COMP 307

Principles of Web Development

Lecture 16

Unit 5 – Backend Design

Dynamic Multi-Paged Website (Part B)

Contents

Class Outline

- MVC vs. Observer design pattern
- Server communication
- Dynamic content (server-side query)
- Dynamic page generation

Readings

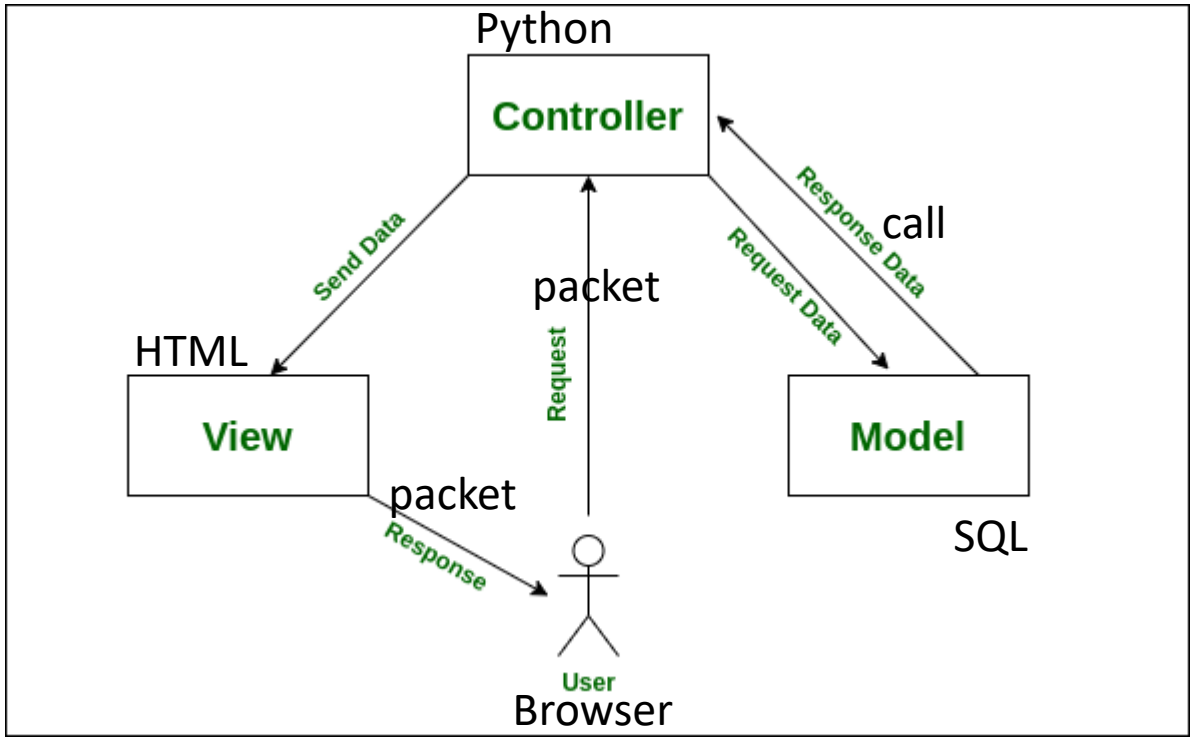
- WWW How to Program
  - Chapter 21
- Online resources
  - Synchronous and Asynchronous requests
    - https://www.w3schools.com/js/js_ajax_intro.asp
    - https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests
  - Dynamic Pages
    - https://www.tutorialspoint.com/internet_technologies/web_pages.htm
    - https://www.seomining.com/web-development/module6/dynamic-webPage-generation.php

Design Patterns

Dynamic Multi-Page Website (part B)

Contents

Model View Controller



MVC

- User = browser
- Controller = server API signature & code
- Model = database or CSV file
- View = response packet data structure (HTML)

