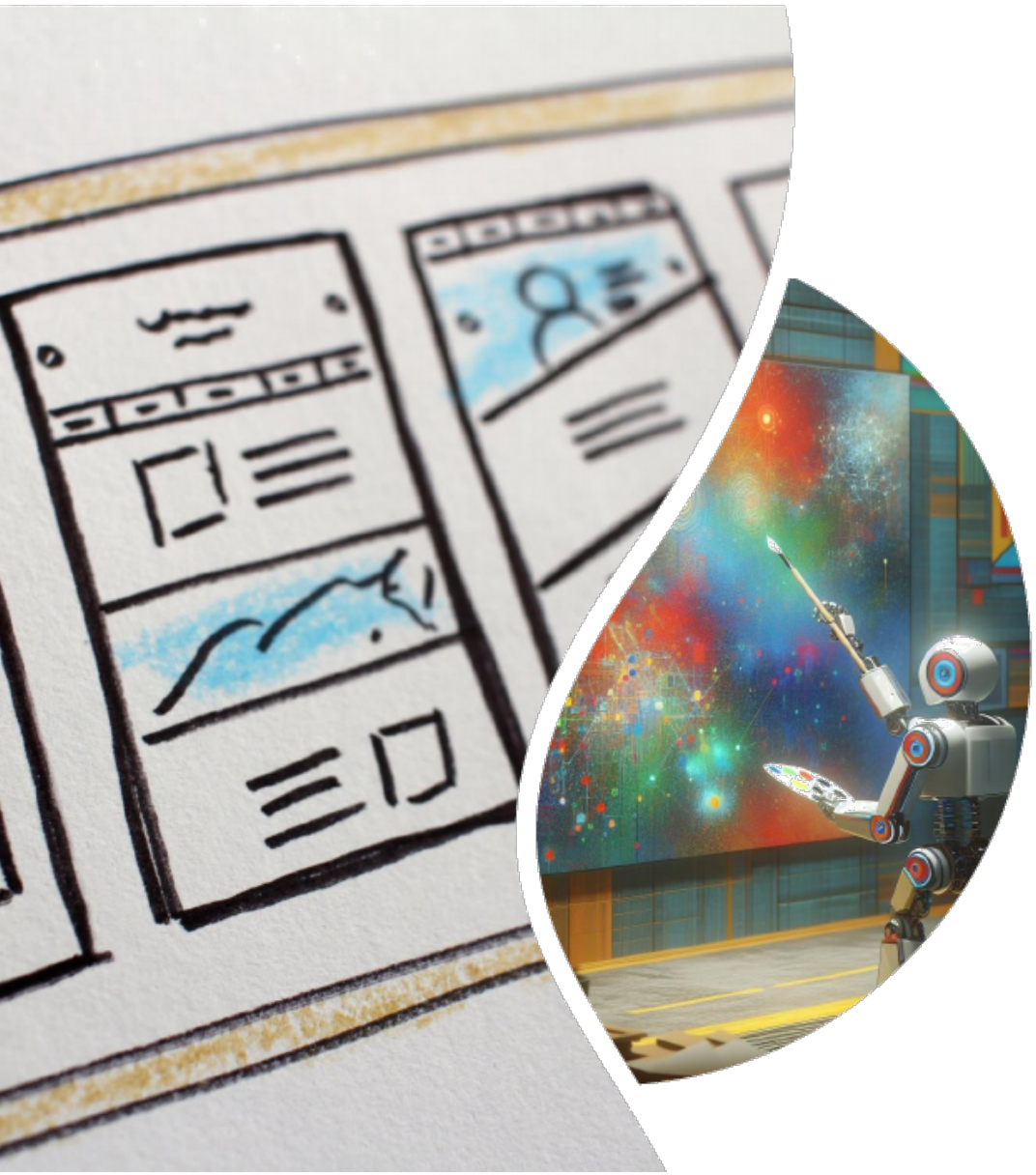


DGIN 5201
Digital Transformation
Lecture 4

Lec 3: Web Server Configuration, Password Protection

Vlado Keselj

Time and date:
13:05–14:25, 14-Jan-2025
Location: LSC C236

Image: DALL-E. Bing Image Creator. Generated by AI

Previous Lecture

- Technical foundations of digital innovation
- Evolution of the Internet
- Course Calendar Overview
- Unit description: Implementing a solution
- MVP, Rapid Prototyping, and Three-tier Architecture
- Starting points: CSID, timberlea, ssh and PuTTY
- Logging into timberlea server
- PuTTY, ssh (Secure SHell) connection
- Creating a simple web page
  - ssh, bash commands,
  - file permissions,

Notes and Announcements

- Lab 1 finished last Friday, due on Thursday by midnight
- Issues with copy-and-paste from PDF
  - tilde character (~)
  - other issues possible, such as quotes
- Solution: do not copy-and-paste but type
- Suggestion: learn touch-typing

Aside: Touch Typing

- If you don't use touch typing, consider learning it
- A relatively simple and not popular skill, but
  - actually important, and even more and more relevant
- Also known as *blind typing*, and *touch keyboarding*
- Reference: `https://en.wikipedia.org/wiki/Touch_typing`

Touch Typing

Touch Typing (2)

Image source of the previous slide:
`https://news.sophos.com/wp-content/uploads/2023/08/tt-1200.png`

Example e2: User Registration, Printable Page, Files Shared

- Consider a Conference Management System: *CoMS*
- Let us build a conference registration form
- We also want to provide them with some material
- First iteration: Create a printable form
- Create directory `public_html/dgin5201/e2`
- Go to that directory
- Add file `index.html` (content to be given)
- Make sure that the permissions of e2 are `rwx--x--x`, and of `index.html` are `rwxr--r--`

Example 2: public_html/dgin5201/e2/index.html

```
<html><head><title>Conference Registration</title></head>
<body>
<h1>Conference Registration</h1>

