Extracted from:

Take My Money

Accepting Payments on the Web

This PDF file contains pages extracted from *Take My Money*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2017 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Take My Money

Accepting Payments on the Web

Noel Rappin

edited by Katharine Dvorak

Take My Money

Accepting Payments on the Web

Noel Rappin

The Pragmatic Bookshelf

Raleigh, North Carolina

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Katharine Dvorak (editor)
Potomac Indexing, LLC (index)
Liz Welch (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Copyright © 2017 The Pragmatic Programmers, LLC.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 978-1-68050-199-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January 2017

Disclaimer: This book is intended only as an informative guide on setting up a financial transaction website. Information in this book is general, and the author, editors, and The Pragmatic Programmers, LLC disclaim all liability for compliance to federal, state, and local laws in connection with the use of this book. This work is sold with the understanding that the author, editors, and The Pragmatic Programmers, LLC do not offer legal, financial, or other professional services. If professional assistance is required, the services of a competent professional person should be sought.

One day, the theater management gathers our development team to make a simple request. A survey of our website patrons determined that 20 percent of them are refusing to give us their credit card information at all. They'd rather use PayPal to pay us. But we don't offer PayPal.

We sigh. We knew this day was going to arrive.

Although many payment gateways work more or less like Stripe, which we set up in Chapter 2, *Take the Money*, on page ?, PayPal is a popular gateway that uses an entirely different payment process. I'm tempted to call PayPal "a more elegant weapon from a more civilized age," except that it's less elegant, and whether the age was more civilized is debatable.

PayPal tries to solve the exact security issue we just solved with our client-side Stripe solution: allowing users to securely make payments without sending their credit card information directly to our servers. Where Stripe uses an Ajax and token mechanism to keep credit card information off our site, PayPal's workflow asks users to log in and present their personal information to PayPal as part of our site's checkout process. (PayPal does have a service that takes credit card information from business sites; however, because this service isn't commonly used, and its functionality is similar to Stripe, I'm not going to discuss it here.)

In this chapter, we'll set up a PayPal business account and learn how to accept PayPal as a source for authorizing transactions.

Setting Up a PayPal Account

To accept payments via PayPal, you need a PayPal business account. If you already have a PayPal personal account, you really want to keep that separate from your developer account. Here's how to create a PayPal business account, at least as of this writing:

Go to the PayPal website.[1] Click "Sign up" and then "Open a business account." You are directed to the business home page, then click "Get Started."

Now it gets a little confusing. You have three options: "PayPal Payments Standard," "PayPal Payments Pro," and "Already accepting credit cards?" That third option, which is usually called "PayPal Express Checkout" and which for some reason PayPal isn't identifying as such on the site, is the one we want. That's the classic "Check out with PayPal button" experience you are probably familiar with.

1. https://www.paypal.com

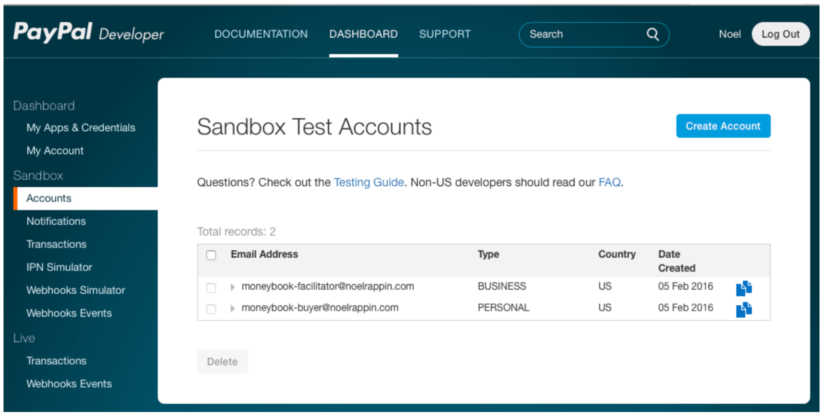
To select the Express Checkout option, click "Learn More" following the "Already accepting credit cards?" option and then click "Get Express Checkout." You are now asked to answer a couple of questions about your business and enter your business's URL. Next you are asked for an email address. When you enter your email address you are taken to a "Sign up for a Business account" form with that email address filled in. (It seems we could have gotten to this form in fewer than four pages, but *c'est la vie*.)

The form asks us to enter a legal name and a legal business name (though the help text says, "If there isn't [a business name] then please enter something that best represents what you are doing"). You need to enter a phone number and mailing address as well. Then you go to *another* form for some additional questions about the business. The next page asks for the last four digits of your Social Security number and your date of birth. (Non-U.S. residents, I assume you have something different here.)

Finally, you are able to submit the form and create the account.

The resulting page says, "Welcome to your PayPal Business Account." You'll need to perform additional setup activities to actually take payments, but at this point we are ready to create a test account for our theater application.

To test PayPal in development mode, we need to create what PayPal calls "sandbox accounts," and as far as I can tell, how to do so is completely opaque from the business account page. But once you are logged into your business account, point your web browser to the Sandbox Test Accounts page.[2] You should see two automatically created sandbox accounts, as shown in the figure on page 8.



