

ÍNDICE

PRESENTACIÓN DEL TRABAJO	3
HERRAMIENTAS	4
DISEÑO DE HERRAMIENTA SOFTWARE	9
NORMALIZACIÓN	10
MODELO CONCEPTUAL	11
MODELO LÓGICO	13
BASE DE DATOS CON POGRESSQL	15
PLANIFICACIÓN PARA HERRAMIENTA SOFTWARE	22
PLANIFICACIÓN PARA HERRAMIENTA SOFTWARE	29
FORMAS DE LENGUAJES DE PROGRAMACIÓN PARA EXTENDER LA FUNCIONALIDAD DE TAREAS Y MANIPULACIÓN DE OBJETOS DE ANÁLISIS DE DATOS.	34
DESARROLLO DE HERRAMIENTA SOFTWARE	39
CONFIGURACIÓN DEL PROYECTO:	41
BACK-END	43
FRONT-END	55
COMPROBACIÓN DE HERRAMIENTA SOFTWARE	63
REVISIÓN DE RESULTADOS CON OTRO LENGUAJE	69
POSIBLES MEJORAS	70
CONCLUSIÓN	72

PRESENTACIÓN DEL TRABAJO

Parte A - Exploración de Herramientas para Manipular Datos:

En la entrada en la empresa, he investigado las funciones esenciales de un lenguaje de análisis de datos, destacando su capacidad para operaciones complejas y manipulación eficiente de datos. He propuesto el uso de PostgreSQL para la base de datos y Python con Pandas para análisis, priorizando herramientas escalables y eficientes.

Parte B - Diseño de la Herramienta de Software:

Seleccioné un conjunto de datos del INE y realicé su normalización, creando un modelo relacional en PostgreSQL. Utilicé UML para representar la manipulación de datos a través de objetos en memoria. Además, planifiqué y ejecuté 10 análisis en PostgreSQL, proporcionando un ejemplo de código Java para obtener y mostrar resultados, y analicé el papel de los lenguajes de programación en las tareas de análisis de datos.

Parte C - Desarrollo de la Herramienta:

En la tercera parte, he comenzado a desarrollar una aplicación web con Spring Boot para que los usuarios accedan a información de análisis desde la base de datos PostgreSQL. Esto incluyó la creación de endpoints y repositorios, estructurando la aplicación de manera efectiva.

Parte D - Comprobación de la Calidad de la Información:

En la última fase, he planificado una batería de pruebas para evaluar la efectividad de la aplicación desarrollada. Ejecuté las pruebas y revisé los resultados, comparándolos con otras herramientas como Excel contra el CSV original. Proporcioné una propuesta de mejoras basada en los resultados obtenidos.

HERRAMIENTAS

PRINCIPALES HERRAMIENTAS PARA MANIPULAR GRAN CONJUNTOS DE DATOS

1) Biblioteca **Pandas** = [**Python**]:

Manipulación de datos tabulares, limpieza, transformación y análisis exploratorio.<

2) Biblioteca **NumPy** = [**Python**]:

Operaciones matriciales y manipulación de arreglos multidimensionales.



3) **R**:

Estadísticas, análisis gráfico y manipulación de datos.



4) **Apache Hadoop**:

Almacenamiento y procesamiento distribuido de conjuntos de datos grandes.



5) **Apache Spark**:

Procesamiento de datos en clústeres, soporte para SQL, machine learning y procesamiento de gráficos.



6) **SQL** (Structured Query Language):

Consultas y manipulación de datos en bases de datos relacionales.



7) **Excel** (Power Query y Power Pivot):

Transformación y análisis de datos tabulares.



8) **Tableau**:

Visualización interactiva y análisis de datos.



9) **MATLAB**:

Análisis numérico, visualización y procesamiento de señales.



10) **Jupyter Notebooks**:

Entorno interactivo para la creación de documentos que integran código, texto y visualizaciones.



FUNCIONES PRINCIPALES DE UN LENGUAJE DE ANÁLISIS DE DATOS

Un lenguaje de análisis de datos efectivo debería ofrecer todas, o casi todas las funciones que veremos a continuación. Para satisfacer las necesidades de datos de diversas aplicaciones.

Manipulación de Datos:

«Capacidad para cargar, limpiar, transformar y combinar conjuntos de datos eficientemente.

Operaciones Estadísticas:

«Capacidad para calcular medidas estadísticas como media, mediana, desviación, etc.

Visualización de Datos:

«Crea gráficos y elementos visuales para interpretar los datos.

Procesamiento Distribuido:

«Soporte para el procesamiento de grandes conjuntos de datos distribuidos en grupos.

Machine Learning:

«Integración de algoritmos de *machine learning*, es decir, IA para desarrollar algoritmos, para la construcción de modelos previsores y clasificatorios.

Soporte para Datos Temporales:

«Manejo y análisis de datos temporales y series temporales.

Interactividad:

«Interactividad en el análisis de datos, como cuadernos interactivos.

Conectividad con Bases de Datos:

«Capacidad para conectarse y realizar consultas a bases de datos.

Documentación Integrada:

«Facilita el documentar y explicar el análisis dentro del entorno de trabajo.

Escalabilidad:

«Capacidad para manejar conjuntos de datos, ya sean pequeños, medianos o grandes.

TECNOLOGÍAS PARA MANIPULACIÓN DE GRAN CANTIDAD DE DATOS

Almacenamiento Distribuido:

Apache Hadoop Distributed File System (HDFS)

- HDFS permite el almacenamiento distribuido de grandes conjuntos de datos, facilitando su crecimiento y exceso.

Procesamiento Distribuido:

Apache Spark

- Spark ofrece un motor de procesamiento rápido y eficiente para grandes operaciones distribuidas, incluyendo soporte para *SQL*, *machine learning* y procesamiento gráfico.

Procesamiento en Memoria:

Apache Flink

- Flink se destaca por su gran capacidad de procesamiento en memoria, acelerando operaciones y consultas.

Bases de Datos NoSQL:

MongoDB (Document Store)

- Las bases de datos *NoSQL*, como MongoDB, son adecuadas para el manejo de grandes volúmenes de datos no estructurados, ofreciendo flexibilidad y rendimiento.

Almacenamiento en la Nube:

Amazon S3 (Simple Storage Service)

- S3 proporciona un almacenamiento en crecimiento y duradero en la nube, permitiendo el acceso rápido a grandes conjuntos de datos desde diversas ubicaciones.

Procesamiento SQL Interactivo:

PrestoDB

- PrestoDB permite realizar consultas *SQL* interactivas en grandes conjuntos de datos almacenados en diferentes fuentes, como sistemas de almacenamiento distribuido.

Cuadernos Interactivos:

Jupyter Notebooks con Apache Zeppelin

- Estas herramientas ofrecen entornos interactivos que combinan código - texto - visualizaciones, facilitando el análisis y colaboración.

Procesamiento de Flujo de Datos:

Apache Kafka

- Kafka se centra en el procesamiento de flujo de datos en tiempo real, permitiendo la absorción y manipulación de datos en movimiento.

Machine Learning Escalable:

Apache Mahout

- Mahout proporciona algoritmos de *machine learning* en crecimiento que operan con grandes conjuntos de datos.

Visualización de Datos:

Tableau

- Tableau ofrece capacidades de visualización para explorar y comunicar datos de manera efectiva.

FORMAS EN QUE LOS LENGUAJES DE PROGRAMACIÓN INTERACTÚAN CON UN LENGUAJE DE ANÁLISIS DE DATOS

Formas más comunes de interacción:

1. **Con librerías específicas:** Muchos lenguajes de programación, como *Python* con *Pandas* o *NumPy*, *R* con *dplyr*, *tidyr* o más, y *Julia* con *DataFrames.jl*, tienen librerías especializadas para análisis de datos. Estas librerías

proporcionan funciones y herramientas específicas para manipular, visualizar y analizar datos.

2. **APIs y bibliotecas:** Los lenguajes de programación pueden interactuar con lenguajes de análisis de datos a través de **APIs** (Interfaz de Programación de Aplicaciones) y **bibliotecas** específicas para lograr esto.
3. **Integración de herramientas:** Algunas herramientas de análisis de datos ofrecen interfaces determinadas para la integración con otros lenguajes de programación. Por ejemplo, herramientas como *Apache Spark*, *Hadoop* y *TensorFlow* se pueden integrar con lenguajes como *JAVA*, *Scala* y *Python* para análisis de datos a gran escala, aprendizaje automático y procesamiento distribuido.
4. **Comunicación entre procesos:** La comunicación entre procesos, como *sockets*, *RPC* (*Remote Procedure Call*), y otros mecanismos de comunicación interprocesos, permiten que diferentes componentes de un sistema interactúen y compartan datos entre sí.
5. **Formatos de intercambio de datos:** El intercambio de datos en formatos comunes como *CSV*, *JSON*, *Parquet*, *Avro* y mas, permiten la compatibilidad entre herramientas y sistemas de análisis de datos.

COMBINAR DIFERENTES LENGUAJES DE PROGRAMACIÓN PARA EXTENSIONES DE FUNCIONALIDAD

En muchos casos, los desarrolladores necesitan combinar funcionalidades de diferentes lenguajes de programación para completar tareas complejas o para aprovechar las fortalezas de cada lenguaje en un proyecto. Esto puede ser especialmente relevante en el análisis de datos, donde se pueden utilizar diferentes lenguajes para diferentes partes del proceso.

Formas para lograr ese cometido:

Interfaz de Llamada Externa (Foreign Function Interface - FFI):

La **FFI** permite que un programa escrito en un lenguaje de programación se comunique con funciones en otro lenguaje. Creando una interfaz común que facilita la llamada a funciones externas.

- ◆ Uso en Análisis de Datos: Permite llamar funciones en lenguajes de bajo nivel, como *C* o *C++*, para realizar operaciones intensivas.

Uso de Bibliotecas Compartidas (Shared Libraries):

Las bibliotecas compartidas son archivos compilados que contienen funciones utilizadas por programas escritos en otros lenguajes. Se enlazan dinámicamente durante el tiempo de ejecución.

- ◆ Uso en Análisis de Datos: Facilita la integración de funciones, como algoritmos de procesamiento de datos, implementados en lenguajes como *Fortran*.

Wrappers y APIs:

Se pueden crear envoltorios, llamadas *wrappers*, o interfaces de programación de aplicaciones, llamadas *APIs*, que actúan como capa intermedia entre el código escrito en diferentes lenguajes. Haciendo que la interfaz sea más fácil de usar.

- ◆ Uso en Análisis de Datos: Permite la integración de módulos específicos desarrollados en lenguajes para análisis estadístico o *machine learning*, siendo una interfaz más accesible.

Uso de Extensiones y Módulos:

Lenguajes como *Python*, permiten la creación de extensiones y módulos escritos en otros lenguajes, como *C* o *C++*. Estos pueden ser llamados y utilizados desde el código principal.

- ◆ Uso en Análisis de Datos: Facilita la incorporación de funcionalidades, como algoritmos de optimización, estando implementados en lenguajes de bajo nivel.

Utilización de Servicios Web y Protocolos:

Los servicios web que utilizan protocolos estándar, como *REST* o *GraphQL*, permiten la comunicación entre aplicaciones creadas por diferentes lenguajes.

Los resultados pueden ser utilizados remotamente.

- ◆ Uso en Análisis de Datos: Posibilita la integración de servicios de análisis externos que pueden ser implementados en cualquier lenguaje compatible con servicios web.

Llamadas a Funciones DLL (Dynamic Link Libraries) en Windows:

En sistemas Windows, las funciones almacenadas en bibliotecas dinámicas (*DLL*) pueden ser llamadas desde otros programas, dando un enlace dinámico.

- ◆ Uso en Análisis de Datos: Permite la utilización de funciones específicas de lenguajes como *C#* o *C++* para operaciones intensivas.

La capacidad de enlazar y llamar código desde diversos lenguajes, permite la expansión y optimización de tareas en el análisis de datos. La elección de la estrategia dependerá de los requisitos del proyecto y la compatibilidad de los lenguajes añadidos.

DISEÑO DE HERRAMIENTA SOFTWARE

En esta parte iniciaremos el diseño de una herramienta de software para analizar grandes conjuntos de datos. Este diseño preliminar servirá de base para el desarrollo detallado de la herramienta en el LO3. En este proceso, se establecen los fundamentos técnicos y conceptuales necesarios para crear una herramienta capaz

de manejar y analizar eficientemente datos complejos, proporcionando así información relevante para el escenario específico.

Hemos seleccionado un conjunto de datos dados en la página del INE, Instituto Nacional de Estadística, basado en **Renta por hogar por comunidades autónomas**. Estos datos definen la Renta media por hogar, desde 2008 hasta 2022, de cada comunidad autónoma. Incluye las comunidades y ciudades autónomas, la Renta anual neta media por hogar, su Periodo y su total. Estos datos están disponibles en formato CSV.

Renta por hogar por comunidades autónomas

Unidades: €

▶ Seleccione valores a consultar

Comunidades y Ciudades Autónomas	Renta anual neta media por hogar	Periodo
<div><input type="text" value=""/></div> <div>Total Nacional 01 Andalucía 02 Aragón 03 Asturias, Principado de 04 Balears, Illes 05 Canarias 06 Cantabria</div> <div>Seleccionados: 20 Total: 20</div>	<div><input type="text" value=""/></div> <div>Renta neta media por hogar Renta media por hogar (con alquiler imputado)</div> <div>Seleccionados: 1 Total: 2</div>	<div><input type="text" value=""/></div> <div>2014 2013 2012 2011 2010 2009 2008</div> <div>Seleccionados: 1 Total: 15</div>

▶ Elija forma de presentación de la tabla

	Renta anual neta media por hogar	Periodo
Comunidades y Ciudades Autónomas	-	-
	-	-
	-	-
	-	-

Decimales a mostrar:

¿Sabías que...?

La realización de la ECV, Encuesta de condiciones de vida, permite poner a disposición de la Comisión Europea un instrumento estadístico de primer orden para el estudio de la pobreza y desigualdad, el seguimiento de la cohesión social en el territorio de su ámbito, el estudio de las necesidades de la población y del impacto de las políticas sociales y económicas sobre los hogares y las personas, así como para el diseño de nuevas políticas.

NORMALIZACIÓN

Aquí aplicaremos la Normalización, proceso de bases de datos para organizar la estructura de los datos, reduciendo la redundancia y evitando problemas, dividiendo las tablas en partes más pequeñas y relacionadas entre sí.

