



GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FINAL DE GRADO

Herramienta para cambio de estado en reserva de actividades

Autor:

Alejandro RODRÍGUEZ SÁNCHEZ

Supervisor:

Miguel SALAS ALEGRE

Tutor académico:

Jorge SALES GIL

Fecha de lectura: 14 de julio de 2022

Curso académico 2021/2022

Resumen

A lo largo de este documento se detallan el análisis, planificación, diseño e implementación del proyecto realizado en la empresa Easygoband de Castellón. Valga este documento como Trabajo de Fin de Grado del itinerario de Ingeniería del Software del Grado en Ingeniería Informática de la Universitat Jaume I.

El proyecto consiste en la creación de una herramienta para facilitar el cambio de estado de las reservas de actividades de un hotel para que los operarios puedan probar de forma mucho más rápida que las notificaciones que generan estos cambios de estado funcionan de forma adecuada. El proyecto pertenece al ámbito de *front-end*, por tanto es un proyecto centrado más en la parte visual. El lenguaje de programación utilizado para llevar a cabo el proyecto será Vue.js.

Como hemos mencionado anteriormente, se trata de un proyecto que debe facilitar la vida a los trabajadores de la empresa para realizar las pruebas pertinentes con los cambios de estado en las actividades, por eso la solución final deberá ser rápida, muy intuitiva y concisa.

Finalmente, el documento refleja todo lo aprendido en lo referente a la parte de ingeniería del software, esto es, saber redactar una memoria técnica de un proyecto informático, ser capaz de crear un planificación según la metodología y ser capaz de realizar el análisis del sistema.

Palabras clave

Reserva, actividad, estado, paginación

Keywords

Booking, activity, state, pagination

Índice general

Introducción	9
1.1. Contexto y motivación del proyecto	9
1.2. Objetivos del proyecto	10
1.3. Descripción del proyecto	11
1.3.1 Alcance funcional	12
1.3.2 Alcance organizativo	12
1.3.3 Alcance informático	12
1.4. Tecnologías y herramientas	13
1.5. Estructura de la memoria	15
Planificación de proyecto	17
2.1. Metodología	17
2.2. Planificación	18
2.2.1 Análisis de riesgos	20
2.3. Estimación de recursos y costes del proyecto	21
2.4. Seguimiento del proyecto	24
Análisis y diseño del sistema	27
3.1. Análisis del sistema	27
3.1.1 Actores	27

3.1.2 Diagrama de casos de uso	28
3.1.3 Requisitos funcionales	29
3.1.4 Diagrama de clases	34
3.2. Diseño de la arquitectura del sistema	35
3.3. Diseño de la interfaz	37
3.3.1 Guía de estilos	38
Implementación y pruebas	47
4.1. Detalles de la implementación	47
4.2. Estructura del proyecto	48
4.3. Verificación y validación	56
Conclusiones	59
5.1 Conclusión técnica	59
5.2 Conclusión laboral	59
5.3 Conclusión personal	60
Bibliografía	61

Índice de figuras

Figura 1. Logotipo de la empresa	10
Figura 2. Logotipo de la plataforma	11
Figura 3. Vue.js logo	14
Figura 4. Vuetify logo	14
Figura 5. Vuex logo	14
Figura 6. Jira logo	15
Figura 7. GitLab logo	15
Figura 8. VSC logo	15
Figura 9. Swagger logo	15
Figura 10. Figma logo	16
Figura 11. Planificación de actividades del proyecto	20
Figura 12. Diagrama de Gantt	21
Figura 13. Planificación de actividades de seguimiento	26
Figura 14. Diagrama de Gantt de seguimiento	26
Figura 15. Diagrama de casos de uso	29
Figura 16. Diagrama de clases	36
Figura 17. Flux logo	37
Figura 18. Arquitectura Flux	38
Figura 19. Colores de la interfaz	39

Figura 20. Colores para los diferentes estados	39
Figura 21. Elementos utilizados	40
Figura 22. Interfaz listado de reservas	42
Figura 23. Interfaz listado de reservas de actividades	43
Figura 24. Interfaz cambio de estado	44
Figura 25. Seleccionar acción para cambiar estado	45
Figura 26. Completar datos para acción de cambio de estado	46
Figura 27. Alcanzado estado final de una reserva de actividad	47
Figura 28. Estructura de los directorios de la aplicación	50
Figura 29. Directorio public	51
Figura 30. Componentes	53
Figura 31. Directorio config	53
Figura 32. Directorio services	55
Figura 33. Directorio store	56
Figura 34. Directorio views	64

Índice de tablas

Tabla 1. Estimación de costes hardware en euros	21
Tabla 2. Estimación de costes software	22
Tabla 3. Estimación de costes humanos	23
Tabla 4. Estimación de costes extras	23
Tabla 5. Estimación de costes totales	24
Tabla 6. Actores	27
Tabla 7. CU01	29
Tabla 8. CU02	30
Tabla 9. CU03	31
Tabla 10. CU04	32
Tabla 11. CU05	33
Tabla 12. Código del componente Navbar.vue	52
Tabla 13. Código archivo server-directions.js	53
Tabla 14. Ejemplo entidad Room	53
Tabla 15. Ruta de la vista del listado de reservas de actividades	54
Tabla 16. Fichero App.vue	62

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto que se presenta en este documento, se llevó a cabo en la empresa **Easygoband World SL** ([Figura 1](#)) situada en Castellón. Esta empresa proviene de la compañía **Paynopain** y actualmente se encuentra ejerciendo de forma independiente en la ciudad de Castellón de la Plana. Easygoband no solo se encuentra en el territorio nacional, sino que se ha extendido también a Latinoamérica. Es una empresa que pertenece al sector terciario y destaca por la creación de tecnología software y que a día de hoy cuenta con dos plataformas creadas por ellos mismos: Gofun y Goguest. De entre estas dos plataformas nos vamos a fijar en la denominada Goguest, ya que nuestro proyecto se va a centrar en crear una nueva herramienta para dicha plataforma.



Figura 1. Logotipo de la empresa

Goguest ([Figura 2](#)) es una plataforma integradora para hoteles que facilita la interacción con el huésped mediante un sistema único, que además permite la comunicación directa e inmediata con todo el hotel (PMS¹, POS², dispositivos y demás software). Con este servicio, los hoteles que hagan uso de él podrán gestionar de manera sencilla las reservas de habitaciones, los diferentes pagos, podrán controlar los accesos a diferentes áreas y las aperturas de puertas, además de gestionar el préstamo de materiales, cortesías y mucho más. Dentro de esta interacción con el hotel, el huésped tiene la posibilidad de reservar diferentes

¹ *Property Management System*. Es un software para automatizar los servicios de un hotel.

² Point Of Sale.

actividades que el hotel ofrezca, y es en este punto donde empieza nuestro proyecto [1].



Figura 2. Logotipo de la plataforma

La motivación de este proyecto surge ante la necesidad tener una herramienta para facilitar a los desarrolladores el cambio de estado de las reservas de actividades de un hotel para poder probar, de forma más cómoda y directa, las diferentes notificaciones que se emiten ante los diferentes cambios de estados de las reservas de actividades realizadas por los clientes. Esto es fundamental ya que se deben probar muy bien las notificaciones que llegan a los cliente para que puedan estar bien informados en todo momento acerca del estado de la reserva de alguna actividad que quieran realizar.

1.2. Objetivos del proyecto

El proyecto tiene diversos objetivos, pero el principal es conseguir crear una herramienta sencilla y ágil de utilizar para que los desarrolladores de la empresa puedan realizar cambios de estados de las reservas de actividades y así poder probar las notificaciones que estos cambios deben generar. A continuación mostramos un listado con el resto de objetivos a cumplir a lo largo de este proyecto:

- Generar una interfaz que permita buscar una reserva en el hotel.
- Generar una interfaz que permita visualizar las diferentes reservas de actividades pertenecientes a la reserva de hotel seleccionada.
- Generar una interfaz que permita al equipo de desarrollo cambiar los estados de las reservas de forma rápida para que puedan probar las diferentes notificaciones relacionadas con la reserva de actividades que se generan ante los cambios de estado.
- Crear un *storyline* para cada reserva de actividad con cada acción realizada y los estados.

- Indicar en el storyline los diferentes estados por los que pasa una reserva de actividad.
- Se pretende que sea más sencillo para los desarrolladores el poder probar más fácilmente las notificaciones relacionadas con las actividades.

1.3. Descripción del proyecto

El proyecto consiste en crear un herramienta que, mediante el uso de la API (*Application Programming Interface*) de Goguest pueda gestionar el estado de las reservas de actividades de forma que se generen eventos de notificación a través de correo. Una actividad en Goguest representa cualquier cosa que puedes reservar en un hotel como huésped: mesa en el restaurante, sesiones de spa, actividades acuáticas, masajes, gimnasio, etc.

Estas actividades constan de dos métodos de reservas: *in venue* y *room service*. En un principio, en este proyecto nos centraremos en tener una funcionalidad completa respecto a las actividades reservadas por los desarrolladores y que sean de tipo *in venue*. Esto nos permitirá centrarnos en tener al menos el objetivo básico del proyecto. En este proyecto nos vamos a centrar solo en las reservas de actividades de tipo *in venue*, pero en un futuro se podría ampliar la funcionalidad para las de tipo *room service*.

Esta herramienta deberá ser una interfaz que permita seleccionar una reserva de actividad y gestionar los cambios de estados de las propias reservas de una forma rápida y sencilla por parte de los desarrolladores. La herramienta se utilizará para realizar diferentes pruebas con dichas reservas y sus estados para poder comprobar de forma rápida las notificaciones que estos cambios generan. Existen diferentes estados en los que puede estar una reserva: *pending*, *accepted*, *pending modification*, *cancelled*, *rejected*, *attended* y *absent*.

La herramienta debe permitir ir añadiendo diferentes acciones sobre la reserva de actividad para su posterior ejecución de forma secuencial. Se irá creando un *storyline* indicando la acción de cambio de estado que se quiere realizar, el estado en el que queda la reserva y se mandarán ejecutar estos cambios de estados seleccionados.

En lo referente al alcance del proyecto, vamos a analizarlo desde tres puntos de vista: (1) alcance funcional, (2) alcance organizativo y (3) alcance informático.

1.3.1 Alcance funcional

La herramienta debe proporcionar una interfaz gráfica que usarán los desarrolladores para probar las notificaciones para los usuarios de la plataforma al haber modificaciones en sus reservas, por tanto, nos vamos a centrar en la parte del *frontend* y no en la parte de *backend*. Debe ofrecer la siguiente funcionalidad:

1. Buscar y seleccionar una reserva.
2. Seleccionar una reserva de actividad de la reserva anteriormente elegida.
3. Crear acciones de cambio de estado sobre esa reserva de actividad y añadirlas al *storyline* de la reserva.
4. Permitir el cambio del estado de una reserva.
5. Permitir la ejecución de las acciones seleccionadas de forma secuencial.

Cada vez que hay un cambio de estado, se activa un trigger para que se produzcan las notificaciones. Generar las notificaciones no entra dentro de nuestro alcance ya que es algo que está implementado, pero es una consecuencia de los cambios de estados que sí realizamos nosotros.

1.3.2 Alcance organizativo

Goguest cuenta con un grupo de personas dedicado al *backend* y otro grupo de personas centradas en la parte *frontend*. Como ya hemos mencionado anteriormente, nuestro proyecto pertenece al ámbito del frontend en cuanto al desarrollo. Pero una vez finalizado, podrán hacer uso de la herramienta tanto el grupo de desarrolladores de la parte *backend* como de la parte *frontend* y cualquier otra persona de la organización que necesite hacer uso de la herramienta, en caso de querer probar cualquier acción relacionada con la generación de reservas de actividades y notificaciones de cambios de estado de las mismas.

1.3.3 Alcance informático

La herramienta hará uso y se comunicará con la API que nos proporciona la empresa para poder gestionar los cambios de estados de reservas de actividades.

1.4. Tecnologías y herramientas

En este apartado se van a enumerar y comentar las tecnologías y herramientas utilizadas a lo largo del desarrollo del proyecto.



Figura 3. Vue.js logo

Vue.js ([Figura 3](#)): es un framework progresivo³ de JavaScript para construir interfaces de usuario. Vue permite la creación de *Single-Page-Applications* de una forma muy sencilla cuando se utiliza en combinación con librerías de apoyo. El framework nos permite manejar los datos de la página de forma sencilla aportando así mucha flexibilidad [\[2\]](#).



Figura 4. Vuetify logo

Vuetify ([Figura 4](#)): Vuetify es un framework que se puede utilizar junto con Vue.js para poder acelerar el desarrollo de aplicaciones web, ya que incorpora una gran cantidad de componentes prediseñados y listos para usar. Vuetify otorga una estética del famoso Material Design [\[3\]](#) [\[4\]](#) [\[5\]](#).



