# IT研修発表

Eチーム:Equipe

# 1. プロジェクト概要

- テーマ：シンプル雑貨オンライン
- 開発期間：2025/07/01 ~ 2025/07/31
- チーム名：Equipe
- チームメンバー：朝枝教人、鶴喰怜、高木優、坂口あやの
  花岡歩、佐藤諒
- 開発目的・ゴール：Spring BootとWeb アプリケーション開発
  の基本スキル習得、チーム開発の経験

# 2-1. 開発したシステムの特徴

- 機能一覧

**ユーザー側：**

ログイン/ログアウト機能、商品一覧/カテゴリー別/詳細表示、カート追加/更新/削除、注文機能、カート情報保存機能*、注文情報自動入力機能*、登録情報確認*、トークン認証*

(*がついているものは会員登録した場合のみ)

**管理者側：**

ログイン/ログアウト機能、カテゴリ編集機能、　商品情報編集機能、注文情報確認機能、管理者編集機能

# 2-2. 開発したシステムの特徴

- アーキテクチャ概要

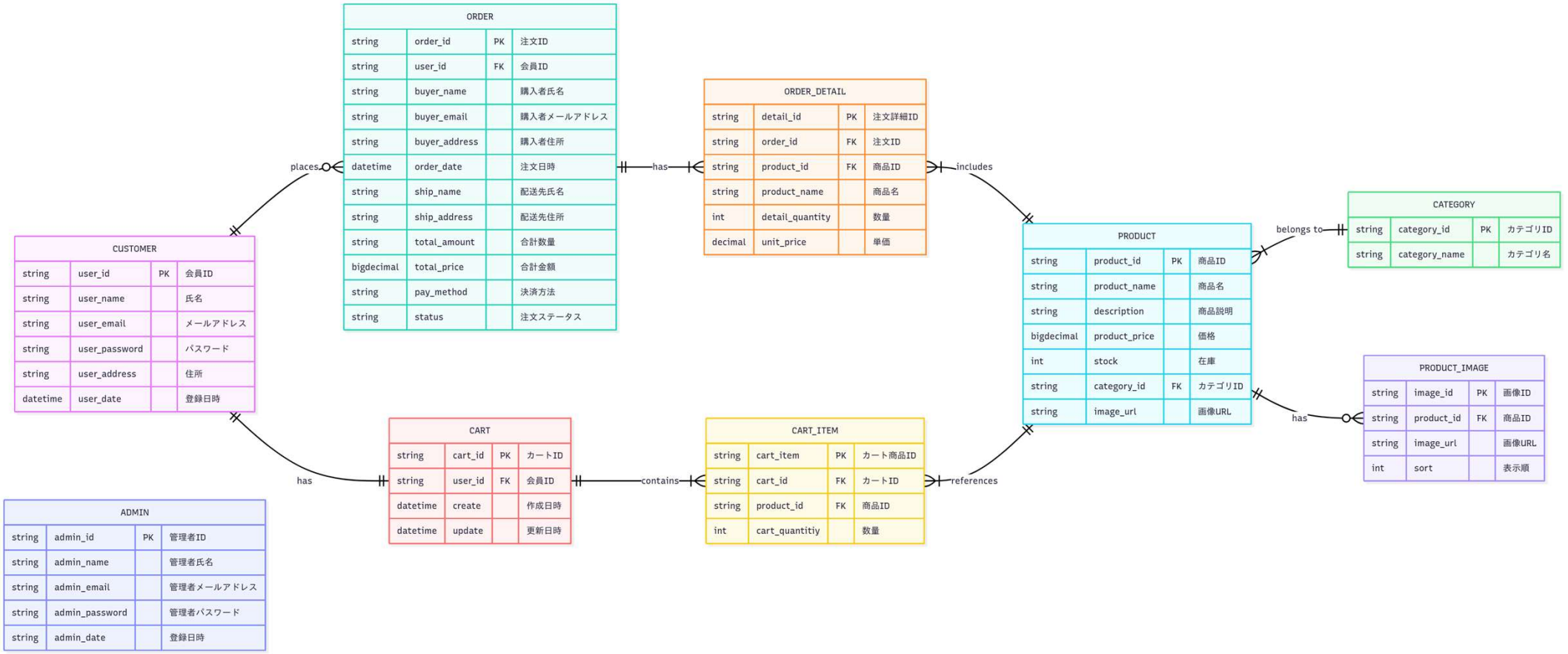
Spring Boot, Java, JPA(Hibernate), H2 Database,
HTML/CSS/JavaScript(Bootstrap), REST API

