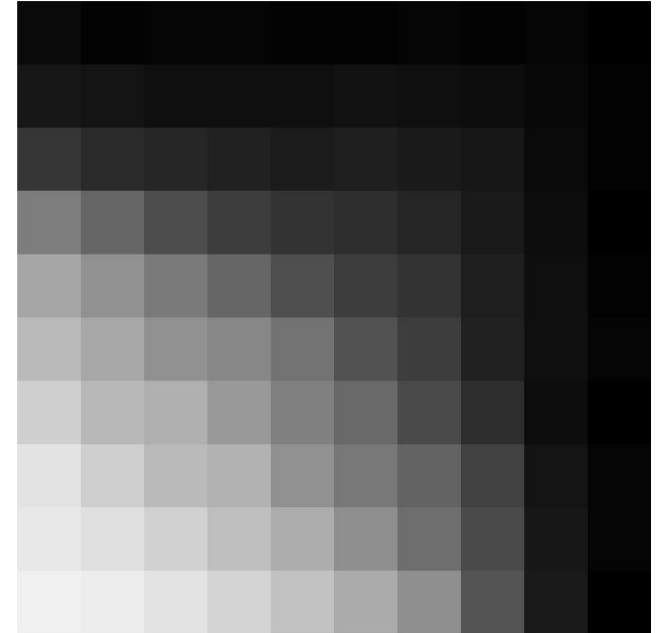


# IT研修 振り返り

チームneko(Dチーム)

Ver.2.7 : 2025/1/21



# ｜ アジェンダ

1. プロジェクト概要
2. 開発したシステムの特徴
3. テスト・品質確保の取り組み
4. ふりかえり結果
5. デモや成果物の画面紹介
6. 今後の展望・改善提案
7. 質疑応答

# 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: 2025/06/16 (月) ~ 2025/07/31 (木)

- チーム体制: チーム名: neko

メンバー紹介: 森旺介、山下棋平、吉岡裕矢、  
中村華、三浦丙都、木村日向子

- 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.機能一覧

### 顧客

- **商品表示**
  - 商品一覧
  - 詳細表示、カテゴリーフィルタ、商品検索
- **会員機能**
  - 会員登録、ログイン
  - マイページ、ログアウト
- **カート機能**
  - 追加、編集、削除
  - 注文確認、注文完了

### 管理者

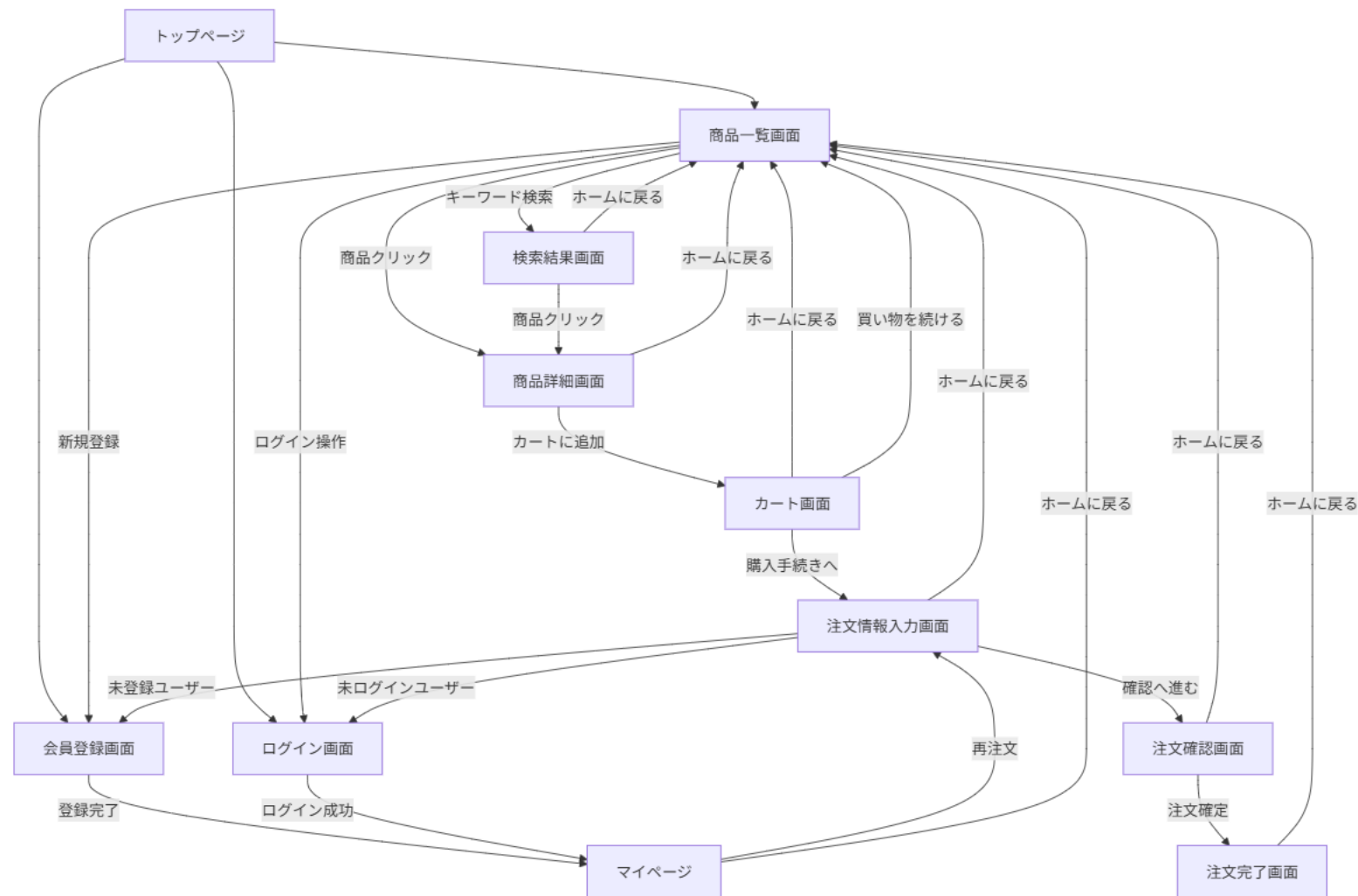
- **ログイン機能**
  - ログイン
- **商品管理**
  - 商品一覧
  - 商品登録
  - 商品編集
  - 商品削除

