Desarrollo de Aplicaciones Web Empresariales

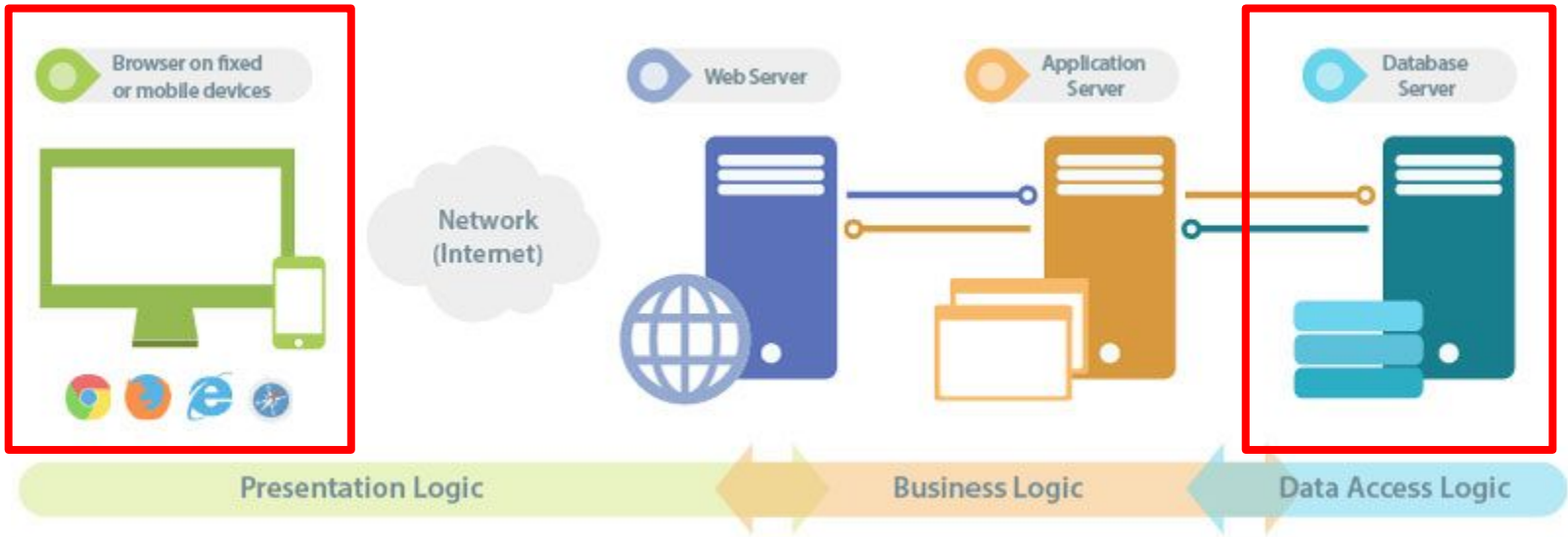
Exequiel Fuentes Lettura
exequiel.fuentes@ucn.cl

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: Sesiones



Cómo sabemos cuál usuario envió una solicitud?

- Nos gustaría autenticar a un usuario y tener la información disponible cada vez que se procese una solicitud.
- Generalmente, las aplicaciones web mantienen el estado por “browser activo”
- Concretamente:
  - expressApp.get('/user/:user_id', function (httpRequest , httpResponse)...

    // Se necesita tomar una decision para aceptar la solicitud o rechazarla

    const sessionState = GetSessionState(httpRequest);

El problema de la sesión

- Una solicitud HTTP llega al servidor
  - No toda la información identifica la sesión
- Solución: Incluir algo en la solicitud para obtener la sesión.
- Una solución inicial fueron las cookies.
  - El estado es asignado por el servidor y el browser lo adjunta por cada solicitud
  - Útil pero con una historia llena de conflictos:
    - Las personas a veces comparten el computador.
    - Se pueden borrar.
    - Problemas con la privacidad.
    - Las cookies han muerto, larga vida a las cookies!
- Los browser modernos soportan Web Storage API

HTTP Cookies: Idea básica

- Servidores web agregan cookies al HTTP response header

  Set-Cookie: cookie_name1=cookie_value1

  Set-Cookie: cookie_name2=cookie_value2; expires=Tue, 9 Nov 2016 08:31:45 GMT

- Cada cookie es sólo una tupla nombre, valor.
- Solicitudes futuras desde el browser al mismo servidor debería incluir las cookies

  Cookie: cookie_name1=cookie_value1; cookie_name2=cookie_value2

El contenido de una cookie

- Cookie: nombre y valor:
  - Dominio: servidor, puerto (opcional), URL (opcional).
  - Las cookies sólo se incluyen en las solicitudes que coinciden con su dominio.
  - Fecha de expiración: El browser puede eliminar las cookies viejas.
- Límite:
  - El tamaño viene dado por el browser, típicamente es < 4 KB.
  - El browser limita el número de cookies por servidor (alrededor de 50).

Cookies como un forma de almacenamiento

- El usuario puede:
  - Ver las cookies.
  - Modificar (corromper) las cookies.
  - Borrar cookies.
  - Crear cookies.
  - Perder cookies debido a hackers.
- Es un método poco confiable para almacenar datos
  - No se puede recuperar si se pierde.

