Payment Processing

Payment Processing

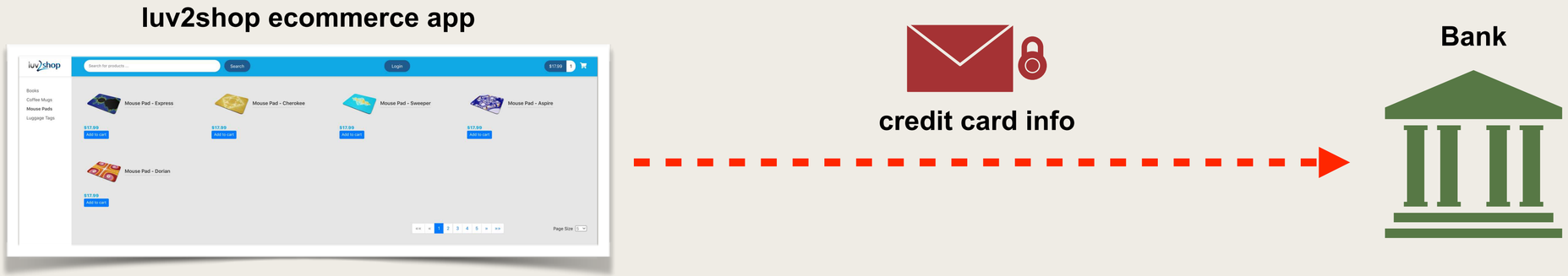
- We have the basic functionality of an ecommerce site

- But what about payment processing???

© luv2code LLC

Payment Processing Requirement

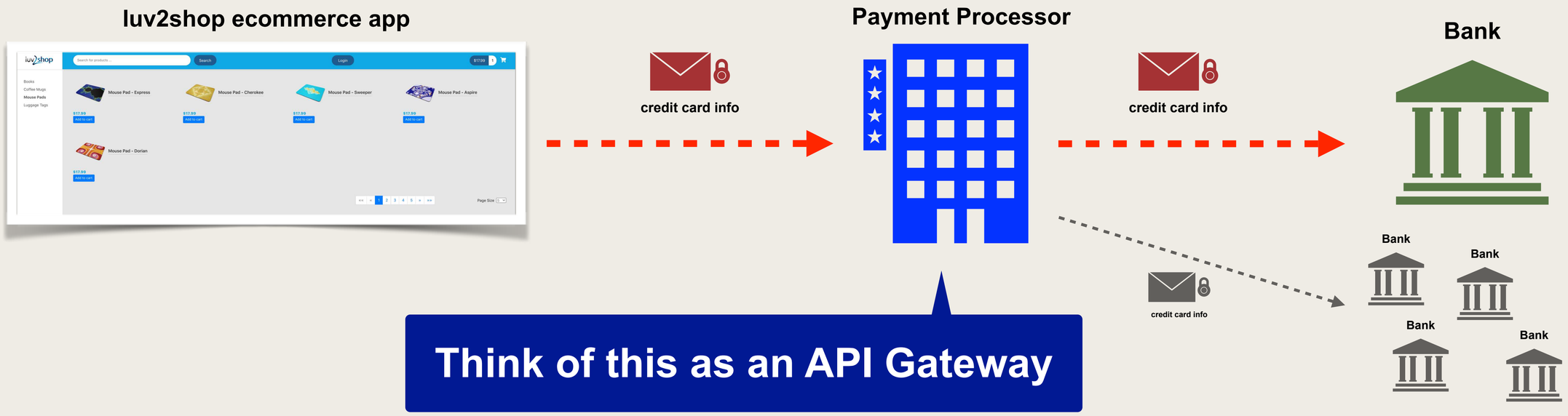
- Accept customer credit card info
- Perform a one-time charge for the shopping cart total



© luv2code LLC

Payment Processors

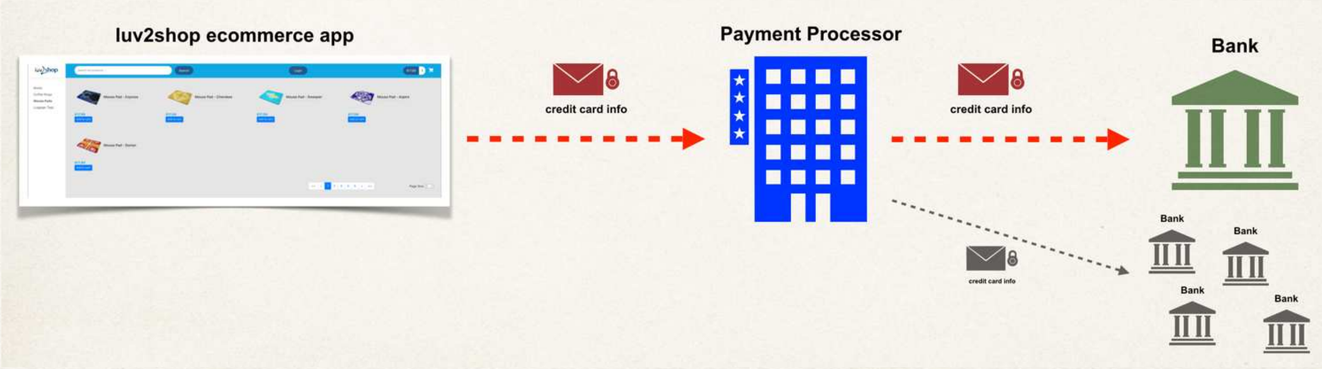
- To process credit cards, need to use an intermediary
- Payment Processor is a company that handles the credit card transaction



© luv2code LLC

Payment Processor Responsibilities

- Relays credit card information to the bank
- Performs fraud detection on the credit card
- If bank authorizes transaction, then process funds
- If bank denies transaction, return error message / code



© luv2code LLC

Payment Processors

- Large number of payment processors in the market
- Stripe
- Paypal
- etc ...

We will use Stripe for our app
Modern API
Supported in a large number of countries

© luv2code LLC

What about Payment Processor XYZ??

- If you need to use a specific payment processor for your project
- You can use the concepts presented in this course
- Refer to the API documentation for your specific payment processor
- All payment processors have API documentation and tutorials

© luv2code LLC

Stripe Integration Overview

Stripe API

- Stripe provides an API for processing credit card payments
- REST API is available
- Libraries also available for JavaScript, Java, .NET, PHP, Go etc ...

https://stripe.com/docs

© luv2code LLC

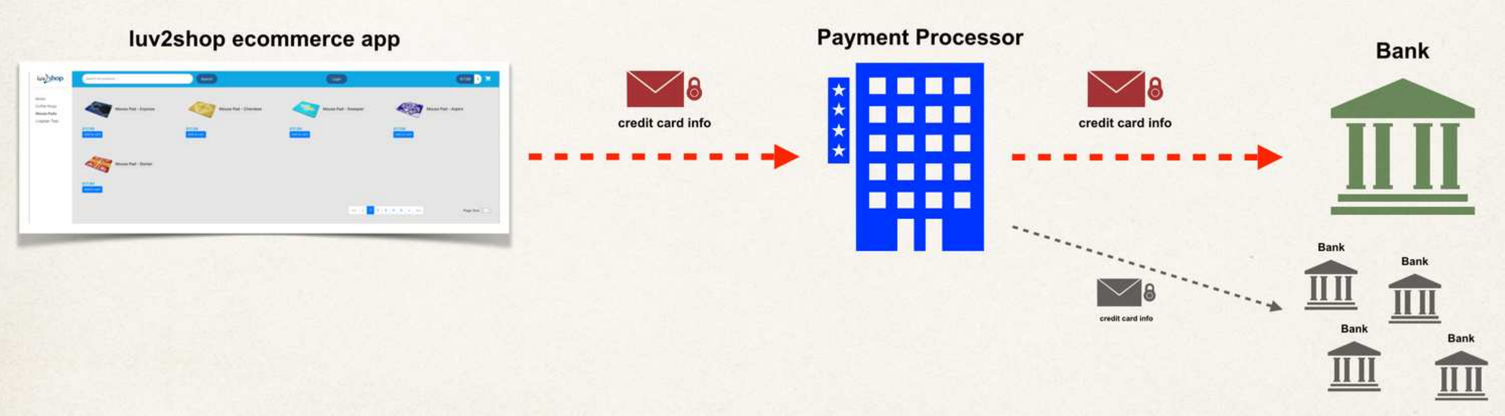
Stripe API Features

- Online payment processing
- Invoicing: creating, collecting and reconciling
- Subscriptions: recurring transactions
- Quotes: give price estimates for invoices and subscriptions

© luv2code LLC

Payment Processing Requirement

- Accept customer credit card info
- Perform a one-time charge for the shopping cart total



© luv2code LLC

Sensitive Data

- When dealing with credit card data, you must protect cardholder data
- Your DevOps process must be **PCI** compliant
- Payment Card Industry Data Security Standard (PCI DSS)
  - Ensure all companies accept, process, store and transmit data in a secure environment

© luv2code LLC

PCI Compliance

- Use and maintain firewalls
- Proper password protection
- Protect card holder data
- Encrypt transmitted data
- Use and maintain anti-virus
- Properly updated software

