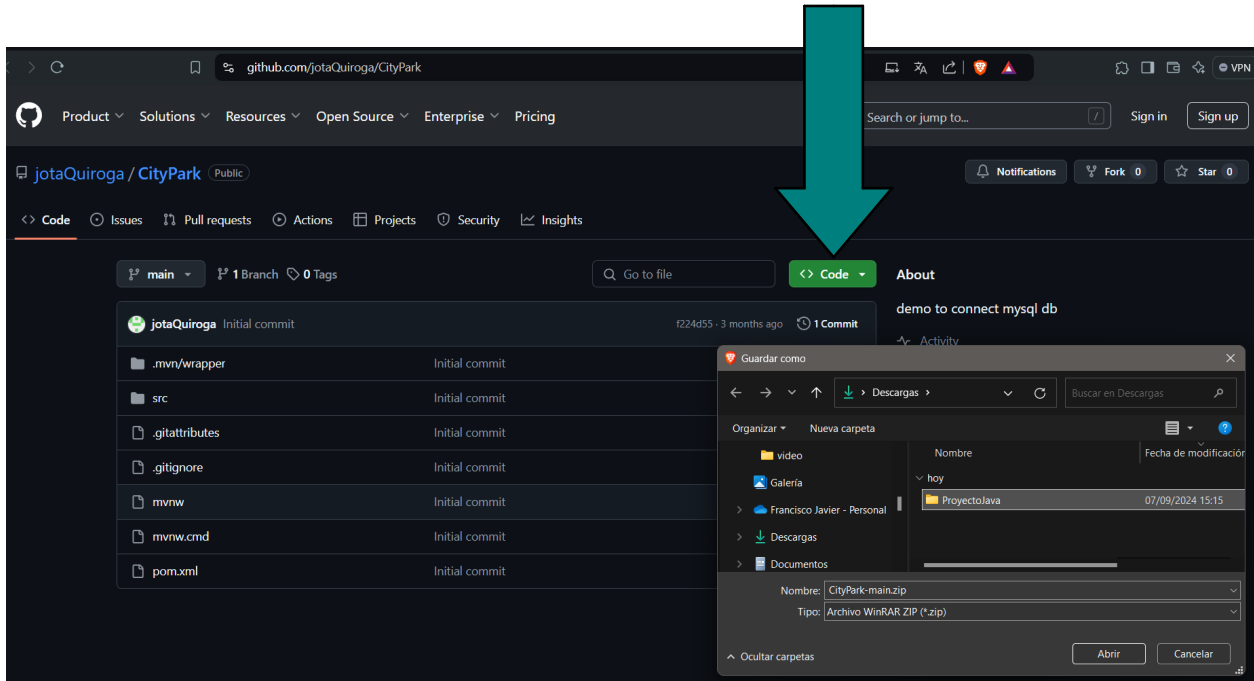


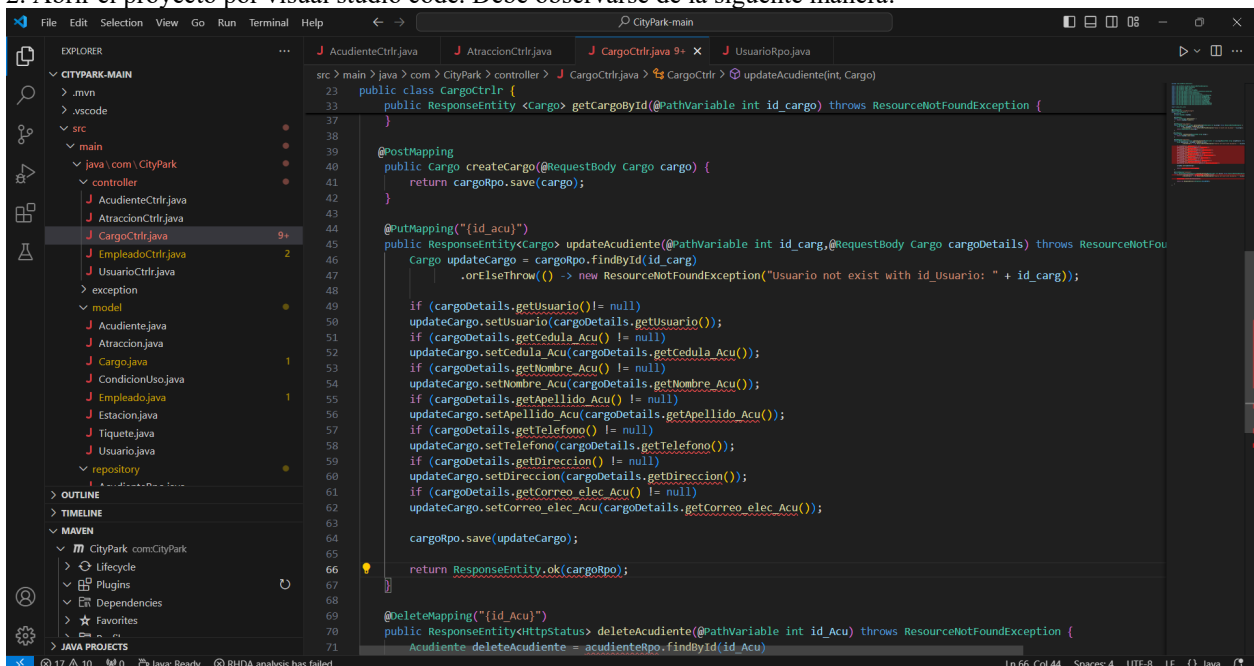
CREAR APLICATIVO DE PUNTA A PUNTA

1. Se utiliza el prototipo que se encuentra publicado en el github <https://github.com/jotaQuiroga/CityPark>

Dar click en el boton señalado y utilizar la opción descargar. Luego descomprimir el archivo.



2. Abrir el proyecto por visual studio code. Debe observarse de la siguiente manera:



Nota: Es posible que el entorno de desarrollo solicite actualización de Plugins para poder utilizar el proyecto, por lo tanto se debe actualizar lo que requiera.

3. Este proyecto está bajo la arquitectura MVC, que consiste en agrupar en 3 grandes grupos toda su funcionalidad. La primera parte corresponde al Modelo, que es la representación de nuestro modelo de datos. En este proyecto corresponden al grupo de clases Model y Repository.

src	23
main	33
java\com\CityPark	37
model	38
Acudiente.java	39
Atraccion.java	40
Cargo.java	41
CondicionUso.java	42
Empleado.java	43
Estacion.java	44
Tiquete.java	45
Usuario.java	46
repository	47
AcudienteRpo.java	48
AtraccionRpo.java	49
CargoRpo.java	50
