第 24 回南陵祭 POS システム 全体構想

1.システムアーキテクチャ

本システムは Google が提供するデータベースツール「Firebase」を用い複数のフロントエンドアプリケーションと外部サービスが連携する、アーキテクチャを採用する。

- フロントエンドアプリケーション(来場者が実際に使う)
  - ホスティング
    - GitHub Pages
      - ynr-cs.github.io/nanryosai-2026/pos/配下に全ての静的ファイル（HTML, CSS, JavaScript）を配置する。
      - UI の表示 メニュー、カート、注文状況、レジ、ユーザーインターフェースなどをブラウザ上に描画。
      - Firebase との接続 後述の Firebase の各サービスを JavaScript から呼び出す。
      - JavaScript 全ての動的な処理を担う。ここで Firebase SDK(Software Development Kit)というライブラリを使い、認証やデータベースへのアクセスを行う。
  - 来場者向け
    - mobile-order.html Google アカウント連携を前提としたパーソナライズされたモバイルオーダー。
    - sok.html 店舗設置のセルフオーダーキオスク(SOK)。
    - sok-pay.html SOK から遷移し Google アカウントログインと与信確保とを行う。
    - status.html オンライン注文者が自分自身の注文状況をリアルタイムで追跡するページ。

第 24 回南陵祭 POS システム 全体構想

- 出店団体向け
  - portal.htm すべての管理機能を集約した運営の司令塔。
  - pos.html 有人レジ用の注文入力ページ。
  - kitchen.html 厨房スタッフ向けのページ。
  - presenter.html 提供口スタッフ向けのページ。
  - monitor.html 来場者に提供準備が完了したことなどを知らせるためのモニター(卓上 iPad 等)。
- バックエンド(Firebase)
  - Firebase Authentication Google アカウントを利用したログイン処理を担当。
  - Cloud Firestore すべてのデータ(注文・ステータス・メニュー)を保持しリアルタイムで各フロントエンドに同期するシステムの SSOT(信頼できる唯一の情報源)。
  - Cloud Functions フロントエンドでは実行できない、または実行するべきではないサーバーサイドの処理を担当。
    - **外部 API 連携** au Pay や SendGrid の API を呼び出す処理。API キーなどの機密情報をフロントエンドに組み込むことは危険なため、必ず Cloud Functions を経由しい実行する。
    - **決済処理** 与信確保や売上確定など、金銭に関わる重要な処理。
    - **トリガー実行** Firestore のデータ変更をきっかけにメール送信や Push 通知などの処理を自動実行する。
    - **定期実行処理 (Scheduled Function):** 1 分ごとに起動し、後述する放置ペナルティの対象となる注文がないか監視する。

第 24 回南陵祭 POS システム　全体構想

- Firebase Cloud Messaging (FCM)
  - Web Push 通知をユーザーのブラウザに送信する役割を担う。「商品の準備ができました」といった重要な更新を、ユーザーがサイトを閉じていても通知できる。

- **外部連携サービス**
  - au Pay　指定売上を利用した安全な決済処理の心臓部。
    - 確実に実現できるかは au Pay との事前協議が必要。
    - Googel スプレッドシートと Google Apps Script(GAS)　取引データをリアルタイムで GAS が書き込み、閲覧と分析を可能にするデータ出力先。
    - **メール送信サービス**　SendGrid 等を用いた来場者への自動通知メールを配信するためのサービス

2.シナリオ別注文ワークフロー

このセクションでは、3 つの異なる注文方法（モバイルオーダー、セルフオーダー、有人 POS）が、どのように処理され、最終的に商品として提供されるかの一連の流れを、技術的な観点と運用上の観点を交えて詳細に記述したものである。

1.モバイルオーダー

(ア) **目的:**　来場者が直接注文に並ばなくても注文ができることによる混雑の緩和。

(イ) **決済フロー**

① **ログインと店舗選択**

1. 来場者は校内に掲示された QR コードから南陵祭 Web サイトへアクセスし、mobile-order.html にアクセスする。

第 24 回南陵祭 POS システム 全体構想

2. 最初に、Google アカウントによるログイン認証が要求される。これにより、以降の操作がすべての特定のユーザーに紐づけられる。

3. 初回ログイン時のみ、認証成功後に以下の初期設定を順番に行う。

   (ア)Web Push 通知の許可要求: 「商品の準備ができた際に通知でお知らせします。」というメッセージを表示し、通知の許可を求める。

   (イ)利用規約・ペナルティへの同意要求: 無断キャンセル(商品完成後の指定時間内の未受け取り)の時の対応に関する規約を提示し、同意を求める。同意を得られない場合、以降のフローへと進むことはできない。

