Welcome!

WWCode San Francisco - Backend Study Group

July 6, 2023

- We'll start in a moment :)
- We are **RECORDING** tonight's event
- We may plan to take screenshots for social media
- If you are comfortable, turn the video ON. If you want to be anonymous, then turn the video off
- We'll make some time for Q&A at the end of the presentation
- Feel free to take notes
- Online event best practices:
    - Don't multitask. Distractions reduce your ability to remember concepts
    - Mute yourself when you aren't talking
    - We want the session to be interactive
    - Use the 'Raise Hand' feature to ask questions
- **By attending our events, you agree to comply with our Code of Conduct**

WOMEN WHO CODE /san-francisco

Introduction & Agenda

- Welcome from WWCode!
- Our mission: Empower women to excel in technology careers
- Backend Study Group: Learn and discuss backend engineering concepts

- Fundamentals of Ruby
- CRUD operations
- Your first Rails Application!
- Q & A

Prachi Shah
Instructor
Sr. Software Engineer, Unity
Director, WWCode SF

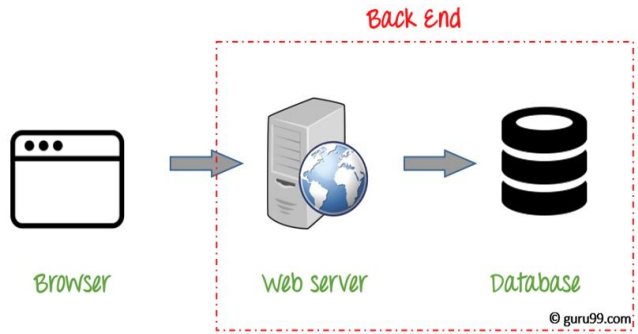
Harini Rajendran
Software Engineer, Confluent
Lead, WWCode SF

WOMEN WHO CODE /san-francisco

Backend Engineering

- Design, build and maintain server-side web applications

- Concepts: Client-server architecture, networking, APIs, web fundamentals, microservices, databases, security, operating systems, etc.



- Tech Stack: Java, PHP, .NET, C#, Ruby, Python, REST, AWS, Node, SQL, NoSQL, etc.

Ruby

- Ruby is an object-oriented scripting language known for its simplicity, flexibility, and elegant syntax. It was created by Yukihiro Matsumoto, also known as "Matz," in the mid-1990s.
- Ruby is designed to prioritize developer productivity and happiness, promoting readable and expressive code. It supports multiple programming paradigms, including procedural, functional, and object-oriented programming.
- Ruby has a rich standard library and a vast ecosystem of community-developed libraries and frameworks.

Rails

- Ruby on Rails, often referred to as Rails, is a web application framework written in Ruby. It was created by David Heinemeier Hansson (DHH) while working on the Basecamp project in 2003.
- Rails follows the Model-View-Controller (MVC) architectural pattern, which promotes separation of concerns and code organization.
- Rails emphasizes convention over configuration, providing a set of conventions and sensible defaults, which allows developers to focus on building the application's unique features instead of repetitive tasks.
- Rails provides a range of features, including an Object-Relational Mapping (ORM) called ActiveRecord for database interactions, a routing system, view templating using Embedded Ruby (ERB), and built-in support for handling common web development tasks such as form handling, session management, and security.

Ruby on Rails

- Together, Ruby and Rails offer a powerful combination for building web applications. Ruby provides a flexible and expressive language, while Rails simplifies the development process by providing a framework that promotes best practices, encourages rapid development, and minimizes repetitive coding tasks.
- The combination of Ruby's elegance and Rails' productivity has contributed to the popularity and success of the Ruby on Rails ecosystem.
- Prerequisites:
  - Ruby and Rails installed on your machine. Visit https://www.ruby-lang.org/en/
  - Basic understanding of Ruby programming.
  - Familiarity with HTML, CSS, and JavaScript.
  - Knowledge of databases and SQL concepts.

Development Setup

- Install Ruby and Rails.
- Set up a development environment on your machine.
- Install Ruby and RubyGems package manager.
- Install Rails using the Gem package manager.

Rails Application

- Generate a new Rails application.
- Use the Rails command-line tool to create a new project.
- Open your terminal and run the following command:

  ```
  $ rails new myapp
  ```
- MVC Architecture: Model-View-Controller
  - Models represent the data and business logic.
  - Views handle the presentation and user interface.
  - Controllers handle the application's logic and interact with models and views.
- Create a model and database table:
  - Define your application's data model using Ruby classes.
  - Use migrations to generate and modify database tables.

