



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

Cursor Auto Version

by [Thinknetika](#) and [Eugene Leontev](#)

© 2025 Thinknetika, Eugene Leontev

Ответ cursor (model: auto)

Я проведу коллективное обсуждение с командой опытных Ruby/Rails разработчиков и составлю список из 30 ключевых правил для работы с Cursor и AI в Ruby on Rails проектах.

Отлично! Я вижу, что у вас уже есть структура для хранения правил по разным AI-инструментам. Теперь проведу коллективное обсуждение с командой опытных Ruby/Rails разработчиков и составлю топ-30 правил для работы с Cursor и AI.

Ruby/Rails Cursor Rules: Топ-30 правил для эффективной разработки



Содержание

- Настройка и конфигурация
- Работа с AI и промптами
- Архитектура и структура кода
- Модели и база данных
- Контроллеры и API
- Тестирование и качество
- Документация и коммуникация
- DevOps и деплой

Настройка и конфигурация

1. Создавай `.cursorrules` в корне каждого проекта

Определяй специфичные для проекта правила: стиль кода, архитектурные паттерны, naming conventions.

Хорошие промпты:

"Создай `.cursorrules` файл для Rails проекта с правилами: используй `snake_case` для методов, `camelCase` для классов, добавляй комментарии к публичным методам"

Плохие практики:

- Использование общих правил для всех проектов
- Отсутствие специфичных для домена правил
- Необновляемые правила

2. Создавай AI-специфичные .gitignore правила

Исключай временные AI файлы, логи генерации, кэш Cursor.

✓ Хорошие промпты:

```
"Добавь в .gitignore правила для AI файлов: .cursor_cache/, ai_generated_*.rb, temp_ai_*.md"
```

✗ Плохие практики:

- Игнорирование всех .md файлов
- Отсутствие правил для временных AI файлов
- Попадание служебных файлов в репозиторий

3. Создавай AI-дружественную структуру папок

Организуй код по принципам, которые AI может легко понять: `models/`, `services/`, `controllers/`, `views/`.

✓ Хорошие промпты:

"Реорганизуй структуру проекта по принципу: `models/` для ActiveRecord, `services/` для бизнес-логики, `controllers/` для HTTP логики"

✗ Плохие практики:

- Смешивание разных типов файлов в одной папке
- Неочевидные названия папок
- Отсутствие четкой иерархии

Работа с AI и промптами

4. Структурируй промпты по принципу "Context + Task + Constraints"

Начинай с контекста (файл, класс), затем задачу, затем ограничения.

Хорошие промпты:

"В файле `app/models/user.rb` есть класс `User` с методами `authenticate` и `validate_email`. Добавь валидацию email с учетом Rails conventions и существующего стиля кода"

Плохие практики:

- "Создай валидацию email" (без контекста)
- "Сделай что-то с User моделью" (неопределенная задача)
- Отсутствие ограничений и требований

5. Используй AI для рефакторинга, а не для создания с нуля

AI лучше работает с существующим кодом, чем с пустым файлом.

✓ Хорошие промпты:

"Рефактори этот метод `authenticate` в `User` модели, добавь обработку ошибок и логирование"

✗ Плохие практики:

- "Создай весь `User` контроллер с нуля"
- "Напиши всю бизнес-логику приложения"
- Отсутствие базовой структуры перед обращением к AI

6. Создавай AI-дружественные комментарии в коде

Пиши комментарии, которые объясняют "почему", а не "что".

✓ Хорошие промпты:

"Добавь комментарии к публичным методам в User модели, объясняющие бизнес-логику и edge cases"

✗ Плохие практики:

- Комментарии типа "# проверяет email"
- Отсутствие комментариев к сложной логике
- Устаревшие комментарии

7. Используй AI для генерации тестов, а не бизнес-логики

Поручай AI создавать RSpec тесты, factory definitions, но критическую бизнес-логику пиши самостоятельно.

✓ Хорошие промпты:

"Создай RSpec тесты для User#authenticate метода, включая edge cases и factory для User"

✗ Плохие практики:

- "Напиши всю бизнес-логику аутентификации"
- "Создай сложный алгоритм шифрования"
- Доверие AI критической логике без проверки



Архитектура и структура кода

8. Используй AI для генерации сервисных объектов

Поручай AI создавать service classes с правильной структурой, но бизнес-логику определяй самостоятельно.



Хорошие промпты:

"Создай сервисный объект `PaymentProcessor` с методами `process_payment` и `refund_payment`, используя стандартную Rails структуру"



Плохие практики:

- "Напиши всю логику обработки платежей"
- Создание сервисов без четкой ответственности
- Смешивание разных доменов в одном сервисе

9. Создавай AI-дружественные имена методов и переменных

Используй описательные имена, которые AI может легко понять и использовать в контексте.

