gesViña- Gestión de viñedos para autoconsumo

Memoria de Proyecto Final de Grado/Máster

Maestría Universitaria en Desarrollo de sitios y aplicaciones Web

Autor: Alejandro Viana Ríos

Consultor: Carlos Caballero González

Profesor: César Pablo Córcoles Briongos y Julià Minguillón Alfonso

Enero de 2020

Licencia

Dedicatoria/Cita

Me gustaría pedirle disculpas a mis alumnos, a quien tanto tiempo de preparación de clases les que robado para terminar satisfactoriamente este máster, y agradecerle su ayuda, tanto a mis padres, como a mi novia, que me lo dan todo sin pedir nada, sin los cuales nada de esto habría sido posible.

Abstracto

El objetivo del presente proyecto es el desarrollo de una aplicación web para gestionar un viñedo, permitiendo la introducción de muestras tomadas a lo largo del proceso de producción del vino, de los parámetros de la cosecha (kg recolectados, fecha, etc.) y del producto final (acidez, grados alcohólicos, litros obtenidos, etc.). A partir de los datos recogidos, la aplicación podría sugerir las actuaciones necesarias para obtener un producto de más calidad, así como crear gráficos que mostrasen la evolución de la cosecha.

Esta aplicación cuenta con un diseño adaptable a cualquier tamaño de pantalla para facilitar la toma de datos sobre el terreno con dispositivos portátiles como móviles o tabletas.

La aplicación se ha estructurado usando HTML5 y se ha maquetado mediante CSS3 y para su desarrollo se han usado los frameworks Laravel para el back-end y Angular para front-end

Palabras clave: vino, vitivinícola, cosecha, uva, laravel, angular

Abstract

The aim of this project is the development of a web application to manage a vineyard, allowing the introduction of samples taken throughout the wine production process, the parameters of the harvest (kg harvested, date, etc.) and the final product (acidity, alcoholic content, litres obtained, etc.). Based on the data collected, the application could suggest the actions needed to obtain a higher quality product, as well as create graphs showing the evolution of the harvest.

This application has a design that can be adapted to any screen size to facilitate data collection in the field with portable devices such as mobile phones or tablets.

The application has been structured using HTML5 and has been laid out using CSS3 and for its development the Laravel framework has been used for the back-end and Angular for the front-end

Keywords: wine, vintage, grape, laravel, angular

Translated with www.DeepL.com/Translator (free version)

Índice

Índice de figuras

Índice de tablas

1.-Introducción

1.1.- Contexto y justificación

Un familiar dispone de un viñedo pequeño que utiliza para autoconsumo y tiene interés en llevar un seguimiento de diversos parámetros en el proceso de la elaboración de vino para compararlos a lo largo de la misma cosecha o de las últimas y determinar los tratamientos que hay que aplicar para obtener un mejor producto.

Hay en el mercado herramientas generalistas que permiten gestionar cualquier negocio que podrían serle útiles. Podría usar un procesador de textos o una hoja de cálculo, pero estaría atándose a un producto y, quizá, a un sistema operativo concreto (caso de Microsoft Word en Microsoft Windows) o a una plataforma concreta (caso de LibreOffice calc para PC).

Otra opción es usar alguna de las aplicaciones webs adaptadas a la gestión de bodegas y viñedos como, por ejemplo vintiOs, Tractur o Isagri. Sin embargo, estas soluciones son demasiado ambiciosas para lo que se necesita en esta explotación familiar. Sería posible usarlas pero, si se deja de lado por un momento su coste, habría que tener en cuenta que dispondrían de una cantidad abrumadora de opciones, de las que se usaría una pequeñísima parte, lo que complicaría su aprendizaje y su uso.

Se ha decidido, por tanto, desarrollar una aplicación web que haga exactamente lo que necesita el familiar. En la presente documentación se desarrollará, de forma completa y detallada, el proceso de desarrollo de la aplicación web de gestión vitivinícola que se ha llamado gesViña.

2.- Descripción

En este TFM se desarrollará una aplicación web que permita hacer un seguimiento a las cosechas de un viñedo familiar de poca superficie cuya producción estará orientada hacia el autoconsumo. Por tanto, deberá ajustarse a lo que se necesita en una explotación con esas características, evitando opciones innecesarias que compliquen el uso del producto y encarezcan su desarrollo.

3.- Objetivos

Arthus SchopenHauer, filósofo alemán, dijo que “No hay ningún viento favorable para el que no sabe a qué puerto se dirige” así que, para que este proyecto llegue a buen puerto, es necesario definir sus objetivos, es decir, hacia qué puerto se dirige. Éstos se pueden dividir entre principales, irrenunciables, y secundarios, muy deseables, pero potencialmente no realizables, según avance el proyecto.

- Principales:
    - Gestionar las muestras en las tres fases: maduración, vendimia y fabricación.
    - Obtener gráficas comparativas de los parámetros principales tomados en las muestras de la última cosecha y de las tres últimas.
    - Gestionar las cosechas.

- Realizar un diseño funcional, agradable y adaptado a dispositivos pequeños para la recogida de datos sobre el terreno.

- Secundarios
  - Sugerir tratamientos y actuaciones para la mejora del producto (inicialmente era un objetivo principal).
  - Establecer alarmas para avisar de tareas requeridas.
  - Montar un sistema que permite que haya usuarios con los siguientes papeles:
    - administrador: tiene acceso completo al sistema y puede hacer cualquier operación con usuarios, cosechas, muestras y tratamientos.
    - operador: puede introducir información sobre cosechas, muestras y tratamientos, pero no eliminar ni editar.
    - Consultor: puede consultar la información.
  - Permitir al usuario elegir qué parámetros mostrar en la sección de estadísticas y el número de cosechas de las que tomas los datos (actualmente solo se muestra PH y acidez de la última y de las tres últimas cosechas).

4.- Marco teórico/escenario

Existen en el mercado aplicaciones adaptadas a la gestión de bodegas y viñedos como, por ejemplo, las siguientes:

- vintiOS: permite gestionar una bodega haciendo uso de un sistema de información geográfica, por lo que permite gestionar una explotación con varias parcelas. También es posible llevar un control de los productos fitosanitarios y consultar sus datos en el Ministerio de Agricultura y Medio Ambiente.
- Tractus: está enfocado en la trazabilidad del vino. Incluye también control de producción y control de insumos, permite llevar un control desde el viñedo hasta la venta final del vino al cliente y resulta de gran utilidad para el control de etiquetado, actividad requerida por los organismos certificadores.
- Solución Isagri para bodegas. Permite, entre otras opciones, lo siguiente:
  - Seguimiento técnico y trazabilidad de los viñedos.
  - Control de los costes de producción
  - Seguimiento de la mano de obra y de la maquinaria
  - Cartografía de las parcelas
  - Gestión de los proveedores de uva
  - Registro de las entradas de uva y albaranes de entrega
  - Control de rendimiento de cartillas del Consejo Regulador
  - Seguimiento gráfico de la trazabilidad ascendente y descendente
  - Órdenes de producción, embotellados y etiquetados

- Pedidos de clientes, presupuestos, albaranes y facturas
- Control de cobros, emisión y contabilización de remesas
- Gestión de clientes, proveedores, transportistas, etc.
- Gestión de empaquetado y etiquetado
- Optimización de cajas y palets
- Contabilidad fiscal y analítica

Parece obvio que estas soluciones son demasiado ambiciosas para lo que se necesita en esta explotación familiar. Si se deja de lado por un momento su coste, sería posible usarlas, pero habría una cantidad abrumadora de opciones, de las que se usaría una pequeñísima parte, lo que complicaría su aprendizaje y su uso. Es por ello que se ha decidido desarrollar una web a medida.

5.- Contenidos

La presente documentación contiene los siguientes puntos:

- Introducción al problema planteado y su solución. El primer paso de un proyecto es reconocer las necesidades y dar una solución inicial.
- Descripción del problema y la solución. Es necesaria desarrollar el problema y sus posibles soluciones con más detalle.
- Objetivos que se pretenden alcanzar. Para llegar a buen puerto es necesario definir hacia dónde dirigirse.
- Descripción de la situación del mercado relacionado con la solución. Hay que conocer a la competencia para saber de qué adolece.
- Contenido de este documento.
- Metodología seguida para alcanzar la solución.
- Arquitectura del proyecto, detallando las partes que la componen: una base de datos para almacenar toda la información, un backend encargado de ofrecer una API al frontend para hacer operaciones con la base de datos y un frontend para presentar los resultados al usuario e interaccionar con él. También se entrará en detalle en las tres partes explicando la arquitectura interna de los diferentes componentes.
- Plataforma de desarrollo utilizada para el proyecto, tanto aplicaciones como máquinas utilizadas.
- Planificación temporal seguida detallando, mediante un diagrama de Gantt, las tareas a realizar, las horas dedicadas a cada y las fechas de entrega.
- Proceso de trabajo donde, para cada una de las partes de la arquitectura se describirá, con cierto nivel de detalle, el desarrollo llevado a cabo. Se describirá el proceso de creación de la base de datos usando Laravel, así como el funcionamiento general de la API y de un solo controlador, para no repetir, ya que el funcionamiento del resto es muy similar. Asimismo, se describirá la funcionalidad más importante del frontend desarrollado en Angular.
- Diagramas de base de datos y de árbol de navegación

- Prototipos de baja resolución usados para guiar el diseño. Al ser un proyecto pequeño, no ha sido necesario realizar prototipos de alta resolución, ya que los de baja resolución resultaron ser muy parecidos al producto final.
- Perfil de los usuarios que se esperan que hagan uso de la aplicación.
- Técnicas de usabilidad se han tenido en cuenta en el desarrollo de la aplicación.
- Descripción de las pruebas realizadas sobre el producto, indicando herramientas, personas y protocolos utilizados.
- Requisitos de instalación y de uso.
- Instrucciones e instalación.
- Instrucciones de uso.
- Bugs detectados y su posible solución.
- Proyección futura. Todo proyecto puede ser continuado y completado tras su entrega.
- Presupuesto.
- Análisis de mercado.
- Conclusiones personales sobre el proyecto, la maestría y mi evolución personal.
- Anexos
    - Descripción de los directorios entregados en el proyecto.
    - Extractos de código fuente.
    - Información sobre bibliotecas externas utilizadas.
    - Guía de usuario.
    - Libro de estilo.
    - Bibliografía.
    - Vita.

6.- Metodología

El modelo elegido para el desarrollo de la aplicación ha sido el modelo en cascada. Como se puede ver en la Figura 1, dicho modelo consiste en concatenar las siguientes fases:

- Requisitos. Mediante una reunión con el cliente final, se analizarán sus necesidades y se plasmarán en un documento escrito que defina los objetivos de la aplicación y qué debe realizar. Es una fase de una importancia capital ya que una vez comenzado el diseño y aplicación, no es posible, ni deseable, cambiar los objetivos de la aplicación.
- Diseño. En esta fase tiene lugar la descomposición del sistema en partes menores, la definición de qué hace cada parte y el establecimiento de relaciones entre ellas. Se elegirán también las tecnologías en las que se basará el desarrollo, en este caso Laravel, MySQL, Angular, HTML y CSS y se configurará el en-

torno local y remoto para trabajar con ellas, en este caso XAMPP y Github como sistema de control de versiones. Además, se realizarán las siguientes tareas:

  - Se creará el mapa web, donde se definirán las pantallas que tendrá la aplicación y el flujo entre ellas.
  - Se crearán bocetos y prototipos que reflejen un estado inicial del estilo visual de la aplicación.
  - Se definirá el modelo de base de datos que usará la aplicación.

- Implementación. En esta fase se comienza a codificar la aplicación usando las tecnologías elegidas en el anterior punto.
- Verificación. En esta última fase se realizan pruebas para ver si la aplicación cumple los objetivos, y lo hace de forma eficiente, y, si tienen éxito, se realiza la implantación del sistema. Estas pruebas debe realizarlas el desarrollador, alguien externo a él y, finalmente, el cliente.

Tras terminar la aplicación, se completará la documentación del proyecto y se hará una presentación y un vídeo explicando su funcionamiento.

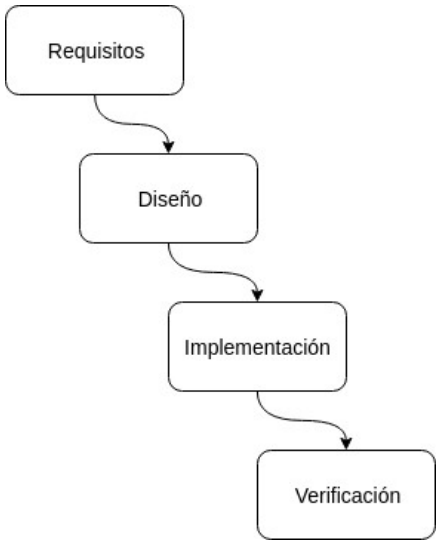


Figura 1: Modelo en cascada

7.- Arquitectura

La aplicación consta de varias partes (Figura 2):

- Una base de datos para almacenar la información.
- Un backend en PHP, programado mediante el framework Laravel, que se encarga de proporcionar una API REST con la que interaccionar con la base de datos.
- Un frontend, programado en TypeScript mediante el framework Angular, que llama a la API REST, obtiene los datos y los presenta de forma agradable al usuario.

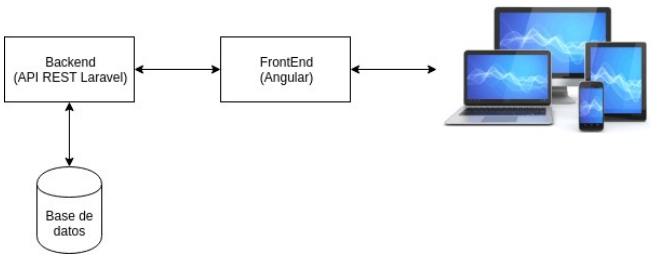


Figura 2: Arquitectura de la aplicación

7.1.- REST

REST es una arquitectura basada en estándares web que usa el protocolo HTTP donde cada componente es un recurso y se accede a ellos mediante métodos HTTP estándar. Una implementación de un servicio web REST sigue cuatro principios de diseño fundamentales:

- utiliza solo los métodos HTTP:
  - POST para crear un recurso en el servidor.
  - GET para obtener un recurso del servidor.
  - PUT para cambiar el estado de un recurso del servidor.
  - DELETE para eliminar un recurso del servidor.
- No mantiene estado. Cada petición HTTP es independiente del resto y contiene toda la información necesaria para ser ejecutada, por lo que ni cliente ni servidor necesitan recordar nada.
- Utiliza URIs con forma de directorios, usando un identificador único para cada recurso.
- Transfiere XML, JSON o ambos lo cual favorece la operatividad entre aplicaciones.

7.2.- Angular

Angular es un framework desarrollado por Google bajo el paradigma de código abierto. Está programado en TypeScript y su objetivo es facilitar la programación de aplicaciones web de una sola página. Para ello, se basa en clases “componente”, con cuyas variables y métodos se enlaza desde el código HTML de cada plantilla.

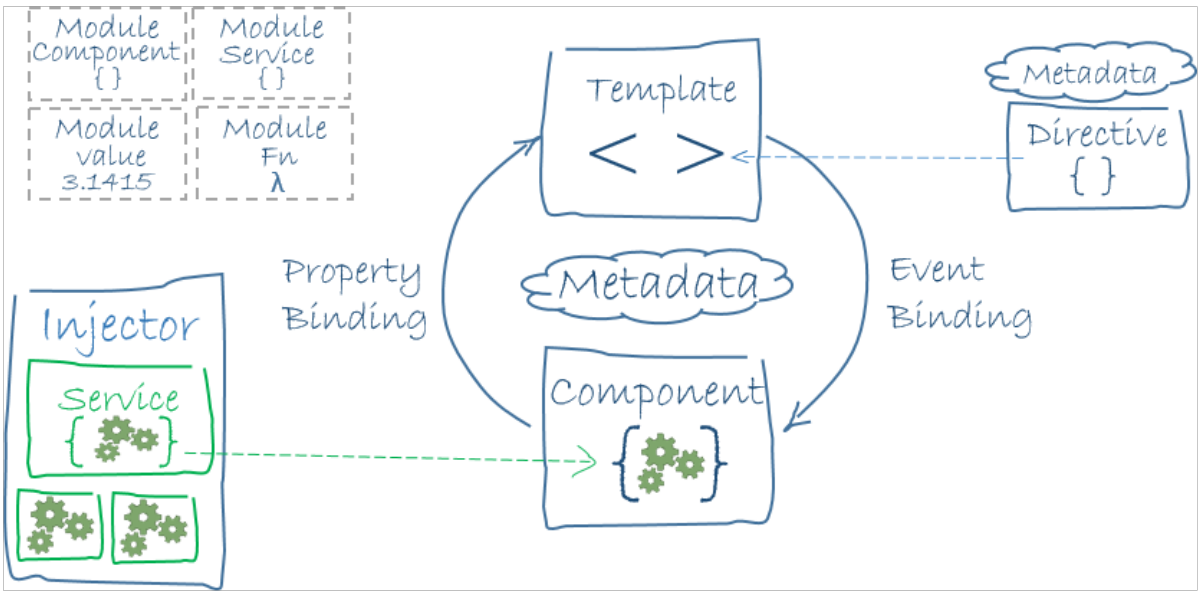


*Figura 3: Arquitectura de Angular (*Wikipedia*)*

7.3.- Laravel

Laravel es un framework PHP de código abierto que facilita la programación de aplicaciones. Está en constante evolución y tiene una muy buena documentación y mucho soporte. Se basa en la filosofía MVC (Modelo, vista, controlador) que divide la aplicación en tres partes:

- Modelo: Cada tabla de la base de datos tiene un modelo asociado, que se usa para interaccionar con ella. Se organiza en ficheros bajo el directorio app.
- Controlador: procesamiento de la información antes de mostrársela al usuario. Se organiza en fiche#ros bajo el directorio “app/Htpp/Controllers”.
- Vista: Recibe la información ya procesada del controlador y se la muestra al usuario. Se organiza en fiche-ros bajo el directorio “resources/views”.

7.4.- Base de datos

La base de datos de nuestra aplicación debe ser capaz de almacenar los siguientes datos:

- Parámetros ideales que debería tener cada tipo de vino según el tipo de uva: acidez, PH, grado alcohólico, volumen de 100 uvas, peso de 100 uvas, resultado visual, sulfuroso, ácido glucónico y ácido málico.
- En la fase de maduración, para cada muestra tomada: cata, acidez, PH, grado alcohólico y fecha de la muestra.
- En la fase de vendimia, para cada muestra tomada: PH, acidez, temperatura, densidad y fecha de la toma. En un futuro, para cada tratamiento realizado: Cantidad de metabisulfito aplicado (Sulfitado), cantidad de ácido tartárico aplicado (Acidificación), cantidad de levaduras aplicadas, cantidad de nutrientes aplicados (Nitrógeno fácilmente asimilable, NFA) y fecha de tratamiento.
- En la fase de conservación, para cada muestra tomada : acidez, sulfuroso, PH y fecha de la muestra.

8.- Plataforma de desarrollo

Desarrollo		
Servidor de bases de datos	MySQL 10.1.38	Integrado en XAMPP 7.3.2
Lenguaje backend	PHP 7.3.2	
Servidor web	Apache 2.4.38	
Interfaz gráfico con la base de datos	PhpMyAdmin 4.8.5	
Frontend	Angular Cli 7.0.4	
	NodeJS 12.13.1 y npm 6.13.4	
Backend	Laravel 5.8	
Bibliotecas externas	Bootstrap 4, ng2-chart, mat-angular, angular-flex	

Tabla 1: Aplicaciones usadas para desarrollar la aplicación

Circuitería	
Ordenador	Sony Vaio SVE1712L1EW, Intel core i7-3610QM, 8GB RAM, 256GB SSD
Sistema operativo	Ubuntu Gnome 18.04.3 LTS 64 bits Windows 10 (para Balsamiq)

Tabla 2: Circuitería usada en el desarrollo de la aplicación

Servicios	
Control de versiones y respaldo	GitHub.com
Diagrama de tablas de base de datos	Draw.io 12.4.8
Diagrama E/R	Yed live 4.4.6 (https://www.yworks.com/yed-live/)

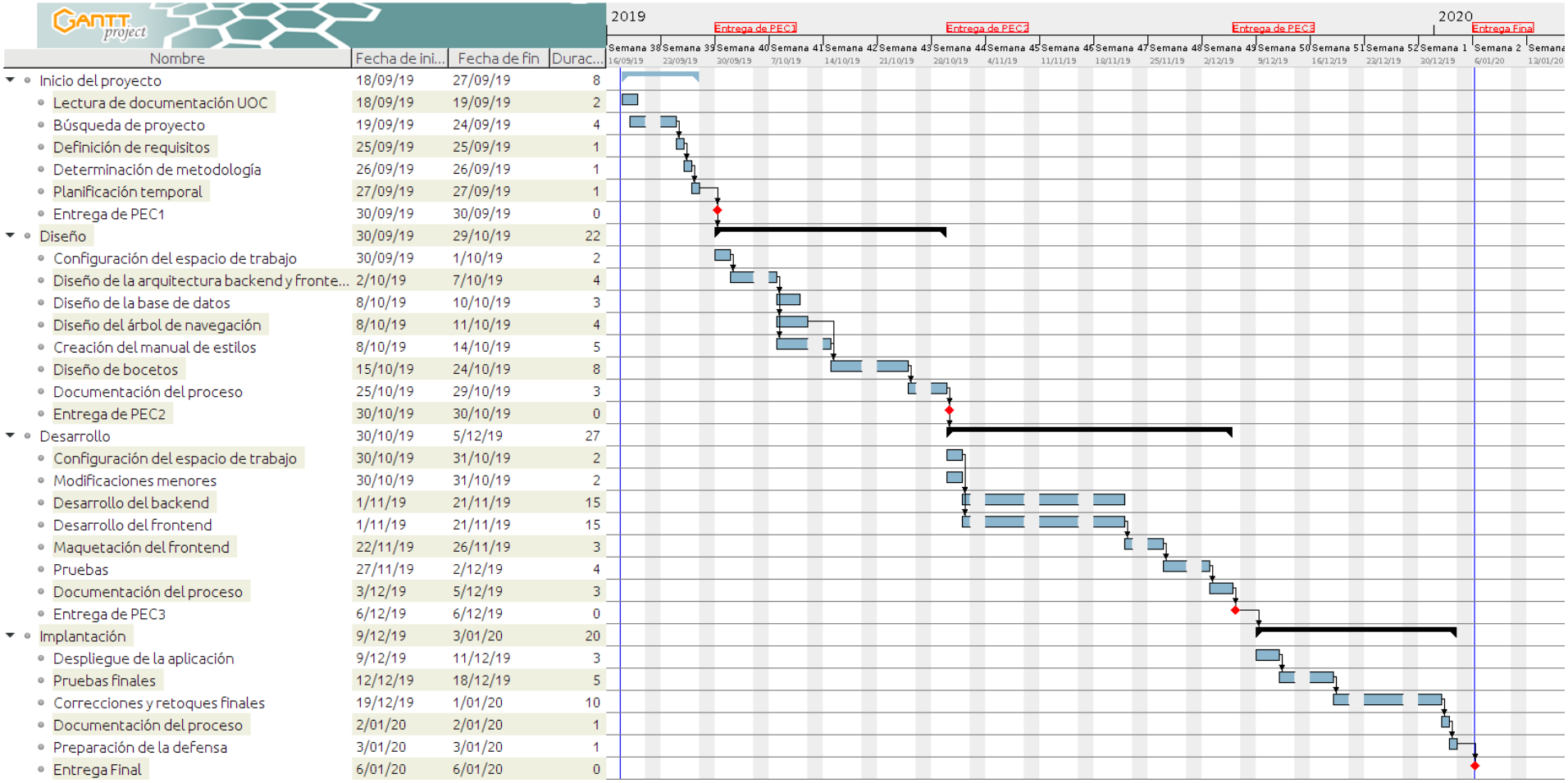
Tabla 3: Servicios usados en el desarrollo de la aplicación

Aplicaciones auxiliares	
Documentación	LibreOffice Writer 6.3
Presentación	LibreOffice Impress 6.3
Creación de maquetas	Balsamiq mockup 3.15
Navegadores para pruebas	Opera 65 y Firefox 71
Grabación de vídeo	Kazam 1.4.5
Edición de imágenes	Gimp 2.10.14
Pruebas de la API	Postman 7.14
Planificación	Gantt Project 2.8.10
Editor	Atom 1.31
Servidor web	Integrado en Laravel 5.8

Tabla 4: Aplicaciones que han ayudado a desarrollar

9.- Planificación

Teniendo en cuenta la tipología del proyecto, la metodología elegida y la división en PECs de la UOC y sus fechas de entrega, el proyecto se puede dividir en las siguientes tareas, tal y como se puede ver en la Figura 4:



Nombre	Fecha de ini...	Fecha de fin	Durac...
Inicio del proyecto	18/09/19	27/09/19	8
Lectura de documentación UOC	18/09/19	19/09/19	2
Búsqueda de proyecto	19/09/19	24/09/19	4
Definición de requisitos	25/09/19	25/09/19	1
Determinación de metodología	26/09/19	26/09/19	1
Planificación temporal	27/09/19	27/09/19	1
Entrega de PEC1	30/09/19	30/09/19	0
Diseño	30/09/19	29/10/19	22
Configuración del espacio de trabajo	30/09/19	1/10/19	2
Diseño de la arquitectura backend y fronte...	2/10/19	7/10/19	4
Diseño de la base de datos	8/10/19	10/10/19	3
Diseño del árbol de navegación	8/10/19	11/10/19	4
Creación del manual de estilos	8/10/19	14/10/19	5
Diseño de bocetos	15/10/19	24/10/19	8
Documentación del proceso	25/10/19	29/10/19	3
Entrega de PEC2	30/10/19	30/10/19	0
Desarrollo	30/10/19	5/12/19	27
Configuración del espacio de trabajo	30/10/19	31/10/19	2
Modificaciones menores	30/10/19	31/10/19	2
Desarrollo del backend	1/11/19	21/11/19	15
Desarrollo del frontend	1/11/19	21/11/19	15
Maquetación del frontend	22/11/19	26/11/19	3
Pruebas	27/11/19	2/12/19	4
Documentación del proceso	3/12/19	5/12/19	3
Entrega de PEC3	6/12/19	6/12/19	0
Implantación	9/12/19	3/01/20	20
Despliegue de la aplicación	9/12/19	11/12/19	3
Pruebas finales	12/12/19	18/12/19	5
Correcciones y retoques finales	19/12/19	1/01/20	10
Documentación del proceso	2/01/20	2/01/20	1
Preparación de la defensa	3/01/20	3/01/20	1
Entrega Final	6/01/20	6/01/20	0

Figura 4: Diagrama de Gantt

10.- Proceso de desarrollo

Una vez establecida la necesidad del proyecto, descrito su cometido, comparado con el mercado y definidos sus objetivos, el proyecto arrancó comprobando que se tenían todas las herramientas necesarias instaladas de anteriores asignaturas de esta maestría. Para desarrollar la aplicación se siguieron los siguientes pasos:

1. Diseño de la base de datos de tal forma que permita almacenar la información que necesita la aplicación.
2. Creación de una API REST para interactuar con la base de datos.
3. Programación del frontend para pedirle los datos a la API REST y presentárselos al usuario.

