JAVA

JAVA上級WEBアプリ開発 実践テキスト

Spring Boot
フリマアプリ風システム

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum finibus orci sit amet quam commodo, mollis pellentesque sapien aliquam. Nunc ut sapien vel augue congue egestas et vel neque. Sed faucibus ornare ultricies.

目　次

10.各ソースコード（解説・説明付き）

フリマアプリ風システム

1.開発方針

項目	内容
サーバーサイド	Java (Spring Boot, Maven)
フロントエンド	HTML / CSS（レスポンシブ） / JavaScript (Vue.js 推奨)
データベース	PostgreSQL
認証	Spring Security (JWT)
外部連携	Stripe API（決済）、Cloudinary（画像管理）、LINE 通知 API
バージョン管理	GitHub
デプロイ想定	AWS (Elastic Beanstalk / S3)

2.要件定義

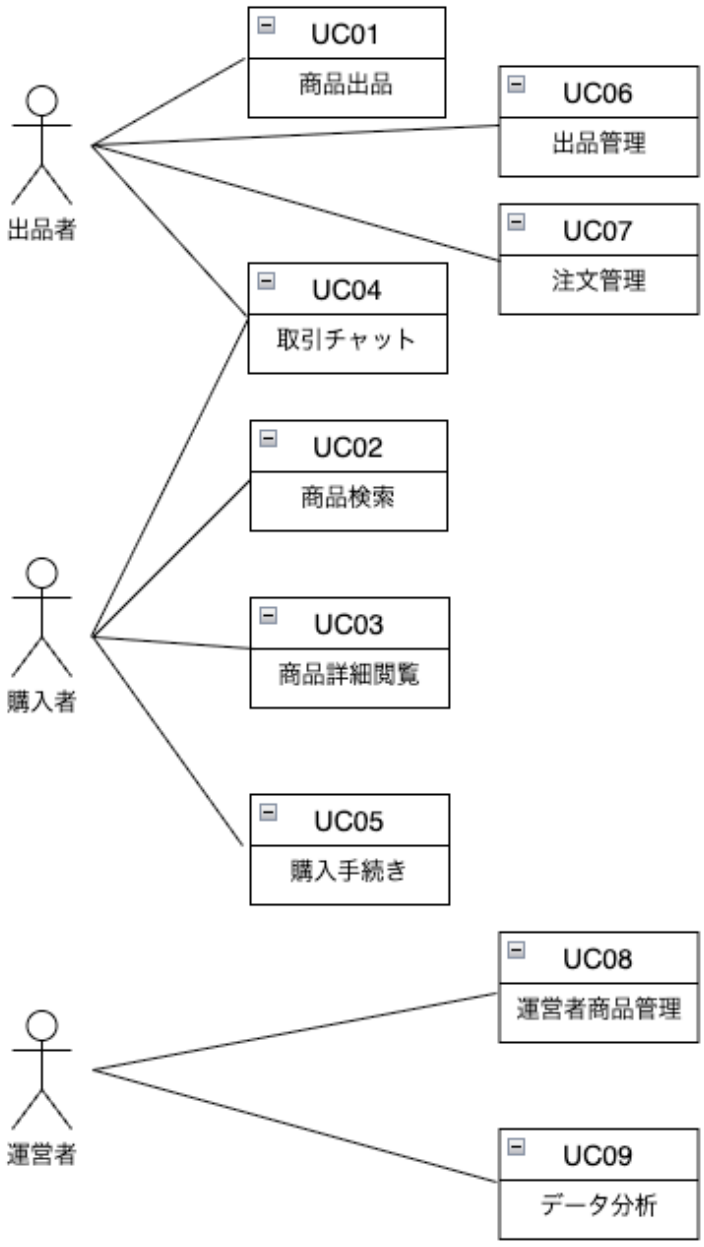
2.1 ユーザーストーリー（テーマ）

- **出品者**
  - 不要な商品を簡単に出品したい
  - 質問にすぐ答えたい
  - 売れたらすぐ通知が欲しい
- **購入者**
  - 商品を検索・比較したい
  - 安心して取引したい
  - 質問・交渉をしたい
  - 決済を安全に済ませたい
- **運営者**
  - 商品やユーザを管理したい
  - トラブル対応や違反商品削除を行いたい
  - 売上データを把握したい

2.2 ユースケース記述

UC-ID	ユースケース名	アクター	概要	事前条件	基本フロー	代替フロー
UC01	商品出品	出品者	商品情報を登録し出品する	ログイン済み	①出品画面を開く → ②商品情報入力 → ③画像アップロード → ④登録	出品途中で保存可能
UC02	商品検索	購入者	商品を条件で検索する	–	①検索フォーム入力 → ②商品一覧表示	並べ替え、絞り込み
UC03	商品詳細閲覧	購入者	商品詳細を確認する	–	①商品一覧から選択 → ②詳細表示	なし
UC04	取引チャット	購入者/出品者	商品の取引に関するチャットを行う	ログイン済み	①商品詳細画面 → ②チャット送信	画像送信対応
UC05	購入手続き	購入者	商品を決済する	ログイン済み	①購入ボタン押下 → ②決済画面 → ③Stripe 決済 → ④完了通知	決済キャンセル
UC06	出品管理	出品者	自分の出品状況を確認・編集する	ログイン済み	①出品一覧表示 → ②編集 or 削除	なし
UC07	注文管理	出品者	自分の商品が売れた注文を管理する	ログイン済み	①注文一覧表示 → ②発送処理 → ③発送完了通知	発送キャンセル
UC08	運営者商品管理	管理者	違反商品や出品者を管理する	ログイン済み	①商品検索 → ②削除 or 停止	なし
UC09	データ分析	管理者	売上や取引数の統計を確認する	ログイン済み	①条件指定 → ②グラフ表示 → ③CSV 出力	なし

2.3 ユースケース図

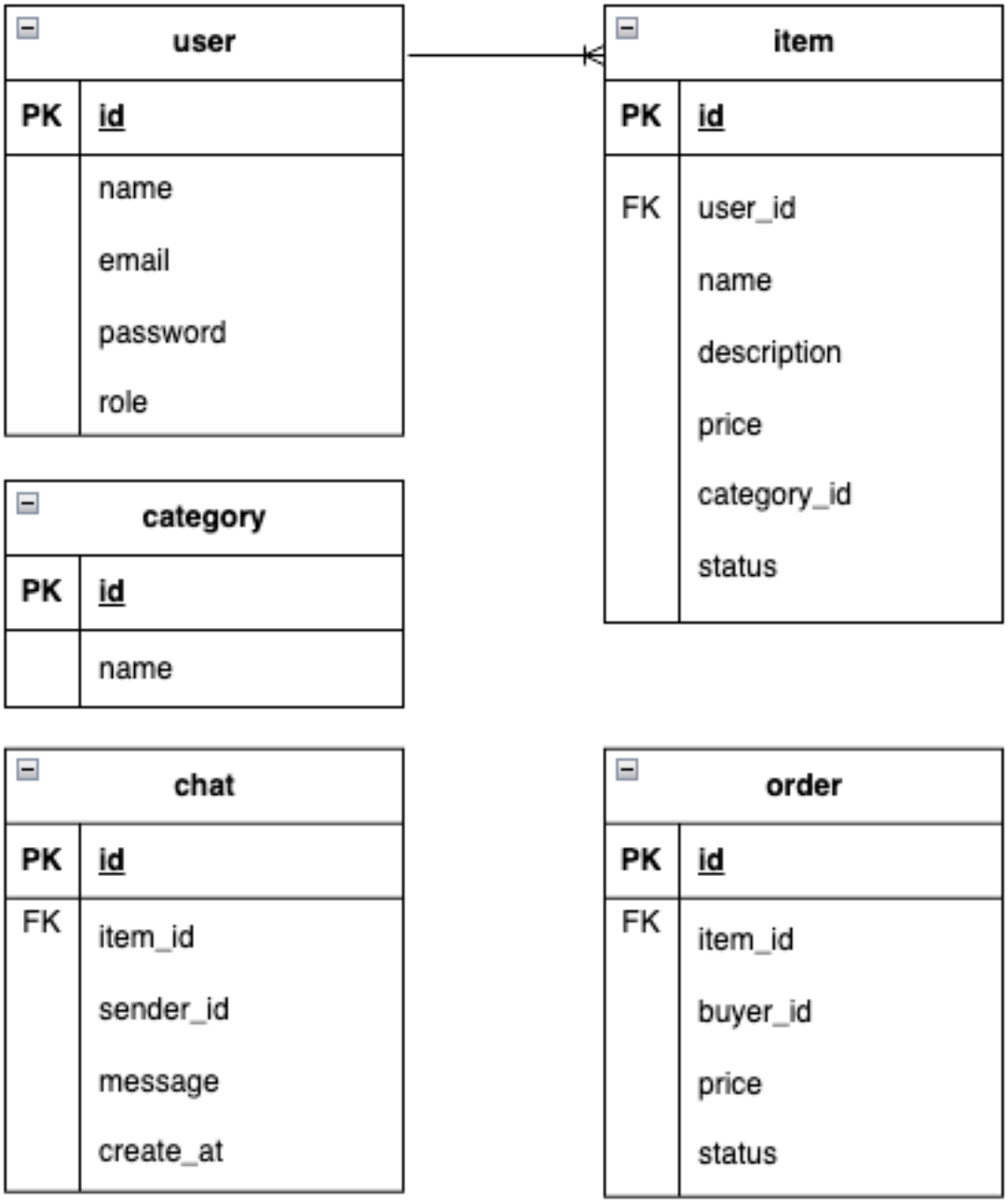


2.4 用語辞典

用語	意味
出品	商品を販売目的で登録すること
購入	商品を買う行為
Stripe	クレジット決済サービス
Cloudinary	画像アップロード&CDN サービス
チャット	取引メッセージ
違反商品	規約違反で削除対象の商品
JWT	JSON Web Token（認証方式）

3. 外部設計

3. 1ER 図


user
PK
id
name
email
password
role
item
PK
id
FK
user_id
name
description
price
category_id
status
category
PK
id
name
chat
PK
id
FK
item_id
sender_id
message
create_at
order
PK
id
FK
item_id
buyer_id
price
status

3.2 テーブル設計書

user

項目名	型	制約	備考
id	serial	PK	
name	varchar(50)	not null	
email	varchar(255)	unique	
password	varchar(255)	not null	bcrypt ハッシュ
role	varchar(20)	not null	USER / ADMIN

item

項目名	型	制約	備考
id	serial	PK	
user_id	int	FK	user.id
name	varchar(255)	not null	
description	text		
price	numeric(10,2)	not null	
category_id	int	FK	category.id
status	varchar(20)		出品中 / 売却済

category

項目名	型	制約	備考
id	serial	PK	
name	varchar(50)	not null	

order

項目名	型	制約	備考
id	serial	PK	
item_id	int	FK	item.id
buyer_id	int	FK	user.id
price	numeric(10,2)	not null	
status	varchar(20)		購入済 / 発送済

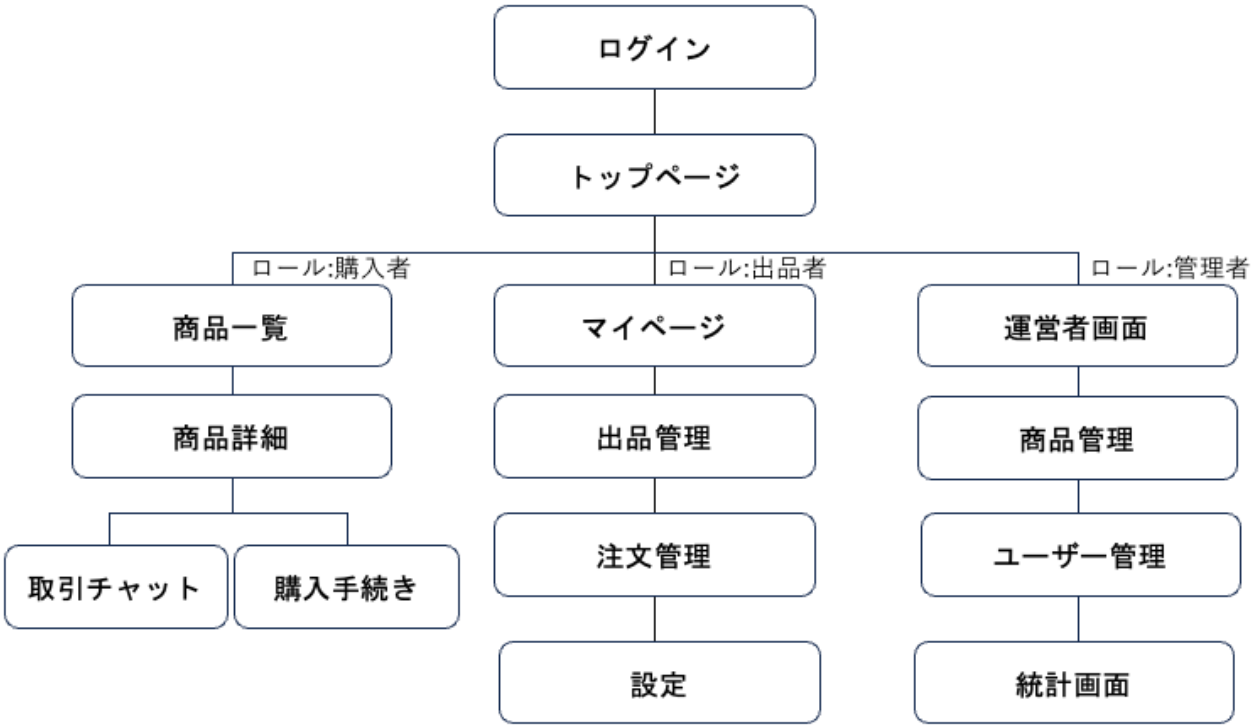
chat

項目名	型	制約	備考
id	serial	PK	
item_id	int	FK	item.id
sender_id	int	FK	user.id
message	text		
created_at	timestamp		

3.3 画面レイアウト設計書

例：商品詳細画面

3.4 画面遷移図



4. 内部設計

4.1 クラス図

User		
	クラス名	説明
1	id	ユーザーの主キー（PK）/連番
2	name	表示名。プロフィールに出す想定/必須（～50 文字目安）
3	email	ログイン用メールアドレス/必須・一意（UNIQUE）
4	password	パスワードのハッシュ（bcrypt）/生パスワードは保存しない
5	role	権限区分/USER（出品者/購入者を含む） / ADMIN（運営者）

Item		
	クラス名	説明
1	id	商品の主キー（PK）
2	userId	出品者のユーザーID（FK → User.id）
3	name	商品名（必須、～255 文字目安）
4	description	商品説明（任意、長文可）
5	price	価格。購入時の税扱い方針に合わせて小数 2 桁（例：numeric(10,2)）
6	categoryId	カテゴリ ID（FK → Category.id）
7	status	出品状態。例：出品中 / 売却済（必要に応じて下書きなど追加可）

Order		
	クラス名	説明
1	id	注文の主キー（PK）
2	itemId	対象商品の ID（FK → Item.id）
3	buyerId	購入者のユーザーID（FK → User.id）
4	price	決済金額のスナップショット（当時の価格を保持）
5	status	注文状態。例：購入済 / 発送済（運用でキャンセルなどを拡張可）

Chat		
	クラス名	説明
1	id	チャットメッセージの主キー（PK）
2	itemId	対象商品の ID（FK → Item.id）
3	senderId	送信者のユーザーID（FK → User.id）
4	message	メッセージ本文（必須、改行可）
5	createdAt	送信日時（タイムスタンプ。UTC 保存＋表示時に JST 変換推奨）

4.2 クラス設計書

User

項目	内容
Entity 名	User
属性	id, name, email, password, role
関連	Item (1:N), Order (1:N), Chat (1:N)

Item

項目	内容
Entity 名	Item
属性	id, userId, name, description, price, categoryId, status
関連	User (N:1), Order (1:N), Chat (1:N)

Order

項目	内容
Entity 名	Order
属性	id, itemId, buyerId, price, status
関連	User (N:1), Item (N:1)

Chat

項目	内容
Entity 名	Chat
属性	id, itemId, senderId, message, createdAt
関連	User (N:1), Item (N:1)

4.3URL マッピング

URL	メソッド	機能
/login	GET	ログイン画面表示
/login	POST	認証処理（JWT 発行）
/items	GET	商品一覧表示
/items/new	GET	出品画面表示
/items	POST	出品登録
/items/{id}	GET	商品詳細表示
/items/{id}/edit	GET	出品編集画面
/items/{id}	POST	出品編集保存
/items/{id}/delete	POST	出品削除
/orders	GET	注文一覧表示
/orders/{id}/ship	POST	発送処理
/chat/{itemId}	GET	チャット画面
/chat/{itemId}	POST	チャット送信
/admin/items	GET	運営者：商品管理画面
/admin/users	GET	運営者：ユーザ管理画面
/admin/statistics	GET	運営者：統計画面
/admin/statistics/csv	GET	統計データ CSV 出力

5.テスト

5.1 単体テスト

- UserRepository
  - findByEmail
  - save
- ItemRepository
  - findByCategory
  - save
- OrderRepository
  - findByBuyerId
  - save
- ChatRepository
  - findByItemId
  - save

5.2 結合テスト

- ログイン → JWT 認証フロー
- 商品出品 → 商品詳細表示
- 購入処理 → Stripe 決済検証
- チャット → メッセージ送受信検証
- 運営者 CSV 出力 → データ内容検証

6.セキュリティ設計

- bcrypt によるパスワードハッシュ
- JWT 認証（トークン失効対応）
- 権限管理（Spring Security）
- 入力バリデーション
- ログ監査（更新履歴記録）

7.外部連携

- Stripe API
  - 決済フロー
- Cloudinary API
  - 商品画像アップロード&CDN 配信
- LINE Notify API
  - 売却通知、メッセージ通知

8.開発順序

8.1プロジェクト初期化

・Maven/Spring Boot最小構成、application.properties（DB・Thymeleaf・ロギング）作成
・依存関係だけ入れる：web/thymeleaf/validation/jpa/postgresql/lombok/security（まだ緩め）/test

8.2DB準備

・Schema.sql（DDL）→ 起動時自動適用
・data.sql（初期データ。ADMIN 1人は BCrypt 済み）
・起動して /actuator/health かログで DB 接続確認

8.3エンティティ（外部キーに依存しない順）

User → Category → Item → AppOrder → Chat → FavoriteItem → Review

8.4リポジトリ

UserRepository → CategoryRepository → ItemRepository → AppOrderRepository → ChatRepository → FavoriteItemRepository → ReviewRepository

8.5最低限の画面土台

・LoginController（GET だけ）／templates/login.html
・DashboardController（トップ遷移だけ）
・style.css（ビルド/表示確認用の最低限）

8.6 SecurityConfig（最小）

・BCrypt、/css/と/loginと/items/は許可、他は認可
・フォームログイン（/login）、ログアウト設定
・ここでログイン画面まで動作確認

8.7サービス層（安全な順）

・CategoryService / UserService（CRUD・検索）
・ItemService（まずは一覧/詳細の読み取りだけ実装）
・FavoriteService（exists チェックだけ先に）
・ChatService（取得だけ先に）
・ReviewService（取得/平均算出だけ先に）
・AppOrderService（読み取り系・統計の枠だけ。Stripe連携は後回し）

・CloudinaryService/LineNotifyService/StripeService は**スタブ**を先に置く（例：未実装は例外 or ダミー返し）

8.8 公開ページの縦スライス①（読み取りのみ）

・ItemController：/items（一覧・検索）、/items/{id}（詳細）、item_list.html / item_detail.html
→ ここまでで “未ログインでも閲覧 OK” を安定稼働

8.9 認証後ページの土台

・UserController：/my-page だけ表示
・my_page.html（リンクがダミーでも OK）

8.10 出品フロー（書き込み）

・ItemService に保存/編集/削除（画像はまだローカル URL か空で OK）
・ItemController に新規/編集/削除を追加
・item_form.html / seller_items.html
→ フォーム→保存→一覧の往復がエラーなくなる

8.11 お気に入り

・FavoriteService（追加/解除/一覧/exists 完実装）
・ItemController にお気に入り操作追加
・my_favorites.html で表示

8.12 チャット（通知なしで先に）

・ChatService（投稿/取得 完実装）
・ChatController（GET/POST）
・item_detail.html のチャット UI 連携
→ この時点では LINE 通知はスタブ

8.13 レビュー

・ReviewService（本人性/重複チェックを実装）
・ReviewController（フォーム表示/投稿）
・review_form.html / user_reviews.html

8.14 注文（Stripe なしの素振り）

・AppOrderService：在庫確定・状態遷移（購入済/発送済）だけ先に
・AppOrderController：
一覧/発送操作（seller_app_orders.html/buyer_app_orders.html）

8.15 tripe 連携を導入

・StripeService：PaymentIntent 作成/取得/確認 を実装
・AppOrderService：paymentIntentId で**厳密に**ひも付ける
・AppOrderController：購入開始→payment_confirmation.html→完了
・Security：Webhook を除外（必要なら）、CSRF の扱い最終調整

8.16 Cloudinary 導入

・CloudinaryService 実装、ItemService の画像保存/差替えに組込み
・URL→public_id の抽出ユーティリティを用意（削除用）

8.17 LINE Notify 導入

・LineNotifyService 実装（失敗は WARN ログに）
・ChatService／AppOrderService の通知ポイントに組込み

8.18 管理者機能

・AdminController：商品/ユーザ管理・統計・CSV
・admin_* テンプレート一式（ダッシュボード/商品/ユーザ/統計）

8.19 例外/検証/安定化

・@ControllerAdvice（例外共通化）、@Valid 入力検証、@Transactional 付与
・ログ、監査、アクセス制御（@PreAuthorize）の仕上げ
・CSRF/セッション/ヘッダの最終見直し

8.20 テスト整備

Repository 単体 → Service 単体（スタブ外部連携） → Controller（MockMvc）
重要ユースケースの結合テスト：ログイン→出品→購入→レビュー

8.21 エラーの出にくい実装手順

1. schema / data / style
2. User, Category, Item, AppOrder, Chat, FavoriteItem, Review
3. UserRepository → CategoryRepository → ItemRepository → AppOrderRepository → ChatRepository → FavoriteItemRepository → ReviewRepository
4. SecurityConfig, LoginController, DashboardController
5. ItemService → ItemController → item_list / item_detail / item_form / seller_items
6. FavoriteService → my_favorites
7. ChatService → ChatController
8. ReviewService → ReviewController → review_form / user_reviews

9. AppOrderService → AppOrderController → buyer_app_orders / seller_app_orders → StripeService → payment_confirmation
10. CloudinaryService / LineNotifyService
11. AdminController → admin_dashboard / admin_items / admin_users / admin_statistics

9.ディレクトリ構成（1/2）

```
flea-market-system-complete/
  └ src/
     └ main/
        └ java/
            └ com/
                └ example/
                   └ fleamarketsystem/
                      ├ congig/
                      |    └ SecurityConfig.java
                      ├ controller/
                      |    ├ AdminController.java
                      |    ├ AppOrderController.java
                      |    ├ ChatController.java
                      |    ├ DashboardController.java
                      |    ├ ItemController.java
                      |    ├ LoginController.java
                      |    ├ ReviewController.java
                      |    └ UserController.java
                      ├entity/
                      |    ├ AppOrder.java
                      |    ├ Category.java
                      |    ├ Chat.java
                      |    ├ FavoriteItem.java
                      |    ├ Item.java
                      |    ├ Review.java
                      |    └ User.java
                      ├repository/
                      |    ├ AppOrderRepository.java
                      |    ├ CategoryRepository.java
                      |    ├ ChatRepository.java
                      |    ├ FavoriteItemRepository.java
                      |    ├ ItemRepository.java
                      |    ├ ReviewRepository.java
                      |    └ UserRepository.java
                      └service/
                          ├ AppOrderService.java
                          ├ CategoryService.java
                          ├ ChatService.java
                          ├ CloudinaryService.java
                          ├ FavoriteService.java
                          ├ ItemService.java
                          ├ LineNotifyService.java
                          ├ ReviewService.java
                          ├ StripeService.java
                          └ UserService.java
```

（続き）ディレクトリ構成（2/2）

```
flea-market-system-complete/
  └ src/
     └ main/
        ├ java/
        └ resources/
            ├ static/
            |   └ css/
            |       └ style.css
            ├ templates/
            |   ├ admin_dashboard.html
            |   ├ admin_items.html
            |   ├ admin_statistics.html
            |   ├ admin_users.html
            |   ├ buyer_app_orders.html
            |   ├ item_detail.html
            |   ├ item_form.html
            |   ├ item_list.html
            |   ├ login.html
            |   ├ my_favorites.html
            |   ├ my_page.html
            |   ├ payment_confirmation.html
            |   ├ review_form.html
            |   ├ seller_app_orders.html
            |   ├ seller_items.html
            |   └ user_reviews.html
            |
            ├ application.properties
            ├ data.sql
            └schema.sql
```