<p>This is a registration page for CoMS.<br/>
For additional documents, please check <a
href="material">here</a>.<br/>
Please enter your information below to register:

<table>
<tr><th align=right>First and last name:</th>
<td>_____</td></tr>
<tr><th align=right>Email:</th>
<td>_____</td></tr>
<tr><th>Area of Interest (DB, HI, DS):</td>
<td>_____</td></tr>
</table>
```

Example 2: Make material available

- Create readable and accessible ('executable') directory `material` (permissions: `rwxr-xr-x`)
- Copy PDF from: `~vlado/public/dt-mini-conf.pdf` into directory `material`
- Setup permissions for the directory `material` to be all readable and accessible (`rwxr-xr-x`), and for the file `dt-mini-conf.pdf` to be all readable (`rw-r--r--`)
- Try to access material link on the page. Does it work? Why?

Example 2: Prepare `.htaccess` in `material` directory

- Prepare file `.htaccess` and make it all readable (`rw-r--r--`):

```
Options Indexes
```

- Check material access now
- Add the following line to `.htaccess` and try accessing again:

```
Options Indexes
AddDescription "DT Conference Poster (PDF)" dt-mini-conf.pdf
```

- Add "and Information" to "DT Conference Poster" and access
- Add the following line and try again:

```
Options Indexes
IndexOptions DescriptionWidth=*
AddDescription "DT Conference Poster..." dt-mini-conf.pdf
```

- `.htaccess` file is used to configure Apache web server behaviour
  - can be used to provide a simple password-protected access

Concepts Review: Example 2

- Creating something that looks like form when printed
- HTML tags: `head`, `title`, `h1`, `p`, `br`, `a`, `table`, `tr`, `th`, `td`
- HTML attribute: `<th align=right>` `<a href="...">`
- `bash` shell: `cp`, using path, `~vlado`
- Accessing directory via browser
- `.htaccess` file for the Apache server: `Options Indexes`, `AddDescription`

Example e3: Password Protection

- Let us make a copy of our e2 site
- First, go back to the directory above e2:

```
cd ../..
```

