Authentication vs Authorization

Don't confuse **authentication** and **authorization**.

Note that none of the bold terms in this article are programming jargons! They are just plain-old English words._

Think of **authentication** as allowing your users to **log into** your app.

Authorization, on the other hand, is all about **granting specific permissions** to your logged in users.

Example of Authorization

In an e-commerce application, there could be many different types of users that can each do different things. For instance,

- A store manager can upload new products and delete products (manage inventory)
- A customer can buy products but cannot manage product inventory
- A store admin can see orders but cannot edit the details of an order (to ship products)

This is where the concept of **authorization** becomes relevant!

Authorization in our PairBnB Clone

Let's implement some type of authorization in our PairBnB clone. What we implement may not be in the actual AirBnB site, but for learning's sake, let's now pretend that we have some form of authorization. As usual, we will have users who can either list their properties or make reservations, let's call them `customer`. On top of that, let's have a `moderator` who has the authority to verify listings, i.e. there will be a checkmark/badge next to a listing (something like Facebook's verified badge). Don't worry about the criteria for a listing to be verified, after all, we just want to implement a `moderator` role here. By the way, you can call these roles other terms if you prefer as long as your naming is clear and consistent. Finally we will have a `superadmin` role. By convention, most developers make it such that superadmins can do everything.

Thinking Step-by-Step

It's often helpful to breakdown the feature you want to implement to basic steps.

In this case, there are probably 2 key steps:

1. Since we can categorize users into various groups of people with various levels of permissions, let's group them up based on their roles. In other words, users should have different **roles** . We can store their **roles** in the database.

   Depending on your app, some apps may require a user to play multiple roles - e.g. a user can both be a teacher and a school administrator.

   However, majority of apps only allows each user to only play 1 role, like our version of PairBnB. A customer is a customer and a superadmin is a superadmin. If they want to be a superadmin and customer at the same time, they would just have to have 2 different accounts.

2. Each **role** should have their various **permissions** (e.g. superadmins can verify listings and no other roles can, customers can host/book listings and no other roles can).

Now that we have some rough ideas, we can conclude that the **first step** for us is probably to implement some kind of logic in our app to allow users to have their own respective role (notice the singular form `role`!).

And the **second step** is to implement some other kind of logic to allow each role to perform only their allowed actions.

Objectives

0. Add a verification column to listings

Add another column to your listings table which has data type boolean to store the verification.

1. Declare roles.

In our case of PairBnB, we should at least have `superadmin`, `moderator` and `customer`. There are many ways to implement roles, e.g.

- A simple boolean in your user columns to state whether this user is `admin` or `not admin`, `moderator` or `not moderator`, `customer` or `not customer`.
- Storing a user's role as a string in the database.
- Use some gem (for your own future's sake, let's practise writing code on our own and not use external gems unnecessarily)
- **[Try this method for this challenge!]** A very clean way to do it is using the `ActiveRecord::Enum` module. Check it out!

2. Check for Permission Now, write some code to prevent a customer from verifying his/her own property. To be safe, let's by default, blacklist every action for all roles and only whitelist the permitted actions for a role manually.

Here's some sample code for you to get started.

```
class ListingsController < ApplicationController
  def new
    # authorization code
    if user.customer?
      flash[:notice] = "Sorry. You are not allowed to perform this action."
      return redirect_to some_other_url, notice: "Sorry. You do not have the permissino to verify a property.
"
    end
    # end authorization code


    # other code to make the new action work!
  end
end
```

3. Don't Repeat Yourself!!
You are pretty much going to perform these authorization checks for almost all controller actions. So, let's not write the same code over and over again in every controller action! (definition: controller actions are just methods in your controller.)

Try extracting out your authorization code in the `Listings#new` action into an application helper. (Application helpers are just methods in your application controller that can be used in pretty much any controller actions that inherits from application controller).

```
# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
```

```
  def allowed?(action:, user:)
    # implement some code
  end
end


# app/controllers/listings_controller.rb
class ListingsController < ApplicationController
  def new
    allowed?(action: <replace with your action>, user: current_user)
  end
end
```

User Experience

To deliver excellent user experience, you probably also want to hide certain elements from the view for different roles. For example, as a customer user, it's definitely a bad user experience if I can see the verify property button if I'm not allowed to do so. As such, figure out a way to hide the `verify property` button from logged in users who plays the **customer** role.

After this challenge

This challenge **isn't** a one off challenge. As you implement new features/user stories in your app, you're going to need to add new authorization logic for those functionalities.

So as you build PairBnB, keep adding code to allow only certain roles from performing the new actions you implement.

Caveat: Complex scenarios

In most applications, a simple role-based authorization like the one in this challenge should suffice. However, there are certain enterprise grade applications that require more complicated permission levels. The idea is the same but we won't deal with that in this bootcamp.