

In [part one](#) of this tutorial, we managed to generate an API-only Rails application, set up a testing framework, and use TDD to implement the todo API.

Json Web Token  
Authorize Api Request  
Authenticate User  
Authentication Controller  
Conclusion

In this part of the tutorial, we'll implement token-based authentication with [JWT](#) (JSON Web Tokens). In this implementation, we'll proceed with our approach of using TDD principles to add the authentication features.

## Authentication

Our API should be able to support user accounts with each user having the ability managing their own resources. We'll adapt [Hristo's](#) approach with notable alterations.

First, generate a user model.

```
$ rails g model User name:string email:string password_digest:string
# run the migrations
$ rails db:migrate
# make sure the test environment is ready
$ rails db:test:prepare
```

If you're wondering why we have `password_digest` field instead of a normal `password` field, hang tight, we'll go over this soon enough. :)

Let's define the user model spec.

```
# spec/models/user_spec.rb
require 'rails_helper'

# Test suite for User model
RSpec.describe User, type: :model do
```

```
# Association test
# ensure User model has a 1:m relationship with the Todo model
it { should have_many(:todos) }
# Validation tests
# ensure name, email and password_digest are present before save
it { should validate_presence_of(:name) }
it { should validate_presence_of(:email) }
it { should validate_presence_of(:password_digest) }
end
```

Users should be able to manage their own todo lists. Thus, the user model should have a one to many relationship with the todo model. We also want to make sure that on every user account creation we have all the required credentials.

Let's add a user factory. This will be used by our test suite to create test users.

```
# spec/factories/users.rb
FactoryBot.define do
  factory :user do
    name { Faker::Name.name }
    email 'foo@bar.com'
    password 'foobar'
  end
end
```

Run the tests and...

```
🐞 todos-api-part-2 git:(98304d3) rspec
```

```
Item
```

```
  should belong to todo
  should validate that :name cannot be empty/falsy
```

```
Todo
```

```
  should have many items dependent => destroy
  should validate that :title cannot be empty/falsy
  should validate that :created_by cannot be empty/falsy
```

```
User
```

```
  should have many todos (FAILED - 1)
  should validate that :name cannot be empty/falsy (FAILED - 2)
  should validate that :email cannot be empty/falsy (FAILED - 3)
  should validate that :password_digest cannot be empty/falsy (FAILED - 4)
```

```
Items API
```

```
GET /todos/:todo_id/items
  when todo exists
    returns status code 200
    returns all todo items
  when todo does not exist
    returns status code 404
    returns a not found message
GET /todos/:todo_id/items/:id
  when todo item exists
```

User specs are failing as expected. Let's fix that by implementing the user model.

```
# app/models/user.rb
class User < ApplicationRecord
  # encrypt password
  has_secure_password

  # Model associations
  has_many :todos, foreign_key: :created_by

  # Validations
  validates_presence_of :name, :email, :password_digest
end
```

Our user model defines a 1:m relationship with the todo model also adds field validations. Note that user model calls the method `has_secure_password`, this adds methods to authenticate against a `bcrypt` password. It's this mechanism

that requires us to have a `password_digest` attribute. Thus, we need to have the `bcrypt` gem as a dependency.

```
# Gemfile
# [...]
# Use ActiveRecord has_secure_password
gem 'bcrypt', '~> 3.1.7'
# [...]
```

Install the gem and run the tests.

```
$ bundle install
$ bundle exec rspec
```

All green!

Model's all set up to save the users. We're going to wire up the rest of the authentication system by adding the following service classes:

- JsonWebToken - Encode and decode `jwt` tokens
- AuthorizeApiRequest - Authorize each API request
- AuthenticateUser - Authenticate users
- AuthenticationController - Orchestrate authentication process

# Json Web Token

We're going to implement [token based authentication](#). We'll make use of the [jwt](#) gem to manage JSON web tokens(jwt). Let's add this to the Gemfile and install it.

```
# Gemfile
# [...]
gem 'jwt'
# [...]
```

```
$ bundle install
```

Our class will live in the [lib](#) directory since it's not domain specific; if we were to move it to a different application it should work with minimal configuration. There's a small caveat, though...

As of Rails 5, [autoloading is disabled in production](#) because of thread safety.

This is a huge concern for us since [lib](#) is part of auto-load paths. To counter this change, we'll add our [lib](#) in [app](#) since all code in app is auto-loaded in

development and eager-loaded in production. Here's a long discussion on [the above](#).