Figura 5. Vuex logo

Vuex ([Figura 5](#)): es una librería que nos permite realizar gestión del estado (State Management) de aplicaciones Vue.js. Sirve como una especie de almacén común a todos los componentes de una aplicación Vue, con reglas que aseguran que el estado se puede alterar según convenga de forma segura [\[6\]](#) [\[7\]](#).

³ Un framework progresivo puede tener muchas características disponibles para usar pero te permite que uses solo las que realmente necesitas sin necesidad de utilizar todas. Son framework versátiles ya que te permiten desarrollar soluciones simples o soluciones muy complejas.



Figura 6. Jira logo

Jira ([Figura 6](#)): Es una herramienta de gestión de trabajo. Jira se diseñó en su origen para gestionar incidencias y errores. Pero en los últimos tiempos se ha convertido en una herramienta de gestión de trabajo para todo tipo de casos de uso, gestión de requisitos y casos de prueba hasta el desarrollo de software ágil. También sirve para mantener un seguimiento de las horas empleadas en cada tarea creada [\[8\]](#).



Figura 7. GitLab logo

GitLab ([Figura 7](#)): es una herramienta de ciclo de vida y repositorio de Git. Es una plataforma que permite a los desarrolladores gestionar y realizar diversas tareas del proyecto. Las tareas incluyen la planificación del proyecto, la gestión del código fuente, el mantenimiento de la seguridad, la corrección y el seguimiento. Se pueden crear diferentes ramas para cada nueva tarea del proyecto de este modo se pueden evitar errores en la producción de código. Una vez acaba dicha tarea si todo es correcto puedes juntar esa rama auxiliar a la principal y de esta forma crece el proyecto [\[9\]](#).



Figura 8. VSC logo

Visual Studio Code ([Figura 8](#)): es un editor de código gratuito desarrollado por Microsoft el cual se adapta perfectamente a las características del framework Vue.js ya que se basa en JavaScript que es el lenguaje de programación más utilizado en Visual Studio Code [\[10\]](#).



Figura 9. Swagger logo

Swagger ([Figura 9](#)): Es una herramienta que nos permite documentar una API. Nos permite crear una documentación sencilla y de forma rápida de una API que todo el mundo podría entender y, además, con Swagger la documentación puede utilizarse directamente para automatizar procesos dependientes de APIs [\[11\]](#).



Figura 10. Figma logo

Figma ([Figura 10](#)): es una herramienta que se utiliza para la creación de prototipos principalmente web aunque también permite crear para móviles. Permite la opción de compartir con varias personas el mismo diseño por tanto permite el trabajo cooperativo [\[12\]](#).

1.5. Estructura de la memoria

Para la mejor comprensión de la organización de este proyecto voy a proceder a realizar una breve explicación de la estructura de esta memoria.

En primer lugar, el capítulo 2 hace referencia a la planificación del proyecto. Explica la información relacionada con la metodología utilizada, la estimación de recursos y costes del proyecto y cuál ha sido el seguimiento del proyecto.

En segundo lugar, el capítulo 3 habla sobre el análisis y el diseño de la herramienta creada, diferenciando entre la arquitectura del sistema y la interfaz del sistema.

A continuación, en el capítulo 4 se detalla toda la implementación de la herramienta y los procesos de validación y verificación.

Finalmente, en el capítulo 5, el documento cerrará con las conclusiones obtenidas a lo largo del proceso de desarrollo del proyecto, así como un conjunto de anexos relacionados y la bibliografía que se ha consultado para desarrollar el proyecto.

Capítulo 2

Planificación de proyecto

2.1. Metodología

Vamos a utilizar una metodología PMBOK (Project Management Body of Knowledge [13]) ya que se trata de un proyecto que sigue un ciclo de vida en cascada. En Ingeniería de software, el desarrollo en cascada, también llamado secuencial, es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior [14]. PMBOK se basa en definir los diferentes grupos de procesos: inicio, planificación, ejecución, seguimiento y control y cierre. Es útil utilizar PMBOK ya que es un estándar orientado a procesos. Indica el conocimiento necesario para manejar el ciclo de vida de cualquier proyecto a través de esos procesos.

Dentro de esta metodología PMBOK vamos a utilizar la metodología ágil SCRUM⁴ [15], por tanto, podemos decir que vamos a utilizar las dos metodologías de forma conjunta y complementaria. La metodología SCRUM no se va aplicar al 100%, ya que en nuestro caso no vamos a basarnos en *sprints*⁵, pero sí vamos a tener breves reuniones de forma diaria, siempre a la misma hora (popularmente conocidas como *dailies*), donde comentaremos el trabajo que hemos realizado el día anterior, el trabajo que tenemos pensado realizar para ese día, y si nos ha surgido alguna problema y si tenemos alguna duda. En caso de existir un problema o duda, simplemente procedemos a resolverla en ese *daily* en caso de que lleve pocos minutos. En el caso de que sea una duda más compleja, que requiere más tiempo su explicación, procedemos a programar una reunión con la líder de *front-end* cuando tuviese disponibilidad para poder tratar esa duda o problema.

Aunque no utilicemos la metodología SCRUM directamente, esta metodología necesita de tres roles bien definidos para que se aplique de forma correcta, los cuales son los siguientes:

⁴SCRUM es un marco de trabajo para desarrollo ágil de software.

⁵Un *sprint* es la unidad básica de desarrollo en Scrum, un esfuerzo limitado en el tiempo y que normalmente oscila entre una semana y un mes, siendo dos semanas lo más común.

- **Product Owner:** es el encargado de asegurarse de que el equipo trabaje con el proyecto de forma correcta desde el punto de vista de negocio. En nuestro caso es el supervisor.
- **Scrum Master:** es el encargado de organizar el equipo y gestionar los problemas que puedan surgir a lo largo del desarrollo. En nuestro caso sería la líder de *front-end*.
- **Scrum Team:** son los trabajadores que se encargan de la implementación del proyecto. En nuestro caso seríamos nosotros mismos.

Aparte de los tres roles anteriores, existen roles auxiliares como los *stakeholders*, ya que, aunque no se involucran directamente en el proceso, hay que tenerlos en cuenta ya que son los que hacen posible el proyecto y quieren obtener un beneficio concreto.

2.2. Planificación

Para la planificación del proyecto vamos a seguir una metodología de planificación predictiva [\[16\]](#), la cual consiste en ir indicando las diferentes fases y tareas que forman el proyecto. A estas tareas se les asignará una duración estimada y también se indicará qué tareas son predecesoras de otras como se puede ver en la [Figura 11](#). De esta forma se podrá saber qué tareas pueden retrasarse o no y qué tareas necesitan que otras hayan finalizado antes de poder empezar. Finalmente obtendremos un diagrama de Gantt en el que quedarán reflejadas las tareas y su organización a lo largo del tiempo hasta la fecha final del proyecto como se puede ver en la [Figura 12](#).

























		Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1			▾ Proyecto	65 días	22/02/22	23/05/22	
2			▾ Inicio	8 días	22/02/22	03/03/22	
3			Descripción y alcance	5 días	22/02/22	28/02/22	
4			Definición bjetivos	1 día	01/03/22	01/03/22	3
5			Definir endpoints	2 días	02/03/22	03/03/22	4
6			▾ Planificacion	5 días	04/03/22	10/03/22	2
7			Definición actividades	2 días	04/03/22	07/03/22	
8			Gestion tiempo	1 día	08/03/22	08/03/22	7
9			Gestión recursos	1 día	09/03/22	09/03/22	8
10			Gestión de costes	1 día	10/03/22	10/03/22	9
11			▾ Modelización	3 días	11/03/22	15/03/22	6
12			Definición requisitos	3 días	11/03/22	15/03/22	
13			Wireframes	2 días	11/03/22	14/03/22	
14			▾ Ejecución	45 días	16/03/22	17/05/22	11
15			Familiarizarse con Vue JS	5 días	16/03/22	22/03/22	
16			Definición componentes	3 días	23/03/22	25/03/22	15
17			Creación componentes	33 días	28/03/22	11/05/22	16
18			Pruebas unitarias, aceptación e integración	4 días	12/05/22	17/05/22	17
19			Seguimiento y control	1 día	18/05/22	18/05/22	14
20			▾ Cierre	3 días	19/05/22	23/05/22	19
21			Evaluar resultados	3 días	19/05/22	23/05/22	
22			▾ Dailys	64,03 días	22/02/22	23/05/22	

Figura 11. Planificación de actividades del proyecto

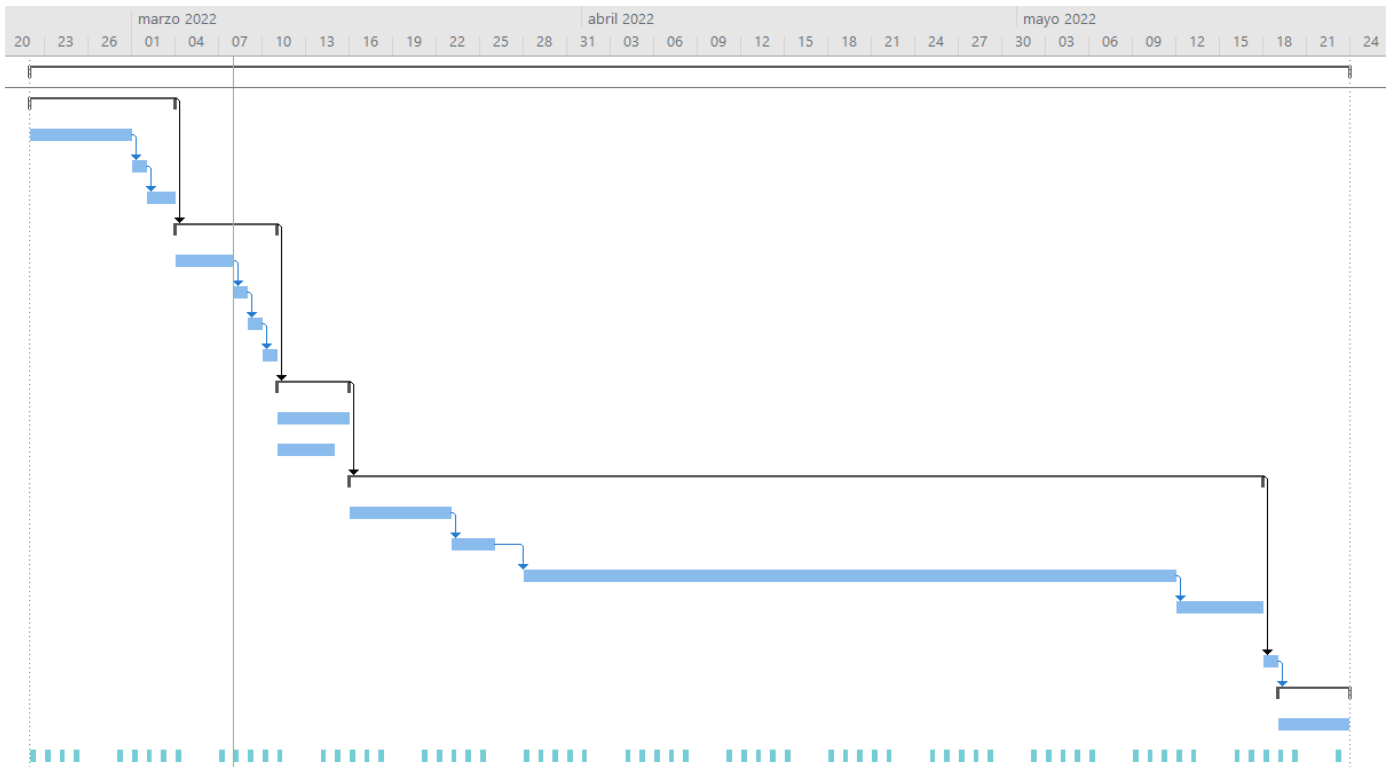


Figura 12. Diagrama de Gantt

2.2.1 Análisis de riesgos

En este apartado de análisis de riesgos se estudian los posibles sucesos o amenazas no deseadas que pueden surgir durante el transcurso del proyecto y que pueden afectar a su duración. A continuación se exponen los riesgos que podrían surgir durante el proyecto:

- **Falta de experiencia con las tecnologías empleadas:** Debido a que era la primera vez que iba a utilizar el framework Vue.js, una biblioteca UI (User Interface) de Vue conocida como Vuetify y también Vuex, que es una librería oficial de Vue que sirve para controlar el estado de la aplicación, podrían surgir ciertos retrasos. Como todo era nuevo para mí, se corre un riesgo de que pueda faltar tiempo, ya que al principio voy a tener que dedicar varias horas a aprender como funciona todo. Desde la empresa nos facilitaron unos vídeos de YouTube donde, en unos tutoriales, explicaban las cosas más importantes que debía aprender. También se puede perder tiempo porque cada vez que surja un problema me va a costar más saber cómo resolverlo.