10.1.- Creación de la base de datos

Se creó un modelo ER de la base de datos teniendo en cuenta los requisitos y, a partir de él, se definieron las tablas (ver punto 7.4). Como la gestión de la base de datos recae en el backend y se va a usar Laravel, se creó un fichero por cada tabla con la sintaxis de Laravel y se migró, proceso mediante el cual Laravel se conecta a la base de datos y crea la estructura de las tablas según lo especificado en los ficheros.

Aunque se podría haber creado las tablas conectándose mediante consola al servidor de MySQL, creando una a una y rellenándolas con datos fila a fila, es mucho más cómodo usar Laravel. Este framework, mediante un proceso llamado migración, es capaz de crear varias tablas consecutivamente con solo describir su estructura en un fichero, y también es capaz de rellenarlas con tantas filas como se quiera de una forma rápida y cómoda, mediante un proceso llamado sembrado. Para ello, se han seguido los siguientes pasos:

- Creación de la base de datos y el usuario para acceder a ella usando phpMyAdmin.
- Configuración de Laravel para conectarse a la base de datos con las credenciales anteriores.
- Creación de las migraciones (Figura 5), un fichero por cada tabla describiendo, con la sintaxis de Laravel, la estructura de la base de datos.

```php
public function up()
{
    Schema::create('maduracion', function (Blueprint $table) {
        $table->bigIncrements('id_muestra'); //por defecto increments hace que sea clave primaria
        $table->unsignedBigInteger('id');
        $table->integer('cata');
        $table->integer('ph');
        $table->integer('acidez');
        $table->integer('grado_alcoholico');
        $table->date('fecha');
        $table->timestamps();
    });
}
```

Figura 5: Fichero de migración de la tabla "maduracion"

- Migración de la base de datos con el comando “php artisan migrate”, Laravel se conectará a la base de datos y creará las tablas según lo descrito en los ficheros.

- Creación de los ficheros de “sembrado” (Figura 6) que permiten llenar de datos las tablas creadas. Para ello se ha usado una biblioteca en PHP que genera datos aleatorios llamada faker (https://github.com/fzaninotto/Faker).

```
public function run()
{
  $faker=Faker::create();
  $insertar=10;

  $matriz_ids=cosechas::pluck('id_cosecha')->toArray();
  $num_ids=count($matriz_ids);

  for ($i=0; $i<$insertar; $i++){
    $maduracion= maduracion::create([
      'id_cosecha'=>$matriz_ids[$faker->numberBetween($min=0, $max=$num_ids-1)],
      'cata'=>$faker->numberBetween($min=1, $max=2),
      'ph'=>$faker->numberBetween($min=1, $max=3),
      'acidez'=>$faker->numberBetween($min=1, $max=4),
      'grado_alcoholico'=>$faker->numberBetween($min=1, $max=5),
      'fecha'=>$faker->date(),
    ]);
  }
}
```

Figura 6: Sembrado de datos en la tabla "maduracion"

- Sembrado de datos con el comando “php artisan db:seed”

10.2.- Creación de la REST API

Siendo el backend la parte encargada de interactuar con la base de datos y el frontend la encargada de interactuar con el usuario, se hace necesario un mecanismo para que haya comunicación entre ambas partes. Ese mecanismo lo proporciona el backend y consiste en una API, mediante la cual le presenta unas funciones perfectamente definidas al frontend para que haga las operaciones que necesite con la información de la base de datos. Ésta API se ha diseñado siguiendo el patrón REST, que sigue las siguientes premisas:

- Cualquier petición se hace mediante el protocolo HTTP.
- Los parámetros que definen qué información se solicita (por ejemplo, el identificador de un registro), cual se borra o cual se actualiza se pasan a través de la URL mediante el esquema http://dirección_IP/API/Método/parámetro1/parámetro2/…
- La información a insertar o modificar se pasa dentro del “body” de una petición HTTP.
- El método que va a recibir la petición HTTP en el backend se indica en la URL mediante el esquema http://dirección_IP/API/Método/parámetro
- Al enviar la petición HTTP solo se utilizan los siguientes métodos
  - GET, para obtener uno o más registros.
  - POST para crear un registro.

- PUT, para actualizar un registro.
- DELETE, para eliminar uno o varios registros.

- El resultado devuelve en formato JSON.
- El código de estado HTTP define el resultado de la operación: 200 todo bien, 404 no encontrado, etc.

Para implementar una API REST en Laravel hacen falta dos cosas:

- Un controlador que contenga métodos que le digan a Laravel qué consulta a la base de datos hacer cuando reciba una petición HTTP. Laravel determina el controlador a elegir en función de un parámetro en la URL. En la Figura 7, la función “mostrarTodos” obtiene todos los registros de la tabla “vendimia”.

```php
<?php

namespace App\Http\Controllers;
use App\vendimia;
use App\cosechas;

use Illuminate\Http\Request;

class controladorVendimia extends Controller
{
  public function mostrarTodos(){
      return vendimia::all();
  }
```

Figura 7: Función mostrarTodos del controlador "controladorVendimia"

- Un fichero de rutas (Figura 8) que contiene una asociación entre el método de llamada HTTP (GET, POST, PUT, DELETE), los parámetros pasador por la URL y qué método dentro de qué controlador se llamará cuando se produzca esa petición HTTP. En la Figura 8, cuando se produce una petición GET con una URL HTTP://localhost:8000/api/vendimia, el servidor laravel llamará a la función “mostrarTodos” del controlador “controladorVendimia”

```php
Route::get('vendimia', 'controladorVendimia@mostrarTodos');
Route::get('vendimia/{id}', 'controladorVendimia@mostrarUno');
Route::post('vendimia', 'controladorVendimia@almacenar');
Route::delete('vendimia/{id}', 'controladorVendimia@eliminar');

Route::get('conservacion', 'controladorConservacion@mostrarTodos');
Route::get('conservacion/{id}', 'controladorConservacion@mostrarUno');
Route::post('conservacion', 'controladorConservacion@almacenar');
Route::delete('conservacion/{id}', 'controladorConservacion@eliminar');
```

Figura 8: Algunas de las rutas en Laravel.

Por tanto, el siguiente paso consistió en crear los controladores de Laravel, que ejecutan las funciones para consultar la base de datos y llenar el fichero de rutas que le dirá a Laravel qué función de qué controlador ejecutar dependiendo de la llamada a la API.

Tras probar la API mediante postman y comprobar que efectivamente se hacen modificaciones y consultas sobre la base de datos, se pasó a desarrollar la parte de frontend

10.3.- Programación del frontend

Para programar el frontend se ha usado el framework angular en su versión 7.0.4. La programación del frontend, comenzó definiendo la estructura que se iba a seguir y los componentes de los que constaba, que cambiaría varias veces a lo largo del desarrollo.

El segundo paso fue crear un servicio que se conectaba a la API del backend y probar que funcionaba en un componente de prueba. Una vez que estuvo listo, se creó el primer módulo, muestras, que se incluyó en el fichero de rutas de Angular. Se programó para que se conectase a la API y se crearon tres sub-componentes, “maduración”, “vendimia” y “conservación”, que contienen un formulario para introducir datos de muestra y se probó que funcionaban.

El siguiente componente fue “configuración”, que me llevó bastante tiempo porque no tenía claro cuál era la mejor forma de proceder con este componente ni de presentarle los datos al usuario. Estuve haciendo pruebas bastante tiempo hasta que di con la forma que consideré más cómoda: redirigir por defecto al usuario a ese componente para obligarlo a tomar decisiones sobre la cosecha actual y sus parámetros. Más adelante vi la necesidad de que, además de redirigirlo, el sistema tomase la decisión automáticamente por no dejar al sistema en un estado inconsistente si el usuario decidía no tomar ninguna decisión.

Tampoco tenía claro si le debía dar todas las opciones disponibles sobre cosechas al usuario desde esta pantalla es decir, cambiar la cosecha por defecto, eliminar y crear. Finalmente, por no agobiar con demasiada información, opté por darle solo las opciones de configuración (al fin y al cabo, el usuario estaba en el apartado de configuración), dejando la de eliminar para otro componente (que llamaría “cosechas”).

Otra decisión que tomé en este componente fue la de crear componentes independientes para presentar los datos y para recogerlos mediante un formulario. Nacieron así dos componentes:

- “parametros” donde se presenta un formulario para introducir los parámetros ideales de un tipo de uva o editarlos si ya existen.
- “nueva_cosecha” donde se presenta un formulario para introducir datos sobre una nueva cosecha.

Se pasó a desarrollar el componente “cosechas” que muestra un listado de las cosechas actuales y permite eliminarlas y crear nuevas, para lo cual llama al mismo componente que “configuracion”, “nueva_cosecha”.

El último componente de la zona de usuarios identificados que desarrollaré fue “estadisticas”, donde hice uso de una biblioteca externa ng2-charts que toma datos de varias matrices y genera gráficos muy bonitos. Este componente me dio guerra varios días porque no fue fácil tomar los datos de la aplicación y adaptarlos a la forma que requiere.

Por último desarrollé los componentes para identificar al usuario (“acceso”) y de “registro”, que acceden a la API para obtener e insertar usuarios en la base de datos y hacen uso del servicio “usuario.service” e introduje en el resto de componentes una consulta al servicio “usuario.service” para comprobar si el usuario se ha identificado. En caso contrario, redirigen a la pantalla de identificación.

Completada la funcionalidad, pasé a la parte de presentación. En un principio usé angular-material, pero cambié a bootstrap 4, ya que me siento más cómodo. Elegí una serie de imágenes relacionadas con la temática, que se muestran en el carrusel de entrada y, a partir de ellas, saqué unos colores que apliqué a toda la página.

Por último estuve probando la interfaz y se la di a mi familiar para que la probase. Anoté los fallos y reparé los que detectamos.

Finalmente la estructura ha quedado divida en dos directorios: “shared”, con la funcionalidad común y vistas, con los componentes. El directorio “shared” está compuesto de lo siguiente (Figura 12):

- componentes. Contiene dos componentes que se usarán en el resto de componentes y módulos de la aplicación:
  - admin-layout, que contiene los elementos comunes (barra de navegación y pie de página) para los componentes en los que el usuario tiene que iniciar sesión.
  - Aterrizaje-disposicion, que contiene los elementos comunes (barra de navegación y pie de página) para los componentes en los que el usuario no tiene que iniciar sesión.
- Modelos, contiene ficheros que definen la estructura de los datos usados en la aplicación: cosecha, muestras en cualquiera de las fases, parámetros y usuarios.
- servicios, que contiene
  - API, que hace llamadas a la API REST.
  - Cosechas, que almacena la cosecha actual, dato que será necesario en algunos componentes para mostrar las muestras asociadas a dicha cosecha.
  - Sesion, que almacena datos sobre la sesión actual para que el usuario pueda ver las páginas para las que es necesario iniciar sesión.

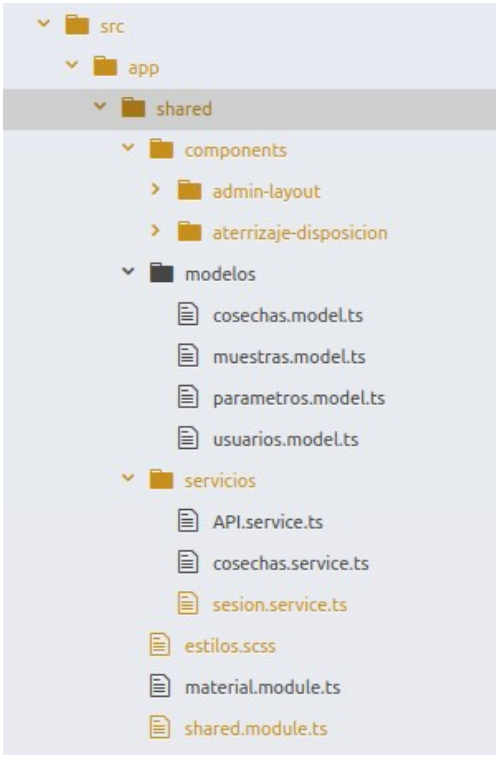


Figura 9: Estructura de directorios de Angular (1)

El otro directorio importante es “vistas” (Figura 10), que contiene los módulos y componentes que forman la aplicación y que son los siguientes:

- Aterrizaje. Contiene una página de aterrizaje con un carrusel (Figura 31). Hace uso del componente común “aterrizaje-disposicion” y desde él se puede acceder a dos componentes:
  - Acceso (Figura 32). Muestra un menú de acceso para identificar al usuario y que pueda acceder al contenido de la aplicación web.
  - Registro (Figura 33). Muestra un menú para registrarse tras lo cual se le da acceso al contenido de la aplicación web. (en una versión futura habrá un sistema de usuarios

- Configuracion. Muestra la configuración actual: cosecha actual sobre la que se va a trabajar (Figura 34) (la última introducida salvo que el usuario la cambie a mano (Figura 1) y los parámetros ideales sobre ese tipo de uva (en un futuro se usará para recomendar acciones y tratamientos en función de los valores obtenidos en las muestras y los de referencia). Pinchando en “definirlos” se llama al componente “parametros”
    - Parametros. Se muestra un formulario (Figura 36) para editar, si ya existen, o definir los parámetros ideales que debe tener una cosecha. En función de ellos se podrían recomendar acciones en un futuro.
- Cosechas (Figura 37). A él se accede pinchando en “cosechas” de la barra de navegación. Muestra las cosechas que hay en la base de datos y da la opción de eliminarlas y crear una nueva pinchando en “nueva”, para lo cual se llama al siguiente componente. Si se elimina la cosecha actual, marcada en rojo, el sistema elige automáticamente la más reciente como cosecha a.
    - nueva_cosecha (Figura 38). Muestra un formulario para almacenar en la base de datos una cosecha nueva y la elige como cosecha actual.
- Muestras. Muestra una pantalla donde se pueden ver las muestras que se han tomado de la cosecha actual en las tres fases (Figura 39). También permite eliminar una muestra o crear una nueva, para lo cual se llama a uno de los siguientes componentes, según sea de una fase u otra.

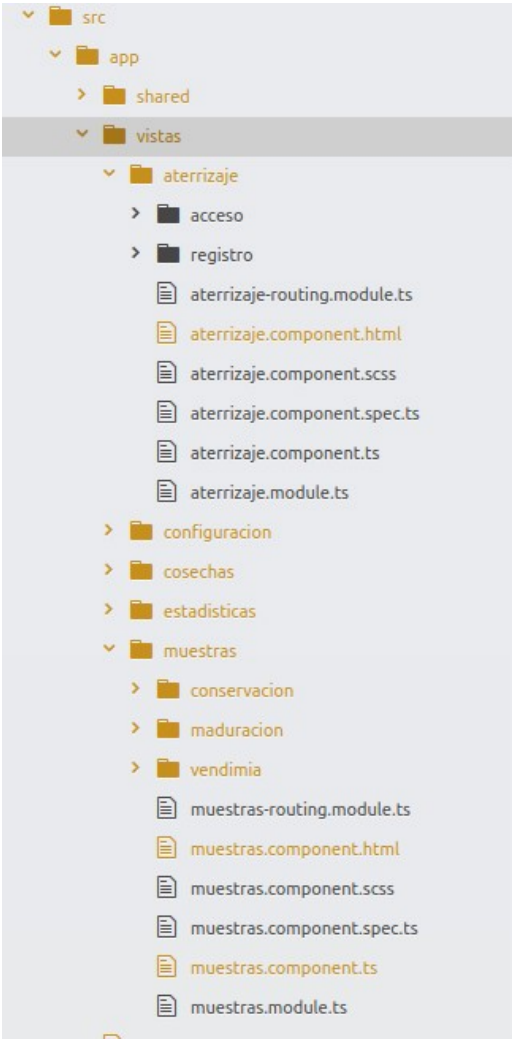


Figura 10: Estructura de directorios de angular (2)

    - Maduracion. Permite introducir la información sobre una muestra realizada en la fase de maduración (Figura 40).
    - Vendimia. Permite introducir la información sobre una muestra realizada en la fase de vendimia.
    - Conservacion. Permite introducir la información sobre una muestra realizada en la fase de conservación.
- Estadisticas. Muestra los siguientes gráficos:
    - los valores de PH de todas las muestras de todas las fases de la cosecha actual
    - los valores de PH de todas las muestras de todas las fases de las tres últimas cosechas.
    - los valores de acidez de todas las muestras de todas las fases de la cosecha actual
    - los valores de acidez de todas las muestras de todas las fases de las tres últimas cosechas.

  Para ello, hace uso del componente “grafico” que es donde se realiza el dibujo del gráfico (Figura 41)

11.- Diagramas

11.1.- Base de datos

El diagrama E/R resultante es el siguiente:

Figura 11: Diagrama E/R

El modelo anterior se traduce en las siguientes tablas (Figura 12):

- parámetros_ideales, cuya clave primaria, “tipo_vino”, es foránea a la tabla “cosecha”.
- Cosechas: con una clave primaria, “id_cosecha”, foránea a la tablas tratamientos, maduracion, vendimia y conservacion.
- maduracion, vendimia y conservacion: con una clave “id_cosecha”, foránea a la tabla “cosecha”.
- Tratamientos (para una versión futura): con una clave foránea “id_cosecha” a la tabla “cosechas”.

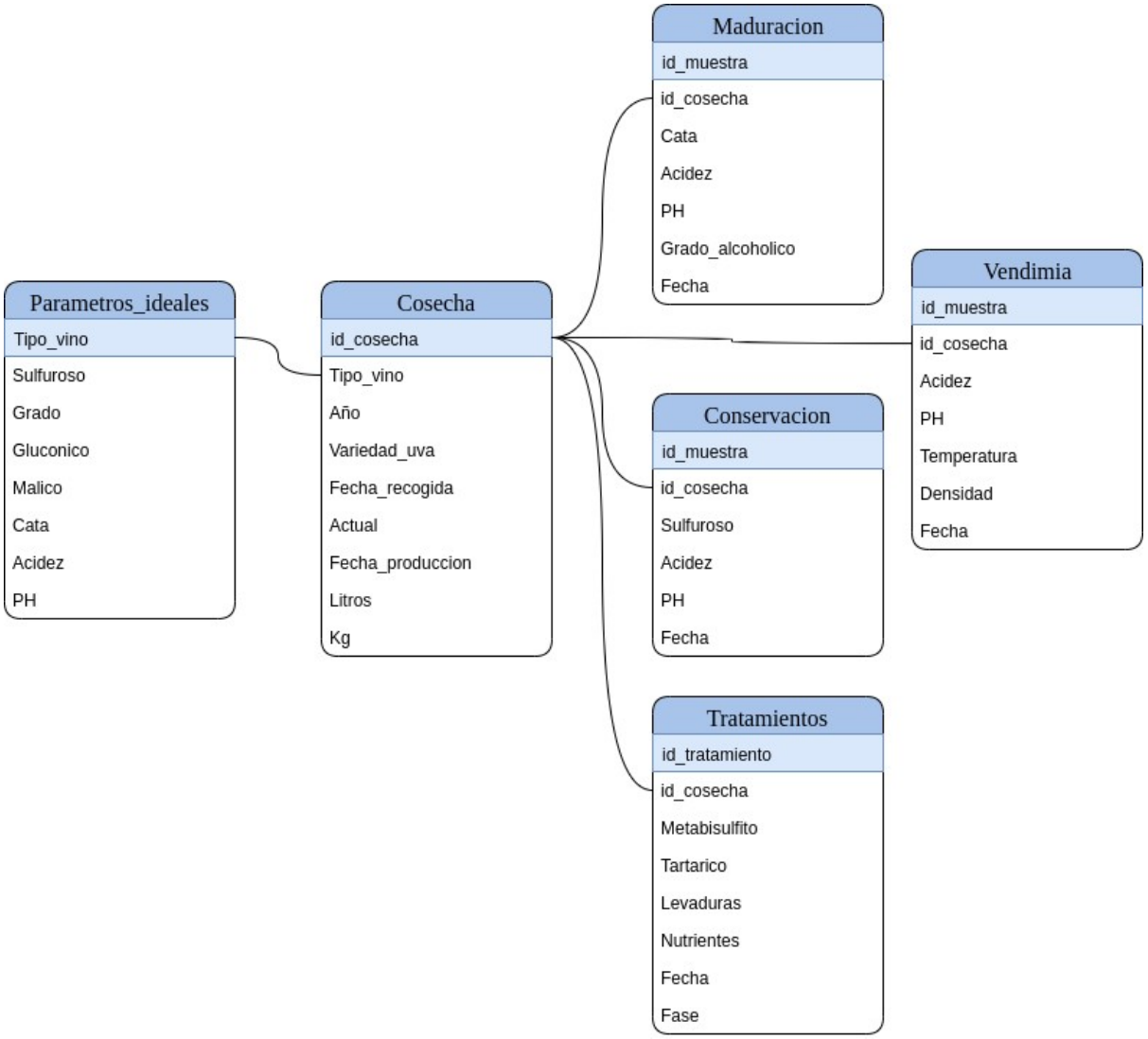


Figura 12: Tablas resultantes. En azul claro la clave primaria

11.2.- Navegación

A continuación se muestra el esquema de navegación de la aplicación web (Figura 13). El usuario aterrizará en una pantalla con información general que describirá la aplicación y desde la cual se podrá iniciar sesión o registrarse.

Para poder usar la aplicación es necesario iniciar sesión o registrarse (que conlleva automáticamente un inicio de sesión). Una vez hecho, se ofrecerá una pantalla donde cambiar la cosecha sobre la que se va a trabajar y los parámetros ideales de dicha cosecha y que contiene ya todos los enlaces de los que dispone la aplicación:

- Configuración de los parámetros ideales para ese vino.
- Cosechas, donde se puede ver las cosechas sobre las que hay datos en el sistema, eliminarlas y añadir una nueva, que pasará a ser la cosecha de trabajo.
- Muestras, donde se podrán ver las muestras de cada una de las fases, eliminarlas y añadir nuevas.
- Estadísticas, donde se podrá ver cuatro gráficos interactivos:

- Valores del PH de todas las fases de la última cosecha.
- Valores de PH de todas las fases de las tres últimas cosechas.
- Valores de acidez de todas las fases de la última cosecha.
- Valores de acidez de todas las fases de las tres últimas cosechas.

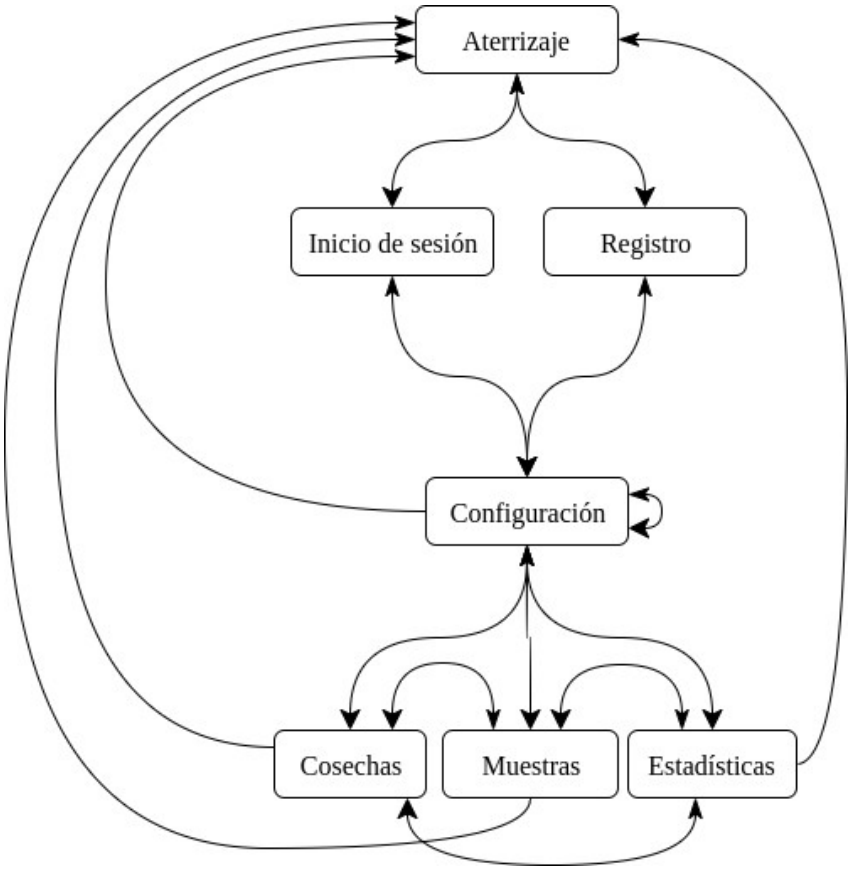


Figura 13: Mapa de navegación

12.- Prototipos

La aplicación está orientada a introducir datos del resultados de unos análisis, así que es de esperar que lo hagan justo cuando los obtengan desde un dispositivo móvil, por lo que se hace necesario usar un diseño adaptable a varias resoluciones (responsive). Sin embargo, tampoco se puede descartar que se use desde un ordenador de sobremesa, por lo que será necesario hacer otro diseño para este último caso, ya que son dos dispositivos con un tamaño de pantalla muy distinto.

Para ahorrar tiempo en la fase de diseño y codificación, se han desarrollado prototipos de todas las pantallas de las que consta la aplicación tanto en su versión de escritorio (Figura 14 a Figura 20) como en su versión móvil.

