Chapter 6 Notes: Modelling Users

Authentication:

- Prebuilt solutions: Clearance, Authlogic, Devise, and CanCan, OpenID, OAuth
- Custom Solution—often the better option, since pre-built solutions require a significant amount of customization to begin with and are often "black-boxes" in terms of how they work

6.1 User model

User class built in console lacked persistence

$ rails generate model User name:string email:string: creates a model with name and email attributes
NOTE: model names are singular: a Users controller, but a User model.

Creates a new file called a **migration**

- provides a way to alter the structure of the database incrementally, so that our data model can adapt to changing requirements
- **t.timestamps**, command that creates two columns called created at and updated at, which are timestamps that automatically record when a given row is created and updated

$ rails db:migrate

- creates a file called db/development.sqlite3
- to see the structure of the database first download the database file to the local disk, as shown in Figure 6.5 then open development.sqlite3 with DB Browser for SQLite

Using a test console

$ rails console --sandbox creates a new console where all changes will be rolled back upon exiting Note: the console will automatically load the current Rails environment (including models)

Commands

User.create	Combines assigning values and save function
User.destroy	Undoes User.create
User.find(*1*)	Finds record with that ID
User.find_by(email: mdoucette@example.edu**)**	Will search User table for specified attribute(s)
User.first	Will return first item in User table
User.all	Will return all users
User.reload	Loads the object from the database; overwriting what is in memory
User.name = "NewUser**"**	Update a single attribute
User.update_attribute(name: "NewUser**", email: "**NewUser@Example.com**"**	Update multiple attributes

Note: both create and destroy return an object and will exist in memory after the command is executed

6.2 User Validations

How to validate presence, length, format and uniqueness as well as using confirmation.

1. Create a passing test – to know that the object itself is valid
    a. Define **@user** using the special **def setup** method—which automatically gets run before each test
2. Next we will use tests to substitute attributes with invalid values and test whether or not **@user** is valid or not
    a. Uses "a" * 244 to repeat a character that number of times
    b. Regex uses pattern matching for numbers and strings and is extremely useful to know, but there are entire books written on this…. There is a good breakdown in Table 6.1
        i. Rublar.com is a great way to test regular expressions
    c. Forcing unique values in Active Record doesn't guarantee there won't be duplicates in database, if two identical records come in at the same time– this has to be done at the database level

Editing the database

Add an index and enforce unique values for emails at the database level.

```
rails generate migration add_index_to_users_email
```

This will create a migration file that we can edit to create an index (to make searching faster) and enforce unique values. To finalize changes run: **$ rails db:migrate**

Note: the fixtures file that was generated automatically (**$ rails generate model User name:string email:string**) will now need to be cleared out since they are not unique.

To alter a record before saving, update model with: `before_save { self.email = email.downcase }`

6.3 Adding a secure password

Add **has_secure_password** to User model which grants the ability to:

- Save a **password_digest**
- Use **password** and **password_confirmation** attributes
- And **authenticate** whether a password is correct

1. Create a new migration file to add **password_digest** column to database

```
$ rails generate migration add_password_digest_to_users
password_digest:string
```

2. Update **app/models/user.rb** to include:
    a. **has_secure_password**
    b. **validates :password, length: { minimum: 6 }**
3. Note: User_test will now fail
    a. Add a **password** and **password_confirmation** for **@user** in the **def setup**
    b. You will also need to set up test to check a minimum password length

```
@user.password = @user.password_confirmation = "a" * 5
assert_not @user.valid?
```