- **Problemas en el entendimiento del proyecto:** Es un riesgo que quizás es más difícil que se dé en un proyecto de este tamaño pero está ahí. Puede que por alguna explicación mal entendida sobre funcionalidad o diseño, esto haga que tengas que hacer pequeñas rectificaciones que te roben un poco de tiempo y te retrasen más de lo que esperas.

2.3. Estimación de recursos y costes del proyecto

Recursos hardware

En cuanto al hardware, nos encontramos que hemos utilizado un portátil ASUS ZenBook, un ratón inalámbrico y un monitor HP, con un coste total de unos 700€. Dicho hardware tiene una duración aproximada de 5 años (1.825 días). A continuación tenemos la [Tabla 1](#) con todo el desglose de precios.

Recurso	Precio	Duración aproximada	Coste por día	Días	Coste total
Portatil	500 €	1.825 días	0,27 €	65	17,55 €
Monitor	180 €	1.825 días	0,10 €	65	6'50 €
Ratón	20 €	1.825 días	0,01 €	65	0'65 €
					24,70 €

Tabla 1. Estimación de costes hardware en euros.

Recursos software

En recursos software encontramos principalmente las herramientas software de las cuales hemos hecho uso a lo largo del proyecto ([Tabla 2](#)). Tenemos tanto herramientas en su versión gratuita como en versiones de pago.

Recurso	Tarifa	Precio mes	Meses	Total
GitLab	Gratuita	0 €	3	0 €
Visual Studio Code	Gratuita	0 €	3	0 €
Figma	Professional	12 €	3	36 €
Jira	Premium	7,50 €	3	43,50 €
				79,50 €

Tabla 2. Estimación de costes software

Recursos Humanos

Respecto a los costes de recursos humanos, tenemos diversos cálculos. En primer lugar tendríamos el coste del programador junior. Para calcular dicho coste nos hace falta saber el número de horas total trabajadas, que en este caso serían unas 325 horas, y el precio por hora, que en este caso el precio hora oscila entre los 8/9€ según [\[17\]](#). Esto supondría un total de alrededor de 2.762,5 €.

Debemos tener en cuenta que la supervisora de *front-end* (programadora senior) revisará el proyecto y dedicará cerca del 15% del tiempo de las horas totales trabajadas. Ese tiempo sale de todos los *dailies* diarios, reuniones extras y tiempo empleado para revisar y comentar el código. El 15% de esas 325 horas totales nos dejaría con 48,75 horas, que multiplicadas por el precio hora de la supervisora (de 12 a 15 euros) quedaría un total de entre 390€ y 487'50€. [\[18\]](#)

En la [Tabla 3](#) podemos ver un desglose de estos gastos en recursos humanos.

Recurso	Salario bruto anual	Salario neto mensual	Salario por hora	Horas	Salario Total
Programador Junior	19.056 €	1.310,10 €	8'50 €	325	2.762,50 €
Programador Senior	37.555 €	2.342,50 €	15 €	48'75	721,25 €
					3.483,75 €

Tabla 3. Estimación de costes humanos

Además de estos gastos de recursos humanos, debemos también tener en cuenta: gastos de seguridad social, gastos de oficina, internet, etc. Por estos gastos, siguiendo criterios de la empresa, se suele contar un 20% para gastos de contratación y otro 20% para gastos indirectos ([Tabla 4](#)).

Recurso	Coste	Porcentaje	Total
Coste contratación	3.483,75 €	20%	696,75 €
Costes indirectos	3.483,75 €	20%	696'75 €
			1.393,50 €

Tabla 4. Estimación de costes extras

En la [Tabla 5](#) nos encontramos la suma total de todos los costes:

Recursos	Coste Total
Costes hardware	24,70 €
Costes software	79,50 €
Costes humanos	3.483,75 €
Costes extras	1.393,50 €
4.981,45 €	

Tabla 5. Estimación de costes totales

El coste final del proyecto se estima en 4.981,45 €.

2.4. Seguimiento del proyecto

En este apartado se va a explicar el seguimiento que ha tenido el proyecto a lo largo de la estancia en la empresa con respecto a la planificación inicial, se van a comentar los problemas surgidos que han hecho que la planificación haya sufrido ciertas modificaciones.

Como hemos comentado antes, dentro de la metodología predictiva utilizamos la parte SCRUM de tener reuniones diarias de pequeñas duración. De este modo se ha ido haciendo el seguimiento. Íbamos solucionando poco a poco los errores y dudas. Si era necesario sí que se hacían reuniones más largas y específicas.

Las primeras partes de “Inicio” y “Planificación” se desarrollaron sin problemas. En primer lugar realizamos las actividades de la parte de “Inicio” ya que lo más idóneo es empezar por entender el proyecto entero y los objetivos que este persigue. Una vez definido todo esto realizamos la parte de la “Planificación”. Aquí nos centramos en crear las diferentes fases y actividades que tendríamos que hacer a lo largo de todo el proyecto. Les asignamos una duración estimada a cada una.

Al principio de la estancia en prácticas es necesario que entreguemos una propuesta técnica acerca del proyecto donde se nos pide exactamente lo explicado en el párrafo anterior, por eso fueron las dos fases que se realizaron nada más empezar la estancia en prácticas.

Una vez entregada la propuesta técnica y habiendo finalizado las primera dos fases, pasamos a la tercera fase denominada “Modelización”. Aquí ya empezamos a profundizar en la funcionalidad que debía ofrecer la herramienta final que íbamos a implementar. Se empezaron a definir algunos casos de uso del sistema a la vez que hacíamos unos *wireframes* sobre cómo pensábamos que tendrían que ser las interfaces. Más adelante la propia empresa nos facilitó los *wireframes* que ellos mismos tenían para que el diseño fuese exactamente igual que lo que ellos querían para esta herramienta.

Una vez habíamos pensado ya un poco en algunas partes de la funcionalidad de la herramienta, pasamos a la parte de ejecución donde lo primero que realizaríamos sería unos cuantos días de aprendizaje sobre las partes más básicas de Vue.js, Vuetify y Vuex mediante unos vídeos de YouTube. Aprendimos lo mínimo necesario

que nos indicaron desde la empresa para poder empezar ya con el proyecto en el ámbito de la programación.

Una vez terminado el aprendizaje vinieron los primeros problemas al empezar la implementación del proyecto ya que tuvimos muchos inconvenientes con la nueva versión de Vuetify. Una vez solucionado esos problemas con la versión pasamos a probar la conexión con la API. Después de eso nos volvimos a retrasar un poco ya que los wireframes, proporcionados por la empresa y que debíamos seguir, se retrasaron un poco por tanto no podía saber muy bien por donde empezar ya que al ser un proyecto front-end es bastante importante conocer el diseño de la interfaz.

Una vez disponibles esos *wireframe* ya empezamos a crear la primera vista de la herramienta sin problemas hasta que llegamos a la parte relacionada con los índices de paginación, los cuales explicaremos más adelante. Esto nos retrasó bastante pero una vez solucionado ya todo fue mucho más rápido y fácil hasta prácticamente el final.

Cerca del final de completar las horas totales y la implementación surgió un problema ya que hubo un malentendido entre un requisito y lo que hice. Me tocó cambiarlo y para asegurarnos de que podíamos llegar al final del proyecto y tener algo funcional decidimos, por mutuo acuerdo, entre la empresa y yo, alargar unos cuantos días más la estancia y el proyecto.

Finalmente el proyecto se quedó muy cerca de completarse al 100% pero hubo pequeños detalles que no dio tiempo a implementar. A continuación, en la [Figura 13](#) y la [Figura 14](#) se muestra como quedó finalmente la planificación de actividades y el diagrama de Gantt final teniendo en cuenta los cambios que surgieron durante el desarrollo del proyecto.

	i	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1			▾ Proyecto	73 días	22/02/22	02/06/22	
2			▾ Inicio	8 días	22/02/22	03/03/22	
3			Descripción y alcance	5 días	22/02/22	28/02/22	
4			Definición bjetivos	1 día	01/03/22	01/03/22	3
5			Definir endpoints	2 días	02/03/22	03/03/22	4
6			▾ Planificación	5 días	04/03/22	10/03/22	2
7			Definición actividades	2 días	04/03/22	07/03/22	
8			Gestion tiempo	1 día	08/03/22	08/03/22	7
9			Gestión recursos	1 día	09/03/22	09/03/22	8
10			Gestión de costes	1 día	10/03/22	10/03/22	9
11			▾ Modelización	3 días	11/03/22	15/03/22	6
12			Definición requisitos	3 días	11/03/22	15/03/22	
13			Wireframes	2 días	11/03/22	14/03/22	
14			▾ Ejecución	53 días	16/03/22	27/05/22	11
15			Familiarizarse con Vue JS	8 días	16/03/22	25/03/22	
16			Definición componentes	3 días	28/03/22	30/03/22	15
17			Creación componentes	42 días	31/03/22	27/05/22	16
18			Seguimiento y control	3 días	30/05/22	01/06/22	14
19			▾ Cierre	1 día	02/06/22	02/06/22	18
20			Evaluar resultados	1 día	02/06/22	02/06/22	
21			▸ Dailys	72,03 días	22/02/22	02/06/22	

Figura 13. Planificación de actividades de seguimiento

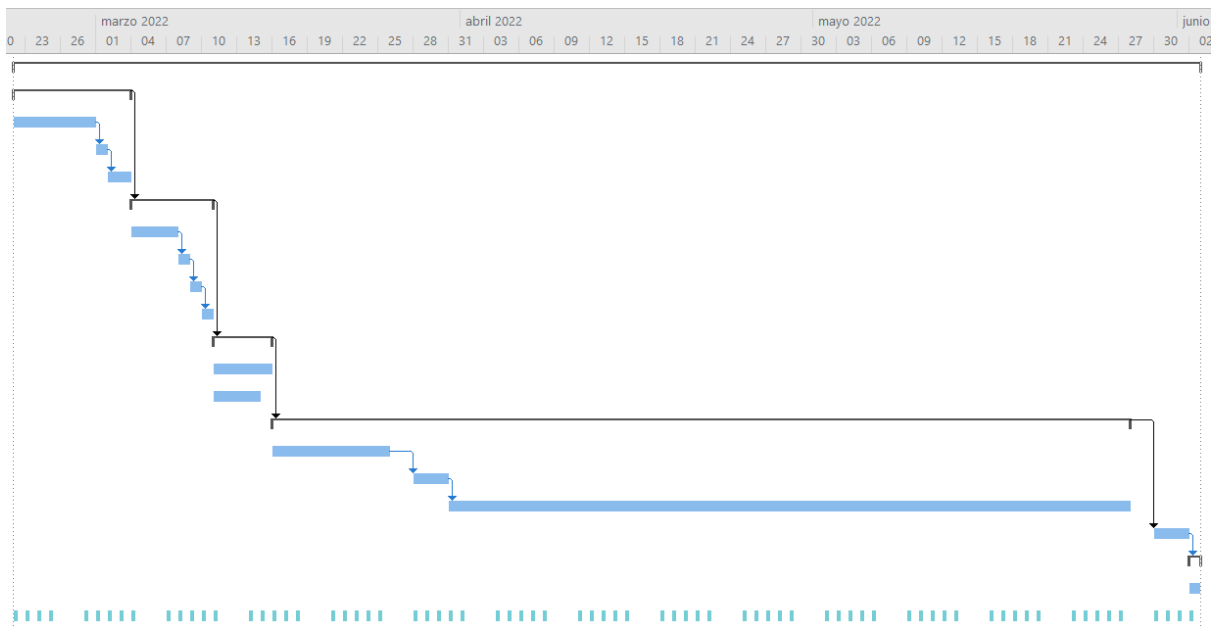


Figura 14. Diagrama de Gantt de seguimiento

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

En este capítulo se va a realizar un análisis detallado del sistema a implementar. La explicación se llevó a cabo mediante el uso de técnicas y diagramas UML (*Unified Modeling Language* [19]) que permiten una mejor comprensión de los requisitos del sistema. Las técnicas más populares y que vamos a ver a continuación son los diagramas de casos de uso, de actividades y de clases.

3.1.1 Actores

En primer lugar debemos definir los actores o actor principal que tendrá el sistema para poder identificar los requisitos y modelar el sistema. En nuestro caso nos encontramos con que nuestro sistema solo tiene un actor tal y como se muestra en la [Tabla 6](#).

Identificador	Actor	Descripción
AC-01	Desarrollador	Este actor representa a cualquier persona de la empresa relacionada con la parte de desarrollo de software de la empresa.

Tabla 6. Actores

3.1.2 Diagrama de casos de uso

El diagrama de casos de uso sirve para representar las diferentes situaciones que se podrán dar en la herramienta. En este caso se representan las acciones que un desarrollador podrá seguir para poder cambiar los estados de una reserva de actividad teniendo en cuenta los requisitos del sistema. A continuación, tenemos la [Figura 15](#) donde podemos ver el diagrama de casos de uso.