- Restrict data access
- Unique ids for access
- Restrict physical access
- Create and maintain access logs
- Scan and test for vulnerabilities
- Document policies

© luv2code LLC

How Stripe Helps with PCI

- Stripe Payment integration greatly simplifies PCI compliance
  - Stripe provides hosted forms for users to enter credit card data
  - Credit Card data is sent directly to Stripe for processing
  - Credit Card data is never sent to your own servers

- As a developer
  - Never send credit data to your own server
  - Never store credit card data on your own server: in memory or DB

© luv2code LLC

Additional Resources

https://stripe.com/docs/security

https://stripe.com/guides/pci-compliance

© luv2code LLC

Stripe Integration Options

Stripe Integration Options

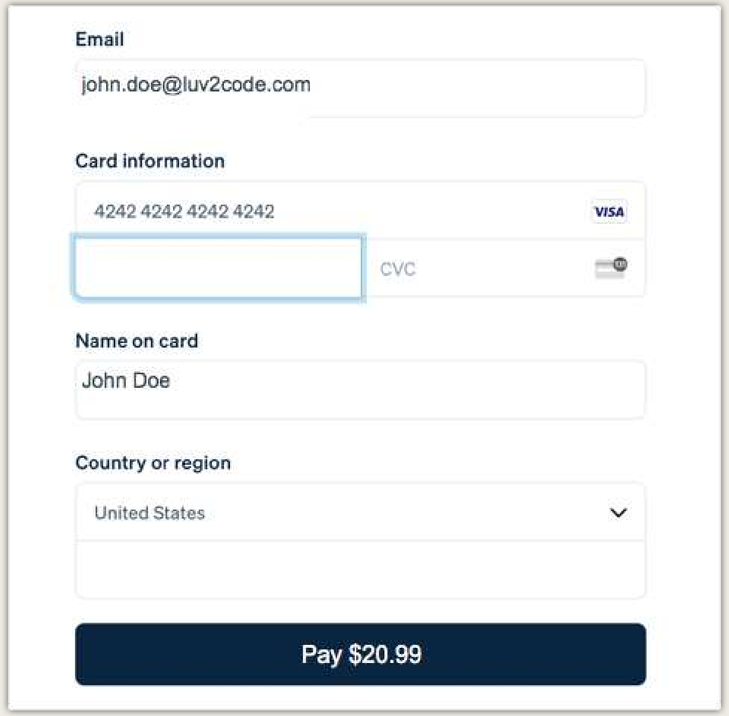
- Low-code integration: Checkout pop-up window

- Custom code integration: Use custom/existing checkout form

© luv2code LLC

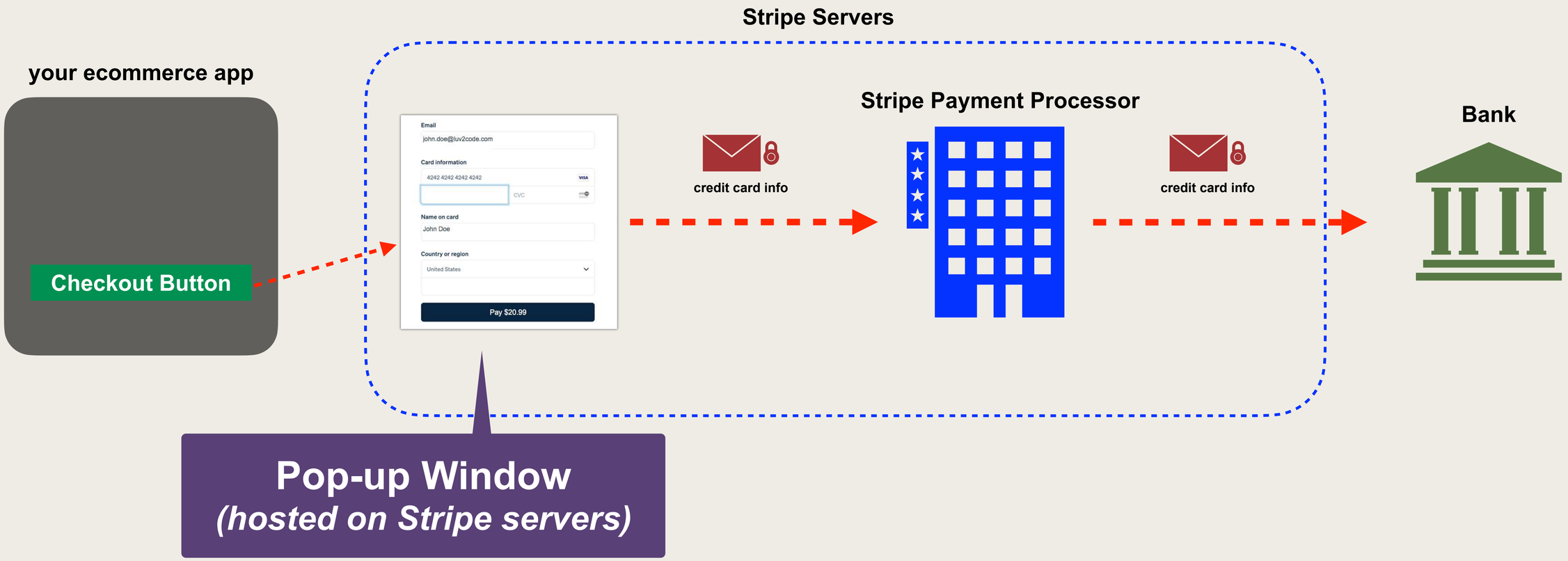
Stripe Integration: Checkout Form

- Stripe provides a pre-built checkout form
- Separate pop-up window
- Checkout form is hosted on Stripe's website
- Form includes validation, responsive design
- Supports customization with CSS and images
- PCI compliance, tax collection etc ...



© luv2code LLC

Stripe Integration: Checkout Form



© luv2code LLC

Stripe Integration: Checkout Form

Advantages

- Low-code integration
- Hosted form on Stripe Servers
- Customization and branding
- PCI compliant

Disadvantages

- Separate pop-up window
- Does not provide seamless user experience

https://stripe.com/payments/checkout

© luv2code LLC

Stripe Integration: Custom Code Integration

- Ideally, you want to use your existing/custom checkout form
- No desire to have separate popup windows

© luv2code LLC

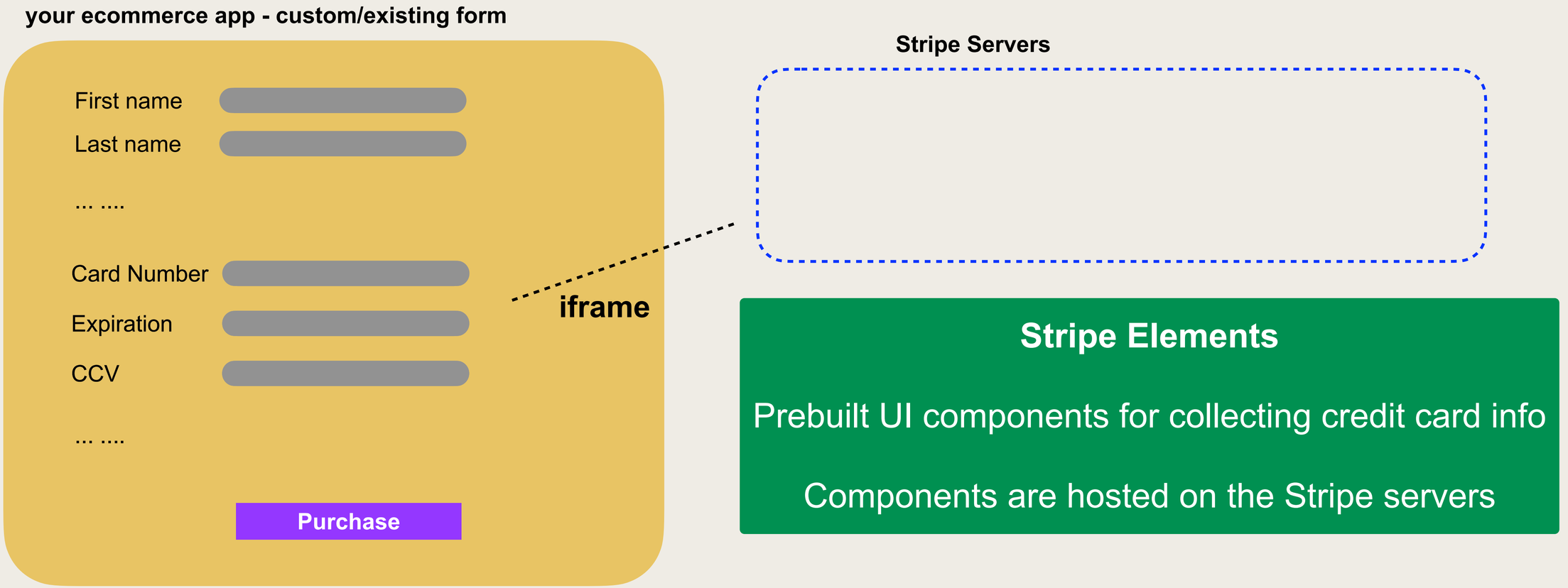
Stripe Integration: Custom Code Integration