Great, let's do this! Add the lib directory and the jwt class file.

```
# create custom lib
$ mkdir app/lib
$ touch app/lib/json_web_token.rb
```

Define jwt singleton.

```
# app/lib/json_web_token.rb
class JsonWebToken
  # secret to encode and decode token
  HMAC_SECRET = Rails.application.secrets.secret_key_base

  def self.encode(payload, exp = 24.hours.from_now)
    # set expiry to 24 hours from creation time
    payload[:exp] = exp.to_i
    # sign token with application secret
    JWT.encode(payload, HMAC_SECRET)
  end

  def self.decode(token)
    # get payload; first index in decoded Array
    body = JWT.decode(token, HMAC_SECRET)[0]
```

```

      HashWithIndifferentAccess.new body
    # rescue from all decode errors
  rescue JWT::DecodeError => e
    # raise custom error to be handled by custom handler
    raise ExceptionHandler::InvalidToken, e.message
  end
end
end

```

This singleton wraps `JWT` to provide token encoding and decoding methods. The encode method will be responsible for creating tokens based on a payload (user id) and expiration period. Since every Rails application has a unique secret key, we'll use that as our secret to sign tokens. The decode method, on the other hand, accepts a token and attempts to decode it using the same secret used in encoding. In the event decoding fails, be it due to expiration or validation, `JWT` will raise respective exceptions which will be caught and handled by the `ExceptionHandler` module.

```

module ExceptionHandler
  extend ActiveSupport::Concern

  # Define custom error subclasses - rescue catches `StandardErrors`
  class AuthenticationError < StandardError; end
  class MissingToken < StandardError; end
  class InvalidToken < StandardError; end

  included do
    # Define custom handlers
    rescue_from ActiveRecord::RecordInvalid, with: :four_twenty_two
    rescue_from ExceptionHandler::AuthenticationError, with: :unauthor:
    rescue_from ExceptionHandler::MissingToken, with: :four_twenty_two
    rescue_from ExceptionHandler::InvalidToken, with: :four_twenty_two

    rescue_from ActiveRecord::RecordNotFound do |e|
      json_response({ message: e.message }, :not_found)
    end
  end
end

```



```

        end
    end

    private

    # JSON response with message; Status code 422 - unprocessable entity
    def four_twenty_two(e)
        json_response({ message: e.message }, :unprocessable_entity)
    end

    # JSON response with message; Status code 401 - Unauthorized
    def unauthorized_request(e)
        json_response({ message: e.message }, :unauthorized)
    end
end
end

```

We've defined custom `Standard Error` sub-classes to help handle exceptions raised. By defining error classes as sub-classes of standard error, we're able to `rescue_from` them once raised.

## Authorize Api Request

This class will be responsible for authorizing all API requests making sure that all requests have a valid token and user payload.

Since this is an authentication service class, it'll live in `app/auth`.

```

# create auth folder to house auth services
$ mkdir app/auth
$ touch app/auth/authorize_api_request.rb
# Create corresponding spec files
$ mkdir spec/auth

```

```
$ touch spec/auth/authorize_api_request_spec.rb
```

Let's define its specifications

```
# spec/auth/authorize_api_request_spec.rb
require 'rails_helper'

RSpec.describe AuthorizeApiRequest do
  # Create test user
  let(:user) { create(:user) }

  # Mock `Authorization` header
  let(:header) { { 'Authorization' => token_generator(user.id) } }

  # Invalid request subject
  subject(:invalid_request_obj) { described_class.new({}) }

  # Valid request subject
  subject(:request_obj) { described_class.new(header) }

  # Test Suite for AuthorizeApiRequest#call
  # This is our entry point into the service class
  describe '#call' do
    # returns user object when request is valid
    context 'when valid request' do
      it 'returns user object' do
        result = request_obj.call
        expect(result[:user]).to eq(user)
      end
    end

    # returns error message when invalid request
    context 'when invalid request' do
      context 'when missing token' do
        it 'raises a MissingToken error' do
          expect { invalid_request_obj.call }
        end
      end
    end
  end
end
```

```

        .to raise_error(ExceptionHandler::MissingToken, 'Missing token')
      end
    end
  end

  context 'when invalid token' do
    subject(:invalid_request_obj) do
      # custom helper method `token_generator`
      described_class.new('Authorization' => token_generator(5))
    end

    it 'raises an InvalidToken error' do
      expect { invalid_request_obj.call }
        .to raise_error(ExceptionHandler::InvalidToken, /Invalid token/)
    end
  end

  context 'when token is expired' do
    let(:header) { { 'Authorization' => expired_token_generator(subject(:request_obj)) } }
    subject(:request_obj) { described_class.new(header) }

    it 'raises ExceptionHandler::ExpiredSignature error' do
      expect { request_obj.call }
        .to raise_error(
          ExceptionHandler::InvalidToken,
          /Signature has expired/
        )
    end
  end

  context 'fake token' do
    let(:header) { { 'Authorization' => 'foobar' } }
    subject(:invalid_request_obj) { described_class.new(header) }

    it 'handles JWT::DecodeError' do
      expect { invalid_request_obj.call }
        .to raise_error(
          ExceptionHandler::InvalidToken,
          /Not enough or too many segments/
        )
    end
  end
end

