Rails Basics

Sang Shin
JPassion.com
“Learn with Passion!”

Topics

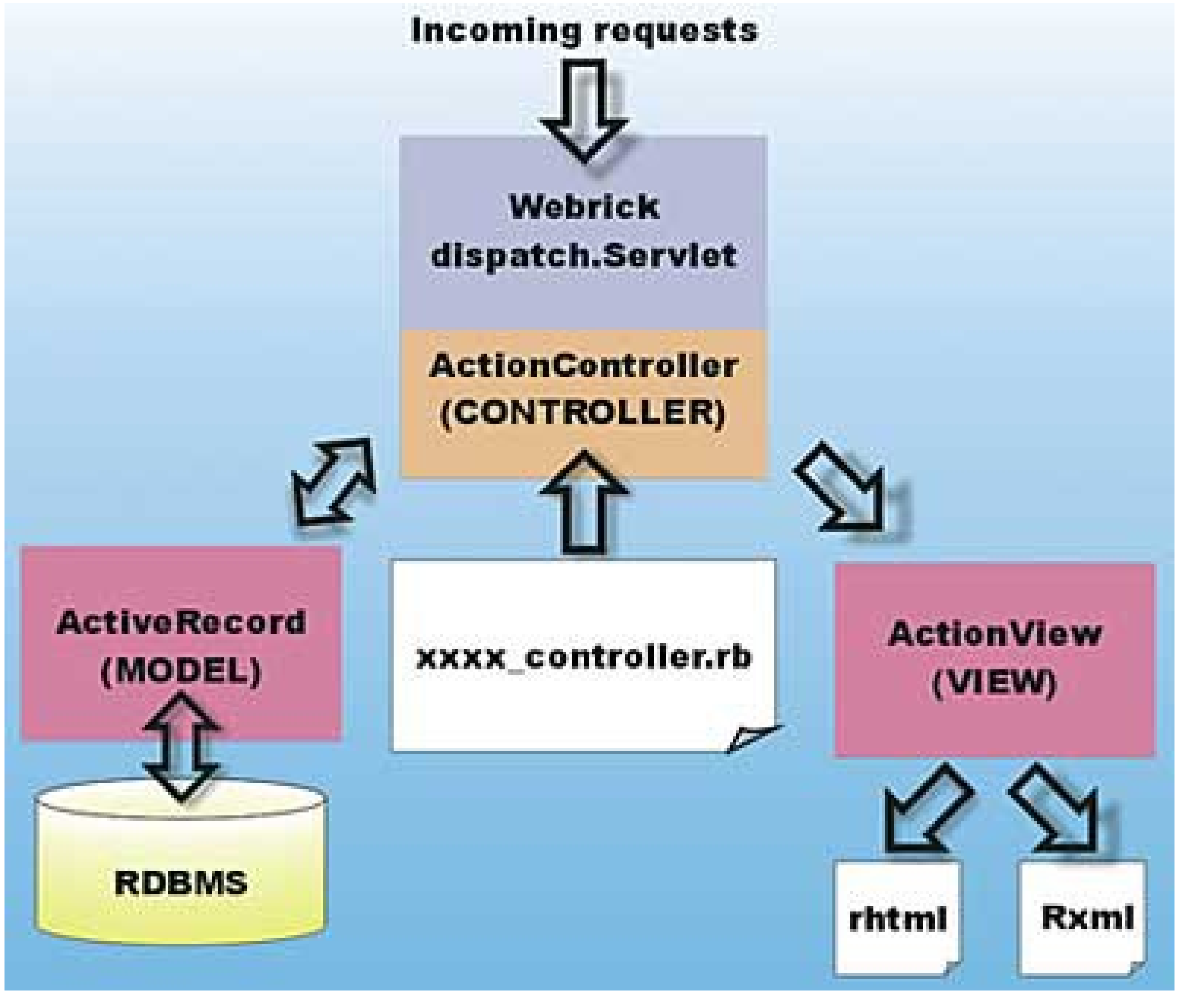
- What is and Why Ruby on Rails?
- Building HelloWorld Rails application step by step
- Key concepts of Rails application development
  - App directory structure (MVC), Environment
  - Rake, Generator, Migration, Rails console
  - Bundle (will be covered in another presentation)
- Add another field to a model
- Add another template (view page)
- Create a form for user input
- Explore Rails learning resources

What is and Why Ruby on Rails (RoR)?

What Is “Ruby on Rails”?

- A full-stack MVC web development framework
- Written in Ruby
  - Rails leverages various characteristics of Ruby language - meta-programming, closure, etc.
- First released in 2004 by David Heinemeier Hansson (DHH)

“Ruby on Rails” MVC



source: http://www.ilug-cal.org

“Ruby on Rails” Principles

- Don’t Repeat Yourself (DRY)
- Convention over configuration
  - Default over configuration
- Agile development environment

Don't Repeat Yourself (DRY)

- What is it?
  - Every piece of knowledge must have a single, unambiguous, authoritative representation within a system
  - Aimed at reducing repetition/duplication of information of all kinds
- Benefits
  - When the DRY principle is applied successfully, a modification of any single element of a system does not change other logically-unrelated elements.
  - Additionally, elements that are logically related all change predictably and uniformly, and are thus kept in sync.

Convention Over Configuration

- What is it?
  - Predefined default values
  - Predefined directory structure
  - Predefined naming conventions
- Benefits
  - Higher developer productivity
  - No need to write configuration files
  - Less coding
  - Easier maintenance
  - Less number of bugs
  - Leverage best practice

Agile Development Environment

- What is it?
  - No recompile, deploy, restart cycles
  - Simple tools to generate code quickly
  - Testing built into the framework
- Benefits
  - Higher developer productivity
  - Higher maintainability

Lab:

Exercise 1: Install Rails 4 (If it was not installed already)

5521_rails_basics.zip

Building “Hello World” Rails Application Step by Step

Steps to Follow

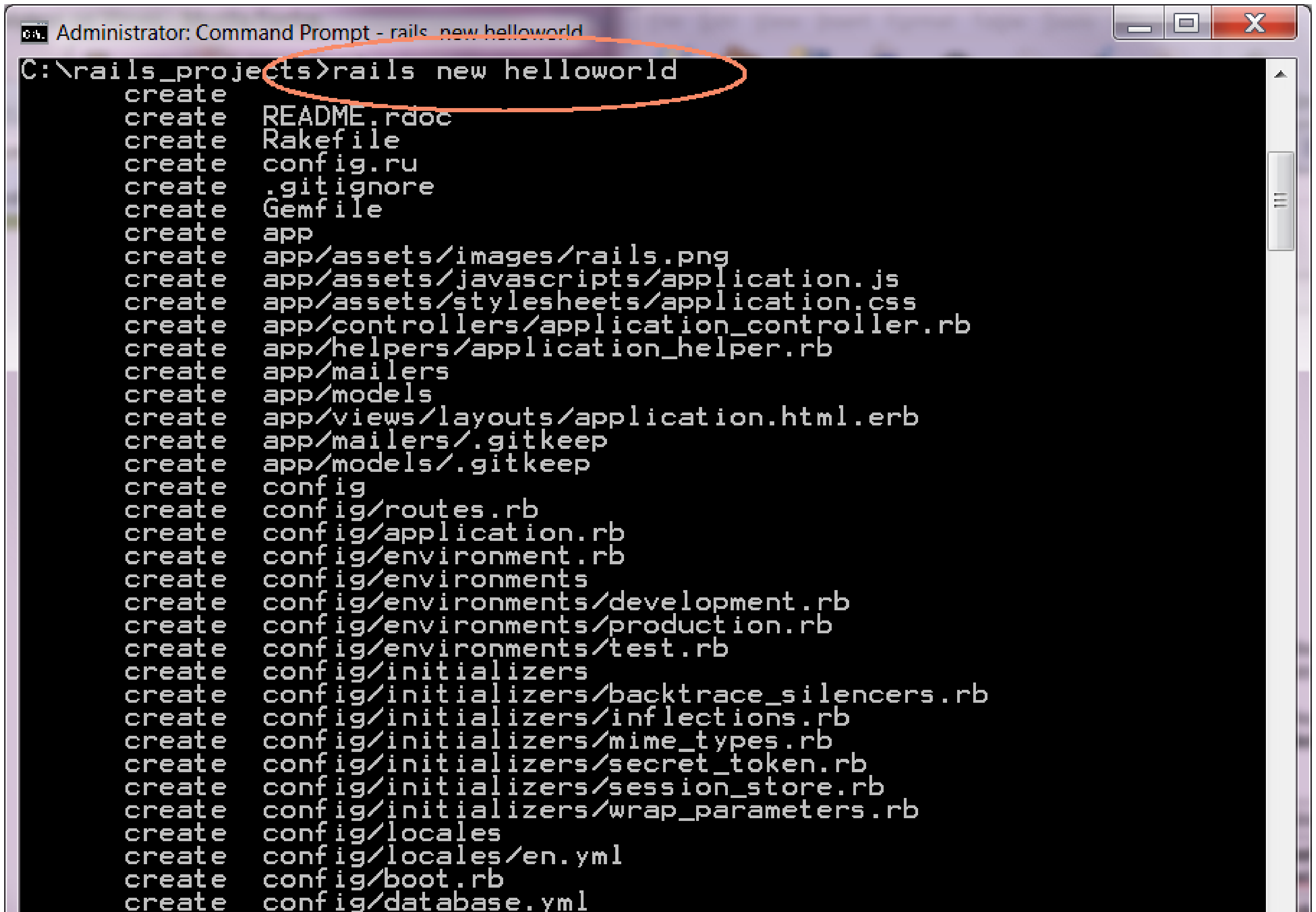
1. Create “Ruby on Rails” project
   > Rails generates directory structure
2. Create Database (using Rake)
3. Create Models (using Generator)
4. Create Database Tables (using Migration)
5. Create Controllers (using Generator)
6. Create Views
7. Set URL Routing

You, as a Rails developer, spend most of your time, writing Model,View, Controller code.

