Funicular on Rails

Write Your Rails Frontend in Pure Ruby

hasumikin

BRUG Meetup

2026-09-16

self.inspect

- Hitoshi HASUMI
  - @hasumikin (GitHub and Twitter)
- Creator of PicoRuby
- Committer of mruby and mruby/c
- Committer of CRuby IRB and Reline

Today's Topic

Part 1. Chat App in 5 Minutes

Part 2. Under the Hood

Part 3. Rails Integration Tour

Part 1

Chat App in 5 Minutes

Demo : Chat App

👀 Terminal + two browser windows

What We Just Did

```
rails new chat_demo
vim Gemfile
  gem "funicular"
  gem "json", "< 3"              # for ActiveSupport compat
bundle install
bin/rails funicular:install
npm install                      # for jsdom (Testing)
bin/rails generate funicular:chat
bin/rails db:migrate
bin/rails funicular:compile
bin/rails server
```

Zero JavaScript written. Zero JavaScript build tool

What the Generator Made (Excerpt)

```
app/models/funicular_chat_message.rb                # Rails
app/controllers/funicular_chat_controller.rb        # Rails
app/channels/funicular_chat_channel.rb              # Rails
app/views/funicular_chat/show.html.erb              # Rails
-------------------------------------------------------------
app/funicular/components/funicular_chat_component.rb # UI
app/funicular/initializer.rb                         # UI
test/funicular/client/funicular_chat_component_picotest.rb
```

- Rails owns persistence and broadcasting
- Funicular owns the stateful, interactive UI

The Component (Excerpt)

```
class FunicularChatComponent < Funicular::Component
  def initialize_state
    { name: "Rails developer", body: "", messages: [], connected: false }
  end

  def component_mounted
    Funicular::HTTP.get("/funicular_chat/messages") do |response|
      patch(messages: response.data)
    end

    @consumer = Funicular::Cable.create_consumer("/cable")

    @consumer.subscriptions.create(channel: "FunicularChatChannel") do |message|
      patch(messages: state[:messages] + [message])
    end
  end
end
```

Wait, Ruby in the browser?

Part 2

Under the Hood

PicoRuby Started on Microcontrollers

- Ruby for Raspberry Pi Pico and ESP32
- Compiler using Prism + mruby VM + Task scheduler
- 512 KB of RAM is plenty
- Bare metal: PicoRuby does NOT depend on an OS
- ...and that turned out to matter in the browser

PicoRuby consists of

- PicoRuby compiler using Prism
- mruby VM + Task scheduler (mruby-task mgem)
- A lot of libraries (Picogems)
- Microcontroller integration: Raspi Picos and ESP32
- Browser-based runtime:
  PicoRuby.WASM (picoruby-wasm mgem)
- Browser app framework:
  Funicular (built in PicoRuby.WASM)

How to Use PicoRuby.WASM

```
<script src="https://cdn.jsdelivr.net/npm/@picoruby/
wasm-wasi@latest/dist/init.iife.js"></script>

<!-- Embedded Ruby Script -->
<script type="text/ruby">
  puts "Hello, World!"
</script>

<!-- Remote Ruby Script file (.rb) -->
<script type="text/ruby" src="hello.rb"></script>

<!-- Remote Precompiled Ruby VM Code file (.mrb) -->
<script type="application/x-mrb" src="hello.mrb"></script>
```

CRuby.WASM vs PicoRuby.WASM

	CRuby.WASM + stdlib	PicoRuby.WASM
`Kernel#sleep`	❌	✅
Multithreading	No `Thread` support	`Task` support
Binary size (compressed)	31.0 MB (8.6 MB)	2.0 MB (713 KB)
ASYNCIFY	Yes	No

Why sleep Works

- CRuby essentially depends on the OS
  →Browser can't use OS's syscall
- PicoRuby runs on a multi-task scheduler
- **`Kernel#sleep`** stops its task and returns to the scheduler
- In PicoRuby.WASM, the JS event loop IS the scheduler
  - **`sleep`** works like **`yield`**
- This mechanism is basically the same as on microcontrollers

