Rails Application Documentation

Build this documentation with either rake doc:* for standard rails doc output, or with the rake doc:ajax task to get nicer doc layout using breakpointer‘s ajax-rdoc (which is available at <a href="http://github.com/breakpointer/ajax-rdoc»github.com/breakpointer/ajax-rdoc</a> and has to installed separately).

Installation

The first step is to run these rake tasks:

```
rake db:migrate
rake db:seed
```

Now the database is fully migrated and seeded. Start the webserver with

```
rails server
```

A setup screen will be ready at <a href="http://localhost:3000»localhost:3000</a>

Starting points for further reading

Search API

For any information on the search functionality of the app, take a look at the Search module.

Matching API

For any information on the matching of Freight and LoadingSpace objects, take a look at the Matching module.

Class: ActiveRecord::Base

Public Class methods

brackets_find_by_(attribute_name)_

Adds a convenient [] find_by to the model utilizing the given attribute and returning the first record matching the condition. Therefore this is best used on attributes that go through validates_uniqueness_of.

```
class Country < ActiveRecord::Base
  brackets_find_by :iso_code
end

Country[:de] => #<Country id: 1, name: "Germany", iso_code: "de">
```

human_attribute_value_(attribute_name, value, i18n_opts = {})_

searchable_(opts = {})_

Adds a model to the search index.

```
class User < ActiveRecord::Base
  searchable
end
```

Public Instance methods

attributes_filled_()_

belongs_to?_(user = current_user)_

Returns if the record belongs to a certain user.

human_attribute_value_(attribute_name, i18n_opts = {})_

mine?_(user = current_user)_

Alias for belongs_to?

Class: Admin::BaseController

The BaseController is the controller every other backend controller inherits from. He is mainly responsible for providing basic functionality to the backend, e.g. rights management.

Module: Admin::BaseHelper

The BaseHelper provides basic helper methods all backend views.

Public Instance methods

link_new*(text = t(ädmin.common.new_link"), url = new_resource_url)*

Renders a link to the new action of the current resource.

render_table*(arel)*

Renders a table for a given ActiveRelation.

Example:

```
<%= render_table User.all %>
```

Class: AppConfig

AppConfig objects store system specific configuration values for the application.

Reading

Read any value from the database (or the default config yaml, if it is not yet in the db) by using the [] accessor.

```
AppConfig[:language]  # => "en"
```

Storing

Store any value in the database by using the []= accessor.

```
AppConfig[:language] = 'de'  # => "de"
AppConfig[:some_option] = 'some value' # => "some value"
```

Public Class methods

AppConfig[key] # => Object

Returns the value stored for key in the database or its default value.

```
AppConfig[:language]  # => "en"
```

AppConfig[key] = value

Stores the value for key in the database.

```
AppConfig[:language] = 'de'  # => "de"
```

Class: ApplicationController

Private Class methods

login_required*(opts = {})*

Use this in a controller to restrict access.

```
class UsersController < ApplicationController
  login_required :only => [:edit, :update, :show]
end
```

ownership_required*(opts = {})*

Use this in a controller to restrict access to owners.

role_or_ownership_required*(roles, opts = {})*

Use this in a controller to restrict access to either users of certain roles (e.g. admins) or the rightful owner of an object.

```
class PostingController < ApplicationController
  role_or_ownership_required [:posting_admin, :administrator]
end
```

role_required*(roles, opts = {})*

Use this in a controller to restrict access to either users of certain roles (e.g. admins).

```
class Admin::BaseController < ApplicationController
  role_required :administrator
end
```

same_company_required*(opts = {})*

Private Instance methods

controller_catalog*(*

Returns the i18n catalog path for the current controller.

```
class UsersController < ApplicationController
  def index
    controller_catalog # => 'users'
  end
end

class Admin::UsersController < Admin::BaseController
  def index
    controller_catalog # => 'admin.users'
  end
end
```

current_company*()*

Returns the Company object of the currently logged in user or `nil` if no user is logged in.

current_person*()*

Returns the Person object of the currently logged in user or `nil` if no user is logged in.

current_user*()*

Returns the User object of the currently logged in user or `nil` if no user is logged in.

demo_mode?*()*

Returns `true` if the application is running in demo mode.

