Programación III - Post Parcial

David Gabriel Corzo Mcmath

2019-09-17 07:12

Índice general

Capítulo 1

2019-09-17

1.1. API's & sus peculiaridades

- **Nos preguntamos:** ¿Por que razón puedo ver el XML en el browser? Es por que estamos usando el método GET.
- AJAX no permite no tiene la versitabilidad de parser. Este es el defecto de beautiful Soup, para el tipo de interacciónes con AJAX se necesita usar elenioum.

1.2. Flask

- Flase es un framework, no es como Django que uno es obligado a usar cirtas cosas oblgatoriamente, ***Definición de "framework":*** *es un set de herramientas.*
- Usualmente utiliza menos memoria usar "from ¡librería¿import ¡funciones o clases a usar¿".
- Está corriendo en un puerto.
- ***Definición de "Socket":*** *la combinación de una IP y un puerto.*
- Con hostname:

```
DAVIDCORZO@DESKTOP-73D7DE2 /cygdrive/c/Users/DAVIDCORZO
$ hostname
    DESKTOP-73D7DE2
```

- app.run(host="0.0.0.0",port=55) el host= 0.0.0.0. permite ver por medio de la red aplicaciones corriendo en otras computadoras.
- ***Definición de "puerto":*** *permite cambiar el socket, tiene un máximo de 65,535 puertos.*
- Debug igual True, uno de los beneficios que permite es correr la app sin tener que iterar el ciclo guardar,correr,ver_resultados, pero el chiste es debugging.
- En python "@" es un decorador, es una manera implicia de llamar funciones.
- Cambiémos la ruta con el decorador a "@app.route(/alumnos)"
- En flask hay dos formas de render: ***Definición de "Server side rendering":*** *fui a la base de datos y la respuesta solo al sabe mi aplicación, soy capaz por ende de imprimir información ingresada en mi aplicación.* ***Definición de "client side rendering":*** *este render lo hace el browser.*
- Lenguages de renderización, pintar de manera bonita en el browser.
- In jinja2 se referencia a una variable asi: "{{ ¡variable¿}}", para hacer un for loop {% for i in algo %}.

1.3. Averiguar después

1. decorators “@”
2. CSS Bootstrap
3. jinja2 en Flask
4. Server side render, client side render.
5. Leer Redis.

Capítulo 2

2019-09-19

2.1. Usar el kanban

- Con esto se puede registrar en GitLab la actividad hasta el momento.
- Metodolgía ágil.

2.2. Peculiaridades de Jinja

1. ***(Paréntesis "historial":****en bash o en shell se puede buscar el historial con el comando "history")*
2. Permite no replicar código
3. usar para un diccionario { %for students in ages.items() %}, posteriormente { %endfor %}, jinja intera en los elementos.
4. Como jinja no tiene el control sobre los elementos se usa {{loop.index}}.
5. Los parentesis con % es para if's, fors por ejemplo.
6. Puedo usar JS en mi página.

2.3. Gitlab

- "git checkout master", me saca del branch actual.

2.4. Buenas prácticas de programación

Agregar requirements.txt, if I add the flask library add the version.

- Correr "python -m pip install -r requirements.txt"

2.5. Dockerfile & .dockerignore

2.5.1. Dockerfile

Correr: docker build -rn -f "Dockerfile -t ¡nombre_de_archivo¿:latest"

EN

EL Dockerfile

```
FROM python:3-alphine

WORKDIR / <app>
COPY requirements.txt

RUN pip intall -r requirements.txt

COPY . ./

CMD ["Python","<app>.py>"]

EXPOSE 5000
```

2.5.2. .dockerignore

No incluye los archivos especificados,
Ejemplo:

EN EL .dockerignore

```
.vscode
Dockerfile
README.md
*.pyc
```

2.6. .gitlab-ci.yml

Sirve para compilar, para hacer pruebas dentro de GitLab, *Citación:"no aprendan GitLabCI"*, ***Definición de "CI":*** *Continous Integration.*

- ***Definición de "except master":*** *ejecutará el job en no master.*
- ***Definición de "only master":*** *ejecutará el job* ***solo*** *en master.*
- ***Definición de "CI":*** *permite probar e identificar fallas, es una metodología realmente para integración continual de mi código.*

EN EL .gitlab-ci.yml

```
#Jenskinsfile
#CircleCI
#Travis CI

stages:
    -test
    -build
    -deploy

<insertar un template de git lab>
```

2.7. Anuncios

Invitación al innovation del entreprenour.
Martes 1 octubre conferencia de Docker.
Peach competition.

