IoT システムを用いた

店舗向け大量万引き検知プロジェクト

- C 言語×Java（SpringBoot）による IoT 重量センサーシステム

1　プロジェクト概要

- **概要**

　店舗における大量万引き対策を目的とし、重量センサーを用いて商品棚の重量変化を検知することで、短時間での大量商品持ち出しを検出する IoT システムである。

　デバイス側、サーバ側の責務を分離し、安全性および拡張性を重視した設計を行った。

- **ユースケース**

　人手の少ない時間帯において、店舗スタッフが他業務を行っている間に、高額商品がまとめて持ち去られるケースを想定している。

2　開発背景・目的

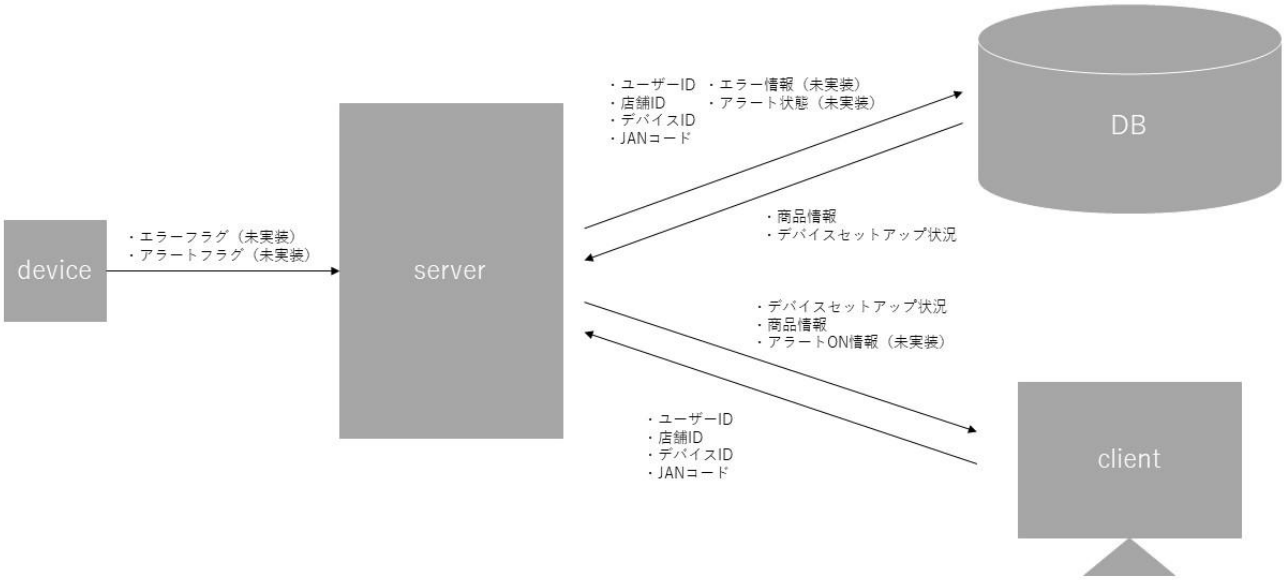
　小売店では、人気商品の大量万引きが問題となっている。一方で、防犯カメラや人員配置などの対策ではコストや運用での負担が大きい。

