

**CFGS DESARROLLO DE APLICACIONES  
MULTIPLATAFORMA**

**Proyecto Final de Ciclo**



**RWells Tattoo PWA**



**Autor:** Aníbal Santos Gómez

**Tutor:** Borja Arnau

**Fecha de entrega:** 07/05/2019

**Convocatoria:** 2018/2019

## Índice de contenidos

<b>1. Introducción.</b> .....	<b>2</b>
1.1. Motivación. ....	4
1.2. Objetivos propuestos.....	5
<b>2. Metodología utilizada.</b> .....	<b>7</b>
2.1. Scrum. ....	9
2.1.1. Herramientas de Scrum.....	10
2.1.2. Eventos de Scrum. ....	10
2.2 Kanban. ....	11
2.2.1. Tableros Kanban.....	12
2.2.2. Tarjetas kanban.....	13
2.2.3. Beneficios de kanban. ....	13
2.3. Kanban vs Scrum.....	14
2.4. Scrumban.....	14
2.4.1. Ciclo de vida del proyecto. ....	15
<b>3. Tecnologías y herramientas utilizadas en el proyecto.</b> .....	<b>21</b>
3.1. Lenguaje de programación, IDE, control de versiones, almacenamiento y herramientas. ....	22
3.2. Frameworks frontend: Vue JS, Nuxt JS, Vuetify, Sass.....	24
3.3. Frameworks backend: Node JS, Express, MongoDB, Mongoose, Passport JS, Webpack, Babel. ....	30
3.4. Administración de sistemas: Cloud Computing, Ubuntu, Nginx, Pm2, Cerbot. ....	35
<b>4. Estimación de recursos y planificación.</b> .....	<b>42</b>
4.1. Diagrama de Gantt.....	42
4.2. Scrumban.....	48
4.3. Presupuesto objetivo. ....	50
4.3.1. Costes directos de desarrollo. ....	50
4.3.2. Costes indirectos de desarrollo. ....	52
4.3.3. Costes de mantenimiento. ....	53
<b>5. Desarrollo del proyecto.</b> .....	<b>54</b>
5.1. Inicio.....	54
5.2. Análisis. ....	57
5.3. Diseño. ....	60
5.3.1. Administración sistemas.....	60
5.3.2. Creación del proyecto y control de versiones.....	62
5.3.3. Desarrollo del proyecto en local.....	64
<b>6. Despliegue y pruebas.</b> .....	<b>80</b>
6.1. Plan de prueba.....	80
6.2. Despliegue. ....	83
<b>7. Conclusiones.</b> .....	<b>84</b>
7.1. Objetivos alcanzados. ....	84
7.2. Conclusiones del trabajo. ....	85

<b>7.3. Vías futuras.....</b>	<b>86</b>
<b>8. Anexos. ....</b>	<b>87</b>
<b>8.1. Glosario. .....</b>	<b>87</b>
<b>8.2. Bibliografía. ....</b>	<b>95</b>
<b>8.3. Manual de instalación.....</b>	<b>98</b>
<b>8.2. Manual de usuario.....</b>	<b>101</b>

## 1. Introducción.

### “Abstract”

English:

The following software project aims to develop a Progressive Web App for managing the services of a small tattoo studio. This project seeks to ensure that the application can run on mobile devices and does not involve an effort for the customer to configure it. It also seeks to use the latest technologies in development, for this we have Javascript, Vue Js, Nuxt Js, Node, Git among others.

Also noteworthy as methodologies, the use of Scrum and Kanban for the organization of development tasks, which have allowed us to achieve our objectives in an iterative, ie consecutive, organized and incremental.

The final result of this project is a functional application that is in production for a particular client.

We can run it directly at the following link: [rwelltattoo.com](http://rwelltattoo.com), where it runs directly. To access the administration panel we will use the following user: [anibal@ilerna.com](mailto:anibal@ilerna.com) and the password: ilerna123.

To proceed with its local installation, we will run inside the directory "npm i" and "npm run dev".

Finally, I would like to thank my family and my girlfriend for their support and patience throughout this period in which I have begun this process of personal transformation towards the world of development.

Salamanca May 7, 2019.

Español:

El siguiente proyecto de software tiene como objetivo desarrollar una Progressive Web App para la gestión de los servicios de un pequeño estudio de tatuaje. Con este proyecto se busca conseguir que la aplicación pueda ejecutarse en dispositivos móviles y que no suponga un esfuerzo para el cliente configurarla. Además, se busca la utilización de últimas tecnologías en desarrollo, para ello contamos con Javascript, Vue Js, Nuxt Js, Node, Git entre otras.

Además, cabe destacar como metodologías, el uso de Scrum y Kanban para la organización de las tareas de desarrollo, que nos han permitido alcanzar nuestros objetivos de una forma iterativa, es decir, consecutiva, organizada e incremental.

El resultado final de este proyecto, es una aplicación funcional que se encuentra en producción para un cliente particular.

Podremos ejecutarla directamente en el siguiente enlace: [rwellstattoo.com](http://rwellstattoo.com), donde se ejecuta directamente. Para acceder al panel de administración utilizaremos el siguiente usuario: [anibal@ilerna.com](mailto:anibal@ilerna.com) y el password: [ilerna123](#).

Para proceder a su instalación local, ejecutaremos dentro del directorio "npm i" y "npm run dev".

Por último agradecer a mi familia y a mi novia el apoyo y la paciencia durante todo este periodo en el que he iniciado este proceso de transformación personal hacia el mundo de desarrollo.

Salamanca 7 de mayo de 2019.

## 1.1. Motivación.

En el presente documento se llevará a cabo la explicación del proceso de desarrollo de una aplicación informática que pretende dar solución a un problema simple de gestión de citas bajo la utilización de tecnologías modernas de desarrollo.

¿En qué consiste el desarrollo ágil? El desarrollo ágil es la habilidad de crear y responder al cambio, con el fin de tener éxito en un ambiente incierto.

En el contexto del desarrollo de una aplicación de software es un conjunto de prácticas y métodos basados en los valores y principios basados en los valores de Manifiesto Ágil. Existen diversas prácticas, cuyo objetivo es, mantener un código simple, pruebas seguidas y entregas funcionales cortas.

Bajo la premisa de la inmediatez en la que vivimos día tras día, es importante abordar los problemas de una manera eficiente y organizada. Cada cliente tiene una visión sobre la sociedad y la economía diferente, pero si hay algo en los que todos se asemejan es en evitar la dilación de los plazos para resolver un conflicto. Los conflictos sólo acarrean quebraderos de cabeza y pérdida de tiempo; lo que se traduce en costes.

En el mundo empresarial existen infinidad de problemas a resolver, y el objetivo siempre es el mismo, encontrar una solución en el menor tiempo posible, esto es, eficientemente.

Teniendo esto claro, el desarrollo del software nos facilita trabajos que de otra manera nos costarían horas. El ejemplo más evidente es la interacción con futuros clientes.

Un futuro cliente contacta con una empresa para que le preste un servicio, este servicio debe ser realizado en un plazo. El cliente necesita acomodar ese plazo a su rutina diaria.

Las rutinas y planes de ambas partes son diferentes en la mayor parte de los casos, pero ambas llegan a un acuerdo mediante el cual se realizará una prestación de servicios a cambio de una contraprestación económica.

El foco principal del problema, es el marco temporal de encuentro. En ocasiones puede ser un infierno manipular una agenda, y la comunicación con determinados clientes

puede ser insatisfactoria, generándonos un mal feedback, que acabará tarde o temprano traduciéndose en costes.

Esto plantea la siguiente cuestión ¿Por qué no reducir el factor humano en situaciones comunicativas que pueden generar conflicto, además de pérdidas de tiempo? Este es un punto importante, que se dividirá en dos aspectos:

- Acabar con un inicial conflicto en el acuerdo entre partes.
- Proporcionar inmediatez al usuario o cliente final.

Esto es sin duda alguna el santo grail que persiguen las empresas que trabajan de cara a un público **exigente**. Acabar con los conflictos comunicativos y acelerar el proceso de ingresos.

## 1.2. Objetivos propuestos.

Los objetivos a los cuales nos referiremos en este apartado tratarán de dar solución a la problemática anteriormente expuesta:

- Acabar con un inicial conflicto en el acuerdo entre partes.
- Proporcionar inmediatez al usuario o cliente final.
- Estructurar, gestionar y organizar la producción.
- Proporcionar visibilidad y destacar por encima de la media.
- Generar ingresos velozmente (atendiendo siempre al mercado oferta/demanda).

A continuación, se presenta el problema concreto para dar forma al contexto explicado en la motivación:

La empresa Not Sorry Tattoo, ante la abundante afluencia de clientes continua, tiene problemas para gestionar los servicios que allí se prestan. Los principales problemas a los que se enfrentan no son complejos, pero el personal que allí trabaja es limitado.

Algunos de estos problemas son:

- Gestionar las citas para tatuajes.
- Comunicación con los clientes.
- Gestión y control de los productos que se utilizan para llevar a cabo las tareas.

¿Cómo solucionaremos digitalmente este tipo de problemas que afectan a las pequeñas empresas? Los **objetivos** a resolver los dividiremos en generales y específicos:

- **Objetivos generales:**

- Crear una aplicación multiplataforma que gestione las citas para dicho negocio.
- La aplicación podrá crear listas de inventario.
- Ejecución tanto en dispositivos móviles como en ordenadores de escritorio, ya que el personal trabaja con ellos.
- Permitirá al cliente final comunicarse de asincrónicamente.
- Facilitará una entrada visual al trabajo artístico desarrollado por el estudio a nivel mundial.

- **Objetivos específicos:**

- La aplicación debe ser actualizable, fácil de mantener y posicionable.
- Debe poder ser escalable ante el supuesto de que se den peticiones masivas.
- Utilización de tecnologías modernas, y de diseño sencillo, intuitivo y con clara orientación al UX/UI.

## 2. Metodología utilizada.

La premisa de este proyecto es la utilización de metodologías ágiles. Como se viene explicando, el desarrollo ágil es la habilidad de crear y responder al cambio, con el fin de tener éxito en un ambiente incierto. Esto puede parecer un tanto ambiguo.

Para entender bien en qué consiste el desarrollo ágil atenderemos sus los **doce principios del Manifiesto Ágil**:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.

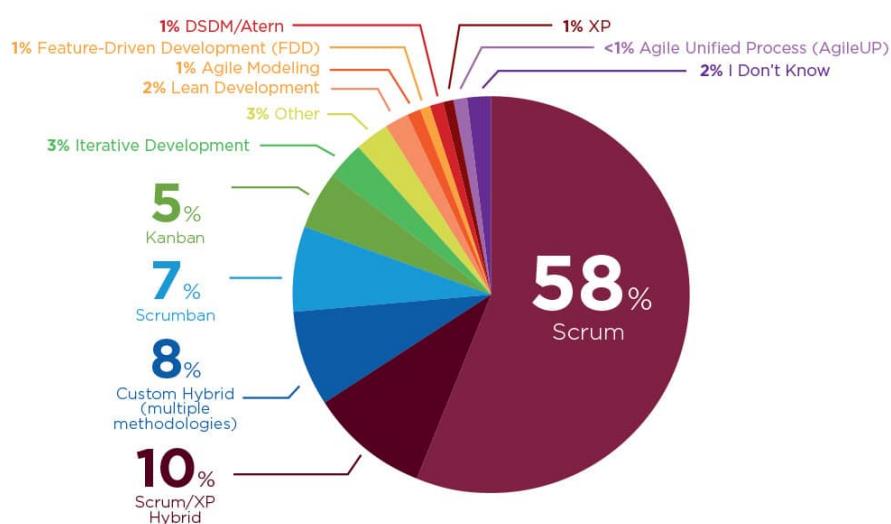
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Interiorizado el enfoque que va a tener nuestra metodología, se enumeran a continuación las **principales opciones** en desarrollo ágil:

1. Scrum.
2. eXtreme Programming.
3. Kanban.
4. Scrumban.
5. Lean.
6. Feature-Driven Development.
7. Test-Driven Development.

## *Agile Methodologies Used*

When asked what agile methodology is followed most closely, nearly 70% of respondents practice Scrum (58%) or Scrum/XP hybrid (10%).



SOURCE: VERSIONONE 10TH ANNUAL STATE OF AGILE™ REPORT  
© 2016 VERSIONONE INC. ALL RIGHTS RESERVED.

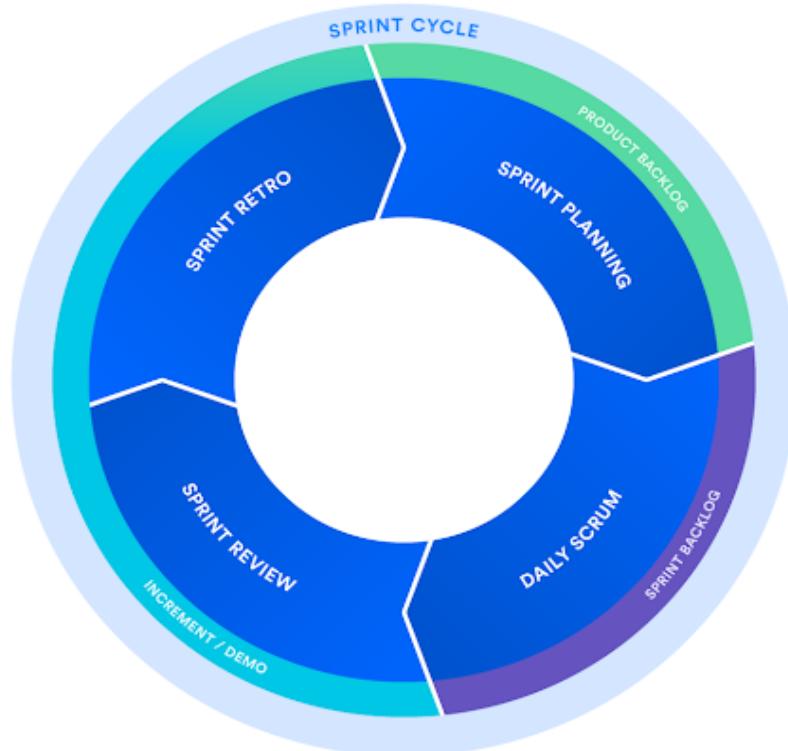
<https://boost.solutions/wp-content/uploads/2018/01/Agile-Methodologies-Used.jpg>

## 2.1. Scrum.

Como hemos visto las metodologías ágiles más representativas se basan en Scrum y/o combinados del mismo.

Scrum anima a los equipos a aprender a través de las experiencias, a autorganizarse mientras se trabaja en un problema y a reflexionar sobre sus victorias y derrotas para mejorar continuamente.

Son los equipos de desarrollo de software los que utilizan con mayor frecuencia, pero se puede aplicar a todo tipo de trabajo en equipo. Esta es una de las razones por las que es tan popular. Se basa en el aprendizaje continuo. Reconoce que el equipo no lo sabe todo al inicio de un proyecto y evolucionará a través de la experiencia.



<https://es.atlassian.com/agile/scrum>

### 2.1.1. Herramientas de Scrum.

1. Backlog del producto: es una lista dinámica de funciones, requisitos, mejoras y correcciones que actúa como la entrada para el backlog de sprint.
2. Backlog de sprint: es la lista de elementos, historias de usuario o correcciones de errores, seleccionadas por el equipo de desarrollo, para su implementación en el ciclo actual de sprint.
3. Objetivo del sprint, es el producto final utilizable de un sprint.

### 2.1.2. Eventos de Scrum.

- Organización del backlog: la lleva a cabo el propietario y su objetivo es, dirigir el producto y estar al tanto del mercado y los clientes.
- Planificación de sprint: todo el equipo de desarrollo planifica el trabajo que se va a realizar durante el sprint actual. Al final de la reunión de planificación, cada miembro debe tener claro qué se puede entregar en el sprint y cómo.
- Sprint: un sprint es el periodo real en que el equipo trabaja de forma conjunta para llegar al objetivo. La duración de un sprint suele ser de dos semanas, aunque algunos equipos pueden necesitar menos o más tiempo para la entrega del objetivo.

Establecido un determinado intervalo de tiempo para un sprint, debe seguir siendo coherente durante todo el periodo de desarrollo.

- Reunión rápida: reunión diaria de muy corta duración que tiene lugar siempre a la misma hora y en el mismo sitio. Es el momento de expresar cualquier inquietud acerca del cumplimiento del objetivo del sprint. Existen tres preguntas clave:
  - ¿Qué hice ayer?
  - ¿Qué tengo planificado para hoy?
  - ¿Hay algún obstáculo?

- Revisión de sprint: al final del sprint, el equipo se reúne en una sesión informal para ver una demostración del objetivo.
- Reflexión acerca del sprint: es donde el equipo se reúne para documentar y analizar qué ha funcionado y qué no ha funcionado en un sprint, un proyecto, en las personas o en las herramientas.

En este proyecto haremos uso de las partes que más nos interesan de Scrum y que se puedan adecuar a nuestro caso concreto. Evidentemente no tendremos reunión con el equipo, pero si tendremos reuniones con el cliente para definir el backlog, realizaremos sprints con el fin de tener un objetivo finalizado.

Esta es la esencia del Scrum, organizar el trabajo bajo un marco, donde no sucede el estancamiento bajo un problema. Si la solución a un problema no funciona se desecha y se busca otra.

Existen otras opciones, las cuales no valoraremos excepto una, que se ha tenido en cuenta a la hora de plantear un cierto tipo de metodología, y esa es **Kanban**.

## 2.2 Kanban.

En kanban los elementos de trabajo se representan visualmente en un tablero, lo que permite a los miembros del equipo ver el estado de cada uno en cualquier momento.

Nace del proceso de optimización de los procesos de ingeniería por parte de Toyota en los años cuarenta. El objetivo era alinear mejor sus niveles de inventario con el consumo real de materiales. Los trabajadores pasaban una tarjeta entre equipos. Cuando se vaciaba una ubicación de materiales en la línea de producción, se pasaba una tarjeta al almacén describiendo el material que se necesitaba.

El almacén tendría una nueva ubicación de este material en espera, que enviaría a la planta de producción y, a su vez, enviaría su propio kanban al proveedor. El proveedor también tendría una ubicación de este material en particular en espera, que enviaría al almacén.

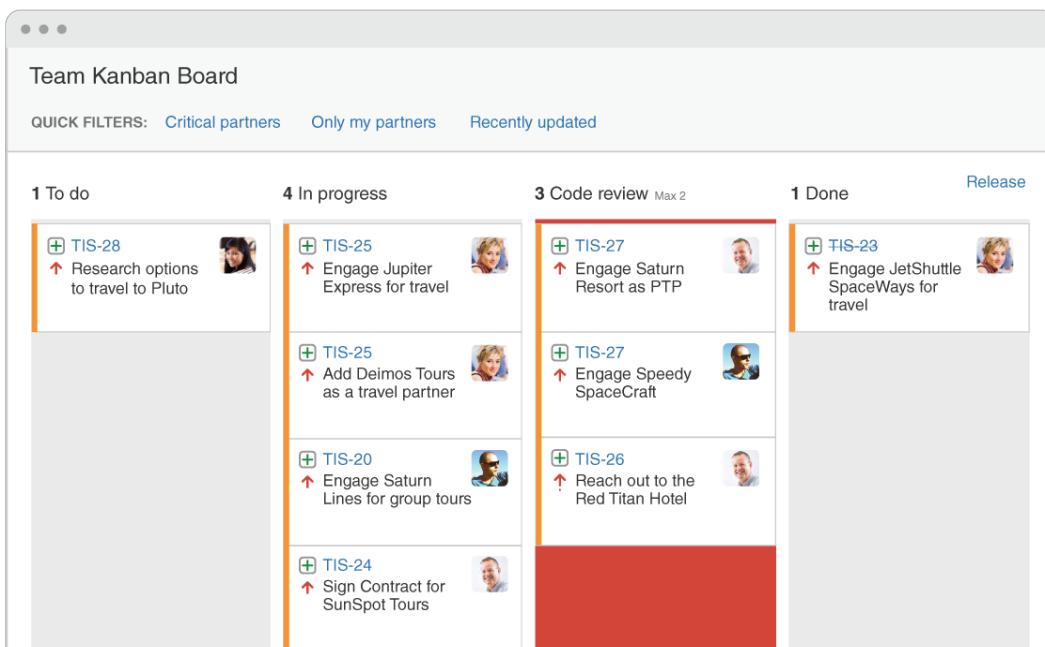
Los equipos de desarrollo de software han obtenido resultados muy positivos con esta práctica.

### 2.2.1. Tableros Kanban.

El trabajo del equipo gira en torno a un tablero, que se usa para visualizar las tareas y optimizar el flujo de trabajo. Su función es garantizar que el workflow se unifique y que se identifiquen y resuelvan inmediatamente todos los factores que lo bloqueen y de los que dependan. El tablero tiene un **workflow** de tres pasos:

- Por hacer (pendiente).
- En curso.
- Hecho.

Esta metodología se basa en transparencia del trabajo. El tablero debería considerarse la única fuente fiable sobre el trabajo del equipo.



<https://es.atlassian.com/agile/kanban>

## 2.2.2. Tarjetas kanban.

Cada elemento de trabajo se representa en una tarjeta distinta del tablero. El objetivo principal es que el equipo realice el seguimiento del progreso del trabajo visualmente.

