Check Review Console

Comprehensive Technical & Product Guide

Version: 1.0.0
Date: January 2026
Classification: Confidential - For Authorized Recipients Only

Table of Contents

1. Executive Overview

1.1 What Is Check Review Console?

Check Review Console is a **bank-grade SaaS platform** designed for community banks and credit unions to streamline their check deposit review workflows. The platform provides:

- **Intelligent Queue Management**: Automated prioritization and routing of check images requiring human review
- **Fraud Detection Integration**: Real-time risk scoring and suspicious activity flagging
- **Dual Control Workflows**: Regulatory-compliant approval processes for high-value transactions
- **Comprehensive Audit Trail**: Immutable logging of every action for regulatory examination
- **Multi-Tenant Architecture**: Secure isolation between financial institution tenants

1.2 Target Market

- Community banks (assets $100M - $10B)
- Credit unions
- Regional banks requiring check review capabilities
- Financial institutions modernizing legacy check processing systems

1.3 Business Value Proposition

Metric	Before	After
Average review time	3-5 minutes	45-90 seconds
Fraud detection rate	72%	94%+
Audit preparation time	2-3 weeks	Real-time
Reviewer capacity	150 checks/day	400+ checks/day

1.4 Regulatory Alignment

The platform is designed to support compliance with:

- **Check 21 Act** - Electronic check processing requirements
- **BSA/AML** - Bank Secrecy Act anti-money laundering controls
- **Regulation CC** - Funds availability and check processing
- **FFIEC Guidelines** - IT examination handbook requirements

- **SOC 2 Type II** - Security, availability, and confidentiality controls

2. Product Capabilities

2.1 Core Features

Check Review Queue

- Real-time queue of checks requiring human review
- Configurable priority rules based on amount, account age, risk score
- Automatic load balancing across available reviewers
- Queue status: `new`, `in_review`, `pending_dual_control`, `approved`, `rejected`, `returned`

High-Resolution Image Viewer

- Secure, time-limited access to check images (front and back)
- Zoom, pan, and enhancement tools
- Side-by-side comparison for signature verification
- No permanent image caching in browser

Decision Workflow

- One-click approve/reject/return actions
- Mandatory reason codes for rejections
- Dual control enforcement for amounts exceeding configurable thresholds
- Real-time notification to second reviewer

Fraud Detection Dashboard

- Risk score visualization (0-100 scale)
- Fraud type classification (check kiting, counterfeit, forged signature, etc.)
- Historical fraud trends by channel, amount bucket, and time period
- SAR (Suspicious Activity Report) preparation assistance

Reporting & Analytics

- Daily/weekly/monthly processing volumes
- Reviewer productivity metrics
- Fraud detection rates and false positive analysis
- Customizable date range exports

Operations Dashboard

- Real-time system health monitoring

- Database, Redis, Prometheus, Alertmanager status
- Active alert summary
- Disaster recovery status and metrics
- Quick links to monitoring tools

Administration

- User management with role-based access control
- Policy configuration (hold amounts, dual control thresholds)
- Connector management for core banking integration
- System configuration and feature flags

2.2 User Roles

Role	Permissions
Reviewer	View queue, review checks, make decisions
Senior Reviewer	All reviewer permissions + dual control approval
Supervisor	All senior permissions + queue management, reassignment
Administrator	All supervisor permissions + user management, policies
Auditor	Read-only access to all data and audit logs
System Admin	Full system access including configuration

2.3 Integration Points

- **Core Banking Systems**: Real-time account lookup, hold placement
- **Check Image Archives**: Retrieval of historical check images
- **Fraud Detection Engines**: Risk score ingestion and alert routing
- **SIEM Platforms**: Structured JSON log export for Splunk/ELK/CloudWatch
- **Identity Providers**: SAML/OIDC federation (roadmap)

3. System Architecture

3.1 High-Level Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                        Load Balancer (TLS)                        │
└─────────────────────────────────────────────────────────────────┘
                                 │
                      ┌──────────┴──────────┐
                      ▼                     ▼
             ┌──────────────────┐  ┌──────────────────┐
             │  Frontend (React) │  │  Frontend (React) │
             │  Static Assets    │  │  Static Assets    │
             └──────────────────┘  └──────────────────┘
                      │                     │
                      └──────────┬──────────┘
                                 ▼
             ┌─────────────────────────────────────────────────────┐
             │                    API Gateway                        │
             │             (Rate Limiting, CORS)                     │
             └─────────────────────────────────────────────────────┘
                                 │
                      ┌──────────┴──────────┐
                      ▼                     ▼
             ┌──────────────────┐  ┌──────────────────┐
             │  Backend API #1   │  │  Backend API #2   │
             │  (FastAPI/Python) │  │  (FastAPI/Python) │
             └──────────────────┘  └──────────────────┘
                      │                     │
             ┌────────┴─────────────────────┴────────┐
             │                                        │
             ▼                    ▼                   ▼
    ┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐
    │   PostgreSQL    │  │      Redis      │  │  Image Storage  │
    │   (Primary)     │  │   (Sessions,    │  │  (S3/MinIO)     │
    │                 │  │    Caching)     │  │                 │
    └─────────────────┘  └─────────────────┘  └─────────────────┘
             │
             ▼
    ┌─────────────────┐
    │   PostgreSQL    │
    │   (Replica)     │
    └─────────────────┘
```

3.2 Technology Stack

Layer	Technology	Version
Frontend	React + TypeScript	18.x
UI Framework	TailwindCSS + Headless UI	3.x
State Management	Zustand	4.x
Backend API	FastAPI (Python)	0.109+
ORM	SQLAlchemy (async)	2.x
Database	PostgreSQL	15+
Cache/Sessions	Redis	7.x
Task Queue	Celery (optional)	5.x
Monitoring	Prometheus + Grafana	Latest
Alerting	Alertmanager	Latest
Container Runtime	Docker	24.x
Orchestration	Kubernetes	1.28+

3.3 API Design

Base URL Structure

