Sinatra - Practice Sheet

10 Q&A on routes, middleware, templates, helpers, and testing

Q1. What is Sinatra and how does it differ from Ruby on Rails?

A: Sinatra is a DSL for building web apps in Ruby with minimal structure. There is no MVC, no ORM, no asset pipeline by default. You define routes with get/post/put/delete and return a string or render a template. Rails is a full framework; Sinatra is a micro-framework.

Q2. How do you define and read URL parameters in Sinatra?

A: get '/users/:id' do; user_id = params[:id]; end. Wildcard: get '/files/*' do; path = params['splat'].first; end. Query strings are also in params: GET /search?q=ruby gives params[:q] == 'ruby'. Route blocks have access to request, response, and session.

Q3. How does Sinatra handle request data (body, headers, forms)?

A: request.body.read returns the raw request body. request.env['HTTP_AUTHORIZATION'] reads a header. params includes both query string and form fields from application/x-www-form-urlencoded or multipart. For JSON: JSON.parse(request.body.read).

Q4. What template engines does Sinatra support and how do you render them?

A: Sinatra supports ERB (erb :view), Haml (haml :view), Slim, Mustache, and others. Install the corresponding gem, then call the helper with a symbol matching the file name in views/. Pass local variables: erb :user, locals: { user: @user }.

Q5. How do Sinatra before/after filters work?

A: before do; @user = User.find(session[:user_id]); end runs before every route. after do; log_response(response); end runs after. Filters accept a pattern: before '/admin/*' do; halt 401 unless @user&.admin?; end scopes to matching paths.

Q6. How do you define and use helper methods in Sinatra?

A: Helpers inside a helpers do block or a module included via helpers MyHelpers become available in both route blocks and templates. def link_to(text, href); "#{h(text)}"; end. h() is the built-in HTML escape helper.

Q7. How do you handle errors and define custom error pages?

A: not_found do; 'Page not found'; end handles 404. error 500 do; 'Server error'; end handles 500. error MyCustomException do; status 400; 'Bad input'; end catches a specific exception class. error do catches any unhandled exception.

Q8. What is the difference between classic-style and modular Sinatra?

A: Classic style (require 'sinatra') defines routes at the top level simple but pollutes the global namespace. Modular style (class MyApp < Sinatra::Base) is a reusable class you mount in a Rack config.ru. Modular is preferred for tests and embedding in larger apps.

Q9. How do you manage sessions and cookies in Sinatra?

A: Enable sessions with enable :sessions (uses Rack::Session::Cookie with a random secret by default). Set a secret with set :session_secret, 'long-random-string'. session[:user_id] = user.id stores data. Cookies: response.set_cookie('name', value: 'x', httponly: true).

Q10. How do you test a Sinatra application with Rack::Test?

A: Include Rack::Test::Methods and define app { Sinatra::Application } (or MyApp). Call get '/path', headers: {'Authorization'=>'Bearer token'}. Assert last_response.status == 200 and last_response.body. set :environment, :test to suppress output.