CondicionUsoRpo.java	51
EmpleadoRpo.java	52
EstacionRpo.java	53
TiqueteRpo.java	54
UsuarioRpo.java	55
CityParkApplication.java	56

La segunda parte es el Controlador, corresponde al grupo de lógica de negocio, es decir, las acciones objetivo que realizará nuestro proyecto, el grupo de clases se ubican en la sección Controller.

.mvn	
.vscode	
src	
main	
java\com\CityPark	
controller	
AcudienteCtrlr.java	
AtraccionCtrlr.java	
CargoCtrlr.java	9+
EmpleadoCtrlr.java	2
UsuarioCtrlr.java	
exception	
model	
repository	
CityParkApplication.java	

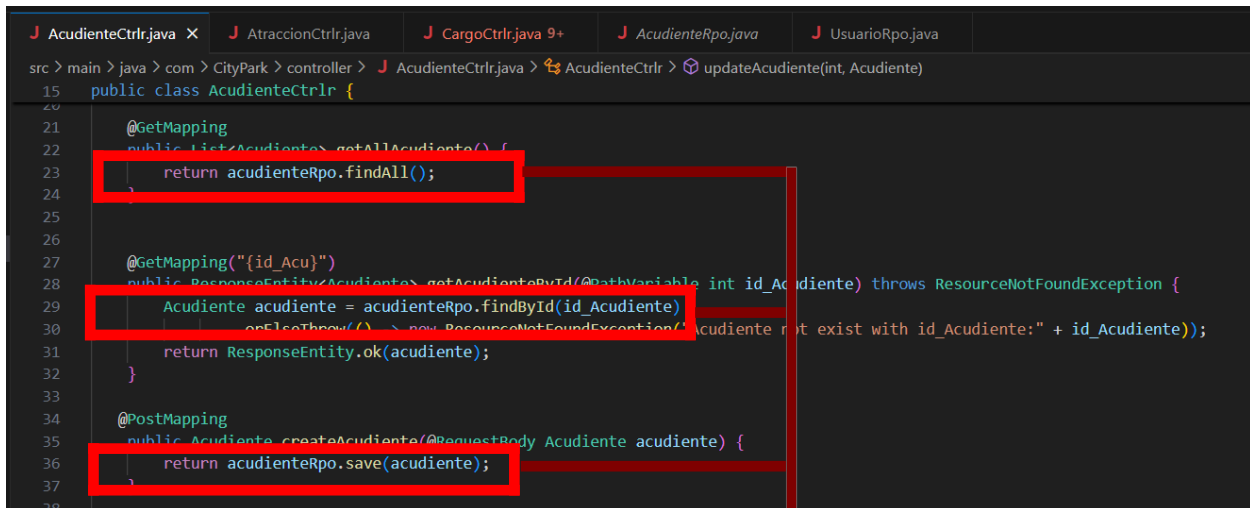
Java brinda una estrategia de desarrollo para interactuar con una base de datos relacional llamado JPA (Java Persistence API), esta estrategia ya cuenta con la lógica para realizar las funciones básicas de interacción con la base de datos llamada CRUD (Create, Read, Update, Delete). Esta estrategia fue utilizada en este proyecto.

```
package com.CityPark.repository;

import com.CityPark.model.Acudiente;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AcudienteRpo extends JpaRepository<Acudiente, Integer> {
}

