Politechnika Świętokrzyska w Kielcach
Wydział Elektrotechniki, Automatyki i Informatyki

Politechnika Świętokrzyska
Kielce University of Technology

Inżynieria Systemów Informacyjnych

Projekt

System Zarządzania Apteką

Skład zespołu:
Artur Szymkiewicz

Kierunek/specjalność: Informatyka 3 rok
Studia: niestacjonarne
Numer grupy: 11A

Sprawozdanie z projektu "System Zarządzania Apteką"

System Zarządzania Apteką to kompleksowa aplikacja oparta na frameworku Spring Boot, która zapewnia pełne funkcjonalności związane z zarządzaniem apteką. Projekt charakteryzuje się zaawansowaną architekturą wielowarstwową, implementacją wzorców projektowych oraz kompleksowym zestawem testów jednostkowych i integracyjnych. System obsługuje uwierzytelnianie JWT, płatności przez Stripe, zarządzanie zamówieniami, inwentaryzację, reklamacje oraz generowanie raportów analitycznych.

Architektura i struktura projektu

Organizacja kodu źródłowego

Projekt jest zorganizowany zgodnie z najlepszymi praktykami Spring Boot w strukturze wielowarstwowej. Główny kod źródłowy znajduje się w pakiecie `com.systemzarzadzaniaapteka` i jest podzielony na logicznie powiązane moduły. Architektura opiera się na wzorcu MVC (Model-View-Controller), gdzie każda warstwa ma określone odpowiedzialności i jest luźno powiązana z pozostałymi warstwami.

Struktura pakietów obejmuje konfigurację bezpieczeństwa (`config`), kontrolery REST API (`controller`), obiekty transferu danych (`dto`), obsługę wyjątków (`exception`), modele biznesowe (`model`), repozytoria dostępu do danych (`repository`), komponenty bezpieczeństwa (`security`) oraz serwisy biznesowe (`service`). Taka organizacja zapewnia wysoką kohezję w ramach każdego pakietu i niskie sprzężenie między pakietami.

Warstwa prezentacji - kontrolery REST

Warstwa kontrolerów składa się z 12 wyspecjalizowanych kontrolerów, z których każdy odpowiada za określony obszar funkcjonalny systemu. **AuthController** zarządza procesami uwierzytelniania i autoryzacji użytkowników. **PaymentController** obsługuje kompleksowy proces płatności, w tym integrację z zewnętrznym dostawcą płatności Stripe. **OrderController** zarządza całym cyklem życia zamówień, od składania przez przetwarzanie do finalizacji.

MedicineController zapewnia funkcjonalności związane z zarządzaniem asortymentem leków, **CartController** obsługuje koszyki zakupowe klientów, a **CustomerController** zarządza danymi i procesami związanymi z klientami apteki. Pozostałe kontrolery obejmują zarządzanie reklamacjami (**ComplaintController**), powiadomieniami (**NotificationController**), promocjami (**PromotionController**), raportami (**ReportController**), bazą danych (**DatabaseController**) oraz użytkownikami systemu (**UserController**).

Warstwa biznesowa i logika aplikacji

Serwisy biznesowe

Warstwa serwisów implementuje główną logikę biznesową aplikacji i składa się z 9 wyspecjalizowanych serwisów. **PaymentService** implementuje kompleksowy proces obsługi płatności, w tym tworzenie intencji płatności w systemie Stripe, walidację danych płatności oraz przetwarzanie transakcji. Serwis wykorzystuje wzorzec Strategy do obsługi różnych metod płatności i implementuje mechanizmy zabezpieczające przed błędami transakcyjnymi.

OrderService zarządza kompleksowym procesem zamówień, implementując logikę biznesową związaną z walidacją dostępności produktów, kalkulacją cen, zarządzaniem stanami zamówień oraz koordynacją z innymi serwisami. **UserService** implementuje funkcjonalności związane z zarządzaniem użytkownikami, w tym rejestrację, aktualizację danych, zarządzanie rolami i uprawnieniami.

Pozostałe serwisy obejmują **CartService** do zarządzania koszykami zakupowymi, **MedicineService** do operacji na asortymencie leków, **NotificationService** do wysyłania powiadomień, **ComplaintService** do obsługi reklamacji, **PromotionService** do zarządzania promocjami oraz **ReportService** do generowania raportów analitycznych.

Modele biznesowe i encje

System definiuje ponad 40 modeli biznesowych, które reprezentują różne aspekty działalności apteki. Kluczowe modele obejmują **Medicine** reprezentujący leki w asortymencie, **Order** i **OrderItem** dla zamówień i ich pozycji, **Payment** dla transakcji płatniczych, **Customer** dla klientów apteki oraz **AppUser** dla użytkowników systemu.

Modele związane z procesami biznesowymi obejmują **Cart** i **CartItem** dla koszyków zakupowych, **Complaint** dla reklamacji, **Promotion** dla promocji i rabatów, **Notification** dla systemu powiadomień oraz **Report** dla raportów analitycznych. System obsługuje również zaawansowane funkcjonalności jak **EPrescription** dla e-recept, **ComplianceAudit** dla audytów zgodności, **InventoryAlert** dla alertów magazynowych oraz **SalesData** dla danych sprzedażowych.

Mechanizmy bezpieczeństwa i uwierzytelniania

Implementacja JWT

System implementuje zaawansowany mechanizm uwierzytelniania oparty na tokenach JWT (JSON Web Tokens). **JwtTokenProvider** jest odpowiedzialny za generowanie, walidację i parsowanie tokenów JWT.

Tokeny są generowane po pomyślnym uwierzytelnieniu użytkownika i zawierają zakodowane informacje o tożsamości i uprawnieniach użytkownika.

JwtAuthenticationFilter działa jako filtr uwierzytelniania, który przechwytuje przychodzące żądania HTTP, wyciąga tokeny JWT z nagłówków Authorization, waliduje je za pomocą JwtTokenProvider i ustawia odpowiedni kontekst bezpieczeństwa w SecurityContextHolder. Mechanizm ten zapewnia bezstanowe uwierzytelnianie, co jest idealne dla aplikacji REST API.

Konfiguracja bezpieczeństwa

SecurityConfig definiuje kompleksową konfigurację bezpieczeństwa aplikacji, w tym reguły autoryzacji dla różnych endpointów, konfigurację OAuth2 oraz integrację z filtrem JWT. Klasa implementuje wzorzec konfiguracyjny Spring Security, definiując łańcuch filtrów bezpieczeństwa i reguły dostępu do zasobów.

System obsługuje również **CustomOidcUser** dla integracji z dostawcami tożsamości OAuth2, co umożliwia uwierzytelnianie przez zewnętrzne systemy przy zachowaniu spójności z wewnętrznym systemem uprawnień.