- Use command `pwd` to check your directory
- Copy e2 to e3 as an exact copy:

```
rsync -av e2/ e3/
```

- Check the new site e3 in the browser
- `rsync` is a very useful utility for copying directory structures
  - it works locally as well as over ssh
  - it copies incrementally differences, which is important if two sites are large and mostly equal
  - it may preserve permissions if we use option `-a`

Example 3: Simple Password Protection

- cd to e3 directory and let us prepare a password
- In a locally-only readable file `pw` (`rw-------`) we can save a password for our reference: `dt dt5201`
- Prepare the password for the site using the command:

```
htpasswd -bc .htpasswd dt dt5201
```

- Make the file `.htpasswd` all-readable and check its contents
- Prepare the file `.htaccess` and make it all readable:

```
AuthType Basic
AuthName dgin5201
AuthUserFile /users/webhome/<your_csid>/dgin5201/e3/.htpasswd
AuthGroupFile /dev/null
<Limit GET POST>
require user dt
</Limit>
```

- Check that site is password-protected

Summary of e3

- Files and permissions copied from `e2`
- `pw` file with permissions `rw-------`
- `.htpasswd` file with permissions `rw-r--r--` and appropriate content set up with the `htpasswd` command
- `.htpasswd` file with permissions `rw-r--r--` and content set up for password protection as given in class

Concepts Review: Example 3

- `rsync` command, `-av` options
- `htpasswd` command, password saved as hash
- Using `.htaccess` for password-controlled access

Unix-style Customization

- Unix-style customization is typically text-based
- Example: bash customization
  - aliases: rm, mv, cp, em
  - .profile and .bashrc files
- Example: Emacs customization
  - .emacs file
- Earlier example: Apache customization
  - .htaccess, .htpasswd files

Example e4: Introducing a Form

- With `rsync` copy e3 to e4, update .htaccess file
- Change the table part of `index.html` to:

```
<form>
<table>
<tr><th align=right>First and last name:</th>
<td><input type="text"></td></tr>
<tr><th align=right>Email:</th>
<td><input type="text"></td></tr>
<tr><th>Area of Interest (DB, HI, DS):</td>
<td><select><option>DB</option><option>HI</option>
  <option>DS</option></select></td></tr>
 </table>
 </form>
```

- Check the page and see that this is usable fillable form, which can be printed

Concepts Review: Example 4

- Creating fillable form in HTML: `<form>...<form>`
- `<input type="text">`
- `<select><option>op1</option>...</select>`

Summary of e4

- Files set up as in e3
- `index.html` modified to make a usable fillable form

Back-end Processing

- How to
  - receive data at the server end from the client (browser)
  - process data and send some results back to the client
- This is called Back-end processing
- Some options to implement:
  - Apache has a way to help:
    - CGI — Common Gateway Interface
  - Use a server: build one or use server-based options

CGI Processing

- CGI — Common Gateway Interface
- Implemented as CGI program; e.g.: `prog.cgi`
- When requested, executed by Web server and output returned
- Input prepared by Web server

Example 5: Backend Server Processing using CGI (started)

- Using `rsync` copy `e4` to `e5`
- Let us first check that CGI scripts are working by creating file `test.cgi` in e5 as follows:

```
#!/usr/bin/perl
use CGI qw/:standard/;

print header;
print "<html><body>Test\n";
```

- The file should be user executable, without permissions to the group and others (`rwx------`)
- Run the command `./test.cgi` and you should get a simple output as follows:

```
Content-Type: text/html; charset=ISO-8859-1

<html><body>Test
```

- Check in browser: `https://web.cs.dal.ca···dgin5201/e5/test.cgi`

Example 5: Backend Server Processing using CGI (continued)

- Using `rsync` copy `e4` to `e5`
- Let us first check that CGI scripts are working by creating file `test.cgi` in e5 as follows:

```
#!/usr/bin/perl
use CGI qw/:standard/;

print header;
print "<html><body>Test\n";
```