PicoRuby.WASM's Main Loop

```
function run() {
  // 1. wake sleeping tasks, fire timers
  Module._mrb_tick_wasm();
  // 2. run Ruby for ~16ms (one frame)
  const sliceStart = performance.now();
  while (performance.now() - sliceStart < BATCH_DURATION) {
    if (Module._mrb_run_step_status() <= 0) break;
  }
  // 3. give way to the browser
  setTimeout(run, 0);
}
run();
```

- No ASYNCIFY: that's why the binary is ~50% smaller

Async Without ASYNCIFY

- **`fetch`**, **`setTimeout`**, **`addEventListener`**
  → suspend the Ruby task **before** the JS call
- Promise resolves → task resumes on a fresh C stack
- **`raise`** / **`rescue`** work across async boundaries
- The Ruby code looks synchronous (no async color)

```ruby
response = JS.global.fetch("/api/posts").await
# The browser stays responsive while waiting
```

JS Bridge from Ruby

```ruby
el = JS.document.getElementById("app")     # JS::Element
el[:textContent] = "Hello"                 # set property
JS.global[:navigator][:language]           # => "en-US"
JS.global.history.pushState({}, "", "/chat")
```

- **`JS::Object`** inherits **`BasicObject`**
  → **`hash`**, **`send`**, **`class`** reach JS via **`method_missing`**
- Strings, numbers, booleans, nil convert automatically

Events Reach Ruby via Task::Queue

```ruby
def self._spawn_event_consumer(callback_id, block)
  q = Task::Queue.new
  Task.new(name: "js-cb-#{callback_id}") do
    while ev = q.pop      # suspends until JS pushes an event
      block.call(ev)
    end
  end
end
```

- One long-lived consumer Task per listener
- JS pushes the event, Ruby pops it

Debugging: binding.irb in the Browser

- Debug build exports `mrb_debug_*` API
- Chrome DevTools extension: **PicoRuby Debugger**
  - REPL in the running app's context
  - `binding.irb` breakpoints, step, next
  - Locals, call stack
  - Funicular component inspector (state, props, tree)
- You won't use a runtime unless it's debuggable

Funicular

- Single-Page Application framework similar to ReactJS
- Virtual DOM + Diffing
- Class-based Components
- Ruby DSL with no JavaScript. No JSX
- Routing, Forms, Models, Realtime, SSR, Local DB...
- ~10K lines of pure Ruby (`mrblib/`)

How to Write Funicular: e.g. Counter

```ruby
class CounterComponent < Funicular::Component
  def initialize_state
    { count: 0 }
  end

  def render          # required override
    div do
      p { "Current count: #{state[:count]}" }
      button(onclick: :increment) { "Increment" }
    end
  end

  def increment       # user-defined
    patch(count: state[:count] + 1) # kicks `render`
  end
end
```

How to Patch

```
Component#patch
 └─ Component#component_will_update # If callback exists
    @state.merge(normalized_state)
    Component#re_render
     └─ patches = VDOM::Differ.diff(@vdom, new_vdom)
        VDOM::Patcher.new.apply(@dom_element, patches)
    Component#component_updated     # If callback exists
```

(Nothing special)

State flows down, events flow up through **patch**

What Ships to the Browser

Artifact	Size (brotli)	What it is
picoruby.wasm	2 MB (713 KB)	The VM. Funicular inside
your_app.rb	your app	Your .rb file
or your_app.mrb	your app	`mrbc` compiled

- The VM binary is the same for every app
- When **`*.mrb`**, bytecode goes straight into the VM
  → no parsing at runtime

Part 3

Rails Integration Tour

The Tour Guide: funicular-demo

- A Slack-like chat app + a tiny blog
  - `/login`, `/chat`, `/settings`, `/blog`
- Rails 8.1, SQLite3, Propshaft, Tailwind
- **github.com/hasumikin/funicular-demo**
- Not every funicular feature is used there
  → snippets on slides fill the gaps

