🔥 Full source code included

Learn how to build and launch a real-world SaaS application in just a few weeks.

Building SaaS with Laravel

By Max Kostinevich

Introduction

In this book you'll learn how to design, build and launch a real-world SaaS application using Laravel framework. This book will be useful to solopreneurs, bootstrappers and indie makers who wanted to launch their SaaS business.

I'm going to explain step-by-step each stage of project development - from designing wireframes to deployment.

You'll learn how to:

- Convert your idea to wireframes
- Design database
- Convert HTML template to Laravel views
- Organize project routes
- Accept payments through Stripe and collect fees
- Distribute payments using Stripe Connect
- Send email notifications
- Install and use Laravel Horizon to manage queues
- Work with 3rd-party API to convert currencies
- Build master administration panel to manage the project
- Deploy the application

I also included the list of tips and advices which will be useful to you when you decide to launch you SaaS.

Learn more about this book at maxkostinevich.com/books/laravel-saas

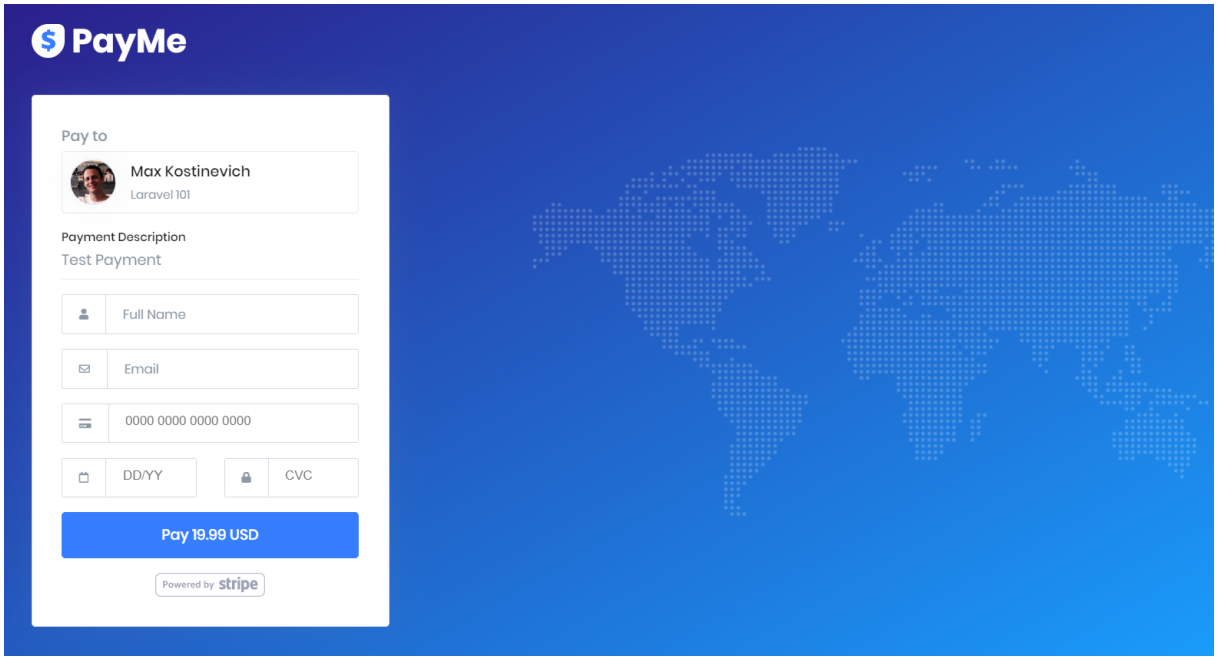
What we are going to build

The project we're going to develop in this book is called **PayMe**.