10.各ソースコード（解説・説明付き）

10.1 パッケージ: config

SecurityConfig.java（BCrypt・Ant パターン・MethodSecurity・Webhook の CSRF 除外）

```
// Spring の設定クラスであることを示す
package com.example.fleamarketsystem.config;

// UserRepository を使ってユーザを読み出すための import
import com.example.fleamarketsystem.repository.UserRepository;
// Bean 定義や設定用アノテーションの import
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
// HTTP セキュリティを構築するための import
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
// WebSecurity を有効化するアノテーション
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
// メソッドレベルの認可アノテーション(@PreAuthorize 等)を有効化
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
// ユーザ詳細を提供するための型
import org.springframework.security.core.userdetails.UserDetailsService;
// ユーザが見つからないときの例外
import org.springframework.security.core.userdetails.UsernameNotFoundException;
// 安全なパスワードエンコーダ（BCrypt）を使うための import
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
// PasswordEncoder インターフェースの import
import org.springframework.security.crypto.password.PasswordEncoder;
// セキュリティフィルタチェーンの型
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

    // セキュリティの主要設定（エンドポイント保護 / 認証 / ログアウト / CSRF 例外）
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        // HttpSecurity ビルダーに対して設定を積み上げる
        http
            // 認可ルールの設定
            .authorizeHttpRequests(authorize -> authorize
                // ログイン画面や静的ファイル、商品一覧・詳細は匿名アクセス許可
                .requestMatchers("/login", "/css/**", "/js/**", "/images/**",
"/items/**").permitAll()
                // Stripe の Webhook は署名検証で守るため認可は permitAll（後で CSRF は除外）
                .requestMatchers("/orders/stripe-webhook").permitAll()
                // /admin 配下は ADMIN ロールのみ
                .requestMatchers("/admin/**").hasRole("ADMIN")
```

```
            // それ以外は認証必須
            .anyRequest().authenticated()
        )
        // フォームログインの設定
        .formLogin(form -> form
            // ログインページのパスを指定
            .loginPage("/login")
            // 成功時は商品一覧へリダイレクト
            .defaultSuccessUrl("/items", true)
            // ログインページ自体は誰でも表示可能
            .permitAll()
        )
        // ログアウト設定
        .logout(logout -> logout
            // ログアウトURL
            .logoutUrl("/logout")
            // 成功時はログイン画面へ
            .logoutSuccessUrl("/login?logout")
            // 誰でも呼べる
            .permitAll()
        )
        // CSRFの設定（基本有効、Stripe Webhookのみ除外）
        .csrf(csrf -> csrf
            // Antパターンで Webhookを除外
            .ignoringRequestMatchers("/orders/stripe-webhook")
        );

    // 構築したフィルタチェーンを返す
    return http.build();
}

// DBからユーザをロードして Spring Securityの UserDetailsに変換する
@Bean
public UserDetailsService userDetailsService(UserRepository userRepository) {
    // email（=username）で検索し、見つかれば UserDetailsを組み立てる
    return email -> userRepository.findByEmail(email)
            // Mapでアプリの Userを Springの Userに詰め替える
            .map(user -> org.springframework.security.core.userdetails.User.builder()
                    // ユーザ名はメール
                    .username(user.getEmail())
                    // パスワード（BCryptハッシュ前提）
                    .password(user.getPassword())
                    // ロールは"ADMIN"や"USER"を渡せば自動で"ROLE_"が付与される
                    .roles(user.getRole())
                    // 有効/無効のフラグを反映
                    .disabled(!user.isEnabled())
                    // Builderを閉じて UserDetailsを作成
                    .build())
```

```
                // 見つからない場合は例外
                .orElseThrow(() -> new UsernameNotFoundException("User not found: " + email));
    }

    // 安全なパスワードハッシュ用エンコーダ（BCrypt）を提供
    @Bean
    public PasswordEncoder passwordEncoder() {
        // 10 程度のストレングスがデフォルトで実用十分
        return new BCryptPasswordEncoder();
    }
}
```

10.2 パッケージ: controller

AdminController.java（管理者：商品/ユーザ管理、統計表示と CSV 出力。期間は直近 1 か月デフォルト）

```
// コントローラのパッケージ宣言
package com.example.fleamarketsystem.controller;
// 商品サービスの import
import com.example.fleamarketsystem.service.ItemService;
// 注文サービスの import
import com.example.fleamarketsystem.service.AppOrderService;
// ユーザサービスの import
import com.example.fleamarketsystem.service.UserService;
// 日付パラメタをフォーマットするアノテーションの import
import org.springframework.format.annotation.DateTimeFormat;
// メソッドレベル認可を使うためのアノテーション（EnableMethodSecurity が前提）
import org.springframework.security.access.prepost.PreAuthorize;
// MVC コントローラのアノテーション
import org.springframework.stereotype.Controller;
// 画面に値を渡す Model の import
import org.springframework.ui.Model;
// ルーティングアノテーション群の import
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
// 入出力例外に備えるための import
import java.io.IOException;
// CSV 出力に使用する Writer の import
import java.io.PrintWriter;
// 期間指定に使う LocalDate の import
import java.time.LocalDate;
// サーブレットの HTTP レスポンスの import
import jakarta.servlet.http.HttpServletResponse;
// MVC コントローラであることを示す
@Controller
// /admin 配下のルーティングを担当
@RequestMapping("/admin")
// このコントローラは ADMIN ロールのみアクセス可
@PreAuthorize("hasRole('ADMIN')")
public class AdminController {
    // 商品サービスの参照
    private final ItemService itemService;
    // 注文サービスの参照
    private final AppOrderService appOrderService;
    // ユーザサービスの参照
    private final UserService userService;
    // 依存関係をコンストラクタで注入
    public AdminController(ItemService itemService, AppOrderService appOrderService,
UserService userService) {
```

```
        // 商品サービスを設定
        this.itemService = itemService;
        // 注文サービスを設定
        this.appOrderService = appOrderService;
        // ユーザサービスを設定
        this.userService = userService;
    }
    // 商品管理画面の表示
    @GetMapping("/items")
    public String manageItems(Model model) {
        // すべての商品を取得
        model.addAttribute("items", itemService.getAllItems());
        // 管理画面テンプレートを返却
        return "admin_items";
    }
    // 管理者による商品削除
    @PostMapping("/items/{id}/delete")
    public String deleteItemByAdmin(@PathVariable("id") Long itemId) {
        // 商品 ID を指定して削除
        itemService.deleteItem(itemId);
        // 削除成功のクエリパラメタ付きで一覧へ
        return "redirect:/admin/items?success=deleted";
    }
    // ユーザ管理画面の表示
    @GetMapping("/users")
    public String manageUsers(Model model) {
        // 全ユーザを取得
        model.addAttribute("users", userService.getAllUsers());
        // ユーザ管理テンプレートを返却
        return "admin_users";
    }
    // ユーザの有効/無効をトグル
    @PostMapping("/users/{id}/toggle-enabled")
    public String toggleUserEnabled(@PathVariable("id") Long userId) {
        // サービスで有効フラグを反転
        userService.toggleUserEnabled(userId);
        // 成功パラメタを付けて戻る
        return "redirect:/admin/users?success=toggled";
    }
    // 統計ダッシュボードの表示
    @GetMapping("/statistics")
    public String showStatistics(
            // 開始日（未指定なら 1 ヶ月前）
            @RequestParam(value = "startDate", required = false) @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate startDate,
            // 終了日（未指定なら本日）
            @RequestParam(value = "endDate", required = false) @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate endDate,
```

```
            // モデルへ値を詰める
            Model model) {
        // 開始日デフォルトを設定
        if (startDate == null) startDate = LocalDate.now().minusMonths(1);
        // 終了日デフォルトを設定
        if (endDate == null) endDate = LocalDate.now();
        // 期間をモデルへセット
        model.addAttribute("startDate", startDate);
        // 終了日をモデルへセット
        model.addAttribute("endDate", endDate);
        // 総売上を計算してモデルへ
        model.addAttribute("totalSales", appOrderService.getTotalSales(startDate, endDate));
        // ステータス別件数をモデルへ
        model.addAttribute("orderCountByStatus",
appOrderService.getOrderCountByStatus(startDate, endDate));
        // 統計画面テンプレートを返却
        return "admin_statistics";
    }
    // 統計 CSV のエクスポート
    @GetMapping("/statistics/csv")
    public void exportStatisticsCsv(
            // 開始日 (任意)
            @RequestParam(value = "startDate", required = false) @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate startDate,
            // 終了日 (任意)
            @RequestParam(value = "endDate", required = false) @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate endDate,
            // HTTP レスポンスに直接書き込む
            HttpServletResponse response) throws IOException {
        // デフォルト開始日
        if (startDate == null) startDate = LocalDate.now().minusMonths(1);
        // デフォルト終了日
        if (endDate == null) endDate = LocalDate.now();
        // コンテンツタイプを CSV に設定 (UTF-8)
        response.setContentType("text/csv; charset=UTF-8");
        // ダウンロードファイル名を設定
        response.setHeader("Content-Disposition", "attachment;
filename=¥"flea_market_statistics.csv¥"");
        // try-with-resources で Writer を確実にクローズ
        try (PrintWriter writer = response.getWriter()) {
            // 期間の見出しを書き出す
            writer.append("統計期間: " + startDate + " から " + endDate + "¥n¥n");
            // 総売上を書き出す
            writer.append("総売上: " + appOrderService.getTotalSales(startDate, endDate) +
"¥n¥n");
            // ステータス別の見出し
            writer.append("ステータス別注文数¥n");
```

```
            // ステータス別件数を 1 行ずつ出力
            appOrderService.getOrderCountByStatus(startDate, endDate).forEach((status, count)
-> {
                // CSV1 行を出力
                writer.append(status + ″,″ + count + ″¥n″);
            });
        }
    }
}
```

AppOrderController（フローはそのまま、例外伝播とフラッシュ属性を整理）

```
// コントローラのパッケージ
package com.example.fleamarketsystem.controller;

// 参照エンティティとサービス
import com.example.fleamarketsystem.entity.User;
import com.example.fleamarketsystem.service.AppOrderService;
import com.example.fleamarketsystem.service.ItemService;
import com.example.fleamarketsystem.service.UserService;
// Stripe 例外と Intent
import com.stripe.exception.StripeException;
import com.stripe.model.PaymentIntent;
// 設定値の読み込み
import org.springframework.beans.factory.annotation.Value;
// 認証中のユーザ情報取得
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.security.core.userdetails.UserDetails;
// Spring MVC のアノテーション
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
// リダイレクト時のメッセージ用
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller
@RequestMapping("/orders")
public class AppOrderController {

    // 依存サービス
    private final AppOrderService appOrderService;
    private final UserService userService;
    private final ItemService itemService;

    // 公開キー（Stripe Elements 用）
    @Value("${stripe.public.key}")
    private String stripePublicKey;

    // 依存を受け取るコンストラクタ
    public AppOrderController(AppOrderService appOrderService, UserService userService,
ItemService itemService) {
        // フィールドに代入
        this.appOrderService = appOrderService;
        this.userService = userService;
        this.itemService = itemService;
    }

    // 購入開始：PaymentIntent 作成→client_secret を Flash で渡す
    @PostMapping("/initiate-purchase")
```

```
    public String initiatePurchase(
            // 認証済みユーザを受け取る
            @AuthenticationPrincipal UserDetails userDetails,
            // フォームから商品 ID
            @RequestParam("itemId") Long itemId,
            // リダイレクト先へ一時メッセージを渡す
            RedirectAttributes redirectAttributes) {
        // 買い手ユーザをメールから取得
        User buyer = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("Buyer not found"));
        try {
            // Stripe に Intent 作成→注文はサービス側で“決済待ち”作成
            PaymentIntent paymentIntent = appOrderService.initiatePurchase(itemId, buyer);
            // クライアント用に client_secret と itemId を Flash に積む
            redirectAttributes.addFlashAttribute("clientSecret",
paymentIntent.getClientSecret());
            redirectAttributes.addFlashAttribute("itemId", itemId);
            // 確認画面へ
            return "redirect:/orders/confirm-payment";
        } catch (IllegalStateException | IllegalArgumentException | StripeException e) {
            // 失敗時はエラーメッセージを載せて商品詳細に戻す
            redirectAttributes.addFlashAttribute("errorMessage", e.getMessage());
            return "redirect:/items/" + itemId;
        }
    }

    // Stripe Elements での支払い確認画面
    @GetMapping("/confirm-payment")
    public String confirmPayment(
            // Flash から clientSecret (無ければ一覧へ)
            @ModelAttribute("clientSecret") String clientSecret,
            // Flash から itemId (無ければ一覧へ)
            @ModelAttribute("itemId") Long itemId,
            // 画面に値を渡す
            Model model) {
        // データ欠如時は一覧に戻す
        if (clientSecret == null || itemId == null) {
            return "redirect:/items";
        }
        // テンプレートに値を積む
        model.addAttribute("clientSecret", clientSecret);
        model.addAttribute("itemId", itemId);
        model.addAttribute("stripePublicKey", stripePublicKey);
        // 確認用テンプレートを返す
        return "payment_confirmation";
    }

    // フロント(Stripe.js)で決済成功後の完了処理
    @GetMapping("/complete-purchase")
```

```
    public String completePurchase(
            // Stripe から受け取った PaymentIntent ID
            @RequestParam("paymentIntentId") String paymentIntentId,
            // リダイレクト用のフラッシュ属性
            RedirectAttributes redirectAttributes) {
        try {
            // サービスで“安全に”購入確定
            appOrderService.completePurchase(paymentIntentId);
            // 成功メッセージ
            redirectAttributes.addFlashAttribute("successMessage", "商品を購入しました！");
            // 直近の購入済 ID が取れればレビュー画面へ
            return appOrderService.getLatestCompletedOrderId()
                    .map(orderId -> "redirect:/reviews/new/" + orderId)
                    .orElseGet(() -> {
                        // 取れない場合は注文履歴へ
                        redirectAttributes.addFlashAttribute("errorMessage", "購入は完了しましたが、評価ページへのリダイレクトに失敗しました。");
                        return "redirect:/my-page/orders";
                    });
        } catch (StripeException | IllegalStateException e) {
            // 例外時はメッセージを積んで一覧へ
            redirectAttributes.addFlashAttribute("errorMessage", "決済処理中にエラーが発生しました: " + e.getMessage());
            return "redirect:/items";
        }
    }

    // Stripe Webhook エンドポイント（署名検証は本番で必須：ここでは受信確認のみ）
    @PostMapping("/stripe-webhook")
    public void handleStripeWebhook(@RequestBody String payload, @RequestHeader("Stripe-Signature") String sigHeader) {
        // 本番では sigHeader で署名検証を必ず行う
        System.out.println("Received Stripe Webhook: " + payload);
        // 例: event type ごとに処理を分岐
    }

    // 出品者の発送操作
    @PostMapping("/{id}/ship")
    public String shipOrder(@PathVariable("id") Long orderId, RedirectAttributes redirectAttributes) {
        try {
            // サービスで発送済みに更新＋通知
            appOrderService.markOrderAsShipped(orderId);
            // 成功メッセージ
            redirectAttributes.addFlashAttribute("successMessage", "商品を発送済みにしました。");
        } catch (IllegalArgumentException e) {
            // エラーメッセージ
            redirectAttributes.addFlashAttribute("errorMessage", e.getMessage());
```

```
        }
        // マイページ売上へ戻る
        return "redirect:/my-page/sales";
    }
}
```

ChatController.java（商品別チャットの表示/送信。認証ユーザを取得し、サービス層へ委譲）

```
// コントローラが属するパッケージ名
package com.example.fleamarketsystem.controller;
// 送信者の特定に使うユーザエンティティ
import com.example.fleamarketsystem.entity.User;
// チャット機能のビジネスロジックサービス
import com.example.fleamarketsystem.service.ChatService;
// 商品取得などで使うサービス
import com.example.fleamarketsystem.service.ItemService;
// 認証ユーザの実体を取得するためのサービス
import com.example.fleamarketsystem.service.UserService;
// ログインユーザをメソッド引数で受けるためのアノテーション
import org.springframework.security.core.annotation.AuthenticationPrincipal;
// Spring Security のユーザ詳細型
import org.springframework.security.core.userdetails.UserDetails;
// MVC コントローラを示すアノテーション
import org.springframework.stereotype.Controller;
// 画面に値を渡すための Model
import org.springframework.ui.Model;
// ルーティングアノテーション群
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

// MVC コントローラとして登録
@Controller
// ベースパス /chat にマッピング
@RequestMapping("/chat")
public class ChatController {

    // チャットサービスの参照
    private final ChatService chatService;
    // 商品サービスの参照
    private final ItemService itemService;
    // ユーザサービスの参照
    private final UserService userService;

    // 必要な依存をコンストラクタで受け取る
    public ChatController(ChatService chatService, ItemService itemService, UserService
userService) {
        // フィールドへ代入
        this.chatService = chatService;
        // フィールドへ代入
        this.itemService = itemService;
        // フィールドへ代入
        this.userService = userService;
```

```
    }

    // 商品単位のチャット画面を表示
    @GetMapping("/{itemId}")
    public String showChatScreen(@PathVariable("itemId") Long itemId, Model model) {
        // 商品を取得（見つからなければ例外送出）
        model.addAttribute("item", itemService.getItemById(itemId)
                .orElseThrow(() -> new RuntimeException("Item not found")));
        // チャット履歴を昇順で取得
        model.addAttribute("chats", chatService.getChatMessagesByItem(itemId));
        // チャットは商品詳細テンプレートに埋め込んで表示
        return "item_detail";
    }

    // チャットの新規メッセージ送信
    @PostMapping("/{itemId}")
    public String sendMessage(
            // パスから商品 ID を受け取る
            @PathVariable("itemId") Long itemId,
            // 認証済みユーザを受け取る
            @AuthenticationPrincipal UserDetails userDetails,
            // 送信するメッセージ本文を受け取る
            @RequestParam("message") String message) {
        // ログインユーザをメールで特定（見つからなければ例外）
        User sender = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("Sender not found"));
        // サービス層に送信処理を委譲（通知などはサービス側で実施）
        chatService.sendMessage(itemId, sender, message);
        // 送信後は同じチャット画面へ遷移
        return "redirect:/chat/{itemId}";
    }
}
```

DashboardController.java（管理者はダッシュボード、一般ユーザは商品一覧へ誘導）

```
// コントローラのパッケージ宣言
package com.example.fleamarketsystem.controller;
// ユーザエンティティの取得に使うリポジトリ
import com.example.fleamarketsystem.repository.UserRepository;
// 商品の一覧取得に使うサービス
import com.example.fleamarketsystem.service.ItemService;
// 注文の一覧取得に使うサービス
import com.example.fleamarketsystem.service.AppOrderService;
// 認証ユーザを受け取るためのアノテーション
import org.springframework.security.core.annotation.AuthenticationPrincipal;
// 認証ユーザの詳細型
import org.springframework.security.core.userdetails.UserDetails;
// MVC コントローラのアノテーション
import org.springframework.stereotype.Controller;
// 画面へ値を渡す Model
import org.springframework.ui.Model;
// GET ハンドラのアノテーション
import org.springframework.web.bind.annotation.GetMapping;

// MVC コントローラとして登録
@Controller
public class DashboardController {

    // ユーザ検索に使うリポジトリ
    private final UserRepository userRepository;
    // 商品サービスの参照
    private final ItemService itemService;
    // 注文サービスの参照
    private final AppOrderService appOrderService;

    // 依存をコンストラクタ注入
    public DashboardController(UserRepository userRepository, ItemService itemService,
AppOrderService appOrderService) {
        // フィールドへ設定
        this.userRepository = userRepository;
        // フィールドへ設定
        this.itemService = itemService;
        // フィールドへ設定
        this.appOrderService = appOrderService;
    }

    // ダッシュボード画面のハンドラ
    @GetMapping("/dashboard")
    public String dashboard(@AuthenticationPrincipal UserDetails userDetails, Model model) {
        // ログインユーザをメールで検索
        com.example.fleamarketsystem.entity.User currentUser =
userRepository.findByEmail(userDetails.getUsername())
```

```
                .orElseThrow(() -> new RuntimeException("User not found"));
        // 管理者であれば管理ダッシュボード表示
        if (currentUser.getRole().equals("ADMIN")) {
            // 最近の商品を表示（ここでは全件）
            model.addAttribute("recentItems", itemService.getAllItems());
            // 最近の注文を表示（ここでは全件）
            model.addAttribute("recentOrders", appOrderService.getAllOrders());
            // 管理者用テンプレートへ
            return "admin_dashboard";
        } else {
            // 一般ユーザは商品一覧へ誘導
            return "redirect:/items";
        }
    }
}
```

