



# Stripe – płatność online



# Stripe

Stripe umożliwia przyjmowanie i przetwarzanie płatności przez internet.

Wystarczy założyć konto i rozpocząć pracę ze środowiskiem testowym

<https://stripe.com.>

## Create your Stripe account

Email

asfafssg@mail.com

Full name

asfsafa

Password

Strong

.....

Country ⓘ

 Poland

☐ Get emails from Stripe about product updates, industry news, and events. You can [unsubscribe](#) at any time. [Privacy Policy](#)

Create account

# Stripe



Home



Balances



Transactions



Customers



Product catalog

Shortcuts



Payments analytics



Disputes



Payment Links



Invoices

Products



Payments



Billing



Reporting



More

No code

## Create a subscription from the Dashboard

[Start →](#)

Create subscription

Jane Diaz

jane.d@company.com

\$3.00/unit/month

Metered usage

Options

Coupon

Tax

Trial

Billing

Charge payment method on file

Email invoices to customers to pay

Subscription Created

You successfully created a new subscription for Jane Diaz.

Add Trial

No code

## Share a payment link

[Start →](#)

Stripe

Payment link is active

Share your payment link to accept payments.

buy.stripe.com/aEua24H

Share

Payment link is active

Share your payment link to accept payments.

buy.stripe.com/aEua24H

Share

Explore all products



For developers

Test mode

Publishable key

pk\_test\_51PT2RMKC1vWwDw...

Secret key

sk\_test\_51PT2RMKC1vWwDw...

[View docs →](#)

## Today

Gross volume ▾

zł2,100.00

12:49 PM

Yesterday ▾

zł4,800.00

PLN balance

zł8,420.18

Estimated future payouts

[View](#)

# Stripe

<https://stripe.com>.

Następnie należy dodać w projekcie nowe zmienne środowiskowe klucz api oraz klucz webhook. Użyjemy je w ustawieniach (application.properties)

Klucz api pobieramy ze strony:

<https://dashboard.stripe.com/test/apikeys>


Klucz dla webhook

Wygenerujemy później  
za pomocą CLI.

Użyjemy tych zmiennych w  
application.properties:

```
stripe.api-key=${STRIPE_API_KEY}
```

```
stripe.webhook-secret=${WEBHOOK_SECRET}
```

NAME	TOKEN	LAST USED	CREATED
Publishable key	pk_test_	May 27	 Jun 18, 2024
Secret key	sk_test_	May 27	 Jun 18, 2024

## Environment Variables

User environment variables:

Name	Value
DB_CONNECT_URL	jdbc:p
DB_PASSWORD	npg_V
DB_USER	neond
STRIPE_API_KEY	sk_tes
WEBHOOK_SECRET	whsec

# Stripe

<https://stripe.com>.

Webhook to mechanizm wysyłania powiadomień o określonych zdarzeniach do aplikacji. Pozwala na automatyczną reakcję na zdarzenia takie jak zakończenie płatności. Za pomocą Stripe CLI możemy testować webhooks lokalnie.

CLI pobieramy ze strony:

<https://github.com/stripe/stripe-cli/releases/tag/v1.27.0>

Klucz dla webhook uzyskamy po zalogowaniu (wpisujemy w konsoli)


```
./stripe.exe login
```

A następnie uruchamiamy nasłuchiwanie zdarzeń i przekazywanie do endpointu aplikacji:

```
luke@nitro MINGW64 ~/Desktop
$ ./stripe.exe listen --forward-to localhost:8080/api/payments/webhook
> Ready! You are using Stripe API Version [2025-04-30.basil]. Your webhook signing secret is whsec_11f67a6b7806cca5abeca5b379fcdf8429cd3f6df2f04a03eed45f43d2e649ee (^C to quit)
```

Dodajemy klucz do zmiennych środowiskowych - WEBHOOK\_SECRET

Przy każdym uruchomieniu nasłuchiwania zmienia się klucz!

