

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

INTRODUCCIÓN	3
OBJETIVOS	3
OBJETIVO GENERAL	3
OBJETIVOS ESPECÍFICOS	4
JUSTIFICACIÓN	4
DESARROLLO	5
CONTEXTUALIZACIÓN	5
ARQUITECTURA DE DESARROLLO	5
METODOLOGÍA O PROCESOS DE DESARROLLO	5
ENTORNO DE TRABAJO	6
ESPECIFICACIONES	6
HERRAMIENTAS DE SOFTWARE	6
CLASES Y MÉTODOS	7
SERVIDOR PHP	7
SERVIDOR SERVLET	10
CLIENTE ANDROID	14
CLIENTE ESCRITORIO	18
COMPARATIVAS Y EXPERIENCIAS	21
CONCLUSIONES	22
ANEXOS	23

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.

En cuanto al flujo de datos, cuando un cliente desea comunicarse con el servidor, realiza los siguientes pasos: Abrir una conexión TCP, enviar un mensaje HTTP, leer la respuesta enviada por el servidor y por último, cerrar o reutilizar la sesión para más solicitudes.

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.

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 tradicional para interactuar con un cliente HTTP no tradicional.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

Desarrollar un software 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 tradicional del lado del servidor y no tradicional por parte del cliente con la finalidad de desarrollar al menos una función en dos servidores de aplicaciones y en dos clientes.

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.

Aunque el protocolo HTTP muchas veces sea comparado con otros protocolos de comunicación, resulta pertinente listar las características que hacen del protocolo HTTP una muy buena opción para el desarrollo de tecnologías:

- Ofrece poco uso de CPU y memoria.
- Permite la canalización de solicitudes y respuestas.
- Tiene una congestión de red reducida.
- Reporta errores sin cerrar la conexión TCP.

Teniendo en cuenta los beneficios que trae la implementación del protocolo HTTP, se puede crear una idea de la relevancia e importancia que este tiene en la implementación de un proyecto, haciéndola 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 tradicional 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 NetBeans IDE permitiendo la creación de dos servidores y un cliente (un servidor en Servlet, otro en PHP y el cliente de escritorio en Java) y Android Studio permitiendo la creación del cliente de app Android.

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:

- **Servlet:** Es una clase de Java utilizada para extender las capacidades de los servidores web.
- **JSON:** Es un formato de intercambio de datos constituido por una colección de pares/valor y una lista ordenada de valores.
- **PHP:** Es un lenguaje de scripting del lado del servidor utilizado para desarrollar sitios web estáticos, dinámicos o aplicaciones web.
- **ServletException:** Define una excepción general que un servlet puede lanzar cuando encuentra dificultades.
- **HttpServlet:** Extiende la clase GenericServlet y proporciona una clase abstracta que se subclasifica para crear un servlet HTTP adecuado para un sitio web.
- **HttpServletRequest:** Extiende la interfaz de ServletRequest para proporcionar información de solicitud para servlets HTTP.

- **HttpServletResponse:** Extiende la interfaz de ServletRequest para proporcionar una funcionalidad específica de HTTP al enviar una respuesta.
- **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. SERVIDOR PHP

5.1.1. Conexion

Esta clase es la encargada de establecer la conexión con la base de datos y de recibir las sentencias SQL que serán enviadas a la base de datos.

5.1.1.1. conectar()

Este método se encarga de establecer la conexión con la base de datos.

```
function conectar(){
    try{
        global $bd;
        $bd = new mysqli("localhost","root","","parqueadero");

    } catch (Exception $error){
        $msg = array("mensaje" => $error->getMessage());
        throw new Exception();
    }
}
```

5.1.1.2. consultar()

Este método se encarga de enviar las sentencias SQL a la base de datos.

```
function consultar($sql){
    global $bd;
    $res = NULL;
    try{
        if($bd==NULL){
            conectar();
        }
        return $bd->query($sql);
    }catch(Exception $error){
        $msg = array("mensaje" => $error->getMessage());
        throw new Exception($msg);
    }
}
```

5.1.2. Operation

Esta clase se encarga de recibir los datos del cliente y de elaborar las sentencias SQL dependiendo de qué función deba ejecutar.

