RoR - Active Storage

# Overview

Active Storage facilitates the uploading of files by a user to local or cloud storage and attaching them to Active Record objects. Active Storage also has the ability to transform image uploads with ImageMagick, and generate image represenations of PDFs and videos.

# Setup

Database

Active Storage uses two tables in the application database: active\_storage\_blobs and active\_storage\_attachments. To create these tables run:

rails active\_storage:install

Config

Active Storage configuration credentials are defined in config/storage.yml. Use rails crendetials to fill in sensitive details securely.

In each application environment the storage location can be set:

#dev

config.active\_storage.service = :local

#test

config.active\_storage.service = :test

#prod

config.active\_storage.service = :amazon

Amazon S3

To use the amazon s3 storage service, add the aws s3 gem to the Gemfile

gem 'aws-sdk-s3', require 'false'

Create the s3 bucket and IAM credentials with the following minimum permissions:

* ListBucket
* PutObject
* GetObject
* DeleteObject

Other Cloud Providers

Active Storage has adapters for many other cloud storage services such as:

* digitalocean
* azure
* google

<https://edgeguides.rubyonrails.org/active_storage_overview.html>

# Uploads

has\_one\_attached

The has\_one\_attached macro can be used to map one file to an Active Record record. To use add the macro to the model:

has\_one\_attached :<file-name>

Then upload using the rails helper method:

<%= form.file\_field :<file-name>, accept: 'image/png,image/gif,image/jpeg'%>

The file can then be permitted as a strong parameter and added to a model as normal during record creation. To update an exisiting record, use the attach method:

model.<file-name>.attach(params[:<file-name>])

To check if a file is attached use the attached? method:

model.<file-name>.attached?

has\_many\_attached

The has\_many\_attached macro can be used to map many files to one record. To use add the macro to the model:

has\_many\_attached :<file-name-plural>

Create an array of files in the html form:

file\_field(:post, :images, multiple: true, accept: 'image/png,image/gif,image/jpeg')

The files can then be permitted as a strong parameters, for example:

params.require(:message).permit(:title, :content, images: [])

Then files can be added to the model using the attach method:

model.<file-name-plural>.attach(params[:<file-name-plural>])

# Processing

Removing Files

To remove a file simply call the purge method on the record:

record.<file-name>.purge

To perform the purge command asynchronously use:

record.<file-name>.purge\_later

Download

To process a file after it has been uploaded, it can be download to binary and stored in system memory:

binary = user.avatar.download

To download to local disk use:

record.<file-name>.open do |file|

system '/path/to/save', file.path

# ...

end

# Using Files

Link to File

Use the url\_for helper method to create a url for the attached file of a record:

url\_for(record.<file-name>)

A download link can be created using:

rails\_blob\_path(user.<file-name>, disposition: "attachment")

Link can also be created out of context using:

Rails.application.routes.url\_helpers.rails\_blob\_path(record.<file-name>, only\_path: true)

# Image Processing

Active Storage currently provides image processing by the ImageProcessing gem. Processing can be performed to uploaded images, allowing for the many different types of transformation provided by MiniMagick.

When transformations are applied to an image blob, such as creating a thumbnail or fixed size avatars, they are saved as a variant to the image and reuploaded to the storage service.

Setup

To transform images first add the image\_processing gem to the application:

gem 'image\_processing'

This sets the default image processor as ImageMagick using the MiniMagick gem. Therefore, the system need to have ImageMagick installed for the gem to function properly:

brew install imagemagick

It is also possible to use the vips processor, by setting:

config.active\_storage.variant\_processor = :vips

Processing

Images can be processed in the controller during upload, for example using:

params[:cat][:images].each do |image|

mini\_image = MiniMagick::Image.new(image.tempfile.path)

mini\_image.resize '1200x1200'

end

Images can also be processed on demand using the variant method, however since processing requires full download of the image to the server and Active Storage lazily transforms the image, this can be slow for the initial process and view:

<%= image\_tag user.avatar.variant(resize\_to\_limit: [100, 100]) %>

Therefore, it is common to perform standard transformations using a background worker such as sidekiq or delayedjob. Use the after\_commit callback to run a job processing the image:

after\_commit :your\_method, if: :persisted?

Inside the job, use the variant processed method to start the processing of the variant. For example:

user.avatar.variant(resize\_to\_limit: [100, 100]).processed

Once processed the variant will be uploaded to the storage service and retrived if the same variant method is called again.

Note, the after\_save callback will be called once the blob is created, but before it is saved to the disk or service.

Removing MetaData

Removing metadata from images can be performed using the before\_save callback to pre-process an attachment before saving it to the database. For example:

before\_save :strip\_exif\_data

private

def strip\_exif\_data

return unless image.attached?

filename = image.filename.to\_s

attachment\_path = "#{Dir.tmpdir}/#{image.filename}"

File.open(attachment\_path, 'wb') do |file|

file.write(image.download)

file.close

end

mm\_image = MiniMagick::Image.open(attachment\_path)

mm\_image.strip

mm\_image.write attachment\_path

image.attach(io: File.open(attachment\_path), filename: filename)

end

<https://stackoverflow.com/questions/50844197/how-to-compress-images-before-uploading-to-the-cloud-using-activestorage>

# Other File Types

Previews

Non-image files can be previewed using the preview method. For example:

<ul>

<% @message.files.each do |file| %>

<li>

<%= image\_tag file.preview(resize\_to\_limit: [100, 100]) %>

</li>

<% end %>

</ul>

By default Active Storage can preview videos and PDFs, however many third party plugins can be used to preview other file types.