```



```
src > main > java > com > CityPark > controller > AcudienteCtrlr > updateAcudiente(int, Acudiente)
15 public class AcudienteCtrlr {
20
21     @GetMapping
22     public List<Acudiente> getAllAcudiente() {
23         return acudienteRpo.findAll();
24     }
25
26
27     @GetMapping("/{id_Acu}")
28     public ResponseEntity<Acudiente> getAcudienteById(@PathVariable int id_Acudiente) throws ResourceNotFoundException {
29         Acudiente acudiente = acudienteRpo.findById(id_Acudiente)
30             .orElseThrow(() -> new ResourceNotFoundException("Acudiente not exist with id_Acudiente:" + id_Acudiente));
31         return ResponseEntity.ok(acudiente);
32     }
33
34     @PostMapping
35     public Acudiente createAcudiente(@RequestBody Acudiente acudiente) {
36         return acudienteRpo.save(acudiente);
37     }
38

```

Funciones ofrecidas por la estrategia JPA

4. Este proyecto cuenta también con otra ayuda adicional, con el fin de evitar la necesidad de realizar actividades redundantes como declarar los getters y setters, los constructores, y otras ayudas. Se denomina proyecto LOMBOK; la manera en que evitamos estas acciones repetitivas es mediante una estrategia que se llama Anotaciones, las cuales funcionan de la siguiente manera: al colocarlas dentro de nuestro código, es como si declaráramos los métodos. La manera en que identificamos que hemos utilizado una anotación es porque las precede el símbolo @.

```
package com.CityPark.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.persistence.*;
import java.util.Set;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "CONDICIONUSO")

public class CondicionUso {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id_condicionUso;

    @Column(name = "descripcion")
    private String descripcion;

    @ManyToMany(fetch = FetchType.LAZY,
        cascade = { CascadeType.PERSIST, CascadeType.MERGE, CascadeType.DETACH, CascadeType.REFRESH })
    @JoinTable(name = "atraccion_condicionuso_map", joinColumns = @JoinColumn(name = "id_cond_uso"),
        inverseJoinColumns = @JoinColumn(name = "id_atrac"))
    private Set<Atraccion> atraccionSet;
}