5.1.2.1. registrar_e()

Método encargado de elaborar la sentencia SQL para registrar entradas en la base de datos, recibiendo como parámetros los datos del cliente.

```
function registrar_e() {
    $nombre = @$_REQUEST["nombre"];
    $cedula = @$_REQUEST["cedula"];
    $placa = @$_REQUEST["placa"];
    $estado = "1";
    try {
        $res = consultar("INSERT INTO `listado` (`nombre`, `cedula`, `placa`, `estado`)
        VALUES('$nombre','$cedula','$placa','$estado')");
        echo json_encode(array("mensaje" => "OK"));
    } catch (Exception $error) {
        echo json_encode(array("mensaje" => "Usuario No Existe"));
    }
}
```

5.1.2.2. registrar_s()

Método encargado de elaborar la sentencia SQL para registrar salidas en la base de datos, recibiendo como parámetros los datos del cliente.

```
function registrar_s(){
    $estado = "0";
    $id = @$_REQUEST["id"];
    try {
        $res = consultar("UPDATE `listado` SET `estado`='$estado' WHERE id='$id'");
        echo json_encode(array("mensaje" => "OK"));
    } catch (Exception $error) {
        echo json_encode(array("mensaje" => "Usuario No Existe"));
    }
}
```

5.1.2.3. listar()

Método encargado de mostrar una lista de todos los clientes registrados.

```
function listar() {
    try {
        $res = consultar("SELECT * FROM `listado`");
        if ($res != NULL && $res->num_rows > 0) {
            $json = json_encode($res->fetch_all(1));
            echo $json;
            /* liberar el conjunto de resultados */
            $res->free();
        } else {
            echo json_encode(array("mensaje" => "No hay Usuarios"));
        }
    } catch (Exception $error) {
        echo json_encode(array("mensaje" => $error->getTraceAsString()));
    }
}
```

5.2. SERVIDOR SERVLET

5.2.1. BD

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

5.2.1.1. registrarEntrada()

Método encargado de elaborar la sentencia SQL para registrar entradas en la base de datos, recibiendo como parámetros los datos del cliente.

```
public void registrarEntrada(String nombre, String cedula, String placa, String estado) {
    SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yy hh:mm:");
    String fecha = sdf.format(new Date());
    System.out.println(fecha);
    String ComandoSQL = "INSERT INTO `listado` "
        + "(`nombre`, `cedula`, `placa`, `estado`) "
        + "VALUES ('"+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."+er.getMessage());
    }
}
```

5.2.1.2. registrarSalida()

Método encargado de elaborar la sentencia SQL para registrar salidas en la base de datos, recibiendo como parámetros los datos del cliente.

```
public void registrarSalida(String id) {
    SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yy hh:mm:");
    String fecha = sdf.format(new Date());
    System.out.println(fecha);
    String ComandoSQL = "UPDATE `listado` SET `estado`=0 WHERE id="+id;
    try {

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

        stmt.close();

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

5.2.1.3. ListarRegistros()

Método encargado de mostrar una lista de todos los clientes registrados.

```
public ArrayList<Registro> ListarRegistros() {
    String nombre, cedula, placa, estado, id, fechaE, fechaS;
    float temp;
    String ComandoSQL = "SELECT * FROM listado";
    ArrayList<Registro> cts = new ArrayList<>();
    Gson js = new Gson();

    try {
        Statement stmt = conexion.createStatement();
        ResultSet resultado = stmt.executeQuery(ComandoSQL);

        while (resultado.next()) {
            nombre = resultado.getString(1);
            cedula = resultado.getString(2);
            placa = resultado.getString(3);
            estado = resultado.getString(4);
            id = resultado.getString(5);
            fechaE = resultado.getString(6);
            fechaS = resultado.getString(6);

            cts.add(new Registro(nombre, cedula, placa, estado, id, fechaE, fechaS));
        }
        stmt.close();
    } catch (java.sql.SQLException er) {
        System.out.println("No se pudo realizar la operación.");
    }
    return cts;
}
```

5.2.2. Registro

Esta clase contiene todos los atributos del cliente, junto a sus constructores y getters and setters.

5.2.3. Server

Esta clase se encarga de obtener el URL enviado por el cliente para conocer la funcionalidad a realizar, y esto luego es enviado a la base de datos.

5.2.3.1. doGet()

Este método se encarga de obtener el URL enviado por el cliente y determinar la funcionalidad a realizar.

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    String action = request.getParameter("accion");
    switch (action){
        case "entrada":
            PrintWriter out = response.getWriter();
            registrar_e(request.getParameter("nombre"),
                request.getParameter("cedula"), request.getParameter("placa"));
            String json="{\"mensaje\":\"OK\"}";
            out.println(json);

            break;
        case "salida":
            registrar_s(request.getParameter("id"));
            break;

        case "listar":
            listar(request, response);
            break;
    }
}
```

5.2.3.2. registrar_e()

Este método recoge los datos del usuario y envía la entrada a la base de datos.

```
private void registrar_e(String nombre, String cedula, String placa){
    BD bd = new BD();
    bd.registrarEntrada(nombre, cedula, placa, "1");
}
```

