Using Ruby on Rails

During my time at ShowMeBug, I contributed to the Enterprise WeChat Integration project. This involved integrating ShowMeBug with Enterprise WeChat, providing seamless access to technical interview tools within the Enterprise WeChat ecosystem. I utilized technologies such as Ruby, Ruby on Rails, PostgreSQL, and the WeChat SDK to create a smooth user experience for both interviewers and candidates.

This blog post was composed with the assistance of AI around February 2025.

Ruby on Rails (often just "Rails") is a powerful web development framework built on the Ruby programming language. It's designed to make building web applications fast and enjoyable by emphasizing conventions over configuration and DRY (Don't Repeat Yourself) principles. Let's walk through setting it up and creating a simple app.

Step 1: Install Ruby and Rails First, you'll need Ruby installed, as Rails is a Ruby gem (library). Most systems don't come with Ruby pre-installed, so here's how to set it up:

- **On macOS/Linux:**
  - Use a version manager like `rbenv` or `rvm` for flexibility. Install it via Homebrew (`brew install rbenv`), then run:

    ```
    rbenv install 3.2.2  # A stable Ruby version as of 2025
    rbenv global 3.2.2
    ```

  - Install Rails:

    ```
    gem install rails
    ```

- **On Windows:**
  - Use RubyInstaller (download from rubyinstaller.org). Pick a version like 3.2.2 with DevKit.
  - After installing Ruby, open a command prompt and run:

    ```
    gem install rails
    ```

Verify the installation:

```
ruby -v  # Should show something like ruby 3.2.2
rails -v # Should show the latest Rails version, e.g., 7.1.x
```

Step 2: Create a New Rails Project Once Rails is installed, generate a new app:

```
rails new myapp --database=sqlite3
cd myapp
```

This creates a folder called `myapp` with a full Rails structure, using SQLite as the default database (great for development).

Step 3: Start the Server Run the built-in Rails server:

```
rails server
```

Open your browser to `http://localhost:3000`. You'll see a welcome page. Congrats, your Rails app is running!

Step 4: Build Something Simple Let's create a basic "Posts"feature to understand Rails'MVC (Model-View-Controller) pattern.

- **Generate a Model and Controller:**

  ```
  rails generate scaffold Post title:string body:text
  ```

  This creates a `Post` model, database migration, controller, and views—all wired together.

- **Run the Migration:**

  ```
  rails db:migrate
  ```

  This sets up the database table for posts.

- **Check It Out:** Restart the server (`rails server`) and visit `http://localhost:3000/posts`. You'll see a CRUD interface to create, read, update, and delete posts.

Step 5: Explore Key Concepts

- **Routes:** Open `config/routes.rb`. You'll see `resources :posts`, which auto-generates RESTful routes like `/posts/new` or `/posts/1/edit`.
- **Controllers:** Look at `app/controllers/posts_controller.rb`. It handles requests and responses.
- **Views:** Check `app/views/posts/`. These are ERB templates (HTML with Ruby embedded).
- **Models:** See `app/models/post.rb`. It ties to the database and can include validations (e.g., `validates :title, presence: true`).

Step 6: Customize and Deploy

- Add some style with CSS in `app/assets/stylesheets/`.
- For production, switch to a database like PostgreSQL (`rails new myapp --database=postgresql`) and deploy to a platform like Render or Heroku. Update `Gemfile` with `gem "pg"` and run `bundle install`.

Pro Tips

- Use `rails console` to experiment with your models in real-time.
- Run `rails generate --help` to see all the shortcuts Rails offers.
- Leverage gems like `devise` for authentication or `pundit` for authorization—add them to your `Gemfile` and configure as needed.

That's it! You've got a basic Rails app up and running. From here, explore the official Rails Guides (guides.rubyonrails.org) or build something real to solidify your skills. What kind of project are you thinking about?