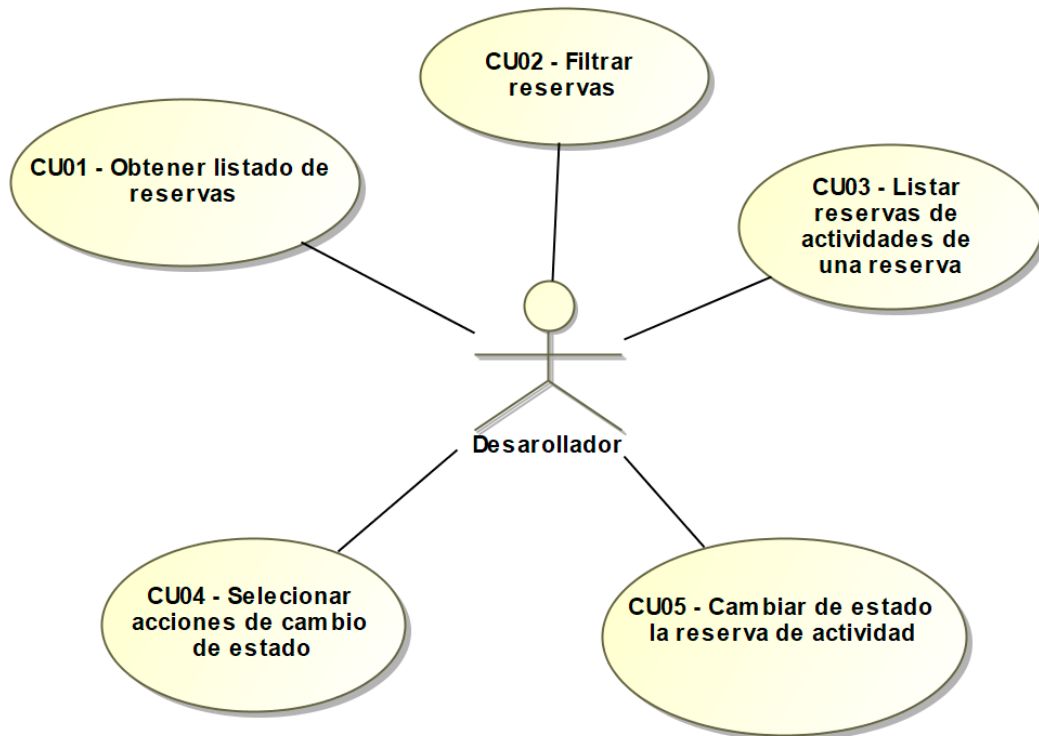


Figura 15. Diagrama de casos de uso

3.1.3 Requisitos funcionales

En este apartado se va a proceder a explicar cada uno de los casos de usos del diagrama de casos de uso anterior ([Figura 15](#)). Cada caso de uso representa un requisito funcional que debe cumplir el sistema. [Tabla 7](#) [Tabla 8](#) [Tabla 9](#) [Tabla 10](#) [Tabla 11](#)

Requisito funcional	
Identificador	CU-01
Nombre	Obtener listado de reservas
Fecha creación	15/03/2022
Autores	Alejandro Rodríguez Sánchez
Versión	1
Actor principal	AC-01
Descripción	La aplicación debe permitir al desarrollador poder acceder al listado de las reservas (bookings).
Relaciones	-
Precondición	Haber iniciado sesión en el panel
Disparador	Abrir la extensión de cambio de estado.
Secuencia normal	<ol style="list-style-type: none">1. El desarrollador accede la extensión2. El sistema abre la extensión3. El sistema pide a la API las reservas4. El sistema muestra el listado de reservas
Secuencia alternativa	-
Excepciones	Si no hay conexión a internet o la API está caída.

Tabla 7. CU01

Requisito funcional	
Identificador	CU-02
Nombre	Filtrar reservas
Fecha creación	15/03/2022
Autores	Alejandro Rodríguez Sánchez
Versión	1
Actor principal	AC-01
Descripción	La aplicación debe permitir al desarrollador filtrar las reservas por diferentes parámetros (Instalación, tipo reserva, checkin, checkout, activas, nombre)
Relaciones	-
Precondición	Estar dentro de la extensión y en la interfaz del listado de reservas
Disparador	Activar un filtro
Secuencia normal	<ol style="list-style-type: none"> 1. El desarrollador selecciona el filtro que quiere aplicar 2. El sistema hace la llamada a la API y devuelve las reservas filtradas
Secuencia alternativa	<ol style="list-style-type: none"> 1. El desarrollador selecciona los filtros que quiere aplicar 2. El sistema hace la llamada a la API y devuelve las reservas filtradas por los diferentes parámetros
Excepciones	Si no hay conexión a internet o la API está caída.

Tabla 8. CU02

Requisito funcional	
Identificador	CU-03
Nombre	Listar reservas de actividades de una reserva
Fecha creación	15/03/2022
Autores	Alejandro Rodríguez Sánchez
Versión	1
Actor principal	AC-01
Descripción	La aplicación debe permitir al desarrollador obtener el listado de actividades reservadas asociadas a una reserva
Relaciones	-
Precondición	Estar dentro de la extensión y en la interfaz del listado de reservas
Disparador	Seleccionar un reserva
Secuencia normal	<ol style="list-style-type: none"> 1. El desarrollador selecciona una reserva 2. El sistema comprueba si la reserva tiene asociado un <i>webapp user</i> 3. El sistema abre la vista si la reserva tenía asociado un <i>webapp user</i> 4. El sistema llama a la API para obtener los datos de las reservas de actividades de la reserva seleccionada 5. El sistema muestra los datos.
Secuencia alternativa	-
Excepciones	Si la reserva seleccionada no tiene <i>webbapp user</i> no se abre la vista. Si no hay conexión con la API no se cargan los datos.

Tabla 9. CU03

Requisito funcional	
Identificador	CU-04
Nombre	Seleccionar acciones de cambio de estado
Fecha creación	15/03/2022
Autores	Alejandro Rodríguez Sánchez
Versión	1
Actor principal	AC-01
Descripción	La aplicación debe permitir al desarrollador seleccionar las diferentes acciones de cambio de estado que quiere realizar para posteriormente ejecutar esas acciones
Relaciones	-
Precondición	Estar dentro de la extensión y en la interfaz del listado de reservas de actividades
Disparador	Seleccionar una reserva de actividad
Secuencia normal	<ol style="list-style-type: none"> 1. El desarrollador selecciona una reserva de actividad 2. El sistema abre la vista correspondiente 3. El desarrollador elige las diferentes acciones de cambio de estado que quiere que se realicen. 4. El sistema almacena las acciones seleccionadas
Secuencia alternativa	-
Excepciones	Si no hay conexión a internet o la API está caída.

Tabla 10. CU04

Requisito funcional	
Identificador	CU-05
Nombre	Cambiar el estado de la reserva de una actividad
Fecha creación	15/03/2022
Autores	Alejandro Rodríguez Sánchez
Versión	1
Actor principal	AC-01
Descripción	La aplicación debe permitir al desarrollador indicar que quiere que se inicie el proceso de ejecución de los cambios de estado que ha seleccionado.
Relaciones	-
Precondición	Haber seleccionado una reserva de actividad y estar dentro de la vista de cambio de estado.
Disparador	Indicar que se quieren ejecutar las acciones seleccionadas
Secuencia normal	<ol style="list-style-type: none"> 1. El desarrollador indica que quiere que se ejecuten las acciones que ha elegido previamente 2. El sistema ejecuta todas las acciones almacenadas de forma secuencial. 3. El sistema muestra que todo ha ido correctamente y cierra la extensión
Secuencia alternativa	-
Excepciones	Si no hay conexión a internet o la API está caída.

Tabla 11. CU05

3.1.4 Diagrama de clases

Un diagrama de clases UML [\[20\]](#) es un diagrama de estructura, por tanto, se utiliza para representar los diferentes elementos que componen un sistema de información desde un punto de vista estático. En los diagramas de clases UML se representan las diferentes clases, sus atributos, las operaciones y las relaciones entre clases.

A continuación, tendremos una lista con cada una de las clases que componen este diagrama y el diagrama de clases ([Figura 16](#)):

- **Booking**: Contendrá la información sobre las reservas que haya en el hotel.
- **BookingType**: Contendrá los datos sobre los tipos de reservas que existen.
- **Rooms**: Contendrá la información de las habitaciones.
- **Profile**: Contendrá datos del perfil al que esté a nombre la reserva.
- **Facilities**: Contendrá los datos de una instalación del hotel.
- **WebappUser**: Entidad para saber si una booking tiene webapp user asociado.
- **ActivityReservations**: Contendra la información sobre las reservas de actividades.
- **Activity**: Contendrá los datos de las actividades.

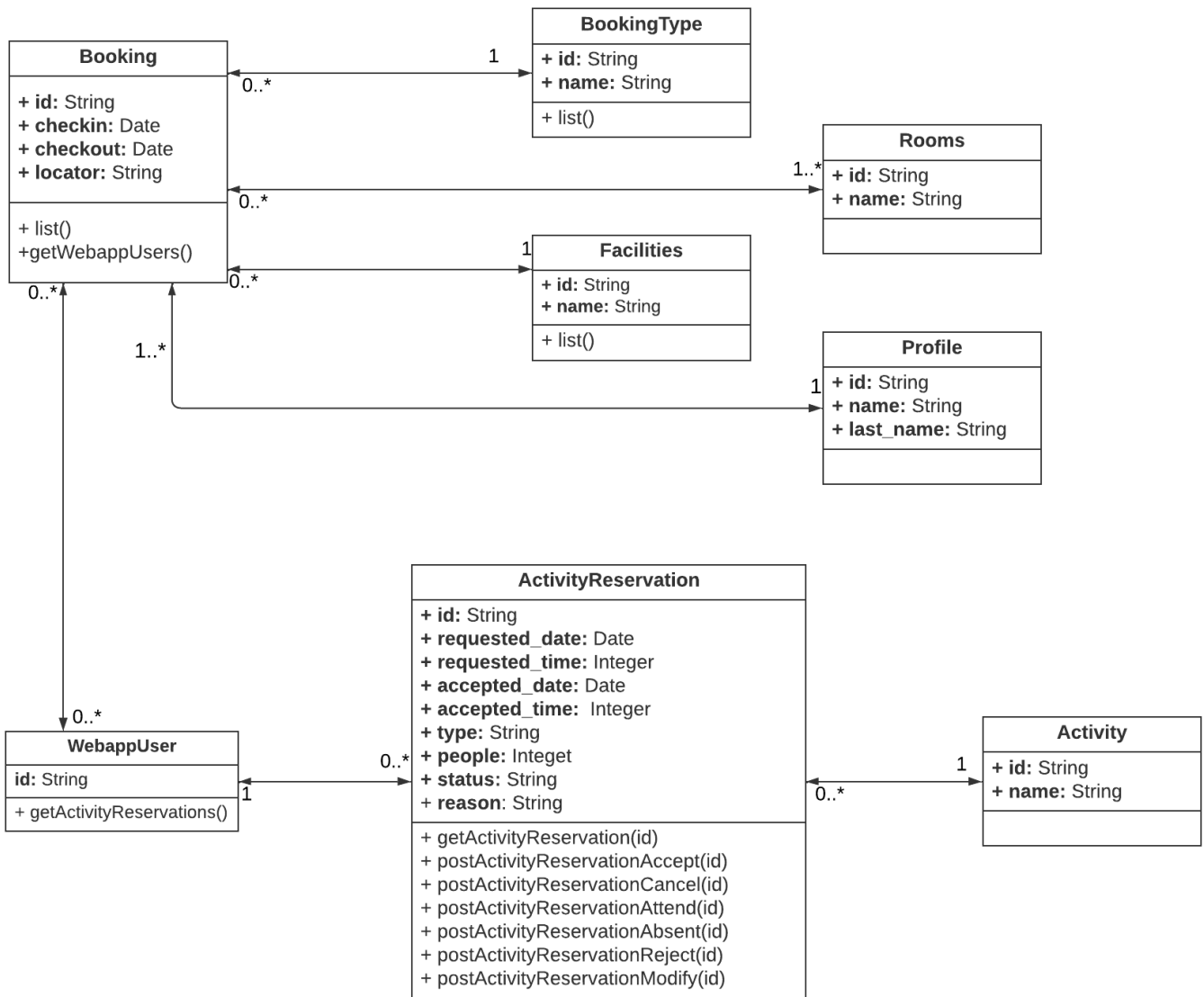


Figura 16. Diagrama de clases

3.2. Diseño de la arquitectura del sistema

En este apartado vamos a comentar la arquitectura que utiliza el sistema que hemos implementado.

