



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



## TFG del Grado en Ingeniería Informática

### Aplicación Web para la recopilación, tratamiento y visualización de datos públicos



Presentado por Iván Arjona Alonso  
en Universidad de Burgos — 31 de mayo  
de 2018

Tutor: Dr. José Francisco Díez Pastor  
y Dr. Jesús Manuel Maudes Raedo



---

# Índice general

---

<b>Índice general</b>	<b>I</b>
<b>Índice de figuras</b>	<b>III</b>
<b>Índice de tablas</b>	<b>v</b>
<b>Apéndice A Plan de Proyecto Software</b>	<b>1</b>
A.1. Introducción . . . . .	1
A.2. Planificación temporal . . . . .	1
A.3. Estudio de viabilidad . . . . .	11
<b>Apéndice B Especificación de Requisitos</b>	<b>15</b>
B.1. Introducción . . . . .	15
B.2. Objetivos generales . . . . .	15
B.3. Catalogo de requisitos . . . . .	16
B.4. Especificación de requisitos . . . . .	17
<b>Apéndice C Especificación de diseño</b>	<b>25</b>
C.1. Introducción . . . . .	25
C.2. Diseño de datos . . . . .	25
C.3. Diseño procedimental . . . . .	26
C.4. Diseño arquitectónico . . . . .	27
<b>Apéndice D Documentación técnica de programación</b>	<b>31</b>
D.1. Introducción . . . . .	31
D.2. Estructura de directorios . . . . .	31
D.3. Manual del programador . . . . .	33

D.4. Instalación y ejecución del proyecto . . . . .	36
D.5. Despliegue . . . . .	37
D.6. Pruebas del sistema . . . . .	38
<b>Apéndice E Documentación de usuario</b> . . . . .	<b>41</b>
E.1. Introducción . . . . .	41
E.2. Requisitos de usuarios . . . . .	41
E.3. Instalación . . . . .	42
E.4. Manual del usuario . . . . .	42

---

# Índice de figuras

---

A.1. Burndown del sprint 0 . . . . .	2
A.2. Burndown del sprint 1 . . . . .	3
A.3. Burndown del sprint 2 . . . . .	4
A.4. Burndown del sprint 3 . . . . .	5
A.5. Burndown del sprint 4 . . . . .	6
A.6. Burndown del sprint 5 . . . . .	6
A.7. Burndown del sprint 6 . . . . .	7
A.8. Burndown del sprint 7 . . . . .	8
A.9. Burndown del sprint 8 . . . . .	8
A.10. Burndown del sprint 9 . . . . .	9
A.11. Burndown del sprint 10 . . . . .	10
 B.1. Diagrama de casos de uso . . . . .	23
 C.1. Diagrama de secuencia para actualizar fuentes de datos . . . . .	27
C.2. Diagrama de clases del paquete Fuentes . . . . .	29
C.3. Diagrama Modelo–Vista–Controlador . . . . .	30
 E.1. Página de consulta . . . . .	43
E.2. Filtros de la consulta . . . . .	44
E.3. Seleccionar columnas a mostrar . . . . .	45
E.4. Visualización de la consulta . . . . .	46
E.5. Límite de filas superado . . . . .	46
E.6. Exportar consulta . . . . .	47
E.7. Botón para añadir una subconsulta . . . . .	47
E.8. Pestañas con varias subconsultas . . . . .	47
E.9. Método para juntar subconsultas . . . . .	48
E.10. Autocompletar en columnas calculadas . . . . .	49

E.11. Visualización de una consulta con una columna calculada . . . . .	49
E.12. Columna calculada no válida . . . . .	50
E.13. Desplegables para visualizar el mapa . . . . .	50
E.14. Desplegable para seleccionar la columna a visualizar . . . . .	51
E.15. Mapa por provincias de la columna calculada . . . . .	51
E.16. Mapa por municipios de la columna calculada . . . . .	52
E.17. Error al intentar visualizar un mapa sin haber seleccionado la clave correspondiente . . . . .	52
E.18. Aplicación de escritorio para juntar ficheros CSV . . . . .	53
E.19. Selección de columnas al juntar CSV . . . . .	54

---

# Índice de tablas

---

A.1. Costes de personal . . . . .	11
A.2. Costes de hardware . . . . .	12
A.3. Costes de software . . . . .	12
A.4. Coste total . . . . .	13
A.5. Dependencias . . . . .	14
B.1. CU-1 Carga de datos. . . . .	18
B.2. CU-2 Consulta de datos. . . . .	19
B.3. CU-3 Visualización de datos. . . . .	20
B.4. CU-4 Exportar consulta. . . . .	21
B.5. CU-5 Juntar ficheros CSV. . . . .	22



## *Apéndice A*

---

# **Plan de Proyecto Software**

---

## **A.1. Introducción**

En esta sección se trata sobre la planificación del proyecto. Se ha realizado un estudio para estimar el tiempo, la viabilidad económica y las consecuencias legales que puedan conllevar la realización de este proyecto. Los resultados de esta fase conllevarán en la decisión de si se va a llevar a cabo el proyecto o no.

Para la organización del proyecto se ha utilizado [GitHub<sup>1</sup>](#) junto con la extensión de [ZenHub<sup>2</sup>](#) para asignar prioridades a las tareas y realizar gráficos del trabajo pendiente de cada iteración.

## **A.2. Planificación temporal**

El desarrollo del proyecto se ha llevado a cabo utilizando metodologías ágiles, basándose en la metodología *scrum* con algunas modificaciones (una sola persona y sin reuniones diarias).

Se aplicó una estrategia de desarrollo incremental, con iteraciones que llamaremos *sprints*.

El resultado de cada iteración es un entregable, sobre el que se discute en la reunión posterior a cada sprint.

---

<sup>1</sup>GitHub: <https://github.com/IvanArjona/TFG-Datos-publicos>

<sup>2</sup>ZenHub: <https://www.zenhub.com/>

Se realizó, en principio, una reunión a la semana con los tutores para exponer las modificaciones realizadas en el sprint anterior y planificar los cambios a realizar en la siguiente iteración.

Estas tareas están priorizadas por el tiempo estimado de su realización, se puede ver esta estimación en el enlace de cada sprint a sus tareas.

A continuación se va a realizar un breve resumen de las tareas realizadas en cada una de las iteraciones, así como la duración de cada sprint y el gráfico *burndown* correspondiente.

## Sprint 0 (16/02/2018 - 02/03/2018)

Primer sprint del proyecto. En la la reunión de planificación de este primer sprint se discute de forma general de lo que va a tratar el proyecto.

Las tareas realizadas durante este sprint fueron la creación y configuración del repositorio y sobre todo investigar sobre las tecnologías y herramientas que se podían utilizar.

Se investigaron posibles fuentes de datos para implementar más adelante: INE, SEPE, AEAT.

La duración fue de dos semanas en lugar de una para poder documentarse sobre todos los aspectos relevantes del proyecto.

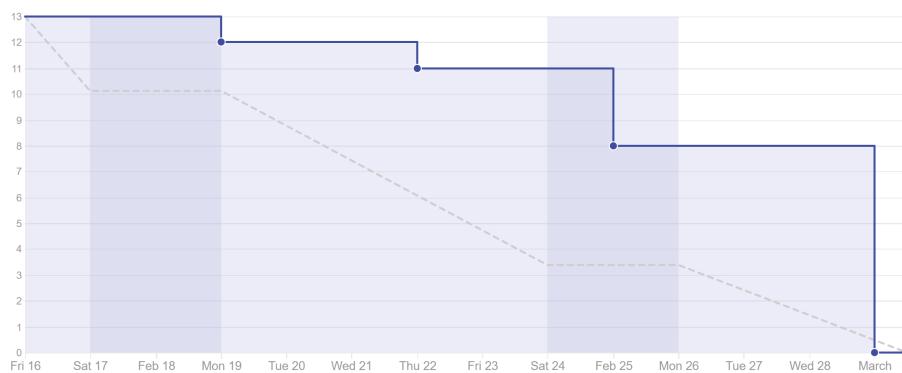


Figura A.1: Burndown del sprint 0

### Tareas del sprint 0 en Github<sup>3</sup>

---

<sup>3</sup>Tareas del sprint 0 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/1?closed=1>

## Sprint 1 (03/03/2018 - 08/03/2018)

Un objetivo de este sprint es investigar alternativas de bases de datos no relacionadas que se podrían utilizar. Se ha elegido MongoDB.

El otro objetivo es empezar a implementar prototipos con las fuentes de datos que se habían encontrado en el sprint anterior. Se implementaron prototipos del INE, de la agencia tributaria y del SEPE.

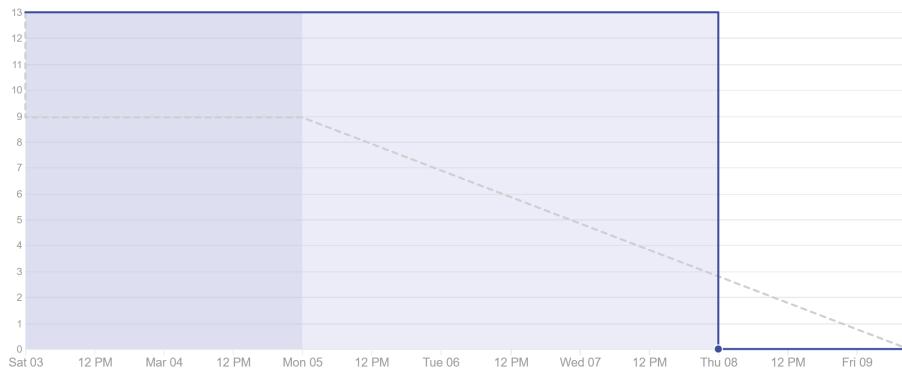


Figura A.2: Burndown del sprint 1

Tareas del sprint 1 en Github<sup>4</sup>

## Sprint 2 (09/03/2018 - 15/03/2018)

El primer objetivo de este sprint es crear la estructura de la página web con Flask. Utilizando un modelo vista-controlador. También se utiliza Bootstrap para ahorrar trabajo en el diseño.

Se implementó un prototipo de la carga de datos hacia la base de datos y otro para la descarga de datos desde la base de datos para ser mostrados.

Se hizo una implementación de las fuentes de datos a partir de los prototipos del sprint 1 de modo que se pueda cargar todas las fuentes de manera automática.

<sup>4</sup>Tareas del sprint 1 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/2?closed=1>



Figura A.3: Burndown del sprint 2

Tareas del sprint 2 en Github<sup>5</sup>

### Sprint 3 (16/03/2018 - 22/03/2018)

En este sprint se sopesaron varias plataformas para hacer el despliegue de la web. De ellas se eligió DigitalOcean y Nanobox.

Se realizó el despliegue utilizando estas plataformas. **Web desplegada**.

Se investigaron los posibles riesgos de seguridad como inyecciones NoSQL.

Se implementó un formulario para la consulta de datos en la página web. En esta primera aproximación se podía hacer una consulta comparando con una columna de una de las fuentes de datos.