2.8. Posterior a la clase investigar

1. html
2. css
3. bootstrap
4. navigation bar
5. jinja commands
6. CI
7. Dockerfile

Capítulo 3

2019-09-24

3.1. Invitado especial 8 de octubre

- Fullstack
- Monorepo

3.2. Buscar posterior a la clase

- Body request vs query parameters
- Typescript
- Esvelte
- React, Angular
- Selenioum
- Flask
- Logging

Capítulo 4

2019-10-01

4.1. Se trabajó en el projecto para el próximo martes

Capítulo 5

2019-10-08-Clase de UX

5.1. Diferencias entre UX/UI

5.1.1. UX

1. User Research
2. User Personas
3. User flow diagrams
4. Usability testing
5. A/B testing
6. Measurements

5.1.2. UI

1. Colors
2. Typography
3. Color constrast
4. Accesibility
5. High fidelity prototypes
6. Overall look and feel

(Paréntesis:*Entrarían html y css, si tenés tiempo.)*

5.2. Proceso

Hay varios métodos pero el más conocido y usado es el siguiente:

1. Research: una de las partes más importantes para determinar qué podemos hacer, evaluar cómo opera la competencia,
2. Sketched: Los bocetos, cuando uno hace el research se le ocurren ideas, estos son machotes, son una lluvia de ideas que se utiliza para descartar ideas, normal mente son a mano.

3. Wireframe: es una versión más refinada de los sketches, en los wireframes no se emite la versión terminada pero tenes que pensar más en lo realístico al producto final, vas a tomar en cuenta el dispositivo el cual tu app va a correrse.

4. Mockups: Es la versión terminada de los mockups, se define casi que nada, esta es la version final del producto.

5. Prototyping: el prototipo es agarrar todos los mockups que tenemos y prototiparlo.

6. Testing: esto es para medir qué tan buenos resultados están los resultados de los mockups, si no están al gusto del project manajer se repite la iteración.

5.3. Aplicación a diseño en la vida real

A continuación

1. Resource: qué tanto personal contamos con.

2. Requerimientos: definen el "qué hacer", son los requisitos básicos que tienen que tener el producto, a veces uno propone algo mejor pero hay que respetar que tenemos esos requisitos.

3. Deadline: uno empieza a economizar y ver si se puede saltar pasos en el proceso, probablemente saltar de sketches a mockups, por ejemplo.

4. Availability: quiénes van a estar disponible, hay feriados en la iteración, alguien va a estar de vacaciones.

5. Product type: qué estoy tratando de lograr desde lo que estoy diseñando, ***Ejemplo:****un botón, una pagina de checkout*

5.4. Buenos y malos ejemplos de UX

- Claridad en el producto, los parqueos y las señales de parqueo por ejemplo.
- ***Ejemplo:*** *Formulario, la gran lista de países.*
- ***Ejemplo:*** *Por ejemplo la eliminación de mensajes de whatsapp*

5.5. Trabajando con Project manajers y developers

Nos preguntamos: ¿Cómo trabajo con tantas personas sin hacer un gran problema?