```

```
        end
      end
    end
  end
end
```

The `AuthorizeApiRequest` service should have an entry method `call` that returns a valid user object when the request is valid and raises an error when invalid. Note that we also have a couple of test helper methods;

- `token_generator` - generate test token
- `expired_token_generator` - generate expired token

We'll define these helpers in `spec/support` .

```
# create module file
$ touch spec/support/controller_spec_helper.rb
```

```
# spec/support/controller_spec_helper.rb
module ControllerSpecHelper
  # generate tokens from user id
  def token_generator(user_id)
    JsonWebToken.encode(user_id: user_id)
  end

  # generate expired tokens from user id
```

```

def expired_token_generator(user_id)
  JsonWebToken.encode({ user_id: user_id }, (Time.now.to_i - 10))
end

# return valid headers
def valid_headers
  {
    "Authorization" => token_generator(user.id),
    "Content-Type" => "application/json"
  }
end

# return invalid headers
def invalid_headers
  {
    "Authorization" => nil,
    "Content-Type" => "application/json"
  }
end
end
end

```

We also have additional test helpers to generate headers. In order to make use of these helper methods, we have to include the module in `rails_helper`. While we're here let's also include `RequestSpecHelper` to all types (not just requests); remove `type: :request`. This way, we'll be able to reuse our handy `json` helper.

```

RSpec.configure do |config|
  # [...]
  # previously `config.include RequestSpecHelper, type: :request`
  config.include RequestSpecHelper
  config.include ControllerSpecHelper
end

```

```
# [...]  
end
```

At this point, if you attempt to run the tests, You should get a load error. You guessed it, this is because we haven't defined the class. Let's do just that!

```
bundler: failed to load command: rspec (/Users/austin/.rbenv/versions/2.3.0/bin/rspec)  
LoadError: Unable to autoload constant AuthorizeApiRequest, expected /Users/austin/projects/akabiru/todos-api-part-2/app/auth/authorize_api_request.rb to define it  
/Users/austin/.rbenv/versions/2.3.0/lib/ruby/gems/2.3.0/gems/activesupport-5.0.1/lib/active_support/dependencies.rb:512:in `load_missing_constant'  
/Users/austin/.rbenv/versions/2.3.0/lib/ruby/gems/2.3.0/gems/activesupport-5.0.1/lib/active_support/dependencies.rb:203:in `const_missing'
```

```
# app/auth/authorize_api_request.rb  
class AuthorizeApiRequest  
  def initialize(headers = {})  
    @headers = headers  
  end  
  
  # Service entry point - return valid user object  
  def call  
    {  
      user: user  
    }  
  end  
  
  private  
  
  attr_reader :headers  
  
  def user  
    # check if user is in the database  
    # memoize user object  
    @user ||= User.find(decoded_auth_token[:user_id]) if decoded_auth_t
```

```

    # handle user not found
  rescue ActiveRecord::RecordNotFound => e
    # raise custom error
    raise(
      ExceptionHandler::InvalidToken,
      ("#{Message.invalid_token} #{e.message}")
    )
  end

  # decode authentication token
  def decoded_auth_token
    @decoded_auth_token ||= JsonWebToken.decode(http_auth_header)
  end

  # check for token in `Authorization` header
  def http_auth_header
    if headers['Authorization'].present?
      return headers['Authorization'].split(' ').last
    end
    raise(ExceptionHandler::MissingToken, Message.missing_token)
  end
end