```
https://api.checkreview.example.com/api/v1/
```

Endpoint Categories

Category	Prefix	Description
Authentication	`/auth/`	Login, logout, token refresh, MFA
Checks	`/checks/`	Check items, images, decisions
Queue	`/queues/`	Queue management, assignment
Users	`/users/`	User management, profiles
Policies	`/policies/`	Review policies, thresholds
Fraud	`/fraud/`	Fraud events, trends, reports
Audit	`/audit/`	Audit log queries, exports
Operations	`/operations/`	System health, metrics, alerts
System	`/system/`	Configuration, health checks

Request/Response Format

- All requests/responses use JSON
- ISO 8601 timestamps with timezone
- UUID v4 for all entity identifiers
- Pagination via `skip` and `limit` parameters
- Consistent error response schema

3.4 Frontend Architecture

```
frontend/
├── src/
│   ├── components/          # Reusable UI components
│   │   ├── common/          # Buttons, forms, modals
│   │   ├── layout/          # Navigation, sidebar, header
│   │   └── check/           # Check-specific components
│   ├── pages/               # Route-level page components
│   ├── services/            # API client and service layer
│   ├── stores/              # Zustand state stores
│   ├── hooks/               # Custom React hooks
│   └── utils/               # Utility functions
```

4. Data Model & Storage

4.1 Core Entities

User

```
users
├── id (UUID, PK)
├── tenant_id (UUID, FK) - Multi-tenant isolation
├── email (String, unique per tenant)
├── hashed_password (String)
├── full_name (String)
├── role (Enum: reviewer, senior_reviewer, supervisor, admin, auditor)
├── is_active (Boolean)
├── mfa_enabled (Boolean)
├── mfa_secret (String, AES-256-GCM encrypted)
├── last_login (DateTime)
├── created_at (DateTime)
└── updated_at (DateTime)
```

Check Item

```
check_items
├── id (UUID, PK)
├── tenant_id (UUID, FK)
├── external_item_id (String) - Core banking reference
├── account_number (String, last 4 visible)
├── routing_number (String)
├── amount (Decimal, precision 2)
├── check_number (String)
├── payee_name (String)
├── payer_name (String)
├── deposit_date (Date)
├── status (Enum: new, in_review, pending_dual_control, approved, rejected, returned)
├── risk_score (Integer, 0-100)
├── risk_factors (JSON)
├── front_image_path (String) - S3/MinIO path
├── back_image_path (String)
├── assigned_to (UUID, FK to users)
├── assigned_at (DateTime)
├── queue_id (UUID, FK to queues)
├── created_at (DateTime)
└── updated_at (DateTime)
```

Decision

```
decisions
├── id (UUID, PK)
├── tenant_id (UUID, FK)
├── check_id (UUID, FK to check_items)
├── reviewer_id (UUID, FK to users)
├── action (Enum: approve, reject, return, escalate, needs_more_info)
├── reason_code (String)
├── notes (Text)
├── is_dual_control (Boolean)
├── dual_control_reviewer_id (UUID, FK to users)
├── dual_control_action (Enum)
├── dual_control_at (DateTime)
├── created_at (DateTime)
└── updated_at (DateTime)
```

Audit Log

```
audit_logs
├── id (UUID, PK)
├── tenant_id (UUID, FK)
├── user_id (UUID, FK to users)
├── action (String) - e.g., "check.approved", "user.login"
├── resource_type (String)
├── resource_id (String)
├── old_values (JSON)
├── new_values (JSON)
├── ip_address (String)
├── user_agent (String)
├── request_id (UUID) - Correlation ID
├── integrity_hash (String, SHA-256)
├── previous_hash (String) - Chain integrity
├── timestamp (DateTime)
└── created_at (DateTime)
```

Fraud Event

```
fraud_events
├── id (UUID, PK)
├── tenant_id (UUID, FK)
├── check_id (UUID, FK to check_items, nullable)
├── fraud_type (Enum: check_kiting, counterfeit_check, forged_signature, ...)
├── channel (Enum: branch, atm, mobile, rdc, mail, online, other)
├── amount_bucket (Enum: under_100, 100_to_500, ..., over_50000)
├── detected_at (DateTime)
├── reported_at (DateTime)
├── status (Enum: draft, submitted, withdrawn)
├── sar_filed (Boolean)
├── sar_number (String)
├── notes (Text)
├── created_at (DateTime)
└── updated_at (DateTime)
```

4.2 Database Indexes

Critical indexes for query performance:

```
-- Queue processing
CREATE INDEX idx_check_items_tenant_status ON check_items(tenant_id, status);
CREATE INDEX idx_check_items_assigned ON check_items(assigned_to, status);
CREATE INDEX idx_check_items_queue ON check_items(queue_id, status);

-- Audit queries
CREATE INDEX idx_audit_logs_tenant_time ON audit_logs(tenant_id, timestamp DESC);
CREATE INDEX idx_audit_logs_resource ON audit_logs(tenant_id, resource_type, resource_id);
CREATE INDEX idx_audit_logs_user ON audit_logs(tenant_id, user_id, timestamp DESC);

