Правила работы с Cursor и AI-инструментами в Ruby/Rails проектах с примерами промптов

Grok Version

by Thinknetika and Eugene Leontev

Thought for 52s

Мы, как команда из 10 опытных Ruby-разработчиков, ежедневно использующих Cursor и AI-инструменты, тщательно обсудили и составили список из 30 самых важных и эффективных правил для проектов на Ruby и Ruby on Rails. Эти правила основаны на нашем опыте, лучших практиках сообщества и стремлении к созданию качественного, поддерживаемого и производительного кода. Каждое правило кратко сформулировано и сопровождается пояснением, почему оно попало в топ.

Эти правила выбраны за их влияние на качество кода, производительность, безопасность и командную работу. Они отражают наш опыт и лучшие практики Ruby on Rails, помогая создавать надежные и масштабируемые приложения.

Топ-30 правил Cursor для проектов на Ruby и Ruby on Rails

Правила для базы данных и моделей

1. Используйте ActiveRecord для всех операций с базой данных

- **Хорошая практика**: Используйте методы ActiveRecord, такие как `User.find_by(email: params[:email])`, для безопасного и удобного доступа к данным.
- **Плохая практика**: Использование сырых SQL-запросов, например, `User.connection.execute("SELECT * FROM users WHERE email = '#{params[:email]}'")`, что уязвимо для SQL-инъекций.

2. Применяйте валидации в моделях

- **Хорошая практика**: Добавьте валидации в модель, например, `validates :email, presence: true, uniqueness: true` в модели User.
- **Плохая практика**: Проверка наличия и уникальности email в контроллере, что усложняет поддержку и тестирование.

3. Используйте модули concerns для разделения ответственности

- **Хорошая практика**: Выносите повторяющуюся логику в concerns, например, `module Searchable; def search; ...; end; end` и подключайте в модели.
- **Плохая практика**: Дублирование кода поиска в нескольких моделях.

4. Применяйте индексы в базе данных

- **Хорошая практика**: Добавьте индексы для часто используемых полей, например, `add_index :users, :email, unique: true`.
- **Плохая практика**: Отсутствие индексов на полях, по которым часто выполняются запросы, что замедляет работу приложения.

Правила для контроллеров и маршрутизации

1. Используйте сильные параметры в контроллерах

- **Хорошая практика**: Используйте `params.require(:user).permit(:email, :password)` для фильтрации входящих данных.
- **Плохая практика**: Присваивание всех параметров напрямую, например, `User.new(params[:user])`, что может привести к уязвимостям.

2. Применяйте RESTful маршрутизацию

- **Хорошая практика**: Используйте ресурсы в маршрутах, например, `resources :users`, для создания стандартных маршрутов.
- **Плохая практика**: Создание нестандартных маршрутов, например, `get '/get_user/:id', to: 'users#show'`, что усложняет понимание API.

Правила для представлений и фронтенда

1. Используйте Hotwire для динамических интерфейсов

- **Хорошая практика**: Используйте Turbo и Stimulus для обновления страниц без перезагрузки, например, `turbo_frame_tag` для частичных обновлений.
- **Плохая практика**: Использование большого количества JavaScript для динамических обновлений, что усложняет код.

2. Применяйте Tailwind CSS для стилизации

- **Хорошая практика**: Используйте утилитарные классы Tailwind, например, `<div class="flex justify-center">`, для быстрой стилизации.
- **Плохая практика**: Написание большого количества кастомного CSS, что замедляет разработку и поддержку.

3. Используйте view helpers и partials для DRY кода

- **Хорошая практика**: Выносите повторяющиеся части представлений в partials, например, `_form.html.erb` для форм.
- **Плохая практика**: Дублирование кода форм в нескольких представлениях.

Правила для тестирования и контроля качества

1. Пишите тесты с использованием RSpec или Minitest

- **Хорошая практика**: Напишите тесты для контроллеров, моделей и представлений, например, `expect(response).to have_http_status(:success)`.
- **Плохая практика**: Отсутствие тестов, что увеличивает риск ошибок в продакшене.

2. Используйте FactoryBot для тестовых данных

- **Хорошая практика**: Создавайте тестовые данные с помощью фабрик, например, `user = create(:user)`, для упрощения тестов.
- **Плохая практика**: Создание данных вручную в каждом тесте, что приводит к дублированию кода.

3. Используйте RuboCop для проверки стиля

- **Хорошая практика**: Настройте RuboCop для автоматической проверки стиля кода, например, `rubocop –a` для автокоррекции.
- **Плохая практика**: Игнорирование стиля кода, что затрудняет чтение и поддержку.

Правила для производительности и оптимизации

1. Применяйте сервисные объекты для сложной бизнес-логики

- **Хорошая практика**: Выносите сложную логику в сервисные объекты, например, `UserRegistrationService.new(user).call`, для улучшения читаемости.
- **Плохая практика**: Размещение всей бизнес-логики в контроллерах или моделях, что усложняет их.

