

Pełny przegląd testów – wyjaśnienie każdego testu

(10 testów) AvailabilityServiceTest

Test 1 **isAvailable_nullParameters_returnsFalse**

Co robi: Przekazuje wartości null dla ID pokoju, daty początkowej lub końcowej

Co testuje: Walidację danych wejściowych – usługa powinna obsługiwać brakujące parametry

Oczekiwane zachowanie: Zwraca false dla dowolnego parametru null

Dlaczego ważne: Zapobiega awariom typu NullPointerException w produkcji

Test 2 **isAvailable_invalidDateRange_returnsFalse**

Co robi: Sprawdza niepoprawne zakresy dat (od > do, lub od = do)

Co testuje: Poprawność logiki dat

Oczekiwane zachowanie: Zwraca false dla niemożliwych zakresów dat

Zasada biznesowa: Nie można wymeldować się przed dniem zameldowania ani tego samego dnia

Test 3 **isAvailable_maintenanceRoom_returnsFalse**

Co robi: Próbuje zarezerwować pokój będący w remoncie

Co testuje: Funkcję blokady pokoi w remoncie

Oczekiwane zachowanie: Zwraca false dla pokoi w remoncie

Zasada biznesowa: Pokoi w remoncie nie można rezerwować

Test 4 **isAvailable_blockedDate_returnsFalse**

Co robi: Próbuje zarezerwować pokój w dniu zablokowanym (np. święta)

Co testuje: Blokadę terminów przez właściciela lub w dni świąteczne

Oczekiwane zachowanie: Zwraca false dla zablokowanych dat

Zasada biznesowa: Niektóre daty są zarezerwowane na użytek właściciela lub święta

Test 5 **isAvailable_overlappingBooking_returnsFalse**

Co robi: Symuluje istniejącą rezerwację i próbuje utworzyć pokrywającą się w czasie

Co testuje: Zapobieganie podwójnym rezerwacjom

Oczekiwane zachowanie: Zwraca false, gdy daty nachodzą na siebie

Kluczowa zasada biznesowa: Brak podwójnych rezerwacji

Test 6 **isAvailable_noOverlappingBookings_returnsTrue**

Co robi: Symuluje istniejącą rezerwację i próbuje zarezerwować inne, niepokrywające się daty

Co testuje: Wykrywanie dostępności pokoju

Oczekiwane zachowanie: Zwraca true, gdy nie ma konfliktów

Pozytywny test: Potwierdza, że wolne pokoje można rezerwować

Test 7 **isAvailable_maximumStayExceeded_returnsFalse**

Co robi: Próbuje zarezerwować pokój na 31 dni (limit to 30 dni)

Co testuje: Zasady dotyczące maksymalnej długości pobytu

Oczekiwane zachowanie: Zwraca false dla pobyków dłuższych niż 30 dni

Zasada biznesowa: Zapobiega ekstremalnie długim rezerwacjom

Test 8 `isAvailable_weekendSuiteOneNight_returnsFalse`

Co robi: Próbuje zarezerwować apartament na jedną noc od soboty

Co testuje: Zasady minimalnego pobytu w weekend dla pokoi premium

Oczekiwane zachowanie: Zwraca false dla jednodniowych rezerwacji apartamentów w weekendy

Zasada biznesowa: Apartamenty wymagają co najmniej 2 nocy w weekendy

Test 9 `isAvailable_weekendSuiteTwoNights_returnsTrue`

Co robi: Próbuje zarezerwować apartament na 2 noce od soboty

Co testuje: Spełnienie wymogu minimalnego pobytu

Oczekiwane zachowanie: Zwraca true, gdy zasada jest spełniona

Pozytywny test: Potwierdza poprawne działanie weekendowych rezerwacji

Test 10 `checkMultipleRooms_mixedAvailability_returnsCorrectMap`

Co robi: Sprawdza dostępność wielu pokoi jednocześnie (wolne, w remoncie, zajęte)

Co testuje: Funkcję masowej weryfikacji dostępności

Oczekiwane zachowanie: Zwraca mapę z poprawnymi statusami dla każdego pokoju

Funkcja efektywnościowa: Umożliwia sprawdzenie wielu pokoi w jednym wywołaniu

(5 testów) BookingServiceTest

Test 1 `createBooking_whenRoomAvailable_andPaymentSucceeds_returnsBooking`

Co robi: Przebiega cały proces rezerwacji z sukcesem

Co testuje: Kompletny przepływ rezerwacji od początku do końca

Dlaczego krytyczny: To najczęstszy scenariusz (90% przypadków)