- Stripe provides **Stripe Elements**
  - Prebuilt UI components for collecting credit card info
  - Components are hosted on the Stripe servers
  - Can easily integrate into your existing/custom checkout form
  - Provide seamless integration
  - Supports PCI ... the credit card data is never sent to your servers

© luv2code LLC

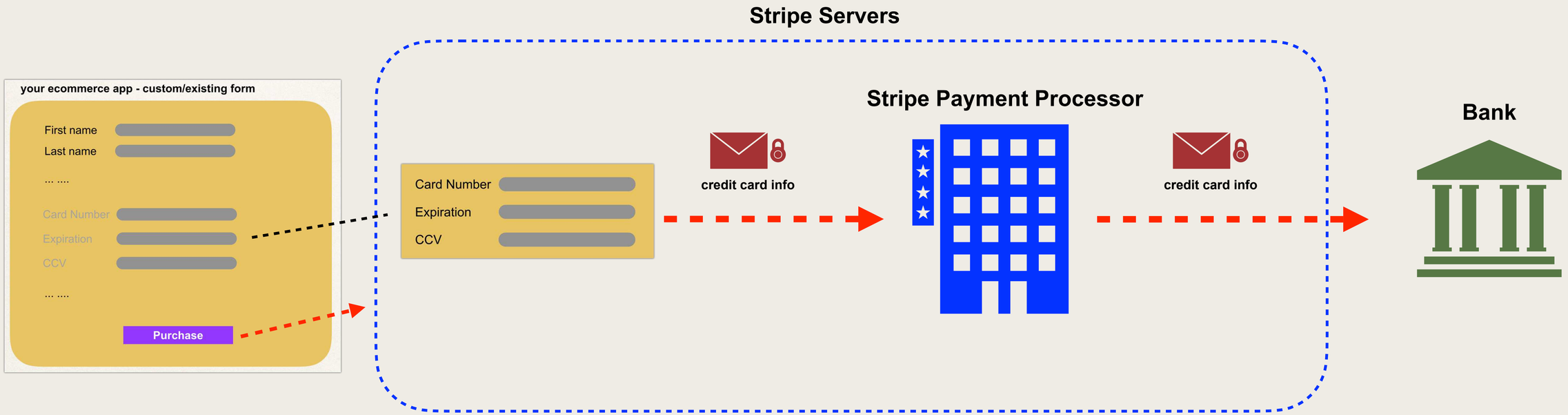
Stripe Integration: Stripe Elements

Provides seamless integration



© luv2code LLC

Stripe Integration: Stripe Elements



© luv2code LLC

Stripe Integration: Stripe Elements

Advantages

- Seamless checkout integration
- Hosted UI elements on Stripe Servers
- Use our existing / custom form
- Customization and branding
- PCI compliant

Disadvantages

- Requires an existing / custom form
- Requires custom coding on frontend and backend

https://stripe.com/payments/elements

© luv2code LLC

Our Integration

- For our integration, we will leverage Stripe Elements
- We don't want a pop up window
- We want to continue to use our existing checkout form (slight mods)
- Provide a seamless user experience
- Leverage PCI compliance features of Stripe

© luv2code LLC

luv2code

Payment Processing with Stripe

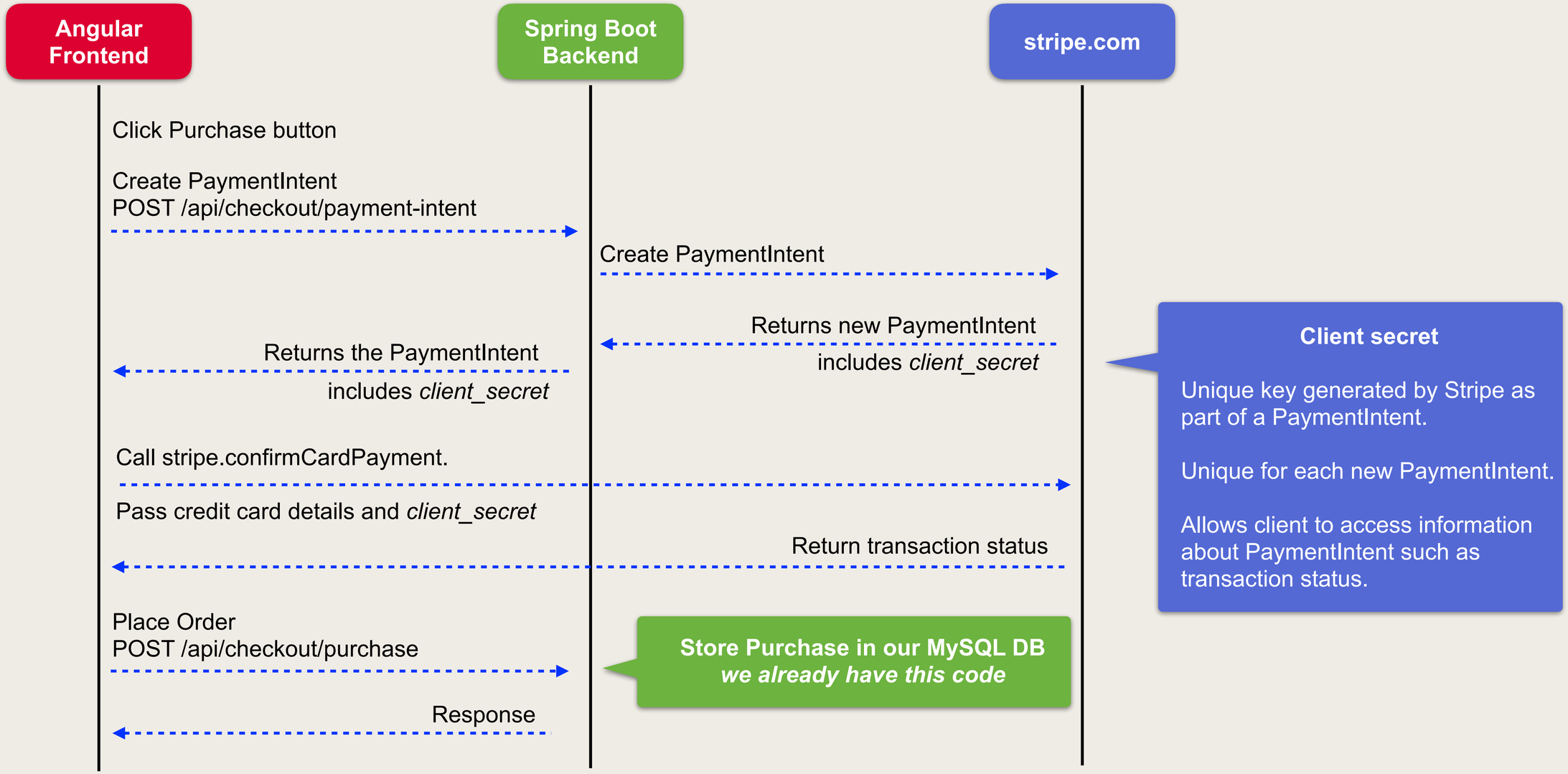
Development Process

Stripe PaymentIntent

- A **PaymentIntent** describes the components of a payment
  - Payment method type
  - Amount
  - Currency

© luv2code LLC

Stripe Payment Processing Flow



© luv2code LLC

Stripe Documentation

www.stripe.com/docs

© luv2code LLC

Development Process Overview

- We will split the process up into backend and frontend

- Start on backend

© luv2code LLC

Development Process - Backend

1. Create Stripe Developer Account
2. Add Stripe Maven Dependency
3. Configure Stripe API Key
4. Create custom PaymentInfo DTO
5. Update CheckoutService to create a PaymentIntent
6. Update CheckoutController to expose /payment-intent endpoint

© luv2code LLC

Step 1: Create Stripe Developer Account

- Create a free account

- Visit: https://dashboard.stripe.com/register

© luv2code LLC

Step 2: Add Stripe Maven Dependency

- Stripe provides a Java library for integration

- Maven dependency

```
<dependency>
  <groupId>com.stripe</groupId>
  <artifactId>stripe-java</artifactId>
  <version>${VERSION-NUMBER}</version>
</dependency>
```


© luv2code LLC

Step 3: Configure Stripe API Key

- Your account has unique API keys
- API keys are available on Stripe Developer Dashboard
- Copy secret key value and place in application.properties

application.properties

```
stripe.key.secret=...
```


© luv2code LLC

Step 4: Create custom PaymentInfo DTO

- We'll create a Data Transfer Object (DTO)
- Contains info regarding payment: amount and currency

PaymentInfo.java

```
package com.luv2code.ecommerce.dto;

import lombok.Data;

@Data
public class PaymentInfo {

    private int amount;
    private String currency;

}
```


