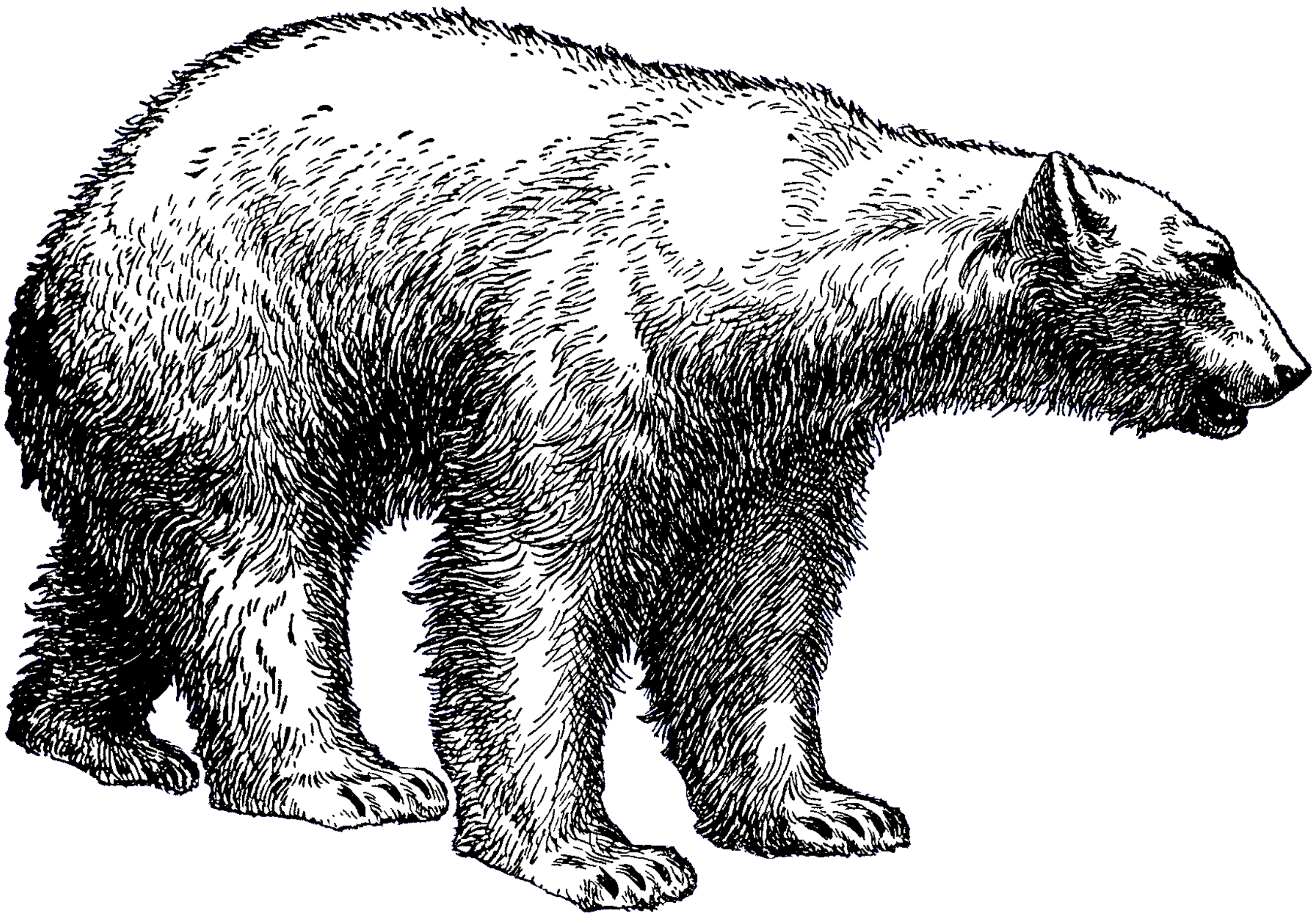
Build your own full stack SaaS product in days!



Full Stack SaaS Product Cookbook

From Soup to Nuts

Christopher J. Frewin

Full Stack Craft LLC™

Full Stack SaaS Product Cookbook

From Soup to Nuts - Create a Profitable SaaS Product as a Solo Developer

Christopher J. Frewin

https://chrisfrew.in

May 6, 2021

Contents

Foreword

If I have seen further it is by standing on the shoulders of Giants.

(Isaac Newtown, 1675)

Book Introduction

SaaS Products. Such a massively overused buzzword in today's internet culture.

Everyone seems to *want* a profitable SaaS product, but rarely is a complete in-depth discussion taken on what exactly that entails. Typically, the bear minimum for a circa 2020s SaaS product includes the following:

- User authentication, authorization, and management
- A custom backend API
- A nice looking and easy-to-use UI
- Email flow and service for welcoming new customers, password resets, etc.
- Logging and alerts throughout the entire stack
- Last and most importantly, *the value of the product itself*.

These design minutia and decisions don't fit into our 280 character Tweet world. A huge majority of the resulting noise online surround SaaS development therefore devolves into the incessant framework vs. framework or language vs. language battles - or worse - paraphrased guru or meme-like slogans that have nothing to do with actually putting in the hard work to build the product itself.

In this book, I cut through all that noise, describing in extreme detail, step-by-step, from frontend to backend, with all configuration in between, how to build all parts of your next profitable SaaS product. The final product will be highly maintainable while at the same time highly customizable. After 10+ years of building my own solo side products, wasting literally *thousands* of hours making countless of mistakes, I've finally arrived at an extensively reusable, fast, and very lean stack that works for solo developers. This book is the refined culmination and best practices of my decade long experience.

Who this Book is For

This book is targeted at solo developers, creators, and makers who want to have full control over their own SaaS Products and know the inner workings at all parts of the stack. It's for those who want to ultimately automate nearly all aspects their product or service with small exceptions like communicating with customers, or personal interactions promoting the product (all of which are *extremely* important, as I'll get to in later sections of the book.)

