Application setup with esbuild

Step one in building Hotwired ATS is using the Rails application generator to create a new Rails application.

In this chapter, we will set up all of the essential tools needed to build our modern Rails application so that we can focus on using all of the new tools the rest of the way. Since this chapter will primarily be spent on installation and setup, we will often be working through well-documented installation steps. This means we will move more quickly in this chapter than we will in the rest of this book.

By the end of this chapter, we will have a new Rails 7 application configured to use:

- Postgres as our primary database
- **esbuild** to bundle the project's JavaScript, configured to automatically refresh the browser as you make changes to the code.
- **Tailwind** and postcss for styling our application.
- The **Hotwire stack** — Stimulus and Turbo — installed for faster page loads (**Turbo Drive**), partial page updates with **Turbo Frames**, reactive page updates (**Turbo Streams**), and frontend interactivity (**Stimulus**).
- **CableReady** and **StimulusReflex** installed for even more, server-powered frontend interactivity and reactive page updates.
- **Mrujs** to replace some features from Rails/UJS and for its powerful **Cable Car plugin**.
- UUIDs used for primary keys, just because I like using UUIDs.

A note on the density of this chapter before we start building: We are going to setup a lot of tools all at once in this chapter. You may not be familiar with all of them and that is completely okay. Please do not feel the need to start reading documentation or deep-diving into any of these tools. We will learn about all of the key pieces in detail throughout the rest of this book.

Setting up your environment

Before we begin, you will need Ruby, Rails, Postgres, Redis, Node, and Yarn installed.

This book is built for Rails 7 and Ruby 3.0 (3.1 will work fine too). If you are setting up your environment for the first time, make sure to install those versions.

Most of what we will build will work on Rails 6.1, but your setup experience will be different and you may encounter unexplained differences. If possible; please use Rails 7 as you follow along with this book so that you can maximize your time spent in the code and minimize your time spent chasing down weird errors.

If you do not already have your development environment setup and you are on a Mac, your first step should be **rails.new**. Using this script from the Bullet Train team should get your development environment fully configured and ready to follow along with this book. Note that as of March 2022, rails.new may not fully support M1 Macs.

If you are on **Linux** or **Windows**, Chris at GoRails has published great guides you can use to get your development environment up and running.

Creating a Rails application with rails new

The first step in the journey is using `` `rails new` `` to generate our application. Rails 7 ships with Stimulus and Turbo by default. We can use the newly released

jsbundling-rails and **cssbundling-rails** gems to install esbuild and Tailwind with a single command.

From your terminal:

```
rails new hotwired_ats -T -d postgresql --css=tailwind --
javascript=esbuild
cd hotwired_ats
rails db:create
```

The passed-in options skip installing a test framework (`-T`), configure the application to use Postgres as the database (`-d`), install TailwindCSS (`--css`), and select esbuild as our JavaScript bundler (`--javascript`). Once the command runs, you will have a Rails application created with Stimulus and Turbo installed, and the basics of esbuild and Tailwind will be in place.

> **Heads up!**
> Before proceeding, check the version of `turbo-rails` installed in `Gemfile.lock` to confirm that version 1.x is installed. An accidental release in 2021 caused some folks to end up with a bad 7.1.1 version of `turbo-rails` in their bundler cache. If you are in this state, you can fix the problem by running `gem uninstall turbo-rails` and then deleting the `Gemfile.lock` and running `bundle install`.

Next we will make a few adjustments to the default Tailwind installation so that we can import CSS from other files throughout this book.

Configure Tailwind

The default node-powered Tailwind installation provided by `cssbundling-rails` does not allow importing other css files into the main `application.css` file our application serves to end users. Fortunately, we can fix this limitation without too much trouble.