```

The `AuthorizeApiRequest` service gets the token from the authorization headers, attempts to decode it to return a valid user object. We also have a singleton `Message` to house all our messages; this an easier way to manage our application messages. We'll define it in `app/lib` since it's non-domain-specific.

```

# app/lib/message.rb
class Message
  def self.not_found(record = 'record')
    "Sorry, #{record} not found."
  end
end

```

```
def self.invalid_credentials
  'Invalid credentials'
end

def self.invalid_token
  'Invalid token'
end

def self.missing_token
  'Missing token'
end

def self.unauthorized
  'Unauthorized request'
end

def self.account_created
  'Account created successfully'
end

def self.account_not_created
  'Account could not be created'
end

def self.expired_token
  'Sorry, your token has expired. Please login to continue.'
end
end
```

Run the auth specs and everything should be green.

```
$ bundle exec rspec spec/auth -fd
```



## Authenticate User

This class will be responsible for authenticating users via email and password.

Since this is also an authentication service class, it'll live in `app/auth` .

```
$ touch app/auth/authenticate_user.rb
# Create corresponding spec file
$ touch spec/auth/authenticate_user_spec.rb
```

Let's define its specifications.

```

# spec/auth/authenticate_user_spec.rb
require 'rails_helper'

RSpec.describe AuthenticateUser do
  # create test user
  let(:user) { create(:user) }
  # valid request subject
  subject(:valid_auth_obj) { described_class.new(user.email, user.password) }
  # invalid request subject
  subject(:invalid_auth_obj) { described_class.new('foo', 'bar') }

  # Test suite for AuthenticateUser#call
  describe '#call' do
    # return token when valid request
    context 'when valid credentials' do
      it 'returns an auth token' do
        token = valid_auth_obj.call
        expect(token).not_to be_nil
      end
    end
  end

  # raise Authentication Error when invalid request
  context 'when invalid credentials' do
    it 'raises an authentication error' do
      expect { invalid_auth_obj.call }
        .to raise_error(
          ExceptionHandler::AuthenticationError,
          /Invalid credentials/
        )
    end
  end
end

```

The `AuthenticateUser` service also has an entry point `#call` . It should return a token when user credentials are valid and raise an error when they're not. Running the auth specs and they should fail with a load error. Let's go ahead and implement the class.

```
# app/auth/authenticate_user.rb
class AuthenticateUser
  def initialize(email, password)
    @email = email
    @password = password
  end

  # Service entry point
  def call
    JsonWebToken.encode(user_id: user.id) if user
  end

  private

  attr_reader :email, :password

  # verify user credentials
  def user
    user = User.find_by(email: email)
    return user if user && user.authenticate(password)
    # raise Authentication error if credentials are invalid
    raise(ExceptionHandler::AuthenticationError, Message.invalid_credentials)
  end
end
```

The `AuthenticateUser` service accepts a user email and password, checks if they are valid and then creates a token with the user id as the payload.

```
$ bundle exec rspec spec/auth -fd
```

```
todos-api-part-2 git:(master) x bundle exec rspec spec/auth -fd
```

```
AuthorizeApiRequest
```

```
#call
```

```
  when valid request
```

```
    returns user object
```

```
  when invalid request
```

```
    when missing token
```

```
      raises a MissingToken error
```

```
    when invalid token
```

```
      raises an InvalidToken error
```

```
    when token is expired
```

```
      raises ExceptionHandler::ExpiredSignature error
```

```
Finished in 0.94746 seconds (files took 6.67 seconds to load)
```

```
4 examples, 0 failures
```



## Authentication Controller

This controller will be responsible for orchestrating the authentication process making use of the auth service we have just created.

```
# generate the Authentication Controller
```

```
$ rails g controller Authentication
```

First thing's first. Tests!

```

# spec/requests/authentication_spec.rb
require 'rails_helper'

RSpec.describe 'Authentication', type: :request do
  # Authentication test suite
  describe 'POST /auth/login' do
    # create test user
    let!(:user) { create(:user) }
    # set headers for authorization
    let(:headers) { valid_headers.except('Authorization') }
    # set test valid and invalid credentials
    let(:valid_credentials) do
      {
        email: user.email,
        password: user.password
      }.to_json
    end
    let(:invalid_credentials) do
      {
        email: Faker::Internet.email,
        password: Faker::Internet.password
      }.to_json
    end

    # set request.headers to our custom headers
    # before { allow(request).to receive(:headers).and_return(headers) }

    # returns auth token when request is valid
    context 'When request is valid' do
      before { post '/auth/login', params: valid_credentials, headers: headers }

      it 'returns an authentication token' do
        expect(json['auth_token']).not_to be_nil
      end
    end
  end
end

