Phase 3: Payment Processing

The Shrey Method Fitness Platform Guide

← Back to Phase 2 | Next: Phase 4 - Scheduling →

In this phase, we'll add payment processing capabilities to your fitness platform using Stripe. This will allow you to:

- Accept payments for your coaching services
- Set up subscription plans
- Manage payment methods securely

We'll break this down into small, manageable steps.

Step 1: Create a Stripe Account

First, you need to create a Stripe account to handle payments.

1. Go to stripe.com
2. Click "Start now" or "Create account"
3. Follow the sign-up process:
    - Enter your email address
    - Create a password
    - Enter your personal information
    - Enter your business information
4. Once you're signed in, go to the Developers section
5. Find your API keys (you'll need these later)

What's happening here? You're creating a Stripe account that will handle payment processing for your fitness platform. Stripe is a secure payment processor that will handle all the sensitive payment information.

Step 2: Install Stripe Libraries

Now let's install the Stripe libraries that will allow you to integrate Stripe with your fitness platform.

1. In the terminal, type:

```
npm install stripe @stripe/stripe-js
```

2. Wait for the installation to complete

What's happening here? You're installing the Stripe libraries that will allow you to integrate Stripe with your fitness platform. These libraries provide the tools to interact with the Stripe API.

Step 3: Create Lambda Function for Payment Processing

Now let's create a Lambda function that will handle payment processing using Stripe.

1. In the terminal, type:

```
amplify add function
```

2. Answer the prompts:

   - Function name: "processPayment"
   - Choose the runtime: "NodeJS"
   - Choose the function template: "Hello World"
   - Do you want to configure advanced settings: "Yes"
   - Do you want to access other resources in this project from your Lambda function: "No"
   - Do you want to invoke this function on a recurring schedule: "No"
   - Do you want to configure Lambda layers for this function: "No"
   - Do you want to edit the local lambda function now: "Yes"

3. It will open the function file in VS Code. Replace the content with:

```
const stripe = require('stripe')('sk_test_your_stripe_secret_key');

exports.handler = async (event) => {
    try {
        const body = JSON.parse(event.body);
        const { operation, payload } = body;

        switch (operation) {
            case 'createPaymentIntent':
                return await createPaymentIntent(payload);
            case 'createSubscription':
                return await createSubscription(payload);
            case 'cancelSubscription':
                return await cancelSubscription(payload);
            default:
                return {
                    statusCode: 400,
                    body: JSON.stringify({ error: 'Invalid operation' })
                };
        }
    } catch (error) {
        return {
            statusCode: 500,
            body: JSON.stringify({ error: error.message })
        };
    }
};

async function createPaymentIntent(payload) {
    const { amount, currency, customerId } = payload;
```

```
    const paymentIntent = await stripe.paymentIntents.create({
        amount,
        currency,
        customer: customerId
    });

    return {
        statusCode: 200,
        body: JSON.stringify({
            clientSecret: paymentIntent.client_secret
        })
    };
}

async function createSubscription(payload) {
    const { customerId, priceId } = payload;

    const subscription = await stripe.subscriptions.create({
        customer: customerId,
        items: [{ price: priceId }],
        expand: ['latest_invoice.payment_intent']
    });

    return {
        statusCode: 200,
        body: JSON.stringify({
            subscriptionId: subscription.id,
            clientSecret: subscription.latest_invoice.payment_intent.client_secret
        })
    };
}

async function cancelSubscription(payload) {
    const { subscriptionId } = payload;

    const canceledSubscription = await stripe.subscriptions.del(subscriptionId);

    return {
        statusCode: 200,
        body: JSON.stringify({
            subscriptionId: canceledSubscription.id,
            status: canceledSubscription.status
        })
    };
}
```

4. Replace `'sk_test_your_stripe_secret_key'` with your actual Stripe secret key (from the Stripe Dashboard > Developers > API keys)
5. Save the file (Ctrl+S or File > Save)

What's happening here? You're creating a Lambda function that will handle payment processing using Stripe. This function will be called by your frontend code to create payment intents, subscriptions, and cancel

subscriptions.

Step 4: Create API Endpoint for Payment Processing

Now let's create an API endpoint that will connect to your Lambda function.