## 2-2.アーキテクチャ概要

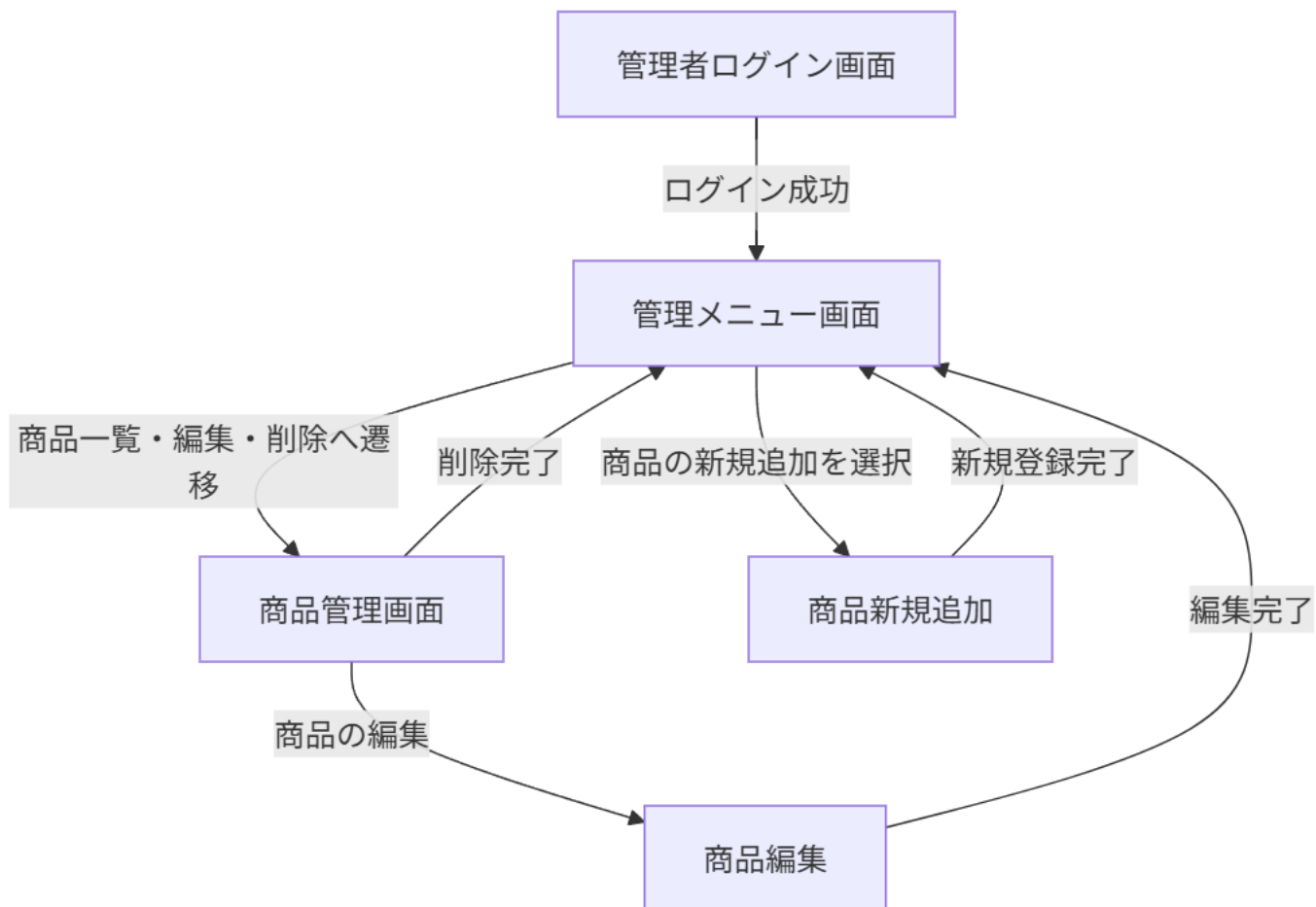
- **クライアント**
  - **顧客**：PC・スマートフォン・タブレット端末のWebブラウザ（Google Chromeの最新バージョン）
  - **管理者**：PC（Windows 10/11） + Google Chromeを想定
- **サーバー**：クラウド環境（AWSを想定）
- **ネットワーク**：公開インターネットを介したHTTPS接続（SSL証明書適用）

言語	JAVA
フレームワーク	Spring Boot
データベース	H2 Database
フロントエンド	HTML5, CSS3, JavaScript(ES6)
Webサーバー	AWS