First, install postcss-import via yarn. From your terminal:

```
yarn add postcss-import
touch postcss.config.js
```

And then update `**postcss.config.js**`:

postcss.config.js

```
module.exports = {
  plugins: [
    require('postcss-import'),
    require('tailwindcss'),
    require('autoprefixer')
  ]
}
```

Update `**application.tailwind.css**` to replace the `**@tailwind**` directives with imports, as described in the **Tailwind installation docs**:

app/assets/stylesheets/application.tailwind.css

```
@import "tailwindcss/base";
@import "tailwindcss/components";
@import "tailwindcss/utilities";
```

With these changes made, we can now import any other css files into `**application.tailwind.css**`.

We will also be using the **Tailwind Forms plugin**, which we can add with:

```
yarn add @tailwindcss/forms
```

And then add it to the Tailwind config found in `**tailwind.config.js**`:

tailwind.config.js

```
module.exports = {
  mode: 'jit',
  content: [
    "./app/**/*.html.erb",
    "./app/helpers/**/*.rb",
    "./app/javascript/**/*.js",
  ],
  plugins: [
    require('@tailwindcss/forms')
  ],
}
```

Update esbuild config

Now we will create a custom configuration to replace the default esbuild script provided by jsbundling-rails. This custom configuration will:

- Enable source maps in development and production.
- Minify the bundle in production.
- Automatically rebuild and refresh the page when assets and view files change.

All of these configuration changes are optional, but the changes, especially automatically refreshing the page when files change, make life easier during development. As a bonus, seeing an example of creating a custom configuration for esbuild should help you feel more comfortable using esbuild in the future.

The **`jsbundling-rails`** gem offers the simplest possible esbuild out of the box. In practice, you will often need to add your own custom configuration to use **source maps** and **plugins**.

First, we will use **chokidar** to enable watching and automatically refreshing. From your terminal:

```
yarn add chokidar -D
touch esbuild.config.js
```

Next, we will fill in `esbuild.config.js` like this:

esbuild.config.js

```
#!/usr/bin/env node

const esbuild = require('esbuild')
const path = require('path')

// Add more entrypoints, if needed
const entryPoints = [
  "application.js",
]
const watchDirectories = [
  "./app/javascript/**/*.js",
  "./app/views/**/*.html.erb",
  "./app/assets/stylesheets/*.css",
  "./app/assets/stylesheets/*.scss"
]

const config = {
  absWorkingDir: path.join(process.cwd(),
"app/javascript"),
  bundle: true,
  entryPoints: entryPoints,
  outdir: path.join(process.cwd(),
"app/assets/builds"),
  sourcemap: true
}

async function rebuild() {
const chokidar = require('chokidar')
const http = require('http')
const clients = []

http.createServer((req, res) => {
```

```
  return clients.push(
    res.writeHead(200, {
      "Content-Type": "text/event-stream",
      "Cache-Control": "no-cache",
      "Access-Control-Allow-Origin": "*",
      Connection: "keep-alive",
    }),
  );
}).listen(8082);

let result = await esbuild.build({
  ...config,
  incremental: true,
  banner: {
    js: ' (() => new
EventSource("http://localhost:8082").onmessage = () =>
location.reload())();',
  },
})

chokidar.watch(watchDirectories).on('all', (event,
path) => {
  if (path.includes("javascript")) {
    result.rebuild()
  }
  clients.forEach((res) => res.write('data:
update\n\n'))
  clients.length = 0
});
}

if (process.argv.includes("--rebuild")) {
  rebuild()
} else {
  esbuild.build({
    ...config,
    minify: process.env.RAILS_ENV == "production",
  }).catch(() => process.exit(1));
}
```

There is a lot of code here; let's pause and break it down.

In the `reload` function, we first create a server on port 8082 and build our JavaScript with esbuild with the **banner** configuration option set.

This option inserts JavaScript into the built file that opens a new **EventSource** connection to the web server living on port 8082 and fires `reload` each time a message is received.

Then we configure chokidar to watch the directories we care about and, each time a change is detected, chokidar broadcasts a new message to the EventSource server. This triggers `reload()` for all subscribed browsers and tells esbuild to rebuild JavaScript if the change detected is in the javascript directory.