---

<sup>5</sup>Tareas del sprint 2 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/3?closed=1>

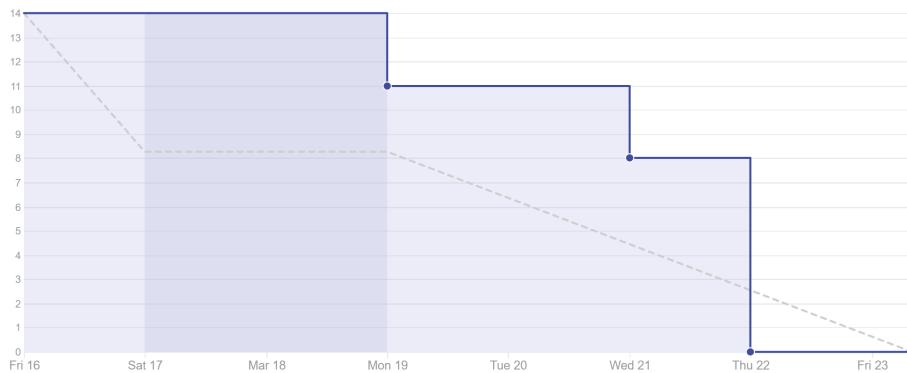


Figura A.4: Burndown del sprint 3

Tareas del sprint 3 en Github<sup>6</sup>

## Sprint 4 (23/03/2018 - 13/04/2018)

Este sprint coincide con Semana Santa, por lo que dura una semana más de lo habitual y la carga de trabajo también es mayor.

Se corrigieron errores en los tipos de las fuentes de datos al tratar con números como cadenas.

Se implementó una forma de descargar las consultas a partir del formulario.

Se mejoró interfaz gráfica y el formulario de consulta.

Se modificaron las fuentes de datos para corregir errores y añadir el código de municipio a todas ellas para más tarde poder unirlas.

<sup>6</sup>Tareas del sprint 3 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/4?closed=1>



Figura A.5: Burndown del sprint 4

Tareas del sprint 4 en Github<sup>7</sup>

### Sprint 5 (14/04/2018 - 25/04/2018)

Este sprint se dedicó a empezar a documentar la memoria y corregir algunos errores en la página de consulta como el reenvío de formularios en firefox y la descarga de consultas en JSON y CSV.

También se añadió una descripción a la fuente de datos para explicar de qué se trata cada una en la interfaz web.



Figura A.6: Burndown del sprint 5

<sup>7</sup>Tareas del sprint 4 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/5?closed=1>

### Tareas del sprint 5 en Github<sup>8</sup>

#### Sprint 6 (26/04/2018 - 02/05/2018)

El objetivo principal de este sprint fue implementar la posibilidad de juntar varias subconsultas mediante un *join*. Otra característica implementada es la de avisar al usuario si se ha sobrepasado el límite de columnas especificado, para que pueda filtrar más fino.

También se consideró hacer cambios en el modelo de datos, pero debido a la alta dimensionalidad se ha dejado como estaba en el sprint anterior.



Figura A.7: Burndown del sprint 6

### Tareas del sprint 6 en Github<sup>9</sup>

#### Sprint 7 (03/05/2018 - 09/05/2018)

En este sprint se implementó un mapa coroplético para mostrar los valores de cualquier atributo en el mapa agrupando los municipios por su provincia.

Se eliminaron los campos duplicados de las consultas que surgían al realizar *join* de varias subconsultas. Como estos campos repetidos siempre son iguales, se ha optado por eliminarlos en lugar de renombrarlos.

---

<sup>8</sup>Tareas del sprint 5 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/6?closed=1>

<sup>9</sup>Tareas del sprint 6 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/7?closed=1>

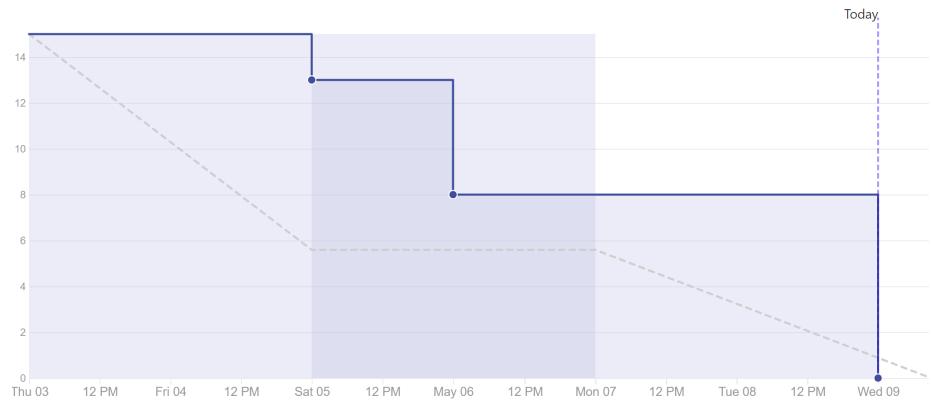


Figura A.8: Burndown del sprint 7

Tareas del sprint 7 en Github<sup>10</sup>

### Sprint 8 (10/05/2018 - 16/05/2018)

En este sprint se implementó una aplicación para juntar varios ficheros CSV en uno sólo utilizando *join*. Para poder aplicar técnicas de minería de datos sobre estos ficheros.

Se añadió una funcionalidad de poder mostrar mapas coropléticos a nivel de municipio, además de a nivel de provincia.



Figura A.9: Burndown del sprint 8

<sup>10</sup>Tareas del sprint 7 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/8?closed=1>

### Tareas del sprint 8 en Github<sup>11</sup>

#### Sprint 9 (17/05/2018 - 23/05/2018)

En este sprint se añadió un campo para introducir columnas calculadas a partir de los campos de una consulta.

Se añadió otra fuente de datos, resultados electorales del congreso, utilizando los datos del ministerio del interior.

Se añadió un campo para seleccionar la columna por la que se quieren juntar varios CSV a partir de las columnas comunes.

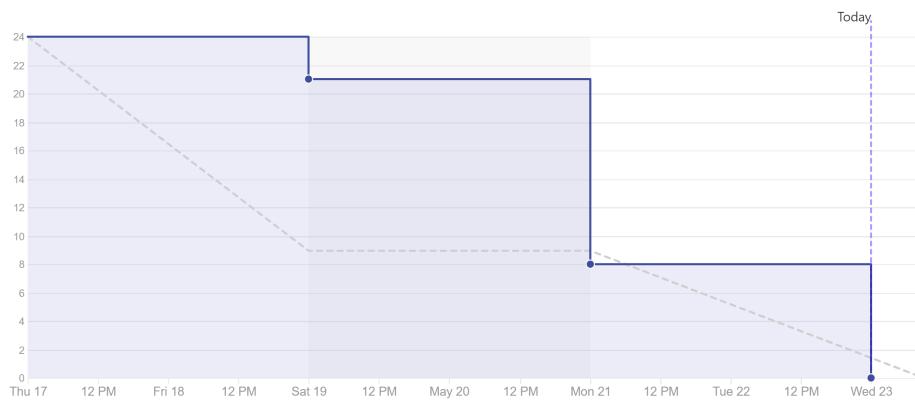


Figura A.10: Burndown del sprint 9

### Tareas del sprint 9 en Github<sup>12</sup>

#### Sprint 10 (24/05/2018 - 30/05/2018)

Este es el último sprint en el que se añade funcionalidad.

Se añadió un desplegable para seleccionar las columnas al escribir una consulta calculada.

En el programa para juntar ficheros CSV, se cambió el campo para seleccionar la columna por la que se van a juntar por un campo de selección multiple para permitir juntarlos por varias columnas.

<sup>11</sup>Tareas del sprint 8 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/9?closed=1>

<sup>12</sup>Tareas del sprint 9 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/10?closed=1>

Se realizaron test unitarios con unittest y de interfaz con Selenium.

Se modificó la fuente de datos del Ministerio del Interior para agrupar los partidos políticos según su posición.



Figura A.11: Burndown del sprint 10

Tareas del sprint 10 en Github<sup>13</sup>

---

<sup>13</sup>Tareas del sprint 10 en Github: <https://github.com/IvanArjona/TFG-Datos-publicos/milestone/11?closed=1>

Concepto	Coste (€)
Salario neto	1000 €
Retención IRPF (19 %)	360,53 €
Seguridad social (28,30 %)	537,00 €
Salario bruto (mensual)	1897,53 €
Total 4 meses	7592,59 €

Tabla A.1: Costes de personal

## A.3. Estudio de viabilidad

### Viabilidad económica

#### Costes de personal

El proyecto se ha llevado a cabo por un desarrollador empleado a tiempo parcial durante 4 meses. Se considera un salario neto de 1000 €. (Ver tabla A.1)

Los porcentajes de cotización a la seguridad social se han calculado a partir del régimen general para 2018 como horas comunes (23,6 % empresa y 4,7 % trabajadores) [?].

Además se sumará el sueldo de los dos tutores asignados al proyecto durante 4 meses, que corresponde a 0,5 créditos [?] [?].

- Ayudante de doctor. Sueldo mensual: 1.815,61 €. Imparte 24 créditos anuales.

$$\frac{1.815,61 \text{ €} * 12 \text{ meses} * 0,5 \text{ créditos}}{24 \text{ créditos}} = 453,90 \text{ €}$$

- Profesor titular de universidad. Sueldo mensual: 3.527,89 €. Imparte 24 créditos anuales.

$$\frac{3.527,89 \text{ €} * 12 \text{ meses} * 0,5 \text{ créditos}}{24 \text{ créditos}} = 881,97 \text{ €}$$

Corresponde un total de 7592,59 € de personal y 1335,87 € de los tutores.

### Costes de material

En esta sección se incluyen los costes de software y hardware.

Como hardware se incluye el coste del servidor web (5€ mensual) y el equipo que se ha utilizado para desarrollar la aplicación (un único pago de 1000€) con amortización a 5 años. (Ver tabla A.2)

$$\frac{1.000 \text{ €} * 4 \text{ meses}}{5 \text{ años} * 12 \text{ meses}} = 66,67 \text{ €}$$

Concepto	Coste (€)	Amortización
Ordenador portátil	1000 €	66,67 €
Servidor web (mensual)	5 €	
Total 4 meses		86,67 €

Tabla A.2: Costes de hardware

Como software incluimos la licencia de Windows como único pago y las licencias de Github developer, Codacy Pro y PyCharm como pago mensual. Gitkraken Pro supone un coste de 41€ anual, como no hay pago mensual, se considerará el pago completo. (Ver tabla A.3)

Concepto	Coste (€)
Windows 10 Home	145 €
GitHub Developer (mensual)	7 €
Codacy Pro Plan (mensual)	15 €
PyCharm (mensual)	8,90 €
GitKraken Pro (anual)	41 €
Total 4 meses	309,60 €

Tabla A.3: Costes de software

### Costes totales

Sumando los costes de los apartados anteriores tendremos un coste total de 9957.4€. (Ver tabla A.4)

Concepto	Coste (€)
Personal	7592,59 €
Tutores	1335,87 €
Hardware	86,67 €
Software	309,60 €
Total	9324,73 €

Tabla A.4: Coste total

## Beneficios

Si se quisiera rentabilizar el proyecto se pueden considerar varias alternativas.