1. Create “Ruby on Rails” Project

1. Create “Ruby on Rails” Project

- Run “rails new <app-name>” at the command-line



```
C:\rails_projects>rails new helloworld
      create
      create  README.rdoc
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      create  app/assets/images/rails.png
      create  app/assets/javascripts/application.js
      create  app/assets/stylesheets/application.css
      create  app/controllers/application_controller.rb
      create  app/helpers/application_helper.rb
      create  app/mailers
      create  app/models
      create  app/views/layouts/application.html.erb
      create  app/mailers/.gitkeep
      create  app/models/.gitkeep
      create  config
      create  config/routes.rb
      create  config/application.rb
      create  config/environment.rb
      create  config/environments
      create  config/environments/development.rb
      create  config/environments/production.rb
      create  config/environments/test.rb
      create  config/initializers
      create  config/initializers/backtrace_silencers.rb
      create  config/initializers/inflections.rb
      create  config/initializers/mime_types.rb
      create  config/initializers/secret_token.rb
      create  config/initializers/session_store.rb
      create  config/initializers/wrap_parameters.rb
      create  config/locales
      create  config/locales/en.yml
      create  config/boot.rb
      create  config/database.yml
```

Lab:

Exercise 2: Build Helloworld App
5521_rails_basics.zip

Directory Structure of a Rails Application

- When you ask Rails to create a Rails project with “rails new <app-name>”, it creates the entire directory structure for the application under <app-name> directory
  - The boiler plate files are also created
  - The names of the directories and files are the same for all Rails projects
- Rails knows where to find things it needs within this structure, so you don't have to tell it explicitly

Directory Structure of a Rails Application

- *app*: Holds all the code that's specific to this particular application according to MVC
  - *app/models*: Holds models
  - *app/views*:
    - *app/views/layouts*: Holds the template files for layouts to be used with views.
    - app/views/<controller>: Holds view files of the <controller>
  - *app/controllers*: Holds controllers
  - *app/assets:* Holds assets (images, Javascript files, stylesheets)

Directory Structure of a Rails Application

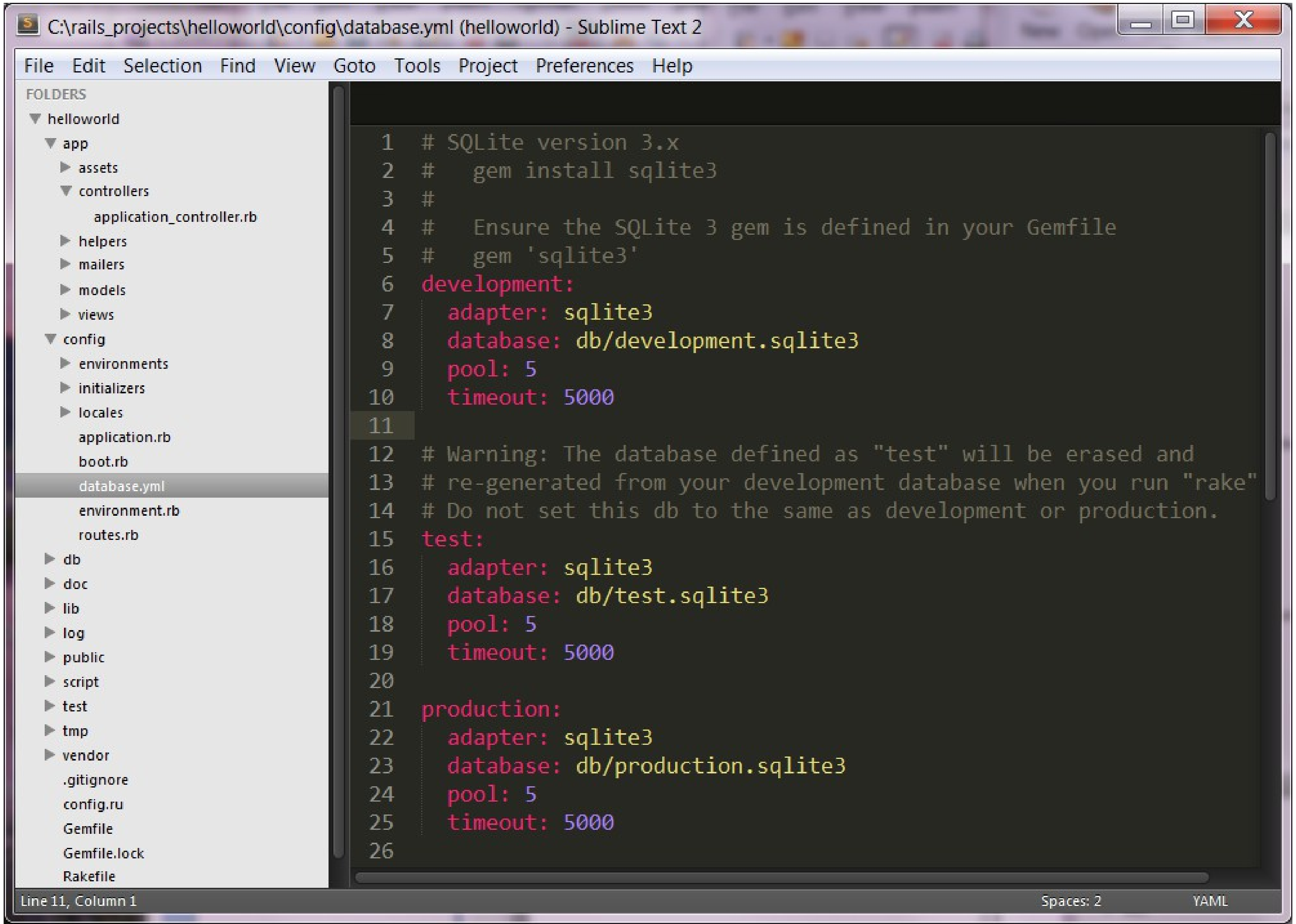
- *config:* Holds configuration files for the Rails environment, the routing map, the database, and other dependencies.
  - > *config/environments*
  - > *config/initializers*
  - > *config/locales*
  - > *Application.rb*
  - > *boot.rb*
  - > *database.yml*
  - > *environment.rb*
  - > *routes.rb*

Learning Point:
Environments

What is an Environment?

- Rails provides the concept of environments
  - *development, test, production*
- As a default, different database is going to be used for different environment.
  - Therefore each environment has its own database connection settings
- It is easy to add custom environments
  - For example, “staging” environment
- Rails always runs in only one environment
  - “Rails.env” shows current environment

config/database.yml



```
# SQLite version 3.x
#   gem install sqlite3
#
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem 'sqlite3'
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake"
# Do not set this db to the same as development or production.
test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000

production:
  adapter: sqlite3
  database: db/production.sqlite3
  pool: 5
  timeout: 5000

