Desarrollo de Aplicaciones Web Empresariales

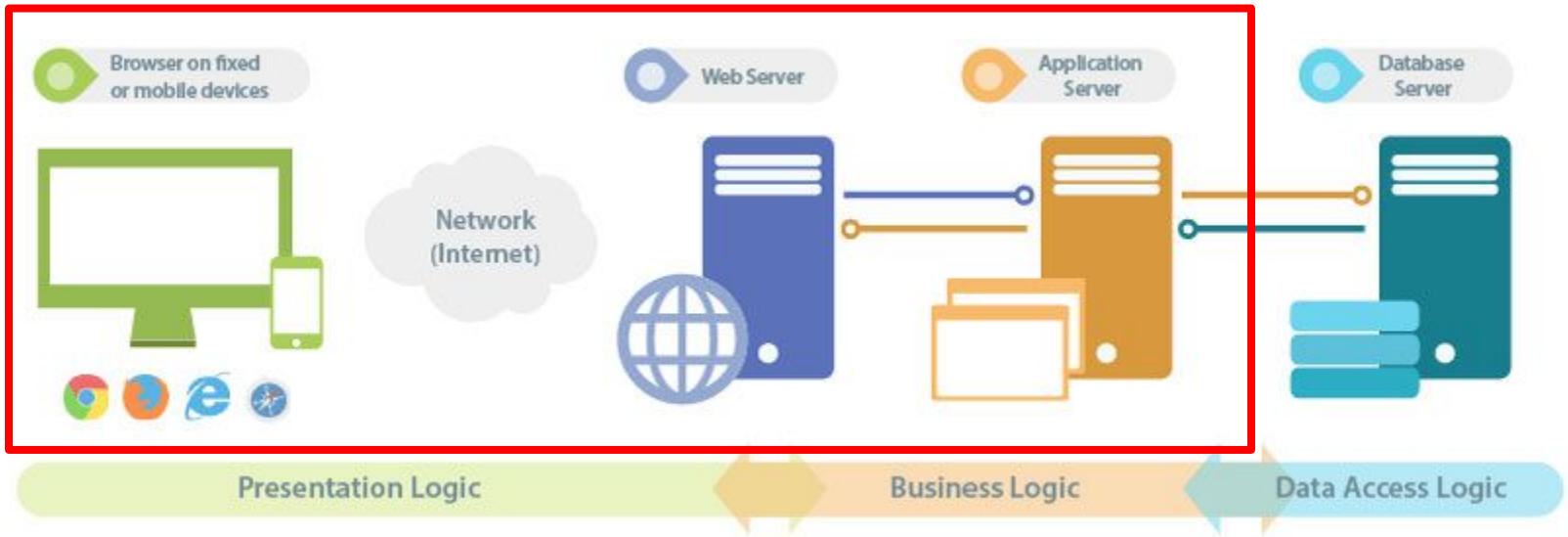
Exequiel Fuentes Lettura
exequiel.fuentes@ucn.cl

Información de contacto

- Exequiel Fuentes Lettura
  - Email: exequiel.fuentes@ucn.cl
  - Horario de Atención: Jueves y Viernes, bloque C

- Departamento de Ingeniería de Sistemas y Computación
  - Oficina: Y1 - 329
  - http://www.disc.ucn.cl

Arquitectura de aplicaciones Web



El rol del Controller en MVC

- El trabajo del Controller es obtener el modelo para la vista
  - Puede requerir comunicación con servidor, ejemplo: servicio de autenticación.
- En un principio, el browser realizaba solicitudes HTTP para obtener el modelo
  - En Microsoft les gustaba XML, así que la extensión del DOM fue llamada XMLHttpRequest
- Se permite a Javascript hacer solicitudes HTTP sin cambiar de página.
- AJAX significa Asynchronous Javascript and XML.
- Permite enviar solicitudes y respuestas en XML o cualquier otro formato
  - Ahora es ampliamente usado JSON.