　そこで本システムでは、重量変化を用いることで誤検知を抑えつつ異常を検知できる仕組みとして設計した。

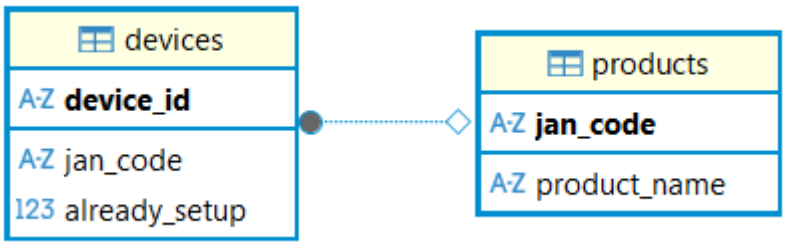
3　システム構成

3.1　データの流れ

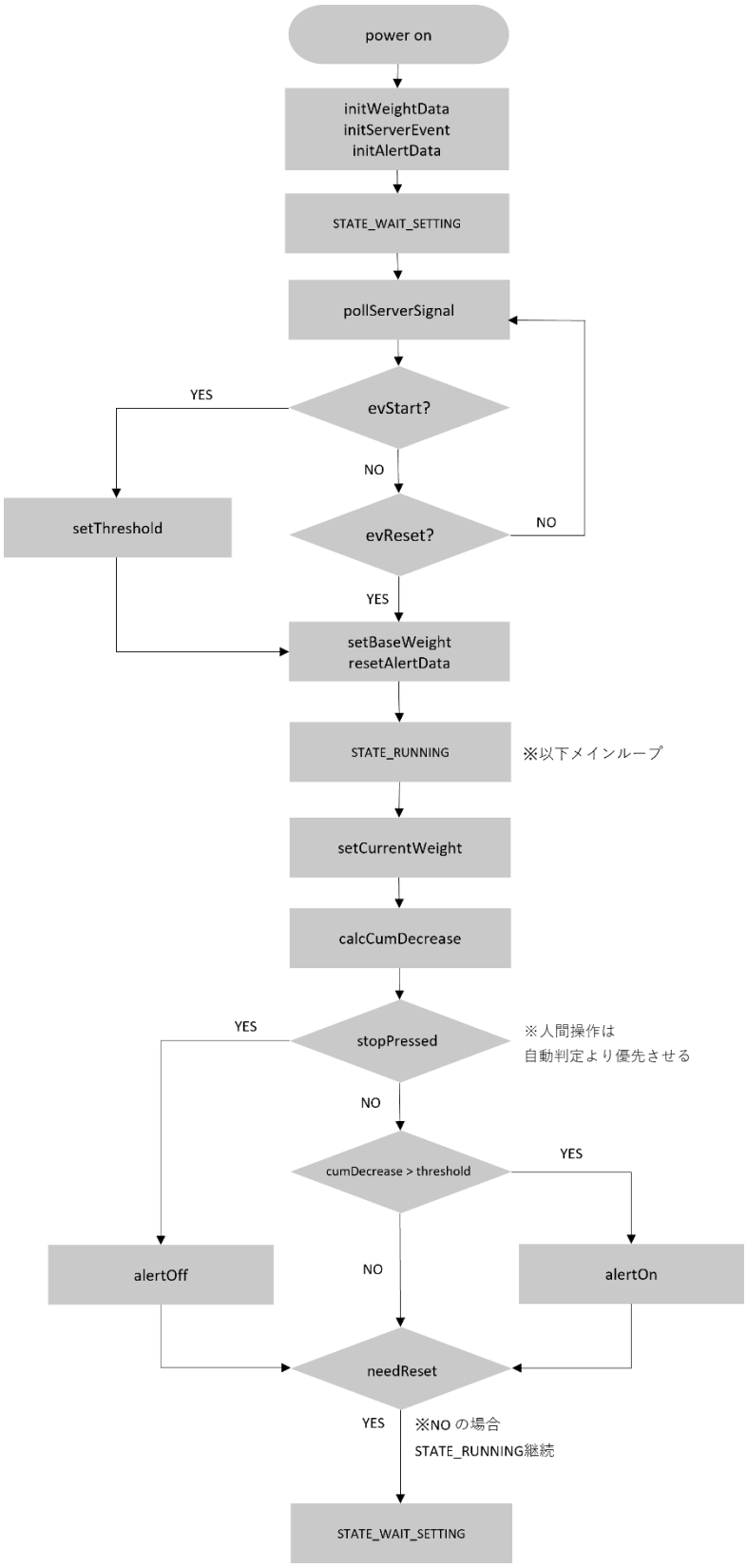
重量センサーから取得したデータは、IoT デバイス側で差分を計算した上で判定結果をフラグとしてサーバへ送信する。状態管理はサーバ側で行う設計としている。



3.2　ER 図



3.3　IoT デバイスフローチャート



4　技術スタック

- デバイス：C 言語
- サーバ：Java（SpringBoot）
- フロントエンド：Thymeleaf
- データベース：MySQL

5　設計方針

5.1　全体設計方針

各レイヤーで役割を明確に分離し、将来の拡張や変更に対応できる構成とした。

5.2　言語ごとの責務分離

デバイス側（C 言語）

- センサー値の取得と重量変化の判定のみを担当
- 意味づけや業務判断は行わず、判定結果はフラグとして出力。上位層で解釈されることを前提としている

サーバ側（Java / SpringBoot）

- エラーやアラート状態の管理や業務ロジックを担当
- ログの蓄積や学習処理の実装も想定している

5.3　デバイス側（C 言語）の安全設計

- 判定と制御ロジックを分離し、副作用のない判定関数として設計
- また、誤動作時の影響を最小限にするため、ボタンによる OFF 操作は常に最優先で処理されるようにした