5.2.3.3. registrar_s()

Este método recoge los datos del usuario y envía la salida a la base de datos.

```
private void registrar_s(String id){
    BD bd = new BD();
    bd.registrarSalida(id);
}
```

5.2.3.4. listar()

Este método recoge los datos del usuario y envía una lista a la base de datos.

```
private void listar(HttpServletRequest request, HttpServletResponse response) throws IOException {
    BD bd = new BD();
    Gson gson = new Gson();
    String json=gson.toJson(bd.ListarRegistros());
    PrintWriter out = response.getWriter();
    out.println(json);
}
```

5.3. CLIENTE ANDROID

5.3.1. Registro

Esta clase contiene todos los atributos del cliente, junto a sus constructores y getters and setters.

5.3.2. Agregar

Esta es la vista encargada de registrar al cliente y enviar los datos de éste en forma de petición a los servidores.

5.3.2.1. guardar()

Este método recibe los datos de las vistas y crea el URL que se enviará como petición.

```
private void guardar(String nombre,String cedula, String placa){

    if(nombre!=null&& cedula!=null&&placa!=null){
        String url = URL+"accion="+ "entrada"+"&nombre="+nombre+"&cedula="+cedula+"&placa="+placa;
        GetXMLTask task = new GetXMLTask();
        task.execute(new String[] { url });
    }
}
```


5.3.2.2. **getOutputFromUrl()**

Este método es el encargado de recibir el URL, armar la petición y enviarla al servidor.

```
private String getOutputFromUrl(String url) {  
    StringBuffer output = new StringBuffer("");  
    try {  
        InputStream stream = getHttpConnection(url);  
        BufferedReader buffer = new BufferedReader(new InputStreamReader(stream));  
        String s = "";  
        while ((s = buffer.readLine()) != null)  
            output.append(s);  
    } catch (IOException e1) {  
        e1.printStackTrace();  
    }  
    return output.toString();  
}
```


5.3.2.3. **getHttpConnection()**

Este método crea un `HttpURLConnection` y retorna un `InputStream`.

```
private InputStream getHttpConnection(String urlString) throws IOException {  
    InputStream stream = null;  
  
    java.net.URL url = new URL(urlString);  
    URLConnection connection = url.openConnection();  
  
    try {  
        HttpURLConnection httpConnection = (HttpURLConnection) connection;  
        httpConnection.setRequestMethod("GET");  
        httpConnection.connect();  
  
        if (httpConnection.getResponseCode() == HttpURLConnection.HTTP_OK) {  
            stream = httpConnection.getInputStream();  
        }  
    } catch (Exception ex) {  
        ex.printStackTrace();  
    }  
    return stream;  
}
```

5.3.3. ConexionHttpPostServer

Esta clase permite al cliente realizar peticiones al servidor, estableciendo una conexión por medio de URL.

5.3.3.1. conexionConElServidor()

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

```
public String conexionConElServidor(List<NameValuePair> parametros, String rutaDeLaAplicacionWeb)throws Exception{
    HttpClient clienteHTTP = new HttpClient();
    BufferedReader br = null;
    PostMethod petitionPOST = new PostMethod(rutaDeLaAplicacionWeb);
    for (NameValuePair parametro: parametros) {
        petitionPOST.addParameter(parametro);
    }
    try{
        int codigoRespuesta = clienteHTTP.executeMethod(petitionPOST);
        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 = petitionPOST.getResponseBodyAsStream();
            respuesta = procesarRespuestaDelServidor();
            return (datosEntrada !=null)? respuesta: null;
        }
    }
    catch (Exception e){
        throw new Exception("Error 3: Conexion fallida: \n" + e.getMessage());
    }finally {
        petitionPOST.releaseConnection();
        if (br != null){
            try{
                br.close();
            }catch (Exception fe){
            }
        }
    }
}
```

5.3.3.2. 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;
}
```

5.3.4. PantallaLista

Esta vista muestra los usuarios listados, de los cuales se marca su salida.

5.3.4.1. mostrarUsuariosEnLista()

Este método es el encargado de mostrar la lista de los usuarios registrados en la base de datos.

```
private void mostrarUsuariosEnLista() {
    int i = 1;
    /* for (Map.Entry<String, Usuario> alguien : listaUsuarios.entrySet() ) {
        for (Registro alguien : listaR) {
            i++;
            System.out.println("No. " + i + " " + alguien);
            String elemento = alguien.getId()+"- "+alguien.getNombre() + " - " + alguien.getPlaca();
            items.add(elemento);
        }
        listaRegistro.setAdapter(items);
    }
}
```

5.3.4.2. procesarRespuestaPetición()

Este método es el encargado de remapear una lista de las peticiones del servidor, convirtiéndola de Gson a una lista de tipo Registro.

