



APIs

Juan “Harek” Urrios  
@xharekx33

# What's an API?

“In computer programming, an application programming interface (API) specifies how some software components should interact with each other”

Basically an API is a computer-computer interface that allows programs to interact with each other.



# Web Services

A Web service is a method of communication between two electronic devices over a network.

One side is called the server and provides the API and the other side is the client and can obtain data from the server through the API and manipulate it.

They communicate through a common protocol: through HTTP requests and responses.



# HTTP request

HTTP requests consist of:

- URL (Uniform Resource Locator): a unique address for a resource
- Method: the kind of action the client wants the server to take
- Headers: meta-information about the request
- Body: the data the client wants to send the server.



# HTTP response

Responses consist of:

- Status Code: tell whether the request was successful or not
- Headers: meta-information about the response
- Body: the data the server sends back to the client



# Web services and HTTP

---

Web services provide a way so clients can access and modify entities on a remote server using HTTP requests and responses.



# REST

REST (Representational State Transfer ) is a software architecture style for building scalable web services:

- Protocol agnostic
- Uses standard HTTP
- Simple
- Scalable
- Performant
- Stateless

APIs that use REST principles are called RESTful APIs



# RESTful APIs

RESTful APIs are defined with these aspects:

- Base URL: The base address for our requests
- Internet media type: For the data
- Resources: Data you expose in your API
- Supported operations: Create, Read, Replace, Update, Delete.





# Creating a basic API

Create a new project to create a basic API: `rails new taskly`

In our app there will be Users and Tasks.

A terminal window with a dark background and a light gray title bar containing three colored window control buttons (red, yellow, green). The terminal displays the command `$ rails new taskly` in a light gray monospaced font.

```
$ rails new taskly
```



@xharekx33

# Adding users

Users will have name and email. Use `rails g resource User name:string email:string` as a shortcut to add the User model, controller, migration file and basic routes. Run `rake db:migrate` afterwards

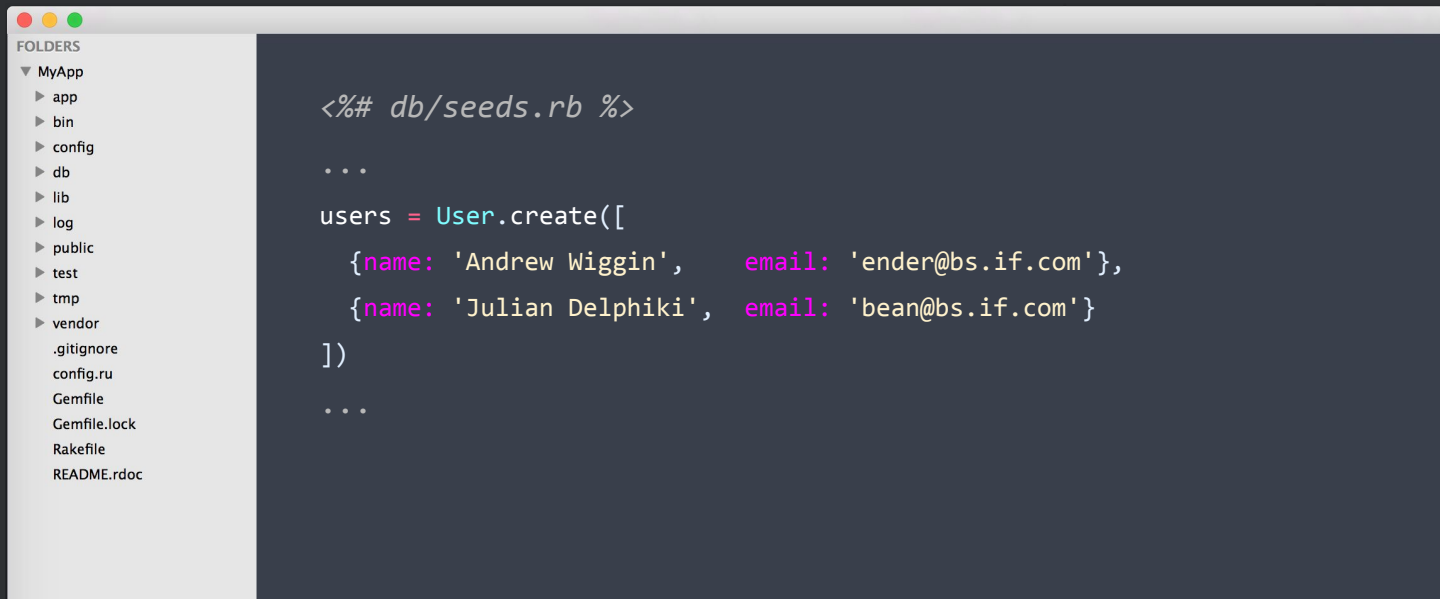
```
$ rails g resource User name:string email:string
  invoke  active_record
  create  db/migrate/20150809190834_create_users.rb
  create  app/models/user.rb
  invoke  test_unit
  create  test/models/user_test.rb
  create  test/fixtures/users.yml
  invoke  controller
  create  app/controllers/users_controller.rb
  invoke  erb
  create  app/views/users
  invoke  test_unit
```



@xharekx33

# Seed database

Let's add some users to the seed file and run `rake db:seed`



The screenshot shows a code editor with a sidebar on the left displaying a file tree under the heading 'FOLDERS'. The tree includes a 'MyApp' folder with subfolders 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor', as well as files '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area shows a Ruby seed file with the following content:

```
<%=# db/seeds.rb %>

...

users = User.create([
  {name: 'Andrew Wiggin',   email: 'ender@bs.if.com'},
  {name: 'Julian Delphiki', email: 'bean@bs.if.com'}
])

...
```