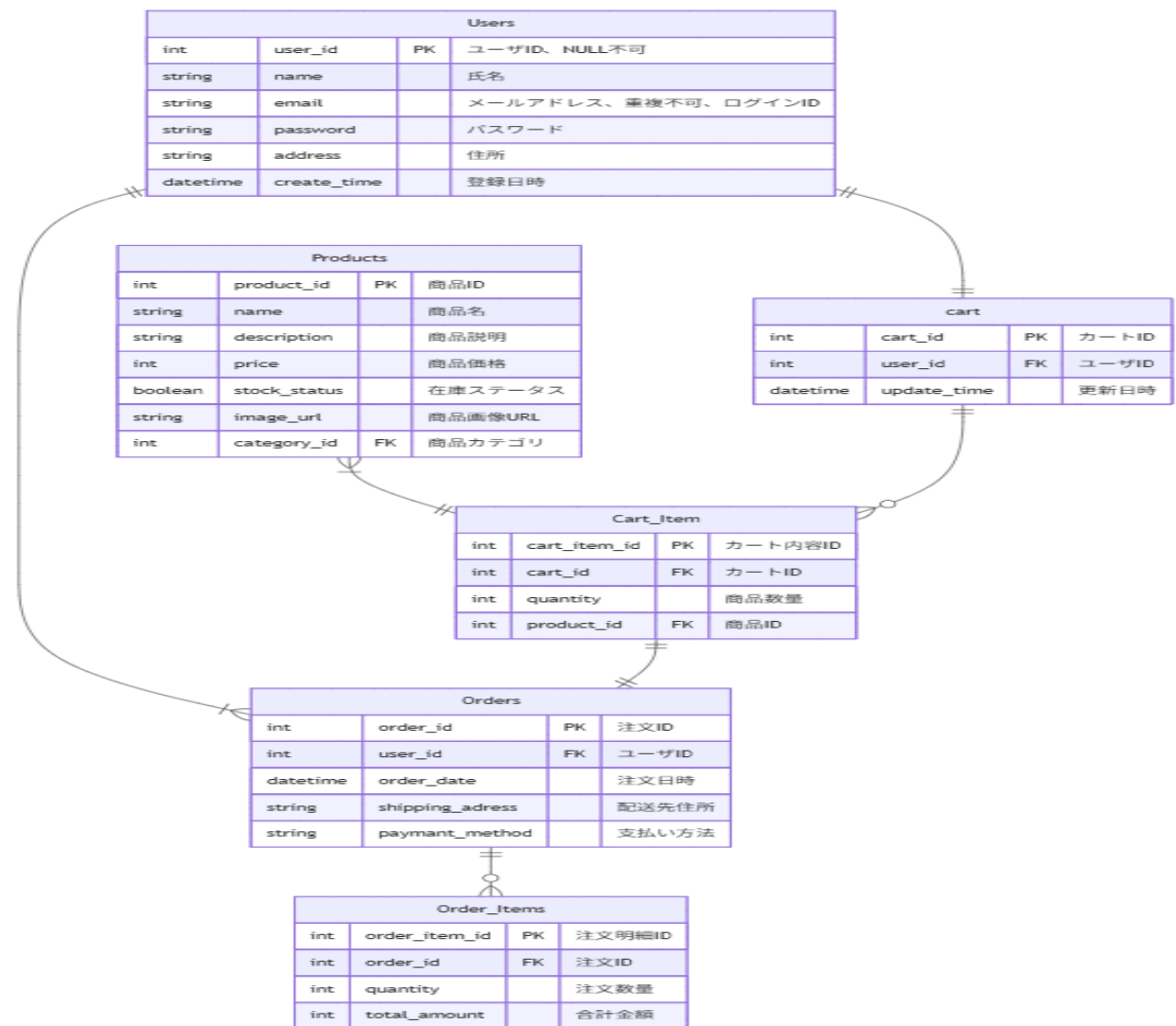
## 2-3. 主要な画面フロー（顧客画面）



## 2-3. 主要な画面フロー（管理者画面）



## 2-4. ER図 (データベース設計)





## 2-5. API設計のポイント

メソッド	エンドポイント	説明
POST	/api/users	新規会員登録
POST	/api/user/login	ログイン処理 入力認証確認
GET	/api/user/mypage	マイページ出力
POST	/api/user/logout	ログアウト処理
POST	/api/products	商品の追加（管理者）
PUT	/api/products/{productId}	商品の更新（管理者）
DELETE	/api/products/{productId}	商品の削除（管理者）

# 3テスト・品質確保の取り組み

## 実施期間

7/16~7/25（7日間）

単体テスト 7/16~7/18

結合テスト 7/22~7/23

総合テスト 7/24~7/25

## 使用ツール

JUnit

REST Client

Mockito

JMeter

# 成果(単体テスト)

- 使用ツール JUnit,Mockito
- 目的 正常に動作するシナリオ、エラーが出るシナリオ共に、プログラミングの工程で作成したシステムが期待通りに動いているかをチェック

## テスト例

```
src > test > java > com > example > simplezakka > controller > LoginControllerTest.java > {} com.example.simplezakka.controller
47 class LoginControllerTest {
75 @Nested
76 public class GetMypageTest {
77     @Nested
78     @DisplayName("GET /api/mypage")
79     class GetMypageSuccessTests {
80         @Test
81         @DisplayName("emailで登録されているユーザが存在する場合、登録情報を返す")
82         void getmypage_WhenUserExists_ShouldReturnLoginInfoWithStatusOk() throws Exception {
83             // Arrange
84             MockHttpSession mockSession = mock(classToMock:MockHttpSession.class);
85             mockSession.setAttribute("userEmail", "success@sample.com");
86             when(mockSession.getAttribute("userEmail")).thenReturn( value:"success@sample.com");
87
88
89
90             when(authService.getUserInfoByEmail(email:"success@sample.com")).thenReturn(successLogininfo);
91
92
93             // Act & Assert
94             mockMvc.perform(get("/api/user/mypage")
95                 .session(mockSession)
96                 .accept(MediaType.APPLICATION_JSON))
97                 .andExpect(status().isOk())
98                 .andExpect(content().contentType(MediaType.APPLICATION_JSON))
99                 .andExpect(jsonPath("$.name", is(successLogininfo.getName())))
100                 .andExpect(jsonPath("$.email", is(successLogininfo.getEmail())))
101                 .andExpect(jsonPath("$.password", is(successLogininfo.getPassword())))
102                 .andExpect(jsonPath("$.address", is(successLogininfo.getAddress())));
103
104
105 }
```

