PROYECTO FINAL

Vamos a crear una aplicación ANDROID del tipo cliente-servidor.

La idea es crear una app que funcione de forma cliente servidor. Queremos que la lógica de la aplicación este en la parte del servidor y podamos acceder a ella desde distintos clientes ya sean móviles, desde una web. La idea es crear una especie de foro sencillo parecido a Twitter, por tener una referencia.

Contenido

[**SERVIDOR** 1](#_Toc480881134)

[EL PATRÓN DE DISEÑO PROXY 1](#_Toc480881135)

[BASE DE DATOS 1](#_Toc480881136)

[SQL 1](#_Toc480881137)

[NUESTRA APLICACIÓN 2](#_Toc480881138)

[LóGICA DEL NEGOCIO 2](#_Toc480881139)

[API REST 2](#_Toc480881140)

[**CLIENTE** 3](#_Toc480881141)

# **SERVIDOR**

### EL PATRÓN DE DISEÑO PROXY

El cliente es el proxy, el cual cree que los métodos que hemos implementado son la lógica del negocio. Pero estos métodos no son los verdaderos, si no que se comunican con los verdaderos métodos.

## BASE DE DATOS

Programaremos en SQL. Tendremos que conseguir una base de datos, en nuestro caso utilizaremos SQLite. <https://www.sqlite.org/cli.html>

La idea es crear una base de datos y dentro de estas creamos tablas. La estructura de la tabla debe estar diseñada desde el principio. Se establece un límite de columnas, pero no de filas, lo cual viene limitado por la memoria.

### SQL

Se crean una serie de tablas con información de distintos elementos. Por ejemplo, imaginemos el caso de un alumno y una asignatura. La asignatura necesitará un código o identificador, un nombre y una descripción. Un alumno necesitará otras propiedades, incluyendo un identificador igualmente. Estos elementos se conocen como **tablas de** **entidades**. En las tablas de entidades habrá siempre **una primary key**, que es el identificador de la clase. Al definir una columna como key, está siempre tendrá que tener un valor definido y deberá ser único. Una clave también puede ser dos columnas al mismo tiempo.

Hay **otro tipo de tablas que relacionan estas entidades**. En el caso anterior podría ser una tabla matrícula que relacionaría los alumnos y las asignaturas. Estos dos serán claves ajenas, ya que son claves de otras tablas. Lo que no se podrá repetir aquí es una misma combinación de claves. Es decir, la clave será la combinación de las claves ajenas.

>char(8) //debe tener 8 a la fuerza

>varchar(8)// puede tener hasta 8

Una vez que ya está la información, nos queda la parte de Query, es decir, la de interrogar. Suele ser más complicado cuando hay varias tablas. La versión más sencilla es preguntar de una sola tabla. Para ello:

>select \* from alumno; //Nos da la información de una tabla entera; Asterisco es todo.

>select nombre, apellido from alumno; //Nos pasa los parámetros que decidamos;

Es más difícil cuando empezamos a ponerle condiciones a lo que buscamos. Por ejemplo, los DNI de los que se apellidan García:

>select dni from alumno where apellido=’García’;

A continuación, veremos cómo navegar de tablas a otras, imaginemos que queremos buscar las asignaturas en las que está matriculado Juan, para lo que:

>select Asignatura.nombre

from Alumno, Matrícula, Asignatura

where Alumno.nombre= ’Juan’

and Alumno.dni = Matricula.dni and Matrícula.cod = Asignatura.cod;

Toda la info queda descrita en el enlace de arriba.

Más tarde podemos crear ficheros de texto con los comandos que solamos utilizar para luego pasárselos al programa SQlite. De esta forma es más sencillo realizar pruebas.

### NUESTRA APLICACIÓN

*Cambiado el 16 de mayo.*

En la base de datos tendremos una entidad llamada **usuario**:

|  |  |  |
| --- | --- | --- |
| id | nombre | password |
| Char(8) | Varchar(20) | Varchar(20) |

Tendremos otra base de datos que se llama **mensajito**:

|  |  |  |
| --- | --- | --- |
| nombre | instante | texto |
|  | integer | Varchar(140) |

IdUsuario es la clave ajena, la clave primaria será la combinación de instante + idUsuario.

En programación suele haber una función que da los milisegundos que pasan desde el inicio de año de 1970 y nos da la fecha actual: THE EPOCH.

## LóGICA DEL NEGOCIO

Utilizaremos JS para programar esta lógica del negocio. Para ello utilizaremos NODEJS. Aquí es donde tenemos la programación.