-- Fraud analysis
CREATE INDEX idx_fraud_events_tenant_type ON fraud_events(tenant_id, fraud_type);
CREATE INDEX idx_fraud_events_detected ON fraud_events(tenant_id, detected_at DESC);
```

4.3 Data Retention

Data Type	Retention Period	Archive Strategy
Check images	7 years	Cold storage after 90 days
Audit logs	7 years	Immutable, no deletion
Check items	7 years	Soft delete only
User sessions	30 days	Hard delete
Fraud events	7 years	No deletion

5. Authentication & Authorization

5.1 Authentication Flow

```
┌──────────┐     ┌──────────┐     ┌──────────┐     ┌──────────┐
│  Client  │────▶│  Login   │────▶│   MFA    │────▶│  Access  │
│          │     │ (email/  │     │  (TOTP)  │     │  Granted │
│          │     │  pass)   │     │          │     │          │
└──────────┘     └──────────┘     └──────────┘     └──────────┘
                      │                │                │
                      ▼                ▼                ▼
               ┌────────────────────────────────────────┐
               │        JWT Access Token (15 min)        │
               │      Stored in memory (JavaScript)      │
               └────────────────────────────────────────┘
               ┌────────────────────────────────────────┐
               │       JWT Refresh Token (7 days)        │
               │      Stored in httpOnly cookie          │
               └────────────────────────────────────────┘
```

5.2 Token Architecture

Access Token

- **Format**: JWT (RS256 signed)
- **Lifetime**: 15 minutes
- **Storage**: JavaScript memory only (not localStorage)
- **Contains**: user_id, tenant_id, role, permissions, exp

Refresh Token

- **Format**: JWT (RS256 signed)
- **Lifetime**: 7 days
- **Storage**: httpOnly, Secure, SameSite=Strict cookie
- **Contains**: user_id, tenant_id, token_id (for revocation)

CSRF Token

- **Format**: Random 32-byte hex string
- **Lifetime**: Matches refresh token
- **Storage**: httpOnly cookie + X-CSRF-Token header requirement
- **Validation**: Required for all POST/PUT/PATCH/DELETE requests

5.3 Multi-Factor Authentication

- **Algorithm**: TOTP (RFC 6238)
- **Period**: 30 seconds
- **Digits**: 6
- **Secret Storage**: AES-256-GCM encrypted at rest
- **Backup Codes**: 10 single-use codes, bcrypt hashed

MFA Secret Encryption

```
# Key derivation using HKDF
hkdf = HKDF(
    algorithm=SHA256(),
    length=32,  # 256 bits
    salt=b"check-review-field-encryption",
    info=b"mfa-secret-encryption",
)
key = hkdf.derive(SECRET_KEY.encode())

# Encryption with unique nonce
nonce = os.urandom(12)  # 96 bits for GCM
ciphertext = AESGCM(key).encrypt(nonce, plaintext, None)
```

5.4 Role-Based Access Control (RBAC)

Permission Matrix

Permission	Reviewer	Senior	Supervisor	Admin	Auditor
view_queue	Yes	Yes	Yes	Yes	Yes
review_check	Yes	Yes	Yes	Yes	No
dual_control	No	Yes	Yes	Yes	No
reassign_check	No	No	Yes	Yes	No
manage_users	No	No	No	Yes	No
view_audit	No	No	Yes	Yes	Yes
export_audit	No	No	No	Yes	Yes
manage_policies	No	No	No	Yes	No

Permission Enforcement

```
def require_permission(resource: str, action: str):
    def decorator(func):
        @wraps(func)
        async def wrapper(*args, current_user: User, **kwargs):
            if not has_permission(current_user.role, resource, action):
                raise HTTPException(403, "Insufficient permissions")
            return await func(*args, current_user=current_user, **kwargs)
        return wrapper
    return decorator
```

5.5 Session Security

- Sessions bound to IP address (configurable)
- Concurrent session limit per user (default: 3)
- Automatic logout after inactivity (default: 30 minutes)
- Forced re-authentication for sensitive operations
- Login anomaly detection (new device, unusual location)

6. Secure Image Handling

6.1 The Problem with JWTs in URLs

Traditional approaches of putting JWTs in image URLs are problematic:

1. **Referrer Leakage**: JWT visible in Referer header if user navigates away
2. **Browser History**: URL with token stored in history
3. **Server Logs**: URLs logged by proxies, CDNs, WAFs
4. **Cache Keys**: CDNs may cache URLs containing tokens
5. **Copy/Paste Risk**: Users may accidentally share URLs with tokens

6.2 One-Time-Use Token Solution

Check Review Console implements a secure, one-time-use token system:

```
┌──────────┐     ┌──────────┐     ┌──────────┐     ┌──────────┐
│  Client  │────▶│ Request  │────▶│ Receive  │────▶│  Fetch   │
│  needs   │     │  image   │     │ one-time │     │  image   │
│  image   │     │  token   │     │  token   │     │  (once)  │
└──────────┘     └──────────┘     └──────────┘     └──────────┘
                      │                │                │
                      ▼                ▼                ▼
                ┌──────────────────────────────────────────────┐
                │  POST /checks/{id}/images/{side}/token       │
                │  → Returns: { token: "abc123", expires: 60s }│
                └──────────────────────────────────────────────┘