1. In the terminal, type:

```
amplify add api
```

2. Answer the prompts:
    - Select "REST" as the API service
    - Enter API name: "paymentapi"
    - Enter path: "/payments"
    - Choose Lambda source: "Use a Lambda function already added in the current Amplify project"
    - Choose the Lambda function: "processPayment"
    - Do you want to restrict API access: "Yes"
    - Who should have access: "Authenticated users only"
    - What kind of access do you want for Authenticated users: "read/write"
    - Do you want to add another path: "No"

What's happening here? You're creating an API endpoint that will connect to your Lambda function for payment processing. This endpoint will be called by your frontend code to process payments.

Step 5: Deploy Your Updated Backend

Now let's update your AWS resources with the new Lambda function and API endpoint.

1. In the terminal, type:

```
amplify push
```

2. Review the changes and confirm by typing "y"
3. Wait for the deployment to complete

What's happening here? AWS is updating your resources with the new Lambda function and API endpoint that you configured.

Step 6: Create Client Payments Page

Now let's create the payments page that clients will see when they want to subscribe to your coaching plans.

1. Create a new file called `client-payments.html`
2. Copy and paste this code:

```
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Payments - The Shrey Method Fitness</title>
    <link rel="stylesheet" href="css/styles.css">
    <link rel="stylesheet" href="css/dashboard.css">
    <script src="https://cdn.jsdelivr.net/npm/aws-amplify@5.0.4/dist/aws-
amplify.min.js"></script>
    <script src="https://js.stripe.com/v3/"></script>
</head>
<body>
    <div class="dashboard-container">
        <!-- Sidebar navigation (same as client-dashboard.html) -->
        <div class="dashboard-sidebar">
            <div class="logo">
                <h2>The Shrey Method</h2>
            </div>
            <nav class="dashboard-nav">
                <ul>
                    <li><a href="client-dashboard.html">Overview</a></li>
                    <li><a href="client-program.html">My Program</a></li>
                    <li><a href="client-nutrition.html">Nutrition</a></li>
                    <li><a href="client-messages.html">Messages</a></li>
                    <li class="active"><a href="client-payments.html">Payments</a>
</li>
                    <li><a href="client-schedule.html">Schedule</a></li>
                </ul>
            </nav>
            <div class="sidebar-footer">
                <button id="logout-btn" class="btn-secondary">Log Out</button>
            </div>
        </div>

        <!-- Main content area -->
        <div class="dashboard-content">
            <header class="dashboard-header">
                <h1>Payments</h1>
                <div class="user-menu">
                    <span id="user-name"></span>
                    <img src="images/default-avatar.jpg" alt="Profile"
class="avatar" id="user-avatar">
                </div>
            </header>

            <div class="content-container">
                <div class="dashboard-grid">
                    <!-- Available Plans Card -->
                    <div class="dashboard-card">
                        <h2>Available Plans</h2>
                        <div id="available-plans">
                            <div class="plan-card">
```

```
                                <h3>Monthly Coaching</h3>
                                <p class="plan-price">$199/month</p>
                                <ul class="plan-features">
                                    <li>Personalized workout plans</li>
                                    <li>Nutrition guidance</li>
                                    <li>Weekly check-ins</li>
                                    <li>24/7 messaging support</li>
                                </ul>
                                <button class="btn-primary subscribe-btn" data-
plan="monthly">Subscribe</button>
                            </div>

                            <div class="plan-card">
                                <h3>3-Month Coaching</h3>
                                <p class="plan-price">$179/month</p>
                                <p class="plan-discount">Save 10%</p>
                                <ul class="plan-features">
                                    <li>All monthly features</li>
                                    <li>Discounted rate</li>
                                    <li>Priority support</li>
                                </ul>
                                <button class="btn-primary subscribe-btn" data-
plan="quarterly">Subscribe</button>
                            </div>
                        </div>
                    </div>

                    <!-- Payment Method Card -->
                    <div class="dashboard-card">
                        <h2>Payment Method</h2>
                        <div id="payment-method">
                            <p>No payment method on file.</p>
                            <button id="add-payment-method-btn" class="btn-
primary">Add Payment Method</button>
                        </div>
                    </div>

                    <!-- Payment History Card -->
                    <div class="dashboard-card">
                        <h2>Payment History</h2>
                        <div id="payment-history">
                            <p>No payment history to display.</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- Payment Modal -->
    <div id="payment-modal" class="modal">
        <div class="modal-content">
            <span class="close">&times;</span>
            <h2>Add Payment Method</h2>
```

```
            <form id="payment-form">
                <div id="card-element">
                    <!-- Stripe Card Element will be inserted here -->
                </div>
                <div id="card-errors" class="error-message"></div>
                <button type="submit" class="btn-primary">Save Payment
Method</button>
            </form>
        </div>
    </div>

    <script src="js/client-payments.js"></script>
</body>
</html>
```