## 結果

テスト

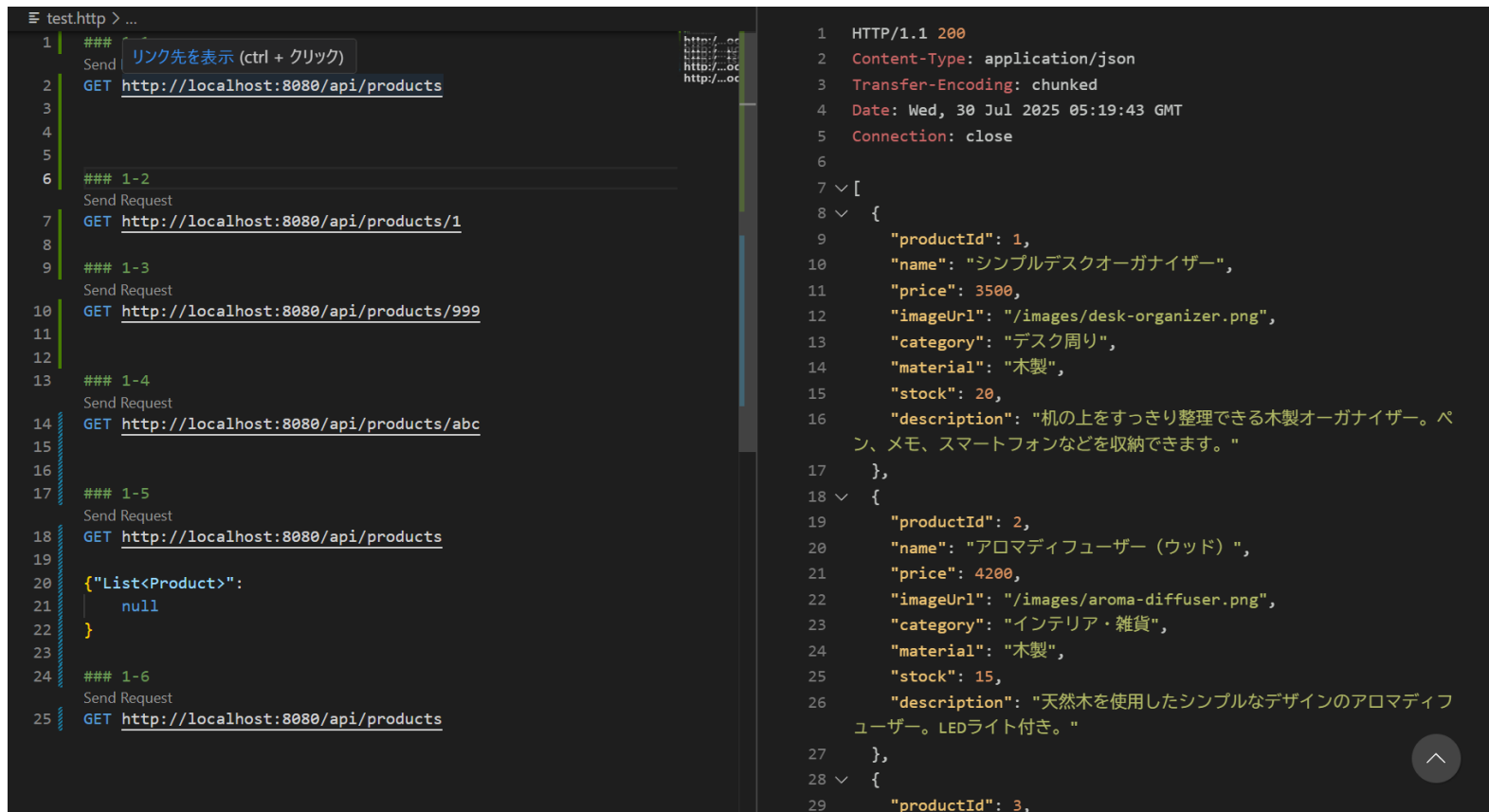
フィルター (例: テキスト、! 除外、@タグ)

1/1 8.7s

- ✓ simplezakka 4.5s
- ✓ {} com.example.simplezakka 595ms
- ✓ SimplezakkaApplicationTests 595ms
- ✓ contextLoads() 595ms
- > ✓ {} com.example.simplezakka.controller 1.1s
- > ✓ {} com.example.simplezakka.repository 777ms
- > ✓ {} com.example.simplezakka.service 2.1s

# 成果(結合テスト)

- 使用ツール REST Client
  - 目的 送信されたAPIに対しController,Service,Repositoryが連動して機能するかのテスト
- ## テスト例



```
test.http > ...
1  ### リンク先を表示 (ctrl + クリック)
2  Send
3  GET http://localhost:8080/api/products
4
5
6  ### 1-2
7  Send Request
8  GET http://localhost:8080/api/products/1
9
10 ### 1-3
11 Send Request
12 GET http://localhost:8080/api/products/999
13
14 ### 1-4
15 Send Request
16 GET http://localhost:8080/api/products/abc
17
18 ### 1-5
19 Send Request
20 GET http://localhost:8080/api/products
21
22 {"List<Product>":
23   null
24 }
25
26 ### 1-6
27 Send Request
28 GET http://localhost:8080/api/products
29
30
31 HTTP/1.1 200
32 Content-Type: application/json
33 Transfer-Encoding: chunked
34 Date: Wed, 30 Jul 2025 05:19:43 GMT
35 Connection: close
36
37 [
38   {
39     "productId": 1,
40     "name": "シンプルデスクオーガナイザー",
41     "price": 3500,
42     "imageUrl": "/images/desk-organizer.png",
43     "category": "デスク周り",
44     "material": "木製",
45     "stock": 20,
46     "description": "机の上をすっきり整理できる木製オーガナイザー。ペン、メモ、スマートフォンなどを収納できます。"
47   },
48   {
49     "productId": 2,
50     "name": "アロマディフューザー (ウッド)",
51     "price": 4200,
52     "imageUrl": "/images/aroma-diffuser.png",
53     "category": "インテリア・雑貨",
54     "material": "木製",
55     "stock": 15,
56     "description": "天然木を使用したシンプルなデザインのアロマディフューザー。LEDライト付き。"
57   },
58   {
59     "productId": 3,
```

# 成果(総合テスト)

- 使用ツール JMeter
- 目的 ブラウザにて手動で動作やユーザビリティを確認する。

## テスト例

No.	テスト対象	テスト種別	目的/観点	前提条件	操作手順
ST-SC-008	管理者ページ、商品登録シナリオ	シナリオ	管理者が商品の登録情報を変更できるか	管理者ユーザが登録されていること	1. 管理者ログイン画面で登録されたユーザ名、パスワードを入力し、ログインボタンをクリック 2. 「商品の登録」をクリック 3. 商品C (name: 商品C, price: 10000, stock: 20)を登録
ST-SC-009	商品編集シナリオ	シナリオ	管理者が商品を削除できるか	ST-SC-008 完了後	1. 「登録済み全商品の表示・編集・削除」をクリック 2. 商品Cの列の編集ボタンをクリック 3. 商品Cの登録情報を (name: 商品C, price: 5000, stock: 30)に変更
ST-SC-010	商品削除シナリオ	シナリオ	管理者が商品を新たに登録できるか	ST-SC-009 完了後	1. 「登録済み全商品の表示・編集・削除」をクリック 2. 商品Cの列の削除ボタンをクリック 3. 本番に削除しますかというメッセージをクリック