La arquitectura utilizada es la denominada arquitectura Flux ([Figura 17](#)) [21]. Esta arquitectura es, por ejemplo, la que utiliza Facebook para crear aplicaciones web del lado del cliente.



Figura 17. Flux logo

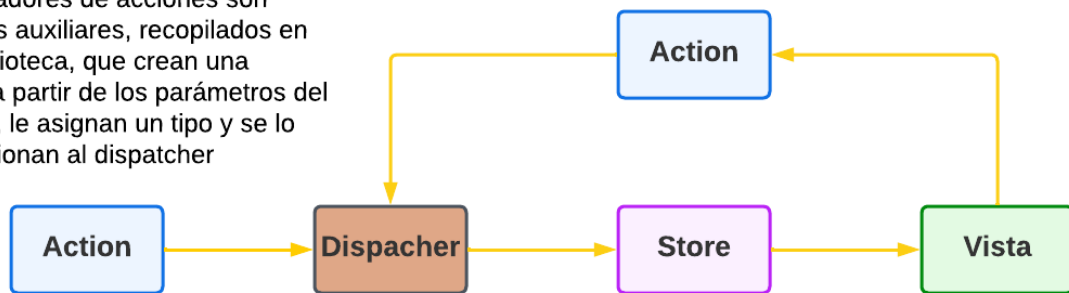
Las aplicaciones que se basan en la arquitectura Flux ([Figura 18](#)) se dividen en tres partes: el dispatcher, la store y las vistas (componentes Vue en nuestro caso). No hay que confundir esta arquitectura con el patrón MVC aunque pueda parecer que tienen rasgos en común.

La arquitectura Flux propone un modelo en el que los datos viajan en un flujo unidireccional. Cada vez que se ejecuta una acción esta se envía mediante un dispatcher a la store y la store se actualiza para mantener el estado de la aplicación actualizado, se emite un evento de cambio y la vista lee los nuevos datos de la store y ésta se actualiza. La propia vista puede emitir acciones que seguirán el flujo descrito anteriormente.

Puede haber una única store para mantener el estado de la aplicación pero lo más normal es dividirla en módulos para que solo contengan los datos y métodos necesarios. En nuestro caso tenemos un módulo para los datos relacionados con las bookings, otro módulo para las facilities, otro para las acciones de cambio de estado, etc.

En nuestro caso muchos de los datos que necesitamos vienen desde una API. Para obtenerlos sería la store la que realizará la llamada a la API, actualizará el estado y la vista se actualizará para mostrar los datos nuevos.

Los creadores de acciones son métodos auxiliares, recopilados en una biblioteca, que crean una acción a partir de los parámetros del método, le asignan un tipo y se lo proporcionan al dispatcher



Cada acción se envía a todas las store a través de las devoluciones de llamada que las tiendas registran con el dispatcher

Después de que las store se actualicen en respuesta a una acción, emiten un evento de cambio.

Una vista especial llamada vistas de controlador, escucha los eventos de cambio, recupera los nuevos datos de las store y proporciona los nuevos datos a todo el árbol de sus vistas hijas.

Figura 18. Arquitectura Flux [\[19\]](#)

3.3. Diseño de la interfaz

Al tratarse de un proyecto centrado en el front-end la parte de la interfaz del usuario es muy importante ya que el objetivo final es tener un producto sencillo, bonito y agradable para su uso.

Al ser una herramienta tan requerida por la empresa fue la misma empresa la que nos proporcionó el diseño final de las interfaces de usuario, que ellos ya habían pensado, para cada vista de la herramienta.

Yo simplemente elaboré un sencillo wireframe sobre la vista del cambio de estado para entender mejor cómo debía ser y funcionar esa parte ya que es la más compleja de la herramienta. Pero una vez terminado ese prototipo fue desechado y sustituido por la versión final que nos proporcionó la empresa.

A continuación, se exponen los diferentes elementos relacionados con todo lo utilizado para crear las interfaces de usuario y finalmente se explicarán y mostrarán las interfaces finalizadas.

3.3.1 Guía de estilos

Una guía de estilos es un conjunto de criterios, reglas y estándares seguidos para completar el desarrollo de una interfaz. Siguiendo esta guía de estilos se consigue homogeneidad y coherencia en el diseño de todas las interfaces del producto.

Colores

En la [Figura 19](#) se muestran los colores principales utilizados para el diseño general de las interfaces y sus elementos.



Figura 19. Colores de la interfaz

La [Figura 20](#) muestra la gama de colores utilizados para indicar y diferenciar los diferentes estados en que puede estar una reserva de actividad.

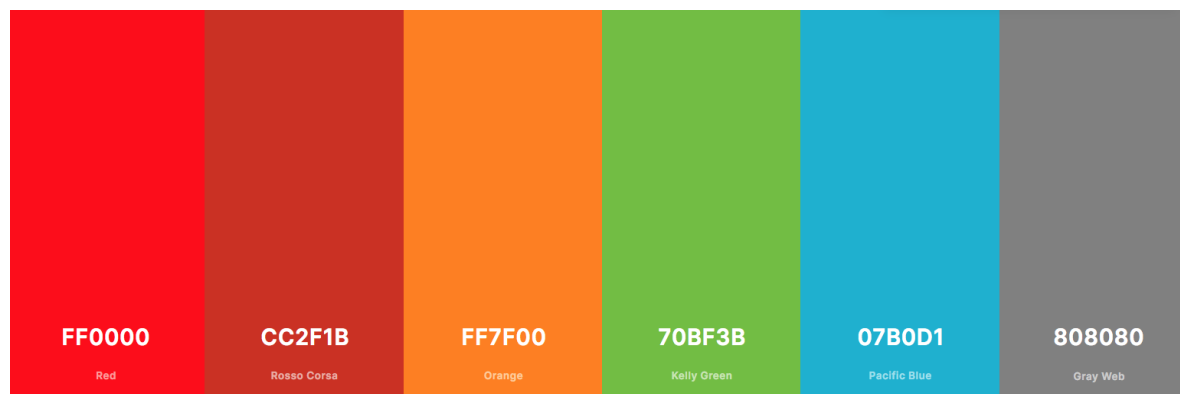


Figura 20. Colores para los diferentes estados

Fuentes

Las fuentes utilizadas para las interfaces han sido las siguientes:

- **Urbanist:** <https://fonts.google.com/specimen/Urbanist?query=urba>
- **Roboto:** <https://fonts.google.com/specimen/Roboto?query=robot>
- **Ubuntu:** <https://fonts.google.com/specimen/Ubuntu?query=ubuntu>

Elementos

En la siguiente figura ([Figura 21](#)) se pueden observar algunos de los diferentes elementos utilizados para las vistas de este proyecto.

Todos los elementos son procedentes del framework anteriormente descrito llamado Vuetify. Son elementos prediseñados listos para usar y adaptar a las necesidades concretas que requieran.

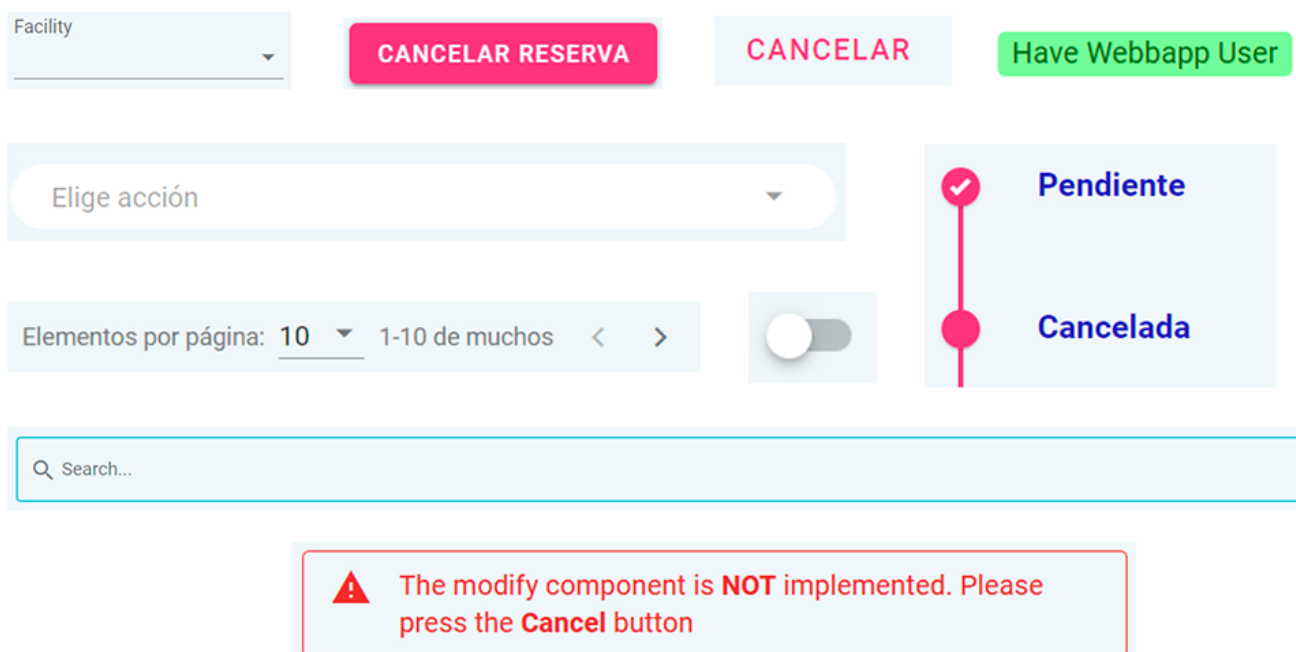


Figura 21. Elementos utilizados

Interfaces

Las interfaces de este proyecto fueron diseñadas por la empresa, en ningún momento tuvimos que centrarnos en su diseño ya que los wireframes nos fueron compartidos.

De las interfaces que se van a mostrar se puede decir que son interfaces limpias, sencillas e intentando tener solo la cantidad de elementos necesarios para hacer ágil, sencillo e intuitivo su uso. Son interfaces que consiguen que el desarrollador llegue a su objetivo con el menor número de clicks posibles y de forma muy directa y efectiva.

A continuación, se van a mostrar cómo lucen las diferentes interfaces del proyecto.

Interfaz de la vista del listado de bookings

En primer lugar tenemos la interfaz de la vista que debe mostrar el listado de bookings ([Figura 22](#)). En la parte superior encontramos el header que tiene el nombre de la extensión y logotipo de la empresa y que es común a todas las interfaces. En la vista el elemento predominante es la tabla donde se muestran las bookings existentes y los datos más relevantes de estas. En la parte superior de dicha tabla podemos encontrar los diferentes elementos que vamos a poder usar para filtrar las bookings para mostrar solo aquellas que nos interesan.

En la parte inferior izquierda encontramos el elemento conocido como paginación, el cual nos va a permitir navegar por las diferentes páginas de la tabla, dependiendo del número de reservas que haya.

Buscar una reserva para encontrar una reserva de actividad

Facility Type Checkin Checkout

Q Search...

Contact Name	Checkin	Checkout	Rooms	Locator	Webbapp User
Néstor López Olaria	16/01/2010 16:50	25/12/2012 16:50	117	AAAAAA	Have Webbapp User
Sven Vavricka	13/10/2016 20:00	14/10/2016 14:00	102		No Webbapp User
no name GUBSE AG	12/01/2017 19:00	14/01/2017 13:00	102		No Webbapp User
AND MONIQUE CLAIRMONT RALPH	12/09/2018 14:00	28/10/2019 19:00		I 41211 1	No Webbapp User
FERNANDO ALTAMIRANO	17/09/2018 14:00	28/10/2019 19:00		I 48375 1	No Webbapp User
UNDERWOOD MONICA DICKENS TONIA	18/09/2018 14:00	23/09/2018 20:00		I 36531 1	No Webbapp User
RAMON COTA ARVAYO	09/09/2019 14:00	11/09/2019 14:00		I 53314 1	No Webbapp User
no name de los plotes	03/10/2019 20:00	04/10/2019 14:00	106		No Webbapp User
Torsten Bellmann	15/10/2019 20:00	16/10/2019 14:00	116		No Webbapp User
PEDRO HIDALGO	28/10/2019 13:00	28/10/2019 19:00		I 53959 1	No Webbapp User

Elementos por página: 10 1-10 de muchos < >

Figura 22. Interfaz listado de reservas.

Interfaz de la vista del listado de reservas de actividades

La siguiente interfaz es la segunda vista de un total de tres. En esta interfaz es donde el desarrollador podrá ver el listado de reservas de actividades asociadas a la booking que haya seleccionado en la vista anterior ([Figura 23](#)).

Es una interfaz mucho más simple que la primera ya que es simplemente una tabla donde se ven las reservas de actividades y los datos más importantes de estas. En este caso no vamos a realizar búsqueda filtrada por tanto los elementos de antes desaparecen.

Abajo a la izquierda nos volvemos a encontrar el elemento paginación que nos permite navegar entre las diferentes páginas de la tabla.