```

5. La estrategia de esta aplicación en su comunicación entre la parte back y la parte front es a través de microservicios, donde la parte front realiza peticiones de tipo GET, POST, PUT, DELETE a la parte back, las cuales son respondidas con datos en formato JSON. Esta estrategia de comunicación entre el back y el front se denomina servicios REST.

```

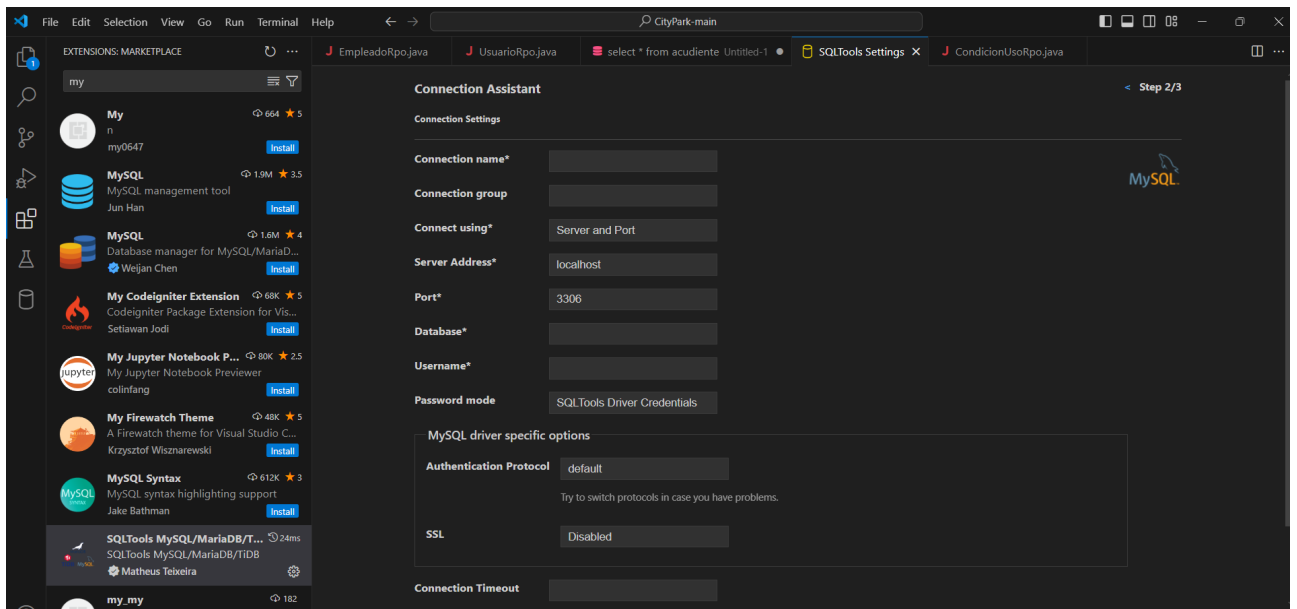
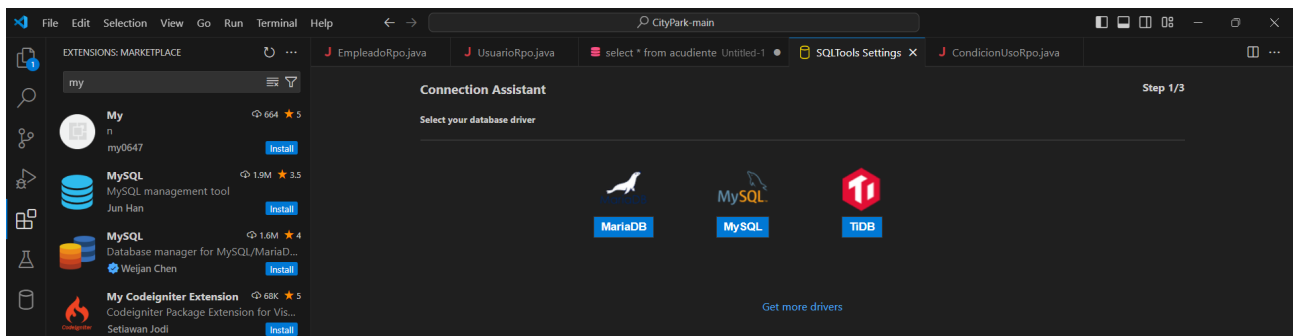
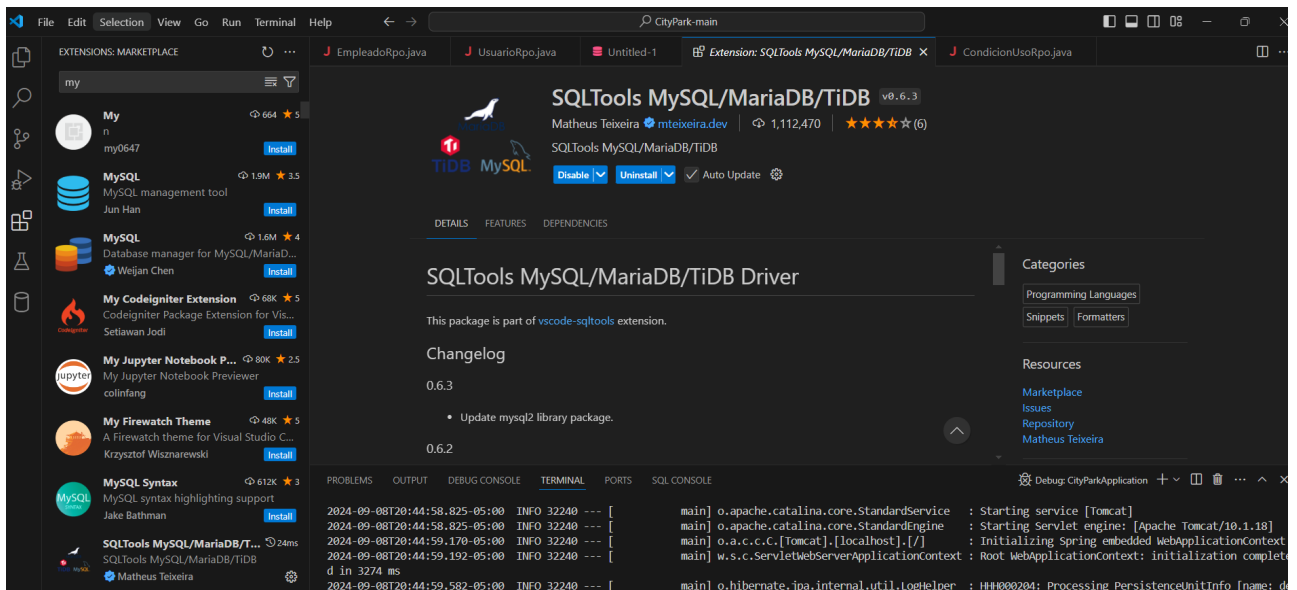
J AcudienteCtrlr.java  J AtraccionCtrlr.java  J CargoCtrlr.java 9+ x  J CondicionUso.java  J UsuarioRpo.java
src > main > java > com > CityPark > controller > J CargoCtrlr.java > CargoCtrlr > updateAcudiente(int, Cargo)
19 import java.util.List;
20
21 @RestController
22 @RequestMapping("/cityPark/cargo")
23 public class CargoCtrlr {
24     @Autowired
25     private CargoRpo cargoRpo;
26
27     @GetMapping
28     public List<Cargo> findAll() {
29         return cargoRpo.findAll();
30     }
31
32     @GetMapping("/{id_cargo}")
33     public Cargo findById(@PathVariable int id_cargo) throws ResourceNotFoundException {
34         Cargo cargo = cargoRpo.findById(id_cargo)
35             .orElseThrow(() -> new ResourceNotFoundException("Cargo no existe con id_Cargo:" + id_cargo));
36         return ResponseEntity.ok(cargo);
37     }
38
39     @PostMapping
40     public Cargo save(Cargo cargo) {
41         return cargoRpo.save(cargo);
42     }
43
44     @PutMapping("/{id_acu}")
45     public Cargo updateAcudiente(@PathVariable int id_carg, @RequestBody Cargo cargoDetails) throws ResourceNotFo
46         Cargo updateCargo = cargoRpo.findById(id_carg)
47             .orElseThrow(() -> new ResourceNotFoundException("Usuario not exist with id_Usuario: " + id_carg));
48