```

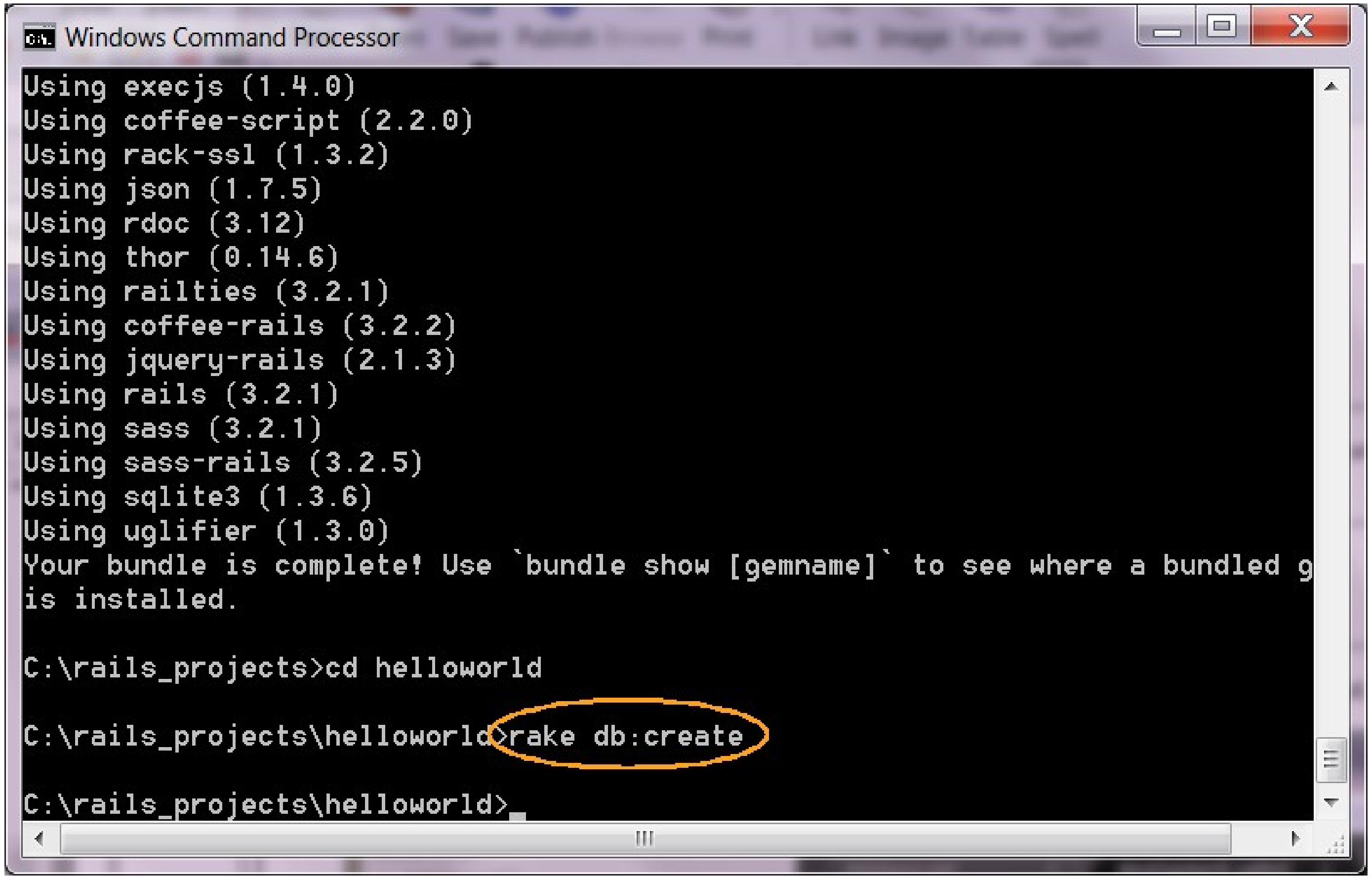
Lab:

Exercise 3.1: Study Directory Structure
5521_rails_basics.zip

2. Create Database using “Rake”

Create Database

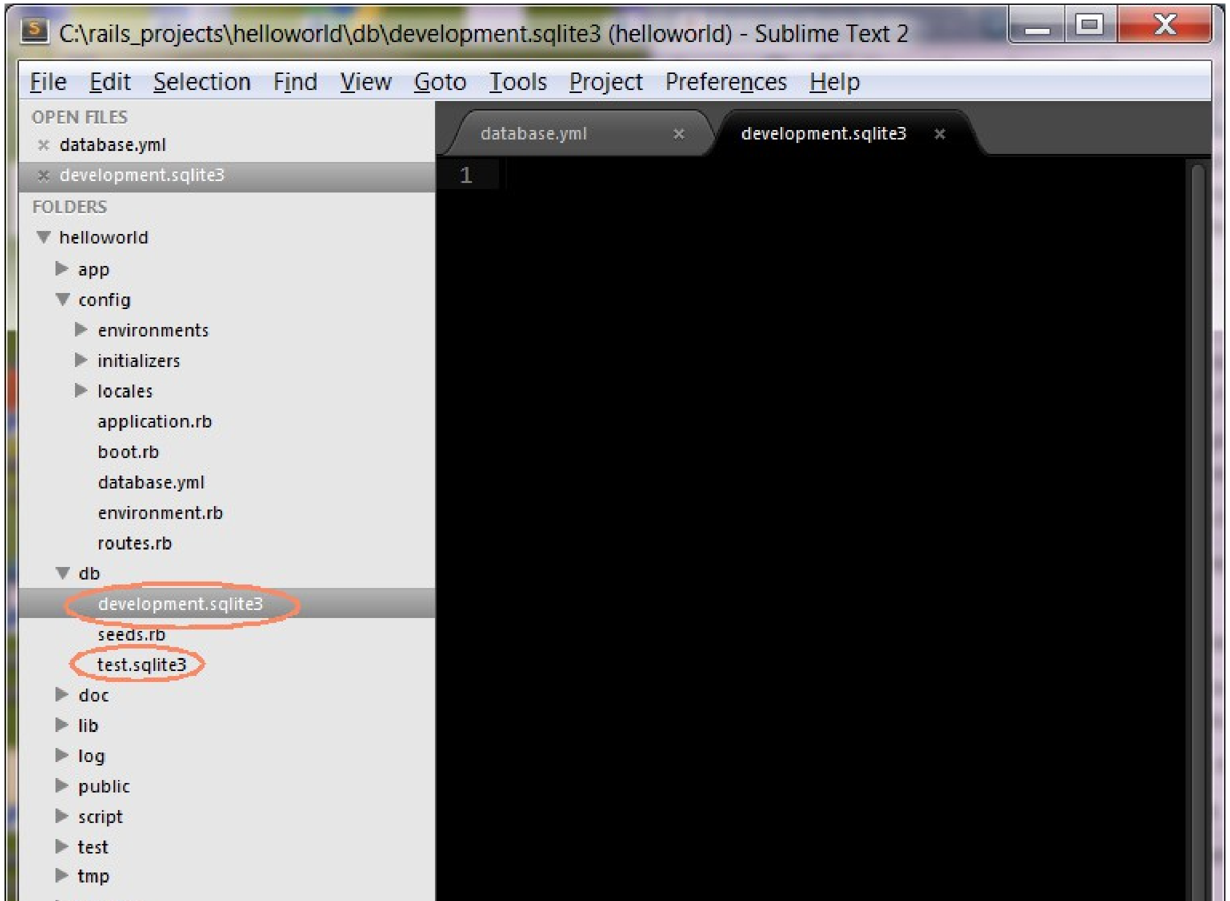
- Use "*rake db:create*" to create database – actually this is no longer needed since later version of Rails create the development database when creating a project



```
Using execjs (1.4.0)
Using coffee-script (2.2.0)
Using rack-ssl (1.3.2)
Using json (1.7.5)
Using rdoc (3.12)
Using thor (0.14.6)
Using railties (3.2.1)
Using coffee-rails (3.2.2)
Using jquery-rails (2.1.3)
Using rails (3.2.1)
Using sass (3.2.1)
Using sass-rails (3.2.5)
Using sqlite3 (1.3.6)
Using uglifier (1.3.0)
Your bundle is complete! Use `bundle show [gemname]` to see where a bundled g
is installed.

C:\rails_projects>cd helloworld

C:\rails_projects\helloworld>rake db:create

C:\rails_projects\helloworld>_
```

