Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ESTUDIO Y DESARROLLO DE UN PROTOTIPO DE APLICACIÓN WEB PARA EL ANÁLISIS DE RESULTADOS EN PROCESOS ELECTORALES

Alumno: Juan Manuel Cañas Delgado

Tutor: Prof. D. José Ramón Balsas Almagro
Dpto: Departamento de Informática

Septiembre, 2019

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don José Ramón Balsas Almagro, tutor del Trabajo Fin de Grado titulado: Estudio y desarrollo de un prototipo de aplicación web para el análisis de resultados en procesos electorales, que presenta Juan Manuel Cañas Delgado, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, septiembre de 2019

El alumno: Los tutores:

Juan Manuel Cañas Delgado José Ramón Balsas Almagro

Índice

Índice de ilustraciones

Índice de tablas

1. INTRODUCCIÓN

1.1. Motivación

Como sabemos, en la actualidad tener un buen sistema electoral es algo fundamental, al ver todo el revuelo de los últimos años con respecto a elecciones y cambios de partidos, pensé en cómo se hacia el reparto de escaños y si habría otras opciones para llevar a cabo esta tarea.

Por ello para ayudar a que las personas puedan realizar una simulación de distribución de escaños por ellos mismo se ha pensado en hacer una herramienta que les facilite esta tarea. Esta herramienta irá dedicada sobre todo al público en general, pero será accesible a todo el mundo pudiendo usarla tanto medios de comunicación como incluso partidos políticos

Para todo esto, en este trabajo se propone realizar una aplicación web, para que todo el mundo con acceso a un navegador pueda hacer uso de ella y beneficiarse de sus utilidades.

1.2. Objetivos

Para comenzar el desarrollo del trabajo, inicialmente se plantearon una serie de objetivos principales a alcanzar que orientarían todo el proceso. Estos se definieron en la propuesta original de TFG y son los siguientes:

Realizar un estudio de necesidades y soluciones existentes en el contexto seleccionado. Por ejemplo, haciendo recopilación de información sobre los principales sistemas electorales actualmente en uso, así como de posibles soluciones *software* existentes que ayudan a su estudio e implantación

Diseñar un sistema informático especialmente orientado a la utilización de arquitecturas, principios de diseño y metodologías de desarrollo web. En principio se tratará de poner en práctica los conocimientos adquiridos en las diferentes asignaturas

a lo largo del Grado, profundizando en aquellos que sea necesario para obtener los resultados adecuados.

Seleccionar un entorno de desarrollo web e implementar un prototipo de aplicación web que satisfaga las necesidades que se hayan determinado.

1.3. Metodología

Una vez fijados los objetivos principales, el siguiente paso será determinar una metodología de trabajo que permita planificar las diferentes tareas que deban realizarse.

En este proyecto se ha decidido utilizar una metodología de desarrollo ágil [1] frente a las tradicionales por varias razones; estas metodologías son iterativas e incrementales, se enfocan en el desarrollo de entregables para el cliente, anticipan el software al diseño y la documentación (sin dejar estas tareas de lado, simplemente se reparten en las diferentes iteraciones), se adaptan mucho mejor al cambio y están orientadas a equipos de proyectos pequeños.

Dentro de las metodologías agiles se ha elegido *Scrum* [1] que nos permite obtener el mayor valor de negocio en el menor tiempo posible. En *Scrum* los equipos son auto-organizados y las iteraciones son llamadas *Sprints*, las cuales suelen ser de dos semanas de duración.

A continuación, vemos como aplicamos *Scrum* siguiendo su ciclo de trabajo como observamos en la Ilustración 1.

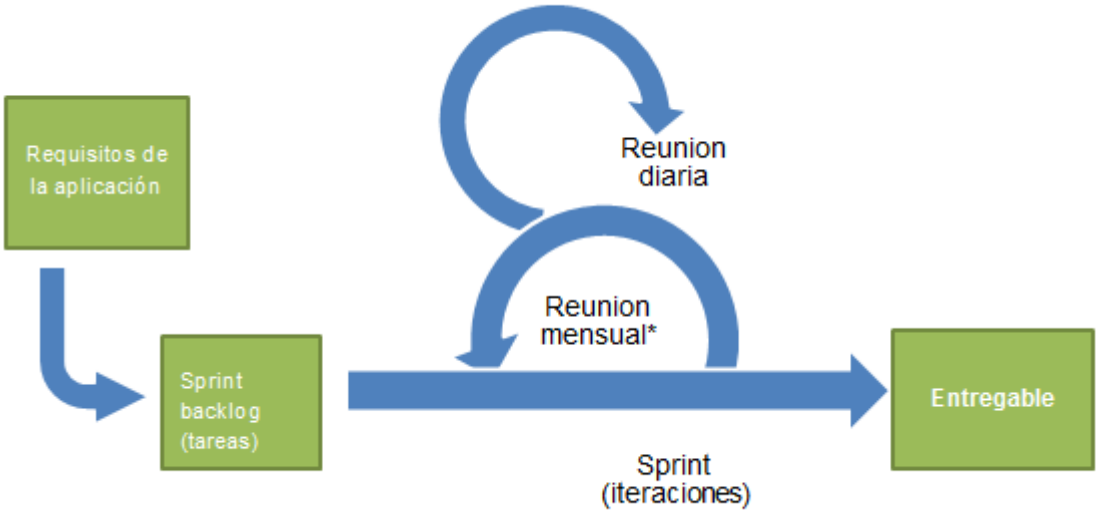


Ilustración 1: Ciclo de trabajo *Scrum*

Tomaremos los requisitos del cliente y crearemos nuestras historias de usuario, descompondremos estas historias de usuario en tareas y las ordenaremos por prioridad en una pila de producto (*Backlog*). Repartiremos estas tareas priorizadas en las diferentes iteraciones o *Sprints*. Una vez terminado cada *Sprint* tendremos una funcionalidad completa, y repetiremos este proceso hasta terminar el *Backlog*.

Dentro del marco de trabajo de *Scrum* también hay diferentes roles:

- El *Product Owner* que es el responsable del producto desde el punto de vista del cliente y busca la máxima rentabilidad de producto.
- El *Scrum Master* que es el encargado de hacer que se cumplan las prácticas de *Scrum* y ayuda al resto del equipo a alcanzar su máxima productividad.
- El *Scrum Team* son los encargados de desarrollo.

En este caso al trabajar solo yo llevaré a cabo todos los roles, además en *Scrum* también hay una serie de reuniones:

- *Sprint planning:* En ella se crea el *Backlog* a partir de las historias de usuario que el equipo se comprometer a completar. Se obtienen y estiman las tareas de cada historia de usuario.

- *Sprint review:* En ella el equipo presenta el trabajo realizado durante un *Sprint.*
- *Sprint retrospective:* Después de cada *Sprint* el equipo hace una revisión de lo que funciona y lo que no.
- *Daily scrum meeting: E*s una reunión diaria en la que todo el mundo comenta que hizo ayer, que va a hacer hoy y si tiene algún problema.

Los *Daily scrum meeting* en este caso no tiene sentido hacerlos ya que yo mismo desempeño todos los roles.

Los *Sprint review*, si fueran necesarios, se podrían consultar con el tutor para aclarar posibles cuestiones que pudieran surgir durante el desarrollo del trabajo

El resto no serán reuniones tampoco, ya que solo estaré yo, pero llevaré a cabo la planificación inicial a modo de *Sprint Planning* y haré una revisión de lo que funciona y no después de cada *Sprint* (*Sprint retrospective*).

1.4. Estructura del documento

A continuación, vamos a presentar una visión general del documento explicando los diferentes apartados en que se ha organizado.

Comenzaremos por el apartado de análisis donde estudiaremos las características del problema a resolver y las necesidades existentes, además también haremos un estudio de las soluciones ya existentes para el problema, para después hacer una propuesta de solución que satisfaga las necesidades y mejore las soluciones existentes. Posteriormente, explicaré brevemente las tecnologías y *framerworks* que usaré para la solución propuesta.

Después empezaremos con el desarrollo ágil, veremos qué es y las partes de las que se compone. Aplicaré este desarrollo ágil al proyecto mostrando las historias de usuario, con más detalle, todas las tareas obtenidas, las priorizaré y expondré la información de cada sprint. A partir de todo esto se estudiarán los posibles costes de desarrollo para obtener la viabilidad del proyecto.

Para finalizar con el análisis definiremos en modelo de dominio donde estarán representadas nuestras clases conceptuales obtenidas de entidades y conceptos procedentes del mundo real y que representan el dominio del problema y un pequeño análisis de la interfaz en el que veremos los dos tipos de usuarios que accederán al sistema.

El siguiente apartado es el diseño en el que obtendremos diferentes diagramas, comenzando por el diagrama de entidad-relación y los diagramas de clases, que permitirán respectivamente modelar la información en un sistema de bases de datos relacional y definir la estructura del software a desarrollar. Después, seguiremos con los diagramas de secuencia en los cuales haremos referencia a los aspectos más relevantes del funcionamiento interno que hemos propuesto en de diagrama de clases.

Continuaremos con un diseño de la interfaz, realizado con una herramienta de prototipado, para tener clara la apariencia visual de la aplicación y estudiar el flujo de trabajo de los usuarios con el sistema propuesto. Después veremos un pequeño diseño del API REST donde estarán expuestas las diferentes URLs disponibles del servicio REST. Para terminar con el apartado de diseño habrá un plan de pruebas, en el que se explicaran los diferentes tipos de pruebas que van a planificarse.

Los siguientes dos apartados son, el apartado de implementación donde veremos unos diagramas de arquitectura y entraremos más en detalles sobre la implementación de la aplicación; y el apartado de pruebas donde veremos los resultados obtenidos de realizar el plan de pruebas obtenido en el apartado de diseño.

Y por último tendremos un apartado en el que expondremos las conclusiones a las que hemos llegado después de la realización del proyecto y la bibliografía utilizada.

Además, habrá unos apéndices sobre la instalación del sistema, un manual de usuario y una descripción del contenido suministrado.

2. ANÁLISIS

El análisis [2] de un proyecto informático se centra en el “qué”, es decir, estudia el sistema desde el punto de vista del dominio del problema.

En este apartado, en primer lugar, explicaremos que son los sistemas electorales y realizaremos un análisis preliminar de los diferentes métodos de reparto de escaños a partir de los votos obtenidos por cada partido; posteriormente veremos una serie de herramientas existentes en el mercado que nos permitan identificar funcionalidades y mejoras que nos aporten ideas interesantes y funcionalidades que deban ser tenidas en cuenta en la solución propuesta.

2.1. Análisis preliminar

Un sistema electoral es un conjunto de reglas que determinan cómo se llevan a cabo las elecciones y los referendos y cómo se determinan sus resultados. Los sistemas electorales políticos están organizados por los gobiernos, mientras que las elecciones no políticas pueden tener lugar en empresas, organizaciones sin ánimo de lucro y organizaciones informales. [3]

Los sistemas de representación proporcional, son una categoría de sistemas electorales, que determinan la forma en que las opiniones políticas de los ciudadanos individuales, que son muchos, dan mandato a los miembros del Parlamento, que son sólo unos pocos. Las mismas técnicas se aplican cuando en el Parlamento, los grupos políticos deben estar representados en un comité de un tamaño mucho menor que el propio Parlamento. A la hora de convertir los votos en escaños, no se puede realizar directamente mediante un reparto proporcional, puesto que entonces repartiríamos números racionales y no números enteros de escaños, debido a esto la representación proporcional inevitablemente culmina en la tarea de traducir los “grandes” números de los que van a ser representados en “pequeños” números de los que van a servir como representantes. La tarea se resuelve mediante procedimientos llamados métodos de reparto o fórmulas electorales. [4]

La fórmula electoral es el cálculo matemático mediante el cual, en una votación, se distribuyen los escaños de una asamblea en función de los votos del electorado. [5]

Un enfoque riguroso del reparto debe hacer uso de las funciones de redondeo y de las reglas de redondeo. Las dos clases principales de métodos de reparto son los métodos de divisores y los métodos de *Quotas* [4]. Por lo general, el número de votos de entrada es mucho mayor que el número de escaños de salida deseado. Por lo tanto, el recuento de votos se reduce a cocientes provisionales de una magnitud adecuada. A continuación, los cocientes intermedios se redondean a números enteros. Los métodos del divisor utilizan un divisor flexible para el primer paso y una regla de redondeo específica para el segundo. Los métodos de *Quotas* utilizan un divisor de fórmula, llamado *quota*, para el primer paso y una regla de redondeo flexible para el segundo.

Algunas de las fórmulas electorales más conocidas son: el Cociente Hare, *Sainte-Lagué, Sainte-Lagué* modificada, voto único transferible, Cociente *Droop*, *Imperiali, Hagenbach-Bischoff, D'Hondt*, voto único no transferible, voto limitado, mayoría absoluta y mayoría relativa (ordenadas de mayor a menor proporcionalidad). [5]. Aquí podemos ver un ejemplo práctico del Sistema D'Hont:

Supongamos que tenemos que repartir 6 escaños y los resultados de una votación han sido los siguientes:

Partidos	Votos
Partido A	1255
Partido B	975
Partido C	500
Partido D	87

Tabla 1: Ejemplo D'Hont - votos

En primer lugar, habría que comprobar que el partido con menos votos tiene al menos el 3% mínimo del total de votos emitidos, que establece la ley, en este caso si lo cumple.

Una vez hecho se creará una tabla con tantas filas como partidos y una de totales y mínimo tantas columnas como escaños tengamos que repartir, estas columnas serán los divisores.

Después se divide el número total de votos del partido por el divisor correspondiente, y se buscan los 6 cocientes más altos de la tabla, los cuales aparecen marcados de rojo.

Partidos	Votos/1	Votos/2	Votos/3	Votos/4	Votos/5	Votos/6	Escaños Asignados	Escaños Proporcionales
Partido A	1255,00	627,50	418,33	313,75	251,00	209,17	3	2,67
Partido B	975,00	487,50	325,00	243,75	195,00	162,50	2	2,08
Partido C	500,00	250,00	166,67	125,00	100,00	83,33	1	1,06
Partido D	87,00	43,50	29,00	21,75	17,40	14,50	0	0,19
Total	**2817,00**	**1408,50**	**939,00**	**704,25**	**563,40**	**469,50**	**6**	**4**

Tabla 2: Ejemplo D'Hont – cocientes, escaños asignados y proporcionales

Por cada cociente marcado de rojo se le asignará un voto al partido correspondiente.

En este ejemplo se puede observar también a diferencia con respecto a un reparto proporcional, el cual repartiría números racionales.

2.2. Soluciones existentes

En el mercado ya existen páginas web que permiten realizar este tipo de cálculos, introducir unos partidos y sus correspondientes votos, seleccionar alguna formular electoral y obtener los escaños, aquí presentamos algunos de los ejemplos más relevantes que hemos encontrado y analizamos sus ventajas y posibles inconvenientes:

2.2.1. Simulador electoral ElectoSIM

Esta herramienta está disponible en http://electosim.brainum.es/ y cuenta con las siguientes funcionalidades:

Sistemas electorales

D'hondt, Sainte Lagüe, Sainte Lagüe Modificado*, Cuota Hare, Cuota Imperiali, Cuota Hagenbach-Bischoff y Winner Takes it All* en una circunscripción o en varias.

Formas de entrada de datos

El sistema acepta como formato de entrada archivos *JSON*[1] con el número de votos de cada partido. También se pueden introducir los datos a mano en un formulario.

Formato de salida de datos

El sistema permite descargar los resultados en un archivo *JSON* con el número de escaños de cada partido.

Personalización

El sistema no utiliza distintos usuarios por lo tanto a personalización es extremadamente limitada.

Visualización

Tras el cálculo del resultado el sistema muestra una tabla y una gráfica asociada, como podemos ver en la Ilustración 2

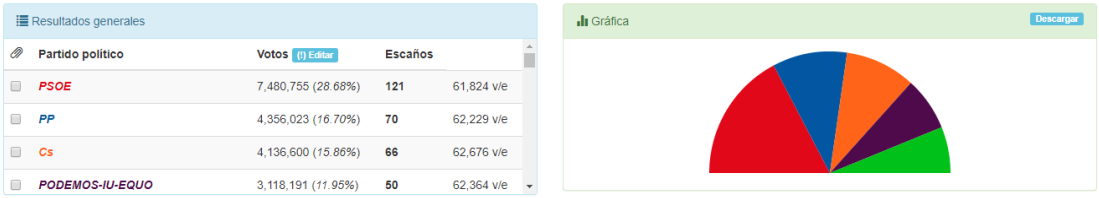


Ilustración 2: Simulador electoral ElectoSIM - tabla y Gráfica

Otros

Este sistema tiene una utilidad para estudiar el impacto de posibles pactos electorales, denominada "pactómetro", con el cual comprobar si un bloque de partidos llega a la mayoría simple o a la mayoría absoluta.

Tiene funcionalidades que permiten modificar los datos de entrada de forma rápida.

[1] JSON (acrónimo de *JavaScript Object Notation,* "notación de objeto de JavaScript") es un formato de texto sencillo para el intercambio de datos.

Utiliza métodos no seguros de reparto de escaños sin preaviso a los usuarios, por ejemplo, el método de cuota *Hagenbach-Bischoff* puede repartir más escaños que el total. Lo que podemos ver en este ejemplo es un reparto de 9 escaños en 5 + 5 = 10 escaños, como vemos en la Ilustración 3.

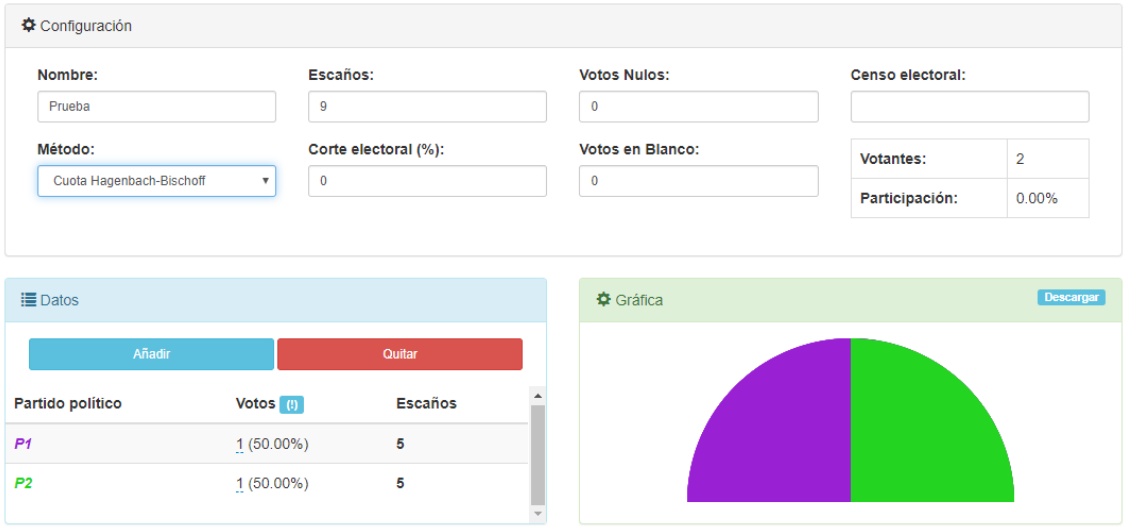


Ilustración 3: Simulador electoral ElectoSIM - Ejemplo de aplicación de métodos de reparto no seguros

También se ha observado que realiza los cálculos directamente en cliente sin llamar a ningún servidor.

Concusiones

Esta aplicación web es bastante completa y es visualmente muy atractiva e incluso incluye algunos datos de ejemplo de las Elecciones Generales. Cabe destacar lo dinámica que es actualizando la taba y la gráfica solo con seleccionar el método o las elecciones deseados.

Sin embargo, utilizar métodos inseguros sin avisar a los usuarios de los posibles resultados incoherentes es un punto muy negativo. Además, la aplicación no permite la posibilidad de almacenar las diferentes configuraciones para usuarios específicos de forma que puedan consultarse posteriormente de forma privada. Sería interesante que un usuario pudiera registrarse para guardar sus propias simulaciones.

2.2.2. Herramienta web para el cálculo de escaños mediante la ley D'Hont

Esta herramienta está disponible en http://www.calculoescanos.com/ y cuenta con las siguientes funcionalidades:

Sistemas electorales

Aparentemente solo usa el método de divisores *D'hondt*, pero también se pueden ver los resultados dados por el método de cuota de *Droop* y el método de cuota de *Hare*, aunque se encuentran escondidos fuera de la página principal y solo permite la configuración de elecciones generales autonómicas y municipales españolas; Por ejemplo, la cuota mínima para obtener escaño.

Formas de entrada de datos

Solo permite la entrada de datos por formulario, tiene una utilidad para introducir los nombres de los partidos reales españoles.

Tiene guardadas varias elecciones obtenidas del periódico el País

Formato de salida de datos:

Permite exportar los datos en un archivo en formato Excel, pero al abrir el archivo el formato no coincide con la extensión y obtenemos una tabla en la que todos los números son considerados texto, como observamos en la Ilustración 4.

