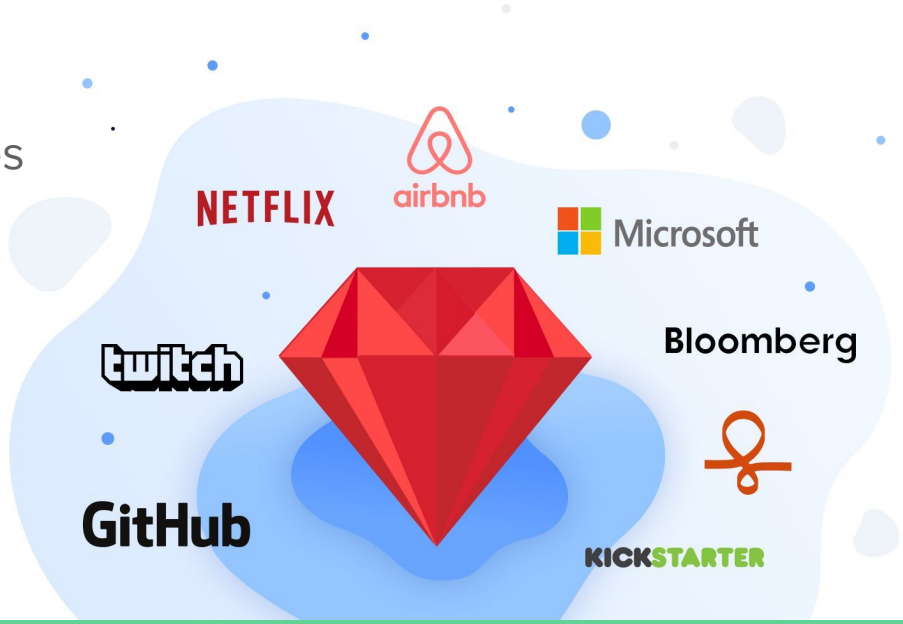
Ruby on Rails

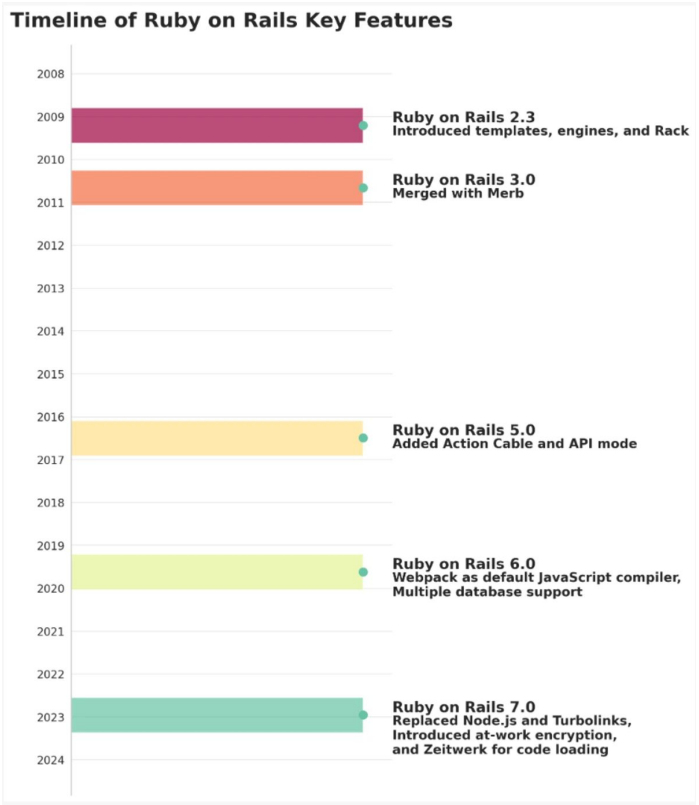
Web Application Framework

Ruby on Rails

- Web application framework
- Model-View-Controller (MVC) architecture
- Convention over Configuration - write configuration only for deviations
- Don't Repeat Yourself (DRY)
- SOLID object oriented design principles
- Easy to setup and build prototypes
- Active developer community
- Usage:
  - Airbnb
  - GitHub
  - Shopify
  - Hulu



History

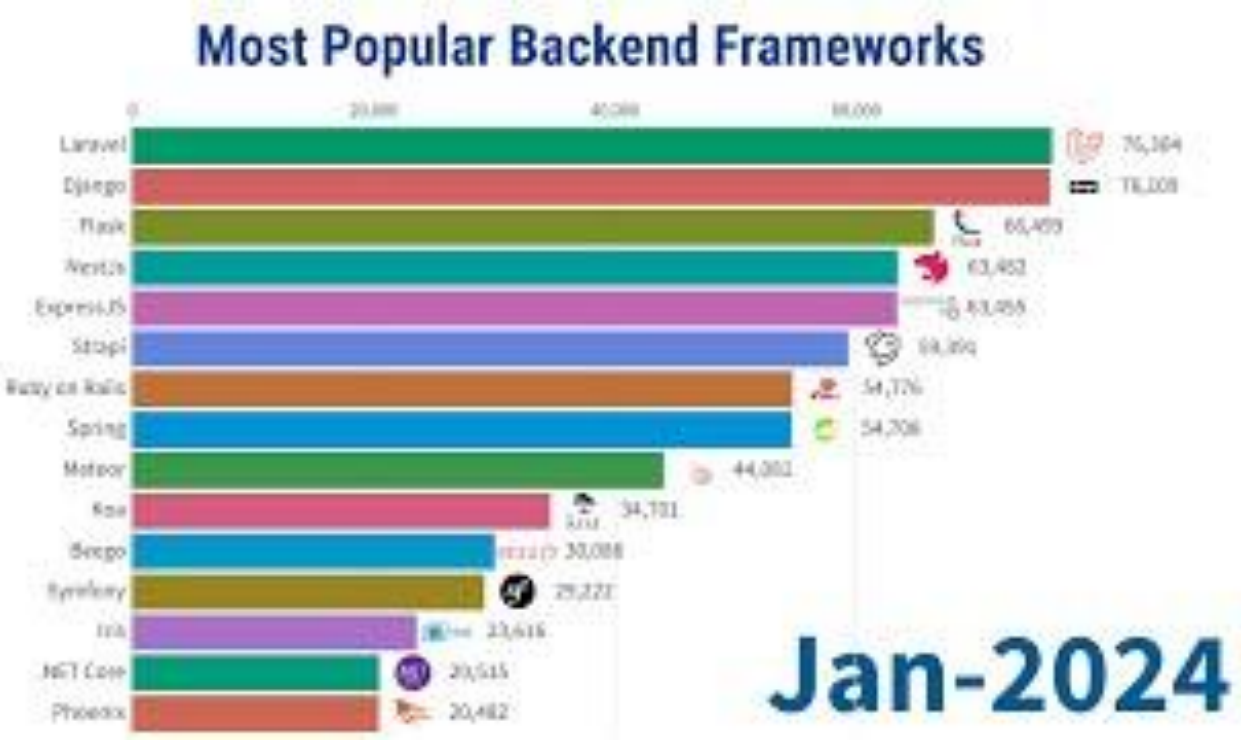


Version History

Version	Release Date	Compatible Ruby Version(s)[29][30][31]
1.0[32]	13 December 2005	1.8.6
1.2[33]	19 January 2007	1.8.6
2.0[34]	7 December 2007	1.8.6
2.1[35]	31 May 2008	1.8.6
2.2[36]	21 November 2008	1.8.7 recommended; 1.8.6 possible
2.3[37]	16 March 2009	1.8.7 recommended; 1.8.6 and 1.9.1 possible
3.0[38]	29 August 2010	1.9.3 recommended; 1.8.7 and 1.9.2 possible
3.1[39]	31 August 2011	1.9.3 recommended; 1.8.7 and 1.9.2 possible
3.2[40]	20 January 2012	1.9.3 recommended; 1.8.7 and 1.9.2 possible
4.0[41]	25 June 2013	2.0 preferred; 1.9.3 or newer required
4.1[19]	8 April 2014	2.0 preferred; 1.9.3 or newer required
4.2[20]	19 December 2014	2.0 preferred; 1.9.3 or newer required
5.0[21]	30 June 2016	2.2.2 or newer
5.1[22]	10 May 2017	2.2.2 or newer
5.2[23]	9 April 2018	2.2.2 or newer
6.0[25]	16 August 2019	2.5.0 or newer
6.1[26]	9 December 2020	2.5.0 or newer
7.0[27]	15 December 2021	2.7.0 or newer
7.1[42]	5 October 2023	2.7.0 or newer
7.2[43]	2024[44]	3.1.0 or newer[45]
8.0[46]	2024[46]	TBC

