Web Development

Higher Diploma in Science in Computer Science

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Sessions

Web Development

How to Make an Application out of a Web Page?

- On the internet, a web page is a web page is a web page…
- If you surf from ./page1.html to ./page2.html these are two unique requests.
- The server doesn‘t know anything about the fact that both pages are visited by the same user.
- Sessions are the technique used to logically group several requests into a “group“ (called a session)
- If you start a session, the server will know that it‘s still the same user who surfed from ./page1.html to ./page2.html

Sessions

- HTTP itself is “stateless”
  - no state stored on the server between requests from the same client
- but many web apps are stateful
  - necessary to connect requests from the same user / browser / browser-window, e.g. shopping cart, appointments calendar etc...
- *Session*
  - multiple requests performed in a stateful context
- *Session tracking*
  - technique that allows sessions in stateless environments

Session Tracking / Handling

- User surfs to http://demo.com
    - Server (on 1st request / if no sessionID stored on client)
        - generates unique session id, which is mapped to ...
        - ... a session-object
            - stored in memory (lost on shutdown), in a file or in database
            - can contain anything (list of articles, game state, counters, ...)
    - Session id is added to the response
- from now on:
    - each subsequent request from the same user (browser) must contain the session id ...
    - ... which is used by the server to map to the session-object
- No data gets stored on the client, except SessionID

Session Tracking Techniques

- Cookies
- Hidden Form Fields
- URL Rewrite

Cookies

1. Server creates a cookie with session-id on first request

2. Server maps id to a new user-specific session object

3. The session-id is sent to the client with the first response

4. ..and automatically added by the browser on each further request (to the same address/domain/…)

5.Server receives request + cookie with session-id

6.Server maps session-id to session-object

- Potential problems:
  - users may disallow the usage of cookies in their browsers

URL Rewrite

- Basic idea:
  - Server adds the session-id to all links the user can follow
    - http://server/myhome
  - is changed to
    - http://server/myhome?sessionid=123
  - session-id must be dynamically added
    - functionality usually offered by scripting frameworks
- Pro
  - simple, works with every browser (no cookies required)
- Contra
  - all URLs in response pages must add session-id
  - URL displayed in browser is the rewritten URL

