Proyecto 2do Bimestre

Grupo:
JDB
Integrantes:

- Luisa Bermeo
- Juan Gahona
- David Paredes
- Daniel Ulloa

Contenido

Laboratorio 2.2

Descripción

La aplicación simula ser un módulo de gestión para generar archivos de salida en diferentes formatos, que contengan la información de los locales que se encuentran repartidos en las diferentes ciudades.

Para llevar a cabo esta finalidad se hizo uso de la base de datos planteada en el desarrollo del proyecto del 1er bimestre, teniendo en cuenta los cambios planteados en el laboratorio 1.1 de l desarrollo del proyecto del bimestre en curso.

Esquema de la base de datos

En la imagen siguiente se puede observar el esquema de la base de datos, como previamente se mencionó, se hace uso del esquema planteado inicialmente para la aparición.

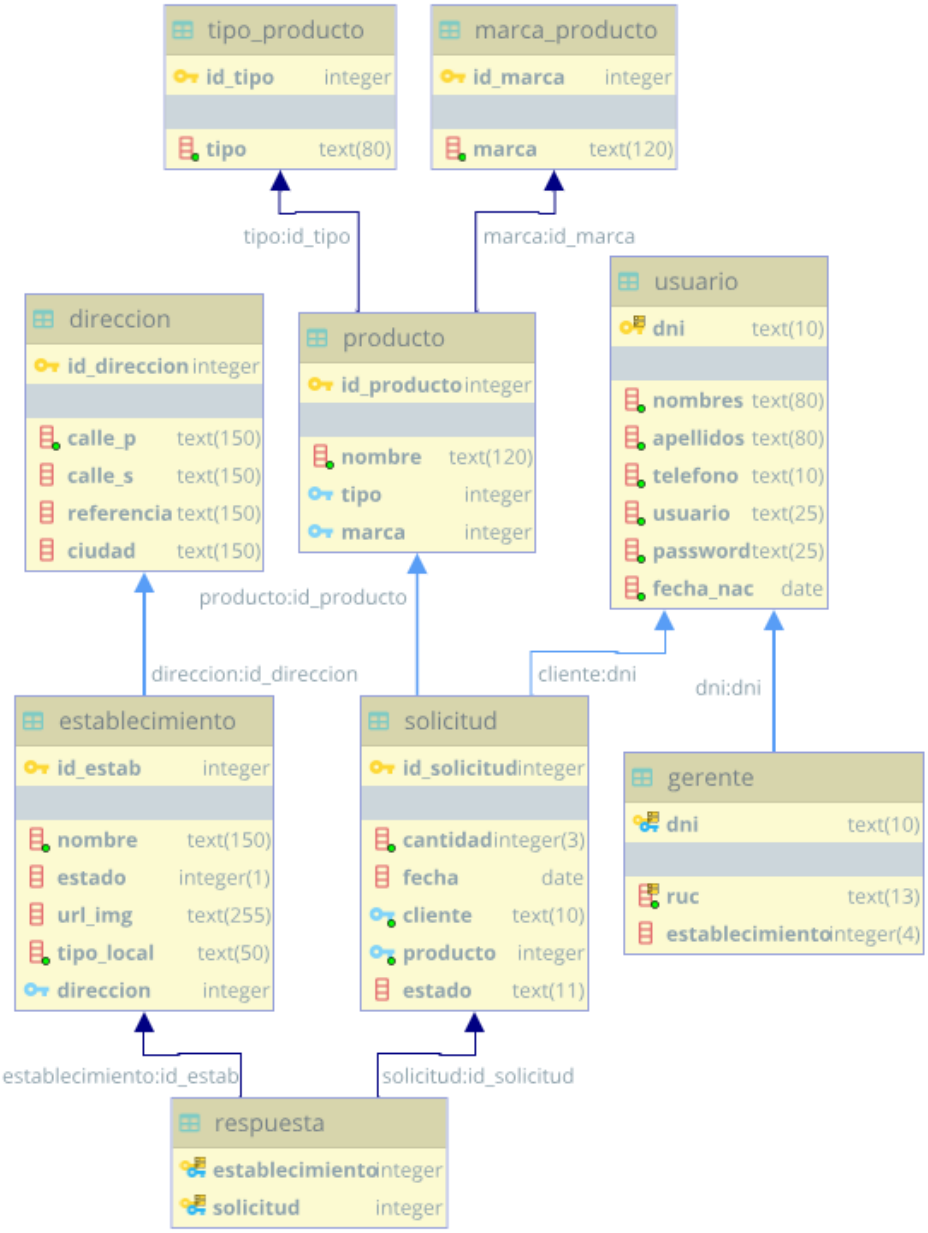

Fig. 1 – Esquema de la base de datos de Find It!

La implementación de la misma en el desarrollo de la aplicación fue mediante SQLite, ya que nos permite mantener una base de datos a nivel local en un simple, pero potente, fichero único.

Código

Obtención de los datos

```java
/**
 * Método encargado de obtener la información de los locales, desde la base de datos.
 *
 * @param city ciudad por la que se filtra la búsqueda.
 * @return una lista de locales.
 */
public List<LocalDetail> getLocals(String city) {
    // Lista vacía
    List<LocalDetail> locals = new ArrayList<>();
    // Consulta SQL
    String query = "SELECT es.id_estab, es.nombre, es.tipo_local, di.calle_p, di.calle_s, di.referencia, di.ciudad\n" +
            "FROM establecimiento es,\n" +
            "     direccion di\n" +
            "WHERE di.id_direccion = es.direccion";

    // Si la opción de ciudad es diferente de ALL, se añade una condición mas al WHERE
    if (!city.equals("ALL")) {
        query += "\n    AND di.ciudad = ?";
    }

    try {
        this.connect(); // Conectamos la DB
        // Preparamos la consulta
        PreparedStatement st = this.conn.prepareStatement(query);
        if (!city.equals("ALL")) {
            st.setString(1, city);
        }
        ResultSet res = st.executeQuery(); // Ejecutamos la consulta

        // Recorremos los resultados de la consulta
        while (res.next()) {
            // Creamos y añadimos los locales