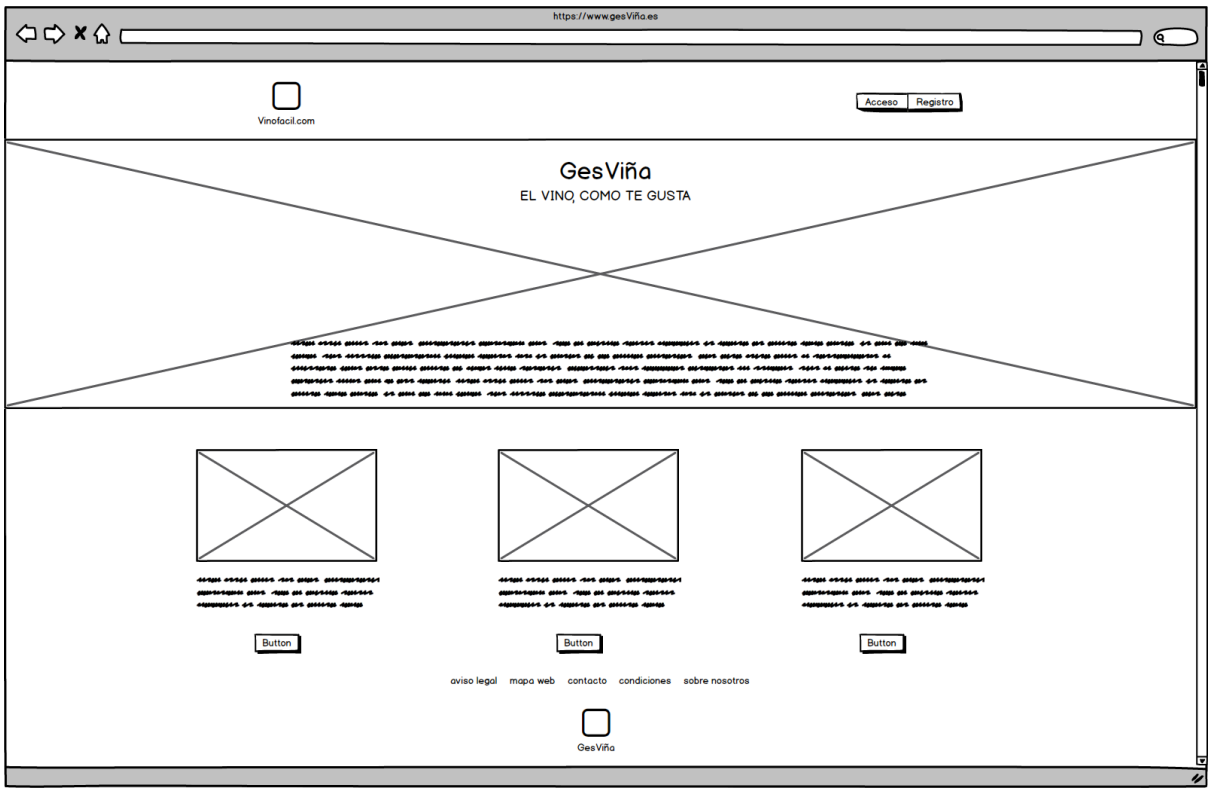


Figura 14: Sesión no iniciada (versión escritorio)

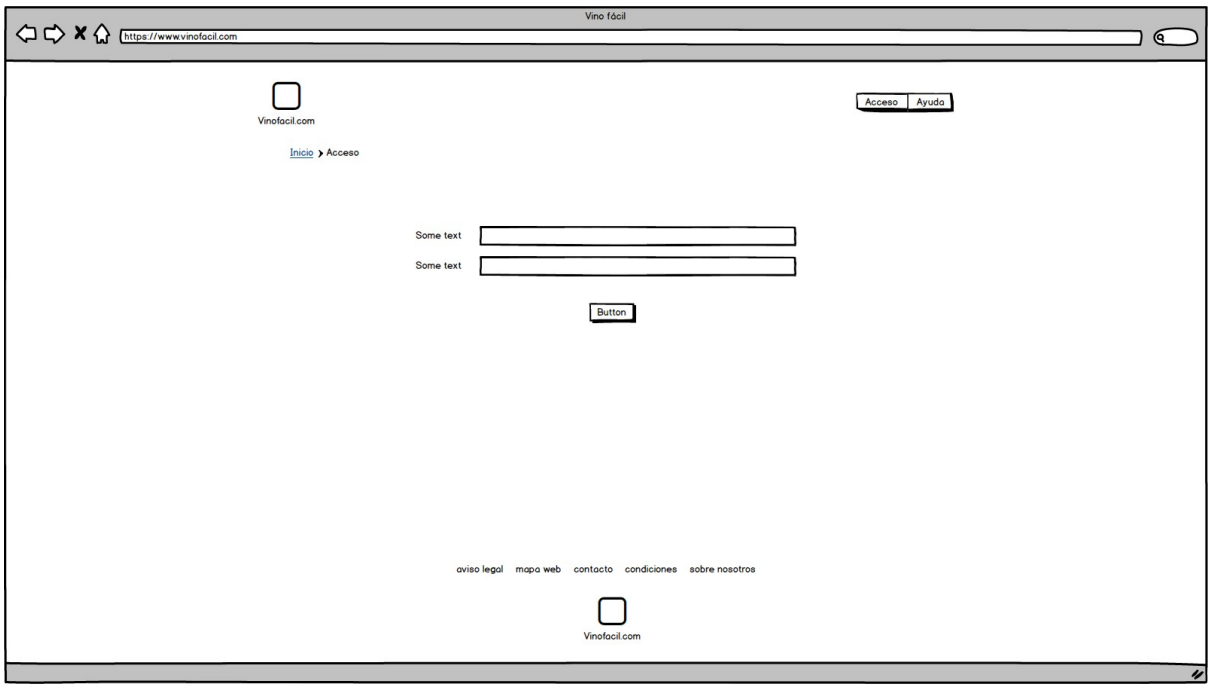


Figura 15: Iniciar sesión (versión escritorio)

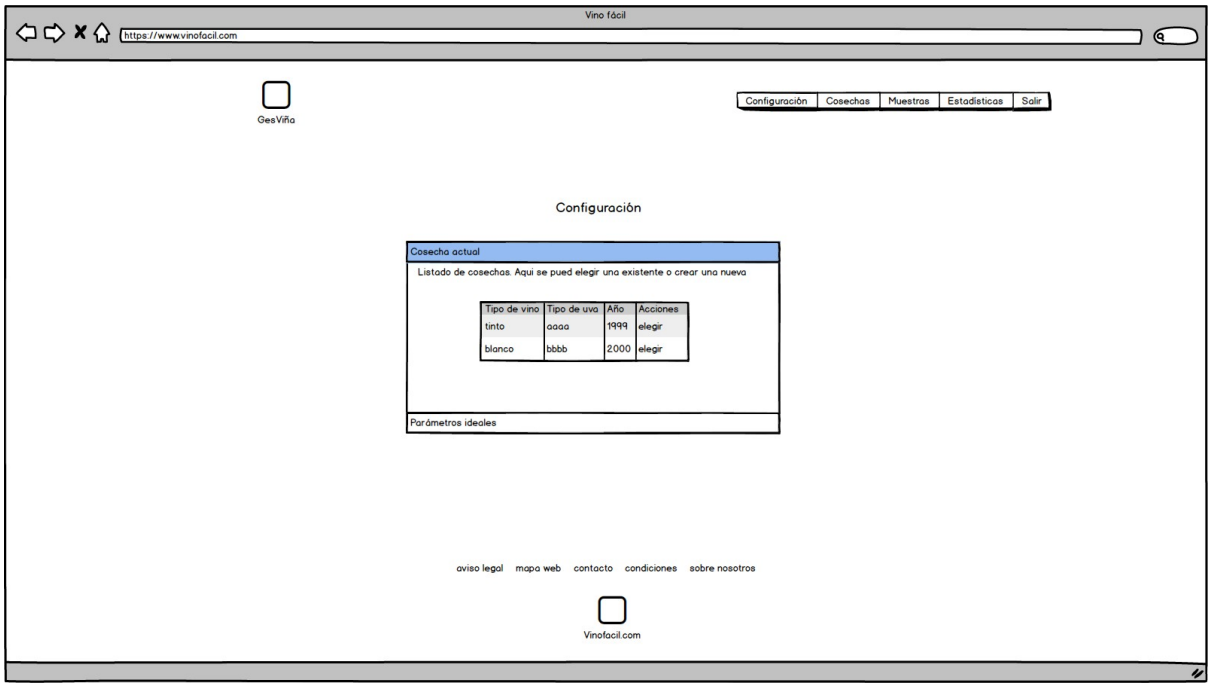


Figura 16: Configuración de la aplicación (versión escritorio)

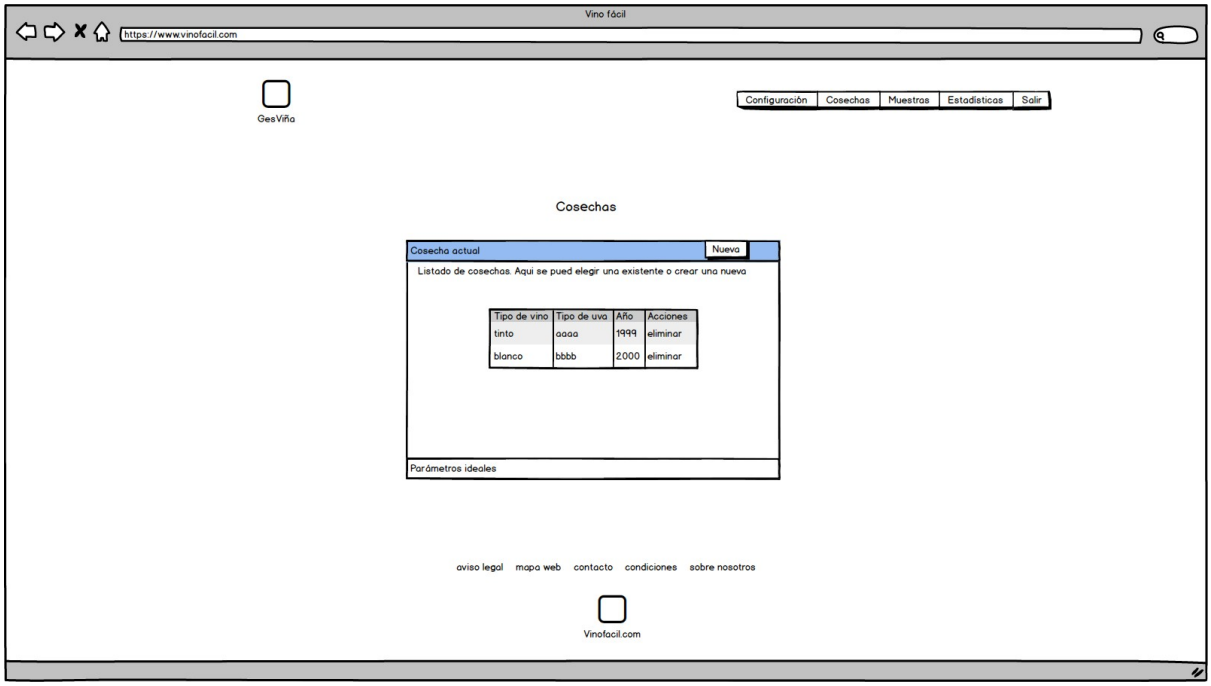


Figura 17: Gestión de cosechas (versión escritorio)

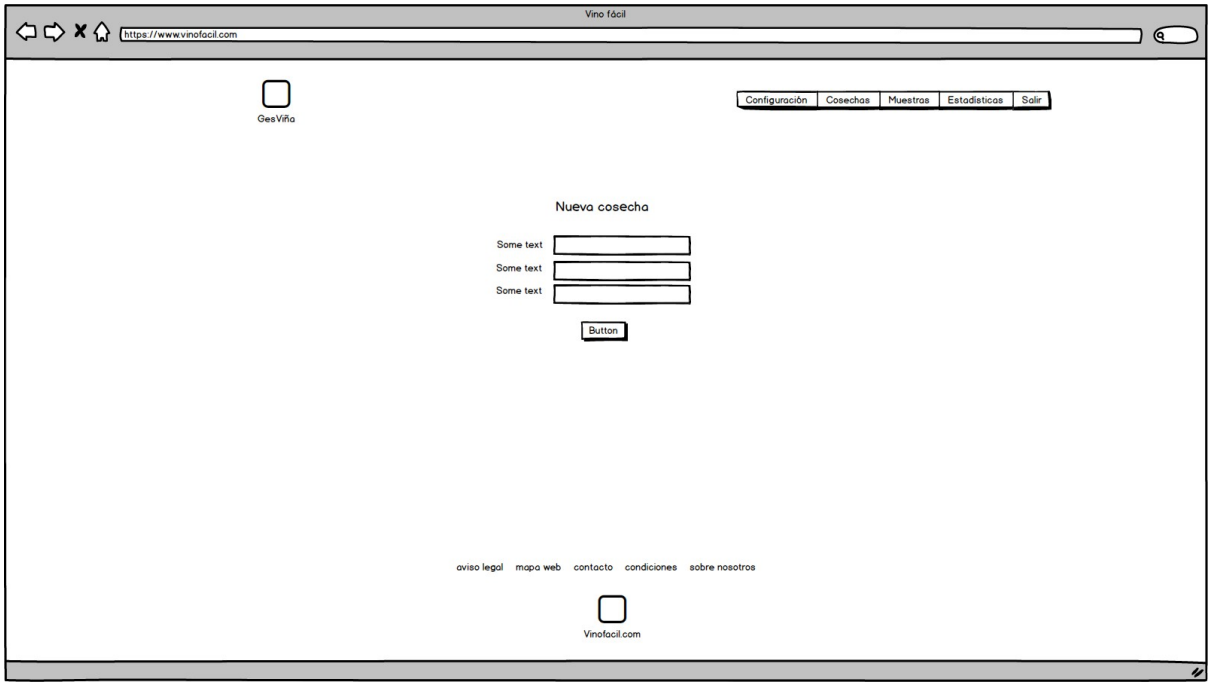


Figura 18: Adición de parámetros (cosecha, muestra, etc.) (versión escritorio)

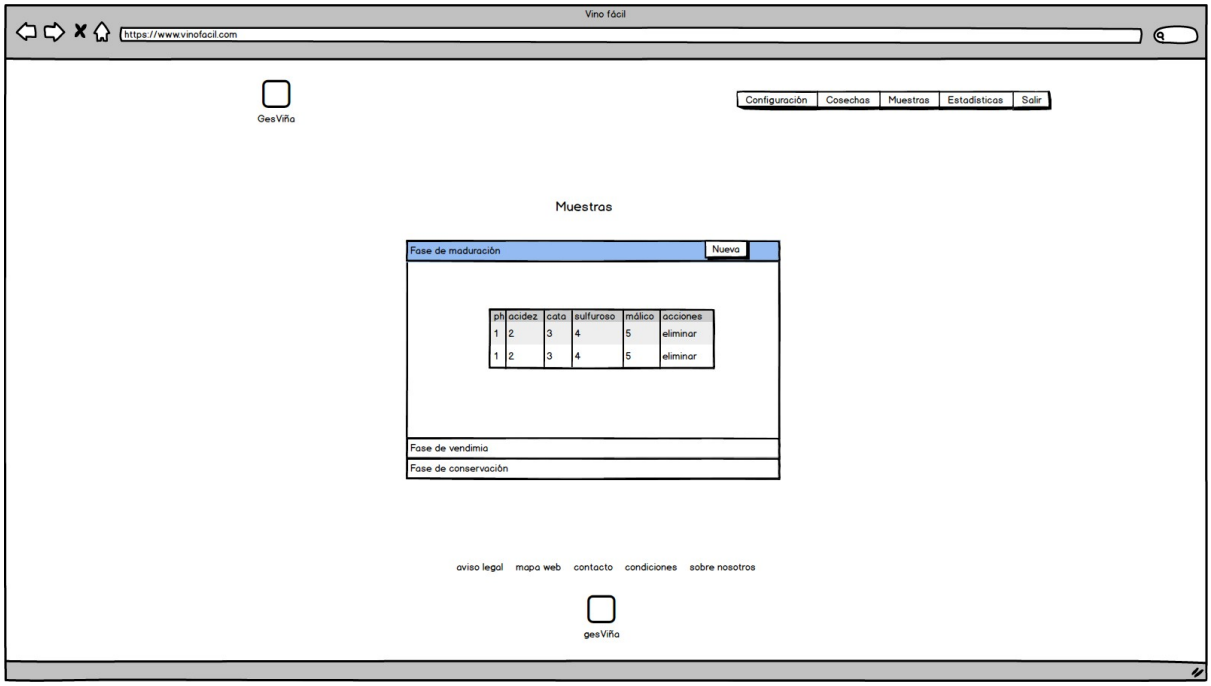


Figura 19: Gestión de muestras (versión escritorio)

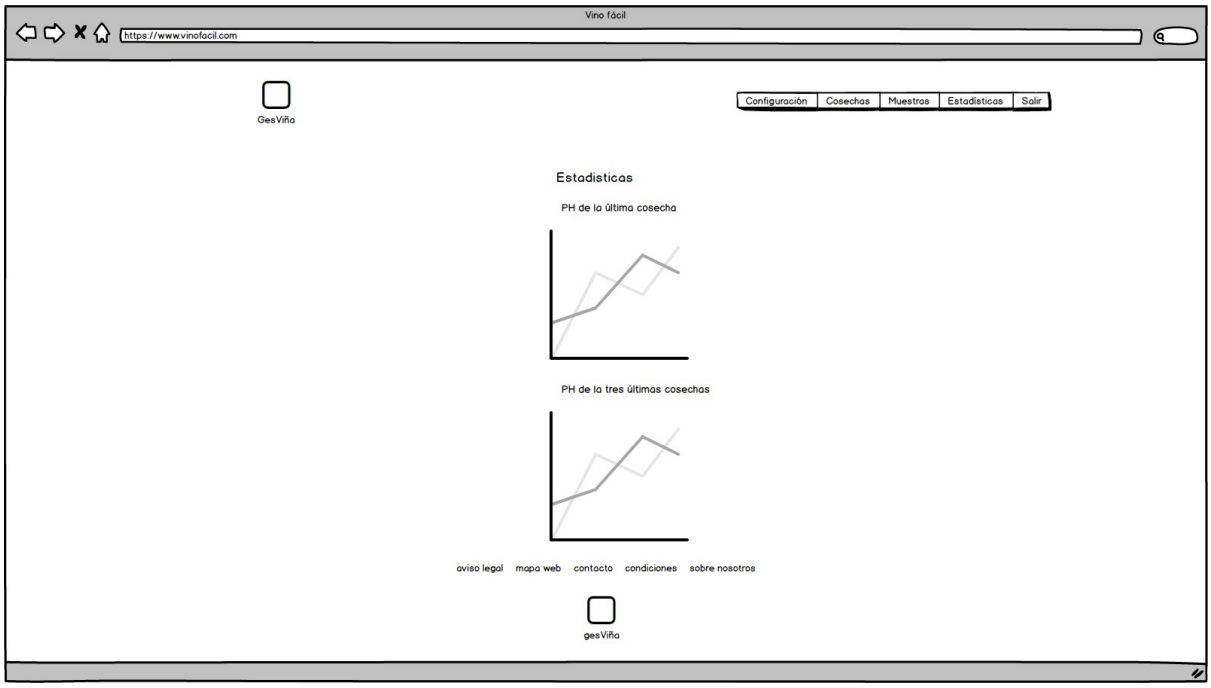


Figura 20: Presentación de resultados (versión escritorio)

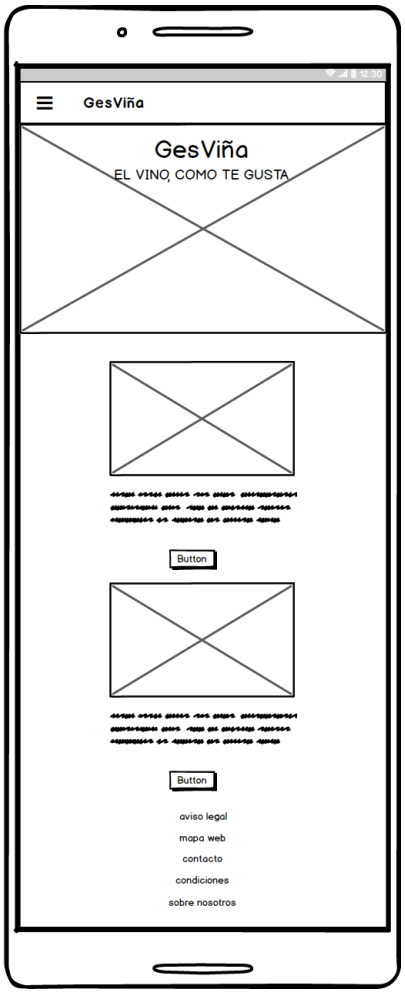


Figura 21: Sesión no iniciada (versión móvil)

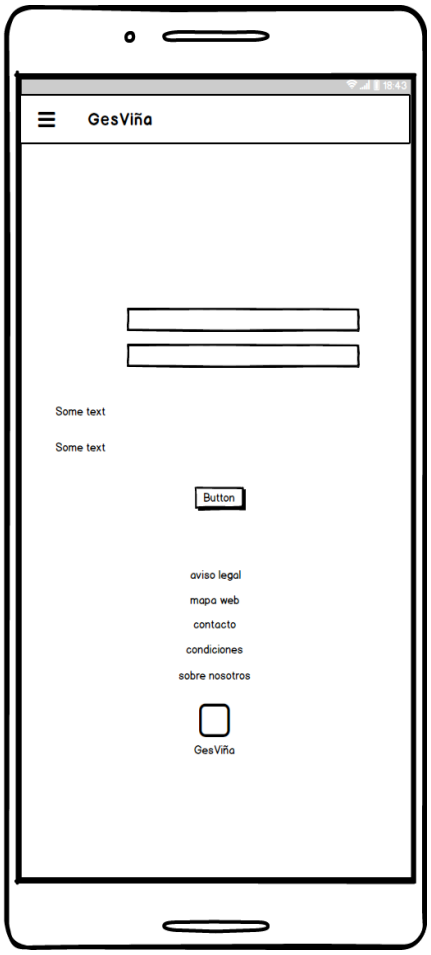


Figura 22: Iniciar sesión (versión móvil)

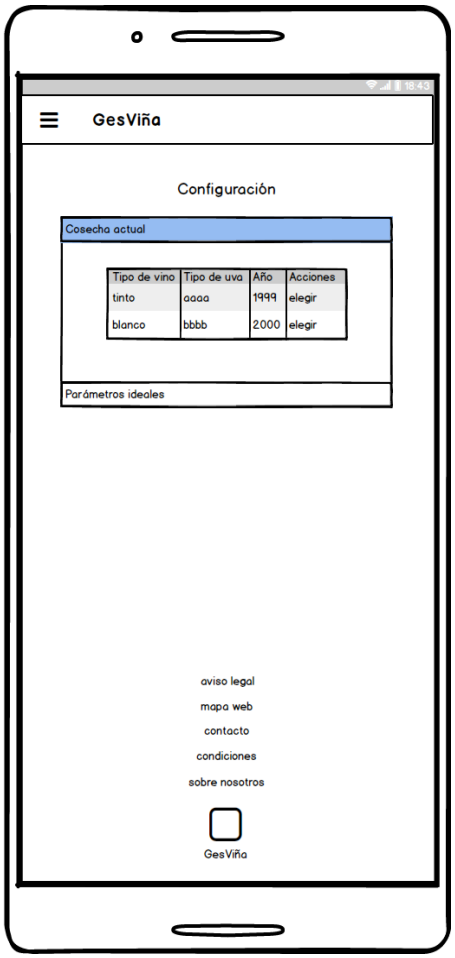


Figura 23: configuración de la aplicación (versión móvil)

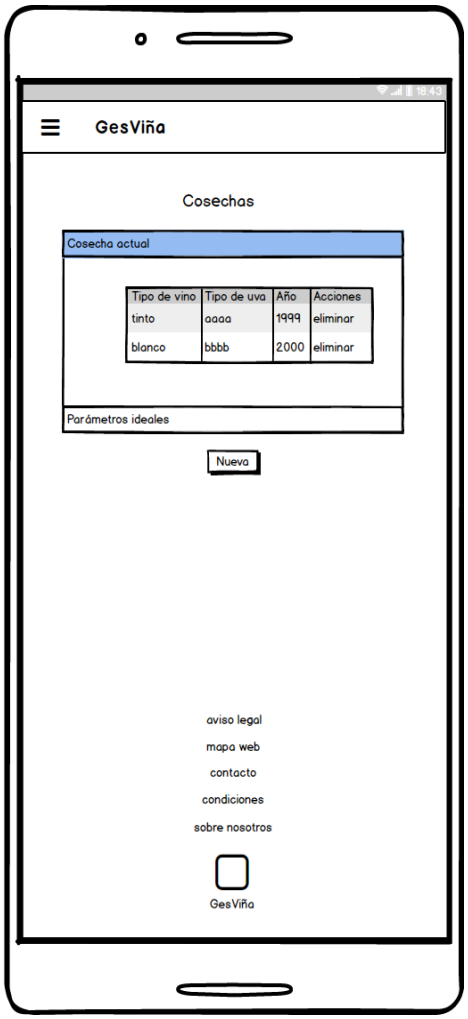


Figura 25: Gestión de cosechas (versión móvil)

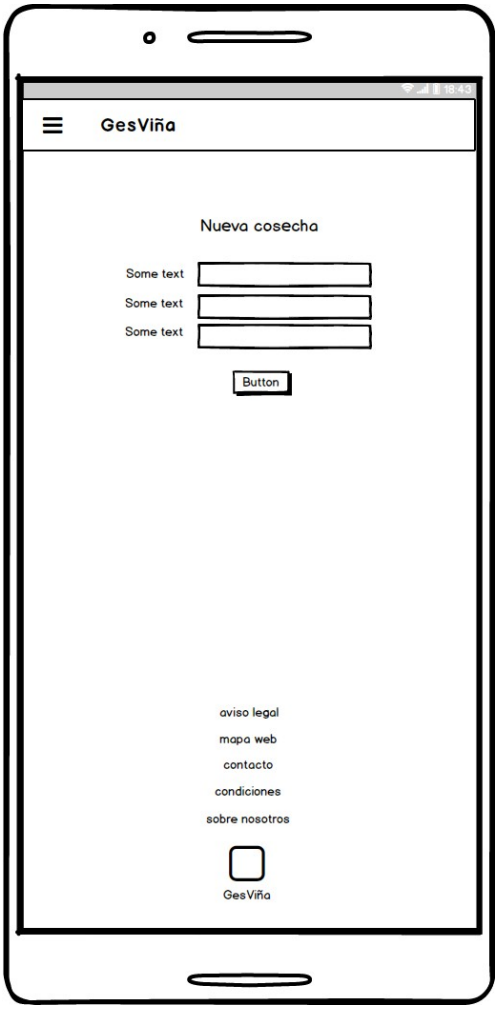


Figura 24: Adición de nuevos parámetros (cosecha, muestra, etc).(versión móvil)

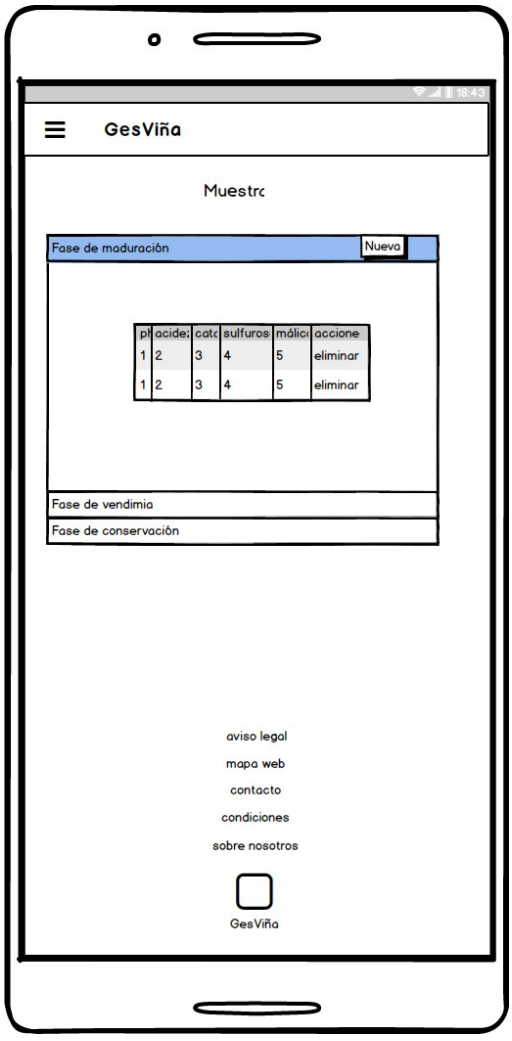


Figura 26: Gestión de muestras. (versión móvil)

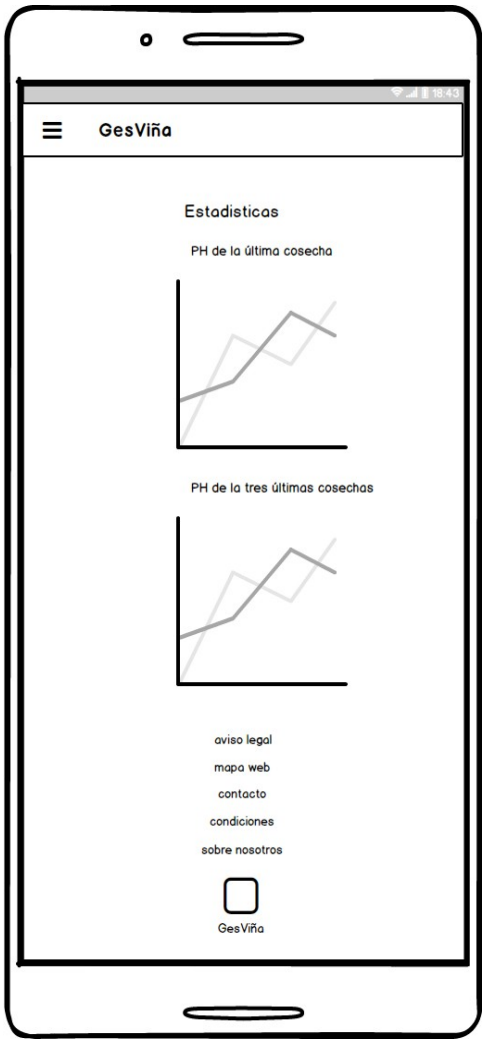


Figura 27: Estadísticas. (versión móvil)

13.- Perfiles de usuario

El primer paso para desarrollar una aplicación es responder a la pregunta, ¿Qué problema resuelve?¿Quién tiene ese problema?. Y es que adaptar la aplicación a quien la va a usar, teniendo en cuenta sus aptitudes, percepciones y forma de interaccionar con un sistema es un paso imprescindible para que triunfe y no caiga en el olvido enseguida.

