SEMINARIO DE ACTUALIZACIÓN

INFORME

ALEXIS ALEXANDER BEDOYA ARIAS

ANTONIO JOSÉ CORTÉS SAMPAYO

CRISTIAN HERNÁNDEZ GARAY

PROF. JOHN CARLOS ARRIETA ARRIETA

UNIVERSIDAD DE CARTAGENA

FACULTAD DE INGENIERÍA

INGENIERÍA DE SISTEMAS

2021

TABLA DE CONTENIDO

1. INTRODUCCIÓN

HTTP es un protocolo que permite obtener recursos y es la base de cualquier intercambio de datos en la web, tratándose de un protocolo cliente-servidor. Este protocolo está diseñado para ser simple y extensible en el que incluso se puede introducir una nueva funcionalidad mediante acuerdos entre el cliente y el servidor sobre la semántica de un nuevo encabezado.

Al tratarse de un protocolo cliente-servidor, los mensajes enviados por HTTP pueden ser de dos tipos: Solicitudes o respuestas, cada uno de estos con su propio formato. Las peticiones por un lado, constan de un método HTTP, una ruta del recurso a buscar, la versión del protocolo HTTP y un encabezado para transmitir información adicional para los servidores. Por otra parte, las respuestas constan de la versión del protocolo HTTP que siguen, un código de estado, un mensaje de estado, encabezados HTTP y en algunas ocasiones, un cuerpo que contiene el recurso obtenido.

Al incluir el factor de los servicios web al protocolo HTTP, se cuenta con SOAP (Simple Object Access Protocol) y RESTful. Por una parte, SOAP es definido como un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Los servicios SOAP funcionan por lo general por el protocolo HTTP que es lo más común al invocar un Web Services, sin embargo, SOAP no está limitado a este protocolo, ya que puede también ser enviado por FTP, POP3, TCP, colas de mensajería (JMS, MQ, etc). Por otro lado, RESTFul es una tecnología que transporta datos por medio del protocolo HTTP, pero este permite utilizar los diversos métodos que proporciona HTTP para comunicarse, como lo son GET, POST, PUT, DELETE, PATCH y a la vez, utiliza los códigos de respuesta nativos de HTTP (404,200,204,409). Es tan flexible que permite transmitir prácticamente cualquier tipo de datos, lo que permite mandar XML, JSON, Binarios (imágenes, documentos), Text, etc.

En este informe se describe de forma detallada el proceso de desarrollo, en donde se mencionan los procesos de instalación, configuración, librerías, métodos y aplicación en el uso de tecnologías de comunicación sobre protocolo HTTP basadas en Servicios Web (SOAP y RESTFul) para interactuar con un clientes HTTP no tradicionales.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

Desarrollar un aplicativo que permita el registro de la entrada y salida de los clientes de un parqueadero, haciendo uso de la tecnología de comunicación sobre protocolo HTTP basadas en Servicios Web (SOAP y RESTFul) del lado del servidor, el cual debe interactuar con otro aplicativo cliente no tradicional. Esto último con la finalidad de desarrollar al menos una función en dos servidores de aplicaciones y en dos clientes, creando un punto de comparación entre tecnologías.

2.2. OBJETIVOS ESPECÍFICOS

- Diseñar la arquitectura del software teniendo en cuenta los requerimientos establecidos
- Implementar el software permitiendo a los clientes registrar su paso por los parqueaderos.
- Ejecutar pruebas en el transcurso del desarrollo e implementación del aplicativo para verificar su buen funcionamiento.

3. JUSTIFICACIÓN

El protocolo HTTP es utilizado tanto en aplicaciones de escritorio para computadores como en dispositivos móviles para solicitar a los servidores que envíen el contenido del sitio en cuestión. El principal objetivo de este protocolo es administrar un estándar para la comunicación cliente-servidor.

Al trabajar usando tecnologías de comunicación sobre protocolo HTTP basadas en servicios web, entran en mención el protocolo SOAP y RESTFul, de los cuales resulta pertinente conocer las características que los hacen una muy buena opción para el desarrollo de tecnologías:

Por una parte, SOAP ofrece las siguientes ventajas en comparación con REST:

- Es independiente del idioma, la plataforma y el transporte.
- Es estandarizado.
- Proporciona extensibilidad significativa previa a la construcción en forma de estándares WS.
- Tiene manejo de errores incorporado.

Por otro lado, REST es más flexible y más fácil de usar. Tiene las siguientes ventajas:

- Utiliza estándares fáciles de entender.
- Tiene una curva de aprendizaje más pequeña.
- Es eficiente (usa pequeños formatos de mensajes como JSON, XML, HTML etc.)
- Es rápido, no requiere un procesamiento extenso.
- Es más cercano a otras tecnologías web en temas de diseño.

Teniendo en cuenta los beneficios que tiene cada uno de estos, se puede crear una idea de la relevancia e importancia que estos tienen en la implementación de un proyecto, haciéndolos una solución que permita el cumplimiento de los objetivos planteados de manera eficiente.

4. DESARROLLO

4.1. CONTEXTUALIZACIÓN

Gestión y organización son dos cosas necesarias en cualquier actividad, la gestión de un parqueadero no es la excepción, puesto que resulta pertinente tener un buen control de los ingresos y salidas que se presentan en este.

Teniendo en cuenta lo anteriormente expuesto, se propone la creación de un aplicativo tecnológico para la gestión de ingresos y salidas de un parqueadero que ayude a mantener el control de este mismo, haciendo uso de tecnologías de comunicación sobre protocolo HTTP basadas en Servicios Web (SOAP y RESTFul) por parte del servidor y una aplicación cliente HTTP no tradicional, tanto para una PC como para móvil.