If you are a solo software developer looking to move into the SaaS landscape and not waste time asking yourself and answering complex questions like:

- What database to use
- What authentication or authorization service to use
- What type of API to use
- How to implement full stack logging, monitoring, and alerts through the entire application
- How to create and automate frontend and backend builds with CI and CD

Then look no further. This book will provide answers to all those questions and more with full code solutions. Note that this book *is* highly opinionated. I do use specific frameworks and services throughout the entirety of the book. But, like I've said, after searching for 10 years for the holy grail of SaaS product generators, I believe I've found it, at least for web-based SaaS products. If you are looking for more theoretical or fundamental-minded books on building apps, this book is not for you, and there are plenty of those out there.

Book GitHub Organization and Repository

✅ Book Code and Source ✅

I have created an **entire GitHub Organization for this book**. It includes all milestone repositories as well as **the repository for this book's source**! (Too meta, right?). While I encourage you to understand and write your own code as you follow along, I also totally understand if you've missed a section or something small and clone the code just to see how it works. Enjoy!

Highlight Boxes

Throughout this book, you'll encounter a variety of highlighted boxes, which are colored coded to provide specific types of information. Examples are as follows:

✅ Green Highlight Boxes ✅

Green highlight boxes have green check emojis and will offer links to various repositories which act as milestones in time of the code base we will be building together. In fact, you've already seen one, just in the above section.

ℹ️ Blue Highlight Boxes ℹ️

Blue highlight boxes have blue information emojis and are more of aside details about my software opinions. They aren't essentially to the workflow of building the product, but offer some nice insights (in my opinion) into the careful thought process I put into my stack.

⚠️ Yellow Highlight Boxes ⚠️

Yellow highlight boxes have yellow warning emojis are warnings of what could go wrong with a particular piece of code, the stack, or a methodology. Take note of these far and few between warning highlights!

Are You Ready?

That should be all you need to know before diving into this book.

I'm proud of how this book came out, and I frequently reap the rewards of my own labor, using it as a handbook myself for each new SaaS product I build. I hope that I've piqued your interest, and that you'll join me on this full stack adventure!

- Christopher Frewin

Feldkirch, Austria, April 2021

1

The Product

It's really rare for people to have a successful start-up in this industry without a breakthrough product. I'll take it a step further. It has to be a radical product. It has to be something where, when people look at it, at first they say, 'I don't get it, I don't understand it. I think it's too weird, I think it's too unusual.

(Marc Andreessen)

1.1 The Product You'll Be Building

The product you'll be making in this book is a product I call 'ReduxPlate'. It's a real, full fledged, profit generating product I own, currently live at **https://reduxplate.com**. It's a $60 / year subscription service that builds your entire Redux code boilerplate from the state of your application alone!

For those who use Redux with TypeScript, you may know how much code needs to be written after adding just one new part of state. (Hint: it's even more than the boilerplate required with vanilla JavaScript!) I had long wanted to build a SaaS product like this, and the motivation to write the book finally spurred me to build it, since it is a good example for a full stack SaaS product.

Don't worry, we'll get into the nitty-gritty of how it actually works, writing all the code step-by-step throughout this book. But more on those details will come later.

My Challenge to You

If you're motivated, I suggest to copying only the *nature* of each of the tutorials throughout the book, modifying code where it is needed, ultimately coming out with your own SaaS product by the end of the book. This is especially useful in the "recipe" sections in the second half of the book - they are actually product agnostic, and should be able to be included for *any* type of SaaS Product.

It's also of course completely acceptable to work through the tutorials exactly step-by-step - you'll come out with an exact clone of what ReduxPlate looks like today! Even if you take this mimicry style of workflow, at the end, you'll still have this book as a reference and can do it all again, already knowing all the steps, for your next profitable SaaS product!

2

The Frontend - Getting Started

You've got to start with the customer experience and work backwards to the technology.

(Steve Jobs, 1997)

2.1 Introduction to the Frontend

Chapter Objectives

- Some notes on naming conventions you'll see throughout the book
- A few of my own personal style techniques when writing frontend code with React and TypeScript
- Define the framework and tool versions used on the frontend

We're going to start off building the frontend, as that side of the stack gives us some immediate visual feedback, and as Steve's quote above touts, we can then work backwards to figure out what sort of technologies we'll need to complete our

SaaS product.

A Word On Naming Conventions

As mentioned in Section I, I'll be going step by step through what I did to build ReduxPlate (**https://reduxplate.com**) Indeed, this book was written *while* I built ReduxPlate! The repositories we'll create for the project will key into the naming convention I will use throughout the book. In fact, the only two repositories we'll need for our entire complete SaaS product will have the following names:

`reduxplate.com` (For the frontend repository, AKA the client. In the case of a web app, which ReduxPlate is, I typically choose the root domain name for the the name of the repository.)

`ReduxPlateApi` (For the backend repository, AKA the API. This is standard capitalized camel case notation that is standard in C#, and will make our namespaces play nice in our .NET code.)

So, we will see this `reduxplate` or `ReduxPlate` moniker over and over again throughout this book. In the case of things like secrets and constants, we will see this moniker used instead in all caps and with an underscore as a space, i.e. `REDUX_PLATE`. In some cases for readability, I will use it lower case with a hyphen, .i.e. `redux-plate`.

If you are going the option of tailoring each step in this book to your own project, whenever you see `reduxplate` or `ReduxPlate`, take it as a signal to rename variables with those monikers to your own product's name. Take a deep breath, there's going to be **a lot** of them.

Some Notes on My Frontend Style

I also have developed my own specific code style. Some of my most important rules, though not all of them, include:

- Avoid `var` and `let` wherever possible; this should almost always be possible.
- Always de-structure `props`
- Keep as much logic out of components as possible - components should generally be only for rendering jsx- style markup
- Use TypeScript
- Use **Redux** with **Redux Toolkit**

Frontend Frameworks and Tools Versioning

On the frontend, I will be using these versions of the following tools and frameworks:

- npm 7.6.13
- Node 14.16.0
- Gatsby 3.0.0
- React (and React DOM) 17.0.2
- Bootstrap 5.0
- TypeScript 4.2

Installation and setup of all these frameworks, including code editor plugins and so on are outside of the scope of the book (excluding Ubuntu 20.04 - I will be going over in detail how to start a Ubuntu 20.04 box with Digital Ocean). There are plenty of awesome resources online for everything else, and for the packages themselves, **it's always best to start with their respective documentation first.**

Everything still okay? Let's finally start building this product!

