Chapter 6 Overview:

Though there are many systems for user authentication, it's usually better to write your own. Need to know how it works and what's going on, can customize it to your personal needs

- Migrations allow us to modify our application's data model.
    - Default data structure is the model, default library for database interactions is Active Record (which you can use without SQL)
    - Migrations (with Active Record [model library]) allow persistence and data definitions to be written in Ruby
    - *rails generate model* to create the user model (similar to generate controller)
        - Generates Model .rb file (user.rb) – inherits from ActiveRecord::Base
        - Generates Migration file – alter data structure to adapt to changes in code
            - Identified with timestamp to avoid collision issues
            - Creates a table with the specified data (Users table has name and email columns) and id, created_at, and updated_at columns
            - Rails naming convention: Model is singular (instance of object) while database is plural (will have many models/objects)
    - *bundle exec rake db:migrate* – migrating up command, creates SQLite database
    - *bundle exec rake db:rollback* – Roll back database/migrate down/undo migrate

- Active Record comes with a large number of methods for creating and manipulating data models.
    - Can make edits in sandbox – Rolls back on exit, undo database changes!! This does lock db, so can't do migrates or rollbacks until all sandboxes are closed
        - *User.new[(name: "", email: "")]* – create new user [void unless specified], return object
        - *valid?* – validate entry, return bool
        - *save* – save user to database, sets the id and timestamps, return bool
        - access attribute by name (user.name/email/id)
        - *.create* will create and save entry, return object
        - *.destroy* destroys object (but still exists in memory), return object
        - *.find(id)* – find user in database, return object or *RecordNotFound*
        - *.find_by(attr: "attribute_name")* – return object
        - *.first* – return first entry; *.all* – return all members as ActiveRecord::Relation object (aka an array)
        - Edit a record by setting the attribute (*user.email = "b@g.com")* and *.save*
            - This will update *.updated_at* to current time stamp
        - Undo an edit by calling *.reload* (*user.reload.email*) BEFORE SAVING
        - *.update_attributes()* updates multiple attributes, return boolean
        - *.update_attribute()* bypasses restrictions for save

- Active Record validations allow us to place constraints on the data in our models.

- Common validations include presence, length, and format.
  - Listing 6.5 for code
  - *bundle exec rake test:models* will test the models
  - to add presence validation, add the following:
    - add following test to test/models/user_test.rb

```
test "name should be present" do
  @user.name = "     "
  assert_not @user.valid?
end
```

    - add *validates :name, presence: true* to app/models/user.rb
  - *.errors.full_messages* – sandbox call, use to see why you can't save/valid object
  - Validate length with *test "___ should not be too long"* (Listing 6.14) and set length (Listing 6.16)
    - (user_test.rb) *@user.name = "a" * 51* //string multiplication, 51 char
    - (user.rb) *validates :name, presence: true,* ***length: { maximum: 50 }***
  - Validate format (like an email) by adding different email styles to user_test.rb (see Listing 6.18)
    - Can add a custom error message with *assert @user.valid?, "#{valid_address.inspect} should be valid"*
    - Need to add VALID_EMAIL_REGEX to user.rb (Listing 6.21)

- Regular expressions are cryptic but powerful.
  - Set a regular expression to validate the email (table 6.1 in section 6.2.4)
  - VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i (explained in table 6.1)
  - rubular.com is a regular expression editor – has a quick reference guide at bottom and has an error check in it so awesome resource!

- Defining a database index improves lookup efficiency while allowing enforcement of uniqueness at the database level.
  - Add *test "email addresses should be unique"* to user_test.rb and *uniqueness: {case_sensative: false}* to user.rb file
  - Due to threading/multiple processes happening at once, this is not enough. If an email is sent twice (hit submit twice), they will be checked against the database which will not have either of them yet and accept them
  - *rails generate migration add_index_to_users_email* to add structure (index) to existing model, it won't be predefined (rails isn't sure what your changing) so add *add_index :users, :email, unique: true* to new db/migrate/[file]
    - code adds an index to email col of users table, sets it to be unique
    - Make sure to migrate db!
  - Now empty test/fixtures/users.yml file – it doesn't think emails should be unique, we'll get back to it in Ch 8 (fixatures are a way of organizing data that you want to test against)
  - Add *before_save { self.email = email.downcase }* to user.rb to take care of case sensitive issues

- We can add a secure password to a model using the built-in `has_secure_password` method.
    - Save a hashed version of the password in the database
        - Applying an irreversible hash function (mapping function) to input data
    - We're taking a password, hashing it, comparing it to hashed password in db
    - *has_secure_password* – (in user.rb) rails method that saves hashed *password_digest* to db, makes *password* and *password_confirmation* upon object creation and requires them to match, and an *authenticate* boolean method to confirm password on login
        - Need to add *password_digest* for this to work: *rails generate migration add_password_digest_to_users* ***password_digest:string***
        - Since we gave rails our new variable, it auto generates the change function, just have to migrate the database
        - Need *bcrypt gem* for a good hash function, add gem and bundle install
    - Add *password* and *password_confirmation* fields to *@user* in user_test.rb
    - Add *test "password should be present (nonblank)"* and *test "password should have a minimum length"* to user_test.rb (Listing 6.38)
    - *validates :password, presence: true, length: { minimum: 6 }* to user.rb to validate user

One last comment: make sure to call *heroku run rake db:migrate* in console since we're using a database now