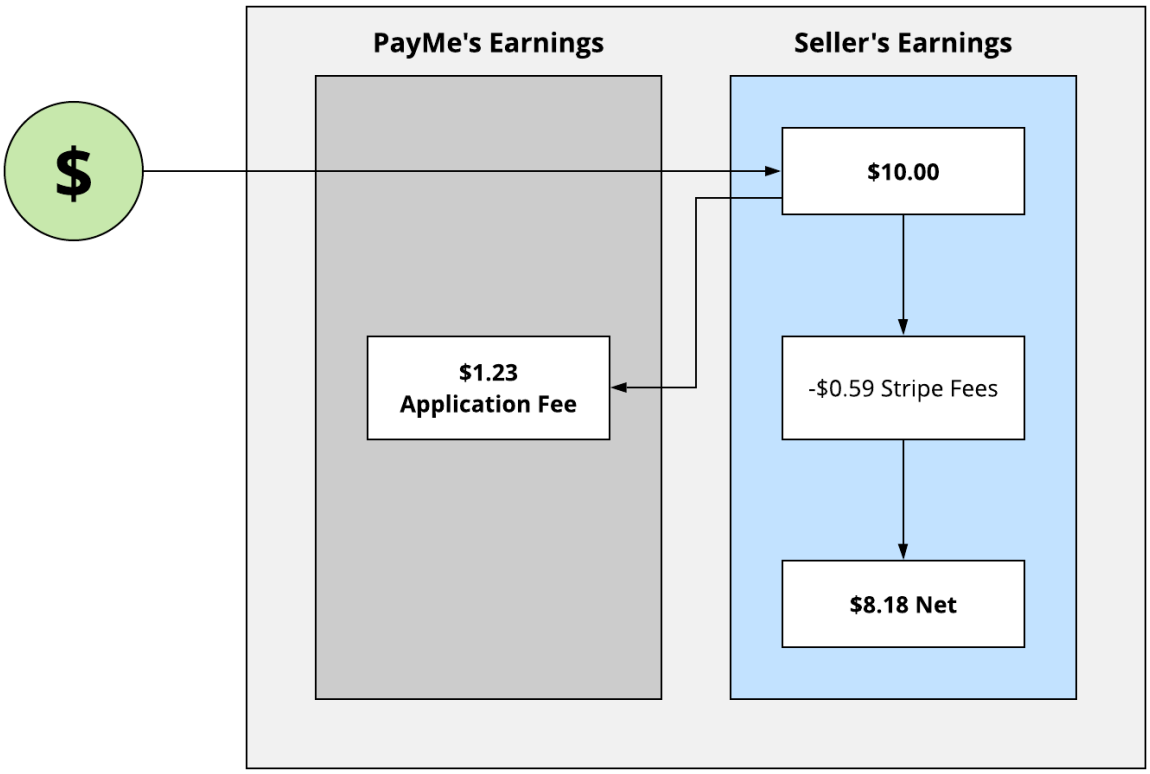
PayMe is a checkout payment solution to accept payments online for freelancers, digital artists and small agencies.

The idea of the project is pretty simple: the Seller (e.g. freelance designer) registers on PayMe, connects their Stripe Account, creates a payment form where he can enter service description, amount, and choose a currency. When the payment form is created, it becomes available via unique URL, which then could be shared with the Customer. When the Customer pays via the payment form, the paid amount is automatically transferred to Seller's Stripe Account.

You may see an example of payment form on the following screenshot:



As our main goal is to validate our idea and build MVP (minimum-viable-product) as soon as possible and spend as less time as we can, we're not going to implement any subscription-based features. Instead of this, we're going to collect some fee (e.g. percent or fixed price) on each payment made through our app, which will be deducted automatically from the Seller's account. You can see the payment flow on the diagram below:



On the example above, the Customer paid $10 to the Seller. From this amount we collected $1.23 as a fee for our application. Additional $0.59 have been deducted by Stripe as their own fee. As a result, the seller earned $8.18 net amount ($10 - $1.23 - $0.59 = $8.18).

PayMe demo could be found using the links below:

- Live demo
- Sample payment form

Prerequisites

The project we are going to develop in this book is designed for beginner developers, however I assume that you're familiar with Laravel framework and know how to work with Controllers, Models, and run Artisan Commands.

We're going to use **Laravel 5.8**.

If you're absolutely new to Laravel, I'd recommend you to check some video courses first, for example:

- Laracasts
- Codecourse
- Udemy

As we're a going to use Stripe Connect and Stripe API to handle payments and payouts, you'll need to have a Stripe Account. Currently Stripe is available in 34+ countries, so if you're living in a country where Stripe isn't yet supported - don't worry, you still will be able to create development account with Stripe.