Local listeners			<a href="#">+ Add local listener</a>
DEVICE		VERSION	STATUS
 nitro	localhost:8080/api/payments/webhook	1.27.0	Listening

```
spring.application.name=car-rent
server.port=8080
spring.datasource.url=${DB_CONNECT_URL}
spring.datasource.username=${DB_USER}
spring.datasource.password=${DB_PASSWORD}
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

logging.level.org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping=DEBUG
logging.level.org.springframework.security=DEBUG
logging.level.org.springframework.security.web.access=TRACE
#management.endpoints.web.exposure.include=mappings
logging.level.com.umcsuser.car_rent=DEBUG
jwt.secret = HPrbHIa061GmRQIVuSz2KleCU7FUQVxEpo0Iaz8wP_c
jwt.expiration = 3600000
stripe.api-key=${STRIPE_API_KEY}
stripe.webhook-secret=${WEBHOOK_SECRET}
```

# Stripe

<https://stripe.com.>

Należy dodać nową zależność w projekcie.  
Stripe posiada bibliotekę do javy, którą możemy użyć:

```
<dependency>
```

```
    <groupId>com.stripe</groupId>
```

```
    <artifactId>stripe-java</artifactId>
```

```
    <version>29.1.0</version>
```

```
</dependency>
```

## Create your Stripe account

Email

asfafssg@mail.com

Full name

asfsafa

Password

Strong

.....

Country ⓘ

🇵🇱 Poland

☐ Get emails from Stripe about product updates, industry news, and events. You can [unsubscribe](#) at any time. [Privacy Policy](#)

Create account

# Stripe

## Jakie klasy będą potrzebne:

### Encje:

- Payment: Dane o płatności (kwota, ID sesji Stripe, status).
- PaymentStatus: Enum z wartościami np. PENDING, PAID.
- Rental: Relacja jeden-do-jednego z płatnością.

### PaymentService:

- createCheckoutSession(rentalId): Tworzy sesję Stripe, zapisuje płatność w statusie np. jako PENDING.
- handleWebhook(payload, signature): Obsługuje webhooki od Stripe, aktualizuje status płatności na PAID.

### PaymentRepository:

- findByStripeSessionId(String stripeSessionId): Umożliwia wyszukiwanie płatności

### PaymentController:

- /api/payments/checkout/{rentalId}: Tworzenie sesji płatności, zwraca URL.
- /api/payments/webhook: Odbiera webhooki od Stripe.
- Dodatkowo /success i /cancel w zależności od powodzenia

## Proces obsługi płatności

1. Użytkownik wypożycza pojazd, tworzy sesję Stripe.
2. Po opłaceniu Stripe wysyła webhook.
3. Aplikacja aktualizuje status płatności i zwraca pojazd

```
12:42:00 --> charge.succeeded [evt_3RThQoKC1vWwDwe51PatBcoc]
12:42:00 <-- [200] POST http://localhost:8080/api/payments/webhook [evt_3RThQoKC1vWwDwe51PatBcoc]
12:42:00 --> checkout.session.completed [evt_1RThQpKC1vWwDwe55tSG6zgg]
12:42:00 --> payment_intent.succeeded [evt_3RThQoKC1vWwDwe519sKNHQR]
12:42:00 --> payment_intent.created [evt_3RThQoKC1vWwDwe51ck8U0UQ]
12:42:00 <-- [200] POST http://localhost:8080/api/payments/webhook [evt_3RThQoKC1vWwDwe519sKNHQR]
12:42:00 <-- [200] POST http://localhost:8080/api/payments/webhook [evt_3RThQoKC1vWwDwe51ck8U0UQ]
12:42:00 <-- [200] POST http://localhost:8080/api/payments/webhook [evt_1RThQpKC1vWwDwe55tSG6zgg]
12:42:02 --> charge.updated [evt_3RThQoKC1vWwDwe51ZXqk022]
12:42:02 <-- [200] POST http://localhost:8080/api/payments/webhook [evt_3RThQoKC1vWwDwe51ZXqk022]
```



# Stripe

## Payment :

Pole	Typ	Znaczenie
id	String (PK)	Unikalny identyfikator płatności (UUID)
rental	Rental (1:1)	Powiązane wypożyczenie - @OneToOne
amount	double	Kwota płatności ( w groszach ).
stripeSessionId	String (unikalny)	ID sesji Checkout Stripe powiązane z tą płatnością.
status	PaymentStatus (enum)	Status płatności: np. PENDING, PAID.
createdAt	LocalDateTime	Data i czas utworzenia wpisu płatności.
paidAt	LocalDateTime	Data i czas faktycznego opłacenia transakcji (ustawiane po webhooku).