```
private boolean procesarRespuestaPetición() {
    ArrayList<NameValuePair> listaParametros = new ArrayList<>();
    NameValuePair parametro = new NameValuePair( name: "accion", value: "listar");
    listaParametros.add(parametro);
    try {
        String resultadoDelServidor = conexionServidor.conexionConElServidor(listaParametros,
            conexionServidor.direccionDelServidor);

        if (resultadoDelServidor != null && resultadoDelServidor.length() > 0) {
            System.out.println("JSON: " + resultadoDelServidor);
            Gson json = new Gson();
            Type lista = new TypeToken<Map<String, Usuario>>().getType();
            Type listaR = new TypeToken<List<Registro>>().getType();
            listaR = json.fromJson(resultadoDelServidor, listaR);
            return true;
        } else {
            return false;
        }
    } catch (Exception error) {
        error.printStackTrace();
        Toast.makeText( context: this, error.getMessage(), Toast.LENGTH_LONG).show();
        return false;
    }
}
```

5.4. CLIENTE ESCRITORIO

5.4.1. ConexionHttp

Esta clase permite que el cliente realice peticiones al servidor, estableciendo una conexión mediante el URL.

5.4.1.1. listar()

Método que permite realizar una petición de la acción listar, el cual retorna un JSON, que se obtiene por medio de JSONArray, que posteriormente es remapeado a una lista de tipo Registro.

```
public List<Registro> listar() throws InterruptedException, ExecutionException, TimeoutException {
    HttpClient client = HttpClient.newHttpClient();
    HttpRequest request = HttpRequest.newBuilder().uri(URI.create(
        ("http://localhost/parqueadero/crud/operation.php?accion=listar")).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"), album.getString("estado"),
            album.getString("id"), album.getString("horaEntrada"),
            album.getString("horaSalida").toString()));
    }
    return l;
}
```

5.4.1.2. salida()

Método que permite realizar una petición de la acción salida, el cual retorna un JSON, que se obtiene por medio de JSONArray.

```
public void salida(String id) {
    HttpClient client = HttpClient.newHttpClient();
    HttpRequest request = HttpRequest.newBuilder().uri(URI.create(
        ("http://localhost/parqueadero/crud/operation.php?accion=salida&id="
            +id)).build());
    client.sendAsync(request, HttpResponse.BodyHandlers.ofString())
        .thenApply(HttpResponse::body).join();
}
```

5.4.1.3. entrada()

Método que permite realizar una petición de la acción listar, el cual retorna un JSON, que se obtiene por medio de JSONArray, que posteriormente es remapeado a una lista de tipo Registro.