- The file should be user executable, without permissions to the group and others (`rwx------`)
- Run the command `./test.cgi` and you should get a simple output as follows:

```
Content-Type: text/html; charset=ISO-8859-1

<html><body>Test
```

- Check in browser: `https://web.cs.dal.ca···dgin5201/e5/test.cgi`

Perl Scripting Language

- What we saw in `test.cgi` program is example of a Perl program
- Perl is a scripting language, similar to Python and PHP
- Provides convenient and quick data preprocessing
- Text processing oriented
- Appropriate for rapid prototyping, and CGI programming

Example 5: Preparing form for processing

- Modify `index.html` the table part:

```
<form method="post" action="register.cgi">
<table>
<tr><th align=right>First and last name:</th>
<td><input type="text" name="name"></td></tr>
<tr><th align=right>Email:</th>
<td><input type="text" name="email"></td></tr>
<tr><th>Certificate (DB, HI, DS):</td>
<td><select name="certificate">
  <option>DB</option><option>HI</option>
  <option>DS</option></select></td></tr>
<tr><td align=center colspan=2>
<input type="submit" value="Submit"/></td></tr>
</table>
</form>
```

Example 5: Processing Data

- Prepare user executable file `register.cgi`:

```
#!/usr/bin/perl
use CGI qw/:standard/;
print header;
print "<html><body><h1>Registration</h1>\n";
print "<p>The following registration is received:\n";
$name  = param('name');  $email = param('email');
$certificate  = param('certificate');
print <<"EOT";
<table>
<tr><th align=right>First and last name:</th>
<td>$name</td></tr>
<tr><th align=right>Email:</th><td>$email</td></tr>
<tr><th>Certificate (DB, HI, DS):</td><td>$certificate
</td></tr><tr><td align=center colspan=2>
<a href="index.html">Back to Registration Page</a></td>
</tr></table>
EOT
```

Example 5: Processing Data and Testing

- Submit some registrations and make sure `register.cgi` works well
- This completes Example 5 (e5)

Concepts Review: Example 5

- Server-side processing, concept of CGI (Common Gateway Interface)
- Perl programming language, Perl with CGI
- `<form method="post" action="...">`
- `<input ... name="x">`
- `<input type="submit" value="Submit/>`
- CGI processing in Perl

Example 6: Saving Registration Data: Implementation

- Using `rsync` copy `e5` to `e6`; adjust `.htaccess`
- To save registration, add the following line in the script `register.cgi`:

```
...
$email = param('email');
$certificate = param('certificate');

&save_registration($name, $email, $certificate);

print <<"EOT"; ...
```

- and we add the following function at the end of the program:

```
sub save_registration {
  my ($name, $email, $certificate) = @_;
  open (my $fh, ">>registrations-saved.txt") or die;
  print $fh "\nname: $name\nemail: $email\n".
    "certificate: $certificate\n";
  close($fh);
}
```

Example 6: Saving Registration Data: Testing

- First check syntax: `perl -c register.cgi`
- Test the web site by making several registrations
- Check that registrations are saved in the file `registrations-saved.txt`
- Check permissions of `registrations-saved.txt`
  - If not all-readable, make them all-readable
  - Verify that the file is accessible on the web **(!)**
- Change the permissions of `registrations-saved.txt` to user-only readable and writeable
- Check accessibility on the web; **Lesson learned!**
- Check that the application still works

Concepts Review: Example 6

- Perl subroutine (similar concepts: procedure, function)
- Saving and appending data to a file
- Importance of file permissions
- Possible additional issues to deal with files: concurrency (race conditions), efficiency
- Alternatives: using databases, server or file-based

Example 7: Sending Registration by Email (started)

- Use `rsync` to copy e6 to e7
- Modify the `register.cgi` file as follows by adding a new line:

```
...
&save_registration($name, $email, $certificate);
&send_email($name, $email, $certificate);
...
```