Getting the source code

The source code of the **PayMe** project is included to your purchase. Be sure to carefully read the *README* for installation instructions.

Disclaimer

The source code is provided for learning purposes without warranty of any kind. You're free to use provided source code (or any parts of it) as you'd like. Redistribution (i.e. reselling) or sharing (e.g. via public GitHub repository) is not allowed without prior written permission.

Working with the source code

For your convenience, all important steps in the source code are marked by git tags.

You can easily switch between specific tags by using *git checkout* command. For example, to switch to tag *step-2.1*, just type in your terminal:

```
git checkout step-2.1
```

To list all available tags, just type the following command:

```
git tag -n
```

You'll see the list of all available tags:

```
➤ src (master -> origin) X git tag -n
step-1.0        Provision new Laravel app
step-1.1        Initial setup
step-2.0        Add guzzle
step-2.1        Add assets compilation and static pages
step-2.2        Add auth pages
step-3.0        Update email verification page
step-3.1        Add Dashboard
step-3.2        Add user settings page
step-3.3        Add payments form crud
step-3.4        Add payment form
step-3.5        Accepting payments
step-3.6        Managing payments
step-3.7        Add stats to the dashboard
step-3.8        Notifications
step-3.9        Small changes
step-4.0        Master Admin
```

During the book you may see the following notes:

> Related tag: step-x.x

That means that the section or chapter has a related tag in the app source

code.

Workspace setup

There are no strict requirements which tools to use for local development, you will be fine with Homestead or Valet. I'd recommend to avoid XAMPP, WAMP and similar software.

Personally, I use the following setup:

- Docker with installed Nginx, PHP 7, Redis and MySQL
- Ngrok to expose local server to internet over secure tunnel
- PHPStorm as my main IDE
- Notepad++ for quick edits
- Cmder as my main command-line tool

If you're new to Docker and haven't worked with it before, I recorded a short video explaining how to get started with Docker for Laravel development, you can watch it here.

I also created a ready-to-go Docker template, which is available on the Github.

Also, if you haven't worked with Ngrok before, I would recommend you to try it out! Ngrok allows you expose your local server to the internet over a secure tunnel. This tool is super-useful for testing OAuth integrations, webhooks, 3rd-party API calls and so on.

About the author

Max Kostinevich is a solutions consultant and web-developer. Max have over 10 years of extensive experience in eCommerce and SaaS consulting and de-

velopment, and have worked with dozens of companies worldwide, including multinational companies on the Inc. 5000.

Contact the author

If you have any questions, ideas, suggestions, or want to report an error, please email me at hello@maxkostinevich.com

For most recent updates, please follow me on Twitter: maxkostinevich

Chapter 1. Planning our app

Detailed plan is the key to successful results. However, when planning the MVP, it's important to keep your requirements list as simple as possible, just because you can easily got drown in all your notes, ideas, and wanted features.

So first, we need to clearly define what our project does and extract most essential features of our project and write them down in project specification file. Based on this file we can create wireframes to get better idea of how our project will looks like.

Designing wireframes

So what the wireframing is? Wireframing is the process of transforming app spec into graphicc representation at the structural level. You may think about wireframing as about low-level design where you focus on features and layout instead of high-level details. Wireframes helps us to get better idea of how our project may looks like, how all features will work together and how long the development process may take before we even write a single line of code.

When designing wireframes it's important to not to focus too much on small details. It's a good idea to ask yourself - how it may looks like and how it should work?'