## JMeter使用例

スレッド数: 100

Ramp-Up 期間 (秒): 10

ループ回数: ☐ 無限ループ 1

☒ Same user on each iteration

Web サーバ

プロトコル: http サーバ名または IP: localhost ポート番号: 8080

HTTP リクエスト

GET バス: /api/products Content encoding:

☐ 自動リダイレクト ☒ リダイレクトに対応 ☒ KeepAlive を有効にする ☐ Use multipart/form-data ☐ Browser-compatible headers

Parameters Body Data Files Upload

リクエストで送るパラメータ:

統計レポート

名前: 統計レポート

コメント:

全てのデータをファイルに出力

ファイル名: 参照... Log/Display Only: ☐ ログエラーのみ ☐ Successes

Label	# Sam...	Average	Median	90% Li...	95% Li...	99% Li...	Min	最大値	Error %	Throu...	Receiv...	Sent K...
HTTP ...	100	861	863	1009	1017	1030	234	1036	0.00%	9.7/sec	2874.73	1.51
合計	100	861	863	1009	1017	1030	234	1036	0.00%	9.7/sec	2874.73	1.51

# 学びと価値

## 学び

- テスト工程において、プログラミングとはまた違ったコードの書き方で苦勞  
→テストコードを書くときに自分たちが書いたコードがどのように動いているかを再度考えることでコードへの理解が深まる。
- 非機能要件の大切さ  
→プログラムとして動いていてもユーザの視点から見ることで不親切な点や改善すべき点が見つかる。

## 価値

- 自分たちで作ったプログラムが想定したとおりに動いていることが確認できた。  
→このサイトの品質について保証でき、顧客に自信をもって引き渡すことができる。

## 4-1 KPIの概要

### Keep（良かったこと）

#### チームワーク・コミュニケーション

- 役割分担がうまくでき、無駄がなかった
- チーム内で助け合う雰囲気があった
- 質問を丁寧に行い、認識のズレを防げた

#### ドキュメント・設計

- コードにコメントが多く、わかりやすかった

#### 技術・理解

- 自力で調べながら進められた
- 機能を一貫して担当し、理解が深まった
- 開発全体の流れをつかめた

#### 進行・スケジュール管理

- 進捗に応じて柔軟に進められた
- gitの運用ルールでトラブルを回避できた

## 4-1 KPIの概要

### Problem（課題）

#### チームワーク・進行

- 情報共有が不足し、作業重複があった
- モチベーション差による作業偏りがあった
- 進捗の共有ができていなかった

#### スケジュール・管理

- 日程が詰まり、作業を諦めた部分があった
- 提出が遅れた
- ファイル混同で無駄が発生

#### 技術・理解

- Git・GitHubの習熟に時間がかかった
- 担当外の理解が浅かった
- 非機能要件があまり考慮されていなかった
- 最初の要件定義の頃にAIに頼りすぎて、後半で整合性が取れていない場面があった