- and add the following subroutine at the end of the file:

```
sub send_email {
  my ($name, $email, $certificate) = @_;
  my $emailmessage = "To: vlado\@dnlp.ca\n".
    "Subject: New registration\n\n".
    "A new registration is received as follows:\n\n".
    "name: $name\nemail: $email\n".
    "certificate: $certificate\n";
  open(my $s, "|/usr/lib/sendmail -ti") or die;
  print $s $emailmessage;
  close($s);
}
```

Example 7: Sending Registration by Email (continued)

- Use `rsync` to copy e6 to e7
- Modify the `register.cgi` file as follows by adding a new line:

```
...
&save_registration($name, $email, $certificate);
&send_email($name, $email, $certificate);
...
```

- and add the following subroutine at the end of the file:

```
sub send_email {
  my ($name, $email, $certificate) = @_;
  my $emailmessage = "To: vlado\@dnlp.ca\n".
    "Subject: New registration\n\n".
    "A new registration is received as follows:\n\n".
    "name: $name\nemail: $email\n".
    "certificate: $certificate\n";
  open(my $s, "|/usr/lib/sendmail -ti") or die;
  print $s $emailmessage;
  close($s);
}
```

Example 7: Sending Registration by Email (2)

- IMPORTANT: Instead of string `vlado@dnlp.ca` use your own email
- No not forget to use backslash (\) just before the at-sign (@) in email, as in `vlado\@dnlp.ca` because the string is delimited by double-quotes. Otherwise, Perl will replace `@dnlp` with the value of that array
- Test the program and make sure that you receive email after each registration

Example 7: Received Email

- If everything is implemented correctly, and if it works, you should receive an email similar to:

```
From: "...your name..." <YourCSID@willow.cs.dal.ca>
Date: Tue, 13 Feb 2024 14:59:34 -0400 (AST)
To: your_email@dal.ca
Subject: New registration

A new registration is received as follows:

name: Test Name
email: test-email@cs.dal.ca
certificate: DB
```

Example 8: Testing Other Scripting Languages

- Copy e7 to e8 using rsync
- Update .htaccess to use passwords from e8/.htaccess
- Create files index-php.html and index-py.html to use PHP and Python as actions: register.php and register.py
- Implement basic register.php and register-py.cgi to print filled form

Example 8: Testing a PHP Script: register.php

```
<html><head><title>Applicant Registration</title></head>
<body>
<h1>Registration</h1>

<p>The following registration is received:

<table>
<tr><th align=right>First and last name:</th>
<td><?php echo $_POST['name'] ?></td></tr>
<tr><th align=right>Email:</th>
<td><?php echo $_POST['email'] ?></td></tr>
<tr><th align=right>Certificate (DB, HI, DS):</th>
<td><?php echo $_POST['certificate'] ?></td></tr>
<tr><td align=center colspan=2>
<a href="index-php.html">Back to Registration Page</a>
</td></tr></table>
```

Example 8: Testing a Python Script: register-py.cgi

```
#!/usr/bin/python
import cgi, cgitb
print "Content-type:text/html\n\n"
print "<html><body><h1>Registration</h1>\n";
print "<p>The following registration is received:\n";

form=cgi.FieldStorage()
name  = form.getvalue('name')
email = form.getvalue('email')
certificate  = form.getvalue('certificate')
print """<table><tr><th align=right>First and last name:</th>
<td>"""+name+"""</td></tr>
<tr><th align=right>Email:</th><td>"""+email+"""</td></tr>
<tr><th>Certificate (DB, HI, DS):</th>
<td>"""+certificate+"""</td></tr>
<tr><td align=center colspan=2>
<a href="index-py.html">Back to Registration Page</a></td>
</tr></table>\n"""
```

Example 8: Renaming Python Script to `register.py`

- We can copy `register-py.cgi` to `register.py` and try if it works (use `index-py2.html` as the index page)
- It does not!
- Solution: Add the following line to `.htaccess` file:

```
AddHandler cgi-script .py
```

Scripting Languages

- Developed as helpful tools for automating tasks, rapid prototyping, gluing together other programs
- Evolved into mainstream programming tools
- Examples
  - shell scripts (e.g., bash)
  - Early text processing: sed, Awk
  - Perl, PHP, Python, Ruby, Tcl, Lua, ...
  - Javascript
  - Visual Basic, VBScript, JScript, CScript, WScript,...
  - ...