```

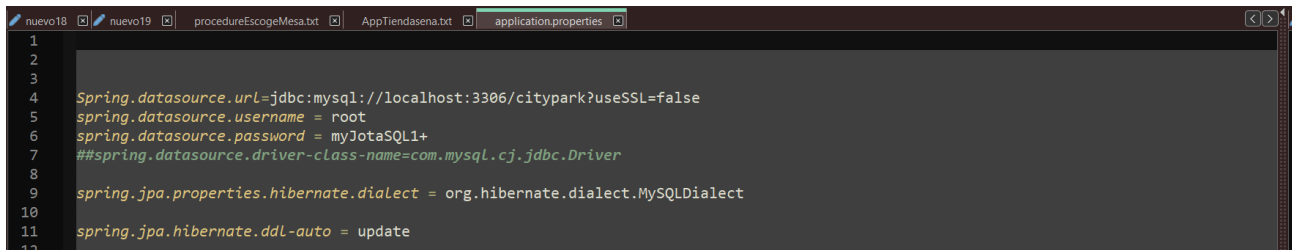
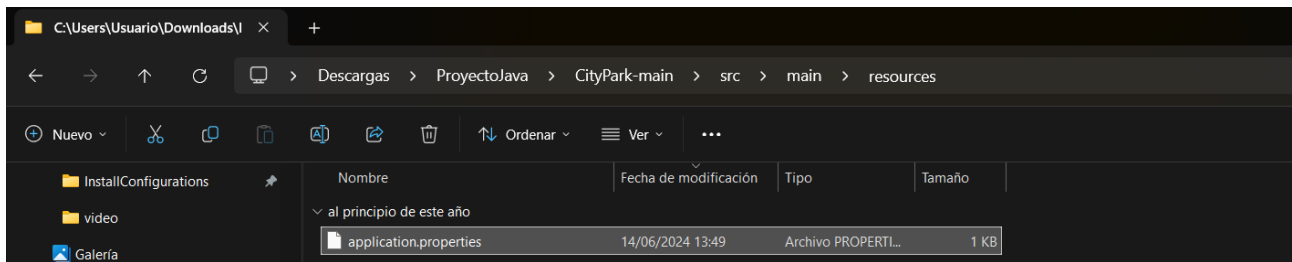
6. Observemos el modelo de datos que dio origen a todos los archivos java del presente proyecto con el fin de evidenciar como están creados, como objetos las tablas y como atributos los campos de cada una, de toda nuestra base de datos.