ItemController.java（一覧/検索/詳細/出品・編集・削除・お気に入り操作。認可チェックと画像連携を内包）

```
// コントローラが属するパッケージ名
package com.example.fleamarketsystem.controller;
// 入出力例外に備えるための import
import java.io.IOException;
// 金額を正確に扱うための BigDecimal の import
import java.math.BigDecimal;
// 一覧描画などで使うコレクションの import
import java.util.List;
// Optional で存在チェックを簡潔にするための import
import java.util.Optional;
// ページング機能を使うための import
import org.springframework.data.domain.Page;
// 認証ユーザ取得用アノテーションの import
import org.springframework.security.core.annotation.AuthenticationPrincipal;
// 認証ユーザの型の import
import org.springframework.security.core.userdetails.UserDetails;
// MVC のコントローラアノテーションの import
import org.springframework.stereotype.Controller;
// 画面へデータを渡すための Model の import
import org.springframework.ui.Model;
// HTTP GET を扱うための import
import org.springframework.web.bind.annotation.GetMapping;
// パス変数を扱うための import
import org.springframework.web.bind.annotation.PathVariable;
// HTTP POST を扱うための import
import org.springframework.web.bind.annotation.PostMapping;
// コントローラ全体のベースパス指定用 import
import org.springframework.web.bind.annotation.RequestMapping;
// クエリ/フォームのパラメタ取得用 import
import org.springframework.web.bind.annotation.RequestParam;
// 画像アップロードのための MultipartFile の import
import org.springframework.web.multipart.MultipartFile;
// リダイレクト時にメッセージを渡すための import
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
// カテゴリエンティティの import
import com.example.fleamarketsystem.entity.Category;
// 商品エンティティの import
import com.example.fleamarketsystem.entity.Item;
// ユーザエンティティの import
import com.example.fleamarketsystem.entity.User;
// カテゴリ関連サービスの import
import com.example.fleamarketsystem.service.CategoryService;
// チャット関連サービスの import
import com.example.fleamarketsystem.service.ChatService;
// お気に入り関連サービスの import
import com.example.fleamarketsystem.service.FavoriteService;
```

```
// 商品関連サービスの import
import com.example.fleamarketsystem.service.ItemService;
// レビュー関連サービスの import
import com.example.fleamarketsystem.service.ReviewService;
// ユーザ関連サービスの import
import com.example.fleamarketsystem.service.UserService;
// MVC コントローラであることを示すアノテーション
@Controller
// /items 配下のリクエストを受け付ける
@RequestMapping("/items")
public class ItemController {
    // 商品サービスへの参照
    private final ItemService itemService;
    // カテゴリサービスへの参照
    private final CategoryService categoryService;
    // ユーザサービスへの参照
    private final UserService userService;
    // チャットサービスへの参照
    private final ChatService chatService;
    // お気に入りサービスへの参照
    private final FavoriteService favoriteService;
    // レビューサービスへの参照
    private final ReviewService reviewService;
    // 依存関係をコンストラクタインジェクションで受け取る
    public ItemController(ItemService itemService, CategoryService categoryService,
UserService userService,
            ChatService chatService, FavoriteService favoriteService, ReviewService
reviewService) {
        // 商品サービスをフィールドに設定
        this.itemService = itemService;
        // カテゴリサービスをフィールドに設定
        this.categoryService = categoryService;
        // ユーザサービスをフィールドに設定
        this.userService = userService;
        // チャットサービスをフィールドに設定
        this.chatService = chatService;
        // お気に入りサービスをフィールドに設定
        this.favoriteService = favoriteService;
        // レビューサービスをフィールドに設定
        this.reviewService = reviewService;
    }
    // 商品一覧を表示する GET エンドポイント
    @GetMapping
    public String listItems(
            // 検索キーワード (任意)
            @RequestParam(value = "keyword", required = false) String keyword,
            // カテゴリ ID (任意)
            @RequestParam(value = "categoryId", required = false) Long categoryId,
```

```
            // ページ番号（0始まり、デフォルト0）
            @RequestParam(value = "page", defaultValue = "0") int page,
            // 1ページ件数（デフォルト10）
            @RequestParam(value = "size", defaultValue = "10") int size,
            // 画面へデータを渡すモデル
            Model model) {
        // 条件に応じて商品を検索（出品中のみ）
        Page<Item> items = itemService.searchItems(keyword, categoryId, page, size);
        // カテゴリ一覧を取得
        List<Category> categories = categoryService.getAllCategories();
        // 商品一覧をテンプレートへ渡す
        model.addAttribute("items", items);
        // カテゴリ一覧をテンプレートへ渡す
        model.addAttribute("categories", categories);
        // 一覧画面のテンプレート名を返す
        return "item_list";
    }
    // 商品詳細表示のGETエンドポイント
    @GetMapping("/{id}")
    public String showItemDetail(@PathVariable("id") Long id, @AuthenticationPrincipal
UserDetails userDetails,
            Model model) {
        // 商品をIDで検索（存在しない場合の判定にOptionalを使う）
        Optional<Item> item = itemService.getItemById(id);
        // 見つからなければ一覧へリダイレクト
        if (item.isEmpty()) {
            // 商品が見つからないため一覧へ
            return "redirect:/items";
        }
        // 商品本体をテンプレートへ渡す
        model.addAttribute("item", item.get());
        // 当該商品のチャット履歴を昇順で取得して渡す
        model.addAttribute("chats", chatService.getChatMessagesByItem(id));
        // 出品者の平均評価があれば1桁小数で埋め込む
        reviewService.getAverageRatingForSeller(item.get().getSeller())
                .ifPresent(avg -> model.addAttribute("sellerAverageRating",
String.format("%.1f", avg)));
        // ログイン済みであればお気に入り状態を判定して渡す
        if (userDetails != null) {
            // 現在のログインユーザをメールで特定
            User currentUser = userService.getUserByEmail(userDetails.getUsername())
                    .orElseThrow(() -> new RuntimeException("User not found"));
            // お気に入り登録済みかどうかを判定
            model.addAttribute("isFavorited", favoriteService.isFavorited(currentUser, id));
        }
        // 商品詳細テンプレートを返す
        return "item_detail";
    }
```

```
// 出品フォーム表示の GET エンドポイント
@GetMapping("/new")
public String showAddItemForm(Model model) {
    // 空の Item をフォームのバインド用に渡す
    model.addAttribute("item", new Item());
    // カテゴリの選択肢を渡す
    model.addAttribute("categories", categoryService.getAllCategories());
    // 入力フォームのテンプレート名
    return "item_form";
}
// 出品登録の POST エンドポイント
@PostMapping
public String addItem(
        // 認証済みユーザを取得
        @AuthenticationPrincipal UserDetails userDetails,
        // 商品名
        @RequestParam("name") String name,
        // 商品説明
        @RequestParam("description") String description,
        // 価格
        @RequestParam("price") BigDecimal price,
        // カテゴリ ID
        @RequestParam("categoryId") Long categoryId,
        // 画像ファイル (任意)
        @RequestParam(value = "image", required = false) MultipartFile imageFile,
        // リダイレクトメッセージ用
        RedirectAttributes redirectAttributes) {
    // 出品者ユーザを取得 (存在しなければ例外)
    User seller = userService.getUserByEmail(userDetails.getUsername())
            .orElseThrow(() -> new RuntimeException("Seller not found"));
    // カテゴリを ID で取得 (存在しなければ 400 相当)
    Category category = categoryService.getCategoryById(categoryId)
            .orElseThrow(() -> new IllegalArgumentException("Category not found"));
    // 新規 Item を作成して各項目を設定
    Item item = new Item();
    // 出品者を設定
    item.setSeller(seller);
    // 商品名を設定
    item.setName(name);
    // 説明を設定
    item.setDescription(description);
    // 価格を設定
    item.setPrice(price);
    // カテゴリを設定
    item.setCategory(category);
    // 画像があればアップロードして保存、なければそのまま保存
    try {
        // 画像アップロードを含めて保存
        itemService.saveItem(item, imageFile);
```

```
            // 成功メッセージをフラッシュ
            redirectAttributes.addFlashAttribute("successMessage", "商品を出品しました！");
        } catch (IOException e) {
            // 画像アップロード失敗時のエラーメッセージ
            redirectAttributes.addFlashAttribute("errorMessage", "画像のアップロードに失敗しました: " + e.getMessage());
            // 入力フォームへ戻す
            return "redirect:/items/new";
        }
        // 一覧へリダイレクト
        return "redirect:/items";
    }
    // 出品編集フォーム表示の GET エンドポイント
    @GetMapping("/{id}/edit")
    public String showEditItemForm(@PathVariable("id") Long id, Model model) {
        // 対象商品を取得
        Optional<Item> item = itemService.getItemById(id);
        // なければ一覧へ
        if (item.isEmpty()) {
            // 商品が存在しない
            return "redirect:/items";
        }
        // 既存商品の内容をフォームへ
        model.addAttribute("item", item.get());
        // カテゴリ選択肢を用意
        model.addAttribute("categories", categoryService.getAllCategories());
        // 入力フォームを返却
        return "item_form";
    }
    // 出品更新の POST エンドポイント（簡便のため POST を使用）
    @PostMapping("/{id}")
    public String updateItem(
            // パスの ID
            @PathVariable("id") Long id,
            // 現在のログインユーザ
            @AuthenticationPrincipal UserDetails userDetails,
            // 更新後の商品名
            @RequestParam("name") String name,
            // 更新後の説明
            @RequestParam("description") String description,
            // 更新後の価格
            @RequestParam("price") BigDecimal price,
            // 更新後のカテゴリ ID
            @RequestParam("categoryId") Long categoryId,
            // 差し替え画像（任意）
            @RequestParam(value = "image", required = false) MultipartFile imageFile,
            // リダイレクトメッセージ
            RedirectAttributes redirectAttributes) {
```

```
        // 既存商品を取得（なければ404相当）
        Item existingItem = itemService.getItemById(id)
                .orElseThrow(() -> new RuntimeException("Item not found"));
        // 現在ユーザを取得
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        // 出品者以外の編集をブロック
        if (!existingItem.getSeller().getId().equals(currentUser.getId())) {
            // 権限エラーをフラッシュ
            redirectAttributes.addFlashAttribute("errorMessage", "この商品は編集できません。
");
            // 一覧へ戻す
            return "redirect:/items";
        }
        // カテゴリ取得（なければ400)
        Category category = categoryService.getCategoryById(categoryId)
                .orElseThrow(() -> new IllegalArgumentException("Category not found"));
        // 値を上書き
        existingItem.setName(name);
        // 説明を上書き
        existingItem.setDescription(description);
        // 価格を上書き
        existingItem.setPrice(price);
        // カテゴリを上書き
        existingItem.setCategory(category);
        // 保存処理（画像差し替えがあればアップロード)
        try {
            // 保存実行
            itemService.saveItem(existingItem, imageFile);
            // 成功メッセージ
            redirectAttributes.addFlashAttribute("successMessage", "商品を更新しました！");
        } catch (IOException e) {
            // 画像アップロードの失敗を通知
            redirectAttributes.addFlashAttribute("errorMessage", "画像のアップロードに失敗しま
した: " + e.getMessage());
            // 編集画面へ戻す
            return "redirect:/items/{id}/edit";
        }
        // 詳細画面へリダイレクト
        return "redirect:/items/{id}";
    }
    // 出品削除のPOSTエンドポイント
    @PostMapping("/{id}/delete")
    public String deleteItem(@PathVariable("id") Long id, @AuthenticationPrincipal UserDetails
userDetails,
            RedirectAttributes redirectAttributes) {
        // 削除対象の商品を取得
        Item itemToDelete = itemService.getItemById(id)
                .orElseThrow(() -> new RuntimeException("Item not found"));
```

```
        // 現在のユーザを取得
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        // 出品者以外は削除不可
        if (!itemToDelete.getSeller().getId().equals(currentUser.getId())) {
            // 権限エラーを通知
            redirectAttributes.addFlashAttribute("errorMessage", "この商品は削除できません。
");
            // 一覧へ
            return "redirect:/items";
        }
        // サービスを通じて削除（画像削除も内包）
        itemService.deleteItem(id);
        // 成功メッセージ
        redirectAttributes.addFlashAttribute("successMessage", "商品を削除しました。");
        // 一覧へ
        return "redirect:/items";
    }
    // お気に入り登録の POST
    @PostMapping("/{id}/favorite")
    public String addFavorite(@PathVariable("id") Long itemId, @AuthenticationPrincipal
UserDetails userDetails,
            RedirectAttributes redirectAttributes) {
        // 現在ユーザを取得
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        // 例外をサービス側で投げ、ここでユーザに伝える
        try {
            // お気に入り追加
            favoriteService.addFavorite(currentUser, itemId);
            // 成功メッセージ
            redirectAttributes.addFlashAttribute("successMessage", "お気に入りに追加しました！
");
        } catch (IllegalStateException e) {
            // エラーメッセージ
            redirectAttributes.addFlashAttribute("errorMessage", e.getMessage());
        }
        // 詳細へ戻る
        return "redirect:/items/{id}";
    }
    // お気に入り解除の POST
    @PostMapping("/{id}/unfavorite")
    public String removeFavorite(@PathVariable("id") Long itemId, @AuthenticationPrincipal
UserDetails userDetails,
            RedirectAttributes redirectAttributes) {
        // 現在ユーザを取得
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
```

```
        // 例外をユーザに伝える
        try {
            // お気に入り削除
            favoriteService.removeFavorite(currentUser, itemId);
            // 成功メッセージ
            redirectAttributes.addFlashAttribute("successMessage", "お気に入りから削除しました。");
        } catch (IllegalStateException e) {
            // エラーメッセージ
            redirectAttributes.addFlashAttribute("errorMessage", e.getMessage());
        }
        // 詳細へ戻る
        return "redirect:/items/{id}";
    }
}
```

LoginController.java（ログイン画面表示のみを担当）

```
// コントローラのパッケージ宣言
package com.example.fleamarketsystem.controller;
// MVC コントローラのアノテーション
import org.springframework.stereotype.Controller;
// GET ハンドラのアノテーション
import org.springframework.web.bind.annotation.GetMapping;

// MVC コントローラとして登録
@Controller
public class LoginController {

    // ログインページ表示のハンドラ
    @GetMapping("/login")
    public String login() {
        // login.html (Thymeleaf) を返す
        return "login";
    }
}
```

ReviewController.java（評価フォーム表示と投稿。本人性と重複はサービスで検証）

```
// コントローラのパッケージ宣言
package com.example.fleamarketsystem.controller;
// 認証ユーザを引数で受けるためのアノテーション
import org.springframework.security.core.annotation.AuthenticationPrincipal;
// Spring Security のユーザ詳細型
import org.springframework.security.core.userdetails.UserDetails;
// MVC コントローラのアノテーション
import org.springframework.stereotype.Controller;
// 画面へ値を渡す Model
import org.springframework.ui.Model;
// ルーティングアノテーション群
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
// リダイレクト先へ一時メッセージを渡すための型
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
// 注文エンティティ（フォームに表示するため）
import com.example.fleamarketsystem.entity.AppOrder;
// ユーザエンティティ（レビュワーの解決に使用）
import com.example.fleamarketsystem.entity.User;
// 注文サービス（注文取得に使用）
import com.example.fleamarketsystem.service.AppOrderService;
// レビューサービス（バリデーションと保存に使用）
import com.example.fleamarketsystem.service.ReviewService;
// ユーザサービス（ログインユーザの解決に使用）
import com.example.fleamarketsystem.service.UserService;

// MVC コントローラとして登録
@Controller
// ベースパス /reviews にマッピング
@RequestMapping("/reviews")
public class ReviewController {

    // レビューサービスの参照
    private final ReviewService reviewService;
    // 注文サービスの参照
    private final AppOrderService appOrderService;
    // ユーザサービスの参照
    private final UserService userService;

    // 依存関係をコンストラクタで受け取る
    public ReviewController(ReviewService reviewService, AppOrderService appOrderService,
UserService userService) {
        // フィールドへ代入
        this.reviewService = reviewService;
```

```
        // フィールドへ代入
        this.appOrderService = appOrderService;
        // フィールドへ代入
        this.userService = userService;
    }

    // 新規レビューの入力フォーム表示
    @GetMapping("/new/{orderId}")
    public String showReviewForm(@PathVariable("orderId") Long orderId, Model model) {
        // 注文を取得（見つからなければ例外）
        AppOrder order = appOrderService.getOrderById(orderId)
                .orElseThrow(() -> new IllegalArgumentException("Order not found."));
        // テンプレートへ注文を渡す（商品名などの表示用）
        model.addAttribute("order", order);
        // フォームのテンプレートを返却
        return "review_form";
    }

    // レビュー投稿の送信
    @PostMapping
    public String submitReview(
            // 認証ユーザを受け取る
            @AuthenticationPrincipal UserDetails userDetails,
            // 評価対象の注文 ID
            @RequestParam("orderId") Long orderId,
            // 評点（1〜5 想定）
            @RequestParam("rating") int rating,
            // 任意コメント
            @RequestParam("comment") String comment,
            // リダイレクト先へメッセージを渡す
            RedirectAttributes redirectAttributes) {
        // ログインユーザを取得（見つからなければ例外）
        User reviewer = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        try {
            // サービス層で検証＋保存を実施
            reviewService.submitReview(orderId, reviewer, rating, comment);
            // 成功メッセージを設定
            redirectAttributes.addFlashAttribute("successMessage", "評価を送信しました！");
        } catch (IllegalStateException | IllegalArgumentException e) {
            // ルール違反などのエラーをメッセージで返す
            redirectAttributes.addFlashAttribute("errorMessage", e.getMessage());
        }
        // 購入者の注文履歴へ遷移
        return "redirect:/my-page/orders";
    }
}
```

UserController.java（マイページ：プロフィール／出品／注文／売上／お気に入り／自分のレビュー一覧）

```
// コントローラのパッケージ宣言
package com.example.fleamarketsystem.controller;
// 認証ユーザをパラメータ注入するアノテーション
import org.springframework.security.core.annotation.AuthenticationPrincipal;
// Spring Security のユーザ詳細型
import org.springframework.security.core.userdetails.UserDetails;
// MVC コントローラのアノテーション
import org.springframework.stereotype.Controller;
// 画面へ値を渡す Model
import org.springframework.ui.Model;
// ルーティング関連のアノテーション
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
// ユーザエンティティ（Model へ載せるため）
import com.example.fleamarketsystem.entity.User;
// 注文サービス（購入/売上一覧で使用）
import com.example.fleamarketsystem.service.AppOrderService;
// お気に入りサービス（お気に入り一覧で使用）
import com.example.fleamarketsystem.service.FavoriteService;
// 商品サービス（出品一覧で使用）
import com.example.fleamarketsystem.service.ItemService;
// レビューサービス（自分のレビュー一覧で使用）
import com.example.fleamarketsystem.service.ReviewService;
// ユーザサービス（ログインユーザの取得で使用）
import com.example.fleamarketsystem.service.UserService;

// MVC コントローラとして登録
@Controller
// ベースパス /my-page にマッピング
@RequestMapping("/my-page")
public class UserController {

    // ユーザサービスの参照
    private final UserService userService;
    // 商品サービスの参照
    private final ItemService itemService;
    // 注文サービスの参照
    private final AppOrderService appOrderService;
    // お気に入りサービスの参照
    private final FavoriteService favoriteService;
    // レビューサービスの参照
    private final ReviewService reviewService;

    // 依存関係をコンストラクタで受け取る
    public UserController(UserService userService, ItemService itemService, AppOrderService
appOrderService,
            FavoriteService favoriteService, ReviewService reviewService) {
```

```
        // フィールドへ代入
        this.userService = userService;
        // フィールドへ代入
        this.itemService = itemService;
        // フィールドへ代入
        this.appOrderService = appOrderService;
        // フィールドへ代入
        this.favoriteService = favoriteService;
        // フィールドへ代入
        this.reviewService = reviewService;
    }

    // マイページのトップ（プロフィール）
    @GetMapping
    public String myPage(@AuthenticationPrincipal UserDetails userDetails, Model model) {
        // 現在のユーザを取得（見つからなければ例外）
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        // 画面へユーザ情報を渡す
        model.addAttribute("user", currentUser);
        // プロフィール画面を返却
        return "my_page";
    }

    // 自分の出品一覧
    @GetMapping("/selling")
    public String mySellingItems(@AuthenticationPrincipal UserDetails userDetails, Model
model) {
        // 現在のユーザ取得
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        // 出品中/過去出品を取得
        model.addAttribute("sellingItems", itemService.getItemsBySeller(currentUser));
        // 出品者向け一覧テンプレート
        return "seller_items";
    }

    // 自分が購入した注文一覧
    @GetMapping("/orders")
    public String myOrders(@AuthenticationPrincipal UserDetails userDetails, Model model) {
        // 現在のユーザ取得
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        // 自分の注文を取得
        model.addAttribute("myOrders", appOrderService.getOrdersByBuyer(currentUser));
        // 購入者向け一覧テンプレート
        return "buyer_app_orders";
    }
```

```
    // 自分が販売した注文一覧
    @GetMapping("/sales")
    public String mySales(@AuthenticationPrincipal UserDetails userDetails, Model model) {
        // 現在のユーザ取得
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        // 自分の売上（自分の商品に紐づく注文）を取得
        model.addAttribute("mySales", appOrderService.getOrdersBySeller(currentUser));
        // 出品者向け売上一覧テンプレート
        return "seller_app_orders";
    }

    // 自分のお気に入り商品一覧
    @GetMapping("/favorites")
    public String myFavorites(@AuthenticationPrincipal UserDetails userDetails, Model model) {
        // 現在のユーザ取得
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        // お気に入り商品を取得
        model.addAttribute("favoriteItems",
favoriteService.getFavoriteItemsByUser(currentUser));
        // お気に入り一覧テンプレート
        return "my_favorites";
    }

    // 自分が書いたレビュー一覧
    @GetMapping("/reviews")
    public String myReviews(@AuthenticationPrincipal UserDetails userDetails, Model model) {
        // 現在のユーザ取得
        User currentUser = userService.getUserByEmail(userDetails.getUsername())
                .orElseThrow(() -> new RuntimeException("User not found"));
        // 自分が投稿したレビュー一覧を取得
        model.addAttribute("reviews", reviewService.getReviewsByReviewer(currentUser));
        // レビュー一覧テンプレート
        return "user_reviews";
    }
}
```

10.3 パッケージ: entity

AppOrder.java エンティティ（paymentIntentId を追加し決済と 1:1 ひも付け）

```
// エンティティが置かれるパッケージ
package com.example.fleamarketsystem.entity;

// JPA アノテーションの import
import jakarta.persistence.*;
// Lombok でゲッター/セッター等を自動生成
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

// 金額表現と日時のための import
import java.math.BigDecimal;
import java.time.LocalDateTime;

@Entity
// テーブル名を明示
@Table(name = "app_order")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AppOrder {
    // 主キーの定義（AUTO_INCREMENT）
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // 注文が紐づく商品（必須）
    @ManyToOne
    @JoinColumn(name = "item_id", nullable = false)
    private Item item;

    // 買い手ユーザ（必須）
    @ManyToOne
    @JoinColumn(name = "buyer_id", nullable = false)
    private User buyer;

    // 決済金額のスナップショット（必須）
    @Column(nullable = false)
    private BigDecimal price;

    // 注文状態（購入済/発送済/決済待ち 等）
    @Column(nullable = false)
    private String status = "購入済";

    // Stripe の PaymentIntent ID を保持（決済と注文を 1 対 1 で特定）
    @Column(name = "payment_intent_id", unique = true)
    private String paymentIntentId;
```

```
    // 作成日時（集計用）
    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt = LocalDateTime.now();
}
```

Category.java（カテゴリの JPA エンティティ。名称は一意）

```
// パッケージ宣言：このクラスが属するパッケージを指定
package com.example.fleamarketsystem.entity;

// JPA アノテーションを利用するためのインポート
import jakarta.persistence.*;
// Lombok でゲッター/セッター等を自動生成
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

// このクラスが JPA エンティティであることを宣言
@Entity
// 対応するテーブル名を明示（category）
@Table(name = "category")
// Lombok：getter/setter, toString, equals/hashCode を自動生成
@Data
// Lombok：引数なしコンストラクタを生成
@NoArgsConstructor
// Lombok：全フィールドを引数に持つコンストラクタを生成
@AllArgsConstructor
public class Category {
    // 主キーを表すフィールド
    @Id
    // 主キーの採番戦略：DB の IDENTITY(シリアル)に委譲
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // NOT NULL かつ一意制約を付与したカテゴリ名
    @Column(nullable = false, unique = true)
    private String name;
}
```

Chat.java（商品別チャットの JPA エンティティ。投稿内容と作成日時を保持）

```
// パッケージ宣言
package com.example.fleamarketsystem.entity;

// JPA アノテーションのインポート
import jakarta.persistence.*;
// Lombok でボイラープレート削減
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

// 日時型を利用するためのインポート
import java.time.LocalDateTime;

// JPA エンティティであることを示す
@Entity
// 対応テーブル名 chat を指定
@Table(name = "chat")
// Lombok：getter/setter 等を自動生成
@Data
// Lombok：デフォルトコンストラクタ
@NoArgsConstructor
// Lombok：全フィールドコンストラクタ
@AllArgsConstructor
public class Chat {
    // 主キー
    @Id
    // IDENTITY 戦略で自動採番
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // 対象商品の参照（外部キー item_id）。NULL 禁止
    @ManyToOne
    @JoinColumn(name = "item_id", nullable = false)
    private Item item;

    // 送信者ユーザーの参照（外部キー sender_id）。NULL 禁止
    @ManyToOne
    @JoinColumn(name = "sender_id", nullable = false)
    private User sender;

    // 本文は長文になる可能性があるため TEXT 型
    @Column(columnDefinition = "TEXT")
    private String message;

    // 作成日時。列名を created_at にし、NULL を許可しない
    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt = LocalDateTime.now();
}
```

FavoriteItem.java（お気に入りの中間エンティティ。ユーザーと商品をひも付け）

```
// パッケージ宣言
package com.example.fleamarketsystem.entity;

// JPA 関連インポート
import jakarta.persistence.*;
// Lombok でメソッド自動生成
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

// 日時型
import java.time.LocalDateTime;

// JPA エンティティ指定
@Entity
// テーブル名 favorite_item を使用
@Table(name = "favorite_item")
// Lombok：getter/setter 等
@Data
// Lombok：デフォルトコンストラクタ
@NoArgsConstructor
// Lombok：全フィールドコンストラクタ
@AllArgsConstructor
public class FavoriteItem {
    // 主キー
    @Id
    // IDENTITY で自動採番
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // お気に入り登録したユーザー。NULL 禁止
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    // お気に入り対象の商品。NULL 禁止
    @ManyToOne
    @JoinColumn(name = "item_id", nullable = false)
    private Item item;

    // お気に入り登録日時。既定で現在時刻
    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt = LocalDateTime.now();
}
```

Item.java（商品エンティティ。価格/画像URL/状態/作成日時を保持）

```
// パッケージ宣言
package com.example.fleamarketsystem.entity;

// JPAアノテーションの読み込み
import jakarta.persistence.*;
// Lombokでボイラープレート削減
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

// 金額を表すBigDecimalと日時を利用
import java.math.BigDecimal;
import java.time.LocalDateTime; // Add this import

// JPAエンティティ指定
@Entity
// テーブル名 item を明示
@Table(name = "item")
// Lombok：getter/setter等
@Data
// Lombok：デフォルトコンストラクタ
@NoArgsConstructor
// Lombok：全フィールドコンストラクタ
@AllArgsConstructor
public class Item {
    // 主キー
    @Id
    // 自動採番（IDENTITY）
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // 出品者（usersテーブルへの外部キー）。NULL禁止
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User seller;

    // 商品名。NULL禁止
    @Column(nullable = false)
    private String name;

    // 商品説明。長文想定でTEXT
    @Column(columnDefinition = "TEXT")
    private String description;

    // 価格。NULL禁止（小数を扱うためBigDecimal）
    @Column(nullable = false)
    private BigDecimal price;
```

```
    // カテゴリ（外部キー）。NULL 可（未分類を許容）
    @ManyToOne
    @JoinColumn(name = "category_id")
    private Category category;

    // 出品ステータス。初期値は「出品中」
    private String status = "出品中"; // default status

    // 画像 URL（Cloudinary にアップロードした結果を格納）
    // For image URLs (Cloudinary)
    private String imageUrl;

    // 作成日時。列名を created_at に固定、初期値は現在時刻
    @Column(name = "created_at", nullable = false) // New field
    private LocalDateTime createdAt = LocalDateTime.now();
}
```

Review.java（注文に対するレビュー。購入者→出品者への評価/コメントを保存）

```
// パッケージ宣言
package com.example.fleamarketsystem.entity;

// JPA インポート
import jakarta.persistence.*;
// Lombok インポート
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

// 日時型
import java.time.LocalDateTime;

// JPA エンティティ宣言
@Entity
// テーブル名 review を指定
@Table(name = "review")
// Lombok：getter/setter 等自動生成
@Data
// Lombok：引数なしコンストラクタ
@NoArgsConstructor
// Lombok：全フィールドコンストラクタ
@AllArgsConstructor
public class Review {
    // 主キー
    @Id
    // IDENTITY 戦略
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // 一つの注文に対してレビューは 1 件（ユニーク制約）→ OneToOne
    @OneToOne
    @JoinColumn(name = "order_id", nullable = false, unique = true)
    private AppOrder order;

    // レビューワ（購入者）。複数レビューに同一ユーザーが存在し得るため ManyToOne
    @ManyToOne
    @JoinColumn(name = "reviewer_id", nullable = false)
    private User reviewer;

    // 出品者（被評価者）
    @ManyToOne
    @JoinColumn(name = "seller_id", nullable = false)
    private User seller;
```

```
    // 対象商品
    @ManyToOne
    @JoinColumn(name = "item_id", nullable = false)
    private Item item;

    // 評価点（1〜5 を想定）
    @Column(nullable = false)
    private Integer rating;

    // コメント本文（任意）→TEXT 型
    @Column(columnDefinition = "TEXT")
    private String comment;

    // 作成日時（既定は現在時刻）
    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt = LocalDateTime.now();
}
```