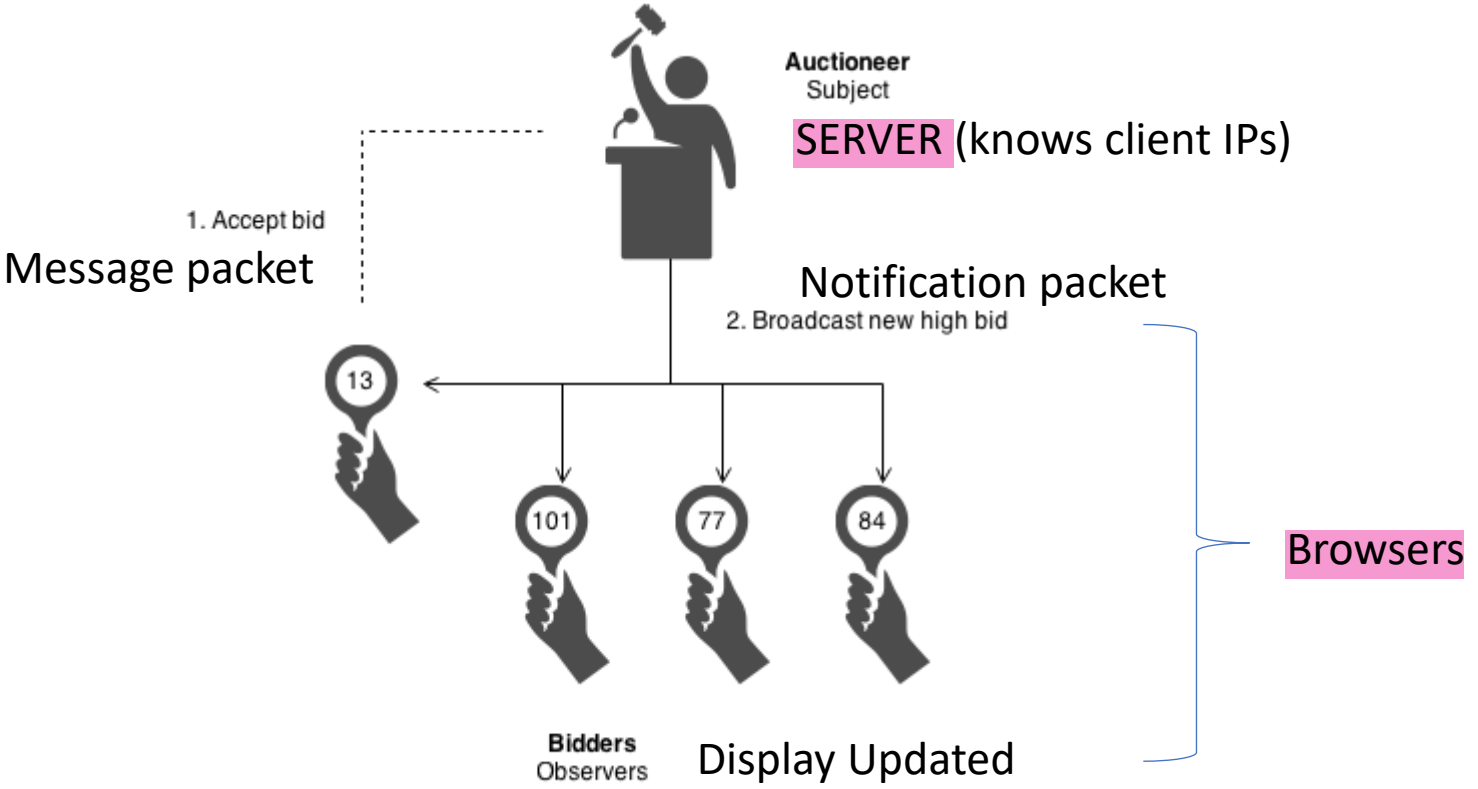
Model View Controller

- Benefits:
  - Each part of the pattern (M, V, and C) is self contained following rules that cannot be damaged by other parts.
    - User = browser
    - Controller = server API code
    - Model = database
    - View = packet data structure (HTML)
  - A convenient way to manage different technologies in the stack.
    - Input rule – processing – output rule
    - Public signature – private function – public string

Dynamic Content and MVC

- Architecture:
  - Client-side skeleton app
  - Server-side APIs
  - Database of content indexed by user (or other)
- Flow:
  - Skeleton calls API to extract data from database to populate skeleton data fields (s)
  - Skeleton auto calls API, or skeleton calls API only after a user interaction

Observer Design Pattern



- Server is in control of the dynamic content on client side.
- Client browser receives a notification as either a popup or content updated on the user's screen.
- Client can optionally respond to this UI change.

Observer Design Pattern

- Requires a thread to be running in the background of the client browser continually listening for a packed from the server.
- To protect the client the packet is either encrypted or uses a special ID number called a Valid Ticket number. The client verifies the validity of the pack (is it from a trusted source).
- Observer's (browsers) need to "register" with the "subject" (server).

We are only looking at MVC today.

