Spaza Shop Registration and Management System (SSRMS) - Security Overview

1. Authentication & Authorization

- Use OAuth 2.0 with JWT for user authentication.
- Role-based access control (RBAC) to distinguish between customers, shop owners, government officials, and admins.
- Enforce strong password policies (length, complexity, expiration, history).
- Multi-Factor Authentication (MFA) for sensitive roles (especially government officials and admins).

2. API Security

- Use HTTPS for all API traffic to protect data in transit.
- Validate all incoming requests with proper API key or token-based access.
- Implement rate limiting and throttling to prevent abuse (e.g., brute-force attacks).
- Input validation and sanitization to prevent injection attacks (SQL, XSS).

3. Backend & Database Security

- Use Spring Security with built-in CSRF, session management, and CORS protections.
- PostgreSQL configured with encrypted connections (SSL), strong passwords, and minimal privilege principles.
- Avoid exposing raw SQL to the frontend - use secure ORM (like JPA/Hibernate).

4. Logging, Monitoring & Alerting

- Log login attempts, access to sensitive resources, failed authentications, and API requests.
- Use tools like ELK Stack or Prometheus + Grafana for real-time monitoring.
- Set up anomaly detection and alerting for suspicious patterns.

5. Data Protection & Privacy

- Encrypt sensitive data at rest (e.g., passwords using bcrypt, shop info using AES if needed).

- Use hashing for verification-related data instead of storing plain tokens.

- Regularly purge unused or expired data.

6. DevSecOps & Code Security

- Code reviews for all backend logic involving authentication or sensitive data.

- Use static analysis tools (e.g., SonarQube) and vulnerability scanners.

- CI/CD pipelines should include security testing stages.

- Secure secrets management (e.g., environment variables or Vault, not hard-coded).

7. Network & Infrastructure Security

- Firewalls and network segmentation between frontend, backend, and database servers.

- Disable unused ports/services.

- Automatic security updates and patch management.

- Use container scanning (if Docker) and infrastructure as code (e.g., Terraform + security rules).

8. User & Admin Awareness

- Educate users about phishing and safe login practices.

- Limit admin accounts and audit admin activity.

Conclusion:

This architecture follows modern secure development practices and addresses key concerns in the SSRMS system.