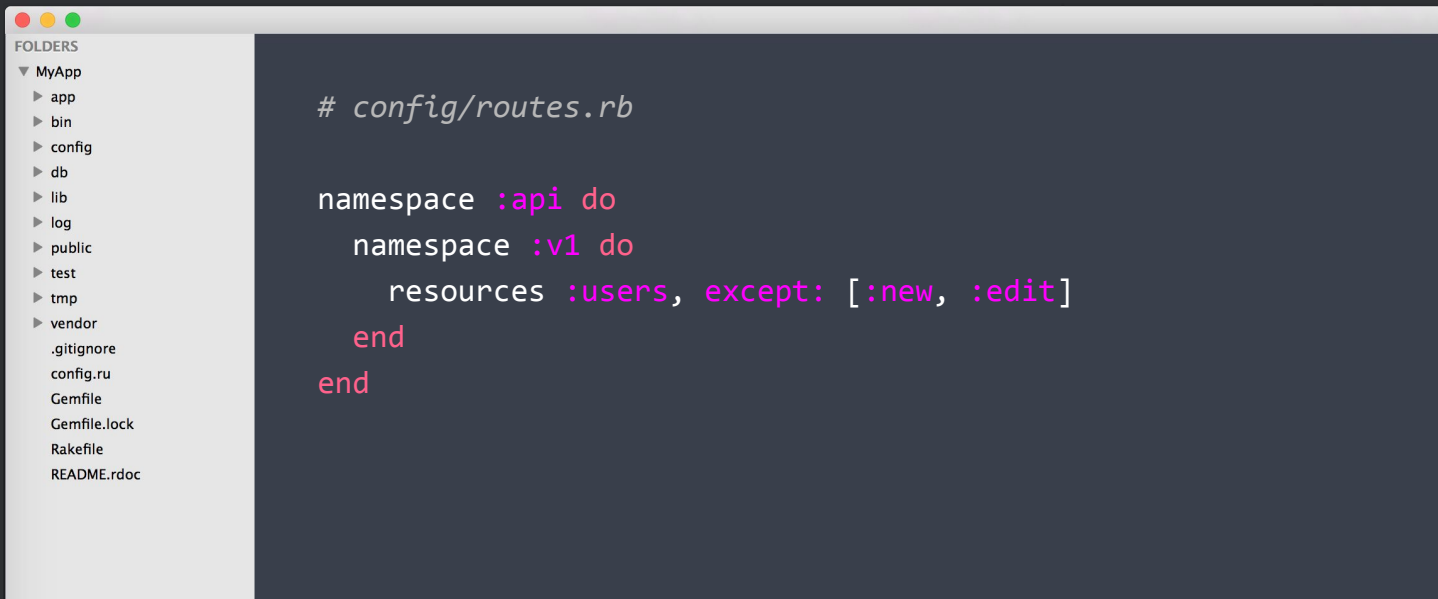


@xharekx33

# Better routes

---

We'll remove unnecessary routes and use namespaces to keep things neat and tidy. Use **rake routes** to check them out.



```
# config/routes.rb

namespace :api do
  namespace :v1 do
    resources :users, except: [:new, :edit]
  end
end
```

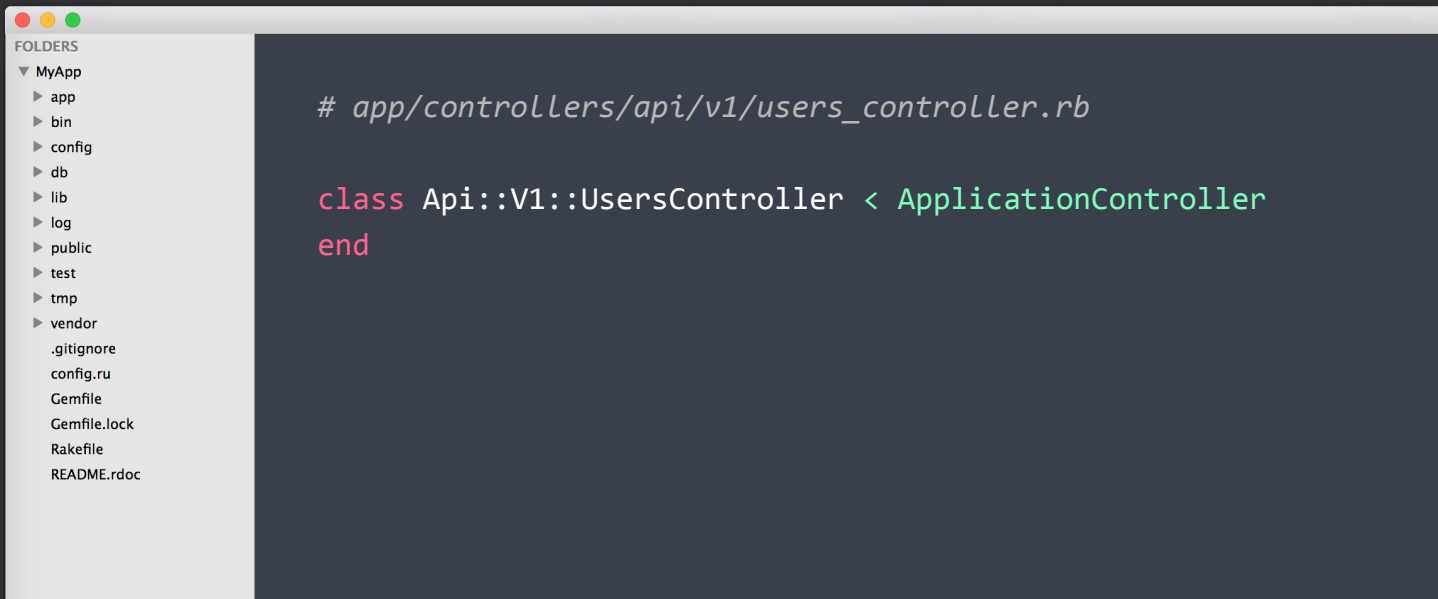


@xharekx33

# Move the Users controller

---

Since we're using namespaces we need to move the controller to the `app/controllers/api/v1` folder and make some changes:



The screenshot shows a code editor with a sidebar on the left displaying the project's file structure under the heading "FOLDERS". The structure includes a "MyApp" directory with subfolders like "app", "bin", "config", "db", "lib", "log", "public", "test", "tmp", and "vendor", as well as files like ".gitignore", "config.ru", "Gemfile", "Gemfile.lock", "Rakefile", and "README.rdoc". The main editor area shows the content of a file named `app/controllers/api/v1/users_controller.rb`. The code defines a new class `Api::V1::UsersController` that inherits from `ApplicationController`.

```
# app/controllers/api/v1/users_controller.rb

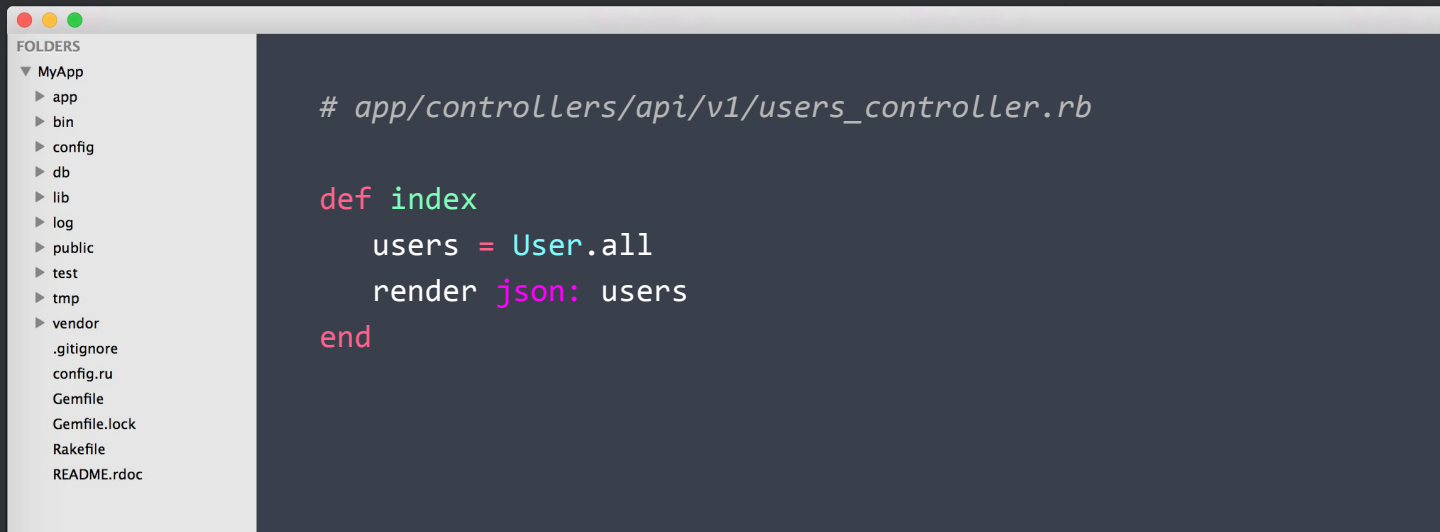
class Api::V1::UsersController < ApplicationController
end
```



@xharekx33

# Listing users

Add a controller action to list all Users. We don't need a corresponding view. Go to <http://localhost:3000/api/v1/users> to check it out.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like app, bin, config, db, lib, log, public, test, tmp, vendor, and files like .gitignore, config.ru, Gemfile, Gemfile.lock, Rakefile, and README.rdoc. The code editor shows the following code:

```
# app/controllers/api/v1/users_controller.rb

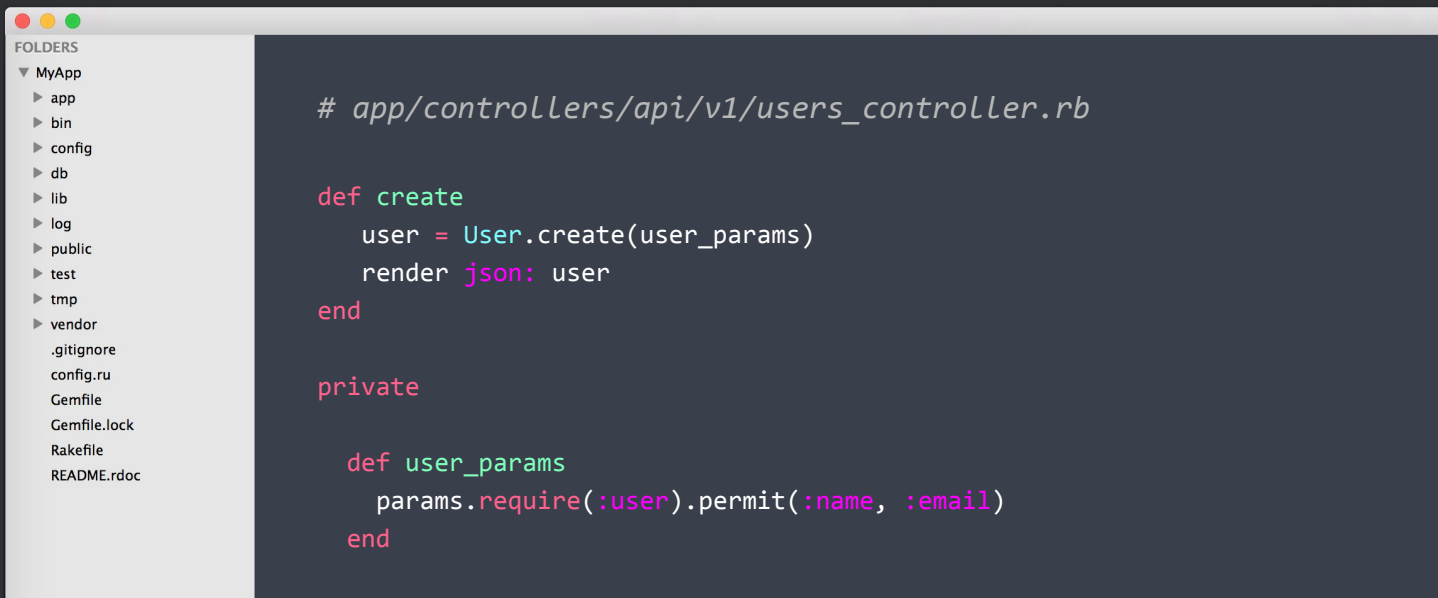
def index
  users = User.all
  render json: users
end
```