Estado de la sesión con cookies

- Web frameworks como Rails soporta almacenar el estado de una sesión en cookies.
  - Rails proporciona una variable de sesión, un objeto tipo javascript donde es posible almacenar información.
- Rails empaca la sesión en una cookie y la agrega a la respuesta HTTP.
  - Así los datos estarán disponibles para futuras solicitudes desde el mismo browser.
- Rails automáticamente verifica la cookie de sesión por cada solicitud.
  - Si la cookie existe, la usa para cargar los datos de la sesión.
  - De lo contrario, crea una nueva sesión.

Manejo de sesión en Rails

```
def init_session
  session[:session_id] = request.session_options[:id]

  if session[:session_id] == nil
    session[:return_to] = nil
    session[:notice] = 'It seems that your previous session has ' \
      'expired. If the problem persists, please email us'

    redirect_to :controller => '/application', :action => 'index',
      :status => request.xhr? ? 500 : 302
  end
end
```

Estado de la sesión en cookies

- Una solución era: Almacenar el estado de la sesión en la cookie:
  - Dado que las cookies pueden ser vistas, cambiadas, borradas, robadas, etc. se deben tomar precauciones. Ejemplo:
    - session.user_id = “efuentes”
    - session.password = “mypass”
  - Usando criptografía se puede:
    - Esconder el contenido.
    - Detectar cambios.
    - No se puede hacer nada con el borrado.
- Una alternativa es guardar un puntero al estado de la sesión en la cookie:
  - Set-Cookie: session=0x4137fd6a; Expires=Wed, 09 Nov 2016 10:18:14 GMT
    Aquí hay menos transferencia de datos, pero aún se necesita proteger con criptografía.

Opciones para almacenar el estado de una sesión

- La memoria del servidor
  - Acceso rápido.
  - Puede que sea grande dependiendo de la cantidad de usuarios.
  - Hace que el balanceo de carga sea difícil.
- Un sistema de almacenamiento
  - Fácil de compartir entre todos los servidores web.
  - Quizás es demasiado carga para un sistema de almacenamiento.
- Un sistema de almacenamiento especializado
  - Soporta rápida obtención de datos pequeños.
  - Ejemplo: memcache, redis en memoria (almacena clave, valor)

Sesión en express

- ExpressJS tiene algunos middleware para lidiar con sesiones
  - Almacena una sessionID de manera segura en una cookie.
  - Como Rails, maneja la creación y la obtención del estado de la sesión desde la solicitud.
- Uso:

```
const session = require('express-session');

const sess = {
  secret: process.env.SESSION_SECRET,
  cookie: {},
  saveUninitialized: false,
  resave: false
}
```

```
app.use(session(sess));

app.get('/login', function (req, res) {
  req.session.user = req.query.email;
  res.redirect("/");
});
```

Sesión en express

- El atributo secret encripta session ID.
- Los datos de la sesión siempre se almacenan en el servidor.
- Se debe tener una variable de ambiente SESSION_SECRET, hay otras formas, pero nunca se debe tener en el código ni explícitamente en un repositorio (y menos público)
- Para mantener conexiones seguras, siempre usar https.

Sesión en express

- Por defecto la sesión se guarda en memoria en Node.js
  - Esto está bien para el ambiente de desarrollo, pero no para producción.
- No olvidar destruir la sesión cuando el usuario salga. También se le puede dar un tiempo de vida.
  - ```
    app.get('/logout', function(req, res) {
        req.session.reset();
        res.redirect('/');
    });
    ```
- Más información de cómo usar express-session aquí: https://github.com/expressjs/session
    ```

Sesión en express

- Es altamente recomendable usar métodos para almacenar sesión, hay que evitar usar la memoria del servidor
  - Redis: https://www.npmjs.com/package/connect-redis
  - Mongo: https://github.com/jdesboeufs/connect-mongo
  - Otras alternativas: https://github.com/expressjs/session#compatible-session-stores

```
var session = require('express-session');
var RedisStore = require('connect-redis')(session);

app.use(session({
    store: new RedisStore(options),
    secret: 'keyboard cat'
}));
```

```
const session = require('express-session');
const MongoStore = require('connect-mongo')(session);

app.use(session({
    secret: 'foo',
    store: new MongoStore(options)
}));
```

Cómo reemplazar las cookies

- Utilizar Web Storage API
- Se tiene (variables globales definidas en el objeto window):
  - sessionStorage: Mismo origen y está disponible mientras esté abierta la página.
  - localStorage: Mismo origen, todas las páginas y hasta que el browser se cierra.
  - Entonces, estos difieren en el alcance y tiempo de vida.
- Tiene una interfaz de clave-valor:
  - localStorage.appSetting = 'Anything';

    localStorage.setItem('appSetting', 'Anything');

    sessionStorage['app2Setting'] = 2;

Cómo reemplazar las cookies

- Tiene un límite aproximado de 10 MB (depende del browser).
- Tiene problemas similares a las cookies en cuanto a la confianza.
- Más información en este link: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

```
if (localStorage.clickcount) {
    localStorage.clickcount = Number(localStorage.clickcount) + 1;
} else {
    localStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "You have clicked the button " +
localStorage.clickcount + " time(s).";
```

```
if (sessionStorage.clickcount) {
    sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
} else {
    sessionStorage.clickcount = 1;
}
document.getElementById("result").innerHTML = "You have clicked the button " +
sessionStorage.clickcount + " time(s) in this session.";
```

¿Preguntas?