	A	B	C	D	E	F
1	**Partido**	**Votos**	**Pje**	**Escaños**	**Faltan**	
2	PP	2518	18.26 %	4	345	360
3	PSG-PSOE	2510	18.20 %	4	355	352
4	AMES NOVO	2474	17.94 %	4	390	316
5	CP	1744	12.65 %	3	548	123
6	P X A	1720	12.47 %	3	0	99
7	BNG	1426	10.34 %	2	294	346
8	C'S	1079	7.82 %	1	66	539
9	En blanco	319	2.31 %			
10	Nulos	291				
11	**Total**	13790		21		

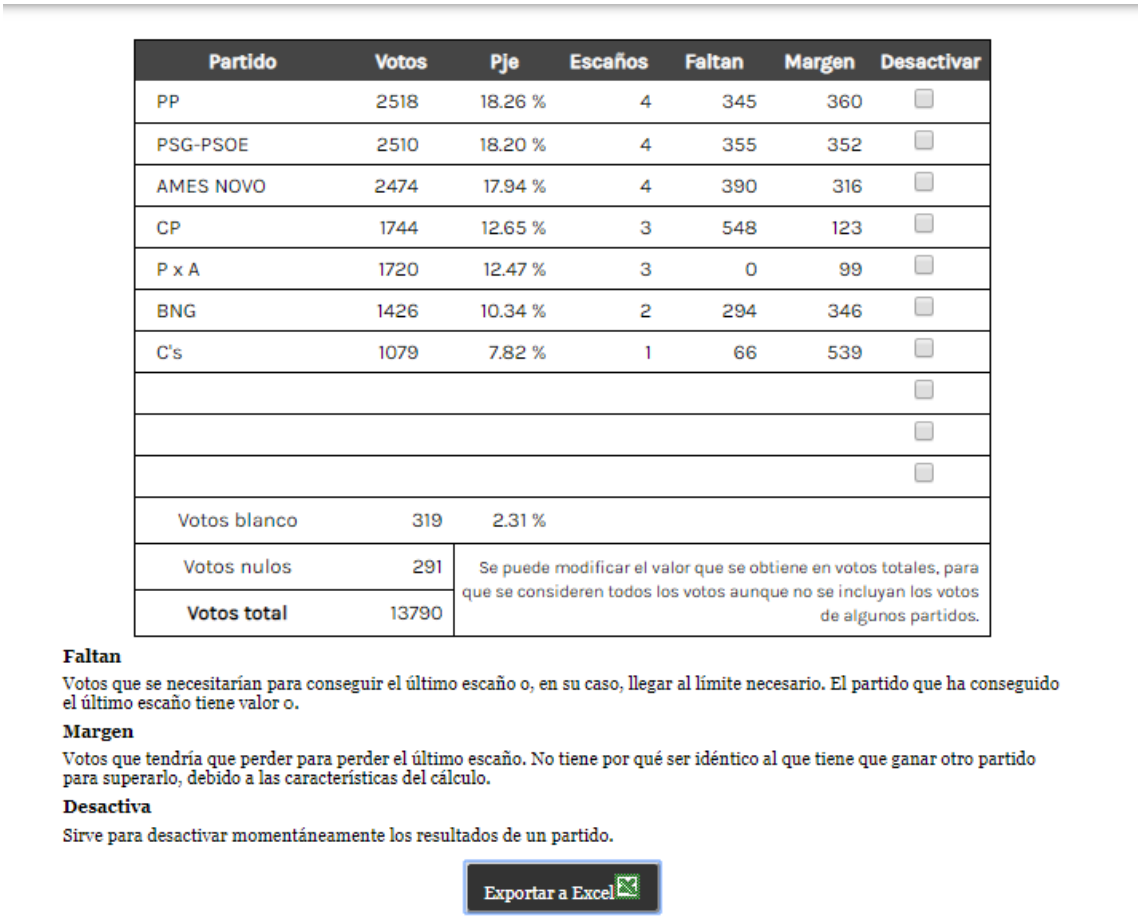
Ilustración 4: Herramienta web para el cálculo de escaños mediante la ley D'Hont - Tabla de datos exportador en Excel

Personalización:

El sistema no utiliza distintos usuarios por lo tanto a personalización es extremadamente limitada.

Visualización

Los resultados se muestran en una tabla con columnas extras cuyo significado es explicado, como observamos en la Ilustración 5



Partido	Votos	Pje	Escaños	Faltan	Margen	Desactivar
PP	2518	18.26 %	4	345	360	☐
PSG-PSOE	2510	18.20 %	4	355	352	☐
AMES NOVO	2474	17.94 %	4	390	316	☐
CP	1744	12.65 %	3	548	123	☐
P x A	1720	12.47 %	3	0	99	☐
BNG	1426	10.34 %	2	294	346	☐
C's	1079	7.82 %	1	66	539	☐
						☐
						☐
						☐
Votos blanco	319	2.31 %				
Votos nulos	291	Se puede modificar el valor que se obtiene en votos totales, para que se consideren todos los votos aunque no se incluyan los votos de algunos partidos.				
Votos total	13790					

Faltan

Votos que se necesitarían para conseguir el último escaño o, en su caso, llegar al límite necesario. El partido que ha conseguido el último escaño tiene valor 0.

Margen

Votos que tendría que perder para perder el último escaño. No tiene por qué ser idéntico al que tiene que ganar otro partido para superarlo, debido a las características del cálculo.

Desactiva

Sirve para desactivar momentáneamente los resultados de un partido.

Exportar a Excel

Ilustración 5: Herramienta web para el cálculo de escaños mediante la ley D'Hont - Ejemplo tabla de resultados

Más abajo en la página aparece una gráfica, como podemos observar en la Ilustración 6, pero no es obvio a priori que el sistema dibuje una gráfica y debido a esto hay que buscarla.

Ilustración 6: Herramienta web para el cálculo de escaños mediante la ley D'Hont - Ejemplo de gráfica poco atractiva

No es visualmente atractiva con los diferentes componentes colocados sin orden y con colores apagados.

Otros:

Tiene muchas páginas explicativas sobre la legislación española y los sistemas de reparto.

En la página principal hay varios componentes de publicidad.

Conclusiones

Todos los métodos que usa tendrían que estar claramente definidos desde el principio. A entrada de datos es demasiado tediosa sobre todo cuando usas la misma simulación varias veces y debes introducirla varias veces.

La tecnología utilizada se basa en JavaScript según su propia explicación y no utilizan ningún *framework* ni CSS que mejore su atracción visual y el orden de los componentes en la página.

Además, la aplicación no permite la posibilidad de almacenar las diferentes configuraciones para usuarios específicos de forma que puedan consultarse posteriormente de forma privada. Sería interesante que un usuario pudiera registrarse para guardar sus propias simulaciones.

2.2.3. Simulador electoral. Cálculo de resultados según el sistema D'Hondt

Esta herramienta está disponible en http://icon.cat/util/elecciones# y cuenta con las siguientes funcionalidades.

Sistemas electorales

Solo utiliza *D'hondt* en una circunscripción.

Formas de entrada de datos:

Solo se pueden introducir datos por un formulario. Los datos opcionales se introducen en huecos en blanco de un texto y los datos obligatorios se van añadiendo dinámicamente según la necesidad, como podemos observar en la Ilustración 7.

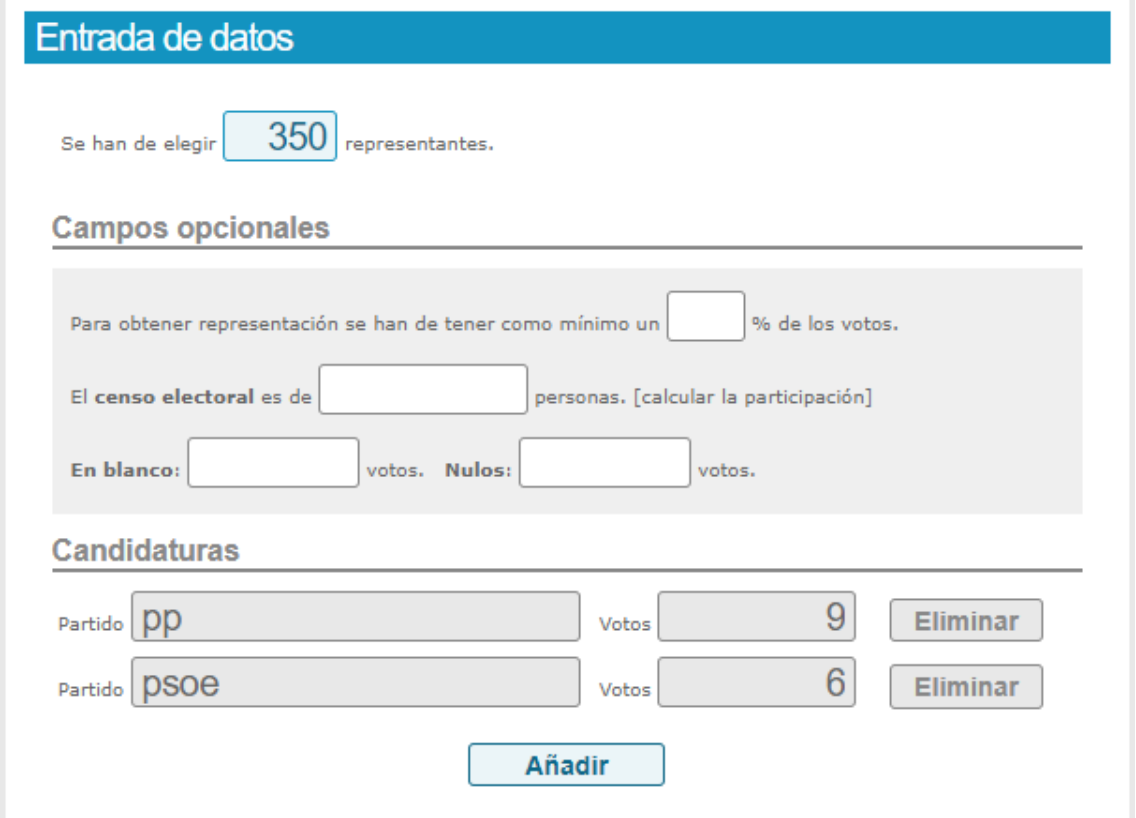


Ilustración 7: Simulador electoral. Cálculo de resultados según el sistema D'Hondt - Ejemplo de formulario de entrada de datos

Formato de salida de datos

No permite la exportación de datos.

Personalización

Es posible guardar la simulación. El procedimiento se basa en realizar un POST con los datos introducidos en el formulario y se recibe una URL donde revisar este resultado una vez perdidos los datos del formulario. La URL debe guardarse puesto que se pierde una vez cerrada la página.

Ilustración 8: Simulador electoral. Cálculo de resultados según el sistema D'Hondt - Ejemplo de enlace para guardar resultados

Tiene integrado el sistema de "*likes*" de Facebook.

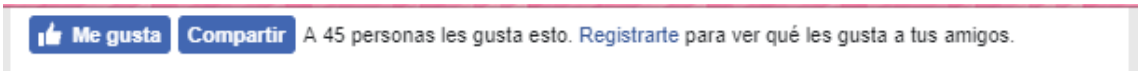


Ilustración 9: Simulador electoral. Cálculo de resultados según el sistema D'Hondt - Ejemplo de sistema de *likes*

Visualización:

Tiene un diseño fluido y atractivo, pero no muestra los resultados en formato de gráficas.

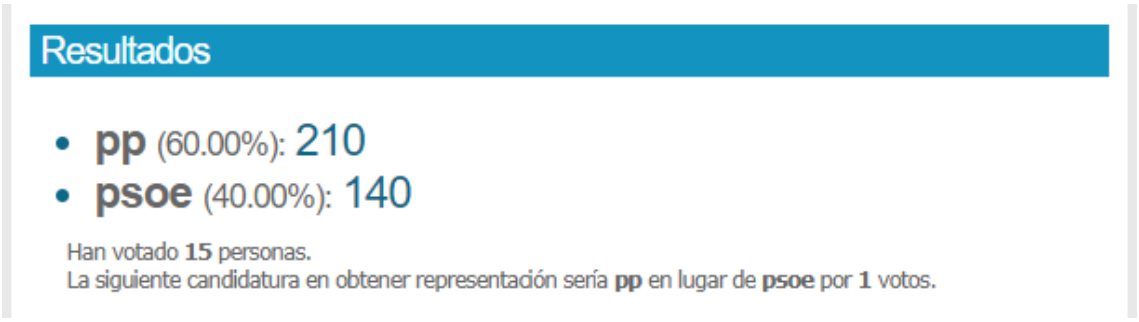


Ilustración 10: Simulador electoral. Cálculo de resultados según el sistema D'Hondt - Ejemplo de resultados sin gráficas

Otros:

Se puede realizar todo el proceso en tres idiomas: español por defecto, inglés y catalán (Hay que notar que es una página .cat).

En la página principal hay varios componentes de publicidad.

Conclusiones:

Tiene un alcance muy limitado al considerar sólo un sistema y en una única circunscripción.

La entrada de datos, aunque por formulario, es más ligera que en otros casos y sería deseable poder exportar los datos o visualizarlos en gráficas.

La personalización no es adecuada porque tener que recordar un identificador aleatorio no es la forma óptima de realizarla, sin embargo, esto permite compartir simulaciones entre usuarios de forma fácil y sencilla.

La lógica de negocio se realiza en JavaScript en tiempo real según se va rellenando el formulario.

2.3. Propuesta de solución

Una vez estudiadas las características más relevantes de este tipo de sistemas, pasamos a determinar cuáles serán los aspectos más importantes del sistema a desarrollar.

Para conseguir una interfaz dinámica y visualmente atractiva, que mejore la interacción y la experiencia de usuario, se ha optado por desarrollar una aplicación web, frente a por ejemplo una aplicación de escritorio, ya que de esta forma su difusión será mayor pudiendo llegar a muchas más personas. La aplicación web centrada en el cliente utilizará un framework JS que se conecte a su vez a un API remota para el intercambio de información con el servidor. Como framework front-end se ha seleccionado Angular. Para la parte back-end se usará un API REST implementada con el framework Spring.

Dicha aplicación web podría ser utilizada por diferentes tipos de clientes y estará basada en una arquitectura MVC (Modelo- Vista-Controlador), compuesta por un API REST que nos proporcionará un servicio, para acceder a la base de datos, realizada utilizando el *Framework Spring* y usando la tecnología JPA (*Java Persistence Api*)[2] para interactuar con la base de datos, la cual la utilizaremos para guardar las votaciones que vayan realizando los usuarios y los propios registros de usuarios. De esta misma forma, la API REST [6] es la encargada de realizar toda la funcionalidad de nuestra pequeña aplicación, ya que será la que nos proporcione los servicios necesarios para gestionar los datos. Por último, la vista, en la que se hará un cliente con Angular para que pueda comunicarse con el servidor y poder mostrar los datos por pantalla de diferentes formas.

Se han pensado, tanto esta arquitectura como algunas de las tecnologías que se emplearan para realizar la aplicación, porque han sido estudiadas durante el Grado facilitando así la obtención de resultados. Además, actualmente están en auge las tecnologías para realizar clientes ligeros, con lo que facilita, su integración en diversos entornos, como aplicaciones web y móviles.

[2] *Java Persistence Api*: Es una especificación para simplificar el intercambio de objetos sobre bases de datos relacionales

2.4. Tecnologías y *frameworks* usados

En este apartado se comentarán algunos de los aspectos principales de las tecnologías, *frameworks* y herramientas que se van a utilizar en el proyecto.

2.4.1. Spring Framework

Spring Framework [7] proporciona un completo modelo de programación y configuración para aplicaciones empresariales modernas basadas en Java, en cualquier tipo de plataforma de implementación.

Un elemento clave de *Spring* es el soporte de infraestructura a nivel de aplicación: *Spring* se enfoca en la "fontanería", es decir, en la interconexión de los diferentes componentes de la aplicación. De esta forma los equipos de desarrollo pueden enfocarse en la lógica de negocios a nivel de aplicación, sin vínculos innecesarios con entornos de implementación específicos.

- Tecnologías principales: inyección de dependencias, eventos, recursos, validación, enlace de datos, conversión de tipos, SpEL, AOP.
- Pruebas: objetos *mock, framework TestContext*, *Spring MVC* Test, *WebTestClient*.
- Acceso a datos: transacciones, soporte DAO, JDBC, ORM, *Marshalling* XML.
- *Spring MVC y Spring WebFlux web frameworks*.
- Integración: *remoting*, JMS, JCA, JMX, correo electrónico, tareas, programación, caché.

2.4.2. Spring Boot

Spring Boot[3] [8] es el punto de partida para construir todas las aplicaciones basadas en *Spring*. *Spring Boot* está diseñado para ponerlo en marcha lo más rápido posible, con una mínima configuración inicial de *Spring*.

[3] *Spring Boot:* https://spring.io/projects/spring-boot

- Crear aplicaciones Spring independientes
- Crear diferentes cosas: REST API, *WebSocket, web, streaming*, tareas y mucho más.
- Seguridad simplificada
- Soporte enriquecido para SQL y NoSQL
- Soporte de tiempo de ejecución integrado: *Tomcat, Jetty y Undertow*
- Herramientas de productividad para desarrolladores como *LiveReload* y Auto Restart
- Proporciona funciones listas para la producción, como métricas, comprobaciones de estado y configuración externalizada.
- Funciona en múltiples IDE*: Spring Tool Suite, IntelliJ IDEA y NetBeans*

2.4.3. IntelliJ IDEA

Este será el IDE que se usará para la programación de la parte del servidor del proyecto, *IntelliJ IDEA*[4]. Se ha optado por utilizar este IDE porque muchas empresas lo usan, en concreto en la empresa que trabajo actualmente se usa en algunos proyectos importantes, y quería familiarizarme con él, además de que Android Studio, el IDE que ofrece Google para crear aplicaciones para Android, está basado en él.

> *IntelliJ IDEA es un entorno de desarrollo integrado(IDE) para el desarrollo de programas informáticos. Es desarrollado por JetBrains (anteriormente conocido como IntelliJ), y está disponible en dos ediciones: edición comunity y edición comercial. IntelliJ IDEA no está basada en Eclipse como MyEclipse u Oracle Enterprise Pack para Eclipse. [9]*

Este IDE cambia un poco la filosofía a lo que estamos acostumbrados en Eclipse o *Netbeans*, ya no se habla de *workspaces*, sino que se habla de proyectos; y lo que antes eran proyectos ahora pasan a ser módulos. Realiza un auto-completado inteligente del código ya que analiza el código de forma estática y mantiene su estructura en memoria mientras trabajas con él, también ofrece soluciones inteligentes a los problemas e incluye una refactorización muy buena.

[4] IntelliJ IDEA: https://www.jetbrains.com/idea/

2.4.4. SonarQube

IntelliJ es capaz de evaluar el código fuente con muy buenos resultados, pero también se ha optado por usar *SonarQube*[5] ya que también es ampliamente utilizado y quería familiarizarme con él.

> *SonarQube (conocido anteriormente como Sonar) es una plataforma para evaluar código fuente. Es software libre y usa diversas herramientas de análisis estático de código fuente como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa. [10]*

Puede realizar análisis en códigos escritos en numerosos lenguajes de programación y proponer una gran cantidad de mejoras para obtener un código de calidad.

2.4.5. Java Persistence API (JPA)

Es un *framework* del lenguaje de programación Java que hace posible establecer una correspondencia entre bases de datos relacionales y un sistema orientado a objetos.

> *El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos (siguiendo el patrón de mapeo objeto-relacional), como sí pasaba con EJB2, y permitir usar objetos regulares (conocidos como POJOs). [11]*

2.4.6. Java SE 8

Se han usado varias características que se incorporaron en la versión 8 de Java, aquí se mencionan algunas de ellas:

> *Las características más importantes de Java SE 8 son la adición de Expresiones Lambda y la API Stream. Con la adición de expresiones lambda podemos crear código más conciso y significativo, además de abrir la puerta hacia la programación funcional en Java, en donde las funciones juegan un papel fundamental. Por otro lado, la API Stream nos permite realizar operaciones de tipo filtro/mapeo/reducción sobre colecciones de datos de forma secuencial o paralela y que su implementación sea transparente para*

[5] *SonarQube:* https://www.sonarqube.org/

el desarrollador. Lambdas y Stream son una combinación muy poderosa que requiere un cambio de paradigma en la forma en la que hemos escrito código Java hasta el momento. [12]

2.4.7. MySQL

Es una de las bases de datos de código abierto más populares, esto se debe a su gran rendimiento y fiabilidad, además de su facilidad de uso.

MySQL[6] es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base datos de código abierto más popular del mundo,12 y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web. [13]

2.4.8. Software de terceros para el cálculo de reparto de escaños

Como comentamos en el apartado de 2.1 Análisis preliminar, a la hora de convertir los votos en escaños, no se puede realizar directamente y por eso se aplican varios métodos matemáticos. Las dos grandes familias de métodos de reparto son los métodos de Divisor y los métodos de *Quotas.*

La librería aportada por mi compañero Daniel De La Concepción, tiene varias funcionalidades y sigue creciendo actualmente, en este caso, se ha tomado una parte encargada de calcular los escaños a partir de los votos, en base a varios de los métodos de estas dos familias.

La aplicación está realizada en Java y se encuentra en GitHub, con licencia GNU *General Public License* v3.0 y accesible desde: https://github.com/dan323/electoral-system-sim.git. Ha sido integrada en este proyecto como una librería externa, para ello se ha descargado de GitHub el código fuente, se ha compilado localmente e instalado en el repositorio local de *maven*.

[6] MySQL: https://www.mysql.com/

2.5. Historias de Usuario

Una forma habitual de plasmar los requisitos obtenidos de las necesidades de los usuarios son las historias de usuario [14]. Describen una tarea concisa de manera simple de forma que la entienda tanto el cliente como el equipo de desarrollo y pueden evolucionar durante el proyecto. Se componen de varias partes:

- Id: Identificador para la historia de usuario
- Título: Título para la historia de usuario
- Descripción: Descripción de la historia de usuario, se suele usar una plantilla:
    - Como <rol>
    - Quiero <característica>
    - Para <obtener un valor de negocio>
- Estimación: Estimación en tiempo teórico para el desarrollo de la historia de usuario
- Prioridad: Numero que representa la prioridad de una historia de usuario frente a otras.
- Dependencias: Aunque unas historias de usuario no deberían depender de otras, aquí se indican los identificadores de las historias de usuario de las que depende. Para comodidad en la revisión del documento, se han añadido enlaces para consultar rápidamente las dependencias de cada historia.
- Criterios de aceptación: Son una serie de pruebas que se tienen que realizar para decir que la historia de usuario se ha completado con éxito. Incluyen:
    - Funcionalidades del sistema
    - Aspecto de la interfaz y funcionamiento
    - Documentación

Para la estimación inicial de las historias de usuario se ha escogido una medida relativa, los puntos de historia, estos expresan el tamaño de un requerimiento comparado con la duración necesaria para completarlo. Para ello usaré una escala creciente muy utilizada: 1, 2, 3, 5, 8, (13, 20, 40, 100).

2.5.1. Introducir partidos y votos

Descripción:

Como Usuario quiero introducir una simulación o una votación real a la aplicación para poder calcular las soluciones de reparto correspondientes con diferentes métodos más adelante.

Aceptación:

- Si un partido se repite, no se aceptará ese dato
- Si los votos son negativos, no se aceptará ese dato
- Si no ocurre nada de lo anterior, comprobar que los datos en la aplicación coinciden con los introducidos por el Usuario

Estimación:

2

2.5.2. Seleccionar método de reparto

Descripción:

Como Usuario quiero seleccionar un tipo de método de reparto para poder calcular las soluciones correspondientes con distintas simulaciones o votaciones reales más adelante.