2.2 Bootstrap the Frontend With Gatsby V3

Chapter Objectives

- Bootstrap the frontend with Gatsby's official starter, `gatsby-starter-default`

With some housekeeping done, let's jump right into code. We'll start by cloning one of the official Gatsby starters, in fact, the default one, `gatsby-starter-default`, and I'll name my project `reduxplate.com`. This will also be the folder that Gatsby creates for us.

So, you'll also need to install Gatsby if you don't have it installed yet:

```
npm install gatsby
```

Then, the command to create our frontend Gatsby project is:

```
gatsby new reduxplate.com
https://github.com/gatsbyjs/gatsby-starter-default
```

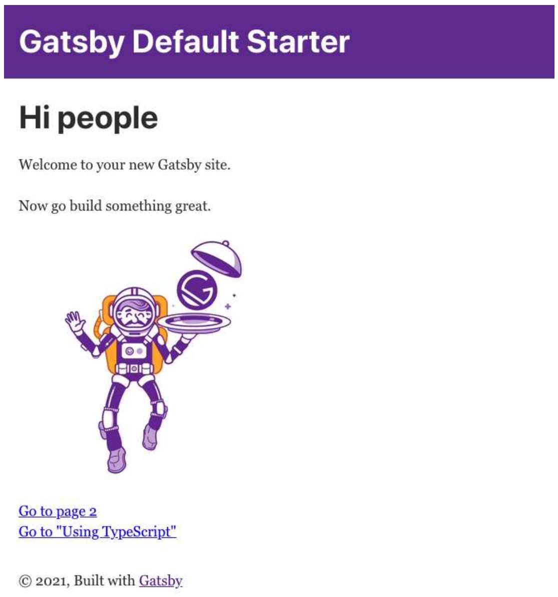
We’ll cd into the directory:

```
cd reduxplate.com
```

and get started with the `develop` command:

```
npm run develop
```

You should see the Gatsby starter spool up at `localhost:8000` in your browser, or a different port if you already had something running at 8000:



Gatsby Default Starter

Hi people

Welcome to your new Gatsby site.

Now go build something great.

Go to page 2
Go to "Using TypeScript"

© 2021, Built with Gatsby

Figure 2.1: Screenshot of the unmodified default Gatsby starter.

2.3 Clean Up the Gatsby Default Starter

Let’s now do some simple house cleaning on this project Gatsby has just scaffolded for us. While doing all of these steps, you should be able to keep running the site in development mode, and see the warnings provided in the terminal. The step by step process to get down to a no-fluff skeleton is as follows:

- hop into `package.json` and modify all the values to fit your project. This likely includes the `"name"`, `"description"`, `"author"`, and `"keywords"` fields.
- Then take a look in `gatsby-config.js`, and follow a similar pattern, modifying the `"title"`, `"description"`, and `"author"` fields. You can also scroll down and active the `gatsby-plugin-offline` plugin and delete the comments about it.
- In the `src/` folder, within `pages/`, delete the `page-2.js` and `using-typescript.tsx` files. You can then delete the two `<Link>` components to each of those pages from `index.js`, as well as the `Link` import there.
- Delete the comments in `gatsby-browser.js`, `gatsby-node.js`, and `gatsby-ssr.js`.
- In the `components` folder, delete `layout.css` (and where it is imported in `layout.js`).
- Delete the `gatsby-astronaut.png` image in the `images/` folder and delete the `<StaticImage>` component from `index.js`.
- Delete the comment fluff on the top of each of the remaining components `header.js`, `layout.js`, and `seo.js`
- Convert all the remaining component files to `.tsx` files, since they are all React components. Also capitalize all the files in the `components/` folder, i.e. `Header`, `Layout`, and `Seo` - we do this as the standard TypeScript pattern for files to match their export names. We won't capitalize the names of the files within the `pages/` folder, since these file names will reflect the actual URL of the page that is produced.
- Remove all references to `propTypes` and `defaultProps` in the codebase.
- After doing that, you'll need to clean up what is now the `Seo.tsx` file. We'll create an `ISeoProps` to use as our props instead. The full resulting component that makes TypeScript happy looks like this:

```
import * as React from "react"
import { Helmet } from "react-helmet"
import { useStaticQuery, graphql } from "gatsby"
import { siteMetadata } from "../../gatsby-config"

export interface ISeoProps {
  title: string
  description?: string
}

function Seo(props: ISeoProps) {
  const { description, title } = props
  const { site } = useStaticQuery(
    graphql`
```

```
      query {
        site {
          siteMetadata {
            title
            description
            author
          }
        }
      }
    `
  )

  return (
    <Helmet>
      {/* General tags */}
      <title>{title}</title>
      <meta
        name="description"
        content={description || siteMetadata.description || ""}
      />
      <meta property="og:title" content={title} />
      <meta
        property="og:description"
        content={description || siteMetadata.description || ""}
      />

      {/* Twitter Card tags */}
      <meta name="twitter:card" content="summary_large_image" />
      <meta name="twitter:creator"
      content={site.siteMetadata?.author || ""} />
      <meta name="twitter:title" content={title} />
      <meta
        name="twitter:description"
        content={description || siteMetadata.description || ""}
      />
    </Helmet>
  )
}

export default Seo
```

it’s not as detailed as an SEO component could be, but we’ll be revisiting and boosting the `Seo` component later in the book.

- Also update the `README.md`. I typically set the title of the README as the name of the repository itself, and then add a small description, something like this:

  ```
  # reduxplate.com

  The website source for ReduxPlate - never write a line of Redux
  again.
  ```

  Since this repository is private, we won't be adding any more information to the README. If you are open-sourcing your project of course, it's wise to include things like install steps, environment variables, and any other examples or requirements to get the product running.
- Finally, update the LICENSE file. You can keep the BSD license, but be sure to change the company name to your company or your own name. I prefer the MIT license. When formatted for my own company, Full Stack Craft LLC, the MIT license looks like this:

  ```
  MIT License

  Copyright (c) Full Stack Craft LLC and its affiliates.

  Permission is hereby granted, free of charge, to any person
  obtaining a copy
  of this software and associated documentation files (the
  "Software"), to deal
  in the Software without restriction, including without
  limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense,
  and/or sell
  copies of the Software, and to permit persons to whom the
  Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be
  included in all
  copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
  EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
  MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO
  EVENT SHALL THE
  ```

```
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE
SOFTWARE.
```

Also remember to update the `"license"` key in `package.json` appropriately if you choose also to switch to a different license, here following my MIT license example:

```
"license": "MIT",
```

So far so good. Right now, the folder structure of the skeleton of the Gatsby default starter should look like this:

```
.
├── LICENSE
├── README.md
├── gatsby-browser.js
├── gatsby-config.js
├── gatsby-node.js
├── gatsby-ssr.js
├── package-lock.json
├── package.json
└── src
    ├── components
    |   ├── header.tsx
    |   ├── layout.tsx
    |   └── seo.tsx
    ├── images
    └── pages
        ├── 404.tsx
        └── index.tsx
```

We've got only two pages, the home page (`index.tsx`) and a 404 (`404.tsx`) page, and a handful of components: a layout, a header, and an seo utility component.