            LocalDetail local = new LocalDetail(
                    res.getInt("id_estab"),
                    res.getString("nombre"),
                    res.getString("tipo_local"),
                    res.getString("calle_p"),
                    res.getString("calle_s"),
                    res.getString("referencia"),
                    res.getString("ciudad")
            );
            locals.add(local);
        }
    } catch (SQLException ex) {
        System.err.println(ex.getMessage());
    }
    return locals; // Retornamos la lista de locales
}
```

Fig. 2 – Fragmento de código donde se obtiene la información desde la DB.

Archivo de salida CSV

```java
/**
 * Método que genera un archivo de salida en formato CSV.
 *
 * @param locals lista de locales.
 */
public static void saveCSV(List<LocalDetail> locals) {
    // Nombres de columnas
    String[] header = new String[]{"id_estab", "nombre", "tipo_local", "calle_p", "calle_s", "referencia", "ciudad"};

    try {
        // Usamos la clase CSVWriter de OpenCSV
        CSVWriter writer = new CSVWriter(new FileWriter(PATH_CSV));
        // Escribimos la primera linea de nombres de columna
        writer.writeNext(header);

        // Recorremos la lista de locales
        for (LocalDetail local : locals) {
            // Por cada local escribimos una fila de elementos en el archivo de salida
            writer.writeNext(new String[]{
                    String.valueOf(local.getIdEstab()),
                    local.getNombre(),
                    local.getTipoLocal(),
                    local.getCalleP(),
                    local.getCalleS(),
                    local.getReferencia(),
                    local.getCiudad()
            });
        }
        // Cerramos el archivo para que los cambios se guarden
        writer.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Fig. 3 – Fragmento de código donde se guarda el archivo en formato CSV.

Archivo de salida JSON

```java
/**
 * Método que genera un archivo de salida en formato JSON.
 *
 * @param locals lista de locales.
 */
public static void saveJSON(List<LocalDetail> locals) {
    // Escribimos un caracter [ para iniciar una lista en JSON
    StringBuilder localsJson = new StringBuilder("[");

