

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



Integrating Stripe payments in Spring Boot



Umasree Kollu · [Follow](#)

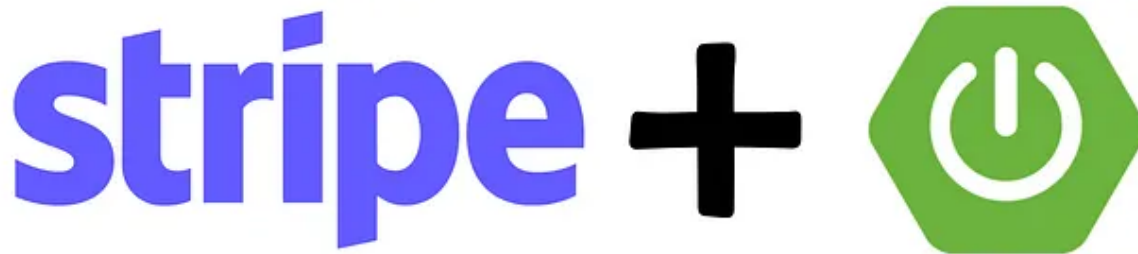
5 min read · Apr 8, 2024



6



1



Discover how to seamlessly integrate Stripe, a leading payment gateway, into your Spring Boot application. Simplify payment processing and enhance user experiences with Stripe's robust API and developer-friendly tools. Follow along as we guide you through the step-by-step process of setting up Stripe, implementing payment logic, and elevating your application's payment capabilities.

To integrate Stripe payment gateway in a Spring Boot application, you can follow these steps:

1. Setting Up Your Stripe Account and Adding Dependencies

- **Stripe Account Setup:**

- Go to the [Stripe website](#) and sign up for an account.
- Navigate to the dashboard and find your API keys under the Developers - > API keys section.
- **Adding Dependencies:**

For Maven, add the following dependency to your `pom.xml` file:

```
<dependency>  
  <groupId>com.stripe</groupId>  
  <artifactId>stripe-java</artifactId>  
  <version>21.1.0</version>  
</dependency>
```

- For Gradle, add this to your `build.gradle` file:

```
implementation 'com.stripe:stripe-java:21.1.0'
```

2. Adding Stripe Keys:

After setting up your Stripe account, obtain your API keys from the dashboard. You'll need both test and live keys. Store these keys securely in your Spring Boot application properties file (`application.properties` or `application.yml`).

```
stripe.apiKey=sk_test_yourSecretKey
```

3. Create Payment Entity:

Define a Payment entity class to represent payment details:

```
@Entity
@Table(name = "payment")
@Data
public class Payment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name="user_email")
    private String userEmail;

    @Column(name = "amount")
    private double amount;
}
```

In order to store payment-related information in your application's database, create a **Payment** entity. This entity represents the structure of your payment data. In this example, we've defined attributes such as **userEmail** and **amount** to capture details about the payer and the transaction amount.

4: Create Payment Repository

Create a repository interface to handle CRUD operations for the Payment entity:

```
public interface PaymentRepository extends JpaRepository<Payment, Long> {
    Payment findByUserEmail(String userEmail);
}
```

A repository interface is responsible for handling database operations related to the **Payment** entity. By extending **JpaRepository**, you inherit methods for common CRUD (Create, Read, Update, Delete) operations. In this case, we've added a custom method to find payment information by user email.

5: Create Payment Info Request

Define a DTO (Data Transfer Object) for capturing payment information:

```
@Data
public class PaymentInfoRequest {
    private int amount;
    private String currency;
    private String receiptEmail;
}
```

The **PaymentInfoRequest** class serves as a data transfer object (DTO) for capturing payment information from the client-side. It contains attributes such as **amount**, **currency**, and **receiptEmail**, which are essential for processing payments.