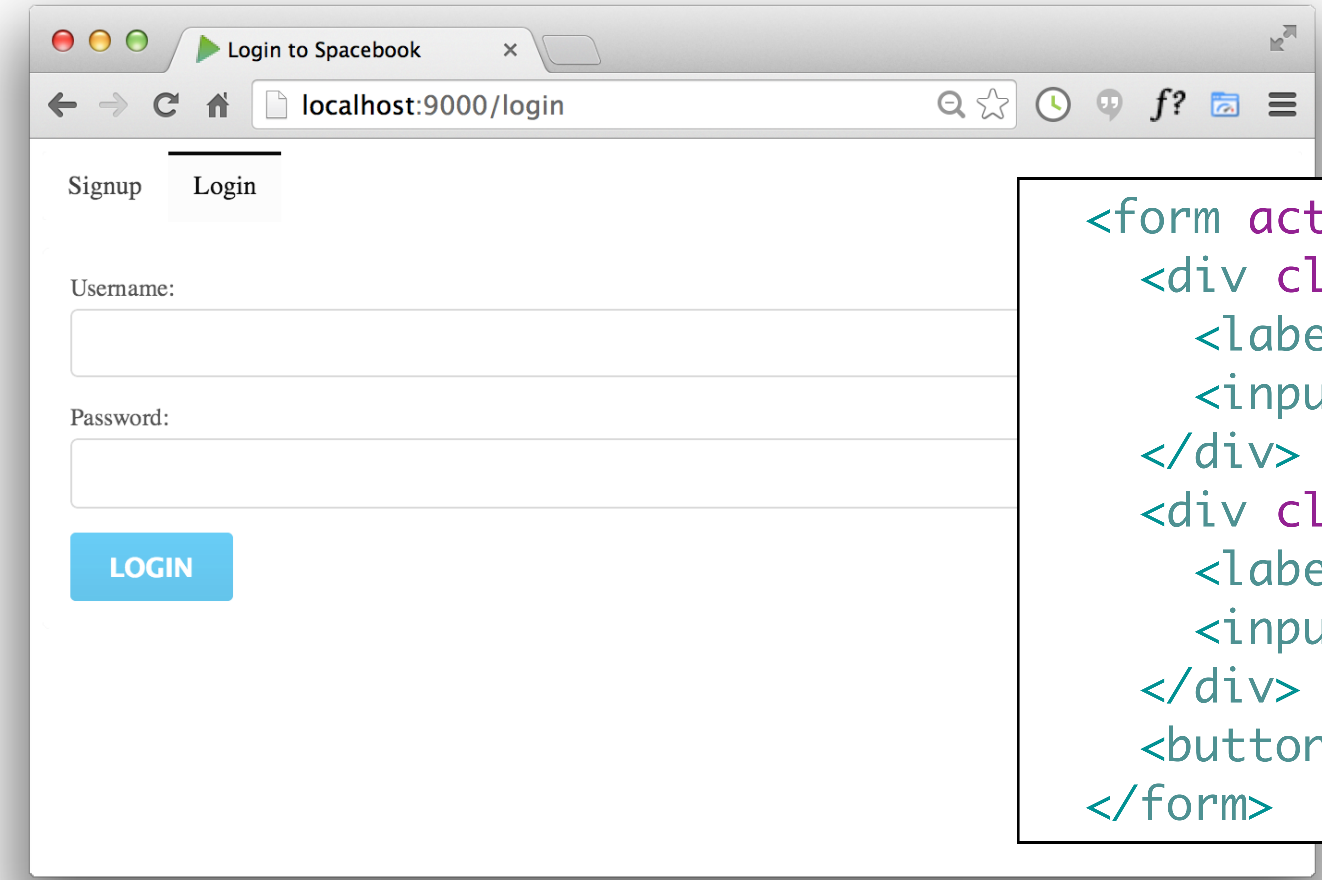
Hidden Form Fields

- In HTML, we can define "hidden" fields in a form
    - <input type="hidden" name="sessionid" value="123">
- These fields are not visible and cannot be changed by the client
- Usage:
    - server creates a session-object for each client and generates a unique ID
    - When HTML documents are created and sent back, the hidden form field is automatically generated containing the actual ID
    - Upon form submit, the session ID is automatically sent back to the server
    - The server can associate this call with an already existing session
- Pro:
    - Simple, works with every browser (no cookies required)
- Contra:
    - Form must be added to all pages
    - Form must be submitted at each request to the server

Web Frameworks

- Cookies generally preferred.
- However, framework will try to ‘abstract away’ specific session management technology, and deliver simpler abstraction to the programmer
- Framework may in fact be able to switch between different techniques depending on circumstances.

Login



```
<form action="/authenticate" method="POST">
  <div class="field">
    <label> Username: </label>
    <input type="text" name="email">
  </div>
  <div class="field">
    <label>  Password: </label>
    <input type="password" name="password">
  </div>
  <button class="ui blue submit button">Login</button>
</form>
```

```
POST    /authenticate                                Accounts.authenticate
```

```
public static void authenticate(String email, String password)
 {
   Home.index();
 }
```

Authenticate Action

```
public static void authenticate(String email, String password)
{
  ...
}
```

- Need to decide whether to allow a user to log in (they must register first), and subsequently 'remember' which user has logged in.
  - In the authenticate method, see if the given user is registered or not.
  - If they are registered, place the user 'id' into a 'session' object
  - This session object will be available to other controllers during subsequent page visits.

Extend User Class ....