Sandbox Test Accounts

Questions? Check out the Testing Guide. Non-US developers should read our FAQ.

Total records: 2

Email Address	Type	Country	Date Created
moneybook-facilitator@noelrappin.com	BUSINESS	US	05 Feb 2016
moneybook-buyer@noelrappin.com	PERSONAL	US	05 Feb 2016

2. https://developer.paypal.com/developer/accounts

For the purposes of this book, we are using two accounts to test our system. One account is our fake business account (moneybook-facilitator@noelrappin.com), and the other account is our fake buyer (moneybook-buyer@noelrappin.com). These accounts are based on the email you used to set up the business account, and you'll start with one business account and one personal account. You can create more from this page if you want.

With our account set up, we can use it as a source of the API keys we need, as well as use it to verify and test our transactions.

Accepting PayPal Transactions

As we build our PayPal transaction manager, we're trying to solve two somewhat separate problems. The immediate problem, of course, is setting up our application to handle PayPal transactions. The larger problem is how to gracefully handle changes to our requirements that break the assumptions the code is already making. In this case, the code we've created so far in this book assumes that all payments use Stripe and take Stripe tokens. As we go forward, we need to be aware that the way we add PayPal into our code base could have important consequences for how easy or difficult it is to make future changes to our business logic.

To accept PayPal for payments, several points in our code need to be adjusted:

- Our shopping cart form needs to allow users to select PayPal as a payment option, and know that if users select PayPal, they don't need to fill in credit card information.
- Once our application receives a user request for PayPal, our application needs to branch off into a PayPal-specific workflow, rather than the Stripe workflow we already have set up.
- As part of the PayPal workflow, our application needs to not complete the entire transaction, but instead tell PayPal about the impending sale. In doing so, the PayPal gem will communicate back to the PayPal REST API and give us a one-time URL associated with the new payment.
- Once PayPal returns a one-time URL, our application needs to redirect users to the URL, where they authenticate in PayPal, choose their payment method, and jump through whatever other hoops PayPal asks them to.
- When PayPal sends a message back to our servers letting us know that authentication is successful, our application needs to be able to finish processing the payment.

From our point of view, the fact that the workflow depends on PayPal sending a message back to our servers makes working with PayPal in test or development mode kind of awkward. In particular, it's difficult to completely run an end-to-end test, and it's also hard to completely test the PayPal workflow when you need PayPal to call back to localhost:3000. We'll go over the code to interact with PayPal first, then talk about how we run this in a developer environment and test.

Setup and Configuration

PayPal has, I think, at least three separate APIs—it's hard to tell because the documentation makes no particular attempt to give a useful overview. In any case, we'll be using the REST version of the API, and the associated official Ruby gem provided by PayPal,[3] which we need to add to our Gemfile. The REST version is the newest version, and the one most likely to be kept up in the future, although it still has some limitations relative to the older APIs.

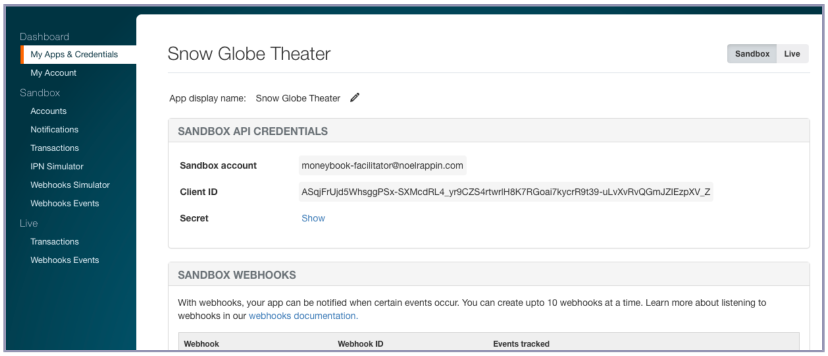
```
gem "paypal-sdk-rest"
```

The gem comes with a handy install script:

```
$ rails g paypal:sdk:install
                        create  config/paypal.yml
      create  config/initializers/paypal.rb
```

Because we're using the REST API, we'll need to create a REST API app to get credentials. Go to https://developer.paypal.com/developer/applications and then click "Create App" under "REST API apps." Give the app a name—we'll use "Snow Globe Theater" and our development merchant account we created in the previous section. Click "Create App" again, and we now have a page with Sandbox API credentials, including a client ID and secret, which we'll need in a moment. In the upper-right corner is a "Sandbox/Live" toggle similar to Stripe's (shown in the following figure) that we can use when we go live.

3. https://github.com/paypal/PayPal-Ruby-SDK



Snow Globe Theater

App display name: Snow Globe Theater

SANDBOX API CREDENTIALS

Sandbox account	moneybook-facilitator@noelrappin.com
Client ID	ASqjFrUjd5WhsggPSx-SXMcdRL4_yr9CZS4rtwrlH8K7RGoai7kycrR9t39-uLvXvRvQGmJZIEzpXV_Z
Secret	Show

SANDBOX WEBHOOKS

With webhooks, your app can be notified when certain events occur. You can create upto 10 webhooks at a time. Learn more about listening to webhooks in our webhooks documentation.