Aceptación:

- Comprobar que la lista de métodos de reparto seleccionables es la lista de métodos de reparto implementados en el sistema.

Estimación:

5

2.5.3. Alta usuario

Descripción:

Como Usuario quiero registrarme en el sistema para poder guardar mis datos para su uso futuro.

Aceptación:

- Si el usuario se ha registrado antes, se obtendrá un error.
- Si la contraseña no contiene al menos una cifra y una letra, se obtendrá un error y comprobar que no ha tenido lugar el registro.
- Si el usuario es nuevo y la contraseña contiene cifras y letras, comprobar que el registro se ha realizado.

Estimación:

5

2.5.4. Baja usuario

Descripción:

Como Usuario quiero darme de baja en el sistema para que mis datos ya no sean guardados.

Aceptación:

- Si el usuario existe, se le eliminará, junto con todos sus datos, del sistema.
- Si el usuario no existe en el sistema, se obtendrá un error.

Estimación:

3

Dependencia:

2.4.3 Alta usuario

2.5.5. Modificación contraseña

Descripción:

Como Usuario quiero modificar mi contraseña para tener más seguridad.

Aceptación:

- Si el usuario el usuario no está registrado, se lanzará un error.
- Si está registrado, comprobar que la contraseña se ha modificado.

Estimación:

2

Dependencia:

2.4.3 Alta usuario

2.5.6. Crear votaciones

Descripción:

Como Usuario quiero guardar una simulación o votación real para su uso futuro.

Aceptación:

- Si el Usuario está registrado y *logeado*, comprobar que la simulación o votación real existe en el sistema.
- Si el Usuario no está registrado, se lanzará un mensaje de error.
- Si el Usuario no está *logeado*, se lanzará un mensaje de error.
- Si la simulación o votación real existe en el sistema, se lanzará un error.

Estimación:

5

Dependencia:

2.4.3 Alta usuario

2.5.7. Borrar votaciones

Descripción:

Como Usuario quiero borrar una simulación o votación real que he guardado con anterioridad para mantener mis datos actualizados.

Aceptación:

- Si la simulación o votación real ha sido guardada por este usuario con anterioridad, comprobar si ha sido eliminada.
- Si la simulación o votación real no ha sido guardada por este usuario con anterioridad, se lanzará un error.

Estimación:

2

Dependencia:

2.4.3 Alta usuario

2.4.6 Crear votaciones

2.5.8. Modificación de una simulación o votación real

Descripción:

Como Usuario quiero modificar una de mis votaciones para tener mis datos actualizados.

Aceptación:

- Si la simulación o votación real no ha sido guardada por este usuario registrado y logueado, se lanzará un error.
- Si el usuario no está registrado, se lanzará un error
- En otro caso, comprobar que la simulación o votación real ha sido modificada.

Estimación:

3

Dependencia:

2.4.3 Alta usuario

2.4.6 Crear votaciones

2.5.9. Calcular el reparto de escaños

Descripción:

Como Usuario quiero calcular reparto de escaños

Aceptación:

- Si no hay una simulación o votación real seleccionada, se lanzará un error.
- Si no hay un método de reparto seleccionado, se lanzará un error.
- Si hay una simulación o votación real seleccionada y un método de reparto seleccionado, comprobar que el reparto dado por el sistema es correcto.

Estimación:

13

Dependencia:

2.4.1 Introducir partidos y votos

2.4.2 Seleccionar Formula electoral

2.5.10. Visualización atractiva de los datos.

Descripción:

Como Usuario quiero visualizar mis datos con tablas y gráficas de forma atractiva y sin perder funcionalidad para ver de formas más clara mis datos.

Aceptación:

- Comprobar que las tablas y gráficas corresponden con los datos originales.

Estimación:

20

Dependencia:

2.4.9 Calcular el reparto de escaños

2.6. Planificación inicial de tareas

En este apartado vamos a realizar una estimación inicial del tiempo y costes del proyecto para conocer su viabilidad. Para esto hablaré de tareas y de los roles encargados de realizarlas

En este caso al trabajar solo yo, tomaré dos roles, si hablamos del rol analista será el encargado de realizar el estudio de mercado, la panificación inicial y todas las tareas de diseño y documentación. Por otro lado, el rol de programador lleva a cabo el resto de tareas de implementación y pruebas.

2.6.1. Estimación inicial de las tareas

En este apartado veremos una estimación del desglose en tareas de cada historia de usuario con sus puntos de historia y el rol encargado de realizarla.

H.U	Tareas	P. historia	Roles
	Estudio de soluciones existentes	13	Analista
	Planificación inicial	13	Analista
	Control de entrada de datos	12	-
	Diseño	1	Analista
2.5.1	**Introducir partidos y votos**	**2**	**Programador**
2.5.2	**Seleccionar método de reparto**	**5**	**Programador**

	Test unitarios	1	Programador
	Documentación	2	Analista
	Manejo de usuarios	18	-
	Diseño	2	Analista
2.5.3	**Alta de usuario**	**5**	**Programador**
2.5.4	**Baja de usuario**	**3**	**Programador**
2.5.5	**Modificación de usuario**	**2**	**Programador**
	Test unitarios	2	Programador
	Documentación	3	Analista
	Manejo de votaciones	13	-
	Diseño	2	Analista
2.5.6	**Crear votaciones**	**4**	**Programador**
2.5.7	**Borrar votaciones**	**2**	**Programador**
2.5.8	**Modificación de una votación**	**2**	**Programador**
	Test unitarios	2	Programador
	Documentación	2	Analista
2.5.9	**Calcular el reparto de escaños**	**13**	**-**
	Estudio métodos de reparto	5	Analista
	Diseño	1	Analista
	Implementación capa REST	5	Programador
	Test unitarios	1	Programador
	Documentación	1	Analista
	Integración módulos	1	Programador
2.5.10	**Visualización atractiva de los datos**	**20**	**-**
	Estudio Angular	5	Analista
	Implementación Servicios	3	Programador
	Diseño de la interfaz	2	Analista
	Implementación de la interfaz	3	Programador
	Navegación de la pagina	1	Programador
	Test	2	Programador
	Documentación	3	Analista
	Total	**103**	**-**

Tabla 3: Estimación inicial de tareas y esfuerzo necesario

Teniendo en cuenta que el tiempo de trabajo de la asignatura de Trabajos Fin de Grado es de unas 300.0 horas de trabajo del alumno (12 créditos ECTS), o 4-6 meses de desarrollo, podríamos hacer la equivalencia de 1 punto de historia 3 horas de trabajo, lo que daría 309 horas de trabajo.

2.6.2. Priorización de tareas

A continuación, se utilizará el enfoque *MoSCoW* [15], que es una técnica de priorización de las tareas, que ayuda a todo el equipo a saber que es importante para el negocio y ayuda a obtener el producto mínimo aceptable.

- *Must have* (es necesario):
  - Estudio de soluciones existentes (13 Pts. historia)
  - Planificación inicial (13 Pts. historia)
  - Introducir partidos y votos (2 Pts. historia)
  - Seleccionar método de reparto (5 Pts. historia)
  - Alta de usuario (5 Pts. historia)
  - Baja de usuario (3 Pts. historia)
  - Crear votaciones (4 Pts. historia)
  - Borrar votaciones (2 Pts. historia)
  - Estudio métodos de reparto (5 Pts. historia)
  - Implementación capa REST (5 Pts. historia)
  - Integración módulos (1 Pts. historia)
  - Estudio Angular (5 Pts. historia)
  - Implementación Servicios (3 Pts. historia)
  - Diseño de la interfaz (2 Pts. historia)
  - Implementación de la interfaz (3 Pts. historia)
  - Navegación de la página (1 Pts. historia)
- *Should have* (es recomendable)
  - Modificación de usuario (2 Pts. historia)
  - Modificación de una votación (2 Pts. historia)
- *Could have* (podría implementarse)
  - Seguridad de Spring

- *Won't have* (no lo queremos... quizá en un futuro)
    - No se ha considerado ninguna tarea de este tipo

2.6.3. Iteraciones

En este apartado vamos a realizar una estimación inicial de las tareas que se acometerán en cada iteración para determinar el alcance, duración y esfuerzo a desarrollar en cada una.

Se ha determinado que en este caso las iteraciones serán de dos semanas y que realizaremos unos 20 puntos de historia por iteración, por lo tanto, eso da lugar a 5,15 iteraciones, pero en la priorización hemos determinado algunas tareas como *Should have* (es recomendable) que dejaremos para una posible 6 iteración.

Iteración 0 (2 Semanas)

- Estudio de soluciones existentes (13 Pts. historia)
- Planificación inicial 1/2 (7 Pts. historia)

En total son 20 puntos de historia.

Iteración 1 (2 Semanas)

- Planificación inicial 2/2 (6 Pts. historia)
- Diseño (1 Pts. historia)
- Introducir partidos y votos (2 Pts. historia)
- Seleccionar método de reparto (5 Pts. historia)
- Test unitarios (1 Pts. historia)
- Documentación (2 Pts. historia)
- Diseño de manejo de usuarios (2 Pts. historia)

En total son 19 puntos de historia.

Iteración 2 (2 Semanas)

- Alta de usuario (5 Pts. historia)
- Baja de usuario (3 Pts. historia)
- Test unitarios (2 Pts. historia)
- Documentación (3 Pts. historia)
- Diseño de manejo de votaciones (2 Pts. historia)
- Crear votaciones (4 Pts. historia)

En total son 19 puntos de historia.

Iteración 3 (2 Semanas)

- Borrar votaciones (2 Pts. historia)
- Test unitarios (2 Pts. historia)
- Documentación (2 Pts. historia)
- Estudio métodos de reparto (5 Pts. historia)
- Diseño (1 Pts. historia)
- Implementación capa REST (5 Pts. historia)
- Test unitarios (1 Pts. historia)
- Documentación (1 Pts. historia)
- Integración módulos (1 Pts. historia)

En total son 20 puntos de historia.

Iteración 4 (2 Semanas)

- Estudio Angular (5 Pts. historia)
- Implementación Servicios (3 Pts. historia)
- Diseño de la interfaz (2 Pts. historia)
- Implementación de la interfaz (3 Pts. historia)
- Navegación de la página (1 Pts. historia)
- Test (2 Pts. historia)
- Documentación (3 Pts. historia)

En total son 19 puntos de historia.

Iteración 5 (2 Semanas)

- Modificación de usuario (2 Pts. historia)
- Modificación de una votación (2 Pts. historia)

2.6.4. Cambios en las estimaciones

En general, las estimaciones iniciales suelen sufrir variaciones a lo largo del proyecto debido a imprevistos o incluso a falta de experiencia en el proceso. Por este motivo, en este apartado se mostrará un gráfico *burn-down* donde se podrá ver cómo fue avanzando el proyecto en las iteraciones y se mencionará un problema ocurrido.

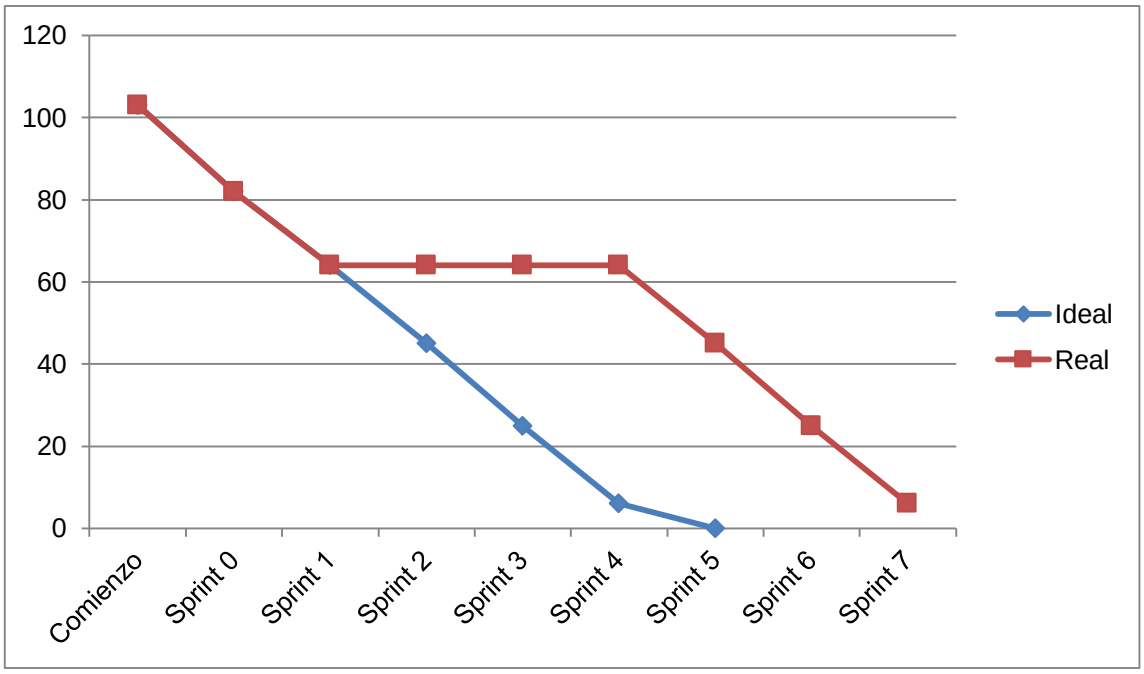


Ilustración 11: Comparativa de diagramas de *burn-down* estimado y real

Como podemos observar en el grafico el avance empezó siendo correcto hasta el sprint 2 en el que debido a una baja médica real se interrumpió el avance durante varias semanas, por lo que hubo que retrasar el tiempo de entrega, esto lo reflejamos en el gráfico añadiendo *sprints* en los que no se avanzaba, después de esto el avance continuó con lo esperado logrando cumplir con las tareas que priorizamos como necesarias (*Must have*) pero teniendo que dejar sin realizar las tareas priorizadas como recomendables (*Should have*).

2.7. Estudio de viabilidad

En este apartado vamos a ampliar la Tabla 3 de la planificación inicial con el coste por rol encargado de las tareas y con una estimación inicial del precio por tarea y del coste total. Para ello basándome en el convenio colectivo de Telefónica de 2018/2019 [16], se tomará como sueldo base de los Analistas 10€ la hora y para los programadores 9€ la hora.

H.U	Tareas	P.historia	Hrs	Roles	Sueldo	Coste/Tarea
	Estudio de soluciones existentes	13	40	Analista	10,00 €	400,00 €
	Planificación inicial	13	40	Analista	10,00 €	400,00 €
	Control de entrada de datos	12	36	-	-	331,80 €
	Diseño	1	4	Analista	10,00 €	42,00 €
2.5.1	**Introducir partidos y votos**	**2**	**6**	**Programador**	**9,00 €**	**54,00 €**
2.5.2	**Seleccionar método de reparto**	**5**	**15**	**Programador**	**9,00 €**	**135,00 €**
	Test unitarios	1	4	Programador	9,00 €	37,80 €
	Documentación	2	6	Analista	10,00 €	63,00 €
	Manejo de usuarios	18	53	-	-	489,80 €
	Diseño	2	6	Analista	10,00 €	62,00 €
2.5.3	**Alta de usuario**	**5**	**15**	**Programador**	**9,00 €**	**135,00 €**
2.5.4	**Baja de usuario**	**3**	**10**	**Programador**	**9,00 €**	**90,00 €**
2.5.5	**Modificación de contraseña**	**2**	**6**	**Programador**	**9,00 €**	**54,00 €**
	Test unitarios	2	6	Programador	9,00 €	55,80 €
	Documentación	3	9	Analista	10,00 €	93,00 €
	Manejo de datos guardados	13	39	-	-	426,60 €
	Diseño	2	5	Analista	10,00 €	54,00 €
2.5.6	**Crear votaciones**	**4**	**12**	**Programador**	**9,00 €**	**135,00 €**
2.5.7	**Borrar votaciones**	**2**	**5**	**Programador**	**9,00 €**	**54,00 €**
2.5.8	**Modificación de una votación**	**2**	**6**	**Programador**	**9,00 €**	**54,00 €**
	Test unitarios	2	5	Programador	9,00 €	48,60 €

	Documentación	2	7	Analista	10,00 €	81,00 €
2.5.9	**Calcular el reparto de escaños**	**13**	**40**	**-**	**-**	**286,40 €**
	Estudio métodos de reparto	5	16	Analista	10,00 €	160,00 €
	Diseño	1	3	Analista	10,00 €	16,00 €
	Implementación capa REST	5	14	Programador	9,00 €	72,00 €
	Test unitarios	1	3	Programador	9,00 €	14,40 €
	Documentación	1	4	Analista	10,00 €	24,00 €
	Integración módulos	1	2	Programador	9,00 €	18,00 €
2.5.10	**Visualización atractiva de los datos**	**20**	**60**	**-**	**-**	**591,80 €**
	Estudio Angular	5	16	Analista	10,00 €	160,00 €
	Implementación Servicios	3	10	Programador	9,00 €	108,00 €
	Diseño de la interfaz	2	6	Analista	10,00 €	40,00 €
	Implementación de la interfaz	3	10	Programador	9,00 €	108,00 €
	Navegación de la pagina	1	3	Programador	9,00 €	27,00 €
	Test	2	6	Programador	9,00 €	55,80 €
	Documentación	3	9	Analista	10,00 €	93,00 €
	Total	**103**	**309**	**-**	**-**	**2.944,40 €**

Tabla 4: Gastos de humanos

Seguidamente, se verán los gastos software

Software	Licencia	Costes
MySQL	Community Edition (GPL)	0 €
IntelliJ IDEA	licencia para estudiantes de la Universidad	0 €
Visual Studio Code	Licencia MIT	0 €
Word	licencia para estudiantes de la Universidad	0 €
Visual Paradigm	licencia para estudiantes de la Universidad	0 €
Total	**-**	**0 €**

Tabla 5: Gastos Software

Y por último tendremos otros gastos materiales:

Gastos materiales	Costes
Lenovo V130 Intel Core i5-7200U/4GB/500GB/15.6"	394,59 €

Yoigo - Fibra Óptica simétrica de 100 Mb.	32,00 € x 4 meses
Total	**522,59 €**

Tabla 6: Gastos materiales

Quedando finalmente unos costes totales de 3.4466,99 €

2.8. Modelo de dominio

Es un modelo conceptual de todos los temas relacionados con un problema, en el que se describen las entidades, atributos y relaciones, además de las restricciones que conciernen al dominio del problema.

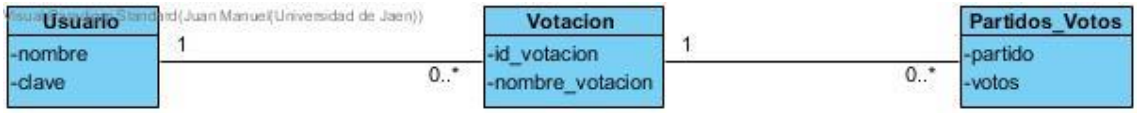


Ilustración 12: Modelo de dominio

En el dominio de nuestro problema se ha observado que mediante un modelo de dominio simple se pueden abarcar todas nuestras necesidades. Por ello se ha optado, por la utilización de un modelo sencillo, que consta de tres clases conceptuales, en nuestro caso tenemos:

- Usuario: cualquier persona que haga uso de nuestra aplicación y que podrá tener cero o más votaciones.
- Votación: contendrá un conjunto de partidos políticos con sus respectivos votos y sirve para diferenciar un conjunto de partidos votos de otros mediante un nombre y un id. Por ejemplo, las elecciones de 2019 y las elecciones de 2016
- Partidos_Votos: corresponde a un partido político y el número de votos de dicho partido.

2.9. Análisis de la interfaz

Los requisitos de la interfaz no son especialmente complejos. Se seguirá un diseño *Responsive*[7] para la interfaz, adaptándose así a las características del dispositivo que la visualice. También se intentarán usar controles de uso sencillo, a ser posibles optimizados para uso táctil.

En la aplicación tendremos en cuenta dos tipos de usuarios, los usuarios registrados y los usuarios no registrados.

Los usuarios registrados podrán guardar sus votaciones y tener acceso a las mismas para calcular los escaños cuando quieran o modificarlas. Dichos usuarios registrados tendrán acceso a su perfil con sus datos almacenados, pudiendo consultarlos, modificarlo o borrarlos cuando quieran e incluso darse de baja.

Los usuarios no registrados solo podrán realizar simulaciones introduciendo los datos para calcular escaños sin posibilidad de almacenarlos para su uso posterior.

[7] Diseño web *responsive*: es una técnica de diseño y desarrollo que busca adaptar una misma página a distintos dispositivos para su correcta visualización

3. DISEÑO

Mientras que el análisis estudia el sistema desde el punto de vista del dominio del problema, el diseño lo hace desde el punto de vista del dominio de la solución centrándose en el "cómo". Pero actualmente con las metodologías agiles no hay una separación entre separación en el tiempo entre análisis y diseño, ya que ambas tareas se suelen llevar a cabo de forma simultánea en cada iteración.

El modelo de diseño proporciona detalles sobre arquitectura del software, estructura de datos, interfaces y componentes que se necesitan para implementar el sistema [2].

En este apartado veremos una serie de diagramas como el de entidad-relación, el de clases y el de secuencia. Además, veremos el diseño de la interfaz y del API REST y por ultimo definiremos un plan de pruebas.

3.1. Diagrama Entidad-Relación

Como se explicó en el apartado de tecnologías empleadas, para el acceso a la BD se utilizaría JPA [17], que es un API para trabajar con ORMs (*object-relational mapping*) que facilita la persistencia de aplicaciones orientadas a objetos en bases de datos relacionales. En el siguiente diagrama observamos las entidades y relaciones que JPA ha generado:

Ilustración 13: Diagrama Entidad-Relación

En el diagrama observamos que aparecen 3 tablas:

- Usuario
- Votacion
- Partidos_Votos

Pero solo tengo dos clases “Usuario” y “Votación” que materializan los objetos de las clases Usuario y Votación además de la relación entre ambas. La tabla Usuario tiene el atributo nombre como clave primaria y la tabla Votación tiene un id_votacion como clave primaria, además también tiene como clave foránea el nombre de usuario.

La tabla “Partidos_Votos” se genera a partir del atributo tipo “map” que contiene la clase Votación. Esta tabla tiene una clave primaria compuesta, que consta de la clave foránea Id_votacion y de su atributo Partido. Por ello, el borrado de la tabla Partidos_Votos se hará en cascada si una votación es borrada. Será explicado con más detalle en el apartado de implementación.