Module: ApplicationHelper

The ApplicationHelper provides basic helper methods for all views.

Public Instance methods

admin?*()*

box*(title = nil, &block)*

clear_both*()*

Returns a DIV tag that clears floating.

collection_choices*(model, attribute_name, const = nil)*

Returns the collection of localized choices for a given attribute. Example:

```
collection_choices(Person, :gender)
```

This will look up `Person::GENDER_CHOICES` and return the keys and localized values.

controller?(name) # => boolean

Returns if c is the current controller. Example:

```
<%= controller?(:root) %>
# => true
```

format_multiline_input*(text)*

Returns a HTML formatted version of `text`. Example:

```
<%= format_multiline_input("First line.\nSecond Line.") %>
# => "First line.\Second line."
```

link_back*(text = t("common.link_back"))*

Returns a link back to the last visited page with a localized caption.

link_to_unless*(condition, name, options = {}, html_options = {}, &block)*

TODO: lookup rails3 implementation

localized_info*(obj, name, lang = I18n.default_locale)*

Returns a formatted string for the associated LocalizedInfo object.

localized_info_field*(f, name, lang)*

```
TODO: localized_info_field f, :type_of_goods, :en
```

BETTA: f.localized_info_field :type_of_goods, :en

only_some_attributes_filled?*(ar)*

render_company_info*(company)*

Renders a partial with the contact information for the given company. Example:

```
<%= render_person_info current_company %>
```

render_partial*(partial, options = {})*

render_person_info*(person)*

Renders a partial with the contact information for the given person. Example:

```
<%= render_person_info current_person %>
```

yes_no*(condition)*

Class: CompaniesController

Public Instance methods

create*()*

dashboard*()*

new*()*

The Companies#new action is actually the "Create a new Accountscreen a user sees when he signs up for the freight exchange.

show*()*

Module: CompaniesHelper

Public Instance methods

registering_new_account?*()*

Class: Company

Companies are organising Users.

Public Instance methods

ensure_admin()

Ensures there is atleast one :company_admin left. If no admin can be found, the first user of the company is assigned the admin role.

Class: Freight

Public Instance methods

localized_info*(name, lang = I18n.default_locale)*

localized_infos=*(array_of_options)*

to_search*()*

update_localized_infos*()*

Class: FreightsController

Public Instance methods

create*()*

new*()*

update*()*

Class: LoadingSpace

Public Instance methods

localized_info*(name, lang = I18n.default_locale)*

localized_infos=*(array_of_options)*

to_search*()*

update_localized_infos*()*

Class: LoadingSpacesController

Public Instance methods

create*()*

new*()*

update*()*

Class: LocalizedInfo

Public Instance methods

update_or_destroy!*()*

Class: Matching::Compare::Base

Compare objects compare two objects A and B based on their type/class.

Creation

Compare objects accept two constructor parameters for the A and the B object.

```
compare = Compare::String.new('one string', 'another string')
compare.result # => 0.6428...
```

Conditions

By default, a compare object compares copies of the entire objects it is passed. It is also possible to only compare certain attributes of an object.

```
class UserComparer < Matching::Compare::Base
  compare :gender, :weight
end
```

Thresholds can be used to ensure that only objects who meet certain criteria are considered alike.

```
class UserComparer < Matching::Compare::Base
  compare :weight, :threshold => 10
  # => User A can be 10 kilos heavier or lighter than user B

  compare :weight, :threshold => 0.05
  # => User A can be 5% heavier or lighter than user B

  compare :weight, :threshold => {:up => 0, :down => 0.1}
  # => User A can be 10% lighter than user B, but not any heavier.

  compare :weight, :threshold => :perfect
  # => User A and B have to have the same weight
end
```

All object-pairs not meeting the threshold criteria are automatically assigned a result of 0.0 (not matching at all).