                ┌──────────────────────────────────────────────┐
                │  GET /images/{token}                          │
                │  → Returns: Image bytes (token invalidated)   │
                └──────────────────────────────────────────────┘
```

Token Properties

Property	Value	Purpose
Format	32-byte random hex	Unpredictable
Lifetime	60 seconds	Minimizes exposure window
Usage	Single-use	Cannot be replayed
Storage	Redis with TTL	Fast validation, auto-expiry
Binding	User ID + Check ID	Prevents token theft

Security Headers on Image Response

```
Cache-Control: no-store, no-cache, must-revalidate, private
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
Referrer-Policy: no-referrer
Content-Security-Policy: default-src 'none'
```

6.3 Image Access Flow

```
@router.post("/checks/{check_id}/images/{side}/token")
async def request_image_token(
    check_id: UUID,
    side: Literal["front", "back"],
    current_user: User,
    db: AsyncSession,
):
    # 1. Verify user has access to this check
    check = await get_check_for_user(db, check_id, current_user)
    if not check:
        raise HTTPException(404)

    # 2. Generate one-time token
    token = secrets.token_hex(32)

    # 3. Store in Redis with metadata
    await redis.setex(
        f"image_token:{token}",
        60,  # 60 second TTL
        json.dumps({
            "check_id": str(check_id),
            "side": side,
            "user_id": str(current_user.id),
            "tenant_id": str(current_user.tenant_id),
        })
    )

    # 4. Audit log the token generation
    await create_audit_log(
        action="image.token_generated",
        resource_type="check",
        resource_id=str(check_id),
        user=current_user,
    )

    return {"token": token, "expires_in": 60}


@router.get("/images/{token}")
async def get_image(token: str):
    # 1. Retrieve and DELETE token atomically
    token_data = await redis.getdel(f"image_token:{token}")
    if not token_data:
        raise HTTPException(404, "Invalid or expired token")

    # 2. Parse token metadata
    metadata = json.loads(token_data)

    # 3. Retrieve image from storage
    image_path = await get_image_path(
        metadata["check_id"],
        metadata["side"]
    )
    image_bytes = await storage.get(image_path)

    # 4. Return with security headers
    return Response(
        content=image_bytes,
        media_type="image/jpeg",
        headers={
            "Cache-Control": "no-store, no-cache, must-revalidate, private",
            "Referrer-Policy": "no-referrer",
            # ... additional headers
        }
    )
```

---

7. Auditability & Compliance

7.1 Audit Log Design Principles

1. **Immutability**: Once written, logs cannot be modified or deleted
2. **Completeness**: Every state change is captured
3. **Integrity**: Cryptographic verification of log chain
4. **Non-repudiation**: Actions tied to authenticated users
5. **Availability**: Logs accessible for 7+ years

7.2 What Gets Logged

Category	Events
Authentication	login, logout, mfa_setup, mfa_verify, password_change, session_timeout
Check Operations	check.viewed, check.assigned, check.approved, check.rejected, check.returned, check.escalated
Dual Control	dual_control.requested, dual_control.approved, dual_control.rejected
Image Access	image.token_generated, image.accessed
User Management	user.created, user.updated, user.deactivated, role.changed
Policy Changes	policy.created, policy.updated, hold_limit.changed
System Events	config.changed, queue.created, queue.modified

7.3 Audit Log Structure

```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "tenant_id": "123e4567-e89b-12d3-a456-426614174000",
  "timestamp": "2026-01-16T14:30:00.000Z",
  "user_id": "789e0123-e45b-67c8-d901-234567890abc",
  "user_email": "reviewer@bank.com",
  "action": "check.approved",
  "resource_type": "check",
  "resource_id": "abc12345-6789-0def-ghij-klmnopqrstuv",
  "old_values": {
    "status": "in_review"
  },
  "new_values": {
    "status": "approved",
    "decision_id": "def67890-1234-5678-9abc-def012345678"
  },
  "ip_address": "192.168.1.100",
  "user_agent": "Mozilla/5.0...",
  "request_id": "req-12345-67890",
  "integrity_hash": "sha256:a1b2c3d4e5f6...",
  "previous_hash": "sha256:9z8y7x6w5v4..."
}
```

7.4 Integrity Chain

Each audit log entry includes:

1. **integrity_hash**: SHA-256 of current record content
2. **previous_hash**: Reference to prior record's hash

This creates a blockchain-like chain where any tampering breaks the chain:

```
def calculate_integrity_hash(record: dict, previous_hash: str) -> str:
    content = json.dumps({
        "tenant_id": record["tenant_id"],
        "timestamp": record["timestamp"],
        "user_id": record["user_id"],
        "action": record["action"],
        "resource_type": record["resource_type"],
        "resource_id": record["resource_id"],
        "old_values": record["old_values"],
        "new_values": record["new_values"],
        "previous_hash": previous_hash,
    }, sort_keys=True)
    return f"sha256:{hashlib.sha256(content.encode()).hexdigest()}"
```

7.5 Audit Query API

```
GET /api/v1/audit/logs?
  start_date=2026-01-01&
  end_date=2026-01-31&
  user_id=789e0123...&
  action=check.approved&
  resource_type=check&
  skip=0&
  limit=100
```

Export Formats

- **JSON**: Native format with full fidelity
- **CSV**: Excel-compatible for manual review
- **SIEM**: Structured JSON for Splunk/ELK ingestion

7.6 Compliance Reporting

Pre-built reports for common examination requests:

Report	Description	Frequency
User Access Report	All users with login history	Monthly
Decision Summary	Approve/reject ratios by reviewer	Weekly
High-Value Transactions	All checks over configurable threshold	Daily
Dual Control Compliance	Verification of dual control enforcement	Weekly
Fraud Detection Summary	Flagged items and outcomes	Monthly

8. Tenant Isolation & Multi-Tenancy

8.1 Isolation Model

Check Review Console uses **database-level row isolation** with `tenant_id`:

```
┌──────────────────────────────────────────────────────┐
│                  Shared Database                       │
├──────────────────────────────────────────────────────┤
│                                                     │
│  ┌───────────────┐  ┌───────────────┐  ┌─────────┐  │
│  │  Tenant A     │  │  Tenant B     │  │Tenant C │  │
│  │  tenant_id=1  │  │  tenant_id=2  │  │ tid=3   │  │
│  │               │  │               │  │         │  │
│  │  users        │  │  users        │  │ users   │  │
│  │  checks       │  │  checks       │  │ checks  │  │
│  │  decisions    │  │  decisions    │  │decisions│  │
│  │  audit_logs   │  │  audit_logs   │  │audit_log│  │
│  └───────────────┘  └───────────────┘  └─────────┘  │
│                                                     │
└──────────────────────────────────────────────────────┘
```

8.2 Enforcement Layers

Layer 1: JWT Token

```
# Access token contains tenant_id
{
  "sub": "user-uuid",
  "tenant_id": "tenant-uuid",
  "role": "reviewer",
  "exp": 1705410000
}
```

Layer 2: Dependency Injection

```
async def get_current_tenant(
    current_user: User = Depends(get_current_active_user)
) -> UUID:
    """Extract tenant_id from authenticated user."""
    return current_user.tenant_id
```

Layer 3: Query Filtering

```
async def get_checks_for_tenant(
    db: AsyncSession,
    tenant_id: UUID,
    status: Optional[str] = None,
) -> list[CheckItem]:
    query = select(CheckItem).where(
        CheckItem.tenant_id == tenant_id  # Always filter by tenant
    )
    if status:
        query = query.where(CheckItem.status == status)
    return (await db.execute(query)).scalars().all()
```

Layer 4: Model Validation

```
class CheckItem(Base):
    __tablename__ = "check_items"

    tenant_id: Mapped[UUID] = mapped_column(
        ForeignKey("tenants.id"),
        nullable=False,
        index=True
    )

    @validates("tenant_id")
    def validate_tenant_id(self, key, tenant_id):
        if tenant_id is None:
            raise ValueError("tenant_id cannot be null")
        return tenant_id
```

8.3 Cross-Tenant Attack Prevention

Attack Vector	Mitigation
Direct ID guessing	UUIDs + tenant filtering on all queries
IDOR (Insecure Direct Object Reference)	Ownership validation before access
SQL injection	Parameterized queries via SQLAlchemy
Token manipulation	JWT signature verification
API enumeration	Rate limiting + audit logging

8.4 Tenant Configuration

Each tenant can customize:

- Hold amount thresholds

- Dual control limits
- Review policies
- User roles and permissions
- Queue configurations
- Fraud detection sensitivity

---

9. Deployment & Operations

9.1 Deployment Architecture

Production Environment

```
                      ┌──────────────────────────┐
                      │   CloudFront / CDN       │
                      │   (Static Assets)        │
                      └────────────┬─────────────┘
                                   │
                      ┌────────────┴─────────────┐
                      │   Application Load       │
                      │   Balancer (TLS)         │
                      └────────────┬─────────────┘
                                   │
             ┌─────────────────────┼─────────────────────┐
             │                     │                     │
             ▼                     ▼                     ▼
      ┌─────────────┐       ┌─────────────┐       ┌─────────────┐
      │  API Pod 1  │       │  API Pod 2  │       │  API Pod 3  │
      │  (FastAPI)  │       │  (FastAPI)  │       │  (FastAPI)  │
      └─────────────┘       └─────────────┘       └─────────────┘
             │                     │                     │
             └─────────────────────┼─────────────────────┘
                                   │
             ┌─────────────────────┼─────────────────────┐
             │                     │                     │
             ▼                     ▼                     ▼
      ┌─────────────┐       ┌─────────────┐       ┌─────────────┐
      │  PostgreSQL │       │    Redis    │       │  S3/MinIO   │
      │  (RDS/HA)   │       │  (Cluster)  │       │  (Images)   │
      └─────────────┘       └─────────────┘       └─────────────┘
```

Kubernetes Resources

```yaml
# Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: check-review-api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: check-review-api
  template:
    spec:
      containers:
      - name: api
        image: check-review-api:1.0.0
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 10
          periodSeconds: 30
        readinessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 5
          periodSeconds: 10
```

9.2 Configuration Management

Environment Variables

Variable	Description	Default
`ENVIRONMENT`	local/development/staging/production	development
`DATABASE_URL`	PostgreSQL connection string	Required
`REDIS_URL`	Redis connection string	Required
`SECRET_KEY`	JWT signing key (256-bit)	Required
`CORS_ORIGINS`	Allowed origins (comma-separated)	http://localhost:3000
`DEBUG`	Enable debug mode	false
`DEMO_MODE`	Enable demo data	false

Secrets Management

- Production: AWS Secrets Manager / HashiCorp Vault
- Kubernetes: Sealed Secrets or External Secrets Operator
- Never in environment variables for production

9.3 Database Migrations

Using Alembic for versioned schema migrations:

```
# Create new migration
alembic revision --autogenerate -m "Add fraud_events table"