4.2. ARQUITECTURA DE DESARROLLO

Para el desarrollo de este proyecto se aplicó la arquitectura cliente-servidor, en la cual el cliente proporciona una interfaz que permite las solicitudes de servicios del servidor y también muestra los resultados arrojados por éste.

4.3. METODOLOGÍA O PROCESOS DE DESARROLLO

Para el desarrollo de este aplicativo se hace uso de la metodología Kanban, teniendo en cuenta los requerimientos del proyecto. Esta metodología es ideal para proyectos con cortos tiempos de entrega, además que permite llevar un buen control sobre las tareas asignadas, eliminando las tareas ineficientes. Por el tema de desarrollo en grupos, esta metodología también permite una gran flexibilidad, ya que al tener un control de tareas óptimo, todos el equipo conoce sus tareas y si surge algún imprevisto, existe una capacidad de respuesta para atenderlo.

4.4. ENTORNO DE TRABAJO

El entorno de trabajo en el que se desarrolló este proyecto es Apache NetBeans IDE permitiendo la creación de dos servidores y un clientes y Android Studio permitiendo la creación del cliente de app Android . (herramienta de alojamiento de servidor en heroku).

4.5. ESPECIFICACIONES

Por cuestiones de practicidad, las pruebas y desarrollo se realizaron en el localhost con un computador que tiene las siguientes especificaciones:

- Ram 8 GB
- Espacio de almacenamiento 500GB
- Procesador Intel core i7 6000U
- Internet Movistar 3 Mb

4.6. HERRAMIENTAS DE SOFTWARE

Para el desarrollo del aplicativo se utilizó el lenguaje Java, haciendo uso de un conjunto de clases y tecnologías:

- **kSOAP2:** Es una biblioteca ligera para usar en dispositivos restringidos.
- **HttpClient:** Proporciona una clase base para enviar solicitudes HTTP y recibir respuestas HTTP de un recurso identificado por un URI.
- **Node.js:** Es un entorno en tiempo de ejecución de JavaScript.
- **Express:** Express es un marco de aplicación web Node.js mínimo y flexible que proporciona un conjunto sólido de funciones para aplicaciones web y móviles.
- **JSON:** Es un formato de intercambio de datos constituido por una colección de pares/valor y una lista ordenada de valores.
- **Connection:** Extiende las interfaces Wrapper y AutoCloseable. La base de datos de un objeto Connection puede proporcionar información que describe sus tablas, su gramática SQL soportada, sus procedimientos almacenados, las capacidades de esta conexión, etc.
- **DriverManager:** Extiende la clase Object y provee el servicio básico para administrar un conjunto de controladores.
- **ResultSet:** Extiende las interfaces Wrapper y AutoCloseable y muestra una tabla de datos que representa un conjunto de resultados de la base de datos, que generalmente se genera al ejecutar una declaración que consulta la base de datos.
- **SQLException:** Una excepción que proporciona información sobre un error de acceso a la base de datos u otros errores.

- **Statement:** Extiende las interfaces Wrapper y AutoCloseable y proporciona el objeto utilizado para ejecutar una declaración SQL estática y devolver los resultados que produce.
- **HttpURLConnection:** Extiende la clase URLConnection. Cada instancia se utiliza para realizar una sola solicitud, pero la conexión de red subyacente al servidor HTTP puede ser compartida de forma transparente por otras instancias.
- **URL:** Extiende la clase Object. Representa un localizador uniforme de recursos, un puntero a un "recurso" en la World Wide Web.
- **URLConnection:** Extiende la clase Object. Es la superclase de todas las clases que representan un enlace de comunicaciones entre la aplicación y una URL.

5. CLASES Y MÉTODOS

5.1. API REST EN NODE.JS

5.1.1. Index.js

Es el encargado de iniciar el servidor configurando el puerto en el cual se conectará.

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

// Settings
app.set('port', process.env.PORT || 3000);

// Middlewares
app.use(express.json());

// Routes
app.use(require('./routes/registro'));

// Starting the server
app.listen(app.get('port'), () => {
  console.log(`Server on port ${app.get('port')}`);
});
```

5.1.2. Database.js

Es en donde se conecta la base de datos haciendo uso de las credenciales de conexión.

```
const mysql = require('mysql');
//bbf89ae680c30c:328fa616@us-cdbr-east-03.cleardb.com/heroku_975e8a21664949b?reconnect=true
const mysqlConnection = mysql.createConnection({
  host: 'j21q532mu148i8ms.cbetxkdyhwsb.us-east-1.rds.amazonaws.com',
  user: 'w4f1agseuwgbanl3',
  password: 't1tpfcm1640win1k',
  database: 'm6fl4gi1gtab1m7l',
  multipleStatements: true
});

mysqlConnection.connect(function (err) {
  if (err) {
    console.error(err);
    return;
  } else {
    console.log('db is connected');
  }
});

module.exports = mysqlConnection;
```

5.1.3. Registro.js

Es el encargado de obtener los listados de la base de datos.