© luv2code LLC

Step 5: Update CheckoutService to create a PaymentIntent

CheckoutService.java

```
package com.luv2code.ecommerce.service;

import com.luv2code.ecommerce.dto.PaymentInfo;
import com.luv2code.ecommerce.dto.Purchase;
import com.luv2code.ecommerce.dto.PurchaseResponse;
import com.stripe.exception.StripeException;
import com.stripe.model.PaymentIntent;

public interface CheckoutService {

    PurchaseResponse placeOrder(Purchase purchase);

    PaymentIntent createPaymentIntent(PaymentInfo paymentInfo) throws StripeException;

}
```

From Stripe API

New method

© luv2code LLC

Step 5: Update CheckoutService to create a PaymentIntent

CheckoutServiceImpl.java

```
@Service
public class CheckoutServiceImpl implements CheckoutService {

    private CustomerRepository customerRepository;

    public CheckoutServiceImpl(CustomerRepository customerRepository,
                               @Value("${stripe.key.secret}") String secretKey) {

        this.customerRepository = customerRepository;

        // initialize Stripe API with secret key
        Stripe.apiKey = secretKey;
    }
```

application.properties

```
stripe.key.secret=...
```

Inject secret key from properties file

Initialize Stripe API with secret key

© luv2code LLC

Step 5: Update CheckoutService to create a PaymentIntent

CheckoutServiceImpl.java

```
@Service
public class CheckoutServiceImpl implements CheckoutService {

   ...

   @Override
   public PaymentIntent createPaymentIntent(PaymentInfo paymentInfo) throws StripeException {

       List<String> paymentMethodTypes = new ArrayList<>();
       paymentMethodTypes.add("card");

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

       return PaymentIntent.create(params);
   }

   ...
}
```

Our custom DTO

PaymentIntent is from Stripe API

© luv2code LLC

Step 6: Update CheckoutController to expose /payment-intent endpoint

CheckoutController.java

```
@RestController
@RequestMapping("/api/checkout")
public class CheckoutController {

    private CheckoutService checkoutService;

    ...

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

        PaymentIntent paymentIntent = checkoutService.createPaymentIntent(paymentInfo);
        String paymentStr = paymentIntent.toJson();

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

}
```

Our custom DTO

PaymentIntent is from Stripe API

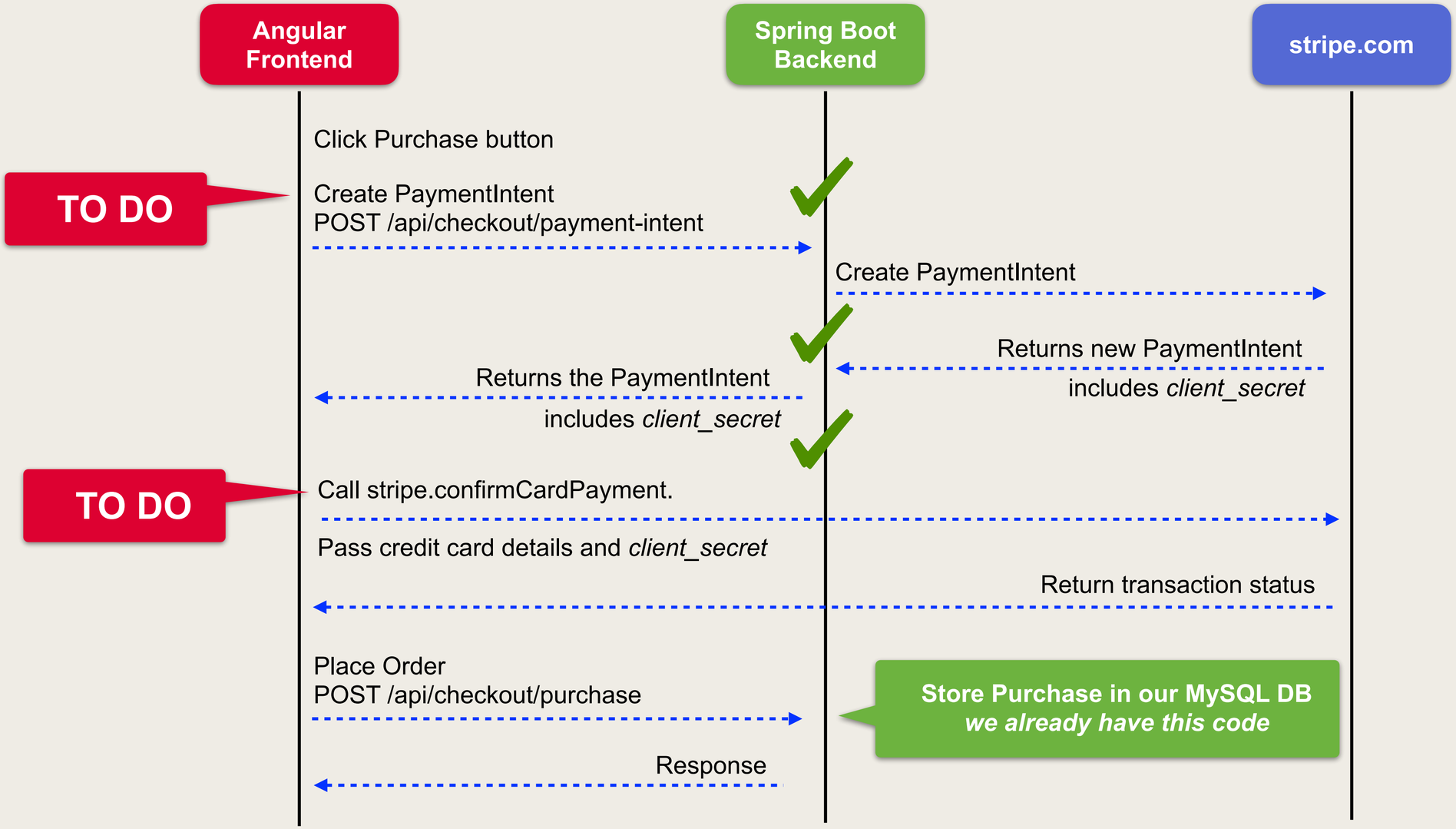
© luv2code LLC

luv2code

Payment Processing with Stripe

Frontend Development Process

Stripe Payment Processing Flow



© luv2code LLC

Stripe Integration: Custom Code Integration

- Stripe provides **Stripe Elements**
  - Prebuilt UI components for collecting credit card info
  - Components are hosted on the Stripe servers
  - Can easily integrate into your existing/custom checkout form
  - Provide seamless integration
  - Supports PCI ... the credit card data is never sent to your servers

© luv2code LLC

Development Process - Frontend

1. Install Stripe API
2. Create a TypeScript declaration file
3. Add Stripe Publishable key
4. Create custom PaymentInfo DTO
5. Update CheckoutService to create a PaymentIntent
6. Load Stripe JavaScript library in index.html
7. Add Stripe Elements to Checkout form

© luv2code LLC

Step 1: Install Stripe API

- Stripe API provides a library for JavaScript development

```
$ npm install stripe
```

- Our Angular app can access the library from TypeScript

© luv2code LLC

Step 2: Create a TypeScript declaration file

- TypeScript declaration file provides type information for JavaScript libraries

- Helps IDEs and compilers to leverage type information of JavaScript libraries

src/typings.d.ts

```
declare var Stripe: any;
declare var elements: any;
```

https://angular.io/guide/typescript-configuration

© luv2code LLC

Step 3: Add Stripe Publishable key

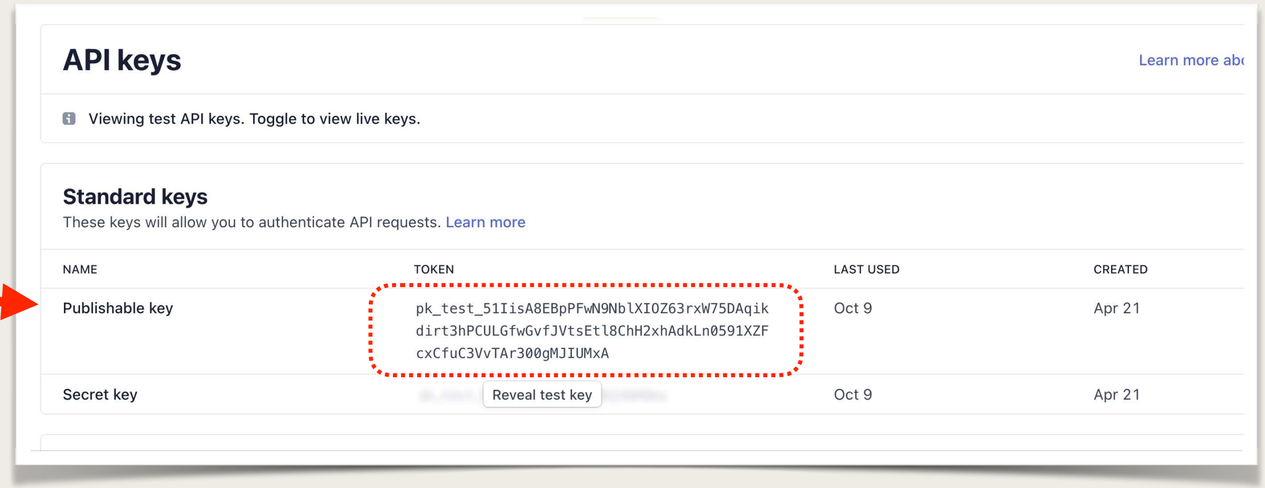
- API keys are available on Stripe Developer Dashboard
- **Publishable key** is used to identify your account with Stripe
- The key isn't secret
- You can safely publish the key in your frontend application