User.java（ユーザーエンティティ。権限/有効フラグ/LINE トークンを保持）

```
// パッケージ宣言
package com.example.fleamarketsystem.entity;

// JPA インポート
import jakarta.persistence.*;
// Lombok インポート
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

// JPA エンティティ宣言
@Entity
// テーブル名 users を使用
@Table(name = "users")
// Lombok：getter/setter 等
@Data
// Lombok：デフォルトコンストラクタ
@NoArgsConstructor
// Lombok：全フィールドコンストラクタ
@AllArgsConstructor
public class User {
    // 主キー
    @Id
    // 自動採番（IDENTITY）
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // 表示名（必須）
    @Column(nullable = false)
    private String name;

    // ログイン ID に使用。ユニーク制約
    @Column(unique = true)
    private String email;

    // ハッシュ化されたパスワード（必須）
    @Column(nullable = false)
    private String password;

    // 役割（USER / ADMIN）（必須）
    @Column(nullable = false)
    private String role;

    // LINE Notify のアクセストークン（任意）
    @Column(name = "line_notify_token")
    private String lineNotifyToken;
```

```
    // アカウントの有効/無効フラグ。初期値は true（有効）
    @Column(nullable = false)
    private boolean enabled = true; // New field
}
```

10.4 パッケージ: repository

AppOrderRepository.java（注文リポジトリ：買い手/出品者別の一覧取得を提供）

```
// リポジトリのパッケージ
package com.example.fleamarketsystem.repository;

// エンティティと関連型の import
import com.example.fleamarketsystem.entity.AppOrder;
import com.example.fleamarketsystem.entity.User;
// Spring Data JPA の import
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

// コレクションや Optional 用
import java.util.List;
import java.util.Optional;

@Repository
public interface AppOrderRepository extends JpaRepository<AppOrder, Long> {
    // 買い手で注文一覧を取得
    List<AppOrder> findByBuyer(User buyer);
    // 出品者で注文一覧を取得（Item の seller 経由）
    List<AppOrder> findByItem_Seller(User seller);
}
```

CategoryRepository.java（カテゴリの CRUD と名称検索を提供）

```
// パッケージ宣言
package com.example.fleamarketsystem.repository;

// エンティティ Category を扱うためのインポート
import com.example.fleamarketsystem.entity.Category;
// Spring Data JPA の基底インタフェース
import org.springframework.data.jpa.repository.JpaRepository;
// コンポーネントスキャン対象にするためのアノテーション
import org.springframework.stereotype.Repository;

// Optional を返すメソッドで利用
import java.util.Optional;

// リポジトリ宣言：Category エンティティ + 主キー型 Long
@Repository
public interface CategoryRepository extends JpaRepository<Category, Long> {
    // カテゴリ名から一件検索（ユニーク想定のため Optional）
    Optional<Category> findByName(String name);
}
```

ChatRepository.java（チャットメッセージの取得。商品ごとに作成日時昇順）

```
// パッケージ宣言
package com.example.fleamarketsystem.repository;

// エンティティのインポート
import com.example.fleamarketsystem.entity.Chat;
import com.example.fleamarketsystem.entity.Item;
// Spring Data JPA
import org.springframework.data.jpa.repository.JpaRepository;
// リポジトリ・ステレオタイプ
import org.springframework.stereotype.Repository;

// 取得結果のリスト型
import java.util.List;

// Chat エンティティ用のリポジトリ
@Repository
public interface ChatRepository extends JpaRepository<Chat, Long> {
    // 指定商品のチャット履歴を作成日時昇順で取得
    List<Chat> findByItemOrderByCreatedAtAsc(Item item);
}
```

FavoriteItemRepository.java（お気に入りの追加/重複チェック/一覧取得）

```
// パッケージ宣言
package com.example.fleamarketsystem.repository;

// エンティティのインポート
import com.example.fleamarketsystem.entity.FavoriteItem;
import com.example.fleamarketsystem.entity.Item;
import com.example.fleamarketsystem.entity.User;
// Spring Data JPA
import org.springframework.data.jpa.repository.JpaRepository;
// リポジトリアノテーション
import org.springframework.stereotype.Repository;

// コレクション/Optional
import java.util.List;
import java.util.Optional;

// FavoriteItem エンティティ用のリポジトリ
@Repository
public interface FavoriteItemRepository extends JpaRepository<FavoriteItem, Long> {
    // ユーザーと商品で一意に検索（ユニーク制約と対応）
    Optional<FavoriteItem> findByUserAndItem(User user, Item item);
    // ユーザーのお気に入り商品を一覧取得
    List<FavoriteItem> findByUser(User user);
    // 既にお気に入り済みか存在チェック（二重登録防止用）
    boolean existsByUserAndItem(User user, Item item);
}
```

ItemRepository.java（商品検索/カテゴリ絞り込み/公開ステータスでのページング取得）

```
// パッケージ宣言
package com.example.fleamarketsystem.repository;

// エンティティのインポート
import com.example.fleamarketsystem.entity.Item;
import com.example.fleamarketsystem.entity.User;
// ページング用の型
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
// Spring Data JPA
import org.springframework.data.jpa.repository.JpaRepository;
// リポジトリアノテーション
import org.springframework.stereotype.Repository;

// 一覧取得で使用
import java.util.List;

// Itemエンティティのリポジトリ
@Repository
public interface ItemRepository extends JpaRepository<Item, Long> {
    // 名前の部分一致 + ステータスでページング検索（大文字小文字無視）
    Page<Item> findByNameContainingIgnoreCaseAndStatus(String name, String status, Pageable
pageable);
    // カテゴリID + ステータスでページング検索
    Page<Item> findByCategoryIdAndStatus(Long categoryId, String status, Pageable pageable);
    // 名前の部分一致 + カテゴリID + ステータスでページング検索
    Page<Item> findByNameContainingIgnoreCaseAndCategoryIdAndStatus(String name, Long
categoryId, String status, Pageable pageable);
    // ステータスのみでページング取得（公開中一覧など）
    Page<Item> findByStatus(String status, Pageable pageable);
    // 出品者ごとの商品一覧
    List<Item> findBySeller(User seller);
}
```

ReviewRepository.java（レビューの検索：出品者別/注文別/レビューワ別）

```
// パッケージ宣言
package com.example.fleamarketsystem.repository;

// コレクション/Optional
import java.util.List;
import java.util.Optional;

// Spring Data JPA
import org.springframework.data.jpa.repository.JpaRepository;
// リポジトリアノテーション
import org.springframework.stereotype.Repository;

// エンティティのインポート
import com.example.fleamarketsystem.entity.Review;
import com.example.fleamarketsystem.entity.User;

// Review エンティティのリポジトリ
@Repository
public interface ReviewRepository extends JpaRepository<Review, Long> {
        // 出品者に紐づくレビュー一覧を取得
        List<Review> findBySeller(User seller);

        // 注文 ID に紐づくレビューを一件取得（レビューは注文に 1 件）
        Optional<Review> findByOrderId(Long orderId);

        // レビューワ（投稿者）別のレビュー一覧を取得
        List<Review> findByReviewer(User reviewer); // Add this line
}
```

UserRepository.java（ユーザーのメール検索を提供）

```
// パッケージ宣言
package com.example.fleamarketsystem.repository;

// エンティティのインポート
import com.example.fleamarketsystem.entity.User;
// Spring Data JPA
import org.springframework.data.jpa.repository.JpaRepository;
// リポジトリアノテーション
import org.springframework.stereotype.Repository;

// Optional を返すために利用
import java.util.Optional;

// User エンティティのリポジトリ
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    // メールアドレスでユーザーを検索（ログイン/認可で使用）
    Optional<User> findByEmail(String email);
}
```

10.5 パッケージ: service

AppOrderService.java（決済と注文の厳密ひも付け／通知／集計）

```
// サービスのパッケージ
package com.example.fleamarketsystem.service;

// 必要なエンティティ/リポジトリ
import com.example.fleamarketsystem.entity.AppOrder;
import com.example.fleamarketsystem.entity.Item;
import com.example.fleamarketsystem.entity.User;
import com.example.fleamarketsystem.repository.ItemRepository;
import com.example.fleamarketsystem.repository.AppOrderRepository;
// Stripe 型
import com.stripe.exception.StripeException;
import com.stripe.model.PaymentIntent;
// Spring 注釈
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

// 金額・日付・コレクション
import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class AppOrderService {

    // リポジトリと周辺サービス
    private final AppOrderRepository appOrderRepository;
    private final ItemRepository itemRepository;
    private final ItemService itemService;
    private final StripeService stripeService;
    private final LineNotifyService lineNotifyService;

    // 依存の注入
    public AppOrderService(AppOrderRepository appOrderRepository, ItemRepository
itemRepository, ItemService itemService, StripeService stripeService, LineNotifyService
lineNotifyService) {
        // 各依存をフィールドに保持
        this.appOrderRepository = appOrderRepository;
        this.itemRepository = itemRepository;
        this.itemService = itemService;
        this.stripeService = stripeService;
        this.lineNotifyService = lineNotifyService;
    }
```

```
    // 購入開始：PaymentIntent 作成＋注文を“決済待ち”で作成（PaymentIntent ID を保存）
    @Transactional
    public PaymentIntent initiatePurchase(Long itemId, User buyer) throws StripeException {
        // 商品を取得（なければ 400）
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new IllegalArgumentException("Item not found"));
        // すでに売却済みならエラー
        if (!"出品中".equals(item.getStatus())) {
            throw new IllegalStateException("Item is not available for purchase.");
        }
        // Stripe へ PaymentIntent 作成（JPY は最小単位が 1 円のため create 側で考慮）
        PaymentIntent paymentIntent = stripeService.createPaymentIntent(item.getPrice(),
"jpy", "購入: " + item.getName());
        // 注文を“決済待ち”で作成し、PaymentIntent ID を確実に保存
        AppOrder appOrder = new AppOrder();
        // 商品を紐付け
        appOrder.setItem(item);
        // 買い手を紐付け
        appOrder.setBuyer(buyer);
        // 金額を固定
        appOrder.setPrice(item.getPrice());
        // ステータスを決済待ちへ
        appOrder.setStatus("決済待ち");
        // PaymentIntent ID を保存（これで後続完了時に 1 件特定できる）
        appOrder.setPaymentIntentId(paymentIntent.getId());
        // 作成日時
        appOrder.setCreatedAt(LocalDateTime.now());
        // DB へ保存
        appOrderRepository.save(appOrder);
        // フロントへ client_secret 等を返すため Intent を返却
        return paymentIntent;
    }

    // 決済完了：PaymentIntent ID で 1 件を厳密に取得して確定処理
    @Transactional
    public AppOrder completePurchase(String paymentIntentId) throws StripeException {
        // Stripe から Intent の最新状態を取得
        PaymentIntent paymentIntent = stripeService.retrievePaymentIntent(paymentIntentId);
        // 成功以外はエラー
        if (!"succeeded".equals(paymentIntent.getStatus())) {
            throw new IllegalStateException("Payment not succeeded. Status: " +
paymentIntent.getStatus());
        }
        // 保存済みの注文を PaymentIntent ID で 1 件特定（ここが安全化の肝）
        AppOrder appOrder = appOrderRepository.findByPaymentIntentId(paymentIntentId)
                .orElseThrow(() -> new IllegalStateException("Order for PaymentIntent not
found."));
        // 既に確定済みなら冪等に成功扱い
        if ("購入済".equals(appOrder.getStatus()) || "発送済".equals(appOrder.getStatus())) {
```

```
            // そのまま返す（再通知などはしない）
            return appOrder;
        }
        // ステータスを購入済へ
        appOrder.setStatus("購入済");
        // 商品を売却済みに更新（在庫1想定）
        itemService.markItemAsSold(appOrder.getItem().getId());
        // 保存
        AppOrder savedOrder = appOrderRepository.save(appOrder);
        // 売り手がLine通知トークンを持っていれば通知
        if (savedOrder.getItem().getSeller().getLineNotifyToken() != null) {
            String message = String.format("¥n商品が購入されました！¥n商品名: %s¥n購入
者: %s¥n価格: ¥%s",
                    savedOrder.getItem().getName(),
                    savedOrder.getBuyer().getName(),
                    savedOrder.getPrice());
            // 例外は内側で処理してログ出し

lineNotifyService.sendMessage(savedOrder.getItem().getSeller().getLineNotifyToken(), message);
        }
        // 確定した注文を返す
        return savedOrder;
    }

    // すべての注文取得（管理者ダッシュボード等）
    public List<AppOrder> getAllOrders() {
        // 全件を返す
        return appOrderRepository.findAll();
    }

    // 買い手別の注文一覧
    public List<AppOrder> getOrdersByBuyer(User buyer) {
        // リポジトリ委譲
        return appOrderRepository.findByBuyer(buyer);
    }

    // 売り手別の注文一覧
    public List<AppOrder> getOrdersBySeller(User seller) {
        // リポジトリ委譲
        return appOrderRepository.findByItem_Seller(seller);
    }

    // 発送処理：ステータスと通知
    @Transactional
    public void markOrderAsShipped(Long orderId) {
        // 注文取得（なければ404相当）
        AppOrder appOrder = appOrderRepository.findById(orderId)
                .orElseThrow(() -> new IllegalArgumentException("Order not found"));
```

```
        // ステータス更新
        appOrder.setStatus("発送済");
        // 保存
        AppOrder savedOrder = appOrderRepository.save(appOrder);
        // 買い手にLINE通知があれば送信
        if (savedOrder.getBuyer().getLineNotifyToken() != null) {
            String message = String.format("¥n購入した商品が発送されました！¥n商品名: %s¥n出
品者: %s",
                    savedOrder.getItem().getName(),
                    savedOrder.getItem().getSeller().getName());
            // 送信試行（失敗はログのみ）
            lineNotifyService.sendMessage(savedOrder.getBuyer().getLineNotifyToken(),
message);
        }
    }

    // IDで1件取得
    public Optional<AppOrder> getOrderById(Long orderId) {
        // Optionalで返す
        return appOrderRepository.findById(orderId);
    }

    // 最新の“購入済”注文ID（レビュー画面遷移用）
    public Optional<Long> getLatestCompletedOrderId() {
        // ステータスが購入済の中で最大IDを返す
        return appOrderRepository.findAll().stream()
                .filter(o -> "購入済".equals(o.getStatus()))
                .map(AppOrder::getId)
                .max(Long::compare);
    }

    // 指定期間の売上合計
    public BigDecimal getTotalSales(LocalDate startDate, LocalDate endDate) {
        // 期間内の購入済/発送済のみ合計
        return appOrderRepository.findAll().stream()
                .filter(order -> order.getStatus().equals("購入済") ||
order.getStatus().equals("発送済"))
                .filter(order ->
order.getCreatedAt().toLocalDate().isAfter(startDate.minusDays(1)) &&
order.getCreatedAt().toLocalDate().isBefore(endDate.plusDays(1)))
                .map(AppOrder::getPrice)
                .reduce(BigDecimal.ZERO, BigDecimal::add);
    }

    // 指定期間のステータス別件数
    public Map<String, Long> getOrderCountByStatus(LocalDate startDate, LocalDate endDate) {
        // 作成日で期間フィルタしてグルーピング
        return appOrderRepository.findAll().stream()
```

```
                .filter(order ->
order.getCreatedAt().toLocalDate().isAfter(startDate.minusDays(1)) &&
order.getCreatedAt().toLocalDate().isBefore(endDate.plusDays(1)))
                .collect(Collectors.groupingBy(AppOrder::getStatus, Collectors.counting()));
    }
}
```

CategoryService.java（カテゴリの CRUD と検索。名称一意の前提）

```
// サービスクラスのパッケージを宣言
package com.example.fleamarketsystem.service;
// カテゴリエンティティを扱うための import
import com.example.fleamarketsystem.entity.Category;
// リポジトリ IF を import
import com.example.fleamarketsystem.repository.CategoryRepository;
// DI 対象サービスを示すアノテーションを import
import org.springframework.stereotype.Service;
// 一覧返却に使う List を import
import java.util.List;
// Optional を返すために import
import java.util.Optional;

// サービス層として登録
@Service
public class CategoryService {

    // カテゴリリポジトリの参照
    private final CategoryRepository categoryRepository;

    // 依存性をコンストラクタで注入
    public CategoryService(CategoryRepository categoryRepository) {
        // フィールドへ設定
        this.categoryRepository = categoryRepository;
    }

    // すべてのカテゴリを取得
    public List<Category> getAllCategories() {
        // 全件取得を委譲
        return categoryRepository.findAll();
    }

    // 主キーでカテゴリを取得
    public Optional<Category> getCategoryById(Long id) {
        // Optional をそのまま返す
        return categoryRepository.findById(id);
    }

    // 名称でカテゴリを取得（名称は一意前提）
    public Optional<Category> getCategoryByName(String name) {
        // 名称検索を委譲
        return categoryRepository.findByName(name);
    }

    // 新規/更新保存
    public Category saveCategory(Category category) {
        // save に委譲
        return categoryRepository.save(category);
```

```
    }

    // 削除
    public void deleteCategory(Long id) {
        // ID指定で削除
        categoryRepository.deleteById(id);
    }
}
```

ChatService.java（商品別のメッセージ投稿／取得。相手への LINE 通知を送信）

```
// サービスクラスのパッケージを宣言
package com.example.fleamarketsystem.service;
// チャットエンティティを扱うための import
import com.example.fleamarketsystem.entity.Chat;
// 商品エンティティを扱うための import
import com.example.fleamarketsystem.entity.Item;
// ユーザエンティティを扱うための import
import com.example.fleamarketsystem.entity.User;
// チャットの検索/保存に使うリポジトリを import
import com.example.fleamarketsystem.repository.ChatRepository;
// 商品の存在確認に使うリポジトリを import
import com.example.fleamarketsystem.repository.ItemRepository;
// DI 対象のサービスであることを示すアノテーションを import
import org.springframework.stereotype.Service;
// 送信日時を記録するために LocalDateTime を import
import java.time.LocalDateTime;
// 履歴表示に使う List を import
import java.util.List;

// サービス層として登録
@Service
public class ChatService {

    // チャットリポジトリの参照
    private final ChatRepository chatRepository;
    // 商品リポジトリの参照
    private final ItemRepository itemRepository;
    // LINE 通知サービスの参照
    private final LineNotifyService lineNotifyService;

    // 依存性をコンストラクタで注入
    public ChatService(ChatRepository chatRepository, ItemRepository itemRepository,
LineNotifyService lineNotifyService) {
        // フィールドへ設定
        this.chatRepository = chatRepository;
        // フィールドへ設定
        this.itemRepository = itemRepository;
        // フィールドへ設定
        this.lineNotifyService = lineNotifyService;
    }

    // 商品 ID に紐づくチャット履歴を昇順で取得
    public List<Chat> getChatMessagesByItem(Long itemId) {
        // 商品の存在を確認（なければ 400 相当の例外）
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new IllegalArgumentException("Item not found"));
```

```
        // 作成日時昇順でリストを返す
        return chatRepository.findByItemOrderByCreatedAtAsc(item);
    }

    // メッセージ送信：保存して相手にLINE通知（可能なら）を行う
    public Chat sendMessage(Long itemId, User sender, String message) {
        // 対象商品を取得（存在しなければ例外）
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new IllegalArgumentException("Item not found"));
        // 新規チャットエンティティを構築
        Chat chat = new Chat();
        // 商品を紐づけ
        chat.setItem(item);
        // 送信者を紐づけ
        chat.setSender(sender);
        // 本文を設定
        chat.setMessage(message);
        // 現在時刻で送信時刻を設定
        chat.setCreatedAt(LocalDateTime.now());
        // 保存して永続化
        Chat savedChat = chatRepository.save(chat);
        // 簡易実装：受信者を出品者とみなして通知（詳細な相手判定は拡張で対応）
        User receiver = item.getSeller();
        // 受信者が通知トークンを設定していれば通知を送る
        if (receiver != null && receiver.getLineNotifyToken() != null) {
            // 通知本文を作成
            String notificationMessage = String.format("¥n商品「%s」に関する新しいメッセージ
が届きました！¥n送信者: %s¥nメッセージ: %s",
                    item.getName(),
                    sender.getName(),
                    message);
            // LINE Notifyへ送信
            lineNotifyService.sendMessage(receiver.getLineNotifyToken(), notificationMessage);
        }
        // 保存結果を返却
        return savedChat;
    }
}
```

CloudinaryService（画像アップロード/削除の薄いラッパ。URL から public_id を推定して削除）

```
// サービスクラスのパッケージを宣言
package com.example.fleamarketsystem.service;
// Cloudinary の Java SDK のエントリポイントを import
import com.cloudinary.Cloudinary;
// アップロード/削除で使うユーティリティを import
import com.cloudinary.utils.ObjectUtils;
// 設定値を外部から注入するためのアノテーションを import
import org.springframework.beans.factory.annotation.Value;
// DI 対象のサービスであることを示すアノテーションを import
import org.springframework.stereotype.Service;
// Spring のファイルアップロード表現を import
import org.springframework.web.multipart.MultipartFile;
// I/O 例外処理のための import
import java.io.IOException;
// アップロード結果を受け取る Map を import
import java.util.Map;

// サービス層として登録
@Service
public class CloudinaryService {

    // Cloudinary クライアントの参照
    private final Cloudinary cloudinary;

    // 必要な認証情報をコンストラクタインジェクションで受け取る
    public CloudinaryService(
            // クラウド名を application.properties から注入
            @Value("${cloudinary.cloud-name}") String cloudName,
            // API キーを注入
            @Value("${cloudinary.api-key}") String apiKey,
            // API シークレットを注入
            @Value("${cloudinary.api-secret}") String apiSecret) {
        // 渡された資格情報で Cloudinary クライアントを初期化
        cloudinary = new Cloudinary(ObjectUtils.asMap(
                "cloud_name", cloudName,
                "api_key", apiKey,
                "api_secret", apiSecret));
    }

    // 画像をアップロードして公開 URL を返す（空ファイルは null）
    public String uploadFile(MultipartFile file) throws IOException {
        // アップロードなしのケースは null を返す
        if (file.isEmpty()) {
            return null;
        }
```

```
        // バイト配列をそのままアップロード（オプションは既定）
        Map uploadResult = cloudinary.uploader().upload(file.getBytes(),
ObjectUtils.emptyMap());
        // 返却 Map から公開 URL を取り出して返す
        return uploadResult.get("url").toString();
    }

    // Cloudinary 上のリソースを削除（URL から public_id を推定）
    public void deleteFile(String publicId) throws IOException {
        // URL を/で分割して末尾のファイル名を取り出す
        String[] parts = publicId.split("/");
        // 配列末尾＝ファイル名部分を取得
        String fileName = parts[parts.length - 1];
        // 拡張子を除いた public_id を推定
        String publicIdWithoutExtension = fileName.substring(0, fileName.lastIndexOf('.'));
        // public_id を指定して削除 API を呼び出す
        cloudinary.uploader().destroy(publicIdWithoutExtension, ObjectUtils.emptyMap());
    }
}
```

FavoriteService.java（お気に入りの追加/解除/判定/一覧取得。二重登録と不正解除を防止）

```
// サービスのパッケージ宣言
package com.example.fleamarketsystem.service;
// お気に入りエンティティの import
import com.example.fleamarketsystem.entity.FavoriteItem;
// 商品エンティティの import
import com.example.fleamarketsystem.entity.Item;
// ユーザエンティティの import
import com.example.fleamarketsystem.entity.User;
// お気に入り用リポジトリの import
import com.example.fleamarketsystem.repository.FavoriteItemRepository;
// 商品リポジトリの import
import com.example.fleamarketsystem.repository.ItemRepository;
// サービスアノテーションの import
import org.springframework.stereotype.Service;
// トランザクション境界を宣言するための import
import org.springframework.transaction.annotation.Transactional;
// コレクション操作のための import
import java.util.List;
// Stream で変換するための import
import java.util.stream.Collectors;
// サービス層としての宣言
@Service
public class FavoriteService {
    // リポジトリの参照（お気に入り）
    private final FavoriteItemRepository favoriteItemRepository;
    // リポジトリの参照（商品）
    private final ItemRepository itemRepository;
    // 依存性をコンストラクタで注入
    public FavoriteService(FavoriteItemRepository favoriteItemRepository, ItemRepository
itemRepository) {
        // お気に入りリポジトリを設定
        this.favoriteItemRepository = favoriteItemRepository;
        // 商品リポジトリを設定
        this.itemRepository = itemRepository;
    }
    // お気に入り追加（同一ユーザ×商品は一意）
    @Transactional
    public FavoriteItem addFavorite(User user, Long itemId) {
        // 商品存在チェック
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new IllegalArgumentException("Item not found"));
        // 既に登録済みならエラー
        if (favoriteItemRepository.existsByUserAndItem(user, item)) {
            // 2 重登録を防止
            throw new IllegalStateException("Item is already favorited by this user.");
        }
```

```
        // 新規のお気に入りエンティティ作成
        FavoriteItem favoriteItem = new FavoriteItem();
        // ユーザを設定
        favoriteItem.setUser(user);
        // 商品を設定
        favoriteItem.setItem(item);
        // 保存して返す
        return favoriteItemRepository.save(favoriteItem);
    }
    // お気に入り解除
    @Transactional
    public void removeFavorite(User user, Long itemId) {
        // 商品取得（存在チェック）
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new IllegalArgumentException("Item not found"));
        // ユーザ×商品でお気に入りを取得（なければエラー）
        FavoriteItem favoriteItem = favoriteItemRepository.findByUserAndItem(user, item)
                .orElseThrow(() -> new IllegalStateException("Favorite not found."));
        // 削除実行
        favoriteItemRepository.delete(favoriteItem);
    }
    // お気に入りかどうかの判定
    public boolean isFavorited(User user, Long itemId) {
        // 商品取得（存在チェック）
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new IllegalArgumentException("Item not found"));
        // 存在するかどうかを返す
        return favoriteItemRepository.existsByUserAndItem(user, item);
    }
    // ユーザのお気に入り商品一覧を返す
    public List<Item> getFavoriteItemsByUser(User user) {
        // FavoriteItem から Item へマッピング
        return favoriteItemRepository.findByUser(user).stream()
                .map(FavoriteItem::getItem)
                .collect(Collectors.toList());
    }
}
```