La normalización consiste en imponer a las tablas ciertas restricciones, para asegurar que éstas solo contienen los atributos necesarios. Se pueden aplicar varias formas normales a las tablas, pero con las tres primeras suele ser suficiente para la mayoría de bases de datos.

- Una tabla está en primera forma normal (FN1) si todos los atributos que no pertenecen a la clave, dependen funcionalmente de la clave.
- Una tabla está en segunda forma normal (FN2) si está en FN1 y además todos los atributos que no pertenecen a la clave dependen funcionalmente de forma completa de ella.
- Una tabla está en tercera forma normal o FN3 si está en FN2 y además no existen atributos que no pertenecen a la clave que dependen transitivamente de la clave.

Se puede apreciar que en este caso las tablas están en FN3 ya que no existen grupos repetitivos para un valor de clave (FN1), tampoco existen grupos repetitivos para un valor de clave o un subconjunto ésta (FN2) y por ultimo no existen atributos que no pertenecen a la clave que tengan dependencias funcionales entre sí.

3.2. Diagrama de Clases

Los diagramas de clases aportan una visión estática o estructural de un sistema sin mostrarnos las comunicaciones dinámicas entre los objetos de las clases. [2]

En el diseño se obtiene el modelo de clases de diseño añadiendo detalles al modelo conceptual de clases, para ello refinaremos el diagrama de clases obtenido en la fase de análisis obteniendo así el diagrama de clases de diseño.

La arquitectura cliente-servidor es un modelo de aplicación distribuida, que consiste en repartir las tareas entre los servidores, lo cuales proveen servicios, y los clientes, los cuales demandan dichos servicios.

En este apartado se presentará el diagrama de clases de diseño de cada parte por separado, por un lado, para el API REST (Servidor), y por otro lado para la aplicación web Angular (Cliente).

3.2.1. Servidor

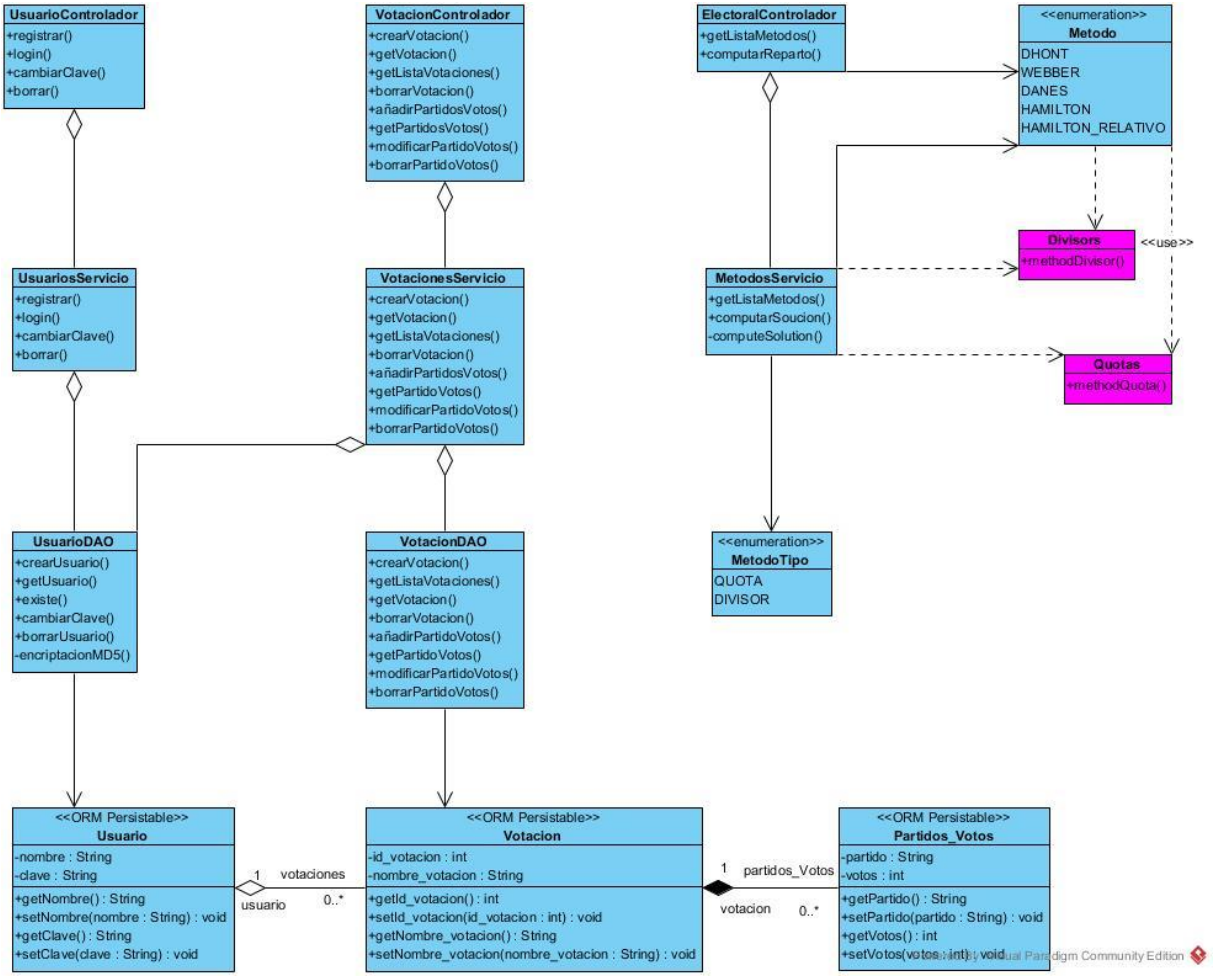


Ilustración 14: Diagrama de Clases del Servidor

En el diagrama de clases podemos observar que un usuario puede tener varias votaciones y que una votación podría tener varios partidos con sus respectivos votos.

En la implementación se han creado una clase “Usuario” y una clase “Votación”. A partir de estas, se ha hecho que la clase “Votación” tenga un atributo de tipo “Usuario” y tenga un mapa que contiene pares de nombres de partidos y sus correspondientes votos.

Además, habrá tres controladores, uno para usuario, otro para votaciones y otro para las elecciones. Cada uno de estos controladores usará un servicio diferente, habiendo tres de estos también.

El servicio de usuario hará uso del DAO correspondiente a usuario que será el que maneje el acceso a la base de datos de usuarios y el servicio de votaciones hará uso del DAO correspondiente a votaciones que llevará a cabo la misma tarea para las votaciones, estos DAOs harán uso de JPA para la gestión de la persistencia de datos

Por otro lado, el servicio de elecciones hará uso directamente de la librería de terceros para hacer el reparto de escaños, como se puede observar en el diagrama las clases de color morado.

En el diagrama de clases parece que la parte del controlador *electoral* no tiene relación con el resto, pero esto se debe a que es el cliente el que interactúa con el controlador *electoral* mandándole los datos necesarios para realizar el cálculo. Estos datos que el cliente le envía al controlador electoral podrá haberlos tomado previamente de los otros controladores.

3.2.2. Cliente

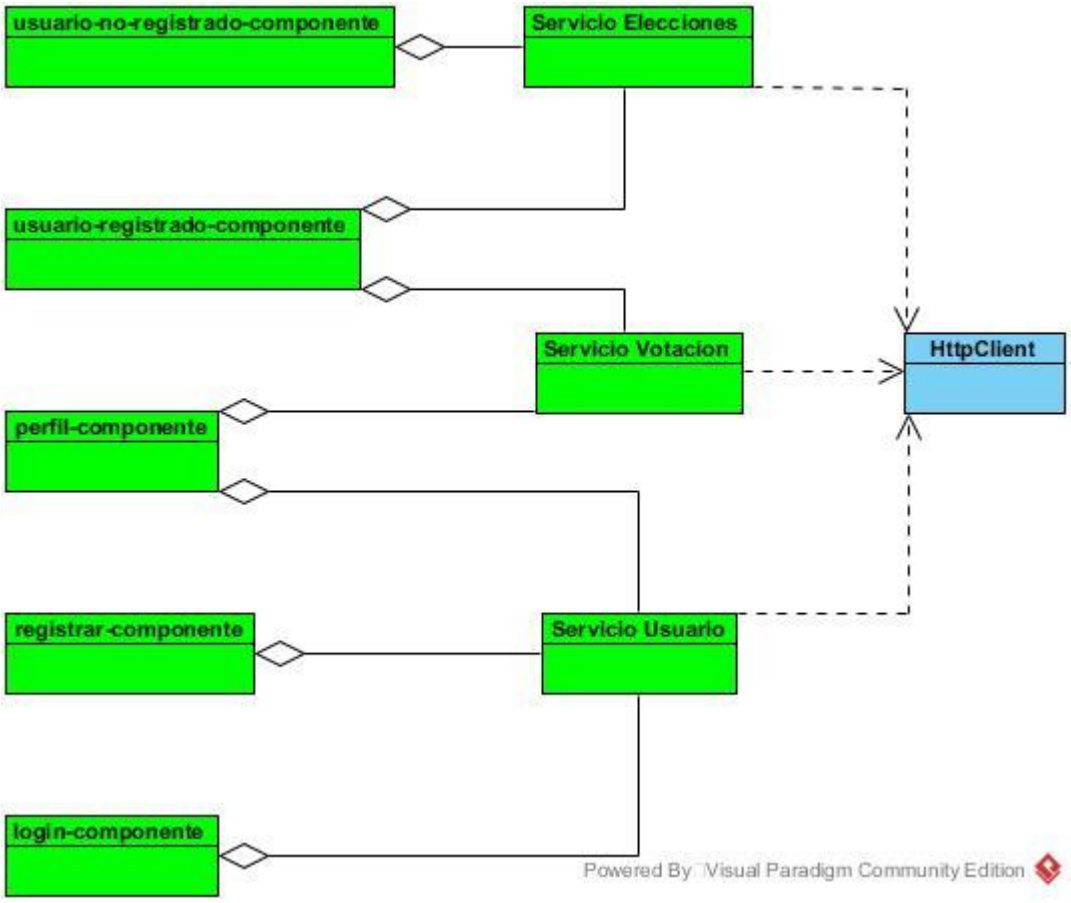


Ilustración 15: Diagrama de clases del cliente

El cliente estará compuesto por componentes y servicios, tendrá 3 servicios que serán los encargados de la comunicación REST con los controladores del servidor, teniendo así acceso a los datos y las funciones del servidor.

Además de los servicios estará compuesto por componentes, que son los bloques de construcción mínimos de una interfaz de usuario en Angular. Aunque estará formada por numerosos componentes, en este diagrama solo se muestran los principales, como los encargados del *login* y del registro, además de dos para los usuarios registrados y uno para los no registrados.

En el diagrama se puede observar que los componentes agregan a los servicios que necesiten para consultar datos del servidor, pero la multiplicidad es de uno, solo agregan un servicio de cada tipo.

3.3. Diagramas de secuencia

Los diagramas de secuencia muestran una interacción entre objetos organizados en una secuencia de tiempo. A diferencia de los diagramas de clases que se encargan de mostrar la estructura estática, los diagramas de secuencia, se usan para mostrar las comunicaciones dinámicas entre objetos durante la ejecución de una tarea. Este tipo de diagrama muestra el orden temporal en el que los mensajes se envían entre los objetos para lograr dicha tarea. Además, pueden tener distintos niveles de detalle. [2]

En este apartado se mostrarán los más representativos para comprender cómo funciona internamente el diseño propuesto en el apartado anterior para las historias de usuario más relevantes.

3.3.1. Diagrama de secuencia Usuario registrado

Se comenzará explicando un diagrama de secuencia en el que se verán las acciones a llevar a cabo para loguearse en la página para un usuario registrado y las comunicaciones entre los objetos involucrados en esta tarea.

Se diferenciará entre el cliente y el servidor para poder observar bien los objetos de cada parte, el cliente estará reflejado de color verde mientras que el servidor será de color azul como veremos en los diagramas.

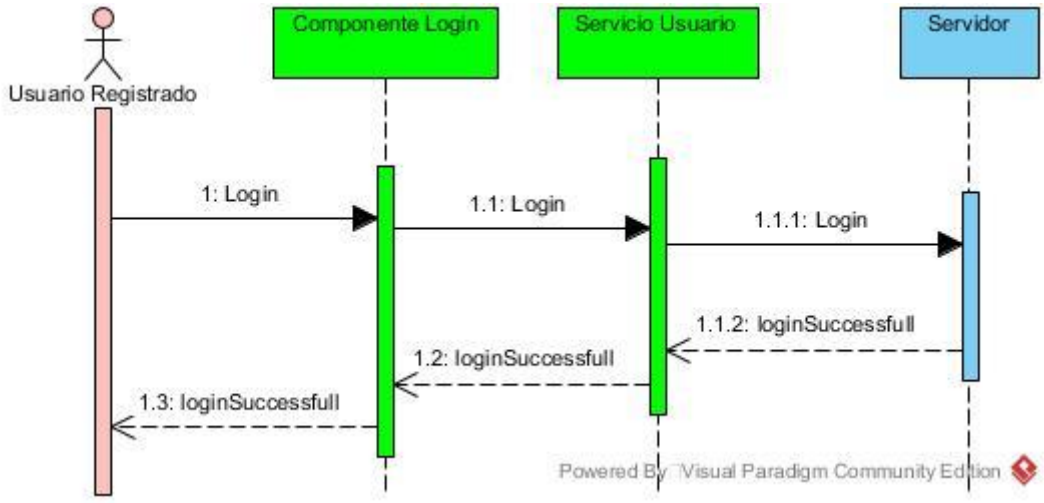


Ilustración 16: Diagrama de secuencia para loguearse (Cliente)

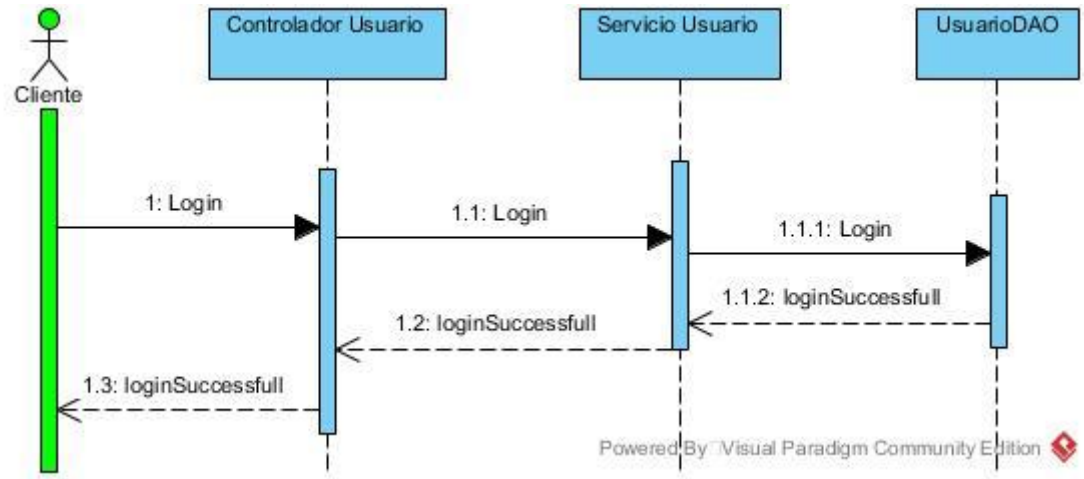


Ilustración 17: Diagrama de secuencia para logearse (Servidor)

En estos diagramas podemos observar como las llamadas que realiza el usuario se propagan por los objetos de nuestro sistema para realizar las tareas.

Para realizar el *login* el usuario se lo solicitará al componente *login* que utilizará el servicio usuario del cliente para solicitar los datos al servidor, esta solicitud legará al controlador de servicio REST que hará uso del servicio usuario para así llegar finalmente hasta los DAO que nos devolverán el usuario logado.

Seguidamente se muestra un diagrama que devuelve el listado de votaciones a un usuario registrado

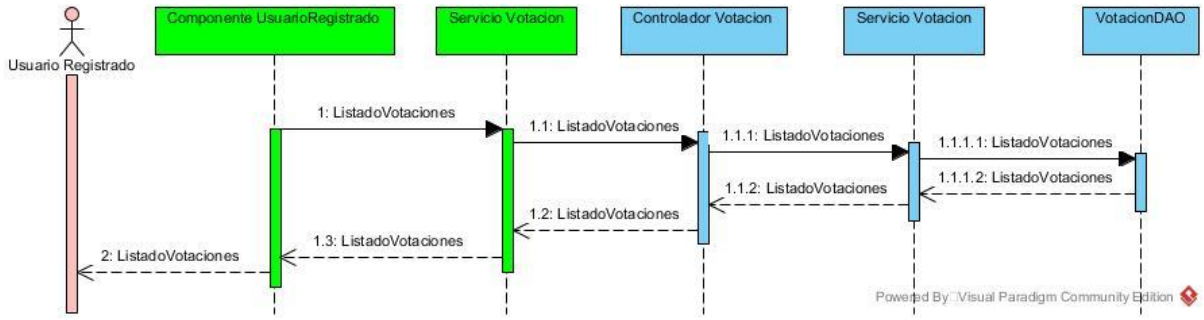


Ilustración 18: Diagrama de secuencia para obtener el listado de votaciones

Una vez logado el componente *usuarioRegistrado* del cliente usará el servicio votaciones del cliente para comunicarse con el servidor y solicitar a su controlador de votaciones el listado de votaciones de ese usuario, el controlador hará uso del servicio de *votaciones* del servidor para, mediante el DAO de *votaciones* obtener el listado de votaciones y devolverlo hasta el usuario.

Se puede ver que es un proceso repetitivo y el resto de operaciones seguirán un esquema similar, los componentes del cliente hacen uso del servicio adecuado para hacer una solicitud a los controladores del servicio REST, estos usan su servicio del servidor correspondiente para acceder a los datos necesarios mediante los DAOs.

En los casos de querer crear o modificar una votación el usuario manda la solicitud a componente *usuarioRegistrado*, este le devuelve el formulario adecuado. Una vez rellenado el formulario se lo manda al componente que propaga la solicitud al servidor hasta llegar al DAO *votación* que guardará los cambios en la base de datos.

Los únicos los únicos dos casos con un esquema diferente serían las tareas para el obtener el listado de métodos de reparto y para realizar el cálculo del reparto de escaños. Para estos casos el usuario a través de componente *usuarioRegistrado* hace uso del servicio *Elección* del cliente para hacer las solicitudes al servidor. Estas solicitudes llegan al controlador de *elecciones* del servidor que hace uso del servicio de *elecciones*, donde directamente se hacen los caculos necesarios utilizando una librería de un tercero, sin tener que utilizar ningún DAO.

3.3.1. Diagrama de secuencia Usuario sin registrar

En el siguiente diagrama mostraremos las acciones a llevar a cabo para hacer una simulación de reparto de escaños para un usuario que no está registrado y las comunicaciones entre los objetos involucrados en esta tarea.

Se hará una separación entre el cliente y el servidor para poder observar bien los objetos de cada parte, el cliente estará reflejado de color verde mientras que el servidor será de color azul como veremos en el diagrama.

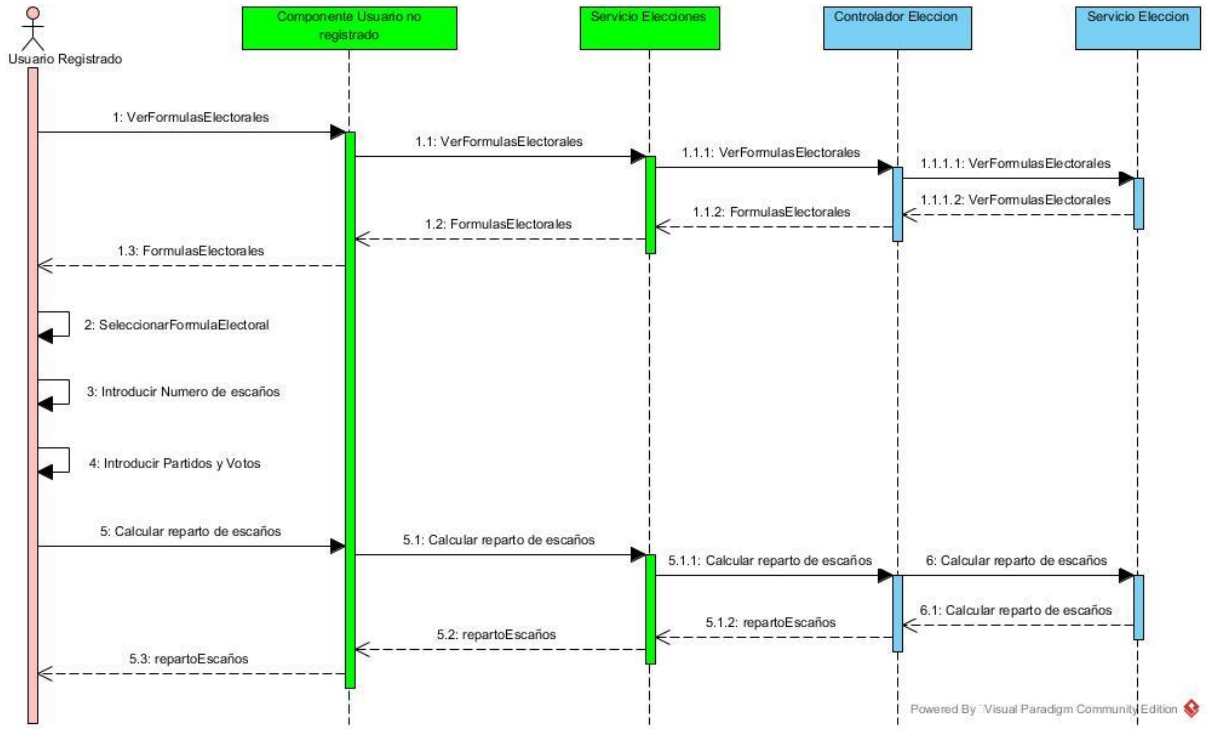


Ilustración 19: Diagrama de secuencia para usuarios no registrados

En el caso de un usuario no registrado el uso de la aplicación WEB es más limitado, directamente accederá por el componente de *usuario no registrado* en el cual solicitará ver los métodos de reparto, este hará uso del servicio elecciones como explicamos en el apartado anterior para comunicarse con el servidor y hacer la solicitud a su controlador de elecciones este usará el servicio elecciones y devolverá los resultados.