@xharekx33

# Creating users

Add a controller action to create new users and the necessary strong parameters.



The screenshot shows a code editor with a sidebar on the left displaying a file tree under the name 'FOLDERS'. The tree includes a 'MyApp' directory with subfolders 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor', as well as files like '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area displays Ruby code for a controller action named 'create' in the file '# app/controllers/api/v1/users\_controller.rb'. The code defines a 'create' method that takes 'user\_params' and uses 'User.create' to create a new user, followed by 'render json: user'. A 'private' section contains a 'user\_params' method that uses 'params.require(:user).permit(:name, :email)' to define the permitted parameters for the create action.

```
# app/controllers/api/v1/users_controller.rb

def create
  user = User.create(user_params)
  render json: user
end

private

def user_params
  params.require(:user).permit(:name, :email)
end
```

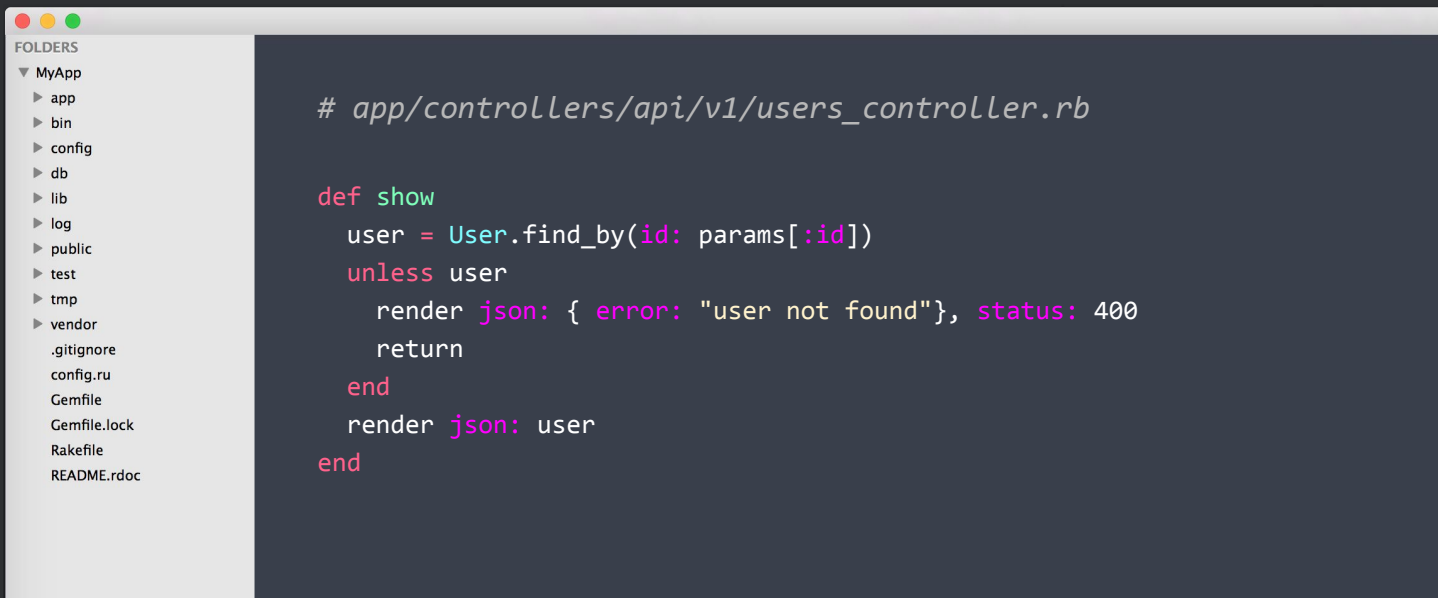


@xharekx33

# Showing a user

---

Add a controller action to return all data for a user.