Las tarjetas presentan información vital sobre ese elemento de trabajo concreto y proporcionan una visibilidad a todo el equipo sobre quién está a cargo de ese elemento de trabajo.

## 2.2.3. Beneficios de kanban.

- *El equipo se centra en el trabajo que está en curso.* Una vez que el equipo completa una tarea, retira la siguiente de la parte de trabajo atrasado.
- *Ciclos de tiempo más cortos,* al optimizar la duración del ciclo, el equipo puede pronosticar con confianza la entrega del trabajo futuro. Es responsabilidad de todo el equipo asegurar que el trabajo progresá de forma fluida.
- *Menos cuellos de botella,* una propuesta clave de kanban es limitar la cantidad de trabajo en curso. Un límite bajo anima al equipo a prestar especial atención a los problemas en el estado de revisión, y a revisar el trabajo de otros antes de plantear sus propias revisiones de código.
- *Entrega continua,* es la práctica de entregar el trabajo a los clientes con frecuencia, esto junto con kanban se complementan porque ambas técnicas se centran en la entrega de valor justo a tiempo.

Cuanto más rápido un equipo pueda llevar la innovación al mercado, más competitivo será su producto en el mercado. Y los equipos de kanban se centran exactamente en eso: optimizar el flujo de trabajo hacia los clientes.

## 2.3. Kanban vs Scrum.

- Las tareas (kanban) tienen un flujo continuo, los sprints tienen una longitud fija periódica.
- En la entrega, las tareas se entregan de forma continua o según el equipo. En cambio, al final de un sprint la entrega la autoriza el propietario del producto.
- En el sprint ganamos en velocidad y en la tarea en el tiempo del ciclo.
- En el sprint evitaremos la previsión priorizando el aprendizaje, mientras que en la tarea los cambios pueden suceder en cualquier momento.

## 2.4. Scrumban.

En un equipo se toman los sprints de longitud fija de Scrum, combinando la atención a los límites del trabajo en curso y el tiempo del ciclo de kanban.

En definitiva, combinamos un proceso de aprendizaje rápido, con un objetivo claro y definido en un tiempo (sprint), pero limitado a un número corto de tareas para focalizar nuestra atención en una parte. Sabiendo siempre de antemano las tareas pendientes y las finalizadas, es decir, en un flujo o ciclo continuo.

Este método es utilizado en numerosos programas de crecimiento empresarial, así como en startups que necesitan avanzar en su producto rápidamente en un mercado inestable o simplemente cambiante.

Esta metodología, es la elegida para desarrollar este proyecto, ya que, aun prescindiendo de un equipo, nos permitirá seguir un ciclo constante de desarrollo, fijar las etapas del mismo e ir aprendiendo todo lo que podamos con un objetivo, dar funcionalidad y salida a la problemática que nos plantea Not Sorry Tattoo.

## 2.4.1. Ciclo de vida del proyecto.

Las principales fases o etapas dentro del ciclo de vida, en un modelo clásico de desarrollo en cascada, de un producto son:

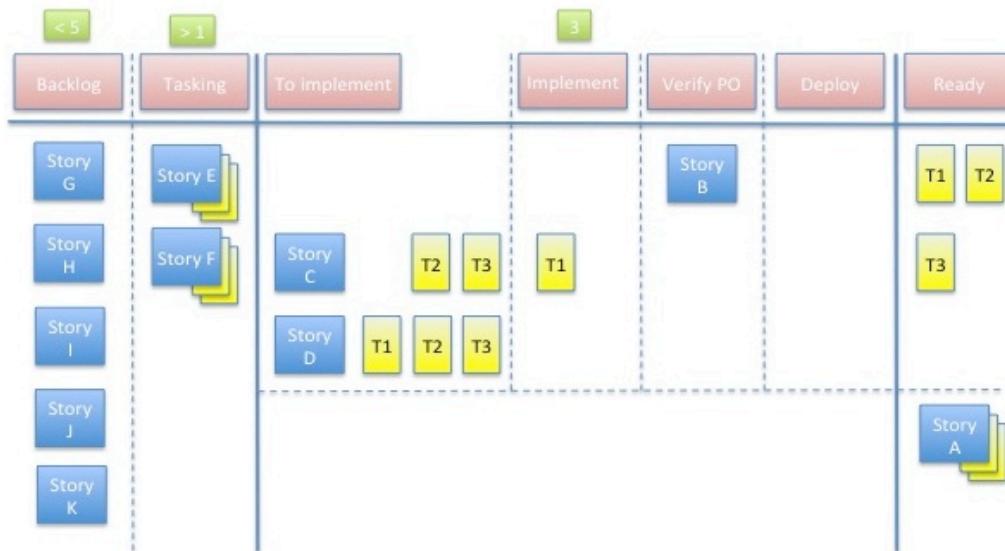
- Definición de necesidades.
- Análisis de requisitos.
- Diseño.
- Codificación.
- Pruebas
- Validación.
- Mantenimiento.

Nuestro ciclo de vida se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto.

El desarrollo iterativo recomienda la construcción de secciones reducidas de software que irán ganando en tamaño para facilitar así la detección de problemas de importancia antes de que sea demasiado tarde. Los procesos iterativos pueden ayudar a desvelar metas del diseño en el caso de clientes que no saben cómo definir lo que quieren.

TO DO			NEXT	DOING 3 / 2	FOR APPROVAL	SPRINT DONE
STORIES	BACKLOG	THIS SPRINT				
+ add task <b>prepare the invoices</b>	+ add task <b>Rework the main page layout</b>	+ add task <b>[high]</b>	+ add task <b>prepare invoice 669</b>	+ add task <b>task 34 C</b>	+ add task <b>RU promo codes M Team</b>	+ add task <b>prepare the invoice 4758</b>
<b>prepare the invoice 45</b>	<b>prepare the invoices</b>	<b>T13</b>	<b>prepare the invoice 4758</b>	<b>T2</b>	<b>task 34 B</b>	<b>RU use case 862 re-do</b>
<b>prepare the presentation for WCY</b>	<b>prepare the invoice 589</b>	<b>give away the marketing infos to DC</b>	<b>prepare a speech for the presentation in DC on December 30 2015</b>	<b>T13</b>	<b>use case 862 re-do</b>	<b>T2 as a user I can edit all of my details</b>
<b>get all archived documents for the tax office</b>	<b>prepare the invoices</b>	<b>Task 1</b>	<b>promo codes MTeam</b>		<b>user case 56 - check for updates</b>	<b>user case 56</b>
<b>prepare the invoices for ZDBank</b>	<b>promo codes MTeam</b>	<b>redo the standign orders for 2016</b>	<b>prepare the invoice 223</b>		<b>Contact Marisha</b>	<b>rr</b>
<b>use case 56</b>	<b>scan the server for malware</b>	<b>hire an office assistant</b>	<b>34 scenario</b>		<b>as a user I can edit all of my details</b>	<b>@AS libit</b>
<b>german text bug fix</b>	<b>use case 56</b>		<b>find the Cycle Time script</b>		<b>rr</b>	<b>set the domain name to new</b>
	<b>as a user I can edit all of my details</b>		<b>prepare the invoice 4758</b>		<b>go go go</b>	<b>get the documents for the tax office</b>
	<b>give away the marketing infos to DC</b>		<b>give away the marketing infos to DC</b>		<b>as a user I can edit all of my details in Prom View</b>	<b>check order 89</b>
	<b>use case 56</b>					<b>give away the marketing infos to DC</b>
						<b>use case 862 re-do</b>
						<b>set the subdomain name to sub_</b>

<https://kanbantool.com/es/scrumban-scrum-y-kanban>



<https://xebia.com/blog/scrumban/>

Se plantea a continuación cómo será el ciclo de vida del software o desarrollo del mismo:

1. Backlog.
2. Sprint.
  - a. Tasking.
3. WIP.
4. Done.
5. Verify.
6. Deploy.
7. Ready.

## 1. Backlog.

Es el conjunto de todos los requisitos de producto, el cual contiene descripciones genéricas de funcionalidades deseables. Los **requisitos** de producto son:

- A. **Creación** de aplicación multiplataforma que gestione las citas del estudio y que permita crear listas de inventario. Estos son los dos requisitos imprescindibles de funcionalidad del producto.
- B. Es muy importante la **adaptabilidad** a dispositivos móviles y que el cliente se comunique solo de forma asíncrona con el estudio.

C. En menor medida y como **complementos** se hace referencia a la importancia de un portfolio con posicionable fácilmente con una interfaz limpia y sencilla orientada a las tendencias de UX/UI (User Experience/User Interface).

Esto se traduce en el siguiente backlog:

I. Administración:

- Login de acceso a un único usuario, sin registro.
- Gestión de citas para el administrador.
- Gestión de stock con pequeñas “shopping lists”.
- Adaptabilidad a dispositivos móviles. Mobile First.

II. Landing page:

- Creación de un portfolio o landing page.
- Formulario de contacto.
- Válido para posicionamiento SEO.

## 2. Sprint.

En esta parte planteamos los sprints para la realización de la aplicación. Los organizaremos en función de la facilidad y la preferencia, para no estancarnos en problemas y dilatar el tiempo en exceso.

Dentro de cada sprint realizaremos un tasking que nos dará una pauta de flujo continuo para continuar con las siguientes fases. El orden del tasking será indiferente, se realizarán las tareas de forma controlada y en pequeños grupos.

**Sprints y tasks:**

- Planteamiento del diseño UX/UI.
- Arquitectura y creación de las bases del proyecto. **Tasks:**
  - Selección de frameworks, lenguajes y herramientas.
  - Selección de proveedores cloud.
  - Creación de repositorio.
  - Creación del proyecto.
  - Creación y configuración de la máquina servidor.
  - Creación y configuración de la base de datos.
  - Configuración de los certificados de seguridad HTTPS.
  - Configuración de los sistemas de despliegue.
- Configuración de la arquitectura.
  - Creación de layouts.
  - Creación de rutas.
  - Creación de vistas.
- Creación y configuración del sistema de autentificación.
- Creación del sistema de API Rest. **Tasks:**
  - Obtención de datos.
  - Registro de datos.
  - Actualización de datos.
  - Eliminación de datos.
- Creación y configuración del store local del navegador.
  - Creación de un store local.
  - Configuración del centralizado de datos.
  - Configuración de peticiones API para registro en base de datos.
- Creación del componente Inventario. **Tasks:**
  - Creación de los componentes.
  - Creación, obtención, modificación y borrado de los datos.
  - Guardado de datos.
- Creación del componente calendario. **Tasks:**
  - Creación de los componentes.
  - Creación, obtención, modificación y borrado de los datos.
  - Representación de los datos actualizados.
- Finalización estética y testeo en dispositivos móviles y de escritorio.

### 3. WIP.

Se pasa a abordar cada sprint esperando un resultado final de entrega. Dentro de cada sprint las tareas se realizarán por unidades, centrándonos en que se finalicen correctamente.

Este será el momento de desarrollo del código y la funcionalidad del producto. Se afronta de una manera segmentada y organizada, sentando unas bases que irán desde la construcción de unos cimientos, como es la arquitectura del proyecto y su despliegue hasta la creación de cada componente y su relación con el entorno y el acceso a datos.

### 4. Done.

Acabada la tarea correspondiente, la trasladaremos a realizada. Esto nos indica que una parte del sprint estará finalizada. Así procederemos sucesivamente hasta acabar con las tareas correspondientes de cada sprint.

Antes de proceder a la parte de verificación es muy importante que probemos la funcionalidad de cada tarea. Esto no quiere decir que verifiquemos su correcto funcionamiento de forma exhaustiva, el objetivo de esta pequeña prueba es demostrar que la tarea se adapta al entorno y no causa problemas en su implementación técnica.

### 5. Verify.

En este momento del sprint, verificaremos que las tareas realizadas funcionan en su conjunto y de forma individual. Finalizado el sprint verificaremos también el resultado del mismo, es decir, el conjunto de las tareas que lo conforman.

El cliente deberá probar cada sprint finalizado, esto se convierte en una obligación ya que el usuario final comprobará los fallos que el desarrollador pueda no tener en cuenta a la hora de programar.

Si este resultado es negativo, el planteamiento deberá reconsiderarse, ya que en una primera instancia buscamos que nuestro cliente entienda el uso de la aplicación.

Entendemos que una aplicación es intuitiva cuando el diseño del producto y su funcionalidad se comprenden en un uso superficial de un usuario medio, con conocimientos tecnológicos.

## 6. Deploy.

Procederemos a desplegar, es decir, muere el sprint y pasamos al siguiente. En este momento se entrega una parte funcional al cliente. Es una parte estable y consumible, por lo que deberá estar exenta de fallos.

El sistema de despliegue en nuestro caso se realizará desde nuestro servidor de producción. El desarrollo pertinente se hace siempre en un entorno local y cada cambio se registrará en un repositorio. Utilizaremos Git como control de versiones, ya que podemos desarrollar offline. Finalizadas las ramas de desarrollo, estas se mezclarán con la rama master. Entonces, podremos publicar los cambios nuevos en nuestro servidor de producción para que estén accesibles al cliente.

## 7. Ready.

Cada sprint se encuentra en producción integrado en el conjunto del producto. Es la parte de desarrollo estable, donde los bugs previos han sido corregidos, testeados y la funcionalidad es la planeada durante el proceso de desarrollo.

### 3. Tecnologías y herramientas utilizadas en el proyecto.

La elección de las tecnologías no es un tema trivial, debemos atender a una serie de factores condicionantes para la selección de las mismas:

- Tiempos de desarrollo y de entrega.
- Innovación.
- Desarrollo multidispositivo.
- Simplicidad y sencillez.

**Los tiempos de desarrollo y de entrega**, son diferentes en cada proyecto, y es un factor condicionante de primer nivel. El cliente quiere una aplicación “para ayer”, es decir, lo más rápido posible. Teniendo en cuenta la elección de metodologías ágiles deberemos aprovechar las mismas para centrarnos en desarrollar lo más rápido posible un producto funcional.

**La Innovación**, atiende sobre todo a la utilización de tecnologías modernas y adaptativas a todo tipo de entorno. Nuestra aplicación será mantenible en el tiempo en proporción a lo actual que sean las tecnologías que utilicemos.

El cliente tiene una necesidad de que el producto sea **multidispositivo**, es decir, que su utilización será en todo tipo de dispositivos móviles, así como de escritorio.

**La simplicidad y sencillez**, son dos factores condicionantes de venta del producto, esto es, un producto que se entiende por sí solo, que no necesita apenas de manual técnico y cuya interfaz guíe de forma amigable al usuario final.

En el mercado existen infinitas opciones para poder desarrollar un producto sencillo como este, desde lenguajes populares como C#, Java, Javascript, Python o IDEs como Netbeans, Eclipse, Visual Studio Code, Visual Studio, IntelliJ IDEA, WebStorm, PhpStorm, etc...

### 3.1. Lenguaje de programación, IDE, control de versiones, almacenamiento y herramientas.

Javascript es un lenguaje versátil, pero que puede llegar a ser engorroso si tenemos en cuenta el desarrollo ágil. Desarrollar un proyecto en Javascript puro puede ser una tarea infernal. Es por esto, que a continuación se detalla la elección de frameworks y otro tipo de herramientas que nos ayudarán a que el desarrollo de nuestra aplicación sea mucho más llevadero y mantenible a lo largo del tiempo.



- **Javascript como lenguaje**, las últimas estadísticas hablan de Javascript como el lenguaje de programación más utilizado del mundo, por delante de lenguajes con un largo recorrido como Java o C++. La elección está más que justificada, existe una comunidad amplia, y con Javascript podemos programar un modelo vista controlador completo, desde el lado del cliente, así como desde el lado del servidor.



- **WebStorm como IDE**, es cierto que existen alternativas open source gratuitas muy potentes para realizar proyectos como Visual Studio Code o Sublime, IDEs como Eclipse, Netbeans, etc... WebStorm nos otorga facilidades de desarrollo en cuestión de integración de un terminal, control de versiones con Git, fácil estructuración de proyectos, actualizaciones de su software y otro tipo de herramientas o plugins como Linters para el formateo del código, que nos ayudarán a que nuestro código esté limpio.



- **Control de versiones: Git.** Es un software de control de versiones diseñado por Linus Torvalds. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos. El control de versiones nos permitirá ver la evolución de nuestro proyecto. Crear ramas de desarrollo, tener en el futuro un repositorio cooperativo de equipo y sincronizar los proyectos en diferentes equipos.

## Bitbucket

Como servicio de almacenamiento basado en la web existen muchas alternativas, GitHub, Gitlab, Bitbucket..., se ha elegido Bitbucket por la posibilidad de generar repositorios privados, ya que este producto será un producto comercial y no estará disponible su código fuente para que otras personas puedan acceder a él.



- **Gestor de paquetes: NPM (Node Package Manager).** Es el gestor de paquetes de Node JS, así como PIP es el gestor para Python. Gracias a él, tenemos casi cualquier librería disponible a tan solo una línea de comando de distancia, permitiéndonos utilizarla en cuestión de segundos. Podemos encontrar mediante su buscador las librerías que necesitemos para nuestros proyectos.  
<https://www.npmjs.com/>



- **Postman**, es una herramienta para desarrolladores que nos permite realizar peticiones HTTP a cualquier API. Nos ayudará en la ardua tarea de probar el router, y las peticiones API REST (get, post, put, delete).

### 3.2. Frameworks frontend: Vue JS, Nuxt JS, Vuetify, Sass.



- **Vue JS**. Es un marco progresivo para construir interfaces de usuario. A diferencia de otros marcos monolíticos, está diseñado desde cero para ser adoptado de forma incremental. La biblioteca principal se centra únicamente en la capa de vista, y es fácil de recoger e integrar con otras bibliotecas o proyectos existentes.

Los puntos fuertes de este framework son:

- **Progresividad**, está modularizado en librerías separadas que permiten ir añadiendo funcionalidad en el momento que las vayamos necesitando.
- **Funcionalidades modernas e intuitivas**, lo principal de este framework es que trata el dato como centro de todo, bajo un patrón MVVM.
- **El sistema de componentes es reactivo**, la comunicación se realiza por medio de eventos asíncronos. Esto se debe a que el modelo de datos del componente es envuelto por getters y setters encargados de gestionar estas reacciones.

- **Ecosistema variado que cubre lo necesario.** Para llevar a cabo la depuración utilizaremos la extensión para Chrome, **Vue Tools**, y los IDEs modernos como WebStorm integran ya plugins que facilitan el formateo del.
- **Comunidad activa**, es un proyecto gestionado, desarrollado, evolucionado y planteado por y para la comunidad.
- **El código de un componente se encuentra en un único fichero**, su estructura se divide en tres partes diferenciadas, **template** (HTML), **script** (Javascript) y **style** (CSS). Esto es lo que se denomina Single File Component.
- **El rendimiento**, gana en velocidad y rendimiento a sus competidores ya que la biblioteca es más ligera.



- **Nuxt JS.** Es un framework que nos permite crear aplicaciones universales utilizando Vue JS. Se enfoca en renderizar y compilar tanto del lado del cliente como del lado del servidor y determinar cuándo corresponde realizar cada tipo de renderización.

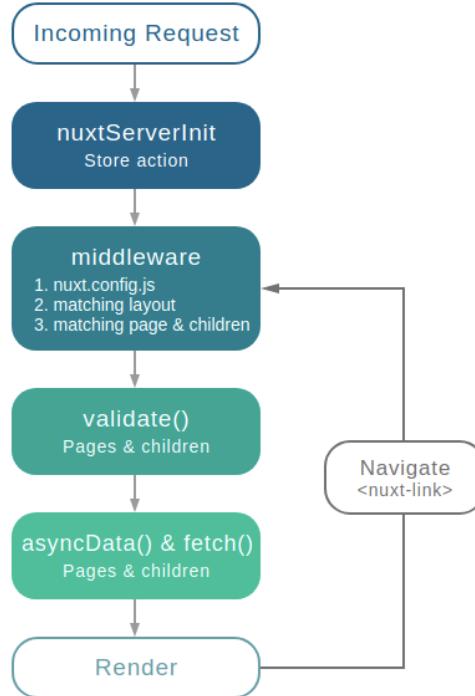
Los **puntos principales** son:

- **Renderizado de la interfaz de usuario** y la abstracción de la distribución cliente/servidor.
- Crear un **marco de trabajo flexible** para que pueda ser utilizado como base principal del proyecto.
- **Preselección de toda la configuración necesaria.**
- Crear **Single Page Application** o **Universal Application**.

- Nuxt JS incluye:
  - **Vue 2**, framework progresivo para generar SPA.
  - **Vue Router**, servicio de routing dinámico.
  - **Vuex (store)**, gestión de estados centralizados
  - **Vue Server Render**, ejecución de código en servidor y cliente.
  - **Vue-meta**, gestión de meta información de la aplicación.
- Escritura de código en **ficheros Vue**.
- **Separación automática del código**, que consiste en dividir el código en varios paquetes, que luego pueden cargarse bajo demanda.
- **Renderizado en servidor**, las vistas de la aplicación serán renderizadas en el servidor. Todos los cambios dinámicos son interceptados por Vue en cliente.
- Nuxt configurará el **routing** de la aplicación dependiendo de la estructura de carpetas que hayamos creado.
- Capacidad de **servir ficheros estáticos** desde servidor.
- **Transpilación de ES6/ES7**, posibilidad de transpilar código escrito en un estándar moderno a código que el navegador pueda entender.
- Manejo de los **elementos de la etiqueta <head>**, nos permitirá cambiar la meta información desde cualquier componente Vue.
- **Carga de aquellos módulos donde hemos realizado cambios**, evitando así cargar toda la página para mostrar los cambios de un único componente.
- **HTTP/2** envío de cabeceras, el servidor nos enviará los assets antes de que el navegador pregunte por ellos.
- **Extensión de la funcionalidad de Nuxt** con arquitecturas modulares.