```
const express = require('express');
const router = express.Router();

const mysqlConnection  = require('../database.js');

// GET all listado
router.get('/', (req, res) => {
  mysqlConnection.query('SELECT * FROM listado', (err, rows, fields) => {
    if(!err) {
      res.json(rows);
    } else {
      console.log(err);
    }
  });

});

// GET An listado
router.get('/:id', (req, res) => {
  const { id } = req.params;
  mysqlConnection.query('SELECT * FROM listado WHERE id = ?', [id], (err, rows, fields) => {
    if (!err) {
      res.json(rows[0]);
    } else {
      console.log(err);
    }
  });
});
```

```
router.post('/', (req, res) => {
  const {id,nombre,cedula,placa,estado} = req.body;
  console.log(id, nombre, cedula,placa,estado);
  const query = `

    CALL listadoAddOrEdit(?,?,?,?,?);
  `;
  mysqlConnection.query(query, [id,nombre,cedula,placa,estado], (err, rows, fields) => {
    if(!err) {
      res.json({status: 'listadod Saved'});
    } else {
      console.log(err);
    }
  });

});

router.put('/:id', (req, res) => {
  const {nombre,cedula,placa,estado} = req.body;
  const { id } = req.params;
  const query = `

    CALL listadoAddOrEdit(?,?,?,?,?);
  `;
  mysqlConnection.query(query, [id, nombre,cedula,placa,estado], (err, rows, fields) => {
    if(!err) {
      res.json({status: 'listado Updated'});
    } else {
      console.log(err);
    }
  });
});
```

5.2. API SOAP JAVA WEB APPLICATION

5.2.1. BD

Esta clase es la encargada de conectarse con la base de datos y enviar las sentencias SQL a esta misma.

```
public class BD {

    String driver, url, login, password;
    Connection conexion = null;

    public BD() {

        driver = "com.mysql.jdbc.Driver";
        url = "jdbc:mysql://j21q532mu148i8ms.cbetxkdyhwsb.us-east-1.rds.amazonaws.com:3306/m6fl4gilgtablm71";
        login = "w4flagseuwgbanl3";
        password = "tltpfcml640winlk";
        try {
            Class.forName(driver).newInstance();
            conexion = DriverManager.getConnection(url, login, password);
            System.out.println("Conexion con Base de datos Ok....");
        } catch (ClassNotFoundException | IllegalAccessException | InstantiationException | SQLException exc) {
            System.out.println("Error al tratar de abrir la base de datos");
            System.out.println(exc.getMessage());
        }

    }
```

5.2.1.1. agregarInfo()

Método encargado de agregar o editar información en la base de datos.

```
public void AgregarInfo(int id,String nombre, String cedula, String placa) {

    String estado = "1";


     String ComandoSQL = "CALL listadoAddOrEdit('"+id+"','" + nombre
             + "' , '"
             + cedula + "' ,'" + placa + "','" + estado + "' )";
    try {

        Statement stmt = conexion.createStatement();
        stmt.executeUpdate(ComandoSQL);
        System.out.println("Registro agregado!");

        stmt.close();

    } catch (java.sql.SQLException er) {
        System.out.println("No se pudo realizar la operación.");
    }
}
```

5.2.2. WebService

5.2.2.1. listarAddorEdit()

Método encargado de llamar al método de la base de datos agregarInfo().

```
    @WebMethod(operationName = "listarAddorEdit")
    public void listarAddorEdit(@WebParam(name = "id") int id,
            @WebParam(name = "nombre") String nombre,
            @WebParam(name = "cedula") String cedula ,
            @WebParam(name = "placa") String placa) {
       bd.AgregarInfo(id, nombre, cedula, placa);
    }


    @WebMethod(operationName = "Listar")
    public List<Registro> Listar() {

        return bd.ListarRegistros();
    }
}
```

5.3. PROCEDURE BASE DE DATOS

Es un procedimiento almacenado con un conjunto de instrucciones, en este caso, permite insertar un dato si el Id es 0 y también permite editar un dato en caso de pasar como parámetro un Id diferente a 0.

```
CREATE DEFINER=`w4f1agseuwgbanl3`@`%` PROCEDURE `listadoAddOrEdit`(IN `_id` INT,
IN `_nombre` VARCHAR(50),
 IN `_cedula` VARCHAR(50), IN `_placa` VARCHAR(50), IN `_estado` INT)
BEGIN
  IF _id = 0 THEN
    INSERT INTO listado (nombre, cedula,placa,estado)
    VALUES (_nombre, _cedula,_placa,1);
    SET _id = LAST_INSERT_ID();
  ELSE
    UPDATE listado
    SET
    estado = 0
    WHERE id = _id;
  END IF;

  SELECT _id AS 'id';
END
```

5.4. CLIENTES

5.4.1. CLIENTES EN ANDROID STUDIO CONEXIÓN HTTPCLIENT API RESTFULL

5.4.1.1. ConexionHttpGetServer

Esta clase es la encargada de hacer peticiones get al servidor de Heroku.

5.4.1.1.1. conexionConElServidor()

Este método recibe una lista de parámetros y una ruta de conexión para realizar una conexión con el servidor.

```
public class ConexionHttpGetServer {
    public static String direccionDelServidor;
    private String respuesta;
    private InputStream datosEntrada;

    public String conexionConElServidor(String rutaDeLaAplicacionWeb)throws  Exception{
        HttpClient clienteHTTP = new HttpClient();
        BufferedReader br = null;
        GetMethod  peticionPOST = new GetMethod(rutaDeLaAplicacionWeb);

        try{
            int codigoRespuesta = clienteHTTP.executeMethod(peticionPOST);
            if(codigoRespuesta == HttpStatus.SC_NOT_IMPLEMENTED){
                throw new Exception("ERROR 1: Parametros mal calificados");
            }else if (codigoRespuesta != HttpStatus.SC_OK) {
                throw  new Exception("ERROR 2: URL Invalida");
            }
            else{
                datosEntrada = peticionPOST.getResponseBodyAsStream();
                respuesta = procesarRespuestaDelServidor();
                return (datosEntrada !=null)? respuesta: null;

            }
        }catch (Exception e){
            throw  new Exception("Error 3: Conexion fallida: \n" + e.getMessage());
        }finally {
            peticionPOST.releaseConnection();
            if (br != null){
                try{
                    br.close();
                }catch (Exception fe){

                }
            }
        }

    }
```

5.4.1.1.2. procesarRespuestaDelServidor()

Este método procesa la respuesta del servidor haciendo uso del formato JSON, retornándolo para su posterior uso.