Esta aplicación va dirigida a propietarios de viñedos familiares y con poca extensión orientados a autoconsumo. No he encontrado datos del INE sobre propietarios de explotaciones agrarias para autoconsumo, sobre su nivel TIC, sexo o edad, pero sí sobre los ocupados por edad, sexo y rama de actividad (Figura 28).

	De 16 a 19 años	De 20 a 24 años	De 25 a 29 años	De 30 a 39 años	De 40 a 49 años	De 50 a 59 años	De 60 a 64 años	De 65 a 69 años
	2019T3	2019T3	2019T3	2019T3	2019T3	2019T3	2019T3	2019T3
Hombres								
A Agricultura, ganadería, silvicultura y pesca	9,0	23,4	45,2	130,6	154,4	155,6	47,0	9,1
Mujeres								
A Agricultura, ganadería, silvicultura y pesca	1,8	3,5	10,7	34,0	45,0	50,5	19,0	3,6

Figura 28: Estadística del INE sobre asalariados según su rama, edad, sexo

Lo único que se puede inferir es que la mayoría son hombres en la franja de edad de 30 a 60 años. Algunos de entre 45 y 60 es posible que no estén muy familiarizados con las tecnologías y con los nuevos diseños de interfaces de usuario. Es posible, asimismo, que deseen introducir las muestras en el programa conforme se realicen, sin esperar a sentarse delante de un ordenador y a plena luz del día.

Estos aspectos tienen condicionantes en cuanto al diseño de la aplicación:

- Dificultad de comprender nuevas formas de interacción.
- Necesidad de una aplicación diseñada para dispositivos móviles.
- Dado que es posible que sea difícil leer la pantalla a plena luz, es necesario que haya un buen contraste entre tipografías y fondo.
- Dado que es posible que trabaje con la aplicación conforme se realicen las muestras, es posible que lo haga con las manos sucias, y por tanto tenga menos precisión, es necesario que se presenten pocas opciones en pantalla para que sea más fácil acertar.

14.- Usabilidad

Los principios y técnicas de usabilidad y experiencia de usuario aplicados han sido los siguientes:

- Diseño adaptable: es fundamental que la aplicación se adapte al tamaño de la pantalla que el usuario esté utilizando para que su uso sea cómodo y se pueda visualizar y manejar correctamente.
- Estructura común en toda la aplicación. Para que el usuario sienta que está en todo momento en la misma aplicación es necesario mostrar un aspecto uniforme en todas las páginas de las que conste.
- Diseño de los menús en un nivel de anidamiento. Para ayudar a que el usuario sepa donde está en cada momento, se ha diseñado un menú plano, que no tiene opciones anidadas. Esto hace innecesario la técnica de migas de pan, puesto que el usuario siempre está a un nivel de profundidad de la página principal.
- Enlace al inicio en un logotipo en la esquina superior izquierda para que el usuario pueda volver al inicio de forma rápida.
- Diapositivas: se usará para mostrar un carrusel con los gráficos que comparan el valor del mismo parámetro durante varias campañas.
- Mapa del sitio: al final de cada página, en su versión para ordenador, estará disponible un mapa del sitio, con toda la estructura. Esto permite al usuario saber en todo momento cómo está estructurado el sitio.
- Navegación principal: En todas las páginas de la versión para escritorio se encontrará en la esquina superior derecha el menú de navegación lo que permitirá al usuario cambiar rápidamente de web y, a la vez, mantener un estilo coherente. En la versión móvil habrá una imagen que desplegará el menú.
- Mantener al usuario informado del estado del sistema mediante mensajes sobre qué cosecha actual tiene seleccionada o cuantas muestras hay para cada fase sin que sea necesario que acceda a ellas para verlas.

15.- Pruebas

No hay artilugio o proceso humano que esté libre de errores y la programación de aplicaciones es uno de ellos. Para que el producto final se comporte lo más parecido a la especificación inicial de requisitos posible es necesario hacer una batería de pruebas. En este desarrollo se han hecho las siguientes:

- Pruebas unitarias. Comprueban que una determinada unidad funcional (método, módulo, componente, etc.) funciona correctamente. Las ha realizado el desarrollador en una fase temprana del desarrollo, con el interfaz aún sin definir con el objetivo de detectar los fallos lo antes posible y solucionarlos con el menor coste en tiempo. En esta fase aún no se le puede presentar al cliente nada. Las pruebas realizadas han sido las siguientes:
  - Pruebas de la API REST
    - Las llamadas HTTP son redirigidas al método adecuado del controlador correcto.
    - El método del controlador hace las operaciones adecuadas en la base de datos.
    - El método del controlador devuelve el resultado correcto.
  - Pruebas de componentes:
    - El componente hace la llamada correcta al servicio que conecta con la API.
    - El componente presentaba los datos de forma adecuada al usuario.
    - El componente accede a los servicios necesarios.
- Pruebas de integración. Comprueban que la aplicación funciona como un todo, que hay una correcta interacción entre las distintas partes que la componen. Además de hacerlas yo mismo, se ha pedido al familiar que la pruebe para ello le he puesto delante del ordenador y, sin explicarle como funciona, le he pedido que interaccione con la aplicación, anotando los resultados.

16.- Requisitos de implantación y uso

Los requisitos recomendados para implantar la aplicación en un servidor externo y utilizarla en un cliente son los mismos que se han usado al desarrollarla. Otras versiones del software podrían funcionar, pero no se han probado exhaustivamente dada la gran cantidad de combinaciones posible y la escasez de tiempo:

- Servidor
  - Alojamiento en servidor compartido (hosting)
  - Dominio que apunte al servidor compartido.
  - Servidor de base de datos: MySQL similar
  - Lenguaje PHP versión 7.3.2
  - Servidor web Apache 2.4.38 o similar.
- Cliente: Navegador Opera65 o Firefox 71.

17.- Instrucciones de instalación

17.1.- Backend

Para desarrollar la parte del backend es necesario usar las siguientes aplicaciones:

- Lenguaje de programación PHP.
- Base de datos.
- (Optativo) Módulo phpMyAdmin para administrar gráficamente el servidor MySQL.
- (Optativo) Servidor Apache, para conectarse mediante web al módulo phpMyAdmin.
- Entorno Laravel

Los cuatro primeros programas vienen incluidos en XAMPP para GNU/Linux (Figura 29) que es tan fácil de instalar como cambiar los permisos al ejecutable con chmod 755 xampp-linux-*.installer.run y lanzarlo con sudo ./xampp-linux-*-installer.run

Figura 29: XAMPP

Para instalar Laravel es necesario seguir los siguientes pasos:

- Instalar la última versión de PHP y las extensiones BCMath, Ctype, Json, Mbstring, OpenSSL, PDO, Tokenizer y XML, instalable desde terminal. Si no se dispone de alguna, seguramente de algún error, aunque no sea al principio.

- Instalar composer, que es un gestor de paquetes para PHP que permite instalar funcionalidad ya desarrollada. Se puede descargar desde los repositorios de Ubuntu, pero para tener última versión, hay que acudir a la web oficial y seguir los siguientes pasos:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') ===
'baf1608c33254d00611ac1705c1d9958c817a1a33bce370c0595974b342601bd80b92a3f
46067da89e3b06bff421f182') { echo 'Installer verified'; } else { echo
'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

- Cambiarle el dueño al directorio de composer con sudo chown -R $USER ~/.composer (por defecto es root, lo que hace que la instalación falle).
- Instalar laravel mediante `composer global require laravel/installer`
- Crear el armazón de la aplicación mediante laravel new backend

17.2.- Frontend

Para desarrollar la parte del frontend se ha utilizado lo siguiente:

- nodeJS + npm. Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de ECMAScript, asíncrono, con E/S de datos basado en eventos y basado en el motor V8 de Google. Npm es un gestor de paquetes desarrollado en JavaScript que permite instalar una serie de bibliotecas puestas a disposición de la comunidad. Para instalarlo es necesario

  nodeJS: curl -sL https://deb.nodesource.com/setup_13.x | sudo -E bash -sudo apt-get install -y nodejs
- El cliente de angular, que se instala como paquete de nodeJS usando su gestor de paquetes, npm:
  `npm install -g @angular/cli@latest`
- Los paquetes angular-material, angular-flex, bootstrap y ng2-chart, que se instalan con los siguientes comandos

```
npm install --save @angular/material @angular/cdk @angular/animations
npm install --save @angular/flex-layout @angular/cdk
npm install --save bootstrap jquery popper.js
npm install --save ng2-charts
```

18.- Instrucciones de uso

La aplicación tiene dos zonas diferenciadas:

- Zona para usuarios no identificados en la que solo se podrá consultar una página de información general sobre la aplicación, identificarse y registrarse. En esta versión, al registrarse, el usuario gana acceso completo a la aplicación. En una versión futura, se montará un sistema donde haya distintos tipos de usuarios

que usen la web de distinta forma. En ese escenario, será necesario que el administrador autorice el registro de los usuarios “operadores” o “consultores”.

- Zona para usuarios identificados en la que se dispone de toda la funcionalidad, que es la siguiente:
  - Consultar la cosecha sobre la que se va a trabajar y modificarla si no es la correcta. Todas las muestras que se introduzcan o borren luego irán asociadas a esta cosecha. La aplicación la elige automáticamente, aunque también permite elegirla.
  - Consultar los parámetros asociados al tipo de vino y tipo de uva de la cosecha y sus valores recomendados o añadirlos si aún no se ha hecho. En una futura revisión, se añadirán consejos sobre qué tratamientos aplicar en función de los valores medidos y los ideales.
  - Consultar un histórico de cosechas, introducir una nueva o eliminar una existente. En el momento en que se introduce una ésta pasa a ser la cosecha de trabajo y si se elimina la cosecha actual, se elegirá como actual la que se haya introducido justo antes de la última.
  - Consultar las muestras de cada fase que se han introducido, eliminarlas o añadir nuevas.
  - Consultar estadísticas de la cosecha actual y de las tres últimas sobre los dos valores más importantes: PH y acidez.
  - Cerrar la sesión.

Para arrancar el programa es necesario arrancar los siguientes servidores:

- base de datos. Aunque se puede hacer mediante terminal, lo más cómodo es hacerlo mediante el panel de control de XAMPP (Figura 30)
- Servidor web incorporado en laravel mediante el comando php artisan serve
- frontend mediante el comando ng serve

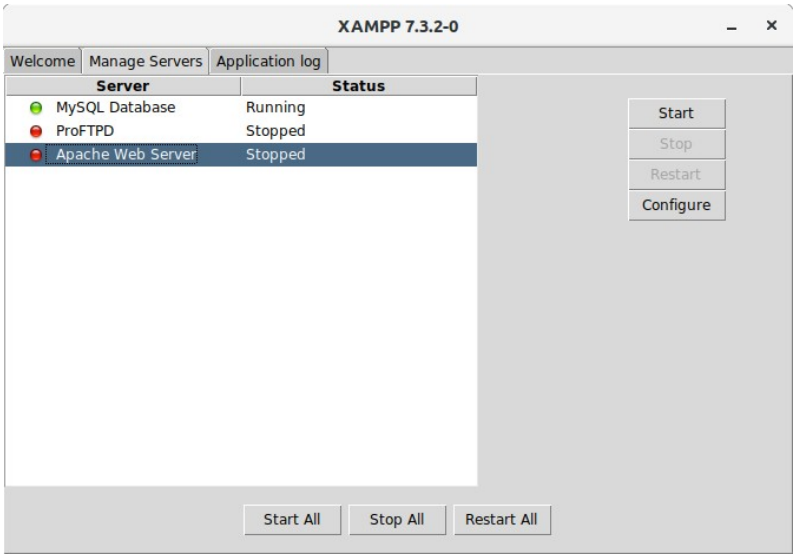


Figura 30: panel de control de XAMPP

Tras arrancar los tres, el usuario debe abrir un navegador en el puerto 4200. Será entonces redirigido a la página de aterrizaje donde se mostrará un carrusel de imágenes y un menú para identificarse o registrarse (Figura 31, Figura 32 y Figura 33). Si se identifica o registra, el usuario tendrá acceso a toda la funcionalidad de la aplicación. Si no, aunque escriba alguna URL de la zona restringida, no podrá acceder, ya que la aplicación comprueba, usando el servicio “sesion.service.ts”, si el usuario ha iniciado sesión.

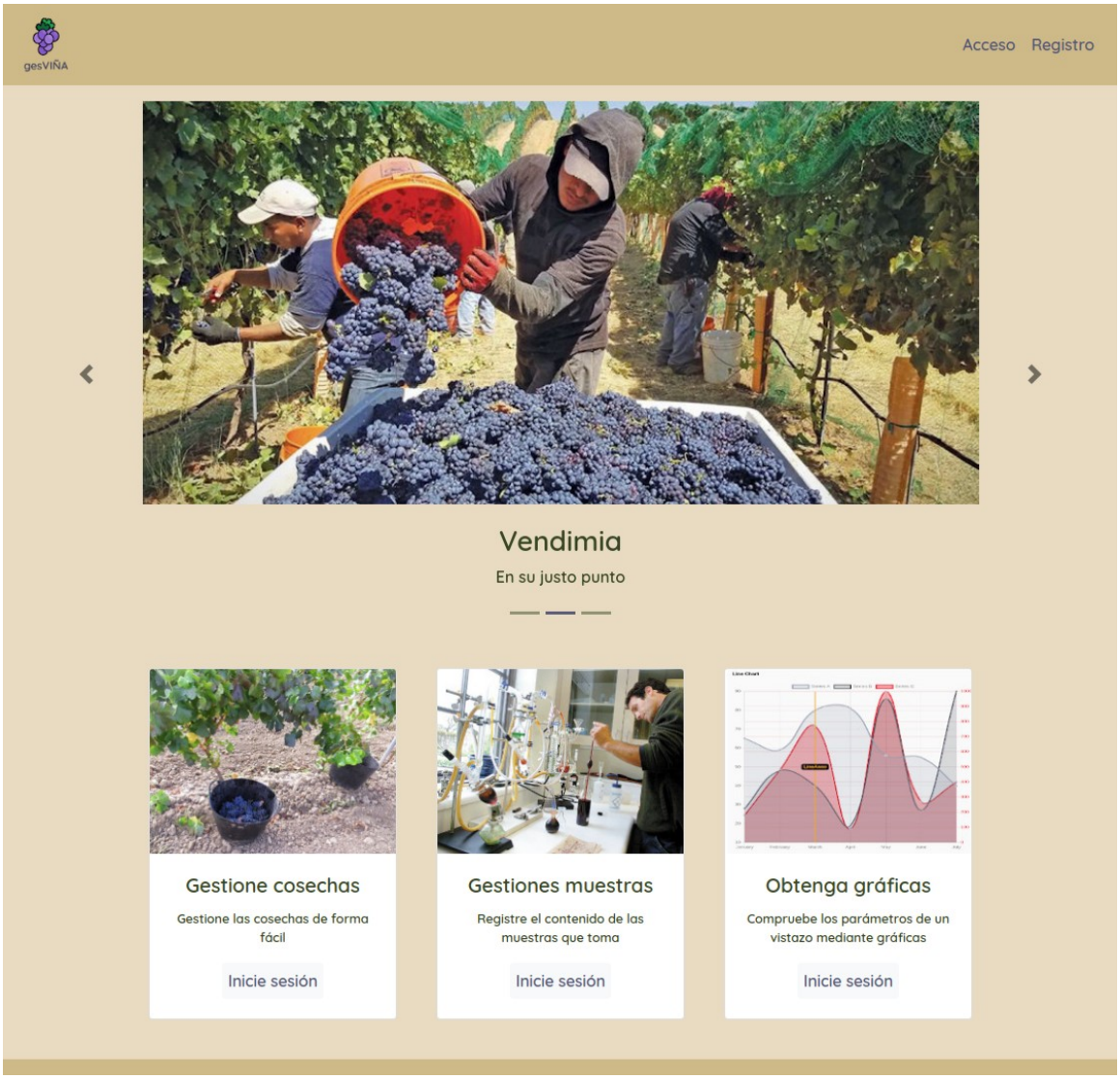


Figura 31: Página de aterrizaje

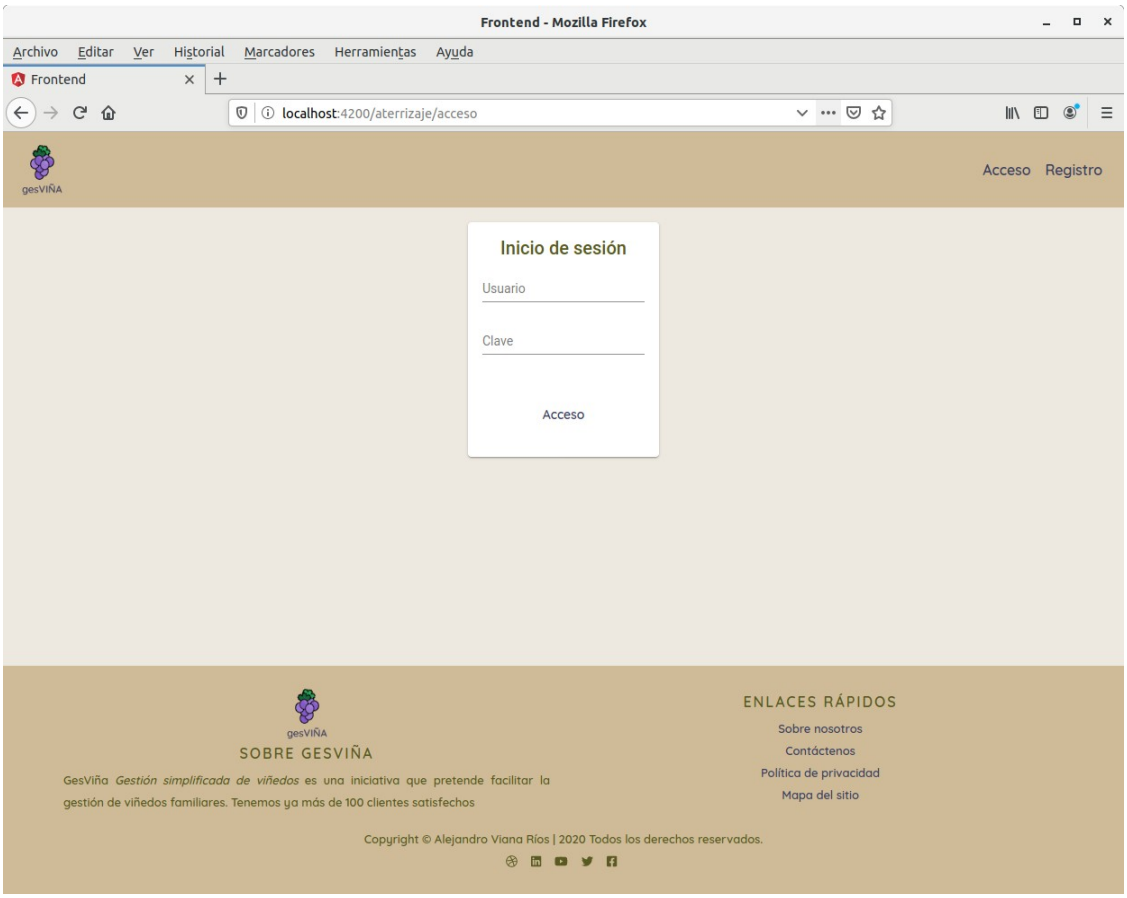


Figura 32: Página de identificación

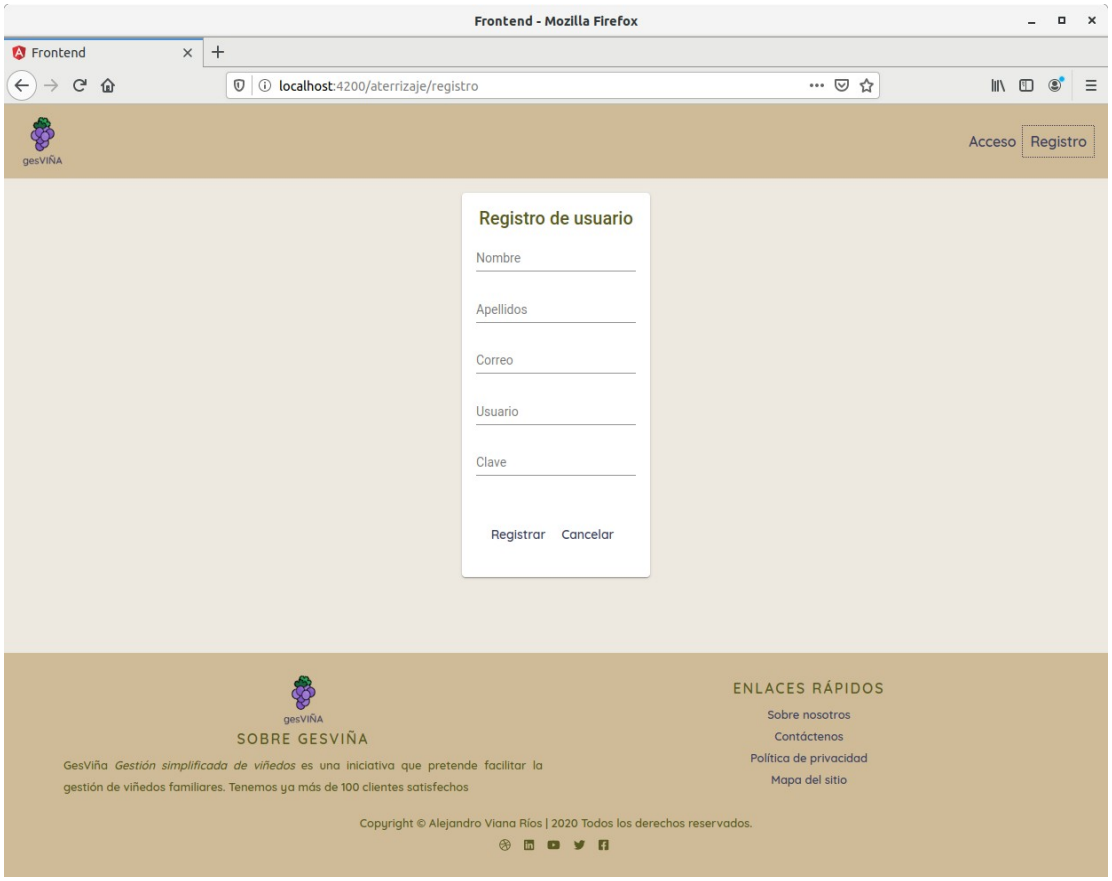


Figura 33: Página de registro

Dado que la aplicación necesita tener definida una cosecha actual, que es aquella sobre la que se introducen las muestras, el sistema escoge actual la última introducida en la base de datos por el usuario. Para que éste la pueda cambiar la aplicación lo redirige tras iniciar sesión a la página de configuración. (Figura 34 y Figura 1).

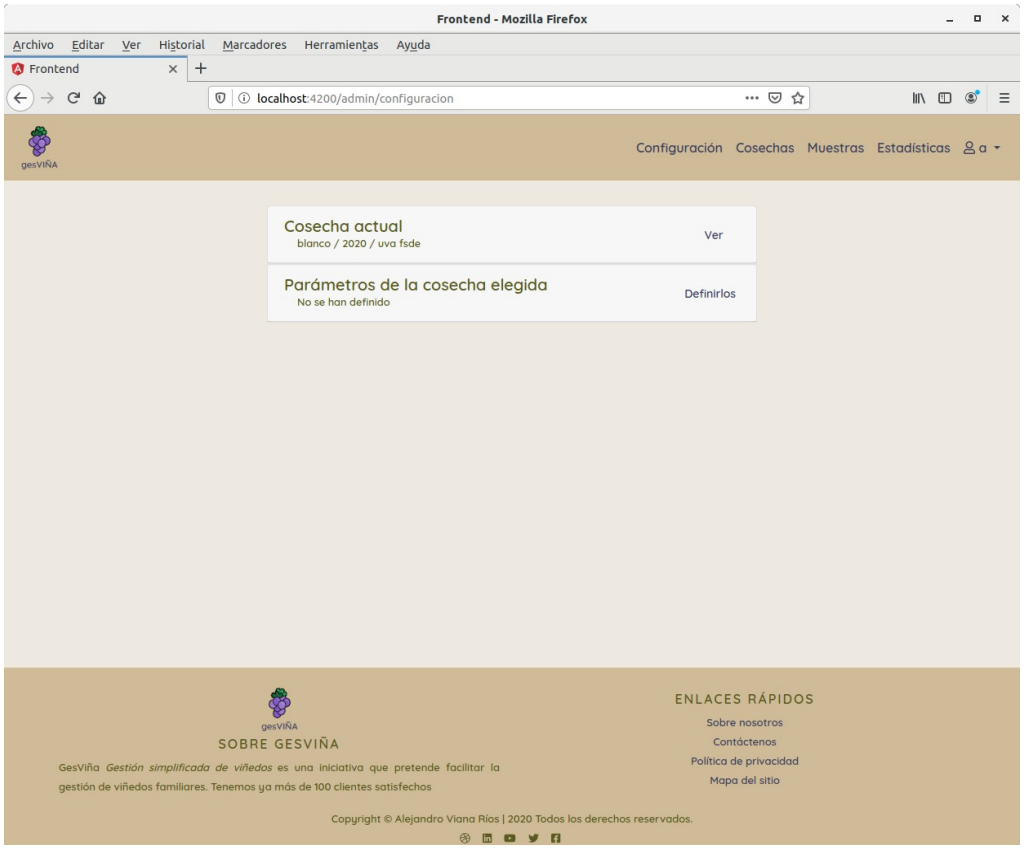


Figura 34: Panel de configuración

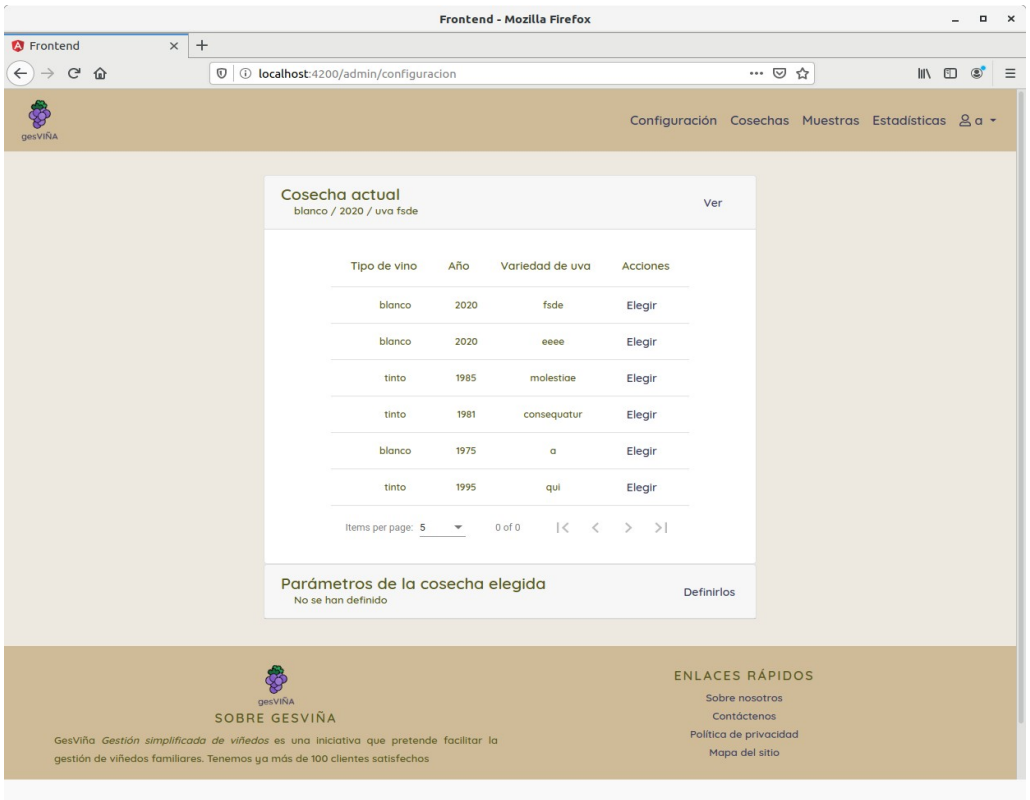


Figura 35: Panel de configuración con la cosecha desplegada para elegir

Esta página también permite editar o añadir, si no los tenía, los parámetros ideales para el tipo de vino de la cosecha actual (Figura 36). Esta característica aún no se utiliza, pero en una futura versión, se podría comparar los parámetros ideales con los valores tomados en las muestras y, en función de la desviación, recomendar determinados tratamientos.