```
public void entrada(String nombre, String cedula, String placa){
    HttpClient client = HttpClient.newHttpClient();
    HttpRequest request = HttpRequest.newBuilder().uri(URI.create
    ("http://localhost:8090/server/Server?accion=entrada&nombre="+
    nombre+"&cedula="+cedula+
    "&placa="+placa)).build();
    client.sendAsync(request, HttpResponse.BodyHandlers.ofString())
        .thenApply(HttpResponse::body).join();
}
```

6. COMPARATIVAS Y EXPERIENCIAS

Alexis Bedoya Arias: A partir del desarrollo de este proyecto se obtiene un claro diferenciador en la implementación de las tecnologías del servidor, siendo el caso el uso de diferentes lenguajes (PHP y servlet java) de programaciones para ofrecer las funcionalidad a los clientes, en el cual se destaca el despliegue del servicio, siendo servlet la que necesita de un contenedor como apache tomcat para poder funcionar, este funcionara como una extensión del mismo servidor, terminando el servicio si ocurre algún error fatal. en caso contrario encontramos a PHP siendo de solicitudes con respuestas mucho más rápidas que la de servlet y algo más flexible con los posibles errores, este lenguaje es básico y conlleva a escribir menos lineas de codigo pero al costo de ofrecen menos funcionalidades que servlet java como es el caso de la robustez, multiplataforma y reusabilidad.