Cambio de estado de la reserva de una actividad

← Reservas de actividades para la reserva seleccionada

Activity	Requested	Type	People	Status
Club de lectura	05/07/2022 11:30	In Venue	1	PENDING_MODIFICATION
Yoga aéreo	30/06/2022 11:00	In Venue	0	REJECTED
Yoga aéreo	29/06/2022 14:15	In Venue	0	REJECTED
Yoga aéreo	10/06/2022 11:00	In Venue	0	PENDING
Yoga aéreo	21/06/2022 11:00	In Venue	0	CANCELLED
Restaurante Mediterraneo	31/05/2022 21:00	In Venue	1	CANCELLED
Yoga aéreo	29/06/2022 15:00	In Venue	0	CANCELLED
Yoga aéreo	20/06/2022 11:00	In Venue	0	ABSENT
Yoga aéreo	15/06/2022 15:45	In Venue	0	ATTENDED
Yoga aéreo	31/05/2022 11:45	In Venue	0	ATTENDED

Elementos por página: 10 1-10 de muchos < >

Figura 23. Interfaz listado de reservas de actividades

Interfaz de la vista del cambio de estado

Aquí nos encontramos con la tercera y última interfaz de todas. La interfaz para realizar el cambio de estado de una reserva de actividad ([Figura 24](#)). Es la más diferente de las tres ya que en este caso no nos encontramos con una tabla como elemento principal sino que en su lugar tenemos un timeline.

A esta interfaz se accede una vez has seleccionado una reserva de actividad en la interfaz anterior. Una vez accedes ves el timeline con su primer elemento que en este caso es el estado actual en el que se encuentra la reserva.

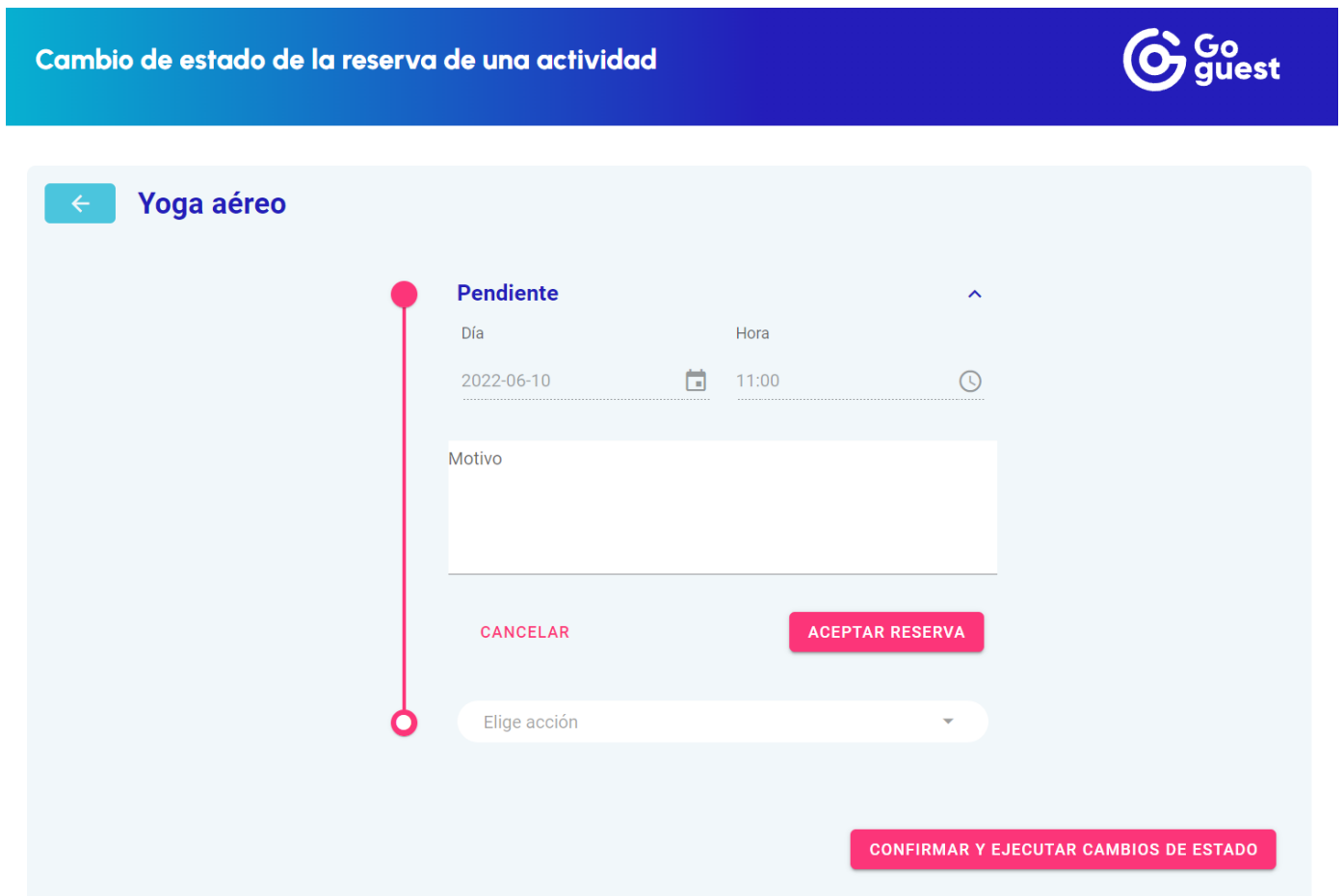


Figura 24. Interfaz cambio de estado

Como estando en ese estado podemos proceder a cambiar el estado de la reserva, por eso en la parte inferior del timeline tenemos un elemento *select* que nos sirve para elegir la acción que deseamos realizar para cambiar el estado ([Figura 25](#)).

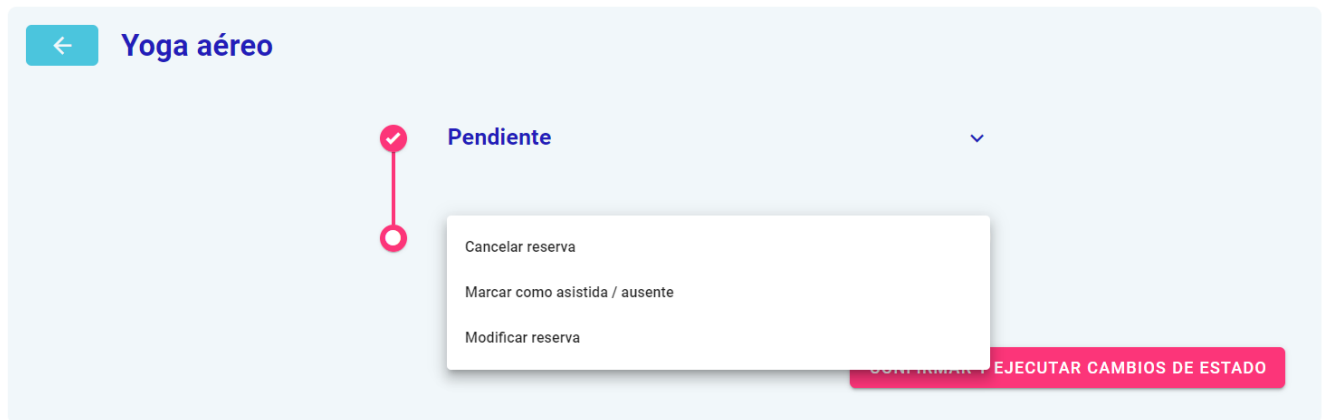
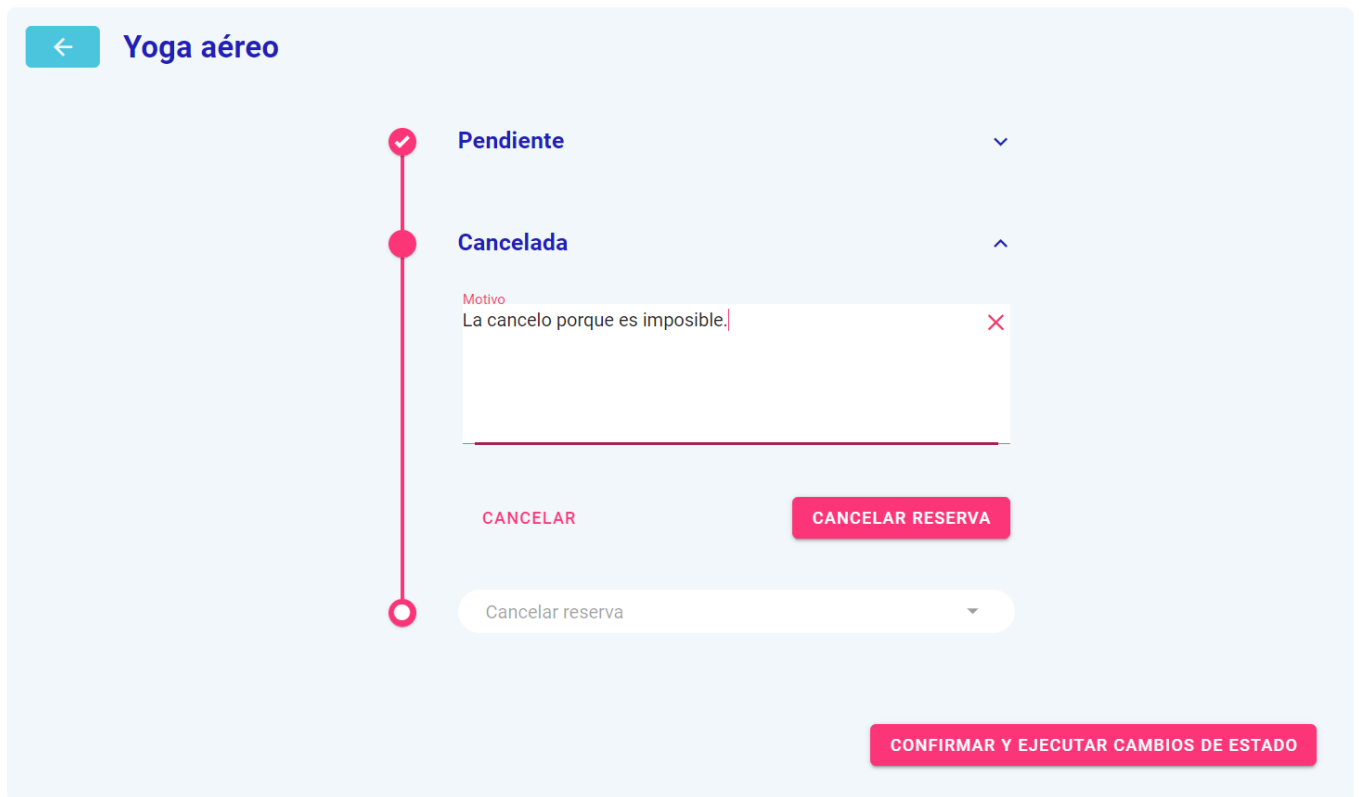


Figura 25. Seleccionar acción para cambiar estado.

Una vez elegida la acción deseada se añade al timeline el componente que representa esa acción. De esta forma podemos rellenar los datos que son necesarios y procedemos a aceptar la acción. Esta acción no se ejecuta en ese momento, sino que se guarda en el estado de la aplicación para ejecutarse cuando indique el desarrollador ([Figura 26](#)).



← Yoga aéreo

✓ Pendiente

● Cancelada

Motivo
La cancelo porque es imposible. ✕

CANCELAR CANCELAR RESERVA

Cancelar reserva

CONFIRMAR Y EJECUTAR CAMBIOS DE ESTADO

Figura 26. Completar datos para acción de cambio de estado

En la siguiente y última interfaz ([Figura 27](#)) vemos que como hemos alcanzado un estado final de la reserva ya no nos permite elegir más acciones ya que sería erróneo. Por tanto se muestra un mensaje indicando lo que sucede. Finalmente en la parte inferior derecha tenemos el botón que nos sirve para ejecutar todas las acciones seleccionadas anteriormente. En este ejemplo primero hemos Aceptado la reserva y luego la hemos Cancelado, esas serán las dos acciones a ejecutar.

← Yoga aéreo

- ✓ Pendiente
- ✓ Cancelada

You can't change the state because it's already in a final state

CONFIRMAR Y EJECUTAR CAMBIOS DE ESTADO

Figura 27. Alcanzado estado final de una reserva de actividad

Capítulo 4

Implementación y pruebas

4.1. Detalles de la implementación

Como hemos comentado en el apartado de las tecnologías, para desarrollar este proyecto hemos utilizado el framework de JavaScript conocido como Vue.js. Es el que utiliza la empresa donde se realizó el proyecto ya que es un framework que en los últimos años ha ganado mucha popularidad para trabajar la parte del frontend.