Overwriting defaults

Blocks can be used to override the default comparisions.

Example:

```
class UserCompanyComparer < Matching::Compare::Base
  # Do not compare the email with the default String processor
  # but compare the email hosts and eliminate the pair if they
  # are not matching.
  compare :email do |a, b|
    email_domain = /[^@]+$/
    a[email_domain] == b[email_domain]
  end
end
```

Public Class methods

compare(*attributes, options = {}, &block)

Specifies one or more attribute(s) that will be compared using the defined options and the block, if given.

Options

- `:as` - A Symbol identifying the Comparer class to be used

(e.g. `:String`, `:Time` etc.)

```
class UserComparer < Matching::Compare::Base
  compare :created_at, :as => :Time
end
```

- `:threshold` - If the attribute of the B object differs more

than the given threshold the comparison fails, resulting in a 0.0 match. `:up` and `:down` options are available as well. Floats are interpreted as relative, Fixnums as absolute thresholds.

```
class UserComparer < Matching::Compare::Base
  compare :weight, :threshold => 10
  # =>  User A can be 10 kilos heavier or lighter than user B
```

```
    compare :weight, :threshold => 0.05
    # =>  User A can be 5% heavier or lighter than user B

    compare :weight, :threshold => {:up => 0, :down => 0.1}
    # =>  User A can be 10% lighter than user B, but not any heavier

    compare :weight, :threshold => :perfect
    # =>  User A and B have to have the same weight
  end
```

Block evaluation If a block is given, the compared attributes are passed and the result of the block is the final result for the comparison (with `true` being interpreted as 1.0).

```
class UserComparer < Matching::Compare::Base
  compare :email do |a, b|
    email_domain = /[^@]+$/
    a[email_domain] == b[email_domain]
  end
end
```

new(a, b)

Create a new Compare object to compare the given objects.

Public Instance methods

result*()*

Compares two objects and returns a result between 0.0 (not alike) and 1.0 (perfect match).

Examples:

```
Comparer::Base.new(true, false) # => 0.0
Comparer::Base.new(true, true) # => 1.0
```

Protected Instance methods

calc_result*(hsh)*

compare_attribute*(attr, opts = {})*

compare_attributes_and_calc_result*()*

compared_attributes*()*

comparer_for*(klass)*

in_threshold*(x, y, result, threshold = {})*

Floats are interpreted as relative, Fixnums as absolute thresholds.

Class: Matching::Compare::Fixnum

Compares two fixnum objects.

Public Instance methods

result*()*

Class: Matching::Compare::FreightToLoadingSpace

FreightToLoadingSpace objects compare Freight with LoadingSpace objects and return a result how good they match.

Class: Matching::Compare::Hash

Compares to two hashes by comparing all values of hash A with their counterparts in hash B.

Public Instance methods

result*()*

Class: Matching::Compare::String

Compares two strings using Levenshtein distance.

Public Instance methods

result*()*

Class: Matching::Compare::Time

Compares two time objects.

Public Instance methods

result_()_

Module: Matching::Compare

The Compare module provides a set of classes and methods to match objects like Strings, Numbers and Dates.

Module: Matching

The Matching module provides a set of classes and methods to match objects. On top of this, it provides an extendable generic API for matching Freight and LoadingSpace objects (see compare_freight_and_loading_space method).

Public Class methods

Match.compare_freight_and_loading_space(freight, loading_space) # => Float Match.fls freight, loading_space # => Float

Returns the likeness of a Freight and a LoadingSpace object.

```
Match.fls Freight.first, LoadingSpace.first # => 0.977920227850516
```

Class: Object

Public Instance methods

obj.full? obj.full? { |f| ... }

Returns wheter or not the given obj is not blank?. If a block is given and the obj is full?, the obj is yielded to that block.

```
salary = nil
salary.full? # => nil
salary.full? { |s| "#{s} $" } # => nil
salary = 100
salary.full? { |s| "#{s} $" } # => "100 $"
```

With Rails' implementation of Symbol#to_proc it is possible to write:

```
current_user.full?(&:name) # => "Dave"
```

Class: Person

Person objects contain personal information about a User.