En los clientes no encontramos prácticamente casi ninguna diferencia puesto a que se hace uso del protocolo HTTP para la comunicación y el formato JSON para los mensajes, ambos clientes utilizan el lenguaje java (Reutilización de código y librerías), diferenciándose por el entorno de desarrollo (NetBeans y Android Studio), y por el manejo de las vista, siendo el caso android studio el que conlleva más problemas puesto a que no se contaba con experiencia previa, para el manejo de las vistas.

Antonio Cortés Sampayo: Al hacer uso de la tecnología de comunicación requerida(Protocolo HTTP) hemos podido evidenciar las diferencias en el desarrollo de cada uno de los servidores y clientes requeridos en este proyecto, la forma de desarrollar las funcionalidades, las conexiones a la base de datos e incluso la conexión cliente-servidor son similares sin importar el lenguaje en el que se trabajó, a pesar de que algunos son más flexibles que otros. Se desarrollaron dos servidores, uno en PHP basado en solicitudes y otro en Java Servlet basado en el uso de Apache Tomcat y siendo este una extensión del servidor 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 elaborarlo con dos servidores y dos clientes diferentes para tener un punto de comparación en el proceso de desarrollo. Por parte de los servidores, uno fue trabajado en PHP, mientras que el otro fue trabajado con Java Servlet; en ambos casos se aplicaron las mismas funcionalidades con la misma base de datos. Como punto de comparación y experiencia de uso entre estos dos servidores, por el lado de PHP la facilidad y el uso previo de éste ayudó mucho en su desarrollo, siendo también un lenguaje más funcional,

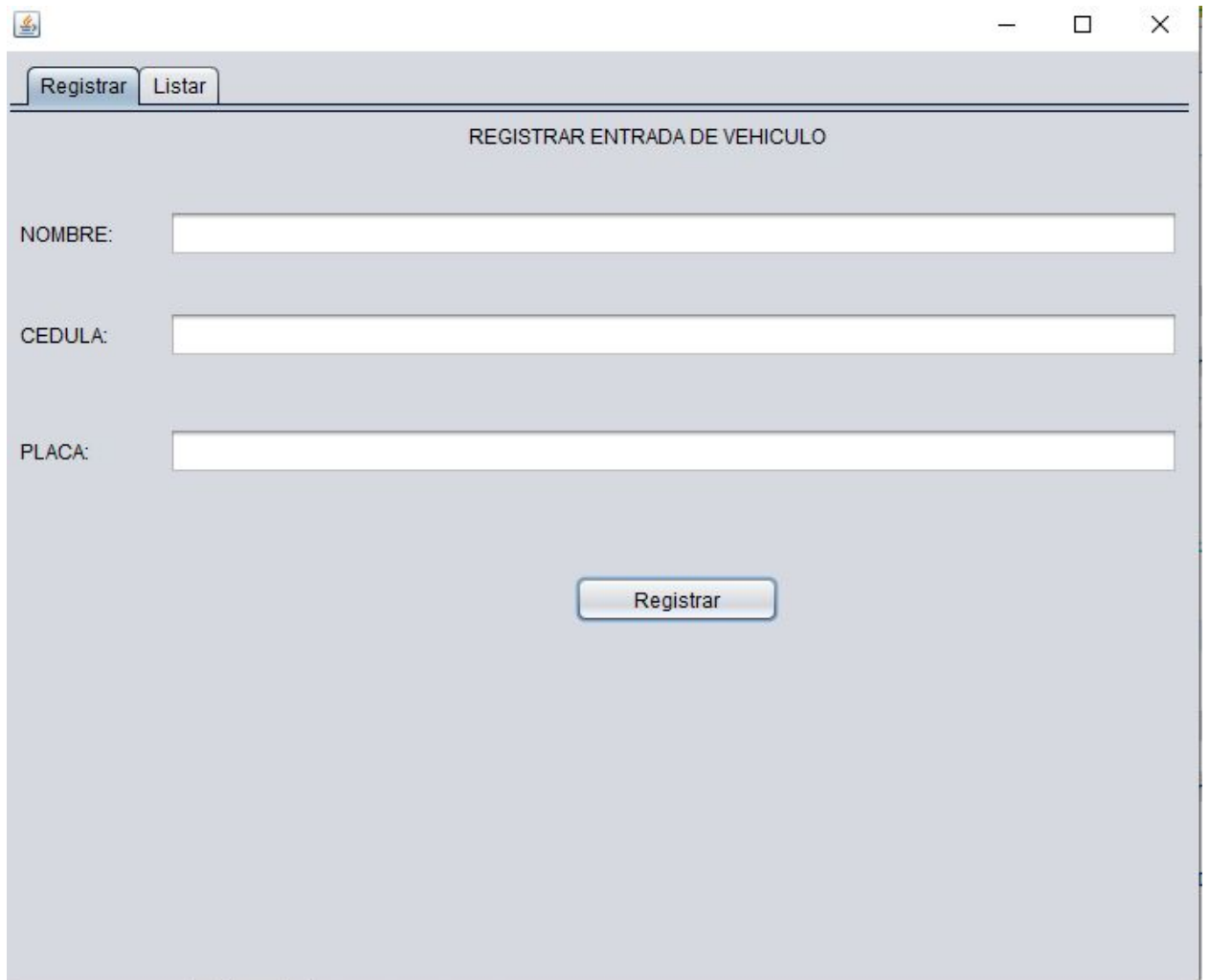
por parte del servidor con Java Servlet, la principal ventaja encontrada en el proceso de desarrollo fue la calidad y cantidad de documentación encontrada.

Por parte de los clientes, uno fue desarrollado como aplicación de escritorio haciendo uso de NETBEANS IDE y otro fue desarrollado como aplicación móvil haciendo uso de Android Studio. Comparando estos clientes, la experiencia fue mucho mejor por el lado de la aplicación de escritorio, no solo por temas de documentación, sino también por la experiencia de trabajo con esta herramienta, además de que se trata de un lenguaje por parte del cliente. Por parte de Android Studio, la experiencia fue un poco más de aprender en el proceso de desarrollo, ya que no se habían realizado muchos trabajos con esta herramienta.