- 主要な画面フロー

ユーザー側：ログイン→商品一覧ページ→商品詳細→カート→
注文フォーム→注文完了

管理者側：ログイン→ダッシュボード→注文情報確認など

# 2-3. ER図（データベース設計）



# 2-4. API設計（ユーザー）

| APIエンドポイント | | メソッド | 機能 |
|---|---|---|---|
| "/api/product" | | GET | 商品一覧取得 |
| | "/{productId}" | GET | 商品詳細取得 |
| | "/category/{categoryId}" | GET | カテゴリ別商品一覧取得 |
| "/api/cart" | | GET | ゲストカート取得（セッション） |
| | | POST | カートに商品追加 |
| | "/items/{itemId}" | PUT | カート内商品の数量変更 |
| | "/items/{itemId}" | DELETE | カート内商品の削除 |
| "/api/order" | | POST | 注文情報登録（ゲスト） |
| "/api/auth" | "/login" | POST | 会員ログイン |
| | "/register" | POST | 新規会員登録 |
| | "/logout" | POST | 会員ログアウト |
| "/api/user" | "/me" | GET | 注文情報入力時の会員情報取得 |
| | "/mypage" | GET | マイページ用会員情報取得 |
| "/api/usercart" | | GET | 会員カート取得（DB） |
| | | POST | カートに商品追加 |
| | "/items/{itemId}" | PUT | カート内商品の数量変更 |
| | "/items/{itemId}" | DELETE | カート内商品の削除 |
| "/api/user/orders" | | POST | 注文情報登録（会員） |

# 2-4. API設計のポイント（管理者）

| APIエンドポイント | | メソッド | 機能 |
|---|---|---|---|
| "/api/admin/auth" | "/login" | POST | 管理者ログイン |
| | "/logout" | POST | 管理者ログアウト |
| "/api/admin/categories" | | POST | カテゴリ登録 |
| | "/{categoryId}" | PUT | カテゴリ更新 |
| | | GET | カテゴリ一覧取得 |
| | "/{categoryId}" | DELETE | カテゴリ削除 |
| "/api/admins" | | GET | 管理者一覧取得 |
| | | POST | 管理者登録 |
| | "/{id}" | DEKETE | 管理者削除 |
| "/api/admin/order" | | GET | 注文一覧取得 |
| | "/{orderId}" | GET | 注文詳細取得 |
| | "/{orderId}" | PATCH | 注文ステータス更新 |
| "/api/admin/products" | | POST | 商品登録 |
| | | GET | 商品一覧取得 |
| | "/{id}" | PUT | 商品情報更新 |
| | "/{id}" | DELETE | 商品削除 |

# 3-1. 単体テスト

- テスト項目表をもとにテストコードを作成
- service,controller,repository単位でロジックを確認
- フレームワークとしてJUnitを使用

**AdminAuthControllerTest.javaの例**

```
16  class AdminAuthServiceTest {
77      void authenticate_EmptyEmail_ThrowsException() {
78
79          when(adminRepository.findByAdminEmail(adminEmail:"")).thenReturn(Optional.empty());
80
81          IllegalArgumentException ex = assertThrows(expectedType:IllegalArgumentException.class, () ->
82              adminAuthService.authenticate(email:"", validPassword));
83
84          assertEquals(expected:"管理者が見つかりません", ex.getMessage());
85      }
86
87      @Test
88      void authenticate_EmptyPassword_ThrowsException() {
89          when(adminRepository.findByAdminEmail(validEmail)).thenReturn(Optional.of(sampleAdmin));
90
91          IllegalArgumentException ex = assertThrows(expectedType:IllegalArgumentException.class, () ->
92              adminAuthService.authenticate(validEmail, password:""));
93
94          assertEquals(expected:"パスワードが正しくありません", ex.getMessage());
95      }
96
97
98
99      @Test
100     void authenticate_NullPassword_ShouldThrowException() {
101         when(adminRepository.findByAdminEmail(validEmail)).thenReturn(Optional.of(sampleAdmin));
102
103         IllegalArgumentException ex = assertThrows(expectedType:IllegalArgumentException.class, () ->
104             adminAuthService.authenticate(validEmail, password:null));
105         assertEquals(expected:"パスワードが正しくありません", ex.getMessage());
106     }
107 }
```