# Apply migrations
alembic upgrade head

# Rollback one version
alembic downgrade -1

# View migration history
alembic history
```

Migration Safety Rules

1. Always test migrations on staging first
2. Migrations must be backwards compatible
3. No data-destructive operations without explicit approval
4. Large table migrations during maintenance windows only

9.4 Monitoring Stack

Prometheus Metrics

```
# Custom metrics
from prometheus_client import Counter, Histogram, Gauge

checks_processed = Counter(
    "checks_processed_total",
    "Total checks processed",
    ["status", "tenant_id"]
)

request_latency = Histogram(
    "http_request_duration_seconds",
    "Request latency",
    ["method", "endpoint"]
)

queue_depth = Gauge(
    "check_queue_depth",
    "Current queue depth",
    ["tenant_id", "queue_id"]
)
```

Grafana Dashboards

Pre-configured dashboards:

1. **Application Overview**: Request rates, latencies, error rates
2. **Database Metrics**: Connections, query times, replication lag
3. **Security Metrics**: Failed logins, rate limit hits, suspicious activity
4. **Business Metrics**: Checks processed, approval rates, fraud detection

Alerting Rules

```
groups:
  - name: check-review-alerts
    rules:
      - alert: HighErrorRate
        expr: rate(http_requests_total{status=~"5.."}[5m]) > 0.1
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: High error rate detected

      - alert: DatabaseConnectionHigh
        expr: pg_stat_activity_count > 80
        for: 5m
        labels:
          severity: warning

      - alert: QueueBacklog
        expr: check_queue_depth > 1000
        for: 15m
        labels:
          severity: warning
```

9.5 Disaster Recovery

Recovery Objectives

Metric	Target
RTO (Recovery Time Objective)	1 hour
RPO (Recovery Point Objective)	15 minutes

Backup Strategy

- **Database**: Continuous WAL archiving + daily snapshots
- **Images**: Cross-region S3 replication
- **Configuration**: Git-versioned, encrypted at rest

DR Procedures

1. **Database Failover**: Promote replica to primary
2. **DNS Cutover**: Update Route53 records
3. **Verification**: Run smoke tests against DR environment
4. **Notification**: Alert operations team and stakeholders

10. Security Posture

10.1 Security Architecture Overview

```
┌─────────────────────────────────────────────────────────────────┐
│                      Security Layers                            │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ Layer 7: Application Security                             │  │
│  │ • Input validation • Output encoding • CSRF protection    │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ Layer 6: Authentication & Authorization                   │  │
│  │ • JWT tokens • MFA • RBAC • Session management            │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ Layer 5: API Security                                     │  │
│  │ • Rate limiting • Request validation • Error handling     │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ Layer 4: Data Security                                    │  │
│  │ • Encryption at rest • Encryption in transit • Masking    │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ Layer 3: Infrastructure Security                          │  │
│  │ • Network isolation • TLS 1.3 • Security groups           │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ Layer 2: Audit & Monitoring                               │  │
│  │ • Immutable logs • Real-time alerts • SIEM integration    │  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │ Layer 1: Operational Security                             │  │
│  │ • Access controls • Change management • Incident response│  │
│  └───────────────────────────────────────────────────────────┘  │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

10.2 OWASP Top 10 Mitigations

Risk	Status	Implementation
A01: Broken Access Control	Mitigated	RBAC, tenant isolation, ownership validation
A02: Cryptographic Failures	Mitigated	AES-256-GCM, TLS 1.3, secure key management
A03: Injection	Mitigated	SQLAlchemy ORM, parameterized queries
A04: Insecure Design	Mitigated	Security-first architecture, threat modeling
A05: Security Misconfiguration	Mitigated	Secure defaults, automated configuration
A06: Vulnerable Components	Monitored	Dependency scanning, automated updates
A07: Authentication Failures	Mitigated	MFA, secure sessions, password policies
A08: Software/Data Integrity	Mitigated	Audit chain integrity, signed deployments
A09: Logging/Monitoring Failures	Mitigated	Comprehensive audit logs, real-time alerts
A10: SSRF	Mitigated	URL validation, network isolation

10.3 Security Headers

All responses include:

```
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'
Referrer-Policy: strict-origin-when-cross-origin
Permissions-Policy: geolocation=(), microphone=(), camera=()
```

10.4 Rate Limiting

Endpoint	Limit	Window
Login	5 requests	1 minute
Token refresh	10 requests	1 minute
MFA verification	3 requests	1 minute
Check image token	30 requests	1 minute
API (general)	100 requests	1 minute
Monitoring events	30 requests	1 minute

10.5 Token Redaction

All log output automatically redacts sensitive tokens:

```
class TokenRedactionFilter(logging.Filter):
    TOKEN_PATTERN = re.compile(r'(Bearer\s+)[A-Za-z0-9\-_=]+\.[A-Za-z0-9\-_=]+\.[A-Za-z0-9\-_=]+')

    def filter(self, record):
        record.msg = self.TOKEN_PATTERN.sub(r'\1[REDACTED]', str(record.msg))
        return True
```

10.6 Security Incident Response

Severity Levels

Level	Description	Response Time
P1 - Critical	Active breach, data exposure	15 minutes
P2 - High	Vulnerability exploitation attempt	1 hour
P3 - Medium	Security misconfiguration detected	4 hours
P4 - Low	Policy violation, non-critical	24 hours

Response Procedures

1. **Detection**: Automated alerts + manual reporting
2. **Triage**: Assess severity and impact
3. **Containment**: Isolate affected systems
4. **Eradication**: Remove threat, patch vulnerabilities
5. **Recovery**: Restore services, verify integrity
6. **Lessons Learned**: Post-incident review, update procedures