Old version | Older version, still maintained | **Latest version** | Future release

Popularity

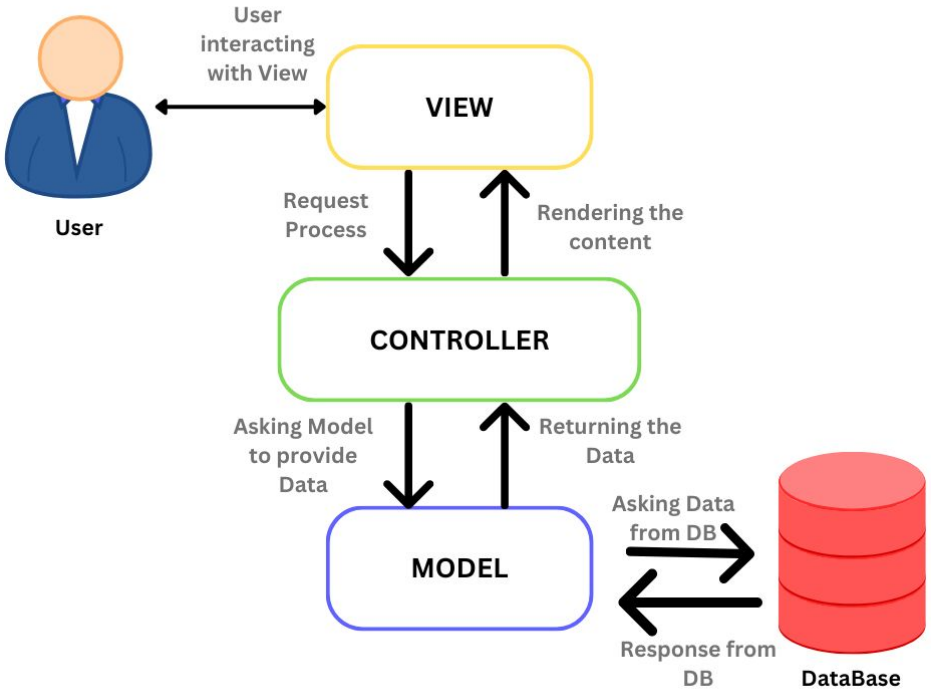

Most Popular Backend Frameworks
Laravel
Django
Flask
Nest.js
Express.JS
Strapi
Ruby on Rails
Spring
Meteor
Koa
Sinatra
Symfony
Fastify
.NET Core
Phoenix
Jan-2024

SOLID

- Single Responsibility Principle
  - Class should have only one reason to change
  - Class should have only one job or responsibility
- Open/Closed Principle
  - Abstraction should be open for extension but closed for modification
- Liskov Substitution Principle
  - Objects of a superclass should be replaceable with objects of a subclass
- Interface Segregation Principle
  - Interfaces should be splitted if not fully used
- Dependency Inversion Principle
  - High-level modules should not depend on low-level modules
  - Abstractions should not depend on details. Details should depend on abstractions.

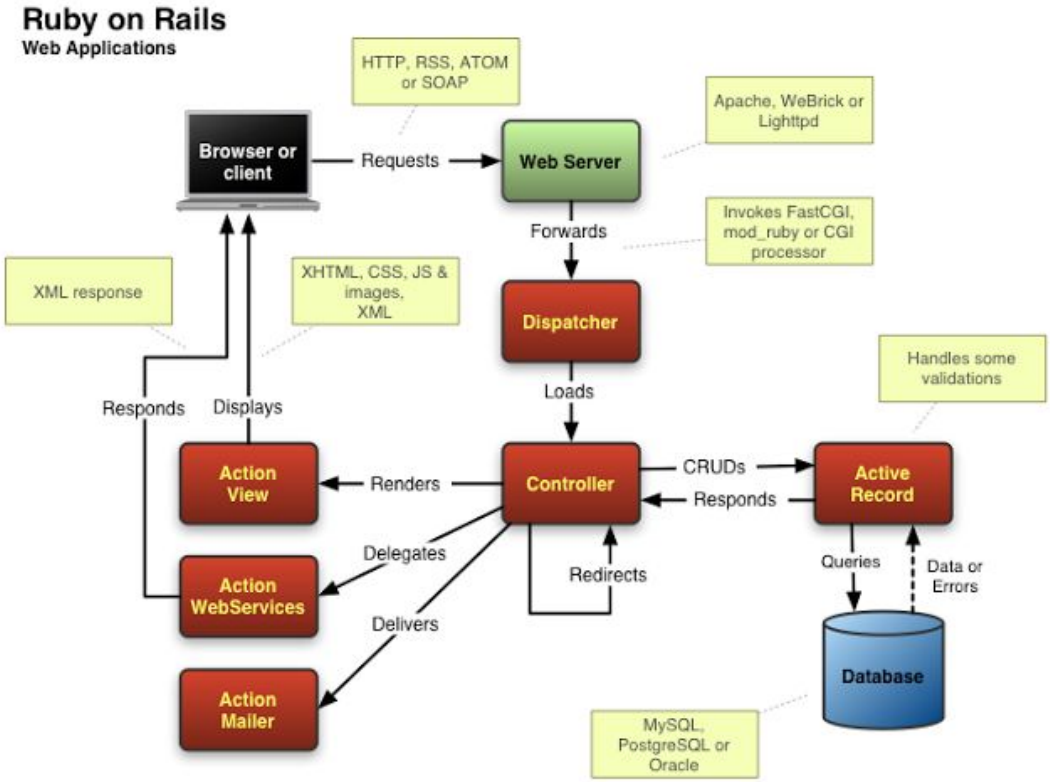
Model-View-Controller

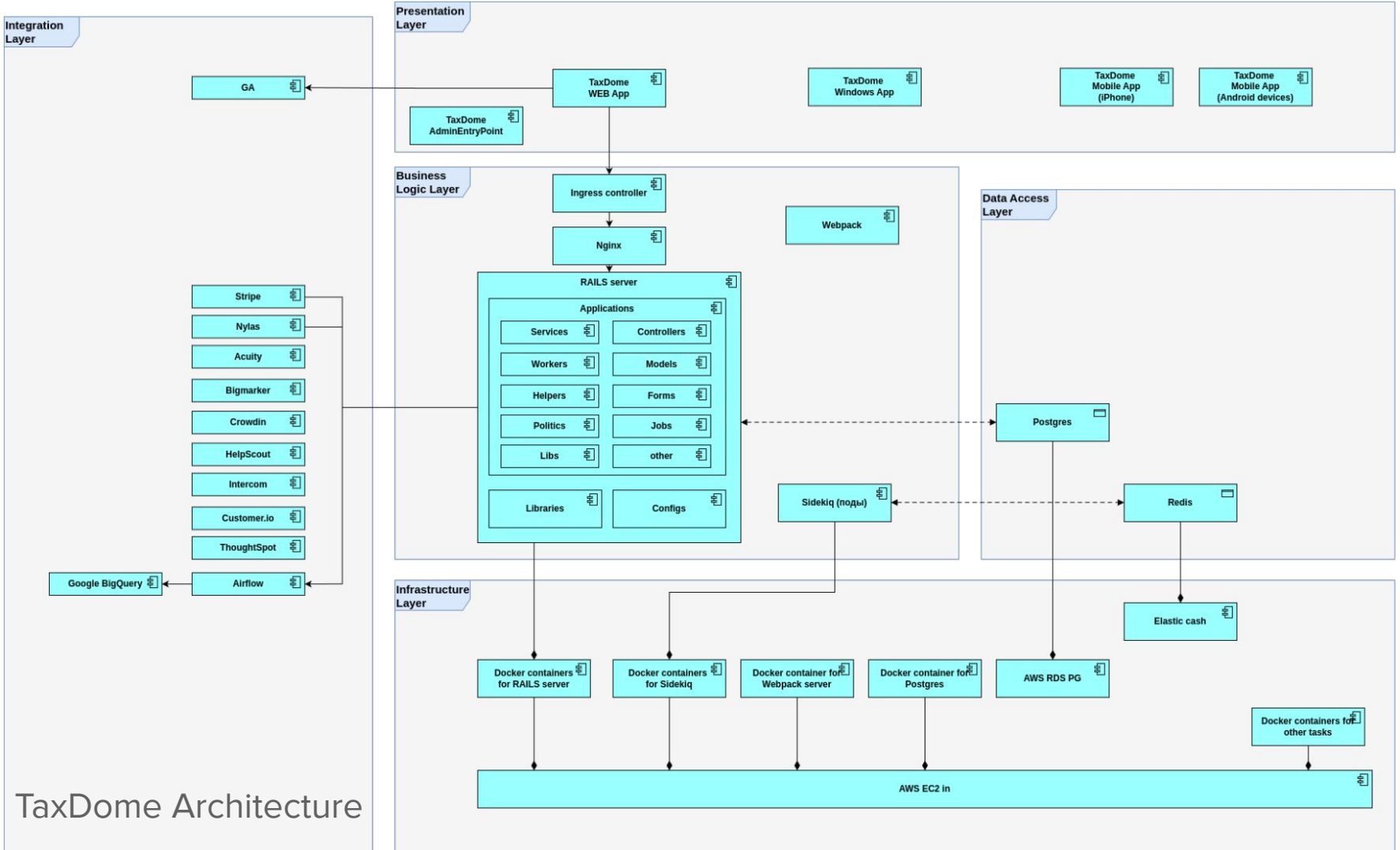
- Software design pattern
- Model - date representation
- View - user interface
- Controller - intermediate logics