Rails Application

- Generate a new model named "Product" with attributes:

  ```
  $ rails generate model Product name:string price:decimal
  ```

- This generates a migration file to create the "products" table in the database.

- Run database migrations.
  - Apply the pending migrations to create the database table:

    ```
    $ rails db:migrate
    ```

  - This sets up the "products" table in the database based on the migration file.
- Creating Views:
  - Create views to display data.
  - Generate a view file for displaying products:

    ```
    $ rails generate controller Products index
    ```

  - This creates an "index" action in the "ProductsController" and the corresponding view file.

Rails Application

- Adding Controller Logic: Implement controller actions.
  - Open the "app/controllers/products_controller.rb" file.
  - Add the following code to the "index" action:

```
def index
    @products = Product.all
end
```

  - This fetches all products from the database and assigns them to the instance variable "@products."
  - Implement controller actions to handle requests.
- Define actions to process user input and interact with models.
- Respond with appropriate data or render views.

Rails Application

- Rendering Views: Render views in the controller.
  - Open the "app/views/products/index.html.erb" file.
  - Add the following code to display the products:

```
<h1>Products</h1>
<ul>
    <% @products.each do |product| %>
        <li><%= product.name %> - <%= product.price %></li>
    <% end %>
</ul>
```

  - This uses embedded Ruby (ERB) tags to embed Ruby code and display product information.

Rails Application

- Routing: Define routes for the application.
  - Set up routes in the config/routes.rb file. Open the file.
  - Add the following line to create a route for the products index page:

    ```
    $ get '/products', to
    ```
  - This maps the URL "/products" to the "index" action in the "ProductsController."
  - Define routes to map URLs to controller actions.
  - Set up routes in the config/routes.rb file.
  - Understand RESTful routing conventions for CRUD operations.
- Launching the Application:
  - Start the Rails server. In your terminal, navigate to the project directory and run:

    ```
    $ rails server
    ```
  - This starts the Rails server, allowing you to access the application in your web browser.

Fundamentals

- Ruby: An object-oriented scripting language.
- Simple syntax and elegant code readability.
- Key concepts:
  - Variables and data types (strings, numbers, arrays, hashes).
  - Control structures (conditionals, loops).
  - Methods and classes.
  - Object-oriented programming principles:
    - All values are objects, and there is no distinction between primitive types and object types.
    - All objects inherit from a class named *Object* and share the methods defined by that class.

Data Types

- Ruby is dynamically typed, so you don't need to declare variable types explicitly.
- You can assign values to variables using the assignment operator (=).
- Examples:

```
# Variables
name = "Stacey"
age = 30
is_employed = true

# Data Types
puts name.class        # String
puts age.class             # Integer
puts is_employed.class     # Boolean
```

Data Types

- Arrays are ordered lists, and hashes are key-value pairs in Ruby.
- Examples:

```
# Arrays
fruits = ["apple", "banana", "orange", "mango"]
puts fruits[2] # "orange"
puts fruits.length # 4

# Hashes
person = {"name" => "Joe", "age" => 30, "is_student" => false}
puts person["name"]# "Joe"
puts person.keys       # ["name", "age", "is_student"]
```

Objects and Classes

- Ruby is an object-oriented language, allowing you to define classes and create objects from those classes.
- Example:

```
#!/usr/bin/ruby
class Customer
    def initialize(id, name)
        @customer_id = id
        @customer_name = name
    end
    def display_customer_info()
        puts "Customer id #@customer_id"
        puts "Customer name #@customer_name"
    end
end
```

Methods

- Methods allow you to encapsulate code into reusable blocks.
- Example:

```
def greet(first_name)
    puts "Hello, #{first_name}!"
end

greet("John")  # "Hello, John!"
greet("Stacey")    # "Hello, Stacey!"
```

Control Structures

- Ruby provides various control structures, such as if statements and loops, to control the flow of your program.
- Examples:

```ruby
# If statement
x = 5
if x > 0
  puts "x is positive"
elsif x < 0
  puts "x is negative"
else
  puts "x is zero"
end
```

```ruby
# Loop - While
i = 0
while i < 5
  puts i
  i += 1
end

# Loop - Times
5.times do |i|
  puts i
end
```

CRUD Operations

- CRUD: Create, Read, Update, Delete.
- Essential operations for interacting with databases.
- Operations:
  - Creating a new record.
  - Reading data from the database.
  - Updating existing records.
  - Deleting records.

