Model-View-Controller in Rails

MVC Architectural Design Pattern

- Doesn't specify how or where the model state is stored
- Doesn't specify the technology or nature of the 'connections' among MVC

Model

- State $\triangleq$ what is stored about the model
- Logic $\triangleq$ operations on model state
- Subclass from `ApplicationRecord`
  - Connects a model to the db
  - Provides db CRUD operations on the model

Models as Resources

- Actions $\triangleq$ CRUD
- Naming $\triangleq$ routes
- Inspect/modify resources $\triangleq$ views and controllers
- State $\triangleq$ db

Backend DB

- Each model gets its own db table (named the plural of the resource)
- Collection of instances $\triangleq$ db table
- Instance $\triangleq$ a single row (model state)
- Attribute of mode $\triangleq$ db column
  - Column names become getters and setters
- Class and instance methods $\triangleq$ model logic
- SQL based
  - Rails has methods to compose SQL queries
  - Returns relations (you can chain these methods) $\Longrightarrow$ materialization at the end
  - Have to be persisted with `save` after manipulating in-memory instance
  - Don't use string interpolation for these b/c can open up to SQL injection

View

- Lets the user see and interact with the model
- Subdirectories in `views` folder for each controller and action

Controller

- Mediates interactions between model and view
- updates view when model state changes
- invokes model logic in response to user actions

Rails as an MVC Framework

- Each thing that an app manipulates will have a model, view, and a controller
- Routing to diff config file for only defining routes
- Models store state in a db
  - Generally each has its own table
- General structure:
  1. App server passes request route to rails
  2. Routing subsystem parses params, identities matching action

3. Controller action is called
4. Model state stored/updated in relational db tables
5. View is rendered

- Everything is stateless besides db