Public Instance methods

name*()*

TODO: Anrede?

Class: Posting

Public Instance methods

to_search*()*

validate*()*

Class: RootController

The RootController is the starting point of the application. On first start, it creates an admin user and guides him to the setup process. On login, it redirects users to their designated location. Additionally, the RootController also holds information about the app (e.g. the about action).

Public Instance methods

index*()*

The index action decides where the current_user is redirected based on whether or not the app is already set up.

welcome*()*

The welcome action decides what to do with a freshly logged in user.

Module: Search

search.rb

The Search module acts as a wrapper to whatever search engine is running in the background.

Public Class methods

Search.clear_index_for(record)

Removes the search index for the given record.

```
user = User.create(:name # => 'Bob')  # => #<User id: 1, name: "Bob">
Search.find 'bob'           # => [#<User id: 1, name: "Bob">]

Search.clear_index_for(user)
Search.find 'bob'           # => []
```

Search.count(query) # => int Search.count(query, models) # => int

Returns the total number of results.

```
Search.count "some query"
Search.count "Berlin", [User, Company])
```

Search.find(query) # => array Search.find(query, models) # => array Search / query # => array

Returns the matching records from the database.

```
Search.find "some query"
Search.find "Berlin", [User, Company])
Search / "some other query"
```

Search.update_index_for(model_or_record) Search

Adds a record or a model to the search index.

```
Search << User.first  # update the index for a specific user
Search << User        # update the index of all users
```

Class: SearchController

Public Instance methods

index_()_

Class: SiteInfo

SiteInfo objects contain information about loading and unloading sites, such as name of the site, address of the site, name of the contractor etc.

Class: Station

Public Instance methods

to_search*()*

Class: User

User objects respresent a user of the system and are used to authenticate users upon login (using acts_as_authentic plugin) and handle permission handling via assigned UserRole objects.

Data concerning the actual, human user (like company, gender, language etc.) is stored in associated Person and Company objects.

Public Instance methods

user.has_role?(role_name) # => boolean

Returns true if a user has a UserRole with the given `name`.

```
user.has_role?(:administrator) # => true
```

is?*(name)*

Alias for has_role?

user.roles # => array

Returns an array of role names.

```
user.roles # => ["administrator", "company_admin"]
```

Class: UserRole

UserRoles grant a logged in User access to certain parts of the application.

Creation

UserRoles are created and identified via their `:name` attribute.

```
UserRole.create(:name => 'employee_of_the_month')
```

Find by name

UserRoles can be found via their `:name` attribute using the `[]` accessor.

```
UserRole[:employee_of_the_month]
```

Assigning

Finally, UserRoles can be eassigned to a User with the « operator.

```
user.user_roles << UserRole[:employee_of_the_month]
```

To access the backend e.g. a user must have administrator priviligues:

```
user.user_roles << UserRole[:administrator]
```

This is also used in the frontend to restrict the priviligues of users in companies.

```
user.user_roles << UserRole[:company_admin]
```

Class: UsersController

Public Instance methods

create*()*

This creates a new user inside the current company. For the original sign up screen, see Companies#new.

index*()*

Lists all users in the current company.

Class: UserSession

Public Class methods

UserSession.login(user) # => boolean

Authenticates a user and logs him in.

```
  UserSession.login(User.first) # => true
```

Class: UserSessionsController

The UserSessionsController handles all requests regarding logging in and out.

Public Instance methods

create_()_

Authenticates a User by creating and saving a UserSession.

demo_login_()_

This action is only available if the application is running in demo mode. It creates a UserSession for a given user without any authentication.

destroy_()_

Logs a user out.

new_()_