1. Tener un timeline

2. Definir prioridades

3. Trabajar en equipo

4. Mantener a todo el equipo al tanto en todo momento

Teninendo un timeline y prioridades se puede evitar conflicto, **Nos preguntamos:** ¿qué pasa cuando dos productos prioritarios? ***La respuesta a esta pregunta es:*** *tenemos que priorizar sólo uno de primero, normalmente se hace esto entre PM y usualmente solo se le informa al developer que deje de hacer lo que está haciendo y que se trabaje en lo que está prioridades*

5.6. Herramientas

1. Figma: es una herramienta multi-plataforma, es administrador de versiones, tiene versión web, tiene plugins que ayudan a diseñar más rápido.
2. Sketch: sólo MAC, plug-ins, smart layouts, es pagado.
3. Adobe XD, MAC/Windows, plugins, es gratis.

5.7. Nos preguntamos: ¿Se puede programar y diseñar?

La respuesta a esta pregunta es: *Sí, es aún mejor tener los conocimientos para ser más ágiles a la hora de tener conflictos, tener en cuenta qué se puede hacer y qué no, entre más sepa de diseño mejor, mientras más sepa de programación mejor.*

5.8. Ventajas de diseñar y programar

1. Diseñar teniendo en cuenta la parte técnica.
2. No vamos a diseñar cosas complejas ni para los usuarios ni para los devs.
3. Diseño teniendo en cuenta la implementación.
4. Hablar el lenguaje de los desarrolladores.
5. Asegurarse que el diseño hecho quede igual una vez implementado.
6. Ofrecer ayuda.

5.9. Nos preguntamos: ¿Qué debemos hacer para programar?

1. HTML: estructura
2. CSS: estilo
3. JavaScript (un poquito solamente lo necesario), hay diseñadores que le tienen miedo a JavaScript.
4. Responsive device

5.10. Tres pilares de csss

1. Herencia: los hijos heredan los estilos de los padres, para ahorrar código.
2. Especificidad: hay elementos en nuestro código son especificamente modificados para esos elementos selectos para que no hereden las características estipuladas si no que se comporten de una manera diferente.
3. Cascada: dos clases que se llaman lo mismo y la que se va a aplicar va a ser la última.

Ver: EDteam.com

5.11. Nos preguntamos: ¿Qué es responsive Web Design?

1. Es básicamente condicionales, tener en cuenta qué dispositivos estarán usando la aplicación.
2. Una condicional que no afecte nada más que defina el comportamiento en diferentes dispositivos para acomodar bien todo según el dispositivo esto es para responsive, la condicional “media query”.

5.12. Resources

5.12.1. U-en-línea

1. Product design, udacity
2. design.io
3. learnux.io
4. platzi.com
5. ed.team
6. codigofacilito.com
7. udemy.com
8. udacity.com
9. youtube.com

5.12.2. Inspiración

1. dribble.com
2. behance.net
3. uplabs.com
4. material.io
5. pttrns.com

Capítulo 6

2019-10-10-Clase-Node.js

6.1. Node.js

Sirve para algunas cosas.

6.2. Historia de JavaScript

1. Fue inventado por Brendan Eich en 10 días, tiene similitudes con java pero solo fue una estrategia de java.
2. Fue innorvador tener la capacidad de un browser, JavaScript se llamaba moca, liveScript, JavaScript.
3. El proposito de esto fue ofrecer una experiencia interactiva, surgen los juegos.
4. Se requería plug-ins para JavaScript para usar calculadoras en línea por ejemplo.
5. Surge una gran infección de viruses que atentaba en contra de industria, era muy vulnerable a la seguridad y el punto de entrada de estos viruses por medio de los plug-ins.
6. Entonces como medida de seguridad se desarrollan nuevos browsers que aseguran que el código de las páginas web cambiara algo de tus archivos locales, pero se volvió un asunto que era incompatible todo con todo.
7. Como solución a la incompatibilidad surge AJAX (ASPX, JAVASCRIPT, AND, XML) & jQuery, entonces surge lo que se le llama "XML requests", al surgir esto surge el deseo en que la experiencia sean más interactivas, entonces surgen librerías de jQuery y esto permitía hacer una interfaz muy gráfica, el problema era que era muy lento.
8. 2008: The reckoning, surgen las "aplicaciones web", todo corria por medio de programas de Windows, surge J8 JavaScript Engine, V8 cambió las cosas, fue diseñada por la mismas personas que hicieron al java vritual machine, permitía ejecutar el código de máquina inmediatamente por medio de un interprete, como consecuencia JavaScript se vuelve rápido, surgen más browsers a raíz de esta innovación, windows saca su propio browser, y hasta el día de hoy se usa V8, V8 introdujo una nueva forma de ejecutar JS, permitía traducir a código de máquina inmediatamente por medio de un interprete.
9. 2019: Node.js, se empiezan a preguntar **Nos preguntamos:** ¿por qué no correr JavaScript directamente en el servidor? Sí, empezó a surgir, usualmente JavaScript corre en el browser.
10. Salió Node,
    - La prima razón para usar JS es por que hay muchos recursos, uno quiere hacer algo y ya probablemente se hizo.
    - Angular framework que compila JS.