    // Usamos la clase Gson de Gson - Google
    Gson gson = new Gson();

    // Recorremos la lista de locales
    for (int i = 0; i < locals.size(); i++) {
        // Instanciamos cada local
        LocalDetail local = locals.get(i);
        // Lo añadimos al string de salida JSON, mediante el método toJson de Gson
        // convierte cada objeto LocalDetail en una representación en formato JSON
        localsJson.append(gson.toJson(local));
        // Se separan por coma cada elemento, menos el ultimo
        if (i != 0) {
            localsJson.append(",");
        }
    }
    // Añadimos un caracter ] para cerrar la lista en JSON
    localsJson.append("]");

    // Creamos un fichero de escritura
    try (BufferedWriter bw = new BufferedWriter(new FileWriter(PATH_JSON))) {
        // Escribimos el String que contien la lista de ciudades en JSON
        bw.write(localsJson.toString());
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Fig. 4 – Fragmento de código donde se guarda el archivo en formato JSON.

Resultado final

Aplicación final

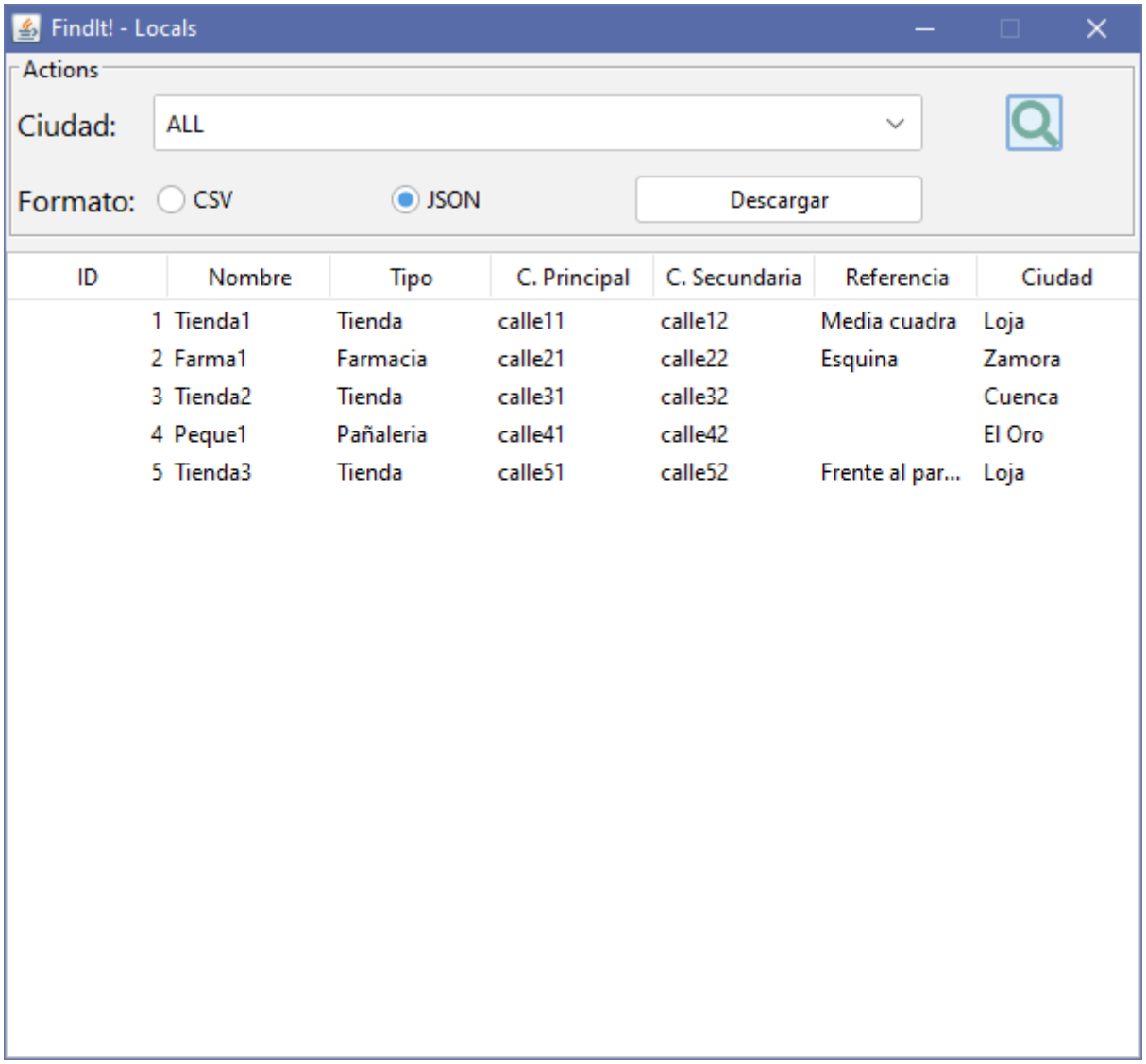

Fig. 5 – Vista general de la aplicación.

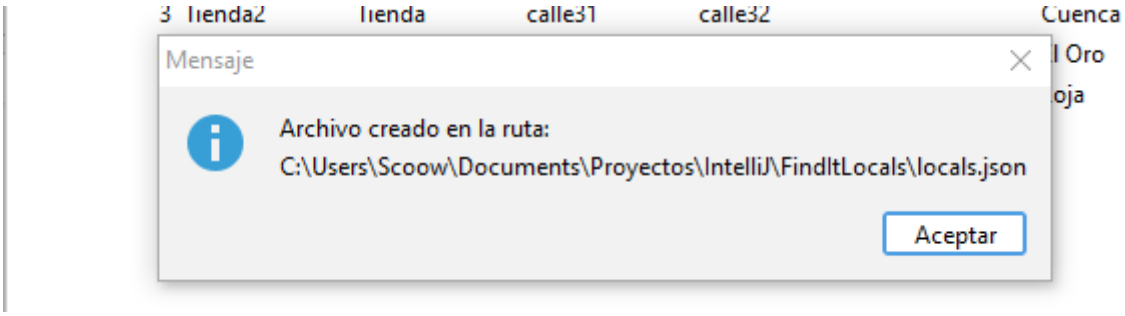

Fig. 6 – Mensaje de archivo generado.

Archivos de salida

```
"id_estab","nombre","tipo_local","calle_p","calle_s","referencia","ciudad"