3. Save the file (Ctrl+S or File > Save)

What's happening here? You're creating the HTML structure for the payments page, which includes available plans, payment methods, and payment history.

Step 7: Add CSS for Payment Page

Now let's add the CSS styling for the payments page.

1. Open `css/dashboard.css`
2. Add this code at the end of the file:

```
/* Payment Styles */
.plan-card {
    border: 1px solid #ddd;
    border-radius: 8px;
    padding: 20px;
    margin-bottom: 20px;
}

.plan-card h3 {
    margin-top: 0;
    color: var(--secondary);
}

.plan-price {
    font-size: 1.5rem;
    font-weight: 600;
    color: var(--primary);
    margin: 10px 0;
}

.plan-discount {
    color: var(--primary);
    font-weight: 500;
    margin-top: -5px;
```

```
    margin-bottom: 10px;
}

.plan-features {
    padding-left: 20px;
    margin-bottom: 20px;
}

.plan-features li {
    margin-bottom: 5px;
}

/* Modal Styles */
.modal {
    display: none;
    position: fixed;
    z-index: 1000;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.5);
}

.modal-content {
    background-color: white;
    margin: 10% auto;
    padding: 30px;
    border-radius: 10px;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
    width: 90%;
    max-width: 500px;
    position: relative;
}

.close {
    position: absolute;
    top: 15px;
    right: 20px;
    font-size: 24px;
    font-weight: bold;
    color: var(--dark-gray);
    cursor: pointer;
}

/* Stripe Elements */
#card-element {
    border: 1px solid #ddd;
    border-radius: 5px;
    padding: 15px;
    margin-bottom: 20px;
}
```

3. Save the file (Ctrl+S or File > Save)

What's happening here? You're adding CSS styling for the payments page, including styles for the plan cards, modal, and Stripe elements.

Step 8: Create Client Payments JavaScript

Now let's create the JavaScript code that will handle the payments functionality.