② メニュー選択と注文確定

1. 選択した店舗のメニュー画面が表示される。在庫状況(inventory_count)や販売ステータス(is_on_sale)などを Firestore からリアルタイムで取得し、「売り切れ」や「販売停止中」の商品は選択できないように UI を制御する。

2. 購入したい商品をカートに追加する。カート情報は Google アカウントと紐づいた Firestore のサブコレクションに保存される。

3. 内容を確認後、「決済へ進む」ボタンを押す。

③ 与信確保(オーソリゼーション)

1. 「決済へ進む」が押された瞬間、処理が開始される。

2. フロントエンドが Cloud Functions の createOnlineOrder を呼び出す。

3. createOnlineOrder はまず counters/mobile ドキュメントを更新し、7000 番台の受付番号を生成する。

4. 次に、カート情報・受付番号・ユーザー情報を含んだ注文

ドキュメントを、Firestore の orders コレクションに

status: 'awaiting_payment'として作成する。

5. 次に、この注文情報を元に au PAY の API に対し、「指定売上」方式での与信確保（Auth）リクエストを送信する。これは来場者の残高から支払い枠を「仮押さえ」するだけで、実際の引き落としは行わない。
6. API から与信確保成功のレスポンスを受け取ると、createOrder 関数は Firestore の該当注文ドキュメントの status を authorized に更新し、各種タイムスタンプ（authorized_at）を記録する。与信失敗の場合は、ドキュメントの status を auth_failed に更新し、フロントエンドにエラーを返す。
7. この status 更新をトリガーに、別の Cloud Functions と FCM が連携し、ユーザーの Google アカウントのメールアドレス宛に「注文受付完了」メール（受付番号、注文内容記載）を自動送信する。
8. フロントエンドは status.html に遷移し、「調理を開始しました」と表示。

④ **以降の「調理」から「提供」までの流れは、全シナリオ共通のフローへ続く。**

2. セルフオーダー(SOK)

(ア) 目的: 注文で並ぶ時間を大幅に削減する。

(イ) 決済フロー

① 端末でのメニュー選択

1. 来場者は店舗カウンターに設置された sok.html が表示された iPad または端末を操作する。
2. メニューを閲覧し、商品を選択してカートを確定させる。基本的なメニュー情報の取得ロジックはモバイルオーダーと

共通。

② 仮注文と QR コード生成

1. フロントエンド(sok.html)はカート情報を Cloud Functions（例:createOrderFromSok）に送信する。
2. Cloud Functions 関数が status: 'awaiting_payment'で注文を仮登録し、そのドキュメント ID（注文 ID）を取得する。
3. 関数は、この注文 ID をクエリパラメータに含んだ決済ページの URL（例: https://…/pay.html?orderId=…）を生成し、フロントエンドに返す。
4. sok.html は、受け取った URL を元に QR コード描画ライブラリを使い、QR コードを画面に表示する。
5. 来場者は自身のスマートフォンで QR コードを読み取る。すると、スマホのブラウザが pay.html を開き、注文内容が表示される。pay.html は、URL のクエリパラメータから orderId を読み取り、Firestore から注文情報を取得して表示する。Google アカウントでのログインを求められた後、「決済へ進む」ボタンを押す。ここから先の**与信確保のバックエンド処理は、シナリオ A のステップ 3 与信確保とほとんど同じ** Cloud Functions を呼び出すことで共通化される。ただし、受付番号は 2000 番台を生成する。

3. 有人 POS レジ

(ア) 目的: 高齢者の方や複雑な注文などに対応する。

(イ) 決済フロー

① 口頭での注文と入力

1. 来場者はレジ担当スタッフに口頭で注文を伝える。スタッフは pos.html が表示された端末で、注文内容を素早く入力し、注文を確定する。

② 未払い注文の登録

1. スタッフが注文を確定すると、処理が実行される。
2. フロントエンドが Cloud Functions の createPosOrder を呼び出す。この関数は、counters/pos[端末番号]から **100/200 番台**の受付番号を生成。
3. フロントエンド（pos.html）は、注文情報を Cloud Functions（例: createOrderFromPos）に送信する。
4. この関数は、決済や与信確保を一切行わず、Firestore の orders コレクションに status: 'unpaid_at_pos'としてドキュメントを作成する。各種タイムスタンプ（created_at）も記録する。