Databases Created

- After “rake db:create”, development and test databases are created

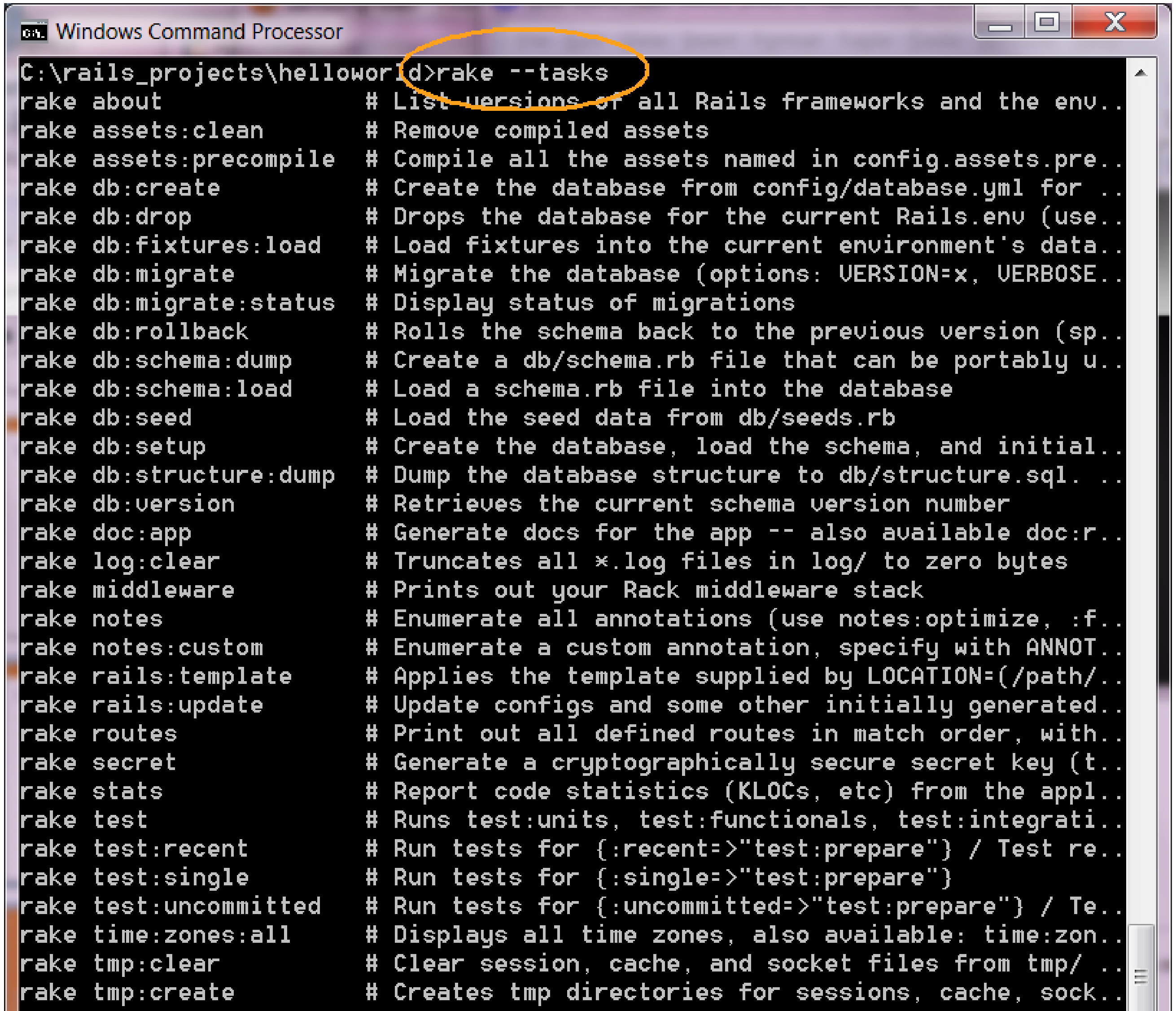


Learning Point:
What is Rake?

What is “Rake”?

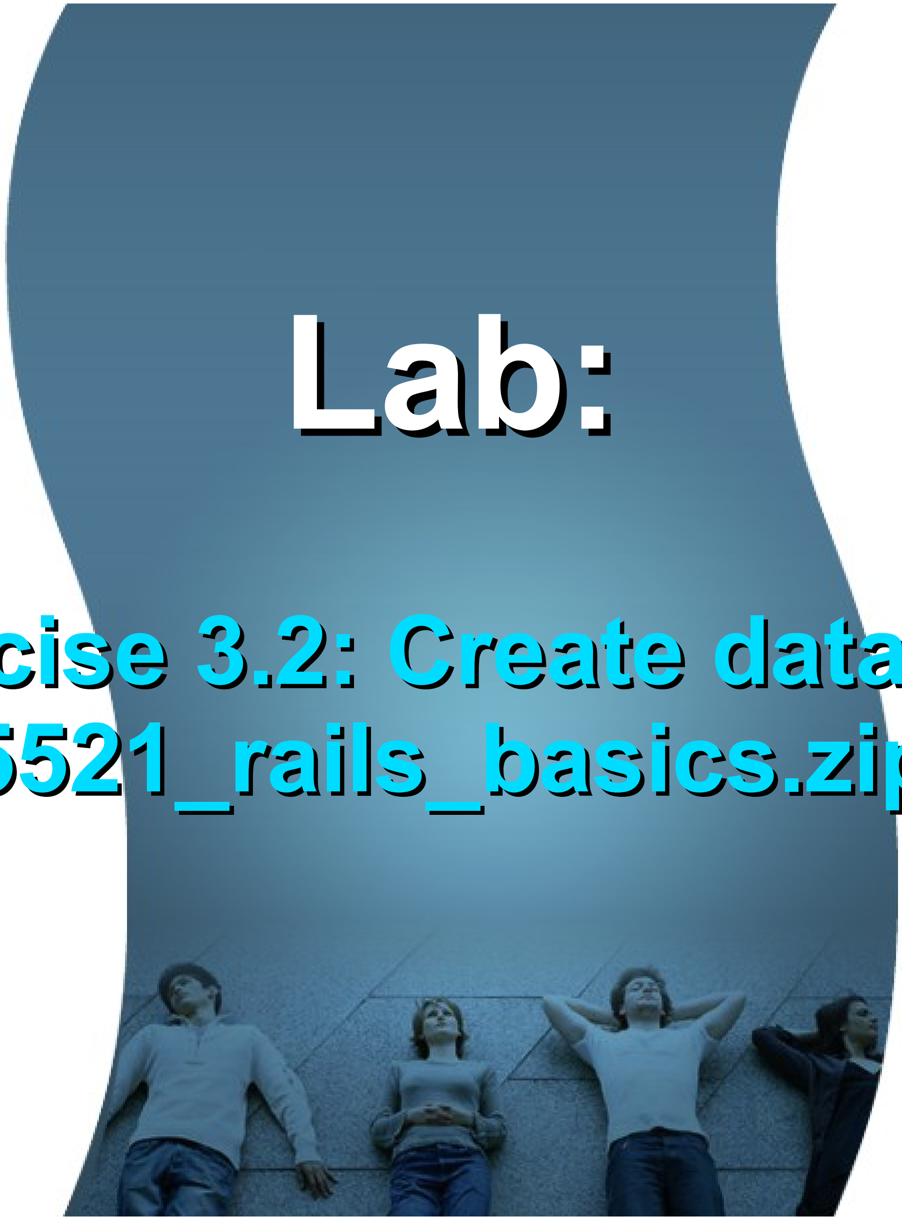
- Rake is a build language for Ruby
  - It is like “make” utility
- Rails uses Rake to automate several tasks such as
  - creating and dropping databases
  - running tests
  - updating Rails support files
- Rake lets you define a dependency tree of tasks to be executed

Listing Rake Tasks (rake –tasks, rake -T)



Windows Command Processor

```
C:\rails_projects\helloworld>rake --tasks
rake about                # List versions of all Rails frameworks and the env..
rake assets:clean         # Remove compiled assets
rake assets:precompile    # Compile all the assets named in config.assets.pre..
rake db:create            # Create the database from config/database.yml for ..
rake db:drop              # Drops the database for the current Rails.env (use..
rake db:fixtures:load     # Load fixtures into the current environment's data..
rake db:migrate           # Migrate the database (options: VERSION=x, VERBOSE..
rake db:migrate:status    # Display status of migrations
rake db:rollback          # Rolls the schema back to the previous version (sp..
rake db:schema:dump       # Create a db/schema.rb file that can be portably u..
rake db:schema:load       # Load a schema.rb file into the database
rake db:seed              # Load the seed data from db/seeds.rb
rake db:setup             # Create the database, load the schema, and initial..
rake db:structure:dump    # Dump the database structure to db/structure.sql. ..
rake db:version           # Retrieves the current schema version number
rake doc:app              # Generate docs for the app -- also available doc:r..
rake log:clear            # Truncates all *.log files in log/ to zero bytes
rake middleware           # Prints out your Rack middleware stack
rake notes                # Enumerate all annotations (use notes:optimize, :f..
rake notes:custom         # Enumerate a custom annotation, specify with ANNOT..
rake rails:template       # Applies the template supplied by LOCATION=(/path/..
rake rails:update         # Update configs and some other initially generated..
rake routes               # Print out all defined routes in match order, with..
rake secret               # Generate a cryptographically secure secret key (t..
rake stats                # Report code statistics (KLOCs, etc) from the appl..
rake test                 # Runs test:units, test:functionals, test:integrati..
rake test:recent          # Run tests for {:recent=>"test:prepare"} / Test re..
rake test:single          # Run tests for {:single=>"test:prepare"}
rake test:uncommitted     # Run tests for {:uncommitted=>"test:prepare"} / Te..
rake time:zones:all       # Displays all time zones, also available: time:zon..
rake tmp:clear            # Clear session, cache, and socket files from tmp/ ..
rake tmp:create           # Creates tmp directories for sessions, cache, sock..
```



Lab:

Exercise 3.2: Create database
5521_rails_basics.zip

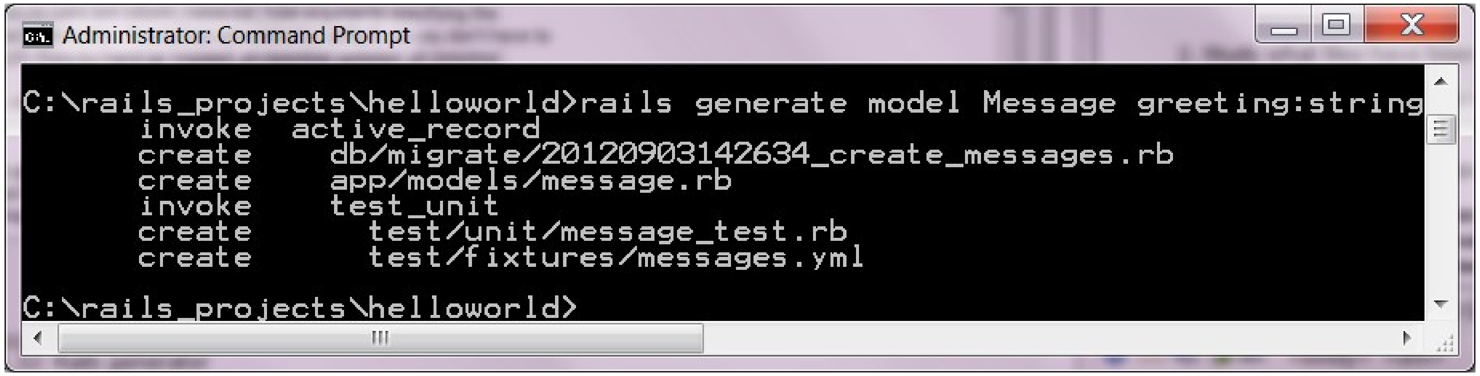
3. Create a Model through “Generator”

What is a Model?

- In the context of MVC pattern, a Model represents domain objects such as message, school, product, etc.
- A model has attributes and methods.
  - The attributes represents the characteristics of the domain object, for example, a “Message” model might have “length”, “creator” as attributes.
  - The methods in a model typically contains simple logic manipulating the attributes
- Most models have corresponding database tables
  - For example, a *message* model will have *messages* table.
- Most model classes are *ActiveRecord* type

Create a Model using Generator

- *rails generate model <name-of-model> <attribute1name:type> <attribute2name:type> ..*
- *rails generate model Message greeting:string sender:string*


Administrator: Command Prompt

```
C:\rails_projects\helloworld>rails generate model Message greeting:string
      invoke  active_record
      create    db/migrate/20120903142634_create_messages.rb
      create    app/models/message.rb
      invoke    test_unit
      create      test/unit/message_test.rb
      create      test/fixtures/messages.yml

C:\rails_projects\helloworld>
```

Files That Are Created

- db/migrate/20140719001000_create_messages.rb
  - A migration file for defining the initial structure of the database.
- app/models/message.rb (Model file)
  - A file that holds the methods for the Message model.
- test/unit/message_test.rb
  - A unit test for checking the Message model.
- test/fixtures/messages.yml
  - A test fixture for populating the model.

Model Class Example

- *Message* model in *message.rb* file

```
class Message < ActiveRecord::Base
end
```

Learning Point:
What is Generator?

What is “Generator”?

- You can often avoid writing boilerplate code by using the built-in generator scripts of Rails to create it for you.
  - This leaves you with more time to concentrate on the code that really matters--your business logic.

Learning Point:
What is Rails Console?

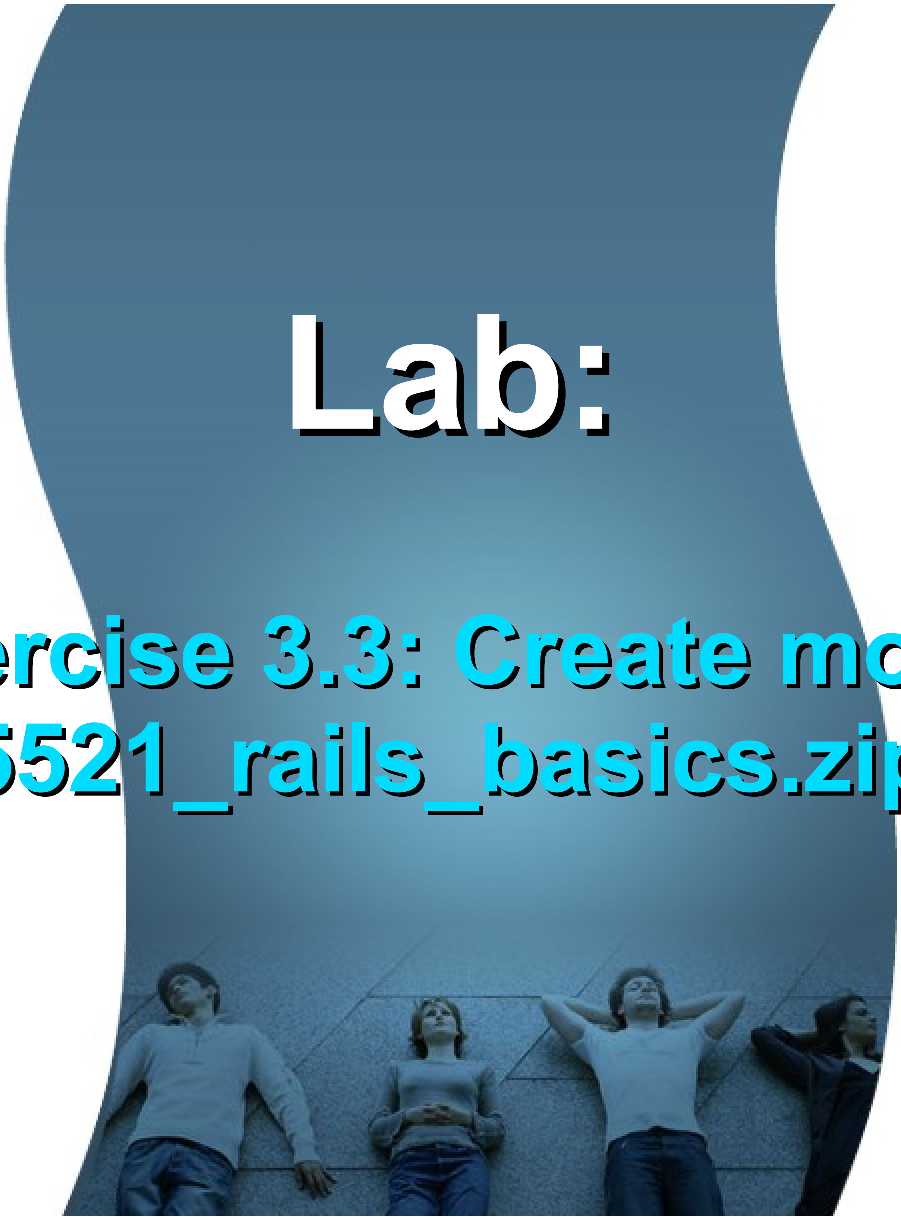
What is Rails Console?

- The Rails console gives you access to your Rails Environment, for example, you can interact with the domain models of your application as if the application is actually running.
  - Things you can do include performing find operations or creating a new active record object and then saving it to the database.
- A great tool for impromptu testing

Learning Point:
“rails” command

Rails Commands

- “rails console” or “rails c”: start Rails console
- “rails generate ..” or “rails g ...”: generate boiler plate code
- “rails server” or “rails s”: start Rails server with the application



Lab:

Exercise 3.3: Create model
5521_rails_basics.zip

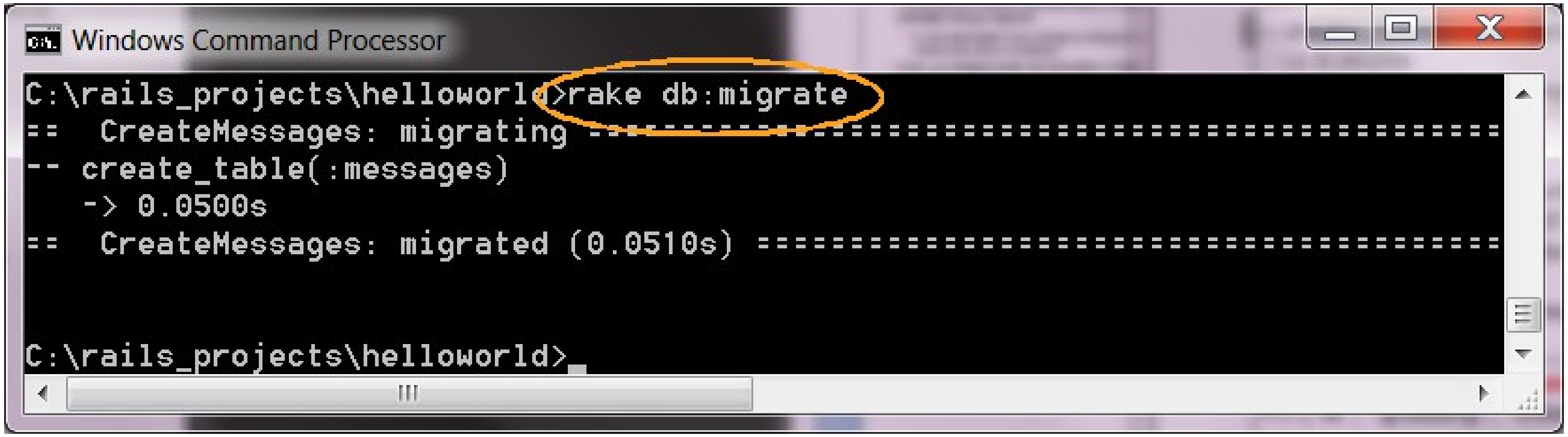
4. Create Database Tables using Migration