Test 2 `createBooking_whenRoomNotAvailable_throws`

Co robi: Próbuje rezerwacji, gdy pokój nie jest dostępny

Co testuje: Integrację z systemem dostępności i obsługę błędów

Oczekiwane zachowanie: Wyrzuca wyjątek

Ochrona biznesowa: Zapobiega niemożliwym rezerwacjom

Test 3 `createBooking_whenPaymentFails_throwsIllegalStateException`

Co robi: Wszystko działa, dopóki płatność nie zawiedzie

Co testuje: Obsługę niepowodzenia płatności

Oczekiwane zachowanie: Wyrzuca wyjątek, brak zapisu rezerwacji

Ochrona finansowa: Brak rezerwacji bez płatności

Test 4 `createBooking_whenValidatorThrows_propagatesException`

Co robi: Walidator odrzuca żądanie rezerwacji

Co testuje: Integrację walidacji danych wejściowych

Oczekiwane zachowanie: Wyjątek zostaje przekazany wyżej

Ochrona wejścia: Błędne dane odrzucone na wstępie

Test 5 **cancelBooking_invokesRepositoryDelete**

Co robi: Anuluje istniejącą rezerwację

Co testuje: Proces anulowania

Oczekiwane zachowanie: Wywołanie metody usuwającej

Uwaga: W realnym systemie byłby tu jeszcze zwrot pieniędzy

BookingValidatorTest (3 testy)

Test 1: **validate_whenFromAfterTo_throws**

Co robi: Data zameldowania jest po dacie wymeldowania

Co testuje: Walidację kolejności dat

Oczekiwane zachowanie: IllegalArgumentException z komunikatem „Invalid dates: from after to”

Przykład: Zameldowanie 10 stycznia, wymeldowanie 5 stycznia

Test 2: **validate_whenZeroNights_throws**

Co robi: Ta sama data zameldowania i wymeldowania

Co testuje: Walidację minimalnej długości pobytu

Oczekiwane zachowanie: IllegalArgumentException z komunikatem „Stay must be at least 1 night”

Reguła biznesowa: Pobyt musi trwać co najmniej jedną noc

Test 3: **validate_whenValid_doesNotThrow**

Co robi: Poprawne żądanie rezerwacji (zameldowanie przed wymeldowaniem)

Co testuje: Pozytywny przypadek walidacji

Oczekiwane zachowanie: Brak wyjątku

Zapewnia: Poprawne żądania przechodzą dalej

DiscountServiceTest (11 testów)

Test 1: **applyDiscount_regularUser_noDiscount**

Co robi: Zwykły użytkownik bez specjalnego statusu

Co testuje: Domyślne zachowanie (brak zniżki)

Wejście: Cena bazowa 100 USD

Oczekiwane: 100 USD (bez zmian)

Test bazowy: Zapewnia, że system nie daje zniżek bez powodu

Test 2: `applyDiscount_zeroPrice_returnsZero`

Co robi: Rezerwacja za 0 USD

Co testuje: Obsługę wartości brzegowej

Oczekiwane: 0.0

Ochrona: Obsługuje nietypowe dane poprawnie

Test 3: `applyDiscount_negativePrice_returnsZero`

Co robi: Wprowadza ujemną cenę

Co testuje: Ochronę przed nieprawidłowymi danymi

Oczekiwane: 0.0

Ochrona: Zapobiega błędom związanym z ujemnymi cenami

Test 4: `applyDiscount_vipCustomer_gets10PercentOff`

Co robi: Użytkownik VIP („vip-user-1”) rezerwuje pokój

Co testuje: Logikę zniżki VIP

Wejście: Cena bazowa 100 USD

Oczekiwane: 90 USD (10% taniej)

Reguła biznesowa: Klienci VIP mają automatyczną zniżkę

Test 5: `applyDiscount_firstTimeCustomer_gets5PercentOff`

Co robi: Użytkownik z prefiksem „new-” w ID

Co testuje: Zniżkę dla pierwszego klienta

Wejście: Cena bazowa 100 USD

Oczekiwane: 95 USD (5% taniej)

Strategia marketingowa: Przyciąganie nowych klientów

Test 6: `applyDiscount_validPromoCode_appliesCorrectDiscount`

Co robi: Testuje różne poprawne kody promocyjne

Co testuje: Funkcję kodów promocyjnych

Przypadki testowe:

- „WELCOME10” → 10% taniej → 90 USD
- „SAVE20” → 20% taniej → 80 USD
- „SUMMER25” → 25% taniej → 75 USD