Una vez el componente de *usuario no registrado* devuelva los métodos de reparto el usuario podrá seleccionar el que desea, introducir el número de escaños a repartir, introducir los partidos y votos, para después poder hacer la solicitud de reparto de escaños que se propagará como la anterior hasta el servicio elecciones del servidor.

3.4. Diseño de la Interfaz

El diseño de la interfaz se encarga de crear un medio de comunicación entre los usuarios y las computadoras para que un sistema se pueda ejecutar [2].

Para ayudarme en la realización del diseño de la interfaz se ha usado la herramienta *Axure*[8], es una herramienta de prototipado rápido con la que puedes crear diseños para web, móviles o de escritorio, se utiliza mediante la colocación de widgets. Además, se ha utilizado también una biblioteca de widgets *Axure Bootstrap*[9], con la que darle un aspecto más similar al que tendrá una vez creada la aplicación real.

El usuario accederá a la página home que podemos ver a continuación, en ella tendrá la opción de registrarse, logarse o hacer una simulación sin darse de alta.

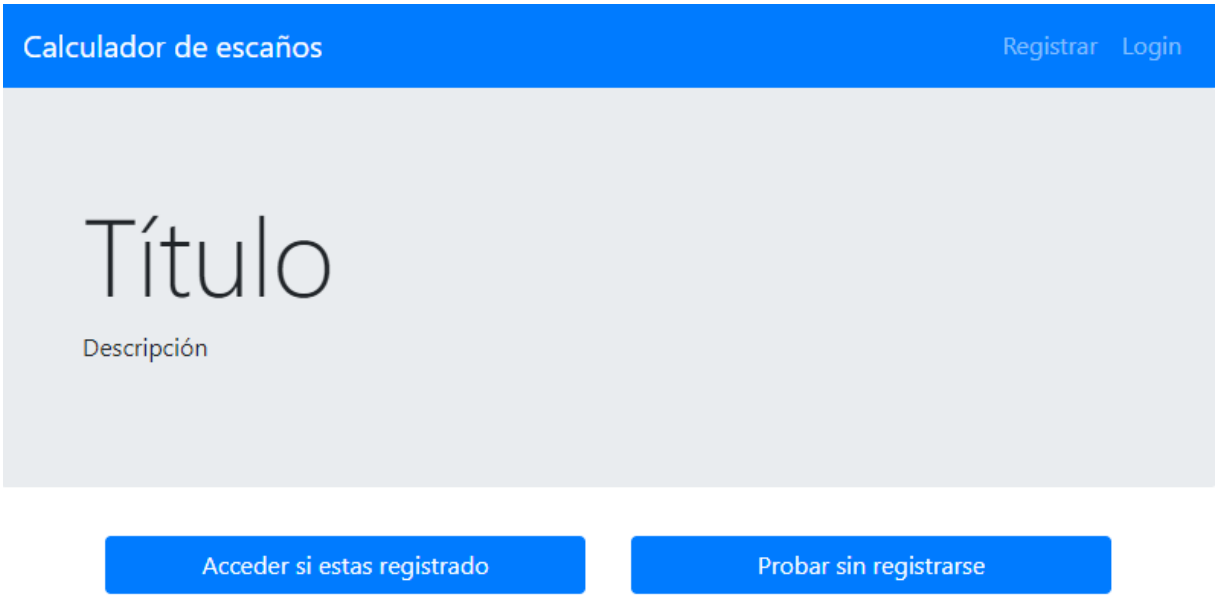


Ilustración 20: Diseño de la interfaz - página *home*

Tanto si pulsan el botón *Login*, como si pulsan el botón "Acceder si estas registrado" se les redirigirá al formulario de *Login*, mientras que si pulsan el botón Registrar se les redirigirá al formulario de registro.

Como podemos ver a continuación tanto el formulario de registro como el de *login* serán similares, ambos contendrán inputs para el nombre y la clave del usuario, un botón para aceptar y una pregunta que les redirigirá el uno al otro.

[8] Axure: https://www.axure.com/
[9] Bootstrap widget library : https://axemplate.com/downloads/bootstrap-widget-library/

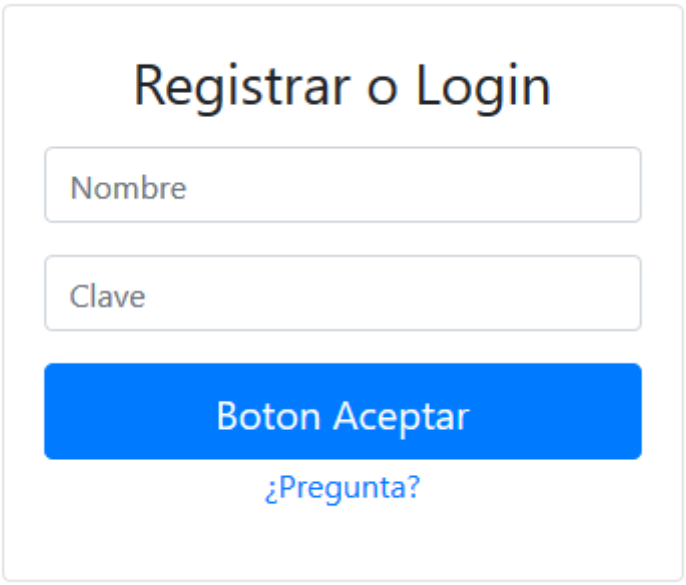


Ilustración 21: Diseño de la interfaz - formulario de registro o login

Una vez introducidos el nombre, si todo es correcto, se les redirigirá a la página de usuario, en la cual como vemos posteriormente se tendrán un conjunto de parámetros a introducir, una lista de votaciones, una lista de los partidos y los votos de las votaciones y por ultimo una gráfica.

Ilustración 22: Diseño de la interfaz – usuario registrado

En la página de usuario también habrá dos botones más en el *navbar* uno que te redirigirá a la página de perfil, como veremos a continuación, y otro que se desconectará de la aplicación y les redirigirá a la página home de nuevo.

Ilustración 23: Diseño de la interfaz - perfil

Por ultimo en la página de perfil estarán los datos del usuario y sus votaciones. Desde aquí se podrá modificar los datos del usuario o eliminar votaciones.

3.5. Diseño del API REST

En este apartado podemos ver las diferentes URLs disponibles del servidor, sus métodos y sus parámetros de entrada y salida.

Descripción	URLs	Métodos	Parámetros de entrada	Parámetros de salida
Registrar usuario	/usuario/	POST	RequestBody: • Nombre • Clave	
Login de usuario	/usuario/{nombreUsuario}	POST	PathVariable: Nombre RequestBody: • Nombre • Clave	Usuario
Modificar usuario	/usuario/{nombreUsuario}?clave=”	PUT	PathVariable: Nombre RequestParam: Clave	
Borrar usuario	/usuario/{nombreUsuario}	DELETE	PathVariable: Nombre	
Crear votación	/votacion/{votacion}/usuario/{usuario}	POST	PathVariable: • Nombre Usuario • Nombre Votación	
Modificar votación	/votacion/{id}	PUT	PathVariable: Id votación RequestParam: Nombre Votación	
Obtener votación	/votacion/{id}	GET	PathVariable: id votación	Votación

Obtener lista de votaciones	/votacion/usuario/{usuario}	GET	PathVariable: Nombre usuario	Lista de votaciones
Borrar votación	/votacion/{id}	DELETE	PathVariable: id votación	
Añadir partidos y votos	/votacion/{id}/partido	POST	PathVariable: id votación RequestBody: Partidos y votos	
Obtener partidos y votos	/votacion/{id}/partido	GET	PathVariable: id votación	Mapa de partidos y votos
Modificar partido y votos	/votacion/{id}/partido/{partido}	PUT	PathVariable: • Id votación • Partido RequestParam: Votos	
Borrar partido y votos	/votacion/{id}/partido/{partido}	DELETE	PathVariable: id votación	
Calcular reparto de escaños	/compute/metodo/{metodo}?esc=”	POST	RequestParam: método RequestBody: Partidos y votos PathVariable: escaños	Mapa de partidos y escaños
Obtener métodos de reparto	/compute/metodos/{tipo}	GET	PathVariable: Tipo	Lista de los métodos de reparto

Tabla 7: Diseño API REST

3.6. Plan de pruebas

En las tareas de desarrollo se ha incluido un tiempo para llevar a cabo los test unitarios de esos desarrollos. Dichos test unitarios se centrarán en las clases de los DAOs para comprobar que se obtienen las funcionalidades especificadas en las tareas propuestas.

Para dichos test utilizaremos *JUnit*[10]:

[10] JUnit: https://junit.org/

Es un *framework* sencillo para escribir test unitarios, hace uso de anotaciones para identificar los métodos que especifican los test [18].

Posteriormente se realizarán los test de integración de la API REST de nuestro servidor con la herramienta *Postman*[11].

Por último, se verificará desde el cliente que todas las historias de usuario funcionan adecuadamente, para ello se contará con al menos un usuario externo que realizará todas las pruebas de aceptación.

[11] Postman: https://www.getpostman.com/

4. IMPLEMENTACIÓN

En este apartado veremos que es el diseño arquitectónico y los diagramas de arquitectura, los cuales serán dos separando el servidor y el cliente. Además, se van a comentar algunos de los aspectos más relevantes en la utilización de las tecnologías y *frameworks* empleados para facilitar su posterior mantenimiento

4.1. Arquitectura

El diseño arquitectónico representa la estructura de los datos y de los componentes del programa que se requieren para construir un sistema basado en computadora. Considera el estilo de arquitectura que adoptará el sistema, la estructura y las propiedades de los componentes que lo constituyen y las interrelaciones que ocurren entre sus componentes arquitectónicos. [2]

4.1.1. Diagrama de paquetes del servidor

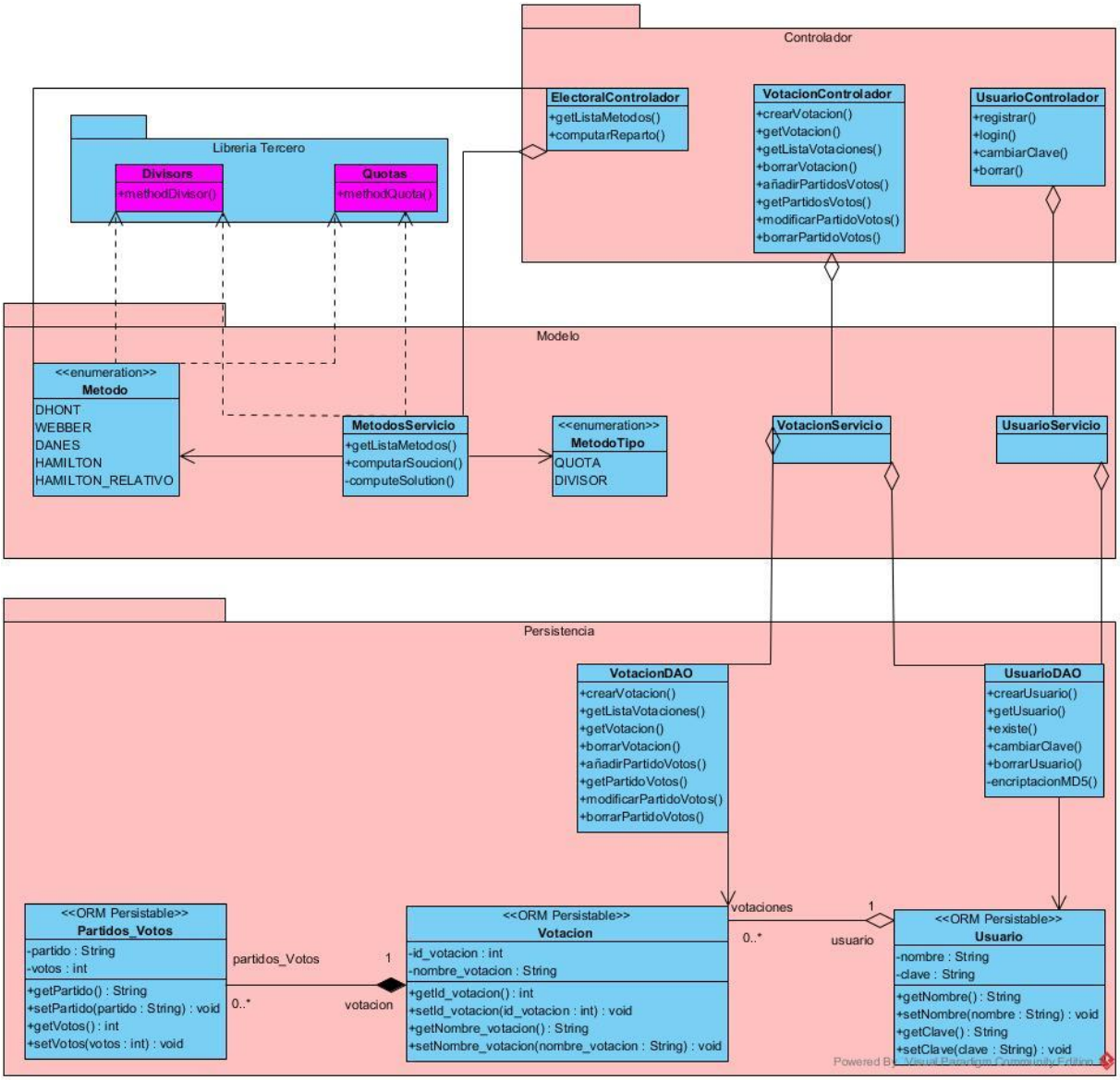


Ilustración 24: Diagrama de paquetes del servidor

En este diagrama podemos ver la separación en la persistencia con las clases encargadas del manejo de la base de datos, el modelo que contendrá la lógica de negocio y por último los controladores que serán los que se comuniquen con el cliente donde estarán las vistas.

4.1.2. Diagrama de paquetes del cliente

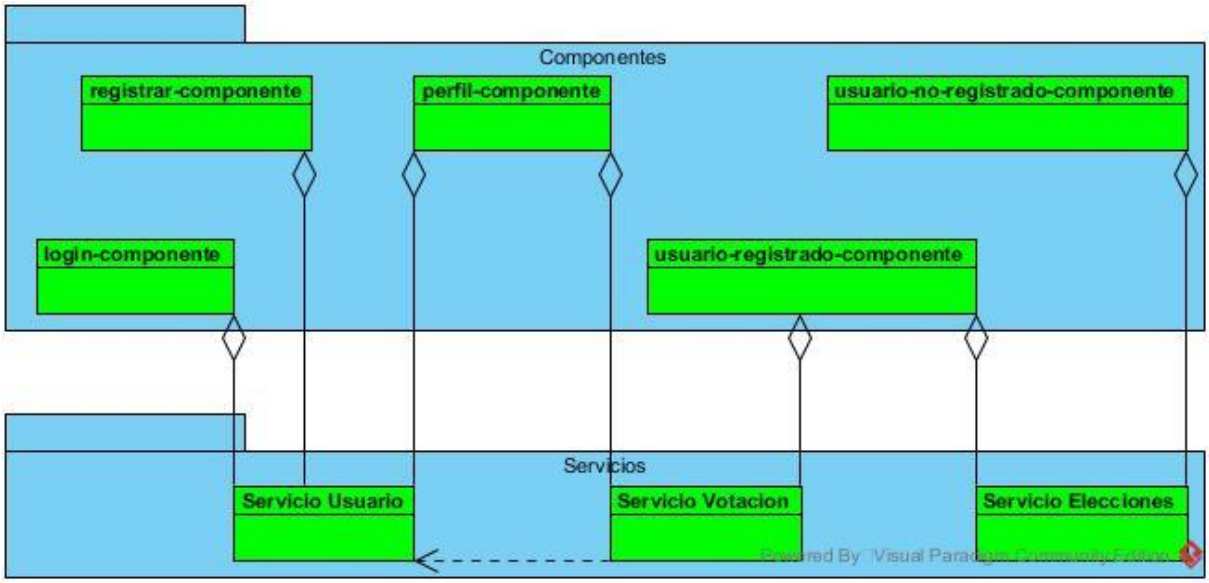


Ilustración 25: Diagrama de paquetes del cliente

En este diagrama podemos observar que la separación será entre componente y servicios, siendo los servicios los que accederán a los controladores del servidor, y siendo los componentes los encargados de las vistas.

4.1.3. Diagrama de arquitectura

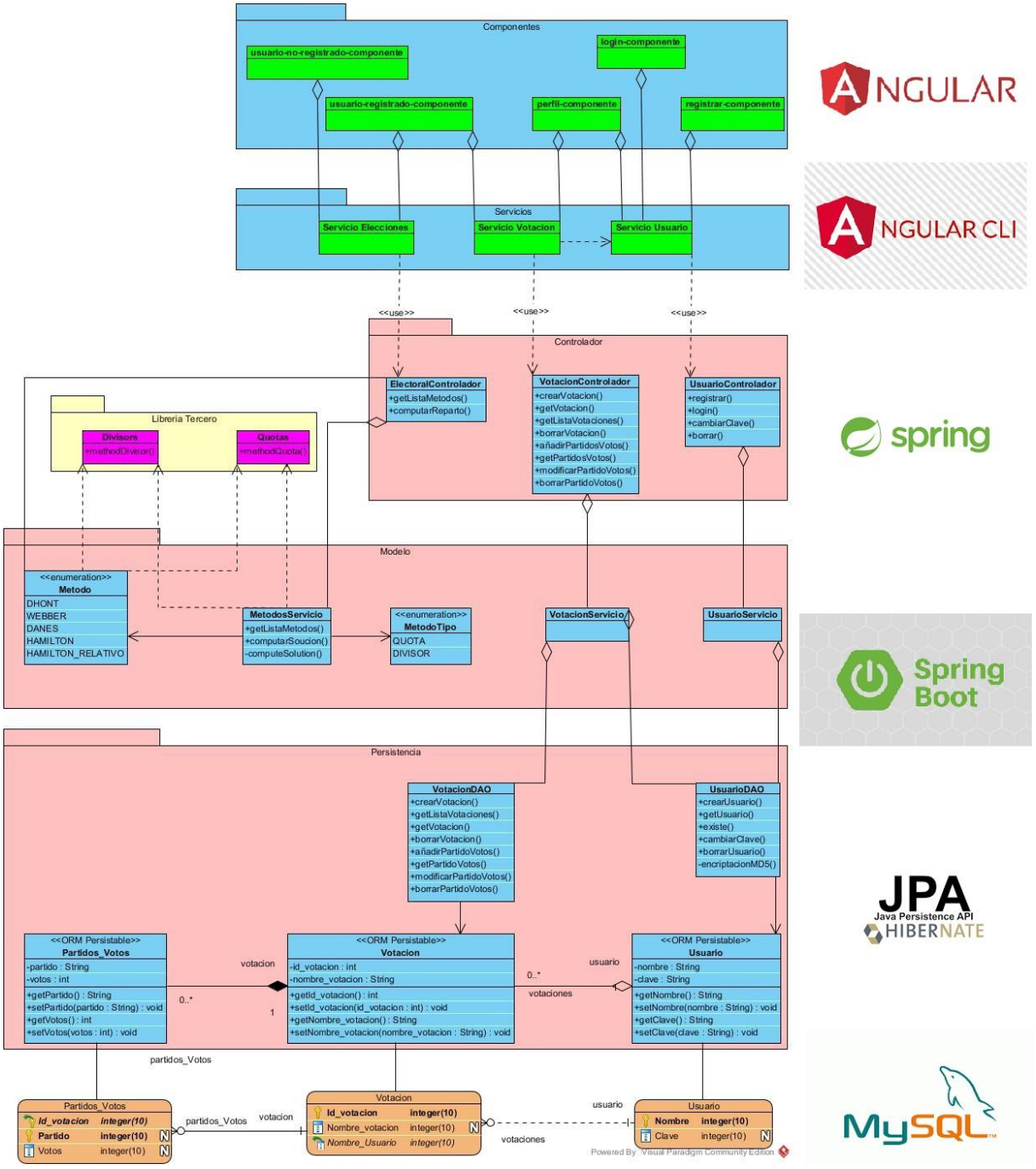


Ilustración 26: Diagrama de arquitectura

En este diagrama se puede observar finalmente la relación entre el servidor y el cliente a través de los controladores del servidor y los servicios del cliente. Además, se muestran en orden las tecnologías que se usan en cada parte.

4.2. Detalles sobre implementación

En este apartado se van a tratar cuestiones concretas sobre los aspectos metodológicos y técnicos de la implementación, recursos empleados, bibliotecas o servicios externos, *framework*, etc.

Para ello dividiré el apartado en dos partes, una que trate todos los aspectos relevantes del servidor y otra que trate los del cliente. Además, mostraré ejemplos breves de cómo se usan los *frameworks.*

4.2.1. Servidor

Esta aplicación se dividirá en 2 proyectos o módulos (como pasan a llamarse en *intellij*). Uno de los módulos, llamado “persistencia”, como su nombre indica será el encargado de la comunicación directa con la base de datos e implementa los DAOs (*Data Access Object*) que proporcionan una interfaz común entre la aplicación y la base de datos. El segundo módulo, llamado “election-rest”, hace uso del primero para acceder a los datos y además utiliza también una librería proporcionada por un tercero para realizar los cálculos necesarios con los datos, para después proporcionar los resultados al cliente mediante un API REST.

Para los dos módulos creados se ha decidido usar *Spring framework*, para ello se ha recurrido al asistente *Spring Boot* para generar un proyecto *Maven* ya que nos ayuda con la selección de jars para *Maven* teniendo que elegir solo el tipo de aplicación que deseamos y también nos permite elegir las dependencias de Spring automáticamente, para ello, a la hora de crear nuestra aplicación accedemos a la página https://start.spring.io/ que se encarga de ayudarnos en todo el proceso.