ItemService.java（商品検索／保存／画像連携／削除／売却確定。検索は公開中のみ）

```
// サービスクラスのパッケージを宣言
package com.example.fleamarketsystem.service;
// Item エンティティを扱うための import
import com.example.fleamarketsystem.entity.Item;
// User エンティティを扱うための import
import com.example.fleamarketsystem.entity.User;
// ページング付き検索で利用するリポジトリを import
import com.example.fleamarketsystem.repository.ItemRepository;
// ページング条件の型を import
import org.springframework.data.domain.Page;
// ページングリクエスト生成用の型を import
import org.springframework.data.domain.PageRequest;
// ページングインターフェイスの型を import
import org.springframework.data.domain.Pageable;
// DI 対象のサービスであることを示すアノテーションを import
import org.springframework.stereotype.Service;
// ファイルアップロードに使う MultipartFile を import
import org.springframework.web.multipart.MultipartFile;
// I/O 例外処理に必要な import
import java.io.IOException;
// 金額保持のために BigDecimal を import
import java.math.BigDecimal;
// 一覧返却で使う List を import
import java.util.List;
// Optional で安全に扱うための import
import java.util.Optional;

// サービス層として登録
@Service
public class ItemService {

    // 商品リポジトリの参照
    private final ItemRepository itemRepository;
    // カテゴリ関連のユースケースに備えてサービス参照を保持
    private final CategoryService categoryService;
    // 画像アップロード/削除のための Cloudinary サービス参照
    private final CloudinaryService cloudinaryService;

    // 依存性はコンストラクタで注入
    public ItemService(ItemRepository itemRepository, CategoryService categoryService,
CloudinaryService cloudinaryService) {
        // フィールドへ商品リポジトリを設定
        this.itemRepository = itemRepository;
        // フィールドへカテゴリサービスを設定
        this.categoryService = categoryService;
        // フィールドへ Cloudinary サービスを設定
        this.cloudinaryService = cloudinaryService;
```

```
    }

    // 商品検索：キーワード/カテゴリ/ページングを組み合わせ、公開中のみ返す
    public Page<Item> searchItems(String keyword, Long categoryId, int page, int size) {
        // ページング指定を生成
        Pageable pageable = PageRequest.of(page, size);
        // キーワードとカテゴリ両方指定時の検索
        if (keyword != null && !keyword.isEmpty() && categoryId != null) {
            // 名前LIKE×カテゴリ×出品中で検索
            return
itemRepository.findByNameContainingIgnoreCaseAndCategoryIdAndStatus(keyword, categoryId, "出品
中", pageable);
        // キーワードのみ指定時の検索
        } else if (keyword != null && !keyword.isEmpty()) {
            // 名前LIKE×出品中で検索
            return itemRepository.findByNameContainingIgnoreCaseAndStatus(keyword, "出品中",
pageable);
        // カテゴリのみ指定時の検索
        } else if (categoryId != null) {
            // カテゴリ×出品中で検索
            return itemRepository.findByCategoryIdAndStatus(categoryId, "出品中", pageable);
        // 条件未指定時のデフォルト検索
        } else {
            // 出品中のみ全件ページングで返す
            return itemRepository.findByStatus("出品中", pageable);
        }
    }

    // 全商品一覧を返す（管理用など）
    public List<Item> getAllItems() {
        // リポジトリの全件取得を委譲
        return itemRepository.findAll();
    }

    // 主キーで商品を取得
    public Optional<Item> getItemById(Long id) {
        // Optionalをそのまま返却
        return itemRepository.findById(id);
    }

    // 商品保存：必要なら画像をCloudinaryへアップロードしてURLを保存
    public Item saveItem(Item item, MultipartFile imageFile) throws IOException {
        // 画像が添付されている場合にのみアップロード処理を実行
        if (imageFile != null && !imageFile.isEmpty()) {
            // Cloudinaryへアップロードし URLを受け取る
            String imageUrl = cloudinaryService.uploadFile(imageFile);
            // 画像URLをエンティティへ設定
            item.setImageUrl(imageUrl);
        }
```

```
        // 商品を保存して返す
        return itemRepository.save(item);
    }

    // 商品削除：Cloudinary 上の画像も可能なら削除してから DB 削除
    public void deleteItem(Long id) {
        // まず対象商品を取得し、存在する場合のみ削除処理を進める
        itemRepository.findById(id).ifPresent(item -> {
            // 画像 URL がある場合は Cloudinary 側の削除を試みる
            if (item.getImageUrl() != null) {
                try {
                    // URL から public id を推定し削除
                    cloudinaryService.deleteFile(item.getImageUrl());
                } catch (IOException e) {
                    // 画像削除失敗は致命ではないためログ出力に留める
                    System.err.println("Failed to delete image from Cloudinary: " +
e.getMessage());
                }
            }
            // 最後に DB から商品レコードを削除
            itemRepository.deleteById(id);
        });
    }

    // 出品者の出品一覧を取得
    public List<Item> getItemsBySeller(User seller) {
        // seller 条件で検索
        return itemRepository.findBySeller(seller);
    }

    // 売却確定：商品ステータスを売却済へ変更
    public void markItemAsSold(Long itemId) {
        // 商品を取得して存在する場合のみ更新
        itemRepository.findById(itemId).ifPresent(item -> {
            // ステータスを売却済に変更
            item.setStatus("売却済");
            // 変更を保存
            itemRepository.save(item);
        });
    }
}
```

LineNotifyService.java（アクセストークンでメッセージを POST。失敗はログに留める）

```
// サービスクラスのパッケージを宣言
package com.example.fleamarketsystem.service;
// 外部設定から既定 URL を読み込むために@Value を import
import org.springframework.beans.factory.annotation.Value;
// HTTP リクエストを構築するための型を import
import org.springframework.http.HttpEntity;
// HTTP ヘッダを構築するための型を import
import org.springframework.http.HttpHeaders;
// コンテントタイプの列挙を import
import org.springframework.http.MediaType;
// DI 対象のサービスであることを示すアノテーションを import
import org.springframework.stereotype.Service;
// フォームパラメータを組み立てるためのユーティリティを import
import org.springframework.util.LinkedMultiValueMap;
// フォームのマップ表現を import
import org.springframework.util.MultiValueMap;
// HTTP クライアントとして RestTemplate を import
import org.springframework.web.client.RestTemplate;

// サービス層として登録
@Service
public class LineNotifyService {

    // API エンドポイント URL（未設定ならデフォルト値を使う）
    @Value("${line.notify.api.url:https://notify-api.line.me/api/notify}")
    private String lineNotifyApiUrl;

    // HTTP クライアントの参照
    private final RestTemplate restTemplate;

    // 依存性をコンストラクタで注入
    public LineNotifyService(RestTemplate restTemplate) {
        // フィールドへ設定
        this.restTemplate = restTemplate;
    }

    // アクセストークンと本文を受け取り、LINE Notify へ送信
    public void sendMessage(String accessToken, String message) {
        // リクエストヘッダを構築
        HttpHeaders headers = new HttpHeaders();
        // フォーム URL エンコードを指定
        headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
        // Bearer トークンをセット
        headers.setBearerAuth(accessToken);

        // フォームボディを構築
        MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
```

```
        // message キーに本文を格納
        map.add("message", message);

        // ヘッダ＋本文でエンティティを生成
        HttpEntity<MultiValueMap<String, String>> request = new HttpEntity<>(map, headers);

        // 送信試行（失敗はログに残して握りつぶす）
        try {
            // POST で API へ投げる
            restTemplate.postForEntity(lineNotifyApiUrl, request, String.class);
            // 成功ログを標準出力へ
            System.out.println("LINE Notify message sent successfully.");
        } catch (Exception e) {
            // 失敗時は標準エラーへ出力して処理継続
            System.err.println("Failed to send LINE Notify message: " + e.getMessage());
        }
    }
}
```

ReviewService.java（レビュー投稿・取得・平均算出。購入者本人性と重複レビューを検証）

```
// サービスのパッケージ宣言
package com.example.fleamarketsystem.service;
// 注文エンティティの import
import com.example.fleamarketsystem.entity.AppOrder;
// レビューエンティティの import
import com.example.fleamarketsystem.entity.Review;
// ユーザエンティティの import
import com.example.fleamarketsystem.entity.User;
// 注文リポジトリの import
import com.example.fleamarketsystem.repository.AppOrderRepository;
// レビューリポジトリの import
import com.example.fleamarketsystem.repository.ReviewRepository;
// サービスアノテーションの import
import org.springframework.stereotype.Service;
// トランザクション境界の宣言
import org.springframework.transaction.annotation.Transactional;
// リストを扱うための import
import java.util.List;
// 平均計算の戻り値に OptionalDouble を使うための import
import java.util.OptionalDouble;
// サービス層の宣言
@Service
public class ReviewService {
    // レビューリポジトリの参照
    private final ReviewRepository reviewRepository;
    // 注文リポジトリの参照
    private final AppOrderRepository appOrderRepository;
    // 依存性をコンストラクタ注入
    public ReviewService(ReviewRepository reviewRepository, AppOrderRepository
appOrderRepository) {
        // レビューリポジトリを設定
        this.reviewRepository = reviewRepository;
        // 注文リポジトリを設定
        this.appOrderRepository = appOrderRepository;
    }
    // レビュー投稿（買い手のみ、1 注文 1 レビュー）
    @Transactional
    public Review submitReview(Long orderId, User reviewer, int rating, String comment) {
        // 注文を取得（存在しなければ 400 相当）
        AppOrder order = appOrderRepository.findById(orderId)
                .orElseThrow(() -> new IllegalArgumentException("Order not found."));
        // 注文の買い手と同一ユーザか検証
        if (!order.getBuyer().getId().equals(reviewer.getId())) {
            // 買い手以外は拒否
            throw new IllegalStateException("Only the buyer can review this order.");
        }
```

```
        // 既にレビュー済みかを検査
        if (reviewRepository.findByOrderId(orderId).isPresent()) {
            // 二重レビューを防ぐ
            throw new IllegalStateException("This order has already been reviewed.");
        }
        // 新しいレビューエンティティを構築
        Review review = new Review();
        // 注文を紐付け
        review.setOrder(order);
        // レビュワーを設定
        review.setReviewer(reviewer);
        // 出品者（注文の商品の出品者）を設定
        review.setSeller(order.getItem().getSeller());
        // 対象商品を設定
        review.setItem(order.getItem());
        // 評価点を設定
        review.setRating(rating);
        // コメントを設定
        review.setComment(comment);
        // 保存して返却
        return reviewRepository.save(review);
    }
    // 出品者に対するレビュー一覧を取得
    public List<Review> getReviewsBySeller(User seller) {
        // リポジトリに委譲
        return reviewRepository.findBySeller(seller);
    }
    // 出品者に対する平均評価を算出
    public OptionalDouble getAverageRatingForSeller(User seller) {
        // ストリームで平均を計算
        return reviewRepository.findBySeller(seller).stream()
                .mapToInt(Review::getRating)
                .average();
    }
    // あるレビュワーが書いたレビューを取得
    public List<Review> getReviewsByReviewer(User reviewer) {
        // リポジトリに委譲
        return reviewRepository.findByReviewer(reviewer);
    }
}
```

StripeService.java（責務：作成／取得のみ、金額は“円→最小単位”変換）

```
// サービスのパッケージ
package com.example.fleamarketsystem.service;

// Stripe SDK の import
import com.stripe.Stripe;
import com.stripe.exception.StripeException;
import com.stripe.model.PaymentIntent;
import com.stripe.param.PaymentIntentCreateParams;
// 設定値の受け取りと Spring サービス化
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

// 金額用
import java.math.BigDecimal;

@Service
public class StripeService {

    // コンストラクタでシークレットキーを初期化
    public StripeService(@Value("${stripe.api.secretKey}") String secretKey) {
        // Stripe SDK に API キーを設定（スレッドセーフ）
        Stripe.apiKey = secretKey;
    }

    // 支払い意図(PaymentIntent)を作成
    public PaymentIntent createPaymentIntent(BigDecimal amount, String currency, String
description) throws StripeException {
        // 通貨の最小単位へ変換（JPY なら 1 円→100 の係数不要だが Stripe は整数で受けるため×100
は不要、しかし他通貨に備え共通化）
        long value = "jpy".equalsIgnoreCase(currency) ? amount.longValue() :
amount.multiply(new BigDecimal(100)).longValue();
        // 作成パラメータをビルド
        PaymentIntentCreateParams params = PaymentIntentCreateParams.builder()
                // 金額（最小単位の整数）
                .setAmount(value)
                // 通貨コード
                .setCurrency(currency)
                // 説明
                .setDescription(description)
                // 自動支払い手段を有効化
                .setAutomaticPaymentMethods(
                        PaymentIntentCreateParams.AutomaticPaymentMethods.builder()
                                .setEnabled(true)
                                .build()
                )
                .build();
```

```
        // PaymentIntent を作成して返す
        return PaymentIntent.create(params);
    }

    // 既存の PaymentIntent を取得
    public PaymentIntent retrievePaymentIntent(String paymentIntentId) throws StripeException
{
        // ID から取得
        return PaymentIntent.retrieve(paymentIntentId);
    }
}
```

UserService.java（ユーザ CRUD と有効/無効切替。メール検索を提供）

```
// サービスクラスのパッケージを宣言
package com.example.fleamarketsystem.service;
// ユーザエンティティを扱うための import
import com.example.fleamarketsystem.entity.User;
// リポジトリ IF を import
import com.example.fleamarketsystem.repository.UserRepository;
// DI 対象のサービスを表すアノテーションを import
import org.springframework.stereotype.Service;
// 変更系でトランザクションを開始するためのアノテーションを import
import org.springframework.transaction.annotation.Transactional;
// 一覧返却に使う List を import
import java.util.List;
// Optional を返すための import
import java.util.Optional;

// サービス層として登録
@Service
public class UserService {

    // ユーザリポジトリの参照
    private final UserRepository userRepository;

    // 依存性をコンストラクタで注入
    public UserService(UserRepository userRepository) {
        // フィールドへ設定
        this.userRepository = userRepository;
    }

    // すべてのユーザを取得
    public List<User> getAllUsers() {
        // 全件取得を委譲
        return userRepository.findAll();
    }

    // 主キーでユーザを取得
    public Optional<User> getUserById(Long id) {
        // Optional を返す
        return userRepository.findById(id);
    }

    // メールアドレスでユーザを取得
    public Optional<User> getUserByEmail(String email) {
        // Optional を返す
        return userRepository.findByEmail(email);
    }
```

```
    // 新規/更新保存
    @Transactional
    public User saveUser(User user) {
        // save に委譲
        return userRepository.save(user);
    }

    // 削除
    @Transactional
    public void deleteUser(Long id) {
        // ID 指定で削除
        userRepository.deleteById(id);
    }

    // 有効/無効フラグのトグル
    @Transactional
    public void toggleUserEnabled(Long userId) {
        // ID でユーザを取得（なければ 400 相当の例外）
        User user = userRepository.findById(userId)
                .orElseThrow(() -> new IllegalArgumentException("User not found"));
        // 既存の属性は変更せず enabled だけ反転
        user.setEnabled(!user.isEnabled());
        // 保存して確定
        userRepository.save(user);
    }
}
```

10.6 テンプレート（resources/templates）

admin_dashboard.html（管理者ダッシュボード。管理メニュー＋最近の出品/注文一覧を表示）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- 文書のルート要素。言語を日本語、Thymeleaf と Spring Security Dialect の XMLNS を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- メタデータと CSS などの外部リソースを定義する領域 -->
<head>
    <!-- 文字エンコーディングを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- ブラウザタブに表示されるページタイトルを設定 -->
    <title>管理者ダッシュボード - フリマアプリ</title>
    <!-- アプリ共通のスタイルシートを読み込む（Thymeleaf でパス解決） -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面に表示される本体部分 -->
<body>
    <!-- レイアウト用の幅制限と余白・影を与えるコンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>管理者ダッシュボード</h1>
        <!-- ログイン中のユーザー名を Spring Security の認証情報から表示 -->
        <p>ようこそ、<span sec:authentication="name"></span>さん！（管理者）</p>

        <!-- 主要ナビゲーションを並べる領域 -->
        <div class="main-nav">
            <!-- 商品管理画面へのリンク -->
            <a th:href="@{/admin/items}" class="button">商品管理</a>
            <!-- ユーザー管理画面へのリンク -->
            <a th:href="@{/admin/users}" class="button">ユーザー管理</a>
            <!-- 統計画面へのリンク -->
            <a th:href="@{/admin/statistics}" class="button">統計画面</a>
            <!-- ログアウト POST フォーム（CSRF 等は Spring Security が自動付与） -->
            <form th:action="@{/logout}" method="post" style="display:inline;">
                <!-- ログアウト送信ボタン（見た目はセカンダリ） -->
                <button type="submit" class="button secondary">ログアウト</button>
            </form>
        </div>

        <!-- 最近の出品セクション見出し -->
        <h2>最近の出品</h2>
        <!-- 最近の出品を表形式で表示 -->
        <table>
            <!-- テーブルの列見出し -->
            <thead>
                <tr>
                    <th>商品名</th>
                    <th>出品者</th>
```

```
                    <th>価格</th>
                    <th>ステータス</th>
                </tr>
            </thead>
            <!-- テーブルの本体。items リストを Thymeleaf で展開 -->
            <tbody>
                <!-- recentItems をループし 1 行ずつ描画 -->
                <tr th:each="item : ${recentItems}">
                    <!-- 商品名を表示 -->
                    <td th:text="${item.name}"></td>
                    <!-- 出品者の表示名を表示 -->
                    <td th:text="${item.seller.name}"></td>
                    <!-- 価格をコンマ区切りで表示し、先頭に¥を付与 -->
                    <td th:text="'¥' + ${#numbers.formatDecimal(item.price, 0, 'COMMA', 0,
'POINT')}"></td>
                    <!-- 商品のステータス（出品中/売却済）を表示 -->
                    <td th:text="${item.status}"></td>
                </tr>
                <!-- 要素が空の場合のメッセージ行 -->
                <tr th:if="${#lists.isEmpty(recentItems)}">
                    <!-- 全列を結合して「最近の出品はありません」を表示 -->
                    <td colspan="4">最近の出品はありません。</td>
                </tr>
            </tbody>
        </table>

        <!-- 最近の注文セクション見出し -->
        <h2>最近の注文</h2>
        <!-- 最近の注文を表形式で表示 -->
        <table>
            <!-- テーブルの列見出し -->
            <thead>
                <tr>
                    <th>商品名</th>
                    <th>購入者</th>
                    <th>価格</th>
                    <th>ステータス</th>
                </tr>
            </thead>
            <!-- テーブル本体。recentOrders をループ表示 -->
            <tbody>
                <!-- recentOrders を 1 件ずつ行として描画 -->
                <tr th:each="order : ${recentOrders}">
                    <!-- 注文の商品名を表示 -->
                    <td th:text="${order.item.name}"></td>
                    <!-- 購入者の表示名を表示 -->
                    <td th:text="${order.buyer.name}"></td>
```

```
                    <!-- 価格をフォーマットして表示 -->
                    <td th:text="'¥' + ${#numbers.formatDecimal(order.price, 0, 'COMMA', 0,
'POINT')}"></td>
                    <!-- 注文ステータス（購入済/発送済）を表示 -->
                    <td th:text="${order.status}"></td>
                </tr>
                <!-- 要素が空の場合のメッセージ行 -->
                <tr th:if="${#lists.isEmpty(recentOrders)}">
                    <!-- 全列結合で「最近の注文はありません」を表示 -->
                    <td colspan="4">最近の注文はありません。</td>
                </tr>
            </tbody>
        </table>
    </div>
</body>
</html>
```

admin_items.html（管理者の全商品一覧と削除操作）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。日本語と Thymeleaf/Spring Security Dialect を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域（メタ・タイトル・スタイル） -->
<head>
    <!-- 文字コードを UTF-8 に指定 -->
    <meta charset="UTF-8">
    <!-- ページタイトルを設定 -->
    <title>管理者: 商品管理 - フリマアプリ</title>
    <!-- 共通スタイルシートを読み込む -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 表示本体 -->
<body>
    <!-- レイアウト用コンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>管理者: 商品管理</h1>
        <!-- 画面の説明文 -->
        <p>システム内の全ての商品を管理します。</p>

        <!-- 商品一覧テーブル -->
        <table>
            <!-- 列見出し -->
            <thead>
                <tr>
                    <th>ID</th>
                    <th>商品名</th>
                    <th>出品者</th>
                    <th>価格</th>
                    <th>カテゴリ</th>
                    <th>ステータス</th>
                    <th>操作</th>
                </tr>
            </thead>
            <!-- テーブル本体 -->
            <tbody>
                <!-- items をループして 1 行ずつ描画 -->
                <tr th:each="item : ${items}">
                    <!-- 商品 ID を表示 -->
                    <td th:text="${item.id}"></td>
                    <!-- 商品名を表示 -->
                    <td th:text="${item.name}"></td>
                    <!-- 出品者名を表示 -->
                    <td th:text="${item.seller.name}"></td>
```

```
                    <!-- 価格をフォーマットして表示 -->
                    <td th:text="'¥' + ${#numbers.formatDecimal(item.price, 0, 'COMMA', 0,
'POINT')}"></td>
                    <!-- カテゴリ名。null なら未分類と表示 -->
                    <td th:text="${item.category != null ? item.category.name : '未分類
'}"></td>
                    <!-- 商品ステータスを表示 -->
                    <td th:text="${item.status}"></td>
                    <!-- 操作列（削除ボタン） -->
                    <td>
                        <!-- 管理者による削除 POST を送信。確認ダイアログで誤操作を防止 -->
                        <form th:action="@{/admin/items/{id}/delete(id=${item.id})}"
method="post" style="display:inline;">
                            <!-- 削除ボタン（赤系スタイル）。confirm で最終確認 -->
                            <button type="submit" class="button danger" onclick="return
confirm('本当にこの商品を削除しますか？');">削除</button>
                        </form>
                    </td>
                </tr>
                <!-- items が空の場合の 1 行表示 -->
                <tr th:if="${#lists.isEmpty(items)}">
                    <!-- 全列結合でメッセージを表示 -->
                    <td colspan="7">商品がありません。</td>
                </tr>
            </tbody>
        </table>

        <!-- 画面下部の戻るボタン群 -->
        <div class="button-group">
            <!-- 管理者ダッシュボードへの戻りリンク -->
            <a th:href="@{/admin/dashboard}" class="button secondary">管理者ダッシュボードに戻
る</a>
        </div>
    </div>
</body>
</html>
```

admin_statistics.html（期間フィルタ＋総売上とステータス別件数の表示／CSV 出力）

```
<!-- HTML5 文書宣言 -->
<!DOCTYPE html>
<!-- ルート要素。日本語と Thymeleaf/Spring Security Dialect を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域 -->
<head>
    <!-- 文字コード指定 -->
    <meta charset="UTF-8">
    <!-- ページタイトル -->
    <title>管理者: 統計データ - フリマアプリ</title>
    <!-- 共通 CSS の読込 -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 本体 -->
<body>
    <!-- レイアウト用の中央コンテナ -->
    <div class="container">
        <!-- 見出し -->
        <h1>管理者: 統計データ</h1>
        <!-- ページ説明 -->
        <p>フリマアプリの売上や注文に関する統計データです。</p>

        <!-- 期間指定フォーム（GET で再表示） -->
        <form th:action="@{/admin/statistics}" method="get" class="filter-form">
            <!-- 開始日の入力欄 -->
            <div>
                <!-- 開始日のラベル -->
                <label for="startDate">開始日:</label>
                <!-- startDate を yyyy-MM-dd でフォーマットして初期値に設定 -->
                <input type="date" id="startDate" name="startDate"
th:value="${#temporals.format(startDate, 'yyyy-MM-dd')}">
            </div>
            <!-- 終了日の入力欄 -->
            <div>
                <!-- 終了日のラベル -->
                <label for="endDate">終了日:</label>
                <!-- endDate を yyyy-MM-dd でフォーマットして初期値に設定 -->
                <input type="date" id="endDate" name="endDate"
th:value="${#temporals.format(endDate, 'yyyy-MM-dd')}">
            </div>
            <!-- フィルタボタン -->
            <button type="submit" class="button">フィルタ</button>
        </form>
```

```
        <!-- CSV エクスポートリンク。現在の期間パラメータを付加 -->
        <a th:href="@{/admin/statistics/csv(startDate=${startDate}, endDate=${endDate})}"
class="button">CSV エクスポート</a>

        <!-- 総売上の見出し -->
        <h2>総売上</h2>
        <!-- 通貨表記で総売上を表示 -->
        <p class="price" th:text="'¥' + ${#numbers.formatDecimal(totalSales, 0, 'COMMA', 0,
'POINT')}"></p>

        <!-- ステータス別件数の見出し -->
        <h2>ステータス別注文数</h2>
        <!-- ステータス別件数の表 -->
        <table>
            <!-- 列見出し -->
            <thead>
                <tr>
                    <th>ステータス</th>
                    <th>注文数</th>
                </tr>
            </thead>
            <!-- 本体 -->
            <tbody>
                <!-- orderCountByStatus (Map) をエントリでループ -->
                <tr th:each="entry : ${orderCountByStatus}">
                    <!-- Map のキー（ステータス）を表示 -->
                    <td th:text="${entry.key}"></td>
                    <!-- Map の値（件数）を表示 -->
                    <td th:text="${entry.value}"></td>
                </tr>
                <!-- データが空の場合の表示 -->
                <tr th:if="${#maps.isEmpty(orderCountByStatus)}">
                    <!-- 全列結合でメッセージ表示 -->
                    <td colspan="2">データがありません。</td>
                </tr>
            </tbody>
        </table>

        <!-- 戻るボタン -->
        <div class="button-group">
            <!-- 管理者ダッシュボードへ戻る -->
            <a th:href="@{/admin/dashboard}" class="button secondary">管理者ダッシュボードに戻
る</a>
        </div>
    </div>
</body>
</html>
```