<div> <div>citypark</div> <div> <div>Tables</div> <div> <div>acudiente 32K</div> <div>Columns</div> <div> <div>id_acudiente (int(11))</div> <div>apellido_acu (varchar(255))</div> <div>cedula_acu (varchar(255))</div> <div>correo_elec_acu (varchar(255))</div> <div>direccion (varchar(255))</div> <div>nombre_acu (varchar(255))</div> <div>telefono (varchar(255))</div> <div>id_usu_acu (int(11))</div> </div> </div> </div> </div> <div> <div>atraccion 16K</div> <div>Columns</div> <div> <div>id_atraccion (int(11))</div> <div>clasificacion (varchar(255))</div> <div>descripcion (varchar(255))</div> <div>estado (bit(1))</div> <div>mantenimiento (bit(1))</div> <div>nombre_atrac (varchar(255))</div> </div> </div>	<div> <div>empleado 32K</div> <div>Columns</div> <div> <div>id_empleado (int(11))</div> <div>apellido (varchar(255))</div> <div>cedula (varchar(255))</div> <div>correo_electronico (varchar(255))</div> <div>direccion (varchar(255))</div> <div>nombre (varchar(255))</div> <div>telefono (varchar(255))</div> <div>id_emple_cargo (int(11))</div> </div> </div>
<div> <div>atraccion_condicionuso_map 32K</div> <div>Columns</div> <div> <div>id_cond_uso (int(11))</div> <div>id_atrac (int(11))</div> </div> </div>	<div> <div>estacion 16K</div> <div>Columns</div> <div> <div>id_estacion (int(11))</div> <div>nombre_estacion (varchar(255))</div> </div> </div>
<div> <div>cargo 16K</div> <div>Columns</div> <div> <div>id_cargo (int(11))</div> <div>nombre_cargo (varchar(255))</div> <div>salario (int(11))</div> </div> </div>	<div> <div>tickete 80K</div> <div>Columns</div> <div> <div>id_tickete (int(11))</div> <div>tipo_tickete (varchar(255))</div> <div>id_atrac_tiq (int(11))</div> <div>id_emp_tiq (int(11))</div> <div>id_est_tiq (int(11))</div> <div>id_usu_tiq (int(11))</div> </div> </div>
<div> <div>condicionuso 16K</div> <div>Columns</div> <div> <div>id_condicion_uso (int(11))</div> <div>descripcion (varchar(255))</div> </div> </div>	<div> <div>usuario 16K</div> <div>Columns</div> <div> <div>id_usuario (int(11))</div> <div>apellido (varchar(255))</div> <div>cedula (varchar(255))</div> <div>correo_electronico (varchar(255))</div> <div>direccion (varchar(255))</div> <div>edad (int(11))</div> <div>estatura (int(11))</div> <div>nombre (varchar(255))</div> <div>numero_visitas (int(11))</div> <div>telefono (varchar(255))</div> </div> </div>

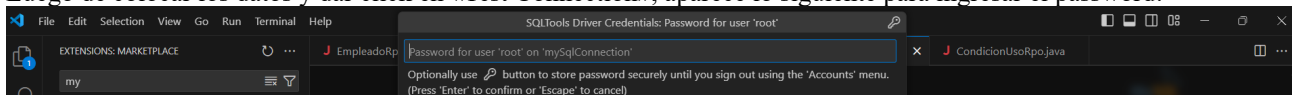
7. Para correr el proyecto, es necesario (si aún no se tiene), incorporar dentro del entorno visual studio un plugin para manejo de bases de datos desde un motor mysql.



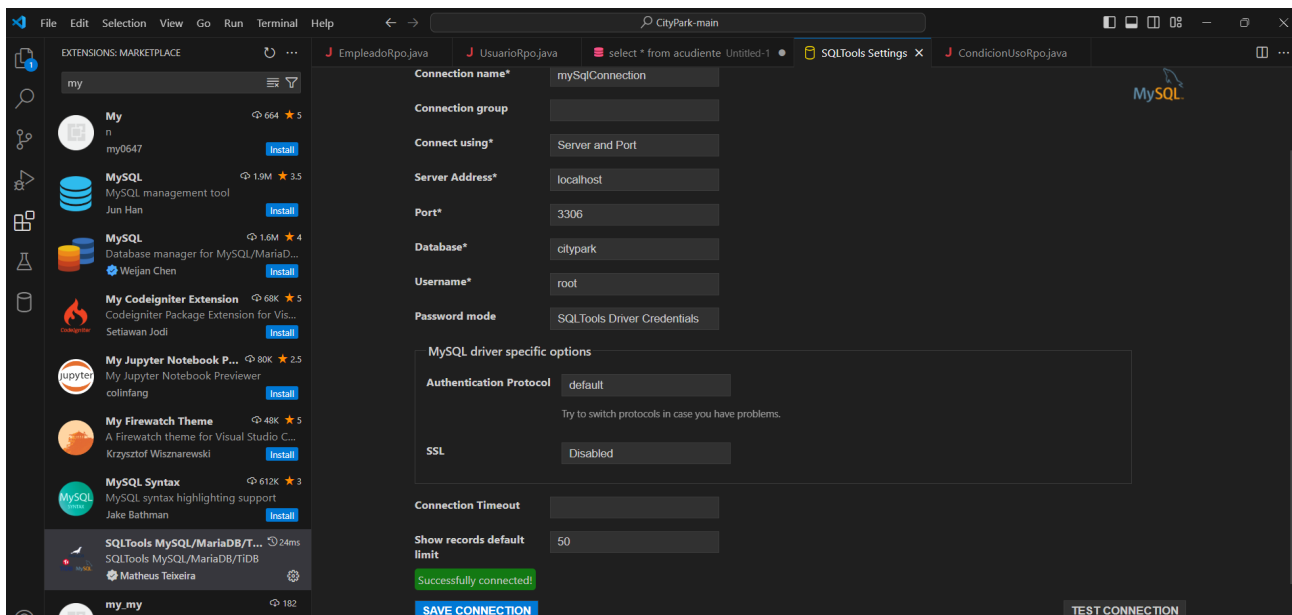
La información necesaria para esta parte la encontramos en el archivo application.properties que se encuentra dentro del proyecto. Se deben realizar los cambios requeridos para colocar la información de la conexión con la cual cuentan, es decir, los datos de conexión de sus respectivas bases de datos. Para mi caso particular son las siguientes:

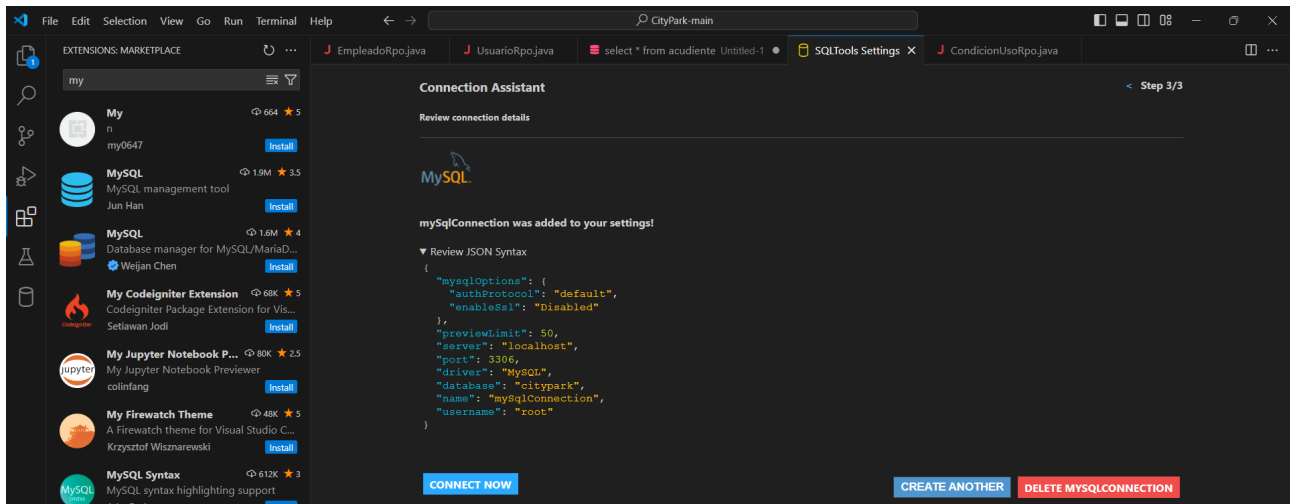


Luego de colocar los datos y dar click en «Test Connection», aparece lo siguiente para ingresar el password:



Finalmente indica que la conexión es satisfactoria. Y doy click a «Save Connection»



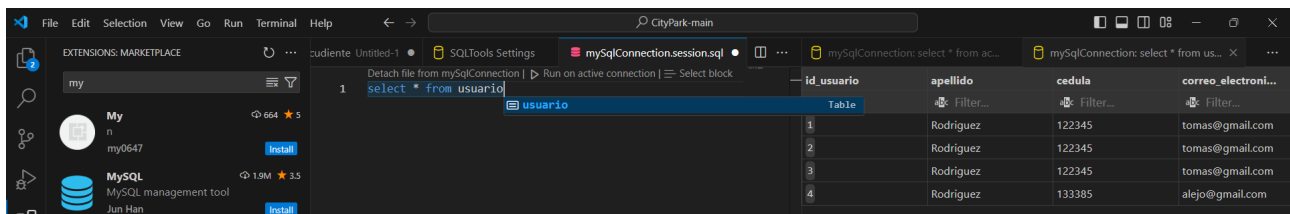


Ahora doy click a «Connect now».