ℹ️ A Word On Typescript ℹ️

For a Full Stack SaaS Product, I would argue that using TypeScript is nearly a necessity. It speeds up development, maintainability, and will help you catch any type errors before you even run your code. We will be using it all across the frontend, including our serverless functions, as we'll see later.

For the Gatsby project, every file we write within the `src` directory will have either a `.tsx` extension, when JSX syntax is needed for React, or a `.ts` extension, for any other non-React code. Luckily, Gatsby supports TypeScript out of the box, so all we need to do is convert the existing files to their respective `.ts` and `.tsx` extensions, and we are all set.

Recap of the Frontend Bootstrapping

We're nearly reading to start actually coding and building our frontend. We've bootstrapped our project with the Gatsby CLI. We've edited our `package.json` and `gatsby-config.js` to reflect our project, converted all components to `.tsx` files, and removed all fluff from all files and code. We also made a few changes to get the codebase to jive nicely with TypeScript. All that is left to get started is to creating a proper git repository so we can start pushing our changes!

✅ Milestone Code ✅

We've reached the first milestone repository of this book: **the skeleton Gatsby repository which we've just finished crafting**! There's not much in it, but it is a perfect minimalist and TypeScript-minded Gatsby boilerplate to start your future SaaS products with.

2.4 Setup a Bitbucket Repository for the Fronte

Chapter Objectives

- Creating a BitBucket repository for the SaaS app's frontend.

Since this will be a private SaaS product, I will be creating a Bitbucket repository for it. Feel free to start yours in a private (or even public!) repository on GitHub. Just keep in mind that further on in this book you will have to take care of things like API secrets and keys in an environment like GitHub by yourself. This is still possible and the workflow is very similar to Bitbucket.

Create the Repository

Create an account on BitBucket if you don't have one already. Then, from your overview dashboard, click the '+' icon in the top left of the screen:

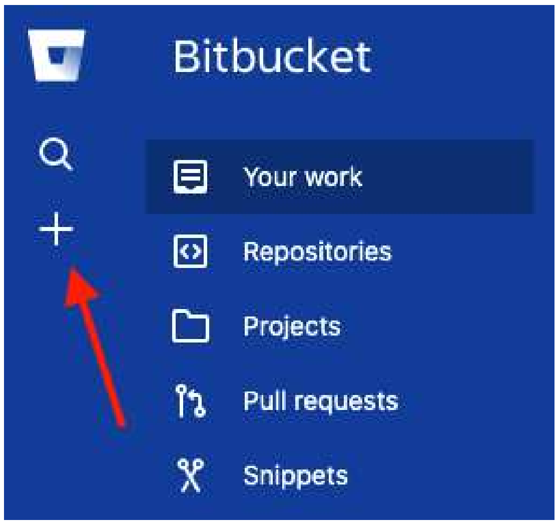


Figure 2.2: Screenshot of adding a repository on Bitbucket.

then select 'Create' > 'Repository':

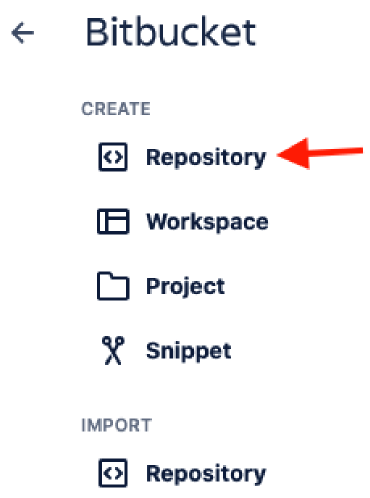
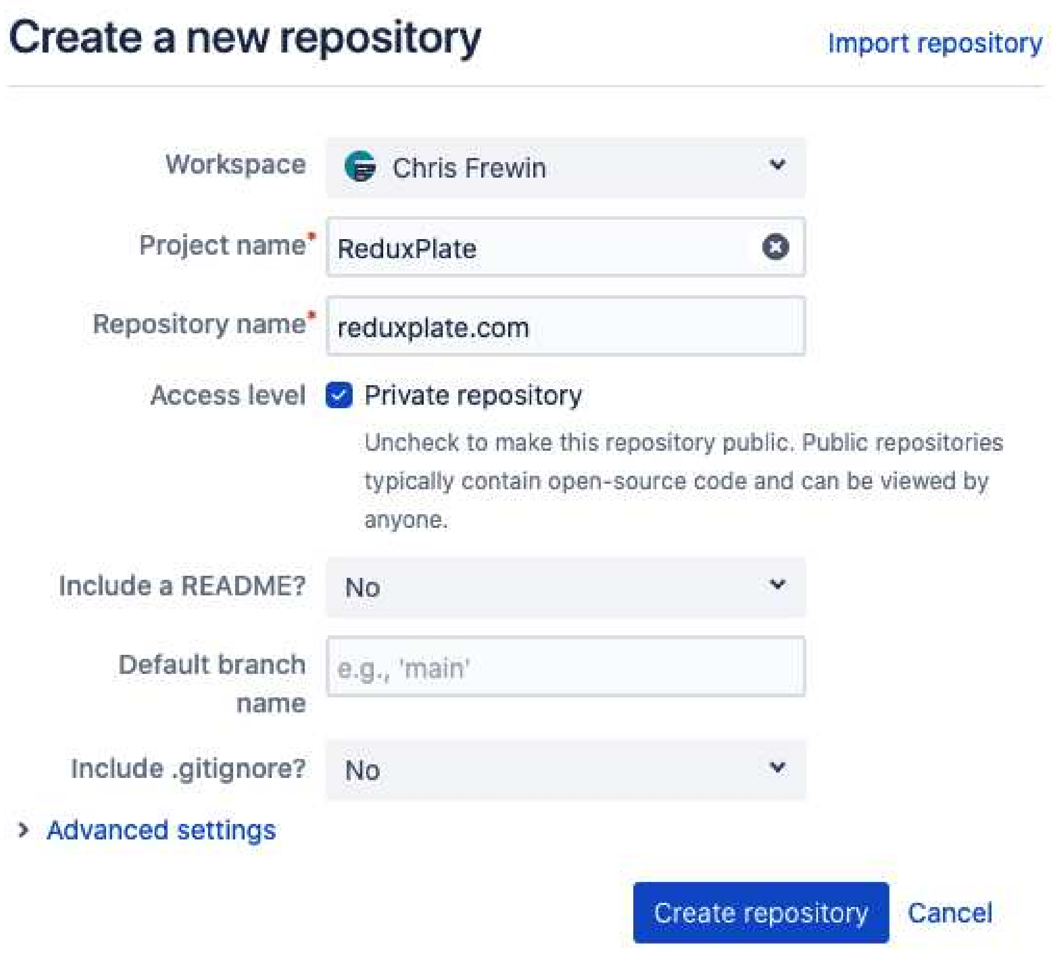