© luv2code LLC

Step 3: Add Stripe Publishable key

environment.ts

```
export const environment = {
  production: false,
  luv2shopApiUrl: "https://localhost:8443/api",
  stripePublishableKey: "pk_test_xxyyzz112233..."
};
```



© luv2code LLC

Step 4: Create custom PaymentInfo DTO

- We'll create a Data Transfer Object (DTO)
- Contains info regarding payment: amount and currency

payment-info.ts

```
export class PaymentInfo {
    amount: number;
    currency: string;
}
```


© luv2code LLC

Step 5: Update CheckoutService to create a PaymentIntent

checkout.service.ts

```
export class CheckoutService {

  private paymentIntentUrl = environment.luv2shopApiUrl + '/checkout/payment-intent';

  constructor(private httpClient: HttpClient) { }


  createPaymentIntent(paymentInfo: PaymentInfo): Observable<any> {
    return this.httpClient.post<PaymentInfo>(this.paymentIntentUrl, paymentInfo);
  }

  ...
}
```

Our custom DTO

Make REST API call

© luv2code LLC

Step 6: Load Stripe JavaScript library in index.html

index.html

```
<!doctype html>
<html lang="en">

<head>
  ...
  <script src="https://js.stripe.com/v3" async></script>

</head>

...

</html>
```

Script is downloaded in parallel to parsing the page

© luv2code LLC

Step 7: Add Stripe Elements to Checkout form

- We are going to **delete** some fields/code from our checkout form

- In previous videos, we created fields for
  - Credit card number, ccv, expiration month and years

- We are removing that code (gasp!)
  - I originally was going to use a different payment processor: **authorize.net**

Older ... traditional

- However, **Stripe** provides these form fields with **Stripe Elements**
  - Simplifies development and helps provide PCI compliance

Cool ... modern

© luv2code LLC

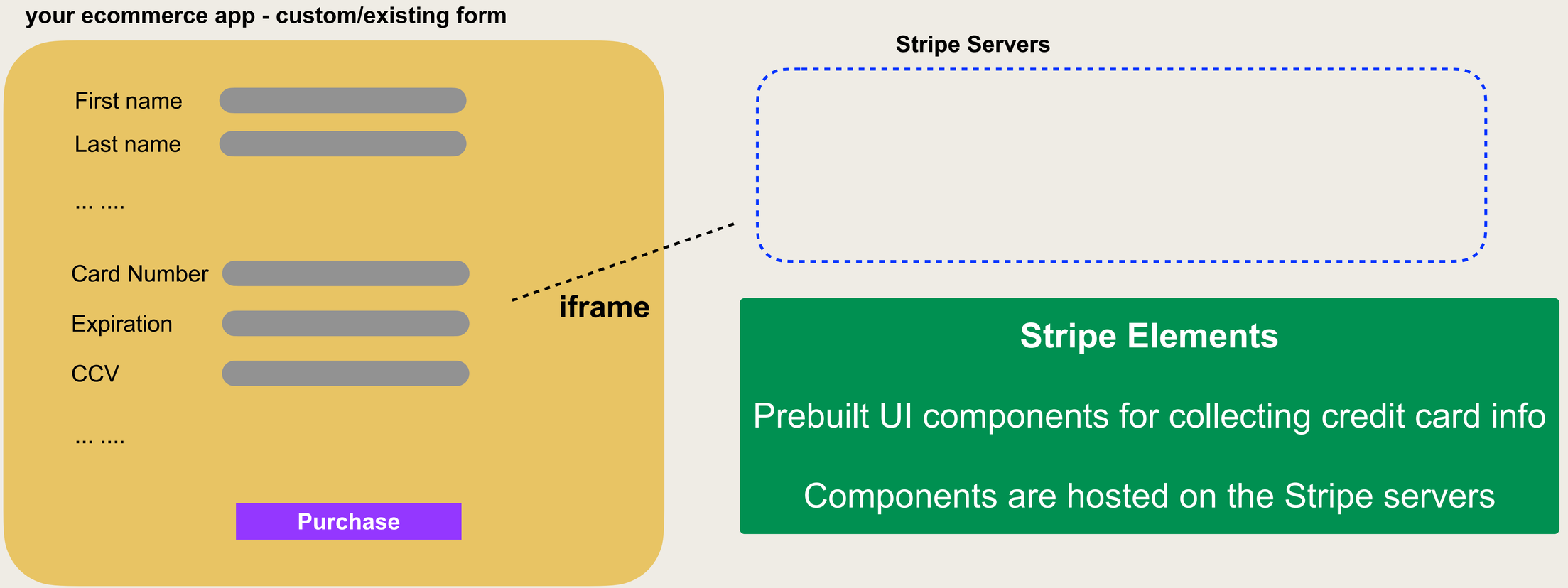
Stripe Integration: Custom Code Integration

- Stripe provides **Stripe Elements**
  - Prebuilt UI components for collecting credit card info
  - Components are hosted on the Stripe servers
  - Can easily integrate into your existing/custom checkout form
  - Provide seamless integration
  - Supports PCI ... the credit card data is never sent to your servers

© luv2code LLC

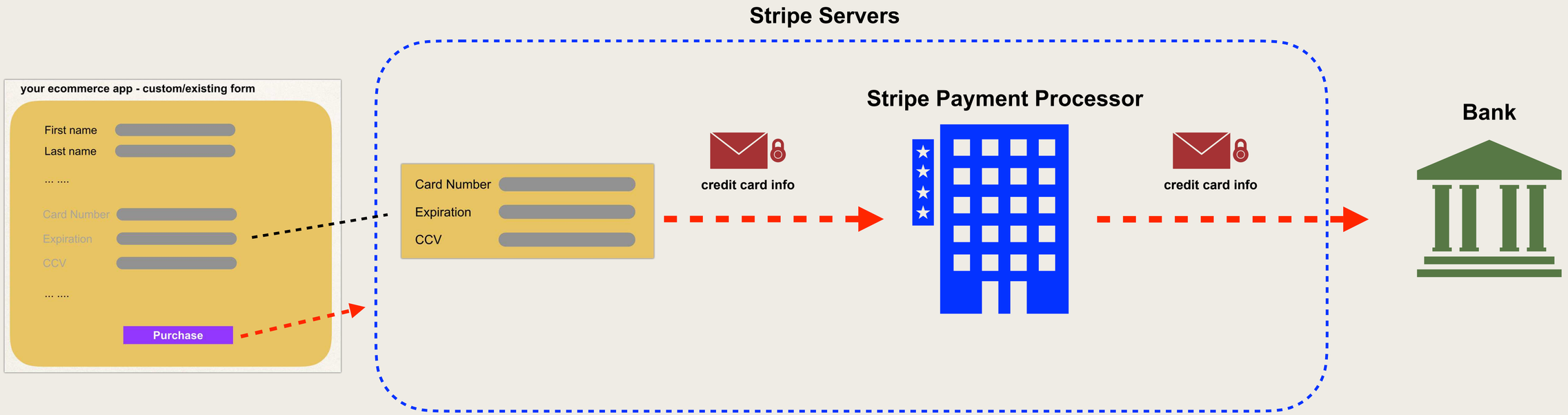
Stripe Integration: Stripe Elements

Provides seamless integration



© luv2code LLC

Stripe Integration: Stripe Elements



© luv2code LLC

Step 7: Add Stripe Elements to Checkout form

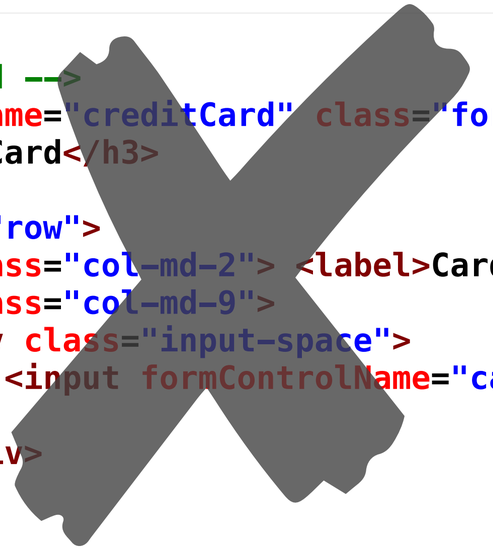
- **Delete** original form fields: card number, ccv, expiration month, year

checkout.component.html

```
...
                <!-- Credit Card -->
                <div formGroupName="creditCard" class="form-area">
                    <h3>Credit Card</h3>

                    <div class="row">
                        <div class="col-md-2"> <label>Card Number</label></div>
                        <div class="col-md-9">
                            <div class="input-space">
                                <input formControlName="cardNumber" type="text">

                            </div>
                        </div>
                    </div>
...
```



© luv2code LLC

Step 7: Add Stripe Elements to Checkout form

- Add support for Stripe Elements

checkout.component.html

```
...
        <!-- Stripe Elements Credit Card Section -->
        <div formGroupName="creditCard" class="form-area">

            <h3>Credit or Debit Card</h3>

            <div id="card-element">
                <!-- a Stripe Element will be inserted here. -->
            </div>

            <!-- Used to display form errors -->
            <div id="card-errors" class="displayError.textContent!==''? 'alert alert-danger mt-1'" ></div>

        </div>
...
```

We will configure Stripe to insert the Stripe Elements here

Covered in next couple of slides ...