admin_users.html（全ユーザー一覧と有効/無効の切替操作）

```
<!-- HTML5 文書宣言 -->
<!DOCTYPE html>
<!-- ルート要素。日本語と Thymeleaf/Spring Security Dialect を有効化 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域 -->
<head>
    <!-- 文字コード設定 -->
    <meta charset="UTF-8">
    <!-- ページタイトル -->
    <title>管理者: ユーザー管理 - フリマアプリ</title>
    <!-- 共通スタイルの読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 本体 -->
<body>
    <!-- レイアウト用コンテナ -->
    <div class="container">
        <!-- 見出し -->
        <h1>管理者: ユーザー管理</h1>
        <!-- 説明テキスト -->
        <p>システム内の全てのユーザーを管理します。</p>

        <!-- ユーザー一覧テーブル -->
        <table>
            <!-- 列見出し -->
            <thead>
                <tr>
                    <th>ID</th>
                    <th>名前</th>
                    <th>メールアドレス</th>
                    <th>役割</th>
                    <th>有効/無効</th>
                    <th>操作</th>
                </tr>
            </thead>
            <!-- テーブル本体 -->
            <tbody>
                <!-- users をループし 1 行ずつ描画 -->
                <tr th:each="user : ${users}">
                    <!-- ユーザーID -->
                    <td th:text="${user.id}"></td>
                    <!-- 表示名 -->
                    <td th:text="${user.name}"></td>
                    <!-- メールアドレス -->
                    <td th:text="${user.email}"></td>
```

```
                    <!-- 権限 (USER/ADMIN) -->
                    <td th:text="${user.role}"></td>
                    <!-- 有効/無効の表示 (三項演算) -->
                    <td th:text="${user.enabled ? '有効' : '無効'}"></td>
                    <!-- 操作列 (有効/無効トグル) -->
                    <td>
                        <!-- トグル用 POST フォーム。ボタンの見た目とラベルを状態で切替 -->
                        <form th:action="@{/admin/users/{id}/toggle-enabled(id=${user.id})}"
method="post" style="display:inline;">
                            <!-- enabled の真偽でボタンのクラスと表示文言を切替し、確認ダイア
ログを表示 -->
                            <button type="submit" class="button"
th:classappend="${user.enabled ? 'danger' : 'secondary'}" th:text="${user.enabled ? '無効にす
る' : '有効にする'}" onclick="return confirm('本当にこのユーザーを' + (${user.enabled} ? '無効
' : '有効') + 'にしますか?');"></button>
                        </form>
                    </td>
                </tr>
                <!-- ユーザーが 1 件もいない場合の行 -->
                <tr th:if="${#lists.isEmpty(users)}">
                    <!-- 全列結合してメッセージ表示 -->
                    <td colspan="6">ユーザーがいません。</td>
                </tr>
            </tbody>
        </table>

        <!-- 戻るボタン群 -->
        <div class="button-group">
            <!-- 管理者ダッシュボードへ戻るリンク -->
            <a th:href="@{/admin/dashboard}" class="button secondary">管理者ダッシュボードに戻
る</a>
        </div>
    </div>
</body>
</html>
```

buyer_app_orders.html（購入履歴一覧を表示）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。言語=日本語、Thymeleaf と Spring Security Dialect の xmlns を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域（メタ・タイトル・スタイル） -->
<head>
    <!-- 文字コードを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- ブラウザタブに表示されるタイトル -->
    <title>購入履歴 - フリマアプリ</title>
    <!-- 共通スタイルシートを読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面本体 -->
<body>
    <!-- レイアウト用の中央コンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>購入履歴</h1>
        <!-- 画面の説明文 -->
        <p>あなたが購入した商品の一覧です。</p>

        <!-- 購入履歴テーブル -->
        <table>
            <!-- テーブルのヘッダ行 -->
            <thead>
                <tr>
                    <!-- 商品名の列見出し -->
                    <th>商品名</th>
                    <!-- 出品者名の列見出し -->
                    <th>出品者</th>
                    <!-- 価格の列見出し -->
                    <th>価格</th>
                    <!-- ステータスの列見出し -->
                    <th>ステータス</th>
                </tr>
            </thead>
            <!-- テーブル本体 -->
            <tbody>
                <!-- myOrders（購入履歴）を 1 件ずつ表示 -->
                <tr th:each="order : ${myOrders}">
                    <!-- 購入した商品の商品名を表示 -->
                    <td th:text="${order.item.name}"></td>
                    <!-- 出品者の表示名を表示 -->
                    <td th:text="${order.item.seller.name}"></td>
```

```
                    <!-- 価格を通貨風に整形表示（¥＋カンマ区切り） -->
                    <td th:text="'¥' + ${#numbers.formatDecimal(order.price, 0, 'COMMA', 0,
'POINT')}"></td>
                    <!-- 注文ステータス（購入済/発送済など）を表示 -->
                    <td th:text="${order.status}"></td>
                </tr>
                <!-- リストが空のときのメッセージ行 -->
                <tr th:if="${#lists.isEmpty(myOrders)}">
                    <!-- 4列を束ねて「購入なし」のメッセージを表示 -->
                    <td colspan="4">購入した商品はありません。</td>
                </tr>
            </tbody>
        </table>

        <!-- 画面下部のボタン群 -->
        <div class="button-group">
            <!-- マイページへ戻るリンク -->
            <a th:href="@{/my-page}" class="button secondary">マイページに戻る</a>
        </div>
    </div>
</body>
</html>
```

item_detail.html（商品詳細表示＋購入/お気に入り操作＋チャット表示・送信）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。言語=日本語、Thymeleaf と Spring Security Dialect の xmlns を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域（メタ・タイトル・スタイル） -->
<head>
    <!-- 文字コードを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- タイトルは商品名 + 固定文言を組み合わせて表示 -->
    <title th:text="${item.name} + ' - 商品詳細'"></title>
    <!-- 共通スタイルシートを読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面本体 -->
<body>
    <!-- レイアウト用の中央コンテナ -->
    <div class="container">
        <!-- 商品名を大見出しに表示 -->
        <h1 th:text="${item.name}"></h1>

        <!-- エラー時メッセージ（Flash 属性など）を表示 -->
        <div th:if="${errorMessage}" class="error-message" th:text="${errorMessage}"></div>
        <!-- 成功時メッセージ（Flash 属性など）を表示 -->
        <div th:if="${successMessage}" class="success-message"
th:text="${successMessage}"></div>

        <!-- 商品詳細領域 -->
        <div class="item-detail">
            <!-- 商品画像（なければプレースホルダ）を表示 -->
            <img th:src="${item.imageUrl != null ? item.imageUrl : '/images/placeholder.png'}"
alt="商品画像" class="item-image">
            <!-- 価格を通貨風に整形して表示 -->
            <p class="price">価格: <span th:text="'¥' + ${#numbers.formatDecimal(item.price,
0, 'COMMA', 0, 'POINT')}"></span></p>
            <!-- 商品説明を表示 -->
            <p>説明: <span th:text="${item.description}"></span></p>
            <!-- カテゴリ名（null なら未分類）を表示 -->
            <p>カテゴリ: <span th:text="${item.category != null ? item.category.name : '未分類
'}"></span></p>
            <!-- 出品者名と平均評価がある場合は併記 -->
            <p>出品者: <span th:text="${item.seller.name}"></span>
                <!-- 平均評価がモデルに存在する場合のみ表示 -->
                <span th:if="${sellerAverageRating}">(平均評価: <span
th:text="${sellerAverageRating}"></span>)</span>
            </p>
```

```
            <!-- 商品の現在ステータスを表示 -->
            <p>ステータス: <span th:text="${item.status}"></span></p>

            <!-- アクションボタン群 -->
            <div class="button-group">
                <!-- 自分以外が出品した「出品中」商品だけ購入ボタンを表示 -->
                <form th:action="@{/orders/initiate-purchase}" method="post"
th:if="${item.status == '出品中' and item.seller.email != #authentication.name}"
onsubmit="return confirm('本当にこの商品を購入しますか？');">
                    <!-- Stripe 連携に渡す itemId を hidden で送信 -->
                    <input type="hidden" name="itemId" th:value="${item.id}">
                    <!-- 購入開始ボタン -->
                    <button type="submit" class="button">購入する</button>
                </form>
                <!-- 対象商品のチャット画面への導線 -->
                <a th:href="@{/chat/{itemId}(itemId=${item.id})}" class="button secondary">チ
ャットで質問/交渉</a>

                <!-- ログイン済みで、かつ自分の出品ではない場合にお気に入り操作を表示 -->
                <div sec:authorize="isAuthenticated()" th:if="${item.seller.email !=
#authentication.name}">
                    <!-- 既にお気に入り済みのときは解除ボタンを表示 -->
                    <form th:if="${isFavorited}"
th:action="@{/items/{id}/unfavorite(id=${item.id})}" method="post" style="display:inline;">
                        <!-- お気に入り解除ボタン（赤系） -->
                        <button type="submit" class="button danger">お気に入り解除</button>
                    </form>
                    <!-- 未登録のときはお気に入り追加ボタンを表示 -->
                    <form th:unless="${isFavorited}"
th:action="@{/items/{id}/favorite(id=${item.id})}" method="post" style="display:inline;">
                        <!-- お気に入り追加ボタン -->
                        <button type="submit" class="button">お気に入りに追加</button>
                    </form>
                </div>
            </div>
        </div>

        <!-- チャット見出し -->
        <h2>チャット</h2>
        <!-- チャットの表示ボックス -->
        <div class="chat-box">
            <!-- チャット履歴を時系列で描画。送信者が自分なら my-message クラスを付与 -->
            <div th:each="chat : ${chats}" class="chat-message"
th:classappend="${chat.sender.email == #authentication.name ? 'my-message' : 'other-
message'}">
                <!-- 送信者名 -->
                <span class="sender-name" th:text="${chat.sender.name}"></span>
```

```
                <!-- 送信日時 (yyyy/MM/dd HH:mm 形式) -->
                <span class="message-time" th:text="${#temporals.format(chat.createdAt,
'yyyy/MM/dd HH:mm')}"></span>
                <!-- メッセージ本文 -->
                <p th:text="${chat.message}"></p>
            </div>
            <!-- チャットが 1 件もない場合のメッセージ -->
            <div th:if="${#lists.isEmpty(chats)}">
                <!-- 空の旨を通知 -->
                <p>まだチャットメッセージはありません。</p>
            </div>
        </div>

        <!-- チャット送信フォーム -->
        <form th:action="@{/chat/{itemId}(itemId=${item.id})}" method="post" class="chat-
form">
            <!-- 入力テキストエリア (必須) -->
            <textarea name="message" placeholder="メッセージを入力してください"
required></textarea>
            <!-- 送信ボタン -->
            <button type="submit">送信</button>
        </form>

        <!-- 下部の戻るボタン -->
        <div class="button-group">
            <!-- 商品一覧ページへの戻りリンク -->
            <a th:href="@{/items}" class="button secondary">商品一覧に戻る</a>
        </div>
    </div>
</body>
</html>
```

item_form.html（商品出品・編集フォーム。画像アップロード対応）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。言語=日本語、Thymeleaf の xmlns を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域（メタ・タイトル・スタイル） -->
<head>
    <!-- 文字コードを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- 新規出品か編集かでタイトルを切り替え -->
    <title th:text="${item.id == null ? '商品出品' : '商品編集'}"></title>
    <!-- 共通スタイルシートを読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面本体 -->
<body>
    <!-- レイアウト用の中央コンテナ -->
    <div class="container">
        <!-- 見出しも新規/編集で切り替え -->
        <h1 th:text="${item.id == null ? '商品出品' : '商品編集'}"></h1>

        <!-- 画像アップロード失敗などのエラーを表示 -->
        <div th:if="${errorMessage}" class="error-message" th:text="${errorMessage}"></div>

        <!-- 新規: /items POST、編集: /items/{id} POST。画像アップロードのため enctype 必須 -->
        <form th:action="${item.id == null ? '/items' : '/items/' + item.id}" method="post"
enctype="multipart/form-data">
            <!-- 編集時のみ HiddenHttpMethodFilter を想定し PUT メソッドを指定（サンプルのまま）
-->
            <input type="hidden" th:if="${item.id != null}" name="_method" value="put">
            <!-- 商品名の入力 -->
            <p>
                <!-- 商品名ラベル -->
                <label for="name">商品名:</label>
                <!-- 商品名テキストボックス（必須） -->
                <input type="text" id="name" name="name" th:value="${item.name}" required>
            </p>
            <!-- 商品説明の入力 -->
            <p>
                <!-- 商品説明ラベル -->
                <label for="description">商品説明:</label>
                <!-- 複数行テキストエリア。既存値を表示 -->
                <textarea id="description" name="description" rows="5"
th:text="${item.description}"></textarea>
            </p>
            <!-- 価格の入力 -->
            <p>
```

```
                <!-- 価格ラベル -->
                <label for="price">価格 (¥):</label>
                <!-- 数値入力（小数2桁ステップ） -->
                <input type="number" id="price" name="price" th:value="${item.price}"
step="0.01" required>
            </p>
            <!-- カテゴリの選択 -->
            <p>
                <!-- カテゴリラベル -->
                <label for="categoryId">カテゴリ:</label>
                <!-- 必須のセレクトボックス。初期値は選択してください -->
                <select id="categoryId" name="categoryId" required>
                    <!-- 空選択のプレースホルダ -->
                    <option value="">選択してください</option>
                    <!-- categories をループして option を生成。既存カテゴリと一致時は選択 -->
                    <option th:each="category : ${categories}" th:value="${category.id}"
th:text="${category.name}" th:selected="${item.category != null and item.category.id ==
category.id}"></option>
                </select>
            </p>
            <!-- 画像アップロード -->
            <p>
                <!-- 画像ラベル -->
                <label for="image">商品画像:</label>
                <!-- 画像ファイル選択（画像のみ許可） -->
                <input type="file" id="image" name="image" accept="image/*">
                <!-- 既存画像があればサムネイル表示 -->
                <span th:if="${item.imageUrl}">現在の画像: <img th:src="${item.imageUrl}"
style="width: 100px; height: auto;"></span>
            </p>
            <!-- 送信/キャンセルボタン -->
            <div class="button-group">
                <!-- 新規なら「出品する」、編集なら「更新する」 -->
                <button type="submit" th:text="${item.id == null ? '出品する' : '更新する
'}"></button>
                <!-- 一覧へ戻る -->
                <a th:href="@{/items}" class="button secondary">キャンセル</a>
            </div>
        </form>
    </div>
</body>
</html>
```

item_list.html（商品一覧＋検索/カテゴリ絞り込み＋ページネーション）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。言語=日本語、Thymeleaf の名前空間を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org">
<!-- ヘッダー領域：メタ情報・タイトル・スタイル読込 -->
<head>
    <!-- 文字コードを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- タブに表示されるページタイトル -->
    <title>商品一覧 - フリマアプリ</title>
    <!-- 共通スタイルシートを読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面本体 -->
<body>
    <!-- レイアウト中央寄せのコンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>商品一覧</h1>

        <!-- 主要ナビゲーション（出品・マイページ・ログアウト） -->
        <div class="main-nav">
            <!-- 新規出品画面へのリンク -->
            <a th:href="@{/items/new}" class="button">商品を出品する</a>
            <!-- マイページへのリンク -->
            <a th:href="@{/my-page}" class="button">マイページ</a>
            <!-- ログアウト送信フォーム（POST で送信） -->
            <form th:action="@{/logout}" method="post" style="display:inline;">
                <!-- ログアウトボタン（見た目だけリンク風） -->
                <button type="submit" class="button secondary">ログアウト</button>
            </form>
        </div>

        <!-- 検索・絞り込みフォーム（GET で自身のページへ） -->
        <form th:action="@{/items}" method="get" class="filter-form">
            <!-- キーワード入力ブロック -->
            <div>
                <!-- キーワード入力欄のラベル -->
                <label for="keyword">キーワード:</label>
                <!-- 直近の検索値を保持して入力欄に反映 -->
                <input type="text" id="keyword" name="keyword" th:value="${param.keyword}">
            </div>
            <!-- カテゴリ選択ブロック -->
            <div>
                <!-- カテゴリ選択のラベル -->
                <label for="categoryId">カテゴリ:</label>
```

```
                <!-- 全カテゴリを選択肢として表示（空=全て） -->
                <select id="categoryId" name="categoryId">
                    <!-- 全カテゴリを対象にするための空選択肢 -->
                    <option value="">全て</option>
                    <!-- categories をループして option 生成。URL パラメータと一致なら選択状態に
-->
                    <option th:each="category : ${categories}" th:value="${category.id}"
th:text="${category.name}" th:selected="${param.categoryId == category.id}"></option>
                </select>
            </div>
            <!-- 検索実行ボタン -->
            <button type="submit" class="button">検索</button>
        </form>

        <!-- 商品カード一覧（レスポンシブグリッド） -->
        <div class="item-grid">
            <!-- Page<Item>の content を回して 1 カードずつ描画 -->
            <div class="item-card" th:each="item : ${items.content}">
                <!-- 詳細画面へのリンク全体でカード化 -->
                <a th:href="@{/items/{id}(id=${item.id})}">
                    <!-- 画像 URL があれば表示、なければプレースホルダ -->
                    <img th:src="${item.imageUrl != null ? item.imageUrl :
'/images/placeholder.png'}" alt="商品画像">
                    <!-- 商品名を見出しとして表示 -->
                    <h3 th:text="${item.name}"></h3>
                    <!-- 価格を通貨風に整形（¥＋カンマ区切り） -->
                    <p class="price" th:text="'¥' + ${#numbers.formatDecimal(item.price, 0,
'COMMA', 0, 'POINT')}"></p>
                    <!-- 出品ステータス（出品中/売却済）を表示 -->
                    <p class="status" th:text="${item.status}"></p>
                </a>
            </div>
            <!-- 検索結果が 0 件のときにメッセージ表示 -->
            <div th:if="${#lists.isEmpty(items.content)}">
                <!-- 商品なしのメッセージ -->
                <p>商品が見つかりませんでした。</p>
            </div>
        </div>

        <!-- ページネーション（前へ/ページ番号/次へ） -->
        <div class="pagination">
            <!-- 前ページが存在する場合のみリンク表示 -->
            <a th:if="${items.hasPrevious()}" th:href="@{/items(page=${items.number - 1},
keyword=${param.keyword}, categoryId=${param.categoryId})}">前へ</a>
            <!-- 0〜(総ページ-1)のシーケンスを生成してページ番号リンクを作成 -->
            <span th:each="i : ${#numbers.sequence(0, items.totalPages - 1)}">
                <!-- 現在ページ以外はリンクとして表示 -->
                <a th:if="${i != items.number}" th:href="@{/items(page=${i},
keyword=${param.keyword}, categoryId=${param.categoryId})}" th:text="${i + 1}"></a>
```

```
                <!-- 現在ページは強調表示（リンクではない） -->
                <span th:if="${i == items.number}" class="current-page" th:text="${i +
1}"></span>
            </span>
            <!-- 次ページが存在する場合のみリンク表示 -->
            <a th:if="${items.hasNext()}" th:href="@{/items(page=${items.number + 1},
keyword=${param.keyword}, categoryId=${param.categoryId})}">次へ</a>
        </div>
    </div>
</body>
</html>
```

login.html（ログインフォームとエラーメッセージ表示）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。言語=日本語、Thymeleaf の名前空間を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org">
<!-- ヘッダー領域：メタ情報・タイトル・スタイル読込 -->
<head>
    <!-- 文字コードを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- ページタイトル -->
    <title>ログイン - フリマアプリ</title>
    <!-- 共通スタイルシートを読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面本体 -->
<body>
    <!-- レイアウト中央寄せのコンテナ -->
    <div class="container">
        <!-- アプリ名の見出し -->
        <h1>フリマアプリ</h1>
        <!-- 機能見出し：ログイン -->
        <h2>ログイン</h2>
        <!-- Spring Security の標準/login エンドポイントへ POST 送信 -->
        <form th:action="@{/login}" method="post">
            <!-- 認証エラー時のメッセージ表示領域（?error 付きで遷移時に表示） -->
            <div th:if="${param.error}" class="error-message">
                <!-- エラーテキスト本体 -->
                メールアドレスまたはパスワードが不正です。
            </div>
            <!-- ログアウト完了時のメッセージ表示領域（?logout で表示） -->
            <div th:if="${param.logout}" class="success-message">
                <!-- 完了テキスト本体 -->
                ログアウトしました。
            </div>
            <!-- ユーザー名（メールアドレス）入力行 -->
            <p>
                <!-- メールアドレス入力ラベル -->
                <label for="username">メールアドレス:</label>
                <!-- Spring Security デフォルトの username パラメータ名を使用 -->
                <input type="text" id="username" name="username" required>
            </p>
            <!-- パスワード入力行 -->
            <p>
                <!-- パスワード入力ラベル -->
                <label for="password">パスワード:</label>
                <!-- Spring Security デフォルトの password パラメータ名を使用 -->
                <input type="password" id="password" name="password" required>
            </p>
```

```
            <!-- ボタン群（送信のみ） -->
            <div class="button-group">
                <!-- 認証要求を送信 -->
                <button type="submit">ログイン</button>
            </div>
        </form>
    </div>
</body>
</html>
```

my_favorites.html（お気に入り一覧をカードで表示）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。言語=日本語、Thymeleaf と Spring Security Dialect の xmlns を宣言（未使用でも将来用に残置可） -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域：メタ情報・タイトル・スタイル読込 -->
<head>
    <!-- 文字コードを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- ページタイトル -->
    <title>お気に入り - フリマアプリ</title>
    <!-- 共通スタイルシートを読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面本体 -->
<body>
    <!-- レイアウト中央寄せのコンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>お気に入り</h1>
        <!-- 画面説明文：お気に入り登録済み商品の一覧 -->
        <p>あなたがお気に入り登録した商品の一覧です。</p>

        <!-- 商品カードのグリッド -->
        <div class="item-grid">
            <!-- favoriteItems を繰り返してカード表示 -->
            <div class="item-card" th:each="item : ${favoriteItems}">
                <!-- 各商品の詳細ページへのリンク -->
                <a th:href="@{/items/{id}(id=${item.id})}">
                    <!-- 商品画像（なければプレースホルダ画像） -->
                    <img th:src="${item.imageUrl != null ? item.imageUrl :
'/images/placeholder.png'}" alt="商品画像">
                    <!-- 商品名 -->
                    <h3 th:text="${item.name}"></h3>
                    <!-- 価格を通貨風に整形 -->
                    <p class="price" th:text="'¥' + ${#numbers.formatDecimal(item.price, 0,
'COMMA', 0, 'POINT')}"></p>
                    <!-- 出品ステータス -->
                    <p class="status" th:text="${item.status}"></p>
                </a>
            </div>
            <!-- お気に入りが空のときの表示 -->
            <div th:if="${#lists.isEmpty(favoriteItems)}">
                <!-- 空である旨の通知 -->
                <p>お気に入りの商品はありません。</p>
            </div>
```

```
        </div>

        <!-- 画面下部の戻る導線 -->
        <div class="button-group">
            <!-- マイページに戻るリンク -->
            <a th:href="@{/my-page}" class="button secondary">マイページに戻る</a>
        </div>
    </div>
</body>
</html>
```

my_page.html（マイページ：各種リンクとアカウント情報の表示）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素：日本語ページ／Thymeleaf と Spring Security Dialect を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域（メタ・タイトル・スタイル） -->
<head>
    <!-- 文字コードを UTF-8 に指定 -->
    <meta charset="UTF-8">
    <!-- ブラウザタブに表示するタイトル -->
    <title>マイページ - フリマアプリ</title>
    <!-- 共通スタイルシートの読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 本文開始 -->
<body>
    <!-- レイアウト用の中央コンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>マイページ</h1>
        <!-- ログイン中ユーザー名を差し込み表示 -->
        <p>ようこそ、<span th:text="${user.name}"></span>さん！</p>