Una vez realizada la normalización del conjunto de datos de la Renta por hogar, para conseguir nuestro diseño de una base de datos relacional. Presentaremos nuestro modelo conceptual de la base de datos:

MODELO CONCEPTUAL

La división en tablas:

1. Tabla renta_hogares:

Esta tabla contendrá todos los datos, de Comunidades Autonomas, Tipos de Renta, Renta Anual Media por hogar, Periodos. Esto es para enlazar los datos, con las demás tablas. Atributos de las columnas del CSV:

ATRIBUTOS: *ComunidadesA, Renta, Periodo, Total.*

2. Tabla de Comunidades Autónomas:

ATRIBUTOS: *ID_Comunidad, Nombre_Comunidad.*

3. Tabla de Tipos de Renta:

ATRIBUTOS: *ID_Tipo_Renta, Tipo_Renta.*

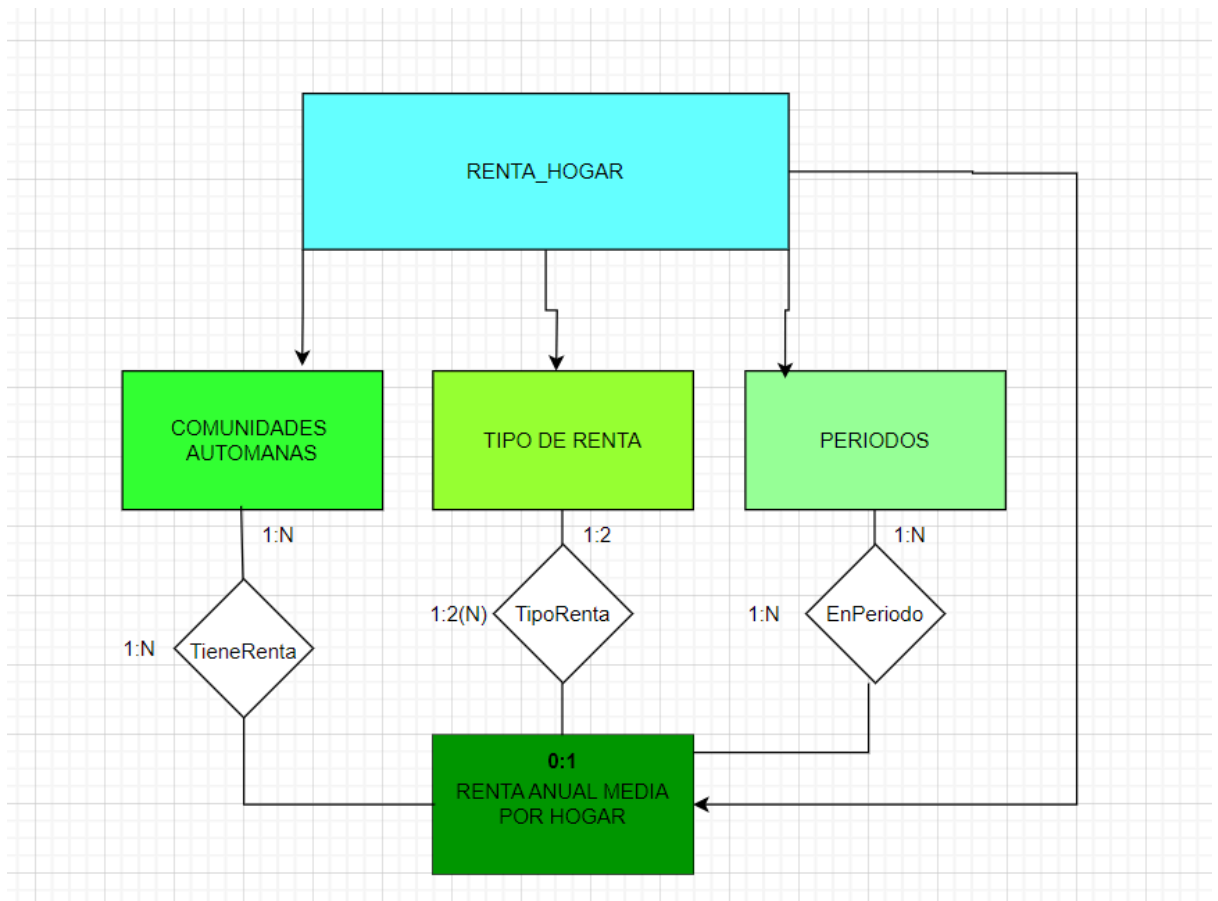
4. Tabla de Renta Anual Media por Hogar:

ATRIBUTOS: *ID_Comunidad , ID_Periodo , Período, Renta_Anual_Media, ID_Tipo_Renta*

5. Tabla de Periodos:

ATRIBUTOS: *ID_Periodo, Año.*

DIAGRAMA DE FLUJO DE DATOS:



EXPLICACIÓN DE RELACIONES:

Tabla de Comunidades Autónomas - Tabla de RentaAnualMedia por Hogar:

- A) Una *COMUNIDAD AUTÓNOMA* puede tener múltiples entradas de Renta Anual Media por Hogar (como para diferentes periodos).
- B) *RENTA ANUAL MEDIA POR HOGAR* pertenece a una sola comunidad autónoma.
- C) Por lo tanto, la relación es de **1:N** (de uno a muchos), donde UNA comunidad autónoma puede tener MÚLTIPLES entradas de renta anual media por hogar.

Tabla de Tipos de Renta - Tabla de Renta Anual Media por Hogar:

- A) *TIPO DE RENTA* puede estar asociado con múltiples entradas de renta anual media por hogar.
- B) *RENTA ANUAL MEDIA POR HOGAR* está asociada con un solo tipo de renta.
- C) Por lo tanto, la relación es de **1:N**, donde UN tipo de renta puede estar asociado con VARIAS entradas de renta anual media por hogar.

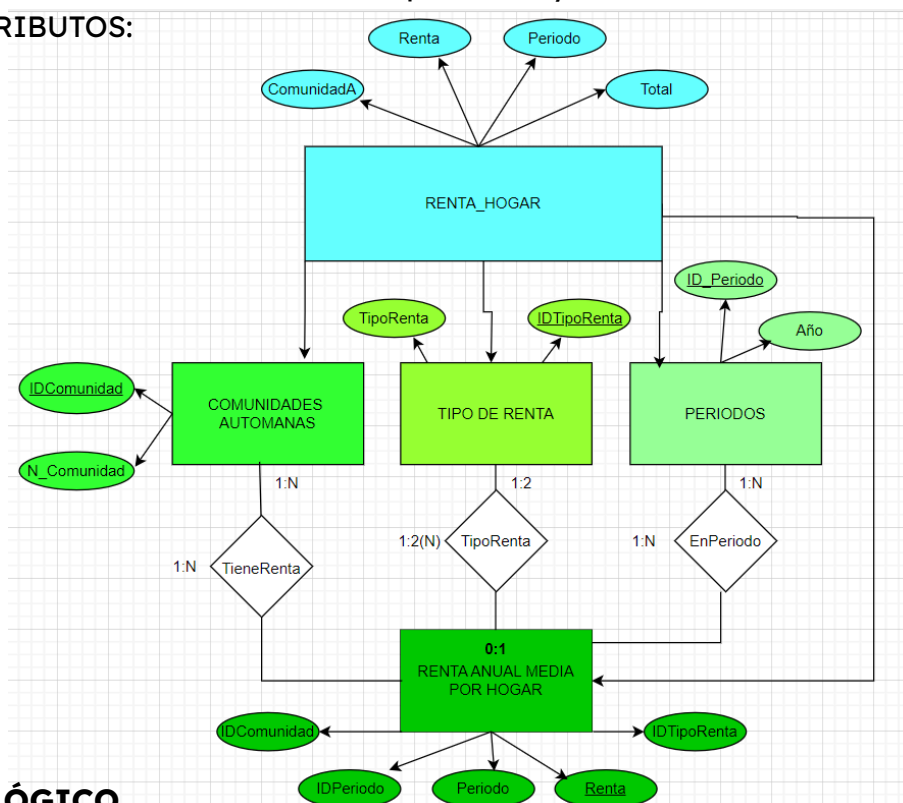
Tabla de Renta Anual Media por Hogar - Tabla de Periodos:

- A) Un solo *PERIODO* puede tener muchas entradas de renta anual media por hogar.
- B) Una entrada de RENTA ANUAL MEDIA POR HOGAR pertenece a un solo periodo.
- C) Por lo tanto, la relación es de **N:1**, donde MUCHAS entradas de renta anual media por hogar pueden estar asociadas con UN SOLO periodo.

Tabla de renta_hogares - Resto de las Tablas:

La tabla de "*renta_hogares*" contiene todos los archivos CSV. No se muestran relaciones directas con las otras tablas porque es una entidad separada que almacena los archivos CSV. Sin embargo, los datos contenidos en los archivos CSV pueden relacionarse con las otras tablas a través de la importación y relación de datos mediante claves primarias y foráneas.

CON LOS ATRIBUTOS:



MODELO LÓGICO

1) Tabla renta_hogares:

- a) Esta tabla actúa como la tabla principal que contiene Comunidades Autonomas, Tipos de Renta, Renta Anual Media por hogar, Periodos.
- Columnas: *ComunidadesA*, *Renta*, *Periodos*, *Total*.
 - SIN CLAVES FORÁNEAS

2) Tabla de Comunidades Autónomas:

- a) Esta tabla actúa como la tabla principal que contiene los nombres de las comunidades autónomas y ciudades autónomas.
- Columnas: *ID_Comunidad* (PK), *Nombre_Comunidad*.
 - SIN CLAVES FORÁNEAS

3) Tabla de Tipos de Renta:

- a) Esta tabla contendrá los tipos de renta anual media por hogar, como "Renta neta media por hogar" y "Renta media por hogar (con alquiler imputado)".
- Columnas: *ID_Tipo_Renta* (PK), *Tipo_Renta*.
 - SIN CLAVES FORÁNEAS

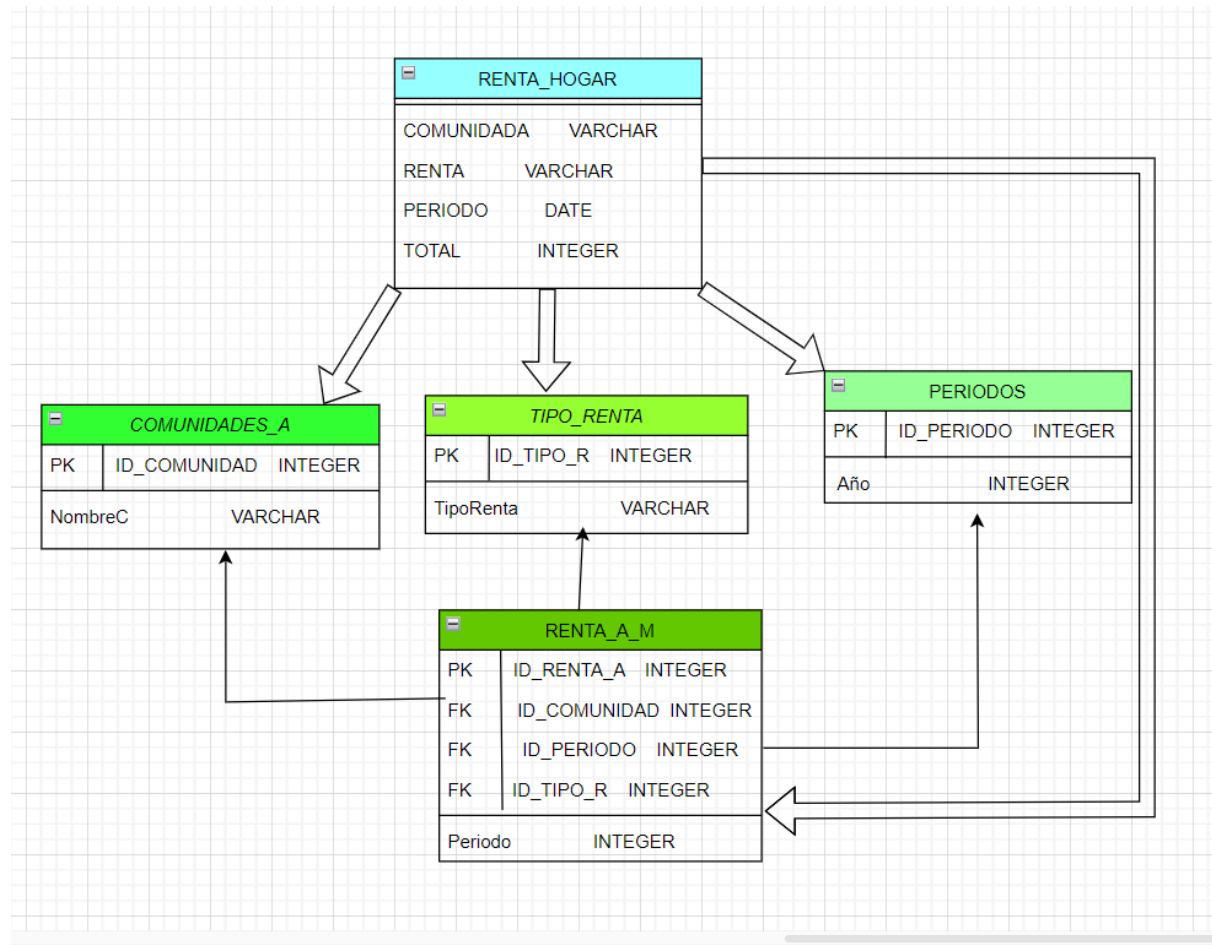
4) Tabla de Renta Anual Media por Hogar:

- a) Esta tabla tendrá los datos de la renta anual media por hogar para cada comunidad autónoma en diferentes periodos e incluye una referencia al tipo de renta.
- Columnas: *ID_Comunidad* (FK), *ID_Periodo* (FK), *Período*, *Renta_Anual_Media* (PK), *ID_Tipo_Renta* (FK).
 - Claves foráneas:
 - *ID_Comunidad*: Se relaciona con la columna *ID_Comunidad* de la tabla "*Comunidades Autónomas*".
 - *ID_Periodo*: Se relaciona con la columna *ID_Periodo* de la tabla "*Periodos*".
 - *ID_Tipo_Renta*: Se relaciona con la columna *ID_Tipo_Renta* de la tabla "*Tipos de Renta*".

5) Tabla de Periodos:

- a) Esta tabla contendrá los distintos periodos en los que se ha recopilado la información de renta anual media por hogar.
- Columnas: *ID_Periodo* (PK), *Año*.
 - SIN CLAVES FORÁNEAS

DIAGRAMA UML:



BASE DE DATOS CON POGRESSQL

Ahora definiremos la estructura de la tabla, que se llamará: **Renta**. Contiene las columnas para **Comunidades_Autónomas**, **Tipos_Renta**, **Renta_Anual_Media_Hogar** y **Periodos**. En esta primera parte del proceso de normalización, definiremos las entidades y los atributos.