Rails Architecture

- Application Server
- Loader
- Rack Middlewares
- Routing
- Controller
- Model
- View
- Mailer

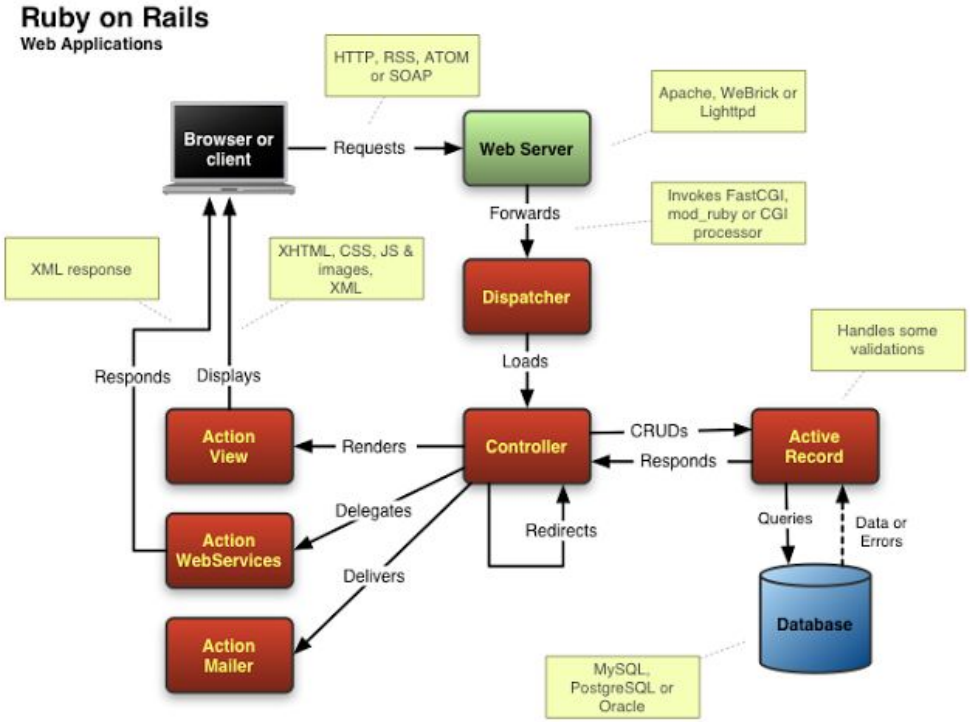




TaxDome Architecture

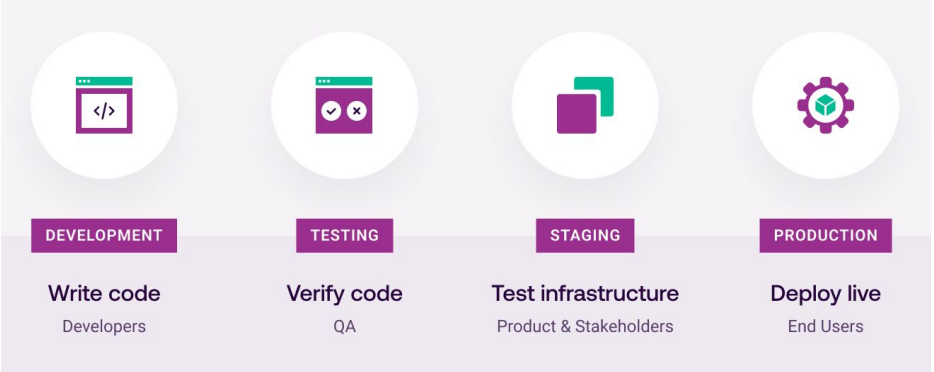
Rails Components

- Migrations (ActiveRecord::Migration)
- Models (ActiveRecord)
- Routes (ActionDispatch)
- Controllers (ActionController)
- Services
- Views
- Serializers (ActiveModel::Serializer)
- Mailers (ActionMailer)
- Workers (Sidekiq::Worker)
- Helpers/Lib



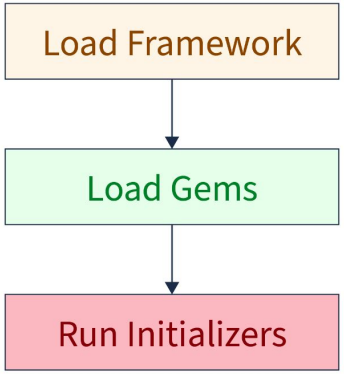
Environments

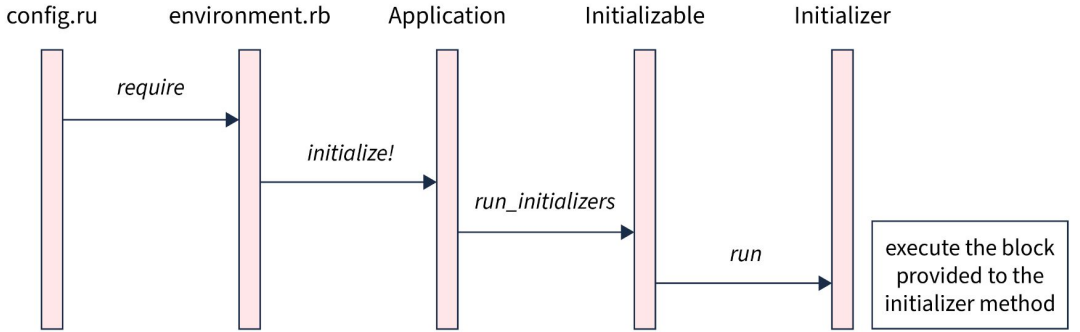
- App runs in different environments
- Each has different ENV variables
- Test
  - Running autotests
  - Has separate database
- Development
  - Local development
  - Debugging and logging
  - Dynamic code reloading
- Staging
  - Running live code on staging
  - Staging integrations
- Production
  - Running live code on staging
  - Production integrations
  - More resources and scalability



Initializers

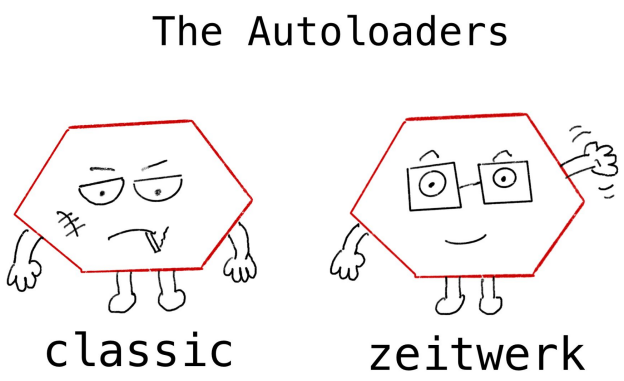
- Scripts that run during the boot of Rails
- Loaded after the framework and gems, but before the application
- Use cases:
  - Set up configuration settings for libraries
  - Customize framework defaults
  - Initialize global settings