        <!-- 機能ナビゲーション -->
        <div class="main-nav">
            <!-- 出品管理ページへのリンク -->
            <a th:href="@{/my-page/selling}" class="button">出品管理</a>
            <!-- 購入履歴ページへのリンク -->
            <a th:href="@{/my-page/orders}" class="button">購入履歴</a>
            <!-- 販売履歴ページへのリンク -->
            <a th:href="@{/my-page/sales}" class="button">販売履歴</a>
            <!-- お気に入りページへのリンク -->
            <a th:href="@{/my-page/favorites}" class="button">お気に入り</a>
            <!-- 自分が送信した評価一覧ページへのリンク -->
            <a th:href="@{/my-page/reviews}" class="button">自分の評価</a>
            <!-- 商品一覧へ戻る導線（サブカラー） -->
            <a th:href="@{/items}" class="button secondary">商品一覧に戻る</a>
            <!-- ログアウト POST フォーム（ボタンレイアウトのためインライン表示） -->
            <form th:action="@{/logout}" method="post" style="display:inline;">
                <!-- ログアウトボタン（強調のため danger クラス） -->
                <button type="submit" class="button danger">ログアウト</button>
            </form>
        </div>

        <!-- アカウント情報の見出し -->
        <h2>アカウント情報</h2>
```

```
        <!-- 名前ラベルと値を表示 -->
        <p><strong>名前:</strong> <span th:text="${user.name}"></span></p>
        <!-- メールラベルと値を表示 -->
        <p><strong>メールアドレス:</strong> <span th:text="${user.email}"></span></p>
        <!-- 役割ラベルと値を表示 (USER/ADMIN)  -->
        <p><strong>役割:</strong> <span th:text="${user.role}"></span></p>

    </div>
</body>
</html>
```

payment_confirmation.html（Stripe 決済確認：Elements の描画と支払い確定）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素：日本語ページ／Thymeleaf を宣言（セキュリティダイアレクトは不要） -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org">
<!-- ヘッダー領域（メタ・タイトル・スタイル・スクリプト） -->
<head>
    <!-- 文字コードを UTF-8 に指定 -->
    <meta charset="UTF-8">
    <!-- ブラウザタブに表示するタイトル -->
    <title>決済確認 - フリマアプリ</title>
    <!-- 共通スタイルシートの読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
    <!-- Stripe.js（公開キーで初期化し、Elements を使用） -->
    <script src="https://js.stripe.com/v3/"></script>
</head>
<!-- 本文開始 -->
<body>
    <!-- レイアウト用の中央コンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>決済確認</h1>
        <!-- 操作説明文：この画面で支払いを完了する旨を表示 -->
        <p>以下の内容で決済を完了します。</p>

        <!-- Stripe Elements の描画ターゲット要素 -->
        <div id="payment-element"></div>
        <!-- 決済確定ボタン -->
        <button id="submit" class="button">決済を完了する</button>
        <!-- エラーメッセージ表示領域 -->
        <div id="payment-message" class="error-message"></div>

        <!-- 戻る導線（商品詳細へ戻る） -->
        <div class="button-group">
            <!-- itemId パラメータで商品詳細へ戻す -->
            <a th:href="@{/items/{id}(id=${itemId})}" class="button secondary">商品詳細に戻る
</a>
        </div>
    </div>

    <!-- 画面内スクリプト（Thymeleaf 変数を JS リテラルに埋め込み） -->
    <script th:inline="javascript">
        // サーバから受け取った Stripe 公開キー
        const stripePublicKey = [[${stripePublicKey}]];
        // サーバサイドで作成された PaymentIntent のクライアントシークレット
        const clientSecret = [[${clientSecret}]];
        // 遷移元の商品 ID（戻り導線などに活用）
        const itemId = [[${itemId}]];
```

```
        // Stripe を公開キーで初期化
        const stripe = Stripe(stripePublicKey);
        // Elements をクライアントシークレット付きで生成
        const elements = stripe.elements({ clientSecret });
        // 決済 UI 要素（支払いフォーム）を作成
        const paymentElement = elements.create('payment');
        // 決済 UI 要素を DOM へマウント
        paymentElement.mount('#payment-element');

        // 送信ボタン要素を取得
        const form = document.getElementById('submit');
        // エラーメッセージ表示領域を取得
        const messageContainer = document.getElementById('payment-message');

        // クリックで支払い確定処理を開始（フォーム送信ではなく JS 制御）
        form.addEventListener('click', async (e) => {
            // ボタンのデフォルト動作（フォーム送信）を止める
            e.preventDefault();

            // Stripe へ支払い確定リクエストを送る（リダイレクト不要時は XHR 内で完了）
            const { error } = await stripe.confirmPayment({
                // 作成済み Elements を指定
                elements,
                // 決済成功後の戻り URL（必要に応じてサーバ側で検証/完了処理）
                confirmParams: {
                    // クライアントシークレットから PaymentIntent ID 部分を組み立てて返送（簡易
実装）
                    return_url: window.location.origin + '/orders/complete-
purchase?paymentIntentId=' + clientSecret.split('_')[0] + '_' + clientSecret.split('_')[1],
                },
                // 3DS などで外部遷移が不要な場合はそのまま JS で継続
                redirect: 'if_required',
            });

            // エラーが返ってきた場合の分岐
            if (error) {
                // カードエラー or 入力検証エラーの場合は詳細を表示
                if (error.type === "card_error" || error.type === "validation_error") {
                    // Stripe が示す具体的なエラーメッセージを表示
                    messageContainer.textContent = error.message;
                } else {
                    // 上記以外の不測のエラー
                    messageContainer.textContent = "予期せぬエラーが発生しました。";
                }
            } else {
                // エラーなし＝即時成功（サーバ検証のため PaymentIntent ID で完了 API へ遷移）
                const paymentIntentId = clientSecret.split('_')[0] + '_' +
clientSecret.split('_')[1];
```

```
                // 完了エンドポイントへGET遷移（サーバで購入完了処理を実行）
                window.location.href = '/orders/complete-purchase?paymentIntentId=' +
paymentIntentId;
            }
        });
    </script>
</body>
</html>
```

review_form.html（購入後の評価投稿フォーム）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素：日本語ページ／Thymeleaf と Spring Security Dialect を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域（メタ・タイトル・スタイル） -->
<head>
    <!-- 文字コードを UTF-8 に指定 -->
    <meta charset="UTF-8">
    <!-- ブラウザタブに表示するタイトル -->
    <title>商品評価 - フリマアプリ</title>
    <!-- 共通スタイルシートの読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 本文開始 -->
<body>
    <!-- レイアウト用の中央コンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>商品評価</h1>
        <!-- 対象商品名と出品者名を差し込み表示 -->
        <p>商品「<span th:text="${order.item.name}"></span>」の出品者「<span
th:text="${order.item.seller.name}"></span>」を評価してください。</p>

        <!-- エラーメッセージ（Flash 属性など）を条件表示 -->
        <div th:if="${errorMessage}" class="error-message" th:text="${errorMessage}"></div>

        <!-- 評価送信フォーム（POST /reviews） -->
        <form th:action="@{/reviews}" method="post">
            <!-- 対象注文 ID を hidden で送信（サーバ側で正当性チェック） -->
            <input type="hidden" name="orderId" th:value="${order.id}">
            <!-- 評価点の入力行 -->
            <p>
                <!-- 評価のラベル（1〜5） -->
                <label for="rating">評価（1-5）:</label>
                <!-- 1〜5 の整数のみ許可し必須化 -->
                <input type="number" id="rating" name="rating" min="1" max="5" required>
            </p>
            <!-- コメント入力行 -->
            <p>
                <!-- コメントのラベル（任意項目） -->
                <label for="comment">コメント（任意）:</label>
                <!-- 任意のテキスト（複数行） -->
                <textarea id="comment" name="comment" rows="5"></textarea>
            </p>
            <!-- ボタン群（送信／キャンセル） -->
            <div class="button-group">
```

```
                <!-- 送信ボタン：評価を登録 -->
                <button type="submit">評価を送信</button>
                <!-- キャンセル：購入履歴へ戻る -->
                <a th:href="@{/my-page/orders}" class="button secondary">キャンセル</a>
            </div>
        </form>
    </div>
</body>
</html>
```

seller_app_orders.html（販売履歴一覧と発送操作ボタン）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。言語は日本語、Thymeleaf と Spring Security Dialect を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域：メタ情報・タイトル・スタイル読込 -->
<head>
    <!-- 文字コードを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- ブラウザタブに表示されるページタイトル -->
    <title>販売履歴 - フリマアプリ</title>
    <!-- 共通スタイルシートの読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面本体開始 -->
<body>
    <!-- レイアウト中央寄せのコンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>販売履歴</h1>
        <!-- ページ説明文 -->
        <p>あなたが販売した商品の一覧です。</p>

        <!-- 販売履歴テーブル -->
        <table>
            <!-- 表ヘッダ行の定義 -->
            <thead>
                <!-- 列見出し行 -->
                <tr>
                    <!-- 商品名列 -->
                    <th>商品名</th>
                    <!-- 購入者列 -->
                    <th>購入者</th>
                    <!-- 価格列 -->
                    <th>価格</th>
                    <!-- ステータス列 -->
                    <th>ステータス</th>
                    <!-- 操作列（発送ボタンなど） -->
                    <th>操作</th>
                </tr>
            </thead>
            <!-- 表本体 -->
            <tbody>
                <!-- mySales コレクションをループして各注文行を表示 -->
                <tr th:each="order : ${mySales}">
                    <!-- 注文に紐づく商品名を表示 -->
                    <td th:text="${order.item.name}"></td>
```

```
                    <!-- 購入者名を表示 -->
                    <td th:text="${order.buyer.name}"></td>
                    <!-- 価格を通貨風（¥＋カンマ区切り）で表示 -->
                    <td th:text="'¥' + ${#numbers.formatDecimal(order.price, 0, 'COMMA', 0,
'POINT')}"></td>
                    <!-- 注文ステータスを表示（購入済/発送済 など） -->
                    <td th:text="${order.status}"></td>
                    <!-- 操作セル：発送済みにするボタンを条件表示 -->
                    <td>
                        <!-- ステータスが購入済のときのみ発送ボタンを表示 -->
                        <form th:action="@{/orders/{id}/ship(id=${order.id})}" method="post"
th:if="${order.status == '購入済'}">
                            <!-- 送信ボタン（発送済みに更新） -->
                            <button type="submit" class="button">発送済みにする</button>
                        </form>
                    </td>
                </tr>
                <!-- データが空の場合の表示（0件メッセージ） -->
                <tr th:if="${#lists.isEmpty(mySales)}">
                    <!-- 全列結合でメッセージを表示 -->
                    <td colspan="5">販売した商品はありません。</td>
                </tr>
            </tbody>
        </table>

        <!-- 画面下部の戻る導線 -->
        <div class="button-group">
            <!-- マイページへ戻るリンク（セカンダリボタン） -->
            <a th:href="@{/my-page}" class="button secondary">マイページに戻る</a>
        </div>
    </div>
</body>
</html>
```

seller_items.html（出品管理：一覧・編集リンク・削除ボタン）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。言語=日本語、Thymeleaf と Spring Security Dialect を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域：メタ情報・タイトル・スタイル読込 -->
<head>
    <!-- 文字コードを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- ブラウザタブに表示されるページタイトル -->
    <title>出品管理 - フリマアプリ</title>
    <!-- 共通スタイルシートの読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面本体開始 -->
<body>
    <!-- レイアウト中央寄せのコンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>出品管理</h1>
        <!-- ページ説明文 -->
        <p>あなたが出品した商品の一覧です。</p>

        <!-- 主要ナビ（新規出品と戻る） -->
        <div class="main-nav">
            <!-- 新規出品ページへのリンク -->
            <a th:href="@{/items/new}" class="button">新しい商品を出品する</a>
            <!-- マイページに戻るリンク（セカンダリ） -->
            <a th:href="@{/my-page}" class="button secondary">マイページに戻る</a>
        </div>

        <!-- 出品中アイテムのカードグリッド -->
        <div class="item-grid">
            <!-- sellingItems をループして各アイテムカードを表示 -->
            <div class="item-card" th:each="item : ${sellingItems}">
                <!-- 商品詳細へのリンクでカード全体を囲む -->
                <a th:href="@{/items/{id}(id=${item.id})}">
                    <!-- 商品画像（なければプレースホルダ） -->
                    <img th:src="${item.imageUrl != null ? item.imageUrl :
'/images/placeholder.png'}" alt="商品画像">
                    <!-- 商品名 -->
                    <h3 th:text="${item.name}"></h3>
                    <!-- 価格を通貨風（¥＋カンマ区切り）で表示 -->
                    <p class="price" th:text="'¥' + ${#numbers.formatDecimal(item.price, 0,
'COMMA', 0, 'POINT')}"></p>
                    <!-- 出品ステータス（出品中/売却済） -->
                    <p class="status" th:text="${item.status}"></p>
```

```
                </a>
                <!-- 各カード下の操作ボタン群 -->
                <div class="button-group">
                    <!-- 編集画面へのリンク（セカンダリ） -->
                    <a th:href="@{/items/{id}/edit(id=${item.id})}" class="button secondary">
編集</a>
                    <!-- 削除フォーム（POST）。インライン表示で横並び -->
                    <form th:action="@{/items/{id}/delete(id=${item.id})}" method="post"
style="display:inline;">
                        <!-- 確認ダイアログ付きの削除ボタン（危険色） -->
                        <button type="submit" class="button danger" onclick="return confirm('
本当にこの商品を削除しますか？');">削除</button>
                    </form>
                </div>
            </div>
            <!-- 出品が0件のときの表示 -->
            <div th:if="${#lists.isEmpty(sellingItems)}">
                <!-- 0件メッセージ -->
                <p>出品中の商品はありません。</p>
            </div>
        </div>
    </div>
</body>
</html>
```

user_reviews.html（自分の評価一覧：★表示と日時整形）

```
<!-- HTML5 文書であることを宣言 -->
<!DOCTYPE html>
<!-- ルート要素。言語=日本語、Thymeleaf と Spring Security Dialect を宣言 -->
<html lang="ja" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<!-- ヘッダー領域：メタ情報・タイトル・スタイル読込 -->
<head>
    <!-- 文字コードを UTF-8 に設定 -->
    <meta charset="UTF-8">
    <!-- ブラウザタブに表示されるページタイトル -->
    <title>自分の評価 - フリマアプリ</title>
    <!-- 共通スタイルシートの読み込み -->
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<!-- 画面本体開始 -->
<body>
    <!-- レイアウト中央寄せのコンテナ -->
    <div class="container">
        <!-- ページ見出し -->
        <h1>自分の評価</h1>
        <!-- ページ説明文 -->
        <p>あなたが送信した評価の一覧です。</p>

        <!-- 評価一覧テーブル -->
        <table>
            <!-- 表ヘッダ行の定義 -->
            <thead>
                <!-- 列見出し行 -->
                <tr>
                    <!-- 商品名列 -->
                    <th>商品名</th>
                    <!-- 出品者列 -->
                    <th>出品者</th>
                    <!-- 評価（★表示）列 -->
                    <th>評価</th>
                    <!-- コメント列 -->
                    <th>コメント</th>
                    <!-- 評価日時列 -->
                    <th>評価日時</th>
                </tr>
            </thead>
            <!-- 表本体 -->
            <tbody>
                <!-- reviews コレクションをループして各レビューを表示 -->
                <tr th:each="review : ${reviews}">
                    <!-- 対象商品名 -->
                    <td th:text="${review.item.name}"></td>
```

```
                    <!-- 対象出品者名 -->
                    <td th:text="${review.seller.name}"></td>
                    <!-- 星評価（1〜5 のシーケンスで★/☆を描画） -->
                    <td>
                        <!-- 1〜5 までの数列を回して評価値以下を★、それ以外を☆で表示 -->
                        <span th:each="i : ${#numbers.sequence(1, 5)}">
                            <!-- i が評価以下なら★ -->
                            <span th:if="${i <= review.rating}">★</span>
                            <!-- i が評価より大きければ☆ -->
                            <span th:if="${i > review.rating}">☆</span>
                        </span>
                    </td>
                    <!-- レビューコメント本文 -->
                    <td th:text="${review.comment}"></td>
                    <!-- 作成日時を YYYY/MM/DD HH:mm 形式で整形表示 -->
                    <td th:text="${#temporals.format(review.createdAt, 'yyyy/MM/dd
HH:mm')}"></td>
                </tr>
                <!-- レビューが 0 件のときの表示 -->
                <tr th:if="${#lists.isEmpty(reviews)}">
                    <!-- 0 件メッセージ -->
                    <td colspan="5">まだ評価を送信していません。</td>
                </tr>
            </tbody>
        </table>

        <!-- 画面下部の戻る導線 -->
        <div class="button-group">
            <!-- マイページへ戻るリンク（セカンダリ） -->
            <a th:href="@{/my-page}" class="button secondary">マイページに戻る</a>
        </div>
    </div>
</body>
</html>
```

10.7 スタイル（resources/static/css）

style.css（共通：フォーム/テーブル/カード/チャット/ページネーション）

```
/* 既定のフォントや背景など全体の基調設定をまとめる */
body {
  /* 可読性の高いサンセリフ体を優先して指定 */
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  /* 画面端にくっつかないよう外周余白 */
  margin: 0;
  /* 内側の基本余白を確保 */
  padding: 20px;
  /* 薄いグレー青系の背景でカードを浮かせる */
  background-color: #eef2f7;
  /* 本文色は濃いグレーで視認性を確保 */
  color: #333;
  /* 行間を広めにして読みやすくする */
  line-height: 1.6;
}

/* 中央の白いカード状のコンテナを作る */
.container {
  /* 横幅を最大 960px に制限して可読性を上げる */
  max-width: 960px;
  /* 余白で上下左右中央寄せ（左右 auto） */
  margin: 30px auto;
  /* カードのベース色 */
  background-color: #ffffff;
  /* 内側パディングで詰まり防止 */
  padding: 30px;
  /* 角丸でやわらかい印象に */
  border-radius: 10px;
  /* うっすらしたドロップシャドウで浮かせる */
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
}

/* ページ大見出しと中見出しの共通装飾 */
h1,
h2 {
  /* 見出し色はやや濃いめの青系 */
  color: #2c3e50;
  /* 中央寄せで視線誘導 */
  text-align: center;
  /* 見出し下に余白を確保 */
  margin-bottom: 30px;
  /* h1 のサイズを少し大きめに */
  font-size: 2.2em;
}
```

```
/* h2 のみ個別微調整 */
h2 {
  /* h2 の標準サイズ */
  font-size: 1.8em;
  /* h2 前にスペースを追加 */
  margin-top: 40px;
  /* 下線風の罫線でセクションを区切る */
  border-bottom: 2px solid #e0e0e0;
  /* 罫線と本文がくっつかないよう下に余白 */
  padding-bottom: 10px;
}

/* 段落の下に適度な余白を付与 */
p {
  /* 要素間隔を取り読みやすく */
  margin-bottom: 15px;
}

/* ラベル要素の共通スタイル */
label {
  /* ブロック表示にして上下に並べやすく */
  display: block;
  /* 下に少し余白を設ける */
  margin-bottom: 8px;
  /* 見出しより弱い強調 */
  font-weight: bold;
  /* 少し薄い文字色で主張を抑える */
  color: #555;
}
```

```
/* 入力系コンポーネントの共通スタイル */
input[type="text"],
input[type="password"],
input[type="email"],
input[type="number"],
textarea,
select {
  /* 親幅に対してパディング等を考慮して計算幅 */
  width: calc(100% - 24px);
  /* 内側パディングで入力しやすく */
  padding: 12px;
  /* ベースの枠線は薄いグレー */
  border: 1px solid #ccc;
  /* 角丸でやわらかい印象 */
  border-radius: 6px;
  /* 本文と同程度の文字サイズ */
  font-size: 1em;
  /* パディング/ボーダー込みで幅を計算 */
  box-sizing: border-box;
  /* フォーカス時のアニメーションをなめらかに */
  transition: border-color 0.3s ease;
}

/* フォーカス時の視覚的フィードバック */
input[type="text"]:focus,
input[type="password"]:focus,
input[type="email"]:focus,
input[type="number"]:focus,
textarea:focus,
select:focus {
  /* フォーカスで青系の枠線に */
  border-color: #007bff;
  /* ブラウザ既定のアウトラインは非表示 */
  outline: none;
}

/* テキストエリアのサイズ指定 */
textarea {
  /* 垂直方向のみリサイズ可能に */
  resize: vertical;
  /* 最小高さを確保して書きやすく */
  min-height: 100px;
}
```

```
/* ボタンを中央にまとめる枠 */
.button-group {
  /* 中央寄せ整列 */
  text-align: center;
  /* 上側に余白を設ける */
  margin-top: 30px;
}

/* ボタンとボタン風リンクの共通装飾 */
button,
.button {
  /* ベース色はブランド青 */
  background-color: #007bff;
  /* 文字色は白でコントラスト確保 */
  color: white;
  /* クリック領域を大きく */
  padding: 12px 25px;
  /* デフォルトの枠線は消す */
  border: none;
  /* 角丸でやわらかい見た目 */
  border-radius: 6px;
  /* クリック可能にするカーソル */
  cursor: pointer;
  /* 見やすいサイズに */
  font-size: 1.1em;
  /* ホバー時の色や押下風の動きにトランジション */
  transition: background-color 0.3s ease, transform 0.2s ease;
  /* インラインブロックで横並びに対応 */
  display: inline-block;
  /* ボタン間の間隔 */
  margin: 0 10px;
}

/* ホバー時の視覚効果 */
button:hover,
.button:hover {
  /* ほんの少し濃くしてアクティブ感 */
  background-color: #0056b3;
  /* わずかに浮かせる演出 */
  transform: translateY(-2px);
}

/* サブ系のボタン色 */
.button.secondary {
  /* グレーでニュートラルな操作を表現 */
  background-color: #6c757d;
}
```

```
/* セカンダリのホバー色 */
.button.secondary:hover {
  /* 少し濃いグレーに */
  background-color: #5a6268;
}

/* 危険（削除など）な操作の色 */
.button.danger {
  /* 赤で警告感を出す */
  background-color: #dc3545;
}

/* デンジャーのホバー色 */
.button.danger:hover {
  /* さらに濃い赤に */
  background-color: #c82333;
}

/* エラーメッセージの見た目 */
.error-message {
  /* 赤系の文字色で注意喚起 */
  color: #dc3545;
  /* 少し小さめの文字で補助情報感 */
  font-size: 0.9em;
  /* 上側に余白 */
  margin-top: 5px;
  /* ブロック要素にして行送り確保 */
  display: block;
  /* 中央寄せで目に入るように */
  text-align: center;
}

/* 成功メッセージの見た目 */
.success-message {
  /* 成功色のグリーンを文字色に */
  color: #28a745;
  /* 少し大きめで嬉しさを演出 */
  font-size: 1.1em;
  /* 中央寄せで見やすく */
  text-align: center;
  /* 上側に余白 */
  margin-top: 20px;
  /* 薄い緑の背景でカード風に */
  padding: 15px;
  /* 境界線でメッセージを区切る */
  border: 1px solid #c3e6cb;
  /* 角丸で柔らかく */
  border-radius: 5px;
}
```

```css
/* ===== テーブル共通装飾 ===== */
/* 表の土台スタイル */
table {
  /* 幅いっぱいに展開 */
  width: 100%;
  /* 罫線の二重線を解消 */
  border-collapse: collapse;
  /* 上側に余白 */
  margin-top: 20px;
  /* 背景色を白でカード風に */
  background-color: #fff;
  /* うっすらした影で浮かせる */
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.05);
  /* 角丸で滑らかに */
  border-radius: 8px;
  /* はみ出す角丸を有効にするため hidden */
  overflow: hidden;
}

/* セルの共通装飾 */
th,
td {
  /* 薄い罫線で区切る */
  border: 1px solid #e0e0e0;
  /* セル内の余白を確保 */
  padding: 12px 15px;
  /* 左寄せで読みやすく */
  text-align: left;
}

/* ヘッダセルの背景と強調 */
th {
  /* うっすらグレーで見出しを目立たせる */
  background-color: #f2f2f2;
  /* 太字で強調 */
  font-weight: bold;
  /* 少し濃いめの文字色 */
  color: #444;
}

/* 偶数行の背景を変えて可読性アップ */
tr:nth-child(even) {
  /* 行ストライプ */
  background-color: #f9f9f9;
}
```

```
/* 行ホバー時の背景で行を追いやすく */
tr:hover {
  /* うっすら強調 */
  background-color: #f1f1f1;
}

/* ===== ナビゲーション ===== */
/* メインナビの枠 */
.main-nav {
  /* 中央寄せ */
  text-align: center;
  /* 下に余白で本文と分離 */
  margin-bottom: 30px;
}

/* ナビ内のリンク装飾 */
.main-nav a {
  /* 濃い青グレーでボタン風 */
  background-color: #34495e;
  /* 文字は白 */
  color: white;
  /* クリック領域を確保 */
  padding: 10px 20px;
  /* 角丸で柔らかく */
  border-radius: 5px;
  /* アンダーライン除去 */
  text-decoration: none;
  /* 隣接リンクと間隔 */
  margin: 0 5px;
  /* ホバー時の変化をスムーズに */
  transition: background-color 0.3s ease;
}
```

```
/* ===== 商品カードグリッド ===== */
/* グリッドレイアウトの土台 */
.item-grid {
  /* 自動で詰まるレスポンシブグリッド（最小200px） */
  display: grid;
  /* 可変列でカードを並べる */
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  /* カード間のスペース */
  gap: 20px;
  /* 上に少し余白 */
  margin-top: 20px;
}