```
private String procesarRespuestaDelServidor() throws Exception{
    BufferedReader lectorDatos = null;
    try{
        lectorDatos = new BufferedReader(new InputStreamReader(datosEntrada,  charsetName: "iso-8859-1"), sz: 8);
    }catch (UnsupportedEncodingException error){
        throw new Exception("ERROR 4: sin respuesta\n" + error.getMessage());
    }
    StringBuilder cadenaDinamica = new StringBuilder();
    String linea = null;
    String json2 = "";
    try{
        while (((linea = lectorDatos.readLine())!= null)){
            if(linea.trim().isEmpty() == false){
                json2 += linea;
            }
        }
    }catch (IOException error){
        throw new Exception("ERROR:5 Sin respuesta\n" + error.getMessage());
    }
    return json2;
}
```

5.4.2. CLIENTES EN ANDROID STUDIO CONEXIÓN HTTP SOAP

5.4.2.1. ConexionHttpSoap

5.4.2.1.1. entrada()

Método que realiza una petición a la API SOAP de tipo listadoAddorEdit, en la cual se pasan los parámetros para añadir un nuevo registro a la tabla listado.

```
public class ConexionHttpSoap {
    final String WSDL_TARGET_NAMESPACE = "http://sercivios/";
    final String SOAP_ADDRESS="http://10.0.2.2:8090/parqueaderoSOAP/webServices";
    final String OPERATION_NAME = "listarAddorEdit";
    final String SOAP_ACTION = "http://sercivios/webServices/listarAddorEdit";
    public ConexionHttpSoap(){}
    public void entrada(Registro r){


        String result="";
        SoapObject request = new SoapObject(WSDL_TARGET_NAMESPACE,
                OPERATION_NAME);

        SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(
                SoapEnvelope.VER10);

        // Con esta opción indicamos que el web service no es .net
        envelope.dotNet = false;

        envelope.setOutputSoapObject(request);

        HttpTransportSE httpTransport = new HttpTransportSE(SOAP_ADDRESS);

        // Enviando un parámetro al web service
        request.addProperty( name: "id", value: 0);
        request.addProperty( name: "nombre",r.getNombre());
        request.addProperty( name: "cedula",r.getCedula());
        request.addProperty( name: "placa", r.getPlaca());

        try {

            // Enviando la petición al web service
            httpTransport.call(SOAP_ACTION, envelope);

            // Recibiendo una respuesta del web service
            SoapPrimitive resultsRequestSOAP = (SoapPrimitive) envelope
                    .getResponse();

            httpTransport.getServiceConnection().disconnect();
        } catch (IOException | XmlPullParserException e) {
            e.getMessage();

        }

    }
```

5.4.2.1.2. salida()

Método que realiza una petición a la API SOAP de tipo listadoAddorEdit, en la cual se pasa el Id como parámetro para realizar un update en la base de datos.

```
public void salida(String id){

    String result="";
    SoapObject request = new SoapObject(WSDL_TARGET_NAMESPACE,
            OPERATION_NAME);

    SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(
            SoapEnvelope.VER10);

    // Con esta opción indicamos que el web service no es .net
    envelope.dotNet = false;

    envelope.setOutputSoapObject(request);

    HttpTransportSE httpTransport = new HttpTransportSE(SOAP_ADDRESS);

    // Enviando un parámetro al web service
    request.addProperty( name: "id",id);

    request.addProperty( name: "nombre", value: "");
    request.addProperty( name: "cedula", value: "");
    request.addProperty( name: "placa",  value: "");

    try {

        // Enviando la petición al web service
        httpTransport.call(SOAP_ACTION, envelope);

        // Recibiendo una respuesta del web service
        SoapPrimitive resultsRequestSOAP = (SoapPrimitive) envelope
                .getResponse();



        httpTransport.getServiceConnection().disconnect();
    } catch (IOException | XmlPullParserException e) {
        e.getMessage();

    }
```

5.4.3. CLIENTE ESCRITORIO

5.4.3.1. ConexionHttpRestySoap

Esta clase es la encargada de hacer peticiones get al servidor de Heroku para obtener una lista de registro.

```
public  List<Registro> listar()throws InterruptedException, ExecutionException, TimeoutException{
    HttpClient client = HttpClient.newHttpClient();
   HttpRequest request = HttpRequest.newBuilder().uri(URI.create
      ("https://seminario-update.herokuapp.com")).build();
    CompletableFuture<HttpResponse<String>> response = client.sendAsync(request, HttpResponse.BodyHandlers.ofString());
      JSONArray albums = new JSONArray(response.thenApply(HttpResponse::body).get(5, TimeUnit.SECONDS));
      List<Registro> l = new ArrayList<Registro>();
      for (int i = 0; i < albums.length(); i++) {
          JSONObject album = albums.getJSONObject(i);
          l.add(new Registro(album.getString("nombre"),album.getString("cedula"),
                  album.getString("placa"),String.valueOf(album.getInt("estado")),
                  String.valueOf(album.getInt("id")),album.getString("horaEntrada"),
                  album.get("horaSalida").toString()));
      }
  return l;
  }
```

5.4.3.1.1. listarAddorEdit()

Este método es generado por java para consumir un Web Service.