```

```

# returns failure message when request is invalid
context 'When request is invalid' do
  before { post '/auth/login', params: invalid_credentials, headers: {} }

  it 'returns a failure message' do
    expect(json['message']).to match(/Invalid credentials/)
  end
end
end
end
end

```

The authentication controller should expose an `/auth/login` endpoint that accepts user credentials and returns a JSON response with the result.

```

# app/controllers/authentication_controller.rb
class AuthenticationController < ApplicationController
  # return auth token once user is authenticated
  def authenticate
    auth_token =
      AuthenticateUser.new(auth_params[:email], auth_params[:password])
      .json_response(auth_token: auth_token)
  end

  private

  def auth_params
    params.permit(:email, :password)
  end
end
end

```

Notice how slim the authentication controller is, we have our service architecture to thank for that. Instead, we make use of the authentication controller to piece everything together... to control authentication. We also need to add routing for authentication action.

```
# config/routes.rb
Rails.application.routes.draw do
  # [...]
  post 'auth/login', to: 'authentication#authenticate'
end
```

In order to have users to authenticate in the first place, we need to have them signup first. This will be handled by the users controller.

```
# generate users controller
$ rails g controller Users
# generate users request spec
$ touch spec/requests/users_spec.rb
```

User signup spec.

```
# spec/requests/users_spec.rb
require 'rails_helper'
```

```

RSpec.describe 'Users API', type: :request do
  let(:user) { build(:user) }
  let(:headers) { valid_headers.except('Authorization') }
  let(:valid_attributes) do
    attributes_for(:user, password_confirmation: user.password)
  end

  # User signup test suite
  describe 'POST /signup' do
    context 'when valid request' do
      before { post '/signup', params: valid_attributes.to_json, headers: headers }

      it 'creates a new user' do
        expect(response).to have_http_status(201)
      end

      it 'returns success message' do
        expect(json['message']).to match(/Account created successfully/)
      end

      it 'returns an authentication token' do
        expect(json['auth_token']).not_to be_nil
      end
    end

    context 'when invalid request' do
      before { post '/signup', params: {}, headers: headers }

      it 'does not create a new user' do
        expect(response).to have_http_status(422)
      end

      it 'returns failure message' do
        expect(json['message'])
          .to match(/Validation failed: Password can't be blank, Name can't be blank/)
      end
    end
  end
end

```



The user controller should expose a `/signup` endpoint that accepts user information and returns a JSON response with the result. Add the signup route.

```
# config/routes.rb
Rails.application.routes.draw do
  # [...]
  post 'signup', to: 'users#create'
end
```

And then implement the controller.

```
# app/controllers/users_controller.rb
class UsersController < ApplicationController
  # POST /signup
  # return authenticated token upon signup
  def create
    user = User.create!(user_params)
    auth_token = AuthenticateUser.new(user.email, user.password).call
    response = { message: Message.account_created, auth_token: auth_token }
    json_response(response, :created)
  end

  private

  def user_params
    params.permit(
      :name,
```

```

      :email,
      :password,
      :password_confirmation
    )
  end
end

```

The users controller attempts to create a user and returns a JSON response with the result. We use Active Record's `create!` method so that in the event there's an error, an exception will be raised and handled in the exception handler.

One more thing, we've wired up the user authentication bit but our API is still open; it does not authorize requests with a token.