Brief Overview of some Programming Languages

- (by Brian Kernighan)
- 1940's — machine language
- 1950's — assembly language
- 1960's — high-level languages: Fortran, Algol, Cobol, Basic
- 1970's — systems programming: C, but also Pascal
- 1980's — object-oriented: Smalltalk, C++
- 1990's — strongly-hyped: Java, modest beginning of JavaScript
- 2000's — lookalike languages: C#, PHP
- 2010's — retry? Scala, Go, Rust, Swift

Overview of Programming (Scripting) Languages

- 1940's — (machine language)
- 1950's — (assembly language)
- 1960's — Fortran, Algol, Cobol — Basic, Snobol
- 1970's — systems programming: C, Pascal — shell
- 1980's — OOP: Smalltalk, C++ — awk, Perl
- 1990's — Web: Java — Perl, Python, PHP
- 2000's — Frameworks: C# — JavaScript
- 2010's — retry? Scala, Go, Rust, Swift — Typescript

Typical Characteristics of Scripting Languages

- Interpreted
- Garbage collection
- Weakly typed; minimal use of types and declarations
- Text strings as an important data type
- Regular expressions support
- Easy execution of external programs

Technical Project Requirements

- Learning objectives and goals
- Choosing project topic
- Presenting a case for your project
  - preparing project specification
- Design and implementation choices
- Planing development

Project Learning Objectives

- Strenghen innovation and startup culture
- Learn to build a functioning three-tier system
- Learn fast Web-based prototyping
- Understand fundamentals of web technology
- Explain trade-offs between web-oriented programming languages and frameworks

Course Project Goals

- A simulation of practical software development, particularly in a start-up environment; this means:
  - building a substantial and useful system
  - working in a team (ideally 3–4 members)
  - having (hopefully) real users
- Building a complete three-tier system
  1. User interface (presentation tier)
  2. Processing logic ("business" logic, control tier)
  3. Database (persistent data, data access tier)
- Related but different than Model-View-Controller design pattern

How to Choose a Project Topic?

- Look at the examples of existing systems
- Find something interesting to you
- We will try to provide project ideas
- Define appropriate scope
- Discuss with other team members

Look at the Existing Systems

- There are many examples of web applications and services:
  - Google, Facebook, Twitter, Amazon, Instagram, LinkedIn, ...
  - email, chat, search, collaboration, maps, mobile apps, ...
  - standalone applications are okay, but may be challenging to recruit beta testers
- You should aim at something of this form, but much simpler and feasible for a term project

Find Interesting Topic *to you*

- The application should be useful to you, to start with
- A quote by Paul Graham, co-founder of Y Combinator (www.paulgraham.com):