Demo : funicular-demo

👀 Browser: /login → /chat

1. Setup

```
$ echo 'gem "funicular"' >> Gemfile
$ bundle install
$ bin/rails funicular:install
#   funicular:install:wasm          -> public/picoruby/{dist,debug}
#   funicular:install:initializer   -> config/initializers/funicular.rb
#   funicular:install:test          -> test/funicular/, jsdom

==========================================

# app/views/layouts/application.html.erb
<%= csrf_meta_tags %>
<%= picoruby_include_tag %>

# app/views/home/index.html.erb
<%= funicular_app_container %>
<script type="application/x-mrb" src="<%= asset_path('app.mrb') %>">
```

1. Setup: app/funicular/

```
app/funicular/
    ├── models/        # Funicular::Model      (compiled 1st)
    ├── components/    # Funicular::Component  (2nd)
    └── initializer.rb                         (last)
```

- They are compiled into one **app.mrb**
  - Excluded from Rails autoloading (We don't want them to run on the server)
- Dev: middleware recompiles on the next request
- Prod: **assets:precompile** runs **funicular:compile**

1. Setup: Just a Railtie Behind the Scenes

```ruby
class Railtie < Rails::Railtie
  initializer "funicular.middleware" do |app|
    if Rails.env.development?
      app.middleware.use Funicular::Middleware # recompile on change
    end
  end

  initializer "funicular.helpers" do
    ActiveSupport.on_load(:action_view) do
      include Funicular::Helpers::PicorubyHelper
    end
  end

  rake_tasks { load "tasks/funicular.rake" }
end
Rake::Task["assets:precompile"].enhance(["funicular:compile"])
```

2. Routing

```ruby
# app/funicular/initializer.rb
Funicular.start(container: 'app') do |router|
  router.get('/login',              to: LoginComponent, as: 'login')
  router.get('/chat/:channel_id', to: ChatComponent, as: 'chat_channel',
                                    constraints: { channel_id: /^\d+/ })
  router.get('/settings',           to: SettingsComponent, as: 'settings')
  router.delete('/messages/:id',  to: MessageComponent, as: 'message')
  router.set_default('/login')
end
```

- Rails-style DSL, Rails-style **`routes.settings_path`**
- **`:channel_id`** arrives as **`props[:channel_id]`**
- **`bin/rails funicular:routes`** to inspect

2. Routing: Navigation

```ruby
link_to routes.settings_path, navigate: true do    # <a href>, History API
  span { "⚙" }
end

link_to routes.message_path(msg), method: :delete # Fetch + CSRF token

Funicular.router.navigate('/chat')                 # programmatic
```

```ruby
def navigation_guard
  @dirty ? "Unsaved changes will be lost. Leave?" : nil
end
```

- Guard covers links, back/forward, and **beforeunload**

Demo : Routing

👀 Browser: /chat → channel switch → ⚙️ → back

3. Components

```
class ChannelListComponent < Funicular::Component
  def render
    ul do
      state[:channels].each do |ch|
        # Component embeds a child: props down, callbacks up
        component(ChannelItemComponent,
          key: ch["id"],
          channel: ch,
          on_select: ->(c) { select_channel(c) }
        )
```

- An html tag is just a method: **`ul`**, **`div`**, **`tag(:main)`**
- **`key:`** tells the Differ who is who in the list

3. Components: Lifecycle

```
new(props)
  -> initialize_state
  -> render
  -> component_mounted          # subscribe, fetch, focus
  -> [patch -> render -> component_updated]*
  -> component_unmounted        # unsubscribe, disconnect
```

- **`state`** is read-only; **`patch`** is the only way
  → that's the **`[patch -> render]*`** loop above
- **`mounted`** / **`unmounted`** mirror each other
- **`render`** takes no arguments; **`self`** is the component

4. Forms and Validation

```
form_for(:user, on_submit: :handle_submit) do |f|
  f.label :username
  f.text_field :username, autofocus: true
  f.label :password
  f.password_field :password
  f.submit(state[:loading] ? "Logging in..." : "Login")
end
```