/* 個々のカードの見た目 */
.item-card {
  /* 白基調のカード */
  background-color: #fff;
  /* 角丸 */
  border-radius: 8px;
  /* うっすら影 */
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.05);
  /* はみ出す画像などを丸みに合わせる */
  overflow: hidden;
  /* 中央寄せ */
  text-align: center;
  /* ホバー時のアニメを滑らかに */
  transition: transform 0.2s ease;
}

/* カードのホバー効果 */
.item-card:hover {
  /* 少しだけ浮かせる */
  transform: translateY(-5px);
}

/* カード全体をリンククリック可能に */
.item-card a {
  /* アンダーラインなしで見出しを保つ */
  text-decoration: none;
  /* 文字色を継承して自然に */
  color: inherit;
  /* クリック領域を広くするためブロック化 */
  display: block;
  /* 内側余白 */
  padding: 15px;
}
```

```
/* カード内の画像調整 */
.item-card img {
  /* 横幅いっぱいに広げる */
  width: 100%;
  /* 高さは固定で整列（一部トリミング） */
  height: 150px;
  /* 画像のトリミング方法を cover に */
  object-fit: cover;
  /* 画像下に細い罫線風の境界 */
  border-bottom: 1px solid #eee;
  /* 下に少し余白 */
  margin-bottom: 10px;
}

/* カード内の商品名 */
.item-card h3 {
  /* 見出しのサイズ */
  font-size: 1.1em;
  /* 下余白 */
  margin-bottom: 5px;
  /* 文字色を少し濃く */
  color: #2c3e50;
}

/* 価格表示を目立たせる */
.item-card .price {
  /* 少し大きめ＋太字で強調 */
  font-size: 1.2em;
  font-weight: bold;
  /* ブランド青で視線誘導 */
  color: #007bff;
  /* 下余白 */
  margin-bottom: 10px;
}

/* ステータスの見た目 */
.item-card .status {
  /* 小さめの補助情報感 */
  font-size: 0.9em;
  /* グレーで抑えめに */
  color: #6c757d;
}
```

```css
/* ===== ページネーション ===== */
/* ページネーションの枠 */
.pagination {
  /* 中央寄せ */
  text-align: center;
  /* 上の要素と間隔 */
  margin-top: 30px;
}

/* ページリンクと現在ページの共通装飾 */
.pagination a,
.pagination span.current-page {
  /* 横並びでクリック領域確保 */
  display: inline-block;
  /* 内側余白で押しやすく */
  padding: 8px 15px;
  /* リンク間のスペース */
  margin: 0 5px;
  /* 枠線はブランド青 */
  border: 1px solid #007bff;
  /* 角丸で柔らかく */
  border-radius: 5px;
  /* 下線を消す */
  text-decoration: none;
  /* 文字色はブランド青 */
  color: #007bff;
  /* ホバー/状態変化のアニメ */
  transition: background-color 0.3s ease, color 0.3s ease;
}

/* ページリンクのホバー時 */
.pagination a:hover {
  /* 背景を青、文字を白へ反転 */
  background-color: #007bff;
  color: white;
}

/* 現在ページの見た目 */
.pagination span.current-page {
  /* 常に反転状態で現在位置を示す */
  background-color: #007bff;
  color: white;
  /* 現在位置を太字で強調 */
  font-weight: bold;
}
```

```
/* ===== 商品詳細 ===== */
/* 商品詳細のレイアウト */
.item-detail {
  /* 縦方向に積む */
  display: flex;
  flex-direction: column;
  /* 中央寄せ */
  align-items: center;
  /* 下側に余白 */
  margin-bottom: 30px;
}

/* 商品画像の見た目 */
.item-detail .item-image {
  /* 最大幅は親に合わせる */
  max-width: 100%;
  /* 高さは自動比率 */
  height: auto;
  /* 角丸 */
  border-radius: 8px;
  /* うっすら影で浮かせる */
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  /* 下に余白 */
  margin-bottom: 20px;
}

/* 商品詳細の段落調整 */
.item-detail p {
  /* 少し大きめで読みやすく */
  font-size: 1.1em;
  /* 下に余白 */
  margin-bottom: 10px;
}

/* 商品詳細の価格強調 */
.item-detail .price {
  /* さらに大きく太字で */
  font-size: 1.5em;
  font-weight: bold;
  /* ブランド青で目立たせる */
  color: #007bff;
}
```

```
/* ===== チャット表示 ===== */
/* チャット全体の枠 */
.chat-box {
  /* 薄い枠線で囲む */
  border: 1px solid #e0e0e0;
  /* 角丸でやさしく */
  border-radius: 8px;
  /* 内側余白 */
  padding: 15px;
  /* 縦方向に長くなりすぎたらスクロール */
  max-height: 300px;
  overflow-y: auto;
  /* 下に余白を確保 */
  margin-bottom: 20px;
  /* 薄いグレー背景で本文と区別 */
  background-color: #f9f9f9;
}

/* 個々のチャットバブル */
.chat-message {
  /* バブル間の余白 */
  margin-bottom: 10px;
  /* 内側余白で読みやすく */
  padding: 8px 12px;
  /* 丸みを付けてバブル風に */
  border-radius: 15px;
  /* 横幅は最大80%に抑える */
  max-width: 80%;
  /* 長い英単語などを折り返す */
  word-wrap: break-word;
}

/* 自分のメッセージの見た目（右寄せ） */
.my-message {
  /* 淡い緑で送信者を識別 */
  background-color: #dcf8c6;
  /* 左側に自動マージンで右寄せ */
  margin-left: auto;
  /* テキストは右揃え */
  text-align: right;
}
```

```
/* 相手のメッセージの見た目（左寄せ） */
.other-message {
  /* 薄いグレーで相手を識別 */
  background-color: #e0e0e0;
  /* 右側にマージンで左寄せ */
  margin-right: auto;
  /* テキストは左揃え */
  text-align: left;
}

/* 送信者名の装飾 */
.sender-name {
  /* 太字で誰の発言か明確に */
  font-weight: bold;
  /* 小さめのサイズで控えめに */
  font-size: 0.9em;
  /* 少し薄いグレー */
  color: #555;
}

/* 送信時刻の装飾 */
.message-time {
  /* 小さめのサイズ */
  font-size: 0.8em;
  /* 薄いグレーで控えめに */
  color: #888;
  /* 送信者名との間隔 */
  margin-left: 10px;
}

/* チャット本文の余白調整 */
.chat-message p {
  /* 上余白を少しだけ確保（デフォルトを打ち消す） */
  margin: 5px 0 0 0;
}

/* チャット送信フォームの横並び */
.chat-form {
  /* 横並びレイアウト */
  display: flex;
  /* 入力とボタンの間隔 */
  gap: 10px;
}
```

```
/* チャットのテキストエリア調整 */
.chat-form textarea {
  /* 余った領域に広がる */
  flex-grow: 1;
  /* 最小高さを指定 */
  min-height: 40px;
  /* 内側余白 */
  padding: 10px;
  /* 角丸 */
  border-radius: 6px;
  /* 枠線は薄いグレー */
  border: 1px solid #ccc;
}

/* 送信ボタンの横幅を固定 */
.chat-form button {
  /* 伸縮しない */
  flex-shrink: 0;
  /* 内側余白を確保 */
  padding: 10px 20px;
}

/* ===== フィルタフォーム ===== */
/* 絞り込みフォームのレイアウト */
.filter-form {
  /* 横並び＋折返し */
  display: flex;
  /* 各要素間の間隔 */
  gap: 15px;
  /* 下に余白 */
  margin-bottom: 20px;
  /* 下端基準で整列 */
  align-items: flex-end;
  /* 狭い画面で折り返す */
  flex-wrap: wrap;
  /* 薄いグレー背景でフォームを強調 */
  background-color: #f8f9fa;
  /* 内側余白 */
  padding: 20px;
  /* 角丸 */
  border-radius: 8px;
  /* うっすら影で浮かせる */
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.05);
}
```

```
/* 入力ブロックを均等配置 */
.filter-form div {
  /* 幅を均等に伸ばす */
  flex: 1;
  /* 最小幅を設定して崩れ防止 */
  min-width: 180px;
}

/* フィルタフォームのラベル微調整 */
.filter-form label {
  /* 少し小さめで控えめ */
  font-size: 0.9em;
  /* 薄いグレーで目立ちすぎないように */
  color: #666;
}

/* フィルタフォームの入力要素共通調整 */
.filter-form input[type="text"],
.filter-form select {
  /* 幅いっぱいに広げる */
  width: 100%;
  /* クリック領域の確保 */
  padding: 8px;
  /* 角丸 */
  border-radius: 4px;
  /* 薄い枠線 */
  border: 1px solid #ddd;
}

/* 絞り込みボタンの見た目 */
.filter-form button {
  /* 押しやすいサイズに */
  padding: 10px 20px;
  /* 一般的なボタンサイズ */
  font-size: 1em;
}
```

10.8 スキーマ/データ（resources）

schema.sql（DDL：テーブル作成・制約・インデックス。Spring 起動時自動実行）

```
-- 依存順に DROP してクリーンスタート（開発用途）
DROP TABLE IF EXISTS review CASCADE;            -- 先に参照の多いテーブルを落とす
DROP TABLE IF EXISTS favorite_item CASCADE;     -- お気に入りテーブルを DROP
DROP TABLE IF EXISTS chat CASCADE;              -- チャットテーブルを DROP
DROP TABLE IF EXISTS app_order CASCADE;         -- 注文テーブルを DROP
DROP TABLE IF EXISTS item CASCADE;              -- 商品テーブルを DROP
DROP TABLE IF EXISTS category CASCADE;          -- カテゴリテーブルを DROP
DROP TABLE IF EXISTS users CASCADE;             -- ユーザテーブルを DROP

-- ===== users（ユーザー）テーブル作成 =====
CREATE TABLE users (
  id SERIAL PRIMARY KEY,                        -- 主キー（連番）
  name VARCHAR(50) NOT NULL,                    -- 表示名（必須）
  email VARCHAR(255) NOT NULL UNIQUE,           -- ログイン用メール（必須・一意）
  password VARCHAR(255) NOT NULL,               -- パスワード（開発時は平文運用を想定）
  role VARCHAR(20) NOT NULL,                    -- 権限（USER / ADMIN）
  line_notify_token VARCHAR(255),               -- LINE Notify アクセストークン
  enabled BOOLEAN NOT NULL DEFAULT TRUE         -- アカウント有効/無効フラグ（既定は有効）
);

-- ===== category（カテゴリ）テーブル作成 =====
CREATE TABLE category (
  id SERIAL PRIMARY KEY,                        -- 主キー（連番）
  name VARCHAR(50) NOT NULL UNIQUE              -- カテゴリ名（必須・一意）
);

-- ===== item（商品）テーブル作成 =====
CREATE TABLE item (
  id SERIAL PRIMARY KEY,                        -- 主キー（連番）
  user_id INT NOT NULL,                         -- 出品者 ID（FK → users.id）
  name VARCHAR(255) NOT NULL,                   -- 商品名（必須）
  description TEXT,                             -- 商品説明（任意）
  price NUMERIC(10,2) NOT NULL,                 -- 価格（小数 2 桁）
  category_id INT,                              -- カテゴリ ID（FK → category.id）
  status VARCHAR(20) DEFAULT '出品中',          -- 出品ステータス（既定：出品中）
  image_url TEXT,                               -- 画像 URL（Cloudinary 等）
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- 作成日時（既定で現在時刻）
  FOREIGN KEY (user_id) REFERENCES users(id),   -- 出品者 FK 制約
  FOREIGN KEY (category_id) REFERENCES category(id) -- カテゴリ FK 制約
);
```

```
-- ===== app_order（注文）テーブル作成 =====
CREATE TABLE app_order (
  id SERIAL PRIMARY KEY,                          -- 主キー（連番）
  item_id INT NOT NULL,                           -- 対象商品 ID（FK → item.id）
  buyer_id INT NOT NULL,                          -- 購入者 ID（FK → users.id）
  price NUMERIC(10,2) NOT NULL,                   -- 購入時の価格スナップショット
  status VARCHAR(20) DEFAULT '購入済',            -- 注文ステータス（購入済/発送済など）
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- 作成日時
  FOREIGN KEY (item_id) REFERENCES item(id),      -- 商品 FK 制約
  FOREIGN KEY (buyer_id) REFERENCES users(id)     -- 購入者 FK 制約
);

-- ===== chat（取引チャット）テーブル作成 =====
CREATE TABLE chat (
  id SERIAL PRIMARY KEY,                          -- 主キー（連番）
  item_id INT NOT NULL,                           -- 対象商品 ID（FK → item.id）
  sender_id INT NOT NULL,                         -- 送信者ユーザ ID（FK → users.id）
  message TEXT,                                   -- メッセージ本文
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- 送信日時
  FOREIGN KEY (item_id) REFERENCES item(id),      -- 商品 FK 制約
  FOREIGN KEY (sender_id) REFERENCES users(id)    -- 送信者 FK 制約
);

-- ===== favorite_item（お気に入り）テーブル作成 =====
CREATE TABLE favorite_item (
  id SERIAL PRIMARY KEY,                          -- 主キー（連番）
  user_id INT NOT NULL,                           -- ユーザ ID（FK → users.id）
  item_id INT NOT NULL,                           -- 商品 ID（FK → item.id）
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- 登録日時
  UNIQUE (user_id, item_id),                      -- 同一ユーザの重複お気に入りを禁止
  FOREIGN KEY (user_id) REFERENCES users(id),     -- ユーザ FK 制約
  FOREIGN KEY (item_id) REFERENCES item(id)       -- 商品 FK 制約
);

-- ===== review（評価）テーブル作成 =====
CREATE TABLE review (
  id SERIAL PRIMARY KEY,                          -- 主キー（連番）
  order_id INT NOT NULL UNIQUE,                   -- 対象注文（1 注文 1 レビュー）
  reviewer_id INT NOT NULL,                       -- レビュワー（購入者）
  seller_id INT NOT NULL,                         -- 出品者
  item_id INT NOT NULL,                           -- 評価対象商品
  rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5), -- 1〜5 の整数
  comment TEXT,                                   -- コメント（任意）
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- 登録日時
  FOREIGN KEY (order_id) REFERENCES app_order(id),    -- 注文 FK
  FOREIGN KEY (reviewer_id) REFERENCES users(id),     -- レビュワーFK
  FOREIGN KEY (seller_id) REFERENCES users(id),       -- 出品者 FK
  FOREIGN KEY (item_id) REFERENCES item(id)           -- 商品 FK
);
```

```
-- ===== パフォーマンス向上のためのインデックス =====
CREATE INDEX IF NOT EXISTS idx_item_user_id      ON item(user_id);         -- 出品者検索用
CREATE INDEX IF NOT EXISTS idx_item_category_id  ON item(category_id);     -- カテゴリ検索用
CREATE INDEX IF NOT EXISTS idx_order_item_id     ON app_order(item_id);    -- 注文→商品参照
CREATE INDEX IF NOT EXISTS idx_order_buyer_id    ON app_order(buyer_id);   -- 注文→購入者参照
CREATE INDEX IF NOT EXISTS idx_chat_item_id      ON chat(item_id);         -- チャット→商品参照
CREATE INDEX IF NOT EXISTS idx_chat_sender_id    ON chat(sender_id);    -- チャット→送信者参照
CREATE INDEX IF NOT EXISTS idx_fav_user_id       ON favorite_item(user_id); -- お気に入り→ユーザ参照
CREATE INDEX IF NOT EXISTS idx_fav_item_id       ON favorite_item(item_id); -- お気に入り→商品参照
CREATE INDEX IF NOT EXISTS idx_review_order_id   ON review(order_id);       -- レビュー→注文参照
```

data.sql（初期データ：ユーザ/カテゴリ/商品）

```
-- 初期ユーザ投入（開発用）。NoOpPasswordEncoder 前提で平文パスワード
INSERT INTO users (name, email, password, role) VALUES
  -- 出品者：メールとパスワード
  ('出品者 A', 'sellerA@example.com', 'password', 'USER'),
  -- 購入者：わかりやすいメールに修正（'z'は誤りだと運用上混乱するため）
  ('購入者 B', 'buyerB@example.com', 'password', 'USER'),
  -- 管理者：管理用アカウント
  ('運営者 C', 'adminC@example.com', 'adminpass', 'ADMIN');

-- 初期カテゴリ投入（よく使う 4 種）
INSERT INTO category (name) VALUES
  -- 書籍カテゴリ
  ('本'),
  -- 家電カテゴリ
  ('家電'),
  -- ファッションカテゴリ
  ('ファッション'),
  -- 玩具カテゴリ
  ('おもちゃ');

-- 初期商品投入（出品者 A が 2 商品を出品）
INSERT INTO item (user_id, name, description, price, category_id, status, image_url)
VALUES
  -- Java 入門書（カテゴリ：本、出品中）
  (
    (SELECT id FROM users WHERE email = 'sellerA@example.com'),
    'Java プログラミング入門',
    '初心者向けの Java 入門書です。',
    1500.00,
    (SELECT id FROM category WHERE name = '本'),
    '出品中',
    NULL
  ),
  -- イヤホン（カテゴリ：家電、出品中）
  (
    (SELECT id FROM users WHERE email = 'sellerA@example.com'),
    'ワイヤレスイヤホン',
    'ノイズキャンセリング機能付き。',
    8000.00,
    (SELECT id FROM category WHERE name = '家電'),
    '出品中',
    NULL
  );
```

```
-- （任意）サンプル注文：コメントアウト例。必要時にコメント解除
-- INSERT INTO app_order (item_id, buyer_id, price, status)
-- VALUES (
--   (SELECT id FROM item WHERE name = 'Javaプログラミング入門'),
--   (SELECT id FROM users WHERE email = 'buyerB@example.com'),
--   1500.00,
--   '購入済'
-- );
```

11.付録（設定・依存関係・起動前セルフチェックの実体）

11.1 環境変数と application.properties（秘密情報は環境変数、プロファイルで dev/prod を分離）

11.1.1 .env.sample（macOS/Linux 用。Windows は後述）

```
# ---- DB (PostgreSQL) ----
export DB_URL=jdbc:postgresql://localhost:5432/fleamarket
export DB_USERNAME=fleauser
export DB_PASSWORD=fleapass

# ---- アプリ基本 ----
export SPRING_PROFILES_ACTIVE=dev            # dev / prod を切替
export SERVER_PORT=8080

# ---- Stripe ----
export STRIPE_PUBLIC_KEY=pk_test_xxxxxxxxxxxxxxxxxxxx
export STRIPE_SECRET_KEY=sk_test_xxxxxxxxxxxxxxxxxxxx

# ---- Cloudinary ----
export CLOUDINARY_CLOUD_NAME=your_cloud_name
export CLOUDINARY_API_KEY=111111111111111
export CLOUDINARY_API_SECRET=xxxxxxxxxxxxxxxxxxxxxxxxxxxx

# ---- LINE Notify ----
export LINE_NOTIFY_ENDPOINT=https://notify-api.line.me/api/notify
# 個人のアクセストークンは users テーブルに保存する想定。全体通知を試すなら下を使う
export LINE_NOTIFY_TOKEN=                                  # （任意）

# ---- ログ ----
export LOGGING_LEVEL=INFO
```

Windows PowerShell 例：

```
$env:DB_URL="jdbc:postgresql://localhost:5432/fleamarket"
$env:DB_USERNAME="fleauser"
$env:DB_PASSWORD="fleapass"
$env:SPRING_PROFILES_ACTIVE="dev"
$env:SERVER_PORT="8080"
$env:STRIPE_PUBLIC_KEY="pk_test_xxx"
$env:STRIPE_SECRET_KEY="sk_test_xxx"
$env:CLOUDINARY_CLOUD_NAME="your_cloud_name"
$env:CLOUDINARY_API_KEY="111111111111111"
$env:CLOUDINARY_API_SECRET="xxxxxxxxxxxxxxxx"
$env:LINE_NOTIFY_ENDPOINT="https://notify-api.line.me/api/notify"
$env:LINE_NOTIFY_TOKEN=""
$env:LOGGING_LEVEL="INFO"
```

11.1.2 src/main/resources/application.properties（共通・最小）

```
# --- 共通（プロファイルごとに上書き前提） ---
spring.application.name=fleamarket-system
server.port=${SERVER_PORT:8080}

# DB は必ず環境変数経由
spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
spring.datasource.hikari.maximum-pool-size=10

# JPA / Hibernate
spring.jpa.open-in-view=false
spring.jpa.hibernate.ddl-auto=none              # schema.sql を使うため none
spring.jpa.properties.hibernate.jdbc.time_zone=UTC
spring.jpa.show-sql=false
spring.jpa.properties.hibernate.format_sql=true

# SQL 初期化（schema.sql / data.sql を使う）
spring.sql.init.mode=always
spring.sql.init.encoding=UTF-8

# Thymeleaf
spring.thymeleaf.cache=${THYMELEAF_CACHE:true}

# Actuator（ヘルスチェックのみ外部公開）
management.endpoints.web.exposure.include=health,info
management.endpoint.health.show-details=never

# セキュリティ（フォームログイン前提の最小セット想定）
# 追加の詳細は SecurityConfig で管理

# 外部連携キー（環境変数必須）
stripe.public-key=${STRIPE_PUBLIC_KEY}
stripe.secret-key=${STRIPE_SECRET_KEY}

cloudinary.cloud_name=${CLOUDINARY_CLOUD_NAME}
cloudinary.api_key=${CLOUDINARY_API_KEY}
cloudinary.api_secret=${CLOUDINARY_API_SECRET}

line.notify.endpoint=${LINE_NOTIFY_ENDPOINT:https://notify-
api.line.me/api/notify}
line.notify.token=${LINE_NOTIFY_TOKEN:}    # 通常は users テーブルの個別トークンを使用

# ログ
logging.level.root=${LOGGING_LEVEL:INFO}
logging.level.org.springframework.security=INFO
```

11.1.3 application-dev.properties（開発）

```
spring.thymeleaf.cache=false
spring.jpa.show-sql=true
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.orm.jdbc.bind=TRACE
```

11.1.4 application-prod.properties（本番）

```
spring.thymeleaf.cache=true
spring.jpa.show-sql=false
server.forward-headers-strategy=framework
```

セッション/CSRF などは SecurityConfig 側も必ず最終確認
プロファイル切替は SPRING_PROFILES_ACTIVE=dev / prod で行います
（application.properties に固定で書かない方が安全）。

11.2 依存関係（Maven）（Stripe/Cloudinary のバージョンを固定）

pom.xml（Java 17、Spring Boot 3 系を想定。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.3.2</version>
    <relativePath/>
  </parent>

  <groupId>com.example</groupId>
  <artifactId>fleamarket-system</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>fleamarket-system</name>
  <description>Flea Market App</description>

  <properties>
    <java.version>17</java.version>
    <!-- 外部ライブラリは“固定” -->
    <stripe.java.version>24.45.0</stripe.java.version>
    <cloudinary.java.version>1.39.0</cloudinary.java.version>
  </properties>

  <dependencies>
    <!-- Spring 基本 -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
```

```
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

    <!-- DB -->
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <scope>runtime</scope>
    </dependency>

    <!-- 外部連携 -->
    <dependency>
      <groupId>com.stripe</groupId>
      <artifactId>stripe-java</artifactId>
      <version>${stripe.java.version}</version>
    </dependency>
    <!-- Cloudinary の HTTPClient 実装付きアーティファクト -->
    <dependency>
      <groupId>com.cloudinary</groupId>
      <artifactId>cloudinary-http44</artifactId>
      <version>${cloudinary.java.version}</version>
    </dependency>

    <!-- 開発支援 -->
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>

    <!-- テスト -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

```
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <excludes>
            <exclude>
              <groupId>org.projectlombok</groupId>
              <artifactId>lombok</artifactId>
            </exclude>
          </excludes>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.11.0</version>
        <configuration>
          <release>${java.version}</release>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Stripe/Cloudinary の version は「固定」。上げるときはローカルで必ずビルド確認してから。

11.3 起動前セルフチェック（“落ちにくい順番”で実行確認）

1. **PostgreSQL を起動**
   - DB を作成：fleamarket
   - ユーザー：fleauser / fleapass（任意で OK。.env と合わせる）
2. **環境変数を読み込む**
   - macOS/Linux: source .env
   - Windows: 上記 PowerShell の $env: でセット
3. **ビルド**
4. mvn -q -DskipTests package
5. **schema.sql / data.sql が効くか確認**
   - アプリを起動してテーブル&初期データが入り、エラーが無いことを確認
6. **起動**
7. java -jar target/fleamarket-system-0.0.1-SNAPSHOT.jar
8. **ヘルスチェック**
9. curl -s http://localhost:${SERVER_PORT:-8080}/actuator/health
10. # => {"status":"UP"} が返れば OK
11. **最低導線の確認**
    - GET /login が表示できる
    - GET /items が空でも表示できる
    - 管理者（data.sql の ADMIN）でログイン → 画面を遷移できる
12. **外部連携の疎通（後からで OK）**
    - **Stripe**：ダッシュボタン無しで PaymentIntent を作成できる（テストキー）
    - **Cloudinary**：ダミー画像をアップロードして URL が返る
    - **LINE Notify**：テストトークンで POST して 200 が返る（失敗は WARN ログに残る）
13. **E2E 確認（開発末期）**
    - ログイン → 出品 → 購入（4242 4242 4242 4242）→ レビュー投稿 → 管理画面で統計