Warstwa dostępu do danych

Repozytoria i persystencja

Warstwa dostępu do danych składa się z 8 wyspecjalizowanych repozytoriów, które implementują wzorzec Repository dla różnych encji biznesowych. **OrderRepository** obsługuje zapytania związane z zamówieniami, **PaymentRepository** zarządza danymi płatności, **MedicineRepository** obsługuje operacje na lekach, a **CustomerRepository** zarządza danymi klientów.

Pozostałe repozytoria obejmują **CartRepository** dla koszyków zakupowych, **ComplaintRepository** dla reklamacji, **PromotionRepository** dla promocji oraz **UserRepository** dla użytkowników systemu. Każde repozytorium dziedziczy z Spring Data JPA i może definiować własne zapytania dostosowane do specyficznych potrzeb biznesowych.

Konwersja danych

System implementuje **LocalDateTimeConverter** do obsługi konwersji typów danych między aplikacją a bazą danych, zapewniając właściwe mapowanie typów temporalnych Java na odpowiednie typy bazodanowe. To rozwiązanie zapewnia spójność w obsłudze dat i czasu w całym systemie.

Mechanizm płatności Stripe

Integracja z bramką płatniczą

System implementuje kompleksową integrację z systemem płatności Stripe przez **PaymentService**. Proces płatności rozpoczyna się od utworzenia intencji płatności (**PaymentIntent**) w systemie Stripe za pomocą metody `createPaymentIntent`, która przelicza kwotę zamówienia na odpowiednie jednostki walutowe i tworzy odpowiedni obiekt w systemie Stripe.

Przetwarzanie płatności odbywa się przez metodę `processPayment`, która waliduje dane płatności, sprawdza zgodność kwot, a następnie inicjuje proces przetwarzania płatności przez bramkę płatniczą. System implementuje mechanizmy obsługi błędów i rollback w przypadku niepowodzenia transakcji, zapewniając integralność danych finansowych.

Zarządzanie stanami płatności

System definiuje enum **PaymentStatus** do zarządzania różnymi stanami płatności, takimi jak PENDING, PAID, FAILED, co umożliwia precyzyjne śledzenie statusu każdej transakcji. Podobnie **OrderStatus** zarządza stanami zamówień, które są synchronizowane ze stanami płatności w celu zapewnienia spójności biznesowej.

Obsługa wyjątków i zarządzanie błędami

Globalna obsługa wyjątków

System implementuje **GlobalExceptionHandler** wykorzystujący adnotację `@ControllerAdvice` do przechwytywania i obsługi wyjątków w całej aplikacji. Ten mechanizm zapewnia spójną obsługę błędów i generowanie odpowiednich odpowiedzi HTTP z właściwymi kodami statusu i komunikatami błędów.

System definiuje specjalizowane wyjątki biznesowe, takie jak **InsufficientStockException** dla sytuacji braku wystarczającego zapasu, **PaymentProcessingException** dla błędów przetwarzania płatności oraz **ProductNotFoundException** dla sytuacji, gdy produkt nie zostanie znaleziony. Każdy wyjątek niesie specyficzne informacje kontekstowe, umożliwiając precyzyjną diagnostykę i obsługę błędów.

Obiekty transferu danych

DTO i mapowanie danych

System wykorzystuje wzorzec Data Transfer Object (DTO) do transferu danych między warstwami aplikacji. **OrderDto** i **OrderItemDto** enkapsulują dane zamówień w sposób optymalny dla transferu przez API, ukrywając wewnętrzną strukturę encji biznesowych przed klientami API.

PaymentRequest służy jako DTO dla żądań płatności, zawierając wszystkie niezbędne informacje do przetworzenia transakcji płatniczej. Wykorzystanie wzorca DTO zapewnia lepszą enkapsulację, optymalizację transferu danych oraz niezależność API od wewnętrznej struktury danych.

Wzorce projektowe i architektoniczne

Wzorce kreacyjne i strukturalne

System implementuje wzorzec **Singleton** przez Spring IoC Container dla serwisów i komponentów oznaczonych adnotacjami `@Service` i `@Component`. Wzorzec **Builder** jest wykorzystywany w integracji ze Stripe API do tworzenia złożonych obiektów konfiguracyjnych.

Wzorzec **Facade** jest implementowany przez klasy serwisowe, które ukrywają złożoność operacji na repozytoriach i innych komponentach przed warstwą kontrolerów. **Adapter** jest wykorzystywany w `LocalDateTimeConverter` do adaptacji typów danych między różnymi warstwami systemu.

Wzorce behawioralne

System implementuje wzorzec **Strategy** w mechanizmie przetwarzania płatności, gdzie różne strategie mogą być stosowane w zależności od typu płatności lub dostawcy usług płatniczych. Wzorzec **Observer** jest wykorzystywany w systemie powiadomień, gdzie różne komponenty systemu mogą subskrybować wydarzenia biznesowe.

Wzorzec **Template Method** jest wykorzystywany w hierarchii klas użytkowników, gdzie wspólne zachowania są definiowane w klasie bazowej, a specyficzne implementacje w klasach pochodnych.

Testowanie i jakość kodu

Kompleksowy zestaw testów

Projekt zawiera rozbudowany zestaw testów jednostkowych i integracyjnych, pokrywający wszystkie warstwy aplikacji. Testy kontrolerów weryfikują poprawność endpointów REST API, obsługę żądań HTTP oraz

generowanie odpowiednich odpowiedzi. Testy serwisów sprawdzają logikę biznesową, a testy modeli weryfikują poprawność mapowania danych i walidacji.

Testy integracyjne weryfikują współpracę między różnymi komponentami systemu, szczególnie w obszarach krytycznych jak przetwarzanie płatności i zarządzanie zamówieniami. System testów obejmuje również testy bezpieczeństwa, weryfikujące poprawność uwierzytelniania i autoryzacji.

Konfiguracja testowa

Projekt zawiera dedykowaną konfigurację testową, w tym **TestSecurityConfig** dla testów bezpieczeństwa, **MockJwtFilter** do mockowania uwierzytelniania w testach oraz `application-test.properties` z konfiguracją dla środowiska testowego. To zapewnia izolację testów od konfiguracji produkcyjnej i umożliwia testowanie w kontrolowanych warunkach.

Zaawansowane funkcjonalności biznesowe

System analityki i raportowania

System implementuje zaawansowane funkcjonalności analityczne przez modele **Analytics**, **SalesData** oraz **SalesReport**. **ReportService** generuje różnorodne raporty biznesowe, w tym raporty sprzedaży, analizy trendów oraz raporty zgodności regulacyjnej. System obsługuje również **ComplianceAudit** i **ComplianceReport** do zarządzania zgodnością z przepisami farmaceutycznymi.