11. Cuando JS salió habían solo Java, Apache, php, pero el modelo de programación de estos otros competidores de JavaScript eran malos, proceso:

    - Cuando se utilizaba un requests había una emisión de request procesaba todo pero en lo que me regresaba un resultado me quedaba esperando, entonces cuando una página tenía muchos requests se tenía que tener una capacidad de procesar esos treads y usualmente se volvía lentísimo, y a veces se caía el servidor, entonces se cambió el modelo de hacer las cosas.

6.3. Simplicity

1. Modelo:

Figura 6.1: Modelo de JavaScript, node.js

2. Se crea una capacidad de poder "atender mientras que esperan que algo pase", JS funcionaba así, se polpularizó con Node.js.

   a) Se volvió en un tread que se le delegó el nombre de "event loop sigle thread".

   b) Podía procesar varios requests al mismo tiempo por que tenía una sistema similar a el de una "sala de espera", en lo que me daba mi resultado me daba un "callback" y atendía al siguiente request.

   c) Consideración: si se computaba algo muy complicado se trababa el tread.

3. Modelo a detalle:

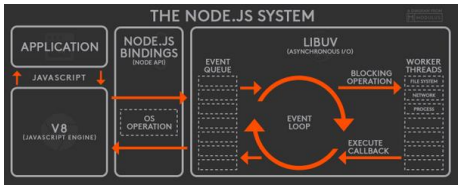

Figura 6.2: Modelo de node a detalle (aún más simplificado a comparación de lo que realmente pasa detrás de cámaras)

6.4. Razones para usar node.js

1. Short turnaround / fast prototyping,

   - ***Ejemplo:*** *Cada vez que se quería prototipar en un cambio se tardaba muchísimo en compilar y en correr que programador en c++, java; JavaScript soluciona eso ya que permite ver y porbar casi inmediatamente, es un lenguaje rápido.*

2. JavaScript Affordability:

- No se necesita linux, permite po ejemplo escribir una función en JS y subirla a algun cloud service que cobra por ejecución de la función.

3. Time to market is faster:

   - Se pueden utilizar lamba functions, subirlo al cloud es algo que rápidamente se prueba, ejecuta, y empezar a venderse en el mercado rápidamente.

6.5. Problemas con node.js & JavaScript

1. Cada cosita es un paquete:

   - No es como python que tiene una gran librería es super dependiente:

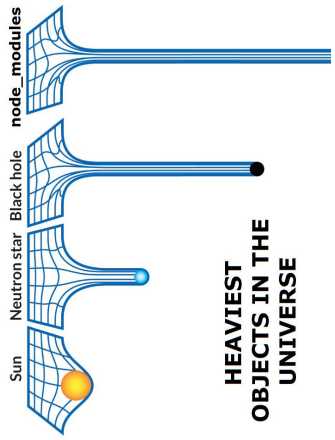

Figura 6.3: Meme de qué tan pesado son las librerías

   - El asunto es que la librerías que tenían dependencias y que node.js dependía de ellas, rápidamente se podía acumular hasta un gigabyte en librerías. Esto daba lugar a otras inconmutabilidades ya que hay librerías que son co-dependientes y cuando se actualiza alguna puede ser que se arruine otra librería que dependa de la librería actualizada.
   - ***Observación:*** *Mala práctica de Docker no incluir los modules en los paquetes de Docker.*
   - Se puede tardar un montón en instalar dependencias, usualmente media vez se corra una vaz el .log lo guarda en cache.

