ECS 162
WEB PROGRAMMING

4/26

Server-side programming

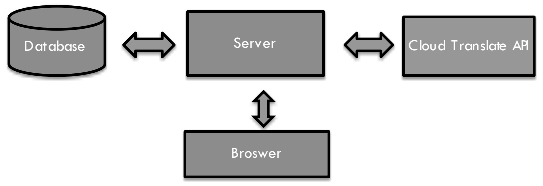
- aka "back end"
- Remainder of class organized around one large project, in three steps.
- This week, we'll do a sort of "hello world"
- After midterm, we'll get started in earnest.

- MIDTERM: Monday May 6, in class
- Uzair put sent out last year's midterm, also on "Labs" page

Flash cards

- Make cards
  - Use Google Translate API to translate from English to another language. Store flash card in database if user approves.
- Use cards
  - User sees other language text, tries to produce English.
  - System keeps track of how user did
- Pick some language with which you are at least a little familiar
- Can we use some transliteration input for non-Roman alphabet languages?

System design



- Now we write both server and browser code
- Database can be running on same machine as server, but behaves like it is on another machine.

This week

- Backend setup
  - Set up server
  - Serve static files (html, css, javascript, images…)
  - Due Thurs May 2
  - Not difficult; we will also be preparing for the midterm.

Design option

- If you want to do your own design for this project:
  - Jamie will give us some wireframes this week
  - You can produce design documents similar in quality and presentation to what we've seen from Jamie (fonts, colors, icons…)
  - Include about half a page of text on your design decisions
  - We will reject designs that show an insufficient level of professionalism

What is a server?

- Any computer on the internet running server software can be a Web server.
- A server recieves HTTP requests and produces HTTP responses.
- Example: OpenWeatherMap

Server

- A server handles static and dynamic requests.
- Static requests download a file, eg. HTML, CSS, Javascript, an image...this is called "serving pages"
- Dynamic requests answer queries, often in JSON; similar to what the OpenWeatherMap API server was doing for us.
- When a Web page sends queries to the server from which it was downloaded, these are AJAX queries (Asynchronous JavaScript and XML, even though often not in XML).

Our server

- We're using a cloud server from a company called Digital Ocean
- Our server has the elegant name:

  server162.site
- Our server is a Unix machine, like most (but not all) servers

Node.js

- Our server code will be written using node.js.
- Node.js is a way to run Javascript programs from the Unix command line, for example:

```
node serverOne.js
```

…runs the Javascript program in the file serverOne.js.

Alternative to Node.js

- The classic Web browser runs on the LAMP stack:

  Linux, Apache (Web server), MongoDB (database), PHP (scripting language).
- Node.js and our Javascript code replaces Apache and PHP. A server still needs an OS and, usually, a database.

Server modules in node.js

- Node.js also includes a set of Javascript modules that help us deal with problems like:
  - serving Web pages,
  - responding to AJAX queries,
  - querying specific APIs
  - and interacting with a database.
- Express is a node module with many server functions and an elegant interface
- We'll use express and several other modules.

Modules

- Modules are something like C or C++ libraries.
- A module is a file containing Javascript code.
- Objects, data and functions that programs that we want other files to see are labeled external.
- Modules provide another level of encapsulation and data hiding (in addition to functions and objects).
- Most modern browsers support them if served using CORS.
- Node.js is built around modules

Ports

- Each of us will be running our own server on the same machine at Digital Ocean.
- To direct incoming traffic to the right server, each of us will get a unique port number
- At the operating system level, a message comes in off the internet, and the system uses the port number to create an interrupt to send to the appropriate Web server

Server code at a lower level

- Mostly hidden by node.js
- We know Web server gets http requests and produces http responses.
- Server code is organized around producing http responses.
- Node.js servers are organized around a central object, usually named "response".

Accessing the server

```
ssh server162.site
```

- You should be able to login using your Kerberos account credentials
- Your port number will be in the file

```
myPortNumber
```

Simple Web server

```
const express = require('express')
```

- Brings in the express module.
- Must be installed before using

```
npm install express
```

- On the command line
- NPM is the node package manager

Handler function

```
function handler (req, res) {…
```

- All node.js servers use a handler function, which is called when a new request arrives at the server. Its job is to put together a response to the request.
- The request object (here req) contains information about the http request.
- We (the handler) use the response object (here res) to build up our response.

Typical handler structure

```
let url = req.url;
```

- Get whatever data we need out of request object
- Here url is the url that we received.

Sending the response

```
res.send('You requested '+url);
```

- Calling res.send() tells express that we have finished filling in the response object, and it is OK to send the response back to the browser.
- Once you call res.send(), the HTTP message is sent and you can't change the response object any more.
- The input to res.send is what to send back to the browser.

Asking express for a server

```
const app = express()
app.get('/*', handler )
```

- Calling express() creates a new application object
- It is traditionally called app
- We want it to handle HTTP GET requests
- We tell it to call the handler function whenever a GET request arrives, with any url (/ followed by anything)
- The handler function is kind of like a callback

listen

```
app.listen(port, function (){console.log('Listening...');} )
```

- This starts the server and tells node.js, Unix and TCP that requests that are labeled with your port should go to your server
- The server hangs, waiting to get a request
- I cannot emphasize too much that your server should listen to YOUR PORT NUMBER, not mine

Running and using the server

- On the server (Digital Ocean), run the simple server program:

  ```
  node miniServer2.js
  ```
- It should hang, waiting for input from the operating system
- From any browser, anywhere, request the URL
  http://server162.site:[your Port number]/anyPageNameYouLike
- Should get response:
  Hello! You asked for anyPageNameYouLike

Summary

- Typical overall node server structure
  1. Make a handler function
     a) In it, get data out of request object
     b) Then construct response
     c) Call app.send() when response is completed
  2. Create a server object specifying your handler
  3. Start it listening to YOUR PORT
  4. Web pages are now visible on the internet!