Ruby on Rails

Week 2 - Scaffolding and Persistence

Scaffolding

Technique used by many frameworks for produce the general structure around which the application can work

Evolved from simple database code generators into full MVC applications

Creates the Model, View, and Controller pieces needed to manage an entity

Scaffolding - Basic Usage

`rails generate scaffold` *`Entity [list of attribute:type]`*

Concrete example

- `rails new ContactList`
- `rails generate scaffold Contact name:string date_of_birth:date`

What did that generate?

- Model, db migrations, tests, routes, controller, views, JSON builders

Go to localhost:3000/contacts

Scaffolding - Using the new Entity

Go to localhost:3000/contacts

(You should have received an error)

Need to migrate the database changes - Let’s look at the generated code

- Model
- Migrations
- Controller
- Views

Scaffolding - Problems

It’s a quick and powerful tool, but…

- Dynamic scaffolding no longer supported
- Changes to scaffolded items (specifically views) are manual and potentially painful
- The UI is ugly

REST

Representational State Transfer

I think we all know what that means!

Essentially just a mechanism for managing state (CRUD operations)

REST - Operations

Operation	Verb	URL
Get all	GET	/products
Get one	GET	/products/{id}
New	POST	/products
Replace	PUT	/products/{id}
Edit	PATCH	/products/{id}
Delete one	DELETE	/products/{id}
Delete all	DELETE	/products

REST - Rails

Requires a bit more complexity because we also need to render a view

```
rails routes
```

REST - JSON Support

Look at Contacts controller, there is URLs with a “.json” file extension

Created for you by scaffolding

Useful for accessing controllers as pure services

Use a tool like Postman to try it out

Testing Rails

```
rails test

rails test:system
```

Models

Attributes inferred from database column names

Class name singular, table name plural

- Table name can be specified if needed
- Some built in known plurals (e.g. person -> people)

Can have validations

Can have relationships

Active Record

Design pattern where objects carry both persistent data and the behaviors to operate on that data.

In Rails it’s also an ORM framework

- In a sufficiently complex schema, mapping objects to underlyuing tables is challenging
- ORM frameworks limit how much we need to know about the underlying data structure
- For legacy or very challenging schemas, most ORM frameworks allow encapsulated use of SQL

Active Record - basic static operations

create - instantiate and save

new - instantiate but not save

all - retrieve all

first - retrieve first

find_by - retrieve first using criteria

where - retrieve all using criteria

Active Record - Basic instance operations

save - saves the specific instance

update - updates the specific instance and saves it

destroy - deletes the specific instance

Active Record - Validations

Built-in mechanism for validating object state

- Check for null fields
- Check for a range of values
- Check format and length
- Cross validation of fields

Active Record - Scopes

Allows definition of commonly used “queries”

Can be chained together

Usage looks like method static method calls

Default scope exists

Active Record - Relationships

has_one - a child has_one parent

has_many - a collector has_many cars

belongs_to - a car belongs_to an enthusiast, a parent belongs_to a child (OK, so sometimes it can get a little iffy)

Adds dynamic methods to the objects

- Ex: enthusiast.cars
- Use enthusiast.methods.sort - Object.methods

Active Record - Relationships

```
rails g scaffold Author name
```

```
rails g scaffold Book title author_id:integer
```

- Or: `rails g scaffold Book title author:references`

Add the appropriate `has_many` and `belongs_to` lines in the models

```
rails db:migrate
```

Active Records - Trying it all out

Create a “library” application

You need books and authors

A book belongs to an author, and an author can have many books