- Binds to **`state[:user]`**, two-way
- No **`preventDefault`**; handler gets a form-data hash
- Inline errors from **`state[:errors]`**

4. Validation on Both Sides

```ruby
# app/funicular/models/user.rb (browser)
class User < Funicular::Model
  validates :display_name, format: { with: /^[^@]+$/ }
end

# app/controllers/api/schema_controller.rb (Rails)
render json: Funicular::Schema.build(User,
  attributes: { "display_name" => { type: "string" } },
  endpoints:  { "update" => { method: "PATCH", path: "/users/:id" } },
  except:     { username: [:format] })
```

- **`Schema.build`** reads **`validators_on`** from ActiveRecord
- Client validates first; invalid → no HTTP request at all
- Server still validates (422): the source of truth

4. Validation: Caveats

- Regexp runs on JavaScript **`RegExp`**, not Onigmo
  - Use **`^...$`**, not **`\A...\z`**. No **`/x`**, no **`[[:alpha:]]`**
- Client **`validates`** overrides the schema-derived one
  → JS-safe one instead of Onigmo regexp in prev page
- Skipped: whatever the browser can't run
  - **`uniqueness`** needs the database
  - Custom validators are server-side Ruby
  - **`if:`**/**`on:`** depend on server context

Demo : Forms

👀 Browser: /settings → type “@” in display name

5. Data: Object-REST Mapper Convention

```ruby
# app/funicular/initializer.rb
Funicular.load_schemas(Post => "post") do ... end

# in components
Post.all(category: "tech")    # GET    /api/posts?category=tech
Post.find(123)                # GET    /api/posts/123
Post.create(title: "Hi")      # POST   /api/posts
post.update(title: "Edited")  # PATCH  /api/posts/123
post.destroy                  # DELETE /api/posts/123
```

- Verb and path come from the schema (**`load_schemas`**)
- Like ActiveRecord, but it maps methods to your **`resources :posts`** routes — not to SQL

5. Data: Callbacks, Not Promises

```
Post.all(category: "tech")    { |posts, error| patch(posts: posts) }
Post.find(123)                { |post,  error| patch(post: post) }
Post.create(title: "Hi")      { |post,  errors| ... }
post.update(title: "Edited") { |updated, errors| ... }
post.destroy                  { |_ok, error| ... }
```

- Every call returns immediately; the block runs later
- One convention: **`(result, error)`** — check the 2nd arg

5. Data: Custom Endpoints

```
class Session < Funicular::Model
  storage :ephemeral   # never cached locally

  def self.login(username, password, &block)
    create({ username: username, password: password },
           model_class: User, &block)
  end

  def self.logout(&block)
    destroy(&block)
  end

  def self.current_user(&block)
    find(endpoint_name: "current", model_class: User) do |user, error|
      block.call(user, error) if block
    end
  end
end
```

5. Data: Suspense for Initial Loading

```
class PostComponent < Funicular::Component
  use_suspense :post, ->(resolve, reject) {
    Post.find(props[:id]) { |post, e| e ? reject.call(e) : resolve.call(post) }
  }

  def render
    suspense(:post,
      fallback: -> { div { "Loading..." } },
      error: ->(e) { button(onclick: -> { reload_suspense(:post) }) { "Retry" } }
    ) do |res|
      h1 { res[:post].title }
    end
  end
end
```

6. Realtime: Funicular::Cable

```
def component_mounted
  @consumer = Funicular::Cable.create_consumer("/cable")
  @subscription = @consumer.subscriptions.create(
    { channel: "ChatChannel", channel_id: channel.id }
  ) do |data|
    case data["type"]
    when "new_message"
      patch(messages: state[:messages] + [data["message"]])
    when "delete_message"
      handle_message_delete(data["message_id"])
    end
  end
end

def component_unmounted
  @subscription&.unsubscribe
  @consumer&.disconnect
end
```