Create Database Table using Migration

- You are going to create a database table (in a previously created database) through migration
  - You also use migration for any change you are going to make in the schema - adding a new column, changing name of a column, for example
- When you create a Model, the first version of the migration file is automatically created
  - *db/migrate/20120719001000_create_messages.rb*, which defines initial structure of the table

```
class CreateMessages < ActiveRecord::Migration
  def change
    create_table :messages do |t|
      t.string :greeting
      t.timestamps
    end
  end
end
```

Performing Migration



```
C:\rails_projects\helloworld>rake db:migrate
==  CreateMessages: migrating =========================================
-- create_table(:messages)
   -> 0.0500s
==  CreateMessages: migrated (0.0510s) ================================


C:\rails_projects\helloworld>_
```

Learning point:
What is Migration?

Issues with Schema Changes

- Database schema's keep changing (along with applications that use them)
  - Example: You need to add “email” column to the “customer” table
- Issues with schema changes
  - How do you version control schema changes (just like you version control Ruby source code)?
  - How do you go back to previous version of the schema?
  - How do people work with different versions of the schema?
  - How do you convey schema changes to other developers and the deployer?

Migration To the Rescue

- Migration can manage the evolution of a schema
- With migrations, you can describe schema changes in self-contained Ruby classes called migration files
- You can check these migration files into a version control system
- Migration files are part of application source
- You (and others) can choose a schema version of choice, for example, several versions back from the current one

Lab:

Exercise 3.4: Create database tables using Migration

5521_rails_basics.zip

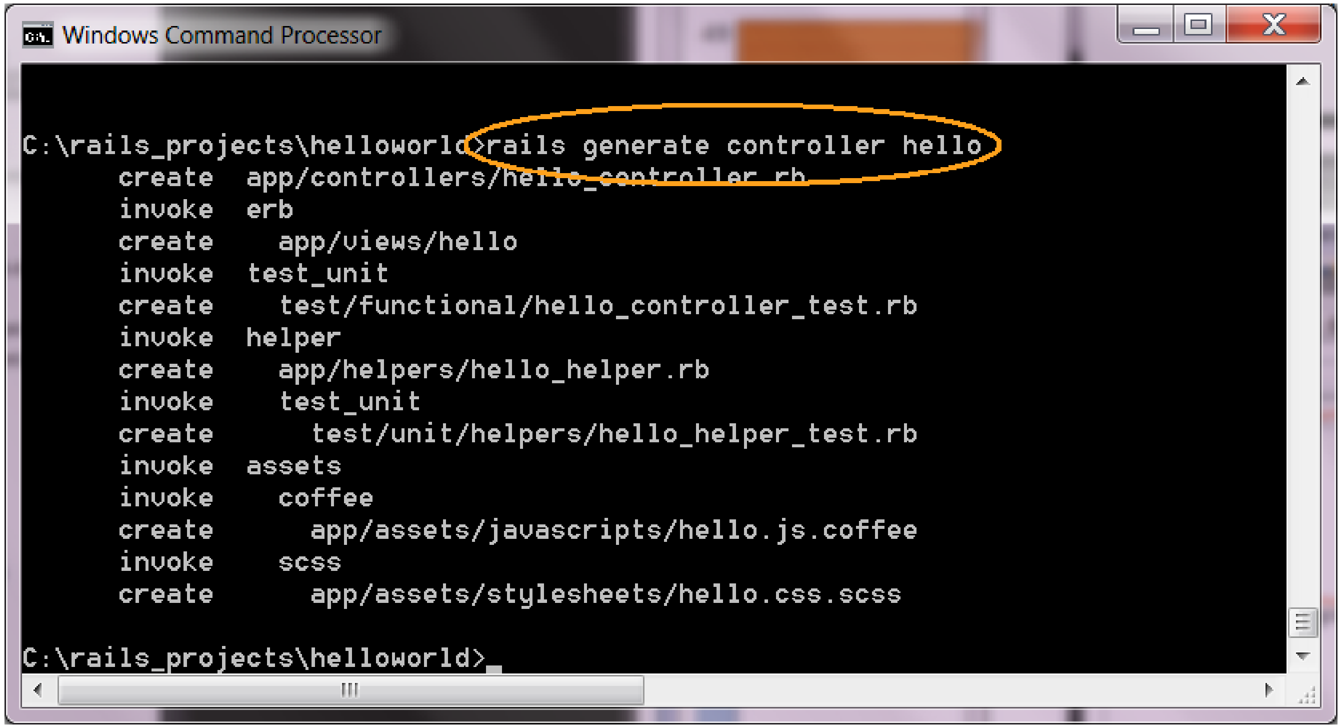
5. Create a Controller

What is a Controller?

- Action Controllers handle incoming Web requests
- A controller is made up of one or more actions
- Actions are executed to handle the incoming requests and then
  - Either render a template/view or
  - Redirect to another action
- An action is defined as a public method of a controller
- Mapping between a request's URL and an action is specified in the Rails routing map (/config/routes.rb)

Create a Controller using Generator

- *rails generate controller <Name-of-controller>*



```
C:\rails_projects\helloworld>rails generate controller hello
      create  app/controllers/hello_controller.rb
      invoke  erb
      create    app/views/hello
      invoke  test_unit
      create    test/functional/hello_controller_test.rb
      invoke  helper
      create    app/helpers/hello_helper.rb
      invoke    test_unit
      create      test/unit/helpers/hello_helper_test.rb
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/hello.js.coffee
      invoke    scss
      create      app/assets/stylesheets/hello.css.scss

C:\rails_projects\helloworld>_
```

Example: HelloController

- Controller contains actions, which are defined with *def*

```
class HelloController < ApplicationController

  def say_hello
    @hello = Message.new(:greeting => "Hello World!")
  end

end
```

an action

Lab:

Exercise 3.5: Create controller
5521_rails_basics.zip

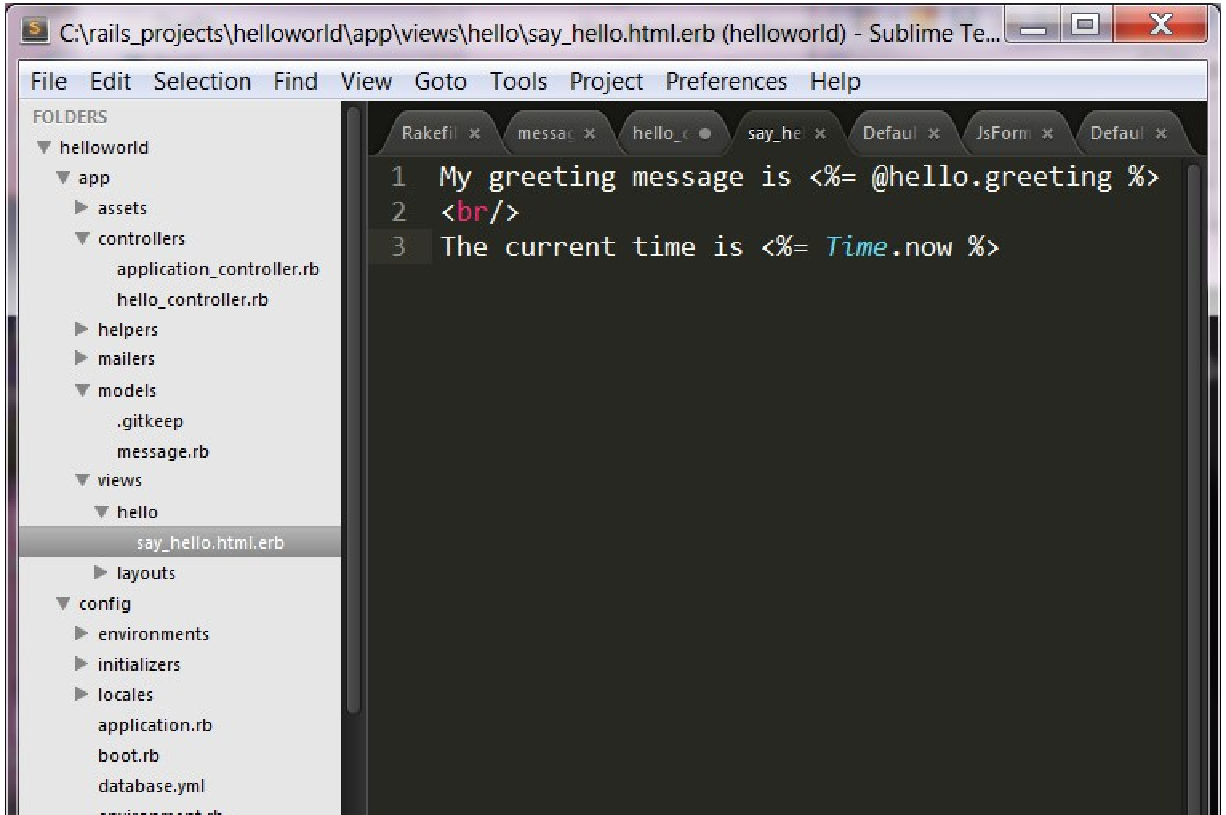
6. Write a View/Template

What is a Template (View)?