At the end of the file, the if/else block simply checks the arguments passed to esbuild, and chooses between the `rebuild` function we just reviewed and a regular `esbuild.build()` function that will be used in production because we we will not be watching for live changes in a production environment.

Update bin/dev Scripts

Now that we have updated the Tailwind and esbuild configurations, the next step is to update the scripts section of `package.json` to use the new configurations.

Update the scripts section of `package.json` like this:

package.json

```
"scripts": {
  "build": "node esbuild.config.js",
  "build:css": "tailwindcss --postcss -i
./app/assets/stylesheets/application.tailwind.css -o
```

```
    ./app/assets/builds/application.css"
    },
```

This section of `package.json` defines scripts that we can call from the command line with `yarn` but we do not need to manually run these scripts. Instead, Rails ships with a script, `bin/dev` that calls out to `Procfile.dev`. When we want to change what `bin/dev` does, `Procfile.dev` is usually the place. Update `Procfile.dev` to pass in the `--rebuild` argument to the `yarn build` command:

Procfile.dev

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

When you are ready to boot the application, use `bin/dev` to start the rails server and build JavaScript and CSS all at once.

Next up, we will install CableReady, StimulusReflex, and Mrujs to fill in the gaps when we need more than Turbo provides.

Install CableReady, StimulusReflex, and Mrujs

The installation process for StimulusReflex, which we will go through manually together, is a temporary requirement caused by the churn in Rails's JavaScript options. In the near future, we can expect StimulusReflex to have an automatic installer that works seamlessly with Rails 7 and esbuild. Fortunately for us, manual setup is possible in the interim.

In this section process, both StimulusReflex and CableReady will be installed and configured to use the latest, prerelease versions of the packages. These prerelease versions are recommended for production use by the maintainers.

Despite the prelease label the prerelease versions of CableReady and StimulusReflex are well-tested and stable.

Note that we are almost directly working from the StimulusReflex **documentation** for this section.

From your terminal:

```
bundle add stimulus_reflex --version 3.5.0.pre8
yarn add stimulus_reflex@3.5.0-pre8
rails dev:cache
rails generate stimulus_reflex:initializer
```

Then update `app/javascript/controllers/application.js` to initialize StimulusReflex:

app/javascript/controllers/application.js

```
import { Application } from "@hotwired/stimulus"
import StimulusReflex from 'stimulus_reflex'

const application = Application.start()

// Configure Stimulus development experience
application.warnings = true
application.debug    = false
window.Stimulus      = application

StimulusReflex.initialize(application, { isolate: true
})

export { application }
```

Next we need to update our cache and session store in local development. Update `config/environments/development.rb`:

config/environments/development.rb

```
# Replace config.cache_store :memory_store with this line
config.cache_store = :redis_cache_store, { url:
ENV.fetch("REDIS_URL") { "redis://localhost:6379/1" } } #
You may need to set a password here, depending on your local
configuration.

# Add this line
config.session_store :cache_store, key:
"_sessions_development", compress: true, pool_size: 5,
expire_after: 1.year
```

When you create a new Rails app with esbuild, **ActionCable's** JavaScript is not configured by default, so we will add that next. StimulusReflex, CableReady, and Turbo Streams all rely on ActionCable being properly configured.

From your terminal:

```
mkdir app/javascript/channels
touch app/javascript/channels/consumer.js
```

And fill in `consumer.js` with:

app/javascript/channels/consumer.js

```
1 import { createConsumer } from "@rails/actioncable"
2
3 export default createConsumer()
```

Then, add the actioncable meta tags to your application layout. In `app/views/layouts/application.html.erb:`

app/views/layouts/application.html.erb

```
    <head>
      <title>Hotwired ATS</title>
      <meta name="viewport" content="width=device-
    width,initial-scale=1">
```

```
    <%= csrf_meta_tags %>
    <%= csp_meta_tag %>
    <%= action_cable_meta_tag %>

    <%= stylesheet_link_tag "application", "data-turbo-track": "reload" %>
    <%= javascript_include_tag "application", "data-turbo-track": "reload", defer: true %>
  </head>
```

Finally, StimulusReflex relies on the `stimulus` package, instead of the `@hotwired/stimulus` package, which is the same package with a different name. For StimulusReflex to work properly, we need to update `package.json` to reference both packages:

package.json

```
"@hotwired/stimulus": "^3.1.0",
"stimulus": "npm:@hotwired/stimulus"
```

I know, it is a little confusing for me too.