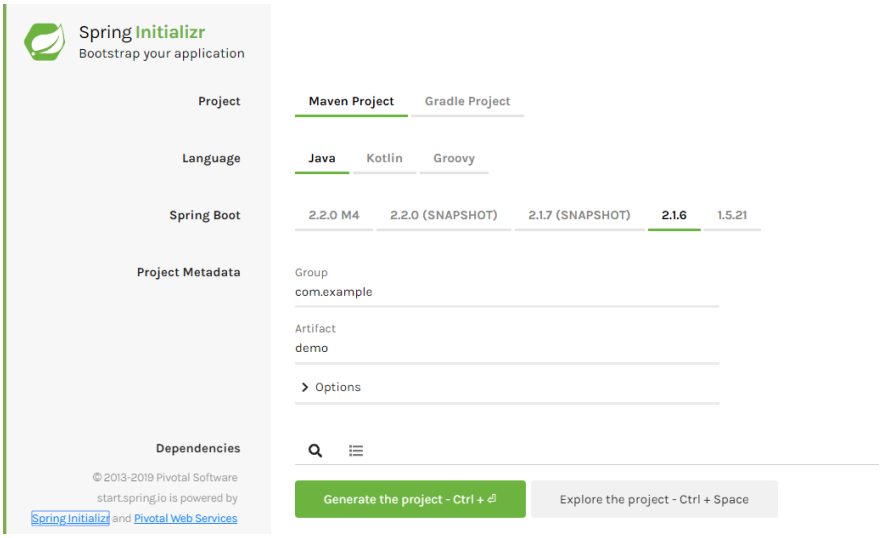


Ilustración 27: *Spring Initializr*

Se ha utilizado esta herramienta, ya que permite crear una aplicación funcional de manera rápida y sencilla.

Modulo Persistencia

Como se describió antes, con *Spring Boot* puedes seleccionar el tipo de aplicación y las dependencias, para crear este módulo se ha creado una aplicación Spring MVC con las dependencias *MySQL Driver* y *Spring Data JPA*.

Este módulo contiene esta jerarquía de paquetes:

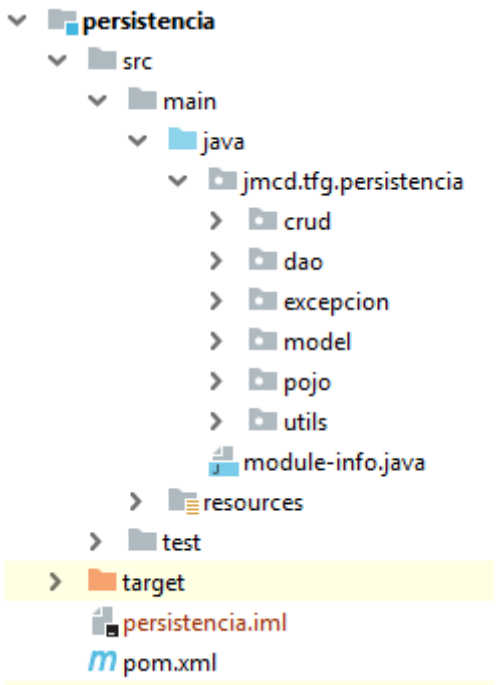


Ilustración 28: Jerarquía de paquetes del servidor

Se comenzará por el paquete *model* en el cual se han creado las clases del modelo “Usuario” y “Votación” las cuales son clases persistentes y en ellas se ha empleado JPA para generar el esquema de base de datos completo. La dependencia JPA ya fue incluida por *Spring Boot* en *maven*, el siguiente paso es configurar el fichero “persistence.xml”:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2"
 xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
<persistence-unit name="persistencia" transaction-type="RESOURCE_LOCAL">
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
<class>jmcd.tfg.persistencia.model.Usuario</class>
<class>jmcd.tfg.persistencia.model.Votacion</class>
<properties>
<property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver" />
<property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/votacion?useSSL=false&amp;useUnicode=true&amp;useJDBCComp
liantTimezoneShift=true&amp;useLegacyDatetimeCode=false&amp;serverTimezone=UTC" />
<property name="javax.persistence.jdbc.user" value="root" />
<property name="javax.persistence.jdbc.password" value="mysql" />
<property name="javax.persistence.schema-generation.database.action" value="create"/>
</properties>
</persistence-unit>
</persistence>
```

Ilustración 29: Fichero “persistence.xml”

En este fichero definimos los parámetros para conectarnos a la base de datos, en este caso *MySql* y también definimos las entidades que vamos a persistir. Por ejemplo:

- El driver que maneja el protocolo de comunicación especifico con la BD es "com.mysql.cj.jdbc.Driver"
- La url de conexión a la BD es “jdbc:mysql://localhost:3306/votación”
- Las credenciales de acceso son:
  - « javax.persistence.jdbc.user » = "root"
  - « javax.persistence.jdbc.password » = "mysql"

Después paso al uso de las anotaciones en las clases, comenzando por Usuario ya que es la clase más sencilla, solo se necesita la anotación @*Entity* para indicar que es una clase persistente y la anotación @Id para indicar su id en la base de datos.

La clase Votación es algo más compleja ya que contiene un mapa que vamos a persistir y también un usuario. Por lo tanto, usaré también @*Entity* para persistir la clase, @Id para indicar el id en la base de datos, además usaré @*ManyToOne* para Usuario, ya que un usuario puede tener muchas votaciones y por último para el mapa utilizaré:

```
@ElementCollection(fetch = FetchType.EAGER)
@CollectionTable(name = "partidosVotos",
joinColumns = @JoinColumn(name = "idPartidosVotos"))
@MapKeyColumn(name = "partido")
@Column(name = "votos")
private Map<String, Integer> mapPartidosVotos;
```

Ilustración 30: Anotaciones para el mapa mapPartidosVotos

Esto dará lugar a la creación de una Tabla más en la base de datos llamada “partidoVotos” cuya clave primaria estará compuesta por el “id_votacion” y por “partido” que representa la *key* de nuestro mapa y además contendrá los votos que son los valores de nuestro mapa.

Después se procedió a realizar el paquete CRUD en el cual se ha creado una clase abstracta *EntityCRUD* en la cual se creará un *EntityManagerFactory* para iniciar JPA y el resto de métodos, que implementaré en las clases que la extienden, para interactuar con la base de datos. Estas clases son UsuarioCRUD y VotacionCRUD las cuales obtienen del *EntityManagerFactory* un *EntityManager* para interactuar con JPA.

Una vez creado el CRUD se ha decidido usar el patrón DAO, ya que es adecuado para aplicaciones con pocas entidades. Por cada una de las tablas en la base de datos tendría que existir un DAO, aunque este caso es un poco diferente ya que una de las tablas creadas es un mapa que pertenece a la clase Votación, por lo tanto, solo se

han implementado dos DAO UsuarioDAO y VotacionDAO que hacen uso de los métodos implementados en el CRUD, Estos DAO son los que finalmente serán accesibles desde el exterior de la aplicación.

En este caso se han creado también dos paquetes más. Uno, el paquete pojo, contiene los pojos (*Plain Old Java Object*) que serán accesibles desde el exterior, y el paquete *utils* que contiene dos clases UsuarioPopulate y VotacionPopulate que se encargan de traducir las clases del modelo a los pojos, de esta forma evitamos que las clases del modelo sean visibles desde el exterior de la aplicación.

Para hacer accesibles los DAO y los POJOS desde el exterior se ha empleado el Java *Platform Module System* (JPMS), o "Módulos" que es un nuevo sistema de abstracción que introdujo Java 9. Un módulo es un conjunto de paquetes del proyecto. En el archivo descriptor “module-info.java” indica el nombre del módulo, sus dependencias, que son una lista de modulo que él mismo requiere, y los paquetes públicos:

```
open module persistencia {
requires spring.beans;
requires java.sql;
requires org.hibernate.orm.core;
requires java.persistence;
requires spring.context;
requires org.slf4j;

exports jmcd.tfg.persistencia.dao;
exports jmcd.tfg.persistencia.pojo;
}
```

Ilustración 31: Fichero module-info.java

Módulo election-rest

Para la generación de este módulo también he4 usado *Spring Boot* incluyendo la dependencia *Spring Web Starter* que incluye RESTfull.

Este módulo hace uso, como se comentó anteriormente, del módulo persistencia y de la librería de un tercero para crear un API REST que ponga a disposición de los clientes los servicios que deseamos, para ello se crean tres controladores:

- UsuarioControlador
- VotacionControlador
- ElectoralControlador

Estos controladores usaran la anotación *@RestController* para que Spring reconozca la clase como servicio web y la trate como tal, además de añadir también la anotación *@RequestMapping* a nivel de clase y especificando después para cada método con las anotaciones *@GetMapping*, *@PostMapping*, *@PutMapping* y *@DeleteMapping* dependiendo de su función.

Se ha tenido que añadir también la anotación a nivel de clase *@CrossOrigin*(*origins* = "http://localhost:4200") para poder acceder a los servicios web desde un cliente y evitar el error:

Access to XMLHttpRequest at '' from origin '' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

Ilustración 32: Error CORS

CORS: Intercambio de recursos de origen cruzado [19], en este caso se producía por utilizar dos puertos diferentes, y esto daba lugar a que las solicitudes eran exitosas, pero el navegador las bloqueaba. Otra posible solución era agregar el encabezado "*Access-Control-Allow-Origin*". En producción se deberá adaptar al nombre de dominio donde se despliegue la aplicación cliente.

UsuarioControlador y VotacionControlador hacen uso del módulo de persistencia mientras que ElectoralControlador hace uso directamente de la librería de un tercero. Para este último se han creado dos clases tipo enum:

- "MetodoTipo" la cual contiene los dos tipos de métodos de reparto que uso, de *quota* o de divisor.
- "Metodo" la cual con tiene los métodos que uso en el sistema.

Además, también se ha creado un servicio, llamado MetodosServicio que hace todos los cálculos ayudándose de la librería de un tercero para que realice toda la lógica de negocio, quedando así el *ElectoralControlador* mucho más limpio. Para ello como se comentó en el apartado Software de terceros para el cálculo de reparto de escaños, se ha descargado de GitHub el código fuente, se ha compilado localmente e instalado en el repositorio local de *maven,* una vez hecho esto se ha incluido la dependencia en el POM del módulo.

```xml
<dependency>
    <groupId>com.dan323.elections</groupId>
    <artifactId>elections</artifactId>
    <version>3.1</version>
    <scope>compile</scope>
</dependency>
```

Ilustración 33: Dependencia maven de la librería de terceros

Seguidamente se puede hacer la importación de las clases y métodos necesarios al servicio.

```
import com.dan323.elections.systems.div.Divisors;
import com.dan323.elections.systems.quo.Quotas;
```

Ilustración 34: Importaciones de las clases Divisors y Quotas

Y finalmente se puede hacer uso de los métodos de las clases en el código.

```
private Map<String, Integer> computeSolution(Map<String, Long> votos, int esc, String metodo) throws NoSuchMethodException{
    Optional<Metodo> met=Arrays.stream(Metodo.values())
            .filter(m->m.getNombre().equals(metodo))
            .findFirst();
    if (met.isPresent()){
        Metodo me=met.get();
        if (me.getTipo().equals(MetodoTipo.DIVISOR)){
            return Divisors.methodDivisor(votos,esc,me.getDivisor());
        }
        else if (me.getTipo().equals(MetodoTipo.QUOTA)){
            return Quotas.methodQuota(votos,esc,me.getQuota(),me.getRemainder());
        }
    }
    return new HashMap<>();
}
```

Ilustración 35: Ejemplo de uso de librería de tercero

En este fragmento de código podemos ver como se utiliza el método *metohdDivisor* y *methodQuota* de sus correspondientes clases.

4.2.2. Cliente

Para la parte del cliente se ha decidido usar el *framework*, de código abierto que está desarrollado en *TypeScript* y es mantenido por Google, Angular [20]. Este apartado es más extenso que el anterior ya que se tenía menos experiencia con este *framework* y se ha querido dejar reflejado todos los conceptos relevantes aprendidos para facilitar su mantenimiento y desarrollo futuros

Mediante la herramienta Angular CLI [21] se pueden generar y estructurar de manera fácil proyectos Angular, es una herramienta de línea de comandos que crea, depura y publica aplicaciones Angular.

Con esta herramienta se ha creado el cliente Angular, para ello, nos desplazaremos al directorio donde queremos crear el cliente Angular y usaremos el siguiente comando para crearlo:

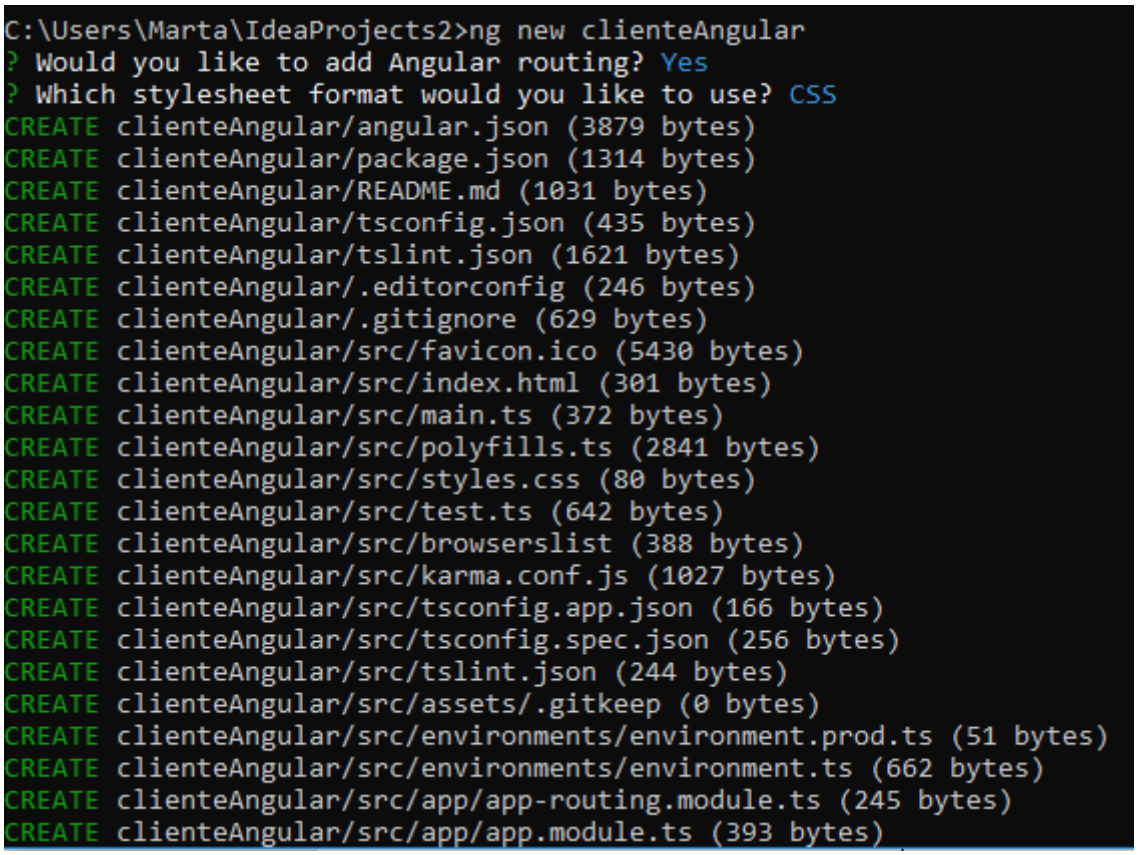


Ilustración 36: Comando de generación de cliente Angular

Con esta herramienta también se podrán instalar las dependencias que van a utilizar en el proyecto como son *Bootstrap, jQuery y Popper.js*, usando el siguiente comando:

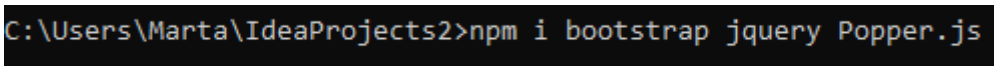


Ilustración 37: Comando de instalación de dependencias

Una vez hecho esto podremos empezar a crear también nuestros servicios y componentes los cuales crearé en las carpetas servicios y componentes respectivamente mediante los comandos:

```
C:\Users\Marta\IdeaProjects2\clienteAngular>ng g c componentes/home
CREATE src/app/componentes/home/home.component.html (23 bytes)
CREATE src/app/componentes/home/home.component.spec.ts (614 bytes)
CREATE src/app/componentes/home/home.component.ts (261 bytes)
CREATE src/app/componentes/home/home.component.css (0 bytes)
UPDATE src/app/app.module.ts (479 bytes)
```

Ilustración 38: Comando de creación de un componente

```
C:\Users\Marta\IdeaProjects2\clienteAngular>ng g s servicios/usuarioApi
CREATE src/app/servicios/usuario-api.service.spec.ts (354 bytes)
CREATE src/app/servicios/usuario-api.service.ts (139 bytes)
```

Ilustración 39: Comando de creación de un servicio

La aplicación tendrá tres servicios:

- Servicio UsuarioAPI
- Servicio VotacionAPI
- Servicio EleccionAPI

Estos servicios contendrán las llamadas al servidor REST, ya que se encargan de acceder a la información y realizar las operaciones con los datos, actuando como proveedores de dichos datos para los componentes, de los cuales hablaré a continuación.

Ejemplo de elementos generados por Angular CLI servicio:

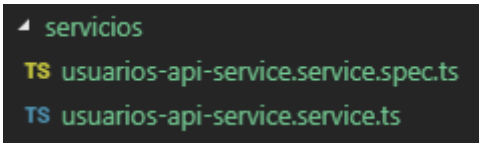


Ilustración 40: Ejemplo de ficheros de un servicio

Uno de los elementos principales de Angular son los componentes que como se explica en su documentación oficial son los bloques de construcción más básicos de una interfaz de usuario en una aplicación Angular, las cuales son desarrolladas en base a un árbol de componentes.

Ejemplo de componente home:

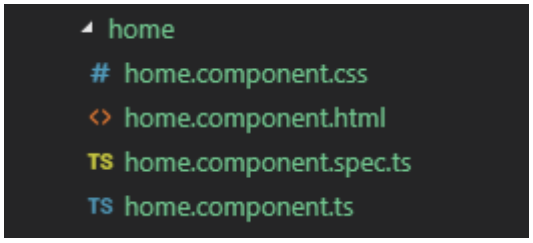


Ilustración 41: Ejemplo de ficheros de un componente

El siguiente paso es modificar el componente principal de la aplicación "app.component.html", incluyendo en él, todos los componentes que quiero para la estructura de la página como son:

- Componente *navbar*
- Componente *hero*
- Y lo más importante el "*router-outlet*" que se encargará de inyectar las rutas de la aplicación.

Las rutas de la aplicación se crean en el fichero "*app-routing.module.ts*" el cual contiene el *array* "*routes*" donde incluyo todas las rutas de la aplicación siguiendo el siguiente patrón:

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'usuario/registrar', component: RegistrarComponent },
```

Ilustración 42: Ejemplo de rutas en el fichero *app-routing.module.ts*

Hay que incluir la ruta con la que queremos trabajar y el componente que queremos *renderizar*[12] para esa ruta.

El orden de estas rutas es muy importante, ya que la forma en que las rutas funcionan en Angular es que la primera ruta positiva que encuentre en el *array* será el que se *renderice*, si por ejemplo tienes un componente que se utilice para todas url que no estén contempladas en la aplicación, tendrá que ser el último en el *array*.

Se han de importar todos los componentes que usamos en las rutas en el fichero *"app-routing.module.ts",* pero hay extensiones para los IDEs como *Visual Studio Code* que ayudan a estas tareas.

[12] *Renderizar*: Anglicismo usado en jerga informática para referirse al proceso de generar una imagen visible e inteligible para el ser humano, a partir de información digital

Otro fichero de configuración importante en Angular es "*app.module.ts*" donde hay varios *arrays*:

- En el *array* "*declarations*" hay que introducir todos los componentes de la aplicación, por ejemplo:

```
declarations: [
  AppComponent,
  HomeComponent,
  NavbarComponent,
  HeroComponent,
```

Ilustración 43: Array de componentes del fichero app.module.ts

- En el *array* "*import*" hay que introducir todos los módulos que hacen falta en la aplicación:

```
imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule,
  FormsModule,
  ChartsModule
],
```

Ilustración 44: Array de modulos del fichero app.module.ts

- En el *array* "*providers*" tengo que inyectar todos los servicios

```
providers: [VotacionesApiService, UsuariosApiServiceService],
```

Ilustración 45: Array de servicios del fichero app.module.ts

Ya hemos visto los ficheros de configuración más importantes, a medida que creemos nuevos componentes, servicios o necesitemos de nuevos módulos tendremos que incluirlos en sus correspondientes *array*.

Ahora empezaré con la implementación de los servicios, para ello necesitaremos importar "*HttpClient*", "*HttpHeaders*" y también "Observable". En los constructores inyectaremos el "*HttpClient*" y crearemos una variable *HttpHeaders* que usaremos en todas las llamadas al servicio REST menos las GET.

```
headers: HttpHeaders = new HttpHeaders({
  "Content-Type": "application/json"
});
```

Ilustración 46: Variable *headers*

Cada servicio tendrá tantas funciones como sean necesarias para comunicarse con el servidor y para ello hará uso de *HttpClient* [22] que es un API HTTP que facilita la comunicación entre el *front-end* y el *back-end*.

HttpClient contiene los métodos *get, post, put* y *delete* que nos devolverán objetos del tipo "Observable"[13] a los cuales nos subscribiremos posteriormente desde los componentes cuando hagamos uso de nuestros servicios.

Todas las funciones de los servicios tienen en común que tendrán una URL y usarán uno de los métodos HTTP mencionados antes, como vemos en este ejemplo simple:

```
getVotacion(id: string) {
  const urlApi = `http://localhost:8080/votacion/${id}`;
  return this.http.get<VotacionInterfaz>(urlApi);
}
```

Ilustración 47: Ejemplo de función con método get de un servicio

En este ejemplo vemos que usamos una URL con un parámetro variable, en este caso el "id" de la votación que queramos obtener. Los métodos HTTP nos devuelven objetos de tipo anónimo, por ellos en algunas ocasiones para que el consumo de esta respuesta sea más fácil hay que indicar el tipo del objeto definiendo una interfaz.

Estas interfaces las crearemos en la carpeta "modelos" y en ella simplemente indicamos los atributos del objeto y sus tipos correspondientes para facilitar su uso posterior. Aquí podemos ver un ejemplo:

```
export interface VotacionInterfaz {
  idVotacion: string;
  nombreVotacion: string;
  usuario: UsuarioInterfaz;
  mapPartidosVotos: { [name: string]: number };
}
```

Ilustración 48: Ejemplo de interfaz en Angular

Continuando con los servicios sus funciones pueden ser más complejas dependiendo de la situación, aquí podemos observar otro ejemplo:

[13] Observable: https://angular.io/guide/observables

```
computarReparto(
  mapPartidosVotos: Map<string, number>,
  metodo: string,
  esc: number
) {
  const urlApi = `http://localhost:8080/compute/metodo/${metodo}?esc=${esc}`;
  return this.http
    .post<{ [name: string]: number }>(urlApi, mapPartidosVotos, {
      headers: this.headers
    });
}
```

Ilustración 49: Ejemplo de función con método post de un servicio

En este caso es un método Post al cual tenemos que pasarle un mapa con los partidos y los votos en el *body*, además tendremos que pasarle nuestro objeto *headers* y la URL tiene dos parámetros variables.