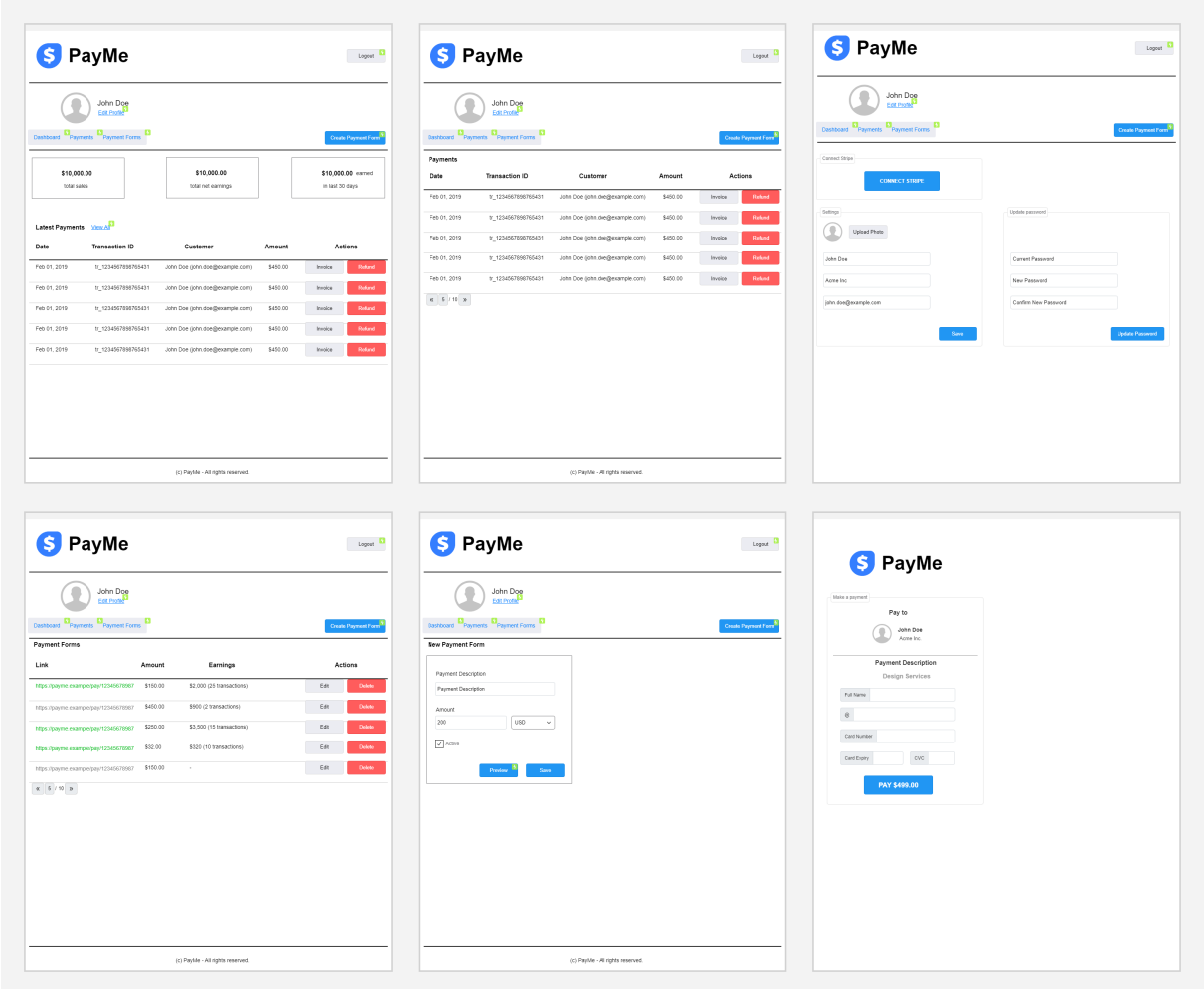
Usually wireframing takes a few iterations before we get clear idea of what we are going to build.

I recommend to make some initial wireframing on paper, as this is most quickest way to do this. Then you may create digital copy of your wireframes. There are several tools which I use for wireframing:

- Sneakpeekit - just a print template for wireframes on the paper

- Balsamiq - a great tool allowing to quickly create wireframes and mock-ups
- UX-App - a web-based application allowing to create interactive wireframes

For PayMe I created interactive wireframes using UX App, see the image below. You can also find these wireframes here.



This is the final version of wireframes for PayMe. Before creating these wireframes in UX App, I spent some time to make a few versions on paper.

After we have our wireframes on file, we can proceed to the next step - designing

database.

Designing database

A good way to design database is to ask yourself the following questions:

- Which models may I need?
- Which attributes each models may need?
- How these models should be relate to one another?

It's important to remember that our database structure may change during the development process, and that's totally fine. At that stage our main goal - is to define starting point from which we can start building something.