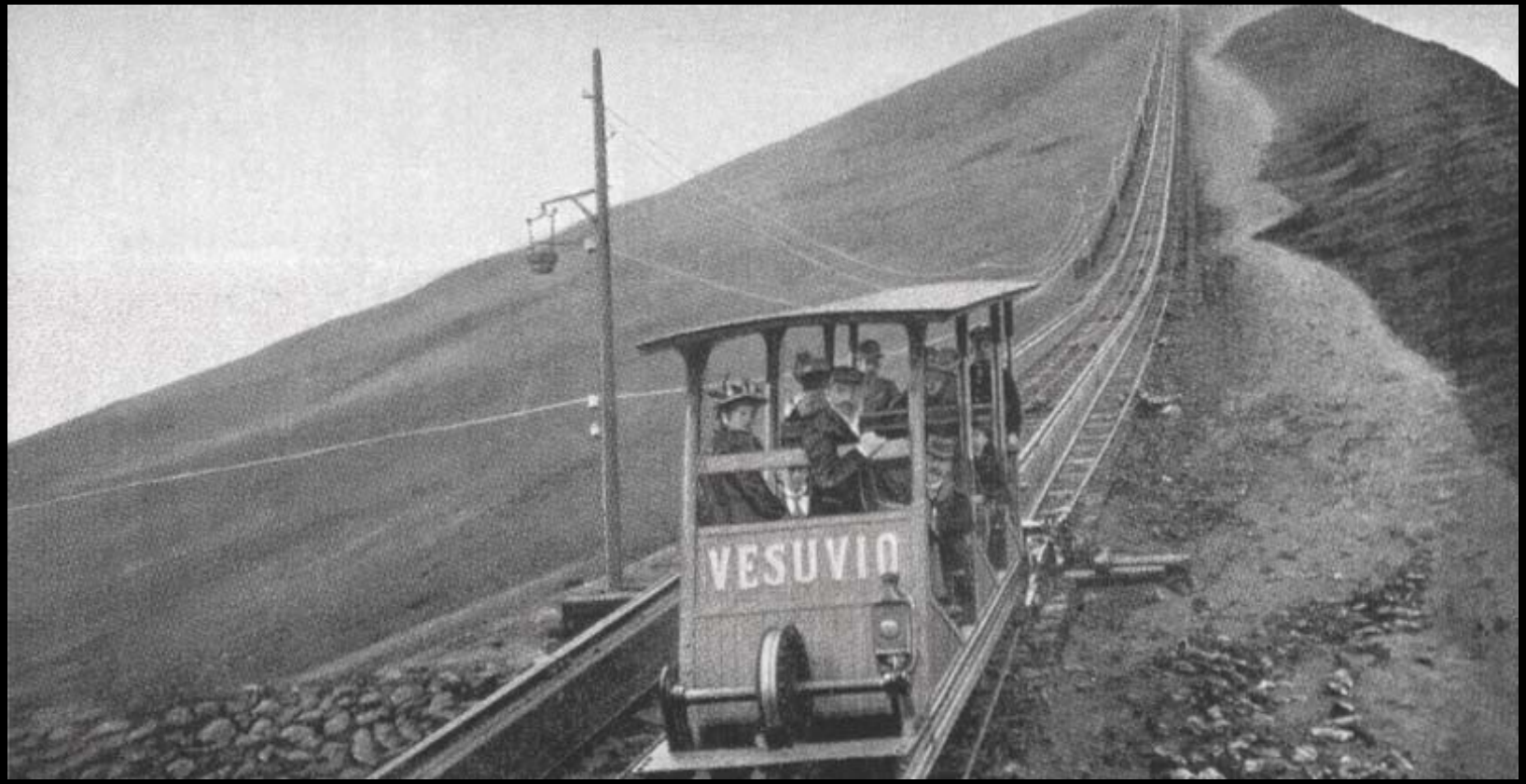
6. Realtime: The Rails Side Is Unchanged

```ruby
class ChatChannel < ApplicationCable::Channel
  def subscribed
    stream_from "chat_#{params[:channel_id]}"
  end

  def speak(data)   # @subscription.perform("speak", message: "Hi")
    ActionCable.server.broadcast("chat_#{params[:channel_id]}", ...)
  end
end
```