③ 受付番号の伝達

1. 注文が登録されると、pos.html に受付番号が表示される。スタッフはその番号を小さな紙に書き、「こちらの番号でお待ちください」と来場者に手渡す。

④ 以降の「調理」から「提供」までの流れは、全シナリオ共通のフローへ続く。

4. 共通オペレーションフロー（調理から提供まで）

(ア)フロー

① 【厨房】調理

1. kitchen.html はつねに status を監視しており、が status authorized または unpaid_at_pos の注文は、全てリアルタイムで厨房の kitchen.html に表示される。
2. 厨房スタッフは、表示された順に調理を行う。
3. 完了後、該当注文の「調理完了」ボタンを押す。Cloud Functions が status を ready_to_serve に更新。

第 24 回南陵祭 POS システム 全体構想

kitchen.html から該当注文が消える。

②【提供口】準備と呼び出し

1. status が ready_to_serve の注文が、提供口の presenter.html に「準備待ち」として表示される。
2. 提供口スタッフはスプーンなどの最終準備を行い、完了後、「提供準備完了」ボタンを押す。
3. Cloud Functions が status を ready_for_pickup に更新。この更新をトリガーに、monitor.html と、オンライン注文者の status.html に、該当番号が「お呼び出し中」として表示される。

③【提供口】商品提供と決済

1. 呼び出しを見た来場者が、受付番号（スマホ画面 or 紙）を提示する。

   (ア)提供できる場合

   ① オンライン注文の場合

   1. スタッフは商品を渡し、presenter.html の「提供・決済完了」ボタンを押す。これをトリガーに、Backend は au PAY に「売上確定（Capture）」リクエストを送信。この瞬間、初めて引き落としが実行され、status は completed_online となる。

   ② 有人 POS 注文の場合

   1. スタッフは「あちらの QR を読み取り、[金額]円をお支払いください」と案内。来場者は店舗設置の QR をスキャンし、金額を手入力して支払い、完了画面をスタッフに見せる。スタッフは目視確認後、商品を渡し、presenter.html の「提供・決済完了」ボタンを押す。Backend は status を completed_at_store に更新する。

（イ）提供できない場合

① 提供不可の場合、スタッフは presenter.html の「提供不可・キャンセル」ボタンを押す。

② オンライン注文なら「与信解放（Cancel/Void）」が実行され（お金は動かない）、status は cancelled に更新される。status.html では店舗都合により注文がキャンセルされた旨と支払い枠の仮押さえが解放された旨を表示し、キャンセルメールを送信される。

③ 有人 POS 注文なら、単に status が cancelled に更新される。monitor.html にはキャンセル情報が表示され、申し訳ないが店舗都合によりキャンセルさせていただいた旨を直接伝え謝罪する。

5.放置キャンセル防止・ペナルティフロー

（ア）Firebase の Scheduled Function が 1 分ごとに自動起動する。

（イ）status が ready_for_pickup のままで、かつ呼び出し開始時刻（ready_for_pickup_at）から 15 分以上経過している注文を Firestore から検索する。

（ウ）対象の注文が見つかった場合、システムは以下の処理を自動的に実行する。

① au PAY API に対し、売上確定(Capture)を強制実行する。
② 注文ステータスを abandoned_and_paid（放棄・決済済）に更新する。
③ 対象ユーザーの ID を、banned_users コレクションに記録する。
④ 対象ユーザーに「規約違反により商品を廃棄、決済を確定し、アカウントを無期限に利用停止しました」という趣旨のメールを送信する。

6.利用制限ユーザーへの対応

（ア）ユーザーが Google アカウントでログインを試みた際、認証成功後

にアプリケーションはまずそのユーザーID が banned_users コレクションに存在するかを確認する。

(イ) 存在した場合、ログイン処理を中断し、「あなたのアカウントは利用が制限されています」というエラーメッセージを表示する。これにより、ペナルティを受けたユーザーは以降のサービス利用ができなくなる。