Entramos en **PogresSQL** y empezamos:

1. Preparar el archivo de datos:

- Debemos tener los archivos de datos en un formato compatible con PostgreSQL, es decir, en CSV, TSV, u otro que pueda manejar. en nuestro caso es CSV.
2. Acceder a la línea de comandos de PostgreSQL:
 - Usamos la línea de comandos de PostgreSQL u otra herramienta.
 3. Crear Base de Datos:

En PostgreSQL seleccionamos la opción de crear la BBDD y añadimos su nombre **RENTA**:



4. Creamos Tabla Principal **renta_hogares** y almacenamos datos del CSV
Debemos exportar los datos a esta tabla y así los tendremos todos almacenados en un sitio seguro.

1º Creamos la Tabla Con el comando **CREATE TABLE** y el nombre de las columnas del CSV:

```
CREATE TABLE renta_hogares (
  comunidad_autonoma VARCHAR(100),
  tipo_renta VARCHAR(100),
  periodo INT,
  renta_neta_media_hogar INT
);
```

2º Cojemos el archivo de CSV y copiamos su ruta. Y añadimos este comando:

```
COPY renta_hogares(comunidad_autonoma, tipo_renta, periodo,
renta_neta_media_hogar)
FROM 'C:\Users\Tecnicos\Desktop\datosRenta.csv' DELIMITER ';' CSV HEADER;
```

- DELIMITER ';' → Especifica el delimitador que se utiliza en el archivo CSV, es decir ;.
- CSV HEADER → Indica que la primera línea del archivo CSV contiene los nombres de las columnas.

Necesitas los permisos adecuados para ejecutar el comando COPY en PostgreSQL.

3º Comprobamos que se halla añadido:

Ponemos el comando **SELECT * FROM renta_hogares**

Y se nos debería de mostrar la tabla:

	comunidad_autonoma character varying (100)	tipo_renta character varying (100)	periodo integer	renta_neta_media_hogar integer
1	Total Nacional	Renta neta media por hogar	2022	32216
2	Total Nacional	Renta neta media por hogar	2021	30552
3	Total Nacional	Renta neta media por hogar	2020	30690
4	Total Nacional	Renta neta media por hogar	2019	29132
5	Total Nacional	Renta neta media por hogar	2018	28417
6	Total Nacional	Renta neta media por hogar	2017	27558
7	Total Nacional	Renta neta media por hogar	2016	26730
8	Total Nacional	Renta neta media por hogar	2015	26092
9	Total Nacional	Renta neta media por hogar	2014	26154
10	Total Nacional	Renta neta media por hogar	2013	26775
11	Total Nacional	Renta neta media por hogar	2012	27747
12	Total Nacional	Renta neta media por hogar	2011	28206
13	Total Nacional	Renta neta media por hogar	2010	29634
14	Total Nacional	Renta neta media por hogar	2009	30045
15	Total Nacional	Renta neta media por hogar	2008	28787
16	Total Nacional	Renta media por hogar (con alquiler imputado)	2022	37363
17	Total Nacional	Renta media por hogar (con alquiler imputado)	2021	35497
18	Total Nacional	Renta media por hogar (con alquiler imputado)	2020	35485
19	Total Nacional	Renta media por hogar (con alquiler imputado)	2019	33794
20	Total Nacional	Renta media por hogar (con alquiler imputado)	2018	32929
21	Total Nacional	Renta media por hogar (con alquiler imputado)	2017	31956
22	Total Nacional	Renta media por hogar (con alquiler imputado)	2016	30822
23	Total Nacional	Renta media por hogar (con alquiler imputado)	2015	30031
24	Total Nacional	Renta media por hogar (con alquiler imputado)	2014	30257
25	Total Nacional	Renta media por hogar (con alquiler imputado)	2013	30501
26	Total Nacional	Renta media por hogar (con alquiler imputado)	2012	31686
27	Total Nacional	Renta media por hogar (con alquiler imputado)	2011	32420

5. Crear el resto de Tablas de la Base de Datos:

Usamos el comando **CREATE TABLE** con el nombre de las tablas y con sus columnas:

```
CREATE TABLE ComunidadesA (
  ID_Comunidad SERIAL PRIMARY KEY,
  Nombre_Comunidad VARCHAR(100) NOT NULL
);
```

Tipo de datos **SERIAL** para que PostgreSQL genere automáticamente valores únicos para esta columna.

```
CREATE TABLE TiposDeRenta (
  ID_Tipo_Renta SERIAL PRIMARY KEY,
```

```
Tipo_Renta VARCHAR(100) NOT NULL
);
```

```
CREATE TABLE Periodos (
  ID_Periodo SERIAL PRIMARY KEY,
  Año INT NOT NULL
);
```

```
CREATE TABLE RentaAnualM (
  ID_Comunidad INT,
  ID_Periodo INT,
  Periodo INT,
  Renta_Anual_Media INT,
  ID_Tipo_Renta INT,
  PRIMARY KEY (ID_Comunidad, ID_Periodo, ID_Tipo_Renta),
  FOREIGN KEY (ID_Comunidad) REFERENCES ComunidadesA(ID_Comunidad),
  FOREIGN KEY (ID_Periodo) REFERENCES Periodos(ID_Periodo),
  FOREIGN KEY (ID_Tipo_Renta) REFERENCES TiposDeRenta(ID_Tipo_Renta)
);
```

Gracias al Diagrama UML, ya sabemos como se estructuran las claves Primarias y Foraneas.

6. Añadir Datos:

Toca añadir los datos a sus respectivas tablas con el comando **INSERT INTO**

Para la tabla *ComunidadesA* añadimos este comando:

```
INSERT INTO ComunidadesA (Nombre_Comunidad)
SELECT DISTINCT comunidad_autonoma
FROM renta_hogares;
```

INSERT INTO ComunidadesA (Nombre_Comunidad):

- **INSERT INTO ComunidadesA:** La consulta indica que los datos se insertarán en la tabla ComunidadesA.
- **(Nombre_Comunidad):** Especifica la columna de la tabla en la que se insertarán los datos. Insertandose en la columna *Nombre_Comunidad*.

SELECT DISTINCT comunidad_autonoma FROM renta_hogares;

- **SELECT DISTINCT comunidad_autonoma:** Selecciona los valores únicos de la columna *comunidad_autonoma* de la tabla *renta_hogares*.
- **DISTINCT ==** Garantiza No halla duplicados.

FROM renta_hogares: Especifica que los datos son seleccionados de la tabla *renta_hogares*.

Verificamos con el comando **SELECT * FROM ComunidadesA:**

id_comunidad [PK] integer	nombre_comunidad character varying (100)
1	02 Aragón
2	03 Asturias, Principado de
3	11 Extremadura
4	14 Murcia, Región de
5	04 Balears, Illes
6	18 Ceuta
7	06 Cantabria
8	12 Galicia
9	10 Comunitat Valenciana
10	08 Castilla - La Mancha
11	Total Nacional
12	13 Madrid, Comunidad de
13	19 Melilla
14	05 Canarias
15	09 Cataluña
16	01 Andalucía
17	16 País Vasco
18	15 Navarra, Comunidad Foral de
19	17 Rioja, La
20	07 Castilla y León

Para la tabla *TiposDeRenta* añadimos este comando:

```
INSERT INTO TiposDeRenta (Tipo_Renta)
SELECT DISTINCT tipo_renta
FROM renta_hogares;
```

Verificamos con el comando **SELECT * FROM TiposDeRenta:**

id_tipo_renta [PK] integer	tipo_renta character varying (100)
1	Renta media por hogar (con alquiler imputado)
2	Renta neta media por hogar

Para la tabla *Periodos* añadimos este comando:

```
INSERT INTO Periodos (Año)
SELECT DISTINCT periodo
FROM renta_hogares;
```

Verificamos con el comando **SELECT * FROM Periodos:**

id_periodo [PK] integer	año integer
1	2013
2	2021
3	2008
4	2015
5	2010

No hay problema porque no esté ordenado, ya lo organizaremos cuando hagamos consultas.

Para la tabla *RentaAnualM* añadimos este comando:

```
INSERT INTO RentaAnualM (ID_Comunidad, ID_Periodo, Periodo,
Renta_Anual_Media, ID_Tipo_Renta)
SELECT
    ca.ID_Comunidad,
    p.ID_Periodo,
    rh.periodo,
    rh.renta_neta_media_hogar,
    tr.ID_Tipo_Renta
FROM
    renta_hogares rh
    INNER JOIN ComunidadesA ca ON rh.comunidad_autonoma =
ca.Nombre_Comunidad
    INNER JOIN Periodos p ON rh.periodo = p.Año
    INNER JOIN TiposDeRenta tr ON rh.tipo_renta = tr.Tipo_Renta;
```

INSERT INTO RentaAnualM → Agrega datos a la tabla *RentaAnualM*.
Especificándose las columnas en las que se insertarán los datos: *ID_Comunidad*,
ID_Periodo, *Periodo*, *Renta_Anual_Media* e *ID_Tipo_Renta*.

SELECT → extrae datos de las otras tablas para insertarlos en esta. Cada columna seleccionada es una columna en RentaAnualM. Las columnas seleccionadas:

- ca.ID_Comunidad: ID_Comunidad de la tabla ComunidadesA.
- p.ID_Periodo: ID_Periodo de la tabla Periodos.

- rh.periodo: Periodo de la tabla renta_hogares.
- rh.renta_neta_media_hogar: Renta neta media por hogar de la tabla renta_hogares.
- tr.ID_Tipo_Renta: ID_Tipo_Renta de la tabla TiposDeRenta.

El FROM renta_hogares rh y los INNER JOIN → combinan filas de *renta_hogares*, *ComunidadesA*, *Periodos* y *TiposDeRenta*, basanose en valores de columnas como *comunidad_autonoma*, *periodo* y *tipo_renta*.

Verificamos con el comando **SELECT * FROM RentaAnualIM :**

id_comunidad [PK] integer	id_periodo [PK] integer	periodo integer	renta_anual_media integer	id_tipo_renta [PK] integer
6	3	2008	34131	1
1	3	2008	29506	2
2	3	2008	33569	1
2	3	2008	29473	2
3	3	2008	24852	1
3	3	2008	21756	2
4	3	2008	30714	1
4	3	2008	26995	2
5	3	2008	32021	1
5	3	2008	28981	2
1	3	2008	33237	1
6	3	2008	29776	2
7	3	2008	33514	1
7	3	2008	29576	2
8	3	2008	29859	1
8	3	2008	25859	2
9	3	2008	30007	1
9	3	2008	25802	2
10	3	2008	28244	1
10	3	2008	24974	2
11	3	2008	32807	1
11	3	2008	28787	2
12	3	2008	38484	1
12	3	2008	33889	2

PLANIFICACIÓN PARA HERRAMIENTA SOFTWARE

Ahora analizaremos detalladamente los datos que hemos ido almacenados en la base de datos, utilizando diferentes consultas SQL. Este análisis nos proporcionará cierta información adicional como segmentación por tipo de renta, comparaciones geográficas, análisis demográfico, correlaciones socioeconómicas y variabilidad de la renta.

Actualmente nuestra Base de Datos tiene cierta información sobre las comunidades autónomas, el tipo de renta que hay, sus periodos y la renta.

En este plan de pruebas se busca identificar y ejecutar consultas SQL que nos permitan obtener resultados significativos y útiles.

A continuación, presentamos diez resultados de análisis con distintas consultas SQL que hemos realizado en la base de datos:

1. Promedio de la renta neta media por hogar por período:

```
SELECT AVG(renta_neta_media_hogar) AS promedio_renta
FROM renta_hogares;
```

promedio_renta	
numeric	
30990.725000000000	

2. Máxima renta neta media por hogar por comunidad autónoma:

```
SELECT comunidad_autonoma, MAX(renta_neta_media_hogar) AS
max_renta
FROM renta_hogares
GROUP BY comunidad_autonoma;
```

comunidad_autonoma character varying (100)	max_renta integer
02 Aragón	38109
03 Asturias, Principado de	35225
11 Extremadura	29030
14 Murcia, Región de	33747
04 Balears, Illes	38258
18 Ceuta	41951
06 Cantabria	37617
12 Galicia	34589
10 Comunitat Valenciana	34178
08 Castilla - La Mancha	32617
Total Nacional	37363
13 Madrid, Comunidad de	45525
19 Melilla	46566
05 Canarias	32131
09 Cataluña	41206
01 Andalucía	32687
16 País Vasco	44538
15 Navarra, Comunidad Foral de	44834
17 Rioja, La	36853
07 Castilla y León	35029

3. Cantidad de hogares por comunidad autónoma:

```
SELECT comunidad_autonoma, COUNT(*) AS cantidad_hogares
FROM renta_hogares
GROUP BY comunidad_autonoma;
```

comunidad_autonoma character varying (100)	cantidad_hogares bigint
02 Aragón	30
03 Asturias, Principado de	30
11 Extremadura	30
14 Murcia, Región de	30
04 Balears, Illes	30
18 Ceuta	30
06 Cantabria	30
12 Galicia	30
10 Comunitat Valenciana	30
08 Castilla - La Mancha	30
Total Nacional	30
13 Madrid, Comunidad de	30
19 Melilla	30
05 Canarias	30
09 Cataluña	30
01 Andalucía	30
16 País Vasco	30
15 Navarra, Comunidad Foral de	30
17 Rioja, La	30
07 Castilla y León	30

4. Número de periodos distintos en los que se han recopilado datos:

```
SELECT COUNT(DISTINCT periodo) AS cantidad_periodos
FROM renta_hogares;
```

cantidad_periodos bigint
15

5. Rentas medias más altas y más bajas por tipo de renta:

```
SELECT tipo_renta, MAX(renta_neta_media_hogar) AS max_renta,
MIN(renta_neta_media_hogar) AS min_renta
FROM renta_hogares
GROUP BY tipo_renta;
```

tipo_renta character varying (100)	max_renta integer	min_renta integer
Renta media por hogar (con alquiler imputado)	46566	23056
Renta neta media por hogar	41714	19364

6. Total de renta neta media por hogar por año:

```
SELECT EXTRACT(YEAR FROM TO_TIMESTAMP(periodo::text, 'YYYY')) AS
año, SUM(renta_neta_media_hogar) AS total_renta
FROM renta_hogares
GROUP BY año;
```

año numeric	total_renta bigint
2021	1314430
2022	1394380
2018	1242851
2011	1231947
2012	1205817
2016	1176509
2010	1277305
2015	1137737
2009	1285884
2019	1266071
2014	1147011
2013	1161335
2008	1232645
2017	1201749
2020	1318764

7. Porcentaje de variación de la renta neta media por hogar entre años consecutivos:

```
SELECT EXTRACT(YEAR FROM TO_TIMESTAMP(periodo::text, 'YYYY')) AS
año,
((SUM(renta_neta_media_hogar) -
LAG(SUM(renta_neta_media_hogar), 1) OVER (ORDER BY año)) /
LAG(SUM(renta_neta_media_hogar), 1) OVER (ORDER BY año)) * 100 AS
porcentaje_variacion
FROM renta_hogares
GROUP BY año;
```

año numeric	porcentaje_variacion bigint
2008	[null]
2009	0
2010	0
2011	0
2012	0
2013	0
2014	0
2015	0
2016	0
2017	0
2018	0
2019	0
2020	0
2021	0
2022	0