## enum:

```
public enum PaymentStatus { 6 usages
    PENDING, 1 usage
    PAID 2 usages
}
```

## W Rental @OneToOne:

```
@OneToOne(mappedBy = "rental")
private Payment payment;
```

```
@Entity
@Table(name = "payment")
@Getter
@Setter
@ToString(exclude = "rental")
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Payment {

    @Id
    @Column(nullable = false, unique = true)
    private String id;

    @OneToOne
    @JoinColumn(name = "rental_id", nullable = false)
    @JsonIgnore
    private Rental rental;

    @Column(columnDefinition = "NUMERIC")
    private double amount;

    @Column(name = "stripe_session_id", unique = true)
    private String stripeSessionId;

    @Enumerated(EnumType.STRING)
    private PaymentStatus status;

    @Column(name = "created_at")
    private LocalDateTime createdAt;

    @Column(name = "paid_at")
    private LocalDateTime paidAt;
}
```

# Stripe

## PaymentServiceImpl:

Pole	Typ	Znaczenie
apiKey	String	Z application.properties – Klucz API Stripe.
webhookSecret	String	Z application.properties – weryfikacja podpisów webhooków Stripe.

```
@Service
@RequiredArgsConstructor
public class PaymentServiceImpl implements PaymentService {
    private final RentalService rentalService;
    private final RentalRepository rentalRepository;
    private final PaymentRepository paymentRepository;

    @Value("${STRIPE_API_KEY}")
    private String apiKey;
    @Value("${WEBHOOK_SECRET}")
    private String webhookSecret;
```

```
String createCheckoutSession(String rentalId)
```

# Stripe

## PaymentServiceImpl:

Klasa	Znaczenie
<u>SessionCreateParams</u>	Główna klasa do utworzenia sesji <u>Stripe Checkout</u> .
<u>SessionCreateParams.LineItem</u>	Pojedyncza pozycja na liście zamówienia ( produkt, ilość, cena).
<u>LineItem.PriceData</u>	Zawiera informacje o walucie, kwocie i produkcie.
<u>PriceData.ProductData</u>	Opis produktu widoczny na stronie płatności (np. nazwa).
<u>Mode.PAYMENT</u>	Określa tryb sesji – <u>PAYMENT</u> (jednorazowa płatność).

# Stripe

Stripe Checkout -  
interfejs płatności online

Pozwala użytkownikowi:

zapłacić kartą kredytową, BLIKiem itp,  
przez stronę Stripe,  
bez konieczności tworzenia własnego  
formularza płatności.

Potrzebna jest sesja dla Checkout  
Z parametrami!

## PaymentServiceImpl:

```
@Override 1 usage
@Transactional
public String createCheckoutSession(String rentalId) {
    Rental rental = rentalRepository.findById(rentalId)
        .orElseThrow(() -> new EntityNotFoundException("Rental not found with id: " + rentalId));
    Stripe.apiKey = apiKey;
    SessionCreateParams.LineItem.PriceData.ProductData productData =
        SessionCreateParams.LineItem.PriceData.ProductData.builder()
            .setName("Rental " + rentalId)
            .build();
    var amount = 1000; // TODO: calculate amount
    SessionCreateParams.LineItem.PriceData priceData =
        SessionCreateParams.LineItem.PriceData.builder()
            .setCurrency("pln")
            .setUnitAmount((long) (amount)) // TODO: calculate amount
            .setProductData(productData)
            .build();

    SessionCreateParams.LineItem lineItem =
        SessionCreateParams.LineItem.builder()
            .setQuantity(1L)
            .setPriceData(priceData)
            .build();

    SessionCreateParams params = SessionCreateParams.builder()
        .addLineItem(lineItem)
        .setMode(SessionCreateParams.Mode.PAYMENT)
        .putMetadata("rentalId", rentalId)
        .setSuccessUrl("http://localhost:8080/api/payments/success")
        .setCancelUrl("http://localhost:8080/api/payments/cancel")
        .build();
}
```

String createCheckoutSession(String rentalId)