With the packages swapped, StimulusReflex and CableReady are installed. Next we will install Mrujs. From your terminal again:

```
yarn add mrujs
```

Update `app/javascript/application.js` like this:

app/javascript/application.js

```
import "@hotwired/turbo-rails"
import "./controllers"
import consumer from './channels/consumer'
import CableReady from "cable_ready"
import mrujs from "mrujs";
import { CableCar } from "mrujs/plugins"
```

```
7
8
9  mrujs.start({
10   plugins: [
11     new CableCar(CableReady)
12   ]
})
```

Whew, that was a lot. Thanks for hanging in there. At this point, our application is ready to use Turbo, StimulusReflex, CableReady, and Mrujs with support for CableReady's JSON serializer, **CableCar** via a **plugin**. We are almost finished with the setup steps and ready to start building features.

Let's wrap up this chapter by configuring our application to **use UUIDs for primary keys** and creating an empty dashboard page so we can see something besides the default Rails welcome screen when we boot the app.

Use uuids by default

To use uuids, we need to enable the pgcrypto Postgres extension, which we can do with a database migration:

```
rails g migration EnableUUID
```

Update the generated migration file:

db/migrate/[timestamp]_enable_uuid.rb

```
1  class EnableUuid < ActiveRecord::Migration[7.0]
2    def change
3      enable_extension 'pgcrypto'
4    end
5  end
```

Run this migration from your terminal to enable the extension:

```
rails db:migrate
```

Next we will add a configuration file to configure the Rails model generator to automatically use uuids as the primary key for new models.

From your terminal:

```
touch config/initializers/generators.rb
```

And fill that file in with:

config/initializers/generators.rb

```
Rails.application.config.generators do |g|
  g.orm :active_record, primary_key_type: :uuid
end
```

Since ordering records by primary key is not very useful when they are not in sequential order, we can tell ActiveRecord to use `**created_at**` to order records when no order is specified.

In `**app/models/application_record.rb**`:

app/models/application_record.rb

```
class ApplicationRecord < ActiveRecord::Base
  primary_abstract_class
  self.implicit_order_column = 'created_at'
end
```

Create an empty dashboard

Near the end of this book, we will build a dashboard with two StimulusReflex-powered charts. Nine chapters early, we will create a placeholder

`DashboardController` to use as a default root route and to store links for which we do not yet have a place for in the UI.

To create the Dashboard controller, use the Rails controller generator from your terminal:

```
rails g controller Dashboard show
```

This will create a DashboardController in `app/controllers` with a single `show` action defined and a corresponding `show.html.erb` view in `app/views/dashboards`.

Head over to the routes file and set the root route to the Dashboard's show action:

config/routes.rb

```
Rails.application.routes.draw do
  get 'dashboard/show'
  root to: 'dashboard#show'
end
```

Start up the Rails app with `bin/dev` and head to localhost:3000. If all is well, you should see a page that looks like this:

```
Dashboard#show
Find me in app/views/dashboard/show.html.erb
```

Beautiful, stunning. You're doing amazing so far.

That's all for this chapter! Now would be a really good time to **`git add .`** and **`git commit`** if you are following along on your own machine.

With the setup complete, we are ready to start building features. In the next chapter, using Devise, we will create Users and Accounts while getting our first look at a Stimulus controller.

To see the full set of changes in this chapter, review **this pull request** on Github.

Change	Date	PR link
Updated the method for importing Stimulus in a way that is compatible with both the newly released Stimulus 3.1 and StimulusReflex.	July 28, 2022	**PR 19 on Github**