Server Communication

Dynamic Multi-Page Website (part B)

Contents

Server Communication

- Synchronous
  - When the browser freezes and waits for the reply
  - Example:
    - **CGI** (built-in synchronous), e.g., <form>
      - Result, the entire page is replaced with new content
    - **Ajax** has a synchronous mode when selected
      - Entire webpage does not need to be replaced using callback JS
- Asynchronous
  - Browser is still usable after request is sent
  - Example: **jQuery lib**, **Ajax**
  - Result, using JS, only a portion of the website is updated using a JS callback function (a thread that waits for a packet) (note: observer pattern)

Synchronous CGI <form>

```
<!DOCTYPE html>
<html>
        <body>
                <h2>HTML Forms</h2>
                <form action="/action_page.php“ method=“post”>
                        <label for="fname">First name:</label><br>
                        <input type="text" id="fname" name="fname" value="John"><br>
                        <label for="lname">Last name:</label><br>
                        <input type="text" id="lname" name="lname" value="Doe"><br><br>
                        <input type="submit" value="Submit">
                </form>
        </body>
</html>
```

The entire webpage will be replaced by the output from action_page.php

Asynchronous Ajax

XML is used to encode the packet's payload

```
<!DOCTYPE html>
<html>
  <body>
        <div>
              <h1>The XMLHttpRequest Object</h1>
              <button type="button" onclick="loadDoc()">Change Content</button>
        </div>
        <p id="demo"></p>

        <script>
              function loadDoc() {
                 var xhttp = new XMLHttpRequest();

                 xhttp.onreadystatechange = function() {
                    if (this.readyState == 4 && this.status == 200) {
                       document.getElementById("demo").innerHTML = this.responseText;
                    }
                 };

                 xhttp.open("GET", "ajax_info.txt", true);
                 xhttp.send();
              }
        </script>
  </body>
</html>
```

The callback

The call

Only a single DOM unit is replaced

https://www.tutorialspoint.com/explain-the-different-ready-states-of-a-request-in-ajax

Ajax Ready States

- ReadyState == 0
  - **Unsent State**: send() method not yet called.
- ReadyState == 1
  - **Opened State**: sent to server and waiting for response.
- ReadyState == 2
  - **Header Received State**: information about the response packet sent, like, content type, page length, date, etc.
- ReadyState == 3
  - **Loading State**: payload received and being read into the browser's buffer for processing.
- ReadyState == 4
  - **Done State**: Packet in buffer and can be returned to your JS program.

Error Messages

- **200** – no error
- **404** – not found
- **401** – not authorized
- **400** – bad request
- **403** – forbidden
- **408** – Request Time-Out
- **500** – internal server error

https://www.pixelfish.com.au/blog/most-common-website-errors/

Asynchronous Ajax

Method	Description
open(*method, url, async*)	Specifies the type of request *method*: the type of request: GET or POST *url*: the server (file) location *async*: true (asynchronous) or false (synchronous)
send()	Sends the request to the server (used for **GET**)
send(*string*)	Sends the request to the server (used for **POST**)

POST Example:

```
xhttp.open("POST", "demo_post2.php", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford"); // CGI format
```

setRequestHeader(*header, value*)	Adds HTTP headers to the request *header*: specifies the header name *value*: specifies the header value

What is jQuery?

- Is a JS Library made by Google
- It simplifies some JS programming
- Include it from google:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
</head>
```

- Or, download it from jquery.com and then link to it:

```
<head>
<script src="jquery-3.6.0.min.js"></script>
</head>
```

Example

```
<!DOCTYPE html>
<html>
<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

<script>
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
</script>
</head>
<body>

<p>If you click on me, I will disappear.</p>
<p>Click me away!</p>
<p>Click me too!</p>

</body>
</html>
```

the library's syntax

All jQuery within the **ready()** event. To make sure page is loaded before execution.

Example

Ajax version

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

<script>
$(document).ready(function(){
  $("button").click(function(){
    $.get("demo_test.php", function(data, status){
                    alert("Data: " + data + "\nStatus: " + status);
    });
  });
});
</script>
</head>
<body>

<button>Send an HTTP GET request to a page and get the result back</button>

</body>
</html>
```

Ajax request

Call back function

It encapsulates all the Ajax calls into its own simplified syntax. (at the expense of a library inclusion)

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $.post("demo_test_post.asp",
    {
      name: "Donald Duck",
      city: "Duckburg"
    },
    function(data,status){
      alert("Data: " + data + "\nStatus: " + status);
    });
  });
});
</script>
</head>
<body>