Webhook	Webhook ID	Events tracked

We now need to add those credentials to our application using the paypal.yml file. We'll use a trick similar to what we did to add the Stripe credentials. We'll add the actual client_id and client_secret to the .env file and then add them to the secrets.yml file:

paypal/01/config/secrets.yml
```
development: &default
  admin_name: First User
  admin_email: user@example.com
  admin_password: changeme
  domain_name: example.com
  secret_key_base: fc65272c27199652b936c28264bf770f385243f00d91b69f7424ca298466
  stripe_publishable_key: <%= ENV["STRIPE_PUBLISHABLE_KEY"] %>
  stripe_secret_key: <%= ENV["STRIPE_SECRET_KEY"] %>
  paypal_client_id: <%= ENV["PAYPAL_CLIENT_ID"] %>
  paypal_client_secret: <%= ENV["PAYPAL_CLIENT_SECRET"] %>
  host_name: "6e0fd751.ngrok.com"

test:
  <<: *default
  secret_key_base: 8489bb21c9497da574dcb42fea4e15d089606d196e367f6f4e166acb281d
```

As you can see in the previous code, we've also added a host_name field, which will come in handy later on when we need to give PayPal a callback URL.

Next we need to add the environment variables to the file PayPal uses, the paypal.yml file:

paypal/01/config/paypal.yml
```
development:
  mode: sandbox
  client_id: <%= ENV["PAYPAL_CLIENT_ID"] %>
  client_secret: <%= ENV["PAYPAL_CLIENT_SECRET"] %>
```

In the paypal.yml configuration file, the same configuration is duplicated in the test section. We might want to keep this out of source control, similar to the secrets.yml file, so that the staging and production versions can be set directly.

View Update

Our shopping cart view now needs to change to allow users to pick PayPal as a payment option. As usual, the following code won't win any design awards, but it's functional:

paypal/01/app/views/shopping_carts/show.html.slim

```
h2 Checkout

h3 Payment Options

= form_tag(payments_path, class: "form-inline", id: "payment-form") do

  .paypal
    img(src="https://www.paypal.com/en_US/i/logo/PayPal_mark_37x23.gif"
        align="left" style="margin-right:7px;")
    span(style="font-size:11px; font-family: Arial, Verdana")
      | The safer, easier way to pay.
    = radio_button_tag(:payment_type,
        :paypal, false,
        class: "payment-type-radio", id: "paypal_radio")
  .credit_card
    | Credit Card
    = radio_button_tag(:payment_type,
        :credit, true,
        class: "payment-type-radio", id: "credit_radio")

    = hidden_field_tag(:purchase_amount_cents, @cart.total_cost.cents)

    #credit-card-info
      h3 Credit Card Info
```

The rest of the credit card information stays as before, subordinate to the #credit-card-info element, which we've added to allow us to easily show or hide the entire credit card portion of the form.

The use of the specific PayPal logo GIF and tagline is required by PayPal if you are using ExpressCheckout.[4] Otherwise, this code defines a standard set of radio buttons setting a form variable called payment_type.

We need add a tiny sprinkle of JavaScript to make sure the credit card information really does hide. This is the JavaScript class formerly known as StripeForm, with the naming updated to better reflect its new function:

paypal/01/app/assets/javascripts/purchases_cart.es6

```
class PaymentFormHandler {

  constructor() {
    this.checkoutForm = new CheckoutForm()
    this.initSubmitHandler()
```

4. https://www.paypal.com/webscr?cmd=xpt/Merchant/merchant/ExpressCheckoutButtonCode-outside

```
    this.initPaymentTypeHandler()
  }

  initSubmitHandler() {
    this.checkoutForm.form().submit((event) => {
      if (!this.checkoutForm.isPayPal()) {
        this.handleSubmit(event)
      }
    })
  }

  initPaymentTypeHandler() {
    this.checkoutForm.paymentTypeRadio().click(() => {
      this.checkoutForm.setCreditCardVisibility()
    })
  }

  handleSubmit(event) {
    event.preventDefault()
    if (this.checkoutForm.isButtonDisabled()) {
      return false
    }
    this.checkoutForm.disableButton()
    Stripe.card.createToken(this.checkoutForm.form(), TokenHandler.handle)
    return false
  }
}

$(() => new PaymentFormHandler())
```

With this code we're adding a new handler to fire on a click to one of the radio buttons and passing the handler to the checkoutForm. We also need to make sure that the client-side call to the Stripe API doesn't happen if the customer selects the PayPal option. You'll also need to add a few methods to the CheckoutForm class, which actually interacts with the DOM:

paypal/01/app/assets/javascripts/purchases_cart.es6

```
paymentTypeRadio() { return $(".payment-type-radio") }

selectedPaymentType() { return $("input[name=payment_type]:checked").val() }

creditCardForm() { return $("#credit-card-info") }

isPayPal() { return this.selectedPaymentType() === "paypal" }

setCreditCardVisibility() {
  this.creditCardForm().toggleClass("hidden", this.isPayPal())
}
```

The customer can now select PayPal as a payment option on our form.