For PayMe we can define 3 main models:

- Users (Sellers)
- Forms
- Payments (Sales)

In additional to default Laravel attributes, for each User (Seller) we'll need to store their connected Stripe Account ID, their profile picture, and company.

For Payment Forms we'll need to store related User ID, UID (which will be used in unique URL), service description, amount and currency.

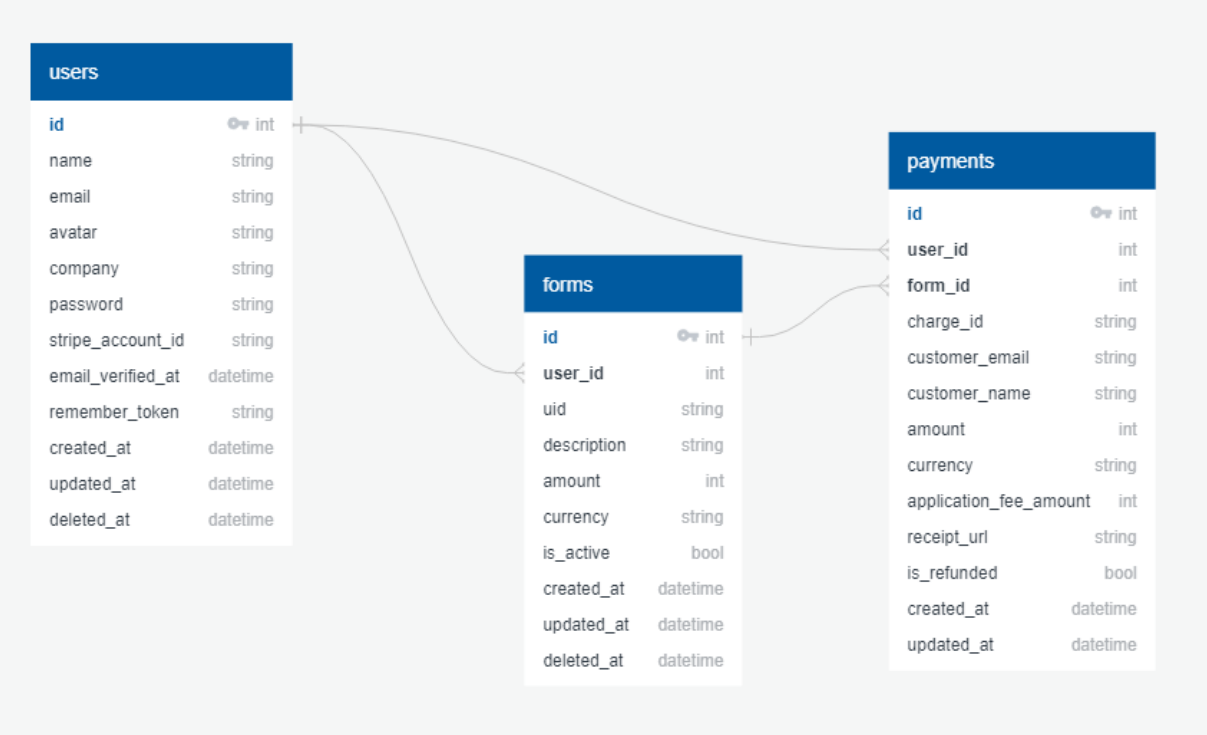
For Payments we'll need to store related Form ID, charge ID (Transaction ID from Stripe), customer name and email, application fee we collected for that payment, and receipt URL (which is automatically generated by Stripe).

We can also assume the following conditions:

- Each User can have multiple Forms
- Each User can have multiple Payments

- Each Form can have multiple Payments
- Each Form relates to only one User
- Each Payment relates to only one Form
- Each Payment relates to only one User (the same as the owner of the Form)
- The Payment could be refunded to the customer

As you can see on the image below, the database structure is pretty simple:



Please, note - payment model may not have a user_id, as we can get this ID from the related Form model. However I decided to store related user_id in Payments model too, as it will make it much easier to us to calculate statistics and get the history of user payments.

Useful tools to design the database:

- QuickDatabaseDiagrams
- DBDiagram

- Lucidchart
- Draw.io

In the next chapter we'll start building our app.

Chapter 2. Building our app

In this chapter we're going to build our application from scratch.