- Inclusión de publicidad.
- Modelo *freemium* [?]: incluir algunas características más avanzadas de pago.
- Limitar la cantidad de datos a mostrar poniendo una barrera económica.

## Viabilidad legal

Con la ayuda de VersionEye [?] se han listado las dependencias del proyecto, con sus correspondientes licencias (Ver tabla A.5).

De las dependencias usadas en el proyecto, la licencia más restrictiva es LGPL-3.0 [?]. Esta licencia permite el uso de la librería con total libertad, por lo que no nos restringe en la licencia que tenemos que utilizar.

Se ha decidido utilizar la licencia GNU-3.0 [?]. Con esta licencia se permite el uso, distribución y modificación del proyecto, siempre que se mantenga la misma licencia y se acredite al autor original.

Dependencia	Versión	Licencia
beautifulsoup4	4.6.0	MIT
branca	0.2.0	MIT
chardet	3.0.4	MIT
click	6.7	BSD
dominate	2.3.1	LGPL-3.0
Flask	0.12.4	BSD
Flask-Bootstrap	3.3.7.1	BSD
Flask-PyMongo	0.5.1	BSD
Flask-WTF	0.14.2	BSD
folium	0.5	MIT
itsdangerous	0.24	BSD
Jinja2	2.10	BSD
MarkupSafe	1.0	BSD
numpy	1.14.3	BSD
pandas	0.22.0	BSD
pymongo	3.6.1	Apache-2.0
requests	2.18.4	Apache-2.0
six	1.11.0	MIT
urllib3	1.22	MIT
WTForms	2.1	BSD
xldr	1.1.0	BSD
Bootstrap	3.3.7	MIT
jQuery	1.12.4	MIT
Dynatable	0.3.1	AGPL

Tabla A.5: Dependencias

## *Apéndice B*

---

# **Especificación de Requisitos**

---

## **B.1. Introducción**

En este apéndice se describirán los objetivos de la aplicación y se detallarán tanto los requisitos funcionales como los no funcionales.

## **B.2. Objetivos generales**

- Integrar varias fuentes de datos públicas en una única base de datos. Estos datos son datos de carácter sociológicos, económicos y demográficos a nivel municipal en España.
- Permitir añadir nuevos conjuntos de datos de forma sencilla.
- Desarrollar un algoritmo para la carga de varias fuentes de datos en una base de datos de manera automatizada, de forma que el administrador pueda añadir nuevos conjuntos de datos de forma sencilla.
- Desarrollo de una aplicación web que permita la consulta de los datos de manera sencilla y visual.
- Facilitar la interpretación de los datos utilizando mapas coropléticos interactivos.
- Desplegar la aplicación web en un servidor de forma que sea fácil de actualizar cada vez que se realice un cambio. Además de funcionar en un entorno local.

## B.3. Catalogo de requisitos

### Requisitos funcionales

- **RF-1 Cargar datos:** El administrador de datos debe poder cargar y actualizar los datos desde sus respectivas fuentes de forma automatizada.
- **RF-2 Consulta:** Los usuarios deben poder consultar información de las fuentes de datos.
  - **RF-2.1:** Podrán realizarse varias subconsultas al mismo tiempo.
  - **RF-2.2:** Se podrá seleccionar las columnas resultantes a mostrar.
  - **RF-2.3:** Se podrá filtrar según los campos de una de las fuentes de datos.
- **RF-3 Visualizar datos:** Los usuarios deben poder visualizar los datos de una consulta en forma de mapa coroplético.
  - **RF-3.1:** Los datos podrán elegirse de cualquier columna numérica.
  - **RF-3.2:** Se podrán mostrar los datos en un mapa a nivel municipal o provincial.
  - **RF-3.3:** Los datos de una columna se agregarán utilizando varios métodos (media, suma y cuenta).
- **RF-4 Exportar:** Los usuarios deben poder exportar datos en varios formatos.
- **RF-5 Juntar CSV:** Los usuarios deben poder juntar varios archivos CSV.
  - **RF-5.1:** Juntar los archivos mediante una columna común.
  - **RF-5.2:** Utilizando varios tipos de join (*inner, outer, left, right*).
  - **RF-5.3:** Seleccionar la ruta donde se exporta el resultado.

### Requisitos no funcionales

- **RNF-1 Usabilidad:** La aplicación debe ser fácil de usar e intuitiva para el usuario.

- **RNF-2 Rendimiento:** La aplicación debe cargar en un tiempo aceptable.
- **RNF-3 Mantenimibildad:** La aplicación debe permitir añadir características de forma sencilla.
- **RNF-4 Compatibilidad:** La aplicación debe funcionar correctamente en los navegadores modernos más utilizados (Edge, Chrome, Firefox, Opera y Safari).
- **RNF-5 Responsividad:** La aplicación debe funcionar en pantallas de cualquier tamaño y adaptar su interfaz a cada pantalla.
- **RNF-6 Escalabilidad:** La aplicación debe poder aumentar su rendimiento al aumentar recursos hardware.
- **RNF-7 Facilidad de despliegue:** La aplicación debe poder desplegarse en un servidor de forma sencilla.
- **RNF-8 Software libre:** Utilizar software libre siempre que sea posible.

## B.4. Especificación de requisitos

En esta sección se desarrollan los casos de uso relacionados con los requisitos funcionales del apartado anterior, se enumeran los actores que interactúan con la aplicación y se incluye el diagrama de casos de uso.

### Descripción de casos de uso

A continuación se desarrolla la tabla de cada uno de los casos de uso.

<b>CU-1</b>	<b>Carga de datos</b>
<b>Versión</b>	1.0
<b>Autor</b>	Iván Arjona Alonso
<b>Requisitos asociados</b>	RF-1
<b>Descripción</b>	Carga el contenido de las fuentes de datos a la base de datos de la aplicación.
<b>Precondición</b>	Se ha iniciado la base de datos
<b>Acciones</b>	<p>1. El administrador de datos ejecuta el script para cargar las fuentes de datos.</p>
<b>Postcondición</b>	La información de las fuentes de datos se cargan en la base de datos.
<b>Excepciones</b>	<ul style="list-style-type: none"> <li>■ Alguna fuente no está disponible.</li> <li>■ La estructura de los datos de alguna fuente ha cambiado.</li> </ul>
<b>Importancia</b>	Alta

Tabla B.1: CU-1 Carga de datos.

CU-2	Consulta de datos
<b>Versión</b>	1.0
<b>Autor</b>	Iván Arjona Alonso
<b>Requisitos asociados</b>	RF-2, RF-2.1, RF-2.2, RF-2.3
<b>Descripción</b>	Permite al usuario consultar una o varias fuentes de datos.
<b>Precondición</b>	Los datos están cargados en la base de datos.
<b>Acciones</b>	<ol style="list-style-type: none"> <li>1. El usuario entra en la página de consulta.</li> <li>2. Selecciona la fuente de datos.</li> <li>3. Selecciona un filtro (columna, comparador y valor).</li> <li>4. Si quiere añadir más fuentes pulsa ‘+’ y repite desde el paso 2.</li> <li>5. Selecciona el tipo de <i>join</i> (<i>inner</i>, <i>outer</i>, <i>left</i>, <i>right</i>)</li> <li>6. Escoge las columnas a mostrar y el número de filas.</li> <li>7. Pulsa el botón consultar.</li> </ol>
<b>Postcondición</b>	Se muestra la consulta realizada por el usuario.
<b>Excepciones</b>	<ul style="list-style-type: none"> <li>■ No hay ningún dato que concuerde con el filtro.</li> <li>■ El número de filas totales es menor que el número a mostrar (avisa al usuario).</li> </ul>
<b>Importancia</b>	Alta

Tabla B.2: CU-2 Consulta de datos.

<b>CU-3</b>	<b>Visualización de datos</b>
<b>Versión</b>	1.0
<b>Autor</b>	Iván Arjona Alonso
<b>Requisitos asociados</b>	RF-3, RF-3.1, RF-3.2, RF-3.3
<b>Descripción</b>	Permite al usuario visualizar un conjunto de datos en el mapa.
<b>Precondición</b>	Se ha realizado una consulta (CU-2 <a href="#">B.2</a> ).
<b>Acciones</b>	<ol style="list-style-type: none"> <li>1. El usuario realiza una consulta.</li> <li>2. Selecciona el método de agregación (mean, sum, count).</li> <li>3. Selecciona el nivel al que mostrar el mapa (Municipios o provincias).</li> <li>4. Escoge la columna de datos a representar.</li> </ol>
<b>Postcondición</b>	Se muestra un mapa coroplético con los datos de la columna seleccionada.
<b>Excepciones</b>	<ul style="list-style-type: none"> <li>■ Error al cargar las divisiones geográficas.</li> <li>■ No hay datos suficientes.</li> </ul>
<b>Importancia</b>	Media

Tabla B.3: CU-3 Visualización de datos.

<b>CU-4</b>	<b>Exportar consulta</b>
<b>Versión</b>	1.0
<b>Autor</b>	Iván Arjona Alonso
<b>Requisitos asociados</b>	RF-4
<b>Descripción</b>	Exporta el resultado de una consulta en varios formatos.
<b>Precondición</b>	Se ha realizado una consulta (CU-2 B.2).
<b>Acciones</b>	<ol style="list-style-type: none"> <li>1. El usuario realiza una consulta.</li> <li>2. Pulsa el botón correspondiente al formato en el que quiere descargar el resultado de la consulta.</li> </ol>
<b>Postcondición</b>	Descarga un archivo con los datos de la consulta en el formato seleccionado.
<b>Excepciones</b>	<ul style="list-style-type: none"> <li>■ No hay datos suficientes.</li> </ul>
<b>Importancia</b>	Media

Tabla B.4: CU-4 Exportar consulta.

CU-5	Juntar ficheros CSV
<b>Versión</b>	1.0
<b>Autor</b>	Iván Arjona Alonso
<b>Requisitos asociados</b>	RF-5, RF-5.1, RF-5.2, RF-5.3
<b>Descripción</b>	Junta dos o más ficheros CSV en uno solo.
<b>Precondición</b>	Se tienen al menos dos ficheros CSV con al menos una columna común.
<b>Acciones</b>	<ol style="list-style-type: none"> <li>1. El usuario abre la aplicación para juntar CSV.</li> <li>2. Sube al menos dos ficheros CSV.</li> <li>3. Selecciona la columna por la que juntarlos.</li> <li>4. Selecciona el tipo de <i>join</i> (<i>inner</i>, <i>outer</i>, <i>left</i>, <i>right</i>)</li> <li>5. Pulsa el botón ‘Join’.</li> <li>6. Seleccionar la ruta donde guardar el archivo resultante.</li> </ol>
<b>Postcondición</b>	Descarga un único fichero CSV con la combinación de los anteriores en la ruta seleccionada.
<b>Excepciones</b>	<ul style="list-style-type: none"> <li>■ Hay menos de dos ficheros.</li> <li>■ No tienen columnas en común.</li> <li>■ No se puede acceder a alguno de los ficheros.</li> <li>■ Alguno de los archivos no está en formato CSV.</li> <li>■ No se puede escribir en la ruta seleccionada.</li> </ul>
<b>Importancia</b>	Media

Tabla B.5: CU-5 Juntar ficheros CSV.