# 3-2. 結合テスト

- テストケースをまとめたシナリオを作成
- APIとDB、画面の連携を確認
- シナリオごとにmd形式で記述、JSONでテスト結果を返す

simplezakka結合テストシナリオ一覧

1. 管理者ログイン機能

| シナリオ分類 | No. | シナリオ名 | 連携確認ポイント | 期待される主な結果 | 結果1回目 |
|---|---|---|---|---|---|
| 正常系 | 1-1 | 正しい資格情報でログイン | POST /admin/auth/login→AdminAuthController →AdminAuthService→AdminRepository→DB | セッションにADMIN_SESSIONが格納され、レスポンスが"success"(HTTP 200 OK) | ok |
| | 1-2 | ログイン済み状態からログアウト | POST /admin/auth/logout→AdminAuthController →HttpSession.invalidate() | セッションが破棄され、HTTP 200 OKが返される | ok |
| 異常系 | 1-3 | 登録していないemailでログイン失敗 | POST /admin/auth/login→AdminAuthController →AdminAuthService→AdminRepository | メールアドレス不一致により例外が投げられ、HTTP 401 Unauthorizedと"ログイン失敗"メッセージが返される | ok |
| | 1-4 | パスワードの不一致でログイン失敗 | POST /admin/auth/login→AdminAuthController →AdminAuthService→AdminRepository | パスワード不一致により例外が投げられ、HTTP 401 Unauthorizedと"ログイン失敗"メッセージが返される | ok |
| | 1-5 | フォーム未入力でログイン失敗 | <form>入力チェック→JavaScript→Controllerへ未送信 | 必須項目未入力のため、JavaScriptやブラウザのバリデーションでリクエストがブロックされる | ok |

ユーザー　管理者　Sheet3　＋

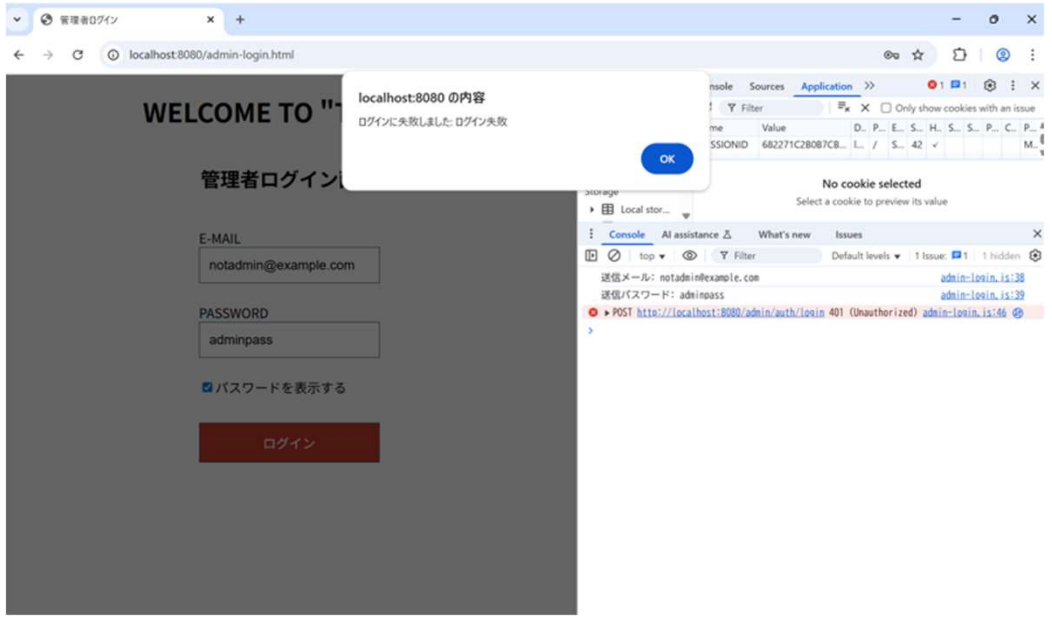
```
## No. 1-5
- テストケース名: ログイン失敗（異常系 - メールアドレスまたはパスワード未入力）

- 前提条件:
  - ブラウザのHTMLフォームで、メールアドレスまたはパスワードが未入力の状態でログインボタンが押される。

- 手順:
  1. 管理者ログインフォームに空の値を入れて送信する。

- 入力データ:
  - email: ""
  - password: ""

- 期待結果:
  1. ブラウザの入力バリデーション、またはJavaScript によりリクエスト自体がサーバーへ送信されない。
  2. サーバー側ではコントローラやサービスロジックは実行されない。
  3. フロントエンド側で「入力必須です」という旨のメッセージが表示されること。
```

