Forms

Requirements

1. Route (resource and action) and view that serves the form itself (usually, `name`, or `edit`)
    - Form fields' `name` attribute appears as keys in `params[]`
    - For resource-based forms, use `form_with()` helper so that attributes associated with the resource get collected into a nested hash
2. Identify the action that receives submission (usually `create`, or `update`)
3. Ensure there are routes, actions, views, etc. for each

Redirection, Flash, and Session

- Idiom: redirect user to a more useful page (eg: index if create is successful or new if unsuccessful)
- Redirect triggers a new HTTP request
    - May want to inform the user why they were redirected
    - Use `flash` which quacks like a hash that persists only until the end of the next request
    - `:notice` conventionally for information, `:warning` conventionally for errors
        - Often rendered in application layout
- Generally don't want to redirect on failure (if we just render something, then the state is persisted)
- Root route: can direct it to a diff page with `root to: <controller>#<method>` or `root to: redirect('<path>')` in `routes.rb`
    - Redirect changes the URI to the correct page

Databases

- Development, production, and test environments each have own db
    - Based on env var `RIALS_ENV` ## Migrations
- Use `$ rails generate migration <migration name>`
- Use `$ rails db:migrate` to migrate environment
- Updates `db/schema.rb`
- Can use `create_table` in migration like:

```ruby
class CreateMovies < ActiveRecord::Migration
    def change
        create_table :movies do |t|
            t.string :title
            t.string :rating
            t.date :release_date
        end
    end
end
```

- Make sure to add a new model in `app/models` if necessary

Debugging

- Issue: errors can appear in many different spots
- Can do `printf` debugging (instrumentation) but doesn't work in prod
- Can use `$ rails console` to interact with code
- Can use `debugger` statements and run `$ rails server --debugger`
- Can use logging
    - Only option that works in prod
- Read, Ask, Search, Post (RASP) $\triangleq$ good idea for how to debug

- Evaluate backtrace (where in the call stack an error occurred)

Instrumentation

- Can use `< %= debug(<value>) %>` or `< %= <value>.inspect %>` in a view
- Can write into logs with `logger.debug(<value>)`
- Beware of `puts` or `printf` b/c it doesn't do anything in prod