Rails Project Guidelines

Basic Guidelines

Installation & Deployment

1. Use Docker & docker-compose to set up the local development/testing environment
2. Use Github for version control.
3. Integrate Github Actions for Continuous Integration (CI)
4. Deploy the application on Heroku or some PAAS.

Documentation

5. All API should have documentation. See Swagger.
6. Add documentation for main modules of the application, and for app architecture
7. Document complex methods

Testing

8. Use Test-Driven Development. Unit tests, Integration tests should be present.
9. Our preferred testing library is RSpec.
10. Measure test code coverage using appropriate libraries
11. Follow best coding conventions for tests, like - http://www.betterspecs.org/

Database Architecture

12. The DB schema should be normalized. Try to avoid denormalized columns
13. Add null constraints, referential integrity constraints, and uniqueness constraints where applicable.
14. Add indexes where applicable.
15. Order DB columns such that important foreign keys and required columns are at the start. Less important columns like timestamps are towards the bottom.

Programming

- Follow: https://rails.rubystyle.guide/
- Follow https://github.com/rubocop-hq/ruby-style-guide
- Use Rubocop https://githudustryb.com/rubocop-hq/rubocop
- Write DRY and scalable code during the development of any project.

- Use the REST principle and the Single Responsibility Principle
- Order variables and methods logically and/or alphabetically when possible.
- Do proper spacing and indentation.
- Do not make any spelling mistakes or grammatical mistakes. Use proper English words.
- Name of the variable/function/scope should reflect its purpose.
- Try to avoid comment messages unless the code is too complicated
- Try to use safe navigation operator instead of if and && conditions
- Always use the latest syntax.
- Use valid syntax. Use Rails 6+
- Use industry best practices: https://thoughtbot.com/upcase/clean-code
- For authentication, using gem- library.is strictly restricted. You have to do it manually.
- For image or file upload you can use Paperclip or CarrierWave gems.
- If you want to export PDF you can use Prawn Pdf and for tables, you can use prawn-table.

Bundler

- Put gems in alphabetical order in the Gemfile.
- Do not use any unnecessary gems in your project.
- Use only established gems in your projects. If you're contemplating on including some little-known gem you should do a careful review of its source code first.
- If your project needs different gems for different environments, then group it for that specific environment.

```
group :development, :test do
  gem 'mysql2'   #mysql for development
  gem 'pry-rails', "~> 0.3.4"
end

group :production do
  gem 'pg'    #postgresql for production
end
```

- Do not remove the `Gemfile.lock` from version control. This is not some randomly generated file - it makes sure that all of your team members get the same gem versions when they do a `bundle install`.

Migrations

- Keep the `schema.rb` under version control and arrange it well, so that it is clearly understandable.
- Add database level validation as well as model level validation
- Enforce foreign-key constraints.
- When writing constructive migrations (adding tables or columns), use the `change` method instead of `up` and `down` methods.
- If you are writing something within `up` migration, write it in the reverse order within `down` migration if needed.
- After migrating a migration, use the `rollback` command to check whether the database can return to its previous state or not.

Controllers

- Keep the controllers skinny - they should only retrieve data for the view layer and shouldn't contain any business logic (all the business logic should naturally reside in the model).
- Remove n + 1 query problems from all places in your project.
- Your code should be precise. Like, use flash in a single line.
- Use memoization where appropriate
- Use Service classes for external API/Service requests
- Use Form classes for complex form processing related code

Model

- Write your model simple and well-formatted. Group same type of method. Follow this coding pattern for models.
- Choose a proper name for methods, scopes etc.
- Properly you have to maintain the model validation and database level validation.
- Controllers should be thin.
- In the case of extra data operation or data manipulation, all logics should be written in the model.
- Instead of repeating blocks of code in a model, use model concerns.

Views

- Please do not use inline style.
- Try to avoid html_safe.
- Try to use partial instead of writing the same code multiple times.
- Double quotation rule and Indentation should be followed strictly.
- Proper Routes declaration should be maintained.
- View Helper Method should be used in case of any data operation/data logic in the view file.