The image shows a code editor window with a sidebar on the left and a main code area on the right. The sidebar, titled 'FOLDERS', lists the project structure under 'MyApp':

- ▼ MyApp
  - ▶ app
  - ▶ bin
  - ▶ config
  - ▶ db
  - ▶ lib
  - ▶ log
  - ▶ public
  - ▶ test
  - ▶ tmp
  - ▶ vendor
  - .gitignore
  - config.ru
  - Gemfile
  - Gemfile.lock
  - Rakefile
  - README.rdoc

The main code area displays the content of `app/controllers/api/v1/users_controller.rb`:

```
# app/controllers/api/v1/users_controller.rb

def show
  user = User.find_by(id: params[:id])
  unless user
    render json: { error: "user not found"}, status: 400
    return
  end
  render json: user
end
```

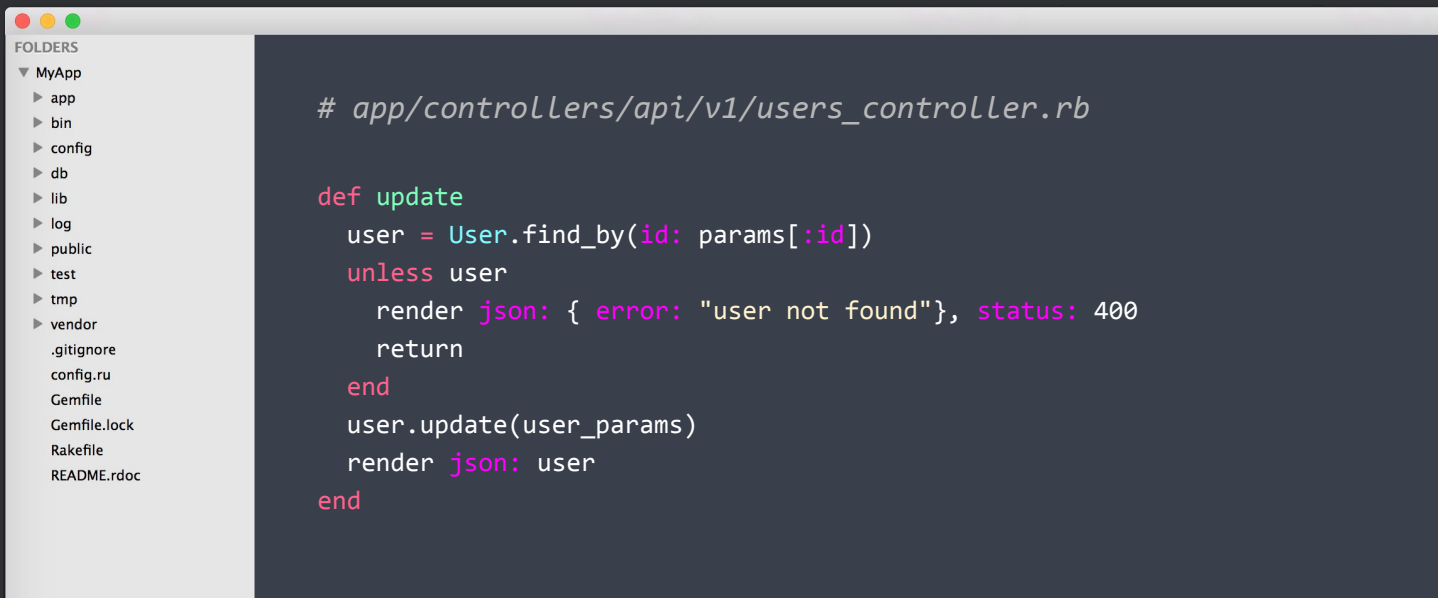


@xharekx33



# Updating a user

Add a controller action to return all data for a user.



The image shows a code editor window with a sidebar on the left displaying a file tree. The main area contains a Ruby code snippet for a controller action. The sidebar lists folders and files under 'MyApp', including 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', 'vendor', '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The code in the main area is for a file named `app/controllers/api/v1/users_controller.rb`. It defines an `update` action that finds a user by ID, checks if it exists, and either renders an error or updates the user and renders the response.

```
# app/controllers/api/v1/users_controller.rb

def update
  user = User.find_by(id: params[:id])
  unless user
    render json: { error: "user not found"}, status: 400
    return
  end
  user.update(user_params)
  render json: user
end
```

