

Open in app ↗

Medium

 Search Write

Building an AI-Powered Content Summarizer in Rails: A Step-by-Step Guide

Ronak Bhatt · [Follow](#)

3 min read · Dec 17, 2024



1



Building AI-powered features doesn't have to be complicated. In this tutorial, we'll create a practical Rails application that automatically generates summaries for articles using OpenAI's GPT model. By the end of this guide, you'll have a fully functional content summarizer that can be integrated into any Rails application.



Why Add AI-Powered Summaries?

Before diving into the code, let's consider why you might want to add automatic summarization:

- Improve user experience by providing quick previews of longer content
- Save editorial time by automating summary generation
- Create consistent summary lengths across all articles
- Generate SEO-friendly meta descriptions automatically

Setting Up the Project

First, let's set up our Rails project with the necessary gems. Add these to your Gemfile:

```
gem 'ruby-openai' # For OpenAI API integration
```

```
gem 'dotenv-rails' # For managing environment variables
```

Run `bundle install` to install the dependencies.

Database Structure

We'll need a simple articles table that includes fields for our content and its AI-generated summary:

```
class CreateArticles < ActiveRecord::Migration[7.1]
  def change
    create_table :articles do |t|
      t.string :title
      t.text :content
      t.text :summary
      t.timestamps
    end
  end
end
```

Run `rails db:migrate` to create the table.

The Magic: Adding AI Summarization

Here's where the real magic happens. We'll create an Article model that automatically generates summaries whenever content is changed:

```
class Article < ApplicationRecord
  validates :title, presence: true
  validates :content, presence: true

  after_save :generate_summary, if: :content_changed?
```

```
private

def generate_summary
  return if content.blank?

  client = OpenAI::Client.new(access_token: ENV['OPENAI_API_KEY'])

  prompt = "Please summarize the following text in 2-3 sentences: #{content}"

  response = client.chat(
    parameters: {
      model: "gpt-3.5-turbo",
      messages: [{ role: "user", content: prompt }],
      max_tokens: 150,
      temperature: 0.7
    }
  )

  update_column(:summary, response.dig("choices", 0, "message", "content"))
rescue => e
  Rails.logger.error("Failed to generate summary: #{e.message}")
end
end
```

Let's break down what's happening here:

1. We use an `after_save` callback to trigger summary generation when content changes
2. The `generate_summary` method connects to OpenAI's API
3. We use GPT-3.5-turbo to generate a concise 2–3 sentence summary
4. The summary is saved directly to the database using `update_column` to avoid triggering callbacks
5. Error handling ensures our application stays robust

The Controller Layer

Our ArticlesController handles the basic CRUD operations:

```
class ArticlesController < ApplicationController
  def index
    @articles = Article.all
  end

  def show
    @article = Article.find(params[:id])
  end

  def new
    @article = Article.new
  end

  def create
    @article = Article.new(article_params)
    if @article.save
      redirect_to @article, notice: 'Article created with AI-generated summary!'
    else
      render :new
    end
  end

  private

  def article_params
    params.require(:article).permit(:title, :content)
  end
end
```

The View Layer

Here's a simple but effective view for displaying articles with their AI-generated summaries:

```
<div class="article">
  <h1><%= @article.title %></h1>
```

```
<div class="summary-box">
  <h3>AI Summary</h3>
  <p><%= @article.summary %></p>
</div>

<div class="content">
  <%= simple_format(@article.content) %>
</div>
</div>
```

Best Practices and Considerations

When implementing AI features in your Rails application, keep these points in mind:

1. **Error Handling:** Always implement robust error handling for API calls
2. **Rate Limiting:** Consider implementing rate limiting if you're generating many summaries
3. **Caching:** Cache summaries to avoid unnecessary API calls
4. **Cost Management:** Monitor your API usage as OpenAI charges based on token usage
5. **Content Quality:** Regularly review generated summaries to ensure quality

Future Enhancements

This basic implementation can be extended in several ways:

- Add a background job for summary generation using Sidekiq
- Implement summary regeneration functionality

- Add length controls for summaries
- Include sentiment analysis along with summarization
- Add support for multiple languages

Conclusion

Building AI-powered features in Rails doesn't have to be complicated. With just a few lines of code, we've created a powerful content summarization system that can be extended and customized to meet various needs.