- Auto-reconnect, ping/pong
- Watches **`visibilitychange`**;
  suspends when hidden (30 s), resumes on return
- Pending commands survive **`beforeunload`** (localStorage)

Funiculí-funiculá funiculí-funiculá



https://ja.wikipedia.org/wiki/ヴェスヴィオ

Demo : Realtime

👀 Browser: two windows on /chat, delete a message

7. SSR and Hydration

```ruby
class HomeController < ApplicationController
  def index
    @ssr = Funicular::SSR.render(path: "/blog", state: { posts: blog_posts })
  end
end
```

```erb
<%= funicular_app_container(@ssr ? @ssr[:html] : "") %>
<%= funicular_state_tag(@ssr[:state]) if @ssr %>
<script type="application/x-mrb" src="<%= asset_path('app.mrb') %>">
```

- **`window.__FUNICULAR_STATE__`** → client hydrates automatically
- No extra client code

7. SSR: Same Ruby, Three Runtimes

Where	Runtime	Purpose
Browser	PicoRuby.WASM	The app
Rails process	CRuby	SSR
Node.js + jsdom	PicoRuby.WASM	bin/rails test

- The same **mrblib/** code runs in all three
- The switch is one flag: **Funicular.server?**

7. SSR: How It Works

```ruby
def self.render(path:, state: {})
  Runtime.boot!                              # load mrblib/ + app/funicular/ into CRuby
  component_class, params = Funicular.router.match(path)
  instance = component_class.new(params)
  instance.seed_state(state)
  html = VDOM::HTMLSerializer.serialize(instance.build_vdom)
  { html: html, state: state, component: component_class }
end
```

- **`Funicular.server?`** no-ops every JS-touching call
- Nested state uses string keys (**`post["id"]`**): JSON on both sides
- Render deterministically: no **`Time.now`** in **`render`**

7. SSR: Static Components in ERB

```
<%# Plain, non-JS ERB page (e.g. checkout) %>
<%= Funicular::SSR.render_component("StorefrontNavComponent",
                                    props: { active: "cart" })[:html] %>
```

- Share the SPA's header with classic ERB pages
- Static: links work, **`onclick`** does not
- Named by string: the constant lives outside autoloading

Demo : SSR

👀 Browser: /blog → view-source → /blog/1

8. Plugins

```ruby
# Gemfile
group :funicular do
  gem "funicular-datepicker"
  gem "funicular-image-uploader"
end
```

- A plugin is just a gem: **`lib/**/*.rb`** + **`assets/*.css`**
- Bundler group membership IS the registration
- Compiled into the same **`app.mrb`**, before your code

8. Plugins: Who Owns What

```
component Funicular::Plugins::ImageUploader::Component,
          upload_url: "/users/#{current_user.id}/avatar",
          file_field: "avatar", auto_upload: true,
          on_upload: ->(result) { ... }
```

- Plugin: file picker, preview, FormData upload (browser)
- Rails: persistence (bytes, Active Storage, S3…)
- CSS via **`<%= funicular_plugin_include_tags %>`**
- Test a plugin without a dummy Rails app:
  `Funicular::Testing.run!`

Demo : Plugins

👀 Browser: /settings → date picker, avatar upload

9. Local Database: SQLite3 in the Browser

- Real SQLite3 compiled to WASM, queried from Ruby
- Rails is the source of truth
- Local DB is a queryable replica + home for client-only data
- Synchronous reads, snapshots to IndexedDB

```ruby
# config/initializers/funicular.rb (Rails)
Funicular.configure do |config|
  config.local_database = true
  config.user_key = ->(controller) { controller.request.session[:user_id]&.to_s }
end
```

9. Local Database: Three Storage Modes

storage	Local table	REST	For
:replica	from schema	yes	Post, Channel (default)
:ephemeral	none	yes	Session, auth
:local	migrate {}	no	Draft, preferences

- **`Channel.all`** → REST, async, upserts into replica
- **`Channel.local.order(:name)`** → SQL, sync, maybe stale