2 new methods:

Search for a User object matching a specific email

Check if a given objects password matches a specific password.

```java
public class User extends Model
{
  public String firstName;
  public String lastName;
  public String email;
  public String password;

  public User(String firstName, String lastName,
              String email,     String password)
  {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.password = password;
  }

  public static User findByEmail(String email)
  {
    return find("email", email).first();
  }

  public boolean checkPassword(String password)
  {
    return this.password.equals(password);
  }

}
```

Authenticate Action

```
public static void authenticate(String email, String password)
{
  Logger.info("Attempting to authenticate with " + email + ":" +  password);

  User user = User.findByEmail(email);
  if ((user != null) && (user.checkPassword(password) == true))
  {
    Logger.info("Authentication successful");
    session.put("logged_in_userid", user.id);
    Home.index();
  }
  else
  {
    Logger.info("Authentication failed");
    login();
  }
}
```

- user.id
    - Although the class User does not explicitly have a field called ‘id’, because User is a ‘model’ class - and id field is always generated.
    - This is unique - and we will use it widely in the application.

Authenticate

```
public static void authenticate(String email, String password)
{
  Logger.info("Attempting to authenticate with " + email + ":" +
```

Search for matching user

```
  User user = User.findByEmail(email);
```

If one is found, see if password matches

```
  if ((user != null) && (user.checkPassword(password) == true))
```

if they match, store user 'id' in 'session'

Let user in to home page

```
  {
    Logger.info("Authentication successful");
    session.put("logged_in_userid", user.id);


    Home.index();
  }
```

if not, revert to start page

```
  else

  {
    Logger.info("Authentication failed");
    login();
  }
```

Sessions

- Every time a user make a ‘request’ - i.e.
  - presses a link
  - navigates to a new page
  - submits a form
- The ‘action’ has no idea who the user is each time such a request arrives
- Remember - there may be hundreds or thousands of requests, from different users, arriving concurrently.

Session Objects

- A mechanism whereby our program can ‘know’ who the ‘current’ user is.
- Implemented by a complex process involving ‘cookies’, ip address, + various other techniques.
- Simplified for the programer in Play as follows:

- If we ‘know’ who the user is, then we store the id in the ‘session’ object:

```
session.put("logged_in_userid", user.id);
```

- Later, in another action, if we want to find out who the is, we ask the session object:

```
String userId = session.get("logged_in_userid");
User user = User.findById(Long.parseLong(userId));
String name = user.firstName;
```

Session - put and set

- put into the session object the user.id value at the key ‘logged_in_userid’

```
session.put("logged_in_userid", user.id);
```

- Ask the session for the value corresponding to the key ‘logged_in_userid’

```
String userId = session.get("logged_in_userid");
```