8. Rentas medias por hogar por tipo de renta y por comunidad autónoma:

```
SELECT comunidad_autonoma, tipo_renta,
ROUND(AVG(renta_neta_media_hogar)::numeric, 2) AS renta_media
FROM renta_hogares
GROUP BY comunidad_autonoma, tipo_renta;
```


comunidad_autonoma character varying (100)	tipo_renta character varying (100)	renta_media numeric
05 Canarias	Renta neta media por hogar	23796.27
17 Rioja, La	Renta neta media por hogar	28369.20
13 Madrid, Comunidad de	Renta media por hogar (con alquiler imputado)	39218.07
Total Nacional	Renta media por hogar (con alquiler imputado)	32898.33
01 Andalucía	Renta neta media por hogar	24119.53
Total Nacional	Renta neta media por hogar	28582.33
15 Navarra, Comunidad Foral de	Renta neta media por hogar	35602.53
10 Comunitat Valenciana	Renta media por hogar (con alquiler imputado)	29681.33
10 Comunitat Valenciana	Renta neta media por hogar	25436.27
03 Asturias, Principado de	Renta media por hogar (con alquiler imputado)	32228.27
14 Murcia, Región de	Renta media por hogar (con alquiler imputado)	28792.93
06 Cantabria	Renta media por hogar (con alquiler imputado)	32150.47
09 Cataluña	Renta neta media por hogar	32500.13
04 Balears, Illes	Renta neta media por hogar	30023.87
14 Murcia, Región de	Renta neta media por hogar	24722.60
12 Galicia	Renta media por hogar (con alquiler imputado)	31307.00
08 Castilla - La Mancha	Renta neta media por hogar	24527.33
18 Ceuta	Renta media por hogar (con alquiler imputado)	36182.53
16 País Vasco	Renta media por hogar (con alquiler imputado)	40669.33

9. Comparación de la renta neta media por hogar entre diferentes comunidades autónomas:

```
SELECT comunidad_autonoma,
ROUND(AVG(renta_neta_media_hogar)::numeric, 2) AS renta_media
FROM renta_hogares
WHERE comunidad_autonoma IN ('19 Melilla', '18 Ceuta', '17 Rioja, La')
GROUP BY comunidad_autonoma;
```

comunidad_autonoma character varying (100)	renta_media numeric
18 Ceuta	33743.33
19 Melilla	38643.30
17 Rioja, La	30411.10

10. Rentas medias por hogar por año y tipo de renta:

```

SELECT EXTRACT(YEAR FROM TO_TIMESTAMP(periodo::text, 'YYYY')) AS
año,tipo_renta,
    ROUND(AVG(renta_neta_media_hogar)::numeric, 2) AS renta_media
FROM renta_hogares
GROUP BY año, tipo_renta;

```

año numeric	tipo_renta character varying (100)	renta_media numeric
2022	Renta neta media por hogar	32334.25
2017	Renta neta media por hogar	27853.55
2018	Renta neta media por hogar	28826.50
2012	Renta media por hogar (con alquiler imputado)	32096.50
2010	Renta media por hogar (con alquiler imputado)	34062.55
2012	Renta neta media por hogar	28194.35
2009	Renta media por hogar (con alquiler imputado)	34079.60
2017	Renta media por hogar (con alquiler imputado)	32233.90
2011	Renta neta media por hogar	28726.10
2019	Renta media por hogar (con alquiler imputado)	33964.15
2016	Renta neta media por hogar	27344.80
2020	Renta neta media por hogar	30599.40
2008	Renta media por hogar (con alquiler imputado)	32756.30
2014	Renta media por hogar (con alquiler imputado)	30725.35
2016	Renta media por hogar (con alquiler imputado)	31480.65
2018	Renta media por hogar (con alquiler imputado)	33316.05
2013	Renta neta media por hogar	27155.80
2021	Renta media por hogar (con alquiler imputado)	35281.85
2015	Renta neta media por hogar	26461.05
2019	Renta neta media por hogar	29339.40
2013	Renta media por hogar (con alquiler imputado)	30910.95
2014	Renta neta media por hogar	26625.20
2009	Renta neta media por hogar	30214.60
2008	Renta neta media por hogar	28875.95

PLANIFICACIÓN PARA HERRAMIENTA SOFTWARE

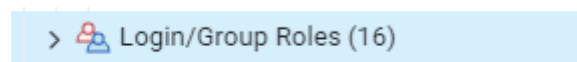
En el desarrollo de una herramientas de software para el análisis de datos, aplicaremos el código de programación tanto de lenguajes generales como de lenguajes específicos orientados al análisis de datos. En este contexto, JAVA es un lenguaje de programación ampliamente utilizado, que puede integrarse con sistemas de bases de datos para ejecutar consultas SQL y procesar los resultados de manera efectiva.

A continuación, se presenta un ejemplo de código en Java que ejecuta una consulta SQL específica y muestra los resultados en la consola, lo que ilustra cómo se pueden integrar las capacidades de Java y SQL en el diseño de herramientas de software para análisis de datos.

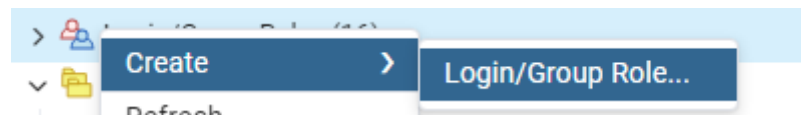
Primero, antes de realizar la acción en JAVA, tenemos que añadir un usuario y contraseña si no lo hemos hecho antes.

Lo haremos de la siguiente manera:

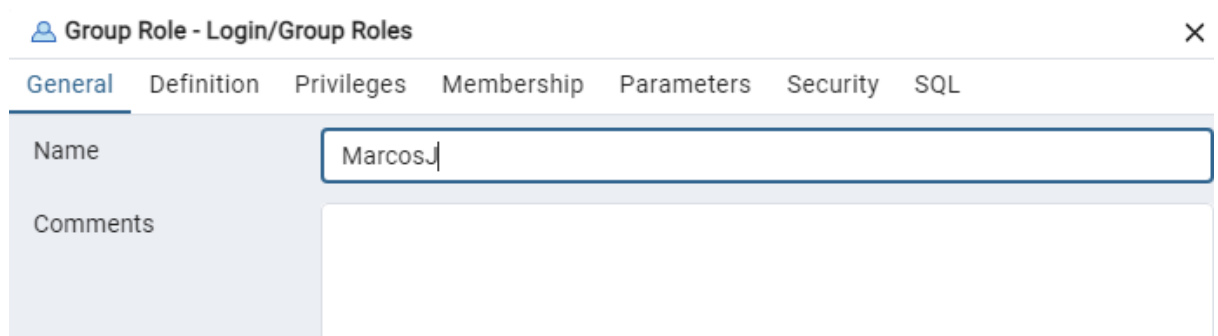
- 1) Vete a el árbol de navegación del lado izquierdo, expande el servidor al que te has conectado para ver las bases de datos y roles.



- 2) Haz clic con el botón derecho del ratón en "Roles" y selecciona "Create" -> "Login/Group Role...".



- 3) Se abrirá una ventana para crear un nuevo rol. En la pestaña "General", ingresa el nombre del nuevo usuario en el campo "Name".


A screenshot of the 'Group Role - Login/Group Roles' dialog box. The 'General' tab is selected. The 'Name' field contains the text 'MarcosJ'. The 'Comments' field is empty. The dialog has tabs for 'General', 'Definition', 'Privileges', 'Membership', 'Parameters', 'Security', and 'SQL'.

- 4) En la pestaña "Definition", ingresa una contraseña para el nuevo usuario en el campo "Password" y marca la casilla "Password" para habilitar la autenticación de contraseña.

Group Role - Login/Group Roles

General Definition Privileges Membership Parameters Security SQL

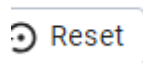

Password

Account expires 

Please note that if you leave this field blank, then password will never expire.

Connection limit

5) Haz clic en "Save" para crear el nuevo usuario.

Así ya tenemos un usuario y contraseña para poder abrir nuestra bbdd en JAVA.

Vamos a aplicar una de las consultas anteriores, en concreto realizaremos la de Total de renta neta media por hogar por año, para saber la media de la renta neta que hay por hogar anualmente, es decir, la media de los ingresos totales que queda una vez que se han recortado todos los costos asociados al generar ingresos en un año, generamos este código de consulta SQL.

```
SELECT EXTRACT(YEAR FROM TO_TIMESTAMP(periodo::text, 'YYYY')) AS  
año, SUM(renta_neta_media_hogar) AS total_renta  
FROM renta_hogares  
GROUP BY año;
```

Ahora aplicaremos el JDBC en tu proyecto en Eclipse. Aquí estan los pasos

1. Debes descargar el controlador JDBC de *PostgreSQL* desde su sitio web oficial, que te dejo aquí = [POSTGRESQL](#) > DOWNLOAD > Te vas a **Older Versions** y te descargas la ultima version (42.7.0)

Older Versions

Many other versions of the JDBC driver are available.
versions of the driver.

 [42.7.0](#)

 [42.6.0](#)

 [42.5.4](#)

 [42.5.3](#)

 [42.5.2](#)

 [42.5.1](#)

 [42.4.2](#)

 [42.3.7](#)

2. Luego agregaremos el archivo JAR a tu proyecto.

Una vez que hayas descargado el archivo JAR, debemos incluirlo en el proyecto.

PASOS:

PASO 1: Copia el archivo JAR que descargaste en un directorio de tu proyecto, por ejemplo, en una carpeta llamada "controlador", en mi caso, aunque comunmente se le llama "lib".

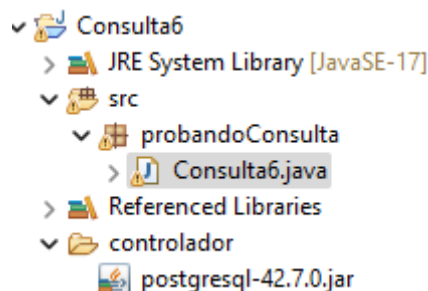
PASO 2: En Eclipse, haz clic derecho en tu proyecto y selecciona "Properties" (Propiedades).

PASO 3: En el panel izquierdo, selecciona "Java Build Path" (Ruta de compilación de Java).

PASO 4: Ve a la pestaña "Libraries" (Bibliotecas) y haz clic en "Add JARs..." (Agregar archivos JAR...).

PASO 5: Selecciona el archivo JAR que has descargado y copiado en tu proyecto y haz clic en "OK".

Cuando hayas agregado el archivo JAR a tu proyecto, dale a "Apply and close". Vuelve a ejecutar tu programa y debería poder encontrar el controlador JDBC de PostgreSQL y establecer la conexión correctamente. Debería quedarte así:



A la hora de aplicarlo a JAVA, con Eclipse EE, quedaria algo así:

1) Configuración de la conexión a la base de datos

Es hora de empezar. Primero colocaremos los datos para para realizar la conexión con la base de datos:

- El **URL** de la bbdd == `jdbc:postgresql://localhost:5432/renta`
- El **usuario** para acceder a ella ==
- La **contraseña** para entrar en la bbdd ==

```
import ...
public class Consulta6 {
    public static void main(String[] args) {
        String url = "jdbc:postgresql://localhost:5432/renta";
        String usuario = "postgres";
        String contraseña = "MarcosGuapo1";
```

2) Colocar consulta

Añadimos la consulta que queremos realizar, todo entera:

```
String consulta =
    "SELECT EXTRACT(YEAR FROM TO_TIMESTAMP(periodo::text, 'YYYY')) AS
año, SUM(renta_neta_media_hogar) AS total_renta FROM renta_hogares GROUP BY
año;"
```

3) Añadir título

Añadimos el titulo para que podamos identificar bien de qué va la consulta que realizaremos:

```
String funcion = "Total de renta neta media por hogar por año";
```

4) Establecemos conexión con la bbdd y ejecutamos la consulta

Establecemos la conexión mediante el método `getConnection`, que coje los parámetros seleccionados anteriormente, `url`, `usuario` y `contraseña`. Después creamos objeto `Statement` para ejecutar declaraciones SQL. El resultado se almacenará en el `ResultSet`. y aparecerá un mensaje para informar de la conexión correcta.

```
try (Connection conexion = DriverManager.getConnection(url, usuario,
contraseña);
    Statement statement = conexion.createStatement();
    ResultSet resultSet = statement.executeQuery(consulta)) {
    System.out.println("Se ha establecido conexion con la bbdd");
```

5) Procesar el resultado

Se itera el `ResultSet` y obtiene la consulta. En cada iteración, se extraen los valores del año y total de renta neta media por hogar por año y se imprimen en la consola.

```
while (resultSet.next()) {
```

```

        int año = resultSet.getInt("año");
        int totalRenta = resultSet.getInt("total_renta");
        System.out.println("Año: " + año + ", Total de renta neta media:
" + totalRenta);
    }

```

6) Mensaje de error

Si esto NO ocurre, aparecerá este mensaje para informar del error

```

//por si da error
    } catch (SQLException e) {
        System.out.println("Error al ejecutar la consulta SQL:");
        e.printStackTrace();
    }
}
}