Vue.js es un framework que nos proporciona la posibilidad de organizar la estructura y el código de forma muy sencilla. Cada componente (archivo con extensión `.vue`) puede tener su parte HTML para la parte visual, a su vez el mismo fichero puede contener código JavaScript para la parte de funcionalidad y además podríamos incluir en el mismo archivo la parte dedicada a los estilos gracias a CSS. Más adelante en este capítulo se mostrarán ejemplos de estos archivos.

También hemos hecho uso de de otras librerías para complementar la implementación de una forma más rápida y sencilla:

- *Vuetify*: Vuetify es un framework que se puede utilizar junto con Vue.js para poder acelerar el desarrollo de aplicaciones web ya que incorpora una gran cantidad de componentes prediseñados y listos para usar. Toda la aplicación ha sido construida utilizando componentes de Vuetify.
- *Vuex*: Esta librería nos ha permitido tener un control sencillo y total sobre el estado de la aplicación. Mediante las acciones y las store hemos podido realizar llamadas a la API y utilizar y guardar los datos recibidos. Hemos podido actualizar el estado a placer para que siempre esté con los datos más recientes.
- *Vue-router*: Nos ha servido para tener un rutado entre componentes muy sencillo. Cada componente va asignado a una ruta específica y eso hace muy fácil la navegación, además de que otorga gran sencillez a la hora de pasar parámetros por ruta de un componente a otro.

4.2. Estructura del proyecto

La estructura del proyecto está relacionada con cómo organizamos los diferentes directorios y archivos que conforman el proyecto. Nuestra estructura nace a raíz de una plantilla que nos proporcionó la empresa que ya tienen preestablecida para la creación de extensiones. En la [Figura 28](#) podemos ver todos los directorios existentes. Entraremos a analizar cada uno de ellos a continuación.

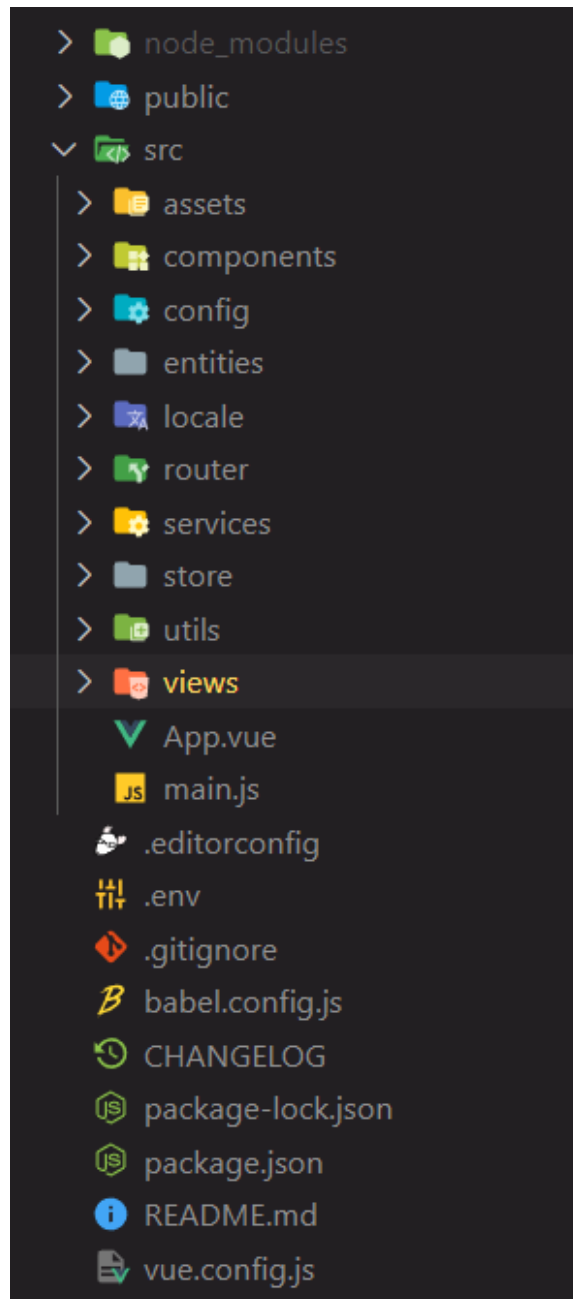


Figura 28. Estructura de los directorios de la aplicación

En primer lugar encontramos el directorio *node_modules*. En este directorio se guardan las herramientas de compilación. También es el directorio donde se almacenarán las librerías que se vayan a instalar durante la creación del proyecto. Para la instalación de librerías externas hemos utilizado **npm** que es sistema de gestión de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript [22].

En el siguiente directorio denominado *public* (Figura 29) se encuentra el archivo *index.html* que se utiliza para iniciar el proyecto. Este archivo leerá la información del archivo *App.vue* para iniciar todo, ya que es así como se inicia *Vue.js*. También encontramos el archivo *config.js* que contiene una variable que guarda la ruta base principal de la API, de esta forma es más sencillo después acceder a ella. También encontramos los diferentes archivos para los logotipos de la empresa.

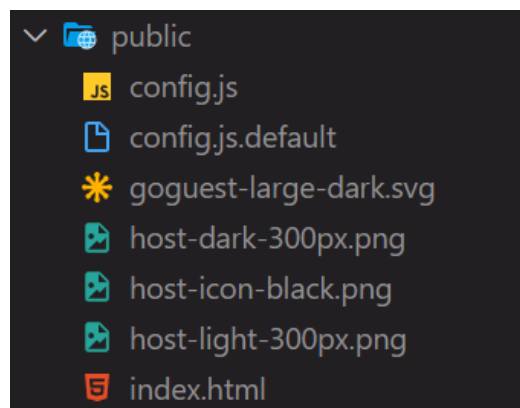


Figura 29. Directorio *public*

El siguiente directorio es el directorio *src*, es de los más importantes sin duda, ya que contiene la mayor cantidad de directorios, archivos y código del proyecto, digamos que es donde se encuentra todo el cuerpo del proyecto. A continuación vamos a describir lo más importante que contiene.

Contenido directorio SRC

- En primer lugar, el directorio *components* (Figura 30) contiene todos los componentes creados por nosotros mismos que serán utilizados para el manejo de diferentes acciones. Se crean componentes para tener una funcionalidad más aislada fácil de cambiar y que permite la posibilidad de reutilización como el caso del componente de *ExternalPagination.vue*.

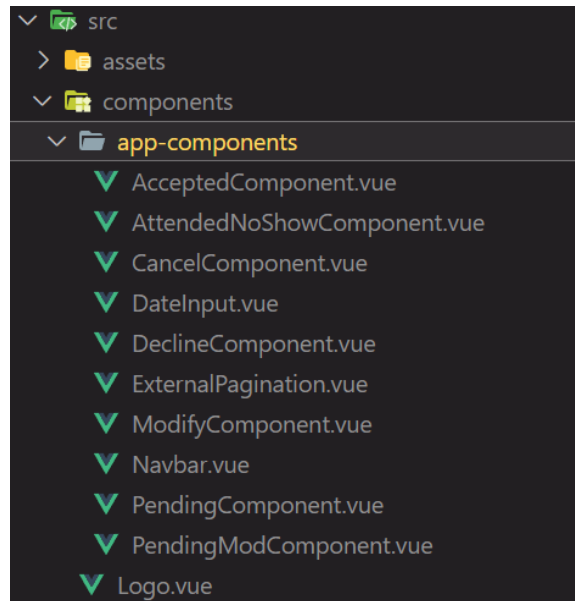


Figura 30. Componentes

También observamos que se encuentran todos los componentes que vamos a utilizar para cada cambio de estado específico que queramos realizar, según la acción que seleccionemos en la interfaz se renderizará uno u otro componente. En el caso del `Navbar.vue` ([Tabla 12](#)) es un componente que está todo el rato visible en la parte superior de la interfaz.

```

<template>
  <nav app>
    <v-app-bar elevation="0" style="background: linear-gradient(88.63deg, #00D7D7
-11.71%, #241DBA 56.83%); height: 116px;" class="py-6 px-10">
      <span class="white--text">
        <span style="font-style: normal; font-weight: 700; font-size:
25px;font-family: 'Urbanist';">{{ this.$t('nav.title') }}</span>
      </span>
      <v-spacer></v-spacer>
      <LogoBigDark style="height: 100px"/>
    </v-app-bar>
  </nav>
</template>

<script>
export default {
  components: {
    LogoBigDark: () => import('@/../public/goguest-large-dark.svg')
  }
}

```

```
}  
</script>  
  
<style>  
</style>
```

Tabla 12. Código del componente Navbar.vue

- Seguimos con el directorio *config* ([Figura 31](#)) donde encontramos archivos que tendrán básicamente variables de entorno para facilitar su uso a lo largo del código, por ejemplo el archivo *server-directions.js* ([Tabla 13](#)) contiene todas las rutas que vamos a utilizar para llamar a la API.

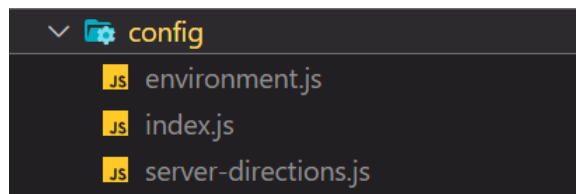


Figura 31. Directorio *config*

```
import { API_ROOT as API_BASE } from './environment'  
  
export default {  
  ACTIVITY_RESERVATIONS: API_BASE + '/activity-reservations',  
  ACTIVITY_CATEGORIES: API_BASE + '/activity-categories',  
  ACTIVITY_RESERVATION: API_BASE +  
'/activity-reservations/{activityReservationId}',  
  ACTIVITY_RESERVATIONS_ID_MODIFY: API_BASE +  
'/activity-reservations/{activityReservationId}/modify',  
  ACTIVITY: API_BASE + '/activities/{id}',  
  BOOKINGS: API_BASE + '/bookings',  
  BOOKING_TYPES: API_BASE + '/booking-types',  
  FACILITIES: API_BASE + '/facilities',  
  BOOKING_ID_WEBAPP_USER: API_BASE + '/bookings/{id}/webapp-users',  
  WEBAPP_USERS_WEBAPP_USER_ID_ACTIVITY_RESERVATIONS: API_BASE +  
'/webapp-users/{id}/activity-reservations/'  
}
```

Tabla 13. Código archivo *server-directions.js*

- El directorio *entities* contiene los archivos que representan cada una de las entidades. Las entidades contienen los datos que llegan de la API que representan una entidad de algo concreto: Booking, Activity, Room ([Tabla 14](#)), etc.

```
import { ExternalId } from './ExternalId'
export class Room {
  constructor (params = {}) {
    this.id = params.id ? params.id : null
    this.externalIds = params.external_ids ? params.external_ids.map((e) =>
new ExternalId(e)) : []
    this.sequence = params.sequence ? params.sequence : null
    this.name = params.name ? params.name : null
    this.type = params.type ? params.type : null
    this.typeId = params.type_id ? params.type_id : null
    this.facility = params.facility ? params.facility : params.facility_id
    this.facilityId = params.facility_id ? params.facility_id : null
  }
  serialize (editedFields) {
    var item = JSON.parse(JSON.stringify(this))
    Object.keys(editedFields).forEach(key => {
    item[key] = editedFields[key]
    })
    return {
    id: item.id,
    name: item.name,
    facilityId: item.facilityId,
    typeId: item.typeId
    }
  }
}
```

Tabla 14. Ejemplo entidad Room

- El directorio *locale* contiene los archivos de traducción de la aplicación. Las traducciones que se almacenan aquí son para los textos fijos en la aplicación. Por ejemplo el texto “Cambio de estado de la reserva de una actividad” que aparece en el componente Navbar es siempre el mismo, lo único que hacemos con estos archivos es mostrarlo en inglés o español dependiendo del lenguaje que se tenga configurado en el navegador.

- El directorio *router* contiene el archivo `index.js` en el cual se almacenan todos los datos de las rutas para poder navegar entre los diferentes componentes y vistas que forman la aplicación. Veamos un pequeño ejemplo en la [Tabla 15](#).

```
{
  path: '/activities/:id',
  name: 'activities',
  component: () => import(/* webpackChunkName: "activities" */
'../views/ActivitiesView.vue')
}
```

Tabla 15. Ruta de la vista del listado de reservas de actividades

Vemos que esta ruta cargará el componente vista llamado `ActivitiesView.vue`, el cual será llamado utilizando esa ruta junto con el parámetro `'id'` que le pasaremos para poder identificar de qué *webbapp user* queremos obtener el listado de actividades.

- El directorio *services* ([Figura 32](#)) contiene los archivos para realizar la llamada a la API. Cada archivo contendrá los métodos necesarios para obtener los datos de entidades concretas: `Activities`, `Bookings`, `Facilities`, etc. Los métodos de estos archivos son llamados desde la store y devolverán los datos necesarios según el método llamado para que estén disponibles desde la store.