11. Demo, Pilot, and Production Modes

11.1 Environment Modes

Mode	Purpose	Data	Security
Demo	Sales demos, training	Synthetic only	Relaxed
Pilot	Customer evaluation	Real customer data	Production-grade
Production	Live operations	Real customer data	Full enforcement

11.2 Demo Mode (`DEMO_MODE=true`)

Safeguards

```
# CRITICAL: Demo mode blocked in production
if settings.DEMO_MODE and settings.ENVIRONMENT == "production":
    raise RuntimeError(
        "FATAL: DEMO_MODE=true is not allowed in production environment!"
    )
```

Demo Features

- Auto-seeded synthetic data (configurable count)
- Demo banner displayed in UI
- Demo indicator in navigation
- Pre-populated sample checks for demonstration
- Safe to reset/reseed without data loss

Synthetic Data Generation

```
async def seed_demo_data(reset: bool = False, count: int = 100):
    """
    Generate synthetic demo data:
    - Users with various roles
    - Check items in different statuses
    - Decisions and audit logs
    - Fraud events for trend analysis
    """
    # All generated data uses clearly fake identifiers
    # Account numbers: 0000-XXXX-DEMO
    # Names: "Demo User", "Sample Bank"
    # Amounts: Reasonable ranges for demonstration
```

11.3 Pilot Mode

Characteristics

- Real customer data with full security
- Limited user count (typically 5-10)
- Enhanced monitoring and logging
- Direct support access
- Feedback collection enabled

Pilot Checklist

- [ ] Security review completed
- [ ] Penetration test passed
- [ ] SOC 2 controls verified
- [ ] DR drill completed
- [ ] Runbook documentation finalized
- [ ] Support team trained
- [ ] Customer onboarding completed

11.4 Production Mode

Security Enforcement

Control	Demo	Pilot	Production
MFA Required	Optional	Recommended	Enforced
Session timeout	8 hours	2 hours	30 minutes
Rate limiting	Relaxed	Standard	Strict
Audit logging	Basic	Full	Full + SIEM
Error details	Verbose	Limited	Hidden
Debug endpoints	Enabled	Disabled	Disabled

Production Hardening

```
# Production-only settings
if settings.ENVIRONMENT == "production":
    # Hide error details to prevent information disclosure
    settings.DEBUG = False

    # Enforce strict CORS
    settings.CORS_ORIGINS = ["https://app.checkreview.com"]

    # Require HTTPS
    settings.SECURE_COOKIES = True

    # Enable all security headers
    settings.SECURITY_HEADERS_ENABLED = True
```

12. Testing & Quality Assurance

12.1 Test Strategy

```
┌─────────────────────────────────────────────────────────────────┐
│                        Test Pyramid                              │
├─────────────────────────────────────────────────────────────────┤
│                         ▲                                        │
│                        / \           E2E Tests (10%)             │
│                       /   \          • User workflows            │
│                      /     \         • Cross-component           │
│                     /───────\                                    │
│                    /         \       Integration Tests (20%)     │
│                   /           \      • API endpoints             │
│                  /             \     • Database operations       │
│                 /───────────────\                                │
│                /                 \   Unit Tests (70%)            │
│               /                   \  • Business logic            │
│              /                     \ • Utility functions         │
│             /───────────────────────\• Data validation           │
└─────────────────────────────────────────────────────────────────┘
```

12.2 Test Categories

Unit Tests (202+ tests)

```
# Run unit tests
pytest tests/unit/ -v

# Coverage report
pytest tests/unit/ --cov=app --cov-report=html
```

Test areas:

- Model validation and constraints
- Service layer business logic
- Utility functions (encryption, hashing)
- Schema validation (Pydantic models)

Integration Tests

```
# Run integration tests (requires database)
pytest tests/integration/ -v --db-url=$TEST_DATABASE_URL
```

Test areas:

- API endpoint request/response
- Database CRUD operations
- Authentication flows
- Multi-tenant isolation

Security Tests

```
# Run security-focused tests
pytest tests/security/ -v
```

Test areas:

- Authentication bypass attempts
- Authorization boundary testing
- Injection attack prevention
- Rate limiting verification
- CSRF protection
- Token handling

12.3 Test Coverage

Module	Coverage	Target
`app.api`	85%	80%
`app.core`	90%	85%
`app.models`	95%	90%
`app.services`	80%	80%
Overall	**87%**	**80%**

12.4 Continuous Integration

GitHub Actions Pipeline

```yaml
name: CI

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:15
        env:
          POSTGRES_PASSWORD: test
        options: >-
          --health-cmd pg_isready
          --health-interval 10s

    steps:
      - uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'

      - name: Install dependencies
        run: pip install -r requirements.txt -r requirements-dev.txt

      - name: Run linting
        run: |
          ruff check .
          mypy app/

      - name: Run unit tests
        run: pytest tests/unit/ --cov=app --cov-fail-under=80

      - name: Run integration tests
        run: pytest tests/integration/
        env:
          DATABASE_URL: postgresql://postgres:test@localhost/test

      - name: Run security tests
        run: pytest tests/security/
```

12.5 Quality Gates

Before merge to main:

- [ ] All unit tests pass
- [ ] All integration tests pass
- [ ] Code coverage >= 80%

- [ ] No linting errors (Ruff)
- [ ] Type checking passes (MyPy)
- [ ] Security scan clean (Bandit)
- [ ] Dependency audit clean