A continuación, veremos el funcionamiento de los componentes, como usan los servicios y un ejemplo. Un componente controla una parte de la pantalla, llamada vista, para ello primero definimos la lógica de la aplicación dentro de una clase (.ts), la cual interactúa con la vista a través de una API de atributos y métodos. Después definimos la vista del componente con un *template*, los cuales parecen un HTML normal, pero incluyen sintaxis adicional de Angular y se utilizan para indicar a Angular como *renderizar* el componente.

Vamos a ver como listar las votaciones de un usuario, para ello el componente usará nuestro VotacionesApiService para obtener los datos del servidor y por ello lo importaremos y lo inyectaremos en el constructor de nuestro componente.

Después crearemos un atributo y un método para interactuar con la vista, el atributo será un *array* de votaciones al cual le añadiremos los valores en el siguiente método:

```
getListVotaciones(): void {
  this.votacionesApi
    .getListaVotaciones()
    .subscribe(
      (votaciones: VotacionInterfaz[]) => (this.votaciones = votaciones)
    );
}
```

Ilustración 50: Ejemplo de método de la clase (.ts) del componente

Como los métodos de los servicios devuelven objetos Observable tenemos que suscribirnos a ellos como comentamos antes, le indicamos el tipo de objeto que esperamos y se lo asignamos a nuestro atributo.

Después este atributo será accesible desde la vista pudiendo acceder a sus valores y, en este caso, al ser un *array* recorrerlo

```
<table class="table table-striped">
  <thead>
    <tr>
      <th scope="col">Id</th>
      <th scope="col">Nombre Votacion</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let votacion of votaciones"
      (click)="getPartidosVotos(votacion.idVotacion, votacion.nombreVotacion)">
      <th scope="row">{{votacion.idVotacion}}</th>
      <td>{{votacion.nombreVotacion}}</td>
    </tr>
  </tbody>
</table>
```

Ilustración 51: Ejemplo de uso de atributos y métodos de componente en la vista

Como ya dijimos y podemos ver en la Ilustración 51, la vista del componente usa la notación HTML además de incluir sintaxis de Angular, en el ejemplo vemos el uso de *ngFor usado para iterar sobre colecciones de datos, para después acceder a los valores mediante {{}}.

En el ejemplo podemos ver como también podemos utilizar los métodos del componente, en este caso llamando al método getPartidosVotos() al clicar sobre una linera de nuestra tabla y pasándole los valores necesarios.

Otra de las partes importantes es mostrar los resultados en una gráfica, para ello se usa la librería Chart.js, es una librería JavaScript que nos permite dibujar diferentes tipos de gráficos simples pero flexibles [23]. Es un proyecto abierto mantenido por la comunidad en el que puedes representar tus datos con hasta 8 tipos diferentes de

gráficos y además es *responsive*. También existe un conjunto de directivas que facilita el uso de esta librería llamado ng2-charts[14].

El primer paso es instalar las librerías ng2-charts y chart.js, aunque esto generó un problema, como vemos a continuación:

```
WARNING in ./node_modules/ng2-charts/fesm5/ng2-charts.js 230:54-72 "export '
ɵɵdefineInjectable' was not found in '@angular/core'
```

Ilustración 52: Error con la instalación de las librerías para el uso de gráficas

Y esto es debido a que la última versión de ng2-charts está pensada para Angular 8, en el cual *defineIndejectable* está en desuso y lo intenta reemplazar con *ɵɵdefineInjectable,* pero en Angular 7 *ɵɵdefineInjectable* no existe, la solución más fácil es utilizar la versión ng2-charts 2.2.3

Una vez solventado este problema podemos empezar a usarla, y como ya comentamos al principio tendremos que incluir el modulo en el *array imports* de nuestro fichero *app.module.ts*.

Después en el componente donde vayamos a usar las gráficas crearemos 3 atributos:

```
// Grafica Doughnut
public doughnutChartType: ChartType = 'doughnut';

public doughnutChartLabels: Label[] = [];

public doughnutChartData: number[] = [];
```

Ilustración 53: Atributos para la creación de la gráfica

En el primero de ellos, *doughnutChartType*, indicamos el tipo de gráfico que vamos a usar, en este caso es un gráfico de *donuts* o rosquilla. Los otros atributos son dos *arrays, doughnutChartLabels* y *doughnutChartData*, los cuales representan los nombres de la leyenda y los valores para el grafico respectivamente y en ellos cargaremos los nombres de los partidos y los votos como vemos a continuación.

[14] ng2-charts: https://valor-software.com/ng2-charts/

```
computarReparto(metodoSeleccionado: string, esc: number): void {
  this.doughnutChartLabels = [];
  this.doughnutChartData = [];

  this.electionsApi
    .computarReparto(this.mapPartidosVotos, metodoSeleccionado, esc)
    .subscribe((mapPartidosEsc: { [name: string]: number }) => {
      console.log(mapPartidosEsc);
      for (const key in mapPartidosEsc) {
        if (mapPartidosEsc.hasOwnProperty(key)) {
          this.doughnutChartLabels.push(key);
          this.doughnutChartData.push(mapPartidosEsc[key]);
        }
      }
    });
}
```

Ilustración 54: Método que realiza el reparto de escaños y cargar los resultados en la gráfica

Este método a partir del número de escaños, el método seleccionado y el mapa con los partidos y votos, hace uso del servicio *ElectionsApi* para obtener un mapa de partidos y sus correspondientes escaños, el cual recorremos cargando los datos en los *arrays* mencionados antes.

Por ultimo solo tendremos que acceder a esos atributos desde la vista de la siguiente manera:

```
<!-- Grafica -->
<div class="col-xs-6 col-sm-6 col-md-6" *ngIf="doughnutChartData.length > 0">
  <div style="display: block">
    <canvas baseChart
      [data]="doughnutChartData"
      [labels]="doughnutChartLabels"
      [chartType]="doughnutChartType">
    </canvas>
  </div>
</div>
```

Ilustración 55: Creación de la gráfica desde la vista del componente

Otro aspecto importante es la protección de rutas en Angular esto se hace mediante *guard*, que son interfaces que protegen las rutas en Angular permitiendo o no el acceso a una ruta o no en función de valor true o false devuelto por una clase que implementa la interfaz.

Para generar el *guard* se utiliza el comando:

```
C:\Users\Marta\IdeaProjects2\clienteAngular>ng g g seguridad/autenticacion
? Which interfaces would you like to implement? (Press <space> to select, <a>
 to toggle all, <i> to invert selection)
CREATE src/app/seguridad/autenticacion.guard.spec.ts (400 bytes)
CREATE src/app/seguridad/autenticacion.guard.ts (257 bytes)
```

Ilustración 56: Comando para generar guard

Hay varios tipos de *guard,* pero en este caso solo se va a utilizar el *CanActivate*, en el que simplemente se verificará si el usuario está *logeado* comprobando el *currentUser* de la siguiente forma:

```
canActivate() {
  if (this.usuariosApi.getCurrentUser()) {
    // login true
    return true;
  } else {
    this.router.navigate(['usuario/login']);
    return false;
  }
}
```

Ilustración 57: Implementación del guard

Es bastante sencillo, utilizando el servicio UsuarioApi obtengo el *currentUser* y en el caso de que no haya será redirigido al *login*.

Una vez hecho esto tendremos que utilizarlo en nuestro fichero *app-routing.module.ts*, para ello lo importaremos y añadiremos un parámetro extra a las rutas que deseamos proteger.

```
{
  path: 'usuario',
  component: UsuarioRegistradoComponent,
  canActivate: [AutenticacionGuard]
},
```

Ilustración 58: Uso del *guard* para proteger una ruta

El parámetro extra como muestro en la Ilustración 58 es *canActivate*, al que le pasamos nuestro *guard*. De esta forma podemos proteger todas as rutas que queramos.

5. PRUEBAS

En el módulo de persistencia las pruebas que llevadas a cabo son test unitarios con *JUnit* para probar el correcto funcionamiento de los DAO, como podemos ver en el siguiente ejemplo sencillo:

```
@Test
public void crearUsuario() {
    LOG.info("Comprobando la funcion UsuarioDAO#crearUsuario");
    usuarioDAO.crearUsuario(usuarioCrear, claveCrear);
    Assertions.assertTrue(usuarioDAO.existe(usuarioCrear, claveCrear));
}
```

Ilustración 59: Ejemplo test unitario para crear un usuario.

En total para UsuarioDAO se han realizado 4 test y para VotacionDAO se han realizado 7 test

Módulo / Clase	Descripción de la prueba	Resultado
Persistencia/UsuarioDAO	Creación de usuario	SUPERADO
Persistencia/UsuarioDAO	Existencia de usuario	SUPERADO
Persistencia/UsuarioDAO	Modificación de la clave del usuario	SUPERADO
Persistencia/UsuarioDAO	Borrado de un usuario	SUPERADO
Persistencia/VotacionDAO	Creación de una votación	SUPERADO
Persistencia/VotacionDAO	Obtención de una votación	SUPERADO
Persistencia/VotacionDAO	Borrado de una votación	SUPERADO
Persistencia/VotacionDAO	Añadir partidos y votos a una votación	SUPERADO
Persistencia/VotacionDAO	Obtener los partidos y votos de una votación	SUPERADO
Persistencia/VotacionDAO	Modificar los partidos y votos de una votación	SUPERADO
Persistencia/VotacionDAO	Borrar los partidos y votos de una votación	SUPERADO

Tabla 8: Tabla de pruebas del módulo Persistencia

En el módulo del servicio REST se me ha empleado la herramienta *Postman* para realizar las pruebas de los tres controladores. A continuación, podemos ver un ejemplo:

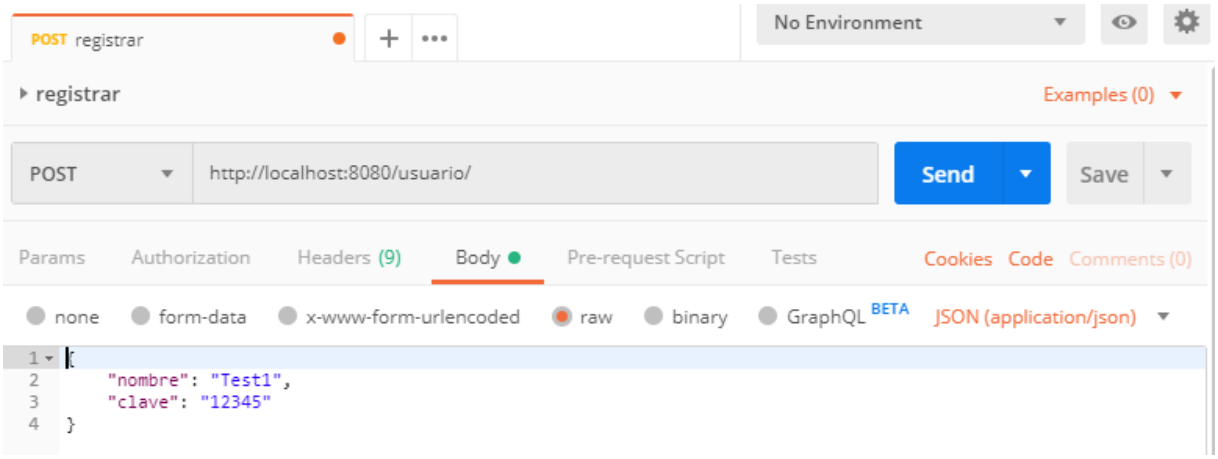


Ilustración 60: Ejemplo test con postman – parámetros de entrada

En esta prueba podemos ver como se crea un usuario con nombre “Test1” y clave “12345”, de la cual obtenemos la siguiente respuesta:

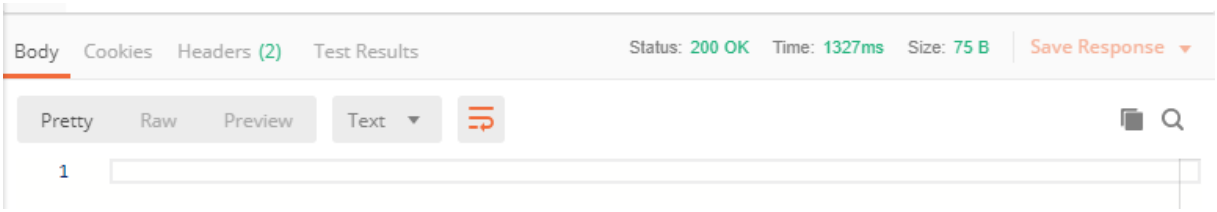


Ilustración 61: Ejemplo test con postman – respuesta

En total para el controlador de usuarios se han creado 4 consultas, para el controlador de votaciones 8 consultas y para el controlador *Election* 2 consultas. Mediante estas consultas, en *Postman,* puedo hacer todas las pruebas que quiera con diferentes valores.

URLs	Método	Descripción	Resultado
http://localhost:8080/usuario/	POST	Creación de usuario	SUPERADO
http://localhost:8080/usuario/pepe	POST	Existencia de usuario	SUPERADO
http://localhost:8080/usuario/juan?clave=123	PUT	Modificación de la clave del usuario	SUPERADO

http://localhost:8080/usuario/juan	DELETE	Borrado de un usuario	SUPERADO
http://localhost:8080/votacion/eleccion5/usuario/pepe	POST	Creación de una votación	SUPERADO
	PUT	Modificación de votación	
http://localhost:8080/votacion/2	GET	Obtención de una votación	SUPERADO
http://localhost:8080/votacion/usuario/pepe	GET	Obtención de todas las votaciones de un usuario	SUPERADO
http://localhost:8080/votacion/1	DELETE	Borrado de una votación	SUPERADO
http://localhost:8080/votacion/6/partido	POST	Añadir partidos y votos a una votación	SUPERADO
http://localhost:8080/votacion/6/partido	GET	Obtener los partidos y votos de una votación	SUPERADO
http://localhost:8080/votacion/2/partido/partido1	PUT	Modificar los partidos y votos de una votación	SUPERADO
http://localhost:8080/votacion/2/partido/partido2	DELETE	Borrar los partidos y votos de una votación	SUPERADO
http://localhost:8080/compute/metodo/Webber?esc=16	POST	Obtener el reparto de escaños	SUPERADO
http://localhost:8080/compute/metodos/divisor	GET	Obtener un lista de métodos por tipo	SUPERADO

Tabla 9: Tabla de pruebas del servicio REST

Finalmente se ha probado en un navegador que el cliente cumple con todos los criterios de aceptación de las historias de usuario.

6. CONCLUSIONES

Se ha logrado cumplir con los objetivos inicialmente propuestos:

Se hizo un estudio de las soluciones existentes, en este caso otras aplicaciones web que sirvieran para repartir los votos en escaños, en este estudio se vieron los puntos positivos y negativos de estas soluciones y se identificaron necesidades que estas tenían y posibles mejoras.

Para cumplir con estas necesidades se ha diseñado un sistema siguiendo una metodología ágil de desarrollo en este caso *Scrum*, en el cual aplicamos una arquitectura Cliente-Servidor.

Además, en este caso se han seleccionado dos entornos de desarrollo web diferentes ya que cada uno se adaptaba mejor a una parte, por un lado, para el servidor he utilizado *Intellij*, mientras que para el cliente he optado por *Visual Code Studio*. Con estos dos IDEs se ha implementado un prototipo de aplicación web que satisface las necesidades determinadas inicialmente.

Como tecnologías principales, por un lado, *Spring Framework* para el servidor es el *framework* de Java más extendido y utilizado, reduce al máximo su configuración inicial dejando que el programador se centre en su código, esto lo facilita aún más su módulo *Spring Boot*. Además, sigue una filosofía modular pudiendo añadir módulos a medida que los necesitemos y que a aplicación sea mayor, esto facilita el mantenimiento y las ampliaciones futuras de la aplicación.

Y por otro lado Angular que es un *framework* de *JavaScript* gratuito y *Open Source* mucho más reciente pero muy potente cuya función es facilitar la creación de aplicaciones web modernas. Su principal lenguaje de programación es *TypeScript* que nos ayuda mucho con el desarrollo, ya que toda la documentación está hecha con ejemplos en *TypeScript* y esto facilita el mantenimiento y las ampliaciones futuras de la aplicación. Además, ya hablamos de los componentes Angular que hacen más fácil la reutilización de código y el modularidad.

En cuanto a la metodología seguida, *Scrum* es una metodología de desarrollo ágil que es más rápida que las tradicionales y mucho más flexible a los cambios, al ir

trabajando en *Sprints* se pueden ir haciendo pruebas y asegurándose el funcionamiento de la aplicación Sprint a Sprint.

También he tenido la oportunidad de trabajar con una librería de un tercero que, si bien es compleja, está bien documentada, lo que facilita su uso. Hay mucho código en internet de buena calidad, disponible para todos y que ahorra mucho trabajo.

Desde mi punto de vista personal, el uso de estas tecnologías ha ayudado en el cumplimiento de los objetivos, y he logrado poner en práctica, y retomar, mis conocimientos de *Java* y *Spring* adquiridos durante el Grado viendo como todo lo estudiado se pone en práctica en el mundo laboral.

Además, durante la realización de este proyecto he podido conocer nuevas herramientas y *frameworks*, en mi caso me ha gustado mucho trabajar con Angular, he llegado a ver el potencial que tiene, toda la información que hay sobre él en internet pudiendo resolver las dudas fácilmente y poder avanzar en el proyecto

También me ha gustado trabajar con nuevos IDEs diferentes a los que he empleado durante el Grado, aunque podría haber usado los habituales, como Eclipse o *NeatbBans*, quería probar nuevos y ver como se adaptaban a las *frameworks* y lenguajes empleados. Y he podido comprobar cómo se adaptaban perfectamente a estos, las numerosas extensiones que tenían y como facilitan el trabajo al programador.

Bibliografía

[1] A. Álvarez, R. De las Heras y C. Lasa, Métodos ágiles y Scrum (Manuales imprescindibles), Anaya multimedia, 2011.

[2] R. S. Pressman, Ingeniería Del Software. Un Enfoque Práctico. Séptima Edición, McGRAW-HILL, 2010.

[3] c. d. Wikipedia, «Sistema Electoral,» [En línea]. Available: https://es.wikipedia.org/wiki/Sistema_electoral.

[4] F. Pukelsheim, Proportional Representation. Apportionment Methods and Their Applications, Springer International Publishing, 2014, 2017.

[5] c. d. Wikipedia, «Fórmula electoral,» [En línea]. Available: https://es.wikipedia.org/wiki/Fórmula_electoral.

[6] J. Webber, S. Parastatidis y I. Robinson, REST in Practice, O'Reilly Media, 2010.

[7] C. Walls, Spring in Action. Fifth Edition, Manning Shelter Island, 2019.

[8] C. Walls, Spring Boot in Action, Manning Shelter Island, 2016.

[9] c. d. Wikipedia, «IntelliJ IDEA,» [En línea]. Available: https://es.wikipedia.org/wiki/IntelliJ_IDEA.

[10] c. d. Wikipedia, «SonarQube,» [En línea]. Available: https://es.wikipedia.org/wiki/SonarQube.

[11] c. d. Wikipedia, «Java Persistence API,» [En línea]. Available: https://es.wikipedia.org/wiki/Java_Persistence_API.

[12] [En línea]. Available: https://www.oracle.com/technetwork/es/articles/java/expresiones-lambda-api-stream-java-2633852-esa.html.

[13] c. d. Wikipedia, «MySQL,» [En línea]. Available: https://es.wikipedia.org/wiki/MySQL.

[14] H. Beyer, User-Centered Agile Methods, Morgan & Claypool, 2010.

[15] c. d. Wikipedia, «MoSCoW method,» [En línea]. Available: https://en.wikipedia.org/wiki/MoSCoW_method.

[16] «Telefónica,» [En línea]. Available: http://www2.fsc.ccoo.es/comunes/recursos/53435/doc298110_Tablas_salariales_2019.pdf. [Último acceso: 2019].

[17] M. Keith y M. Schincariol, Pro JPA 2: Mastering the Java™ Persistence API, APRESS, 2009.

[18] c. d. Wikipedia, «JUnit,» [En línea]. Available: https://es.wikipedia.org/wiki/JUnit.

[19] A. Ranganathan, «cross-site xmlhttprequest with CORS,» 6 Julio 2009. [En línea]. Available: https://hacks.mozilla.org/2009/07/cross-site-xmlhttprequest-with-cors/.

[20] c. d. Wikipedia, «Angular (framework),» [En línea]. Available: https://es.wikipedia.org/wiki/Angular_(framework).

[21] «Angular CLI,» [En línea]. Available: https://cli.angular.io/.

[22] «Angular,» [En línea]. Available: https://angular.io/docs.

[23] «Chart.js,» [En línea]. Available: https://www.chartjs.org/.

Apéndice I. Manual de Instalación del sistema

Para la instalación de la parte del servidor hay que instalar el siguiente software:

- MySQL (GPL)
  - Nombre usuario (por defecto root): user
  - con clave: mysql
  - Nombre base de datos: votacion
- MySQL Workbench (GPL)
- Java 11
- JDK
- En este caso se ha usado como IDE IntelliJ IDEA incluye su propio servidor de aplicaciones.
- Al ser una aplicación generada con Spring Boot solo hay que lanzar el jar:
  - *java -jar election-rest-0.0.1-SNAPSHOT.jar*
- En caso de error revisar la dependencia de la libreria de tercero, jar incluido en el material del entregado

Para la instalación de la parte del cliente hay que instalar el siguiente software:

- El primer paso es instalar la última versión de Node.js y su gestor de paquetes NPM, desde su página oficial https://nodejs.org/es/download/
- Una vez instalado NPM ya podemos instalar Angular CLI siguiendo las instrucciones de su página oficial https://cli.angular.io/
- Situarse en el directorio del cliente. Ej:
  - *cd clienteAngular*
- Actualizar las dependencias:
  - *npm i -g npm-check-updates*
  - *npm-check-updates -u*
- Ejecutar la aplicación Angular:
  - *ng serve*

Nota: usar cualquier línea de comandos para ejecutar las instrucciones de este apartado.