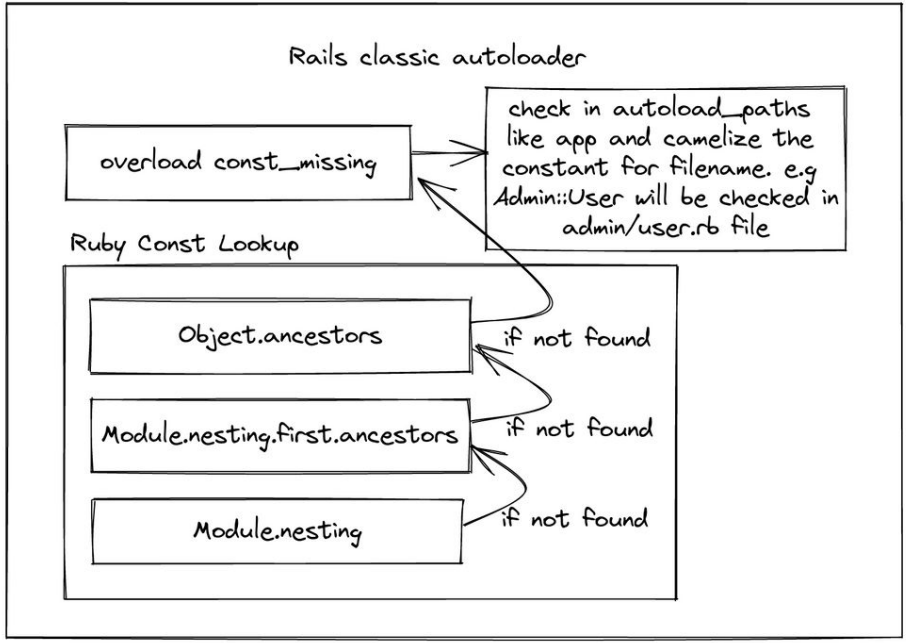


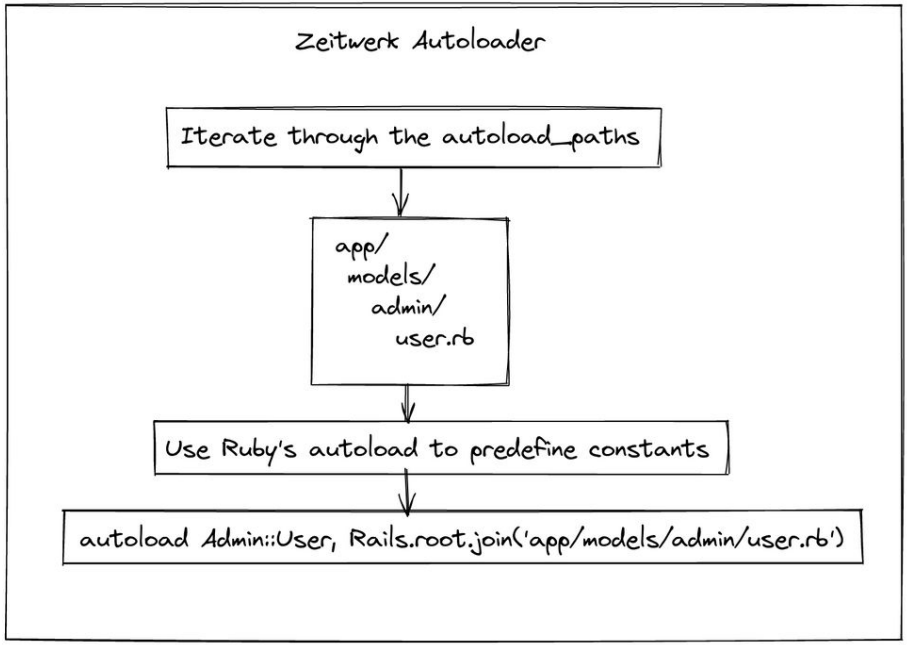
Loader

- Rails 6 introduced Zeitwerk Autoloader
- Has eager and lazy loading modes
- Maps file paths to constant names
- Improved performance
- Faster update on code change
- Enforces code and file structure



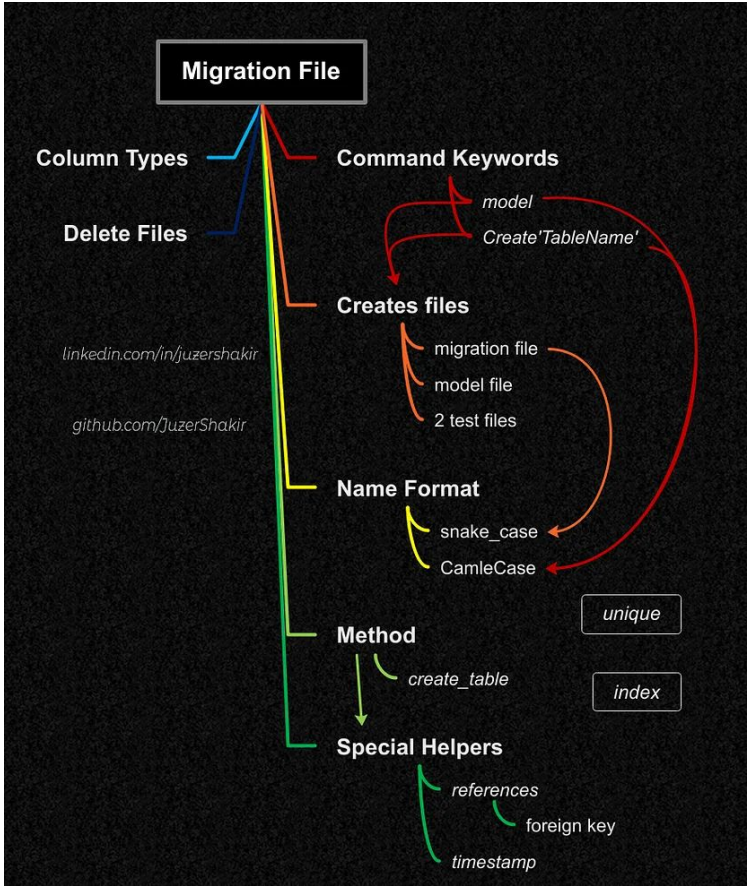
Loader


Rails classic autoloader
overload const_missing
check in autoload_paths like app and camelize the constant for filename. e.g Admin::User will be checked in admin/user.rb file
Ruby Const Lookup
Object.ancestors
if not found
Module.nesting.first.ancestors
if not found
Module.nesting
if not found


Zeitwerk Autoloader
Iterate through the autoload_paths
app/
models/
admin/
user.rb
Use Ruby's autoload to predefine constants
autoload Admin::User, Rails.root.join('app/models/admin/user.rb')

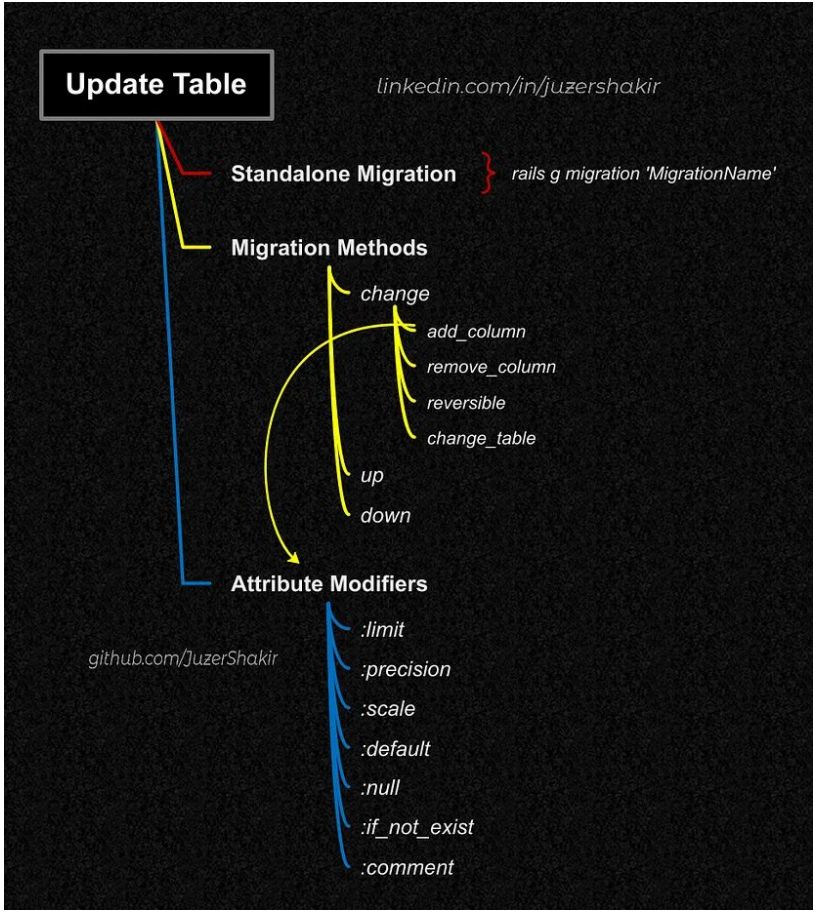
Migrations

- Transformations database structure without SQL
- SQL Methods:
  - create/change/rename/drop table
  - add/remove/rename column
  - add/remove/rename index
- Helper Methods:
  - timestamp
  - references
- Column Types:
  - binary, boolean
  - date, datetime
  - decimal, float, integer, bigint
  - primary_key, string, text, enum
  - time, timestamp