- Template represents what gets displayed
- A template shares data with a controller (actually an action within the controller) through mutually accessible variables
- A template is in the form of <action>.html.erb file

Write <action>.html.erb file

- <action>.html.erb has to be created under /app/views/<controller>/ directory - for example, /app/views/hello/say_hello.html.erb

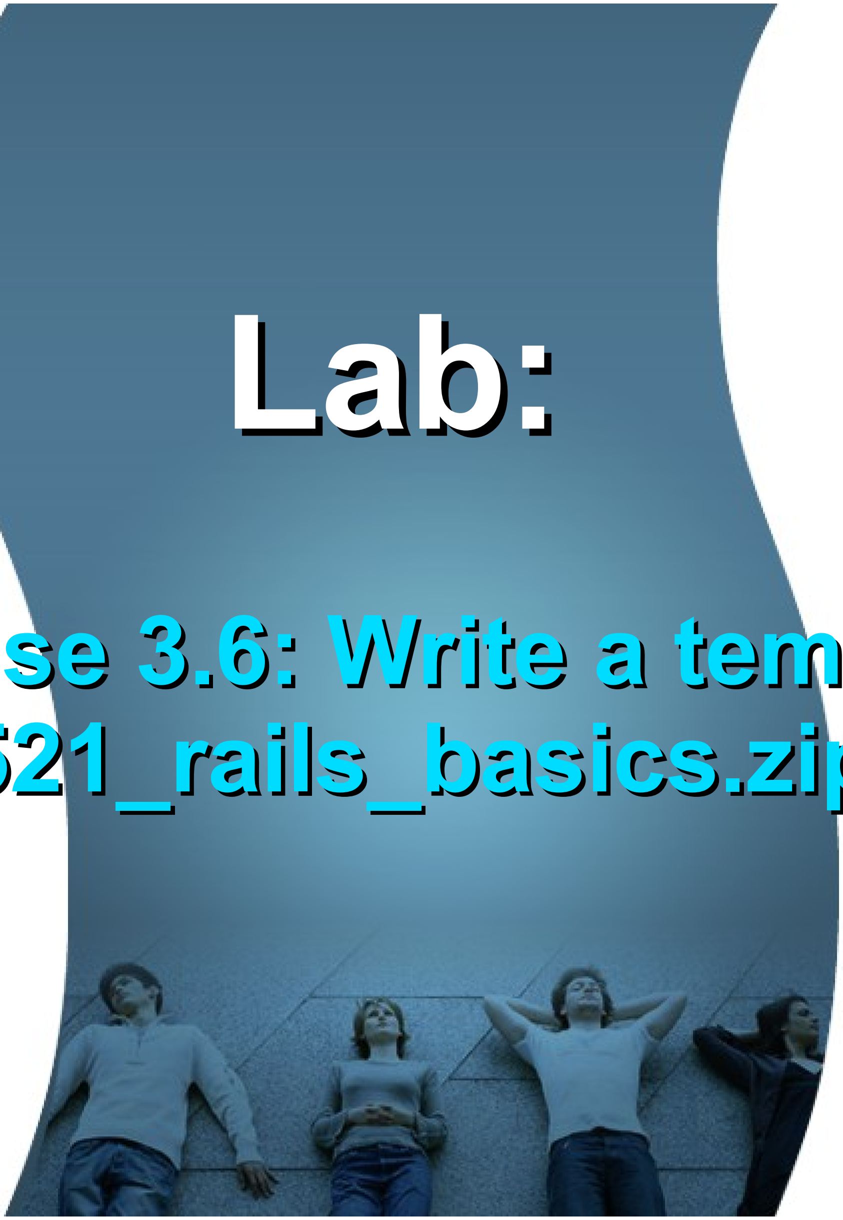


```
My greeting message is <%= @hello.greeting %>
<br/>
The current time is <%= Time.now %>
```

say_hello.html.erb file

```
My greeting message is <%= @hello.greeting %>
<br/>
The current time is <%= Time.now %>
```

Instance variable defined in the controller



Lab:

Exercise 3.6: Write a template
5521_rails_basics.zip

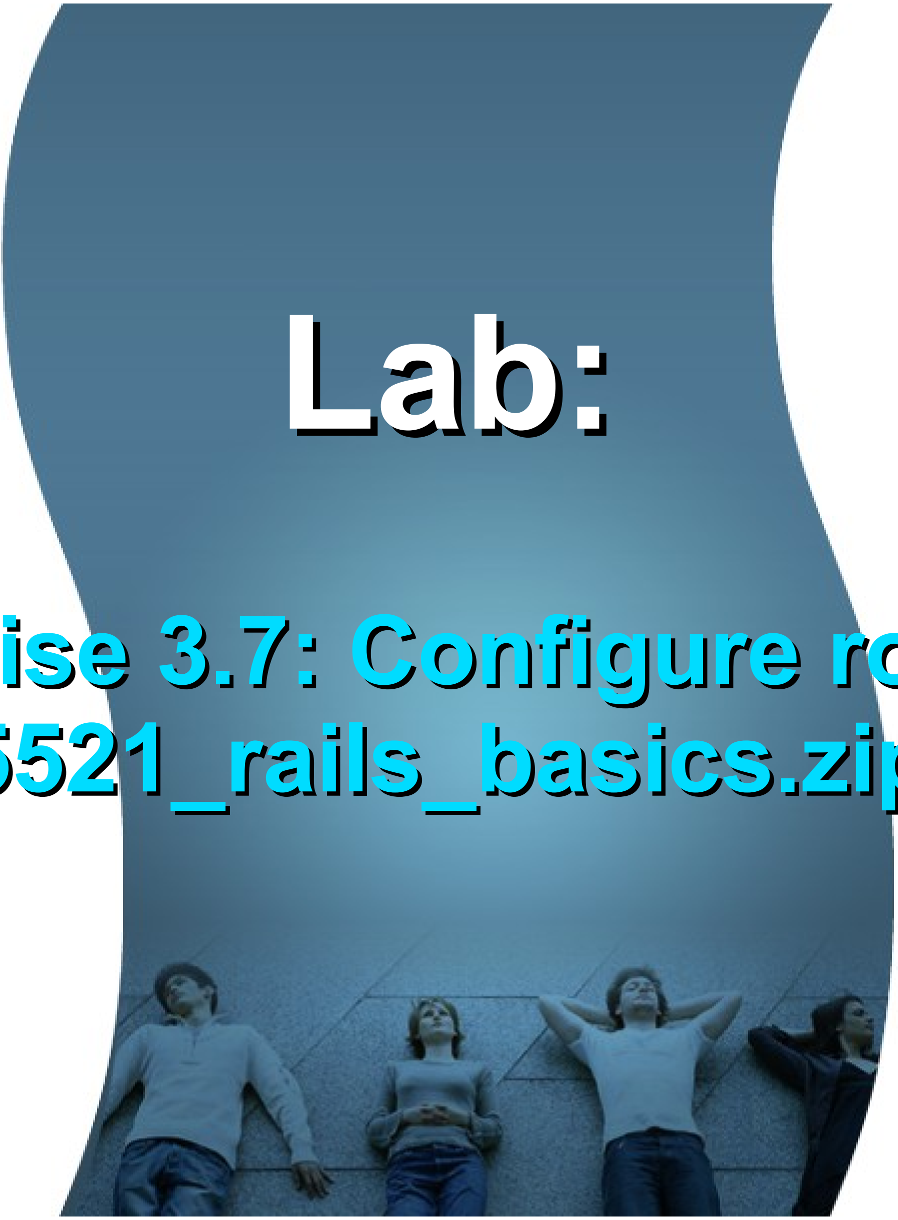
7. Configure URL Routing

URL Routing

- Maps a URL (in the HTTP request) to the controller/action
  - There has to be a matching mapping (routing) for a URL, otherwise 404 error will occur
- *config/routes.rb* file contains the routing setting
  - Rails routing facility is pure Ruby code that even allows you to use regular expressions

config/routes.rb

```
Rails.application.routes.draw do
  # The priority is based upon order of creation: first created -> highest priority.
  # See how all your routes lay out with "rake routes".

  # You can have the root of your site routed with "root"
  root 'hello#say_hello'

  # Example of regular route:
  #   get 'products/:id' => 'catalog#view'

  # Example of named route that can be invoked with purchase_url(id: product.id)
  #   get 'products/:id/purchase' => 'catalog#purchase', as: :purchase
 …


end
```