2. Utiliza mucho RAM:

   - V8 no estaba optimizado en absoluto para uso de RAM, V8 va intentar usar toda la Ram
   - Esto eventualmente se vuelve insostenible, este problema de memoria viene desde las épocas antiguas de C, a muchas personas se les olvidaba desocupar la memoria y lo mismo pasaba con node (denominados "memory leak"), en node es muy complicado encontrar "memory leaks", el garbage colector tiene límites.
   - ***Recordar lo siguiente:*** *Affordability, si se utiliza algo mucho pesado en términos de pesado, es mejor no considerar node.js y mejor usar go, .net, c#.*

3. Slow maths:

   - Si se intenta computar algo muy es muy lento, JavaScript es pésimo para matemática, JavaScript tiene un tipo de dato de número "number" custom made, adicionalmente es inconsistente con los floats.

- JavaScript nunca contempló esto por que se hizo a la carrera, pero si uno tiene operaciónes que hacer operaciones "sincronas" no use JavaScript.
- **Nos preguntamos:** ¿si tengo una compu de cuatro cores JavaScript va a usarlos todos o solo uno? Solo va a usar un core, por que el event loop es de **un sigle thread**, para operaciónes síncronas.
- ***Observación:*** *Es así por asuntos de simplificad y de seguridad por eso es ineficiente con operaciones matemáticas.*

6.6. Nos preguntamos: ¿Hay que usar node.js?

1. Sí, apréndalo y evalúe que necesita su aplicación, porque node.js no es bueno para todo hay evaluar si sería una mejor opción usar go, etc.
2. **Nos preguntamos:** ¿Node.js será la solución del futuro?
   - No exactamente pero se va a seguir usando por lo menos para los siguientes 10 años.

6.7. Cómo aprender node.js

1. Encontrar algo que quiera hacer e intentar hacerlo en node.js.

6.8. Consideraciones

1. Ver: REDIS
2. Proyecto próximo se desarrollará en python o node.js.

Capítulo 7

2019-10-24

7.1. Levantar Redis

7.2. JavaScript

1. JavaScript, se puede hacer todo con JS.
2. Ventajas: comunidad demasiado grande.
3. Vanilla JavaScript, es plain, Angular $\neq$ JavaScript.

7.3. Traducción de Python $\Rightarrow$ JavaScript.

Python	JavaScript

def(...):
if ...:
while ...:
	function(...)...
	if(...)...
	while (...)...

7.4. Homework

1. ECMA Script
2. Weakly typed / Strongly typed
3. Asynchronous of JavaScript - Call Stack
4. Event loop
5. ***Observación:*** *8 diapositivas máximo 4 mínimo.*
6. Opcional, TypeScript.

Capítulo 8

2019-10-29-Presentación de ECMAScript, weakly typed / strongly typed, asynchronous / call stack, event loop:

8.1. ECMA Script - Katherine Garcia

1. JavaScript, el lenguaje de JavaScript se define bajo un estándar de ECMA, ECMAScript se creó para tener un estándar común entre los navegadores.
2. Son un set de reglas, detalles y pautas que describen cómo es un lenguaje script.
3. Las versiones:
4. Son transpilados los lenguajes ECMA, compila el código y lo transcribe a funciones que sí funcionan en ECMAScript, generalmente se transpila a una versión de ECMAScript en una versión.
5. Se pueden mandar parametros por default, dentro de una función puede tomar un valor y fuera puede tomar otro.
6. Es un set de standards.
7. Transpilar, ***Definición de "transpilación":*** *es traducir código que no es complient con el ECMA standard.*
8. "By reading the ECMAScript specification, you learn how to create a scripting language. By reading the JavaScript documentation, you learn how to use a scripting language."
9. En python hay su propio ECMA compliance, es PEP, escribir en PEP es como vanila python.
10. Hay muchos lenguajes tienen estos estándares, ***Interesante:*** *ECMAScript versión 6, introdujo arrow functions, para los que vienen de un contexto como C#.*