The image shows two terminal windows side-by-side. The left window, titled 'rails (fsevent\_watch)', shows the Rails server starting up. It includes messages about booting Puma, Rails 5.0.1 application starting on http://localhost:3000, and various startup options. It then shows a GET request to '/todos' being processed, with Active Record queries for schema migrations and todos. The right window, titled '..os-api-part-2 (zsh)', shows the response to the GET request: 'HTTP/1.1 200 OK'. It lists headers: 'Cache-Control: max-age=0, private, must-revalidate', 'Content-Type: application/json; charset=utf-8', 'ETag: W/"4f53cda18c2baa0c0354bb5f9a3ecbe5"', 'Transfer-Encoding: chunked', 'X-Content-Type-Options: nosniff', 'X-Frame-Options: SAMEORIGIN', 'X-Request-Id: 9f93bb53-d109-457a-9cd2-0f0d0ffbc8a5', 'X-Runtime: 0.770962', and 'X-XSS-Protection: 1; mode=block'.

```

rails (fsevent_watch)
todos-api-part-2 git:(master) x rails s
=> Booting Puma
=> Rails 5.0.1 application starting in development on http://localhost:3000
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.6.2 (ruby 2.3.0-p0), codename: Sleepy Sunday Serenity
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://localhost:3000
Use Ctrl-C to stop
Started GET "/" for ::1 at 2017-03-05 23:41:09 +0300
ActiveRecord::SchemaMigration Load (0.1ms) SELECT "schema_migrations".* FROM "schema_migrations"
Processing by TodosController#index as */*
  Todo Load (0.3ms) SELECT "todos".* FROM "todos"
Completed 200 OK in 11ms (Views: 1.4ms | ActiveRecord: 0.6ms)

..os-api-part-2 (zsh)
todos-api-part-2 git:(master) x http :3000/todos
HTTP/1.1 200 OK
Cache-Control: max-age=0, private, must-revalidate
Content-Type: application/json; charset=utf-8
ETag: W/"4f53cda18c2baa0c0354bb5f9a3ecbe5"
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Request-Id: 9f93bb53-d109-457a-9cd2-0f0d0ffbc8a5
X-Runtime: 0.770962
X-XSS-Protection: 1; mode=block

```

To fix this, we have to make sure that on every request (except authentication) our API checks for a valid token. To achieve this, we'll implement a callback in the application controller that authenticates every request. Since all controllers inherit from application controller, it will be propagated to all controllers.

```

# spec/controllers/application_controller_spec.rb
require "rails_helper"

RSpec.describe ApplicationController, type: :controller do
  # create test user
  let!(:user) { create(:user) }
  # set headers for authorization
  let(:headers) { { 'Authorization' => token_generator(user.id) } }
  let(:invalid_headers) { { 'Authorization' => nil } }

  describe "#authorize_request" do
    context "when auth token is passed" do
      before { allow(request).to receive(:headers).and_return(headers) }

      # private method authorize_request returns current user
      it "sets the current user" do
        expect(subject.instance_eval { authorize_request }).to eq(user)
      end
    end

    context "when auth token is not passed" do
      before do
        allow(request).to receive(:headers).and_return(invalid_headers)
      end

      it "raises MissingToken error" do
        expect { subject.instance_eval { authorize_request } }.
          to raise_error(ExceptionHandler::MissingToken, /Missing token/)
      end
    end
  end
end

```

Cool, now that we have the tests, let's implement the authorization.

```

# app/controllers/application_controller.rb
class ApplicationController < ActionController::API
  include Response
  include ErrorHandler

  # called before every action on controllers
  before_action :authorize_request
  attr_reader :current_user

  private

  # Check for valid request token and return user
  def authorize_request
    @current_user = (AuthorizeApiRequest.new(request.headers).call)[:u
  end
end

```

On every request, the application will verify the request by calling the request authorization service. If the request is authorized, it will set the `current user` object to be used in the other controllers.

Notice how we don't have lots of guard clauses and conditionals in our controllers, this is because of our error handling implementation.

Let's remember that when signing up and authenticating a user we won't need a token. We'll only require user credentials. Thus, let's skip request authentication for these two actions.

First, the authentication action.

```
# app/controllers/authentication_controller.rb
class AuthenticationController < ApplicationController

  skip_before_action :authorize_request, only: :authenticate
  # [...]
end
```

Then the user signup action.

```
# app/controllers/users_controller.rb
class UsersController < ApplicationController
  skip_before_action :authorize_request, only: :create
  # [...]
end
```

Run the tests and you'll notice, our Todo and TodoItems API is failing. Don't fret, this is exactly what we want; means our request authorization is working as intended. Let's update the API to cater for this.