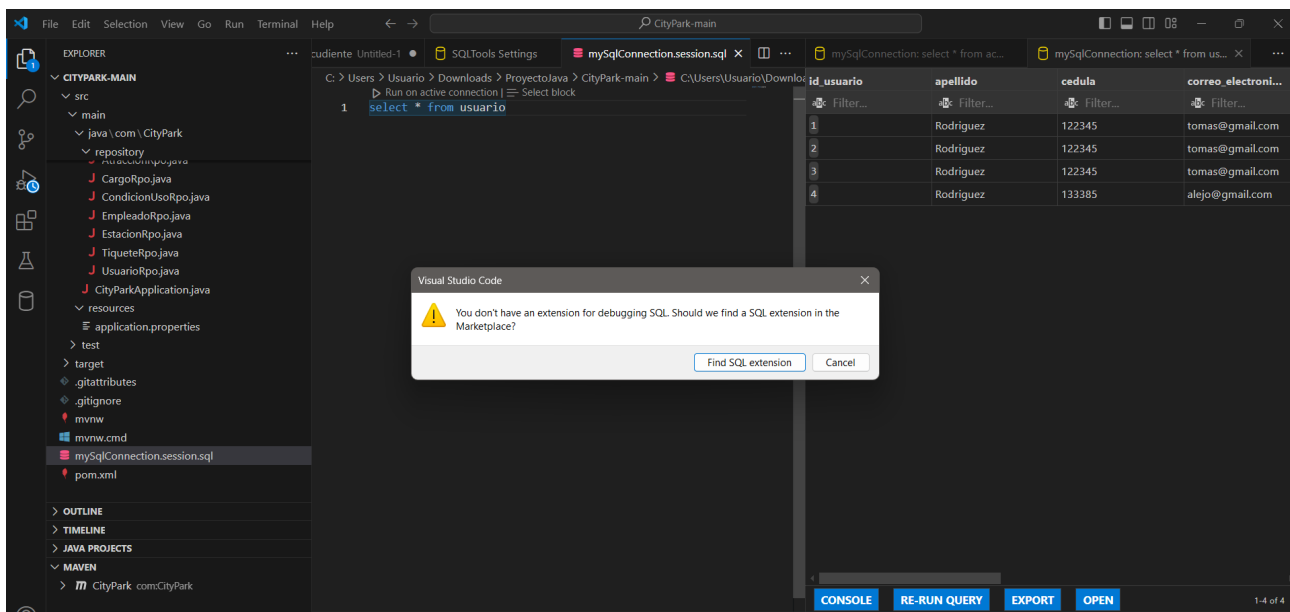
Los pasos detallados de crear la conexión se muestran en el siguiente enlace, cabe aclarar que estos pasos toman como ejemplo una conexión al motor SQL Server, lo único que se debe hacer es utilizar el motor de nuestro interés en vez de ese, es decir MySQL.

<https://learn.microsoft.com/es-es/azure/azure-sql/database/connect-query-vscode?view=azuresql>

Luego de dar click a connect now me genera una nueva pestaña donde puedo realizar consultas como si fuera un entorno de base de datos. Escribo una sentencia para confirmar que funciona y el resultado es el siguiente:



Al intentar ejecutar el proyecto me sugiere que instale un plugin que me permitirá realizar seguimientos a sentencias de sql igual que si nos encontraramos dentro del entorno MYSQL Workbench, al cual ya estamos familiarizados.



FileEditSelectionViewGoRunTerminalHelp

CityPark-main

SQLTools SettingsmysqlConnection.session.sqlExtension: sql funmysqlConnection: select * from ac...mysqlConnection: select * from us...

EXTENSIONS: MARKETPLACE@categorydebuggers SQL

sql fun1972msA.M

IBM Db2 for z/OS Devel...1Provides support for developing IBM ...IBMInstall

GetBotAI Code assistant13K5GetBotAI is your AI assistant designed ...FutureTechNexusInstall

AirOps33K5Powerful AI apps you can customize &...AirOpsInstall

ChatGPT GPT-4, AI Cod...41K4.5ChatGPT and GPT-4 AI Coding Assista...SixthInstall

Alva - AI Assistant, Chat ...25K5Autocorrect, secure, test, and improve ...Alva AIInstall

sql funv0.0.15A.M1,731☆☆☆☆

DisableUninstallAuto Update

DETAILSFEATURES

How to use?

It's very simple

1. Install

2. Put the cursor on any part of your code for 2 seconds

3. A small hover window will show up

4. Enjoy what you'll get in it

CATEGORIESProgrammingLanguagesSnippetsThemesLintersDebuggersKeymapsFormattersExtension PacksSCM ProvidersAzureLanguage PacksData ScienceMachine LearningVisualizationNotebooksEducationTesting

id	usuario	apellido	cedula	correo_electrónico
	Filter...	Filter...	Filter...	Filter...
1		Rodriguez	122345	tomas@gmail.com
2		Rodriguez	122345	tomas@gmail.com
3		Rodriguez	122345	tomas@gmail.com
4		Rodriguez	133385	alejo@gmail.com