## ¿Cómo funciona?

1. Un **usuario realiza una petición** de una ruta determinada al servidor.
2. El servidor ejecuta la **acción nuxtServerInit** del store principal si la tiene implementada. Esta acción nos permite cargar datos iniciales (prefetching de datos globales).
3. Se **ejecutan todos aquellos middlewares** que se encuentren en el fichero de configuración **nuxt.config.js** y los relacionados con el **layout, la página raíz y las páginas hijas** coincidentes que se hayan implementado.
4. Si existe un **validador**, se ejecuta. Si se resuelve con un true se sigue el proceso, si no se devuelve un error 404.
5. Se **obtienen aquellos datos de la página** para que sean renderizados.
6. Se **renderiza en servidor** y se **sirve al usuario**.
7. Si el usuario navega por la aplicación **hacia otra ruta, se repite el ciclo**.



**Ventajas:**

- **Mejor SEO**, los motores de búsqueda verán directamente la página completamente renderizada.
- **Tiempo de carga de contenido más rápido**, la generación de HTML en el servidor no necesita esperar hasta que se haya descargado y ejecutado todo el JavaScript para que se muestre, esto se traduce en una mejor experiencia del usuario.

**Desventajas:**

- **Restricciones de desarrollo**. El código específico del navegador solo se puede usar dentro de ciertas etapas del ciclo de vida del componente.
- **Requisitos de instalación y despliegue más complicados**. Una aplicación procesada por servidor requiere un entorno donde se puede ejecutar un servidor Node JS.
- **Más carga en el servidor**. Renderizar una aplicación completa en Node JS va a requerir más CPU que solo servir archivos estáticos.

En nuestro proyecto necesitamos Server Side Rendering, porque necesitaremos posicionar en un futuro nuestro portfolio. Se resumen a continuación los **tipos de renderizados en Nuxt JS**:

- **Server Rendered (Universal SSR)**, usamos Nuxt JS como framework para manejar todo el renderizado UI de nuestro proyecto.
- **Single Page Applications (SPA)**, podemos habilitar la opción de SPA y usar Nuxt como si trabajáramos con Vue.
- **Static Generated (Pre Rendering)**, nos permite generar los estáticos de nuestro proyecto y poder alojarlos en algún CDN.



- **Vuetify.** Es un framework de componentes semánticos para Vue JS. Su objetivo es proporcionar componentes limpios, semánticos y reutilizables que facilitan la construcción de aplicaciones.

Con esta herramienta construiremos nuestro proyecto con Vue JS, Material Design y una biblioteca masiva de componentes.

Los componentes se encuentran según las especificaciones de diseño de Material Design de Google, presentan un diseño semántico fácil de recordar.

Es compatible con todos los navegadores modernos, incluidos IE11 y Safari 9+ (que utilizan polyfills). Desde el móvil hasta el portátil o el escritorio, estaremos seguros de que el proyecto funcionará como se espera.

Se basa en el sistema “grid” para la ordenación del layout de la página. Está formado por una extensa librería de componentes que incluye desde elementos de formulario sencillos como botones, combobox, inputs, slides, a componentes más avanzados típicos en aplicaciones Android como cards o snack bars.

Todos los componentes pueden configurarse con distintas opciones para cambiar por completo su estética y comportamiento. Así, tenemos funciones para ordenación de componentes, estilos de texto, colores, iconos, sombras, diálogos, animaciones o soporte touch. Con Vuetify aceleraremos la implantación de estilos visuales de nuestra aplicación de una forma visualmente moderna.



- **Sass.** Syntactically Awesome Stylesheets es un metalenguaje de Hojas de Estilo en Cascada (CSS). Es un lenguaje de script que es traducido a CSS. Incluye las siguientes características:
  - Compatible 100% con **CSS3**.
  - Permite el **uso de variables**, anidamiento de estilos y mixins.
  - **Funciones** para manipular con facilidad colores y otros valores.
  - **Elementos de programación** como directivas de control y las librerías.
  - Genera **archivos CSS bien formateados**.

Sass nos otorgará a nuestro proyecto una arquitectura de estilos mantenible en el tiempo y, sobre todo, centralizada.

### 3.3. Frameworks backend: Node JS, Express, MongoDB, Mongoose, Passport JS, Webpack, Babel.



- **Node JS.** Es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

Es una Máquina Virtual ultra rápida y de gran calidad. La ejecución en tiempo real está pensada para ejecutarse directamente en una computadora o sistema operativo de servidor. El entorno omite las APIs de JavaScript específicas del explorador web

y añade soporte para APIs de sistema operativo más tradicionales que incluyen HTTP y bibliotecas de sistemas de ficheros.

Node soporta protocolos TCP, DNS y HTTP.

Es capaz de mantener muchas conexiones abiertas y esperando. En Apache, por ejemplo, el parámetro MaxClients por defecto es 256. Este valor puede ser aumentado para servir contenido estático, sin embargo, si se sirven aplicaciones web dinámicas en PHP u cualquier otro lenguaje, es probable que al poner un valor alto el servidor se quede bloqueado ante muchas conexiones, esto depende del trabajo que realiza la aplicación web en el lado del servidor y el hardware que posea.

En resumen, sus ventajas son:

- **Gran rendimiento.** Diseñado para optimizar el rendimiento y la escalabilidad en aplicaciones web.
- **El código está escrito en Javascript,** se pierde menos en comutaciones de contexto entre lenguajes.
- **El gestor de paquetes (NPM)** proporciona acceso a miles de paquetes reutilizables.
- Es **portable**, con versiones que funcionan en Microsoft Windows, OS X, Linux. Está bien soportado por muchos de los proveedores de hosting.
- Tiene una **comunidad de desarrolladores** de terceros muy activa.

# Express

- **Express.** Es un framework para Node JS que sirve para ayudarnos a crear aplicaciones web en menos tiempo, nos otorga funcionalidades como el enrutamiento, opciones para gestionar sesiones y cookies.

Está diseñado para construir aplicaciones web y APIs. Está pensado para ser el marco de trabajo estándar para NodeJS. Es un framework mínimo con muchas opciones de configuración, así como plugins.

Suele ser uno de los componentes del Full Stack web moderno, combinado con MongoDB en bases de datos, y Angular, React o Vue JS en la parte de cliente o frontend.



- **MongoDB.** Es el sistema de bases de datos NOSQL más utilizado para aplicaciones modernas. En lugar de guardar los datos en tablas, como en las bases de datos relacionales, MongoDB guarda los datos en formato BSON (especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Lo más importante de MongoDB, es que no necesitamos seguir un esquema. Los documentos de una misma colección pueden tener esquemas diferentes. Esto es algo impensable en las bases de datos relacionales, mientras que en MongoDB se puede redundar el dato.

Lo utilizaremos en este proyecto para construir los modelos de CREATE, READ, UPDATE y DELETE.



- **Mongoose.** Es un framework ODM (Object Data Modeling) de Javascript utilizado en una aplicación Node JS con una base de datos MongoDB. Realizaremos un modelado de los esquemas que utilizaremos en la base de datos de MongoDB.

Nos permite escribir todas y cada una de las consultas en Javascript a través de un modelo que contiene un esquema. Este y la base de datos se requerirá desde el servidor de Express en Node JS.

El esquema es sencillo de utilizar, mapea a una colección de MongoDB y define el molde de los documentos que se encuentran dentro de la colección. Para utilizar el esquema convertiremos cada uno en un modelo para trabajar con él.

Los tipos de esquemas en Mongoose son:

- String
- Number
- Date
- Buffer
- Boolean
- Mixed
- ObjectId
- Array

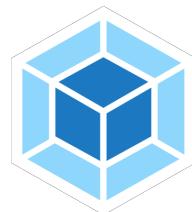


- **Passport JS.** Es un framework para Node JS, que nos ayudará a gestionar la autenticación de usuarios, trabaja con Express JS y la integración en este caso se realizará con un módulo de Auth que nos facilita Nuxt JS. Cuenta con más de treinta plugins para realizar OpenID, OAuth o login con password local (será el que realicemos en este caso).

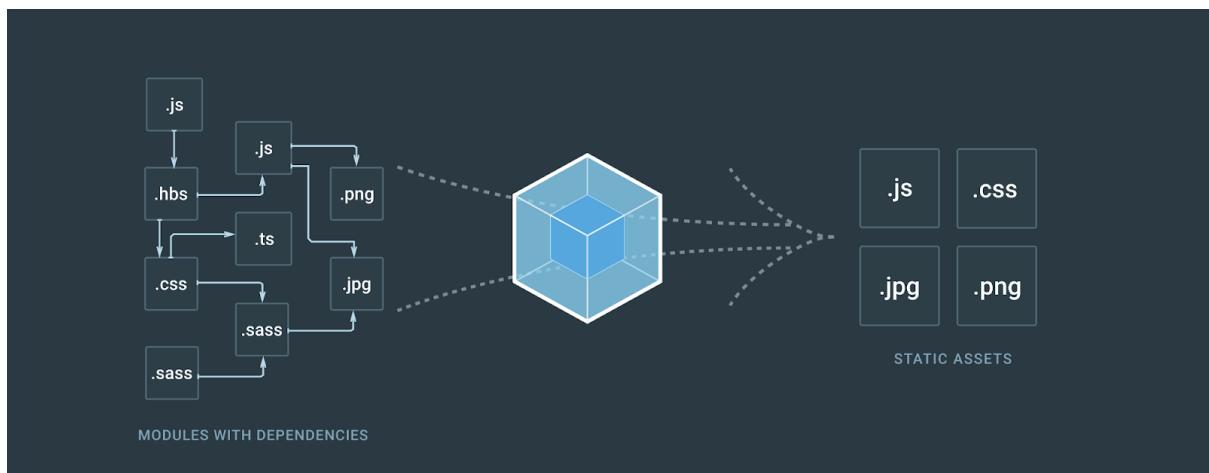
El funcionamiento de la API no es demasiado complejo, se realiza una petición de autenticación por medio de la librería y ésta nos proporciona los componentes para controlar qué sucede cuando la autenticación es un éxito o devuelve un error. Sus características principales son:

- Más de **30 estrategias** de autenticación.
- Single sign-on utilizando **OpenID y OAuth**.
- Manejo de la **autenticación para controlar el éxito** o el fallo del proceso.

- Soporte de **sesiones persistentes**.
- Scope dinámico y **distintos permisos** de aplicación.
- Podemos implementar nuestras **propias estrategias** de autenticación.



- **Webpack.** Es un empaquetador de módulos. Separa el código en módulos que luego se utilizan como dependencias en otros módulos. Gestiona dichas dependencias, además de poder concatenar código, minimizarlo, verificación de buenas prácticas mediante linters, carga bajo demanda de módulos (a la carta).



También podemos utilizar como módulos los archivos HTML y CSS además de Javascript. Las principales ventajas de Webpack son las siguientes:

- **Eliminación de recursos no utilizados.**
- **Control absoluto sobre procesado de los recursos.**
- **Modularización para Javascript.**
- **Es muy rápido.**
- **Despliegues confiables.**
- **Velocidad de desarrollo.**

# BABEL

- **Babel JS.** Es un transcompilador que nos permite convertir nuestro código de Javascript ES6 en código de ES5.

Esta característica se está convirtiendo en algo muy relevante para una gran parte de los desarrolladores ya que las nuevas características de ES6 hacen deseable trabajar con el lenguaje lo antes posible.

Por desgracia no todos los navegadores soportan ES6 y necesitan transpilar este nuevo estándar de escribir Javascript en otras versiones más antiguas del mismo. Pues bien, Babel JS nos ayudará a que esto sea posible.

## 3.4. Administración de sistemas: Cloud Computing, Ubuntu, Nginx, Pm2, Cerbot.



- **Cloud Computing.** La computación en la nube consiste en la creación de servidores desde Internet encargados de atender las peticiones en cualquier momento. Reduce los costes, garantiza un mejor tiempo de actividad y que los sitios web sean invulnerables a los ataques.

Permite a un usuario acceder a servicios y responder con ellos a las necesidades de su negocio, de forma flexible y adaptativa, en caso de demandas no previsibles o de picos de trabajo, pagando únicamente por el consumo efectuado.

Nos da la posibilidad de aumentar el número de servicios basados en la red. Esto genera beneficios tanto para los proveedores, que pueden ofrecer, de forma más

rápida y eficiente, un mayor número de servicios, como para los usuarios que tienen la posibilidad de acceder a ellos.

El concepto de cloud computing se refiere a alguna de estas tres modalidades:

- Software como servicio.
- Plataforma como Servicio.
- Infraestructura como Servicio.

El cloud computing presenta las siguientes **características**:

- **Agilidad:** mejora la oferta de recursos al usuario por parte del proveedor.
- **Coste:** los recursos en la nube suelen tener costes menores.
- **Escalabilidad y elasticidad.**
- **Independencia entre el dispositivo y la ubicación.**
- **Rendimiento:** estos sistemas optimizan el uso de recursos automáticamente.
- **Seguridad:** los proveedores dedican recursos a los problemas de seguridad.
- **Mantenimiento:** no se necesita instalación en el ordenador personal.

Las principales **ventajas** son:

- **Integración de servicios red.** Se puede integrar fácil y rápidamente con el resto de las aplicaciones empresariales.
- **Prestación de servicios a nivel mundial.** Mayor capacidad de adaptación, y recuperación completa de pérdida de datos.
- **Prescindir de instalar software,** ya que este es provisto por la plataforma en la nube. Requiere menor inversión para empezar a trabajar.
- **Implementación rápida y con menos riesgos,** las aplicaciones están disponibles en cuestión de días u horas en lugar de semanas o meses.

- **Actualizaciones automáticas** que no afectan negativamente a los recursos generados por el usuario. No hay que decidir entre actualizar y conservar el trabajo.
- **Uso eficiente de la energía.** La energía consumida es sólo la necesaria, reduciendo el desperdicio.

Existen muchas alternativas para llevar a cabo cloud computing, los proveedores más importantes son, Google Cloud Platform, Amazon Web Services, Microsoft Azure, Digital Ocean entre otros muchos.



En nuestro proyecto utilizaremos el sistema de cloud de Hetzner, que nos ofrece una CPUs Intel Skylane Xeon y SSDs NVMe, la contratación mínima es de un servidor que nos ofrecerá las siguientes características:

- 1 x vCPU.
- 2 GB RAM.
- 20 GB de almacenamiento.
- 20 TB Tráfico.
- Ubicaciones en Alemania y Finlandia.

Además, nos aportará las siguientes **herramientas**:

- **Tomos:** almacenamiento SSD de alta disponibilidad para servidores en nube. El almacenamiento puede ampliarse en cualquier momento hasta 10 TB y conectarse de forma flexible a diferentes servidores en nube.
- **Rendimiento:** CPU Intel Skylake Xeon y unidades SSD NVMe. Alta velocidad gracias a la conexión de red redundante de 10 GBit.

- **API:** administración automática, todas las funciones también están disponibles a través de una API REST y una herramienta CLI.
- **Panel de control:** para crear instancias de servidor en menos de diez segundos.
- **Snapshots:** un servidor puede ser creado o transferido entre proyectos.
- **Backups:** copias de seguridad creadas automáticamente a diario.
- **Sistema de IPs flotantes:** asignación gratuita y orientada a la demanda de IPs a un servidor redundante o a clústeres de servidores de alta disponibilidad.
- **Imágenes de Sistemas Operativos:** Hay varias distribuciones de sistemas operativos para elegir. Se proporciona la versión más reciente. En nuestro caso utilizaremos Ubuntu 18.04 LTS.
- **Tráfico:** 20 TB de tráfico inclusivo, suficiente para aplicaciones de ancho de banda intensivo.
- **Ubicaciones:** las instancias están alojadas en los centros de datos certificados por ISO-27001 en Núremberg y Falkenstein.
- **Protección contra ataques DDOS.**
- **Protección de datos.**

Como vemos, las características de este servicio son amplias y nos ahorrará mucho trabajo en planificación de sistemas. El precio mensual es de 3,01 euros, un gasto ínfimo en comparación con otro tipo de soluciones.



Utilizaremos además el servicio de compra y registro de dominio con 1&1 Ionos de rwellstattoo.com al que dirigiremos el hosting de nuestra aplicación.



- **Ubuntu 18.04 LTS.** Es un sistema operativo de código abierto. Es una distribución de Linux basada en la arquitectura de Debian. Actualmente corre en computadores de escritorio y servidores, en arquitecturas Intel, AMD y ARM. Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario.

Hetzner nos proporciona la imagen de esta distribución de Linux que utilizaremos bajo terminal de comandos para aprovechar la eficiencia y el potencial de Linux y Debian y no malgastar recursos en interfaces gráficas.

Utilizaremos Ubuntu para toda la configuración de sistemas, servidores, bases de datos y despliegue.



- **Nginx.** Es un servidor web/proxy inverso ligero de alto rendimiento. Es un software libre y de código abierto. Es multiplataforma y corre bajo sistemas Unix y Windows.

Sus ventajas principales son el menor consumo de memoria en comparación con Apache, el manejo de hasta cuatro veces más peticiones por segundo. Sus **características** son:

- Servidor de archivos estáticos.
- Proxy inverso.
- Balanceo de carga.
- HTTPS y HTTP2
- Streaming de archivos FLV y MP4.
- Compatible con IPv6.

- Compresión con gzip.
- Soporte de más de diez mil peticiones simultáneas.



- **Pm2.** Es un gestor de procesos en producción para aplicaciones Node JS que tiene un balanceador de carga incorporado. Nos permite mantener siempre activas las aplicaciones y volver a cargarlas, evitando los tiempos de inactividad, a la vez que nos facilita las tareas de administración de sistemas, además de gestionar el registro de aplicaciones, su supervisión y la agrupación en clúster.

#### Principales **características**:

- Capacidad de manejar un amplio número de aplicaciones.
- Capacidad de monitoreo de memoria y cps de nuestros procesos.
- Manejo de logs.
- Balanceo de carga.
- Iniciar aplicaciones una vez el servidor se inicia.
- Observadores de código por si este cambia.



- **Cerbot.** Es una herramienta creada por Electronic Frontier Foundation que nos permite gestionar de forma automática certificados TLS/SSL y configurar automáticamente el cifrado HTTPS en nuestro servidor. Nos dará la posibilidad de obtener los certificados Let's Encrypt y auto configurar nuestro sistema.

Let's Encrypt es una de las CA más grandes a nivel mundial, co-fundada por la EFF y Mozilla, permite a cualquier persona obtener un certificado TLS/SSL para incorporar a su página web el protocolo HTTPS de forma fácil y gratuita.