✓ Хорошие промпты:

```
"Переименуй методы в User модели, используя описательные имена:  
auth -> authenticate_user, val_email -> validate_email_format"
```

✗ Плохие практики:

- Использование аббревиатур (usr, pwd, auth)
- Неочевидные сокращения
- Однобуквенные переменные

10. Используй AI для генерации view helpers и partials

Поручай AI создавать reusable компоненты, но дизайн и UX определяй самостоятельно.

✓ Хорошие промпты:

"Создай helper метод `format_currency` для отображения цен и `partial _product_card` для отображения товаров"

✗ Плохие практики:

- "Создай весь дизайн приложения"
- Смешивание логики и представления
- Дублирование кода в partials

11. Используй AI для создания background jobs

Используй AI для генерации Sidekiq/ActiveJob классов, но сложную логику обработки пиши вручную.

✓ Хорошие промпты:

```
"Создай background job EmailNotificationJob для отправки уведомлений с retry логикой и error handling"
```

✗ Плохие практики:

- "Напиши всю логику обработки email"
- Создание jobs без обработки ошибок
- Отсутствие retry механизмов

12. Используй AI для генерации middleware и initializers

Поручай AI создавать custom middleware, но критическую логику безопасности и производительности проверяй вручную.

✓ Хорошие промпты:

```
"Создай middleware для логирования запросов и initializer для настройки Redis соединения"
```

✗ Плохие практики:

- "Напиши всю security логику"
- Создание middleware без тестирования
- Отсутствие обработки ошибок

Модели и база данных

13. Используй AI для генерации валидаций и ассоциаций

Поручай AI создавать `validates`, `belongs_to`, `has_many` на основе схемы БД.

Хорошие промпты:

"Добавь валидации в User модель: email должен быть уникальным и валидным, password минимум 8 символов"

Плохие практики:

- "Напиши всю модель с нуля"
- Отсутствие валидаций
- Неправильные ассоциации

14. Применяй AI для миграций и seed данных

Используй AI для создания миграций, seed файлов, fixtures.

✓ Хорошие промпты:

"Создай миграцию для добавления поля phone_number в таблицу users и seed данные для тестирования"

✗ Плохие практики:

- "Напиши всю схему БД"
- Отсутствие rollback логики
- Небезопасные seed данные

15. Используй AI для генерации конфигурационных файлов

Поручай AI создавать `database.yml`, `application.rb`, `production.rb`, но секреты и специфичные настройки добавляй вручную.

✓ Хорошие промпты:

"Создай `database.yml` для development и test окружений с правильными настройками PostgreSQL"

✗ Плохие практики:

- "Напиши все конфигурации с секретами"
- Хардкод секретов в файлах
- Отсутствие environment-specific настроек

Контроллеры и API

16. Применяй AI для создания контроллеров по REST принципам

Используй AI для генерации стандартных CRUD операций, но кастомную логику пиши вручную.

Хорошие промпты:

"Создай REST контроллер для Products с CRUD операциями, используя strong parameters и proper error handling"

Плохие практики:

- "Напиши всю логику контроллера"
- Отсутствие strong parameters
- Неправильная обработка ошибок

17. Используй AI для генерации API документации

Поручай AI создавать Swagger/OpenAPI спецификации на основе контроллеров.

✓ Хорошие промпты:

"Создай OpenAPI спецификацию для Users API с описанием всех endpoints и параметров"

✗ Плохие практики:

- "Напиши всю документацию с нуля"
- Отсутствие примеров запросов
- Неактуальная документация

18. Используй AI для создания консольных команд

Используй AI для генерации Rails console helpers, но сложную логику проверяй вручную.

✓ Хорошие промпты:

```
"Создай консольные команды для управления пользователями: create_admin, bulk_import_users"
```

✗ Плохие практики:

- "Напиши всю административную логику"
- Отсутствие валидации входных данных
- Небезопасные команды



Тестирование и качество

19. Используй AI для code review, но проверяй критически

AI может найти очевидные проблемы, но не доверяй ему полностью.



Хорошие промпты:

"Проведи code review этого User контроллера, найди потенциальные security issues и performance проблемы"



Плохие практики:

- Полное доверие AI без проверки
- Игнорирование security аспектов
- Отсутствие manual review

20. Создавай AI-дружественные error handling

Используй стандартные Rails error handling паттерны, которые AI может легко понять и воспроизвести.