## Actores

Actores que utilizan la aplicación:

- **Usuario:** usuario final de la aplicación. Interactúa con la aplicación web y la herramienta para juntar CSV.
- **Administrador de datos:** Encargado de mantener los datos en la aplicación. Actualiza las y añade fuentes de datos.

## Diagrama de casos de uso

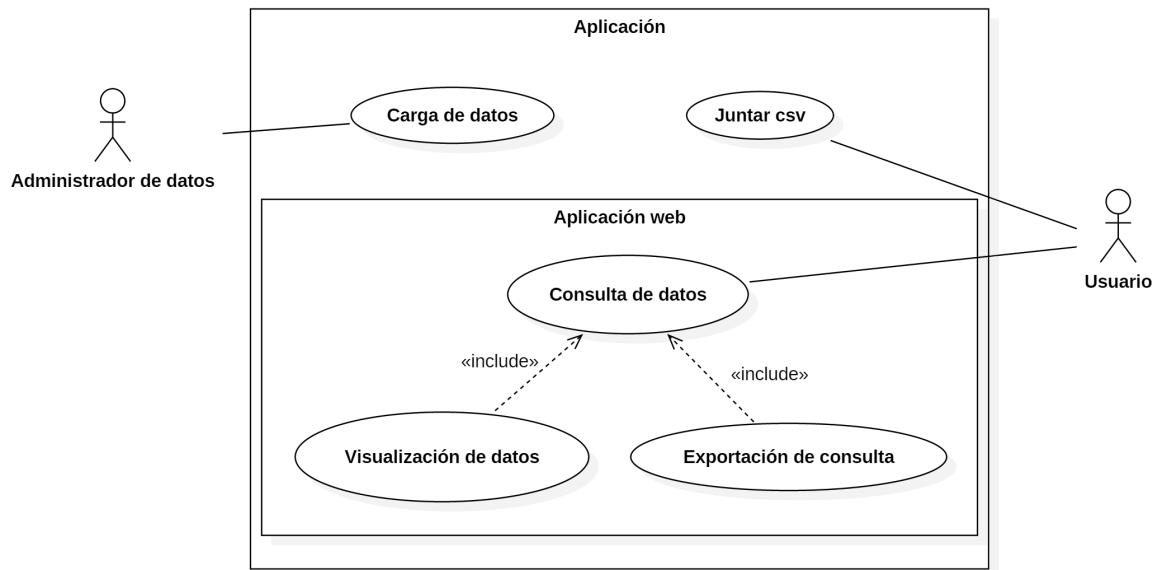


Figura B.1: Diagrama de casos de uso



## *Apéndice C*

---

# Especificación de diseño

---

## C.1. Introducción

En este anexo se explican los diseños que se han llevado a cabo para realizar los objetivos anteriores. Se incluye el diseño de datos, diseño procedimental y diseño arquitectónico.

## C.2. Diseño de datos

La base de datos utilizada en este proyecto es una base de datos de tipo NoSQL, por lo que no hay un sistema relacional de tablas.

Como en la aplicación se trabaja con datos públicos, no se consideró realizar un sistema de usuarios, por lo que cualquiera puede usarla.

### Fuentes de datos

Las fuentes de datos se almacenan cada una de ellas en una colección con sus campos correspondientes.

Todas estas fuentes de datos tiene dos campos comunes: código de municipio y código de comunidad. Estos campos se utilizan para agregar varias fuentes de datos en la consulta y dibujar los mapas coropléticos por provincia o municipio.

## Fronteras geográficas

Para poder dibujar los mapas coropléticos es necesario tener almacenados los límites geográficos para pintar las porciones correspondientes.

Estos límites geográficos están almacenados en archivos *GeoJSON* [?], que son estructuras siguiendo el formato *JSON* para representar elementos geográficos sencillos.

Los GeoJSON utilizados se han obtenido de [?] en el caso de los límites geográficos por provincias y [?] para los límites municipales.

En ambos casos se ha utilizado [mapshaper<sup>1</sup>](#) [?] para minimizar los archivos y que el renderizado sea menos pesado en el navegador.

## C.3. Diseño procedimental

El caso más interesante respecto a la ejecución de los algoritmos es el caso de uso actualizar fuentes de datos.

En la figura C.1 se muestra un diagrama de secuencia al actualizar las fuentes de datos.

---

<sup>1</sup>mapshaper: <http://mapshaper.org/>

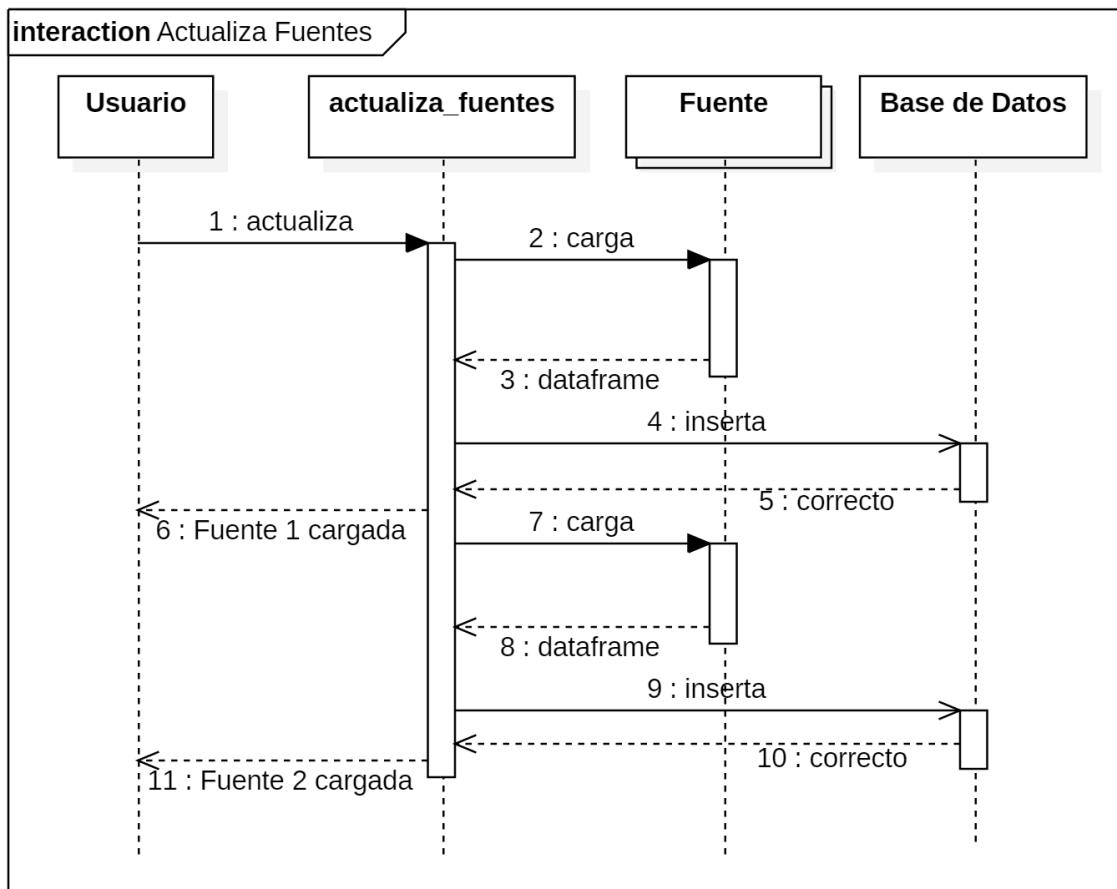


Figura C.1: Diagrama de secuencia para actualizar fuentes de datos

## C.4. Diseño arquitectónico

El diseño de esta aplicación está dividido en dos partes, las fuentes de datos y la aplicación web. Cada una de ellas situada en un paquete.

A continuación se explicará con más detalle cada uno de estos apartados.

### Fuentes de datos

Este paquete contiene los ficheros necesarios para guardar los datos desde cada una de las fuentes en la base de datos.

En primer lugar hay una clase abstracta *Fuente* que contiene las operaciones comunes a todas las fuentes de datos y métodos para acceder a el nombre de la colección y la descripción.

De esta clase *Fuente* heredan todas las fuentes de datos, en el diagrama se muestra la clase *Sepe* para no complicarlo demasiado mostrando todas. Esta clase implementa la lógica común a todos los datos obtenidos de esta fuente, en este caso el SEPE. Se implementa el método *carga()* heredado de *Fuente*, en este método se guarda en un dataframe de Pandas los datos procesados de esta fuente.

De la clase de cada fuente de datos heredan las diferentes consultas que vayamos a almacenar. Por ejemplo para el SEPE tenemos datos de contratos, empleo y paro; cada una en una clase. Esta clase lo que hace es parametrizar a su superclase, por ejemplo, con información de la url base donde se consulta y la lista de años que se usará para acceder a varias urls con la misma estructura dependiendo del año.

Por último, hay que explicar que la función *actualiza\_fuentes* es la encargada de acceder a todas las fuentes, recoge el dataframe de Pandas con la función *carga()* y guarda estos dataframes en la base de datos.

En la figura C.2 se muestra un diagrama de clases explicado anteriormente.