Zarządzanie promocjami i marketingiem

System zawiera kompleksowy moduł zarządzania promocjami przez **Promotion**, **PromotionManagement** oraz **MarketingCampaign**. **PromotionService** implementuje logikę biznesową związaną z tworzeniem, zarządzaniem i stosowaniem różnych typów promocji i rabatów. System obsługuje również **Discount** dla indywidualnych rabatów oraz **CustomerCard** dla programów lojalnościowych.

E-recepty i compliance

System implementuje obsługę e-recept przez model **EPrescription** oraz **PrescriptionCheck** do weryfikacji recept. Te funkcjonalności są kluczowe dla zgodności z przepisami farmaceutycznymi i zapewniają właściwą kontrolę wydawania leków na receptę.

Zarządzanie zasobami i inwentaryzacja / System alertów i kontroli zapasów

System implementuje proaktywne zarządzanie zapasami przez **InventoryAlert**, **InventoryCheck** oraz **StockCheck**. Te komponenty monitorują stany magazynowe i generują alerty przy niskich poziomach zapasów, umożliwiając proaktywne zarządzanie asortymentem i zapobieganie brakom towaru.

PriceChange umożliwia śledzenie zmian cen produktów, co jest istotne dla analiz finansowych i zarządzania marżami. System obsługuje również **Supplier** i **Wholesaler** do zarządzania relacjami z dostawcami i hurtowniami.

Fakty:

Backend

Pakiet „com.systemzarzadzaniaapteka”
- SystemZarzadzaniaAptekaApplication.java: Główna klasa aplikacji Spring Boot, odpowiedzialna za uruchamianie aplikacji. Jest to punkt wejścia do systemu.

Pakiet „com.systemzarzadzaniaapteka.config”
- SecurityConfig.java: Klasa konfiguracyjna dla Spring Security, definiująca reguły uwierzytelniania i autoryzacji, w tym konfigurację OAuth2 i filtrowania tokenów JWT.

Pakiet „com.systemzarzadzaniaapteka.controller”
- AuthController.java: Kontroler obsługujący uwierzytelnianie użytkowników, w tym logowanie i rejestrację.
- CartController.java: Kontroler zarządzający koszykiem zakupowym klientów, umożliwia dodawanie i usuwanie produktów.
- ComplaintController.java: Kontroler do obsługi reklamacji składanych przez klientów.
- CustomerController.java: Kontroler zarządzający danymi klientów, w tym rejestracją i aktualizacją danych.
- DatabaseController.java: Kontroler do zarządzania operacjami na bazie danych, np. resetowanie danych.
- MedicineController.java: Kontroler obsługujący operacje na lekach, takie jak przeglądanie i zarządzanie asortymentem.
- NotificationController.java: Kontroler do wysyłania i zarządzania powiadomieniami dla użytkowników.
- OrderController.java: Kontroler zarządzający zamówieniami, w tym składanie i śledzenie zamówień.
- PaymentController.java: Kontroler obsługujący płatności, w tym tworzenie intencji płatności i przetwarzanie płatności za pomocą Stripe.
- PromotionController.java: Kontroler do zarządzania promocjami i rabatami na leki.
- ReportController.java: Kontroler generujący raporty, np. sprzedaży lub zgodności.
- UserController.java: Kontroler zarządzający użytkownikami systemu, w tym ich rolami i uprawnieniami.

Pakiet „com.systemzarzadzaniaapteka.dto”
- OrderDto.java: Obiekt transferu danych dla zamówień, używany do przekazywania informacji o zamówieniu między warstwami aplikacji.
- OrderItemDto.java: Obiekt transferu danych dla pozycji zamówienia, zawierający informacje o produkcie i ilości.
- PaymentRequest.java: Obiekt transferu danych dla żądań płatności, zawierający informacje o zamówieniu i metodzie płatności.

Pakiet „com.systemzarzadzaniaapteka.exception”
- GlobalExceptionHandler.java: Globalny obsługiwacz wyjątków, który przechwytuje i obsługuje wyjątki rzucane w aplikacji.
- InsufficientStockException.java: Wyjątek rzucany, gdy brak jest wystarczającej ilości produktu w magazynie.
- PaymentProcessingException.java: Wyjątek rzucany w przypadku problemów z przetwarzaniem płatności.
- ProductNotFoundException.java: Wyjątek rzucany, gdy produkt nie zostanie znaleziony w systemie.

Pakiet „com.systemzarzadzaniaapteka.model”
- Admin.java: Model reprezentujący administratora systemu.
- Analytics.java: Model do przechowywania danych analitycznych, np. sprzedaży.
- AppUser.java: Model reprezentujący użytkownika aplikacji, z informacjami o roli i danych uwierzytelniających.
- Cart.java: Model koszyka zakupowego klienta.
- CartItem.java: Model pozycji w koszyku, zawierający informacje o produkcie i ilości.
- Complaint.java: Model reklamacji składanej przez klienta.
- ComplianceAudit.java: Model audytu zgodności.
- ComplianceReport.java: Model raportu zgodności.
- Customer.java: Model klienta apteki.
- CustomerCard.java: Model karty klienta, np. lojalnościowej.
- CustomerRegistration.java: Model rejestracji klienta.
- Delivery.java: Model dostawy związanej z zamówieniem.
- Discount.java: Model rabatu na leki.
- DrugComplaint.java: Model reklamacji dotyczącej konkretnego leku.
- EPrescription.java: Model e-recepty.
- Feedback.java: Model opinii klienta.
- InventoryAlert.java: Model alertu o niskim stanie magazynowym.
- InventoryCheck.java: Model kontroli stanu magazynowego.
- Invoice.java: Model faktury związanej z zamówieniem.
- LocalDateTimeConverter.java: Konwerter daty i czasu dla bazy danych.
- Manager.java: Model managera apteki.
- MarketingCampaign.java: Model kampanii marketingowej.
- Medicine.java: Model leku w asortymencie apteki.
- Notification.java: Model powiadomienia wysyłanego do użytkownika.
- OnlineOrder.java: Model zamówienia online.
- Order.java: Model zamówienia klienta.
- OrderItem.java: Model pozycji w zamówieniu.
- OrderStatus.java: Enum reprezentujący status zamówienia.
- Owner.java: Model właściciela apteki.
- Payment.java: Model płatności związanej z zamówieniem.
- PaymentIntent.java: Model intencji płatności Stripe.
- PaymentStatus.java: Enum reprezentujący status płatności.
- Pharmacist.java: Model farmaceuty.
- PrescriptionCheck.java: Model weryfikacji recepty.
- PriceChange.java: Model zmiany ceny leku.
- Promotion.java: Model promocji na leki.
- PromotionManagement.java: Model zarządzania promocjami.
- Reminder.java: Model przypomnienia dla klienta.
- Report.java: Model raportu generowanego w systemie.
- ReportRequest.java: Model żądania raportu.
- SalesData.java: Model danych sprzedaży.
- SalesReport.java: Model raportu sprzedaży.
- StockCheck.java: Model kontroli zapasów.
- Supplier.java: Model dostawcy leków.