CRUD Operations

- Creating a new record:
  - To create a new record in a database, you typically need an ORM (Object-Relational Mapping) library like ActiveRecord.

```ruby
require 'active_record'

ActiveRecord::Base.establish_connection( # Connect to the database
  adapter: 'postgresql',
  host: 'localhost',
  database: 'your_database_name',
  username: 'your_username',
  password: 'your_password'
)

# Define a model representing a table in the database
class User < ActiveRecord::Base
end

# Create a new user record
user = User.new(name: 'John Doe', age: 30, email: 'john@example.com')
user.save
```

CRUD Operations

- Reading data from the database:
  - To read data from the database, you can use ActiveRecord's query methods.

```ruby
# Find a specific user by their ID
user = User.find(1)
puts user.name

# Find all users
users = User.all
users.each do |user|
  puts user.name
end

# Find users matching specific conditions
users = User.where(age: 25)
users.each do |user|
  puts user.name
end
```

CRUD Operations

- Updating existing records:
  - To update an existing record, you can retrieve it from the database, modify its attributes, and save the changes.

```
# Find a user by their ID
user = User.find(1)
user.name = 'Updated Name'
user.save
```

- Deleting records:
  - To delete a record, you can find it and call the destroy method.

```
# Find a user by their ID and delete it
user = User.find(1)
user.destroy
```

Testing

- Testing Ruby code is typically done using a testing framework like RSpec, MiniTest, or Test::Unit.
- These frameworks provide a structured way to write and execute tests for your Ruby code.
- Example using RSpec:
  - Install the RSpec gem by running the following command in your terminal:

    ```
    $ gem install rspec
    ```

Testing

- Create a new test file. Conventionally, test files are named with the _spec.rb suffix.

```ruby
# controller_spec.rb

require 'rspec'
require_relative 'controller'  # Assuming your code is in a file named controller.rb

describe Controller do
  context 'when given valid input' do
    it 'returns the correct result' do
      # Create an instance of your class or module
      controller = Controller.new

      # Perform the operation you want to test
      result = controller.calculate_multiplication(5)

      # Make assertions to validate the result
      expect(result).to eq(10)
    end
  end
end
```

Testing

- Create the corresponding Ruby file that contains the code you want to test.

```ruby
# controller.rb

class Controller
  def calculate_multiplication(x)
    x * 2
  end
end
```

- Run the test using the rspec command in your terminal, passing the test file as an argument.

```
$ rspec controller_spec.rb
```

Shell

- The Interactive Ruby Shell, also known as IRB, is a different type of shell.
- IRB is a program that actively waits for user input, requiring us to enter code and press "Enter."
- IRB is designed for interacting with Ruby code.
- Example:

```
$ irb
    > puts "Good Morning!"
    Good Morning!
    => nil
    > exit ↩
```

- Shell also prints out the return value for the *puts* statement, which is *nil*. The rocket (=>) indicates that a value is being returned by the executed statement.

Runtime

- Run Ruby code in a shell.

```
$ ruby goodmorning.rb
```

- The Ruby runtime is instructed to interpret and execute the Ruby code within the goodmorning.rb file.
- Print the version of Ruby:

```
$ ruby --version
ruby 3.0.0p0 (2020-12-25 revision 95aff21468)
```

- Execute Ruby code without file storage:

```
$ ruby -e 'puts 101010'
101010
$ ruby -e 'puts "Good morning, San Francisco!"'
Good morning, San Francisco!
```

Backend Study Group

Backend Study Group:

- Presentations on GitHub and session recordings available on WWCode YouTube channel
- August 3rd, 2023: Domain Driver Design (DDD)
- September 21st, 2023: Web Development 101
- October 12th, 2023: DevOps 101
- November 2nd, 2023: How to be a good Backend Engineer?

Women Who Code:

- Technical Tracks and Digital Events for more events
- Join the Digital mailing list for updates about WWCode
- Contacts us at: contact@womenwhocode.com
- Join our Slack workspace and join *#backend-study-group*!

You can unmute and talk or use the chat

WOMEN WHO CODE /san-francisco