© luv2code LLC

Step 7: Add Stripe Elements to Checkout form

- Initialize Stripe API
- Create supporting fields

environment.ts

```
export const environment = {
  production: false,
  luv2shopApiUrl: "https://localhost:8443/api",
  stripePublishableKey: "pk_test_xxyyzz112233..."
};
```

checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  // initialize Stripe API
  stripe = Stripe(environment.stripePublishableKey);

  paymentInfo: PaymentInfo = new PaymentInfo();
  cardElement: any;
  displayError: any =

  ...

}
```

Our custom DTO that we created earlier

Will hold a reference to Stripe Elements card component

© luv2code LLC

Step 7: Add Stripe Elements to Checkout form

- Update ngOnInit() method to set up the Stripe payment form

checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  ngOnInit(): void {

    // setup Stripe payment form
    this.setupStripePaymentForm();

    ...

  }

}
```

We'll create code for this method on next slide

© luv2code LLC

Step 7: Add Stripe Elements to Checkout form

checkout.component.ts

```
setupStripePaymentForm() {

  // get a handle to stripe elements
  var elements = this.stripe.elements();

  // Create a card element ... and hide the zip-code field
  this.cardElement = elements.create('card', { hidePostalCode: true });

  // Add an instance of card UI component into the 'card-element' div
  this.cardElement.mount('#card-element');

  // Add event binding for the 'change' event on the card element
  this.cardElement.on('change', (event) => {

    // get a handle to card-errors element
    this.displayError = document.getElementById('card-errors');

    if (event.complete) {
      this.displayError.textContent = "";
    } else if (event.error) {
      // show validation error to customer
      this.displayError.textContent = event.error.message;
    }

  });

}
```

checkout.component.html

```
...
        <!-- Stripe Elements Credit Card Section -->
        <div formGroupName="creditCard" class="form-area">

            <h3>Credit or Debit Card</h3>

            <div id="card-element">
                <!-- a Stripe Element will be inserted here. -->
            </div>

            <!-- Used to display form errors -->
            <div id="card-errors" class="displayError.textContent!==''': 'alert

        </div>
...
```


© luv2code LLC

Step 7: Add Stripe Elements to Checkout form

- In onSubmit() method ... compute the total amount
- Convert amount in dollars to amount in cents. Multiply by 100

checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  onSubmit() {

    // compute payment info
    this.paymentInfo.amount = this.totalPrice * 100;
    this.paymentInfo.currency = "USD";

    ...
  }

}
```


© luv2code LLC

Step 7: Add Stripe Elements to Checkout form

- Make REST API calls for payment processing
  - Create payment intent
  - Confirm card payment
  - Place order

© luv2code LLC

```
export class CheckoutComponent implements OnInit {

  onSubmit() {
      ...
      this.checkoutService.createPaymentIntent(this.paymentInfo).subscribe(
        (paymentIntentResponse) => {
          this.stripe.confirmCardPayment(paymentIntentResponse.client_secret,
            {
              payment_method: {
                card: this.cardElement
              }
            }, { handleActions: false })
          .then(function(result) {
            if (result.error) {
              // inform the customer there was an error
              alert(`There was an error: ${result.error.message}`);
            } else {
              // call REST API via the CheckoutService
              this.checkoutService.placeOrder(purchase).subscribe({
                next: response => {
                  alert(`Your order has been received.\nOrder tracking number: ${response.orderTrackingNumber}`);