Figure 2.3: Screenshot of the 'create repository' on Bitbucket.

On the resulting page, apply the following:

- Workspace: can just be your workspace, or your team's if you have one.
- Project: I created a new project called 'ReduxPlate', you can choose whatever project you'd like here
- Repository name: should match the folder name that Gatsby made for us, in my case `reduxplate.com`
- Include a README? > No
- Default branch name > Leave blank
- Include .gitignore > No (Gatsby includes one for us!)

All configured, the repository you are about to create should look something like this:


Create a new repository

Workspace	Chris Frewin
Project name*	ReduxPlate
Repository name*	reduxplate.com
Access level	✓ Private repository
	Uncheck to make this repository public. Public repositories typically contain open-source code and can be viewed by anyone.
Include a README?	No
Default branch name	e.g., 'main'
Include .gitignore?	No

Figure 2.4: Screenshot of repository fields for your SaaS product.

Go ahead and click the blue 'Create repository' button. You should be redirected to your repository's homepage.

Add the Repository URL to Project and Push the Code

Nice, so we've successfully created out Bitbucket repository. Let's do the signature 'initial commit' with our current scaffolded project as a sanity check to make sure things are working.

To achieve this, first make sure your `package.json` reflects the new repository you've just created. As an example, here is the `"url"` key with my own Bitbucket git URL:

```json
"repository": {
  "type": "git",
  "url":
  "https://princefishthrower@bitbucket.org/princefishthrower/reduxplate.com.git"
},
```

We also need to update the git origin url from the Gatsby starter to our new repository:

```
git remote set-url origin
https://princefishthrower@bitbucket.org/princefishthrower/reduxplate.com.git
```

We are ready to push. Do that with:

```
git add .
git commit -m "initial commit"
git push
```

Don't worry about adding files or patterns to the `.gitignore` file, the Gatsby starter has already included one for us!

2.5 Use Netlify for the Frontend DevOps Frame

Chapter Objectives

- Using Netlify and the Netlify CLI to build and deploy our site to a live URL whenever we push to the `master` branch

Alright. So we've got our skeleton Gatsby project and a Bitbucket git repository to track our changes as we build the project. Let's connect Netlify now for automatic builds and publishes to master.

Log In or Create an Account for Netlify

Like Bitbucket, Netlify accounts are free for individuals on the most basic plan. From you dashboard, navigate to the 'Sites' section and click the green button 'New site from Git':

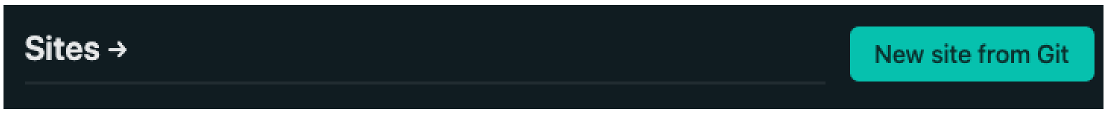
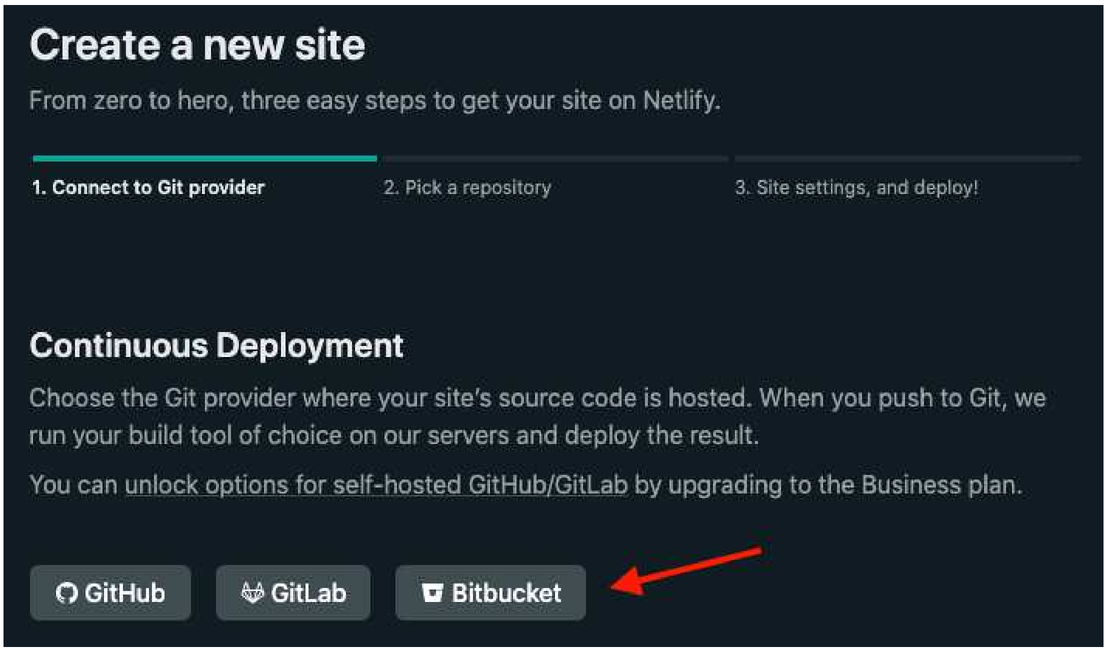


Figure 2.5: Screenshot of the 'New site from Git' button.