### NODE.JS

Con NODE podemos utilizar JavaScript en la parte del servidor. Durante mucho tiempo se ha utilizado PHP. Para no tener que aprender otro lenguaje utilizaremos JavaScript. Este no se le parece tanto a Java como nos pueda parecer. JavaScript es un lenguaje interpretado, es decir que directamente lo ejecutamos y va leyendo línea a línea y ejecutando. Es, además, asíncrono. Primero pensemos en el concepto de síncrono. Al llamar una función, vamos a esta función, quedándose la ejecución en espera, y hasta que no llega algo de vuelta no continuamos ejecutándonos. En este caso podemos crear una serie de funciones que no tengan este sincronismo. Nos enfrentaremos a funciones de biblioteca asíncronas. Esto se debe a que esta aplicado a servidores web, donde deben atenderse muchos clientes al mismo tiempo creándose distintos hilos. VER ARCHIVOS DE EJEMPLO

Para abrir un archi .js desde la consola:

### JSON

JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. Tanto HTML como JSON, son particularizaciones de XML. Estos lenguajes funcionan mediante marcas, que nos permiten incluir una serie de metadatos en el texto. Nosotros podemos incluir nuestras propias marcas. Enviaremos del servidor al cliente mensajes del tipo XML, más concreto JSON, ya que estamos utilizando JavaScript. Veamos algo de código:

var myObj = { "name":"John", "age":31, "city":"New York" };  
console.log (myObj.name);

Podemos también incluir nuestros resultados dentro de etiquetas.

console.log (“<nombre>” + myObj + “</nombre>”);

Utilizando esto, en recepción podemos utilizar una biblioteca que pueda realizar parsing en el texto que enviemos de forma que sea más fácil.

Lo bueno es que podemos utilizar una función para poder realizar esto automáticamente. Es decir, nos permite coger un objeto y pasarlo a texto.

var mensaje = JSON.stringify(myObj);

En la recepción podemos volver a convertir el texto que hemos recibido a objeto.

var myObj2 = JSON.parse(mensaje);

### NUESTRA APLICACIÓN

## API REST

Convertirá el resultado de los métodos a mensajes http. Nos adaptaremos a los mensajes ya que establecen un protocolo cliente servidor: GET, PUT/ POST, DELETE… etc. Estos se llaman verbos. Utilizaremos una librería para procesar estos mensajes. También en JavaScript.

# CLIENTE

La idea es que cualquier tipo de dispositivo pueda acceder al servidor de forma que podamos diseñar distintas plataformas que se ajusten a nuestra aplicación web.

#### ENLACES DE INTERÉS

[www.w3schools.com](file:///C:\Users\Alejandro\AppData\Roaming\Microsoft\Word\www.w3schools.com)

#### CRONOLOGÍA

* **Tue, Apr 25:** Practicamos con las bases de datos SQL y establecemos las entidades básicas para nuestra aplicación. Instalar node.js y ver cómo funciona, familiarizarnos.
* Tue, May 9: Aprender el funcionamiento de las funciones asíncronas en Node.
* Tue, May 16: Programar la lógica en node y comunicarla con la base de datos.

SOLUCIÓN PASO A PASO

Para crear las bases de datos:

En la carpeta de nuestro proyecto creamos una bat para abrir la cmd.

**cd C:\Users\Alejandro\Desktop\Android\proyectos\2017-04-25-ProyectoFinal\basededatos**

**cmd**

A continuación escribimos el directorio del ejecutable de sqlite para poder inicializarlo:

**path C:\Users\Alejandro\Desktop\Android\doc\SQLITE**

Ahora ya podemos empezar a crear las distintas tablas que se piden. Comenzaremos creando la base de datos y la tabla usuario.

sqlite3 pruebas.bd

**sqlite> create table usuario(id char(8) primary key ,nombre varchar(20), password varchar(20));**

Igual para la entidad de los mensajes.

**sqlite> create table mensajito(**

**idUsuario char(8) NOT NULL,**

**instante integer NOT NULL,**

**texto varchar(140),**

**primary key(instante),**

**foreign key(idUsuario) references usuario(id));**

## TIPS

Un truco para que el formato de las tablas quede mejor:

**sqlite> .mode column**

**sqlite> .header on**

Para saber las tablas que hemos creado podemos utilizar el comando de abajo.

**.table**

**.schema**

Si creamos la tabla y olvidamos poner cual es la primary key existe la siguiente solución:

<http://stackoverflow.com/questions/16900552/change-the-primary-key-of-a-table-in-sqlite->

Para crear una base de datos:

**.open nombre\_bd**

Para claves externas:

**sqlite> create table Matricula(dni char(8), cod char(3), FOREIGN KEY(dni) REFERENCES Alumno(dni));**