- Use that value to look up the database for a corresponding user object

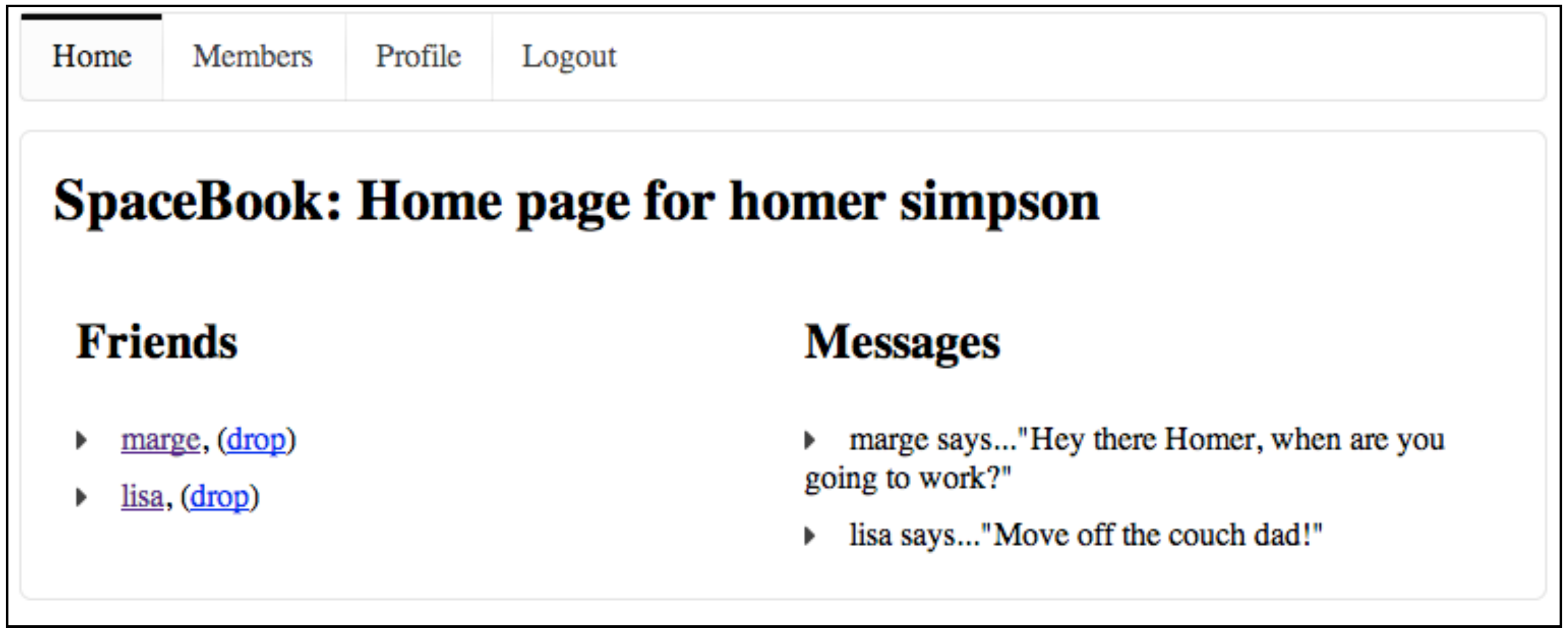
```
User user = User.findById(Long.parseLong(userId));
```

- Get the name of the user from the user object

```
String name = user.firstName;
```

Home Page Heading

- Once a user is successfully logged in, we would like to display the user name in the title of some of the pages.
- Currently 'hard coded' to "Homer Simpson"



Home Members Profile Logout

SpaceBook: Home page for homer simpson

Friends

- marge, (drop)
- lisa, (drop)

Messages

- marge says..."Hey there Homer, when are you going to work?"
- lisa says..."Move off the couch dad!"

```
...
 <h2 class="ui header">SpaceBook: Home page for homer simpson </h2>
...
```

views/Home/index.html

```
public class Home extends Controller
{
  public static void index()
  {
    render();
  }
```