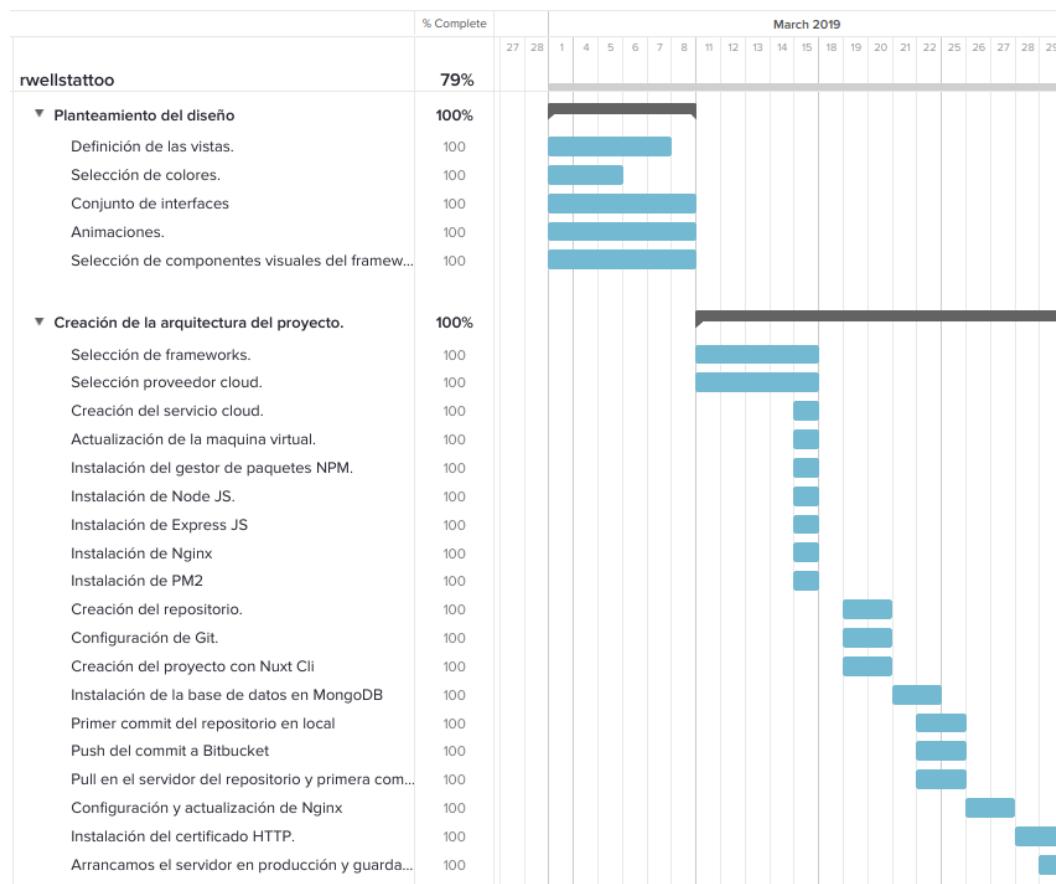
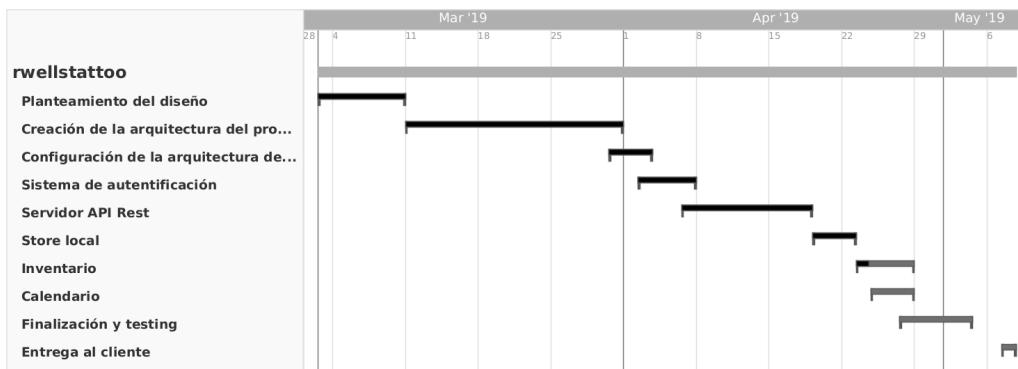
Las tecnologías que se utilizarán en este proyecto, nos ayudarán a economizar el tiempo para construir un producto robusto, escalable, seguro y moderno. Las ventajas son más que evidentes, aunque para ello deberemos renunciar a profundizar en qué es lo que hace internamente cada herramienta.

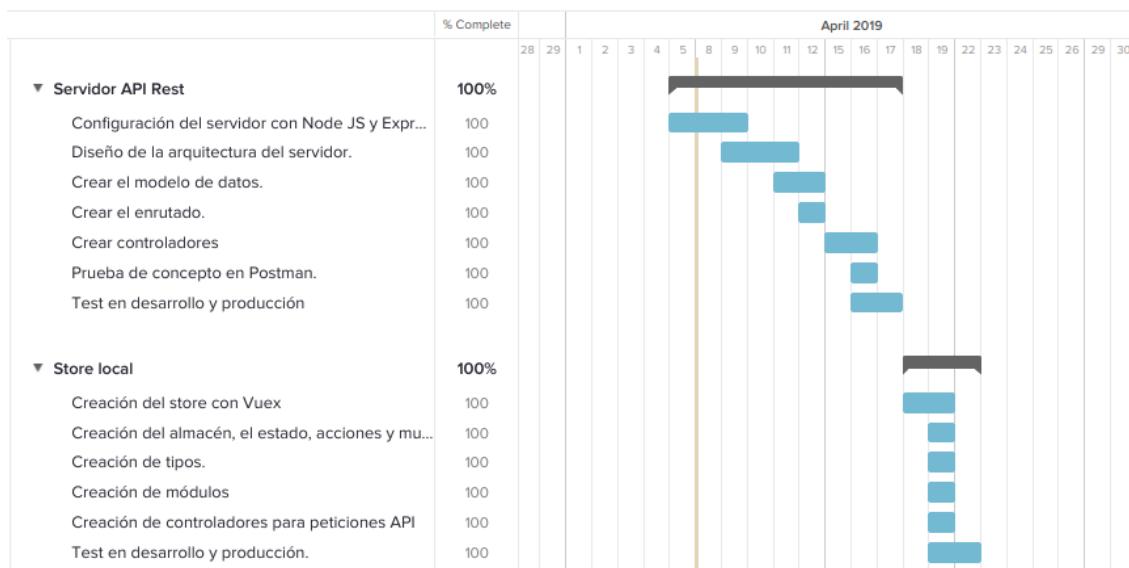
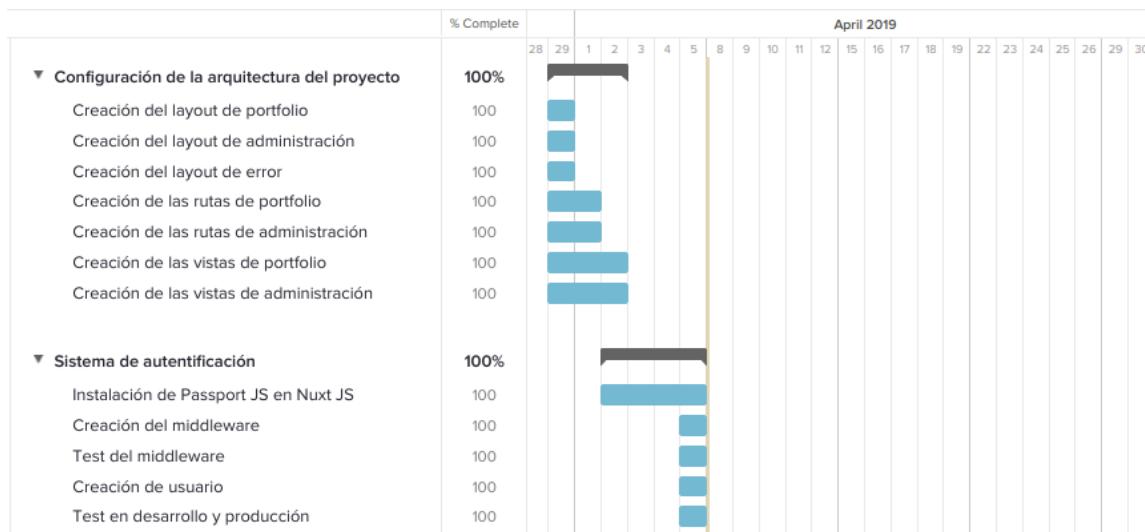
Nuestro objetivo no es otro que producir software en el menor tiempo posible y centrarnos en la tarea de programar, pero no dejaremos de lado el aprendizaje de la implantación de sistemas operativos, la configuración de servidores, bases de datos y el control de nuestros servicios.

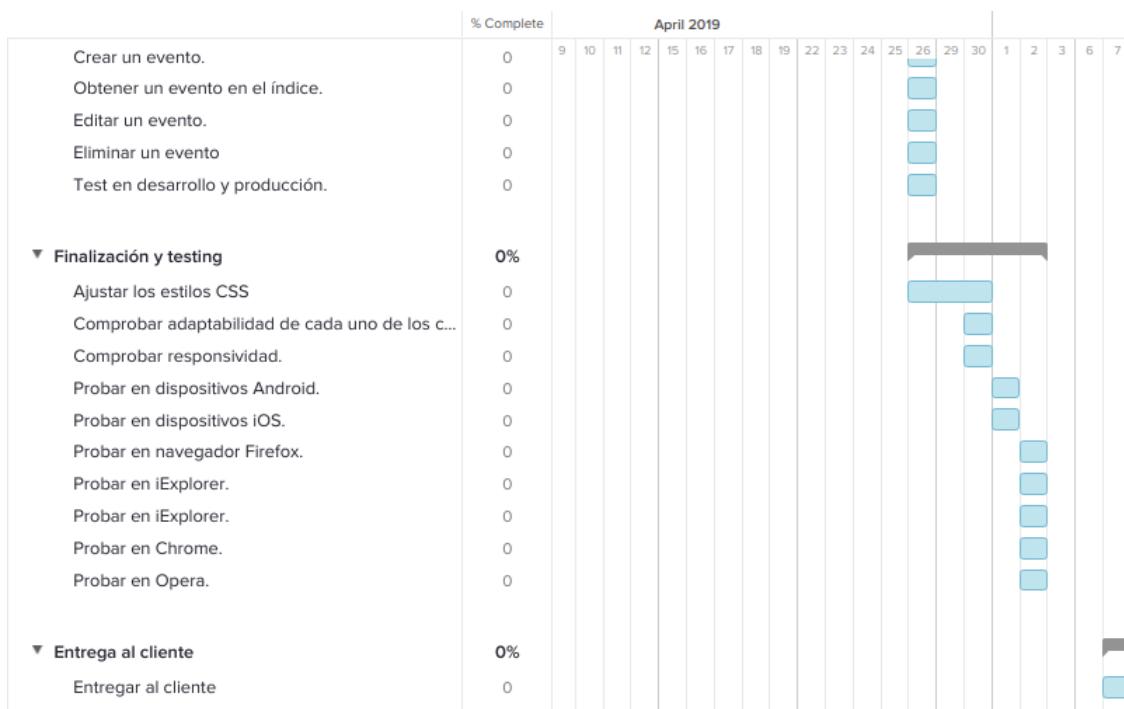
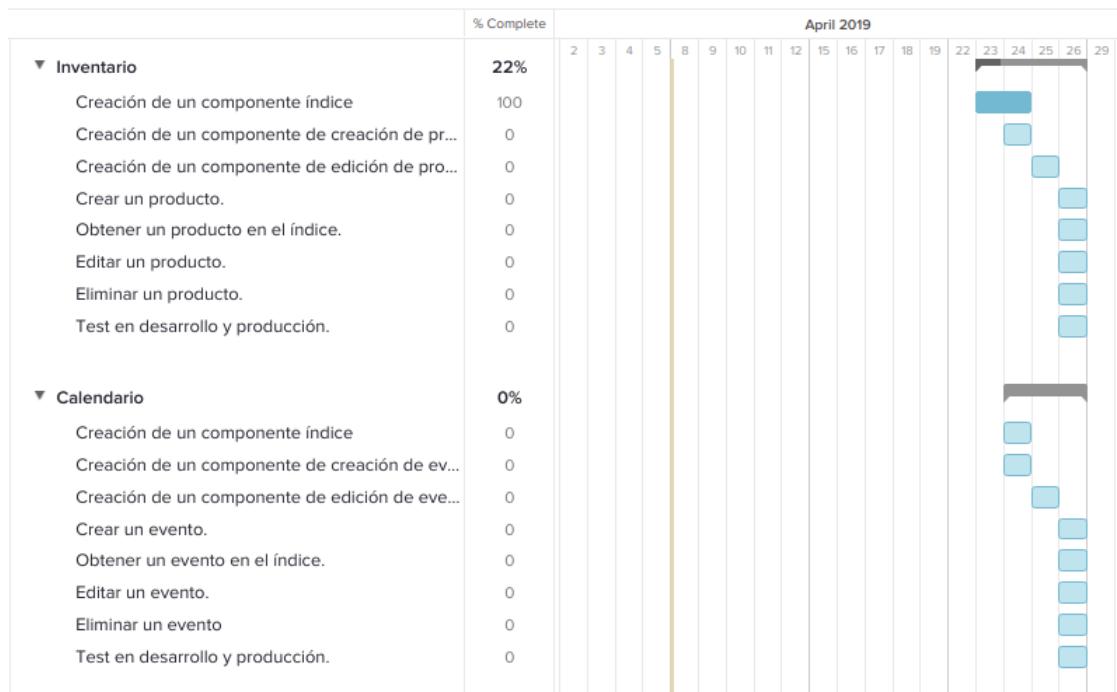
## 4. Estimación de recursos y planificación.

A continuación, se mostrarán las etapas de desarrollo del producto como ya hemos mencionado en el apartado de metodología. Al ser desarrollo ágil hemos utilizado Scrumban, pero además hemos incluido la realización de un diagrama de Gantt para fijarnos los plazos de entrega de cada sprint.

### 4.1. Diagrama de Gantt.







Los sprints han quedado desarrollados de la siguiente forma:

### **1. Planteamiento del diseño.**

- Selección de framework: Vuetify.
- Definición de vistas
- Colores.
- Interfaces.
- Animaciones.
- Selección de componentes visuales.

### **2. Creación de la arquitectura del proyecto.**

- Selección de frameworks.
- Selección proveedor cloud.
- Creación del servicio cloud.
- Actualización de la máquina virtual.
- Instalación del gestor de paquetes NPM.
- Instalación de Node JS.
- Instalación de Express JS
- Instalación de Nginx
- Instalación de PM2
- Creación del repositorio.
- Configuración de Git.
- Creación del proyecto con Nuxt Cli (Babel, Webpack, Sass, Linter...).
- Instalación de la base de datos en MongoDB
- Primer commit del repositorio en local
- Push del commit a Bitbucket
- Pull en el servidor del repositorio y primera compilación.
- Configuración y actualización de Nginx
- Instalación del certificado HTTP.
- Arrancamos el servidor en producción y guardamos cambios con PM2 para mantener los servicios activos.

### 3. Configuración de la arquitectura del proyecto.

- Creación del layout de portfolio.
- Creación del layout de administración.
- Creación del layout de error.
- Creación de las rutas de portfolio.
- Creación de las rutas de administración.
- Creación de las vistas de portfolio.
- Creación de las vistas de administración.

### 4. Sistema de autenticación.

- Instalación de Passport JS en Nuxt JS.
- Creación del middleware.
- Test del middleware.
- Creación de usuario.
- Test en desarrollo y producción.

### 5. Servidor API Rest.

- Configuración del servidor con Node JS y Express JS.
- Diseño de la arquitectura del servidor.
- Crear el modelo de datos.
- Crear el enrutado.
- Crear controladores.
- Prueba de concepto en Postman.
- Test en desarrollo y producción.

### 6. Store local.

- Creación del store con Vuex.
- Creación del almacén, el estado, acciones y mutaciones.
- Creación de tipos.
- Creación de módulos.
- Creación de controladores para peticiones API.
- Test en desarrollo y producción.

## 7. Inventario.

- Creación de un componente índice
- Creación de un componente de creación de productos.
- Creación de un componente de edición de productos.
- Crear un producto.
- Obtener un producto en el índice.
- Editar un producto.
- Eliminar un producto.
- Test en desarrollo y producción.

## 8. Calendario.

- Creación de un componente índice.
- Creación de un componente de creación de eventos.
- Creación de un componente de edición de eventos.
- Crear un evento.
- Obtener un evento en el índice.
- Editar un evento.
- Eliminar un evento.
- Test en desarrollo y producción.

## 9. Finalización y testing.

- Ajustar los estilos CSS.
- Comprobar adaptabilidad de cada uno de los componentes.
- Comprobar responsividad.
- Probar en dispositivos Android.
- Probar en dispositivos iOS.
- Probar en Firefox.
- Probar en iExplorer.
- Probar en Safari.
- Probar en Chrome.
- Probar en Opera.

## 10. Entrega al cliente.

## 4.2. Scrumban.

Además del diagrama de Gantt se han realizado los esquemas correspondientes al Scrumban, que a su vez conforman finalmente por fechas el ya mencionado diagrama. Primero hemos definido un tablero llamado Master; aquí encontraremos el backlog y el conjunto de sprints:



A continuación, hemos definido el tablero de Sprints que realizaremos y que ya hemos descrito anteriormente. Hemos realizado una descomposición de algo más genérico hacia algo más específico para poder abordar las tareas de forma progresiva y dinámica.

En cada sprint posteriormente observaremos que las tareas están más definidas y podremos acomodarnos a las dificultades técnicas modularizando correctamente los retos, hitos y problemas.





### 4.3. Presupuesto objetivo.

Respecto a la estimación de recursos, detallaremos en los sucesivos párrafos un presupuesto inicial y orientativo para llevar a cabo el desarrollo de los servicios informáticos.

#### 4.3.1. Costes directos de desarrollo.

Estos costes engloban el 90% del presupuesto, y se corresponden con todas las fases del desarrollo del software, con desarrollo de software hacemos referencia al análisis y fases de entrevistas con el cliente, la realización del código fuente, la depuración, las pruebas, las fases de testing en diferentes entornos, las pruebas por usuarios finales y la resolución de incidencias, y la publicación final que será la otorgada al cliente en una versión siempre estable.

La estimación de tiempos se encuentra en un orden secuencial en el presupuesto, aunque el ciclo del desarrollo será iterativo, es decir, incremental, se irá desarrollando en pequeñas etapas repetitivas. Esto se realizará así debido a que las necesidades pueden cambiar en un momento dado, y consecuencia de ello los procesos cambiarán.

- Planteamiento del diseño: durará aproximadamente **8 días**. (1 de marzo - 8 de marzo).

La cuantía económica de esta fase será ..... **160 euros.**

- Creación de la arquitectura del proyecto: el plazo temporal será de **18 días** (11 de marzo - 29 de marzo).

La cuantía económica de esta fase será ..... **360 euros.**

- Configuración de la arquitectura del proyecto: la duración de esta fase serán **5 días** (29 de marzo - 2 de abril).

La cuantía económica de esta fase será ..... **100 euros.**

- Sistema de autenticación: esta fase tendrá un plazo de **4 días** (2 de abril - 5 de abril).

La cuantía económica de esta fase será ..... **80 euros.**

- Servidor API Rest: el plazo será de aproximadamente **13 días** (5 de abril - 17 de abril).

La cuantía económica de esta fase será ..... **260 euros.**

- Store local: el plazo será de aproximadamente **5 días** (18 de abril - 22 de abril).

La cuantía económica de esta fase será ..... **100 euros.**

- Inventario: la fase tendrá un plazo de entrega aproximado de **5 días** (22 de abril - 26 de abril).

La cuantía económica de esta fase será ..... **100 euros.**

- Calendario: durará aproximadamente **4 días** (23 de abril - 26 de abril).

La cuantía económica de esta fase será ..... **80 euros.**

- Finalización y testing: comprobaremos que todo funciona correctamente según lo estipulado en unos **7 días** (26 de abril - 2 de mayo).

La cuantía económica de esta fase será ..... **140 euros.**

- Entrega al cliente (7 de mayo).

La tasación se estipula en **20 euros** por hora de trabajo.

#### **4.3.2. Costes indirectos de desarrollo.**

Los costes indirectos surgen como una necesidad de previsión para cumplir los plazos de entrega y necesidades especiales del cliente. Estos costes son orientativos y podrán ampliarse en caso de urgencia. Se comunicaría previamente para contar con el consentimiento de la parte contratante.

- **Consultoría a terceros:** en ocasiones es necesario contratar los servicios de un tercero para la resolución de un problema en el desarrollo, siempre de manera puntual. Este tipo de servicios incluyen, expertos en software, administradores de sistemas, administradores de bases de datos, helpdesk, ingenieros de redes, entre otros.

La cuantía económica será ..... **50 euros / hora.**

- **Servicios de almacenamiento:** la construcción de máquinas y la escalabilidad del servicio y/o producto varía según los usuarios por lo que podrá aumentar o disminuir en función de determinadas circunstancias, las migraciones ocupan una parte importante dentro de este aspecto. Por lo que tratamos de cubrir todos estos contratiempos evitando así daños futuros y retrasos innecesarios.

La cuantía económica será ..... **50 euros / mes.**

- **Repositorios privados:** nuestro centro de datos quedará cifrado a terceros y permanecerá en una nube privada para poder solucionar cualquier incidencia donde sea y cuando sea. Esto requiere siempre uno o varios repositorios privados.

La cuantía económica será ..... **70 euros / mes.**

- **Gastos de prueba:** aunque el testeo se produzca en una fase de desarrollo y se considere como coste directo, se plantean fuera de esta fase otras situaciones y condiciones de prueba que se incluyen siempre en este tipo de costes.

La cuantía económica será ..... **30 euros / hora.**

- **Licencias** de herramientas de desarrollo como **WebStorm**.

La cuantía económica será ..... **12 euros / mes.**

#### 4.3.3. Costes de mantenimiento.

Por último hablamos de unos costes temporales que son los costes de mantener el producto y los servicios. Este tipo de costes incluyen los siguientes servicios, con una cuantía de **30 euros** la hora:

- Resolución de incidencias.
- Atención al cliente 24 horas durante los 60 primeros días de implantación del producto y posterior adaptación de la empresa. (Podrá ampliarse por el cliente bajo sobrecoste).
- Atención al cliente en horario laboral a concretar con el cliente, después de los 60 primeros días.
- Modificaciones mínimas que pueda solicitar el cliente.

## 5. Desarrollo del proyecto.

### 5.1. Inicio.

A la hora de establecer las necesidades del proyecto y su finalidad podemos atender a las siguientes necesidades funcionales:

- Aplicación multiplataforma que se desarrollará como Progressive Web App bajo el framework de Javascript Nuxt JS.
- Infraestructura funcional bajo la premisa del cloud computing para minimizar los costes de infraestructura y optimización de los recursos, mediante la creación de una máquina virtual que gestione la base de datos en MongoDB, el proxy gestionado por Nginx y los certificados pertinentes.
- Control de versiones en producción y desarrollo bajo el sistema de versionado Git.
- Tecnologías modernas para el desarrollo. Las tecnologías empleadas así como los lenguajes deben de ser modernos, actualizables y escalables.
- Posibilidad de versionado y actualización sin costes excesivos.