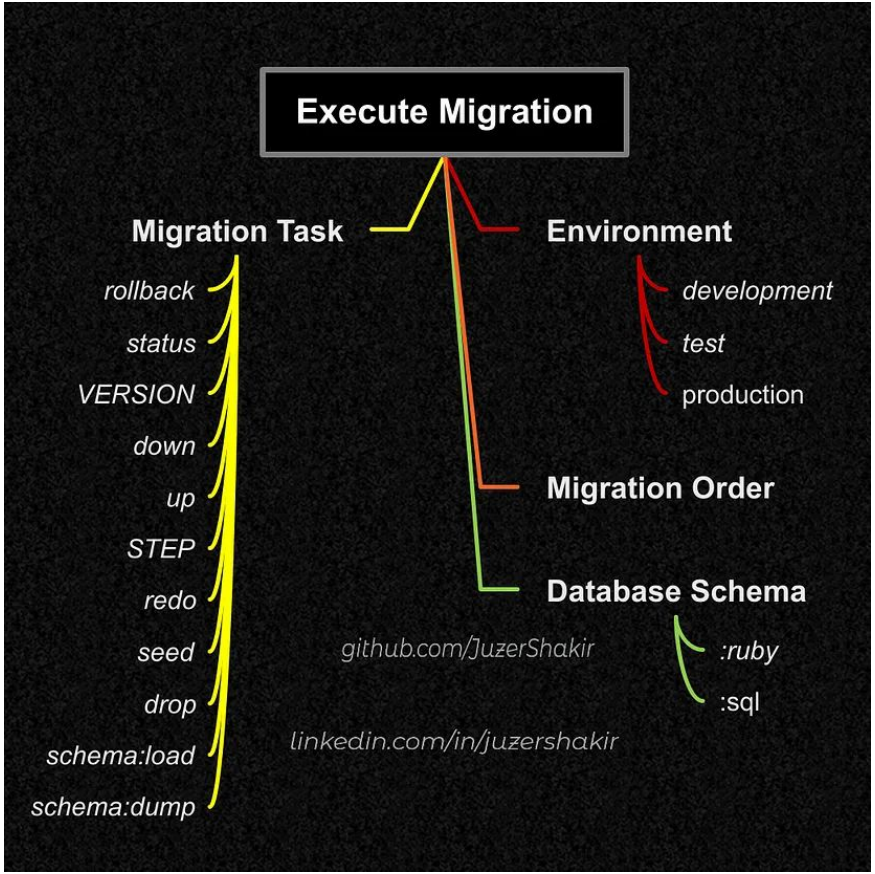
Migrations

- Options:
  - collation
  - limit
  - default
  - null
  - precision, scale
- Revertable:
  - change
  - up
  - down



Migrations

- Schema reflects the current state of the database
- System evolves in incremental steps
- Enables version control
- Commands:
  - db:migrate/rollback[VERSION,STEP]
  - db:migrate:status
  - db:migrate:up/down[VERSION]
  - db:rollback:redo[VERSION,STEP]
  - db:seed
  - db:drop
  - db:schema:load/dump

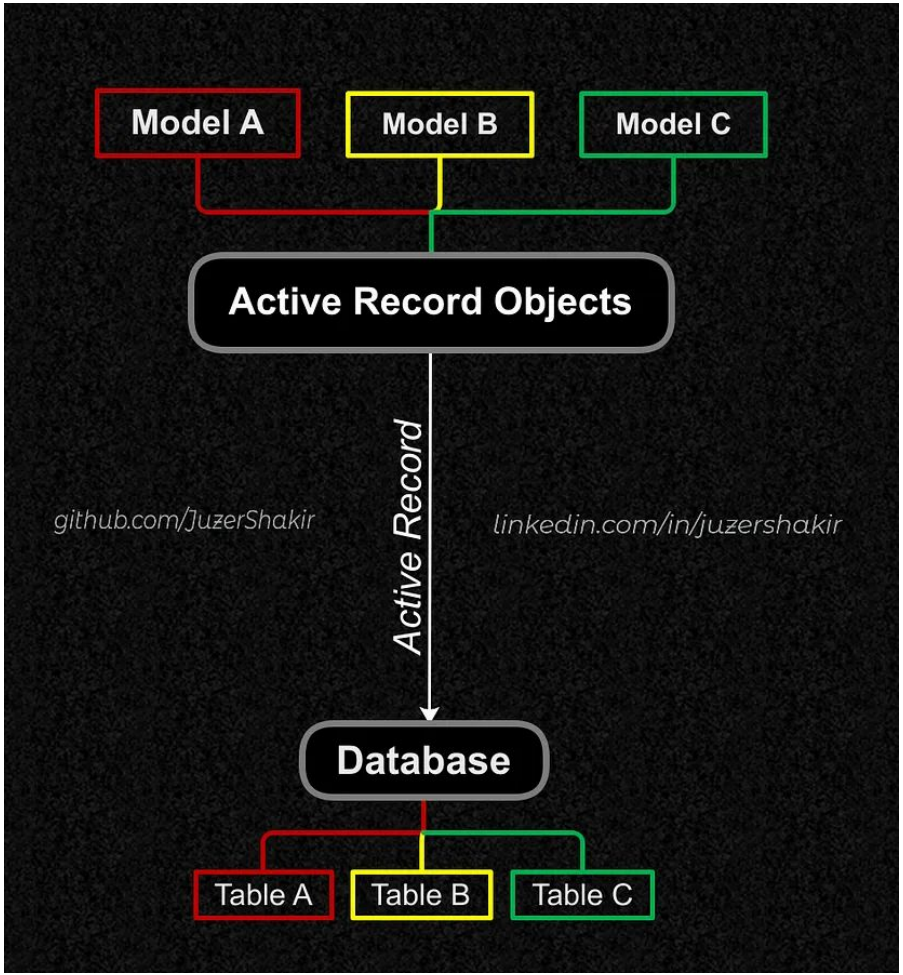


Data Migrations

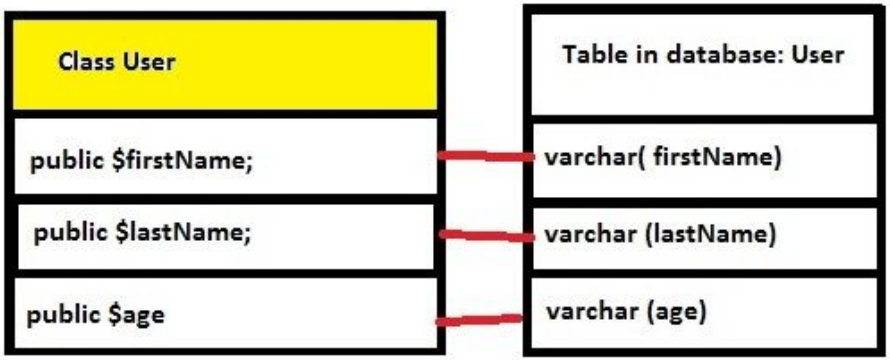
Seeds

ORM

- Object-Relational Mapping (ORM) converts data between OOP and databases
- Two Main Patterns: Active Record and Data Mapper
- ActiveRecord is the default Rails ORM
- Each database table corresponds to a class
- Each record in the table is represented by an instance of the class