```
private static void listarAddorEdit(int id, java.lang.String nombre,
        java.lang.String cedula, java.lang.String placa) {
    parqueaderoclienteescritorio.ws.WebServices_Service service =
            new parqueaderoclienteescritorio.ws.WebServices_Service();
    parqueaderoclienteescritorio.ws.WebServices port =
            service.getWebServicesPort();
    port.listarAddorEdit(id, nombre, cedula, placa);
}
```

6. INSTALACIÓN Y CONFIGURACIÓN

Este aplicativo está dividido en dos partes: una api RESTFul montada en los servicios de alojamiento de Heroku que no necesita configuración (API RESTFUL) y una api SOAP que se tiene que montar localmente mediante el uso de Apache netbeans(java web application) con las siguientes credenciales de base de datos:

```
driver = "com.mysql.jdbc.Driver";

url= "jdbc:mysql://j21q532mu148i8ms.cbetxkdyhwsb.us-east-1.rds.amazonaws.com:3306/m6fl4gi1gtab1m7l";

login = "w4f1agseuwgbanl3";

password = "t1tpfcm1640win1k";
```

Se hace uso del driver mysql-connector-java-5.1.48-bin, el cual se encuentra alojado en la carpeta del mismo proyecto (java soap), además de tener en cuenta el uso del JDK 11 o superiores, puesto que es necesario para ejecutar los métodos de conexión del cliente de escritorio.

7. COMPARATIVAS Y EXPERIENCIAS

Alexis Bedoya Arias: Uno de los objetivos para el desarrollo de este proyecto fue elaborar dos API una en REST y otra en SOAP, para tener un punto de comparación con los servicios requeridos, encontrado un claro diferenciador en ambas siendo SOAP un protocolo estandarizado basado en reglas estrictas poco flexibles, con característica de seguridad avanzada necesitando de más tiempo de carga por su complejidad mientras que REST es más flexible con la posibilidad de escoger el formatos de comunicación y rápida si se escoge el formato más ligero. teniendo en cuenta lo anterior ambas APIS se aplicaron a este proyecto teniendo más popularidad y documentación la basa en REST lo que permitió