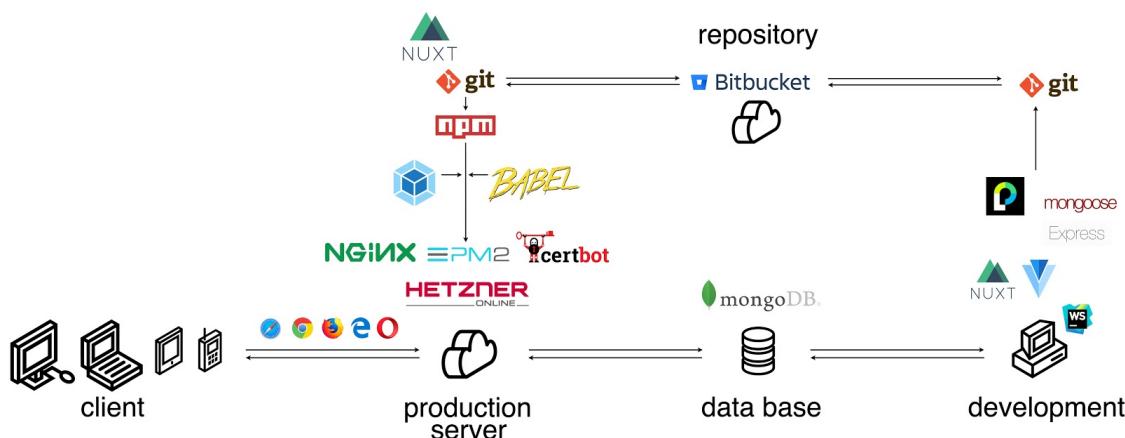
Llegado a este momento, es muy importante reflexionar acerca de todos estos puntos, ya que con ellos diseñaremos la arquitectura y los patrones de comunicación de cada parte del proyecto.

Cada paso en falso nos hará fallar en la estimación temporal, por eso la selección de tecnología y sus herramientas ha resultado imprescindible en esta parte del desarrollo. La elección de cada herramienta tecnológica no ha sido azarosa, sino que responde a necesidades concretas para un desarrollo ágil del producto.

Así como el proceso de desarrollo se basa en sprints y hemos conseguido diseñar una implementación por fases asumibles, hay que tener en cuenta que la mayor parte de las tecnologías están aún por explorar, por ello en el momento en que no se consiga el sprint deberemos desechar la forma de implementar las mismas.

El fin del proyecto es dar al cliente una aplicación funcional, intuitiva y sobre todo ejecutable en cualquier dispositivo.

Antes de proceder a analizar la estructura del proyecto, hemos realizado un diagrama que nos explicará el patrón de comunicación entre el cliente, el servidor de producción, el repositorio, la base datos, y desarrollo:



- **Cliente:** realiza la petición a través del navegador al servidor de producción. El servidor de producción devuelve mediante una respuesta el repositorio alojado en el mismo. El cliente descarga la versión más actualizada del servidor de producción y ejecuta la Progressive Web App. De este modo el cliente tendrá siempre y en todo momento la versión más reciente y actualizada de la aplicación, sin tener que descargarla e instalarla en su equipo. Como vemos el coste de mantenimiento versus una aplicación nativa es simplemente muy inferior.
- **Producción:** el servidor de producción devuelve la respuesta al cliente anteriormente explicada, pero además aloja el repositorio de nuestro proyecto y se comunica con la base de datos. En este caso la tenemos alojada dentro de la misma máquina. Para realizar el despliegue de la aplicación primero deberemos descargarnos a través de nuestro servicio de control de versiones la última versión subida (Git y Bitbucket).

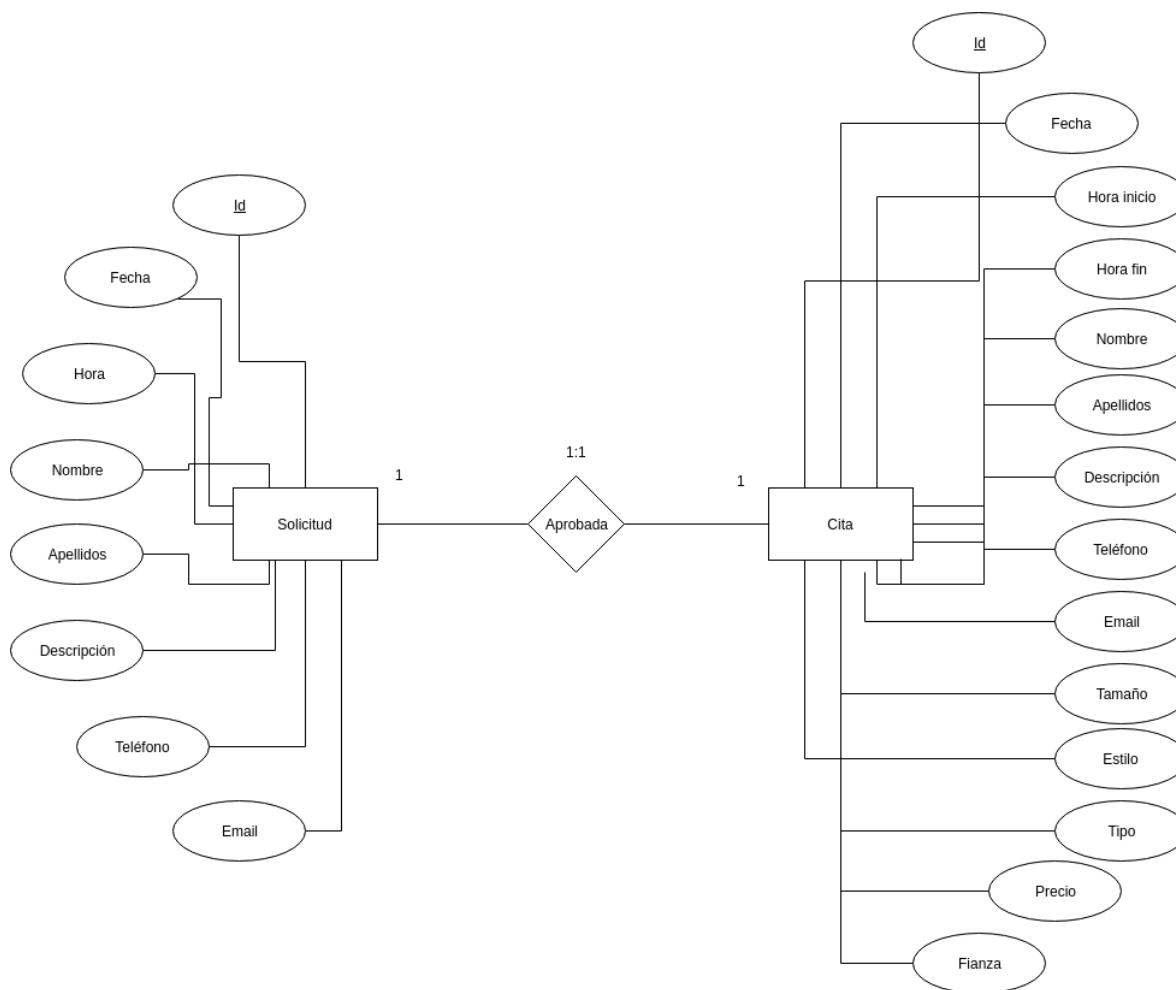
Una vez instalaremos las dependencias que se hayan creado nuevas a través de nuestro gestor de paquetes (NPM). Seguiremos con la construcción a través del gestor de paquetes para producción (build), es decir, la compilación del proyecto;

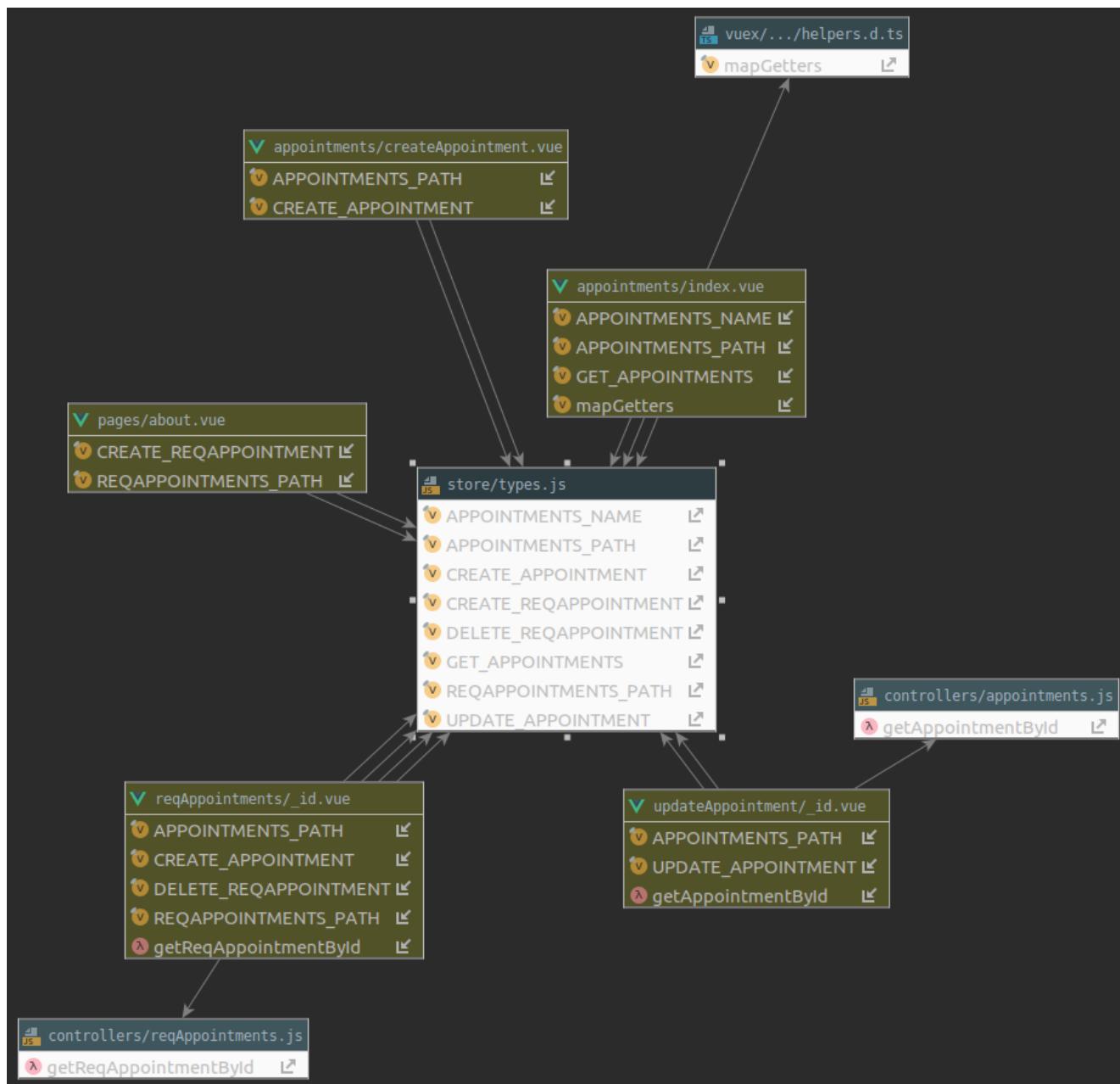
una vez realizada, configuraremos nuestro proxy para dirigir las peticiones del servidor local (Express) a nuestro dominio (con su IP) a través de nuestro framework proxy (Nginx), reseteamos nuestro manager de procesos (PM2) o lo arrancamos si es la primera vez que publicamos nuestra aplicación.

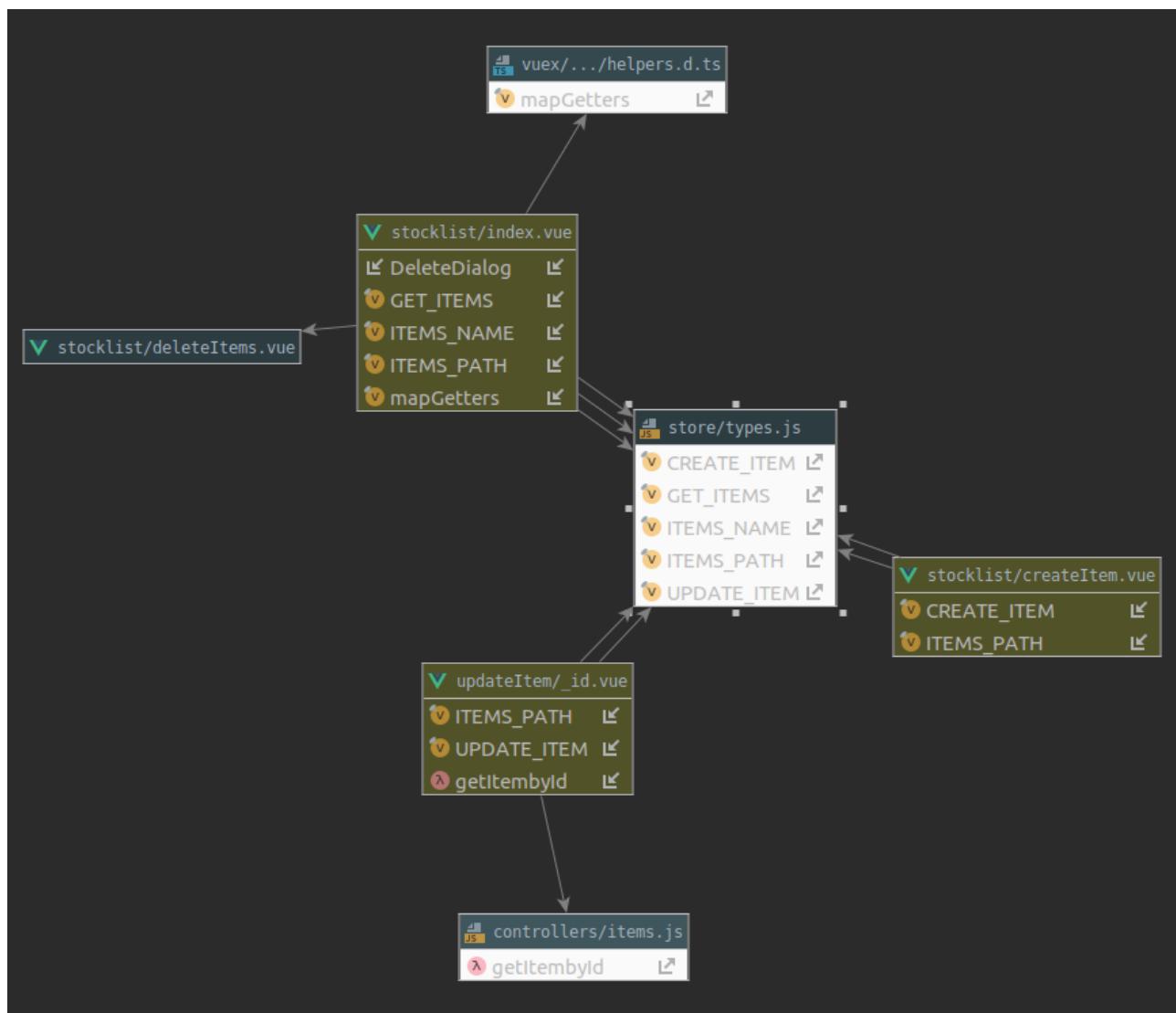
- **Base de datos:** almacenará los datos que crearemos a través de nuestra aplicación. Se comunicará a través del servidor, que le enviará las peticiones, create, read, update y delete. Existen dos bases de datos, una de producción y otra de desarrollo, para garantizar la individualidad de los sistemas y sus configuraciones.
- **Desarrollo:** realizaremos el versionado y el desarrollo de la aplicación, que subiremos a producción a través del repositorio. Deberemos testear en local o desarrollo y posteriormente en producción. Será nuestro entorno local para programar la aplicación, donde testeamos hasta finalizar las tareas y sprints.
- **Repositorio:** es otro servicio de alojamiento, en el realizaremos el control de versiones, muy importante para restaurar posibles fallos en producción, o simplemente para mantener nuestro versionado y una parte de las copias de seguridad del código, en un proveedor independiente. Se comunica con el servidor de producción para actualizarlo y con el de desarrollo que será el que nos envía las nuevas versiones.

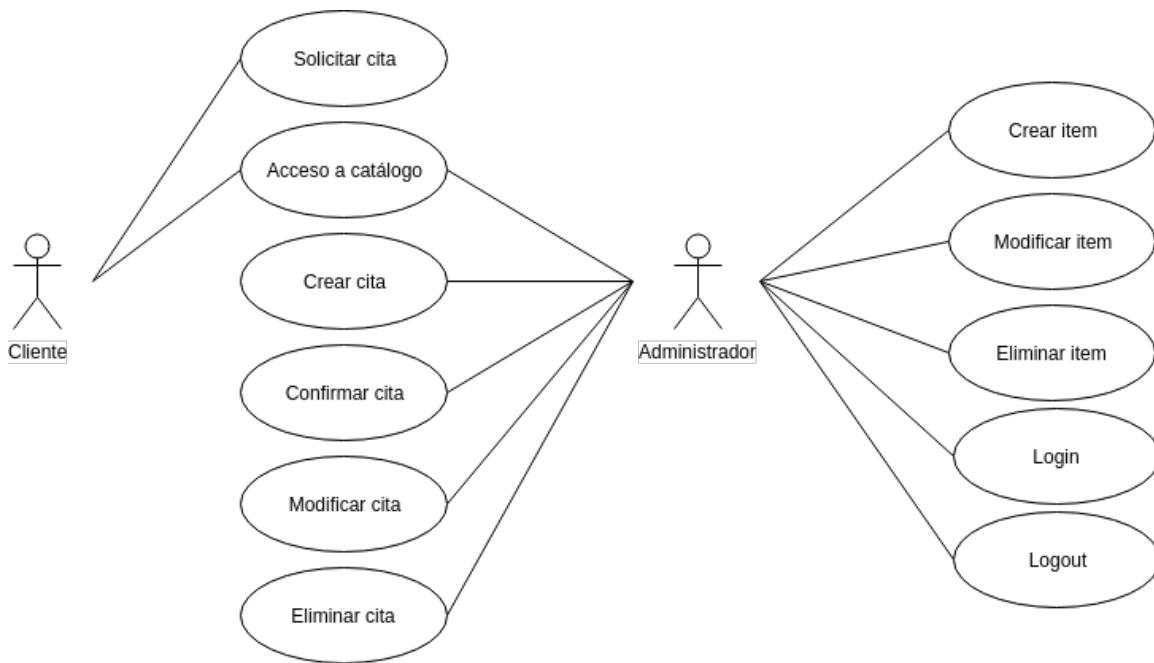
## 5.2. Análisis.

En cuanto al análisis del proyecto se muestran a continuación los siguientes diagramas del modelo entidad relación, diagramas de clases y diagramas de usos. En el apartado siguiente se detalla la comunicación y el desarrollo de cada parte del producto desarrollado de una forma más lógica.









### 5.3. Diseño.

Antes de explicar las partes del código más relevantes y como se comunican entre sí, se resumirá a continuación la constitución de la arquitectura del proyecto, tanto a nivel de frontend como a nivel de backend y su administración de sistemas y control de versionado.

#### 5.3.1. Administración sistemas.

En primer lugar deberemos crear una máquina virtual en Hetzner, la cual será generada con un sistema operativo Ubuntu 18. Lo siguiente que realizaremos es el cambio de la ip en 1&1 para redirigir nuestro dominio a la ip de nuestra máquina virtual.

Ahora entraremos en nuestra máquina virtual y actualizaremos los sistemas mediante los comandos apt update y apt upgrade.

Después instalaremos NPM como gestor de paquetes mediante el comando apt install npm. Instalamos NGINX con el comando apt install nginx.