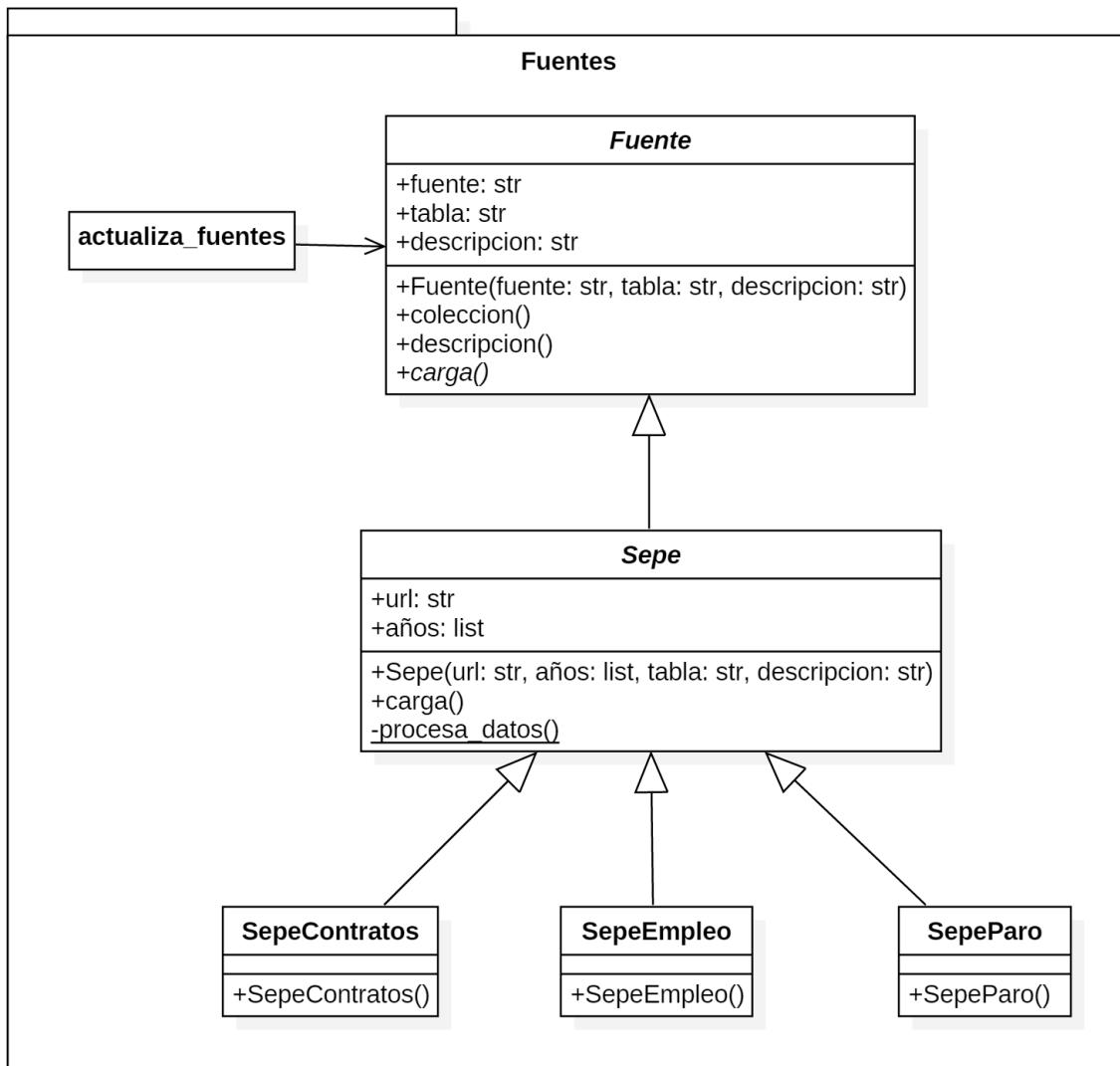


Figura C.2: Diagrama de clases del paquete Fuentes

## Aplicación web

Este paquete está situado en el directorio ‘web’.

Para el diseño de la aplicación web se ha seguido un patrón Modelo–Vista–Controlador. Al utilizar este patrón conseguimos tener una separación entre la vista, que es la parte con la que interactúa el usuario; la lógica que

interacciona con la aplicación y el modelo.

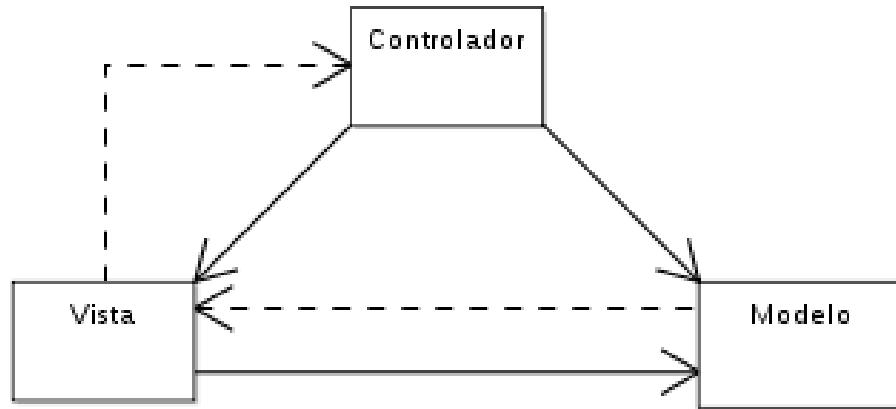


Figura C.3: Diagrama Modelo–Vista–Controlador [?]

Se ha seguido la estructura seguida en el libro *Flask Web Development* de Miguel Grinberg [?]. En este proyecto los modelos se encuentran en la carpeta *forms*, las vistas en la subcarpeta *template* y los controlador en la raíz de la aplicación web.

## *Apéndice D*

---

# **Documentación técnica de programación**

---

## **D.1. Introducción**

En este anexo se explica todo lo que tiene que conocer el programador tanto para instalar y ejecutar la aplicación como para poder seguir con el desarrollo.

## **D.2. Estructura de directorios**

A continuación se van a mencionar los directorios con una breve descripción de su contenido para ayudar a comprender la estructura del proyecto. Además de los directorios, también se van a incluir los archivos más importantes.

```
/  
└ config/ ... Archivos de configuración.  
    └ config.py ... Fichero de configuración de la base  
        de datos.  
└ docs/ ... Documentación: memoria y anexos.  
    └ latex/ ... Código en Latex para generar la  
        documentación.  
        └ img/ ... Imágenes de la documentación.  
        └ tex/ ... Secciones de la documentación en  
            Latex.  
└ memoria.pdf ... Memoria.
```

```
anexos.pdf ... Anexos.
fuentes/ ... Paquete para la carga de datos a
            partir de las fuentes. Un fichero
            por cada fuente de datos.
prototipos/ ... Notebooks de IPython utilizados en
                un principio para implementar las
                fuentes de datos antes de pasarlos a
                la aplicación. Es más fácil trabajar
                con notebooks porque se puede ver el
                resultado en cada paso.
test/ ... Pruebas de la aplicación.
    |__ interfaz/ ... Pruebas de interfaz hechas con
        |__ selenium.
    |__ unitarios/ ... Pruebas unitarias hechas con
        |__ unittest.
web/ ... Raíz de la aplicación web hecha con
        Flask.
    |__ forms/ ... Formularios de Wtforms.
    |__ geojson/ ... Archivos GeoJSON que delimitan
        los límites geográficos en la
        representación de mapas.
    |__ static/ ... Ficheros estáticos.
        |__ css/ ... Hojas de estilo CSS.
        |__ imagenes/ ... Imágenes.

        |__ js/ ... Scripts para el comportamiento
            dinámico en Javascript.
    |__ templates/ ... Plantillas para generar HTML en
        Jinja2.
actualiza-fuentes.py ... Script para actualizar las fuentes
                        de datos en la base de datos.

boxfile.yml ... Archivo de configuración de Nanobox.
joincsv.py ... Aplicación para juntar ficheros CSV.
requirements.txt ... Dependencias de la aplicación
                    (paquetes de Python).

run.py ... Script para ejecutar la aplicación
            web.
```

## D.3. Manual del programador

Esta sección tiene como objetivo explicar los puntos más importantes a tener en cuenta por desarrolladores que puedan seguir ampliando este proyecto.

### Añadir fuentes de datos

El funcionamiento de esta aplicación se basa en los datos obtenidos de fuentes públicas, por lo que un punto importante es explicar cómo añadir más fuentes a nuestra aplicación.

La lógica de esta parte de la aplicación se encuentra en el paquete *Fuentes*.

Lo primero que tenemos que hacer al añadir una fuente es crear un nuevo fichero con el nombre de la fuente. En este fichero crearemos una clase con el mismo nombre. Esta clase heredará de *Fuente* (situado en *Fuente.py*).

Esta clase tiene que implementar el método *carga()*. La funcionalidad de este método es devolver un dataframe de Pandas a partir de los datos realizando una consulta a la fuente que se está implementando. Esta clase tiene que tener un constructor para parametrizar las diferentes consultas que se puedan realizar a la misma fuente de datos.

Por ejemplo, observamos que en el SEPE, la url para consultar el número de contratos en 2018 es la siguiente:

```
https://sede.sepe.gob.es/es/portaltrabaja/resources/sede/
datos_abiertos/datos/Contratos_por_municipios_2018_csv.csv
```

Y la url para consultar los datos del paro en 2018 es la siguiente:

```
https://sede.sepe.gob.es/es/portaltrabaja/resources/sede/
datos_abiertos/datos/Paro_por_municipios_2018_csv.csv
```

Observamos que el principio de la url es el mismo, por lo que lo guardaremos en una variable. Podemos observar que lo único que cambia es el nombre del fichero, en el que también aparece el año. De modo que podemos dividirlo como dos parámetros en el constructor: nombre de fichero y año. Además se incluirá la descripción, que depende de cada consulta.

```
class Sepe(Fuente):
    """
    Fuente de datos para el Servicio Público de Empleo Estatal
```

```

"""
def __init__(self, url, anios, tabla, descripcion_):
    url_sepe = 'https://sede.sepe.gob.es/es/portaltrabaja/
resources/sede/datos_abiertos/datos/'
    self.url = url_sepe + url
    self.anios = anios
    descripcion = descripcion_ + " Servicio Público de Empleo
Estatal."
    super().__init__('sepe', tabla, descripcion)

@staticmethod
def procesa_datos(url):
    # Procesa los datos
    df = pd.read_csv(url, sep=';', encoding='latin1', header=1)
    return df

def carga(self):
    dataframes = []
    for anio in self.anios:
        url = self.url.format(anio)
        df = self.procesa_datos(url)
        dataframes.append(df)
    df = pd.concat(dataframes)
    return df

```

Listing D.1: Ejemplo de fuente de datos (Sepe)

Heredando de la clase *Sepe* crearemos una subclase por cada una de las consultas que queramos realizar. En esta subclase añadiremos un constructor sin parámetros, en el que se llamará al constructor de su superclase parametrizándolo.

Por ejemplo, se crearía la subclase *SepeContratos*, parametrizando el constructor del padre con el nombre del fichero y los años disponibles. Así quedaría la clase *SepeContratos*:

```

class SepeContratos(Sepe):
    """
    Contratos por sexo y por edades por cada municipio
    """

    def __init__(self):
        url = 'Contratos_por_municipios_{0}_csv.csv'
        anios = range(2006, 2019)
        descripcion = 'Contratos por sexo y por edades.'
        super().__init__(url, anios, 'contratos', descripcion)

```

---

Listing D.2: Ejemplo de fuente de datos (SepeContratos)

Una vez implementadas estas clases hay que añadirlas a la lista de fuentes de datos para que se carguen al ejecutar el script de actualizar fuentes. Esta lista está en el fichero `__init__.py` del paquetes *Fuentes*.

```
from fuentes.aeat import AeatRenta
from fuentes.municipios import Municipios
from fuentes.ine import InePoblacion
from fuentes.sepe import SepeContratos, SepeEmpleo, SepeParo
from fuentes.mir import MirCongreso

# Lista de todas las fuentes disponibles
fuentes = [
    Municipios,
    AeatRenta,
    InePoblacion,
    SepeContratos,
    SepeEmpleo,
    SepeParo,
    MirCongreso
]
```

Listing D.3: Lista con todas las fuentes

En el fichero *Fuente.py* tenemos además dos decoradores que se pueden utilizar sobre la función `carga()` (o cualquier otra función que devuelva un dataframe) para realizar ciertas acciones comunes.