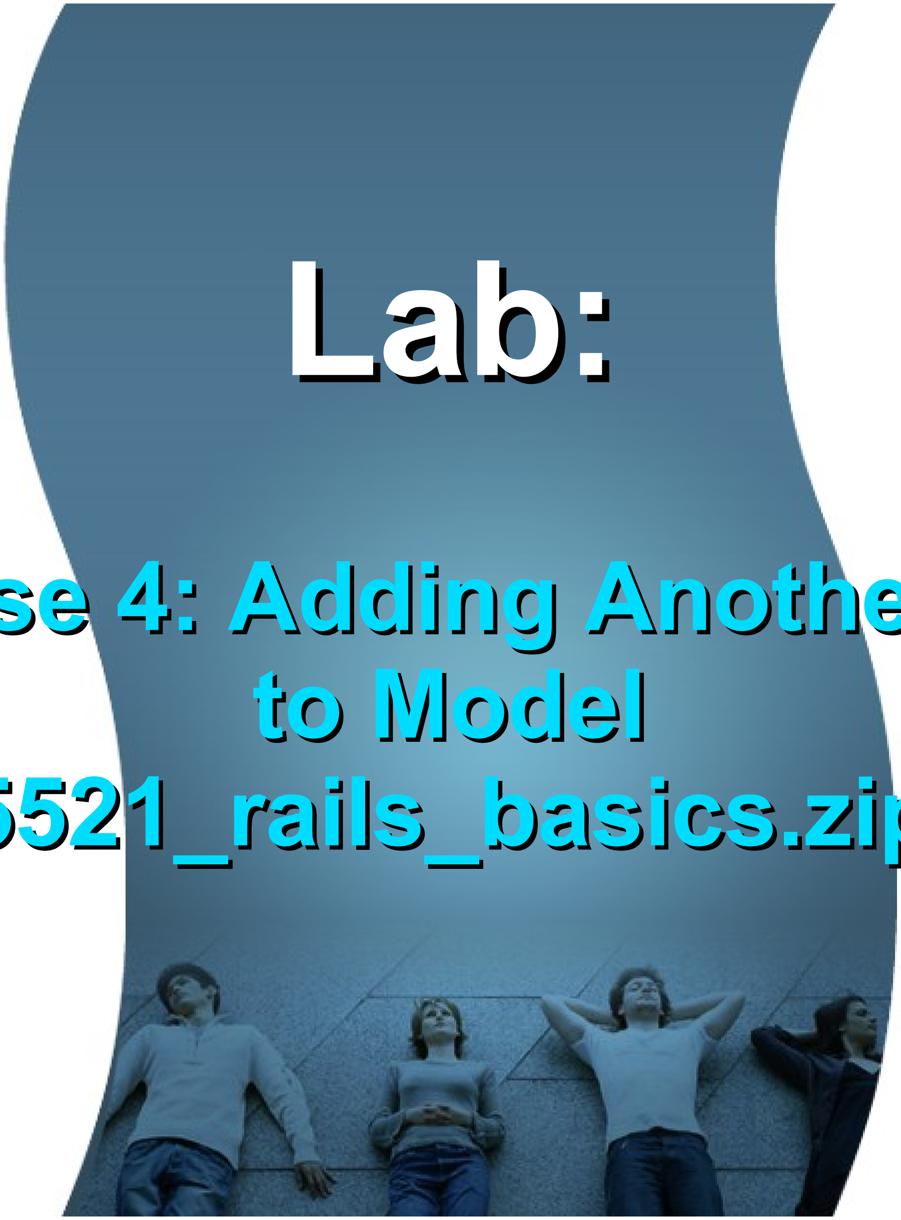
Lab:

Exercise 3.7: Configure routing
5521_rails_basics.zip

Adding another field to Model

Adding another Field to Model

- We now want to add “author” field to “Message” model
- It also means we need to add “author” column to the “messages” table
  - We are going to create a migration file
- We also want to change controller and view to reflect the change



Lab:

Exercise 4: Adding Another Field to Model
5521_rails_basics.zip

Add another template

Add another template

- In the existing template, we want to add a link, through which a user goes to a different template
- We will need to add a new route

Lab:

Exercise 5: Add another template
5521_rails_basics.zip

Create a form for getting user input

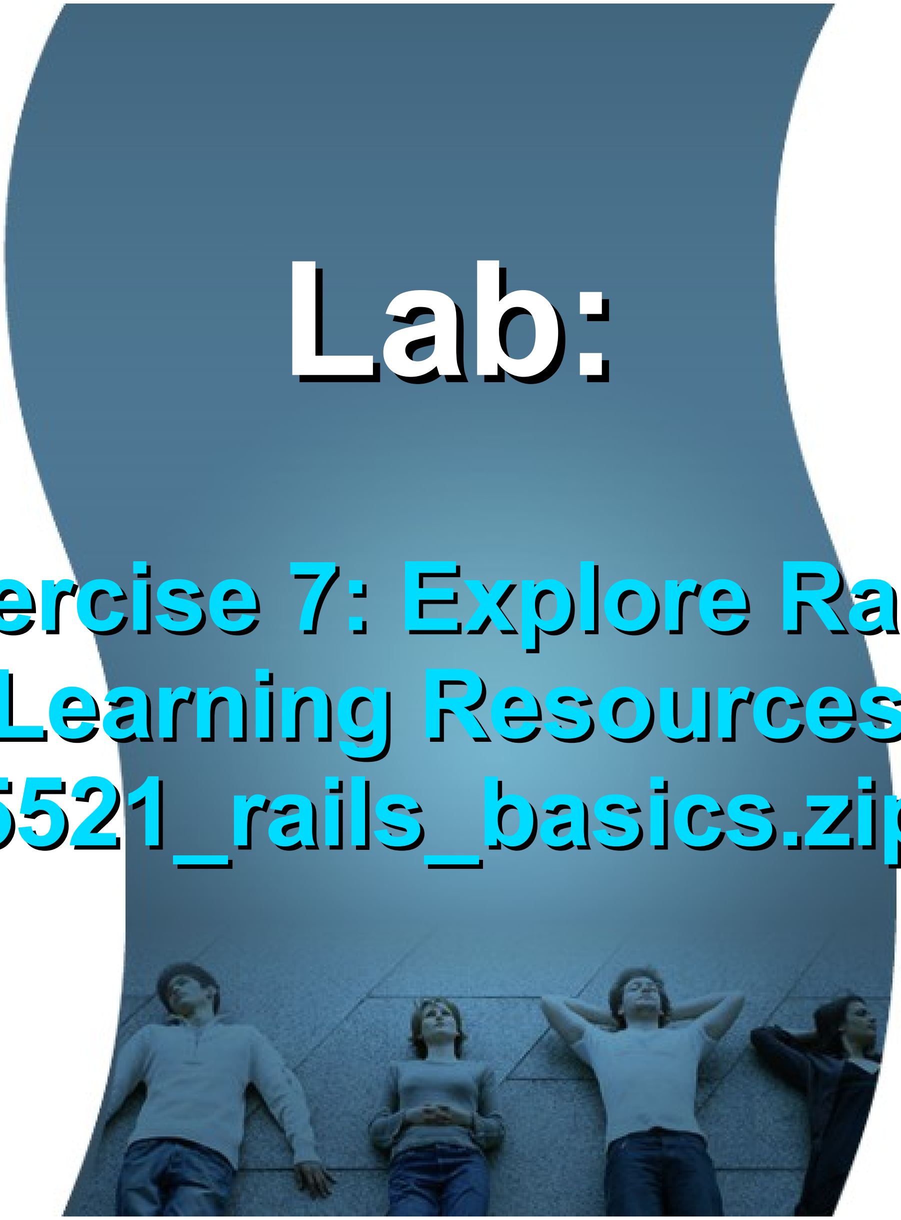
Create a form

- We want to let users to enter some data and we want to pass this data to the controller

Lab:

Exercise 6: Create a form for getting user input 5521_rails_basics.zip

Rails Resources



Lab:

Exercise 7: Explore Rails Learning Resources 5521_rails_basics.zip

Learn with Passion!
JPassion.com