On the resulting page, click the 'Bitbucket' button:


Create a new site

From zero to hero, three easy steps to get your site on Netlify.

Continuous Deployment

Choose the Git provider where your site's source code is hosted. When you push to Git, we run your build tool of choice on our servers and deploy the result.

You can unlock options for self-hosted GitHub/GitLab by upgrading to the Business plan.

Figure 2.6: Screenshot of the 'Bitbucket' button.

You'll then be guided through the OAuth process to connect your Bitbucket account to Netlify. After authenticating, you'll be redirected back to Netlify, where you should see a list of all your Bitbucket repositories. You can scroll through or search for the repository you want to connect. In my case that is 'reduxplate.com'. Then click that repository.

Netlify is so smart that it detects we are using the Gatbsy framework and suggests the build command as `npm run build` and the resulting build directory as (which indeed are both correct!) Click 'OK' to confirm these two variables and feast your eyes as your first build takes off!

A Word On Netlify

I've only been using Netlify since February 2021, but I am already hooked on it as a service, and it deserves it's own section here. I find the tiers of their service so generous, that sometimes, it almost feels like *stealing*. On the free plan, you immediately receive 100 GB of bandwidth, 300 build minutes, *and both quotas complete reset each month*. It truly is an incredible service.

As we'll see later, even with their authentication / authorization service, known as Netlify Identity, you don't pay until you have over *1000 active users*, and if we get that many users on our SaaS product, we don't have to worry about a few additional service fees that'll we'll be more than happy to pay Netlify for! 😉

So, hats off to you, Netlify team, your service is awesome! 👍

Rename the Assigned Netlify Domain Name

Netlify supplies us with a random name for our subdomain, but we can rename this to whatever we'd like! If you've picked a unique enough name for your product, chances are it will be available for your Netlify site domain. If it's already taken, consider adding an hyphen or other small changes so it is a human readable reminder of what the product or project name is. In my case, `reduxplate` was available.

2.6 Add a Primary Domain to Netlify via Namec

Chapter Objectives

- Buying a domain via Namecheap, and setting that as our primary domain on Netlify

It's great that Netlify right away gives us a live domain (now `reduxplate.netlify.app` in my case), but a custom domain is always better, right? Luckily, Netlify shines through once again, allowing us to point a primary domain.

I've already purchased `reduxplate.com` as my primary domain, so I'll use that as an example here.

ℹ️ A Word on Namecheap ℹ️

While Namecheap does not have perhaps the best UI or services, they are true to their word in that they are *cheap*. For DNS setups outside of what we will need for Netlify, their DNS manager UI has a few quirks that takes some getting used to, but that is outside of the scope of this book. As an overall rule of thumb I *do* recommend Namecheap, as their domain prices are quite competitive.

⚠ Domain Name Shopping ⚠

Choosing and purchasing a domain is an important step to consider *before* you even start writing code for your product - you don't want to get in the classic trap of building out a brand and logo without an applicable domain to use first! Nowadays you can always find a `.app`, `.us` or similar top level domains for whatever domain name you are looking for, but the classic top level domain `.com` is what I recommend you try and get a hold of. Also realize that this may take some compromising and / or creativity, and that shorter domain names can be rather expensive!

Adding Netlify DNS

To do this, you will have to go to the domain provider . As previously stated, mine is Namecheap, so I can go to my site's dashboard on Namecheap, and click the dropdown for the 'Nameservers' tab, and click the 'Custom DNS' option:

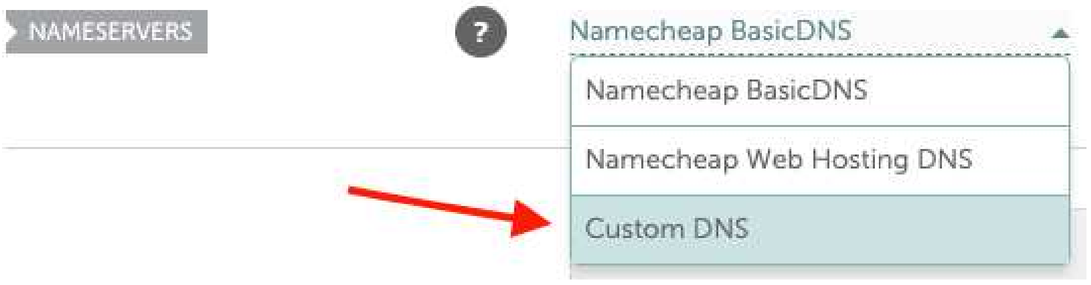


Figure 2.7: Screenshot of the nameservers dropdown on the Namecheap site dashboard.

In the fields that appear, simply apply the handful of values that Netlify provided us with:

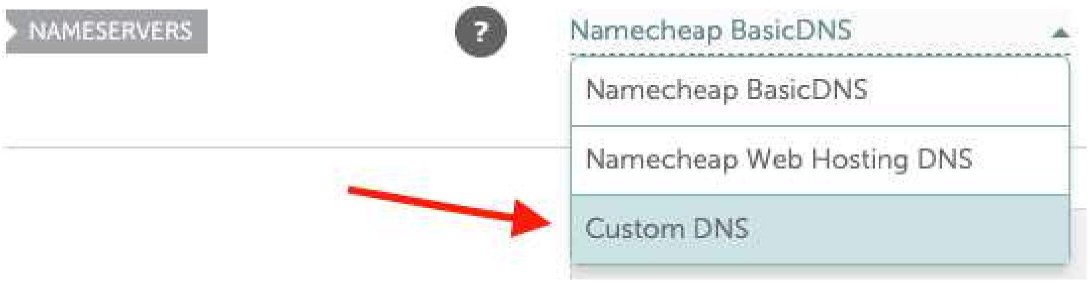


Figure 2.8: Screenshot of the Netlify nameservers applied to the custom DNS configuration on Namecheap.