Narzędzie marketingowe: Kampanie promocyjne

Test 7: `applyDiscount_invalidPromoCode_noDiscount`

Co robi: Używa nieistniejącego kodu promocyjnego

Co testuje: Obsługę niepoprawnych kodów

Oczekiwane: Brak zniżki

Ochrona: Zapobiega nieautoryzowanym zniżkom

Test 8: `applyDiscount_vipCustomerWithPromo_stacksDiscounts`

Co robi: Klient VIP używa kodu promocyjnego

Co testuje: Logikę łączenia zniżek

Obliczenie: VIP (10%) + WELCOME10 (10%) = $100 \times 0.9 \times 0.9 = 81$ USD

Funkcja zaawansowana: Możliwość łączenia wielu rabatów

Test 9: `applyDiscount_longStay_gets15PercentOff`

Co robi: Pobyt 7+ nocy (1–8 czerwca)

Co testuje: Zniżkę za długi pobyt + weekendową

Obliczenie: Długi pobyt (15%) + weekend (8%) = $100 \times 0.85 \times 0.92 = 78,2$ USD

Cel: Zachęcanie do dłuższych pobyków

Test 10: `applyDiscount_multipleDiscounts_cappedAt60Percent`

Co robi: Próbuje nałożyć wiele rabatów przekraczających 60%

Co testuje: Ograniczenie maksymalnej zniżki

Scenariusz: VIP + nowy klient + promo 50% + długi pobyt + weekend

Oczekiwane: Maksymalnie 60% rabatu (cena końcowa 40 USD)

Ochrona biznesowa: Zapobiega nadmiernym zniżkom

Test 11: `applyDiscount_lowPrice_hasMinimumFloor`

Co robi: Nakłada 90% zniżki na rezerwację za 15 USD

Co testuje: Minimalny próg ceny

Oczekiwane: Minimum 10 USD (nie 1,50 USD)

Ochrona biznesowa: Utrzymanie minimalnych przychodów

PricingServiceTest (11 testów)**Test 1: `calculatePrice_nullParameters_returnsZero`**

Co robi: Przekazuje null dla roomId, daty od lub do

Co testuje: Obsługę wartości null

Oczekiwane: Zwraca 0.0 dla dowolnego parametru null

Ochrona: Zapobiega awariom przez złe dane wejściowe

Test 2: `calculatePrice_invalidDateRange_returnsZero`

Co robi: Niepoprawne zakresy dat (od > do, od = do)

Co testuje: Walidację dat w kalkulacji cen

Oczekiwane: Zwraca 0.0 dla niepoprawnych zakresów

Reguła biznesowa: Nie można wycenić niemożliwych pobyków

Test 3: `calculatePrice_usesRoomRepositoryBasePrice`

Co robi: Symuluje pokój z ceną bazową 250 USD, liczy cenę za 1 noc wiosną (poniedziałek)

Co testuje: Integrację z repozytorium pokoi

Oczekiwane: Używa faktycznej ceny bazowej pokoju (250 USD), a nie domyślnej (100 USD)

Integracja danych: Kalkulacja korzysta z rzeczywistych danych pokoju

Test 4: `calculatePrice_fridayNight_hasWeekendPremium`

Co robi: Rezerwuje piątkową noc latem

Co testuje: Dopłatę weekendową

Obliczenie: Cena bazowa × weekend (1.3) × lato (1.4)

Strategia biznesowa: Wyższe ceny w weekend

Test 5: `calculatePrice_winterDiscount`

Co robi: Rezerwacja wtorku w styczniu (zima)

Co testuje: Ceny sezonowe i zależne od popytu

Obliczenie: $100 \times \text{zima} (0.8) \times \text{wtorek} (0.9) = 72 \text{ USD}$

Strategia sezonowa: Niższe ceny zimą, by przyciągnąć klientów

Test 6: `calculatePrice_summerPremium`

Co robi: Rezerwacja poniedziałku w lipcu

Co testuje: Ceny w sezonie letnim

Oczekiwane: 140 USD (100×1.4)

Sezon szczytowy: Wyższe ceny w lecie

Test 7: `calculatePrice_holidayPremium`

Co robi: Rezerwacja w Boże Narodzenie (czwartek)

Co testuje: Dopłatę świąteczną

Obliczenie: Cena bazowa × zima × święto (1.5)