<button>Send an HTTP POST request to a page and get the result back</button>

</body>
</html>
```

← part 1: URL

← part 2: send data **JSON** format

← part 3: returned data

call back function

COMP 307
Principles of Web Development

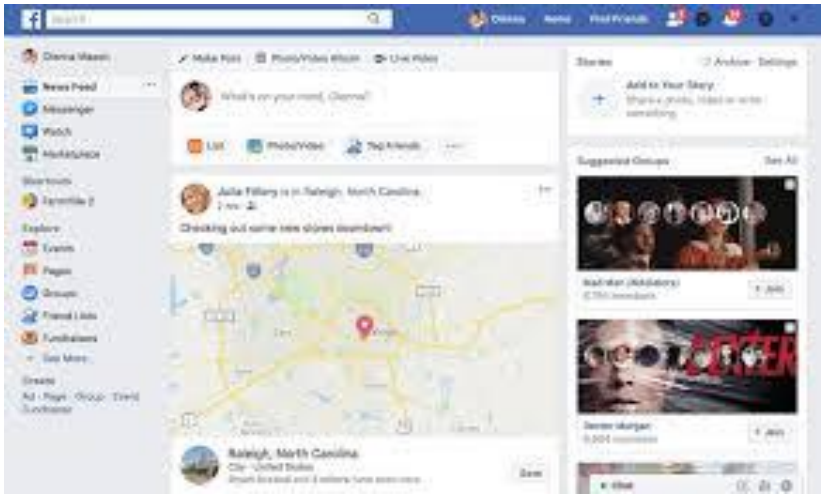
Dynamic Content

Dynamic Multi-Page Website (part B)

Contents

What is a dynamic web page?

- Synchronous and Asynchronous versions
  - Synchronous – entire webpage generated dynamic
  - Asynchronous – portion of webpage generated dynamic
- Displayed information is dependent on…
  - User query (literally, or as a page interaction)
  - User identity
- Example
  - Basic user layout same
  - Content identity based



Architecture

```
<html>
        <body>
                : html
                : html

                sql = “select * from x where user=‘username’”
                insert html based on sql result