```

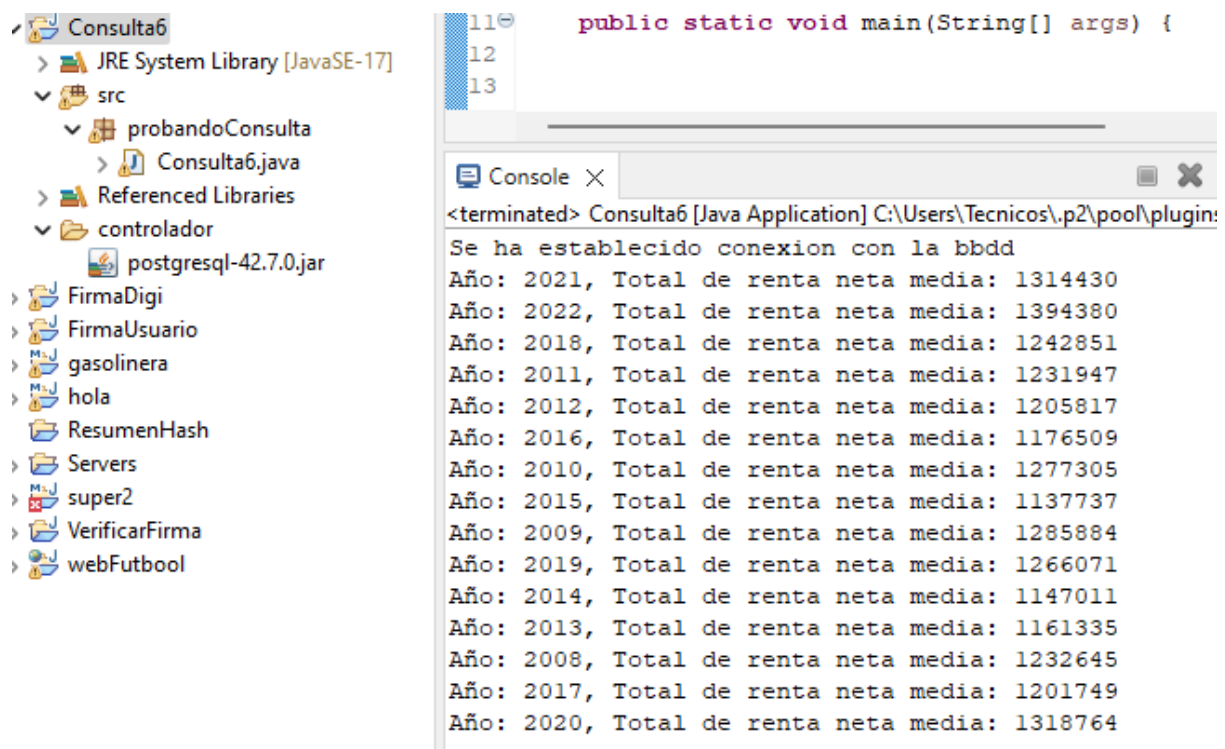
EJEMPLO DE ERROR:

```

Error al ejecutar la consulta SQL:
org.postgresql.util.PSQLException: FATAL: la autentificación password falló
para el usuario postgres
    at
    org.postgresql.core.v3.ConnectionFactoryImpl.doAuthentication(ConnectionFac
    toryImpl.java:693)
    at
    org.postgresql.core.v3.ConnectionFactoryImpl.tryConnect(ConnectionFactoryIm
    pl.java:203)
    at
    org.postgresql.core.v3.ConnectionFactoryImpl.openConnectionImpl(ConnectionF
    actoryImpl.java:258)
    at
    org.postgresql.core.ConnectionFactory.openConnection(ConnectionFactory.java
    :54)
    at org.postgresql.jdbc.PgConnection.<init>(PgConnection.java:263)
    at org.postgresql.Driver.makeConnection(Driver.java:443)
    at org.postgresql.Driver.connect(Driver.java:297)
    at
    java.sql/java.sql.DriverManager.getConnection(DriverManager.java:681)
    at
    java.sql/java.sql.DriverManager.getConnection(DriverManager.java:229)
    at probandoConsulta.Consulta6.main(Consulta6.java:28)

```

RESULTADO:



FORMAS DE LENGUAJES DE PROGRAMACIÓN PARA EXTENDER LA FUNCIONALIDAD DE TAREAS Y MANIPULACIÓN DE OBJETOS DE ANÁLISIS DE DATOS.

Para extender la funcionalidad de tareas intensivas y manipular objetos de análisis de datos directamente, es crucial poder integrar y llamar código escrito en diferentes lenguajes de programación en tiempo de ejecución. Esto se logra a través de varios mecanismos de interoperabilidad (capacidad para comunicarse, cooperar y trabajar juntos eficientemente) entre lenguajes. Lenguajes de programación que pueden lograr esto:

Interoperabilidad entre Lenguajes:

❖ Python con C/C++:

- Python ofrece la capacidad de extender su funcionalidad a través de módulos escritos en C/C++. Estos módulos se pueden enlazar y llamar desde Python usando la interfaz Python/C API.

❖ Java con JNI (Java Native Interface):

- Java proporciona JNI para integrar código nativo escrito en C/C++ en aplicaciones Java. JNI permite llamar funciones desde Java y viceversa.

❖ R con C/C++:

- R es un lenguaje popular para el análisis de datos, integrando con código C/C++ para mejorar el rendimiento de operaciones intensivas en datos.

❖ **TensorFlow con Python y C++:**

- TensorFlow es una biblioteca popular para el aprendizaje automático y la inteligencia artificial (IA). Conocido por su interfaz de Python, y de C++ que permite integrar modelos de TensorFlow en aplicaciones que NO son de Python, como sistemas embebidos o aplicaciones de alto rendimiento.

❖ **Hadoop con Python y Java:**

- Apache Hadoop procesa datos a gran escala. Se puede interactuar con *Hadoop* utilizando Java para desarrollar aplicaciones *MapReduce* (*procesar y analizar grandes volúmenes de datos en sistemas distribuidos*) y también utilizando Python con el módulo *Hadoop Streaming*, que permite escribir programas MapReduce en Python.

❖ **NumPy y SciPy con C y Fortran:**

- *NumPy* y *SciPy* son bibliotecas de Python para uso científico y análisis numérico. Muchas de las funciones de rendimiento de estas bibliotecas están implementadas en *C* y *Fortran*, permitiendo una gran eficiencia y rendimiento en operaciones numéricas.

❖ **Dask con Python:**

- Dask es una biblioteca de Python que permite la computación paralela y distribuida. Aunque está escrito en Python, también utiliza bibliotecas como *NumPy*, *Pandas* y *Scikit-learn*, proporcionando una forma eficiente de trabajar con grandes volúmenes de datos en paralelo.

❖ **PySpark con Scala/Java:**

- Además de la interoperabilidad de Apache Spark con Java, *PySpark* permite a los usuarios interactuar con *Apache Spark* desde Python. *PySpark* proporciona una API similar a la de Spark en Scala o Java, lo que facilita a los desarrolladores trabajar con Spark desde el ecosistema de Python.

Apache Spark con Scala/Java/Python:

Apache Spark es un potente motor de procesamiento de datos de código abierto diseñado para realizar análisis de datos a gran escala y procesamiento de datos distribuido de manera eficiente. ofrece API para Scala, Java y Python. Los usuarios pueden escribir extensiones personalizadas en Scala o Java para Spark, y también pueden aprovechar la biblioteca PySpark para interactuar con Spark desde Python. Fue desarrollado originalmente en la Universidad de California, y posteriormente se convirtió en un proyecto de la *Apache Software Foundation*.



Ejemplo con Apache Spark en Scala:

Apache Spark, escrito en Scala, permite procesamiento de datos distribuido en clústeres.

```
import org.apache.spark.sql.SparkSession

object SparkExample {
  def main(args: Array[String]) {
    val spark = SparkSession.builder()
      .appName("SparkExample")
      .getOrCreate()

    // Crear un DataFrame a partir de una lista
    val data = Seq(("Juan", 25), ("María", 30), ("Pedro", 35), ("Luis", 40))
    val df = spark.createDataFrame(data).toDF("Nombre", "Edad")

    // Mostrar el DataFrame
    df.show()

    // Realizar transformaciones y acciones en el DataFrame
    // ...

    spark.stop()
  }
}
```

En este ejemplo, utilizamos Apache Spark con Scala para crear un DataFrame a partir de una lista de tuplas y realizar operaciones de procesamiento distribuido en los datos.

Ejemplo con Pandas en Python:

Pandas es una biblioteca popular para manipulación y análisis de datos en Python desde distintos tipos de tablas (csv, json, html, excel,...). Utiliza estructuras de datos como *DataFrames* y *Series* para manejar datos eficientemente.

```
import pandas as pd

# Crear un DataFrame de ejemplo
data = {'Nombre': ['Juan', 'María', 'Pedro', 'Luis'],
        'Edad': [25, 30, 35, 40]}
df = pd.DataFrame(data)
```

```
# Mostrar el DataFrame
print("DataFrame original:")
print(df)

# Agregar una nueva columna
df['Profesión'] = ['Ingeniero', 'Doctor', 'Abogado', 'Profesor']

# Mostrar el DataFrame actualizado
print("\nDataFrame con nueva columna:")
print(df)
```

En este ejemplo, Pandas facilita la manipulación de datos mediante la adición de una nueva columna al DataFrame existente.

Estos ejemplos ilustran cómo los lenguajes de programación y las herramientas pueden integrarse para extender la funcionalidad y manipular objetos de análisis de datos de manera eficiente. La elección del lenguaje y las herramientas depende de los requisitos específicos del proyecto y de las preferencias del equipo de desarrollo.

PARA CONECTAR MI BBDD PostgreSQL con Python:

```
import pandas as pd
import psycopg2
from urllib import request as rq
import ssl

ssl._create_default_https_context = ssl._create_unverified_context

# Conectar a la base de datos PostgreSQL (reemplaza con tus propias
credenciales y detalles)
conn = psycopg2.connect(
    host="127.0.0.1",
    database="renta",
    user="postgres",
    password="MarcosGuapo1"
)

# Crear un cursor
cursor = conn.cursor()

# Ejecutar una consulta SQL (reemplaza con tu propia consulta)
consulta_sql = "SELECT * FROM renta_hogares;"
```

```

cursor.execute(consulta_sql)

# Obtener los resultados en un DataFrame de pandas
columnas = [desc[0] for desc in cursor.description]
dataframe = pd.DataFrame(cursor.fetchall(), columns=columnas)

# Cerrar el cursor y la conexión
cursor.close()
conn.close()

# Rellenar los valores faltantes con espacios en blanco
dataframe = dataframe.fillna('')

# Imprimir el DataFrame
print(dataframe.to_string(index=False))

```

comunidad_autonoma	tipo_renta	periodo	renta_neta_media_hogar
Total Nacional	Renta neta media por hogar	2022	32216
Total Nacional	Renta neta media por hogar	2021	30552
Total Nacional	Renta neta media por hogar	2020	30690
Total Nacional	Renta neta media por hogar	2019	29132
Total Nacional	Renta neta media por hogar	2018	28417
Total Nacional	Renta neta media por hogar	2017	27558
Total Nacional	Renta neta media por hogar	2016	26730
Total Nacional	Renta neta media por hogar	2015	26092
Total Nacional	Renta neta media por hogar	2014	26154
Total Nacional	Renta neta media por hogar	2013	26775
Total Nacional	Renta neta media por hogar	2012	27747
Total Nacional	Renta neta media por hogar	2011	28206
Total Nacional	Renta neta media por hogar	2010	29634
Total Nacional	Renta neta media por hogar	2009	30045
Total Nacional	Renta neta media por hogar	2008	28787
Total Nacional Renta media por hogar (con alquiler imputado)		2022	37363
Total Nacional Renta media por hogar (con alquiler imputado)		2021	35497
Total Nacional Renta media por hogar (con alquiler imputado)		2020	35485
Total Nacional Renta media por hogar (con alquiler imputado)		2019	33794
Total Nacional Renta media por hogar (con alquiler imputado)		2018	32929
Total Nacional Renta media por hogar (con alquiler imputado)		2017	31956
Total Nacional Renta media por hogar (con alquiler imputado)		2016	30822
Total Nacional Renta media por hogar (con alquiler imputado)		2015	30031
Total Nacional Renta media por hogar (con alquiler imputado)		2014	30257
...			
19 Melilla Renta media por hogar (con alquiler imputado)		2011	40890
19 Melilla Renta media por hogar (con alquiler imputado)		2010	41844
19 Melilla Renta media por hogar (con alquiler imputado)		2009	38928
19 Melilla Renta media por hogar (con alquiler imputado)		2008	38099

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

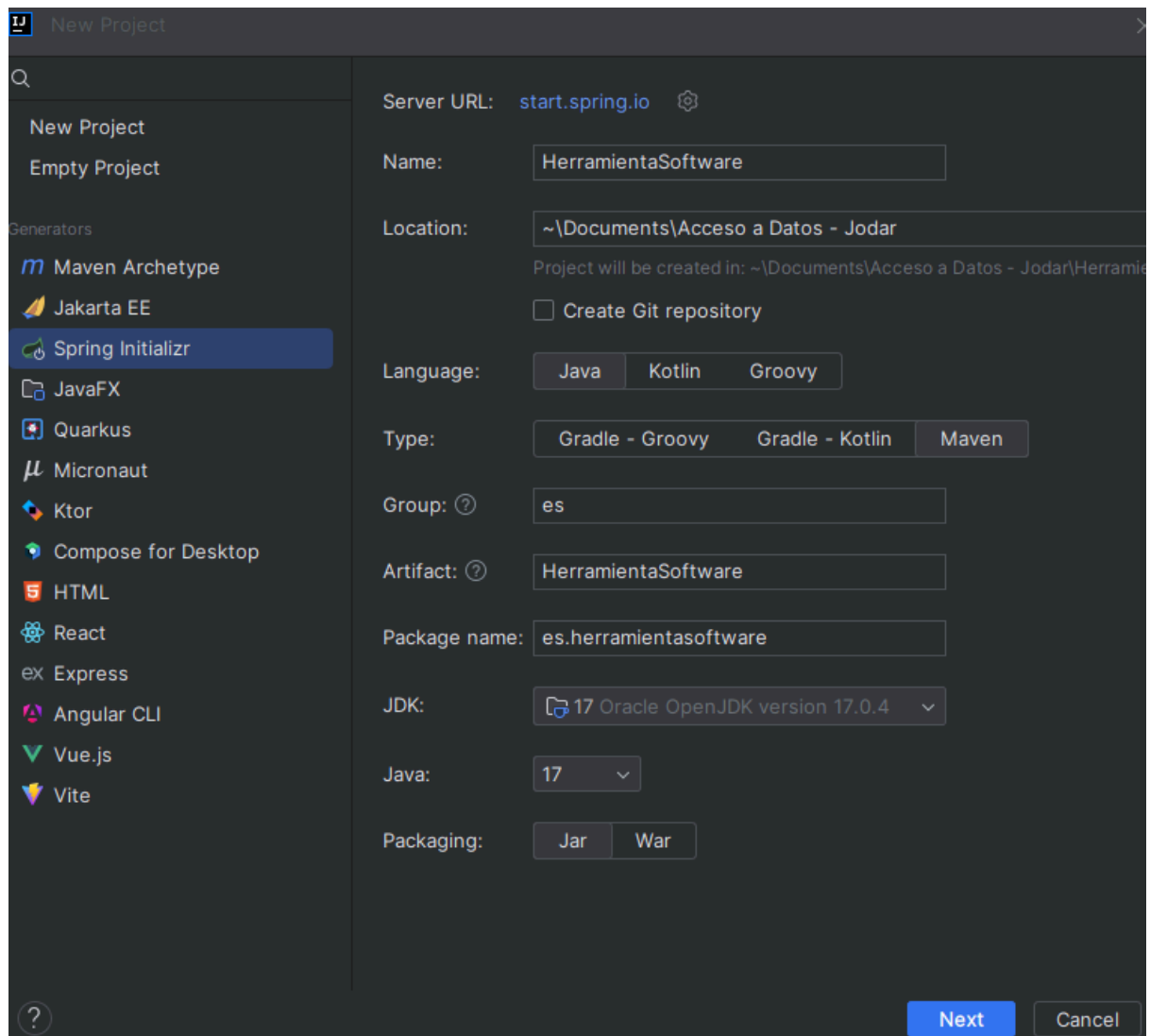
Conecta con nuestra base de datos PostgreSQL, ejecutando la consulta para recuperar todos los datos de *renta_hogares* y los muestra en forma de DataFrame de pandas. Esta es la forma de trabajar con datos almacenados en bases de datos relacionales usando Python.