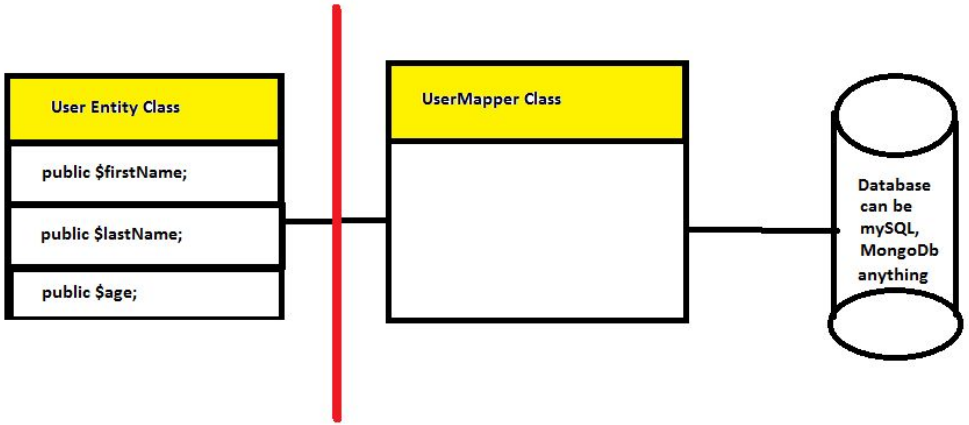


Active Record vs Data Mapper



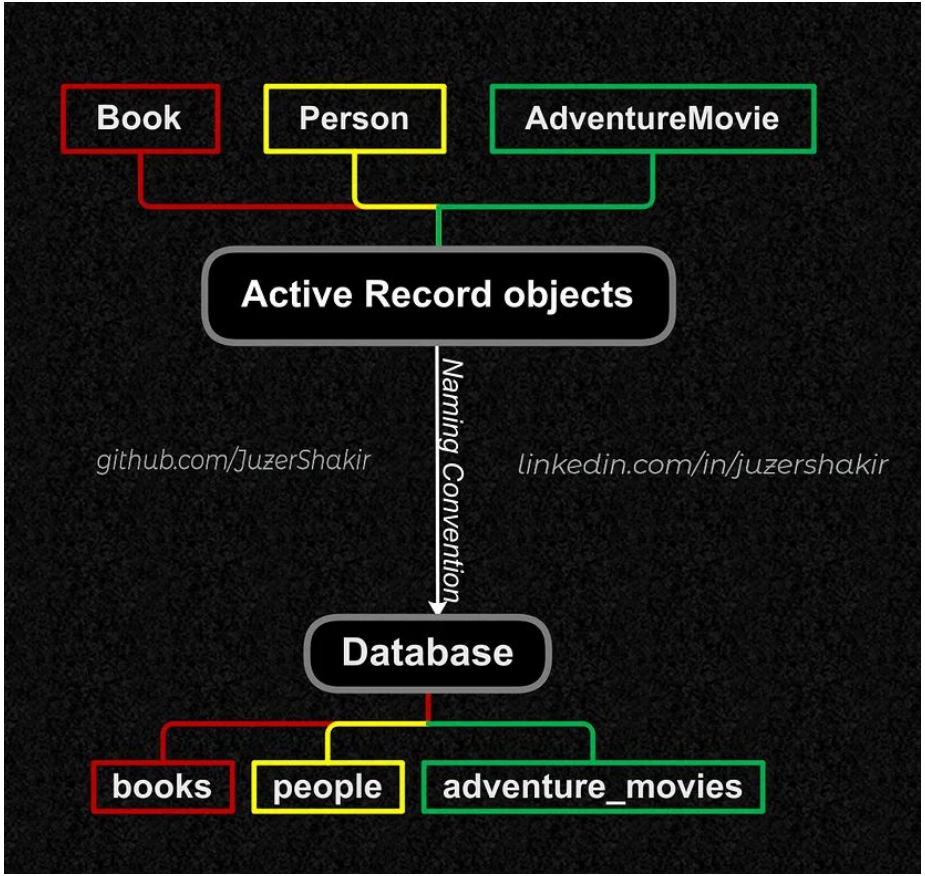
- Each model class corresponds directly to a database table
- Can lead to bloated models with mixed responsibilities
- Tight coupling between model and database schema

- Models are plain objects with no knowledge of the database.
- A separate mapper class handles database interactions.
- Clear separation of concerns
- Increased complexity



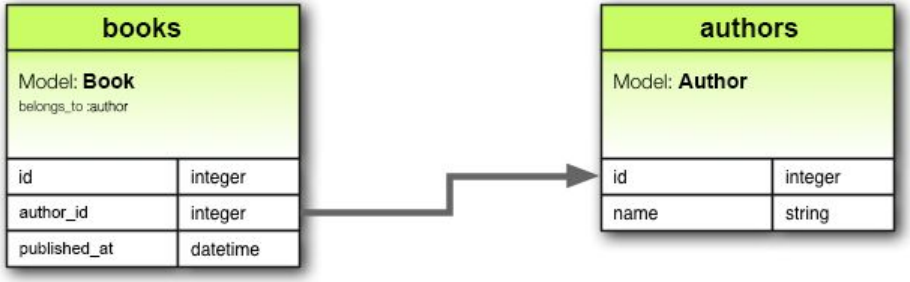
Models

- Provide an interface between tables in the database and Ruby code to modify or create attributes and instances of a table
- Class Methods:
  - new/create
  - update_all
  - where/all/find/find_by
  - destroy_by/destroy_all
- Instance Methods:
  - update
  - destroy

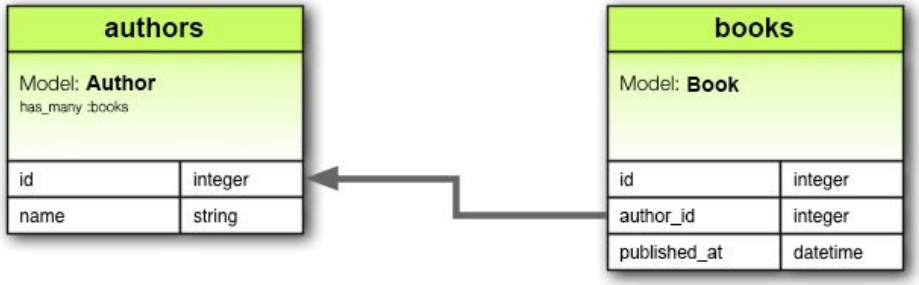


Associations

- Associations establish relationships between models
- Associations:
  - belongs_to
  - has_one/many
  - has_one/many_through
- Has eager loading (includes)
- Allows polymorphic associations
- Methods:
  - create_x
  - build_x
  - x_changed?




```
class Book < ApplicationRecord
  belongs_to :author
end
```




```
class Author < ApplicationRecord
  has_many :books
end
```