                : html
                : html
        </body>
</html>
```

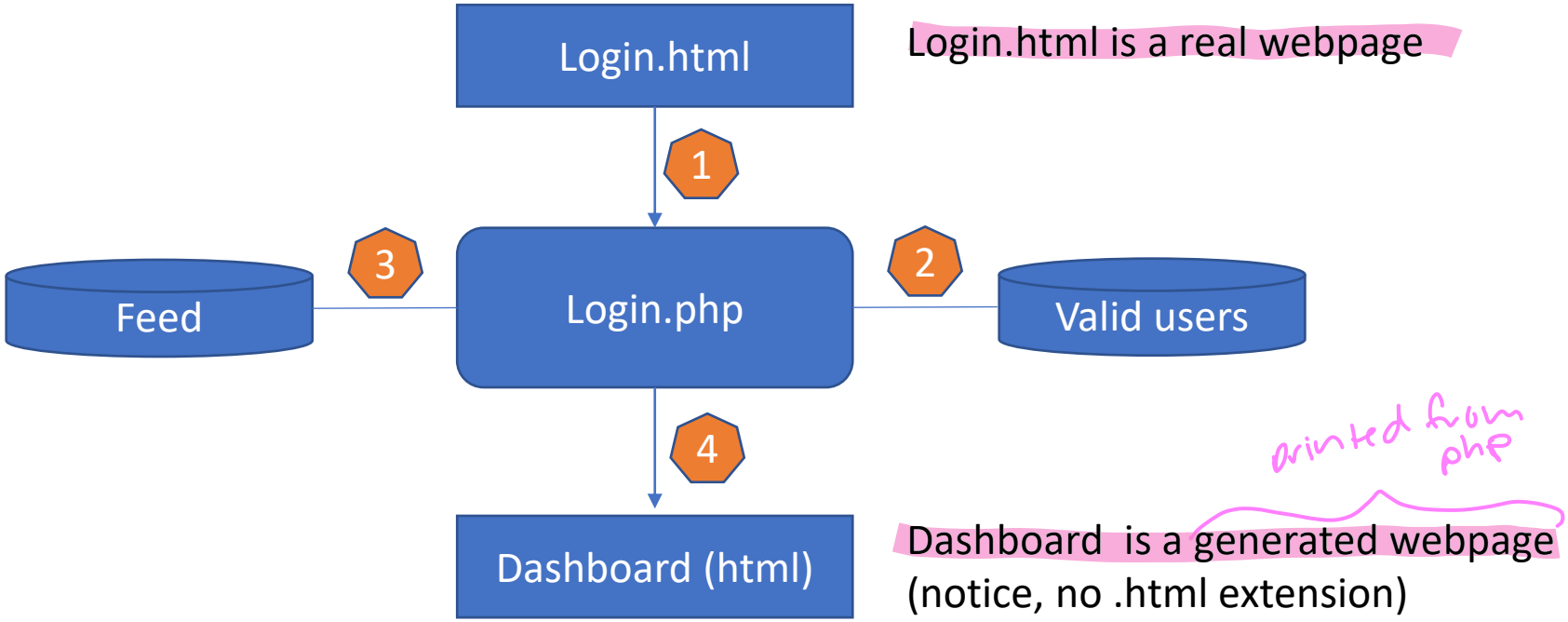
Where:

- The :html is regular HTML code
- Page is modularized into page header, page footer, menu and content area.
- Menu options and Content area are sql dependent.

Good or Bad?

- Benefits
  - Different web page content for same layout
  - Easy to program
  - Flexible
- Drawbacks
  - Loading an entire new webpage is more resource intensive (larger packets, more CPU processing time)
  - Each user query tends to be a heavy query, long server compute time (database calls), and a large response packet (the specific user content resulting from query)
  - **MITIGATION**: make the load large enough so that the user has content to look at for a while ~ amortization.

Example



Login.html is a real webpage

printed from php

Dashboard is a generated webpage
(notice, no .html extension)

The PHP program could have been C or Python as well.

2 Open users file, search, validate.

3 Open database, search username, collect information.

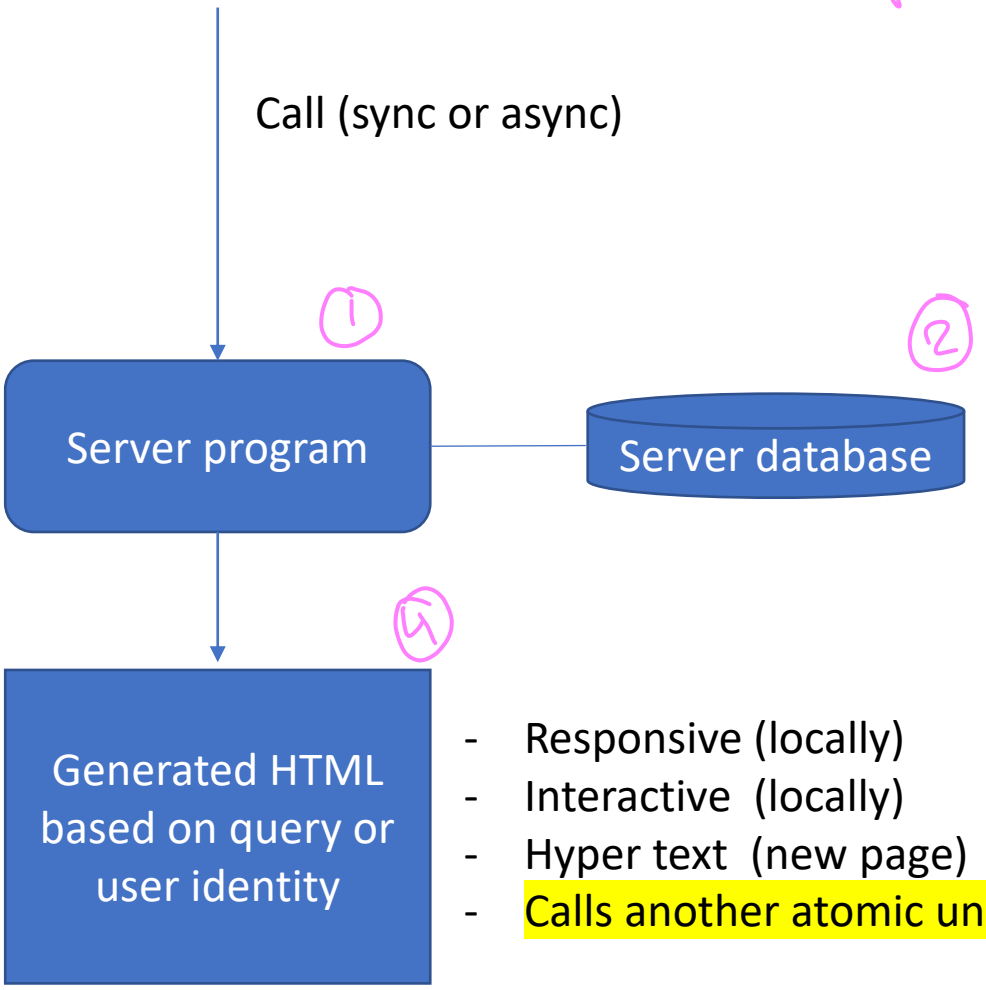