In the Todos request spec, we'll make partial update all our requests to have authorization headers and a JSON payload.

```
# spec/requests/todos_spec.rb
require 'rails_helper'

RSpec.describe 'Todos API', type: :request do
  # add todos owner
```

```

let(:user) { create(:user) }
let!(:todos) { create_list(:todo, 10, created_by: user.id) }
let(:todo_id) { todos.first.id }
# authorize request
let(:headers) { valid_headers }

describe 'GET /todos' do
  # update request with headers
  before { get '/todos', params: {}, headers: headers }

  # [...]
end

describe 'GET /todos/:id' do
  before { get "/todos/#{todo_id}", params: {}, headers: headers }
  # [...]
  end
  # [...]
end

describe 'POST /todos' do
  let(:valid_attributes) do
    # send json payload
    { title: 'Learn Elm', created_by: user.id.to_s }.to_json
  end

  context 'when request is valid' do
    before { post '/todos', params: valid_attributes, headers: headers }
    # [...]
  end

  context 'when the request is invalid' do
    let(:invalid_attributes) { { title: nil }.to_json }
    before { post '/todos', params: invalid_attributes, headers: headers }

    it 'returns status code 422' do
      expect(response).to have_http_status(422)
    end
  end
end

```

```

    it 'returns a validation failure message' do
      expect(json['message'])
        .to match(/Validation failed: Title can't be blank/)
    end
  end
end

describe 'PUT /todos/:id' do
  let(:valid_attributes) { { title: 'Shopping' }.to_json }

  context 'when the record exists' do
    before { put "/todos/#{todo_id}", params: valid_attributes, headers: headers }
    # [...]
  end
end

describe 'DELETE /todos/:id' do
  before { delete "/todos/#{todo_id}", params: {}, headers: headers }
  # [...]
end
end

```

Our todos controller doesn't know about users yet. Let's fix that.

```

# app/controllers/todos_controller.rb
class TodosController < ApplicationController
  # [...]
  # GET /todos
  def index
    # get current user todos
    @todos = current_user.todos
    json_response(@todos)
  end
  # [...]
  # POST /todos

```

```

# POST /todos
def create
  # create todos belonging to current user
  @todo = current_user.todos.create!(todo_params)
  json_response(@todo, :created)
end

# [...]
private

# remove `created_by` from list of permitted parameters
def todo_params
  params.permit(:title)
end

# [...]
end

```

Let's update the Items API with the same.

```

# spec/requests/items_spec.rb
require 'rails_helper'

RSpec.describe 'Items API' do
  let(:user) { create(:user) }
  let!(:todo) { create(:todo, created_by: user.id) }
  let!(:items) { create_list(:item, 20, todo_id: todo.id) }
  let(:todo_id) { todo.id }
  let(:id) { items.first.id }
  let(:headers) { valid_headers }

  describe 'GET /todos/:todo_id/items' do
    before { get "/todos/#{todo_id}/items", params: {}, headers: headers }

    # [...]
  end
end

```



```
end
```

```
describe 'GET /todos/:todo_id/items/:id' do
  before { get "/todos/#{todo_id}/items/#{id}", params: {}, headers:
```

```
    # [...]
```

```
end
```

```
describe 'POST /todos/:todo_id/items' do
  let(:valid_attributes) { { name: 'Visit Narnia', done: false }.to_h }
```

```
  context 'when request attributes are valid' do
```

```
    before do
```

```
      post "/todos/#{todo_id}/items", params: valid_attributes, headers:
```

```
    end
```

```
    # [...]
```

```
  end
```

```
  context 'when an invalid request' do
```

```
    before { post "/todos/#{todo_id}/items", params: {}, headers: headers }
```

```
    # [...]
```

```
  end
```

```
end
```

```
describe 'PUT /todos/:todo_id/items/:id' do
  let(:valid_attributes) { { name: 'Mozart' }.to_json }
```

```
  before do
```

```
    put "/todos/#{todo_id}/items/#{id}", params: valid_attributes, headers:
```

```
  end
```

```
  # [...]
```

```
  # [...]
```

```
end
```

```
describe 'DELETE /todos/:id' do
```

```
  before { delete "/todos/#{todo_id}/items/#{id}", params: {}, headers:
```

```

        # [...]
    end
end

```

Awesome, our specs are now up to date! Phew!

```
👉 todos-api-part-2 git:(master) ✖ bundle exec rspec
.....

Finished in 3.14 seconds (files took 7.23 seconds to load)
53 examples, 0 failures
```

Let's fire up the server for some manual testing.