9. Local Database: Client-Only Model

```ruby
class Draft < Funicular::Model
  storage :local do
    migrate 1 do |t|
      t.integer :channel_id, null: false
      t.text    :body
      t.timestamps
      t.index   :channel_id
    end
  end

  def self.store(channel_id, body)     # upsert; sync, no blocks
    draft = local.find_by(channel_id: channel_id)
    draft ? draft.update(body: body) : create(channel_id: channel_id, body: body)
  end
end

Draft.store(channel_id, text)
rescue Funicular::DB::ReadOnlyTabError # another tab is the writer
```

9. Local Database: `Component#watch`

```ruby
def component_mounted
  watch(:channels) { Channel.local.order(:name) }   # reactive
  Channel.all { |_, error| patch(error: error) if error }  # refresh
end
```

- Runs once, re-runs on any change to the table
- One writer tab elected via Web Locks; others read

Demo : Local DB

👀 Browser: type a draft, switch channel, reload, DevTools > IndexedDB

10. Styling

```ruby
class LoginComponent < Funicular::Component
  styles do
    container "min-h-screen flex items-center justify-center bg-gray-100"
    button base: "px-6 py-2 rounded-lg", variants: { normal: "bg-blue-600", loading: "opacity-50" }
  end

  def render
    div(class: styles.container) do
      button(class: styles.button(:loading)) { "..." }
    end
  end
end
```

- Plain class strings → Tailwind scans **`app/funicular/`**
- Typo → **`NoMethodError`** listing declared names
- Same in the browser and in SSR

11. Testing: Built-in Picotest

```ruby
# test/funicular/client/login_component_picotest.rb
class LoginComponentTest < Funicular::Testing::DOMTest
  def test_empty_submit_shows_validation_error
    mount LoginComponent
    submit "form"
    assert_text "Please enter username and password"
  end
end

# test/funicular/application_test.rb
class FunicularApplicationTest < ActiveSupport::TestCase
  test "client-side Funicular tests" do
    result = Funicular::Testing.run!(timeout_ms: 10_000)
    Funicular::Testing.assert_picotests(self, result)
  end
end
```

11. Testing: How It Runs

- **`bin/rails test`** → spawns Node.js → jsdom → PicoRuby.WASM
- **`*_picotest.rb`**: kept out of the CRuby Minitest loader
- **`mount`**, **`click`**, **`submit`**, **`input`**, **`assert_selector`**, **`drain`**
- Assertion count folded into the Rails report
- jsdom is not a browser: use system tests for layout

12. Debugging

```ruby
# app/funicular/initializer.rb
Funicular.debug_color = "pink"    # outlines every component
```

```ruby
def handle_submit(data)
  binding.irb    # pauses here, in the browser
  Comment.create(post_id: state[:post]["id"], body: data["body"]) { ... }
end
```

- Development serves the debug build by default
- PicoRuby Debugger: REPL, step, inspector
- Available in the Chrome Web Store

Demo : Debugging

👀 Browser: debug_color, DevTools > PicoRuby > component tree

Recap

Funicular on Rails

Rails	Funicular (browser)
routes.rb	router.get(...) / routes.*_path
ActiveRecord validations	Funicular::Schema -> validates
form_with	form_for
ActionCable channel	Funicular::Cable
ERB view	Component#render (+ SSR on CRuby)
Bundler group	Plugins
SQLite3 on the server	SQLite3 in the browser
bin/rails test	*_picotest.rb on jsdom

Funicular: When NOT to Use?

- Need raw performance: WASM + VM overhead
- Huge state with complex management: no best practice yet
- Reliability required: nobody uses it yet (except me)
- Hotwire/Turbo already does the job: keep it

Funicular: When to Use?

- One part of your Rails app becomes a stateful client app
- You would rather not maintain a JS toolchain
- You want to write Ruby, only Ruby, all the way down

Truly Practical Full-Stack Ruby

Stargaze at 🌟

github.com/picoruby/funicular

Docs and tutorials 👀

picoruby.org/funicular