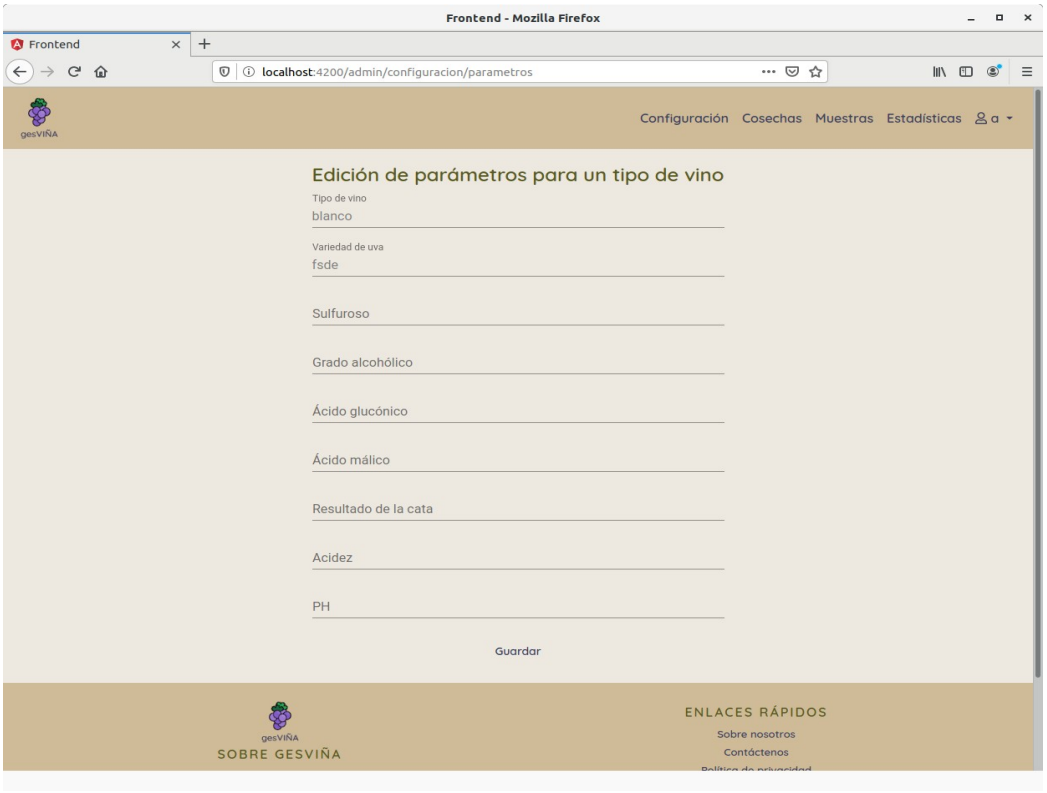


Figura 36: Panel para definir los parámetros ideales para un tipo de vino

Además de la configuración, el usuario puede navegar por los siguientes enlaces:

- cosechas, donde se pueden eliminar y añadir cosechas. Si se elimina una cosecha, también se eliminan sus muestras en la base de datos y si la eliminada es la actual, el sistema designa como actual la anterior. Al añadir una cosecha el sistema la elige como actual. (Figura 37 y Figura 38).
- muestras, donde se ven las muestras que hay en cada fase de la cosecha actual y se pueden añadir nuevas muestras o eliminar las que hay hay (Figura 39 y Figura 40).
- estadísticas, donde se ven, de la cosecha actual, los siguientes gráficos (Figura 41):
  - PH de todas las muestras de todas las fases de la cosecha actual.
  - PH de todas las muestras de todas las fases de las tres últimas cosechas.
  - acidez de todas las muestras de todas las fases de la cosecha actual.
  - acidez de todas las muestras de todas las fases de las tres últimas cosechas.
- Salir, que cierra la sesión y redirige a la página de aterrizaje.

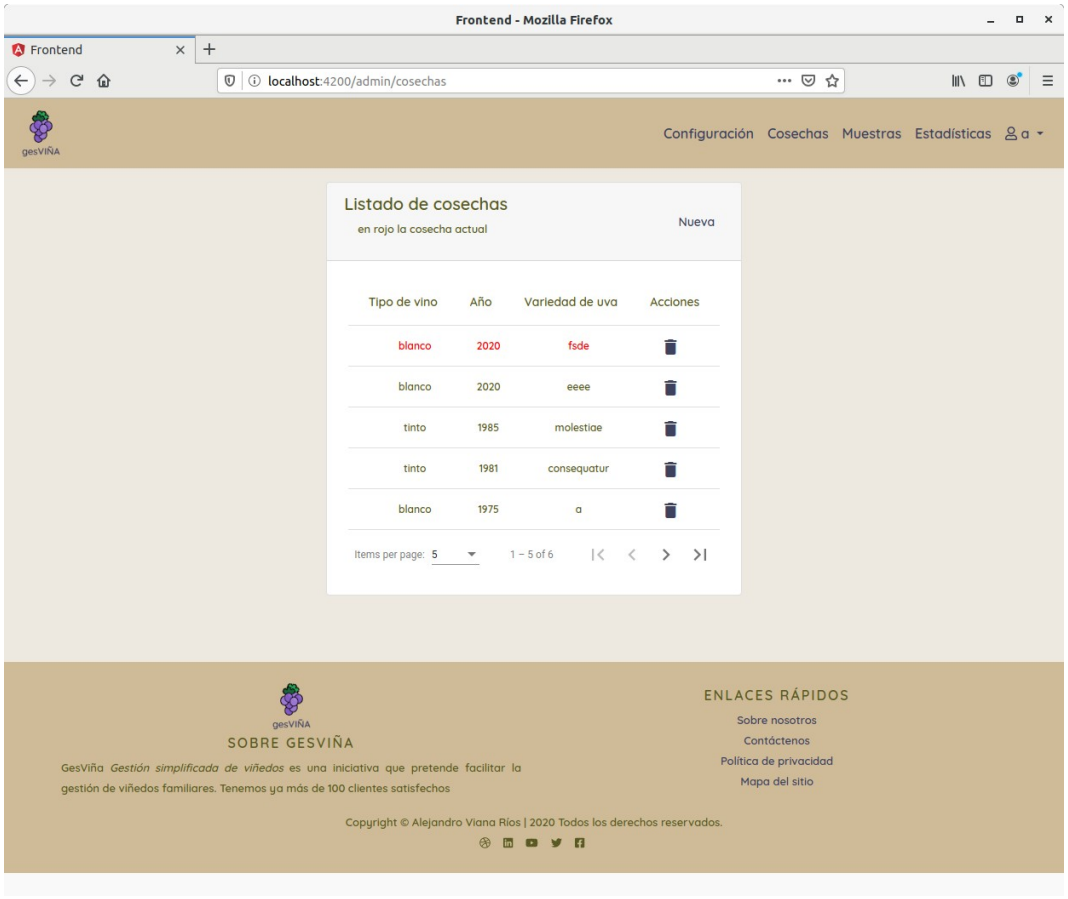


Figura 37: Panel de gestión de cosechas

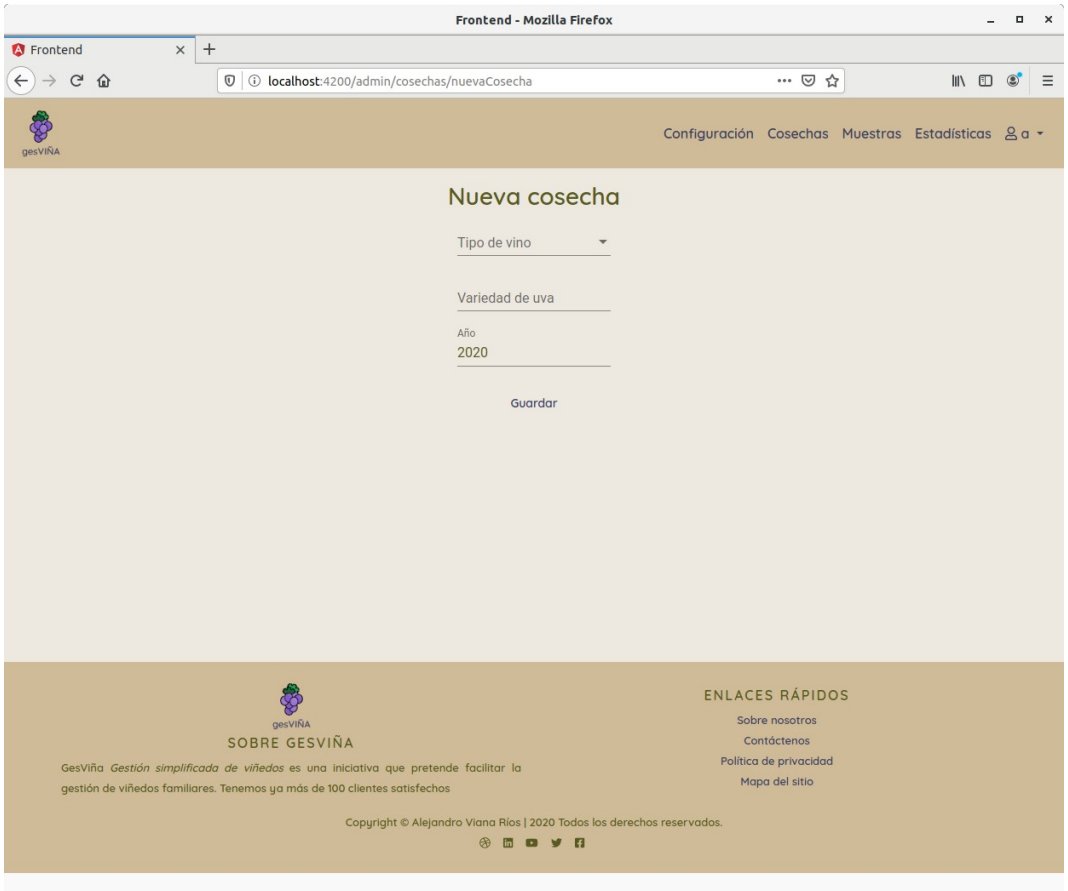


Figura 38: Panel de creación de nueva cosecha

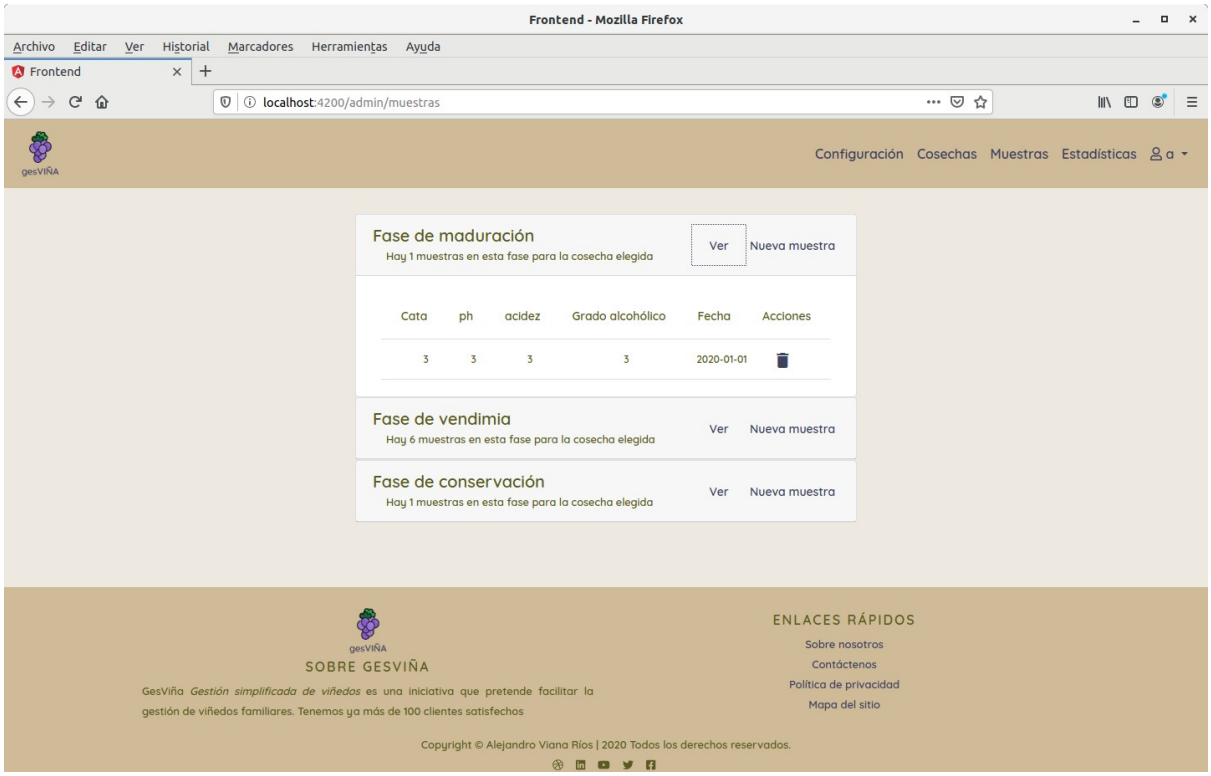


Figura 39: Página de muestras

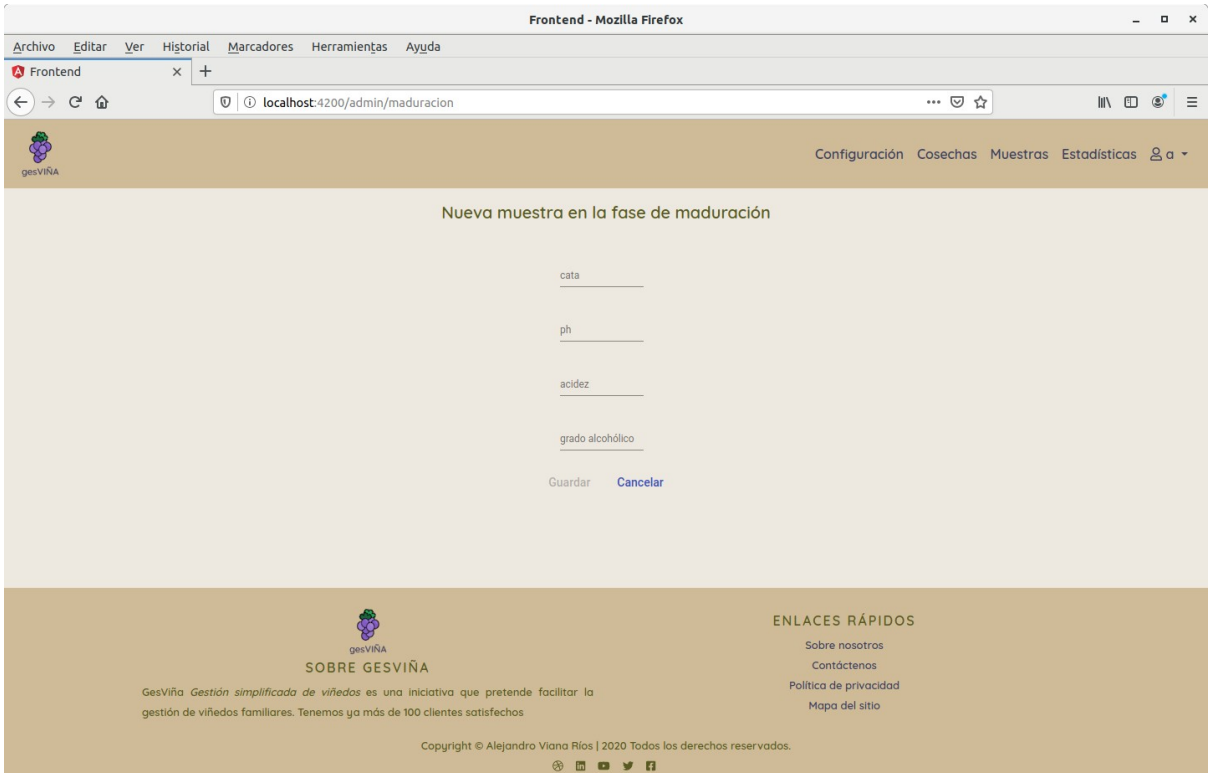


Figura 40: Página para introducir los valores de una muestra

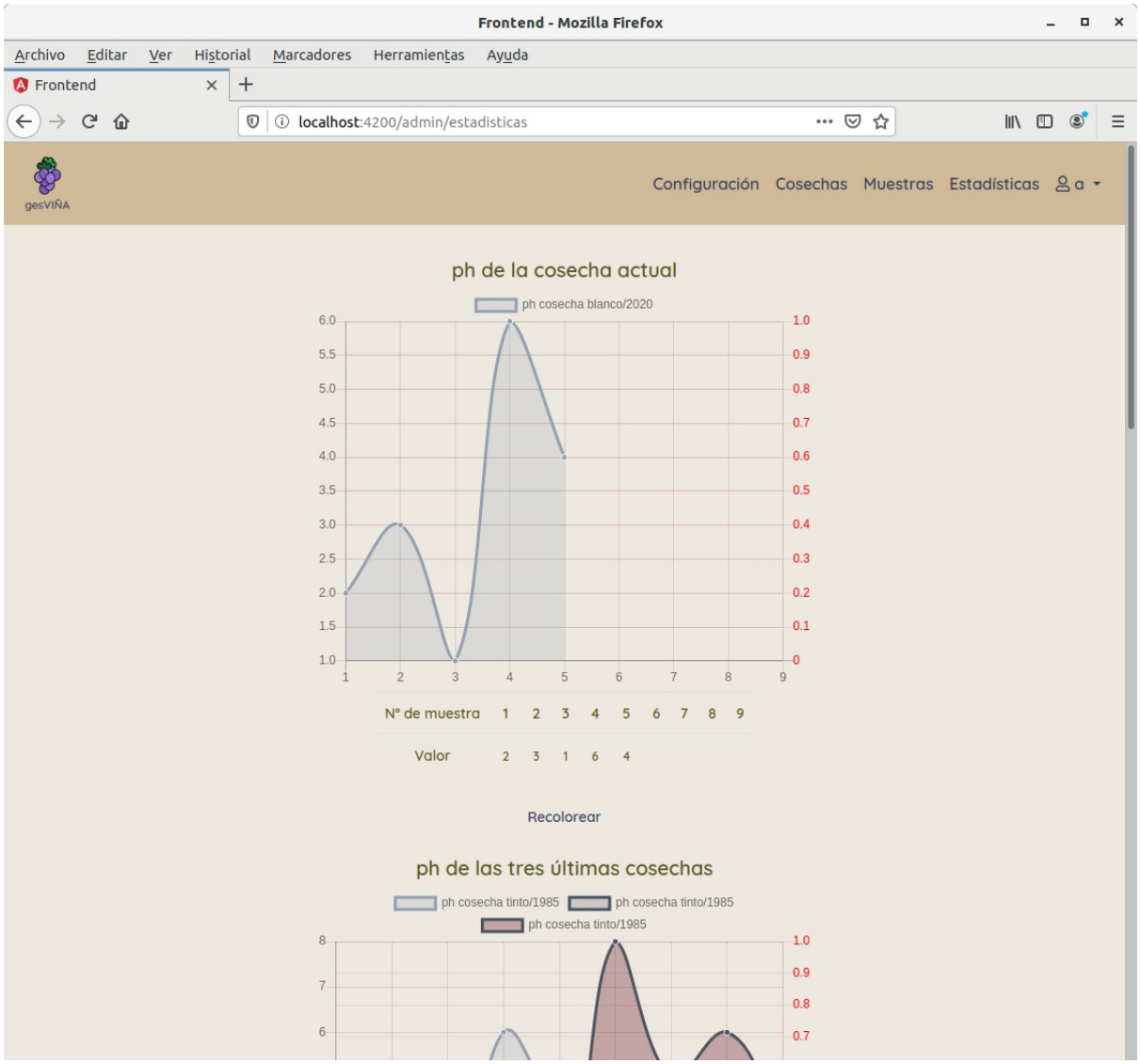


Figura 41: Pantalla con los gráficos

19.- Bugs

Hasta los mayores estudios de software, aquellos con cientos de programadores, cometen errores al programar. Malas prácticas, fallos de comunicación entre equipos, o situaciones no previstas. Así, ninguna aplicación, no importa lo compleja o sencilla que sea, nace ni 100% libre de errores y esta no podía ser una excepción.

Se ha detectado un fallo que, por motivos de escasez de tiempo, no va a poder ser resuelto. Se ha usado el componente mat-table de angular y, desgraciadamente, este no estala bien cuando se muestra en dispositivos con pantalla pequeña. La solución podría ser usar una tabla HTML no el componente mat-table de angular, pero se deja para futuras revisiones de la aplicación.

20.- Proyección a futuro

Ninguna aplicación, no importa lo compleja o sencilla que sea, nace ni 100% libre de errores ni con 100% de la funcionalidad incorporada. Y es que un desarrollo está limitado por varios factores: el presupuesto, el personal con

el que se cuenta o el tiempo. Estas limitaciones hacen que haya que elegir entre desarrollar todo lo previsto de forma rápida, con la consiguiente probabilidad de que contenga un buen número de fallos y la sensación del usuario sea que “casi nada funciona” o desarrollar solo parte de lo previsto pero dedicándole más tiempo para que esté bien depurado y sacar ampliaciones en un futuro.

Esta última es la opción que se ha elegido en este desarrollo y por ello se han dejado fuera varias funcionalidades interesantes que podrían ser incorporadas en un futuro, como podrían ser las siguientes:

- Comparar las muestras de la cosecha actual con los parámetros ideales para ese tipo de uva y sugerir tratamientos para mejorar esos parámetros y obtener un producto de más calidad.
- Establecer algún tipo de aviso para que el usuario recuerde que se requiere hacer una determinada tarea.
- Montar un sistema que permite que haya usuarios con los siguientes papeles:
  - administrador: tiene acceso completo al sistema y puede hacer cualquier operación con usuarios, cosechas, muestras y tratamientos.
  - operador: puede introducir información sobre cosechas, muestras y tratamientos, pero no eliminar ni editar.
  - Consultor: puede consultar la información.
- Permitir al usuario elegir qué parámetros mostrar en la sección de estadísticas y el número de cosechas de las que tomas los datos (actualmente solo se muestra PH y acidez de la última y de las tres últimas cosechas).

21.- Presupuesto

La siguiente tabla toma los datos de la planificación realizada en el apartado 9. A la hora de interpretar la tabla hay que tener en cuenta los siguientes puntos:

- que la duración se midió en días y, por tanto, el total de días de cada fase no es igual a la suma de las dedicaciones de cada tarea de esa fase, puesto que algunas se superponen.
- se ha usado un precio por hora de 35€.
- la dedicación diaria es de 8 horas.

Con estos datos, el presupuesto total para este desarrollo es de 21280€. Añadiéndole un 10% mas por imprevistos, saldría un coste de 23408€

Tarea	Fecha de inicio	Fecha de fin	Duración (Días)	Coste
Fase 1- Inicio del proyecto				
Lectura de documentación UOC	18/09/19	19/09/19	2	
Búsqueda de proyecto	19/09/19	24/09/19	4	
Definición de requisitos	25/09/19	25/09/19	1	

Determinación de metodología	26/09/19	26/09/19	1	
Planificación temporal	27/09/19	27/09/19	1	
Entrega de PEC1	30/09/19	30/09/19	0	
Total fase inicio	18/09/19	27/09/19	7	1960
Fase 2- Diseño				
Configuración del espacio de trabajo	30/09/19	1/10/19	2	
Diseño de la arquitectura backend y frontend	2/10/19	7/10/19	4	
Diseño de la base de datos	8/10/19	10/10/19	3	
Diseño del árbol de navegación	8/10/19	11/10/19	4	
Creación del manual de estilos	8/10/19	14/10/19	5	
Diseño de bocetos	15/10/19	24/10/19	8	
Documentación del proceso	25/10/19	29/10/19	3	
Entrega de PEC2	30/10/19	30/10/19	0	
Total fase diseño	30/09/19	29/10/19	22	6160
Fase 3- Desarrollo				
Configuración del espacio de trabajo	30/10/19	31/10/19	2	
Modificaciones menores	30/10/19	31/10/19	2	
Desarrollo del backend	1/11/19	21/11/19	15	
Desarrollo del frontend	1/11/19	21/11/19	15	
Maquetación del frontend	22/11/19	26/11/19	3	
Pruebas	27/11/19	2/12/19	4	
Documentación del proceso	3/12/19	5/12/19	3	
Entrega de PEC3	6/12/19	6/12/19	0	
Total fase Desarrollo	30/10/19	5/12/19	27	7560
Fase 4 - Implantación				
Despliegue de la aplicación	9/12/19	11/12/19	3	
Pruebas finales	12/12/19	18/12/19	5	
Correcciones y retoques finales	19/12/19	1/01/20	10	
Total fase 4	9/12/19	3/01/20	20	5600
Total				21280
Imprevistos (10%)				2128
Total proyecto				23408

Tabla 5: Cálculo del presupuesto de desarrollo

22.- Análisis de mercado

En el punto 1.1 de la presente documentación se justificaba el desarrollo de esta aplicación aduciendo que las aplicaciones disponibles en el mercado como, por ejemplo tractus, EnolData, isagri o vintiOS permiten gestionar el proceso integral de la fabricación del vino así como su comercialización, etiquetado, trazabilidad, parcelas, bodega, inventario, etiquetado, material, proveedores, etc. pero la mayoría de esas opciones no se utilizan en una explotación familiar de menos de una hectárea pero complican el uso de la aplicación y aumentan su precio.