6: Create Payment Service

Implement a service layer to handle payment-related business logic:

```
@Service
@Transactional
public class PaymentService {

    private PaymentRepository paymentRepository;

    @Autowired
```

```

public PaymentService(PaymentRepository paymentRepository, @Value("${stripe.
    this.paymentRepository = paymentRepository;
    Stripe.apiKey = secretKey;
}

public PaymentIntent createPaymentIntent(PaymentInfoRequest paymentInfoReque
    List<String> paymentMethodTypes = new ArrayList<>();
    paymentMethodTypes.add("card");

    Map<String, Object> params = new HashMap<>();
    params.put("amount", paymentInfoRequest.getAmount());
    params.put("currency", paymentInfoRequest.getCurrency());
    params.put("payment_method_types", paymentMethodTypes);

    return PaymentIntent.create(params);
}

public ResponseEntity<String> stripePayment(String userEmail) throws Excepti
    Payment payment = paymentRepository.findByUserEmail(userEmail);

    if (payment == null) {
        throw new Exception("Payment information is missing");
    }
    payment.setAmount(00.00);
    paymentRepository.save(payment);
    return new ResponseEntity<>(HttpStatus.OK);
}

}

```

- **public PaymentService(...)**: Constructor for the **PaymentService** class. It takes **PaymentRepository** and **secretKey** as parameters and initializes them. The **secretKey** is retrieved from the **application.properties** file using **@Value("\${stripe.key.secret}")**.
- **Stripe.apiKey = secretKey;**: Sets the Stripe API key to authenticate requests. This is a one-time setup when the service is instantiated.
- **createPaymentIntent**: This method generates a payment intent using the Stripe API. It takes a **PaymentInfoRequest** object containing payment

details as a parameter.

- **List<String> paymentMethodTypes** : Defines the types of payment methods accepted, in this case, just "card".
- **Map<String, Object> params** : Constructs a map containing parameters required to create a payment intent, such as **amount** , **currency** , and **payment_method_types** .
- **PaymentIntent.create(params)** : Calls the Stripe API to create a payment intent with the provided parameters.
- **stripePayment** : This method completes a payment transaction by updating the payment amount in the database. It takes the user's email as a parameter.
- **Payment payment = paymentRepository.findByUserEmail(userEmail);** : Retrieves the payment information associated with the given user email.
- **if (payment == null) { throw new Exception("Payment information is missing"); }** : Checks if the payment information exists. If not, it throws an exception.
- **payment.setAmount(00.00);** : Sets the payment amount to 0.00. (Note: This seems to be a placeholder. In a real-world scenario, you would update the payment status or perform other actions here.)
- **paymentRepository.save(payment);** : Saves the updated payment information to the database.

This **PaymentService** class encapsulates the logic for creating payment intents and completing payment transactions, integrating with both the Stripe API and the application's database.

7: Create Payment Controller

Create a controller to handle payment-related requests:

```
@RestController
@RequestMapping("/api/payment/secure")
public class PaymentController {

    private PaymentService paymentService;

    @Autowired
    public PaymentController(PaymentService paymentService) {
        this.paymentService = paymentService;
    }

    @PostMapping("/payment-intent")
    public ResponseEntity<String> createPaymentIntent(@RequestBody PaymentInfoRe
        throws StripeException {

        PaymentIntent paymentIntent = paymentService.createPaymentIntent(payment
        String paymentStr = paymentIntent.toJson();

        return new ResponseEntity<>(paymentStr, HttpStatus.OK);
    }

    @PutMapping("/payment-complete")
    public ResponseEntity<String> stripePaymentComplete(@RequestHeader(value="Au
        throws Exception {
        String userEmail = ExtractJWT.payloadJWTExtraction(token, "\"sub\"");
        if (userEmail == null) {
            throw new Exception("User email is missing");
        }
        return paymentService.stripePayment(userEmail);
    }
}
```

- **@RequestBody PaymentInfoRequest paymentInfoRequest** : Binds the request body to a **PaymentInfoRequest** object.
- **throws StripeException** : Indicates that this method may throw a **StripeException** if there's an issue with the Stripe API.

- **paymentService.createPaymentIntent(paymentInfoRequest)** : Calls the **createPaymentIntent** method of the **paymentService** to generate a payment intent based on the provided payment information.
- **paymentIntent.toJson()** : Converts the payment intent object to JSON format.
- **@RequestHeader(value="Authorization") String token** : Retrieves the authorization token from the request header.
- **ExtractJWT.payloadJWTExtraction(token, "\"sub\"")** : Extracts the user email from the JWT token payload.
- **if (userEmail == null) { throw new Exception("User email is missing"); }** : Checks if the user email is null. If so, throws an exception.
- **paymentService.stripePayment(userEmail)** : Calls the **stripePayment** method of the **paymentService** to complete the payment transaction using the user's email.
- Returns the response entity returned by the **stripePayment** method.

the **PaymentController** class defines two endpoints for handling payment-related operations: one for creating a payment intent and another for completing a payment transaction. It delegates the actual business logic to the **PaymentService** class and returns appropriate responses based on the results.