## 4-1 KPIの概要

### Try（次に向けて）

#### チームワーク・共有

- 進捗共有の機会を増やす
- 参考資料やコードを積極的に共有する

#### スケジュール・管理

- 日程に合わせた要件定義を行う
- WBSなどで進捗を見える化する

#### ツール活用

- 使用ツールのマニュアルを整備する
- AIの提案は検証して採用する

#### 技術・設計

- 機能ごとに簡単なテストを行う
- 設計段階で修正が出ないよう精度を上げる
- 非機能要件（ログ・セキュリティ）にも対応する

#### 機能改善

- ログイン・マイページ機能の充実
- 管理者機能の強化
- わかりやすいエラー表示
- 妥協せず品質を追求する

## 4-2 全体を振り返って

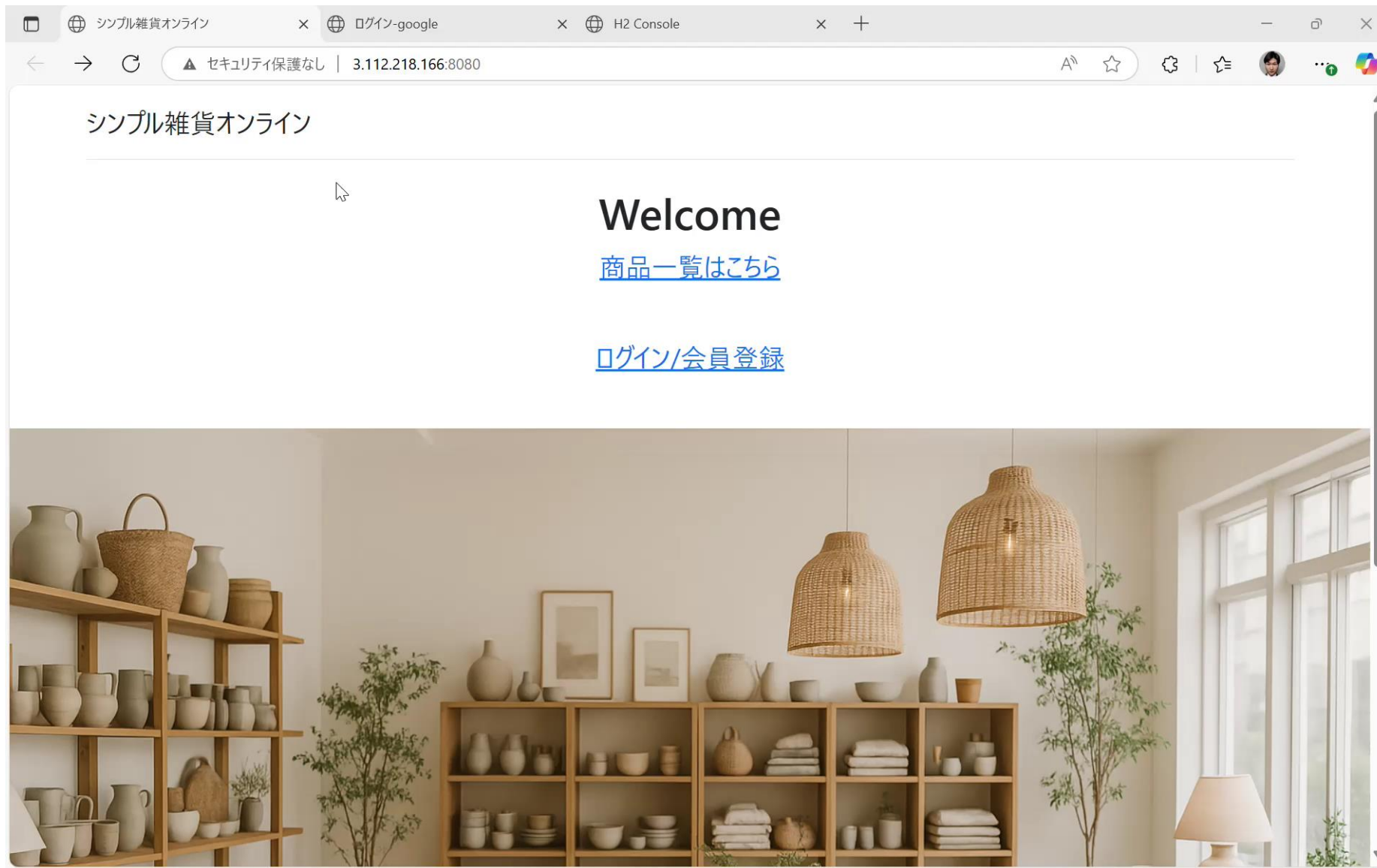
### 特に印象に残った成功体験や困難だった点

- バックエンドと繋がった瞬間  
(会員情報や管理者画面で行った登録がDBに反映されたこと)
- HTTPセッションでログイン状態を保持できたこと

### チームビルディングやプロジェクト管理で工夫した点

- 2人×3組でコードを作成し、役割分担がうまくできた。
- 機能ごとに分担することで、全員が開発の一連の流れを経験でき、知識が深まった。
- 全員がリーダー＆発表する機会を半強制的に持つことで、全体を理解しようとする姿勢があった。

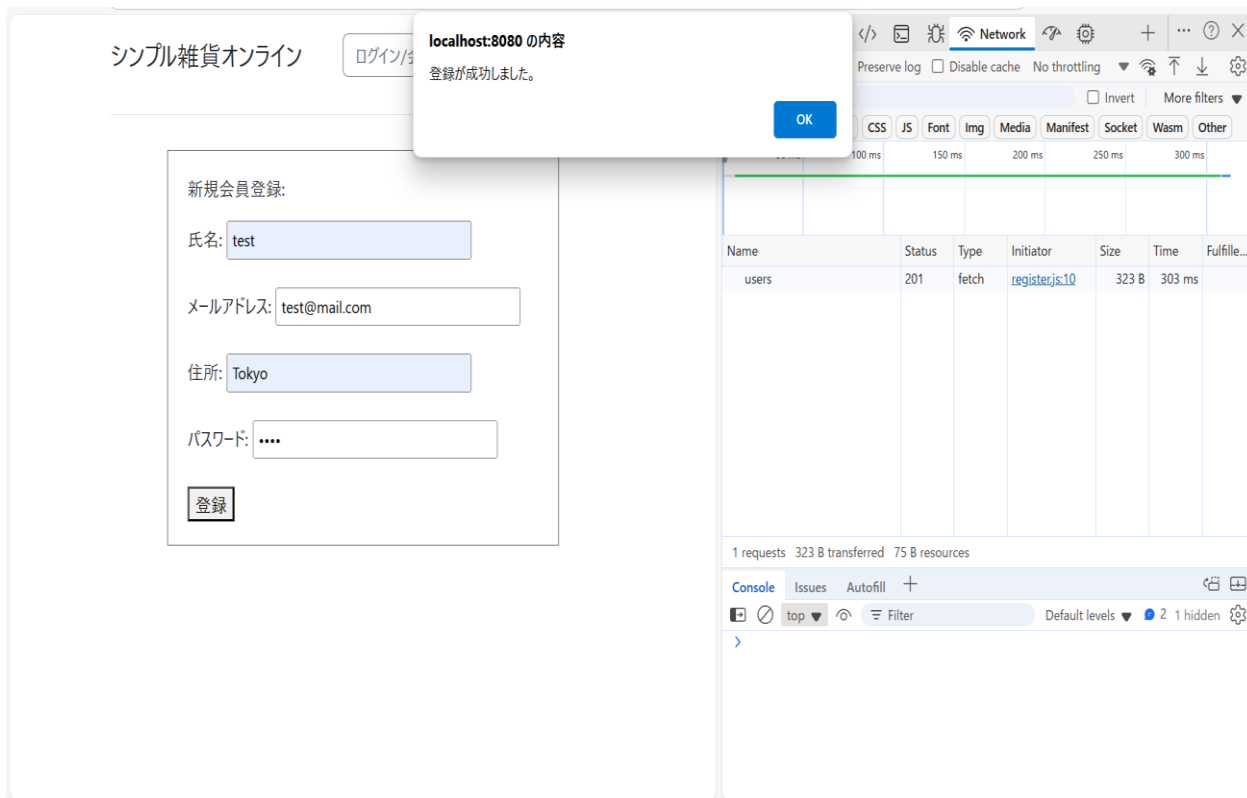
## 5 デモ画面



## 6. 今後の展望・改善提案

- ユーザー認証

現時点では会員登録からログインしてマイページに登録した住所や名前が表示されるだけにとどまっているが、注文の簡便化（自動入力）などを行いたい。



## 6. 今後の展望・改善提案

- 決済機能追加

現時点では決済機能として現金決済（代引き）しか実装されていないのは有用性が薄い  
ため、クレジットカード決済やバーコード決済などを支払方法に追加したい

The screenshot displays a web application interface. In the background, there is a product catalog with items like 'シンプル雑貨' (Simple Goods) and 'シンプルデスクオーガ' (Simple Desk Organizer) with prices ranging from ¥3,500 to ¥5,800. Overlaid on this is a modal window titled 'お客様情報入力' (Customer Information Input). The form contains the following fields:

- お名前 (Name): test
- メールアドレス (Email Address): test@mail.com
- 住所 (Address): Tokyo
- 電話番号 (Phone Number): 000-0000-000

At the bottom of the modal, there are three icons representing different payment methods: a bank building (代引き), a credit card (クレジットカード), and a stack of coins (現金). To the right of these icons are two buttons: 'キャンセル' (Cancel) and '注文を確定する' (Confirm Order).