- Wholesaler.java: Model hurtowni leków.

Pakiet „com.systemzarzadzaniaapteka.repository”
- CartRepository.java: Repozytorium dla operacji na koszyku.
- ComplaintRepository.java: Repozytorium dla reklamacji.
- CustomerRepository.java: Repozytorium dla klientów.
- MedicineRepository.java: Repozytorium dla leków.
- OrderRepository.java: Repozytorium dla zamówień.
- PaymentRepository.java: Repozytorium dla płatności.
- PromotionRepository.java: Repozytorium dla promocji.
- UserRepository.java: Repozytorium dla użytkowników.

Pakiet „com.systemzarzadzaniaapteka.security”
- CustomOidcUser.java: Klasa reprezentująca użytkownika OAuth2.
- JwtAuthenticationFilter.java: Filtr uwierzytelniania JWT, który sprawdza tokeny w nagłówkach żądań HTTP i ustawia kontekst bezpieczeństwa.
- JwtTokenProvider.java: Dostawca tokenów JWT, odpowiedzialny za generowanie, walidację i parsowanie tokenów.

Pakiet „com.systemzarzadzaniaapteka.service”
- CartService.java: Serwis zarządzający koszykiem zakupowym.
- ComplaintService.java: Serwis obsługujący reklamacje.
- MedicineService.java: Serwis zarządzający lekami.
- NotificationService.java: Serwis wysyłający powiadomienia.
- OrderService.java: Serwis zarządzający zamówieniami.
- PaymentService.java: Serwis obsługujący płatności, w tym integrację ze Stripe.
- PromotionService.java: Serwis zarządzający promocjami.
- ReportService.java: Serwis generujący raporty.
- UserService.java: Serwis zarządzający użytkownikami.

Mechanizm przekazywania i generowania tokenów JWT

Tokeny JWT (JSON Web Tokens) są używane w systemie do uwierzytelniania i autoryzacji użytkowników. Mechanizm ten jest zaimplementowany w pakiecie „security”:

- Generowanie tokenów: Klasa „JwtTokenProvider.java” generuje tokeny JWT dla użytkowników po pomyślnym uwierzytelnieniu. Token zawiera informacje o użytkowniku, takie jak ID i rola, oraz jest podpisany tajnym kluczem. Metoda „generateToken(AppUser user)” tworzy token z ważnością 1 dnia (86400000 ms). Kod znajduje się w pliku „src/main/java/com/systemzarzadzaniaapteka/security/JwtTokenProvider.java”.

- Walidacja tokenów: Metoda „validateToken(String authToken)” w klasie „JwtTokenProvider.java” sprawdza, czy token jest poprawny i nie wygasł. Jeśli token jest nieprawidłowy, zwracane jest „false”.

- Przekazywanie tokenów: Tokeny są przekazywane w nagłówku HTTP „Authorization” z prefiksem „Bearer „. Filtr „JwtAuthenticationFilter.java” przechwytuje żądania HTTP, wyciąga token z nagłówka, waliduje go za pomocą „JwtTokenProvider” i ustawia kontekst bezpieczeństwa („SecurityContextHolder”) dla uwierzytelnionego użytkownika. Kod znajduje się w pliku „src/main/java/com/systemzarzadzaniaapteka/security/JwtAuthenticationFilter.java”.

Proces płatności w Stripe

Integracja z Stripe umożliwia przetwarzanie płatności online. Mechanizm ten jest zaimplementowany w klasach „PaymentService.java” i „PaymentController.java”:

- Tworzenie intencji płatności: Metoda „createPaymentIntent(Long orderId)” w klasie „PaymentService.java” tworzy obiekt „PaymentIntent” w Stripe dla danego zamówienia. Kwota zamówienia jest przeliczana na grosze (dla waluty PLN), a następnie tworzony jest obiekt „PaymentIntentCreateParams” z odpowiednią kwotą i walutą. Po utworzeniu intencji płatności, zapisuje się ją w bazie danych jako „Payment” ze statusem „PENDING”. Kod znajduje się w pliku „src/main/java/com/systemzarzadzaniaapteka/service/PaymentService.java”.

- Przetwarzanie płatności: Metoda „processPayment(PaymentRequest request)” w klasie „PaymentService.java” przetwarza płatność dla zamówienia. Sprawdza, czy kwota płatności zgadza się z kwotą zamówienia, a następnie wywołuje metodę „processPaymentWithGateway(PaymentRequest paymentRequest)” do symulacji przetwarzania płatności (w tym przypadku sprawdza, czy ostatnia cyfra numeru karty jest parzysta). Jeśli płatność jest успешна, status płatności jest ustawiany na „PAID”, a status zamówienia na „PAID” i „PROCESSING”. Kod znajduje się w pliku „src/main/java/com/systemzarzadzaniaapteka/service/PaymentService.java”.

- Endpointy płatności: Klasa „PaymentController.java” udostępnia endpointy REST do obsługi płatności. Endpoint „/api/payments/create-intent” tworzy intencję płatności dla zamówienia, a endpoint „/api/payments” przetwarza płatność na podstawie danych z „PaymentRequest”. Kod znajduje się w pliku „src/main/java/com/systemzarzadzaniaapteka/controller/PaymentController.java”.

Użyte wzorce projektowe i ich lokalizacja w kodzie

W projekcie zastosowano kilka wzorców projektowych, które ułatwiają utrzymanie i rozwój kodu:

- Wzorce architektoniczne:
  - MVC (Model-View-Controller): Wzorzec MVC jest zaimplementowany w strukturze projektu Spring Boot. Modele znajdują się w pakiecie „model”, kontrolery w pakiecie „controller”, a widoki (choć nie są częścią backendu) byłyby obsługiwane przez frontend. Przykład: „OrderController.java” w „src/main/java/com/systemzarzadzaniaapteka/controller/OrderController.java” jako kontroler, „Order.java” w „src/main/java/com/systemzarzadzaniaapteka/model/Order.java” jako model.
  - RESTful API: Projekt implementuje API zgodne z zasadami REST, co widać w kontrolerach takich jak „PaymentController.java” z endpointami zgodnymi z REST (np. „POST /api/payments” do przetwarzania płatności). Kod znajduje się w całym pakiecie „controller”.