# 3-3. 総合テスト

- 機能/非機能に分けテストケース一覧を作成
- 実際にブラウザを操作して動作を確認し、画面キャプチャ
- 不具合に対して開発者ツールによる原因究明
- Excelファイルに総合テスト仕様書としてまとめる

1. 管理者ログイン/ログアウト機能

| No. | テスト対象 | テスト種別 | 目的/観点 | 前提条件 | 操作手順 | 期待結果 |
|---|---|---|---|---|---|---|
| ST-AL-001 | 管理者ログイン画面 | 機能 | 有効な管理者アカウントで正常にログインできるか | 管理者<br>email:admin@example.com<br>pass:adminpass | 1. http://localhost:8080/admin-login.html にアクセス<br>2. emailに"admin@example.com"、パスワードに"adminpass"を入力<br>3.「ログイン」ボタンをクリック | 1. POST /admin/auth/login が実行され、HTTPステータス200が返ること<br>2. セッションが開始されること<br>3. admin-top.html に遷移されること |
| ST-AL-002 | 管理者トップ画面 | 機能 | ログイン済み管理者が正常にログアウトできるか | ST-AL-001 を実行してログイン済みであること | 1.http://localhost:8080/admin-top.htmlの画面下の「ログアウト」ボタンをクリックする | 1. POST /admin/auth/logout が実行され、HTTPステータス200が返ること<br>2. セッションが破棄されること<br>3. admin-login.html にリダイレクトされること |
| ST-AL-003 | 管理者ログイン画面 | 機能 | 未登録のIDでログインしようとした場合の挙動確認 | 管理者アカウント(email):"notadmin@example.com"は存在しない | 1. ログイン画面にアクセス<br>2. emailに"notadmin@example.com"、パスワードに"adminpass"を入力<br>3.「ログイン」ボタンをクリック | 1. POST /admin/auth/login が実行され、HTTP 401 Unauthorized が返ること<br>2. 画面にエラーメッセージが表示され、ログインできないこと |



# 3-4. 品質確保の工夫

- 機能ごとにテスト項目を洗い出し、網羅性を意識
- 不具合が見つかったら即修正＆再テスト
- テスト結果のエビデンスのいかに残すか
- 開発完了前に動作確認を実施
- 実際の利用を意識して発生しうる挙動ごとにテスト

# ４．ふりかえり

■困難だった点：

- ログの取得や表示が不十分だったため、エラー時の原因追跡が困難だった。
- tokenやセッションの設計に苦労した。
- 進捗管理がうまくいかず、個々のタスクの遅れが全体に影響した。
- 仕様変更への対応が遅れたことで修正作業が増え、スケジュールが逼迫した。

# ４．ふりかえり

■チームビルディング：

- フェーズごとにリーダーを交代で担当することで、全員が主体的に関われるように工夫した。
- オンライン会議を定期的に実施し、課題の早期発見に努めた。
- 情報共有ツールやタスク管理ツールを活用し、各メンバーの作業状況が見えるようにした。
- 意見を出しやすい雰囲気づくりを意識し、互いの提案に耳を傾ける文化を育てた。

# 5. システムデモンストレーション

■ユーザーホーム画面の紹介
■ゲストユーザーによる注文完了まで
■ログイン画面に戻り新規会員登録から注文完了まで
■登録した会員情報が保存されていることをログイン画面から確認
■管理者画面から在庫数の減少、完了した注文情報を確認
■管理者画面の紹介

# 6. 今後の展望・改善提案

- 追加したい機能：
- 決済方法追加（カード、ネット決済）
- マイページから会員登録情報編集
- 商品検索
- 注文履歴
- メール自動送信
- 問い合わせ・チャットボット
- お気に入り
- 複数の商品画像