7. 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 tradicional 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.

8. ANEXOS



REGISTRAR ENTRADA DE VEHICULO

NOMBRE:

CEDULA:

PLACA:

Registrar

Imagen 1. Vista Registrar cliente Escritorio.

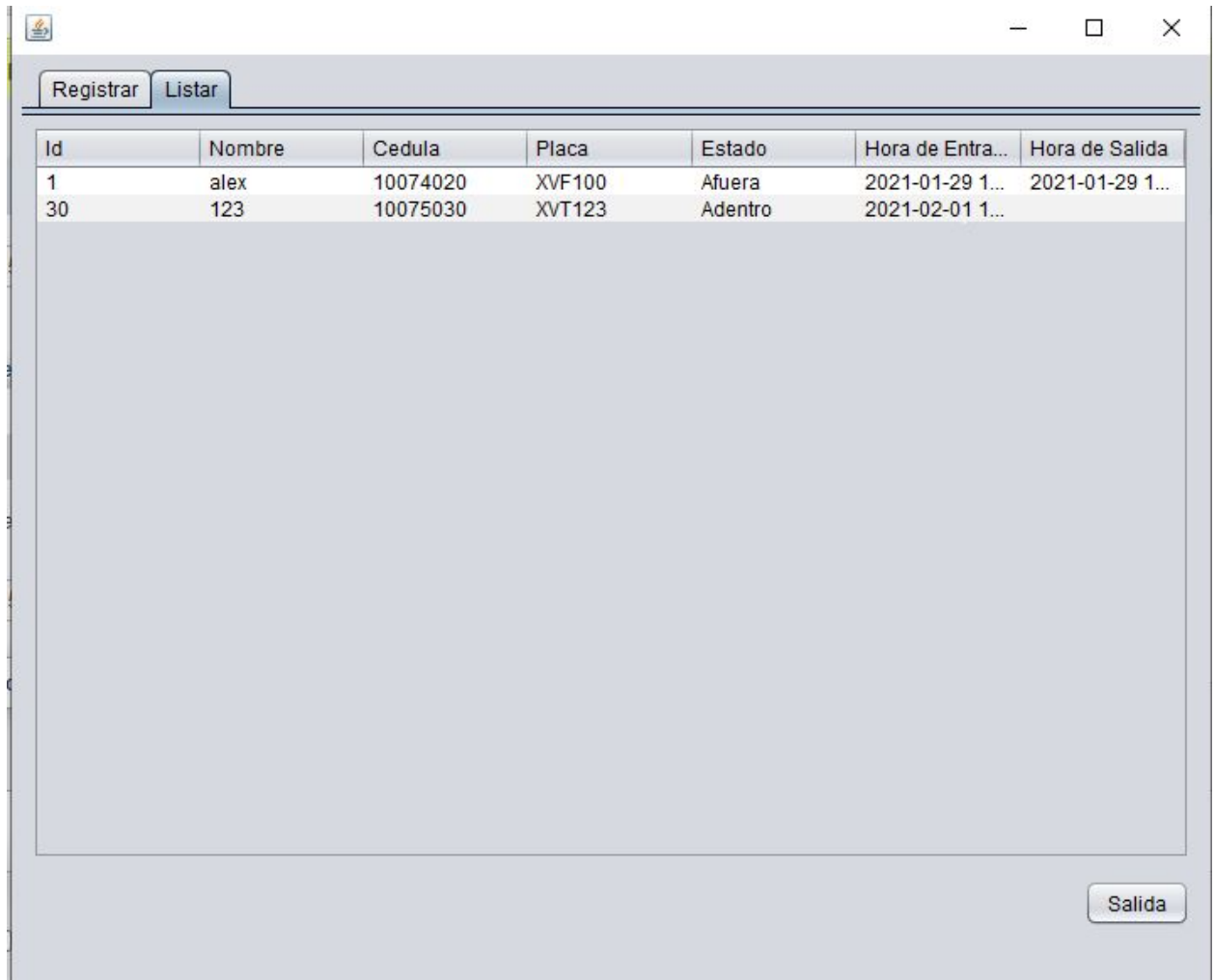


Imagen 2. Vista Listar cliente Escritorio.

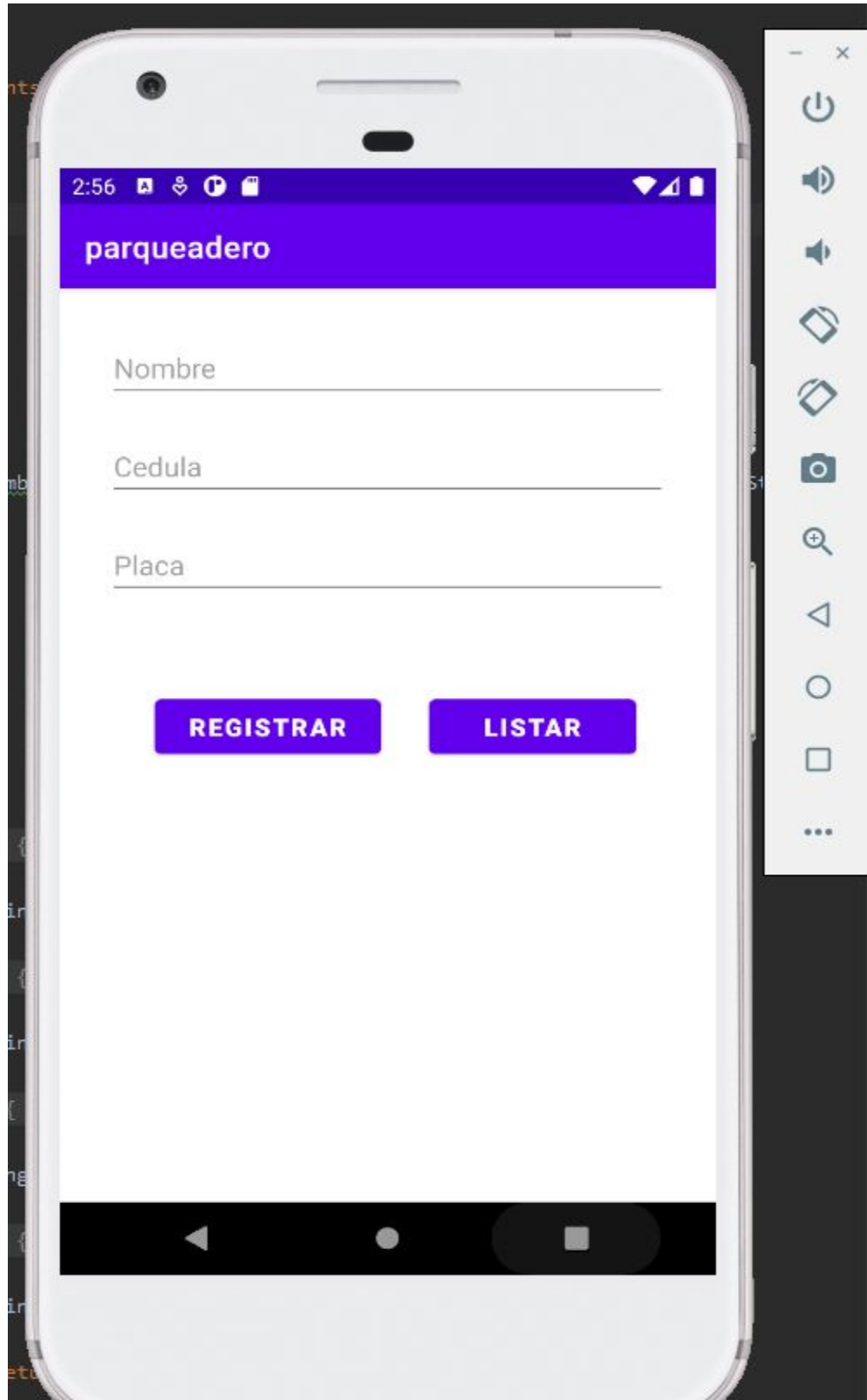


Imagen 3. Vista PantallaInicial cliente Android.

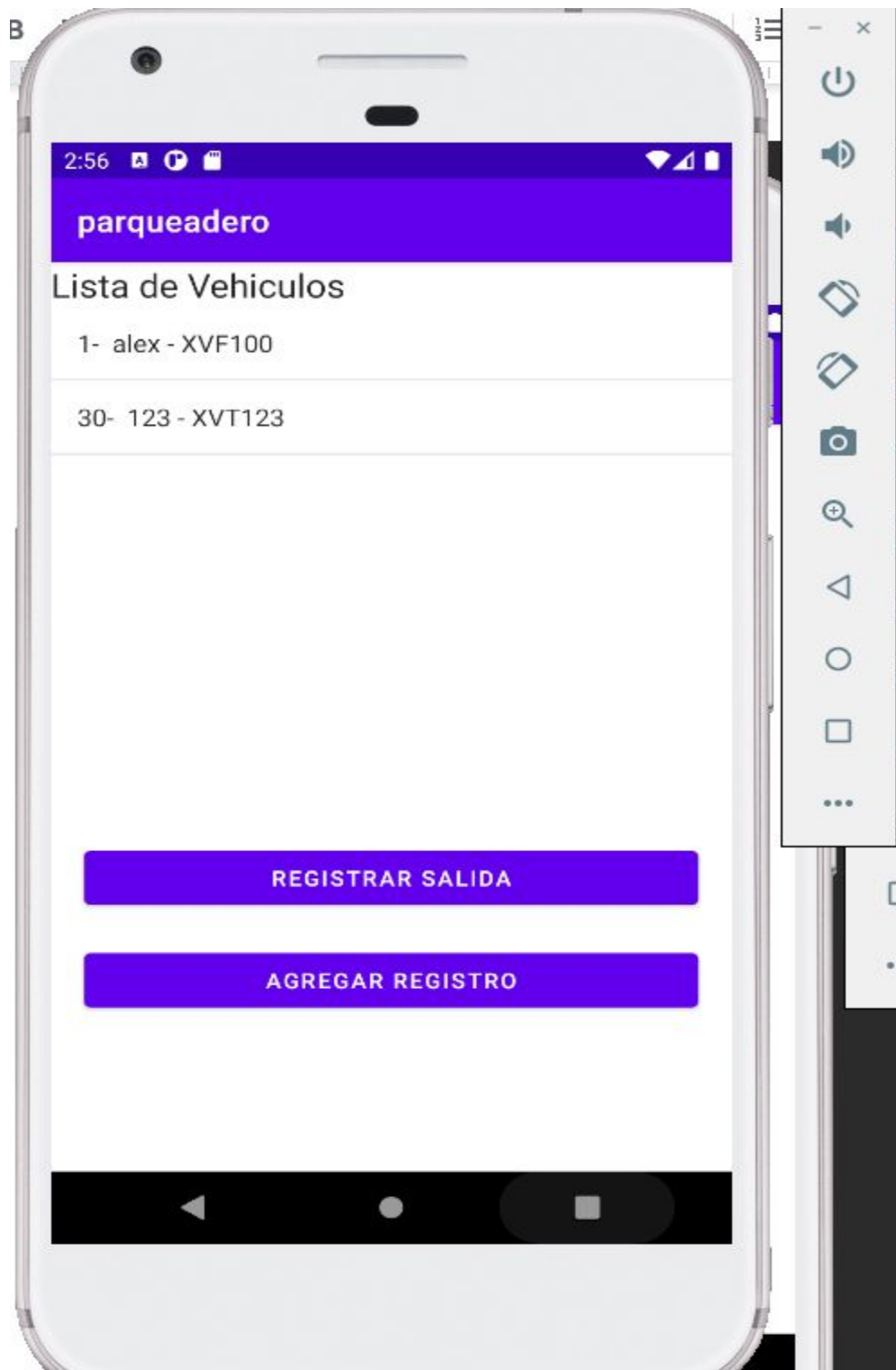


Imagen 4. Vista PantallaLista Android.

9. BIBLIOGRAFÍA

- ServletException (Java(TM) EE 7 Specification APIs). (s. f.). Oracle.**
<https://docs.oracle.com/javaee/7/api/javax/servlet/ServletException.html>
- HttpServlet (Java(TM) EE 7 Specification APIs). (s. f.). Oracle.**
<https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html>
- HttpServletRequest (Java(TM) EE 6 Specification APIs). (s. f.). Oracle.**
<https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html>
- HttpServletResponse (Java(TM) EE 6 Specification APIs). (s. f.). Oracle.**
<https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html>
- Connection (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>
- URLConnection (Java Platform SE 8). (s. f.). Oracle.**
<https://docs.oracle.com/javase/8/docs/api/java/net/URLConnection.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>