Según las estadísticas agrarias sobre la situación del viñedo en España que publica el Ministerio de agricultura, pesca y alimentación (Figura 42), en España un 32,72% de los viñedos tienen una superficie menor de 0,1ha y un 33,27%, una superficie de entre 0,1 y 0,49ha. Con esa superficie, lo más probable es que la mayoría sean explotaciones familiares y/o artesanales para las cuales sería más interesante una herramienta que deje de lado muchas de las opciones mencionadas en el punto 1.1 de esta documentación, que son más interesantes en explotaciones comerciales. Es, por tanto, un nicho de mercado bastante interesante al que no se ha dirigido ninguna aplicación.

Tabla 5: Explotaciones vitícolas por clase de tamaño a nivel nacional. Explotaciones Nº.

	Clase de Tamaño							
Tipo de Producción	Menos de 0,10 ha	De 0,10 a 0,49 ha	De 0,50 a 0,99 ha	De 1 a 2,9 ha	De 3 a 4,9 ha	De 5 a 9,9 ha	10 ha o más	Total
Superf. Total de viñedo (en producción/no en producción)	169.272	172.218	55.072	59.524	19.722	20.427	21.380	517.615
Viñas que producen uvas para vinificación - Total	169.241	172.059	54.965	59.399	19.697	20.408	21.375	517.144
Viñas para vinificación de vinos DOP	66.955	99.358	41.881	50.365	17.953	19.322	20.731	316.565
Viñas para vinificación de vinos IGP	23.635	31.285	8.304	5.960	1.498	1.490	2.323	74.495
Viñas para vinificación de vinos que no son ni DOP ni IGP	82.063	51.664	6.687	4.784	1.177	1.039	1.334	148.748
Viñas para uvas de doble propósito	38	755	906	1.077	162	32	16	2.986
Viñas para pasas		1	6	2	1			10
Otras viñas no consideradas anteriormente	35	173	133	208	73	64	117	803

Figura 42: Número de explotaciones vitícolas por tamaño (Ministerio de Agricultura y pesca, 2015)

23.- Conclusiones

Cuando comencé este TFM me sentía un poco perdido porque, al ser un tema que aún no domino y tener un formato mucho más libre, menos guiado, no sabía muy bien cómo comenzar. Sin embargo, sacando tiempo de donde no hay y gracias a la ayuda del tutor, Carlos, he conseguido que llegue a buen término.

En su desarrollo ha ocurrido como con todo proyecto, que dadas unas especificaciones y una idea inicial, el proyecto va evolucionando y cambiando a lo largo de su desarrollo, tanto la lógica de programa, como la base de datos y la interfaz de usuario.

Este TFM ha supuesto un broche final excelente a la maestría y en él se han trabajado aspectos muy interesantes como presupuesto, planificación y metodología, integrando en un único proyecto todas las fases de un desarrollo moderno lo cual ha sido muy enriquecedor al ofrecer una visión general del proceso completo.

Poco podía imaginar cuando me matriculé en la maestría en Desarrollo de aplicaciones web de la UOC que sería capaz de llevarlo a buen término según mis planes. Y es que cuando lo hice, hacía ya varios años que no programaba nada y lo que había hecho era siempre backend en PHP. En aquella época, los servidores necesarios para cualquier desarrollo, web, php y apache, se instalaban y configuraban para integrarse entre ellos, manual e individualmente. Aún faltaba varios años para que surgieran programas todo en uno como XAMPP. Además, la

idea de un entorno de ayuda que automatizase procesos o “endulzara” la sintaxis del lenguaje de programación era, simplemente, una fantasía que nadie podía imaginar.

Además, mis nociones de optimización, diseño, interfaces de usuario o usabilidad eran, en el mejor de los casos, muy limitadas y, encima, el mercado laboral no me había llevado por la senda de JavaScript, del que conocía poco más que su nombre. Con la tremenda evolución del desarrollo de software y el limitado tiempo que me dejaban mis obligaciones laborales, sentía que ya no podría re engancharme nunca.

La maestría de la UOC me ha dado la oportunidad de meterme de lleno en el mundo del desarrollo web. Gracias a ella he aprendido, entre otras cosas, la importancia del prototipado, la necesidad de diseñar interfaces centrados en el usuario y de mimar la usabilidad, la importancia de optimizar el código y las imágenes porque no todos los dispositivos tienen una conexión de banda ancha, a formatear la página usando hojas de estilo, la importancia de los preprocesadores, la existencia de un mundo de bibliotecas con funcionalidad ya probada, el (al principio) extraño concepto de programación asíncrona, la necesidad de usar entornos (frameworks) en desarrollos mayores y un muy necesario refresco en cuanto a programación en PHP.

Desgraciadamente es imposible que ninguna maestría, y esta no podía ser una excepción, abarque todo el espectro actual de tecnologías, tendencias, buenas prácticas, frameworks, etc de un determinado campo por la falta de horas. A pesar de ello, en mi opinión, la de la UOC está suficientemente equilibrada y consigue que encuentre el camino de la programación del que salí hace tiempo.

Anexo 1.- Entregables del proyecto

- Archivo: PAC_FINAL_Viana_Ríos_Alejandro.zip. Contiene
  - Directorio “proyecto”
    - backend.zip: archivos fuente del backend.
    - frontend.zip: archivos fuente del frontend.
  - Directorio “documentación”
    - esta documentación (PAC_FINAL_mem_Viana_Ríos_Alejandro.pdf).
    - informe de autoevaluación del alumno (Informe_Autoevaluacion_TFM_Viana_Ríos_Alejandro.pdf).
    - presentación visual del proyecto (PAC_FINAL_prs_Viana_Ríos_Alejandro.zip).
- Archivo TFM final.mp4. Contiene el vídeo donde se muestra un resumen del proceso de desarrollo y el funcionamiento de la aplicación.

Anexo 2.– Código fuente (extractos)

Backend

En el siguiente código (Figura 43) se muestran algunas entradas en la tabla de rutas de la API. En las líneas 27 a 30, se controla que las URL con la forma http://localhost:8000/vendimia vayan dirigidas al controlador vendimia. La función a la que irán dirigidas dependerá del método enviado con el protocolo HTTP. Así...

- Si el método es GET y la ULR es http://localhost:8000/vendimia, se ejecutará la función mostrarTodos del controlador “controladorVendimia”.
- Si el método es GET y la URL es http://localhost:8000/vendimia/1, se ejecutará la función mostrarUno del controlador “controladorVendimia” a la que se le pasará como argumento el número 1.
- Si el método es POST y la URL es http://localhost:8000/vendimia, se ejecutará la función almacenar del controlador “controladorVendimia” que almacenará en la base de datos el valor de la muestra pasado en el “body” de la petición HTTP.
- Si el método es DELETE y la URL es http://localhost:8000/vendimia/1, se ejecutará la función eliminar del controlador “controladorVendimia” que eliminará de la base de datos la muestra cuya clave primaria sea 1.

```
27  Route::get('vendimia', 'controladorVendimia@mostrarTodos');
28  Route::get('vendimia/{id}', 'controladorVendimia@mostrarUno');
29  Route::post('vendimia', 'controladorVendimia@almacenar');
30  Route::delete('vendimia/{id}', 'controladorVendimia@eliminar');
31
32  Route::get('conservacion', 'controladorConservacion@mostrarTodos');
33  Route::get('conservacion/{id}', 'controladorConservacion@mostrarUno');
34  Route::post('conservacion', 'controladorConservacion@almacenar');
35  Route::delete('conservacion/{id}', 'controladorConservacion@eliminar');
36
37  Route::get('usuarios', 'controladorUsuarios@mostrarTodos');
38  Route::post('usuarios', 'controladorUsuarios@almacenar');
39
40  Route::get('cosecha', 'controladorCosechas@mostrarTodos');
41  Route::get('cosecha/{id}', 'controladorCosechas@mostrarUno');
42  Route::delete('cosecha/{id}', 'controladorCosechas@eliminar');
43  Route::post('cosecha', 'controladorCosechas@almacenar');
44  Route::put('cosecha/{id}', 'controladorCosechas@actualizar');
```

Figura 43: Ejemplo de entradas en la tabla de rutas en la API

En el siguiente código (Figura 44) se muestra el controlador para las llamadas a la API con el parámetro “vendimia” para hacer operaciones en la base de datos con las muestras tomadas en la fase de vendimia. Se puede observar lo siguiente:

- la función “mostrarTodos” devuelve todos los registros de la tabla “vendimia”
- la función “mostrarUno” devuelve el registro de la tabla “vendimia” cuya clave primaria coincide con la pasada como argumento (en la URL).
- La función “almacenar toma el “body” de la petición HTTP y almacena esos valores en la tabla “vendimia”

- la función “eliminar” eliminar el registro de la tabla “vendimia” cuya clave primaria coincide con la pasada como argumento (en la URL)

```php
<?php

namespace App\Http\Controllers;
use App\vendimia;
use App\cosechas;

use Illuminate\Http\Request;

class controladorVendimia extends Controller
{
  public function mostrarTodos(){
      return vendimia::all();
  }

  public function mostrarUno($id){
    return vendimia::find($id);
  }

  public function almacenar(Request $request){
    if (cosechas::find($request -> input('id_cosecha')) ) {
      $vendimia=vendimia::create($request->all());
      return response()->json(["mensaje" => "Almacenado correctamente"], 200);
    }else {
      return response()->json(['errors'=>array(['code'=>421,'message'=>'Esa cosecha no existe'])],422);
    }
   }

  public function eliminar($id_muestra){
    $vendimia=vendimia::findOrFail($id_muestra);
    if ($vendimia->delete()){
      return response()->json(["mensaje" => "Eliminado correctamente"], 200);
    }
  }
}
```

Figura 44: Controlador "vendimia"

En el siguiente código (Figura 45) se muestra un fichero de migración de la tabla “vendimia” donde se puede observar cómo se define la estructura de la tabla. Ese fichero lo tomará laravel con el comando “php artisan migrate” y creará en el servidor de base de datos la tabla.

```php
public function up()
{
    Schema::create('vendimia', function (Blueprint $table) {
      $table->bigIncrements('id_muestra');
      $table->unsignedBigInteger('id_cosecha');
      $table->integer('acidez');
      $table->integer('ph');
      $table->integer('temperatura');
      $table->integer('densidad');
      $table->date('fecha');
      $table->timestamps();
    });
}
```

Figura 45: Fichero de migración de la tabla "vendimia"

En el siguiente código (Figura 46) se muestra un fichero de sembrado de la tabla “vendimia” donde se puede observar cómo se hace la llamada a la biblioteca “faker” de generación de datos falsos y cómo se le asigna un valor a cada campo que se definió en el fichero de migración.

```
15      public function run()
16      {
17        $faker=Faker::create();
18        $insertar=10;
19
20        $matriz_ids=cosechas::pluck('id_cosecha')->toArray();
21        $num_ids=count($matriz_ids);
22
23        for ($i=0; $i<$insertar; $i++){
24          $vendimia= vendimia::create([
25            'id_cosecha'=>$matriz_ids[$faker->numberBetween($min=0, $max=$num_ids-1)],
26            'acidez'=>$faker->numberBetween($min=1, $max=2),
27            'ph'=>$faker->numberBetween($min=1, $max=3),
28            'temperatura'=>$faker->numberBetween($min=1, $max=4),
29            'densidad'=>$faker->numberBetween($min=1, $max=5),
30            'fecha'=>$faker->date(),
31          ]);
32        }
33      }
```

Figura 46: Fichero de sembrado de la tabla "vendimia"

Frontend

El siguiente código (Figura 47) muestra el servicio que conecta con la API del backend. Se puede observar cómo se hace la conexión con la API del backend mediante una llamada HTTP mediante el método adecuado según la función (GET, POST, PUT O DELETE) usando HttpClient y cómo se le pasa como argumento la url y un punto de contacto (“vendimia”, por ejemplo).

Así, por ejemplo se puede

- usar la función devolverTodas para obtener…
  - todas las muestras de la fase de vendimia llamando a this._servicioAPI.devolverTodas(“vendimia”). Esta función hará una petición GET HTTP://localhost:8000/vendimia que interceptará la API del frontend. Éste consultará su fichero de rutas y, si encuentra una coincidencia, llamará a la función del controlador que en ella se especifique.
  - todas las cosechas, llamando a this._servicioAPI.devolverTodas(“cosecha”)
- usar la función eliminar para borrar…
  - la cosecha con id=3 llamando a this._servicioAPI.eliminar(“cosecha”, 3)
  - la muestra de la fase de maduración con id=5 llamando a this._servicioAPI.elminar(“maduracion”, 5)

```
@Injectable()
export class APIService {
 base_url: string = 'http://127.0.0.1:8000/api/';
 cabecera = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })
 };

 constructor(private _http: HttpClient) {}

 devolverTodas($url_endpoint): Observable<any>{
   return this._http.get<any>(this.base_url+$url_endpoint);
 }

  devolverUna($url_endpoint, $id): Observable<any>{
    return this._http.get<any>(this.base_url+$url_endpoint+"/"+$id);
  }

  guardar($url_endpoint, cuerpo): Observable<any>{
    return this._http.post(this.base_url+$url_endpoint, cuerpo, this.cabecera);
  }

  eliminar($url_endpoint, $id: string): Observable<any>{
    return this._http.delete(this.base_url+$url_endpoint+"/"+$id);
  }

  actualizar($url_endpoint, $id, $cuerpo): Observable<any>{
    return this._http.put(this.base_url+$url_endpoint+"/"+$id, $cuerpo);
  }

  devolverParametro($url_endpoint, $parametro, $id_cosecha){
    return this._http.get<any>(this.base_url+$url_endpoint+"/"+$parametro+"/"+$id_cosecha);
  }
}
```

Figura 47: Fichero "API.service.ts"

El siguiente código (Figura 48) muestra el servicio “cosechas”, que sirve para almacenar la información sobre la cosecha actual y que todos los módulos puedan consultarla sin tener que hacer una llamada a la API del backend. Es necesario que consulten esta información para sacar por pantalla las muestras y las estadísticas de una cosecha concreta. Se pueden observar las siguientes funciones:

- elegirCosecha, que guarda en una variable, cosechaActual, la cosecha sobre la que se va a trabajar.
- devolverCosechaElegida, que devuelve la cosecha actual sobre la que se está trabajando.
- guardarCosecha, que llama a la API del frontend para reflejar el cambio que el usuario ha hecho en la cosecha actual y que al cargar la aplicación la próxima vez, se siga usando la misma.
- elegirParametros, que guarda en una variable los parámetros actuales. Servirá en un futuro para no tener que hacer una llamada a la API del frontend cuando se trate de recomendar tratamientos en función de los resultados obtenidos en las muestras y los parámetros ideales.

- devolverParametros, que devuelve los parámetros actuales. Servirá en un futuro para no tener que hacer una llamada a la API del frontend cuando se trate de recomendar tratamientos en función de los resultados obtenidos en las muestras y los parámetros ideales.

```
import { Injectable } from '@angular/core';
import { HttpClient,HttpHeaders } from '@angular/common/http';
import { definicionCosecha } from 'src/app/shared/modelos/cosechas.model';
import { definicionParametros } from 'src/app/shared/modelos/parametros.model';
import { Observable } from 'rxjs';
import { APIService } from 'src/app/shared/servicios/API.service';

@Injectable({
  providedIn: 'root'
})
export class CosechasService {
   cosechaActual: definicionCosecha;
   parametrosActuales: definicionParametros;

  constructor(
    private _servicioAPI: APIService
  ) {}

  elegirCosecha($nuevaCosechaActual){
      this.cosechaActual=$nuevaCosechaActual;
  }

  devolverCosechaElegida(){
    return (this.cosechaActual);
  }

  guardarCosecha($nuevaCosecha){
    const sinActual={"actual":"0"};
    this._servicioAPI.actualizar("cosecha", this.cosechaActual["id_cosecha"], sinActual).subscribe(datos=>{
      this._servicioAPI.guardar("cosecha", $nuevaCosecha).subscribe(datos=>{
        this.cosechaActual=$nuevaCosecha;
      });
    });
  }

  elegirParametros(parametros){
    this.parametrosActuales=parametros;
  }

  devolverParametros(){
    return (this.parametrosActuales);
  }
}
```

Figura 48: Fichero “cosechas.service.ts”

En el siguiente código (Figura 49) se muestra el servicio “sesion” que permite mantener información centralizada sobre el usuario que ha iniciado sesión. Es un servicio del que hacen uso todos los componentes para darle acceso al usuario, si ha iniciado sesión, o redirigirlo a la página de aterrizaje, si no. Se pueden observar las siguientes funciones:

- cerrarSesion, que elimina la variable “usuario” y pone sesionIniciada a false

- sesionEstaIniciada, que devuelve true si la sesión está iniciada y false si no.
- usuarioSesion, que devuelve el usuario que ha iniciado sesión.
- IniciarSesion, que almacena el usuario que está iniciando sesión. Esta función es llamada por el componente “acceso” y “registrar”.

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class SesionService {
7    sesionIniciada:boolean=false;
8    usuario: Array<any>=[];
9
10   constructor() {
11   }
12
13   cerrarSesion(){
14     this.sesionIniciada=false;
15     this.usuario=[];
16   }
17
18   sesionEstaIniciada(){
19     return (this.sesionIniciada);
20   }
21
22   usuarioSesion(){
23       return this.usuario;
24   }
25
26   iniciarSesion (usuario){
27     this.sesionIniciada=true;
28     this.usuario=usuario;
29   }
30 }
```

Figura 49: Fichero "sesion.service.ts"

En el siguiente código (Figura 50), que forma parte del fichero “configuracion.component.ts” que es parte del componente “configuracion” el cual se encarga de elegir la cosecha actual y los parámetros ideales de ese tipo de vino, se observan tres funciones:

- cargarCosechas, que se conecta al servicio “API.service.ts” para hacer una llamada a la API del backend y obtener información sobre las cosechas.
- cargarParametros, que se conecta al servicio “API.service.ts” para hacer una llamada a la API del backend y obtener información sobre los parámetros de ese tipo de uva.
- cambiarEleccion, se conecta al servicio “API.service.ts” para hacer una llamada a la API del backend y actualizar la tabla de cosechas para cambiar la cosecha actual. Esta función es llamada cuando el usuario cambia la cosecha actual.

```
60    cargarCosechas(){
61      this.hayParametros=false;
62      this._servicioAPI.devolverTodas("cosecha").subscribe(datos=>{
63        this.datosCosechas.data = datos;
64        datos.forEach(cosecha=>{
65          if (cosecha.actual){
66            this.cosechaActual=cosecha;
67            this._servicioCosechas.elegirCosecha(cosecha);
68          }
69        });
70        this.cargarParametros();
71      });
72      this.datosCosechas.paginator = this.paginador;
73      this.datosCosechas.sort = this.sort;
74    }
75
76    cargarParametros(){
77  //para cambiar variedad_uva por tipo_vino, es necesario hacerlo aquí y en controladorParametros
78      this._servicioAPI.devolverUna("parametros", this.cosechaActual["variedad_uva"]).subscribe(datos=>{
79        if (datos.length>0){
80          this.parametrosActuales=datos[0];
81          this.hayParametros=true;
82          //para esperar a que termine de cargar para renderizar el HTML
83          this.parametrosActualesCargados=Promise.resolve(true);
84          this.datosParametros.data = datos;
85          this._servicioCosechas.elegirParametros(datos[0]);
86        }
87
88      });
89    }
90
91    cambiarEleccion($nuevaCosechaActual){
92     //this.cosechaActual contiene la que, hasta ahora, era la cosecha actual. $nuevaCosechaActual es la nueva, elegida por el usuario
93     const sinActual={"actual":"0"};
94     const conActual={"actual":"1"};
95     this._servicioAPI.actualizar("cosecha", this.cosechaActual["id_cosecha"], sinActual).subscribe(datos=>{
96       this._servicioAPI.actualizar("cosecha", $nuevaCosechaActual["id_cosecha"], conActual).subscribe(datos=>{
97         this.cargarCosechas();
98       });
99     });
100     this.paso=2;
101   }
```

Figura 50: Parte del fichero "configuracion.component.ts"

En el siguiente código (Figura 51), que forma parte del fichero “parametros.component.ts” que es parte del componente “parametros” el cual se encarga de mostrar un formulario para editar los parámetros ideales del tipo de vino actual (o introducirlos si no existen en la base de datos), se observan las siguientes funciones:

- crearFormulario, que crea un formulario reactivo y proporciona un validador para cada campo.
- cargarParametros, que carga los parámetros ideales de la cosecha actual.
- rellenarFormulario, que rellena el formulario en caso de que ya existiesen datos sobre los parámetros ideales del tipo de vino de la cosecha actual.
- guardarParametros, que hace una llamada al servicio “API.service.ts” que conecta con la API del backend para almacenar en la base de datos los parámetros, si no había, o acualizarlos, si ya había.

```
46  crearFormulario() {
47    this.formulario = this._builder.group({
48      tipo_vino: new FormControl (''),
49      variedad_uva: new FormControl (''),
50      sulfuroso: new FormControl('', [Validators.required, Validators.pattern("^[0-9]*$")]),
51      grado: new FormControl('', [Validators.required, Validators.pattern("^[0-9]*$")]),
52      gluconico: new FormControl('', [Validators.required, Validators.pattern("^[0-9]*$")]),
53      malico: new FormControl('', [Validators.required, Validators.pattern("^[0-9]*$")]),
54      cata: new FormControl('', [Validators.required, Validators.pattern("^[0-9]*$")]),
55      acidez: new FormControl('', [Validators.required, Validators.pattern("^[0-9]*$")]),
56      ph: new FormControl('', [Validators.required, Validators.pattern("^[0-9]*$")])
57    });
58  }
59
60  rellenarFormulario(){
61    this.formulario.get('tipo_vino').setValue(this.cosechaActual.tipo_vino);
62    this.formulario.get('variedad_uva').setValue(this.cosechaActual.variedad_uva);
63    //si hay parámetros ya definidos
64    if (this.parametros){
65      this.formulario.get('sulfuroso').setValue(this.parametros.sulfuroso);
66      this.formulario.get('grado').setValue(this.parametros.grado);
67      this.formulario.get('gluconico').setValue(this.parametros.gluconico);
68      this.formulario.get('malico').setValue(this.parametros.malico);
69      this.formulario.get('cata').setValue(this.parametros.cata);
70      this.formulario.get('acidez').setValue(this.parametros.acidez);
71      this.formulario.get('ph').setValue(this.parametros.ph);
72    }
73  }
74
75  cargarParametros(){
76    this.parametros=this._servicioCosechas.devolverParametros();
77    this.cosechaActual=this._servicioCosechas.devolverCosechaElegida();
78  }
79
80  guardarParametros(){
81    if (this.parametros){
82      this._servicioAPI.actualizar ("parametros", this.parametros.id_parametros, this.formulario.value).subscribe (datos=>{
83      });
84    }else{
85      this._servicioAPI.guardar("parametros", this.formulario.value).subscribe(datos=>{
86        this._servicioCosechas.elegirParametros(this.formulario.value);
87      });
88    }
89    this._location.back();
90    this.terminado=true;
91  }
```

Figura 51: Parte del fichero "parametros.component.ts"

El siguiente código (Figura 52), forma parte del fichero “estadisticas.component.ts” que es parte del componente “estadisticas”. Al cargarlo, se ejecuta la función ngOninit, donde se llama al servicio “cosechas.service” para obtener la cosecha actual y se hacen 4 llamadas a dos funciones cuyo objetivo es conectarse con la API del backend para tomar datos y adaptarlos al formato que la biblioteca ng2-charts necesita para mostrar los datos:

- rellenarDatosUltima, que almacena los siguientes valores de la cosecha actual:
  - en un objeto, pasado como argumento (por referencia), los valores del parámetro pasado por argumento (ph y acidez) en todas las fases de la cosecha actual.
  - en una matriz, global (no puede ser pasada como referencia, sino como valor, por no ser un objeto), una etiqueta describiendo la cosecha actual (tipo de vino y año).
- rellenarDatosHistoricos, que almacena los siguientes valores de las tres últimas cosechas.

- en un objeto, pasado como argumento (por referencia), los valores del parámetro pasado por argumento (ph y acidez) en todas las fases de las tres últimas cosechas.
- en una matriz, global (no puede ser pasada como referencia, sino como valor, por no ser un objeto), las etiquetas que desriben las tres últimas cosechas de las que se están tomando los datos (tipo de vino y año).

```
58  ngOnInit() {
59    this.cosechaActual=this._servicioCosechas.devolverCosechaElegida();
60
61    //lineChartLabels es pasado como valor, por lo que no se modifica en el interior de la función si se pasa como argumento
62    //lineChartData, al ser un objeto, es pasado como referencia, por lo que sí se modifica dentro de la función
63    this.rellenarDatosUltima("ph", this.lineChartData);
64    this.rellenarDatosHistoricos("ph", 3, this.lineChartData2);
65    this.rellenarDatosUltima("acidez", this.lineChartData3);
66    this.rellenarDatosHistoricos("acidez", 3, this.lineChartData4);
67  }
68
69
70  //No se puede pasar lineChartLabels como argumento, ya que no es un objeto, sin ouna matriz. Es pasado como valor
71  rellenarDatosUltima ($parametro, $datosGrafico){
72    let leyenda="";
73    let etiquetasLinea=[];
74    /*Hay que definir previamente lineChartData, pero cuando se hace todavía no se sabe los datos que se crearán.
75    Por eso se crea con 3 filas vacías y ahora hay que dejar solo una para que no aparezcan en la tabla filas sin datos*/
76    $datosGrafico.length=1
77
78    let datosGraficoTMP:Array<number>=[];
79    //etiqueta visible en la leyenda
80    leyenda=$parametro+" cosecha "+this.cosechaActual['tipo_vino']+"/"+this.cosechaActual['anyo'];
81
82    //obtengo los datos de ese parámetro de todas las muestras para cada la cosecha actual
83    this._servicioAPI.devolverValorMuestras("estadisticas", $parametro, this.cosechaActual['id_cosecha']).subscribe(muestras=>{
84      //meto los valores de cada parámetro
85      muestras.forEach(valoresMuestra=>{
86        //como el gráfico solo muestra una serie, se usa como leyenda de la línea X la fecha de toma de la muestra
87        let fecha=((valoresMuestra['fecha']) as Date);
88        etiquetasLinea.push(this._datepipe.transform(fecha,"dd-MM-yyyy"));
89        //los meto en una matriz temporal
90        datosGraficoTMP.push((valoresMuestra[$parametro]) as number);
91      });
92      $datosGrafico[0].data=datosGraficoTMP;
93      $datosGrafico[0].label=leyenda;
94      this.lineChartLabels=etiquetasLinea;
95    });
96  }
```

Figura 52: Parte del fichero "estadisticas.component.ts"

El siguiente código (Figura 53), que forma parte del fichero “estadisticas.component.html” muestra como se hacen cuatro llamadas al componente app-grafico, que es donde se realiza el pintado de la gráfica, con los parámetros que contienen los datos necesarios para pintar el gráfico.