[illegible]

```

...-api-part-2 (ssh)
● todos-api-part-2 git:(master) http :3000/todos
HTTP/1.1 422 Unprocessable Entity
Cache-Control: no-cache
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Request-Id: 2672a7c4-85c7-4789-b581-ec13f25dcfad
X-Runtime: 0.021686
X-XSS-Protection: 1; mode=block

{
  "message": "Missing token"
}

● todos-api-part-2 git:(master) http :3000/signup name=ash email=ash@email.com password=foobar password_confirmation=foobar
HTTP/1.1 201 Created
Cache-Control: max-age=0, private, must-revalidate
Content-Type: application/json; charset=utf-8
ETag: W/"900bf169cd316128c993db5a32799e9a"
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Request-Id: 3c7957c2-6a77-41ac-9ca0-056a516700fc
X-Runtime: 0.223338
X-XSS-Protection: 1; mode=block

{
  "auth_token": "eyJ0eXA0IjKXVlQlCjhbGciOiJIUzI1NiJ9.eyJ1c2VyX2kiIjozIjEhA10jE000gSMDEyNjR9.7txvLgDzFdXSNIUGYb3M45oNIXlmeB_1TU3jdIG50ds",
  "message": "Account created successfully"
}

● todos-api-part-2 git:(master) http :3000/todos \
  > Authorization: 'eyJ0eXA0IjKXVlQlCjhbGciOiJIUzI1NiJ9.eyJ1c2VyX2kiIjozIjEhA10jE000gSMDEyNjR9.7txvLgDzFdXSNIUGYb3M45oNIXlmeB_1TU3jdIG50ds'
HTTP/1.1 200 OK
Cache-Control: max-age=0, private, must-revalidate
Content-Type: application/json; charset=utf-8
ETag: W/"4f53da18c2ba08c0354bb5f9a3eche5"
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Request-Id: 5881048b-b589-4fba-a1fb-dc4239efc968
X-Runtime: 0.010866
X-XSS-Protection: 1; mode=block

[]

● todos-api-part-2 git:(master) _

...-api-part-2 (ssh)
● todos-api-part-2 git:(master) http POST :3000/todos title=Beethoven \
  > Authorization: 'eyJ0eXA0IjKXVlQlCjhbGciOiJIUzI1NiJ9.eyJ1c2VyX2kiIjozIjEhA10jE000gSMDEyNjR9.7txvLgDzFdXSNIUGYb3M45oNIXlmeB_1TU3jdIG50ds'
HTTP/1.1 201 Created
Cache-Control: max-age=0, private, must-revalidate
Content-Type: application/json; charset=utf-8
ETag: W/"f0e124e32c83a550a3df5bde3160cd"
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Request-Id: a479fbb9-2645-4abf-88c7-21cabe6d740c
X-Runtime: 0.181199
X-XSS-Protection: 1; mode=block

{
  "created_at": "2017-03-06T15:44:21.718Z",
  "created_by": "3",
  "id": 1,
  "title": "Beethoven",
  "updated_at": "2017-03-06T15:44:21.718Z"
}

● todos-api-part-2 git:(master) x http :3000/todos \
  > Authorization: 'eyJ0eXA0IjKXVlQlCjhbGciOiJIUzI1NiJ9.eyJ1c2VyX2kiIjozIjEhA10jE000gSMDEyNjR9.7txvLgDzFdXSNIUGYb3M45oNIXlmeB_1TU3jdIG50ds'
HTTP/1.1 200 OK
Cache-Control: max-age=0, private, must-revalidate
Content-Type: application/json; charset=utf-8
ETag: W/"f22dacl0edd2d565685e85e56490d0b5a"
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Request-Id: 9955e7e8-c697-465a-96dc-ee148bdc548f
X-Runtime: 0.006240
X-XSS-Protection: 1; mode=block

[
  {
    "created_at": "2017-03-06T15:44:21.718Z",
    "created_by": "3",
    "id": 1,
    "title": "Beethoven",
    "updated_at": "2017-03-06T15:44:21.718Z"
  }
]

● todos-api-part-2 git:(master) x _
```

# Conclusion

That's it for part two! At this point, you should have learned how to implement token based authentication with JWT.

In the [next part](#) of this tutorial, we'll wrap up with [API versioning](#) , [pagination](#) and [serialization](#) . As always, hope to see you there. Cheers!