1. Este script de Python importa las **bibliotecas**:
 - **pandas** para manejar datos en forma de **DataFrames**
 - **psycopg2** para conectarse a la base de datos PostgreSQL
 - **ssl** para configurar la conexión segura HTTPS sin verificar.
2. Establece una conexión con una base de datos PostgreSQL local.

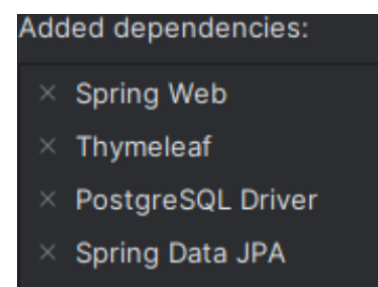
3. Creamos un cursor para ejecutar consultas SQL en la base de datos.
4. Colocamos la consulta SQL que selecciona TODOS los registros de nuestra tabla *renta_hogares*.
5. Obtiene los resultados de la consulta y los almacena en un *DataFrame* de pandas.
6. Cierra el cursor y la conexión a la bbdd una vez que se completan las consultas y se recuperan los datos.
7. Rellena los valores faltantes en el *DataFrame* con espacios en blanco.
8. Imprime el DataFrame resultante sin incluir el índice de fila.

DESARROLLO DE HERRAMIENTA SOFTWARE

Ahora, una vez creada la Base de Datos y mirar diferentes lenguajes para analizar datos, toca desarrollar una aplicación utilizando **Java Spring Boot** para obtener y analizar datos relevantes de nuestra base de datos llamada Renta. Utilizaremos Spring Boot para construir una aplicación eficiente que nos permitirá acceder a los diferentes datos, como comunidades, Periodos, etc., procesarlos y realizar análisis. Con Java Spring Boot, simplificamos el desarrollo y extraemos información valiosa.



Añadimos las siguientes Dependencias:



1. Spring Web:

Spring Web proporciona soporte para el desarrollo de aplicaciones WEB basadas en *Spring Framework*. Con funcionalidades como la *gestión de peticiones HTTP*, que nos sera de mucha importancia en el proyecto, la *implementación de controladores*, la *gestión de vistas* y la *integración con otros componentes de Spring*.

2. Thymeleaf

Thymeleaf es un motor de plantillas para el desarrollo de vistas en aplicaciones WEB JAVA. Permite la creación de plantillas HTML que se pueden integrar fácilmente con el código Java, lo que facilita la presentación dinámica de datos en las páginas web.

3. PostgreSQL Driver

PostgreSQL es el sistema de gestión de bases de datos relacional que hemos estado viendo a lo largo del trabajo, donde tenemos la base de datos que utilizaremos. En este caso, este es su Driver, biblioteca que proporciona la conexión entre la aplicación y la base de datos, permitiendo a la aplicación realizar consultas, actualizaciones y otras operaciones en la base de datos.

4. Spring Data JPA

Spring Data JPA (Java Persistence API) es una especificación de Java que permite el mapeo objeto-relacional (ORM), para el acceso y manipulación de datos en la base de datos, desde una aplicación Java. Simplifica el desarrollo de repositorios de datos, con una interfaz común y métodos predefinidos para interactuar con la base de datos.

Cada una de las dependencias desempeña un papel específico en el desarrollo de una aplicación web con Spring Framework. **Juntas**, estas tecnologías se combinan y forman un entorno completo para el desarrollo de aplicaciones web robustas y escalables, desde el manejo de peticiones HTTP hasta el acceso y la manipulación de datos en una base de datos relacional, pasando por la generación dinámica de contenido HTML mediante plantillas. Juntas, proporcionan un conjunto poderoso de herramientas para el desarrollo ágil y eficiente de aplicaciones web empresariales.

CONFIGURACIÓN DEL PROYECTO:

Las dependencias se añadirán a nuestro Pom.

POM:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.2</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>es</groupId>
```

```

<artifactId>HHerramientaSoftware</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>HHerramientaSoftware</name>
<description>HHerramientaSoftware</description>
<properties>
    <java.version>17</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

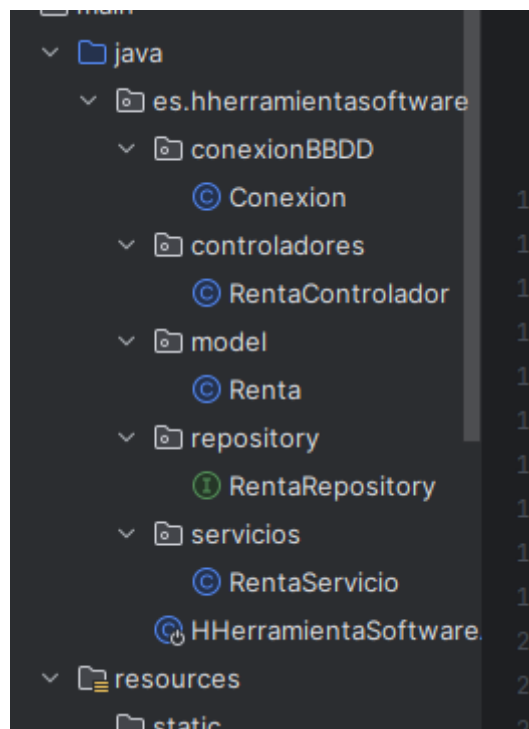
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>

```

BACK-END

En el backend de la aplicación, veremos como se establece una **conexión con la base de datos PostgreSQL** y se **configura un controlador para manejar las solicitudes HTTP**. El **modelo de datos** está representado por la clase Renta, y el repositorio RentaRepository proporciona **métodos para acceder a la base de datos**. El servicio RentaServicio realiza **operaciones de negocio** y utiliza el repositorio para **interactuar con los datos**.

- 1) Conexión
- 2) Controlador → RentaControlador
- 3) Model → Renta
- 4) Repository → RentaRepository
- 5) Servicios → RentaServicio



1) Conexión:

```
package es.herramientasoftware.conexionBBDD;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;
import java.sql.DriverManager;

@Configuration
public class Conexion {
```

```

@Bean
public DataSource ds() {
    DriverManagerDataSource drivers = new
DriverManagerDataSource();

    drivers.setDriverClassName("org.postgresql.Driver");
    drivers.setUrl("jdbc:postgresql://localhost:5432/renta");
    drivers.setUsername("postgres");
    drivers.setPassword("MarcosGuapo1");
    return drivers;
}
}

```

Se está creando la conexión a la base de datos "renta" de PostgreSQL utilizando Spring Framework en la de **Renta de hogares Media por Comunidad Autónoma** en una aplicación Java.

La clase **Conexion**, configura el Spring para que se conecte a la base de datos.

Anotaciones:

- **@Configuration** → anotación que indica a Spring que la clase contendrá métodos de configuración, que deberán procesarse durante el inicio de la aplicación.
- **El método ds() + anotación @Bean** → Anotación para que Spring administre un bean en su contenedor de beans.

Método ds():

- Devuelve un objeto **DataSource**.
- Dentro del método, se instancia un *DriverManagerDataSource*, implementación de la interfaz *DataSource* proporcionada por Spring para configurar la conexión a la base de datos.
- Se configuran las propiedades del *DriverManagerDataSource* con:
 1. Nombre del controlador *JDBC* (org.postgresql.Driver)
 2. URL de la base de datos:
(jdbc:postgresql://localhost:5432/renta)
 3. Nombre de usuario (postgres)
 4. Contraseña (MarcosGuapo1).
- Finalmente, el método devuelve el objeto *DriverManagerDataSource*.

2) Controlador:

```

package es.hherramientasoftware.controladores;

import org.springframework.ui.Model;
import es.hherramientasoftware.model.Renta;
import es.hherramientasoftware.servicios.RentaServicio;
import org.springframework.beans.factory.annotation.Autowired;

```

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

import java.util.List;

@Controller
public class RentaControlador {

    private final RentaServicio datosServicio;

    @Autowired
    public RentaControlador(RentaServicio datosServicio) {
        this.datosServicio = datosServicio;
    }

    @GetMapping("/consulta1")
    public String cojerTodoslosDatos(Model model){
        List<Renta> datos = datosServicio.cojerTodoslosDatos();
        model.addAttribute("datos", datos);
        return "consulta1";
    }

    @GetMapping("/consulta2")
    public String consult2(Model model) { // Ejecutar la consulta
en la base de datos
        List<Object[]> resultado = datosServicio.consulta2();
        // Pasar los resultados a la plantilla Thymeleaf
        model.addAttribute("resultados", resultado);

        // Renderizar la vista correspondiente
        return "consulta2";
    }

    // Endpoint para mostrar la consulta 3
    @GetMapping("/consulta3")
    public String consult3(Model model) {
        // Ejecutar la consulta en la base de datos
        List<Object[]> resultado = datosServicio.consulta3();

        // Pasar los resultados a la plantilla Thymeleaf
        model.addAttribute("rentas", resultado);

        // Renderizar la vista correspondiente
        return "consulta3";
    }
}
```

```
// Endpoint para mostrar la consulta 4
@GetMapping("/consulta4")
public String consult4(Model model) {
    // Ejecutar la consulta en la base de datos
    List<Object[]> resultado = datosServicio.consulta4();

    // Pasar los resultados a la plantilla Thymeleaf
    model.addAttribute("rentas", resultado);

    // Renderizar la vista correspondiente
    return "consulta4";
}

// Endpoint para mostrar la consulta 5
@GetMapping("/consulta5")
public String consult5(Model model) {

    List<Object[]> resultado = datosServicio.consulta5();

    model.addAttribute("resultados", resultado);

    return "consulta5";
}

// Endpoint para mostrar la consulta 6
@GetMapping("/consulta6")
public String consult6(Model model) {
    // Ejecutar la consulta en la base de datos
    List<Object[]> resultado = datosServicio.consulta6();

    model.addAttribute("resultados", resultado);

    return "consulta6";
}

// Endpoint para mostrar la consulta 7
@GetMapping("/consulta7")
public String consult7(Model model) {
    // Ejecutar la consulta en la base de datos
    List<Object[]> resultado = datosServicio.consulta7();

    model.addAttribute("resultados", resultado);

    return "consulta7";
}
```

```

}

// Endpoint para mostrar la consulta 8
@GetMapping("/consulta8")
public String consult8(Model model) {
    // Ejecutar la consulta en la base de datos
    List<Object[]> resultado = datosServicio.consulta8();

    // Pasar los resultados a la plantilla Thymeleaf
    model.addAttribute("resultados", resultado);
    // Renderizar la vista correspondiente
    return "consulta8";
}

// Endpoint para mostrar la consulta 9
@GetMapping("/consulta9")
public String consult9(Model model) {
    // Ejecutar la consulta en la base de datos
    List<Object[]> resultado = datosServicio.consulta9();

    // Pasar los resultados a la plantilla Thymeleaf
    model.addAttribute("resultados", resultado);
    // Renderizar la vista correspondiente
    return "consulta9";
}

// Endpoint para mostrar la consulta 10
@GetMapping("/consulta10")
public String consult10(Model model) {
    // Ejecutar la consulta en la base de datos
    List<Object[]> resultado = datosServicio.consulta10();

    model.addAttribute("resultados", resultado);
    return "consulta10";
}
}

```

Un controlador de *Spring MVC* ==> Maneja solicitudes HTTP para consultar y mostrar datos relacionados con rentas. Cada consulta se realiza a través del servicio **RentaServicio**, y los resultados se pasan a plantillas Thymeleaf para su renderización.

Funciones:

Constructor: Se define un constructor con un objeto *RentaServicio* como parámetro. Este objeto es **@Autowired** (inyectado) por Spring. El *RentaServicio* proporciona métodos para acceder a los datos relacionados con las rentas.

Funciones:

- Se definen muchas funciones de consulta, cada una asociada a una ruta específica (*/consulta1*, */consulta2*, ..., */consulta10*). Cada función de consulta acepta un parámetro *Model*, para pasar datos a las plantillas **Thymeleaf**, que utilizaremos más tarde para su renderización.
- Cada función de consulta llama a un método correspondiente en el *RentaServicio* para obtener los datos necesarios.
- Los resultados de las consultas se agregan al modelo (*Model*) con un nombre específico, para que puedan ser accedidos desde las plantillas.

Retorno de Vistas:

- Cada función de consulta devuelve el nombre de una vista *Thymeleaf*. Este nombre se utilizará para saber qué plantilla se debe renderizar después de la solicitud.

Inyección de Dependencias:

- Anotación **@Autowired** → inyecta una instancia de *RentaServicio* en el constructor del controlador.

3) Renta:

```
package es.herramientasoftware.model;

import jakarta.persistence.*;

import javax.xml.crypto.Data;
import java.time.LocalDate;

@Entity
@Table(name = "renta_hogares")
public class Renta {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "comunidad_autonoma")
    private String comunidadA;

    @Column(name = "tipo_renta")
    private String tipoR;
```

```
@Column(name = "periodo")
private int periodo;

@Column(name = "renta_neta_media_hogar")
private int rentaM;

public Renta(String comunidadA, String tipoR, int periodo, int
rentaM) {
    this.comunidadA = comunidadA;
    this.tipoR = tipoR;
    this.periodo = periodo;
    this.rentaM = rentaM;
}

public Renta() {}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getComunidadA() {
    return comunidadA;
}

public void setComunidadA(String comunidadA) {
    this.comunidadA = comunidadA;
}

public String getTipoR() {
    return tipoR;
}

public void setTipoR(String tipoR) {
    this.tipoR = tipoR;
}

public int getPeriodo() {
    return periodo;
}

public void setPeriodo(int periodo) {
    this.periodo = periodo;
}
```

```

    }

    public int getRentaM() {
        return rentaM;
    }

    public void setRentaM(int rentaM) {
        this.rentaM = rentaM;
    }
}

```

La clase Renta, es una entidad que representa y modela los datos relacionados de la renta de hogares. Proporciona métodos para acceder y modificar estos datos, y está diseñada para integrarse fácilmente con una bbdd relacional utilizando Java Persistence API (**JPA**) y la anotación de mapeo de columnas.

Funciones:

Anotaciones de JPA:

- **@Entity** → Indica la clase es una entidad *JPA*, es decir, que está mapeada a una tabla en la bbdd.
- **@Table(name = "renta_hogares")** → Especifica el nombre de la tabla en la bbdd a la que se asignará esta entidad.