- Wzorce kreacyjne:
  - Singleton: Wzorzec Singleton jest używany w klasach oznaczonych adnotacją „@Component” lub „@Service”, które są zarządzane przez Spring IoC Container jako pojedyncze instancje. Przykład: „JwtTokenProvider.java” w „src/main/java/com/systemzarzadzaniaapteka/security/JwtTokenProvider.java”.
  - Builder: Wzorzec Builder jest używany w Stripe API do tworzenia obiektów, np. „PaymentIntentCreateParams.builder()” w metodzie „createPaymentIntent” w pliku „src/main/java/com/systemzarzadzaniaapteka/service/PaymentService.java”.

- Wzorce strukturalne:
  - Facade: Wzorzec Facade jest używany w klasach serwisowych, które ukrywają złożoność operacji na repozytoriach i innych komponentach. Przykład: „PaymentService.java” w „src/main/java/com/systemzarzadzaniaapteka/service/PaymentService.java” jako fasada dla operacji płatnościowych.
  - Adapter: Wzorzec Adapter jest używany w „LocalDateTimeConverter.java”, który dostosowuje typy danych między aplikacją a bazą danych. Kod znajduje się w „src/main/java/com/systemzarzadzaniaapteka/model/LocalDateTimeConverter.java”.

- Wzorce behawioralne:
  - Strategy: Wzorzec Strategy jest używany w metodzie „processPaymentWithGateway” w „PaymentService.java”, gdzie różne strategie przetwarzania płatności mogą być zastosowane w zależności od

danych wejściowych (np. numer karty). Kod znajduje się w „src/main/java/com/systemzarzadzaniaapteka/service/PaymentService.java”.
  - Observer: Wzorzec Observer jest zaimplementowany w mechanizmie powiadomień, gdzie „NotificationService.java” może informować użytkowników o zmianach (np. statusu zamówienia). Kod znajduje się w „src/main/java/com/systemzarzadzaniaapteka/service/NotificationService.java”.

- Wzorce Spring:
  - Dependency Injection: Wzorzec wstrzykiwania zależności jest szeroko stosowany w całym projekcie dzięki adnotacjom „@Autowired” i konstruktorom. Przykład: Wstrzykiwanie „PaymentService” do „PaymentController” w pliku „src/main/java/com/systemzarzadzaniaapteka/controller/PaymentController.java”.
  - AOP (Aspect-Oriented Programming): Wzorzec AOP jest używany w obsłudze wyjątków za pomocą „GlobalExceptionHandler.java”, który przechwytuje wyjątki w całej aplikacji. Kod znajduje się w „src/main/java/com/systemzarzadzaniaapteka/exception/GlobalExceptionHandler.java”.

Frontend

Projekt jest aplikacją React Native/Expo, która implementuje system zarządzania apteką z funkcjonalnościami takimi jak przeglądanie leków, koszyk zakupowy, autentykacja użytkowników i proces płatności. Poniżej opisano główne klasy i komponenty:

- RootLayout („app/_layout.tsx”): Główny układ aplikacji, który definiuje strukturę nawigacji lub ogólny layout dla całej aplikacji.
- NotFoundScreen („app/+not-found.tsx”): Komponent obsługujący strony nieznalezione (404), wyświetlany, gdy użytkownik próbuje uzyskać dostęp do nieistniejącej ścieżki.
- CheckoutScreen („app/checkout.tsx”): Ekran realizacji zamówienia, który umożliwia użytkownikowi wprowadzenie adresu dostawy, wybór metody płatności i złożenie zamówienia. Zbiera dane karty płatniczej i przesyła je do backendu.
- TabLayout („app/(tabs)/_layout.tsx”): Układ dla zakładek aplikacji, definiujący nawigację między głównymi sekcjami, takimi jak strona główna, koszyk czy profil.
- HomeScreen („app/(tabs)/index.tsx”): Strona główna aplikacji wyświetlająca listę leków i powitanie użytkownika.
- CartScreen („app/(tabs)/cart.tsx”): Ekran koszyka, który pokazuje dodane produkty i umożliwia przejście do realizacji zamówienia.
- LoginScreen („app/auth/login.tsx”): Ekran logowania, gdzie użytkownicy mogą wprowadzić swoje dane uwierzytelniające lub zalogować się przez Google (na platformie web).
- RegisterScreen („app/auth/register.tsx”): Ekran rejestracji, umożliwiający tworzenie nowych kont użytkowników.
- MedicationCard („components/MedicationCard.tsx”): Komponent wyświetlający informacje o pojedynczym leku, z możliwością kliknięcia w celu uzyskania szczegółów lub dodania do koszyka.

Przekazywanie i generowanie tokenów

Mechanizm autentykacji i zarządzania tokenami jest zaimplementowany w pliku „contexts/AuthContext.tsx”. Poniżej opisano proces:

- Generowanie tokenów: Tokeny są generowane przez backend podczas logowania lub uwierzytelniania przez Google OAuth (na platformie web). Po udanym logowaniu, backend zwraca „accessToken” (prawdopodobnie JWT), który jest zapisywany w „AsyncStorage” pod kluczem „@accessToken”.
- Przechowywanie tokenów: Tokeny są przechowywane lokalnie za pomocą „AsyncStorage”, co pozwala na utrzymanie sesji użytkownika między uruchomieniami aplikacji.
- Przekazywanie tokenów: W pliku „lib/api.ts”, tokeny są dołączane do nagłówków żądań HTTP jako „Authorization: Bearer <token> w klasie „BaseApiService”. Token jest pobierany przed każdym żądaniem za pomocą funkcji „getToken”, co zapewnia, że wszystkie żądania do API są uwierzytelnione, jeśli użytkownik jest zalogowany.

- Integracja z Google OAuth: Na platformie web, aplikacja obsługuje tokeny „id_token” i „access_token” zwracane przez Google, przesyłając „id_token” do backendu w celu uwierzytelnienia i otrzymania własnego tokenu JWT backendu.

Płatności w Stripe
Proces płatności jest częściowo zaimplementowany w pliku „app/checkout.tsx”, ale integracja ze Stripe nie jest widoczna na poziomie frontendu. Poniżej opisano, co udało się ustalić:

- Proces płatności: Użytkownik wprowadza dane karty (numer, imię posiadacza, data ważności, CVV) na ekranie „CheckoutScreen”. Dane te są przesyłane do backendu za pomocą metody „paymentService.processPayment” z klasy „PaymentService” (zdefiniowanej w „lib/api.ts”).
- Integracja ze Stripe: Bezpośrednia integracja ze Stripe nie jest widoczna w kodzie frontendu. Stripe jest obsługiwane po stronie backendu, gdzie dane karty są przesyłane do API Stripe w celu przetworzenia płatności. Frontend jedynie zbiera dane i przesyła je do endpointu „/payments” na backendzie.
- Potwierdzenie płatności: Po udanej płatności, koszyk jest czyszczony, a użytkownik otrzymuje potwierdzenie zamówienia.

