Technical Stack for POS Desktop Application

Desktop Application Framework

- **Tauri**
  - Rust-based, lightweight native application shell
  - Uses platform-native webviews (WebView2 on Windows, WebKit on macOS/Linux)
  - Produces small (~3 MB) binaries with low memory consumption (50-100 MB)
  - Built-in secure sandboxing and auto-update capabilities

Frontend UI

- **React with TypeScript**
  - Component-based declarative UI with strong typing
  - Supports scalable architecture and maintainable codebase
  - UI frameworks compatible: Material UI
  - Bundled and optimized with Vite for development and production builds

Embedded Backend

- **Java Spring Boot sidecar**
  - Runs locally as a background HTTP/REST server on localhost
  - Handles business logic, authentication, and hardware APIs
  - Utilizes Spring Data JPA with Hibernate for ORM on SQLite database
  - Spring Security for role-based access control and JWT authentication

    - Bundle as a JAR and launch with Tauri application startup

Local Database

- **SQLite with Write-Ahead Logging (WAL) mode**
    - Supports ACID-compliant transactions with excellent reliability
    - Provides concurrent read access with crash recovery
    - Stores database file locally in encrypted form (AES-256)
    - Automated backup and integrity check mechanisms

Communication between Frontend and Backend

- HTTP/JSON requests over `http://localhost:8080` or configured port
- CORS policies configured to allow local requests
- JWT tokens used for authentication, stored securely on frontend
- Robust error handling and automatic retry strategies

Hardware Integration

- **Node.js SerialPort via Tauri Plugin**
    - Interfaces with barcode scanners (USB and Bluetooth)
    - Sends ESC/POS commands to thermal receipt printers
    - Controls cash drawer via connection through printer interfaces
    - Provides automatic device detection and live status updates

Offline Operation Support

- Local authentication caching enables login without network
- Encrypted token storage with limited validity period for offline use
- Offline data capture and queueing for delayed synchronization
- Emergency manager override codes for extended offline operation

Data Synchronization

- Synchronizes local SQLite data with central backend periodically
- Implements conflict resolution based on timestamps and rules
- Supports delta and batch sync to minimize bandwidth usage
- Configurable sync frequencies and manual trigger options

Security Strategies

- Certificate pinning to prevent man-in-the-middle attacks on sync
- Use of role-based access controls in frontend and backend
- Multi-factor authentication support for staff logins

Deployment & Updates

- Tauri bundler creates platform-specific installers (MSI/DMG/AppImage)
- Auto-update feature configured to fetch releases from GitHub or private server
- CI/CD pipelines automate build, test, and release processes
- Semantic versioning and changelog generation for release management