un desarrollo más rápido en su despliegue (HEROKU) y su consumo de servicios, por otro lado en SOAP se encontró el problema de la información desactualizada, librerías sin soporte y poca documentación sobre servicios gratuitos actuales que permitieran su despliegue por lo mismo se opto de desplegarla de manera local consumiendo una base de datos remota (HEROKU), ambos servicios consumido por los dos tipos de cliente uno en android y otro en java, siendo android el que presentó más problemas por su poco soporte en servicios tipo soap, solucionado usando la librería ksoap2 que permitió establecer una conexión para consumir el servicio.

considerando los problemas anteriores, la experiencia de trabajo en cuanto elaboración de API y consumo de la misma es REST independiente del entorno de desarrollo, por documentación, popularidad, flexibilidad y curva de aprendizaje.

Antonio Cortés Sampayo: Al hacer uso de la tecnología de comunicación requerida (Protocolo HTTP) y estas basadas en servicios web tales como SOAP y Restful hemos podido evidenciar las diferencias en el uso de estos servicios en este proyecto, SOAP tiene la capacidad de soportar XML es un protocolo mucho más robusto que posee un tipeado mucho más fuerte, permitiendo agregar metadatos mediante los atributos, también es un formato más pesado, tanto en tamaño como en procesamiento, en cambio con Restful, resultó ser una tecnología sumamente flexible y con mejor perfomance, permite transmitir prácticamente cualquier tipo de datos, aunque usualmente se utiliza JSON debido a son considerablemente más livianos en peso y mucho más rápido en su procesamiento pero su performance tiene un costo, y es la robustez del mensaje como tal.

Se desarrollaron dos servidores, uno Node.js utilizando el servicio Rest el cual es un servicio bastante flexible que facilita su uso, y otro servidor en Java Web Aplication utilizando el servicio SOAP, este servicio resultó ser más complicado debido a la poca flexibilidad, y se desarrollaron dos clientes, uno de escritorio en el entorno de desarrollo de NetBeans en el cual resultó más cómodo trabajar y otro de Android en el entorno de desarrollo de Android Studio el donde se generaron más contratiempos debido a la falta