4 Populate an HTML doc with the buffer of information from step 3.

Note: php or C or Python.

Atomic Units

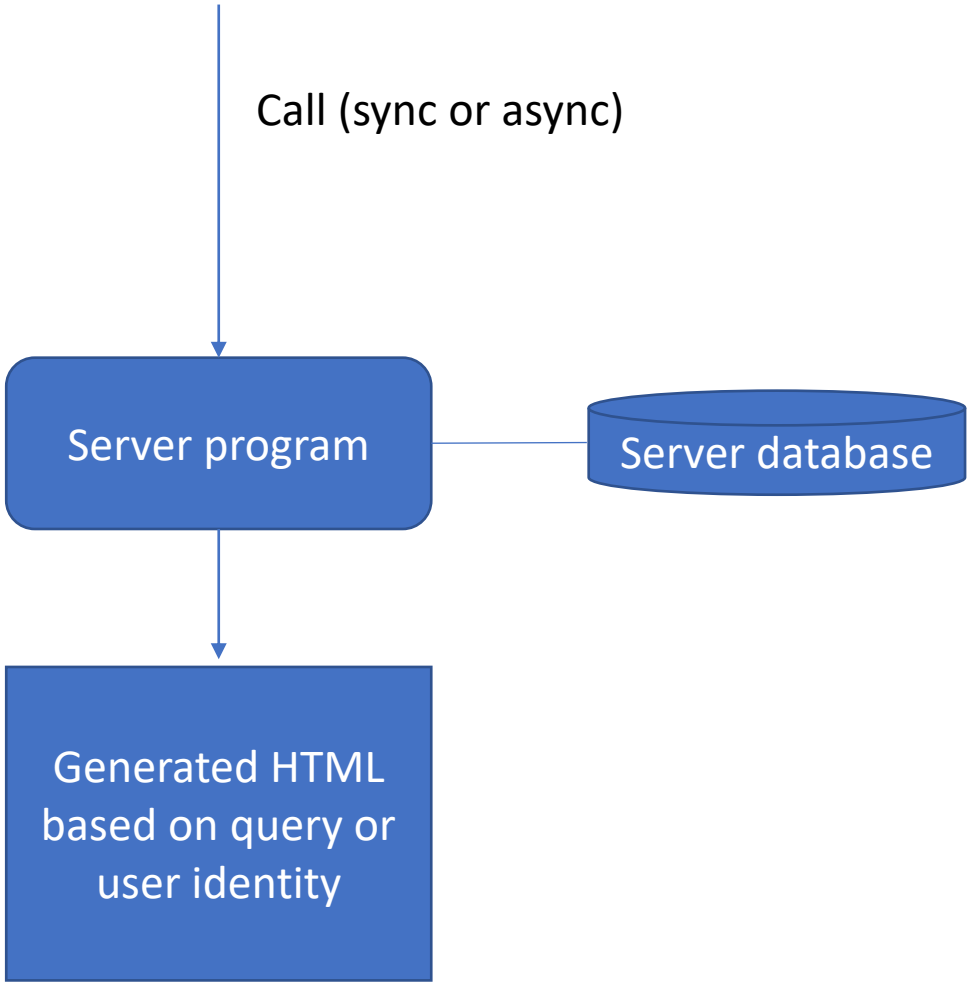
from server perspective

The website is interconnected through these atomic calls



Atomic Units

The website is interconnected through these atomic calls



(**Atomic Calls**) API Signature: URL/PATH/PROGRAM?QUERY_DATA

Heavy / Light Server Calls

- Heavy Server Call
  - Entire new webpage
  - Long server-side computation
  - More than one packet is required to store response
- Light Server Call
  - Since the webpage already exists, the server calls are only for specific data that fits within a single packet.
  - Data replaces a DOM element and not the entire page.

Case Study Examples

- Event based
  - Onload : specific page layout
  - Onhover : query server for popup window
  - Onscroll : query server for more feed information
- OnClick user interactions
  - New page - Display a specific layout
  - Same page - Perform a server query and display result innerHTML using an Async call.
- Programmatic reason
  - When data in array empty query server for more data

Full Page Generated