Atributos de la clase:

- **id** → Clave primaria de la entidad. Con anotación == **@Id** para indicar que es la clave primaria.
- **comunidadA** → Representa el nombre de la comunidad autónoma, siendo la columna **comunidad_autonoma** en la tabla.
- **tipoR** → Tipo de renta, siendo la columna **tipo_renta** en la tabla.
- **periodo** → Periodo, siendo la columna **periodo** en la tabla.
- **rentaM** → Renta neta media del hogar, siendo la columna **renta_neta_media_hogar** en la tabla.

Constructores:

- Con un constructor que inicializa todos los atributos de la clase.
- Con otro constructor vacío, que puede ser útil para inicializar objetos Renta sin proporcionar todos los detalles de inmediato.

Métodos de acceso (Getters y Setters):

- Se proporcionan métodos get y set para todos los atributos de la clase. Métodos para acceder y modificar los valores de los atributos de la clase desde otras clases.
- **Métodos get** ⇒ permiten recuperar los valores de los atributos.
- **Métodos set** ⇒ permiten establecer los valores de los atributos.

4) Repository:

```
package es.herramientasoftware.repository;
```



```

import es.hherramientasoftware.model.Renta;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface RentaRepository extends JpaRepository<Renta,
Long> {

    List<Renta> findAll(); //cojer los datos

    // Consulta 2
    @Query("SELECT AVG(r.rentaM) AS promedio_renta FROM Renta r")
    List<Object[]> consulta2();

    // Consulta 3
    @Query("SELECT r.comunidadA , MAX(r.rentaM) AS max_renta FROM Renta r GROUP BY r.comunidadA")
    List<Object[]> consulta3();

    // Consulta 4
    @Query("SELECT r.comunidadA, COUNT(*) AS cantidad_hogares FROM Renta r GROUP BY r.comunidadA")
    List<Object[]> consulta4();

    // Consulta 5
    @Query("SELECT r.tipoR, MAX(r.rentaM) AS max_renta, MIN(r.rentaM) AS min_renta FROM Renta r GROUP BY r.tipoR")
    List<Object[]> consulta5();

    // Consulta 6
    @Query("SELECT SUBSTRING(CAST(r.periodo AS STRING), 1, 4), SUM(r.rentaM) FROM Renta r GROUP BY SUBSTRING(CAST(r.periodo AS STRING), 1, 4)")
    List<Object[]> consulta6();

    // Consulta 7

```

```

    @Query("SELECT r.comunidadA, r.tipoR, AVG(r.rentaM) FROM Renta r GROUP BY r.comunidadA, r.tipoR")
    List<Object[]> consulta7();

    // Consulta 8
    @Query("SELECT r.comunidadA, AVG(r.rentaM) FROM Renta r WHERE r.comunidadA IN ('19 Melilla', '18 Ceuta', '17 Rioja, La') GROUP BY r.comunidadA")
    List<Object[]> consulta8();

    // Consulta 9
    @Query("SELECT r.periodo, r.tipoR, AVG(r.rentaM) FROM Renta r GROUP BY r.periodo, r.tipoR")
    List<Object[]> consulta9();

    // Consulta 10
    @Query("SELECT COUNT(DISTINCT r.periodo) AS cantidad_periodos FROM Renta r")
    List<Object[]> consulta10();
}

```

RentaRepository ==> actúa como un **repositorio de datos para la entidad Renta**. Interfaz proporcionada por *Spring Data JPA* que extiende *JpaRepository*, es decir, que hereda métodos predefinidos para interactuar con la base de datos y realizar operaciones **CRUD (Crear, Leer, Actualizar, Eliminar)** en la entidad Renta.

Funciones:

findAll():

- Método que devuelve una lista de todas las instancias de la entidad Renta almacenadas en la base de datos.

Consultas personalizadas:

- Definimos las consultas que hemos elegido y utilizamos la anotación **@Query** de Spring Data JPA. Consultas escritas en *JPQL* (Java Persistence Query Language), utilizándose para realizar operaciones más específicas en la bbdd.
- Cada consulta está diseñada para recuperar datos de **Renta**

Interfaz JpaRepository:

- *RentaRepository* hereda métodos de la interfaz *JpaRepository*.
 - **save()**
 - **findById()**
 - **deleteById()**

Realizan operaciones CRUD de Renta.

5) Service:

```
package es.hherramientassoftware.servicios;

import es.hherramientassoftware.model.Renta;
import es.hherramientassoftware.repository.RentaRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class RentaServicio {

    private final RentaRepository rentaR;

    @Autowired
    public RentaServicio(RentaRepository rentaR) {
        this.rentaR = rentaR;
    }

    public List<Renta> cojerTodoslosDatos() {
        return rentaR.findAll();
    }

    //ejecutar la consulta 2
    public List<Object[]> consulta2() {
        return rentaR.consulta2();
    }

    public List<Object[]> consulta3() {
        return rentaR.consulta3();
    }

    public List<Object[]> consulta4() {
        return rentaR.consulta4();
    }

    public List<Object[]> consulta5() {
        return rentaR.consulta5();
    }

    public List<Object[]> consulta6() {
        return rentaR.consulta6();
    }

    public List<Object[]> consulta7() {
        return rentaR.consulta7();
    }
}
```

```

    }

    public List<Object[]> consulta8() {
        return rentaR.consulta8();
    }

    public List<Object[]> consulta9() {
        return rentaR.consulta9();
    }

    public List<Object[]> consulta10() {
        return rentaR.consulta10();
    }
}

```

RentaServicio proporciona una capa intermedia entre los controladores y el repositorio *RentaRepository*, encapsulando la lógica de negocio y proporcionando métodos para realizar consultas específicas sobre los datos de renta en la base de datos. Esto mejora la modularidad y la mantenibilidad del código al tiempo que promueve una separación clara de responsabilidades en la aplicación.

Inyección de dependencias:

- *RentaServicio* utiliza la anotación **@Autowired**, que como ya hemos dicho antes, inyecta una instancia de *RentaRepository* en su constructor, llamándose **inyección de dependencias**. Permitiendo que *RentaServicio* utilice los métodos de *RentaRepository* para interactuar con la base de datos.

Métodos de consulta:

- **cojerTodoslosDatos()** ==> Método unido al método **findAll()** de *RentaRepository* para recuperar TODOS los datos de renta almacenados en la bbdd.
- **consulta2()** a **consulta10()** ==> Métodos que llaman a las consultas de *RentaRepository*, representando una consulta específica

Capa de abstracción:

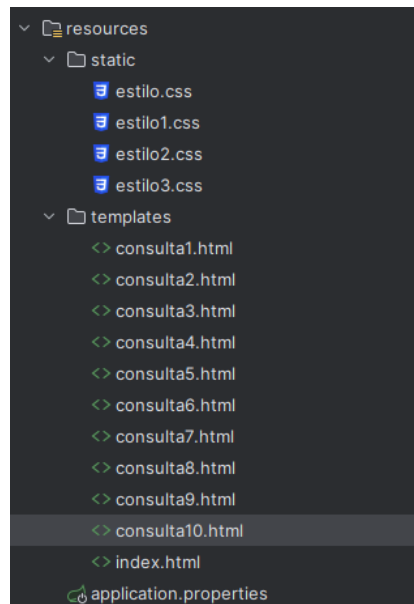
- La lógica de cómo se accede y se manipulan los datos de renta está encapsulada dentro del servicio, lo que hace que el código sea más modular y fácil de mantener.

Promoción de la cohesión:

- Si agrupamos las operaciones de los datos de renta en un solo servicio, promovemos la idea de que las partes relacionadas del código deberían estar juntas y funcionar en conjunto.

FRONT-END

En esta parte de Front-end diseñaremos nuestra aplicación, organizando y mostrando los datos de las consultas realizadas con código HTML y con ayuda de Thymeleaf :



1) Interfaz Principal:

Esta sera la Principal interfaz que se vera al iniciar la aplicación.

Cuenta con un titulo y varias partes con donde con un boton te llevarán a diferentes interfaces para hacer las consultas.

```
<link rel="stylesheet" href="estilo.css">
```

Ruta para conectar con css, aunque nosotros lo conectamos a Bootstrap, código en el que esta todo mas compacto en el propio código HTML, además de ser multiplataforma.

```
<!DOCTYPE html>
<html lang="es">
<!-- -----Cabecera----->

<head >
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Renta por hogar por comunidades autónomas</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
">

  <link rel="stylesheet" href="estilo.css">
</head>
<body>
<!-- -----Sub-Cabecera----->
```

```

<header class="bg-dark text-white py-4 text-center cabeza">
  <div class="container">
    <h1 class="mb-0" style="font-family: 'Lexend', sans-serif;">RENTA
POR HOGAR POR COMUNIDADES AUTÓNOMAS</h1>
  </div>
</header>
<main class="py-5">
  <div class="container">
    <div class="row">
<!-- -----Consulta 1----->

      <div class="col-md-12 mb-4">
        <div class="card">
          <div class="card-body">
            <h2 class="card-title">Consulta 1 - DATOS
TOTALES</h2>
            <p class="card-text">Aquí puedes realizar una
consulta sobre el total de datos disponibles en el INE.</p>
            <a href="/consulta1" class="btn btn-dark">Realizar
Consulta</a>
          </div>
        </div>
      </div>
<!-- -----Consulta 2----->

      <div class="col-md-12 mb-4">
        <div class="card">
          <div class="card-body">
            <h2 class="card-title">Consulta 2 - Promedio de la
renta neta media por hogar por período</h2>
            <p class="card-text">Consulta la tasa de actividad
por género utilizando los datos del INE.</p>
            <a href="/consulta2" class="btn btn-dark">Realizar
Consulta</a>
          </div>
        </div>
      </div>
<!-- -----Consulta 3----->

```

```

        <div class="col-md-12 mb-4">
            <div class="card">
                <div class="card-body">
                    <h2 class="card-title">Consulta 3 - Máxima renta
neta media por hogar por comunidad autónoma:
                    </h2>
                    <p class="card-text">Consulta el total de datos
por titulación.</p>
                    <a href="/consulta3" class="btn btn-dark">Realizar
Consulta</a>
                </div>
            </div>
        </div>

<!-- -----Consulta 4----->

        <div class="col-md-12 mb-4">
            <div class="card">
                <div class="card-body">
                    <h2 class="card-title">Consulta 4 - Rentas medias
más altas y más bajas por tipo de renta:</h2>
                    <p class="card-text">Consulta el total de datos
por indicador.</p>
                    <a href="/consulta4" class="btn btn-dark">Realizar
Consulta</a>
                </div>
            </div>
        </div>

<!-- -----Consulta 5----->

        <div class="col-md-12 mb-4">
            <div class="card">
                <div class="card-body">
                    <h2 class="card-title">Consulta 5 - Total de renta
neta media por hogar por año:
                    </h2>
                    <p class="card-text">Consulta el valor promedio
por indicador.</p>
                    <a href="/consulta5" class="btn btn-dark">Realizar
Consulta</a>
                </div>
            </div>
        </div>
    </div>

```

```

<!-- -----Consulta 6----->

        <div class="col-md-12 mb-4">
            <div class="card">
                <div class="card-body">
                    <h2 class="card-title">Consulta 6 - Porcentaje de
variación de la renta neta media por hogar entre años consecutivos:
                    </h2>
                    <p class="card-text">Consulta la distribución de
valores por género.</p>
                    <a href="/consulta6" class="btn btn-dark">Realizar
Consulta</a>
                </div>
            </div>
        </div>

<!-- -----Consulta 7----->

        <div class="col-md-12 mb-4">
            <div class="card">
                <div class="card-body">
                    <h2 class="card-title">Consulta 7 - Rentas medias
por hogar por tipo de renta y por comunidad autónoma:</h2>
                    <p class="card-text">Consulta el total de datos
por indicador y rango de valor.</p>
                    <a href="/consulta7" class="btn btn-dark">Realizar
Consulta</a>
                </div>
            </div>
        </div>

<!-- -----Consulta 8----->

        <div class="col-md-12 mb-4">
            <div class="card">
                <div class="card-body">
                    <h2 class="card-title">Consulta 8 - Comparación de
la renta neta media por hogar entre diferentes comunidades autónomas:</h2>
                    <p class="card-text">Consulta el total de datos
por titulación y género con valor superior a 80.</p>

```



```

                <a href="/consulta8" class="btn btn-dark">Realizar
Consulta</a>

            </div>
        </div>
    </div>
<!-- -----Consulta 9----->

    <div class="col-md-12 mb-4">
        <div class="card">
            <div class="card-body">
                <h2 class="card-title">Consulta 9 - Rentas medias
por hogar por año y tipo de renta:
                </h2>
                <p class="card-text">Consulta el total de datos
por indicador y género.</p>
                <a href="/consulta9" class="btn btn-dark">Realizar
Consulta</a>

            </div>
        </div>
    </div>

<!-- -----Consulta 10----->

    <div class="col-md-12 mb-4">
        <div class="card">
            <div class="card-body">
                <h2 class="card-title">Consulta 10 - Número de
periodos distintos en los que se han recopilado datos:</h2>
                <p class="card-text">Consulta el total de datos
por titulación y rango de valor.</p>
                <a href="/consulta10" class="btn
btn-dark">Realizar Consulta</a>

            </div>
        </div>
    </div>
</div>
</main>

<!-- -----Pie de Web----->

<footer class="bg-dark text-white py-3">
    <div class="container text-center">

```

```

        <p class="mb-0">&copy; 2024 Análisis de Datos del INE. Todos los
derechos reservados.</p>
    </div>
</footer>
<!-- -----JavaScript----->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"
></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"><
/script>
</body>
</html>

```

EXPLICACIÓN ESTRUCTURA DE CONSULTAS:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" href="estilo1.css">
    <title>Consulta 1</title>
</head>
<body>
<h1>Consulta 1 - Resultados</h1>
<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>Comunidades Autonomas</th>
            <th>Tipos de Rentas</th>
            <th>Periodos</th>
            <th>Renta Media Anual</th>
        </tr>
    </thead>
    <tbody>
        <!-- Aquí utilizamos Thymeleaf para iterar sobre la lista de
datos y mostrar cada fila -->
        <tr th:each="renta : ${renta}">

```