Validations

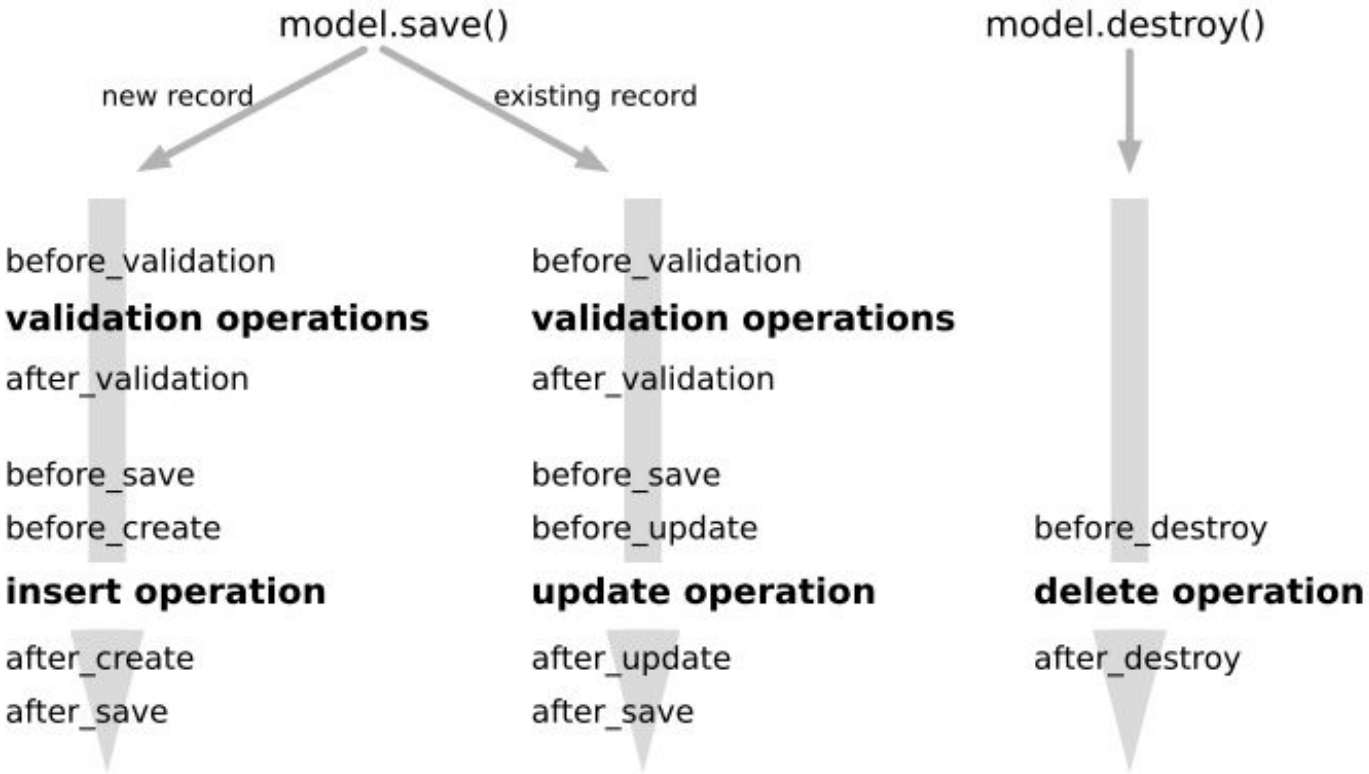
- Ensure that only valid data is saved
- Types:
  - acceptance
  - confirmation
  - comparison
  - format
  - inclusion
  - exclusion
  - length
  - numericality
  - presence
  - absence
  - uniqueness
  - validates_associated
  - validates_each
  - validates_with

```ruby
class Person < ApplicationRecord
  validates :name, presence: true
end
```

```ruby
class Person < ApplicationRecord
  validates :name, length: { minimum: 2 }
  validates :bio, length: { maximum: 500 }
  validates :password, length: { in: 6..20 }
  validates :registration_number, length: { is: 6 }
end
```

```ruby
class Promotion < ApplicationRecord
  validates :end_date, comparison: { greater_than: :start_date }
end
```

Callbacks


model.save()
new record
existing record
model.destroy()
before_validation
validation operations
after_validation
before_save
before_create
insert operation
after_create
after_save
before_validation
validation operations
after_validation
before_save
before_update
update operation
after_update
after_save
before_destroy
delete operation
after_destroy

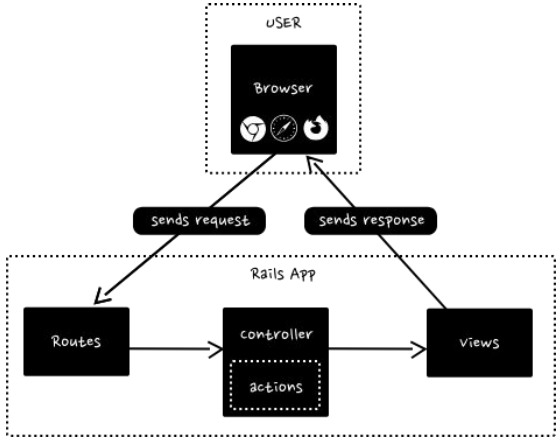
Scopes

Store Model

AASM

Routers

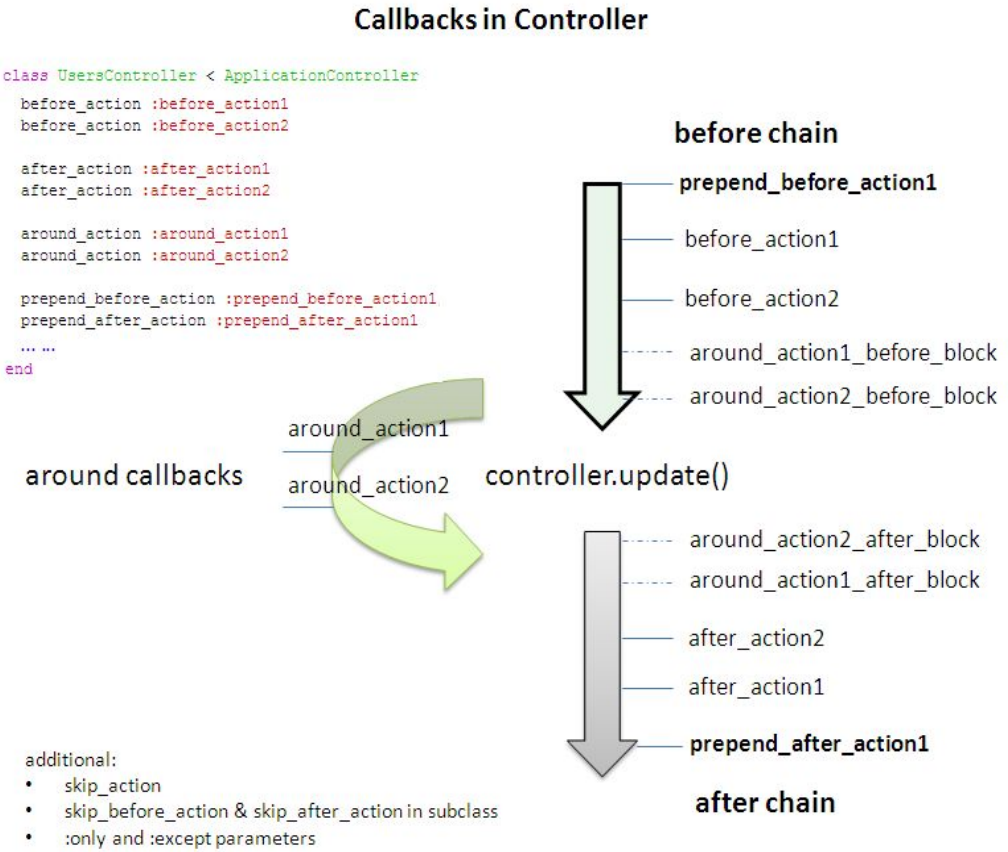
- Handles URL requests and directs them to the appropriate controller actions
- Layers:
  - namespace
  - scope
  - resource
- Custom Actions:
  - member
  - collection



HTTP Verb	Path	Controller#Action	Named Helper
GET	/admin/posts	admin/posts#index	admin_posts_path
GET	/admin/posts/new	admin/posts#new	new_admin_post_path
POST	/admin/posts	admin/posts#create	admin_posts_path
GET	/admin/posts/:id	admin/posts#show	admin_post_path(:id)
GET	/admin/posts/:id/edit	admin/posts#edit	edit_admin_post_path(:id)
PATCH/PUT	/admin/posts/:id	admin/posts#update	admin_post_path(:id)
DELETE	/admin/posts/:id	admin/posts#destroy	admin_post_path(:id)

Controllers

- Handle incoming web requests and return responses
- ActionController::Parameters handles params
- Has callbacks
- Controls response format
- Delegates:
  - params validations -> FormObject
  - complex logics -> ServiceObject
  - complex queries -> QueryObject
  - access restriction -> Policy
  - response format -> Serializer



делегировал,
но... куда?

Forms

- Keeps models clean by moving form-related logic to a separate object
- Located in ***app/forms***
- Handles forms that interact with multiple models or have complex validation rules
- Loose coupling
- Implementation:
  - ApplicationForm
  - dry-rb Form Validation

```ruby
class Api::V1::Firm::Payments::SummaryForm < Api::V1::Firm::Payments::ApplicationForm
  PAYMENT_TYPES = {
    credit_card: %i[stripe stripe_prepayment credit_card credit cpacharge_prepayment cpacharge],
    ach: %i[ach cpacharge_ach cpacharge_ach_prepayment],
    cash: [:cash],
    check: [:check]
  }.freeze

  def initialize(params)
    @q = params[:q] || ActionController::Parameters.new
  end

  def params
    sources = @q[:by_sources] || []
    payment_types = sources.flat_map { |source| payment_types_by_source(source) }
    permit_params.merge(payment_type_in: payment_types)
  end

  private

  def permit_params
    @q.permit(:id_eq, :search_text, :client_id_eq, by_years: [])
  end

  def payment_types_by_source(source)
    Payment.payment_types.slice(*PAYMENT_TYPES[source.to_sym]).keys
  end
end
```

```ruby
schema = Dry::Validation.Params do
  required(:email).filled(:str?)

  required(:age).filled(:int?, gt?: 18)
end

errors = schema.call('email' => '', 'age' => '18').messages
```