controllers/Home.java

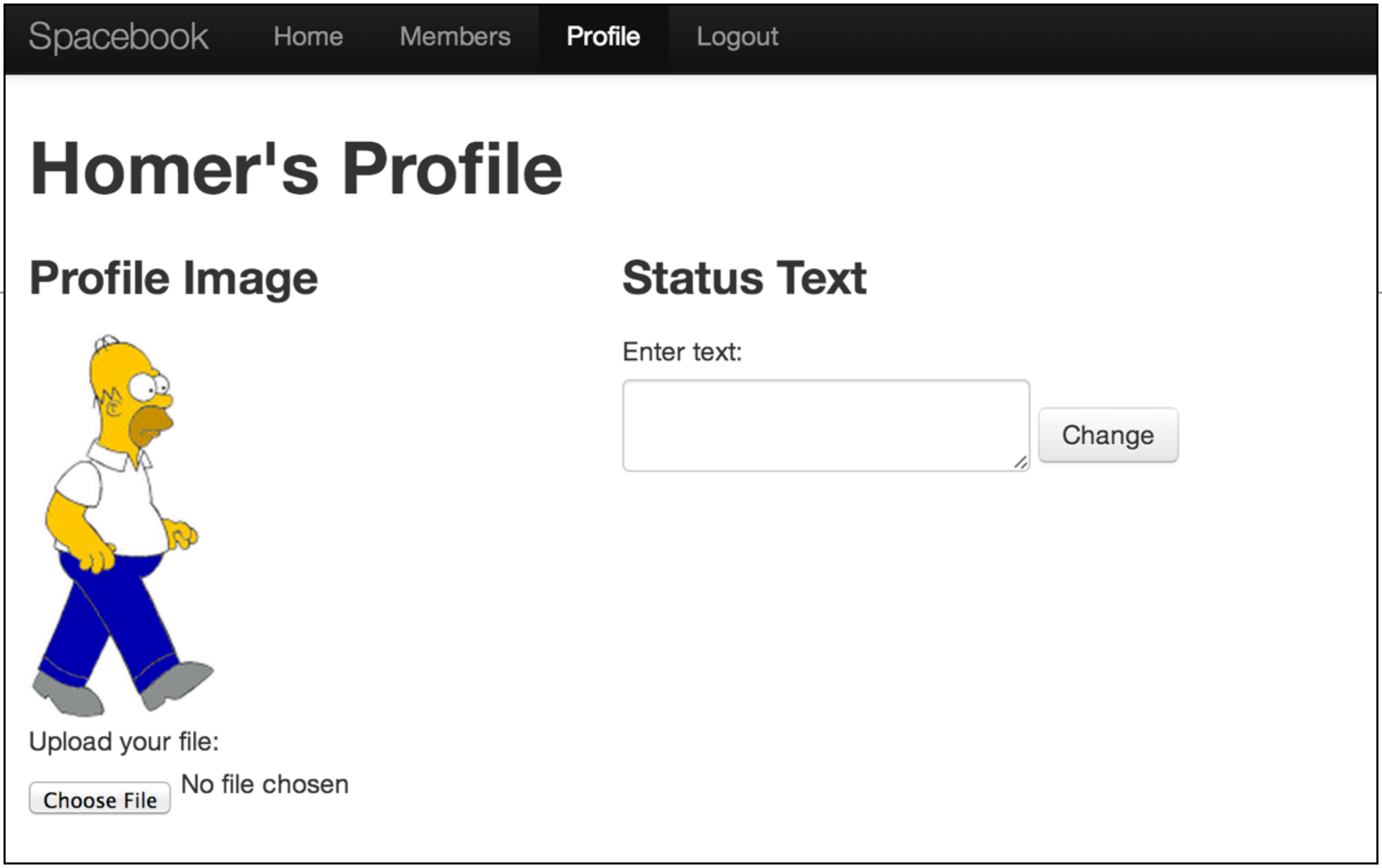
controllers/Home.java

```
public static void index()
{
  String userId = session.get("logged_in_userid");
  User user = User.findById(Long.parseLong(userId));
  render(user);
}
```

views/Home/index.html

```
<h2 class="ui header">SpaceBook: Home page for ${user.firstName} ${user.lastName}</h2>
```

- Assuming the user is ‘logged in’,
  - retrieve the user identity from the session
  - look up the database of users to get the user object
  - get the user name
  - pass the name to the view