2. Используйте фоновые задачи (Sidekiq) для длительных операций

- **Хорошая практика**: Выполняйте отправку email в фоне, например, `UserMailer.welcome_email.deliver_later`, для ускорения ответов.
- **Плохая практика**: Выполнение длительных операций в основном потоке, что замедляет интерфейс.

3. Применяйте кеширование для оптимизации производительности

- **Хорошая практика**: Кешируйте часто используемые данные, например, `Rails.cache.fetch("users_count", expires_in: 1.hour) { User.count }`.
- **Плохая практика**: Повторные запросы к базе данных для получения одних и тех же данных.

4. Используйте встроенные хелперы Rails

- **Хорошая практика**: Используйте хелперы, такие как `link_to`, `form_with`, для упрощения кода представлений.
- **Плохая практика**: Написание HTML вручную, что увеличивает объем кода и вероятность ошибок.

Правила для безопасности

1. Используйте исключения только для ошибок

- **Хорошая практика**: Используйте исключения для обработки ошибок, например, `raise ActiveRecord::RecordNotFound`, а не для управления потоком.
- **Плохая практика**: Использование исключений для управления логикой приложения, что усложняет отладку.

2. Применяйте Devise и Pundit для аутентификации и авторизации

- **Хорошая практика**: Используйте Devise для аутентификации и Pundit для авторизации, например, `before_action :authenticate_user!`.
- **Плохая практика**: Реализация собственной системы аутентификации, что может быть уязвимо.

3. Обеспечивайте безопасность (XSS, CSRF, SQL-инъекции)

- **Хорошая практика**: Используйте встроенные механизмы Rails, такие как `protect_from_forgery`, для защиты от CSRF.
- **Плохая практика**: Игнорирование мер безопасности, что может привести к уязвимостям.

Правила для стиля кода и лучших практик

1. Следуйте Ruby Style Guide

- **Хорошая практика**: Пишите код в соответствии с руководством, например, используйте 2 пробела для отступов.
- **Плохая практика**: Непоследовательный стиль кода, что затрудняет чтение.

2. Используйте описательные имена переменных и методов

- **Хорошая практика**: Называйте методы понятно, например, `def calculate_total_price`, а не `def calc`.
- **Плохая практика**: Использование неинформативных имен, таких как `def x`, что усложняет понимание кода.

3. Следуйте принципу единственной ответственности

- **Хорошая практика**: Каждый класс выполняет одну задачу, например, класс `User` только для управления пользователями.
- **Плохая практика**: Классы с множеством несвязанных методов, что усложняет поддержку.

4. Избегайте дублирования через итерацию и модуляризацию

- **Хорошая практика**: Используйте циклы и модули для устранения дублирования, например, `each` вместо повторения кода.
- **Плохая практика**: Копирование и вставка кода, что приводит к избыточности.

5. Применяйте ООП и функциональное программирование

- **Хорошая практика**: Используйте классы и модули для организации кода, а также функциональные методы, такие как `map`, `reduce`.
- **Плохая практика**: Написание процедурного кода без структуры, что затрудняет масштабирование.

6. Используйте snake_case для имен файлов и переменных

- **Хорошая практика**: Называйте файлы и переменные в snake_case, например, `user_profile.rb`, `first_name`.
- **Плохая практика**: Использование camelCase или других стилей, что не соответствует конвенциям Ruby.

7. Используйте CamelCase для классов и модулей

- **Хорошая практика**: Называйте классы и модули в CamelCase, например, `UserProfile`, `Searchable`.
- **Плохая практика**: Использование snake_case для классов, что не соответствует стандартам.

Правила для конвенций и стандартов Rails

1. Следуйте конвенциям Rails

- **Хорошая практика**: Используйте стандартные имена контроллеров, моделей и представлений, например, `UsersController`.
- **Плохая практика**: Использование нестандартных имен, что усложняет понимание структуры приложения.

2. Используйте YAML для конфигураций

- **Хорошая практика**: Храните конфигурации в YAML-файлах, например, `database.yml`, для удобства управления.
- **Плохая практика**: Хранение конфигураций в коде, что усложняет изменения.

3. Используйте возможности Ruby 3.x

- **Хорошая практика**: Используйте новые функции, такие как pattern matching, для упрощения кода.
- **Плохая практика**: Игнорирование новых возможностей, что может привести к более громоздкому коду.

4. Следуйте структуре MVC

- **Хорошая практика**: Разделяйте код на модели, контроллеры и представления, например, бизнес-логика в моделях, а не в контроллерах.
- **Плохая практика**: Смешивание логики в контроллерах и представлениях, что усложняет поддержку.


THINKNETICA
онлайн-школа для разработчиков