✓ Хорошие промпты:

"Добавь proper error handling в PaymentService с использованием стандартных Rails rescue блоков"

✗ Плохие практики:

- "Напиши всю error handling логику"
- Нестандартные исключения
- Отсутствие логирования ошибок

21. Используй AI для генерации cache strategies

Поручай AI создавать Redis/Memcached интеграции, но критичные cache invalidation стратегии определяй самостоятельно.

✓ Хорошие промпты:

"Создай cache стратегию для Product модели с использованием Redis, включая cache keys и TTL"

✗ Плохие практики:

- "Напиши всю cache логику"
- Отсутствие cache invalidation
- Неправильные cache keys

Документация и коммуникация

22. **Используй AI для документации API и README**

Поручай AI генерировать документацию на основе существующего кода.

Хорошие промпты:

"Создай README для проекта с описанием setup, API endpoints и deployment инструкциями"

Плохие практики:

- "Напиши всю документацию с нуля"
- Отсутствие примеров использования
- Неактуальная информация

23. Создавай AI-дружественную документацию проекта

Пиши README и документацию в формате, который AI может легко понять и обновить.

✓ Хорошие промпты:

```
"Обнови README с добавлением новой секции о background jobs и обнови API документацию"
```

✗ Плохие практики:

- Неструктурированная документация
- Отсутствие стандартных секций
- Устаревшая информация

24. Используй семантические коммиты с префиксами AI

Структурируй коммиты как `feat(ai): add GPT-4 integration for code review`.

✓ Хорошие промпты:

"Создай commit message для изменений в User модели с добавлением AI-generated валидаций"

✗ Плохие практики:

- "fix stuff" или "update code"
- Отсутствие описания изменений
- Неинформативные сообщения

DevOps и деплой

25. **Используй AI для генерации deployment scripts**

Поручай AI создавать Capistrano/Heroku конфигурации, но production секреты и специфичные настройки добавляй вручную.

Хорошие промпты:

```
"Создай Capistrano конфигурацию для production деплоя с правильными environment variables"
```

Плохие практики:

- "Напиши все deployment настройки с секретами"
- Хардкод production данных
- Отсутствие rollback стратегии

26. Применяй AI для создания monitoring и logging

Используй AI для генерации New Relic/Sentry интеграций, но критичные алерты и метрики настраивай вручную.

✓ Хорошие промпты:

"Создай конфигурацию для New Relic monitoring с базовыми метриками и алертами"

✗ Плохие практики:

- "Напиши всю monitoring логику"
- Отсутствие error tracking
- Неправильные алерты

27. **Используй AI для генерации security headers**

Поручай AI создавать security middleware, но сложные security policies определяй самостоятельно.

Хорошие промпты:

```
"Создай security headers middleware с CSP, HSTS и XSS protection"
```

Плохие практики:

- "Напиши всю security логику"
- Отсутствие security headers
- Неправильные security настройки

28. Создавай AI-дружественные environment variables

Используй стандартные имена для env переменных, которые AI может легко понять и использовать.

✓ Хорошие промпты:

```
"Создай .env.example файл с описанием всех environment variables для проекта"
```

✗ Плохие практики:

- Неочевидные имена переменных
- Отсутствие документации
- Хардкод значений

29. Используй AI для создания rake tasks

Используй AI для генерации административных задач, но сложную бизнес-логику в них пиши самостоятельно.

✓ Хорошие промпты:

"Создай rake task для импорта пользователей из CSV файла с валидацией данных"

✗ Плохие практики:

- "Напиши всю бизнес-логику в rake task"
- Отсутствие валидации
- Небезопасные операции

30. **Используй AI для генерации mailer templates**

Поручай AI создавать email шаблоны, но дизайн и брендинг определяй самостоятельно.

Хорошие промпты:

`"Создай mailer template для welcome email с HTML и text версиями"`

Плохие практики:

- "Создай весь email дизайн"
- Отсутствие text версии
- Неправильные ссылки



Заключение

Эти правила обеспечивают оптимальный баланс между продуктивностью AI-инструментов и качеством кода. Ключевые принципы:

1. **AI для рутины, человек для критики** - используй AI для повторяющихся задач
2. **Контекст превыше всего** - всегда предоставляй полный контекст AI
3. **Проверяй критически** - не доверяй AI полностью в важных решениях
4. **Стандартизируй** - используй общепринятые паттерны и конвенции
5. **Документируй** - создавай понятную документацию для AI и людей

Следуя этим правилам, команда может значительно повысить продуктивность разработки, сохраняя при этом высокое качество кода и архитектурную целостность проекта.

THiNKNETCSA

онлайн-школа для разработчиков