Homer's Profile

Profile Image

Status Text

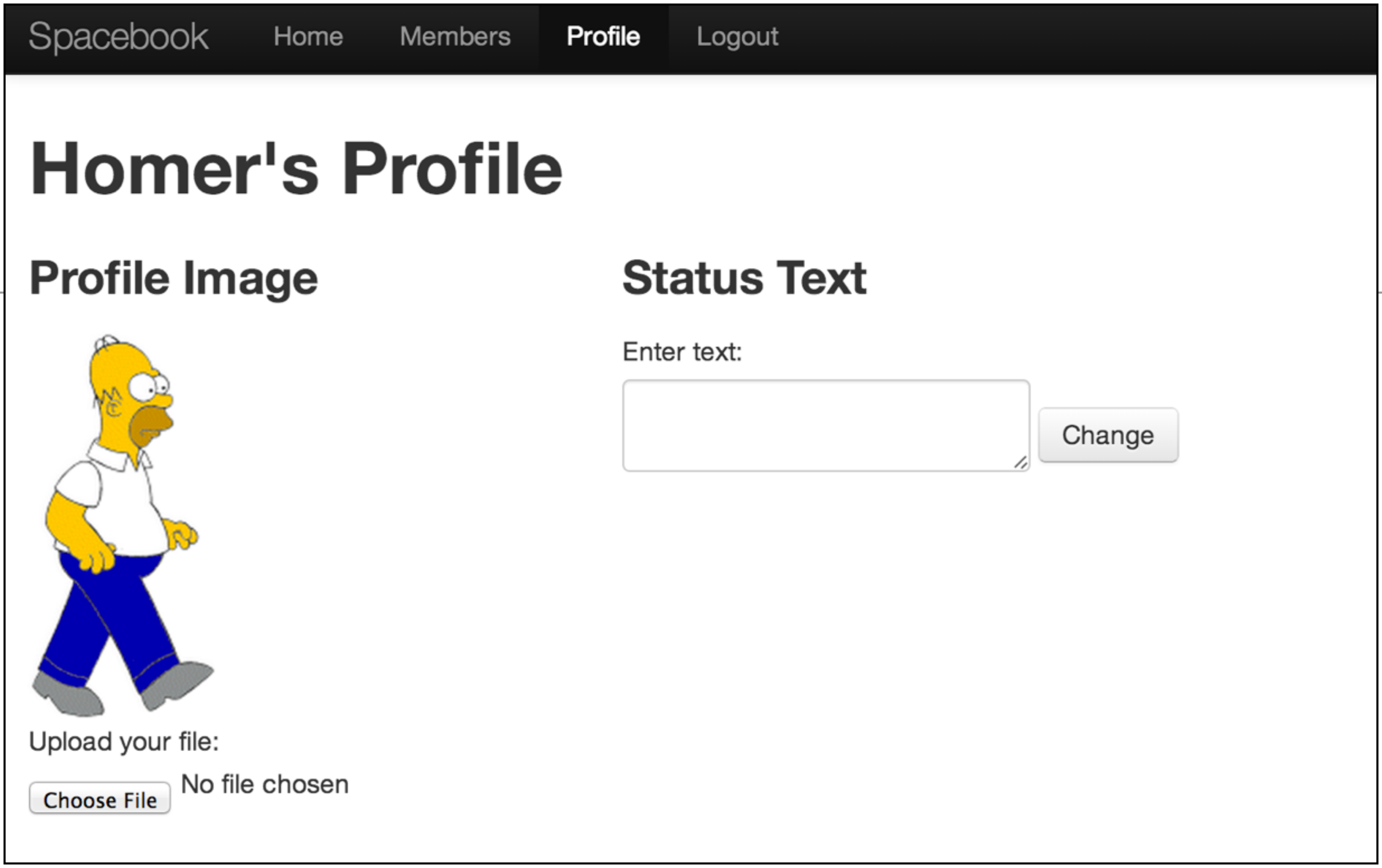
Enter text:

Upload your file:

Home
Profile Title
(hardcoded)

```
<h1>Homers's Profile</h1>
```

```
public static void index()
{
  render();
}
```



Homer's Profile

Profile Image

Status Text

Enter text:

Upload your file:

No file chosen

Home Profile Title (Dynamic)

```
<h1>${user.firstName} 's Profile</h1>
```

```
public static void index()
{
  String userId = session.get("logged_in_userid");
  User user = User.findById(Long.parseLong(userId));
  render(user);
}
```

Destroy the Session

- In the corresponding action, delete the session

```
public static void logout()
{
  session.clear();
  index();
}
```

- Any attempts to recover the information from the session object will fail

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning
support unit