```
<app-grafico
  [titulo]="titulos[0]"
  [datosGrafico]="lineChartData"
  [etiquetasGrafico]="lineChartLabels"
  >
</app-grafico>

<app-grafico
  [titulo]="titulos[1]"
  [datosGrafico]="lineChartData2"
  [etiquetasGrafico]="lineChartLabels"
  >
</app-grafico>
<app-grafico
  [titulo]="titulos[2]"
  [datosGrafico]="lineChartData3"
  [etiquetasGrafico]="lineChartLabels"
  >
</app-grafico>

<app-grafico
  [titulo]="titulos[3]"
  [datosGrafico]="lineChartData4"
  [etiquetasGrafico]="lineChartLabels"
  >
</app-grafico>

```

Figura 53: Fichero "estadisticas.component.html"

Anexo 3.– Bibliotecas externas

Para el desarrollo de la aplicación se han usado las siguientes bibliotecas externas

- Bootstrap, que es una biblioteca multiplataforma para la maquetación de sitios web. Se ha usado para permitir un comportamiento adatpable (responsive) y para facilitar el desarrollo. Para usarla, es necesario instalarla usando

```
npm install --save bootstrap jquery popper.js
```

Una vez instalada, su estilo se aplica definiendo las clases deseadas de todo el juego que tiene dentro de las etiquetas HTML. Por ejemplo, en el siguiente código (Figura 54) se puede observar el uso de bootstrap, en las etiquetas `nav, a, div,` etc. mediante el uso de las clases `navbar, navbar-expand, nav-item`, etc.

```
<section>
  <nav class="navbar navbar-expand-md navbar-light bg-light">
    <a class="navbar-brand" routerLink="/aterrizaje/portada">
      <img src="assets/imagenes/logotipo.png" alt="">
  </a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="r
      <span class="navbar-toggler-icon"></span>
    </button>
   <div class="navbar-collapse collapse w-100 order-3" id="navbarSupportedContent">
        <ul class="navbar-nav ml-auto">
          <li class="nav-item">
            <a class="nav-link" routerLink="/aterrizaje/acceso/">Acceso</a>
          </li>
          <li class="nav-item active">
            <a class="nav-link" routerLink="/aterrizaje/registro/">Registro <span class="sr-only">(current)</span></a>
          </li>
        </ul>
    </div>
  </nav>
</section>
<section class="contenedorCuerpo">
  <router-outlet></router-outlet>
</section>
```

Figura 54: Parte del fichero "aterrizaje-disposicion.component.ts"

- ng2-chart, que es una biblioteca que permite dibujar gráficos, siempre que se le suministren los datos según un formato determinado. Para instalarla es necesario usar el siguiente comando:

```
npm install --save ng2-charts
```

La he utilizado para dibujar las estadísticas en el componente “estadisticas.component.ts” (Figura 52), que usa el servicio “API.service.ts” para conectarse a la API del frontend, obtener los datos, tratarlos para dejarlos con el formato que necesita el gráfico y llamar cuatro veces (Figura 53), una por cada gráfico, al componente “grafico.component.ts” (Figura 55) que es el que pinta el gráfico.

```
<div fxFlex fxLayout="column" fxLayoutAlign="center center">
  <div class="flex-item">
    <div style="display: block;">
      <h5 class="text-center mt-3">{{titulo}}</h5>
    <canvas baseChart width="500" height="400"
                [datasets]="datosGrafico"
                [labels]="etiquetasGrafico"
                [options]="lineChartOptions"
                [colors]="lineChartColors"
                [legend]="lineChartLegend"
                [chartType]="lineChartType"
                [plugins]="lineChartPlugins"
                (chartClick)="chartClicked($event)"></canvas>
    </div>
  </div>
  <div class="flex-item">
    <table class="table table-responsive table-condensed">
      <tr>
        <th>
          Nº de muestra
        </th>
        <th *ngFor="let label of etiquetasGrafico">{{label}}</th>
      </tr>
      <tr *ngFor="let d of datosGrafico; let i=index" [class]="'line-'+i">
        <th>
          Valor
        </th>
        <td *ngFor="let label of etiquetasGrafico; let j=index">{{d && d.data[j]}}</td>
      </tr>
    </table>
  </div>
    <div>
      <button mat-button color="primary" (click)="changeColor()">Recolorear</button>
    </div>
</div>
```

Figura 55: Fichero "grafico.component.html"

- Fontawesome, que es una biblioteca que contiene una vasta cantidad de iconos, divididos en colecciones (core, icons, regular, brands, solid) muchos de ellos gratuitos. Los iconos que contiene son de calidad, puesto que su formato es vectorial por lo que se pueden reescalar sin perder calidad. Dispone de versión para angular, instalable mediante el siguiente comando:

```
npm install @fortawesome/fontawesome-svg-core
@fortawesome/fontawesome-svg-icons @fortawesome/free-regular-svg-icons
@fortawesome/free-brands-svg-icons @fortawesome/free-solid-svg-icons
@fortawesome/angular-fontawesome
```

I

Una vez instalada es necesario, en shared.module.ts importar algunos ficheros, importar cada icono que se vaya a usar y añadirlos a una biblioteca (Figura 56). También es necesario importar la línea 11 de la Figura 56 en el componente donde se vaya a usar.