⚠ **Domain Name Shopping** ⚠

Depending on a variety of factors, like your domain name, your provider, and the nameservers that Netlify gives you, this custom DNS setup could unfortunately take *days* to propagate around the world. As an anecdotal story, when I published my product **The Wheel Screener**, my friends in the United States were able to see the live site within a few hours of me setting up the custom DNS on Namecheap. Here on my internet in Austria, it took about *two days*, and even a bit longer to show up on the cellular network here. So just be prepared for a definitely non-zero lag time for the DNS propagation. It shouldn't be a problem, you'll have plenty to build in the meantime. 😉

Redirect the www Subdomain

We'll also add an automatic redirect from `www.reduxplate.com` to `reduxplate.com`.

3

The Frontend - Implementation

3.1 Adding SCSS and Bootstrap as the Styling

Chapter Objectives

- Use scss as our main styling language.
- Add the `gatsby-plugin-sass` plugin, which allows us to import our .scss files directly and will compile our styles inline during the build process.
- Add Bootstrap as our main styling framework.

Following the official Gatsby documentation on **how to add SASS to Gatsby**, first install both the `sass` package and the `gatsby-plugin-sass` plugin:

```
npm install sass gatsby-plugin-sass
```

also include the `gatsby-plugin-sass` plugin in your `gatsby-config.js`:

```
plugins: [
...
`gatsby-plugin-sass`
]
```

We can now install bootstrap:

```
npm install bootstrap@next
```

Go ahead and create a **`styles/`** folder within the **`src/`** folder, and then add a file simply called **`styles.scss`**. This will be our root SASS file, and we'll import all custom modules we write into it, including Bootstrap.

Installing and Including Bootstrap

We will follow **the Bootstrap official documentation on how to add Bootstrap to our SASS styles**. For now, we can import the Bootstrap SASS directly in our **`styles.scss`** file:

```
@import "../node_modules/bootstrap/scss/bootstrap";
```

As the official docs state, this import should be the first one. All other imports should follow it. In a later section we will work on tree shaking out only the CSS classes which are used in our project to keep our CSS footprint low.

Theming For Bootstrap

Again, following the official documentation,

3.2 Add Netlify Identity as the Authentication

Chapter Objectives

- Setting up Stripe to accept subscriptions

3.3 Use Stripe for the First Payments Platform

Chapter Objectives

- Setting up Stripe to accept subscriptions

3.4 Resolving User Roles

Chapter Objectives

- Buying a domain via Namecheap, and setting that as our primary domain on Netlify

Note that this style of solution is future proof as well: it doesn't matter if we add additional roles later, for example a 'deluxe' or 'corporate' plan. To achieve this future-proofing, we don't use any `switch`, `if`, or `else if` logic on the user's role. Instead we opting for the `Object.values()` of the available roles, and compare them against every role with `roles.find()`.

Excellent. We've add custom styling and themeing to our website, as well as authentication, authorization, and automatic deploys to our live custom URL. The next step is to build our initial endpoint on our custom API and call it from the frontend to ensure that it is working.

4

The Backend - Getting Started

4.1 Introduction to the Backend

Chapter Objectives

- A few of my own opinions when writing backend code with .NET
- Define the framework and tool versions used on the backend

Some Notes on My Backend Style

- Use Repositories
- Use Service Classes

Backend Frameworks and Tools Versioning

On the backend, I will be using the following versions of the following tools and frameworks:

- .NET 5.0

- PostgreSQL 13.2
- Nginx 1.17
- Ubuntu 20.04 (Focal Fossa)

4.2 Bootstrap the Backend With the .NET CLI

4.3 Clean Up the Backend Boiletplate Code

Just as we did for the frontend, we'll clean up some of the extra fluff that .NET has created in our API codebase.

4.4 Setup a Bitbucket Repository for the Backe

Just as we used Bitbucket for the frontend, we will do the same for the backend.

4.5 Use Bitbucket Pipelines for the DevOps

Just as we used Netlify for automatic builds on the frontend, let's hook up Bitbucket Pipelines for automatic builds on our API. To get started with Bitbucket Pipelines, create a **`bitbucket-pipelines.yml`** file in the root of your .NET project:

```
touch bitbucket-pipelines.yml
```

Great. It appears that our pipeline is working, but we have nowhere to send it to! In the next chapter, we'll configure a Digital Ocean droplet with Ubuntu 20.04, PostgreSQL 13.2, and .NET 5.0.

4.6 Create a Digital Ocean Droplet

Our custom .NET API will live on our Digital Ocean Droplet.

4.7 Creating a Droplet

Head over to Digital Ocean and create an account if you don't have one already. Once you're logged in, click the 'Droplets' tab in the sidebar:

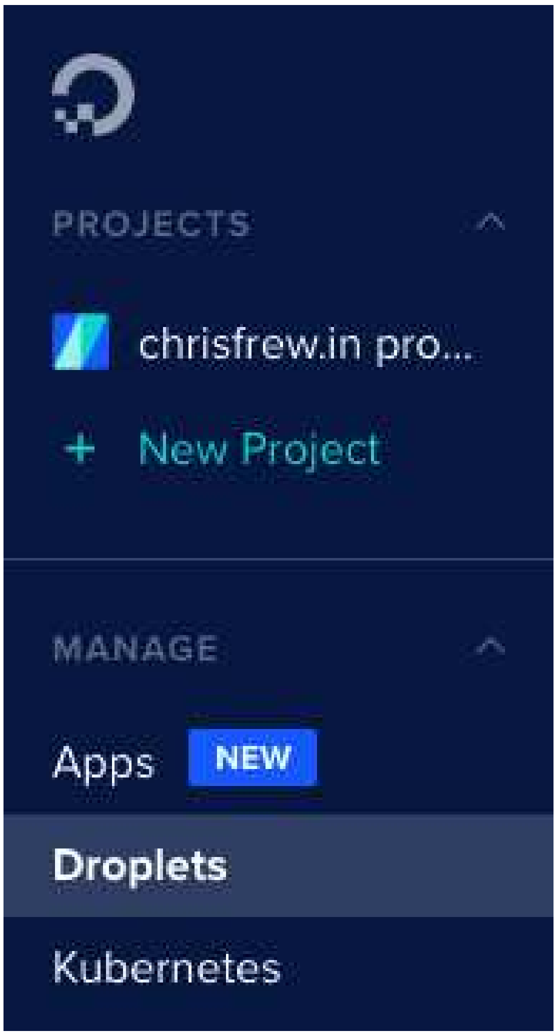


Figure 4.1: Screenshot of the droplets tab.