8. Testing Payment Integration

To test the payment integration, you can send a POST request to the **/api/payment/secure/payment-intent** endpoint with the required parameters, including the Stripe token and the amount to be charged. Follow these steps to test the payment integration:

1. Send POST Request: Use a tool like cURL, Postman, or any HTTP client to send a POST request to the `/api/payment/secure/payment-intent` endpoint with the following JSON:

```
{  
  "amount": 1000,  
  "currency": "usd",  
  "receiptEmail": "example@example.com"  
}
```

2. Verify Response:

- If the payment is successful, you should receive a success message with the charge ID in the response body.
- If the payment fails, you will receive an error message with the reason for the failure.

3. Check Stripe Dashboard:

Additionally, you can check the Stripe Dashboard to verify the payment transaction details, including the charge ID, amount, and status.

Conclusion:

integrating Stripe payments into a Spring Boot application offers a seamless and secure way to process transactions. With clear endpoints for initiating and completing payments, developers can easily implement and test payment functionality. By following best practices and handling error cases effectively, users can enjoy a smooth payment experience.

Thank you for reading this blog post! I hope you found the information helpful and insightful. If you have any questions or feedback, please feel free to reach out. Happy coding!

Stripe

Stripe Integration

Spring Boot

**Written by Umasree Kollu**

19 Followers

Follow

**More from Umasree Kollu**

Umasree Kollu

Building Payment Functionality with Spring Boot: Step-by-Step Guide

Welcome to ' Building Payment Functionality with Spring Boot: Step-by-Step Guide '! In this tutorial, we'll navigate through the...




Apr 3



[See all from Umasree Kollu](#)

Recommended from Medium




 Data Backend Tech

Spring Boot Java Framework: Implementing Rate Limiting Usin...

Introduction

★ Oct 3 🖱 1



 Master Musili

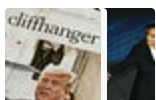
Spring Microservices Implementing API Gateway and...

Spring Cloud Gateway aims to provide a simple, yet effective way to route to APIs and...

Jun 12 🖱 6

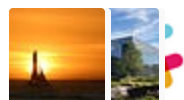


Lists



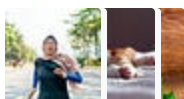
Staff Picks

756 stories · 1420 saves



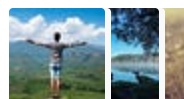
Stories to Help You Level-Up at Work

19 stories · 854 saves



Self-Improvement 101

20 stories · 2969 saves



Productivity 101

20 stories · 2516 saves



Serxan Hamzayev in JavaToDev

Spring Data JPA: Optimizing Batch Inserts and Updates

When handling large datasets in Spring Data JPA, executing inserts and updates in bulk...



6d ago



36

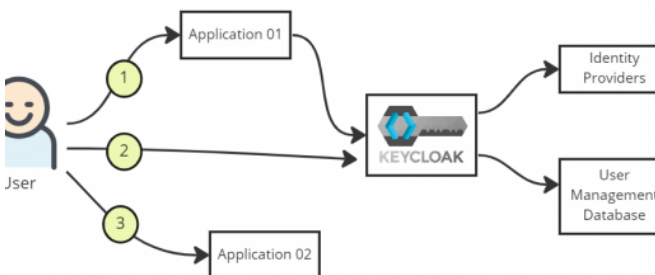


Plexify GmbH

Integrating multiple OAuth flows into your SpringBoot project

This article demonstrates how to integrate multiple OAuth-Clients into a Spring-Boot...

Jun 30



Vinotech

How Keycloak works and Execution Flow using Spring Boot

Keycloak is an open-source identity and access management solution designed to...



5d ago



1



Gopesh Jangid

Connecting a React Native App to AWS : Using Amplify Gen2 (Part 2)

Welcome to Part 2 of our series on integrating AWS AppSync and GraphQL into a React...



Oct 11



1



See more recommendations