Instalaremos además NVM para realizar instalaciones y actualizaciones de Node JS mediante el siguiente comando:

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash
```

Reiniciamos nuestro terminal y accederemos de nuevo a la maquina para aplicar cambios. Una vez ejecutado el terminal instalaremos node a través de NVM y elegiremos la versión más actual y estable con largo desarrollo a través de nvm ls-remote, para listar las versiones y nvm install 10.15.1, para instalar la versión 10.15.1 de Node JS.

A continuación configuraremos NGINX mediante la siguiente documentación:

<https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-18-04-quickstart>

A través de esta configuración queda establecido nuestro proxy y las redirecciones a nuestro dominio.

El siguiente paso es instalar PM2 mediante npm i -g pm2, con ello podremos guardar el arranque de nuestro proyecto en producción en el servidor y resetear cuando realicemos actualizaciones y despliegues a producción nuevos.

Instalaremos además Let's Encrypt mediante Cerbot para proteger nuestro proyecto mediante la capa https. Lo realizaremos introduciendo los siguientes comandos:

```
add-apt-repository ppa:certbot/certbot
sudo apt install python-certbot-nginx
sudo systemctl reload nginx
sudo certbot --nginx -d example.com -d www.example.com
```

Nos queda además instalar MongoDB en nuestro servidor comprobar la actividad del servicio y arrancarlo a través de los siguientes comandos:

```
sudo apt install -y mongodb
```

```
systemctl status mongodb
sudo systemctl start mongodb
```

Realizado esto deberemos crear un usuario administrador dentro de mongo para que Hetzner no nos bloquee la máquina por infringir medidas de seguridad. Entraremos en mongo y ejecutaremos lo siguiente:

```
mongo --host 127.0.0.1:27017
use admin
db.createUser({user:"admin",      pwd:"new_password_here",      roles:[{role:"root",
db:"admin"}]})
```

Saldremos de la consola de mongo y entraremos en el archivo de configuración y cambiaremos lo siguiente:

```
sudo nano /lib/systemd/system/mongod.service
ExecStart=/usr/bin/mongod --auth --config /etc/mongod.conf
```

Por último reseteamos mongo:

```
sudo systemctl daemon-reload
sudo service mongod restart
```

Para acceder a mongo entraremos con el siguiente comando:

```
mongo -u admin -p new_password_here --authenticationDatabase admin
```

### 5.3.2. Creación del proyecto y control de versiones.

A continuación, en nuestra máquina personal crearemos un nuevo proyecto en NuxtJS en el directorio que queramos. Ejecutaremos el siguiente comando:

```
npx create-nuxt-app <project-name>
```

El instalador de Nuxt nos ofrecerá diferentes frameworks para el servidor, elegiremos entonces Express, posteriormente en el UI framework elegiremos Vuetify, y en el modo de Nuxt elegiremos Universal. Añadiremos además ESLint y Prettier.

Creado nuestro boilerplate de proyecto en NuxtJS, accederemos al directorio y ejecutaremos mediante npm run dev en el puerto <http://localhost:3000>.

A continuación procederemos a configurar git en nuestro proyecto para poder realizar nuestro control de versiones. Dentro de nuestro proyecto ejecutaremos lo siguiente a través de nuestra terminal:

```
git init
echo '{name_project}' > README.md
git add README.md
git commit -m 'Initial Commit'
git remote add origin
https://username@your.bitbucket.domain:7999/yourproject/repo.git
git push -u origin master
```

Para arrancar nuestro proyecto, si nos descargamos una versión anterior del repositorio, deberemos en primer lugar instalar las dependencias y después arrancar en el puerto https://localhost:3000:

```
npm i
npm run dev
```

Realizado nuestro primer push a master tendremos alojado en Bitbucket nuestro código. En el apartado 6. Despliegue y pruebas detallaremos como se realizará el despliegue y el sistema de integración continua.

### 5.3.3. Desarrollo del proyecto en local.

En este apartado explicaremos las partes más importantes del proyecto, su estructura y la comunicación entre ambas y con el servidor de producción.

El proyecto en NuxtJS está formado por los siguientes directorios:

- **rwellstattoo:**
  - .nuxt
  - assets
  - components
  - controllers
  - layouts
  - node\_modules
  - pages
  - plugins
  - server
  - static
  - store

Además en la carpeta principal encontraremos una serie de archivos de configuración:

- babelrc
- .env
- .eslintrc.js
- nuxt.config.js
- package.json
- package-lock.json
- README.md

Cada directorio cumple una función importante dentro de la arquitectura NuxtJS, que se detalla a continuación:

- .nuxt

Es el directorio de automatización de tareas realizadas internamente por NuxtJS, en el encontraremos la lógica interna del router dinámico, el servidor, el store, el middleware, el módulo de autenticación, los layouts y las vistas entre otras opciones que nos permite automatizar este framework.

- assets

En este directorio encontraremos los medios que se encuentran alojados en el proyecto, como las imágenes, iconografía y otros recursos visuales.

- static

En static se encuentran los medios utilizados para generar el fav.icon y el ícono de acceso a la PWA.

- node\_modules

Aquí se sitúan todas las dependencias instaladas de NPM.

- plugins

En plugins podremos instalar cualquier plugin o librería externa que necesitemos acoplar a nuestro proyecto. En este caso tendremos un plugin de Vuetify para generar la paleta de colores de nuestro tema y no tener que hacer un escalado en todos los componentes. Quedará definida a través de un objeto en Javascript:

```
theme: {  
    primary: colors.purple.darken3,  
    accent: colors.grey.darken3,  
    secondary: colors.amber.lighten5,  
    info: colors.teal.lighten1,  
    warning: colors.amber.base,  
    error: colors.deepOrange.accent4,
```

```
    success: colors.green.accent3
}
```

- layouts

Aquí crearemos los componentes que definirán nuestros layouts para el portfolio, las páginas de error, o la parte de administración. Existen tres componentes en .vue:

- landing
- error
- admin

Aquí cabe destacar la función que utilizaremos en el componente de admin que nos realizará un logout del sistema de usuarios, a través del módulo de autenticación, una vez presionemos sobre el botón nos redirigirá a la parte de login y habremos perdido la sesión.

```
logout() {
  this.$auth.logout()
  .then(() => {
    this.$router.push({ path: '/admin/login' })
  })
}
```

- components

En esta carpeta encontraremos los componentes reutilizados en varias partes del proyecto, estos externalizan el borrado en appointments, reqAppointments y el stocklist.

Cada componente consiste en un modal con una función muy parecida que nos elimina el registro en la base de datos y en el store de Vue. El ejemplo de deleteAppointment es el siguiente:

```
deleteAppointment() {  
    this.$store.dispatch(APPOINTMENTS_PATH + DELETE_APPOINTMENT,  
    this.appointmentId)  
    this.deleteBtn = false  
}
```

Lo que hace este tipo de función es mandar un evento de borrado al store en el que pasamos el id del appointment y a su vez el store se comunica con la base de datos para eliminarlo de su registro. En las partes de server y store se detallan con más exactitud el funcionamiento de estas capas y su relación con el resto de componentes.

- pages

Pages define por componentes las rutas dinámicas de la aplicación, en una primera instancia encontraremos las que organizan el portfolio:

- index.vue
- gallery.vue
- about.vue

En el componente about.vue encontraremos el formulario para la petición de citas por parte de un cliente donde recogeremos los datos en un modelo reqAppointment y que cuando le demos a enviar utilizará la siguiente función para crear un nuevo reqAppointment para la parte de administración:

```
addRequest() {  
    if (this.$refs.form.validate()) {  
        this.$store.dispatch(REQAPPOINTMENTS_PATH +  
        CREATE_REQAPPOINTMENT, this.reqAppointment)  
        .then((response) => {  
            this.$router.push({ path: '/' })  
        })  
        this.dialogBtn = false  
    }  
}
```

```
    }
    this.dialogBtn = false
}
```

La parte de administración se encuentra formada por la siguiente estructura:

- index.vue
- login.vue
- appointments
  - updateAppointment
    - \_id.vue
  - createAppointment.vue
  - index.vue
- dates
  - updateDate
    - \_id.vue
  - index.vue
- reqAppointments
  - \_id.vue
- stocklist
  - updateItem
    - \_id.vue
  - createItem.vue
  - index.vue

En la parte de administración tenemos el componente de **login.vue** que contiene la vista del login de usuarios a través de la siguiente función:

```
login() {
  this.$auth.loginWith('local', {
    data: {
      email: this.email,
      password: this.password
    }
  })
}
```

```
.then(() => {
    this.$router.push({ path: '/admin' })
})
.catch((err) => {
    this.errorsMsg = err.response.data
    if (err.response.data.msg) {
        this.snackbarMsg = err.response.data.msg
        this.snackbar = true
    }
})
}
```

Una vez autenticados podremos acceder a **index.vue** que coincide con la ruta /admin de nuestra aplicación. Accediendo a index.vue encontraremos las peticiones de cita (reqAppointment), estas se obtienen a través de la siguiente función:

```
computed: {
    ...mapGetters(REQAPPOINTMENTS_NAME, ['getReqAppointments'])
},
mounted() {
    this.$store.dispatch(REQAPPOINTMENTS_PATH + GET_REQAPPOINTMENTS)
},
```

Se encuentran en un mounted de vue que lo que hace es pedir los datos siempre que se monte el componente. El computed extenderá los cambios reactivos del store para mostrarnos los cambios en todo momento.

Cuando pulsemos sobre el botón de confirmación en cada reqAppointment nos llevará a un formulario donde podremos confirmar la cita y cuando esto suceda, se eliminará del store y la base de datos el reqAppointment.

Accediendo a esta vista en primer lugar se carga el reqAppointment creado por el cliente, una vez montado igualaremos el reqAppointment a un appointment y eliminaremos el \_id del mismo (para que mongo nos genere uno nuevo) mediante la siguiente función:

```
mounted() {
  getReqAppointmentById(this.$route.params.id)
    .then((response) => {
      this.reqAppointment = response.data
      this.appointment = Object.assign({}, this.reqAppointment)
      this.appointment.date = new Date(this.reqAppointment)
      delete this.appointment._id
    })
}
```

Cuando enviamos el formulario, se eliminará el reqAppointment y se creará el Appointment en el store y la base de datos:

```
createAppointment() {
  /*eslint-disable*/
  console.log(this.appointment.date)

  this.$store.dispatch(APPOINTMENTS_PATH + CREATE_APPOINTMENT,
  this.appointment)
    this.$store.dispatch(REQAPPOINTMENTS_PATH + +
DELETE_REQAPPOINTMENT, this.reqAppointment._id)
    .then((response) => {
      this.$router.push({ path: '/admin/appointments' })
    })
    .catch((err) => {
      alert(err)
    })
}
```

Podremos ver cada Appointment generado en su vista correspondiente, que se encuentra en /appointments/index.vue en las dependencias del proyecto.

Aquí encontraremos el calendario que nos permite ver la disposición de cada cita almacenada. Cargaremos los datos mediante un get en un computado y formateamos los datos para adaptarlos al formato del componente del calendario de Vuetify:

```
computed: {
    ...mapGetters(APPOINTMENTS_NAME, ['getAppointments']),
    formattedAppointments() {
        const map = {}
        this.getAppointments.forEach(e => (map[e.date] = map[e.date] || []).push({
            date: e.date,
            details: e.description,
            open: false,
            title: e.name,
            id: e._id,
            startTime: e.startTime,
            endTime: e.endTime,
            price: e.price,
            deposit: e.deposit
        }))
        return map
    }
}

mounted() {
    this.$store.dispatch(APPOINTMENTS_PATH + GET_APPOINTMENTS)
},
```

Cuando seleccionemos un evento podremos editarla en su vista. Podremos actualizar cambios en el componente `_id.vue` (dentro de `updateAppointment`), con la siguiente función:

```
updateAppointment(appointment) {
    if (this.$refs.form.validate()) {
        this.$store.dispatch(APPOINTMENTS_PATH + UPDATE_APPOINTMENT,
            appointment)
```

```
.then((response) => {
  this.$router.push({ path: '/admin/appointments' })
})
}
```

El proceso siempre es el mismo, se pasa por el store, se actualiza el cambio y se dispara una petición API Rest a la base de datos que obtiene, actualiza, crea o elimina los datos.

Además la creación de citas directamente por el usuario se realizará mediante una función en `createAppointment.vue`:

```
addAppointment() {
  if (this.$refs.form.validate()) {
    this.$store.dispatch(APPOINTMENTS_PATH + CREATE_APPOINTMENT,
    this.appointment)
    .then((response) => {
      this.$router.push({path: '/admin/appointments'})
    })
  }
},
```

En las dependencias de stocklist el código es similar, las funciones realizan los mismos patrones de comunicación. Se crea un item, se manda al store, y de este se dispara una petición por Axios a la base de datos.

- controllers

En esta parte encontraremos las funciones que comunican el store con las peticiones axios a la base de datos. Cada fichero corresponde su homónimo del directorio store. Cada función será importada en el store para ejecutar peticiones. Así separaremos nuestro código para hacerlo mantenible. Un ejemplo de una petición get Appointment sería la siguiente:

```
export function getAppointments() {  
    return request({  
        url: '/appointment/getAppointments',  
        method: 'get'  
    })  
}
```

La url, corresponde a la ruta de las APIS creadas en servidor que detallaremos en los siguientes puntos. Debemos además usar el método que utilizamos, que puede ser get, post y delete.

- store

El store de Vue es un estado local que nos permite generar reactividad en los componentes sin el uso del servidor, es decir, los cambios se generan primero en el lado del cliente y después se envían al servidor y así conseguimos una baja carga de peticiones en el servidor. Lo que se traduce en mayor velocidad.

Dentro de este directorio encontramos un fichero **index.js**, donde instanciamos las partes del store y las separaremos en otros componentes para hacer el código mantenible. El store de Vue se forma por un estado (state), unos getters, unas acciones (actions), y unas mutaciones (mutations).

Además de index.js, tendremos otro fichero muy importante denominado **types.js**, donde instanciaremos en el front y en cada componente del store cada acción, mutación, nombre o ruta.

Dicho esto, en cada componente del store, por ejemplo, **appointments.js**, importamos las funciones de los controladores de axios, e importamos los tipos de **types.js**. Exportaremos el state del array de appointments. Además definiremos un getter en el estado que nos generará los appointments del store mediante la función que hayamos definido en axios, en este caso:

```
// Getters
```

```
export const getters = {
  getAppointments: state => state.appointments
}
```

Con respecto a **las acciones** sucederá igual, exportamos acciones cuando los cambios se hagan en base de datos, a través de un commit y mediante una promesa o un async, están los dos ejemplos, aunque el modo más utilizado en este proyecto es mediante promesas:

```
// ejemplo promise
[GET_APPOINTMENTS]({ state, commit }) {
  return new Promise((resolve, reject) => {
    getAppointments()
      .then((response) => {
        commit(SET_APPOINTMENTS, response.data)
        resolve()
      })
      .catch((err) => {
        reject(err)
      })
  })
},
```

```
// ejemplo async
[CREATE_APPOINTMENT]({ state, commit }, appointment) {
  return new Promise(async (resolve, reject) => {
    try {
      const response = await createAppointment(appointment)
      commit(SET_APPOINTMENTS, response.data.appointments)
      resolve()
    } catch (err) {
      reject(err)
    }
  })
}
```

```
    })  
},
```

Las **mutaciones** se utilizarán cuando se den cambios en el store de vue:

```
export const mutations = {  
  [SET_APPOINTMENTS](state, appointments) {  
    state.appointments = appointments  
  }  
}
```

- server

Por último explicaremos la parte de servidor, que en sí es la más compleja. Esta parte se ha modularizado para poder controlar y mantener el código, de la siguiente manera:

- server:
  - config
    - passport.js
  - controllers
    - api.js
    - auth.js
    - appointment.js
    - item.js
    - reqAppointment.js
    - user.js
  - models
    - appointment.js
    - item.js
    - reqAppointment.js
    - user.js
  - utils
  - index.js

- config: encontramos dentro de passport.js la configuración de login que nos da la librería passport donde requerimos este módulo, crearemos un usuario que sigue un modelo user.js que encontramos en la carpeta server/models/user.js. Además se definen el registro de mail, el middleware y la autorización.

```
const User = require('../models/user')
```

```
passport.serializeUser((user, done) => {  
    done(null, user.id)  
})
```

```
passport.deserializeUser((id, done) => {  
    User.findById(id, (err, user) => {  
        done(err, user)  
    })  
})
```

- controllers:

- auth.js: esta parte se comunicará con passport y con el modelo de usuario, exportará la función de logeo y su validación, además de la autenticación, y el logout y el registro. Estas son las funciones:

```
exports.login = function (req, res, next) {  
    ...  
    passport.authenticate('local', (err, user, info) => {  
        if (err) { return responses.error(res, err) }  
        if (!user) {  
            return responses.error(res, info)  
        }  
        req.login(user, (err) => {  
            if (err) { return responses.error(res, err) }
```

```
        responses.success(res, { msg: 'Success! You are logged in.'  
    })  
    })  
})(req, res, next)  
  
}  
  
exports.logout = function (req, res) {  
    responses.success(res, { status: 'OK' })  
}  
  
exports.register = function (req, res) {  
    const user = new User(req.body)  
    user.save()  
    .then((userInfo) => {  
        responses.success(res, userInfo)  
    })  
    .catch((err) => {  
        responses.error(res, { msg: err })  
    })  
}
```

- api.js: Aquí definiremos las rutas del backend de nuestro servidor. Importaremos, express, router de express, la configuración de passport y los controladores del servidor. Y cada ruta quedará definida de la siguiente manera:

```
router.post('/auth/login', authController.login)  
router.get('/user', passportConfig.isAuthenticated,  
userController.getUser)  
router.post('/addItem', itemController.addItem)
```

- appointment.js: este será un ejemplo de item.js, reqAppointment.js y appointment.js, ya que la estructura será siempre la misma en cada uno, variando únicamente las funciones y su contenido. En este componente requeriremos el

modelo correspondiente y las responses ubicadas en utils. Cada función se exportará y será utilizada en api.js como controlador para obtener los datos que requiramos en cada ruta, tal y como hemos visto anteriormente. Así pues un ejemplo de añadir un appointment será:

```
exports.addAppointment = function (req, res) {
  const appointment = new Appointment(req.body)
  appointment.save((err) => {
    if (err) {
      return responses.error(res, { msg: err })
    }
    Appointment.find({})
      .then((appointments) => {
        return responses.success(res, { msg: 'appointment created',
          appointments })
      })
      .catch((err) => {
        return responses.error(res, { msg: err })
      })
  })
}
```

Esta función lo que hace es crear una nueva instancia del modelo appointment que ejecutará un método find({}) que responderá creando un objeto en la base de datos. Este es el punto de modelización de datos, donde hacemos uso de mongoose. Es donde mediante funciones javascript creamos, leemos, actualizamos y borramos datos de la base de datos. El resto del código son bloques de control de errores.

Cada componente de controllers funciona de la misma manera.

- Models: aquí definiremos como será nuestro modelo y los datos que se almacenarán en MongoDB. Así pues, por ejemplo en appointment.js tendremos:

```
const mongoose = require('mongoose')

const AppointmentSchema = new mongoose.Schema({
    startTime: String,
    endTime: String,
    date: String,
    name: { type: String, default: 'Anonymous' },
    lastname: { type: String, default: 'Anonymous' },
    phone: String,
    email: String,
    description: String,
    type: String,
    style: String,
    size: Number,
    price: Number,
    deposit: Number
}, { timestamps: true })

module.exports = mongoose.model('Appointment', AppointmentSchema)
```

- utils: aquí encontramos el fichero responses.js, que nos generará las respuestas para el control de errores, como por ejemplo:

```
exports.success = (res, message) => {
    res.status(200).json(message)
}
```

## 6. Despliegue y pruebas.

### 6.1. Plan de prueba.

Nº	ESPECIFICACIÓN DE PRUEBAS
1	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Registro de único usuario.</b></li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Único usuario registrado.</li><li>• Validación y verificación de inputs mail y nombre.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Dar de alta usuario con Postman y no habilitar vista.</li><li>• Email válido con caracteres.</li><li>• Invalido campos en blanco.</li><li>• Mínimo de caracteres.</li></ul></li></ul>
2	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Login de usuario.</b></li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Acceso mediante el usuario registrado.</li><li>• Validación de inputs.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Enviar datos no registrados.</li><li>• Enviar datos registrados.</li><li>• Escribir datos incorrectamente validados.</li></ul></li></ul>
3	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Logout de usuario.</b></li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Perdida de sesión al realizar logout.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Loguearse en el sistema y hacer logout.</li><li>• Refrescar página habiendo hecho logout.</li><li>• Tratar de acceder sin hacer login en administración.</li></ul></li></ul>
4	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Creación de petición de cita.</b></li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Envío de campos.</li><li>• Validación en cliente.</li></ul></li></ul>

	<ul style="list-style-type: none"><li>• Reflejo en administración sin edición directa.</li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Relleno de datos y envío a la base de datos y al store.</li><li>• Campos en blanco y/o datos incorrectos.</li><li>• Acceso a administración y reflejo de la petición creada.</li></ul></li></ul>
5	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Aceptación de cita</b>.</li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Envio de campos.</li><li>• Validación en cliente.</li><li>• Eliminación de petición.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Relleno de datos y envío a la base de datos y al store.</li><li>• Campos en blanco y/o datos incorrectos.</li><li>• Acceso a cita y reflejo de la cita creada.</li><li>• Acceso a administración index y reflejo de eliminación.</li></ul></li></ul>
6	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Creación de cita</b>.</li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Envio de campos.</li><li>• Validación en cliente.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Relleno de datos y envío a la base de datos y al store.</li><li>• Campos en blanco y/o datos incorrectos.</li><li>• Acceso a cita y reflejo de la cita creada.</li></ul></li></ul>
7	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Modificación de cita</b>.</li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Envio de campos.</li><li>• Validación en cliente.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Relleno de datos y envío a la base de datos y al store.</li><li>• Campos en blanco y/o datos incorrectos.</li><li>• Acceso a cita y reflejo de la cita modificación.</li></ul></li></ul>

8	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Supresión de cita.</b></li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Eliminación de registro.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Eliminamos registro y refrescamos la página.</li><li>• Eliminamos registro, hacemos logout y refrescamos la página.</li></ul></li></ul>
9	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Creación de Item.</b></li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Envío de campos.</li><li>• Validación en cliente.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Relleno de datos y envío a la base de datos y al store.</li><li>• Campos en blanco y/o datos incorrectos.</li><li>• Acceso a stocklist y reflejo del item creado.</li></ul></li></ul>
10	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Modificación de item.</b></li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Envío de campos.</li><li>• Validación en cliente.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Relleno de datos y envío a la base de datos y al store.</li><li>• Campos en blanco y/o datos incorrectos.</li><li>• Acceso a stocklist y reflejo del item modificado.</li></ul></li></ul>
11	<ul style="list-style-type: none"><li>• Objetivo probado: <b>Supresión de item.</b></li><li>• Requisitos probados:<ul style="list-style-type: none"><li>• Eliminación de registro.</li></ul></li><li>• Pruebas a Realizar:<ul style="list-style-type: none"><li>• Eliminamos registro y refrescamos la página.</li><li>• Eliminamos registro, hacemos logout y refrescamos la página.</li></ul></li></ul>

## 6.2. Despliegue.

En la fase de desarrollo iremos además de probando la funcionalidad en local desplegando cada fase del proyecto en producción. Para ello cada parte desarrollada y probada en local será comiteada y enviada a nuestro repositorio en Bitbucket.

Accederemos entonces nuestra máquina en Hetzner y en primer lugar configuraremos nuestro sistema de automatización de despliegue. Para ello realizaremos lo siguiente:

```
git init --bare /opt/my-first-app.git
git clone /opt/my-first-app.git /opt/live/my-first-app
```

Y realizamos un script de post-receive mediante nano en /opt/my-first-app.git/hooks. Donde realizaremos el proceso de automatización con lo siguiente:

```
#!/bin/bash
echo 'post-receive: Triggered.'
cd /opt/live/my-first-app
echo 'post-receive: git check out...'
git --git-dir=/opt/my-first-app.git --work-tree=/opt/live/my-first-app checkout master -f
echo 'post-receive: npm install...'
npm install \
&& echo 'post-receive: building...' \
&& npm run build \
&& echo 'post-receive: → done.' \
&& (pm2 delete 'my-first-app-by-githook' || true) \
&& pm2 start npm --name 'my-first-app-by-githook' -- start \
&& echo 'post-receive: app started successfully with pm2.'
```

Realizado esto cada vez que ejecutemos git push live master habremos desplegado en nuestro servidor el último cambio.

Es imprescindible testear todo en local comitearlo en Bitbucket y luego desplegar.

## 7. Conclusiones.

### 7.1. Objetivos alcanzados.

La prioridad de este proyecto era desarrollar una aplicación multidispositivo en poco tiempo y que fuera funcional, en una primera instancia los objetivos conseguidos han sido los explicados en el apartado de objetivos propuestos:

- Crear una aplicación multiplataforma que gestione las citas para el salón de tatuaje.
- Crear listas de inventario para revisar los pedidos de stock.
- Ejecución tanto en dispositivos móviles como en ordenadores.
- Acceso a cualquier usuario al portfolio.
- Actualizable, fácil mantenimiento y posicionable.

Así como la totalidad de los objetivos propuestos en un inicio han sido alcanzados, hemos ido modificando en el transcurso del desarrollo algunas necesidades técnicas que quedarían justificadas de la siguiente manera:

- Se ha desestimado la utilización de Firebase para el registro de usuarios, APIS y base de datos no relacional, en favor de MongoDB. Esto se debe a que MongoDB se encuentra integrado en nuestro sistema de cloud y asumimos el control absoluto de la base de datos, mientras que Firebase es un servicio externo, además la gestión de usuarios se ha realizado mediante la librería Passport JS que nos otorga la misma ventaja que MongoDB.
- El uso de Vue JS ha sido mediante la librería Nuxt JS debido a la realización de una Progressive Web App posicionable, y esto solo puede hacerse actualmente mediante el server side rendering y no mediante Single Page Application, que es como Vue JS renderiza en cliente las aplicaciones en el navegador.

- No se ha desarrollado una aplicación nativa de escritorio con Electron JS debido a los plazos de entrega, y puesto que la aplicación es funcional y todos los dispositivos cuentan con un navegador de forma nativa, queda considerado con un requisito futuro y que se realizará en versiones posteriores al desarrollo.

## 7.2. Conclusiones del trabajo.

El producto que se ha realizado ha servido en primer lugar, para aprender metodologías nuevas de desarrollo, esto es, estructurar el trabajo, dividirlo en tareas más o menos independientes y gestionables en un tiempo mínimo determinado cuantitativamente.

Cabe resaltar la circunstancia de proyecto real en sí mismo, con ello nos referimos, a que existe un cliente final real, con un plazo de entrega real y un presupuesto adecuado a una necesidad de mercado.

Estas dos circunstancias anteriores nos llevan a un resultado fruto de la aplicación de ingeniería de software, y nos ayudan a demostrar una destreza técnica y otra empresarial, que es la de gestionar un proyecto con las responsabilidades que ello conlleva.

Además, el aprendizaje de un lenguaje de programación actual, no contemplado, que es uno de los más utilizados a nivel mundial y sobre todo moderno, así como, librerías y frameworks modernos y con un gran potencial productivo.

No todo es un camino de rosas, puesto que en poco tiempo se ha tenido que aprender, Scrum, kanban, Javascript, Mongo DB, Mongoose, sistemas de integración y administración, Nodejs, Express, Passport, Vue JS, Nuxt JS y Vuetify.

El tiempo de aprendizaje, es tiempo invertido que no emplearemos en aprender en ese momento en otras herramientas que por familiaridad nos hubieran hecho el trabajo más llevadero, aunque no por ello más corto.

Para finalizar esta breve experiencia personal, comentar que este desarrollo ha servido para contrastar el potencial que están adquiriendo cada día los navegadores y que poco a poco vemos como son capaces de competir con el desarrollo de aplicaciones nativas,

sobre todo por su versatilidad. Gracias a ello, podemos ver que sistemas como XAMPP y las páginas web pierden su razón de ser ante el potencial de Node JS y Vue JS (React, Angular...) y el resto de frameworks que aún están por llegar.

Si se puede soñar, se puede programar.

### 7.3. Vías futuras.

Poco a poco y según ha ido evolucionando el proyecto, han ido apareciendo nuevas propuestas o futuros requisitos funcionales que nombramos a continuación:

- Sistema de notificación vía email con Nodemailer.
- Sistema de notificación vía Twilio o AWS.
- Integración de pasarela de pago con PayPal para el depósito o fianza.
- Integración de pasarela de pago con Sprite como alternativa a PayPal.
- Integración de Facebook chat mediante su API.
- Generación de contenedores con Docker.
- Generar una agenda de clientes y recopilar datos para realizar estadísticas.
- Campañas de mailing con Mailchimp.
- Crear una tienda dentro del portfolio para venta de productos.

## 8. Anexos.

En este apartado se detallarán el **glosario**, que está compuesto por todos aquellos términos y definiciones que en el documento no se explican con demasiada claridad o superficialmente, la **bibliografía**, donde encontraremos las principales fuentes y referencias de información, y en último lugar **los manuales** tanto de **instalación**, como de **usuario**, donde visualmente se demuestra el uso de la aplicación.

### 8.1. Glosario.

Se definen brevemente algunos de los términos contemplados en el documento, a modo de clarificar posibles dudas o cuestiones surgidas con el significado de dichos términos:

- **User Experience (UX):** La experiencia de usuario es el conjunto de factores y elementos relativos a la interacción del usuario con un entorno o dispositivo concretos, dando como resultado una percepción positiva o negativa de dicho servicio, producto o dispositivo.
- **User interface (UI):** es el espacio donde se producen las interacciones entre seres humanos y máquinas. El objetivo de esta interacción es permitir el funcionamiento y control más efectivo de la máquina desde la interacción con el humano.
- **Framework:** es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Puede incluir soporte de programas, bibliotecas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.
- **Scrum:** es un marco de trabajo para desarrollar, entregar y mantener productos complejos. Permite el trabajo colaborativo entre equipos.
- **Kanban:** sistema de información que controla de modo armónico la fabricación de los productos necesarios en la cantidad y tiempo necesarios en cada uno de los procesos.

- **Scrumban:** metodología que utiliza lo mejor de Scrum y de kanban.
- **Lean:** es un marco de trabajo originado en el Sistema de Producción de Toyota, este método ofrece todo un marco teórico sólido y basado en la experiencia, para las prácticas ágiles de gestión.
- **eXtreme Programming (XP):** es un marco de desarrollo de software ágil que tiene como objetivo producir software de mayor calidad y una mejor calidad de vida para el equipo de desarrollo.
- **Feature-Driven Development (FDD):** es una metodología ágil basada en la calidad y el monitoreo constante del proyecto.
- **Test-Driven Development (TDD):** es un proceso de desarrollo de software que se basa en la repetición de un ciclo de desarrollo muy corto: los requisitos se convierten en casos de prueba muy específicos, luego se mejora el software para pasar las nuevas pruebas.
- **Backlog del producto:** es una lista dinámica de funciones, requisitos, mejoras y correcciones que actúa como la entrada para el backlog de sprint. Es la lista de "cosas por hacer" del equipo. Se está constantemente revisando ya que, a medida que sabemos más o que cambia el mercado, es posible que los elementos ya no sean relevantes o que los problemas se solucionen de otras maneras. Contiene estimaciones realizadas a grandes rasgos, tanto del valor para el negocio, como del esfuerzo de desarrollo requerido.
- **Sprint:** es el periodo real en que el equipo trabaja de forma conjunta para llegar al objetivo. La duración de un sprint suele ser de dos semanas, aunque algunos equipos pueden necesitar menos o más tiempo para la entrega del objetivo.
- **Backlog de sprint:** es la lista de elementos, historias de usuario o correcciones de errores, seleccionadas por el equipo de desarrollo, para su implementación en el ciclo actual de sprint. Tiene un objetivo concreto y aunque el backlog de sprint puede ser flexible y evolucionar no se puede poner en peligro el objetivo.

- **Objetivo del sprint:** es el producto final utilizable de un sprint. A menudo se la conoce como la definición finalizado. Cuanto más se demore el lanzamiento del software, mayor será el riesgo de que el software no cumpla lo que se espera de él.
- **Task:** es un trabajo asignado que debe realizarse en tiempo limitado
- **Workflow:** es el estudio de los aspectos operacionales de una actividad de trabajo; cómo se estructuran las tareas, cómo se realizan, cuál es su orden correlativo, cómo se sincronizan, cómo fluye la información que soporta las tareas y cómo se le hace seguimiento al cumplimiento de las tareas.
- **WIP:** es un estado en kanban que indica que un task o tarea está en proceso de desarrollo.
- **Done:** es un estado en kanban que indica que un task o tarea está finalizada.
- **Verify:** es un estado en kanban que indica que un task o tarea está verificándose para comprobar que la funcionalidad es la esperada.
- **Deploy:** es un estado en kanban que indica que un task o tarea verificada está desplegándose o entregándose al cliente.
- **Ready:** es un estado en kanban que indica que un task o tarea está lista para ser entregada al cliente.
- **Javascript:** es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.
- **Control de versiones:** es la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.

- **Gestor de paquetes:** es una colección de herramientas que sirven para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software.
- **Frontend:** es la parte de un sitio web que interactúa con los usuarios. Es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios.
- **Backend:** es la capa de acceso a datos de un software o cualquier dispositivo, que no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos. El Backend también accede al servidor, que es una aplicación especializada que entiende la forma como el navegador solicita cosas.
- **HTML:** HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web.
- **CSS:** Cascading Style Sheets, es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado.
- **MVVM:** es un patrón de arquitectura de software modelo–vista–modelo de vista. Se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación.
- **ES6:** ECMAScript v6 (ES6) es el estándar que sigue JavaScript desde junio de 2015.
- **Middleware:** es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware o sistemas operativos. Simplifica el trabajo en la tarea de generar conexiones y sincronizaciones necesarias en los sistemas distribuidos.
- **SEO:** es el posicionamiento en buscadores, optimización en motores de búsqueda (Search Engine Optimization), es un conjunto de acciones orientadas a mejorar el

posicionamiento de un sitio web en la lista de resultados de Google, Bing, u otros buscadores de internet.

- **Single Page Application (SPA):** es una aplicación web o es un sitio web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios como una aplicación de escritorio. En un SPA todos los códigos de HTML, JavaScript, y CSS se carga de una vez.
- **Server Side Rendering (SSR):** se basa en la posibilidad de poder renderizar el HTML de nuestros componentes en cadenas de texto en la parte servidor, vez de la parte cliente.
- **Content Delivery Network (CDN):** es una red superpuesta de computadoras que contienen copias de datos, colocados en varios puntos de una red con el fin de maximizar el ancho de banda para el acceso a los datos de clientes por la red.
- **HTTP:** es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.
- **TCP:** es el protocolo de control de transmisión, que garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron.
- **DNS:** es un sistema de nomenclatura jerárquico descentralizado para dispositivos conectados a redes IP como Internet o una red privada.
- **Full Stack:** un desarrollador Full Stack es el encargado de manejar cada uno de los aspectos relacionados con la creación y el mantenimiento de una aplicación web. Para ello el desarrollador tiene que tener conocimientos frontend y backend.