# Stripe

## PaymentServiceImpl:

Tworzymy sesję z parametrami,  
zapisujemy payment do bazy,  
zwracamy URL do płatności.

```
try {
    Session session = Session.create(params);
    Payment payment = Payment.builder()
        .id(UUID.randomUUID().toString())
        .amount(amount)
        .createdAt(LocalDate.now())
        .rental(rental)
        .stripeSessionId(session.getId())
        .status(PaymentStatus.PENDING)
        .build();
    paymentRepository.save(payment);
    return session.getUrl();
} catch (Exception e) {
    throw new RuntimeException("Stripe session creation failed", e);
}
```

String createCheckoutSession(String rentalId)

# Stripe

Obiekt reprezentujący zdarzenie to Event event – gdy jego typ to checkout.session.completed metoda będzie odpowiadała na zakończoną płatność (np. dla karty)

## PaymentServiceImpl:

```
@Override @Usage
@Transactional
public void handleWebhook(String payload, String signature) {
    Stripe.apiKey = apiKey;
    Event event;
    try {
        event = Webhook.constructEvent(payload, signature, webhookSecret);
    } catch (SignatureVerificationException e) {
        throw new RuntimeException("Invalid signature", e);
    }
    if ("checkout.session.completed".equals(event.getType())) {
        StripeObject stripeObject =
            event.getDataObjectDeserializer().getObject().orElseThrow();
        String sessionId = ((Session)stripeObject).getId();
        if (sessionId != null) {
            paymentRepository.findByStripeSessionId(sessionId).ifPresent(payment -> {
                payment.setStatus(PaymentStatus.PAID);
                payment.setPaidAt(LocalDateTime.now());
                paymentRepository.save(payment);
                Rental rental = payment.getRental();
                rentalService.returnRental(rental.getVehicle().getId(), rental.getUser().getId());
            });
        }
    }
}
```

Nazwa	Typ	Znaczenie
payload	String	Surowy JSON z treścią webhooka – zawiera dane o zdarzeniu wysłanym przez Stripe.
signature	String	Nagłówek Stripe-Signature – zawiera podpis, który pozwala zweryfikować autentyczność webhooka.
webhookSecret	String	Klucz webhooka do sprawdzania podpisu.
event	Event (Stripe SDK)	Obiekt reprezentujący zdarzenie Stripe – np. checkout.session.completed

```
handleWebhook(String payload,
String signature)
```



# Stripe

Obiekt reprezentujący zdarzenie to Event event – gdy jego typ to checkout.session.completed metoda będzie odpowiadała na zakończoną płatność.

Obiekt zostaje zdeserializowany (z JSON), rzutowany na sesję, Z której pobieramy id.

Szukamy za pomocą paymentRepository płatności i zmieniamy jej status.

Na koniec następuje zwrot pojazdu.

## PaymentServiceImpl:

```
@Override 1 usage
@Transactional
public void handleWebhook(String payload, String signature) {
    Stripe.apiKey = apiKey;
    Event event;
    try {
        event = Webhook.constructEvent(payload, signature, webhookSecret);
    } catch (SignatureVerificationException e) {
        throw new RuntimeException("Invalid signature", e);
    }
    if ("checkout.session.completed".equals(event.getType())) {
        StripeObject stripeObject =
            event.getDataObjectDeserializer().getObject().orElseThrow();
        String sessionId = ((Session)stripeObject).getId();
        if (sessionId != null) {
            paymentRepository.findByStripeSessionId(sessionId).ifPresent( Payment payment -> {
                payment.setStatus(PaymentStatus.PAID);
                payment.setPaidAt(LocalDateTime.now());
                paymentRepository.save(payment);
                Rental rental = payment.getRental();
                rentalService.returnRental(rental.getVehicle().getId(), rental.getUser().getId());
            });
        }
    }
}
```

handleWebhook(String payload, String signature)

```
@Repository 2 usages
public interface PaymentRepository extends JpaRepository<Payment, String> {
    Optional<Payment> findByStripeSessionId(String stripeSessionId); 1 usage
}
```