- `@to_numeric`: convierte todos las columnas con datos numéricos almacenados como cadenas a tipos de datos numéricos.
- `@rename(dict)`: renombra las columnas de un dataframe de Pandas a las pasadas en el diccionario. Siendo la clave el nombre actual y el valor el nuevo nombre.

## D.4. Instalación y ejecución del proyecto

### Instalación

#### MongoDB

Antes de empezar con la instalación de la aplicación tenemos que instalar la base de datos. Se ha utilizado una base de datos no relacional MongoDB.

Concretamente se instaló la versión *3.6.4 Community Server* [?] para Windows.

También se ha utilizado *MongoDB Compass* [?] para visualizar el contenido de la base de datos. Esta herramienta es opcional.

#### Python

Esta aplicación se ha desarrollado utilizando la versión 3.6.4 [?], por lo que se recomienda utilizar esta versión o una posterior. En cualquier caso debe de instalarse Python 3 o superior para evitar incompatibilidades.

Todos los paquetes utilizados en la aplicación están listados en ficheros *requirements.txt* junto con sus correspondientes versiones.

Habrá que instalar todas las dependencias utilizando pip<sup>1</sup>:

```
pip install -r requirements.txt
```

### Ejecución

#### Configuración

Para configurar los datos de acceso a la base de dato podemos utilizar las variables de entorno: *DATA\_DB\_HOST*, *DATA\_DB\_PORT*, *DATA\_DB\_NAME*, *DATA\_DB\_USER*, *DATA\_DB\_PASS*.

En caso de usar Nanobox estas variables estarán configuradas por defecto, por lo que no tendríamos que hacer nada.

Si no dejamos los valores por defecto al instalar MongoDB, tampoco hará falta hacer nada, se cogerán estos valores automáticamente.

---

<sup>1</sup>Gestor de paquetes de Python [?]

El archivo *config/config.py* también podemos modificar esta configuración. Si se añade alguna característica más en el futuro que requiera algún tipo de configuración debería añadirse una variable a esta clase.

### Actualización de la base de datos

Antes de ejecutar la aplicación tendremos que construir la base de datos con el contenido de todas nuestras fuentes por primera vez.

Para ello simplemente hay que ejecutar el fichero *actualiza-fuentes.py*. Puede tardar un rato debido a la gran cantidad de datos que se van a cargar.

Este paso puede repetirse cada vez que se quiera actualizar las fuentes de datos. Puede ser utilizar ejecutarlo una vez al mes para mantener al día las fuentes con datos mensuales.

```
python actualiza-fuentes.py
```

### Servidor web

El servidor Flask ya se ha instalado como un paquete, por lo que no necesita más instalación. Para ejecutarlo hay un script *run.py* que nos lanza el servidor en el puerto 5000.

Una vez lanzado podremos acceder a la página web desde **localhost:5000**.

```
python run.py
```

## D.5. Despliegue

Para desplegar la aplicación se ha optado utilizar como servidor en la nube DigitalOcean y Nanobox como microservicio para facilitarnos la instalación de la máquina en la nube y la configuración del servidor.

### Instalación

#### DigitalOcean

Primero tendremos que registrarnos en [DigitalOcean<sup>2</sup>](#) y obtener el *token* de nuestro usuario. Tendremos que poner una tarjeta de crédito de la que nos cobrarán el importe del servidor.

---

<sup>2</sup>DigitalOcean: <https://www.digitalocean.com/>

Podría utilizarse otro proveedor de hosting como *Amazon Web services* o *Google Compute*.

### Nanobox

Después nos registraremos en [Nanobox<sup>3</sup>](#) y creamos una nueva aplicación utilizando el token que hemos obtenido antes en DigitalOcean.

Ahora elegiremos el plan que queramos contratar. Para este proyecto será suficiente con el plan más basico de 5€, 1 CPU y 1GB de ram.

Tras crear la aplicación instalamos el [cliente de nanobox<sup>4</sup>](#). La primera vez que lancemos un comando nos pedirá los datos para iniciar sesión en nuestra cuenta.

En el fichero *nanobox.yml* tenemos la configuración con los componentes que se van a instalar y los comandos que se ejecutan al desplegar la aplicación.

### Despliegue

Una vez todo instalado y configurado, cada vez que queramos actualizar la aplicación del servidor nos situamos en la carpeta del proyecto y lanzamos el siguiente comando:

```
nanobox deploy
```

Con esto, de forma transparente para nosotros, se crea una máquina virtual con nuestro proyecto en la que se instalan la base de datos, el entorno de ejecución y los paquetes y se ejecuta la aplicación en el servidor.

## D.6. Pruebas del sistema

Se ha realizado una batería de pruebas para verificar el correcto funcionamiento de la aplicación.

### Pruebas unitarias

Se han realizado pruebas de cada módulo para comprobar su funcionamiento. Estas pruebas están situadas en la carpeta */test/unitarios* y se han realizado con el framework de pruebas *Unittest* [?].

---

<sup>3</sup>Nanobox: <https://nanobox.io/>

<sup>4</sup>cliente de nanobox: <https://docs.nanobox.io/install/>

Estas pruebas se han centrado en la ejecución de todos los métodos utilizados en la consulta (*consulta.py*) y en la representación de mapas *mapas.py*.

Unittest está integrado en la librería estándar de Python, por lo que no hará falta instalar ningún paquete adicional.

Para ejecutar los test utilizamos el siguiente comando en la consola:

```
python -m unittest discover test.unitarios
```

## Pruebas de interfaz

Para probar la interfaz web se ha utilizado *Selenium* con la API que proporciona para Python [?], en lugar de grabar las acciones mediante el plugin de Firefox.

Estas pruebas están situadas en la carpeta */test/interfaz* y se ejecutan de manera similar a las pruebas unitarias, ya que también están hechas con el framework *unittest*.

Para poder ejecutar estas pruebas primero tenemos que instalar el paquete de Selenium:

```
pip install selenium
```

Una vez instalado podemos ejecutar las pruebas con el siguiente comando:

```
python -m unittest discover test.interfaz
```

## Todas las pruebas

También podemos ejecutar todas las pruebas al mismo tiempo:

```
python -m unittest discover
```



## *Apéndice E*

---

# **Documentación de usuario**

---

## **E.1. Introducción**

En este apéndice se explica cuáles son los requisitos que deberá cumplir el usuario para ejecutar la aplicación, como instalarla y cómo utilizarla.

## **E.2. Requisitos de usuarios**

Como se trata de una aplicación web, el único requisito que necesita el usuario es un navegador web que soporte, *Javascript*, *cookies* y hojas de estilo *CSS*. Como nuestra aplicación también utiliza *jQuery*, nuestros navegadores soportados serán los mismos [?]:

- Google Chrome.
- Microsoft Edge.
- Mozilla Firefox.
- Internet Explorer 9 o superior.
- Safari para Mac.
- Opera.
- Navegador de Android 4.0 o superior.
- Safari para iOS 7 o superior.

En el caso de la aplicación para juntar ficheros CSV, haría falta tener instalado Python 3 y *Pandas 0.23*.

## E.3. Instalación

### Aplicación web

Como se trata de una aplicación web, los usuarios no tendrán que realizar ninguna instalación. Sólo serían necesario acceder a la aplicación desde la siguiente url:

<https://tfg-datos-publicos.nanoapp.io>

En caso de que se quisiera instalar la aplicación en local habría que seguir los pasos explicados en la documentación para programador ([D.4](#)).

### Aplicación para juntar CSV

La otra herramienta proporcionada en este trabajo es la aplicación para juntar ficheros CSV. En este caso, tampoco haría falta instalación como tal, pero si tener instalado Python 3 y Pandas 0.23. Una forma rápida de instalarlo es usando el instalador de *Anaconda*<sup>1</sup> con *Python 3.6*.

Para ejecutar la aplicación:

```
python joincsv.py
```

## E.4. Manual del usuario

En esta sección se va a explicar al usuario cómo utilizar la aplicación.

### Consulta

Para empezar, nos dirigiremos a la [página de consulta](#)<sup>2</sup>.

---

<sup>1</sup>Anaconda: <https://www.anaconda.com/download/>

<sup>2</sup>página de consulta: <https://tfg-datos-publicos.nanoapp.io/consulta>

The screenshot shows a user interface for querying data. At the top left is a title 'Consulta'. Below it, there's a section for 'Consulta 1' with a '+' button to add more. A dropdown menu labeled 'Fuente de datos' is set to 'municipios'. To the right, a detailed description of 'municipios' is provided: 'Lista de municipios con sus códigos de comunidad, provincia y municipio del instituto nacional de estadística'. There are several input fields: 'Columna' (set to 'Todas'), 'Comparador' (set to '='), and 'Valor' (empty). A 'Consultar' button is at the bottom left of this section. On the right side, there are three main sections: 'Columnas a mostrar' (listing 'Codigo comunidad', 'Codigo Provincia', 'Codigo municipio (local)', 'DC', 'Municipio', and 'Codigo Municipio'), 'Columna calculada' (empty), and 'Filas a mostrar' (set to '1000').

Figura E.1: Página de consulta

En la primera columna, seleccionaremos la fuente de datos de la que queramos consultar la información y los filtros que queramos. Por ejemplo, si queremos consultar los datos de la renta en Miranda de Ebro, seleccionaremos rellenaríamos lo siguiente:

**Fuente de datos**

aeat\_renta ▾

**Columna**

Municipio ▾

**Comparador**

= ▾

**Valor**

Miranda de Ebro

Figura E.2: Filtros de la consulta

Podemos filtrar por cualquier columna de la fuente, utilizando los siguientes comparadores: igual (=), distinto (!=), menor (<), mayor (>), menor o igual (<=), mayor o igual (>=).

En la columna de la derecha nos mostrará la descripción de la fuente que hayamos seleccionado y podremos seleccionar las columnas que queramos ver.

Por último, elegiremos el número de filas a mostrar, este es el número máximo de filas que se pueden ver en la previsualización de la tabla, al descargar la consulta o mostrar el mapa, aparecerán todos los datos (por defecto 1000).