## 6.今後の展望・改善提案

- 商品の在庫切れ機能

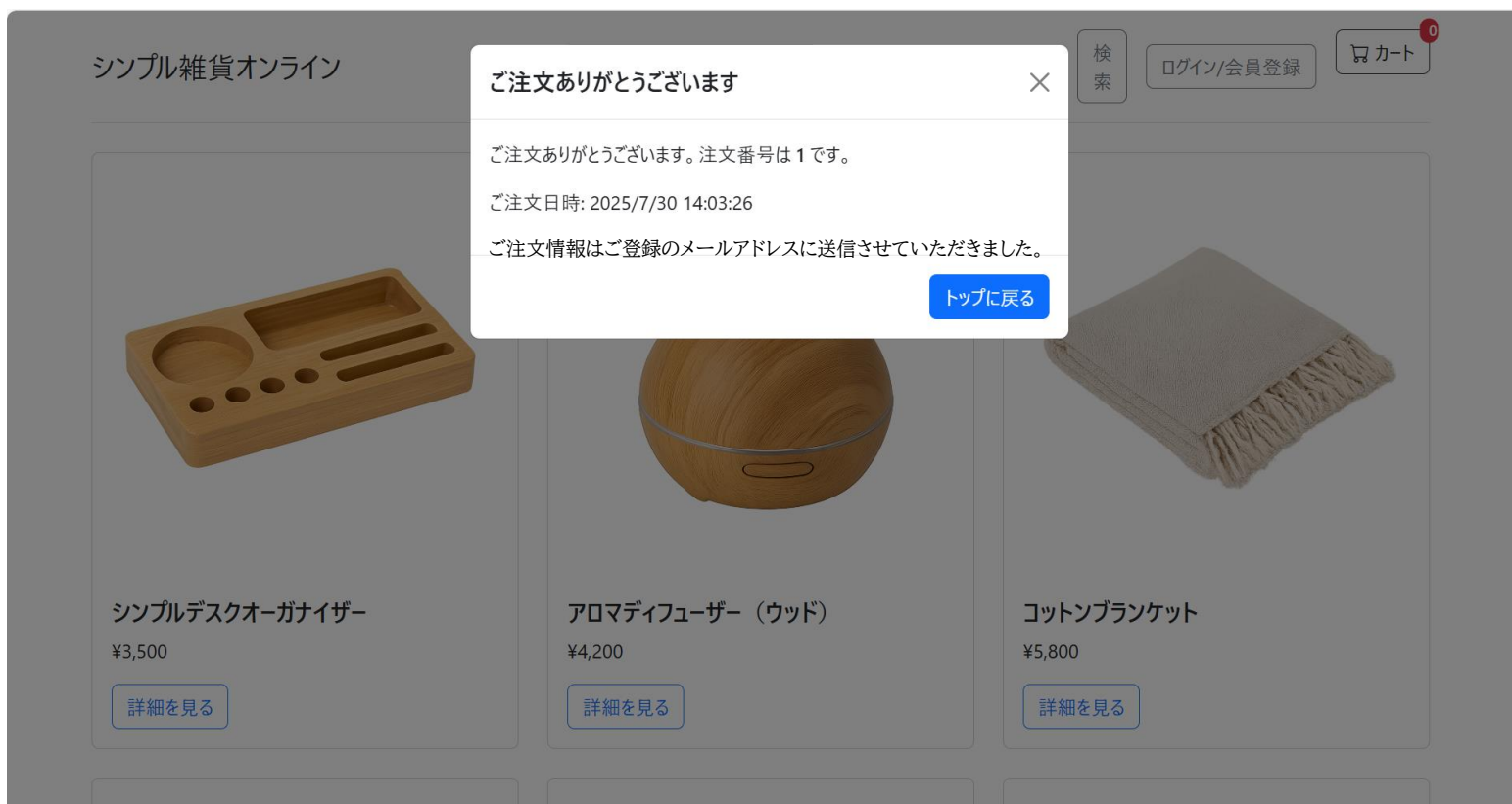
在庫以上の注文にエラーが起きるタイミングが注文情報を送信する際にはじかれる仕組みになってしまっており在庫以上の数量の商品をカートに入れることができてしまっている。そのためカート追加の段階でエラーが起きる仕組みをしたい。