Remember to store your OpenAI API key in your `.env` file:

```
OPENAI_API_KEY=your_key_here
```

This article is part of our series on modernizing Rails applications with AI capabilities. Stay tuned for more tutorials on integrating AI features into your Rails apps.

Ruby



Written by Ronak Bhatt

101 Followers · 0 Following

Follow

A software engineer with solid experience mostly with Ruby on rails and Javascripts.

No responses yet



What are your thoughts?

Respond

More from Ronak Bhatt



Rails 8.0 Beta 1: No PaaS Required



Ronak Bhatt

What's New in Ruby on Rails 8

The first Rails 8 beta has officially been released, bringing an exciting set of features...

Oct 16, 2024 🖱 13



RAILS 6 & 7 API
Authentication with JWT
(Token-based authentication)



Ronak Bhatt


RAILS 6 & 7 API Authentication with JWT (Token-based...

What is JWT?

May 29, 2024 🖱 53 🗨 3





 Ronak Bhatt

Enhancing Ruby on Rails Applications with AI-Powered Dat...

Introduction


Oct 16, 2024  11



RUBY

on

RAILS 8.0

 Ronak Bhatt

Exploring Rails 8: A Look at the Latest Features

As the newest addition to the Rails family, Rails 8 introduces features designed to...

Nov 5, 2024  7  1



See all from Ronak Bhatt

Recommended from Medium





Bhavesh Saluja

Behind the Scenes: Anatomy of a Ruby Gem

Ruby gems are the building blocks of the Ruby on Rails ecosystem, providing reusabl...



Nov 18, 2024



Harendra

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free



Oct 26, 2024



8.5K



129

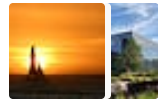


Lists



Staff picks

796 stories · 1561 saves



Stories to Help You Level-Up at Work

19 stories · 912 saves



Self-Improvement 101

20 stories · 3191 saves



Productivity 101

20 stories · 2706 saves



In Ruby on Rails by Rails to Rescue

Action Cable: Beyond Chat Applications, Ruby on Rails

Action Cable, Rails' robust WebSocket framework, isn't just for simple chat...



Nov 1, 2024



4



Patryk Rogala

Understanding Maintenance Tasks in Rails

Introduction




Nov 19, 2024



12



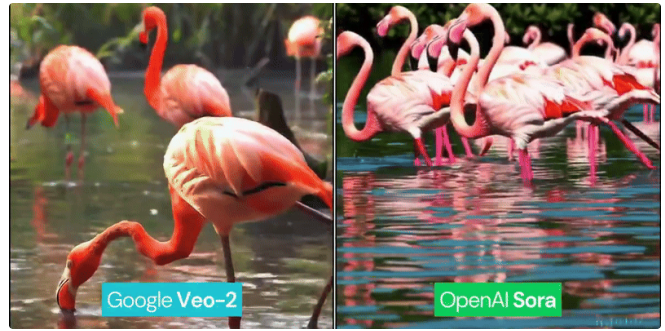


 Jessica Stillman

Jeff Bezos Says the 1-Hour Rule Makes Him Smarter. New...

Jeff Bezos's morning routine has long included the one-hour rule. New...

★ Oct 30, 2024 🖱 18.7K 💬 478 📌⁺ ⋮



 In Coding Beauty by Tari Ibaba

Google really destroyed OpenAI and Sora without even trying

Just when Sam Altman thought they were far ahead of the competition with Sora...

★ Dec 22, 2024 🖱 2.1K 💬 51 📌⁺ ⋮

See more recommendations