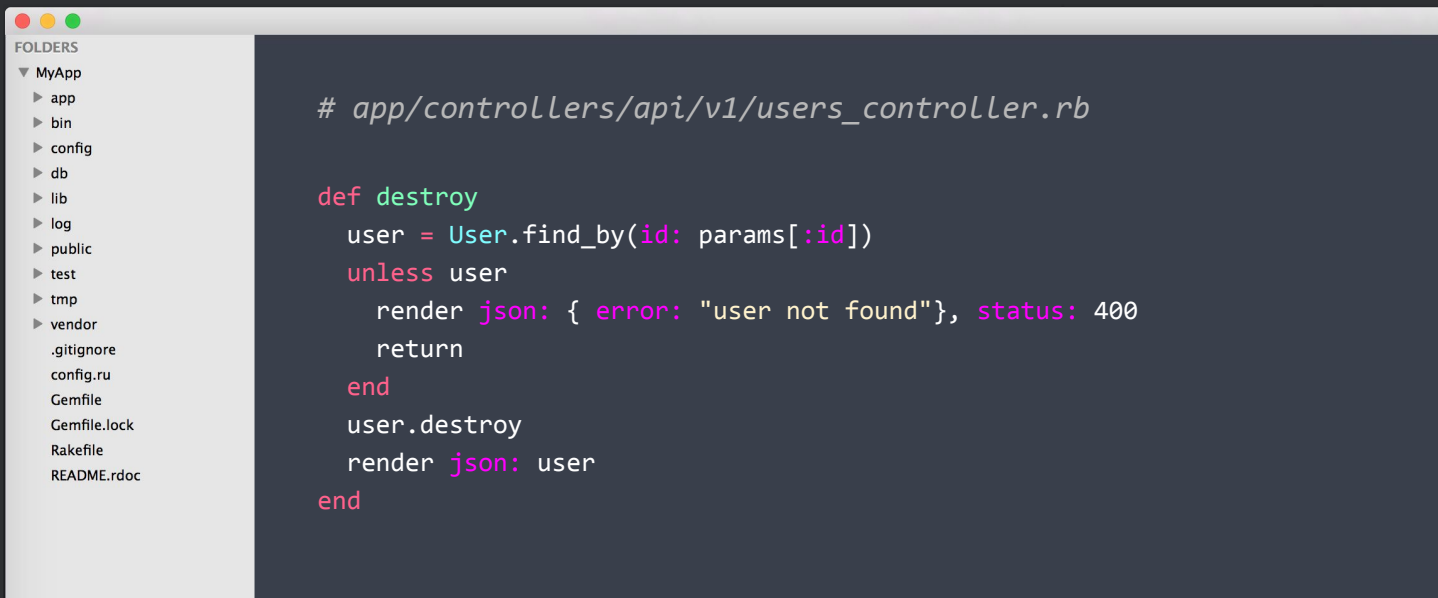


@xharekx33

# Delete a user

---

Add a controller action to destroy a user.



The screenshot shows a code editor with a sidebar on the left displaying a file tree under the name 'MyApp'. The tree includes folders like 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor', as well as files like '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area displays the content of the file `app/controllers/api/v1/users_controller.rb`. The code defines a `destroy` action that finds a user by ID, checks if it exists, and either returns a 400 status if not found or destroys the user and returns the user object as JSON.

```
# app/controllers/api/v1/users_controller.rb

def destroy
  user = User.find_by(id: params[:id])
  unless user
    render json: { error: "user not found"}, status: 400
    return
  end
  user.destroy
  render json: user
end
```



@xharekx33

# Testing you API with Postman

We can test the index action with our browser, but to test the other actions we'd need to create views and forms. It's too much work.

Instead we can use Postman, a Chrome extension that helps you build, test, and document APIs

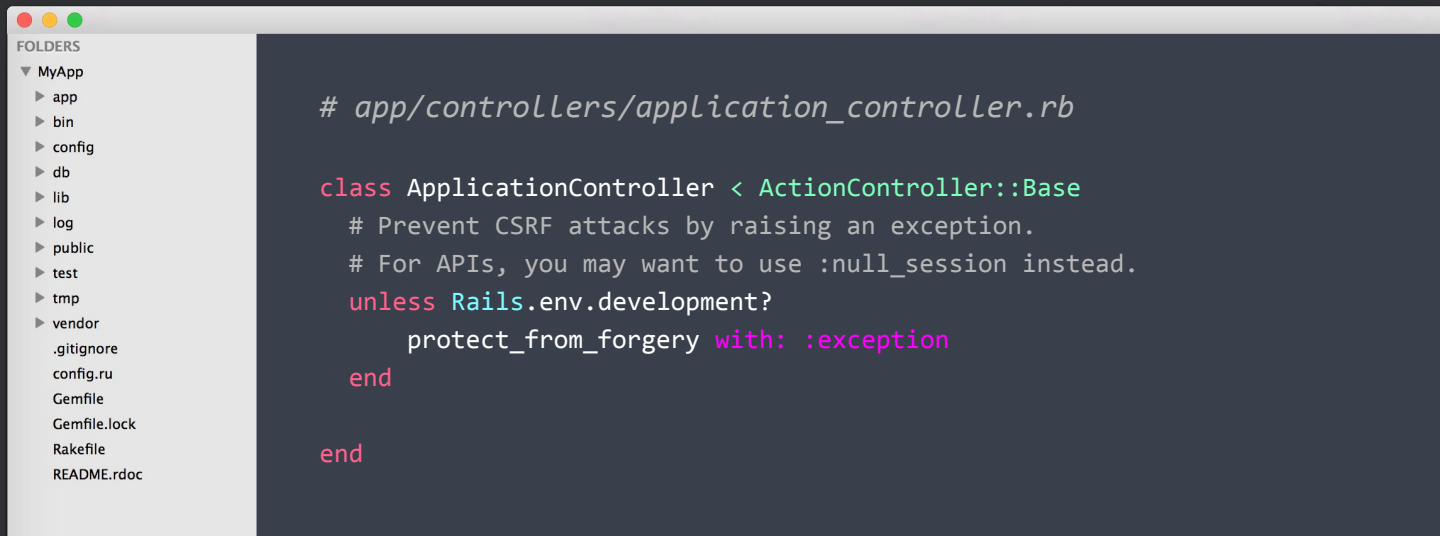
Download it here:

<https://chrome.google.com/webstore/detail/postman/fhbjgfbiflinjbdgggehcddcbncdddomop>



# Make the app work with Postman

There is a small thing to change so Postman can work:  
`protect_from_forgery` needs to be removed from the application controller.



```
# app/controllers/application_controller.rb

class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  unless Rails.env.development?
    protect_from_forgery with: :exception
  end
end
```



@xharekx33

# Adding Tasks

---

Tasks will have names and due dates and will belong to a User. We can use the `rails g resource` shortcut to create the resource, and make the necessary changes to namespace everything as before before running `rake db:migrate`