Apéndice II. Manual de Usuario

En este manual de usuario se describen las funcionalidades implementadas, se aportan capturas de pantalla para mayor aclaración.

Inicio

Cuando el usuario accede a la aplicación se encuentra con la pantalla inicial que cuenta con las siguientes opciones:

Ilustración 62: Página de inicio

En esta pantalla el usuario tendrá 3 opciones que serán:

- Registrarse si accede desde el botón registrar del *navbar*.
- Acceder con sus credenciales si accede desde el botón *login* del *navbar* o desde el botón central usuario registrado.

- Entrar a una simulación sin registrarse si accede desde el botón central usuario sin registrar.

Registro

Si decide registrarse pasará a una pantalla con un formulario como vemos a continuación.

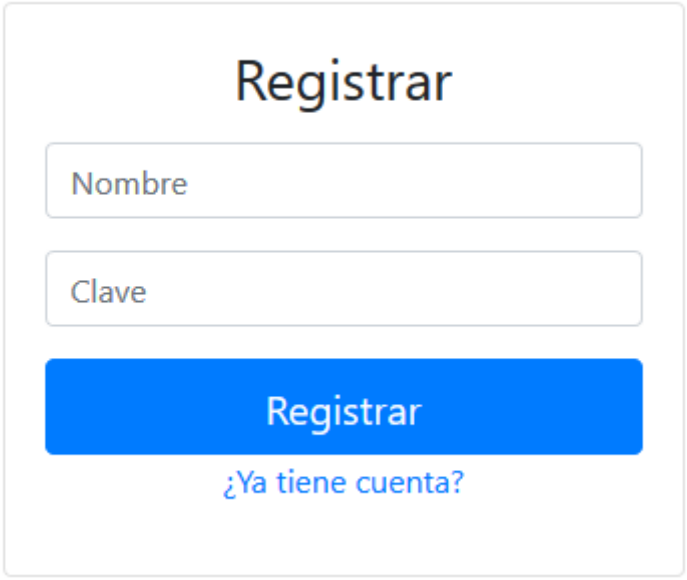


Ilustración 63: Formulario de registro

En esta pantalla el usuario debe introducir el nombre y la clave correspondientes, y al pulsar el botón registrar entrará a la pantalla de usuario registrado. También hay una opción "¿Ya tiene cuenta?" por si el usuario ya está registrado que pueda acceder directamente a la pantalla de *login* que veremos posteriormente.

Login

Si el usuario accede desde el botón *login* del *navbar* o desde el botón central usuario registrado de la página principal se encontrará con el siguiente formulario.

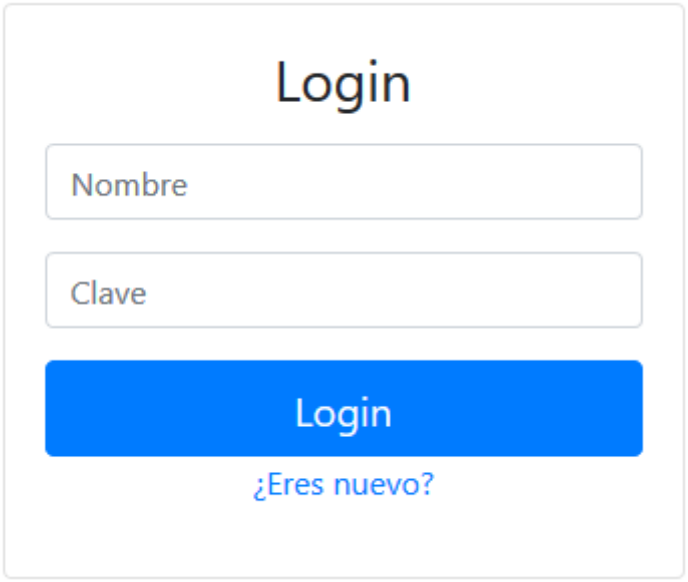


Ilustración 64: Formulario de *login*

Este formulario es muy similar al de registro, el usuario deberá introducir su nombre y clave, y al pulsar sobre el botón *Login* se le redirigirá a la pantalla de usuario registrado. También incluye una opción "¿Eres nuevo?" por si el usuario no está registrado que acceda directamente a la pantalla de registro.

Usuario Registrado

Tanto si accede desde el formulario de registro como desde el formulario de *login* se accederá a la pantalla de usuario registrado que vemos a continuación.

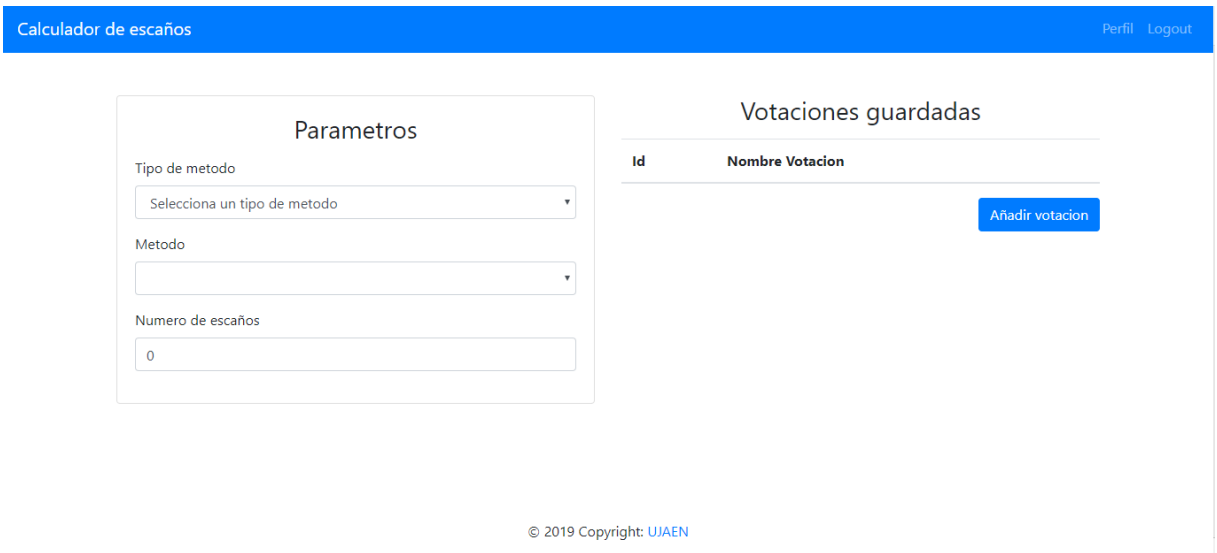


Ilustración 65: Pagina de usuario registrado inicial

En esta pantalla como vemos los botones del *navbar* han cambiado pudiendo ahora acceder al perfil el cual veremos después o cerrar sesión de la aplicación. Además, inicialmente tendremos a la izquierda un formulario para seleccionar el tipo de método, una vez seleccionado el tipo de método se cargarán los métodos correspondientes y podremos seleccionar un método de entre los propuestos y por ultimo introducir el número de escaños a repartir.

Nueva Votación

A la derecha tendremos una tabla con las votaciones guardadas del usuario, inicialmente vacía. Hay un botón para añadir una nueva votación, el cual abre un modal con un formulario en el que introducir el nombre de la votación y un botón para guardarla.

Ilustración 66: Formulario para crear una votación

Una vez guardada la votación podremos seleccionarla en la tabla para que aparezca una tabla nueva con los partidos y votos de esa votación, inicialmente vacía.

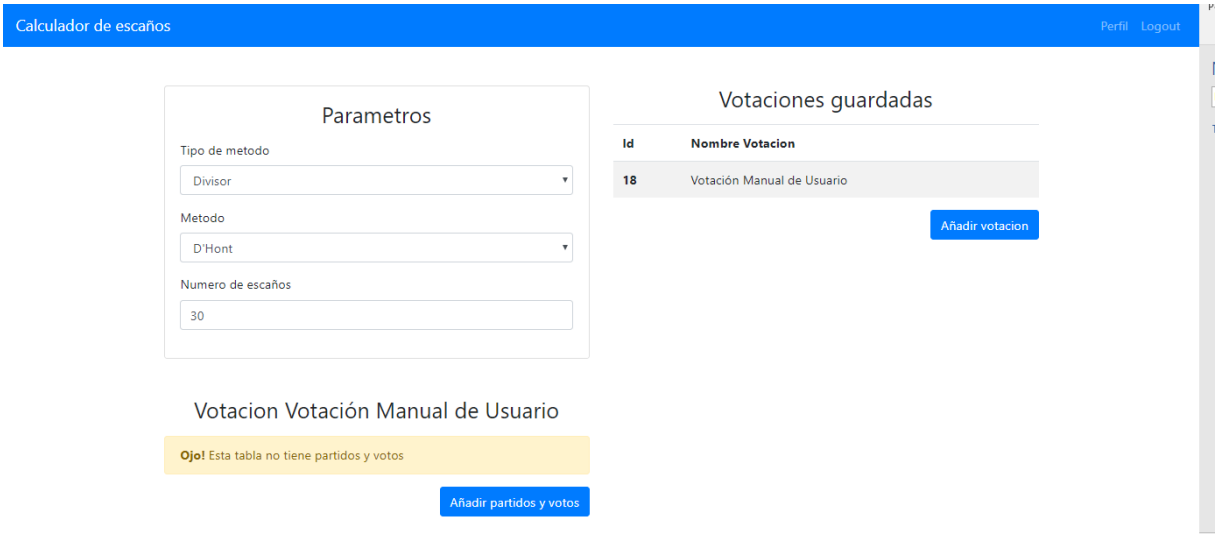


Ilustración 67: Página de usuario registrado con tabla de partidos y votos vacía

En esta nueva tabla tendremos un botón para añadir partidos y votos a la votación, el cual abrirá un modal con un formulario para hacerlo

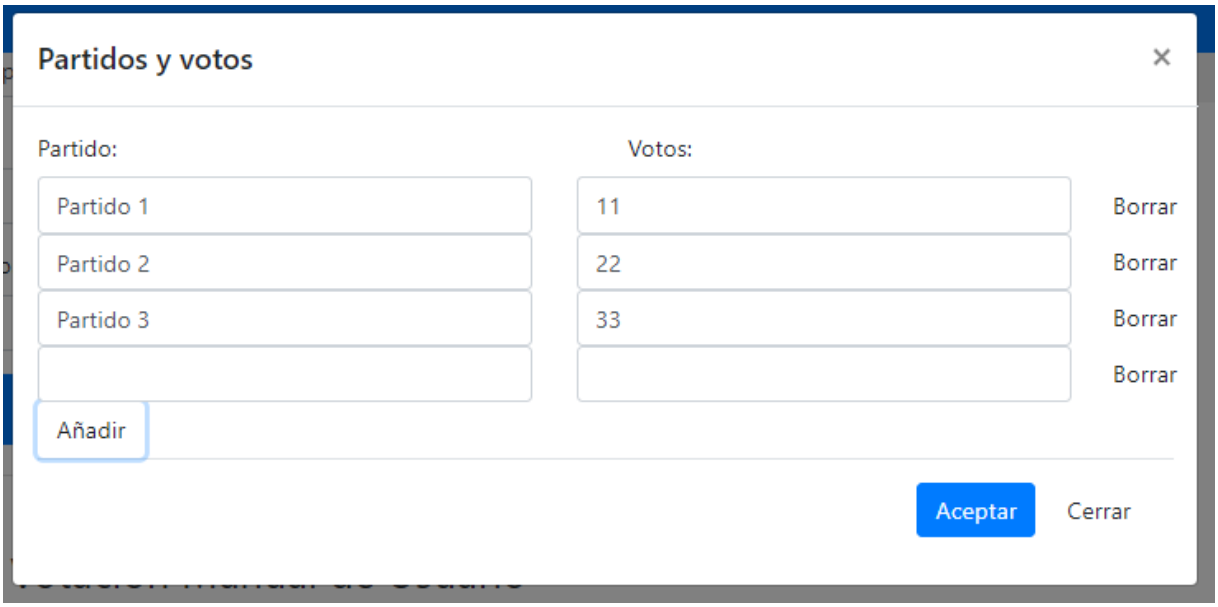


Ilustración 68: Formulario dinámico para añadir partidos y votos

En este formulario habrá varios botones, los de añadir y borrar son para poder aumentar o reducir el número de partidos a introducir, si se pulsa el botón cerrar, se cerrará el modal sin guardar los partidos y votos mientras que si se pulsa aceptar se guardaran los partidos y los votos de esa votación.

Reparto de escaños

Una vez seleccionado el método, la votación e introducido el número de escaños aparecerá un botón en el formulario para calcular el reparto de escaños quedando la página de la siguiente forma.

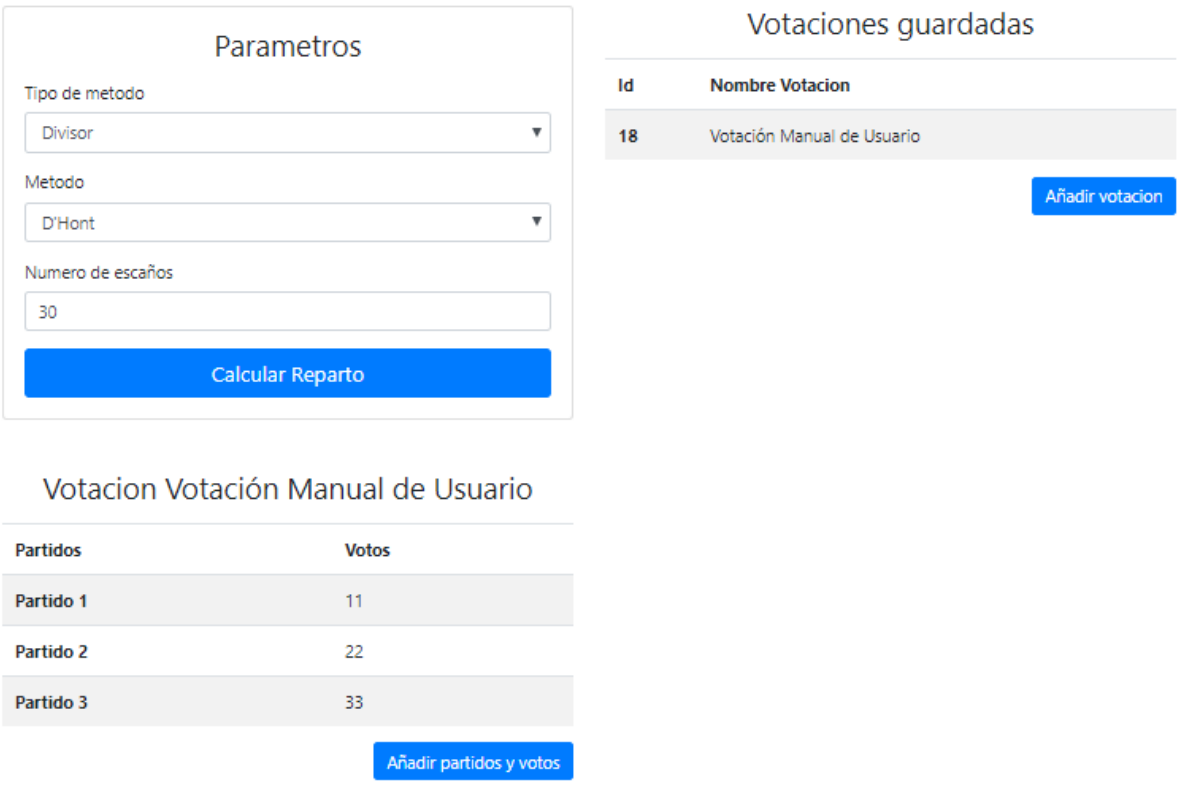


Ilustración 69: Página de usuario registrado con tabla de partidos y votos llena

En esta página podremos cambiar los parámetros del formulario o a votación y una vez pulsado el botón calcular reparto se añadirá una nueva columna con el número de escaños a la taba de los partidos y votos y se mostrará una gráfica con estos escaños también.

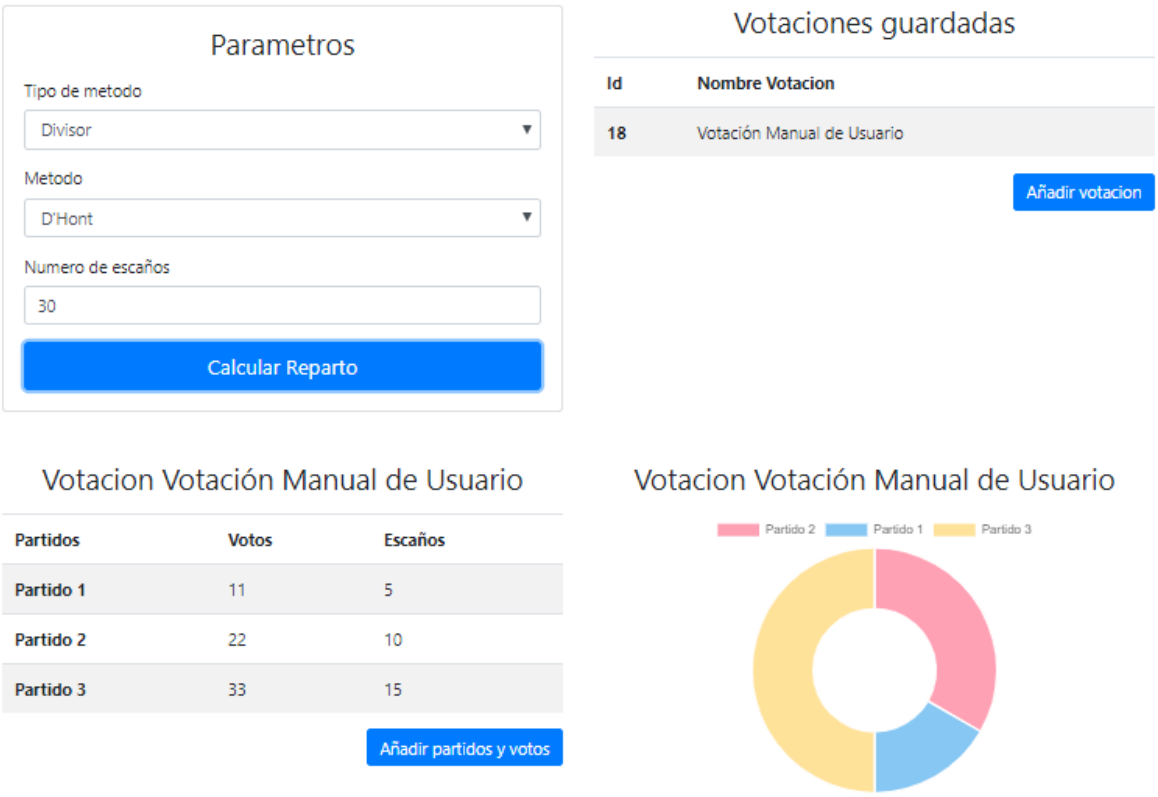


Ilustración 70: Página de usuario registrado con tabla de partidos, votos y escaños y con gráfica

Perfil

Los usuarios registrados como dijimos antes tienen acceso a su perfil mediante el botón de perfil en el *navbar*, accediendo a la siguiente página.

Ilustración 71: Página de perfil

En esta página podremos ver los datos del usuario a la izquierda y una tabla de votaciones a la derecha en la cual podremos borrar las o modificarlas. Además de un botón volver con el cual ir de nuevo a la página de usuario registrado.

Usuario sin registrar

La tercera opción de la página principal es acceder mediante el botón central a una simulación sin registrarse, con lo cual accederíamos a la siguiente página.

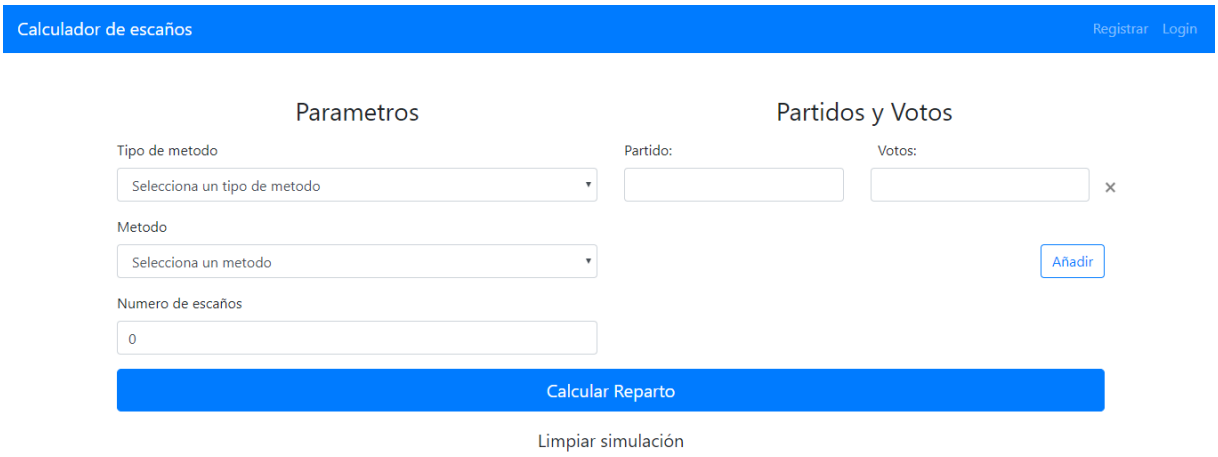


Ilustración 72: Pagina de simulación para usuario sin registrar vacía

En ella tenemos directamente un formulario para seleccionar el tipo de método, seleccionar el método, introducir el número de escaños a repartir e introducir tantos partidos con sus correspondientes votos como queramos. Una vez introducidos todos los parámetros podremos pulsar el botón calcular reparto para generar una tabla con los partidos, votos y escaños correspondientes, además de su gráfica asociada como podemos ver a continuación.

Ilustración 73: Pagina de simulación para usuario sin registrar con tabla y gráfico de los escaños

También disponemos del botón limpiar simulación para comenzar de cero.

Apéndice III. Descripción de contenidos suministrados

Hago entrega de un DVD que contiene:

- PDF con la memoria del proyecto
- Un archivo ".vpp" de Visual Paradigm que contiene todos los diagramas
- Ejecutable .jar de la librería de tercero
- Ejecutable .jar del servidor
- El código fuente del servidor comprimido en formato zip. También accesible desde:
    - https://github.com/jmcd0002/tfg.git
- El código fuente del cliente angular comprimido en formato zip: También accesible desde:
    - https://github.com/jmcd0002/clienteAngular.git