Oczekiwane: Wyższa cena w święta

Wydarzenia specjalne: Premium w okresie świątecznym

Test 8: `calculatePrice_weeklyStay_gets5PercentOff`

Co robi: Rezerwacja na 7 nocy wiosną

Co testuje: Rabat tygodniowy

Oczekiwane: 5% zniżki od całej kwoty

Zachęta: Dłuższe pobyty

Test 9: `calculatePrice_monthlyStay_gets20PercentOff`

Co robi: Rezerwacja na 28 nocy

Co testuje: Rabat miesięczny

Oczekiwane: 20% zniżki od całej kwoty

Strategia długoterminowa: Przyciąganie długich pobyków

Test 10: `calculatePrice_earlyBooking_gets5PercentOff`

Co robi: Rezerwacja 90+ dni wcześniej

Co testuje: Rabat za wcześniejszą rezerwację

Oczekiwane: 5% zniżki za rezerwację z wyprzedzeniem

Płynność finansowa: Nagroda za wcześniejsze zobowiązanie

Test 11: `calculatePrice_multipleNights_sumsCorrectly`

Co robi: Rezerwacja na 3 noce (pon-śr) wiosną

Co testuje: Poprawność sumowania wielu nocy

Oczekiwane: 280 USD łącznie (dokładna weryfikacja obliczeń)

Dokładność: Gwarancja poprawnych kalkulacji

ReportServiceTest (8 testów)

Test 1: `monthlyReport_validInput_returnsReport`

Co robi: Generuje raport dla czerwca 2025

Co testuje: Standardowe generowanie raportu

Weryfikuje: Raport zawiera nagłówki i sekcje

Oczekiwane: „ROOMIFY MONTHLY REPORT”, „June 2025”, „REVENUE SUMMARY”

Funkcja kluczowa: Podstawowe raporty działają

Test 2: `monthlyReport_invalidMonth_returnsError`

Co robi: Próbuje wygenerować raport dla miesiąca 13

Co testuje: Walidację miesiąca

Oczekiwane: Komunikat o błędzie „niepoprawny miesiąc”

Ochrona: Obsługuje błędne dane

Test 3: `monthlyReport_invalidYear_returnsError`

Co robi: Próbuje wygenerować raport dla roku 1999

Co testuje: Walidację roku

Oczekiwane: Komunikat o błędzie „niepoprawny rok”

Reguła biznesowa: Akceptowane lata tylko 2000–3000

Test 4: `getMonthlyMetrics_returnsCorrectMetrics`

Co robi: Pobiera metryki dla czerwca 2025

Co testuje: Obliczanie wskaźników

Weryfikuje obecność:

- totalRevenue
- totalBookings
- averageBookingValue
- occupancyRate
- uniqueCustomers

Funkcja API: Dane dla dashboardów

Test 5: roomPerformanceReport_returnsReport

Co robi: Generuje raport wydajności dla konkretnego pokoju

Co testuje: Analizę na poziomie pokoju

Oczekiwane: „Room Performance Report: room-1”, „June 2025”

Narzędzie zarządcze: Monitorowanie wydajności pokoi

Test 6: yearlyReport_returnsReport

Co robi: Generuje raport roczny dla 2025

Co testuje: Raporty roczne

Oczekiwane: „ROOMIFY YEARLY REPORT – 2025”, „MONTHLY BREAKDOWN”

Narzędzie strategiczne: Analiza biznesu rocznego

Test 7: occupancyReport_returnsReport

Co robi: Generuje raport obłożenia dla czerwca 2025

Co testuje: Obliczanie obłożenia i raportowanie

Oczekiwane: „OCCUPANCY REPORT”, „June 2025”, „Overall Occupancy Rate”

Metryka wydajnościowa: Śledzenie wykorzystania pokoi

Test 8: monthlyReport_edgeCases_handledCorrectly

Co robi: Testuje luty 2024 (rok przestępny), grudzień, styczeń

Co testuje: Obsługę nietypowych miesięcy

Weryfikuje: Poprawne nazwy miesięcy w raportach

Odporność: Obsługuje wszystkie warianty kalendarzowe

RoomServiceTest (2 testy)

Test 1: getRoom_whenExists_returnsRoom

Co robi: Symuluje istniejący pokój, pobiera go po ID

Co testuje: Poprawne pobieranie pokoju

Oczekiwane: Zwraca właściwy obiekt pokoju

Test pozytywny: Normalne działanie

Test 2: getRoom_whenMissing_throws

Co robi: Symuluje brakujący pokój, próbuje go pobrać

Co testuje: Obsługę błędu przy braku pokoju

Oczekiwane: IllegalArgumentException

Obsługa błędu: Łagodne zakończenie dla złych zapytań

UserServiceTest (3 testy)