```
11  import { FaIconLibrary, FontAwesomeModule } from '@fortawesome/angular-fontawesome'
12  import { faFacebookSquare, faTwitter, faLinkedin, faDribbble, faYoutube } from '@fortawesome/free-brands-svg-icons';
13  import { faTrash, faTrashAlt, faPlus, faMinus/*, faUser*/ } from '@fortawesome/free-solid-svg-icons';
14  import { faUser, faEye } from '@fortawesome/free-regular-svg-icons';
15
16  // DIRECTIVES
17
18  // PIPES
19
20  // SERVICES
21  //import { AppConfirmService } from './services/app-confirm/app-confirm.service';
22
23  const declarations = [
24    AterrizajeDisposicionComponent,
25    AdminLayoutComponent];
26  const exports = [
27    CommonModule,
28    FormsModule,
29    ReactiveFormsModule,
30    RouterModule,
31    AterrizajeDisposicionComponent,
32    AdminLayoutComponent
33  ];
34  //const providers = [AppConfirmService];
35
36  @NgModule({
37    imports: [CommonModule, FormsModule, ReactiveFormsModule, RouterModule,
38      MaterialModule,
39      FontAwesomeModule,
40     ],
41    //entryComponents: [AppComfirmComponent],
42    //providers,
43    declarations,
44    exports
45  })
46
47  export class SharedModule {
48    constructor (library: FaIconLibrary){
49      library.addIcons(faFacebookSquare, faTwitter, faLinkedin, faDribbble, faYoutube, faTrash, faTrashAlt, faUser, faEye, faPlus, faMinus);
50    }
51  }
```

Figura 56: Fichero shared.module.ts donde se incluye *FontAwesome*

Anexo 4.– Guía de usuario

Cuando usted acceda a la web se encontrará con una pantalla de bienvenida con información general sobre la aplicación (Figura 57). Para ver el resto de funcionalidad de la aplicación es necesario identificarse. Para ello, tiene dos opciones (ver cuadrado rojo):

- pinche en “acceso”, lo que le llevará a una pantalla (Figura 58) donde tendrá que rellenar sus datos y pinchra en “acceso”.
- Pinche en “registro”, lo que le llevará a una pantalla (Figura 59), donde tendrá que rellenar sus datos y pinchar en “registrarse”

Si todo ha ido bien, tendrá acceso al resto de funcionalidad.

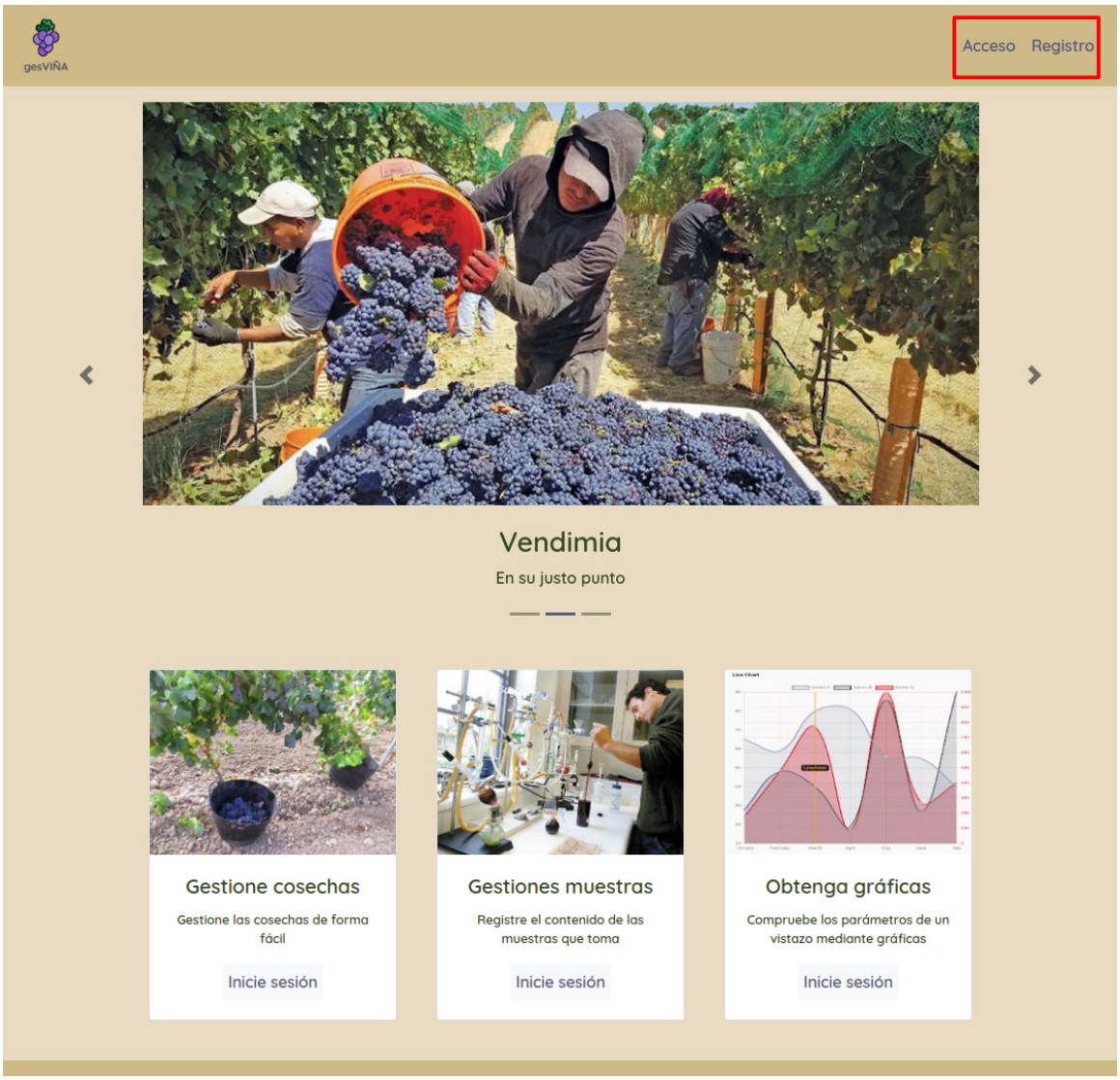


Figura 57: Página de aterrizaje

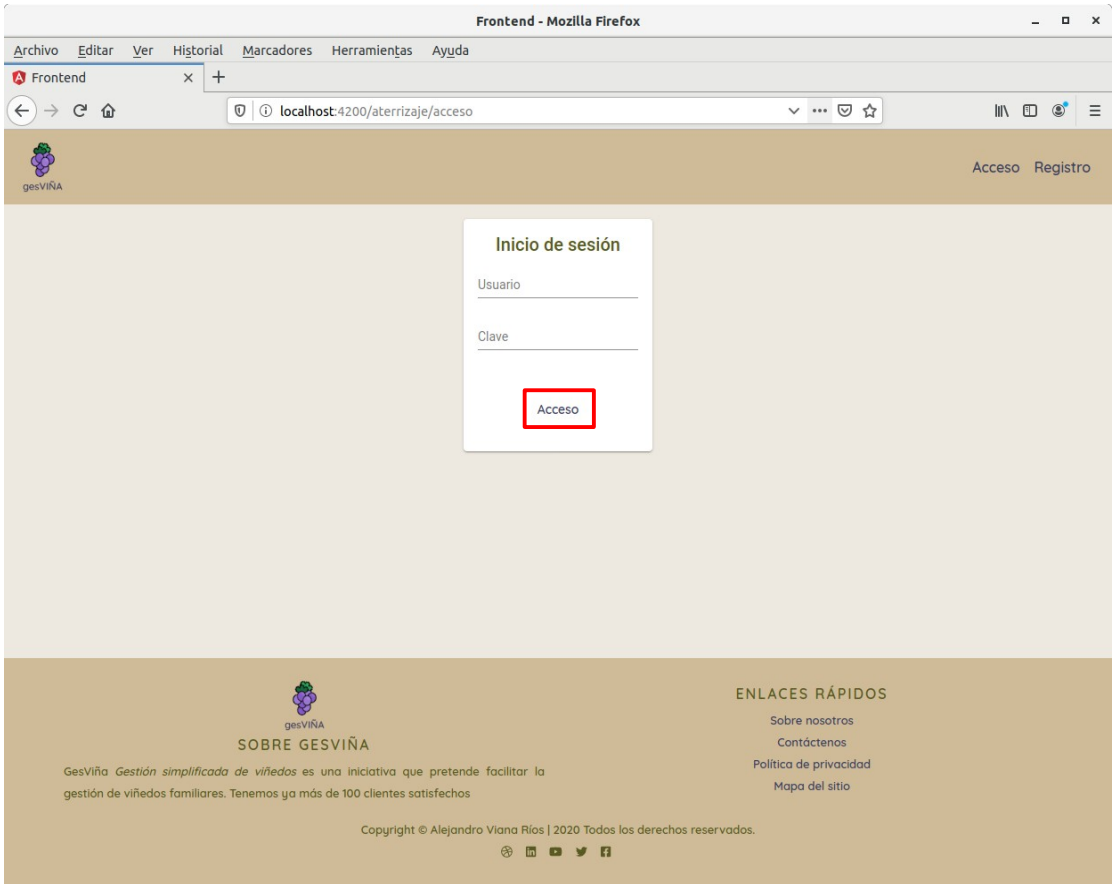


Figura 58: Página de identificación

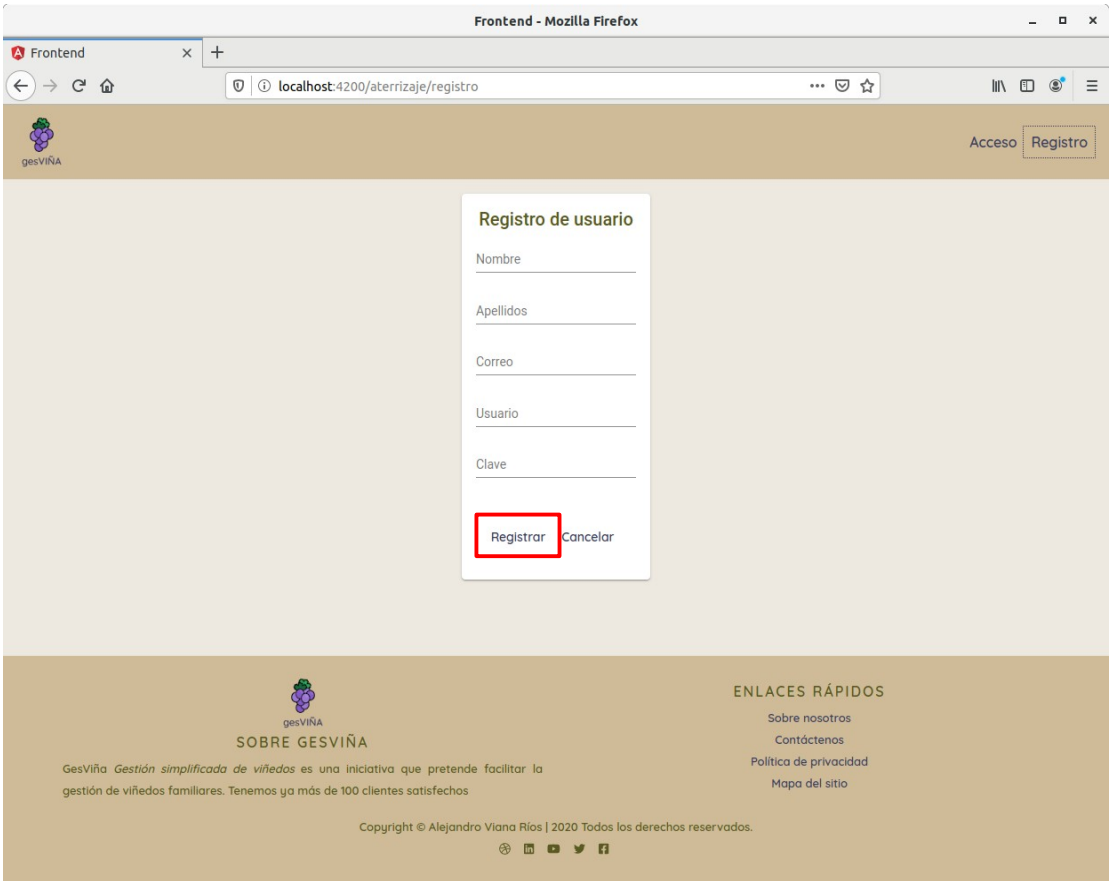


Figura 59: Página de registro

Cuando haya iniciado sesión correctamente, la aplicación le redirigirá a la página de configuración, ya que es necesario tener definida una cosecha actual, que es aquella sobre la que se introducen las muestras. El sistema escogerá como actual la última que haya sido creada o elegida por usted, aunque es posible cambiarla pinchando en “ver” (cuadrado rojo de la Figura 60) y luego en elegir sobre la que desee (Figura 61).

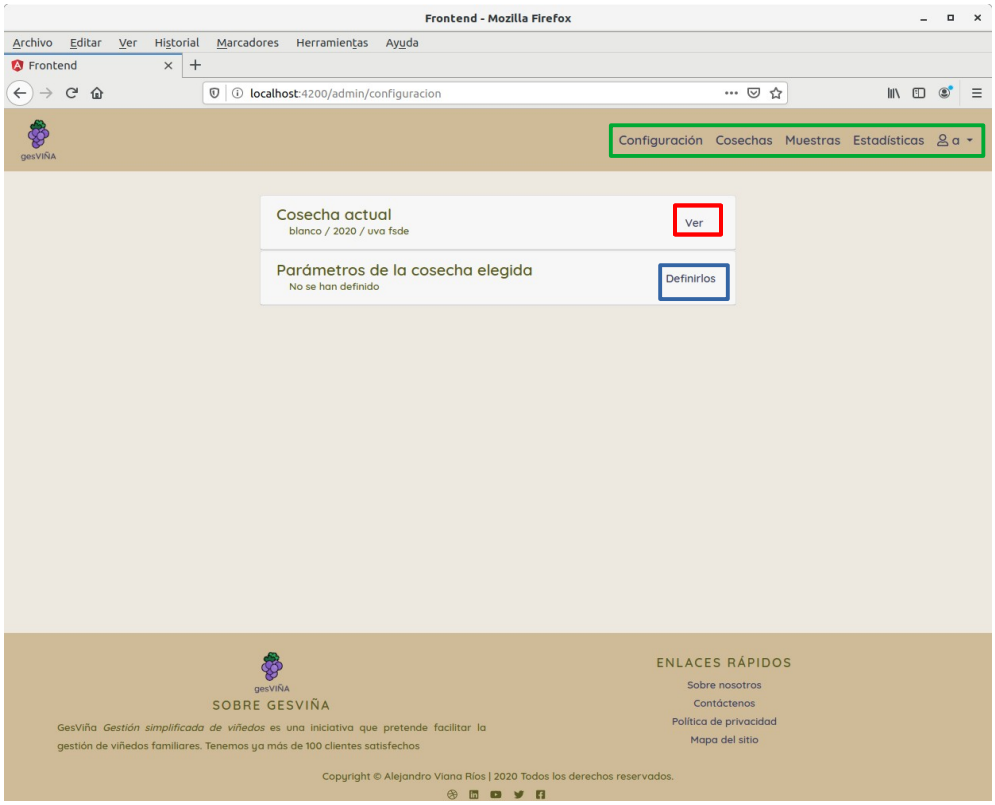


Figura 60: Panel de configuración

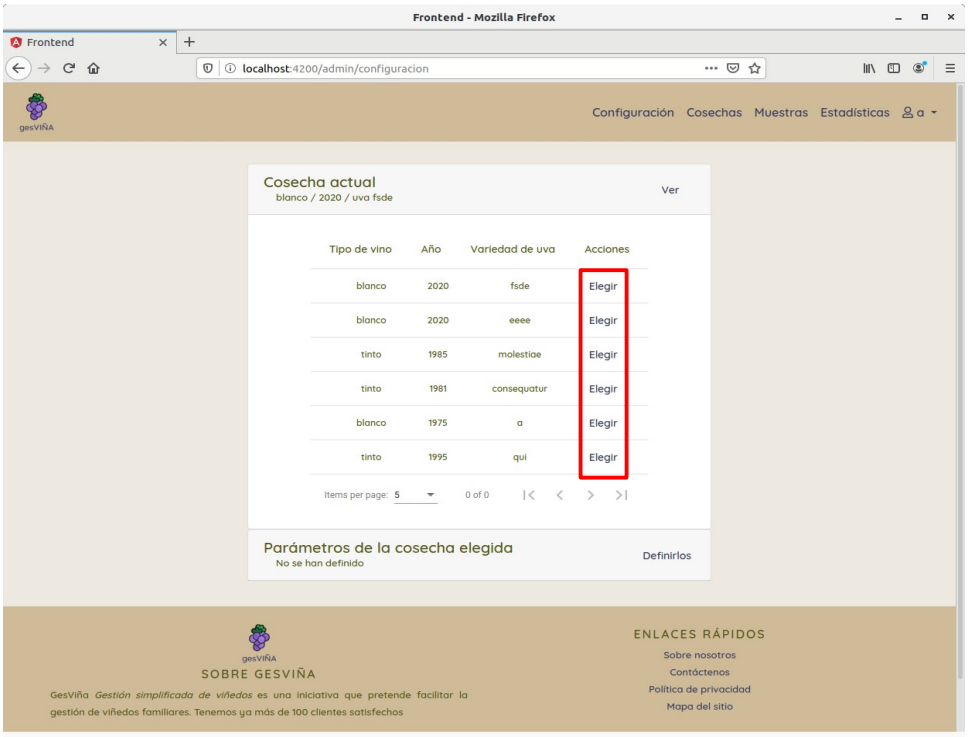


Figura 61: Panel de configuración con la cosecha desplegada para elegir

El panel de configuración (cuadrado azul de la Figura 60) también permite editar o añadir, si no los tenía, los parámetros ideales para el tipo de vino de la cosecha actual (Figura 62). Esta característica aún no se utiliza, pero en una futura versión, se podría comparar los parámetros ideales con los valores tomados en las muestras y, en función de la desviación, recomendar determinados tratamientos, por lo que es recomendable definirlos ya.

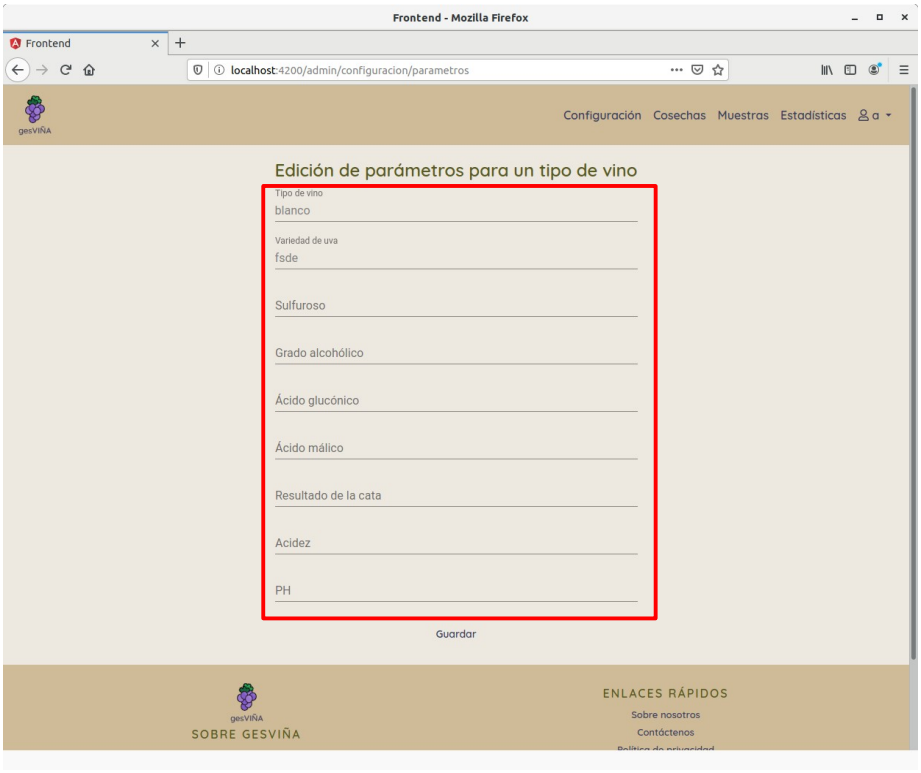


Figura 62: Panel para definir los parámetros ideales para un tipo de vino

Dado que el sistema elige la cosecha actual, puede elegir no configurar nada y navegar por las opciones que son, además de la página de configuración, las siguientes (Cuadrado verde de la Figura 60):

- cosechas, donde se pueden eliminar y añadir cosechas.
  - Para eliminar una cosecha pulse en el icono del cubo de basura a la derecha de la cosecha elegida (cuadrado rojo de la Figura 63). Si se elimina una cosecha, también se eliminan sus muestras en la base de datos y si la eliminada es la actual (que está marcada en rojo), el sistema designa como actual la anterior.
  - Para crear una cosecha nueva, pulse en el cuadrado azul de la Figura 63, lo que le llevará a la Figura 64. Rellene los datos y pulse en “guardar”. Al añadir una cosecha el sistema la elige como actual.
- muestras, donde se ven las muestras que hay en cada fase de la cosecha actual y se puede hacer lo siguiente:
  - añadir nuevas muestras de cualquier fase. Para ello en la Figura 65, pinche en el botón “nueva muestra” (cuadrado rojo) a la derecha de la fase donde desee añadirla, lo que le llevará a un formulario como el de la Figura 66.

    - eliminar las que hay. Para ello, pinche en “ver” las muestras en la fase que desee (cuadrado azul de la Figura 65) y luego en el cubo de basura a la derecha de la muestra que desee eliminar (cuadrado verde de la Figura 65).

- estadísticas, donde se ven, de la cosecha actual, los siguientes gráficos (Figura 67):
    - PH de todas las muestras de todas las fases de la cosecha actual.
    - PH de todas las muestras de todas las fases de las tres últimas cosechas.
    - acidez de todas las muestras de todas las fases de la cosecha actual.
    - acidez de todas las muestras de todas las fases de las tres últimas cosechas.
- Salir, que cierra la sesión y redirige a la página de aterrizaje.

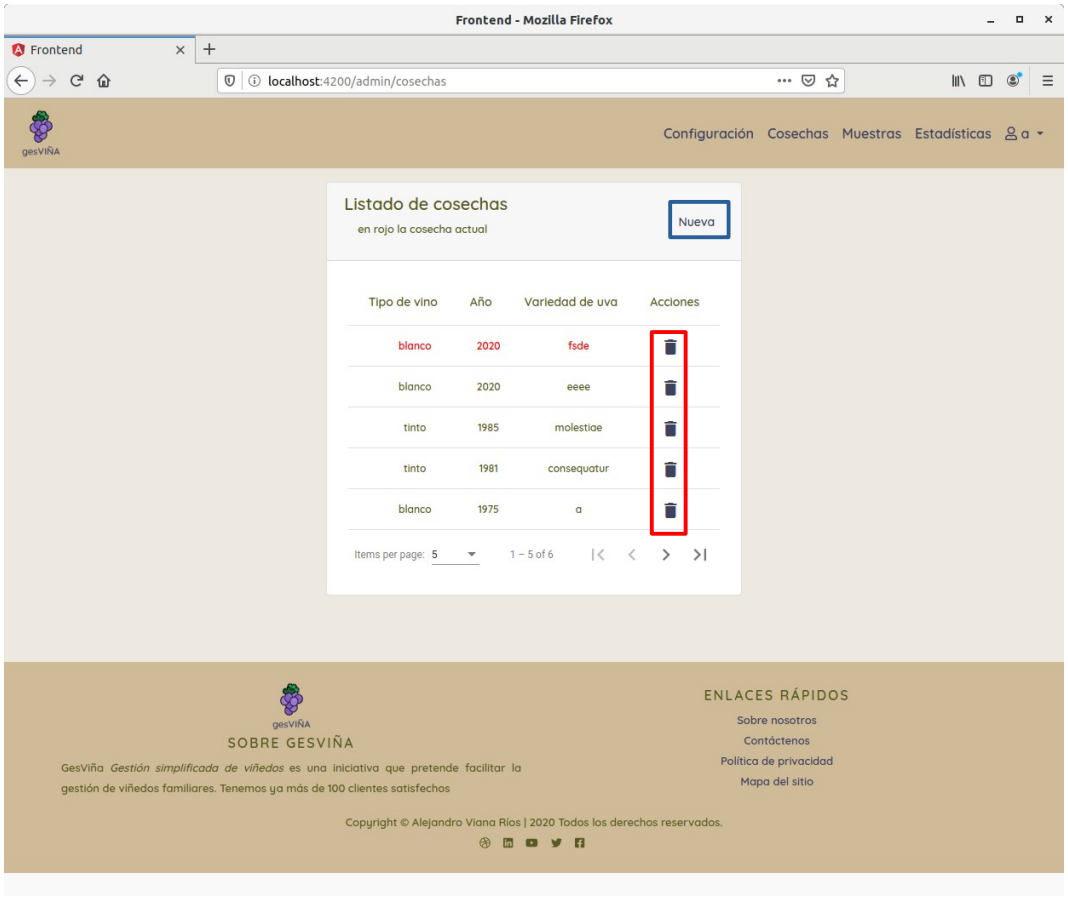


Figura 63: Panel de gestión de cosechas

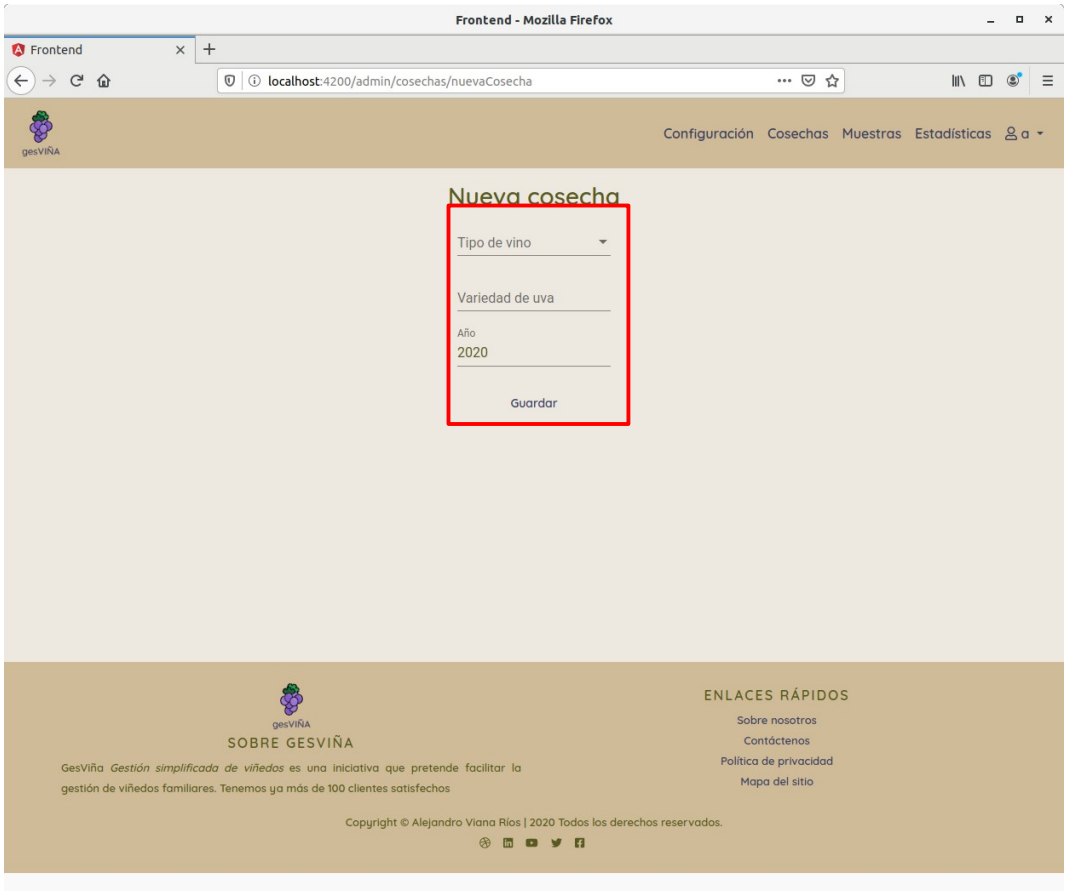


Figura 64: Panel de creación de nueva cosecha

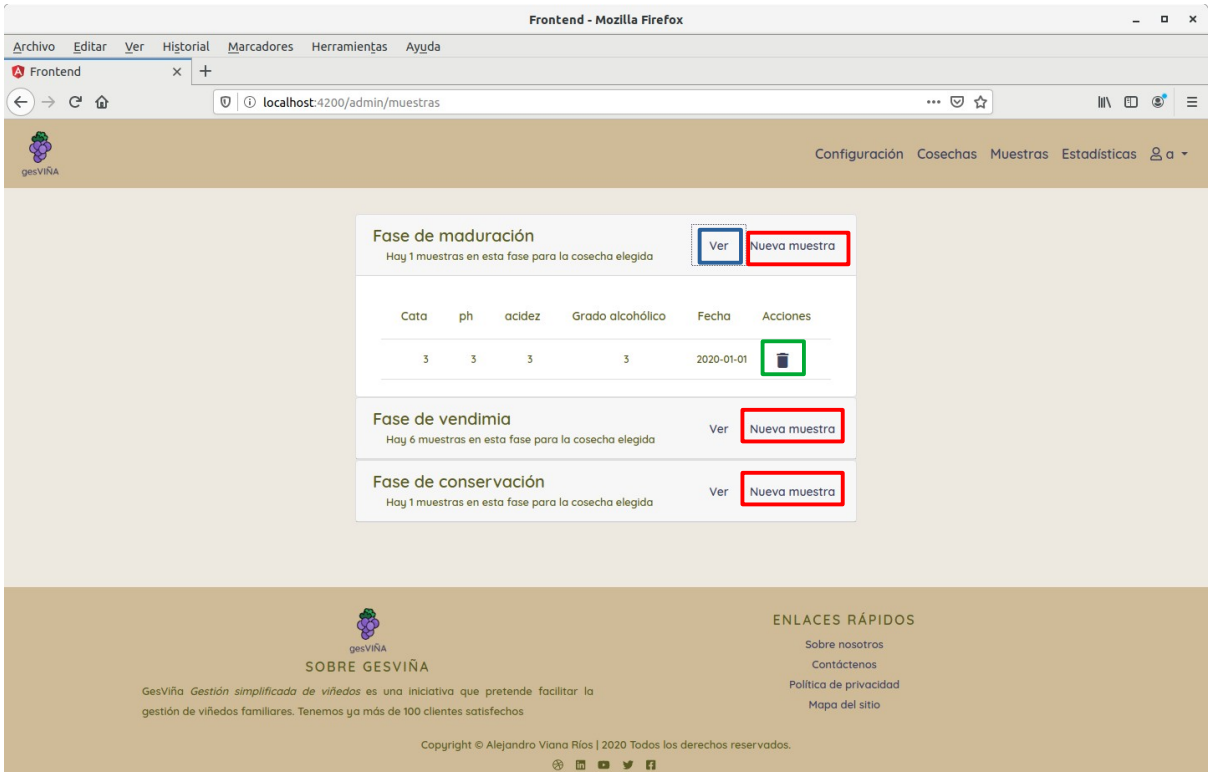


Figura 65: Página de muestras

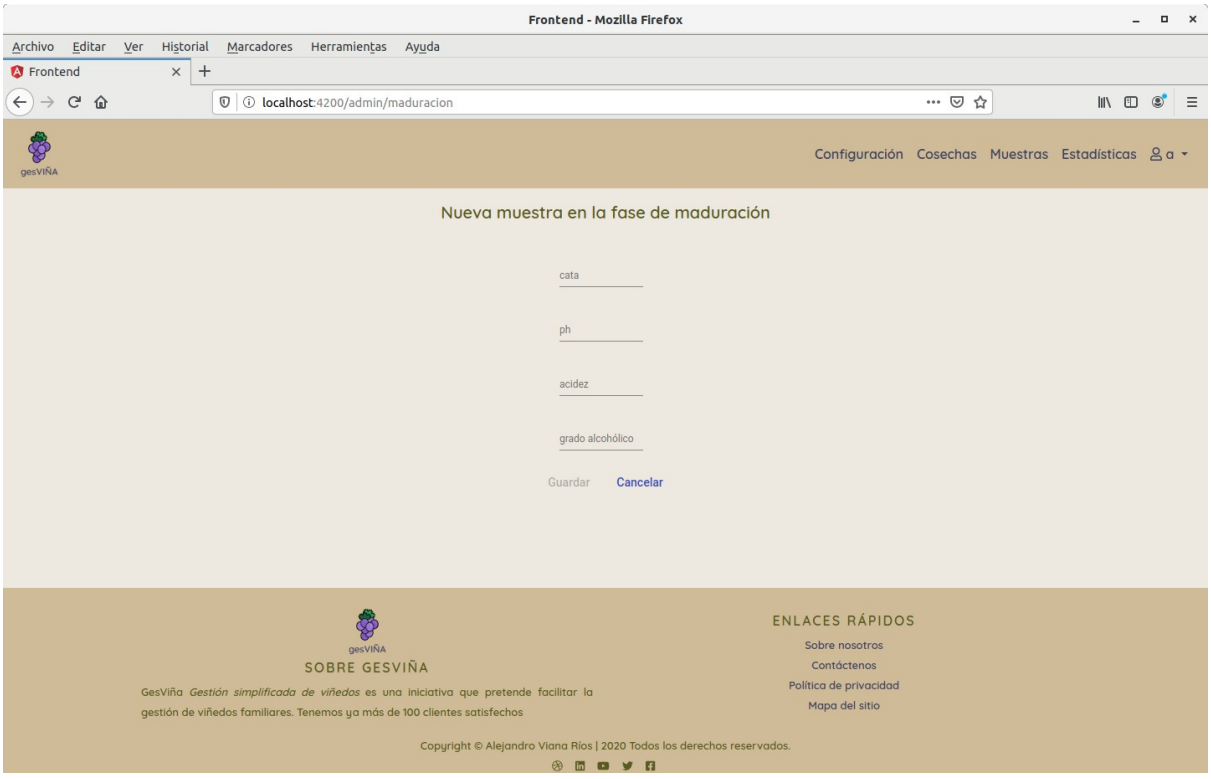


Figura 66: Página para introducir los valores de una muestra

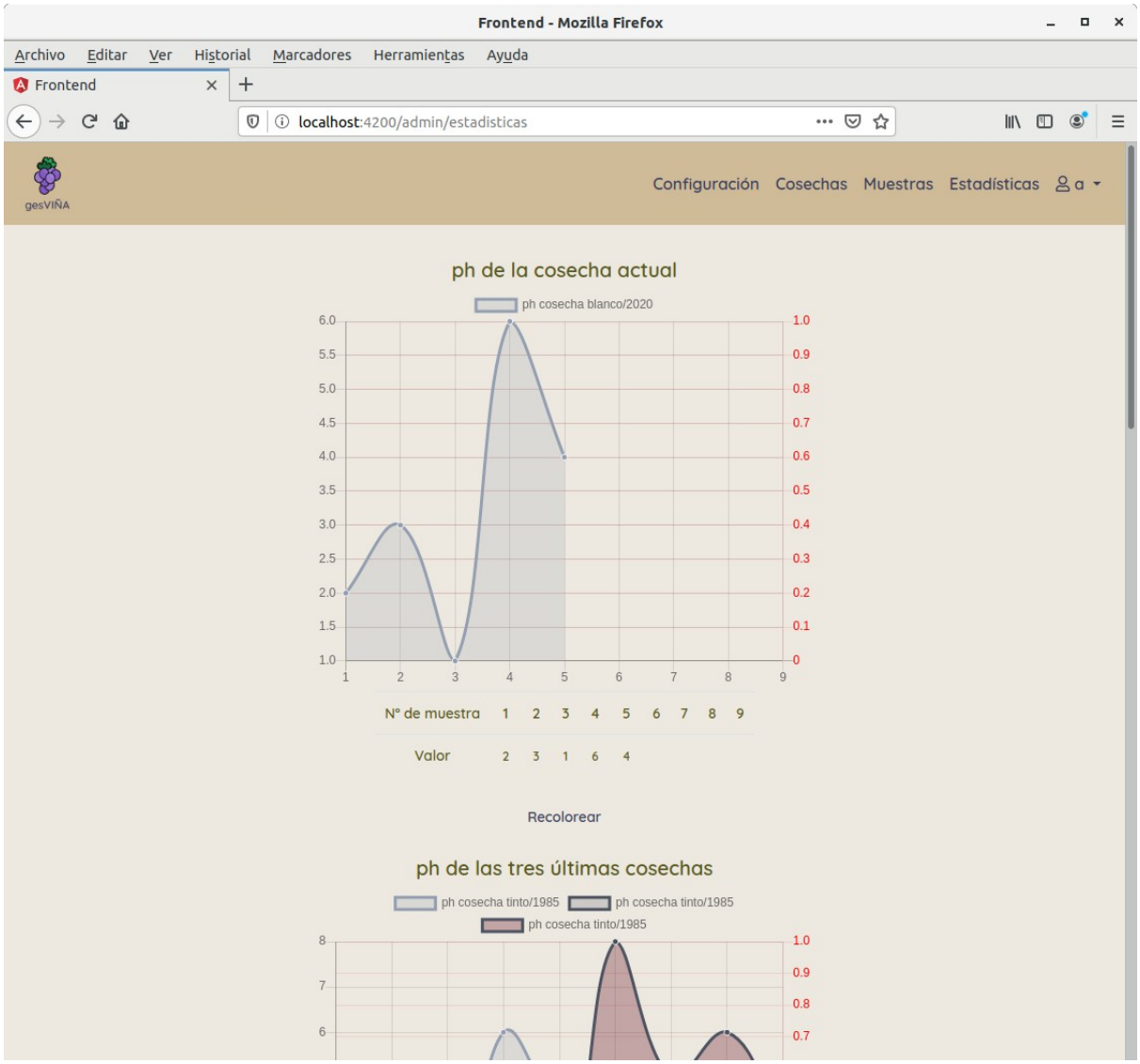


Figura 67: Pantalla con los gráficos

Anexo 5.– Libro de estilos

El libro de estilo es aquel documento que contiene los criterios y pautas necesarias para homogeneizar la representación gráfica de un proyecto. En esta aplicación se han usado las siguientes convenciones gráficas:

- Icono: Se ha usado un icono simbolizando un racimo de uvas, obtenido de la web de iconos gratis icon-icons.com (https://icon-icons.com/icon/grapes/100772)
- Imágenes: se han obtenido las siguientes imágenes de fuentes donde no se especifica que estén protegidas por derechos de autor:
  - Carrusel: Tendrán un tamaño de 800 x 385 px aproximadamente
    - Racimo de uvas: obtenida de una revista cristiana, directamente desde el buscador. No he encontrado el artículo donde está encuadrada.
    - vendimia: obtenida de este blog.
    - Barriles: obtenida de este blog.
  - Imágenes de las tarjetas en la página de aterrizaje. Tendrán un tamaño de 250 x 190 px aproximadamente.
    - Cosecha: obtenida de blog
    - Medición de parámetros: obtenida de blog.
    - Gráfico: página oficial de ng2-chart
- Colores: He creído que le daría consistencia a la página elegir una paleta de colores relacionada con el ámbito al que va dirigido la web: un racimo de uvas creciendo en un viñedo. Para ello he usado la web palettegenerator que permite, a partir de una imagen, obtener una paleta de entre 2 y 10 colores. He subido la imagen del racimo de uvas y eligiendo un paleta de 8 colores he escogido los siguientes:
  - Para el color de la barra de navegación y el pie de página un tierra oscuro: el tercero de la Figura 68, #D1B886.
  - Para el color del cuerpo de todas las páginas HTML un tierra claro: el primero de la Figura 68, #EADAC1.
  - Para el color de los enlaces, un color azul uva: el quinto de la Figura 68, #555a79.

- Para el color del texto normal, un color verde hoja: el último de la Figura 68, #374523.

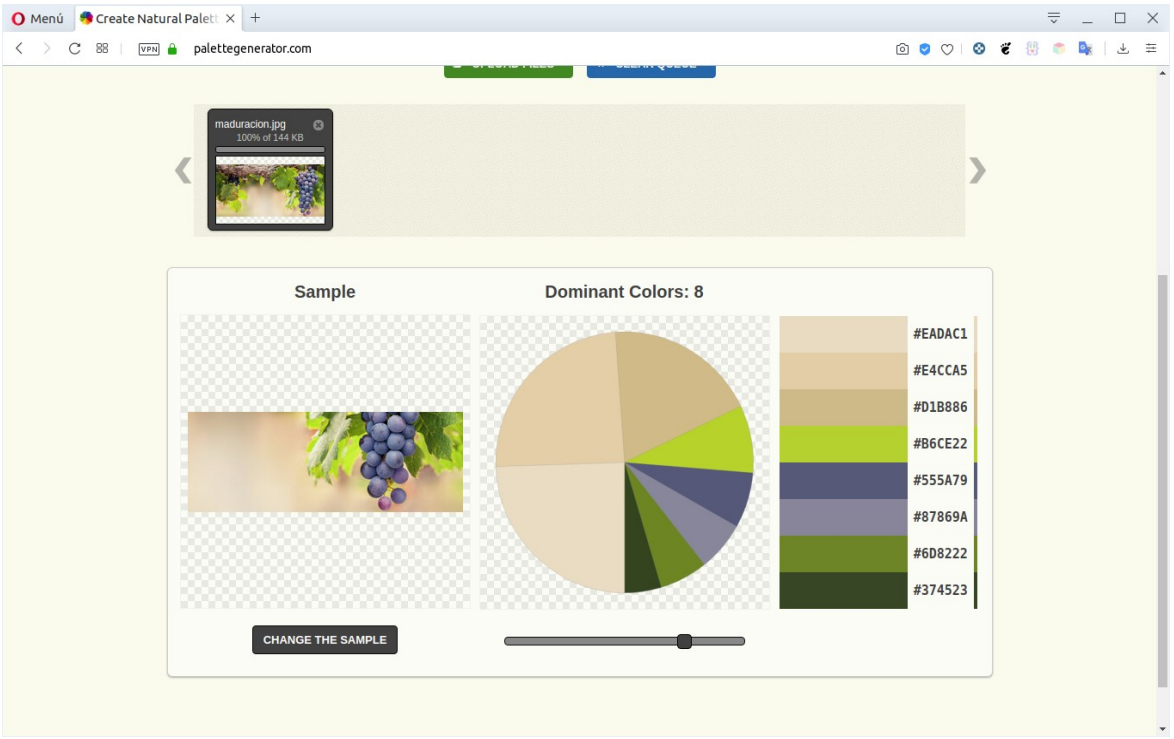


Figura 68: Paleta de colores presente en la imagen del racimo de uvas

- Tipografía: Se ha usado una tipografía Quicksand a 600pt de google fonts (Figura 69). Es una tipografía sin serifa, bonita, fresca y compacta pero con gran legilibilidad, adecuada a la lectura en pantalla.

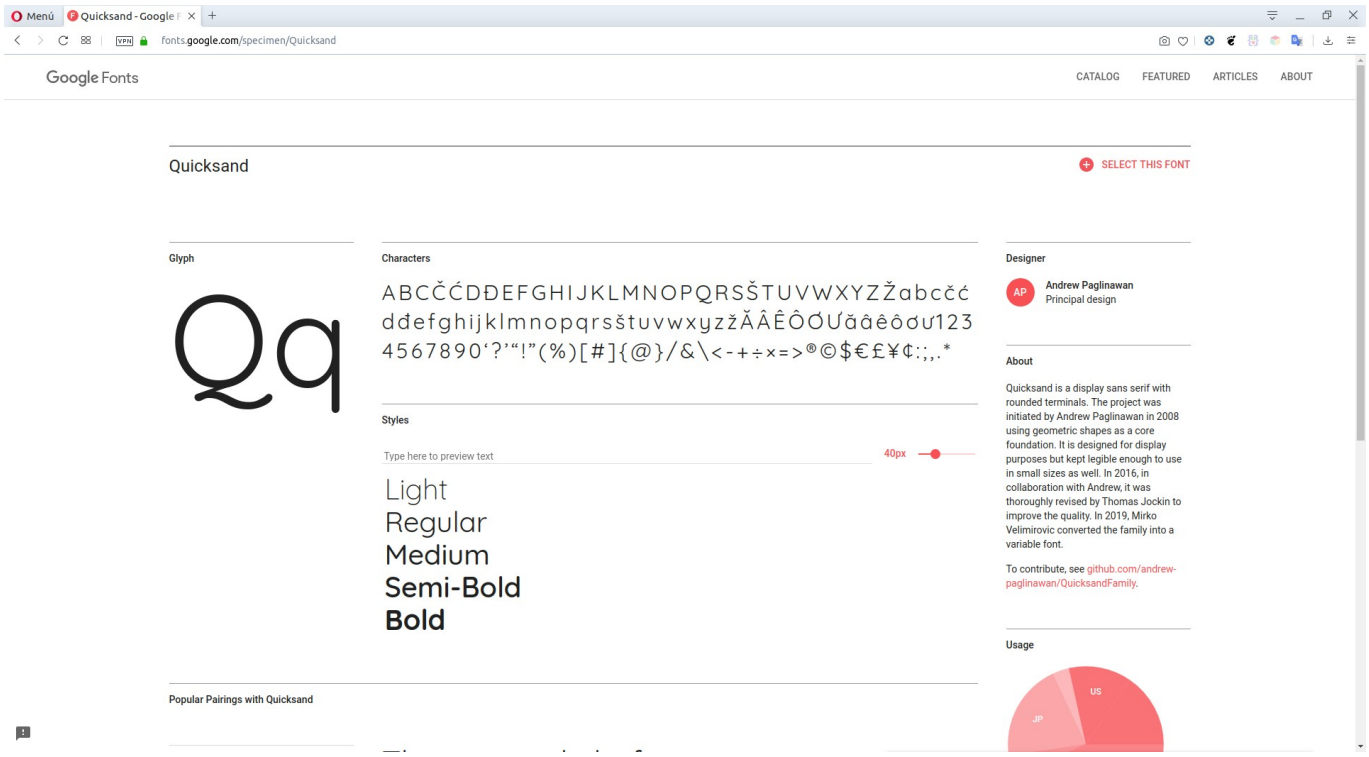


Figura 69: Tipografía elegida

Anexo 6.– Bibliografía

Para el desarrollo de este proyecto se han usado los siguientes recursos.

- Apuntes y PECs del resto de asignaturas de la maestría de desarrollo de sitios web de la UOC.
- Backend
  - Building and consuming a RESTful API in Laravel PHP
  - Usando faker para generar información de relleno para pruebas automatizadas
  - Biblioteca faker
  - Laravel
- Frontend
  - Is Angular Flex Layout (with angular material) better than bootstrap for responsive layout?
  - Integrate FontAwesome icons in an Angular application
  - Angular Components: Pass by reference of pass by value?
  - 3 ways to pass async data to angular 2+ child components
  - Angular module loading…
  - ¿Cómo implementar Lazy Loading en Angular?
  - Working with environment variables in angular 6
  - CRUD operation with Fake HTTP Service in Angular – Part Nine
  - Angular material data table: a complete example (server, pagination, filtering, sorting)
  - Angular responsive layout using flex layout
  - https://medium.com/angular-in-depth/angular-flex-layout-the-masquerade-58fb58d0ab45
  - ng2-charts

Anexo 7.– Vita

Alejandro Viana Ríos es natural de Granada, donde estudia Ingeniería Técnica en informática de Gestión y el segundo ciclo de Ingeniería Informática. Termina sus estudios de la facultad en 2004, pero no se relaja y sigue formándose durante algunos años en programación, administración de sistemas y redes, seguridad e incluso el antiguo CAP (certificado de aptitud pedagógica).

Empieza a trabajar antes de terminar la carrera, alrededor de 2001, como personal laboral en una empresa pública dando soporte informático de todo tipo tanto a compañeros como a clientes. Además, por las tardes realiza trabajos de programación en PHP a medida y clases a grupos de informática básica.

En 2007 deja la empresa y es contratado como el único informático de supermercados DANI donde está poco menos de un año. Ante la llamada de una empresa internacional, la extinta Telvent, perteneciente al, entonces muy boyante, grupo Abengoa se marcha a Sevilla a comenzar una nueva etapa de gran proyección internacional.

Mientras está en nómina de Telvent, estudia las necesidades de los clientes, les realiza ofertas de hardware y, cuando las acepta, viaja a varios países desplegándo el hardware presupuestado.

A finales de 2010 se acaba la relación laboral y Alejandro se plantea un cambio en su vida, orientándola hacia la docencia en el sector público, comenzando a trabajar como profesor técnico de formación profesional de la especialidad de sistemas y aplicaciones informáticas, donde sigue hasta hoy.