                  // reset cart
                  this.resetCart();
                },
                error: err => {
                  alert(`There was an error: ${err.message}`);
                }
              })
            }
          }.bind(this));
        }
      );
    ...
```

Create payment intent
Spring Boot REST API

Confirm card payment
stripe.com

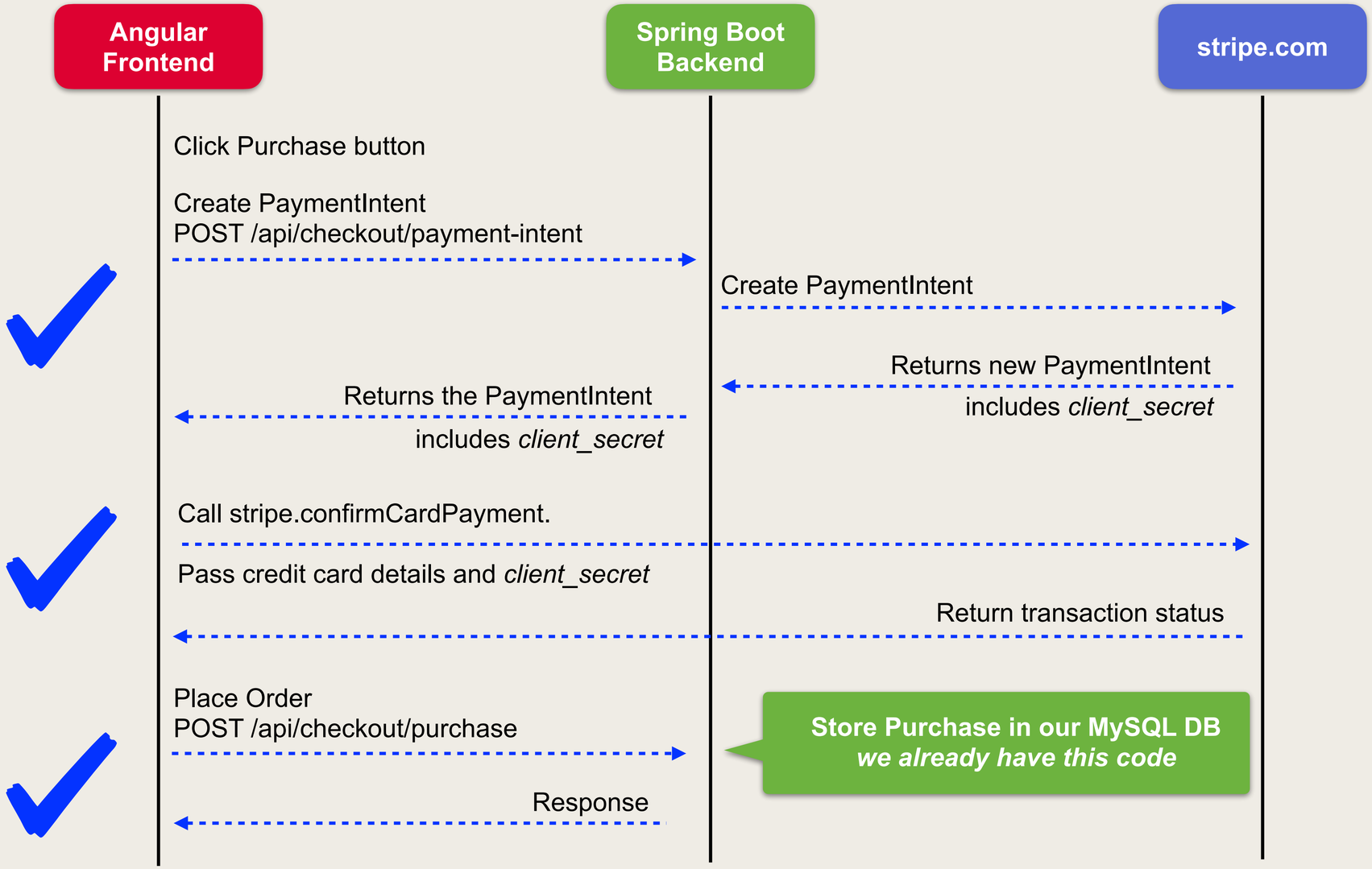
Send credit card data
directly to stripe.com servers

Reference the Stripe Elements component: cardElement

Place order ... store in MySQL DB
Spring Boot REST API

© luv2code LLC

Stripe Payment Processing Flow



© luv2code LLC

Testing Credit Card Payments

- Stripe provides a **test** sandbox ... no charges are sent to banks
- Sample test card numbers are available for various scenarios
- Real credit card information can not be used in test mode
    - Your personal credit card information will not work in test mode

© luv2code LLC

Testing Credit Card Payments

- Test using any of the following test card numbers
- Valid expiration date in the future
- CVC number can be any three digits

Number	CVC	Expiration Date	Result
4242 4242 4242 4242	Any 3 digits	Any future date	Success
4000 0000 0000 0002	Any 3 digits	Any future date	Card Declined
4000 0000 0000 9995	Any 3 digits	Any future date	Card Declined - Insufficient funds
4000 0000 0000 9979	Any 3 digits	Any future date	Card Declined - Stolen card
...	...	...	...

© luv2code LLC

Testing Credit Card Payments

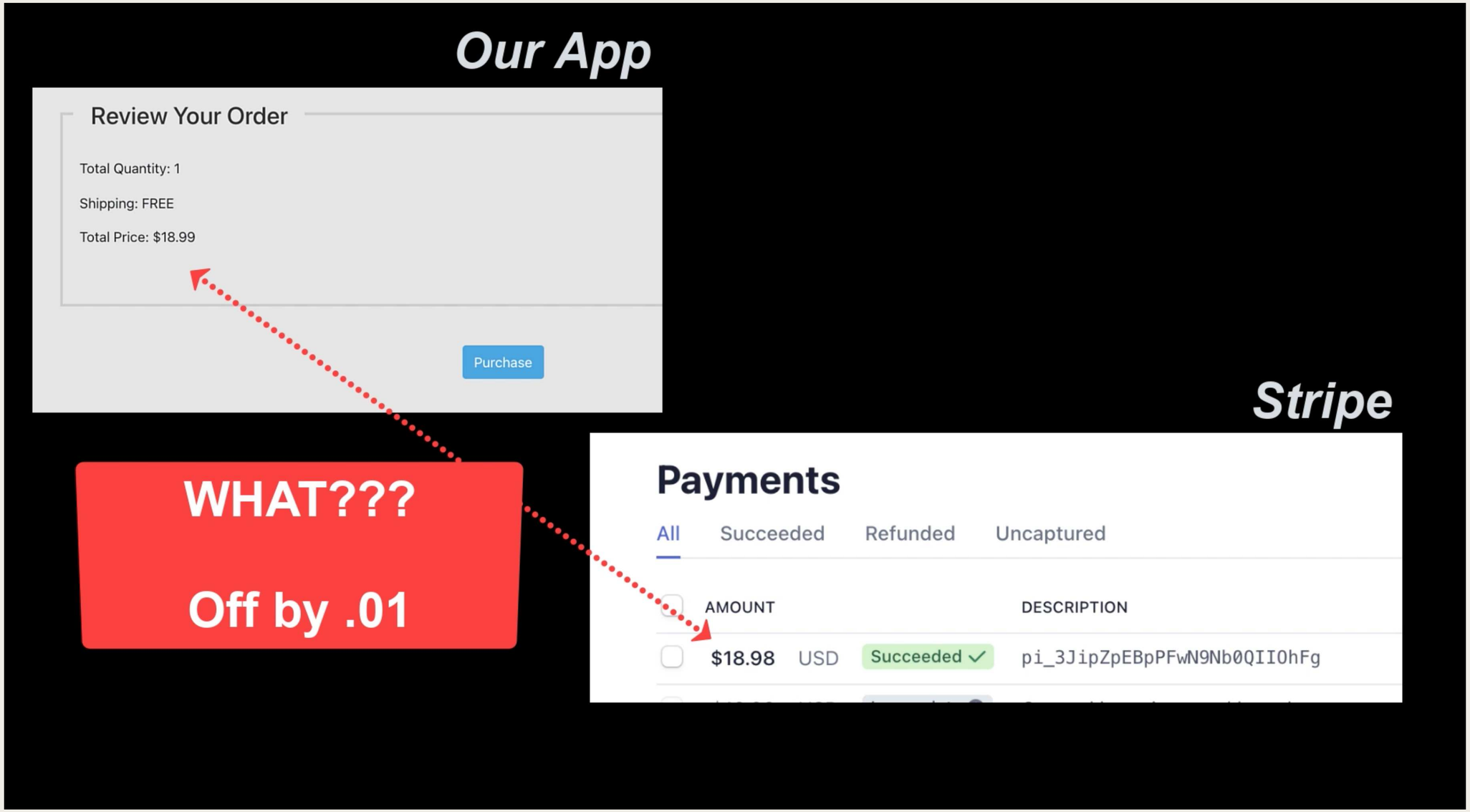
- Additional test card numbers and scenarios available

https://stripe.com/docs/testing

© luv2code LLC

Payment Processing with Stripe

Bug Fix for Payment Amount



© luv2code LLC

© luv2code LLC

JavaScript sent over a floating point number

When the data makes it to Java Spring Boot backend,
the floating point number is truncated to a Java int

JavaScript 1898.99999 is TRUNCATED to a Java int 1898

The value of 1898 is passed to Stripe via Java Stripe API
(creating PaymentIntent)

That is why Stripe has $18.98

© luv2code LLC

One possible solution

On JavaScript side, ROUND the number before sending

Math.round(1898.9999) ==> 1899.0

No issues with truncation when passed to Java

Gives us $18.99 in Stripe as desired

© luv2code LLC

File: checkout.component.ts

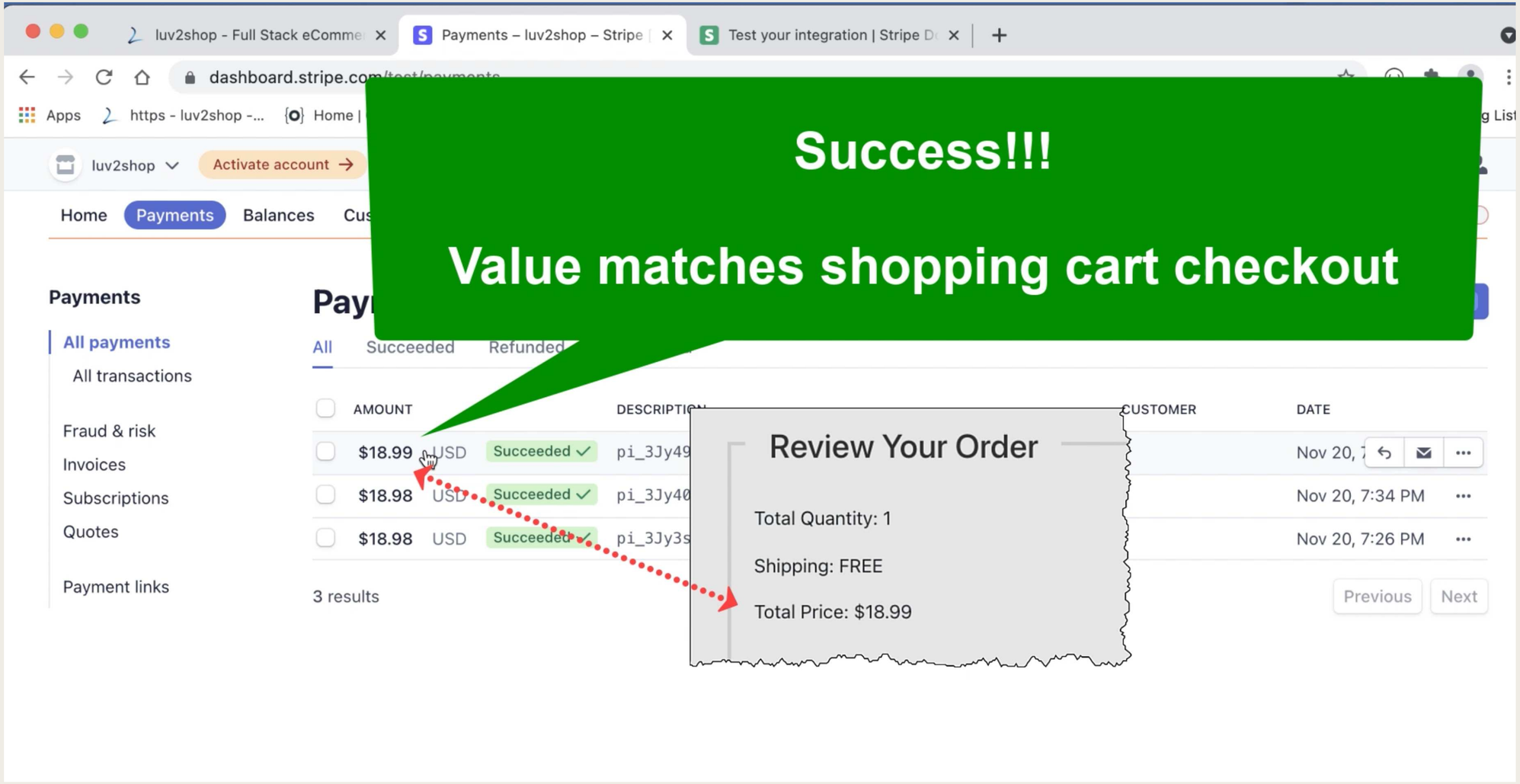
Replace

```
this.paymentInfo.amount = this.totalPrice*100;
```

With

```
this.paymentInfo.amount = Math.round(this.totalPrice*100);
```


© luv2code LLC



© luv2code LLC

Payment Processing with Stripe

Bug Fix for Cart Reload after Check out

PROBLEM

After the checkout
if we RELOAD the page,
then we see old shopping cart contents :-(

However, the cart should be empty

© luv2code LLC

This is the result of "browser reload" logic that we added earlier in the course

Solution
When we reset the cart,
clear out the storage

© luv2code LLC

File: checkout.component.ts

In the `resetCart()` method

Add this line

