CS 4990: Ruby on Rails Web Application Development

Assignment: Logic

Requirements

- Create a Ruby on Rails project (or use an existing project) and add business logic to the application in the form of custom model logic, model associations, and model validations.

- Add custom logic to your application by adding at least two new methods to your models that each return a value computed using at least one other attribute. Display each of these computed values within at least one view template.

- Define at least one association between two models. Create a one-to-many relationship between the two models by defining a `belongs_to` association on one model and a `has_many` association on the other model. Allow the user to establish relationships between instances of the models by adding the necessary form inputs to the view templates. Also, update the view templates to display the associated resources. For the `belongs_to` association, create a link to the single associated resource. For the `has_many` association, create a link to each of the many associated resources. For example, each moon has one link to its associated planet, whereas each planet has a link to each of its many associated moons.

- Define at least three validations for each of at least two different models. Use at least two different types of validation (presence, length, uniqueness, etc.). Modify the controller actions and form view templates as needed to communicate validation errors to the user, allowing the user to correct any errors before continuing.

Hints and Tips

- When defining the associations for a one-to-many relationship between two models, be sure to create the necessary foreign key. For example, in the Planet and Moon example from class, the Moon belongs to a Planet and will therefore need a foreign key named `planet_id`. To create the foreign key, use a command similar to the following to generate a new database migration. Modify the migration name and foreign key name appropriately for your models.

  ```bash
  $ bin/rails generate migration add_planet_id_to_moons planet_id:integer
  ```

- Don’t forget to apply the new migration after you’ve successfully generated it, using the following command:

  ```bash
  $ bin/rake db:migrate
  ```

- After creating the foreign key for the `belongs_to` association, be sure to add it to the list of permitted parameters in the appropriate controller. For example, in the class example, `planet_id` was added to the list of permitted parameters within the moon controller.

- When defining validations for your models, be sure to modify the `create` and `update` actions for each of your controllers to appropriately handle validation failures. Reference the planet controller in the example from class to see how to do this.

Resources

- Use the example from class as a reference when writing your controllers, models, and view templates.

- For a list of validation helpers that you can use when validating your models, refer to this section of the Rails Guides.

Submission

- Show your completed assignment to the instructor during class or office hours to receive credit.