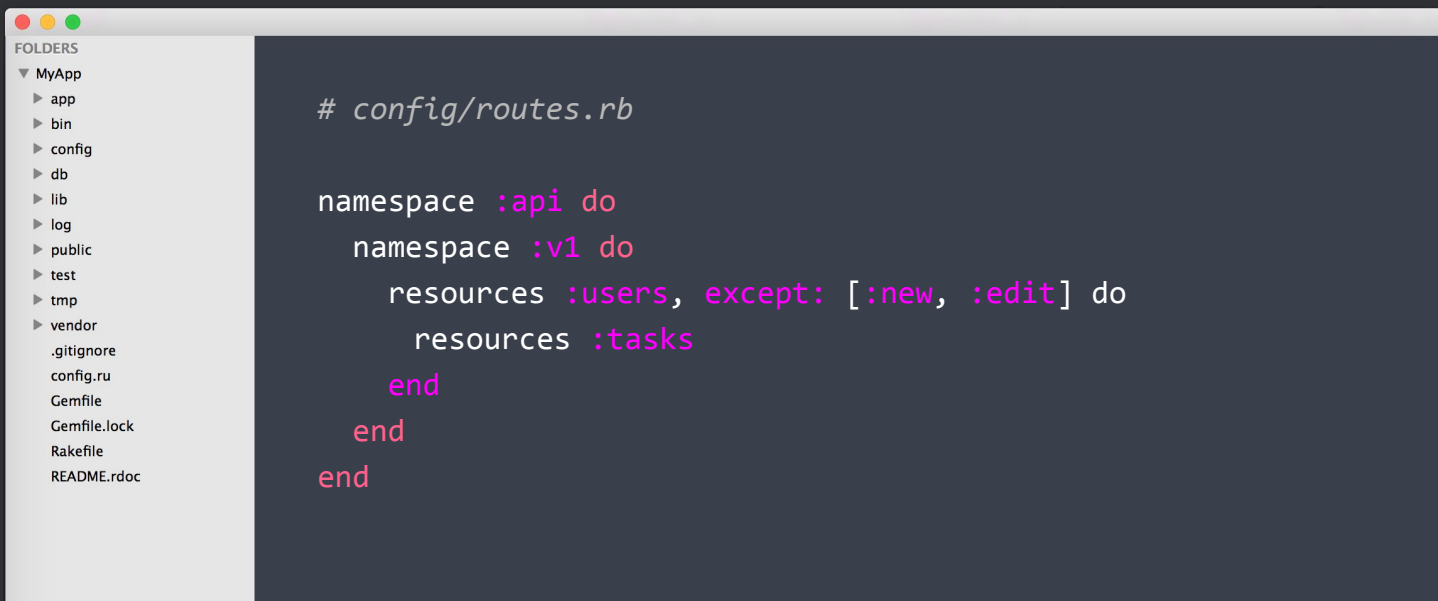
```
$ rails g resource task name:string due_date:datetime
user_id:integer
invoke active_record
  create db/migrate/20150809211032_create_tasks.rb
  create app/models/task.rb
  invoke test_unit
    create      test/models/task_test.rb
    create      test/fixtures/tasks.yml
  invoke controller
```



@xharekx33

# Nested resources

Remember that Tasks belong to Users. We'll change the routes file to capture this relationship. Use **rake routes** to check them out.



```
# config/routes.rb

namespace :api do
  namespace :v1 do
    resources :users, except: [:new, :edit] do
      resources :tasks
    end
  end
end
```

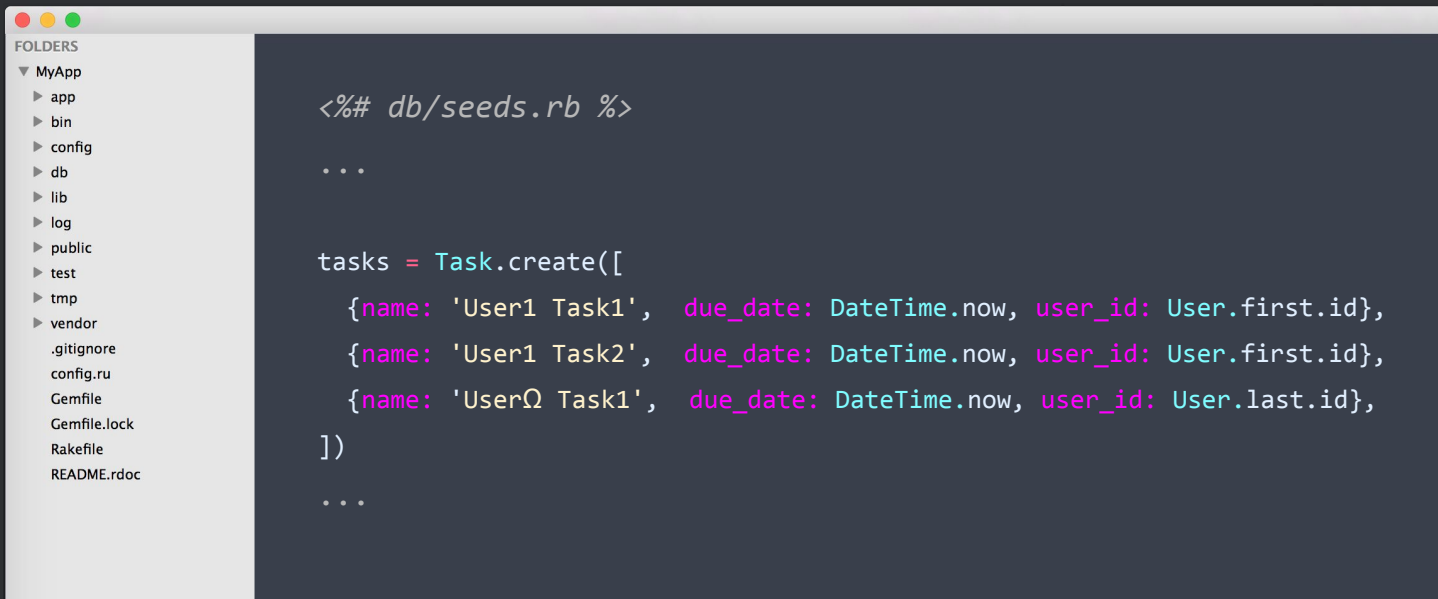
The screenshot shows a code editor window with a sidebar on the left titled "FOLDERS". The sidebar lists the following items: MyApp (expanded), app, bin, config, db, lib, log, public, test, tmp, vendor, .gitignore, config.ru, Gemfile, Gemfile.lock, Rakefile, and README.rdoc. The main editor area displays the content of the routes.rb file, which defines a nested resource structure for users and tasks within the :api namespace.