Użyte wzorce projektowe i ich lokalizacja
W projekcie zastosowano kilka wzorców projektowych, typowych dla aplikacji React Native:

- Context API (Wzorce zarządzania stanem): Użyty w plikach „contexts/AuthContext.tsx”, „contexts/CartContext.tsx” i „contexts/ApiContext.tsx” do zarządzania globalnym stanem aplikacji, takim jak dane użytkownika, tokeny, zawartość koszyka i dostęp do API.
- Modularność i dziedziczenie (Wzorce obiektowe): W pliku „lib/api.ts” zastosowano wzorzec dziedziczenia, gdzie różne serwisy (np. „UserService”, „PaymentService”) dziedziczą po klasie bazowej „BaseApiService”, co zapewnia spójność w obsłudze żądań API i ponowne użycie kodu.
- Komponenty funkcyjne i Hooks (Wzorce React): Wszystkie komponenty, takie jak „CheckoutScreen”, „LoginScreen” czy „MedicationCard”, są zaimplementowane jako komponenty funkcyjne z użyciem hooks (np. „useState”, „useEffect”), co jest standardem w nowoczesnym React. Przykłady można znaleźć w „app/checkout.tsx” i „contexts/AuthContext.tsx”.
- Nawigacja oparta na zakładkach (Wzorce nawigacyjne): W katalogu „app/(tabs)” zastosowano wzorzec nawigacji opartej na zakładkach, zdefiniowany w „app/(tabs)/_layout.tsx”, co umożliwia łatwe przełączanie między głównymi sekcjami aplikacji.

Diagramy UML
Diagram użycia

Diagram klas

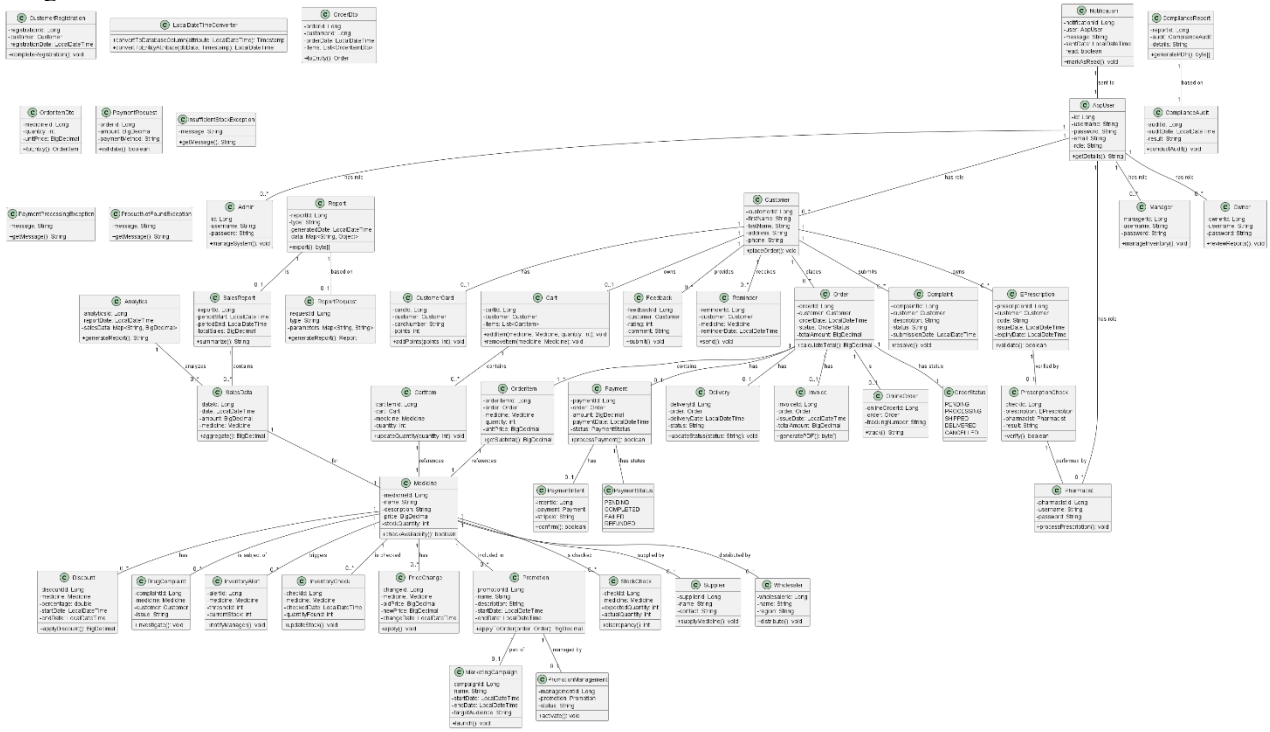
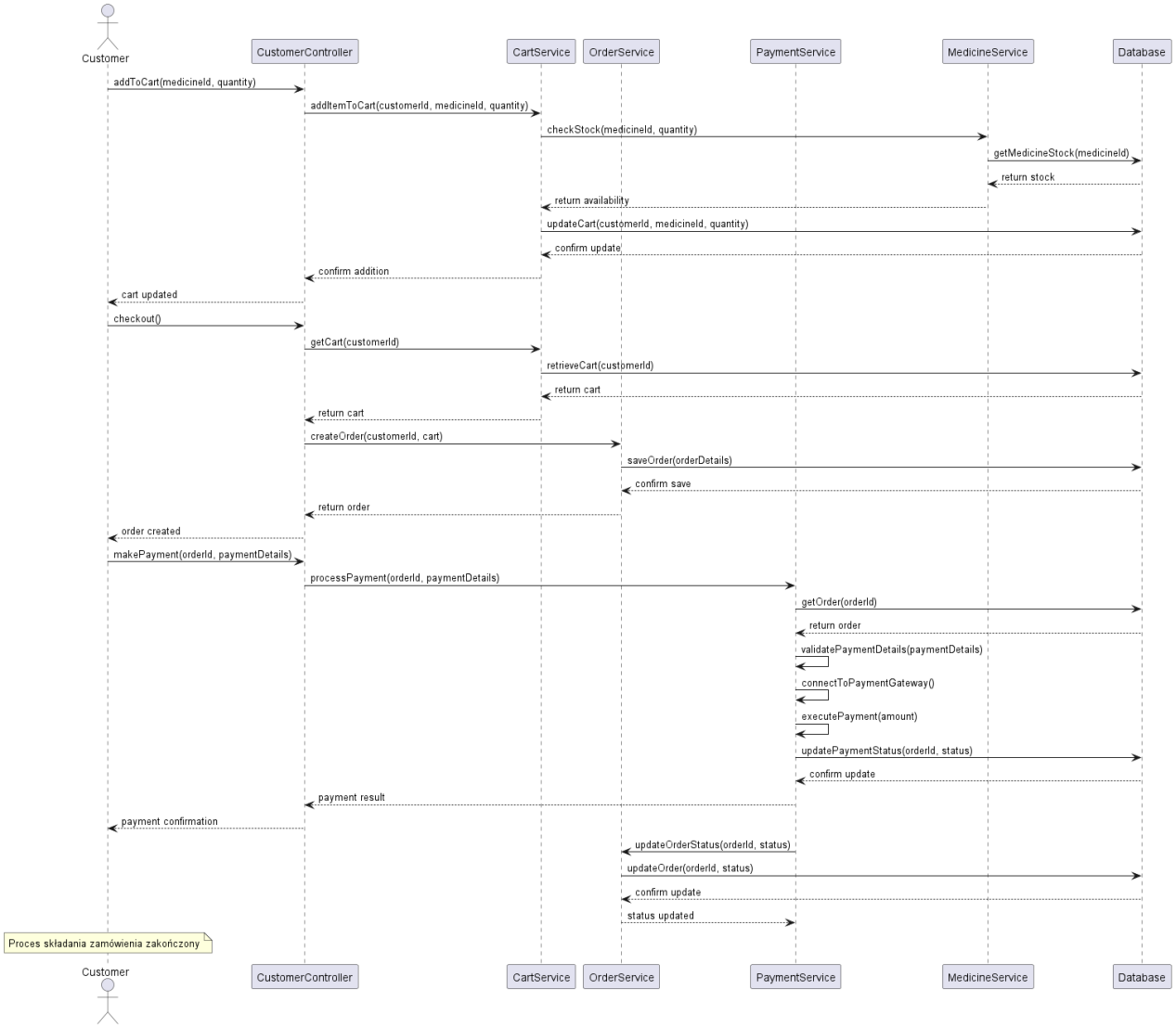