- **SQL:** es un lenguaje específico del dominio utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.

- **NOSQL:** es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico de SGBDR (Sistema de Gestión de Bases de Datos Relacionales).
- **JSON:** es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript.
- **Object Data Modeling (ODM):** es un modelo de datos basado en la programación orientada a objetos, asociando métodos (procedimientos) con objetos que pueden beneficiarse de las jerarquías de clases.
- **OpenID:** es un estándar de identificación digital descentralizado, con el que un usuario puede identificarse en una página web a través de una URL y puede ser verificado por cualquier servidor que soporte el protocolo.
- **OAuth:** es un estándar abierto que permite flujos simples de autorización para sitios web o aplicaciones informáticas. Permite autorización segura de una API de modo estándar y simple para aplicaciones de escritorio, móviles y web.
- **Transcompilador:** es un compilador de traducción que toma el código fuente de un lenguaje de programación como entrada y sale el código fuente a otro lenguaje de programación.
- **Software como servicio:** es un modelo de distribución de software en el que las aplicaciones están alojadas por un proveedor de servicio y puestas a disposición de los usuarios a través de Internet.
- **Plataforma como servicio:** es un conjunto de utilidades que abastecen al usuario de sistemas operativos y servicios asociados a través de Internet sin necesidad de descargas o instalación.
- **Infraestructura como servicio:** hace referencia a la tercerización de los equipos utilizados para apoyar las operaciones, incluido el almacenamiento, hardware, servidores y componentes de red.

- **Proxy:** es un servidor que hace de intermediario en las peticiones de recursos que realiza un cliente a otro servidor.
- **Streaming:** es la distribución digital de contenido multimedia a través de una red de computadoras, de manera que el usuario utiliza el producto a la vez que se descarga.
- **IPV6:** es una versión del Internet Protocol (IP) diseñada para reemplazar a Internet Protocol version 4 (IPv4).
- **Gzip:** es una herramienta de compresión de datos.
- **TLS:** es un protocolo de seguridad de la capa de transporte.
- **SSL:** es un protocolo de seguridad para la capa de puertos seguros.
- **API:** es una interfaz de programación de aplicaciones. Es un conjunto de rutinas que provee acceso a funciones de un determinado software.
- **XAMPP:** es un paquete de software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script PHP y Perl.
- **React JS:** es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre.
- **Angular JS:** es un framework de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador.
- **Twilio:** es una plataforma de desarrollo que permite a los desarrolladores construir aplicaciones de comunicación en la nube y sistemas web. Las API de comunicaciones de Twilio permiten a las empresas proporcionar la experiencia de

comunicación adecuada para sus clientes dentro de la web y las aplicaciones móviles.

- **AWS:** Amazon Web Services es una colección de servicios de computación en la nube pública que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de Internet por Amazon.com.
- **Docker:** Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.
- **Mailchimp:** es un proveedor de servicios de marketing por correo electrónico.

## 8.2. Bibliografía.

- Wikipedia.
  - <https://es.wikipedia.org>
- W3School.
  - <https://www.w3schools.com/js/>
- Atlassian.
  - <https://es.atlassian.com/agile/scrum>
  - <https://es.atlassian.com/agile/kanban>
- Mozilla.
  - [https://developer.mozilla.org/es/docs/Learn/JavaScript/First\\_steps/Understanding\\_JavaScript](https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Understanding_JavaScript)
  - <https://developer.mozilla.org/es/docs/Learn>
  - <https://developer.mozilla.org/es/docs/Web/Reference>
  - <https://developer.mozilla.org/es/docs/Web/API>
- VueJS.
  - <https://vuejs.org/v2/guide/>
  - <https://vuejs.org/v2/guide/comparison.html>
  - <https://vuejs.org/v2/guide/routing.html>
  - <https://vuejs.org/v2/guide/ssr.html>
  - <https://vuejs.org/v2/guide/comparison.html#React>
  - <https://vuejs.org/v2/guide/comparison.html#AngularJS-Angular-1>
  - <https://vuejs.org/v2/guide/comparison.html#Angular-Formerly-known-as-Angular-2>
  - <https://vuejs.org/v2/guide/single-file-components.html>
  - <https://vuejs.org/v2/guide/deployment.html>
  - <https://vuex.vuejs.org/guide/state.html>
  - <https://router.vuejs.org/>
- NuxtJS.
  - <https://nuxtjs.org/guide/installation>

- <https://nuxtjs.org/guide/directory-structure>
  - <https://nuxtjs.org/guide/configuration>
  - <https://nuxtjs.org/guide/routing>
  - <https://nuxtjs.org/guide/views>
  - <https://nuxtjs.org/guide/vuex-store>
  - <https://pwa.nuxtjs.org/>
  - <https://auth.nuxtjs.org/>
- 
- Vuetify.
    - <https://vuetifyjs.com/es-MX/getting-started/quick-start>
- 
- Medium.
    - <https://medium.freecodecamp.org/write-less-do-more-with-javascript-es6-5fd4a8e50ee2>
    - <https://medium.com/@jmz12/patrones-de-dise%C3%B1o-en-js-43beab8f5756>
    - <https://medium.com/gomycode/authentication-with-passport-js-73ca65b25feb>
    - <https://medium.com/@aunnnn/automate-digitalocean-deployment-for-node-js-with-git-and-pm2-67a3cfa7a02b>
    - <https://medium.com/quick-code/building-a-simple-rest-api-introduction-to-nodejs-and-express-fc25daf57baf>
- 
- MongoDB.
    - <https://docs.mongodb.com/manual/tutorial/getting-started/>
    - <https://docs.mongodb.com/manual/installation/>
    - <https://docs.mongodb.com/manual/crud/>
    - <https://mongoosejs.com/docs/guide.html>
- 
- Webpack.
    - <https://webpack.js.org/concepts>
- 
- Hetzner.
    - <https://docs.hetzner.cloud/>

- DigitalOcean.
  - <https://www.digitalocean.com/community/tutorials/how-to-set-up-nginx-with-http-2-support-on-ubuntu-18-04>
  - <https://www.digitalocean.com/community/tutorials/how-to-install-moodle-on-ubuntu-16-04>
  - <https://www.digitalocean.com/community/tutorials/how-to-install-mongodb-on-ubuntu-18-04>
  - <https://www.digitalocean.com/docs/>

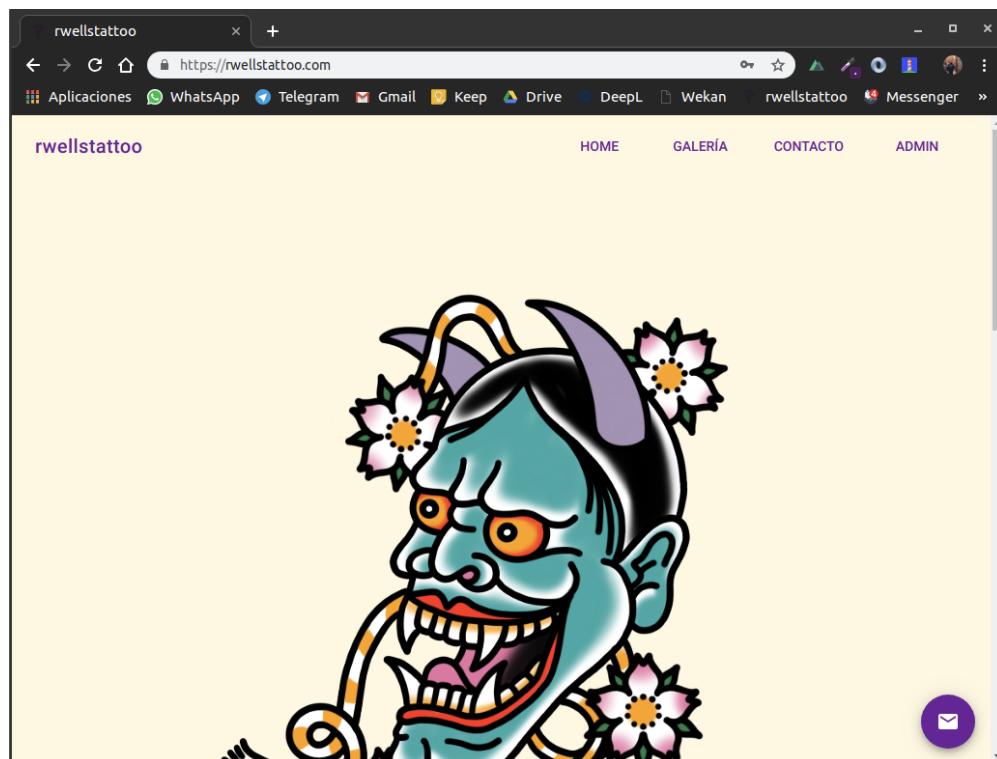
### 8.3. Manual de instalación.

Aclarar que al ser una aplicación web haremos una diferenciación de instalación según el tipo de dispositivo, que podrá ser:

- Equipos de escritorio o con sistemas operativos de escritorio (Windows, OSX, Linux, etc..).

En equipos de escritorio, no necesitaremos una instalación desde ningún paquete puesto que al estar alojada la aplicación en la nube y servida en la dirección <https://rwellstattoo.com> o [www.rwellstattoo.com](http://www.rwellstattoo.com) simplemente deberemos ingresar en nuestra barra de direcciones la dirección anteriormente nombrada para acceder a nuestra aplicación.

Esto es una ventaja ya que nos ahorra costes técnicos y podremos ejecutarla siempre que lo necesitemos sin necesidad de instalación.



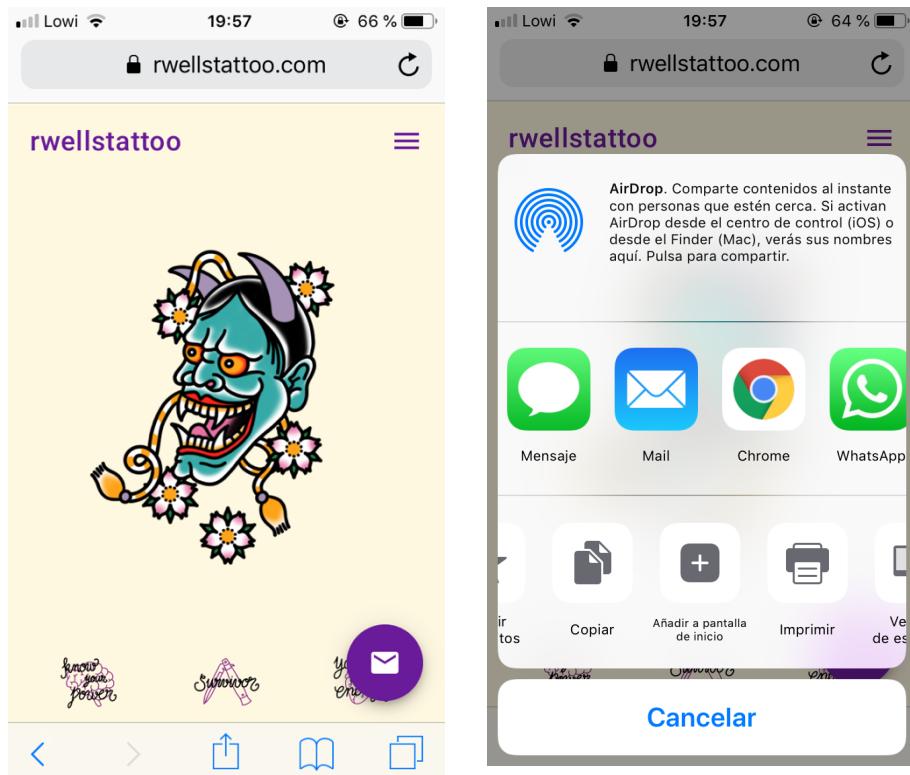
- Dispositivos móviles con sistemas operativos Android o iOS.

En dispositivos móviles sucederá de la misma manera que en equipos de escritorio. No necesitaremos realizar ninguna instalación. Simplemente introduciendo la dirección en nuestro navegador. Salvo con la excepción de que podemos agregar a nuestra pantalla de inicio un acceso directo que nos generará la apariencia de una aplicación nativa suprimiendo las barras de navegación del navegador.

Se explica brevemente como instalar nuestra Progressive Web App en un móvil con iOS; en un dispositivo Android, realizaremos los mismos pasos en el navegador nativo del terminal o a través del navegador Chrome.

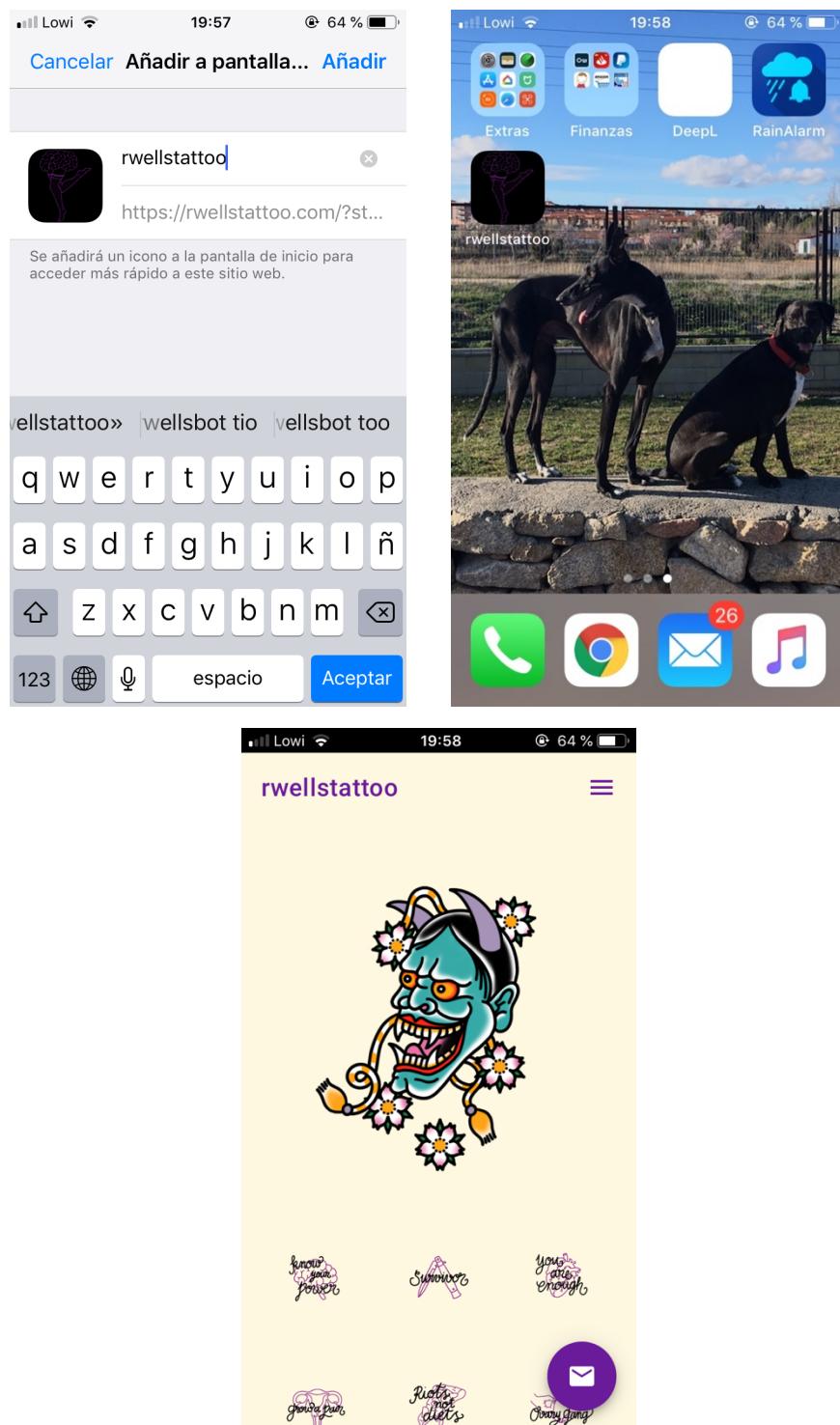
1 - Accedemos mediante Safari a la dirección web <https://rwellstattoo.com> y seleccionamos el ícono de compartir situado en la barra inferior del navegador.

2 - En el submenú de compartir, seleccionamos “Añadir a pantalla de inicio”.



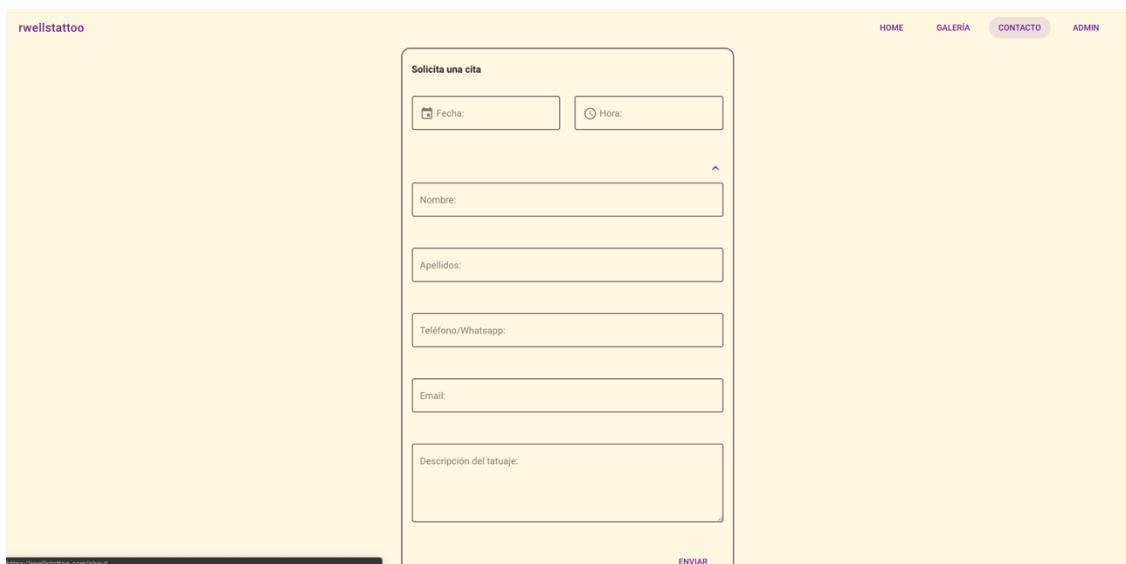
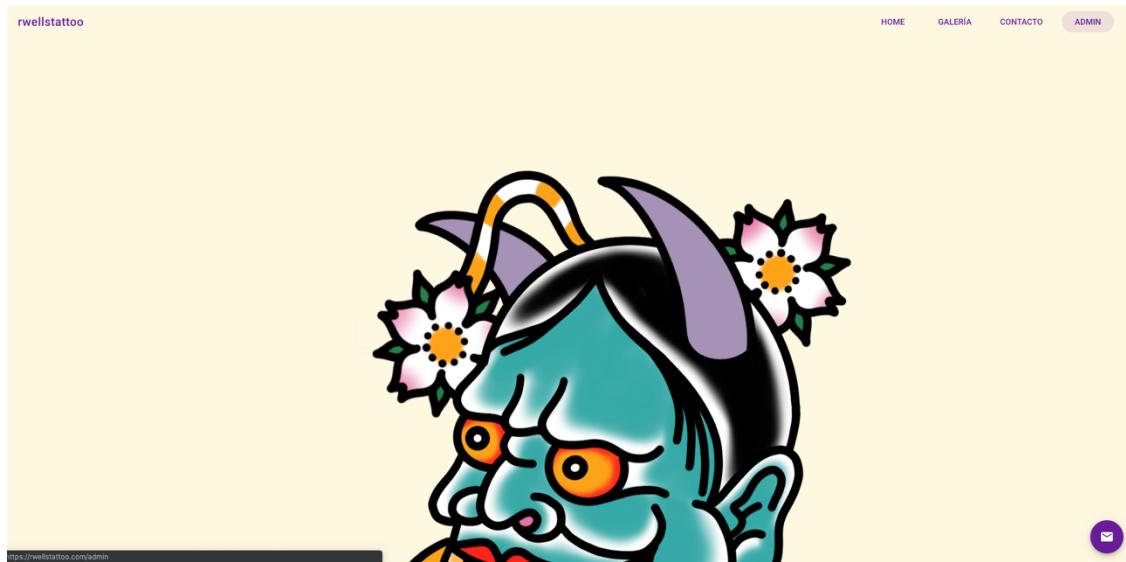
3 - Elegimos el nombre que queramos y le damos a añadir.

4 - Añadida a la pantalla de inicio podremos acceder a nuestra aplicación web como si de una aplicación nativa se tratara.

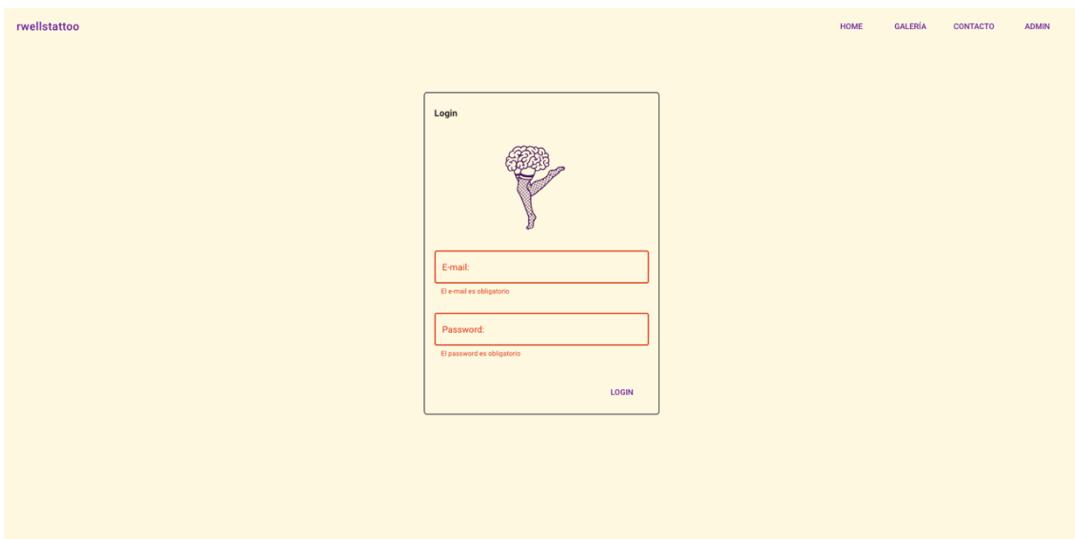


## 8.2. Manual de usuario.

1. Accedemos a nuestro dominio [www.rwellstattoo.com](http://www.rwellstattoo.com) y registramos una cita en el apartado de contacto.



2. Posteriormente ingresamos en nuestro sistema de Login con nuestro usuario y contraseña, y vemos que una cita ha sido creada por un cliente:

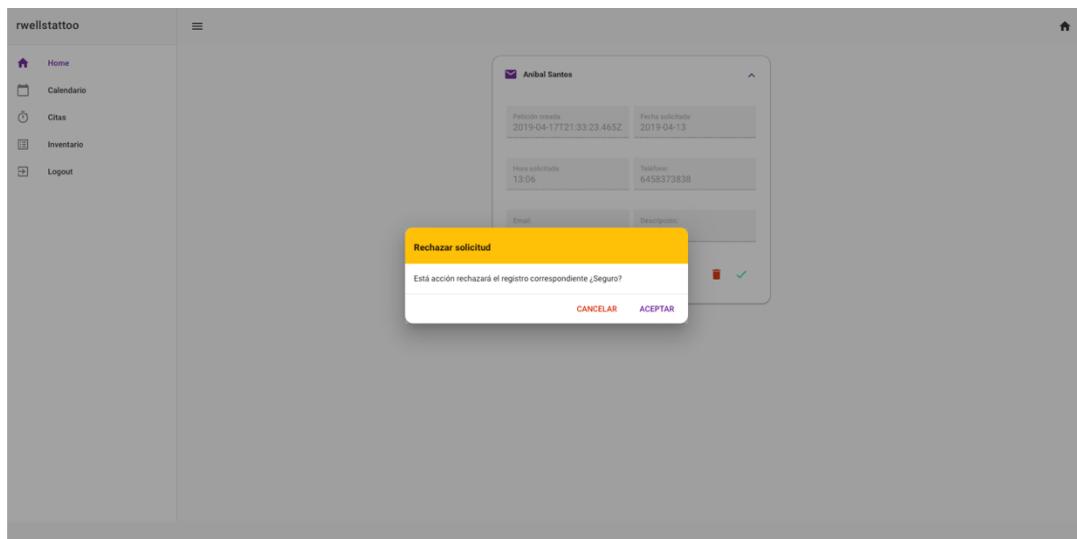


The screenshot shows the 'rwellsstattoo' admin dashboard. On the left, there is a sidebar with navigation links: Home, Calendario, Citas, Inventory, and Logout. The main area displays a modal window for a new appointment. The modal header says 'Aníbal Santos' with a checkmark icon. Inside the modal, there are several input fields and their corresponding values:

- Petición creada: 2019-04-17T21:33:23.465Z
- Fecha solicitada: 2019-04-13
- Hora solicitada: 13:06
- Teléfono: 6458373838
- Email: An@as.es
- Descripción: Andjddjds

A red error icon is visible next to the 'Hora solicitada' field, while a green success icon is next to the 'Email' field. At the bottom right of the modal, there are two buttons: a red one and a green one with a checkmark.

3. O bien la rechazamos o bien la confirmamos:



The screenshot shows a form for creating a new tattoo request. The form fields include: Telefono/WhatsApp: 6458373838; Email: An@as.es; Precio: (empty); Fianza: (empty); Descripción del tatuaje: Andjddjsjs; Estilo: (empty dropdown); Tipo: (empty dropdown). At the bottom of the form are two buttons: "LIMPIAR" (Clear) and "ACEPTAR" (Accept).

4. Confirmada la cita, la aplicación nos redirige a la vista del calendario que además cuenta con una vista alternativa donde podemos listar las citas que tenemos:

This screenshot shows a monthly calendar grid for April and May. The grid is organized by day of the week (Sunday to Saturday) and date (31, Apr 1 to May 4). Each cell in the grid represents a specific day and time slot. Some cells are highlighted in purple, indicating scheduled appointments. The names of the clients are listed within these purple boxes. Navigation buttons for 'PREV' and 'NEXT' are at the bottom, along with a '+' button for adding new events.

Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	1 Apr	2	3	4	5	6
7	8	9	10 Aníbal	11	12 chcfh	13 Aníbal
14	15	16	17 Juan Conchi	18 Juan Alejandra	19	20 Pepe Aníbal
21 Juan	22	23	24 Aníbal	25	26	27
28	29	(20)	1 May Holl	2	3	4

This screenshot shows a list view of scheduled appointments. The left sidebar has navigation links for Home, Calendario, Citas, Inventory, and Logout. The main area lists 15 entries, each with a small red envelope icon and a name. The names listed are Pepe adgsadg, Juan Perez, Juan Perez, Juan Perez, Alejandra Rodriguez, Aníbal Santos, chcfh chg, Pepe P, Conchi PErez, Aníbal Santos, Aníbal Santos, Aníbal Santos, and Holl Yes. Each entry is preceded by a small downward arrow icon, likely for expanding or collapsing details.

✉ Pepe adgsadg
✉ Juan Perez
✉ Juan Perez
✉ Juan Perez
✉ Alejandra Rodriguez
✉ Aníbal Santos
✉ chcfh chg
✉ Pepe P
✉ Conchi PErez
✉ Aníbal Santos
✉ Aníbal Santos
✉ Aníbal Santos
✉ Holl Yes

5. Por último, podemos editar o eliminar las citas y en la barra de navegación de la izquierda podremos seleccionar la vista citas para ver un resumen informativo de cada cita un poco más desglosado.

Editar cita

Fecha: 2019-04-10 | Hora: 22:50 | Fin:

Nombre: Anibal

Apellidos: Santos

Teléfono/WhatsApp: 7892603419325417698

Email: p@p.es

Precio: \_\_\_\_\_ | Fianza: \_\_\_\_\_

Descripción del tatuaje: \_\_\_\_\_

Pepe adgsadg	Juan Perez	Juan Perez
Fecha: 2019-04-20	Fecha: 2019-04-18	Fecha: 2019-04-17
Hora: 13:05	Hora: 13:00	Hora: 13:05
Tatuaje: Asidito	Tatuaje: Black work	Tatuaje: Black work
Tipo: Negro	Tipo: Negro	Tipo: Línea
Precio: 900	Precio: 900	Precio: 900
Fianza: 90	Fianza: 9	Fianza: 9
Total a cobrar: 810	Total a cobrar: 891	Total a cobrar: 891

Juan Perez	Alejandra Rodriguez	Anibal Santos
Fecha: 2019-04-21	Fecha: 2019-04-18	Fecha: 2019-04-24
Hora: 13:05	Hora: 13:05	Hora: 12:20
Tatuaje: Tradicional	Tatuaje: Black work	Tatuaje: Tradicional
Tipo: Línea	Tipo: Negro	Tipo: Color
Precio: 100	Precio: 300	Precio: 300
Fianza: 90	Fianza: 20	Fianza: 50
Total a cobrar: 10	Total a cobrar: 280	Total a cobrar: 250

chcfh chg	Pepe P	Conchi PErez
Fecha: 2019-04-21	Fecha: 2019-04-18	Fecha: 2019-04-24
Hora: 13:05	Hora: 13:05	Hora: 12:20
Tatuaje: Tradicional	Tatuaje: Black work	Tatuaje: Tradicional
Tipo: Línea	Tipo: Negro	Tipo: Color
Precio: 100	Precio: 300	Precio: 300
Fianza: 90	Fianza: 20	Fianza: 50
Total a cobrar: 10	Total a cobrar: 280	Total a cobrar: 250

<https://rwellstattoo.com/admin/stocklist>