Services

- Service objects encapsulate complex business logic
- Located in ***app/services***
- Each service object has one specific job
- Services can be reused
- Loose coupling
- Must contain constructor and call methods

```ruby
module Service
  def self.included(base)
    base.extend self
  end

  def call(*args)
    new(*args).call
  end
end


class CoolService
  include Service

  def initialize(cool_params)
    @cool_params = cool_params
  end

  def call
    puts @cool_params
  end
end

CoolService.call(a: :is_cool, b: :is_ok, you: :are_bad)
```

Queries

- Encapsulate complex query logic into a reusable object
- Located in ***app/queries***
- Thin controller, thin model
- Loose coupling

```ruby
class LoggenInClientsQuery
  class << self
    def with_never_logged_in(query = Client)
      query.where(current_sign_in_at: nil)
    end

    def with_logged_in_since(time, query = Client)
      query.where('clients.current_sign_in_at > ?', time)
    end

    def with_logged_users_in(period, query = Client)
      case period
      when 'week', 'month', 'year'
        with_logged_in_since(DateTimeHelper.beginning_of_period(period), query)
      when 'never'
        with_never_logged_in(query)
      end
    end
  end
end
```

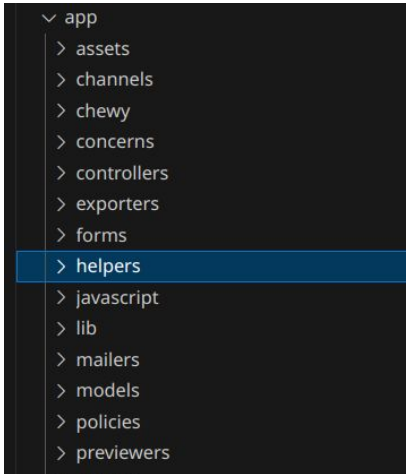
```ruby
class NeverLoggenInClientsQuery
  def initialize(query = Client)
    @query = query
  end

  def call
    @query.where(current_sign_in_at: nil)
  end
end
```

```ruby
class LoggenInSinceClientsQuery
  def initialize(time, query = Client)
    @time = time
    @query = query
  end

  def call
    @query.where('clients.current_sign_in_at > ?', @time)
  end
end
```

```ruby
class LoggenInPeriodClientsQuery
  def initialize(period, query = Client)
    @period = period
    @query = query
  end

  def call
    case period
    when 'week', 'month', 'year'
      time = DateTimeHelper.beginning_of_period(period)
      LoggenInSinceClientsQuery.new(time, @query).call
    when 'never'
      NeverLoggenInClientsQuery.new(@query).call
    end
  end
end
```

Helpers

- Methods that assist in rendering views
- Located in ***app/helpers***
- Can be included in controllers
- Default helper methods:
  - form_with
  - fields_for
  - link_to
  - image_tag




```ruby
module FirmsHelper
  def owner_name(firm)
    firm.primary_name.presence || firm.owner.full_name
  end

  def owner_phone(firm)
    firm.primary_phone? ? firm.primary_phone : firm.owner.user.phone_number
  end

  def owner_email(firm)
    firm.primary_email? ? firm.primary_email : firm.owner.user.email
  end
end
```

Lib

- Custom libraries, modules, and reusable code that do not fit into other patterns
- Located in ***lib***
- Not autoloaded by default
- Examples:
  - Rake tasks
  - Monkey patching
  - Mixins
  - Middlewares

Views

- Templates that represent controller data to the user
- Located in ***app/views***
- Controller renders templates by CoC (can be overridden)
- Views can be shared between controllers
- Locals support
- Rails Helpers integration
- Type:
  - HTML
  - JSON
  - XML
  - Partial

application_cont... | delete.erb | edit.erb | show.erb | index.erb | **new.erb**

```erb
<h1>New Post Form</h1>
<form action="/posts" method="post">
  <label for="name">Name</label>
  <input type="text" name="name">
  <label for="content">Content</label>
  <input type="text" name="content">
  <input type="submit" value="submit">
</form>
```

Mailers

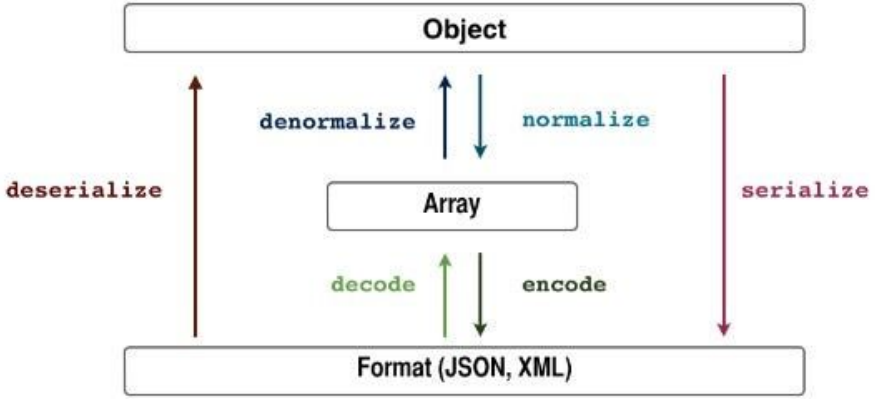
- Action Mailer is a framework for designing email service
- Located in ***app/mailers*** and ***app/views/mailers***
- Handles the creation and delivery of emails
- Consists of mailers and views
- Supports locales

```
mailers > client_mailer.rb > ...
# frozen_string_literal: true

class ClientMailer < ApplicationMailer
  def signup_confirmation(user_id)
    @user = User.find(user_id)
    @firm = @user.firm
    @client_help_url = client_help_url
    url_options = url_options_for(@firm)
    @login_url = new_session_for_firm_url(url_options)
    @account_names = @user.clients.pluck(:account_name).join(', ')

    mail(to: @user.email, bcc: bcc_emails(@firm),
         reply_to: reply_to_firm(@firm), from: from_firm(@firm),
         subject: default_i18n_subject(firm_name: @firm.name))
  end
end
```

```
<% content_for :header do %>
  <%= render partial: '/mailers/shared/header_client', locals: {firm: @firm} %>
<% end %>

<div class="email-bordered-pane" style="border-color: #ffc5c5">
  <%= t('.login_details_html', login_url: link_to(sanitize(@login_url), @login_url), user_email: mail_to(@user.email, sanitize(@user.email))) %>

  <div class="email-button-wrapper email-button-wrapper_r-1-padding">
    <%= render partial: '/mailers/shared/email_button', locals: {href: @login_url, button_label: t('.login_button')} %>
  </div>

  <%= render partial: '/mailers/shared/href', locals: {href: @login_url} %>

</div>

<p><%= t('.greeting') %></p>

<%= t('.body_html', account_names: @account_names, firm_name: @firm.name, help_url: link_to(@client_help_url, @client_help_url, target: '_blank')) %>

<% content_for :footer do %>
  <%= render partial: '/mailers/shared/footer_client', locals: {firm: @firm} %>
<% end %>
```

Serializers

- Format and structure data when rendering API responses
- Located in ***app/serializers***
- Supports JSON, JSON-API, XML
- Implementations:
  - ActiveModel::Serializers (AMS)
  - Fast JSONAPI
  - Blueprinter
  - JSONAPI-RB




```ruby
class Base::CountrySerializer < BaseSerializer
  extend IncludesSerializer

  attributes :name, :iso_code, :default_currency, :default_locale
  includes_attributes :locale_name, :phone_code, :popularity, :region_type, :regions

  def read_attribute_for_serialization(attr)
    object[attr.to_s]
  end
end
```

Workers

- Workers handle background jobs
- Located in ***app/workers***
- Clockwork gem manages recurring events
- Implementations:
  - Sidekiq
  - Rescue
  - Delayed Job

```
every(2.minutes, 'run_reminder_job') do
  ReminderWorker.perform_async
end
```

```
# frozen_string_literal: true

class ReminderWorker < ApplicationWorker
  include Sidekiq::Worker
  sidekiq_options lock: :until_and_while_executing, on_conflict: :log, log_duplicate: true, queue: :default_unique

  def perform
    Reminder.pending.should_start.find_each do |reminder|
      next unless reminder.allowed_to_notify?

      result = SendReminderService.call(reminder)
      ErrorReporter.report(result.errors, {reminder_id: result.data.id}) if result.failed?
    end
  end
end
```

Thank you for your attention!