Test 1: **userService_instantiatesSuccessfully**

Co robi: Tworzy instancję UserService

Co testuje: Konstruktor

Oczekiwane: Brak wyjątków przy tworzeniu

Test podstawowy: Usługa może być utworzona

Test 2: **userService_storesRepository**

Co robi: Tworzy serwis i sprawdza, czy nie jest null

Co testuje: Poprawne utworzenie obiektu

Oczekiwane: Obiekt serwisu nie jest null

Kontrola poprawności: Instancja poprawnie działa

Test 3: **constructor_withNullRepository_doesNotThrow**

Co robi: Tworzy serwis z repozytorium null

Co testuje: Obsługę parametru null w konstruktorze

Oczekiwane: Brak wyjątku

Defensywne programowanie: Obsługa wartości brzegowych

InvoiceServiceTest (1 test)

Test 1: **generateInvoiceId_notNull_andUnique**

Co robi: Generuje dwa ID faktur i porównuje je

Co testuje: Generowanie UUID

Weryfikacje:

- ID nie są null
- ID są unikalne (różne)

Krytyczne dla: Prowadzenia dokumentacji finansowej

ExternalCalendarAdapterTest (1 test)

Test 1: **pushBooking_noException**

Co robi: Wywołuje metodę pushBooking z ID rezerwacji

Co testuje: Brak awarii metody

Oczekiwane: Brak wyjątku

Punkt integracji: Interfejs do systemu zewnętrznego

CancellationPolicyServiceTest (4 testy)

Test 1: refundAmount_returnsZeroForNow

Co robi: Wywołuje kalkulację zwrotu dla rezerwacji

Co testuje: Podstawową kalkulację zwrotu

Oczekiwane: 0.0 (aktualna implementacja)

Placeholder: Gotowe na przyszłe reguły biznesowe

Test 2: refundAmount_handlesNullBookingId

Co robi: Przekazuje null jako ID rezerwacji

Co testuje: Obsługę wartości null

Oczekiwane: 0.0 (łagodna obsługa)

Defensywne programowanie: Obsługa błędnych danych

Test 3: refundAmount_handlesEmptyBookingId

Co robi: Przekazuje pusty string jako ID rezerwacji

Co testuje: Obsługę pustego wejścia

Oczekiwane: 0.0

Przypadek brzegowy: Różne błędne dane

Test 4: refundAmount_multipleCallsConsistent

Co robi: Wywołuje kalkulację zwrotu dwa razy dla tego samego wejścia

Co testuje: Spójność/deterministyczność metody

Oczekiwane: Identyczny wynik przy każdym wywołaniu

Niezawodność: Przewidywalne zachowanie

Analiza wyników testów:

Łączna liczba klas testowych: 11

Łączna liczba metod testowych: 59

Wskaźnik sukcesu: 100% (59/59 ZALICZONYCH)

Nieudane testy: 0

Pominięte testy: 0




Czas wykonania: ~2–3 sekundy

Pokrycie kodu: 80%+

Jaccoco:














roomify

roomify

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.roomify.core.service		92%		73%	63	210	54	542	13	97	0	11
com.roomify.core.dto		52%	n/a		17	36	28	55	17	36	1	5
com.roomify.controller		0%	n/a		4	4	4	4	4	4	2	2
com.roomify		0%	n/a		2	2	3	3	2	2	1	1
com.roomify.core.exception		0%	n/a		1	1	1	1	1	1	1	1
com.roomify.config		0%	n/a		1	1	1	1	1	1	1	1
Total	349 of 3,225	89%	59 of 226	73%	88	254	91	606	38	141	6	21

Created with JaCoCo 0.8

com.roomify.core.service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
AvailabilityService		75%		62%	26	56	26	85	7	21	0	1
PricingService		81%		79%	17	54	20	99	6	22	0	1
ReportService		97%		73%	14	53	7	253	0	27	0	1
DiscountService		98%		81%	6	26	1	55	0	10	0	1
BookingService		100%		100%	0	7	0	34	0	5	0	1
BookingValidator		100%		100%	0	4	0	5	0	2	0	1
RoomService		100%	n/a		0	3	0	4	0	3	0	1
InvoiceService		100%	n/a		0	2	0	2	0	2	0	1
UserService		100%	n/a		0	1	0	1	0	1	0	1
CancellationPolicyService		100%	n/a		0	2	0	2	0	2	0	1
ExternalCalendarAdapter		100%	n/a		0	2	0	2	0	2	0	1
Total	228 of 3,011	92%	59 of 226	73%	63	210	54	542	13	97	0	11

Created with JaCoCo 0.8

Rozkład pokrycia

- Krytyczna logika biznesowa: 95%+ pokrycia
- Usługi wspierające: 85%+ pokrycia
- Klasy narzędziowe: 100% pokrycia
- Cały projekt: 80%+