"1","Tienda1","Tienda","calle11","calle12","Media cuadra","Loja"
"2","Farma1","Farmacia","calle21","calle22","Esquina","Zamora"
"3","Tienda2","Tienda","calle31","calle32",,"Cuenca"
"4","Peque1","Pañaleria","calle41","calle42",,"El Oro"
"5","Tienda3","Tienda","calle51","calle52","Frente al parque","Loja"

```

Fig. 7 – Archivo de salida en formato CSV.

```json
[
  {
    "idEstab": 1,
    "nombre": "Tienda1",
    "tipoLocal": "Tienda",
    "calleP": "calle11",
    "calleS": "calle12",
    "referencia": "Media cuadra",
    "ciudad": "Loja"
  }
  {
    "idEstab": 2,
    "nombre": "Farma1",
    "tipoLocal": "Farmacia",
    "calleP": "calle21",
    "calleS": "calle22",
    "referencia": "Esquina",
    "ciudad": "Zamora"
  },
  {
    "idEstab": 3,
    "nombre": "Tienda2",
    "tipoLocal": "Tienda",
    "calleP": "calle31",
    "calleS": "calle32",
    "ciudad": "Cuenca"
  },
  {
    "idEstab": 4,
    "nombre": "Peque1",
    "tipoLocal": "Pañaleria",
    "calleP": "calle41",
    "calleS": "calle42",
    "ciudad": "El Oro"
  },
  {
    "idEstab": 5,
    "nombre": "Tienda3",
    "tipoLocal": "Tienda",
    "calleP": "calle51",
    "calleS": "calle52",
    "referencia": "Frente al parque",
    "ciudad": "Loja"
  }
]
```


Fig. 8 – Archivo de salida en formato JSON.