@xharekx33

# Seed database

Let's add some tasks for our users to the seed file and run `rake db:seed`



The screenshot shows a code editor with a sidebar on the left displaying the project structure under 'FOLDERS'. The main editor area shows the content of `db/seeds.rb`. The code defines three tasks for users: 'User1 Task1', 'User1 Task2', and 'UserΩ Task1', each with a due date of `DateTime.now` and assigned to a specific user.

```
<%= db/seeds.rb %>

...

tasks = Task.create([
  {name: 'User1 Task1', due_date: DateTime.now, user_id: User.first.id},
  {name: 'User1 Task2', due_date: DateTime.now, user_id: User.first.id},
  {name: 'UserΩ Task1', due_date: DateTime.now, user_id: User.last.id},
])

...
```

**FOLDERS**

- ▼ MyApp
  - ▶ app
  - ▶ bin
  - ▶ config
  - ▶ db
  - ▶ lib
  - ▶ log
  - ▶ public
  - ▶ test
  - ▶ tmp
  - ▶ vendor
- .gitignore
- config.ru
- Gemfile
- Gemfile.lock
- Rakefile
- README.rdoc



@xharekx33

# Exercise

Add the necessary controller methods so tasks can be:

- created
- shown
- deleted
- completed





# Control the output

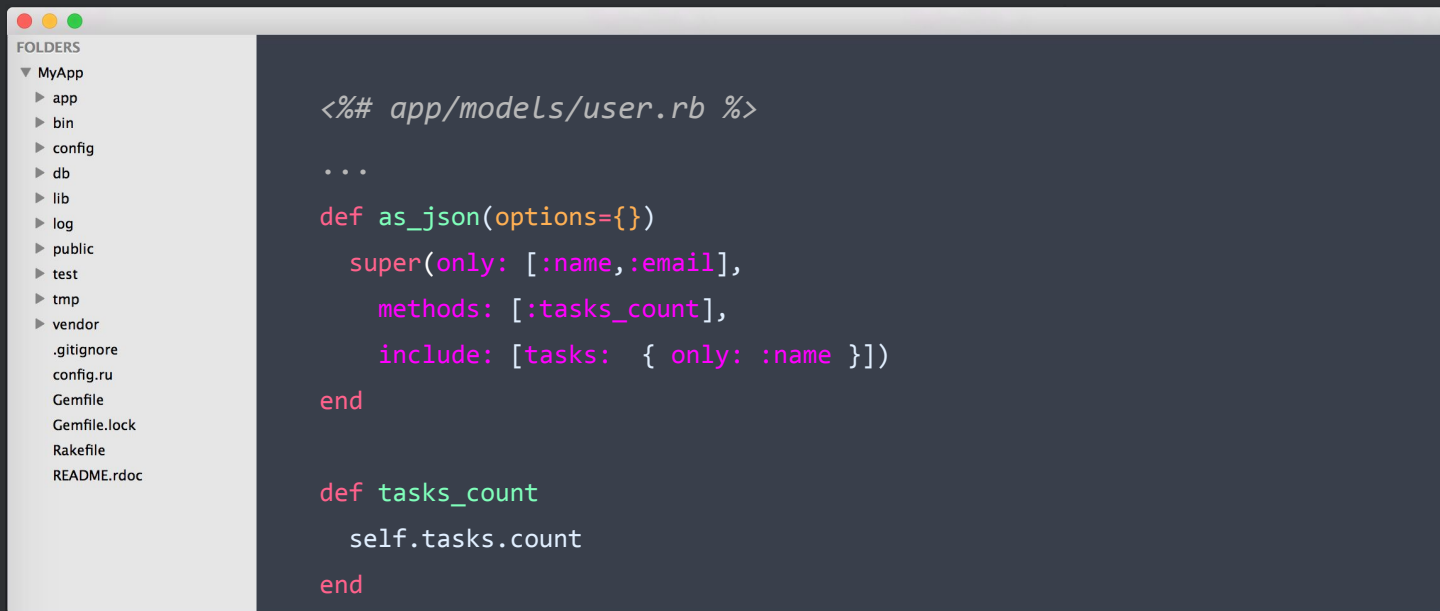
To control what is included in your JSON response, just add a `as_json` method to your model, make a call to its implementation in the superclass and include options:

- `:only` and `:except` options can be used to limit the attributes included
- `:method` to include the result of some method calls on the model
- `:include` to include associations



# to\_json

An example for the User model:



The screenshot shows a code editor with a sidebar on the left displaying a file tree for a project named 'MyApp'. The main editor area contains the following Ruby code for the `to_json` method in `app/models/user.rb`:

```
<%= app/models/user.rb %>

...

def as_json(options={})
  super(only: [:name, :email],
        methods: [:tasks_count],
        include: [tasks: { only: :name }])
end

def tasks_count
  self.tasks.count
end
```



@xharekx33

# Exercise

Make sure that:

- Tasks return no id, created\_at or updated\_at
- Group completed and pending tasks in the index