```
this.cartService.persistCartItems();
```

This happens right after we empty the cart.

The storage will not have any items.

Hence when we reload the page after checkout then the cart will be empty as desired. :-)

© luv2code LLC

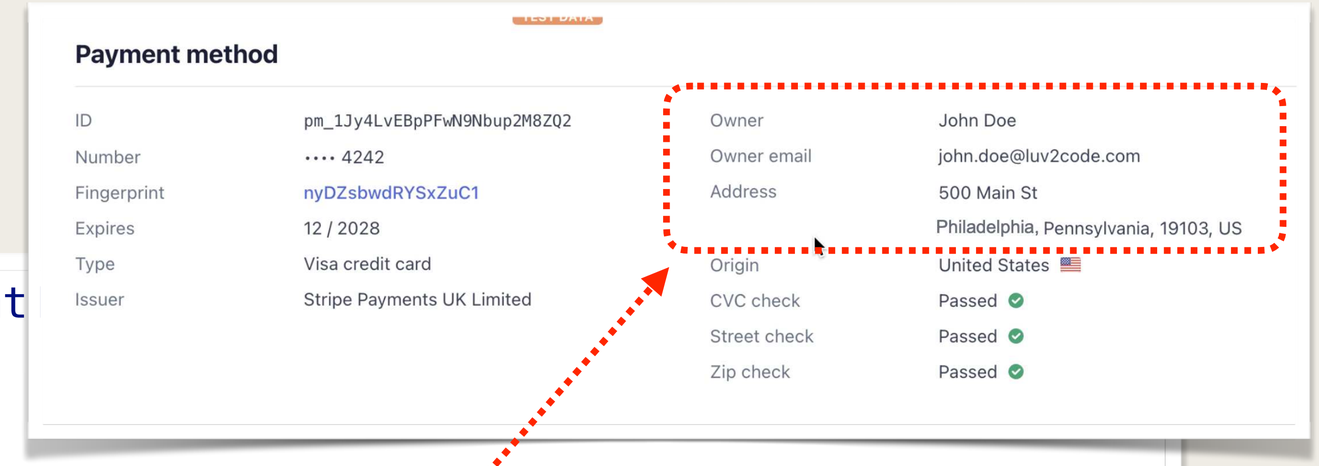
Payment Processing with Stripe

Add Customer Details

Currently Missing Customer Details

© luv2code LLC

Add Customer Details



checkout.component.ts

```
this.stripe.confirmCardPayment(payment
  payment_method: {
      card: this.cardElement,

      billing_details: {
         email: purchase.customer.email,
         name: `${purchase.customer.firstName} ${purchase.customer.lastName}`,
         address: {
           line1: purchase.billingAddress.street,
           city: purchase.billingAddress.city,
           state: purchase.billingAddress.state,
           postal_code: purchase.billingAddress.zipCode,
           country: this.billingAddressCountry.value.code
         }
      }

 },
```


© luv2code LLC

Payment Processing with Stripe

Disable Purchase Button

Disable Purchase Button to prevent multiple form submissions

© luv2code LLC

Development Process

1. Update checkout.component.ts
    1. Add a boolean field `isDisabled`
    2. Before REST API calls, set `isDisabled` to `true`
    3. Once REST API calls complete, set `isDisabled` to `false`

2. Update checkout.component.html
    1. Bind `disabled` attribute to `isDisabled` field

© luv2code LLC

checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  isDisabled: boolean = false;
  ...

  onSubmit() {
    ...
    if (!this.checkoutFormGroup.invalid && this.displayError.textContent === "") {

      this.isDisabled = true;

      this.checkoutService.createPaymentIntent(this.pa
      ...
              if (result.error) {
                // Inform the customer that there was an error
                alert(`There was an error: ${result.error.message}`);
                this.isDisabled = false;
              } else {
                // call REST API via the CheckoutServi
                this.checkoutService.placeOrder(purcha
                  next: response => {
                    alert(`Your order has been received.\norder tracking number: ${response.orderTrackingNumber}`);

                    // reset cart
                    this.resetCart();
                    this.isDisabled = false;
                  },
                  error: err => {
                    alert(`There was an error: ${err.message}`);
                    this.isDisabled = false;
                  }
                })
```

Add new field

Before REST API calls, set field to true

After REST API calls, set field to false

After REST API calls, set field to false

© luv2code LLC

checkout.component.html

```
<!-- submit button-->
<div class="text-center">
       <button type="submit" class="btn btn-info" [disabled]="isDisabled">Purchase</button>
</div>
```

Bind button to `isDisabled` field of checkout.component.ts

Button is automatically disabled/enabled

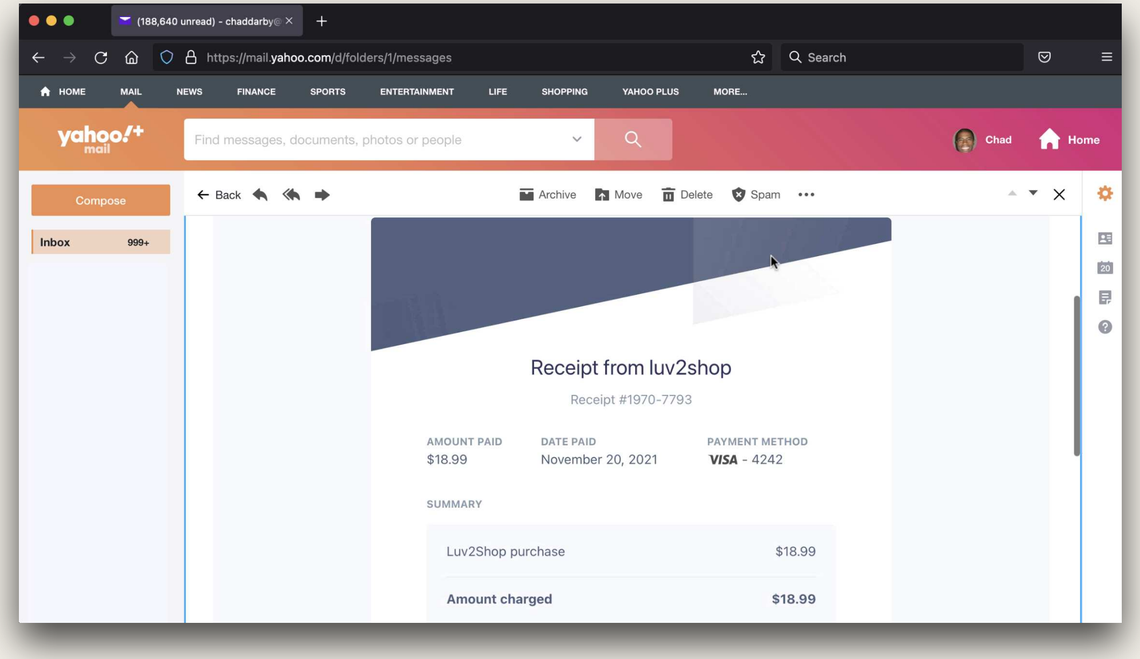
© luv2code LLC

luv2code

Payment Processing with Stripe

Email Receipts

Let's send an email receipt after checkout



© luv2code LLC

Stripe Email Receipts

- Stripe can automatically send email receipts after a successful payment
- Developer provides customer's email when creating a `PaymentIntent`

- Note: In **Test** mode ... using Test API keys
  - Email receipts are NOT sent automatically
  - You need to manually send email receipts from Stripe Dashboard

© luv2code LLC

Development Process

1. Angular
    1. Add email field to payment-info.ts
    2. In onSubmit(), assign email field to customer's email address

2. Spring Boot
    1. Add email field to PaymentInfo.java
    2. In createPaymentIntent(), assign email address

© luv2code LLC

Step 1.1: Add email field to payment-info.ts

payment-info.ts

```
export class PaymentInfo {
    amount: number;
    currency: string;
    receiptEmail: string;
}
```


© luv2code LLC

Step 1.2: In onSubmit(), assign email field to customer's email address

checkout.component.ts

```
  onSubmit() {
    ...

    // compute payment info
    this.paymentInfo.amount = Math.round(this.totalPrice*100);
    this.paymentInfo.currency = "USD";

    this.paymentInfo.receiptEmail = purchase.customer.email;

    ...
  }
```


© luv2code LLC

Step 2.1: Add email field to PaymentInfo.java

PaymentInfo.java

```
@Data
public class PaymentInfo {

    private int amount;
    private String currency;
    private String receiptEmail;

}
```


© luv2code LLC

Step 2.2: In createPaymentIntent(), assign email address

CheckoutServiceImpl.java

```
@Service
public class CheckoutServiceImpl implements CheckoutService {

  ...

  @Override
  public PaymentIntent createPaymentIntent(PaymentInfo paymentInfo) throws StripeException {

    List<String> paymentMethodTypes = new ArrayList<>();
    paymentMethodTypes.add("card");

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

    params.put("receipt_email", paymentInfo.getReceiptEmail());

    return PaymentIntent.create(params);
  }

}
```


© luv2code LLC