1. Create a new file called `js/client-payments.js`
2. Copy and paste this code:

```
// Configure Amplify
const awsConfig = {
    Auth: {
        region: 'us-west-2', // Your AWS region
        userPoolId: 'us-west-2_xxxxxxxx', // Your Cognito User Pool ID
        userPoolWebClientId: 'xxxxxxxxxxxxxxxxxxxxxxxxxx', // Your App Client ID
    },
    API: {
        endpoints: [{
            name: "shreymethodapi",
            endpoint: "https://xxxxxxxxxx.execute-api.us-west-2.amazonaws.com/dev"
        }, {
            name: "paymentapi",
            endpoint: "https://xxxxxxxxxx.execute-api.us-west-2.amazonaws.com/dev"
        }]
    }
};

// Replace with your actual values (same as in auth.js)

Amplify.configure(awsConfig);

// Initialize Stripe
const stripe = Stripe('pk_test_your_stripe_publishable_key'); // Replace with your
Stripe publishable key
let cardElement;

// Check authentication
async function checkAuth() {
    try {
        const user = await Amplify.Auth.currentAuthenticatedUser();
        return user;
    } catch (error) {
        window.location.href = 'client-login.html';
    }
}

// Initialize
document.addEventListener('DOMContentLoaded', async () => {
    try {
```

```
        const user = await checkAuth();

        // Set user name
        const userName = document.getElementById('user-name');
        userName.textContent = user.attributes.name || user.username;

        // Set up payment modal
        const modal = document.getElementById('payment-modal');
        const addPaymentMethodBtn = document.getElementById('add-payment-method-
btn');
        const closeBtn = document.querySelector('.close');

        addPaymentMethodBtn.addEventListener('click', () => {
            modal.style.display = 'block';
            setupStripeElements();
        });

        closeBtn.addEventListener('click', () => {
            modal.style.display = 'none';
        });

        window.addEventListener('click', (event) => {
            if (event.target === modal) {
                modal.style.display = 'none';
            }
        });

        // Set up subscription buttons
        document.querySelectorAll('.subscribe-btn').forEach(button => {
            button.addEventListener('click', () => {
                alert('This feature will be implemented in a future update.');
            });
        });

        // Set up payment form
        document.getElementById('payment-form').addEventListener('submit', async
(event) => {
            event.preventDefault();
            alert('This feature will be implemented in a future update.');
        });

        // Set up logout
        document.getElementById('logout-btn').addEventListener('click', async ()
=> {
            try {
                await Amplify.Auth.signOut();
                window.location.href = 'client-login.html';
            } catch (error) {
                console.error('Error signing out: ', error);
            }
        });

    } catch (error) {
        console.error('Error initializing payments page:', error);
```

```
    }
});

// Set up Stripe Elements
function setupStripeElements() {
    const elements = stripe.elements();

    // Create card element
    cardElement = elements.create('card');
    cardElement.mount('#card-element');

    // Handle validation errors
    cardElement.addEventListener('change', (event) => {
        const displayError = document.getElementById('card-errors');
        if (event.error) {
            displayError.textContent = event.error.message;
        } else {
            displayError.textContent = '';
        }
    });
}
```

3. Save the file (Ctrl+S or File > Save)
4. Update the configuration values with your actual values (same as in Phase 1, Step 7)
5. Replace `'pk_test_your_stripe_publishable_key'` with your actual Stripe publishable key (from the Stripe Dashboard > Developers > API keys)

What's happening here? You're creating the JavaScript code that will handle the payments functionality, including setting up Stripe Elements, handling payment methods, and processing subscriptions.

Step 9: Set Up Stripe Products and Prices

Now let's set up your coaching plans in Stripe.

1. Go to the Stripe Dashboard
2. Click on "Products" in the left sidebar
3. Click "Add product"
4. Create your first product:
    - Name: "Monthly Coaching"
    - Description: "Personalized workout plans, nutrition guidance, weekly check-ins, and 24/7 messaging support"
    - Pricing: "Recurring"
    - Price: $199 per month
    - Click "Save product"
5. Click "Add product" again
6. Create your second product:
    - Name: "3-Month Coaching"
    - Description: "All monthly features plus discounted rate and priority support"
    - Pricing: "Recurring"
    - Price: $179 per month

    - Billing period: Every 3 months
    - Click "Save product"

What's happening here? You're setting up your coaching plans in Stripe so that clients can subscribe to them.

Step 10: Test the Payments Page

Now let's test your payments page to make sure everything works correctly.

1. Open your website in a browser:
    - Right-click on `client-login.html` in VS Code
    - Select "Open with Live Server" (if you have the Live Server extension)
    - Or open the file directly in your browser
2. Log in with a client account
3. Navigate to the Payments page
4. Test the "Add Payment Method" button:
    - Click the button
    - The payment modal should appear
    - You should see the Stripe card element
    - Close the modal
5. Test the subscription buttons:
    - Click on a "Subscribe" button
    - You should see an alert saying "This feature will be implemented in a future update"

What's happening here? You're testing the payments page to make sure the basic functionality works correctly.

Next Steps

In this phase, we've set up the foundation for payment processing with Stripe. In a real-world scenario, you would now implement the full payment processing functionality, including:

- Creating Stripe customers for your users
- Saving and managing payment methods
- Processing subscriptions
- Handling webhooks for payment events
- Displaying payment history

However, for the purposes of this guide, we'll move on to the next phase: adding scheduling capabilities with Calendly.

Congratulations!

You've completed Phase 3 of your fitness platform. You now have the foundation for payment processing with Stripe. In the next phase, we'll add scheduling capabilities to your platform.

← Back to Phase 2 | Next: Phase 4 - Scheduling →