Hotrails

← Back to the list of chapters

Turbo Rails tutorial introduction

Published on January 05, 2022

In this chapter, we will explain what we are going to learn, have a look at the finished product and kickstart our brand new Rails 7 application!

What are we going to build?

In this tutorial, we will build a **single-page, reactive quote editor with only a single line of custom JavaScript** for the flash message animation. You can look at the finished project on the quote editor page of this website. Go ahead! Create a quote, click on it and start adding dates and items to see what we'll build.

With the quote editor, you can create, update and delete quotes. When clicking on a quote, you access the page to create, update and delete dates. On each date, you can add line items. Every time you create, update, or delete a line item, the quote total gets updated.

This project is heavily inspired by a project I had to build with React. When `turbo-rails` was released in December 2020, the first line of the README was: ***"Turbo gives you the speed of a single-page web application without having to write any JavaScript"***.

I wanted to see if that was true, so I read the Turbo Rails source code and rebuilt the quote editor with Turbo. **I was blown away by how easy it was to**

work with. And the best part? No more React, no more Redux, no more Formik! Instead, I could work with the tools I love and know well: Ruby on Rails and Simple Form. Boring? Yes, but I could get the benefits from React with a tenth of the effort.

Why learn Ruby on Rails 7 with Turbo?

With the release of Ruby on Rails 7 in December 2021, **Hotwire**, which is the combination of **Stimulus and Turbo**, became the default front-end framework for Rails applications.

Stimulus has been out there for a while now, and it is a well-known library in the Rails ecosystem. On the other hand, Turbo and its integration for Ruby on Rails are brand new tools with impressive features.

- First, all clicks on links and form submissions are now AJAX requests, which speeds up our applications thanks to *Turbo Drive*. **We get this benefit for free as it does not require any work; we only have to import the library**.
- Second, it is now effortless, with just a few lines of code to build dynamic applications by slicing pages in different pieces with *Turbo Frames*. We develop our CRUD controllers just like we did before, and just by adding a few lines of code, **we can replace or lazy-load independent parts of the page**!
- Third, it becomes **trivial to add real-time features** with the help of *Turbo Streams*. Want to add real-time notifications to your application, build a real-time multiplayer game, or a real-time bug monitoring system? The real-time part is just a few lines of code with Turbo!

Turbo speeds up Rails applications, reduces the amount of JavaScript we have to write, and makes it easy to work with real-time features. The best part of it is that it's simple to learn, and **by reading this tutorial, you will know everything there is to know about the three parts of Turbo**!

Who is this book for?

In this book, we will:

- Create CRUD controllers

- Create our design system
- Setup authentication with the Devise gem
- Learn about Turbo Drive, Turbo Frames, and Turbo Streams

If you are already familiar with about 1 - 3 and want to learn about 4, this tutorial is for you!

Application setup

Let's create our brand new Rails application. We will use Sass as a CSS pre-processor to create our design system, esbuild to bundle our single line of JavaScript, and a Postgresql database to be able to deploy our app on Heroku at the end of the tutorial.

We can now create our application:

```
rails new quote-editor --css=sass --javascript=esbuild --database=postgr
```

This tutorial was written for turbo-rails version **1.0.X** (the version of turbo-rails that was released at the same time as Rails 7).

As this tutorial was written at the time of Rails 7.0.0 let's make sure we use the version of turbo-rails that was used at the time to avoid unexpected issues. Let's update our Gemfile by locking the turbo-rails version:

```
# Gemfile

gem "turbo-rails", "~> 1.0"
```

We can now run `bundle install` to install the correct version of the gem.

Now that our application is ready, let's type the `bin/setup` command to install the dependencies and create the database:

```
bin/setup
```

We can now run the rails server, and the scripts that precompile the CSS and the JavaScript code with the `bin/dev` command:

```
bin/dev
```

We can now go to `http://localhost:3000`, and we should see the Rails boot screen.

Note about the `bin/setup` and the `bin/dev` scripts:

It is a good practice to have a solid `bin/setup` script to set up the application for us: install the gems, the JavaScript dependencies, create, migrate and seed the database.

It's almost a must-have when you work in a team, as it should be easy for new developers to set up the development environment. Of course, we could document the process, but documentation gets out of date, and the code does not! If the script breaks, someone will fix it.

Even when working on a small project, using the script has benefits. Every time we need to start from a blank state, we know `bin/setup` has our back.

The `bin/dev` script installs foreman locally and runs the application based on the `Procfile.dev` file. When running the `bin/dev` command, we are running three commands at once:

```
# Procfile.dev

web: bin/rails server -p 3000
js: yarn build --watch
css: yarn build:css --watch
```

We already know the first command `bin/rails server -p 3000` to launch the Rails server. The two other commands `yarn build --watch` and `yarn build:css --watch` are defined in the scripts section of the `Package.json`. They are in charge of precompiling our CSS and JavaScript code before handing them to the asset pipeline. The `--watch` option is here to ensure the CSS and JavaScript code is compiled every time we save a CSS/Sass or JavaScript file.

Both the scripts live in the `/bin` folder of your Rails app if you want to have a look at them.

Everything is ready now for us to start coding. We will start building our application in the next chapter. See you there!

next →