5.4　テスト設計方針

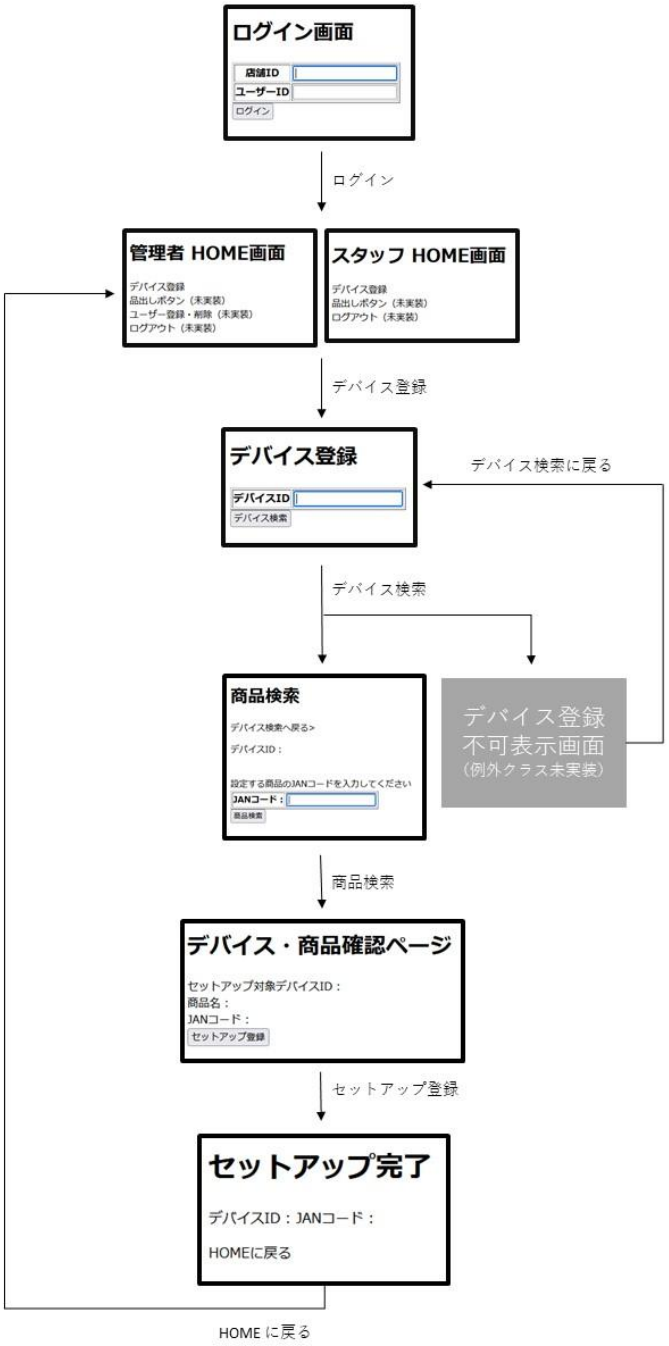
- デバイス側（C 言語）は本番コードとテストコードを分離し、main 関数を差し替える構成とした。本番ロジックはテストを意識せずに実装し、検証はテスト側で行う方針
- サーバ側（SpringBoot）はテストクラス未実装

6　サーバ側構成

6.1　パッケージ構成

SpringBoot の標準的な構成に基づき、 Controller / Service / Entity / Repository に分離している。

6.2　画面構成・遷移



7　実装状況と完成ライン

7.1　デバイス（C 言語）

センサー値の取得と重量変化に基づく基本的な判定ロジックまでを設計し、ハードウェアを想定したシミュレーション実装を行った。

実機条件に左右される処理や、ログやエラーハンドリング、サーバへの出力については、本ポートフォリオでは未実装としている。

7.2　サーバ側（Java / SpringBoot）

SpringBoot アプリは Tomcat（ポート ：8080）で起動済み。

データベース接続も完了し、HTML フォームを通じてブラウザから localhost にアクセス可能。

Controller→Service → Repository→の流れで、データベースの参照（SELECT）およびレコード更新（UPDATE）の動作確認を行った。

データベースはテスト用の仮テーブルを作成して検証している。

8　今後の拡張案

8.1　デバイス（C 言語）

- 実機への接続実装
- アラートやフラグのログ収集
- ログ転送処理

8.2　サーバ側（Java / SpringBoot）

- SpringSecurity によるログイン機能の実装
- 実機稼働状況の取得およびデバイスの管理、アラートの管理
- ログ処理の実装