```

        <td th:text="{renta.id}"></td>
        <td th:text="{renta.comunidad_autonoma}"></td>
        <td th:text="{renta.tipo_renta}"></td>
        <td th:number="{renta.periodo}"></td>
        <td th:text="{renta.renta_neta_media_hogar}"></td>
    </tr>
</tbody>
</table>
</body></html>

```

Al partir el código en 2 partes, valorando las mas importantes:

1. CABECERA

- <!DOCTYPE html> Declara el tipo de documento HTML. Este fragmento declara que el documento sigue la especificación de HTML5.
- <html lang="en"> → Define el comienzo del documento HTML y especifica el idioma, en este caso, inglés.
- <head> → Contiene meta información sobre el documento HTML, como los metadatos, enlaces a hojas de estilo (CSS), enlaces a scripts, entre otros.
- <meta charset="UTF-8"> → Define la codificación de caracteres del documento como UTF-8, que es una codificación que incluye una amplia gama de caracteres.
- <meta name="viewport" content="width=device-width, initial-scale=1.0">: Establece las propiedades de la ventana de visualización, adaptándola al ancho del dispositivo y estableciendo el nivel de escala inicial.
- <link rel="stylesheet" href="estilo1.css"> → Enlaza la hoja de estilos estilo1.css al documento HTML. Esto significa que las reglas de estilo definidas en ese archivo CSS se aplicarán al contenido HTML.
- <title>Consulta 1</title> → Define el título de la página que aparecerá en la pestaña del navegador.

2. CUERPO

- <table> → Define una tabla para organizar los datos.
 - <thead> → Define el encabezado de la tabla.
 - <tr> → Define una fila en la tabla.
 - <th> → Define celdas de encabezado en la tabla.
 - <tbody> → Define el cuerpo de la tabla.
- Se está utilizando Thymeleaf, motor de plantillas, para iterar sobre la lista de datos y mostrar cada fila en la tabla.

- `<tr th:each="renta : ${renta}">` → Utiliza la directiva `th:each` de **Thymeleaf** para iterar sobre una lista de objetos llamada `renta`. Por cada objeto en la lista, se crea una nueva fila en la tabla.
- `<td th:text="${renta.id}"></td>` → Utiliza la directiva `th:text` de Thymeleaf para mostrar el valor del atributo `id` del objeto `renta` en la celda de la tabla.
- `<td th:text="${renta.comunidad_autonoma}"></td>` → Similar al anterior, pero muestra el valor del atributo `comunidad_autonoma`.
- `<td th:text="${renta.tipo_renta}"></td>` → Muestra el valor del atributo `tipo_renta`.
- `<td th:number="${renta.periodo}"></td>` → Muestra el valor del atributo `periodo`. En este caso, parece que se está utilizando `th:number` para asegurarse de que el valor sea numérico.
- `<td th:text="${renta.renta_neta_media_hogar}"></td>` → Muestra el valor del atributo `renta_neta_media_hogar`.

COMPROBACIÓN DE HERRAMIENTA SOFTWARE

PÁGINA PRINCIPAL

Esto es lo primero que se verá, la página principal con 10 consultas con su título y botón que llevan a la interfaz de consulta, listas para mostrar los resultados esperados:

RENTA POR HOGAR POR COMUNIDADES AUTÓNOMAS

Consulta 1 - DATOS TOTALES

Aquí puedes realizar una consulta sobre el total de los datos del INE.

Realizar Consulta

Consulta 2 - Promedio de la renta neta media por hogar por período

Realizar Consulta

Consulta 3 - Máxima renta neta media por hogar por comunidad autónoma:

Realizar Consulta

Consulta 4 - Rentas medias más altas y más bajas por tipo de renta:

Realizar Consulta

Consulta 5 - Total de renta neta media por hogar por año:

Realizar Consulta

Entre años consecutivos:

Realizar Consulta

Consulta 7 - Rentas medias por hogar por tipo de renta y por comunidad autónoma:

Realizar Consulta

Consulta 8 - Comparación de la renta neta media por hogar entre diferentes comunidades autónomas:

Realizar Consulta

Consulta 9 - Rentas medias por hogar por año y tipo de renta:

Realizar Consulta

Consulta 10 - Número de periodos distintos en los que se han recopilado datos:

Realizar Consulta

Consulta 1

Consulta que muestra todos los resultados de la bbdd.

← ↻ ⓘ localhost:8080/consulta1 🔍 ☆ 📄 🗨 📄 📄 ⋮

Rentas por Comunidad Autónoma				
ID	Comunidades Autónomas	Tipos de Rentas	Periodos	Renta Media Anual
1	Total Nacional	Renta neta media por hogar	2022	32216
2	Total Nacional	Renta neta media por hogar	2021	30552
3	Total Nacional	Renta neta media por hogar	2020	30690
4	Total Nacional	Renta neta media por hogar	2019	29132
5	Total Nacional	Renta neta media por hogar	2018	28417
6	Total Nacional	Renta neta media por hogar	2017	27558
7	Total Nacional	Renta neta media por hogar	2016	26730
8	Total Nacional	Renta neta media por hogar	2015	26092
9	Total Nacional	Renta neta media por hogar	2014	26154
10	Total Nacional	Renta neta media por hogar	2013	26775
11	Total Nacional	Renta neta media por hogar	2012	27747
12	Total Nacional	Renta neta media por hogar	2011	28206
13	Total Nacional	Renta neta media por hogar	2010	29634
14	Total Nacional	Renta neta media por hogar	2009	30045
15	Total Nacional	Renta neta media por hogar	2008	28787

Consulta 2

Promedio de la renta neta media por hogar por periodo:

Promedio de Rentas

El promedio de rentas es: 30990.725

Consulta 3

Maxima renta neta media por comunidad Autonoma:

← ↻ ⓘ localhost:8080/consulta3 A ☆ □ ☆ □

Rentas por Comunidad Autónoma

Comunidad Autónoma	Renta Máxima
02 Aragón	38109
03 Asturias, Principado de	35225
11 Extremadura	29030
14 Murcia, Región de	33747
04 Balears, Illes	38258
18 Ceuta	41951
06 Cantabria	37617
12 Galicia	34589
10 Comunitat Valenciana	34178
08 Castilla - La Mancha	32617
Total Nacional	37363
13 Madrid, Comunidad de	45525
19 Melilla	46566
05 Canarias	32131
09 Cataluña	41206
01 Andalucía	32687

Consulta 4

Medias más altas y bajas de rentas, por tipo de renta.

Consulta de cantidad de hogares por comunidad autónoma

Comunidad Autónoma	Cantidad de Hogares
02 Aragón	30
03 Asturias, Principado de	30
11 Extremadura	30
14 Murcia, Región de	30
04 Balears, Illes	30
18 Ceuta	30
06 Cantabria	30
12 Galicia	30
10 Comunitat Valenciana	30
08 Castilla - La Mancha	30
Total Nacional	30
13 Madrid, Comunidad de	30
19 Melilla	30
05 Canarias	30
09 Cataluña	30

Consulta 5

Total de renta neta media por hogar por año:

Consulta de Rentas

Tipo de Renta	Renta Máxima	Renta Mínima
Renta media por hogar (con alquiler imputado)	46566	23056
Renta neta media por hogar	41714	19364

Consulta 6

Renta media por año:

Año	Suma de Rentas
2013	1161335
2014	1147011
2016	1176509
2015	1137737
2011	1231947
2008	1232645
2010	1277305
2019	1266071
2012	1205817
2022	1394380
2017	1201749
2018	1242851
2009	1285884
2020	1318764
2021	1314430

Comunidad Autónoma	Tipo de Renta	Promedio de Renta
05 Canarias	Renta neta media por hogar	23796.266666666666
17 Rioja, La	Renta neta media por hogar	28369.2
13 Madrid, Comunidad de	Renta media por hogar (con alquiler imputado)	39218.066666666666
Total Nacional	Renta media por hogar (con alquiler imputado)	32898.333333333336
01 Andalucía	Renta neta media por hogar	24119.533333333333
Total Nacional	Renta neta media por hogar	28582.333333333332
15 Navarra, Comunidad Foral de	Renta neta media por hogar	35602.533333333333
10 Comunitat Valenciana	Renta media por hogar (con alquiler imputado)	29681.333333333332
10 Comunitat Valenciana	Renta neta media por hogar	25436.266666666666
03 Asturias, Principado de	Renta media por hogar (con alquiler imputado)	32228.266666666666
14 Murcia, Región de	Renta media por hogar (con alquiler imputado)	28792.933333333334
06 Cantabria	Renta media por hogar (con alquiler imputado)	32150.466666666667
09 Cataluña	Renta neta media por hogar	32500.133333333333
04 Balears, Illes	Renta neta media por hogar	30023.866666666667
14 Murcia, Región de	Renta neta media por hogar	24722.6
12 Galicia	Renta media por hogar (con alquiler imputado)	31307.0

Comparación de la renta neta media por hogar de las comunidades autónomas:

localhost:8080/consulta8

Comparación de la renta neta media por hogar entre diferentes comunidades autónomas

Comunidad Autónoma	Promedio de Renta
18 Ceuta	33743.333333333336
19 Melilla	38643.3
17 Rioja, La	30411.1

Consulta 9

Rentas medias por hogar, con su tipo:

localhost:8080/consulta9

Rentas medias por hogar por año y tipo de renta

Año	Tipo de Renta	Promedio de Renta
2019	Renta neta media por hogar	29339.4
2021	Renta neta media por hogar	30439.65
2017	Renta media por hogar (con alquiler imputado)	32233.9
2014	Renta media por hogar (con alquiler imputado)	30725.35
2016	Renta media por hogar (con alquiler imputado)	31480.65
2013	Renta media por hogar (con alquiler imputado)	30910.95
2017	Renta neta media por hogar	27853.55
2008	Renta media por hogar (con alquiler imputado)	32756.3
2015	Renta media por hogar (con alquiler imputado)	30425.8
2012	Renta media por hogar (con alquiler imputado)	32096.5
2019	Renta media por hogar (con alquiler imputado)	33964.15
2014	Renta neta media por hogar	26625.2
2011	Renta media por hogar (con alquiler imputado)	32871.25
2009	Renta neta media por hogar	30214.6
2011	Renta neta media por hogar	28726.1
2012	Renta neta media por hogar	28194.35
2010	Renta neta media por hogar	29802.7

Consulta 10

Número de periodos:

localhost:8080/consulta10

Número de periodos distintos

Cantidad de Periodos
15

REVISIÓN DE RESULTADOS CON OTRO LENGUAJE

Si probamos a utilizar Pandas, de Python para ejecutar una de las Consultas, por ejemplo la 3, ¿Se ejecutaría?

Pandas Python.

Consulta 3:

```
consulta_sql = """
    SELECT comunidad_autonoma, MAX(renta_neta_media_hogar) AS max_renta
    FROM renta_hogares
    GROUP BY comunidad_autonoma;
"""

dataframe = pd.read_sql_query(consulta_sql, conn)

conn.close()

print(dataframe)
```

	comunidad_autonoma	max_renta
0	02 Aragón	38109
1	03 Asturias, Principado de	35225
2	11 Extremadura	29030
3	14 Murcia, Región de	33747
4	04 Balears, Illes	38258
5	18 Ceuta	41951
6	06 Cantabria	37617
7	12 Galicia	34589
8	10 Comunitat Valenciana	34178
9	08 Castilla - La Mancha	32617
10	Total Nacional	37363
11	13 Madrid, Comunidad de	45525
12	19 Melilla	46566
13	05 Canarias	32131
14	09 Cataluña	41206
15	01 Andalucía	32687
16	16 País Vasco	44538
17	15 Navarra, Comunidad Foral de	44834
18	17 Rioja, La	36853
19	07 Castilla y León	35029

POSIBLES MEJORAS

- 1. Optimización del rendimiento de las consultas:** Si pudiera optimizar las consultas SQL, mejoraría el rendimiento de la aplicación, haciendo que sea más rápida al cargar y al dar una respuesta. Al utilizar índices en las columnas de búsqueda, evitaría consultas redundantes y ordenaría a la hora de mostrar grandes conjuntos de datos.
- 2. Mejorar la interfaz de usuario y la Usabilidad:** Si mejoró la interfaz de usuario con un diseño más atractivo, se haría más visible para otras personas, creando un diseño responsive y moderno. Además de simplificar la navegación, y proporcionar retroalimentación sobre las acciones del usuario.
- 3. Funcionalidad de búsqueda:** Agregar una barra de búsqueda para que los usuarios puedan encontrar más rápidamente la información que necesitan dentro de los conjuntos de datos.
- 4. Filtrado de datos:** Permite a los usuarios filtrar las consultas según sus necesidades, como fechas, categorías o valores específicos.
- 5. Gestión de sesiones de usuario:** Implementar un sistema de gestión de usuarios para autenticar y autorizarlos, además de mantener su estado de inicio de sesión abierto.
- 6. Documentación de ayuda:** Proporcionar documentación clara dentro de la aplicación para guiar a los usuarios sobre cómo utilizarla.
- 7. Añadir idiomas:** Agregar soporte para múltiples idiomas y formatos de fecha y hora locales para los usuarios de diferentes regiones.
- 8. Retroalimentación de los usuarios:** Solicitar comentarios de los usuarios para identificar problemas o errores de diseño, áreas de mejora o nuevas características que podrían mejorar la aplicación.
- 9. Monitorización y registro de errores:** Configurar herramientas de monitorización y registro de errores para supervisar el rendimiento de la aplicación y detectar problemas o anomalías.
- 10. Pruebas automatizadas:** Implementar pruebas automatizadas para verificar la funcionalidad y la estabilidad de la aplicación, incluyendo pruebas unitarias e integradas.
- 11. Realizar actualizaciones y un mantenimiento continuo:** Debería actualizar la aplicación con sus últimas versiones de las bibliotecas, frameworks y dependencias. Realizar parches de seguridad y actualizaciones de software continuas, para proteger la aplicación contra vulnerabilidades y mejorar su funcionalidad y seguridad en general.

CONCLUSIÓN

En esencia, este proyecto es como una semilla plantada en el vasto jardín de la tecnología, una idea que ha germinado y ahora florece con cada línea de código que escribo, representa más que solo líneas de código, información y datos; es una ventana hacia el entendimiento y la exploración de datos que impactan la vida de las personas. Al ofrecer una plataforma para analizar la renta por hogar, toca temas fundamentales de bienestar y progreso en nuestras comunidades autónomas.

Cada consulta realizada es una oportunidad para descubrir historias detrás de las cifras, para comprender mejor las dinámicas socioeconómicas que moldean nuestras vidas y entornos. La aplicación no solo proporciona datos, sino que invita a reflexionar, a comprender, y a buscar soluciones basadas en información concreta.

Detrás de cada consulta, hay un esfuerzo por democratizar el acceso a la información, por hacer que los datos sean comprensibles y accesibles para todos. Es un puente entre la complejidad de los datos y la comprensión humana, una herramienta que busca iluminar y guiar en la toma de decisiones informadas.

Es un testimonio de tu creatividad y determinación para crear algo significativo. Aunque mi aplicación actualmente brilla con la promesa de lo que podría ser, aún no ha alcanzado su máximo potencial. Cada mejora que implemente será como un rayo de sol que ilumina el camino hacia un futuro brillante y emocionante.

El proyecto actualmente es una aplicación que extrae datos de una tabla de BBDD y los presenta de forma organizada. Aunque funciona bien, hay espacio para mejorar. Pero en general, es un buen comienzo.