```
<html>
  <body>
        <h1><php? … validate login and say Welcome or Sorry…></h1>

        <div class="menu floatLeft">
            <php? … show menu options based on username using <li> <a>…>
        </div>

        <div class="content floatRight">
            <php? … show feed based on username use <li> …>
        </div>

        <div class="footer">
            …same HTML code for all users…
        </div>
  </body>
</html>
```

Webpage might have multiple <?php> for each dynamic area of the webpage

Notice it generates the new webpage using the php.

Partial Page Generated

```
<!DOCTYPE html>
<html>
  <body>
        <div>
              <h1>The XMLHttpRequest Object</h1>
              <button type="button" onclick="loadDoc()">Change Content</button>
        </div>
        <p id="demo"></p>

        <script>
              function loadDoc() {
                 var xhttp = new XMLHttpRequest();

                 xhttp.onreadystatechange = function() {
                    if (this.readyState == 4 && this.status == 200) {
                       document.getElementById("demo").innerHTML = this.responseText;
                    }
                 };

                 xhttp.open("GET", "ajax_info.txt", true);
                 xhttp.send();
              }
        </script>
  </body>
</html>
```

The callback

The call

async
false => synch

Generating Web Pages

Dynamic Multi-Page Website (part B)

Contents

What is dynamic page generation?

- A website that is created from a single server-side program and a "database" of content
  - The database can be:
    - SQL or No-SQL
    - CSV files
    - Code snippets ← looking at this today
    - A combination of the above

Single entry point of calls that generates all pages

Good or Bad?

- Benefits
  - Each page generator creates a different look
  - Easy to standardize the look of the site
  - Can reuse elements
  - Can add security
- Drawbacks
  - More server load
    - More CPU execution time
    - More hard disk space for website & languages

Demo

www.cs.mcgill.ca/~jvybihal

WINSCP server-side

Inspect with postman (if time permits)

index.php (1)

Notice the query string:

www.cs.mcgill.ca/~jvybihal/index.php?Page=About

context changes depending on this.

index.php (1)

```
<?php
//
// ----------------- MAIN PROGRAM -------------------------------
//
        // --------- COMMON WEBPAGE TOP ---------

        displayActive("matter/top_matter.txt",$_GET["Page"]);

        // --------- ROUTING WEBPAGE BODY -----------

        if (sizeof($_GET)==0 || $_GET["Page"]=="Home") {
                    // HOME PAGE
                    display("matter/home_matter.txt");
        } else if ($_GET["Page"]=="About") {
                    // INFO PAGE
                    display("matter/about_matter.txt");
        } else if ($_GET["Page"]=="Research") {
                    // RESEARCH
                    display("matter/research_matter.txt");
        } else if ($_GET["Page"]=="Hall") {
                    // HALL OF FAME
                    display("matter/hall.txt");
        } else {
                    // ERROR PAGE
                    echo "404: Invalid Page!";
        }

        // --------- COMMON WEBPAGE BOTTOM ----------

        display("matter/bottom_matter.txt");

// END MAIN
```

display contains the path

index.php (2)

```
// Prints a file into the packet positionally dependent

        function display($path) {
                $file = fopen($path,"r");
                while(!feof($file)) {
                        $line = fgets($file);
                        echo $line;
                }
                fclose($file);
        }
```

echoing happens in order

The function does not do this directly.

Instead, where it is called results in HTML placement.

Other supporting functions

index.php (3)

like changing active tab to red

```
// Prints a file to the packet, but switches class="Active" by target

        function displayActive($path,$target) {
                $file = fopen($path,"r");

                if (sizeof($target)==0) $target="Page=Home";
                else $target="Page=".$target;

                while(!feof($file)) {
                        $line = fgets($file);

                        if (strstr($line,$target))
$line=str_replace("class=\"empty\"","class=\"active\"",$line);
                        else
$line=str_replace("class=\"active\"","class=\"empty\"",$line);

                        echo $line;
                }
                fclose($file);
        }
```

Other supporting functions

index.php (3)

```
// Prints formatted announcements into packet positionally dependent

        function announcements($path) {
                $title=0;
                $body=0;

                $file = fopen($path,"r");

                while(!feof($file)) {
                        $line = trim(fgets($file));

                        if ($line == "== title ==") {
                                echo "TITLE: ";
                                $title=1;
                                $body=0;
                        } else if ($line == "== body ==") {
                                echo "BODY: ";
                                $title=0;
                                $body=1;
                        } else if ($line == "== end ==") {
                                $title=0;
                                $body=0;
                        } else if ($title == 1) {
                                echo $line."<br>";
                        } else if ($body == 1) {
                                echo $line."<br>";
                        }
                }