## Columnas a mostrar

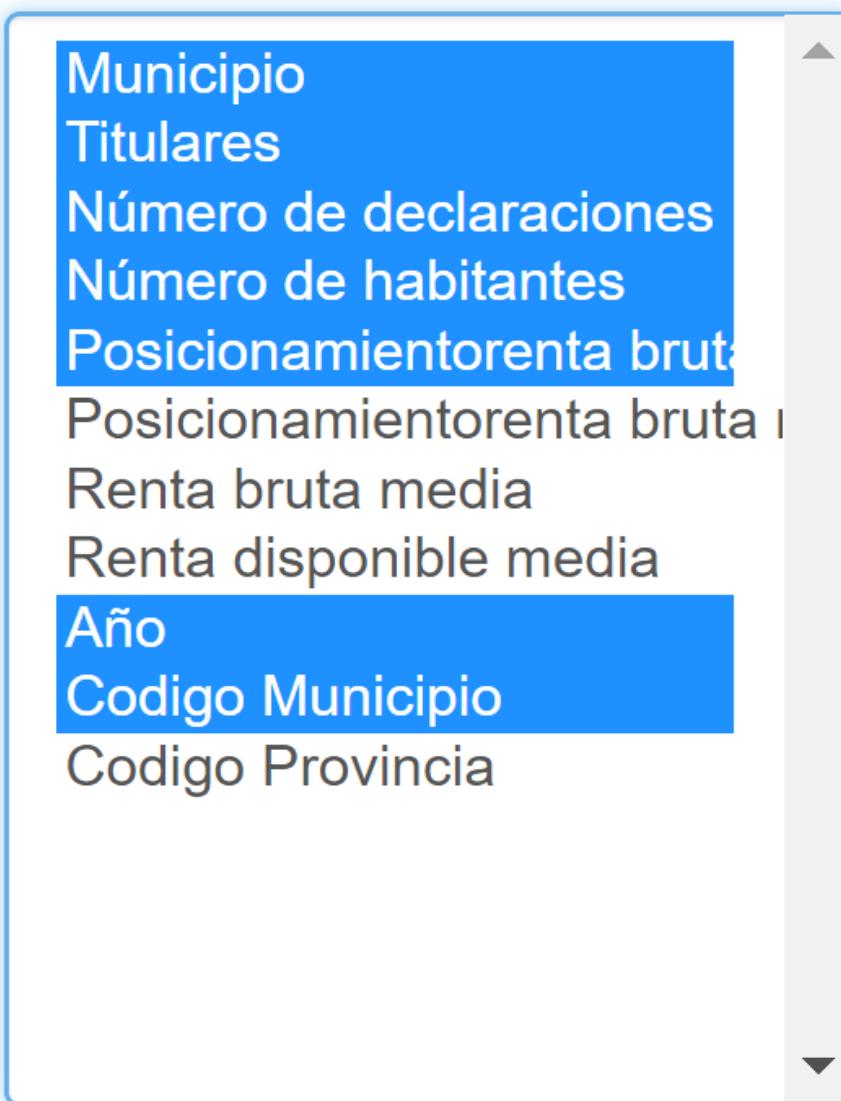


Figura E.3: Seleccionar columnas a mostrar

Una vez rellenados todos los datos para la consulta, le damos al botón ‘Consultar’ y nos llevará a la visualización.

## Visualización

Tras haber realizado la consulta, podremos ver los resultados. Se mostrará una tabla con los datos filtrados.

Podemos elegir el número de filas mostradas por cada página, ordenar los datos por la columna que queramos o buscar dentro de la tabla.



The screenshot shows a data visualization interface titled 'Consulta'. At the top, there are download options for 'csv' and 'json', and dropdown menus for 'mean', 'provincias', and 'Visualizar mapa'. Below these are filters for 'Show: 10' and a search bar. The main area displays a table with the following data:

	Año	Código Municipio	Municipio	Número de declaraciones	Número de habitantes	Posicionamiento renta bruta media nivel nacional	Titulares
0	2013	09219	Miranda de Ebro	19106	36724	370	22959
1	2014	09219	Miranda de Ebro	18827	36173	375	22582
2	2015	09219	Miranda de Ebro	18873	35922	431	22534

At the bottom, it says 'Showing 3 of 3 records' and 'Pages: Previous 1 Next'.

Figura E.4: Visualización de la consulta

En caso de que hayamos superado el número de filas a mostrar, nos aparecerá un mensaje informándonos de cuántas filas se están mostrando y cuál es el número real de filas.



Figura E.5: Límite de filas superado

## Exportación de consultas

Para exportar una consulta, simplemente presionamos el botón 'csv' o 'json', dependiendo del formato que nos interese.

Se descargará los datos de la consulta completa. Pueden ser más de los que aparecen en la tabla, si ha superado el número de filas a mostrar que hemos elegido al realizar la consulta.



Figura E.6: Exportar consulta

## Consulta de varias fuentes

Empezamos de nuevo en la [página de consulta](#)<sup>3</sup>.

Una vez rellenado la primera consulta, como se ha explicado en [E.4](#), pulsaremos el botón '+' y nos aparecerá otra pestaña con una nueva consulta.



Figura E.7: Botón para añadir una subconsulta

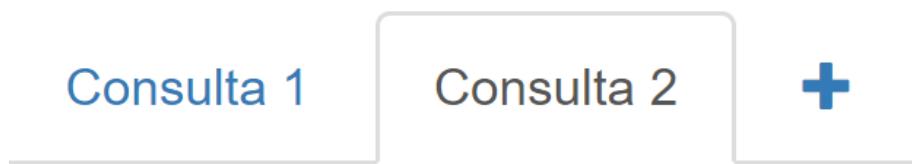


Figura E.8: Pestañas con varias subconsultas

<sup>3</sup>página de consulta: <https://tfg-datos-publicos.nanoapp.io/consulta>

Las consultas se llenan independientemente igual que en E.4. Podemos cambiar de subconsulta pulsando su pestaña.

Hay que tener en cuenta que al seleccionar las columnas a mostrar, tenemos que seleccionar ‘Código Municipio’, que es la columna común entre todas las subconsultas por la que se agruparán.

Una vez llenados los campos de todas las subconsultas, elegiremos cómo queremos que se junten. Para ello en el campo *join*, que habrá aparecido al presionar ‘+’ por primera vez, seleccionamos el método por el que queremos que se junten.

Los posibles métodos son: *inner join*, *outer join*, *left join*, *right join*.

- *inner*: Intersección de todos los municipios entre las fuentes.
- *outer*: Unión de los municipios entre fuentes.
- *left*: Todas las entradas de la primera subconsulta unidas con las demás subconsultas.
- *right*: Todas las entradas de la última subconsulta unidas con las demás subconsultas.

## Join

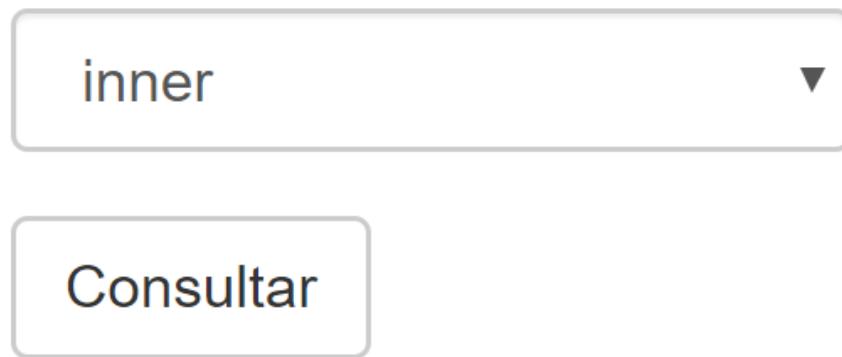


Figura E.9: Método para juntar subconsultas

Cuando hayamos terminado pulsamos ‘Consultar’ y el resultado aparecerá de igual modo que en E.4.

## Columna calculada

Al hacer una consulta, podemos añadir un campo combinando varias columnas como queramos.

Para ello, utilizaremos el campo ‘Columna calculada’.

Al empezar a escribir el nombre de una columna existente, se autocompletará con las columnas que contengan esos caracteres. Para que esto funcione, habrá que escribir un espacio antes de cada campo.

## Columna calculada

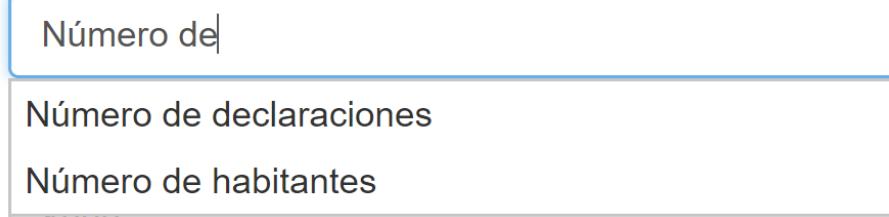


Figura E.10: Autocompletar en columnas calculadas

En este campo podemos utilizar cualquier operador matemático. Incluyendo sumas, restas, divisiones, multiplicaciones, potencias, mínimo (`.min()`), máximo (`.max()`), media (`.mean()`), entre otros.

Por ejemplo, usando la fuente ‘aeat\_renta’, si queremos saber el porcentaje de habitantes que han hecho declaraciones, usaremos la siguiente consulta calculada:

Número de declaraciones / Número de habitantes * 100
--

	Municipio	Número de declaraciones	Número de habitantes	Titulares	Calculada
0	Almería	80509	193351	95335	41.638781
1	Carboneras	3051	7852	3879	38.856342
2	Roquetas de Mar	34158	91682	39781	37.257041
3	Huércal de Almería	7410	16663	8887	44.469783

Figura E.11: Visualización de una consulta con una columna calculada

Número de declaraciones entre el valor máximo de declaraciones:

Número de declaraciones / Número de declaraciones .max()
--

Es importante destacar que hay que seleccionar todas las columnas que usemos en una consulta calculada en el campo ‘columnas a mostrar’.

Si la consulta calculada no es válida, nos mostrará un mensaje y no se mostrará, pero sí que se mostrará el resto de la consulta.

Figura E.12: Columna calculada no válida

## Mapas coropléticos

Para visualizar los mapas coropléticos, primero tendremos que haber realizado una consulta.

En la página donde se muestra el resultado de la consulta, aparece una serie de desplegable para mostrar los mapas.

Figura E.13: Desplegables para visualizar el mapa

El primer desplegable indica el método por el que agrupar los datos. Posibilidades: *mean, sum, count*.

El segundo desplegable es el nivel al que se representa el mapa. Puede ser a nivel de provincia o a nivel de municipio. Cuando se selecciona provincia, se agrupan todos los registros de cada provincia y cuando se selecciona municipio, se agrupan todos los registros del mismo municipio.

El tercer desplegable ‘Visualizar mapa’, es la columna que se quiere visualizar, se mostrarán sólo las columnas numéricas. Al pulsarlo nos abrirá el mapa visualizando la columna que pulsemos, con las configuración de los dos desplegables anteriores.

mean ▾ provincias ▾ Visualizar mapa ▾

Search:

Antorenta nacional	Renta bruta media	Renta disponible media
	22868	18837
	20672	17294
	20582	17072
	20386	17150

Año  
Número de declaraciones  
Número de habitantes  
Posicionamiento renta bruta media nivel autonómico  
Posicionamiento renta bruta media nivel nacional  
Renta bruta media  
Renta disponible media  
Titulares  
Calculada

Figura E.14: Desplegable para seleccionar la columna a visualizar

En este caso visualizaremos la consulta calculada que habíamos elegido en la sección E.4, agrupando por media (*mean*).

Mapa por provincias (Figura E.15).

