Security of the application

Security of sensitive data like database passwords

- The use of gem devise guarantee encryption of passwords stored in database with a hash function.

- We will use the rails filters in class processing sensitive data : the_role_required, before_filter.

- To avoid session hijacking :
  - In an unencrypted wireless LAN it is especially easy to listen to the traffic of all connected clients.
    So we will provide a secure connection over SSL to encrypt informations : this will be accomplished by always forcing SSL.
    config.force_ssl = true

  - Most people don't clear out the cookies after working at a public terminal. So if the last user didn't log out of a web application, you would be able to use it as this user. We'll provide a log-out button and make it prominent.

To avoid SQL injection

- We will always convert the user input to the expected type before using it.

- When it's possible, instead of passing a string to the conditions option, we can pass an array to sanitize tainted strings like this :
  Model.where(login: entered_user_name, password: entered_password).first

To avoid xss attack

- We will filter input using sanitize function. It will be a whitelist input filtering.

Example :

tags = %w(a b strong i em li ul ol h1 h2 h3 h4 h5 h6 blockquote br cite sub sup ins p)
s = sanitize(user_input, tags: tags, attributes: %w(href title))

This allows the given tags. But we can use sanitize without tags if it's not needed.

- When re-displaying user input which hasn't been filtered, we will use the escapeHTML method to replace the html character. The plugin name is SafeErb.

  This is an example of a vicious input by a user in a field named profile :

  Hey, nice forum!<script>alert("Guess who just got owned?")</script>

  This code will execute the alert script.
  <%= raw @user.profile %>

  If we use escapeHTML : **<%= html_escape @user.profile %>**
  That will display :
  Hey, nice forum!&lt;script&gt;alert(&quot;Guess who just got owned?&quot;)&lt;/script>