                fclose($file);
        }
```

matter

Demo:

- Top matter
  - Notice code starts the HTML but then is cut-off
- Body
  - Notice code is both cut-off at the beginning and the end
- Bottom matter
  - Notice code starts cut-off but ends the HTML

Index.php can pick and choose which code is inserted and the order it is inserted.

What is dynamic page generation? **(version 2)**

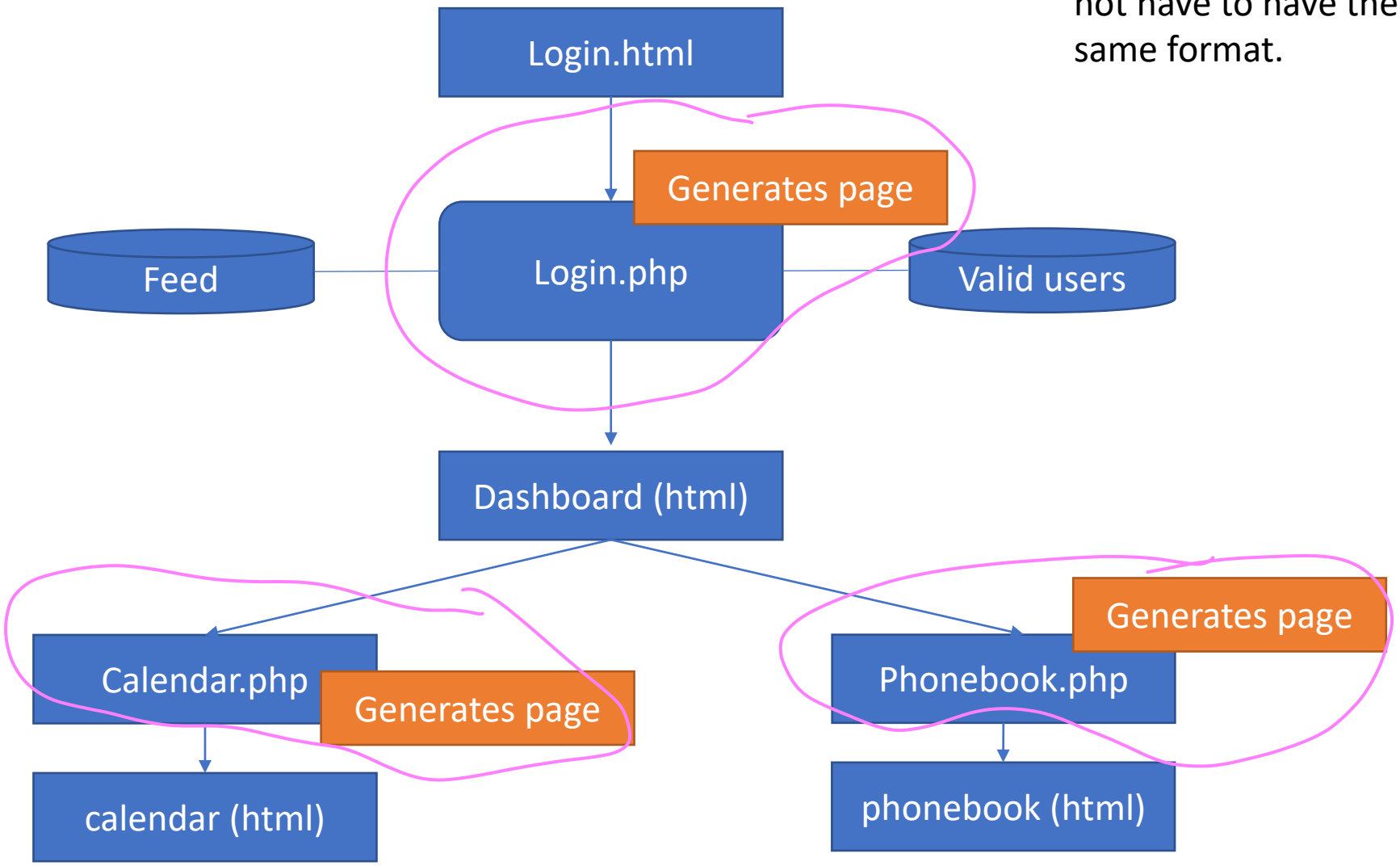
- A website that is created from a single server-side program and a "database" of content
  - The database can be:
    - SQL or No-SQL
    - CSV files
    - Code snippets
    - A combination of the above

Technically, your website can have more than one page generator like index.php.

Each one would support a **different layout** for different parts of the website.

Example Website

Note 1: php or C or Python.
Note 2: generated pages do not have to have the same format.



Prepare for Next Class

- Assignments
  - Mini 6 (how is it going)
  - Project handout on Nov 7
- No labs this week
  -
- Do on your own
  - Try out dynamic page generation using “matter” as seen in class.
  - Try out dynamic content with a browser skeleton page that calls a PHP program to return a sentence that updates a on your skeleton page.