After we have our wireframes and database design finished, we can start building our app.

There are no strict rules on how exactly to build the app, personally I prefer the following sequence:

1. Create wireframes.
2. Design the database structure.
3. Prepare application plain HTML templates for each important page/layout.
4. Install fresh Laravel application.
5. Convert plain HTML templates to Laravel views.
6. Start building main functionality.

For this book I will leave the creation of plain HTML templates behind the scenes, as this is not the main focus of this book and the process is different for each project. You may find all plain HTML templates in the source code attached to the book.

Creating new app

> Related tag: step-1.x

I assume that you already prepared your local environment. So let's create a new Laravel application by running the following command in our console:

```
laravel new payme
```

Then let's update our `.env` file and update database credentials and `APP_URL` variable.

As I use Ngrok, my `APP_URL` variable looks as following:

```
APP_URL=https://payme.ngrok.io
```

I always force `https` protocol in my apps. To do this, we'll just need to add `\URL::forceScheme('https');` to `boot()` method in our `app/Providers/AppServiceProvider.php`:

```
class AppServiceProvider extends ServiceProvider
{

    //...

    public function boot()
    {
        // Force SSL
        \URL::forceScheme('https');
    }
}
```

As our app requires user registration and authentication, we need to enable Laravel's built-in authentication feature. To do this we'll need just run `php artisan make:auth` and `php artisan migrate` in our console.

After we enabled authentication, we can also enable Laravel's built-in email verification feature. This feature will force newly registered users to verify their email addresses. To do this, we'll need to make just a few things:

1. Make sure that our `User` model implements `Illuminate\Contracts\Auth\MustVerifyEmail` contract:

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable implements MustVerifyEmail
{
    use Notifiable;

    // ...
}
```

2. Make sure that `users` table have `email_verified_at` column (it's included by default).
3. After that we can pass the `verify` option to the `Auth::routes` method to activate email verification routes:

```
Auth::routes(['verify' => true]);
```

For email testing on local development I use Mailhog which catches all email sent by our app to it's own local tiny SMTP server with a web-based UI. Mailhog is included by default to my Docker template. To use Mailhog we'll need to update email settings in our `.env` file:

```
MAIL_DRIVER=smtp
MAIL_HOST=mailhog
MAIL_PORT=1025
```

Next, we can copy our HTML templates into `/resources/views/_HTML` folder. This step is not necessary and it's just my personal preference as I'd like to

keep these files at one place. After we convert our HTML templates to Laravel views, we'll remove this folder.

At this stage we can also install some libraries which we'll use later. I usually install Guzzle, which helps us to make HTTP requests (e.g. API calls). To install this library, just run the following command in your console:

```
composer require guzzlehttp/guzzle
```

Another useful package I usually install is a Laravel Horizon. Horizon helps us to manage our Redis queues by providing web-based UI. If you're not familiar with queues or Horizon - don't worry' we'll talk about it a little bit later. To install horizon just run the following command in your console:

```
composer require laravel/horizon
```

After that we can install Horizon:

```
php artisan horizon:install
```

And create `failed_jobs` table (in this table Laravel will store all information about our failed jobs) using the following command:

```
php artisan queue:failed-table

php artisan migrate
```

In order to use Horizon and Redis, we need update change `QUEUE_CONNECTION` variable in our `.env` file:

```
QUEUE_CONNECTION=redis
```

That's all, at the next step we'll convert our HTML templates into Laravel views and customize our authentication pages.

Deploying our app

At first attempt I wanted to make this chapter in the form of a step-by-step tutorial about application deployment. However, as UI of some tools may change and deployment process might be a little bit different depending on your preferred workflow, I decided to give a quick overview of popular deployment options and provice you a few useful link where you can learn more.

There are a number of tools and deployment options we can use to deploy our application. For example, some of popular choices are:

- Laravel Forge - Server management and deployments via Git;
- Envoyer - Zero-downtime deployments;
- Ploi.io - Server management, zero-downtime deployments and auto-backups;
- PaaS and managed hosting (e.g. Heroku and Cloudways);
- Containerized hosting and deployment (e.g. Docker);
- Serverless deployment through Laravel Vapor;

Personally, I prefer to use DigitalOcean for hosting, Forge for server management and Envoyer for zero-downtime deployments. If a short period of downtime isn't crucial to your app, you may use Forge without Envoyer for deployment.

If you want to learn more about Forge and Envoyer, I would recommend you to check these awesome series on Laracasts:

- Learn Forge
- Learn Envoyer

Another great tool for deployment and server management is Ploi, created by Dennis Smink. Ploi includes all features Forge and Envoyer has, and cost a

little bit less than Forge and Envoyer in total.

If you're beginner and do not know a lot about server management, you may take a look at Cloudways, they offer easy-to-use managed hosting platform for Laravel apps.

Laravel Vapor - is entire tool in Laravel eco-system, it allows you to host Laravel application in Serverless infrastructure. Serverless approach have it's own pros and cons, and out of scope of this book.

Regardless of tools you choose, deployment process includes the following steps:

- Setup git repository;
- Provision new server;
- Create new database;
- Configure DNS records and setup SSL certificates;
- Prepare deployment recipe;
- Get all required API keys and prepare `.env` file for production;
- Create daemon and queue worker (if your app uses queues like beanstalk or redis);

It's also a good idea to keep deployment instructions in `Readme` file of your project.

When going to production, do not forget to setup auto-backups. I also recommend to setup some app monitoring tools to be notified when any error occur. For example, you may use Bugsnag or Sentry, both tools provide a free plan.

Chapter 3. Useful tips

In this chapter I'll share my thoughts and useful tips about making and running Sofware-as-a-Service applications.

Types of SaaS

I would distinct two different types of SaaS: - **Standalone SaaS** - This is type of SaaS, which is not depending on any other service.

- **Extensions for other products** - This type of SaaS depends on another product or service. For example - Shopify Apps, Apps for Quickbooks, Apps for Salesforce, etc. (e.g. Shopify Apps, Apps for Quickbooks, etc)

Both of types have their own pros and cons. For example, making a standalone SaaS gives you more freedom and flexibility, as you're not depending on other product. On other side, it could be harder to get your first users of standalone SaaS, as many established products have their own marketplace where developers like you can publish extensions (plugins or apps) for that big product. Making an extension for established product usually requires you to follow some design guidelines, rules, and marketplace terms. However, it gives you an access to existing and loyal customers who might be happy to use your extension (and pay you!).

Thoughts on pricing and customer retention

There are several tips about pricing I would like to share:

- If you're planning to charge pretty small amount for your SaaS (e.g. $3/mo), it's a good idea to offer yearly package, as it will gives you

more money on inital period, which you can re-invest to marketing.
- To get more money on inital period, you may also offer a lifetime deal to your first X customers.
- Providing a reasonable trial period may increase your conversion rate. In most cases, 7-14 days is enough to try a product and make a decision.
- Be careful with the free plan, as it may increase amount of support requests you need to handle.
- Be proactive with your first customers, try to follow-up with each sign-up and get a quick feedback about your product.

If you're willing to learn more about different aspects of running SaaS, I would recommend to join SaaS Club by Omer Khan. SaaS Club provides a lot of useful resources, such as group coaching sessions, a huge content library, access to private community and expert master classes.

Thoughts on legal aspects

If you're active on Twitter, you probably heard about Reilly Chase and his story. So if you're working on your project while keeping your daily job, it's a good idea to get a legal advice and consult your lawyer to make sure you do not violate the terms of your employment contract.

You may also consider to create a new company for your new project. Depending on your requirements, there are several services allowing you to open company remotely. Most of popular options are:

- Stripe Atlas - for US-based company;
- e-Residency - for Estonian company;

Before opening a company, it's a good idea to consult with your accountant regarding all important questions (e.g. Tax/VAT handling, local law compliance,

etc).

Recommended books

There are a lot of books about making and running software projects, I'll just share my Top-3:

- Rework by Jason Fried and David Heinemeier Hansson
- Making ideas happen by Scott Belsky
- Start Small, Stay Small by Rob Walling

Afterword

Thank you for reading this book, I hope you found it useful!

If you have any questions, found a typo or just want to provide a feedback, feel free to shoot me a tweet at maxkostinevich or email me at hello@maxkostinevich.com

— *Max Kostinevich*