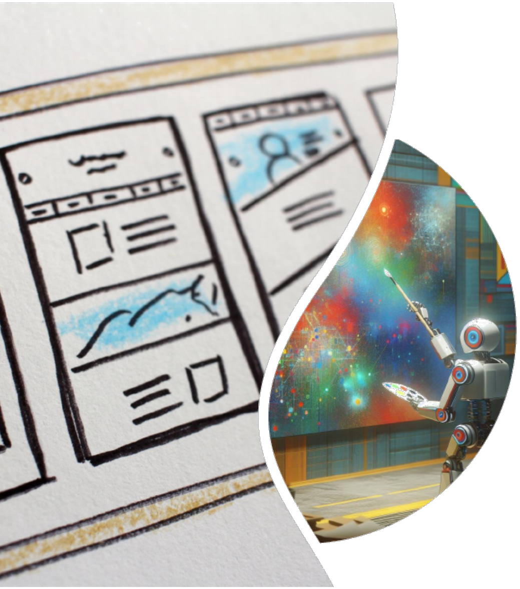

DGIN 5201
Digital Transformation
Lecture 11

Back-end Processing

Vlado Keselj

Time and date:
10:05–11:25, 24-Feb-2026
Location: McCain 2170

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

Previous Lecture

- Source Version Control
  - manual file saving
- Introduction to Git and GitLab
- FCS GitLab Login and course repository
- Login to timberlea and git preparation
- Setting local git repository
  - including e1, e2, e3, e4
- Connecting remote git repository
- Local git repository
  - init, add, commit
- Pushing e1, e2, e3, e4 to course repostiroy

Exercise e5

- Login to `timberlea`
- Change directory to `~/public_html/dgin5201/`

Example e5: Introducing a Form

- With rsync copy e4 to e5, 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 5

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

Summary of e5

- Files set up as in `e4`
- `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 e6: Backend Server Processing using CGI

- Using `rsync` copy e5 to e6
- Let us first check that CGI scripts are working by creating file `test.cgi` in e6 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/e6/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 e6: 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 e6: 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 e6: Processing Data and Testing

- Submit some registrations and make sure `register.cgi` works well
- This completes Example e6

Concepts Review: Example e6

- 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 e7: Saving Registration Data: Implementation

- Using rsync copy e6 to e7; 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 e7: 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 e7

- 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 e8: Sending Registration by Email

- Use `rsync` to copy e7 to e8
- 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 e8: 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 e8: 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: Thu, 23 Jan 2025 13:30:34 -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 e9: Testing Other Scripting Languages

- Copy e8 to e9 using rsync
- Update `.htaccess` to use passwords from `e9/.htpasswd`
- Create files index-php.html and index-py.html to use PHP and Python as actions: register.php and register-py.cgi
- Implement basic register.php and register-py.cgi to print filled form

Example e9: 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 e9: Testing a Python Script: `register-py.cgi`

```
#!/usr/bin/python
import cgi
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 e9: 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!   (i.e.., probably does not)
- Solution: Add the following line to `.htaccess` file:

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

Final Step: GitLab Submissions

- Remember that only e1–e4 are submitted to GitLab
- We need to submit e5–e9
- First, make sure that you are in directory: `~/public_html/dgin5201`
- Copy directories e5–e9 using the `rsync` command:

```
rsync -av e5/ git/e5/
```

- and similarly for the rest e6–e9
- all directories are now prepared in the `git` directory

GitLab Submission: e5–e9

- First go to the `git` directory using command:

```
cd git
```

- Add (stage) all new directories for git submission:

```
git add e5 e6 e7 e8 e9
```

- Commit all changes using the git command:

```
git commit -m"Added e5 to e9"
```

- And push the changes to the GitLab:

```
git push origin
```

- The log message does not have to be exactly to one given here
- You should check now your GitLab contents

Scripting Languages

- These are a couple of notes about 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