Diagram sekwencji



Podział pracy:
Artur Szymkiewicz: Backend – bezpieczeństwo, użytkownicy, konfiguracja, płatności, zamówienia, koszyki, klienci, reklamacje, raporty, leki, promocje, magazyn, integracje, adaptery

Zakres odpowiedzialności:

- Konfiguracja aplikacji i bezpieczeństwa
- Uwierzytelnianie, autoryzacja (w tym JWT, OAuth2)
- Obsługa użytkowników, ról, powiadomień
- Cały proces płatności (Stripe)
- Globalna obsługa wyjątków
- Obsługa zamówień, koszyków, klientów
- Obsługa reklamacji i raportów
- DTO i modele związane z zamówieniami i klientami
- Audyty zgodności, raporty sprzedaży, analityka
- Zarządzanie asortymentem leków i promocjami
- Zarządzanie magazynem, dostawcami, hurtowniami
- Obsługa rabatów, kampanii marketingowych

- Adaptery, konwertery, walidacje
- Integracja z zewnętrznymi systemami (np. bazy, Stripe Builder)
-

Pliki i pakiety:

- SystemZarzadzaniaAptekaApplication.java
- Pakiet config:
  - SecurityConfig.java
- Pakiet security:
  - CustomOidcUser.java
  - JwtAuthenticationFilter.java
  - JwtTokenProvider.java
- Pakiet controller:
  - AuthController.java
  - CartController.java
  - ComplaintController.java
  - CustomerController.java
  - DatabaseController.java
  - OrderController.java
  - ReportController.java
  - NotificationController.java
  - MedicineController.java
  - PromotionController.java
  - PaymentController.java
  - UserController.java
- Pakiet service:
  - NotificationService.java
  - PaymentService.java
  - UserService.java
  - CartService.java
  - ComplaintService.java
  - OrderService.java
  - ReportService.java
  - MedicineService.java
  - PromotionService.java
  -
- Pakiet dto:
  - OrderDto.java
  - OrderItemDto.java
  - PaymentRequest.java
- Pakiet exception:
  - InsufficientStockException.java
  - ProductNotFoundException.java
  - GlobalExceptionHandler.java
  - PaymentProcessingException.java
- Pakiet model:
  - AppUser.java
  - Admin.java
  - Analytics.java
  - Cart.java
  - CartItem.java
  - Complaint.java
  - ComplianceAudit.java
  - ComplianceReport.java
  - Customer.java

    - CustomerCard.java
    - CustomerRegistration.java
    - Delivery.java
    - DrugComplaint.java
    - EPrescription.java
    - Feedback.java
    - Invoice.java
    - Discount.java
    - InventoryAlert.java
    - InventoryCheck.java
    - LocalDateTimeConverter.java
    - Medicine.java
    - PriceChange.java
    - Promotion.java
    - PromotionManagement.java
    - StockCheck.java
    - Supplier.java
    - Wholesaler.java
    - Manager.java
    - MarketingCampaign.java
    - OnlineOrder.java
    - Order.java
    - OrderItem.java
    - OrderStatus.java
    - Owner.java
    - Pharmacist.java
    - PrescriptionCheck.java
    - Reminder.java
    - Report.java
    - ReportRequest.java
    - SalesData.java
    - SalesReport.java
    - Payment.java
    - PaymentIntent.java
    - PaymentStatus.java
    - Notification.java
- Pakiet repository:
    - CartRepository.java
    - ComplaintRepository.java
    - CustomerRepository.java
    - OrderRepository.java
    - PaymentRepository.java
    - MedicineRepository.java
    - PromotionRepository.java
    - UserRepository.java

Uwagi do podziału:

- Każda osoba odpowiada za testy jednostkowe i integracyjne swoich modułów.
- Współpraca przy integracji całości i testowaniu end-to-end.
- Każda osoba odpowiada za dokumentację swojego zakresu.

Podsumowanie

System Zarządzania Apteką reprezentuje zaawansowaną aplikację enterprise, która implementuje najlepsze praktyki inżynierii oprogramowania i wzorce projektowe. Architektura wielowarstwowa zapewnia separację odpowiedzialności, wysoką testowalność oraz łatwość utrzymania i rozwoju systemu.