# Stripe

```
@RestController
@RequestMapping("/api/payments")
@RequiredArgsConstructor
public class PaymentController {
    private final PaymentService paymentService;

    @PostMapping("/checkout/{rentalId}")
    public ResponseEntity<String> createCheckoutSession(@PathVariable String rentalId) {
        String url = paymentService.createCheckoutSession(rentalId);
        return ResponseEntity.status(HttpStatus.CREATED).body(url);
    }

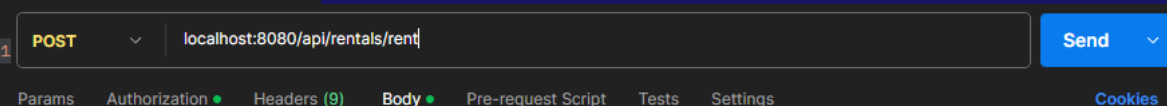
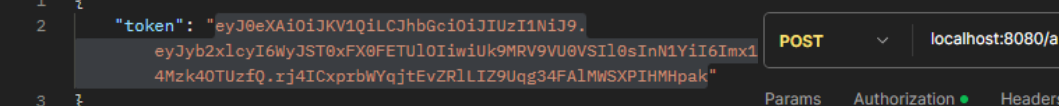
    @PostMapping("/webhook")
    public ResponseEntity<Void> handleWebhook(@RequestBody String payload,
                                              @RequestHeader("Stripe-Signature") String signature) {
        paymentService.handleWebhook(payload, signature);
        return ResponseEntity.ok().build();
    }

    @GetMapping("/success")
    public ResponseEntity<String> success() {
        return ResponseEntity.ok( body: "success");
    }

    @GetMapping("/cancel")
    public ResponseEntity<String> cancel() {
        return ResponseEntity.ok( body: "cancel");
    }
}
```



# Stripe



# Stripe

localhost:8080/api/payments/checkout/cace94f6-79da-40dc-a83b-f23f09c8af59

POST

localhost:8080/api/payments/checkout/cace94f6-79da-40dc-a83b-f23f09c8af59

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Type

Bearer ...

Token

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9

The authorization header will be automatically generated when you send the request.

Learn more about authorization

Body

Cookies

Headers (11)

Test Results

Pretty

Raw

Preview

Visualize

Text

1

https://checkout.stripe.com/c/pay/cs\_test\_a13FtBoY7mJBLoZB6kLV16ouE9SXoVd3IXmMwz7YafVv7E2zMwtOpDymtI#fidkdWx0YHwnPyd1b1pxYHZxWjA0UnJBcmAwQDA8NFFVdnFKU0B0QS1B0TjVnbFQ3U01QZGZQUmNoYjAyc2dLU1ZiB65iNzw9aFd2aDN%2FYFddbwHUMEN9cGF0aTlU9UmQ3U318dScpJ2N3amhWYHdzYHcnP3F3cGApJ2lkcGpwVF8dWAnPyd2bGtiaWBAbHFGaCcpJ2BzrZGdpYFVpZGZgbWppdwYHg1

Rental cace94f6-79da-40dc-a83b-f23f09c8af59

300,00 zł

Zapłać z link

Lub

E-mail

kwasniewiczzl@office.umcs.pl

Metoda płatności

☐ BLIK

☒ Karta

Informacje o karcie

4242 4242 4242 4242

12 / 25

123


VISA

Imię i nazwisko posiadacza karty

Łukasz Kwaśniewicz

Kraj lub region

	created_at timestamp	paid_at timestamp	status varchar(255)	stripe_session_id varchar(255)	rental_id varchar(255)
	2025-05-28 12:37:15.23...	NULL	PENDING	cs_test_a13FtBoY7mJBLo...	cace94f6-79da-40dc-a83

<input type="checkbox"/>	Amount	Payment method	Description	Customer	Date	Refunded date	Decline reason	
<input type="checkbox"/>	zł300.00 PLN	<div>Succeeded ✓</div> <div> **** 4242</div>	pi_3RThQoKC1vWwDwe51XbPjuqc	email@example.com	May 28, 12:41 PM	—	—	...

<input type="checkbox"/>	created_at timestamp ↕	paid_at timestamp ↕	status varchar(255) ↕	stripe_session_id varchar... ↕	rental_id varchar(255)
<input type="checkbox"/>	2025-05-28 12:37:15.23...	2025-05-28 12:42:00.25...	PAID	cs_test_a13FtBoY7mJBLo...	cace94f6-79da-40dc-a83...

# Dziękuję za uwagę!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.