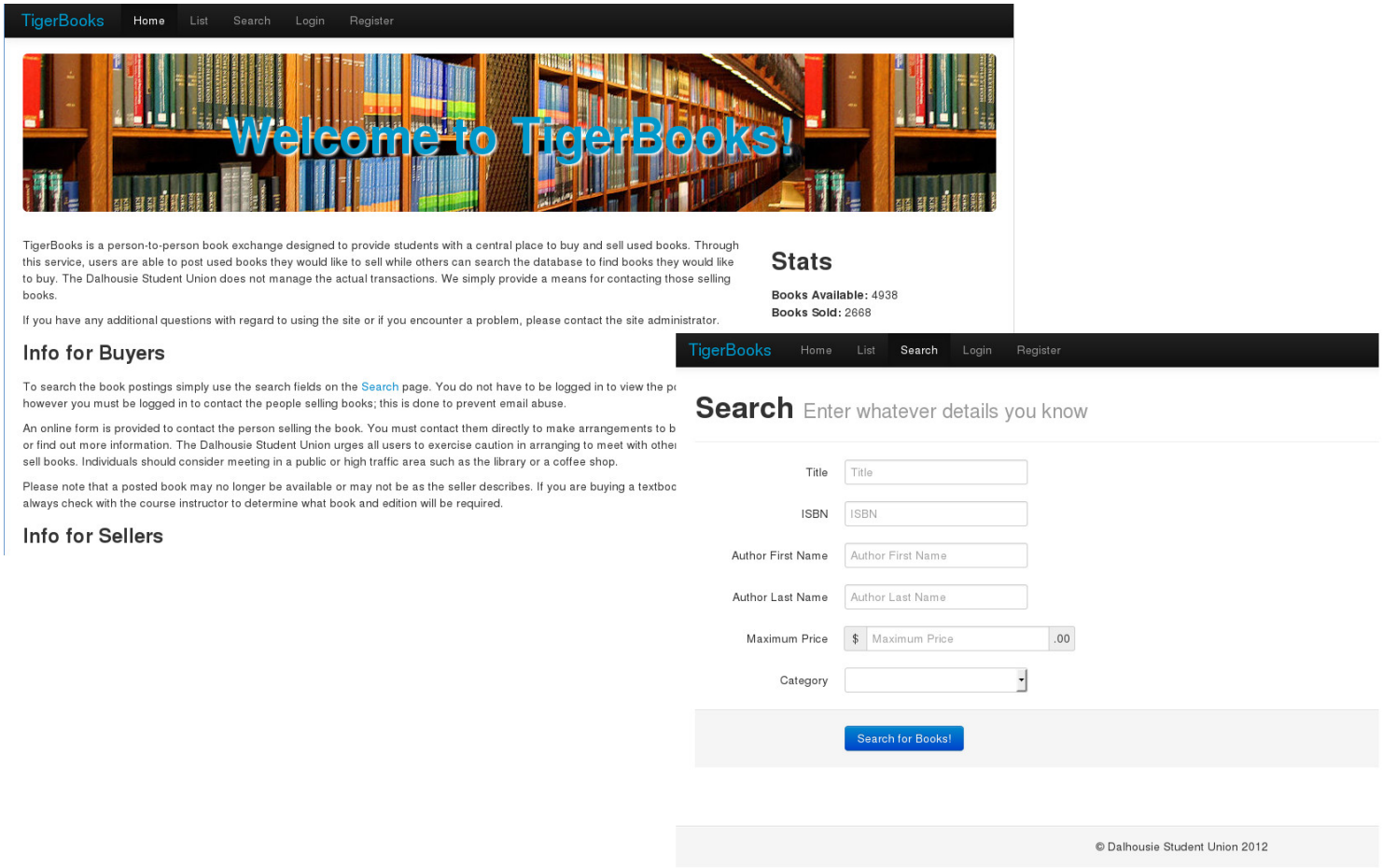
  "The way to get startup ideas is not to think of startup ideas. It's to look for problems, preferably problems you have yourself.
  The very best startup ideas tend to have three things in common: they're something the founders themselves want, that they themselves can build, and that few others realize are worth doing...

Define Appropriate Scope

- By the end of term (probably sooner), you need to have an MVP (Minimal Viable Product)
- There is not much time by the end of term, and you have other courses, so your product must be much simpler than typical interesting examples out there

A (Former) Local Example

- DaI TigerBooks



Presenting Strong Case for Your Project

Different approaches to think about your project:

1. Elevator speech: How would you make a case for your project to an investor during a 60sec elevator ride
2. 1-page ad: what would you put there; what information you would put on your landing web page
3. 5–7 slides for a 5–10min presentation of the project
4. Demo: what case scenarios you would include in a short demo to a client
5. Business plan: value proposition, competitive environment, revenue model and similar
6. Job interview: how would you talk about the project to a potential employer