de conocimientos de esta, ambos clientes son muy similares y presentan pequeños cambios a causa del uso de los diferentes entornos de desarrollo.

Cristian Hernández Garay: Uno de los requerimientos para el desarrollo de este proyecto fue elaborar una API con RESTFul y otra con SOAP. Como se mencionó en las ventajas que posee cada uno de estos servicios, SOAP, por un lado, es un servicio mucho más robusto y estricto, siendo considerado el mejor protocolo para la comunicación de servidor a servidor o cliente a cliente, sin embargo, es esta misma robustez la que que convierte en una desventaja a la hora de trabajar con este servicio, puesto que se convierte en un formato pesado por temas de tamaño y procesamiento, a esto se le adiciona que SOAP solo permite trabajar con formato XML, quitándole flexibilidad al trabajo realizado. Por otra parte, RESTFul fue una tecnología mucho más adaptable y sencilla de usar, debido a ser una tecnología con documentación actualizada y a su flexibilidad que permite transmitir casi cualquier tipo de datos, a diferencia de SOAP que su aplicación está estrictamente ligada al uso de XML.
Teniendo en cuenta lo anteriormente mencionado, aunque SOAP puede ser el servicio preferido para muchos gracias a su robustez, como experiencia personal, fue mucho más cómodo trabajar con RESTFul por temas de escalabilidad, flexibilidad y rendimiento.

8. CONCLUSIONES

- Se aprendió a trabajar con la metodología propuesta, consiguiendo un buen control de tareas y una flexibilidad de trabajo óptima.
- Se consiguió cada uno de los objetivos específicos. Por una parte, se logró desarrollar e implementar el aplicativo propuesto, permitiendo a los clientes registrar sus entradas y salidas al parqueadero. Por otro lado, se ejecutaron pruebas para velar por el correcto funcionamiento del aplicativo.