## 6. 今後の展望・改善提案

- 注文情報のメール送信

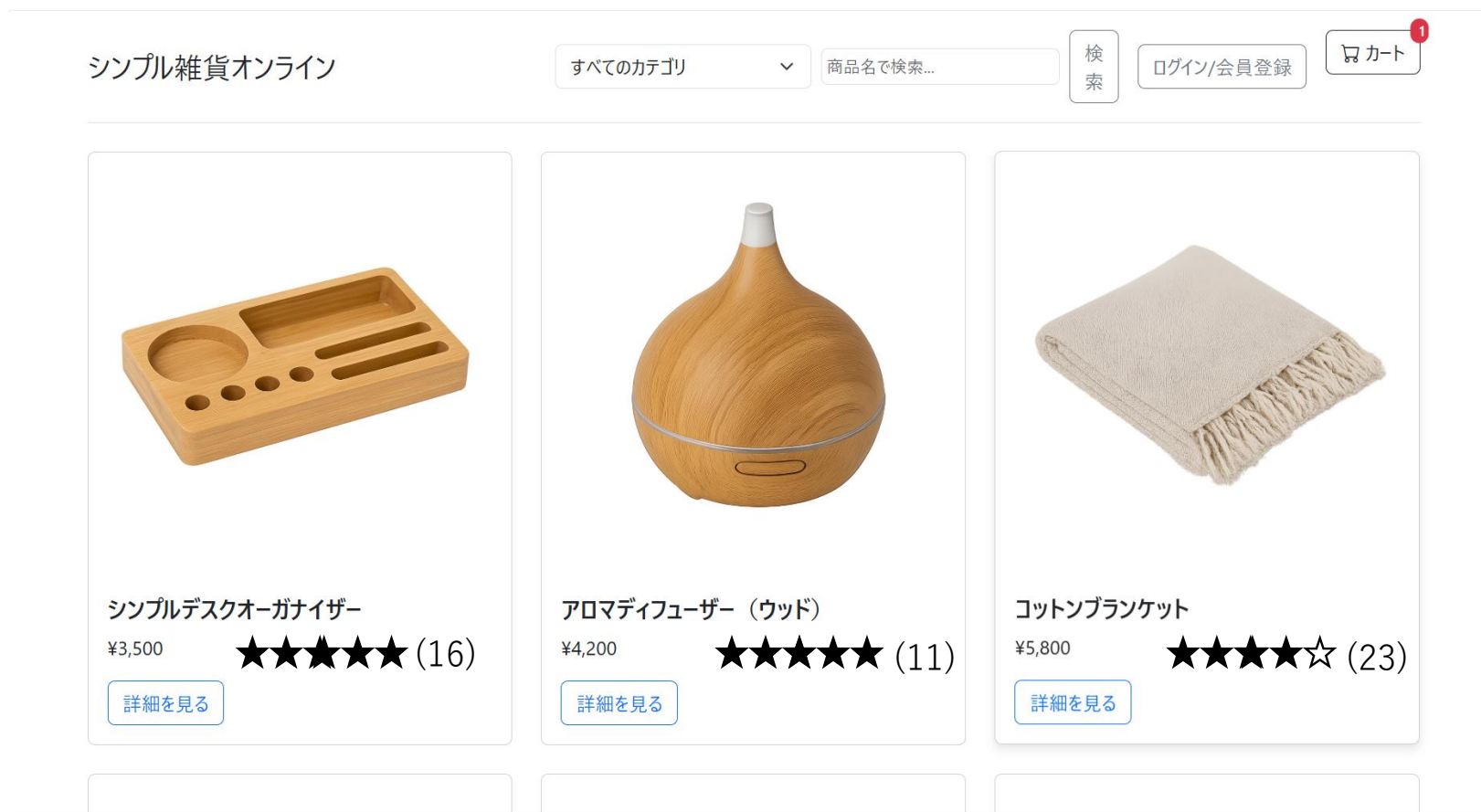
注文内容の確認や注文情報の共有がユーザーに行われていないため、注文確定した時点でメールを送信したい。また会員の注文ではマイページに注文情報も記載したい。



## 6. 今後の展望・改善提案

- 口コミ、レビュー機能

シンプルながら顧客満足度が高いということをデータとして可視化されている環境を作りたい。そのため、各商品に対して口コミやレビュー機能を実装したい。





## 6.今後の展望・改善提案

- 注文履歴閲覧機能（管理者）  
注文内容の確認や注文情報の共有がユーザーに行われていないため、注文確定した時点でメールを送信したい。また会員の注文ではマイページに注文情報も記載したい。

### 注文履歴

ORDER_ID	CUSTOMER_NAME	CUSTOMER_EMAIL	SHIPPING_ADDRESS	SHIPPING_PHONE_NUMBER	TOTAL_AMOUNT	ORDER_DATE	STATUS	CREATED_AT	UPDATED_AT
1	test	test@mail.com	Tokyo	000-0000-000	45500.00	2025-07-30 14:03:26.894252	PENDING	2025-07-30 14:03:26.949189	2025-07-30 14:03:26.949189
33	test2	test2@mail.com	Osaka	000-0000-001	12600.00	2025-07-30 14:26:51.543432	PENDING	2025-07-30 14:26:51.604161	2025-07-30 14:26:51.604161
34	test3	test3@mail.com	Kyoto	000-0000-002	17500.00	2025-07-30 14:27:30.38473	PENDING	2025-07-30 14:27:30.385463	2025-07-30 14:27:30.385463
35	test4	test4@mail.com	Gunma	000-0000-003	3200.00	2025-07-30 14:28:10.885533	PENDING	2025-07-30 14:28:10.888961	2025-07-30 14:28:10.888961
36	test5	test5@mail.com	Okinawa	000-0000-004	97200.00	2025-07-30 14:29:08.562357	PENDING	2025-07-30 14:29:08.566098	2025-07-30 14:29:08.566098

## 6. 今後の展望・改善提案

- ・ 今回発見された課題に対する具体的な改善策

1. チームワーク・進行

WBSなどで進捗が見える化する。（進捗共有の機会を増やすため）

要件定義や詳細定義を細分化して行いバッファを作る。（スケジュール管理するため）

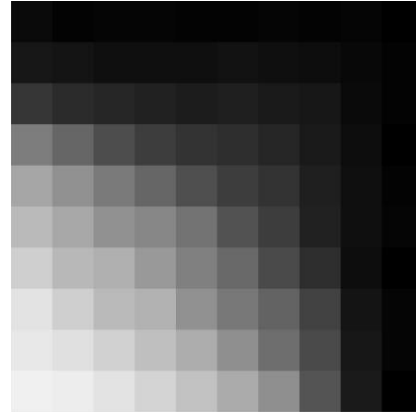
2. 技術理解

Git・GitHubに関する資料を作成し、共有する

担当箇所をその都度進捗共有し、メンバーに共有する。

AIは極力使用を控え、またAIが出力した内容をそのままコピー＆ペーストせず内容理解に励む

## 7. 質疑応答



AVANT

SUSTAINABILITY IS VALUE