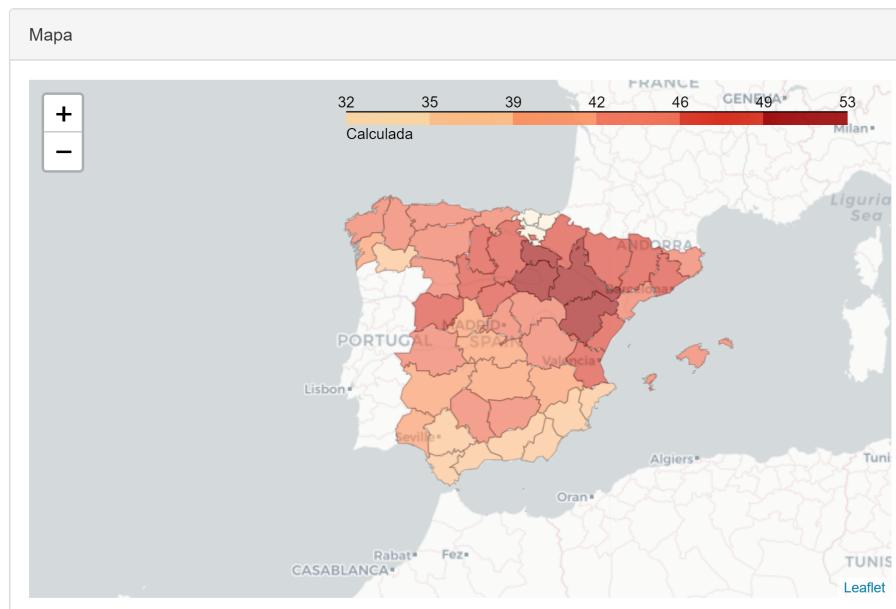


Figura E.15: Mapa por provincias de la columna calculada

Mapa por municipios (Figura E.16).

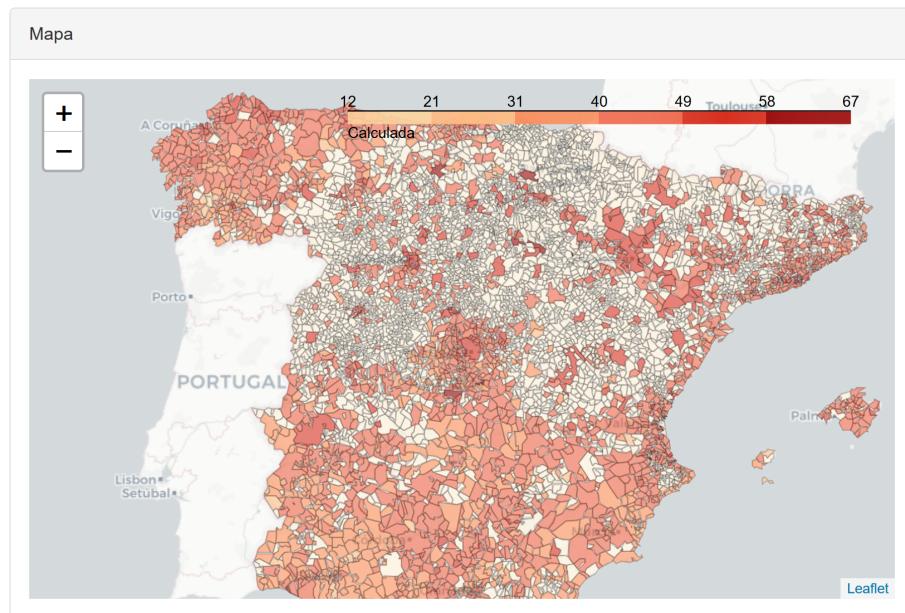


Figura E.16: Mapa por municipios de la columna calculada

Es importante mencionar que si queremos visualizar el mapa por provincia hayamos seleccionado ‘Codigo Provincia’ en columnas a mostrar, y si se va a representar el mapa a nivel de municipio, se deberá haber seleccionado ‘Codigo Municipio’.

Selecciona 'Codigo Provincia' en columnas a mostrar

×

Figura E.17: Error al intentar visualizar un mapa sin haber seleccionado la clave correspondiente

En el caso de mostrar el mapa por municipios, esta opción sólo funciona el Firefox, en Chrome no funciona por limitaciones del framework (lineas futuras).

## Juntar ficheros CSV

Se ha hecho una herramienta para juntar ficheros CSV. Se ha realizado como una aplicación de escritorio para que los usuarios no tengan que subir sus ficheros a una aplicación web por cuestiones de confianza y privacidad.

Para iniciarla:

```
python joincsv.py
```

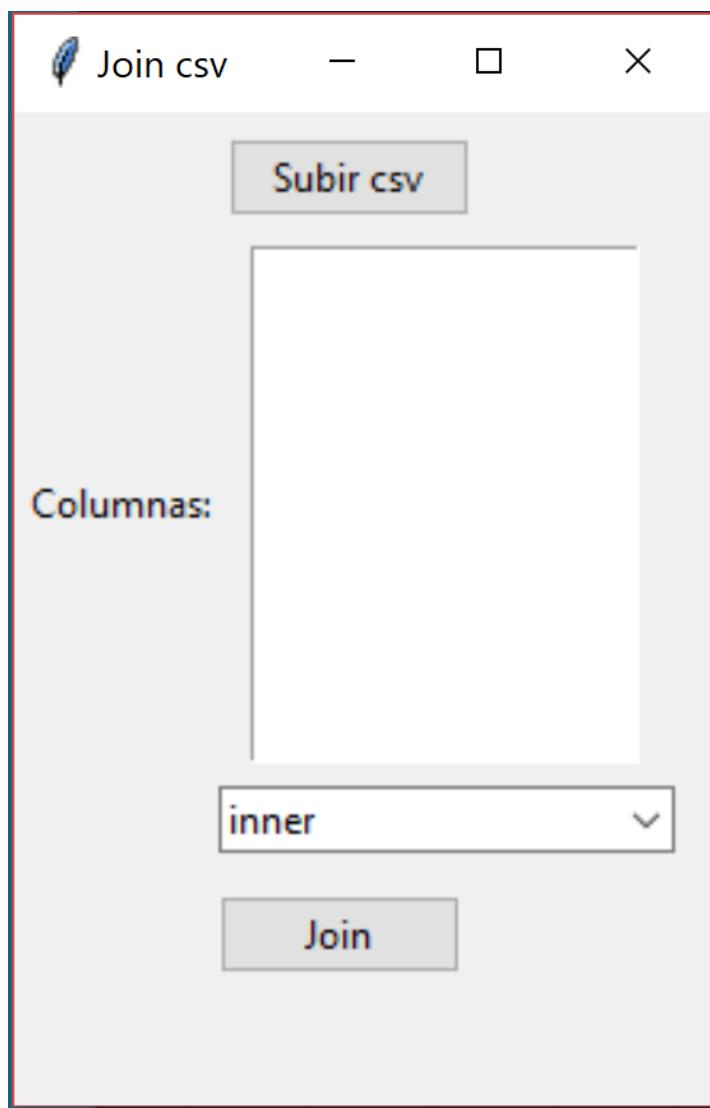


Figura E.18: Aplicación de escritorio para juntar ficheros CSV

Una vez iniciada, habrá que pulsar el botón ‘Subir csv’ y seleccionar todos los ficheros que quiera juntar. Otra opción es seleccionarlos de uno en uno, pulsando varias veces ‘Subir csv’.

Podemos usar los ficheros resultantes de exportar como CSV de nuestra aplicación ([E.4](#)) junto con algún fichero del usuario.

Tras subir los ficheros, nos mostrará cuales son los ficheros mostrados (Figura [E.19](#)).

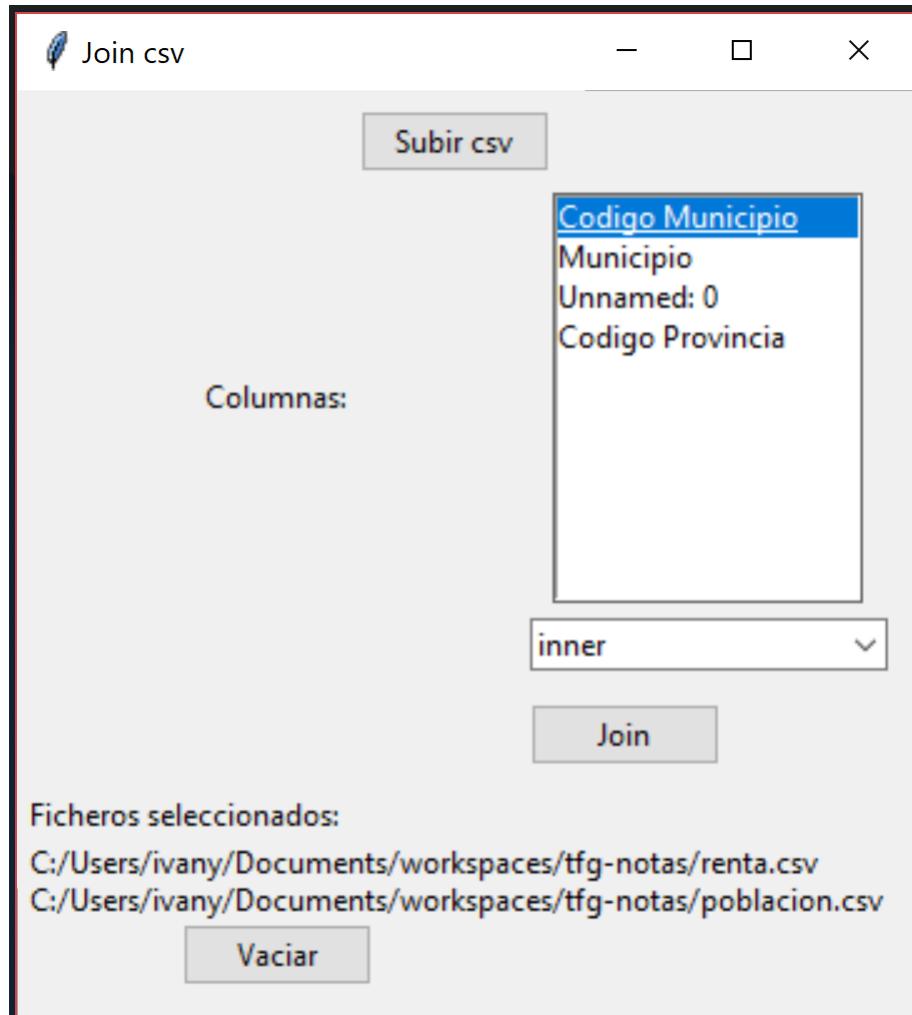


Figura E.19: Selección de columnas al juntar CSV

Si nos hemos confundido y queremos volver a empezar, pulsamos el botón ‘Vaciar’ y volveremos al principio de la aplicación.

Se mostrará una lista de selección múltiple con las columnas comunes a todos los ficheros. Ahora tendremos que seleccionar cuáles son las columnas por las que queremos hacer join (por defecto se seleccionar ‘Codigo Municipio’, si existe en todos los ficheros) y el tipo de join que queramos utilizar (*inner, outer, left, right*).

Cuando hayamos terminado, pulsamos ‘Join’ y nos aparecerá una ventana para seleccionar la ruta donde guardar el fichero resultante.