- Se aprendió a usar tecnologías de comunicación sobre protocolo HTTP, en el caso de este proyecto, se trabajó con protocolo HTTP basadas en Servicios Web (SOAP y RESTFul) por parte del servidor que interactúa con una aplicación cliente HTTP no tradicional. Para este proyecto se trabajó con dos clientes y dos servidores, en los que cada cliente hacía uso de al menos una funcionalidad de cada servidor, esto para que pueda existir un punto de comparación.

9. ANEXOS

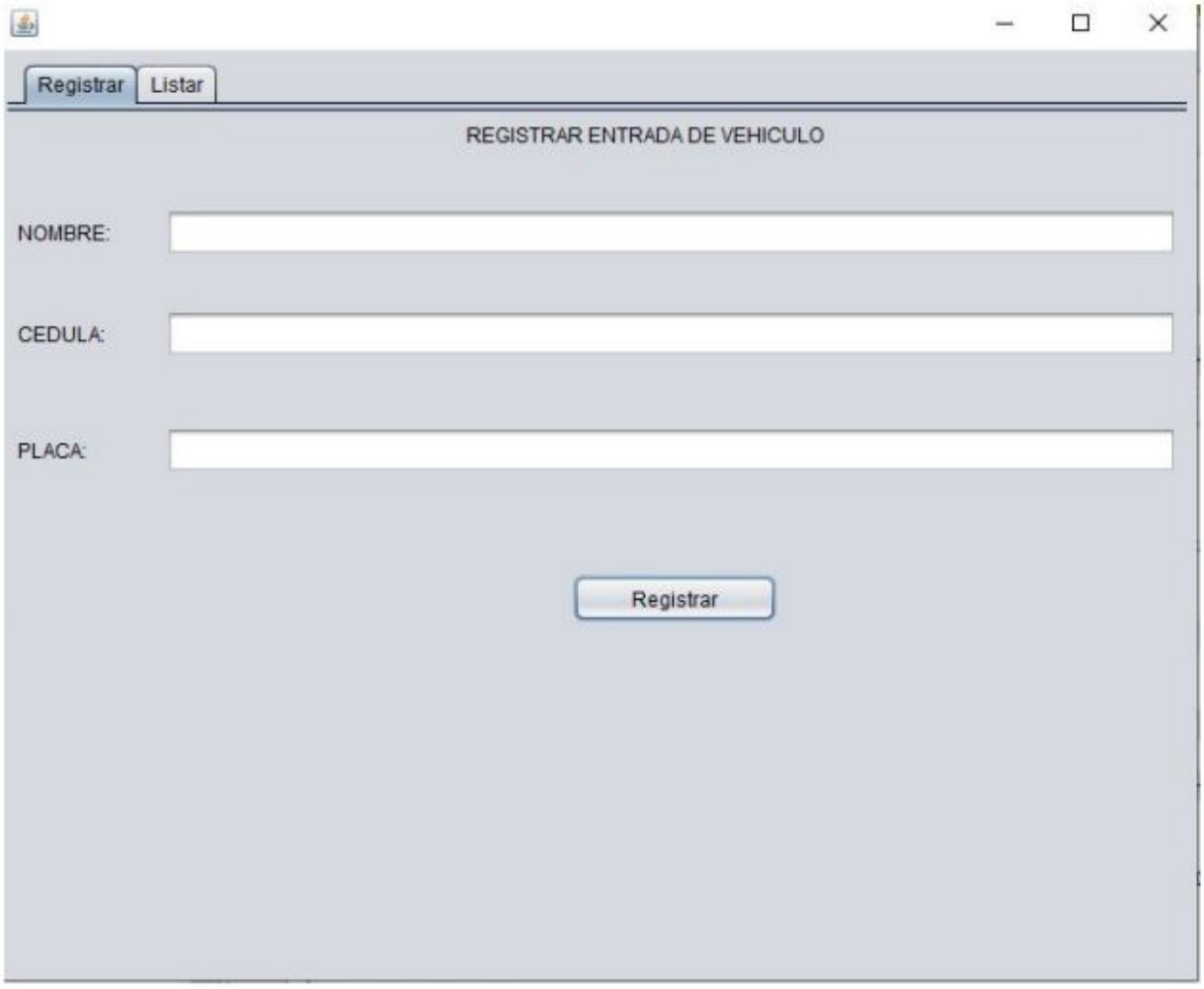

Imagen 1. Vista Registrar cliente Escritorio

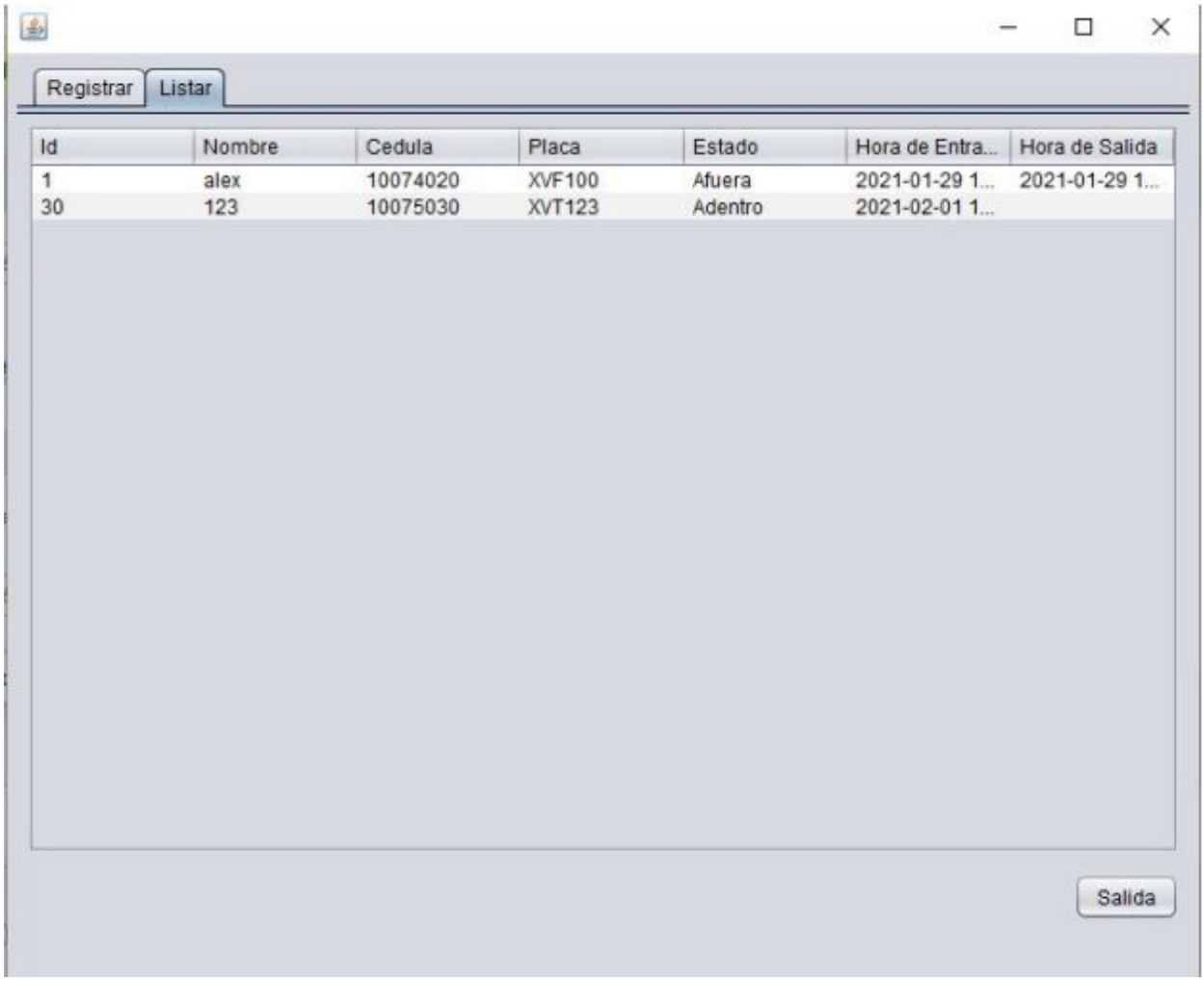

Imagen 2. Vista Listar cliente Escritorio.

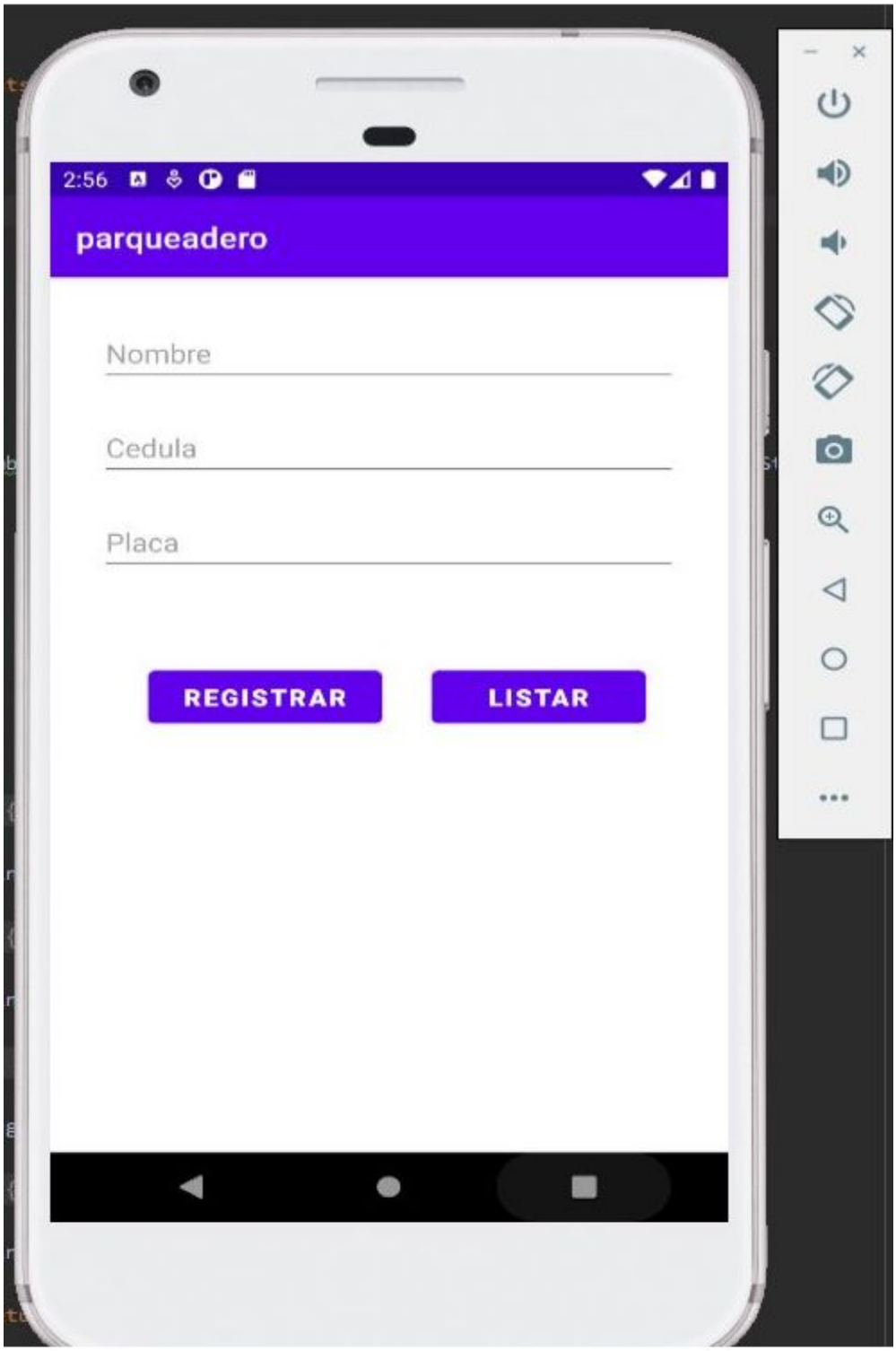

Imagen 3. Vista PantallaInicial cliente Android.

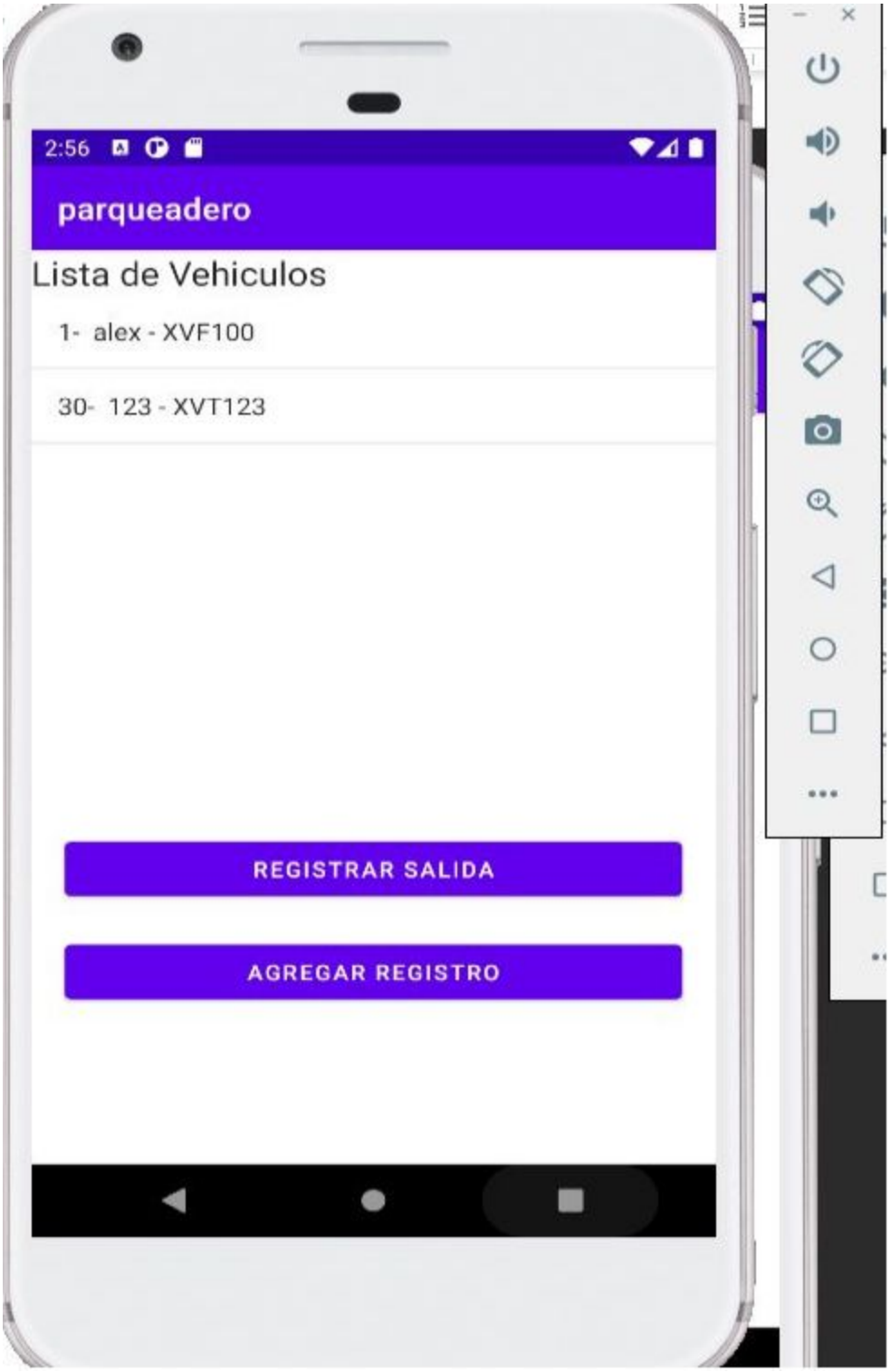

Imagen 4. Vista PantallaLista Android.

10. BIBLIOGRAFÍA

onnection (Java Platform SE 7). (s. f.). Oracle.
https://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html

DriverManager (Java Platform SE 8). (s. f.). Oracle.
https://docs.oracle.com/javase/8/docs/api/java/sql/DriverManager.html

ResultSet (Java Platform SE 7). (s. f.). Oracle.
https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html

SQLException(Java Platform SE 7). (s. f.). Oracle.
https://docs.oracle.com/javase/7/docs/api/java/sql/SQLException.html

Statement(Java Platform SE 7). (s. f.). Oracle.
https://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html

HttpURLConnection(Java Platform SE 8). (s. f.). Oracle.
https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html

URL(Java Platform SE 7). (s. f.). Oracle.
https://docs.oracle.com/javase/7/docs/api/java/net/URL.html

URLConnection(Java Platform SE 7). (s. f.). Oracle.
https://docs.oracle.com/javase/7/docs/api/java/net/URLConnection.html

Express - Node.js web application framework. https://expressjs.com/

kSOAP 2. ksoap2.sourceforge.net