---

13. Known Gaps & Roadmap

13.1 Current Limitations

Area	Gap	Severity	Mitigation
SSO	No SAML/OIDC federation	Medium	Planned for Q2
Mobile	No native mobile app	Low	Responsive web works on mobile
Offline	No offline capability	Low	Bank operations require connectivity
Batch	No bulk upload API	Medium	Individual check submission works
Reporting	Limited custom reports	Low	CSV export available

13.2 Technical Debt

Item	Impact	Priority
Legacy API v1 endpoints	Maintenance burden	Medium
Sync database operations	Performance	Low
Frontend bundle size	Load time	Low

13.3 Roadmap

Q1 2026

- Production pilot with 3 banks
- SOC 2 Type II certification
- Performance optimization (P95 < 200ms)

Q2 2026

- SAML/OIDC SSO integration
- Advanced fraud detection models
- Mobile-optimized UI refresh

Q3 2026

- Multi-region deployment

- Real-time core banking webhooks
- Machine learning check verification

Q4 2026

- Batch processing API
- Custom reporting builder
- Public API for fintech integrations

---

14. FAQ / Hard Questions

Security Questions

Q: How do you prevent one bank from seeing another bank's data?

A: Multi-tenant isolation is enforced at multiple layers:

1. **Token Level**: JWT access tokens include `tenant_id` that cannot be modified
2. **Query Level**: All database queries filter by `tenant_id` as the first condition
3. **Model Level**: Foreign key constraints prevent orphaned records
4. **API Level**: All endpoints validate resource ownership before access
5. **Audit Level**: Cross-tenant access attempts are logged and alerted

Q: What happens if your SECRET_KEY is compromised?

A: We have a documented key rotation procedure:

1. Generate new SECRET_KEY and deploy to all instances
2. Existing refresh tokens remain valid until expiration (7 days max)
3. MFA secrets use HKDF-derived keys, so rotation requires re-encryption
4. Run migration script to re-encrypt all MFA secrets with new key
5. Force password reset for all users as precaution

Q: How do you protect check images in transit and at rest?

A:

- **Transit**: TLS 1.3 for all connections, no fallback to older versions
- **At Rest**: S3 server-side encryption (AES-256)
- **Access**: One-time-use tokens that expire in 60 seconds
- **Caching**: `Cache-Control: no-store` prevents browser/CDN caching
- **Logging**: Token generation and image access are fully audited

Q: What audit evidence can you provide for regulatory exams?

A:

- Complete audit log of every action (7-year retention)
- Integrity chain verification to prove no tampering
- User access reports with login/logout times
- Decision reports with reviewer identification
- Export in JSON, CSV, or direct SIEM integration

Architecture Questions

Q: Can this handle our volume of 50,000 checks/day?

A: Yes, the architecture is designed for horizontal scaling:

- Stateless API servers scale to N instances behind load balancer
- PostgreSQL handles 100K+ writes/day with proper indexing
- Redis provides sub-millisecond session lookups
- Image storage in S3/MinIO scales infinitely
- Rate limiting prevents individual abuse without blocking throughput

Q: What's your uptime guarantee?

A:

- Target SLA: 99.9% (8.76 hours downtime/year max)
- RTO: 1 hour (recovery time objective)
- RPO: 15 minutes (recovery point objective)
- Active-passive DR in separate availability zone
- Quarterly DR drills with documented results

Q: How does dual control work technically?

A:

1. Reviewer makes initial decision (approve/reject)
2. If check amount exceeds threshold, status becomes `pending_dual_control`
3. Original reviewer cannot be the second approver (enforced in code)
4. Second reviewer sees pending items in separate queue
5. Second reviewer confirms or overrides
6. Both decisions recorded with timestamps and user IDs
7. Final status updated only after dual control completion

Operational Questions

Q: How do we migrate from our current system?

A:

1. Data mapping session to identify field correspondence
2. One-time historical data import (optional)
3. Real-time feed integration with your core banking system
4. Parallel operation period (both systems active)
5. Gradual traffic shift with rollback capability

6. Full cutover after validation period

Q: What support do you provide during pilot?

A:

- Dedicated implementation manager
- Daily check-in calls first week
- Direct Slack/Teams channel access
- Same-day response for P1/P2 issues
- Weekly business review meetings
- Full documentation and training materials

Q: Can we customize the workflows?

A:

Current customization options:

- Review policy thresholds (amounts, risk scores)
- Dual control limits per queue
- User roles and permissions
- Queue prioritization rules
- Hold amount defaults

Roadmap customization:

- Custom workflow states
- Configurable approval chains
- Integration webhooks

Compliance Questions

Q: Are you SOC 2 certified?

A: SOC 2 Type II certification is in progress, expected Q1 2026. Current controls:

- Access control policies documented and enforced
- Change management process in place
- Incident response procedures defined
- Data encryption at rest and in transit
- Annual penetration testing

Q: How do you handle PII/PHI data?

A:

- All data encrypted at rest (AES-256)

- All data encrypted in transit (TLS 1.3)
- Access logged and auditable
- Data masking in non-production environments
- Right to deletion supported (soft delete with audit trail)
- No data shared with third parties

Q: What happens at contract termination?

A:

1. 30-day notice period
2. Full data export in standard formats
3. Data retained for 90 days post-termination
4. Certified data destruction after retention period
5. Deletion certificate provided

Document Control

Version	Date	Author	Changes
1.0.0	January 2026	Engineering	Initial release

End of Document

Check Review Console - Bank-Grade Security for Community Banking