XMLHttpRequest

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

Event handling

```
function xhrHandler() {
  if (this.readyState != 4) { // DONE
      return;
  }
  if (this.status != 200) { // OK
      // Handle error ...
      return;
  }
  ...
  var text = this.responseText;
  ...
```

Eventos en XMLHttpRequest

- Los eventos tienen varias etapas en el proceso de solicitud:

0	UNSENT	open() has not been called yet.
1	OPENED	send() has been called.
2	HEADERS_RECEIVED	send() has been called, and headers and status are available.
3	LOADING	Downloading; responseText holds partial data.
4	**DONE**	**The operation is complete.**

- Las respuestas están disponibles en:
  - Texto crudo: responseText
  - Documento XML: responseXML
- Se puede modificar las cabeceras en la solicitud y leer las cabeceras de la respuesta.

AJAX tradicional

- Cuando la respuesta es HTML:
  - elem.innerHTML = xhr.responseText;
- Cuando la respuesta es Javascript:
  - eval(xhr.responseText);
- Cuando la respuesta es un modelo de datos (JSON por ejemplo):
  - JSON.parse(xhr.responseText);
- Ninguna de estas es la forma como lo hace AngularJS:
  - Más información aquí: https://docs.angularjs.org/api/ng/service/$http

Obteniendo modelos con XMLHttpRequest

- Un Controller necesita comunicar en la solicitud que modelo necesita.
- Puede codificar la información del modelo en la URL:
  - Ruta: xhr.open("GET","userModel/78237489/fullname");
  - Parámetros: xhr.open("GET","userModel?id=78237489&type=fullname");
  - En el cuerpo de la solicitud:

```
xhr.open("POST", url);
xhr.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded");
xhr.send("id=78237489&type=fullname");
```

REST

- REST: Representational State Transfer.
- Fue presentada por primera vez por Roy Thomas Fielding en la disertación de su PhD.
- Antes de REST se usaban otros sistemas más complejos, ejemplo: RPC
- Define la forma como una aplicación web se comunica con el servidor

Algunos atributos de REST

- El servidor debería exportar recursos a los clientes usando nombres únicos (URI):
  - Una colección: http://www.example.com/photo/
  - Un recurso: http://www.example.com/photo/78237489
- Mantener el servidor “sin estado”.
- Permitir el almacenamiento en caché de los recursos.
- El servidor debe soportar métodos que mapean a CRUD:
  - GET: Leer un recurso.
  - PUT: Actualizar un recurso.
  - POST: Crear un recurso.
  - DELETE: Borrar un recurso.

Cómo diseñar en REST

- Defina un recurso del servicio y darle un nombre único (URI)
- Permitir acceder al recurso con las operaciones CRUD usando HTTP.
- Extender cuando sea necesario. Por ejemplo: hacer consultas a varios recursos.

REST y AngularJS

- $http: Envía una solicitud HTTP ($http.get, $http.post, etc.)
- $resource: Interactúa con los recursos del lado del servidor.
- Cómo:
  - Define un recurso REST con $resource
    - var resource = $resource(resourceURL, parameters);
  - Ejecuta métodos REST sobre los recursos:
    - resource.get(parameters, callback);
      resource.save(parameters, callback);
- Más información en: https://docs.angularjs.org/api/ngResource/service/$resource

Ejemplo, obtener el modelo:

```
let PhotoListOfUser = $resource('/photos/:id', {id: '@id'}, {
    get: {method: 'get', isArray: true}
});

PhotoListOfUser.get({id: userId}, function(userPhotos) {
    console.log('userPhotos', userPhotos);
});
```

- Genera una solicitud HTTP GET a la URL y retorna el modelo.

Ejemplo, almacenar el modelo:

```
let AddComment = $resource('/commentsOfPhoto/:id', {id: photoId});

AddComment.save({commentText: 'New Comment!'}, function (comment) {
    console.log('Added comment', comment);
});
```

- Genera una solicitud HTTP POST a la URL y crea un modelo.

HTML5 WebSockets

- Para mantener esta ilusión de siempre conectado, hay muchas conexiones HTTP.
- Esto crea un problema que la latencia en las conexiones. Ejemplo: un juego multijugador en el browser.
- Un conexión usando socket establece una conexión persistente entre el cliente y el servidor.
- HTML5 proporciona sockets al browser embebida con una interfaz similar a XMLHttpRequest.

HTML5 WebSockets

```
var socket = new WebSocket("ws://www.example.com/socketserver");

socket.onopen = function (event) {
    socket.send(JSON.stringify(request));
};

socket.onmessage = function (event) {
    JSON.parse(event.data);
};
```

¿Preguntas?