Kompleksowy zestaw funkcjonalności obejmuje wszystkie aspekty działalności apteki, od zarządzania asortymentem i zamówieniami, przez przetwarzanie płatności i obsługę klientów, po zaawansowane analizy biznesowe i compliance regulacyjny. System bezpieczeństwa oparty na JWT zapewnia właściwą autoryzację i uwierzytelnianie, a integracja ze Stripe umożliwia bezpieczne przetwarzanie płatności online.

Literatura

1. **React Native – Oficjalna dokumentacja (aplikacja mobilna)**
   https://reactnative.dev/docs/getting-started
   Dostęp: 17 kwietnia 2025
2. **Expo – Oficjalna dokumentacja (środowisko uruchomieniowe)**
   https://docs.expo.dev/
   Dostęp: 19 kwietnia 2025
3. **React Native Paper – Dokumentacja komponentów UI (UI komponenty)**
   https://callstack.github.io/react-native-paper/
   Dostęp: 22 kwietnia 2025
4. **React Navigation/Expo Router – Nawigacja w aplikacjach mobilnych (nawigacja)**
   https://reactnavigation.org/docs/getting-started
   https://expo.github.io/router/docs
   Dostęp: 25 kwietnia 2025
5. **TypeScript – Oficjalna dokumentacja (typowanie)**
   https://www.typescriptlang.org/docs/
   Dostęp: 27 kwietnia 2025
6. **Expo Location – Dokumentacja (lokalizacja GPS)**
   https://docs.expo.dev/versions/latest/sdk/location
   Dostęp: 30 kwietnia 2025
7. **Expo Camera – Dokumentacja (aparat)**
   https://docs.expo.dev/versions/latest/sdk/camera
   Dostęp: 3 maja 2025
8. **Expo Image Picker – Dokumentacja (wybór zdjęć)**
   https://docs.expo.dev/versions/latest/sdk/imagepicker/
   Dostęp: 6 maja 2025
9. **Lucide React Native (ikony) (ikony)**
   https://lucide.dev/docs/react-native/
   Dostęp: 9 maja 2025

10. **JSON Server – Dokumentacja (REST API backend)**
    https://github.com/typicode/json-server
    Dostęp: 12 maja 2025
11. **Jest – Dokumentacja testów jednostkowych (testy jednostkowe)**
    https://jestjs.io/docs/getting-started
    Dostęp: 16 maja 2025
12. **React Native Testing Library – Dokumentacja testowania komponentów (testy komponentów)**
    https://callstack.github.io/react-native-testing-library/
    Dostęp: 19 maja 2025
13. **Android Emulator – Dokumentacja (testy na emulatorze)**
    https://developer.android.com/studio/run/emulator
    Dostęp: 21 maja 2025
14. **React Context – Oficjalny przewodnik (globalny stan)**
    https://react.dev/learn/passing-data-deeply-with-context
    Dostęp: 25 maja 2025
15. **Expo Notifications – Dokumentacja (powiadomienia)**
    https://docs.expo.dev/versions/latest/sdk/notifications/
    Dostęp: 28 maja 2025
16. **React – Oficjalna dokumentacja (JSX, logika komponentów) (komponenty, JSX)**
    https://reactjs.org/docs/getting-started.html
    Dostęp: 2 czerwca 2025
    Spring Boot – Oficjalna dokumentacja (framework główny backendu)
    https://spring.io/projects/spring-boot
    Dostęp: 5 czerwca 2025
17. **Spring Security – Dokumentacja bezpieczeństwa (uwierzytelnianie i autoryzacja)**
    https://spring.io/projects/spring-security
    Dostęp: 12 kwietnia 2025
18. **JSON Web Tokens (JWT) – Specyfikacja RFC 7519 (tokeny uwierzytelniania)**
    https://jwt.io/introduction/
    Dostęp: 12 kwietnia 2025
19. **Spring Data JPA – Dokumentacja (warstwa dostępu do danych)**
    https://spring.io/projects/spring-data-jpa
    Dostęp: 15 kwietnia 2025
20. **Maven – Oficjalna dokumentacja (zarządzanie zależnościami)**
    https://maven.apache.org/guides/index.html
    Dostęp: 18 kwietnia 2025
21. **JUnit 5 – Dokumentacja testów jednostkowych (testy backendu)**
    https://junit.org/junit5/docs/current/user-guide/
    Dostęp: 22 kwietnia 2025
22. **Mockito – Framework mockowania (testy jednostkowe z mockami)**
    https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html
    Dostęp: 25 kwietnia 2025

23. **Stripe API – Dokumentacja płatności (bramka płatnicza)**
    https://stripe.com/docs/api
    Dostęp: 28 kwietnia 2025
24. **Spring Boot Test – Dokumentacja testowania (testy integracyjne)**
    https://spring.io/guides/gs/testing-web/
    Dostęp: 2 maj 2025
25. **Hibernate ORM – Dokumentacja (mapowanie obiektowo-relacyjne)**
    https://hibernate.org/orm/documentation/
    Dostęp: 5 maj 2025
26. **Bean Validation (JSR-303) – Specyfikacja walidacji (walidacja danych)**
    https://beanvalidation.org/2.0/spec/
    Dostęp: 8 maj 2025
27. **Jackson – Dokumentacja serializacji JSON (przetwarzanie JSON)**
    https://github.com/FasterXML/jackson-docs
    Dostęp: 12 maj 2025
28. **Spring Web MVC – Dokumentacja (kontrolery REST)**
    https://docs.spring.io/spring-framework/docs/current/reference/html/web.html
    Dostęp: 15 maj 2025
29. **PostgreSQL – Dokumentacja bazy danych (główna baza danych)**
    https://www.postgresql.org/docs/
    Dostęp: 18 maj 2025
30. **H2 Database – Dokumentacja (baza danych testowa)**
    https://www.h2database.com/html/main.html
    Dostęp: 22 maj 2025
31. **Spring Boot Actuator – Dokumentacja monitorowania (monitoring aplikacji)**
    https://spring.io/guides/gs/actuator-service/
    Dostęp: 25 maj 2025
32. **OAuth 2.0 – Specyfikacja RFC 6749 (autoryzacja zewnętrzna)**
    https://oauth.net/2/
    Dostęp: 28 maj 2025
33. **Firebase Admin SDK – Dokumentacja Java (powiadomienia push)**
    https://firebase.google.com/docs/admin/setup
    Dostęp: 1 czerwca 2025
34. **Spring Profile – Dokumentacja konfiguracji środowisk (zarządzanie konfiguracją)**
    https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.profiles
    Dostęp: 4 czerwca 2025
35. **OpenAPI 3.0 (Swagger) – Specyfikacja dokumentacji API (dokumentacja REST API)**
    https://swagger.io/specification/
    Dostęp: 7 czerwca 2025