In the new page that opens, click the big green 'Create Droplet' button:

Figure 4.2: Screenshot of the new droplet button.

On the resulting page, choose the following settings:

- Image > Distributions > Ubuntu 20.04 (LTS) x64
- Plan > Shared CPU > Basic
- CPU Options > Regular Intel with SSD > $5 / month
- Datacenter Region > Choose the option that is closet to where you think most of your customers will be!
- Authentication > SSH Keys > If you have an SSH key registered, that's great. If not read on below.
- Choose a hostname > Pick a hostname that matches your project. Of course following the naming convention of this book I will be using `redux-plate`

Generating a New SSH Key

If you don't have an SSH key saved with Digital Ocean yet, no worries. Click the 'New SSH Key' button to get started:

So, our Digital Ocean Droplet is spooled up and runs at the insane price of just $5 / month! Don't think our app will be able to run on such a tiny little instance? Just wait and see!

4.8 Using Secrets

Secret keeping is always a major disussion when it comes to production environments. To handle our app secrets like our PostgreSQL connection string, we'll be using. SOMETHING

5

The Backend - Implementation

5.1 Writing the First Endpoint for the Custom A

Great. Our API is up and running, and, like the frontend, will automatically deploy upon pushing to the `master branch` Typically, when I get to this point, as a sanity check, I create what I call a `Root` controller which just returns some plain text of an API version string, or really whatever you'd like to have.

6

Building a Staging (or Testing) Environment

So far we've focused on building out the frontend and custom backend API for ReduxPlate. We write code in our `develop` git branch, but every time we merge to the `master` branch in either our frontend or backend repositories, the continuous integration process is fired off and shipped to our live SaaS product immediately. Our continuos integration tool for the frontend is Netlify, and with the backend it is Bitbucket pipelines. That's been great so far for prototyping our MVP, but it's fairly risky once we start having customers.

In this section of the book, we'll get into building out what is known as a staging environment. With all of the tooling available in netlify on the frontend side, and .NET on the backend side, the challenge is not too great, but there will be some important considerations and distinctions which we'll look at in detail.

6.1 The Essential need for a Testing Environme

A staging environment is important, because it mimics your live product almost exactly. As we'll see in this section, in comparison to your live product, the staging version of your product will differ only in small configuration changes. Perhaps the exact quality of what certain API endpoints return may differ, but other than that, your staging site is essentially a production-like, risk-free playground where you can test new features, or catch bugs before they ship to production.

6.2 Staging CI / CD for the Frontend

We'll get the client side of things out of the way first. Again, Netlify's powers come to the rescue and setting up a staging version of the frontend is absolute peanuts.

6.3 Create a Staging Branch for the Frontend

To get started, we'll branch off our develop branch into a new:

6.4 Configure Netlify to Build According to the

On Netlify, head to your product's DNS.

The staging site is up and running! We've got the correct staging environment variables up, builds are firing when we merge to staging; all is well. But if we open a console while looking - we can see . We're getting a bunch of 404 errors when we try to call the staging API endpoint we defined at staging.api.reduxplate.com. Let's switch gears into backend mode and rectify this issue.

6.5 Staging CI / CD for the Backend

Our .NET application will unfortunately be a bit more involved than what it took with Netlify due to it's custom nature. But, .NET and BitBucket offer a lot of powerful features which make the process not too difficult.

6.6 Create a Staging Branch for the Backend

As we did with the frontend, branch of of the development repository for the backend:

Staging Environment Recap

Perfect. We've successfully built out a staging environment from layers as deep as the database, all the way to the frontend. Tools like Bitbucket Pipelines and Netlify's branch builds made this a relatively painless task as well, since we already had the production environments working.

What's Next?

Our SaaS app is in a pretty good position right now: we have staging and production environments running successfully side by side (on both the frontend and backend), and we have fully working user onboarding flow thanks to Netlify and Fauna DB, and are able to process payments and subscriptions with Stripe.

The remainder of this book takes will take our SaaS app to the next level. The remaining sections consist of a variety of "recipes" on how to integrate things like additional payment providers, application-wide logging, and examples of automation tasks you may want to add to your application. I would recommend trying to implement them *all*, as they will bring your SaaS app above and beyond intergalactic standards! 🚀

7

Recipe No. #1: Additional Payment Platform Integrations

7.1 Introduction

Payment integrations are of course an essential part of any SaaS production. In this chapter, we’ll learn how to connect Stripe, PayPal, and Gumroad into the frontend flow, be notified of both new subscriptions and unsubscriptions, and automatically update the role in the user’s netlify Identity user automatically.

8

Recipe No. #2: Add Application-Wide Logging

9

Recipe No. #3: Adding Custom Emails

While Netlify takes care of the user email flow (welcome emails, reset password, forgot password)

10

Recipe No. #4: Adding Automation

Afterword

You’ve Done It!

Well, that was quite an adventure. We’ve both made it out alive! I hope you’ve found this book immensely useful, and that you’re ready to refine your SaaS building skills even further.

Cheers! 🍻

-Chris

Credits and Thanks

Credit where credit is due! (Note that I am not sponsored or supported by any of these platforms or individuals in anyway):

1. Netlify, for their simply awesome "feels like stealing" free tier

2. Bitbucket, for their great UI and tooling, including Bitbucket Pipelines

3. Digital Ocean, for the sheer ease of to start up a Linux instance with a few clicks

4. .NET, for just being an absolute joy of a framework to write and run code in

5. Jason Lengstorf, **@jlengstorf**, who ultimately was responsible in sending me down the Netlify / Identity / Stripe rabbithole with his free course video, **Sell Products on the JAMstack**

6. Josh W. Comeau, **@JoshWComeau** who also released a book independently which inspired me to do the same (somewhat unrelated: I consider him my blog rival, though I suppose that feeling is not mutual 😂)

7. All my family and friends, who had to deal with my near daily spamming of PDF drafts of this book, probably overloading all their memory on all their devices. (It's addicting and too easy to do when you're working with LaTeX!)