8.2. Weakly typed / Strongly typed

1. Strong: "siempre que que un objeto pase de una función de llamada una a función llamada, su tipo debe ser compatible con el tipo declarado e la función llamada".
2. ***Ejemplo:*** *Java, Pascal, C*, and Lisp.*

3. ***Interesante:*** *En C si se utiliza punteros uno usa la memoria directamente de la computadora por ende se puede ver una característica de un weakly typed.*
4. Weak: "En lenguaje no cheque que los tipos de datos y solo asume que sí va a comportar como le dice el developer".
5. ***Ejemplo:*** *JavaScript, Perl.*
6. Dynamic Static, un lenguaje strongly typed tienden a ser Static, los weakly typed tienden a ser dynamic. No son synonimos,
7. Característica básicas de strongly typed:
   - Type safety
   - Memory safety
   - Static type checking
8. Aprender typescript asíncrono.
9. ***Interesante:*** *Codling, go,*

- Statically typed: no es restrictivo.
- Python es un hybrido.
- Dynamycally Strongly typed: restrictivo en qué tipos de datos se pueden operar.
- Usualmente los lenguajes compilados en algún nivel tienden a ser strongly typed.

8.3. Asynchronous - Call Stack

1. Asynchronous:
   - Varias operaciones están ocurriendo en threads separados.
   - JavaScript, funciona en base a una naturaleza síncrona pero se puede volver asíncrono.
2. Call stack:
   - Es una parte del event loop, está diciendo que funciones o variables están metidos en el stack.
   - ¿Asínrono y paralelo? asíncrono , **no blockea**, paralelo **todo se ejecuta al mismo tiempo.**
   - Call stack $\neq$ quee.
   - Un stack es apilar algo, se puede ver como un to-do list de funciones.
   - LIFO, Last in first out.

8.4. Event loop - Alejandra Lemus

1. El single thread en V8.
2. Cuando no estamos usando el browser sino node.js podemos usar librerías de C.
3. JavaScript si es un single thread, es un single threads, si tenemos ocho cores JavaScript solo va a utilizar uno, sin embargo se puede programar para que se haga una ejecución más inteligente.
4. Event loop revisa si hay algo en el call stack para poder ejecutarlo, el event loop revisa el call-stack y si no hay nada ve el quee para meterlo al call stack.
5. Verlo como un reloj, revisa cada ciclo de lo que está en el quue y en el stack.

6. Promesas, eran para evitar hacer un montón de call-backs.
7. Call-back hell, es un montón de call-backs.
8. "Programar en promesas"
9. ***Definición de "promesas":*** *es una forma más bonita de hacer callbacks.*
10. ***Interesante:*** *cosas como un thread se queda esperando a otro thread es un asunto de ponerle atención a multiples variables, cosas como un thread blockea a otro aun que sea asíncrono, en Java se puede trabajar con single threads.*
11. ***Interesante:*** *GoldRoutine, uno puede mandar multiples threads pero requieren de un orquestador.*

8.5. Love-my-movies

- Usen JS en el proyecto.
- ***Ojo: considerar lo siguiente...*** *usen JavaScript para las alertas y para las cosas del client side.*

Capítulo 9

2019-10-31-Ejemplos de JS promesas y anidación

9.1. Call-stack, Clase por Giovany

1. Una promesa va a ejecutar código cuando el call-stack esté vacío.
2. Para no anidar se pueden usar promesas, las promesas dejan código pendiente.
3. Concepto de await: solo se puede utilizar para una función “async” , “async” convierte esta función es una promesa.

Capítulo 10

2019-11-05

10.1. Discusión de love my robots

Capítulo 11

2019-11-07

11.1. Express

- Express ≡ Flask
- Public ≡ Static
- Para llamar a una variable en pug `#{<nombre de la variable>}`