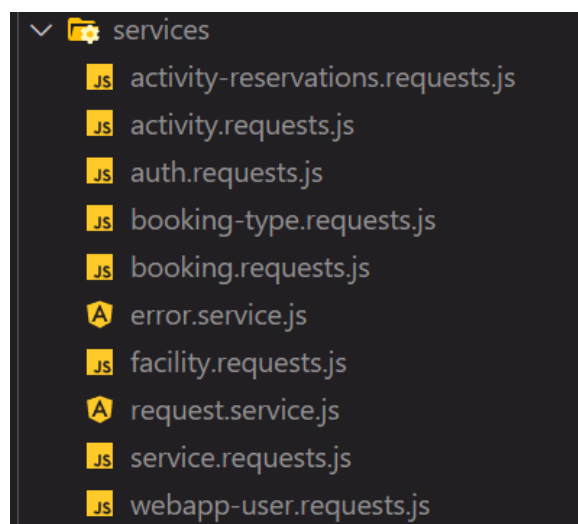


Figura 32. Directorio services

- El directorio *store* ([Figura 33](#)) contiene todos los módulos de las diferentes store. Cada módulo hace referencia a una entidad de datos concreta. Por ejemplo, tenemos el módulo *bookings.js*, *facilities.js*, *activity-reservations.js*, etc. La store nos proporciona unas acciones que podremos ejecutar desde las vistas para obtener por ejemplo el listado de bookings, o para obtener el listado de facilities o para obtener una reserva de actividades concreta. Ejecutas la acción que quieres desde un componente Vue, la acción de la store concreta realizará la llamada al servicio necesario y el servicio llamará a la API y devolverá los datos recibidos los cuales podrás utilizar libremente.

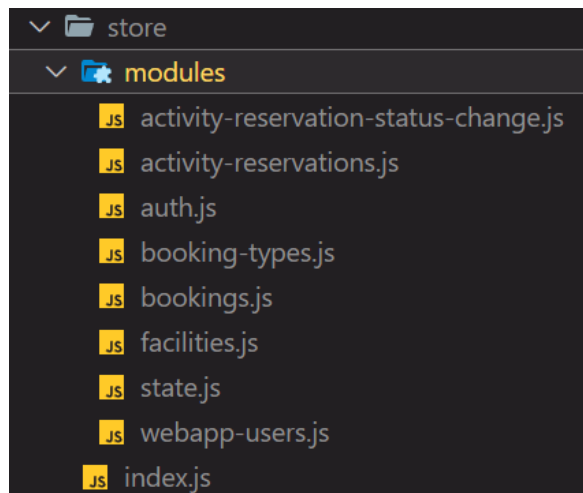


Figura 33. Directorio store

- Finalmente llegamos al directorio *view* ([Figura 34](#)) que contiene las vistas principales de nuestra aplicación. Tenemos tres vistas en total, la vista del listado de reservas, la vista del listado de reservas de actividades y la vista de cambio de estado. Cada una de estas vistas utiliza los diferentes componentes que nos ofrece Vuetify junto con los creados específicamente para este proyecto.

Desde las vistas se ejecutan las acciones para que nos devuelvan los datos necesarios. Por ejemplo, en la primera vista, utilizaremos la acción de la store necesaria para obtener el listado de reservas. Una vez obtenido el listado procederemos a mostrarlos en la tabla que nos ofrece Vuetify. Y de esta forma iremos construyendo las vistas y mostrando los datos necesarios.

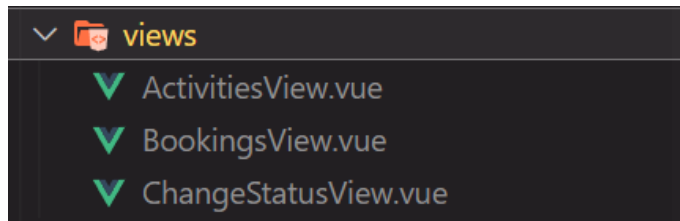


Figura 34. Directorio views

Una vez fuera del directorio src nos encontramos con el archivo App.vue ([Tabla 16](#)) donde gracias a etiqueta `<router-view />` se cargará el componente asociado a la ruta "/" y el componente asociado a esa ruta es la vista de el listado de reservas que tiene que ser nuestra interfaz principal. La vista irá cambiando en ese lugar de forma dinámica según vayamos cambiando la ruta al pasar de una vista a otra. También vemos que cargamos el Navbar para que quede fijo durante toda la ejecución de la aplicación.

```
<template>
  <v-app>
    <Navbar />
    <v-main>
      <v-container fluid class="pt-6">
        <v-layout justify-center align-center>
          <router-view />
        </v-layout>
      </v-container>
    </v-main>
    <v-snackbar
      v-model="state.message.show"
      bottom
      :color="state.message.color"
      :timeout="state.message.timeout">
      {{ state.message.text }}
    </v-snackbar>
  </v-app>
</template>

<script>
import { mapState } from 'vuex'
import Navbar from './components/app-components/Navbar.vue'

export default {
  components: {
    Navbar
  },
}
```

```

    created () {
      const params = new URLSearchParams(window.location.search)
      const token = params.get('x-api-key')
      this.$store.commit('state/setToken', token)
    },
    computed: {
      ...mapState(['state'])
    }
  }
</script>

<style lang="scss">
@import
url('https://fonts.googleapis.com/css2?family=Urbanist:wght@400;800&display=swap');

.big-icon {
  font-size: 38px !important;
}
.flex-none {
  flex: none !important;
}

// VUETIFY
.v-content {
  max-height: 100vh;
  .v-content__wrap {
    overflow: auto;
    position: relative;
  }
}
.v-btn:not(.v-btn--floating):not(.v-btn--icon) {
  border-radius: 5px !important;
}
</style>

```

Tabla 16. Fichero App.vue

4.3. Verificación y validación

La verificación de un programa informático consiste en comprobar que, en ejecución, funciona correctamente aquello que se haya implementado. Esto permite detectar y corregir los errores que van a ir apareciendo a lo largo de su desarrollo.

En nuestro proyecto, para la verificación, se ha utilizado la consola del navegador y el monitor de red ya que son unas de las herramientas más útiles para detectar errores en las aplicaciones web.

Mediante la consola se pueden ver los fallos de llamada a la API por ejemplo. Puedes ver si los datos que recibes y que vas filtrando durante la ejecución del código son los correctos. Desde la red puedes ver que llamada a la API concreta es la que falla y que mensaje de error nos está devolviendo, de esta forma sabes exactamente dónde está el origen del fallo que hay que solucionar.

La consola sirve mucho también para entender y ver que el flujo de los datos funciona de forma correcta a través de todas las llamadas de métodos.

También se hacían revisiones del código por parte de la supervisora de front-end. Mediante GitLab generábamos una *merge request*⁶ una vez finalizamos una funcionalidad concreta del código, y desde esta *merge request* la supervisora podía revisar todo el código e irnos haciendo comentarios de las partes del código que podíamos y debíamos mejorar y cambiar. Una vez corregidos los cambios se procedía a aceptar esa *merge request* de forma que la nueva funcionalidad se juntaba con el resto del código ya operativo.

Como validación lo que hacemos es ejecutar la herramienta como si fuese una demo de prueba e íbamos comprobando que podíamos asegurar que todos los requisitos se podían cumplir. De esta forma nos aseguramos que la herramienta permitía hacer lo que la empresa nos encargó que hiciese, ni nada más, ni nada menos.

⁶ Una merge request sirve para poder corregir el código nuevo que has creado y una vez aceptada que ese nuevo fragmento se junte al código principal sin crear errores.

Capítulo 5

Conclusiones

5.1 Conclusión técnica

Desde el punto de vista técnico creo ha sido bastante enriquecedora la experiencia ya que me ha permitido descubrir tecnologías nuevas como es el caso de Vue.js que como hemos comentado es un lenguaje bastante de moda en la actualidad que viene bien conocerlo.

Haber aprendido un poquito como funciona GitLab me servirá bastante en el futuro ya que muchas empresas lo utilizan incluso puedo usarlo para proyectos personales.

Trabajar de una forma más inmersiva con la API también me ha venido bien para terminar de fortalecer algunos de los conocimientos adquiridos durante el grado.

Prácticamente todas las tecnologías y herramientas utilizadas a lo largo de la estancia en prácticas han sido nuevas para mi por tanto me voy bastante contento por haber podido aprender sobre ellas ya que quizás en el futuro me puedan ser útiles.

Finalmente cabe mencionar que el proyecto no ha quedado completado al 100% ya que faltan por implementar el cambio de estado cuando se quiere modificar una reserva de actividad (la hora, el día, etc). Pero respecto a los objetivos desde los que partimos creo que ha quedado un herramienta funcional para determinados casos y creo que la empresa le podrá sacar partido.

5.2 Conclusión laboral

En cuanto a la parte laboral ha sido nueva para mí porque nunca había estado en una empresa realizando un trabajo del ámbito de la informática por tanto ha sido una experiencia bastante enriquecedora ya que me ha permitido conocer de forma bastante cercana el como puede ser trabajar en una empresa de estas

características. El ver cómo se reúnen, como se organizan, como resuelven dudas, horarios y reuniones me ha dado una visión general de cómo sería trabajar en un proyecto del mundo de la informática.

Las prácticas las he realizado de forma no presencial ya que la empresa me daba libertad de poder elegir la forma en que quisiera hacerlas y por un tema de distancia con el lugar de residencia la mejor opción era sin duda no presencialidad. Yo creo bastante en que la no presencialidad debería ser una opción disponible para todo el mundo en el futuro, ya que precisamente nuestro sector por suerte puede aplicarlo sin muchos problemas.

Respecto a la empresa no tengo ninguna queja. Son un equipo genial y al ser todavía una empresa con “poca” gente se nota que se conocen bien y que son capaces de generar muy buen ambiente de trabajo.

5.3 Conclusión personal

Finalmente, como conclusión personal, creo que puedo decir que ha sido bueno tener esta experiencia laboral, realmente es donde por fin ves cómo se trabaja y si estás preparado o no. Siendo sincero considero que he estado a la altura, pero por los pelos, quizás me he notado más justo de lo que pensaba. Me hubiese gustado hacerlo mejor de lo que lo he hecho pero al menos si me voy contento porque ha quedado un proyecto terminado.

En este tipo de proyecto la mayoría de las cosas eran nuevas y tenía que buscarme la vida. Lo aprendido en la carrera seguramente ha ayudado en esa tarea de buscarme la vida ya que sí te prepara para ello y seguro que también ha ayudado a la hora de saber como crear un documento técnico como este.

De cara al futuro espero aprender de los errores cometidos y tratar de ser un poquito mejor cada día.

Bibliografía

- [1] Página web de la empresa Easygoband <https://www.easygoband.com/> [Consulta: 22 de febrero de 2022]
- [2] Documentación Vue.js. <https://vuejs.org/> [Consulta: 9 de marzo de 2022]
- [3] Documentación Vuetify. <https://vuetifyjs.com/en/> [Consulta: 15 de marzo de 2022]
- [4] Material Design Icons. <https://materialdesignicons.com/> [Consulta: 9 de marzo de 2022] Documentación Vuex. <https://vuex.vuejs.org/> [Consulta: 10 de marzo de 2022]
- [5] Vuetify, estética Material Design para tus Apps en VueJS, Luis Llamas <https://www.luisllamas.es/vuetify-estetica-material-design-para-tus-apps-en-vuejs/> [Consulta: 31 de mayo de 2022]
- [6] Documentación Vuex. <https://vuex.vuejs.org/> [Consulta: 10 de marzo de 2022]
- [7] ¿Qué es Vuex?, Un poco de Java <https://unpocodejava.com/2019/05/09/que-es-vuex/> [Consulta: 31 de mayo de 2022]
- [8] ¿Para qué sirve Jira?, Atlassian <https://www.atlassian.com/es/software/jira/guides/use-cases/what-is-jira-used-for> [Consulta: 31 de mayo de 2022]
- [9] ¿Qué es GitLab y donde alojarlo? Geekflare. <https://geekflare.com/es/gitlab-hosting/>
- [10] Visual Studio Code. Wikipedia. https://es.wikipedia.org/wiki/Visual_Studio_Code
- [11] Swagger y swagger UI: ¿Qué es y por qué es imprescindible para tus APIs?, Chakray <https://www.chakray.com/es/swagger-y-swagger-ui-por-que-es-imprescindible-para-tus-apis/> [Consulta: 31 de mayo de 2022]
- [12] ¿Qué es Figma? CEI.. <https://cei.es/que-es-figma/> [Consulta: 13 de junio de 2022]

- [13] PMBOK Guide, Project Magement Institut
<https://www.pmi.org/pmbok-guide-standards/foundational/PMBOK>
- [14] S. Pressman, Roger. Ingeniería del software: Un enfoque práctico, 3.^a Edición, Pag. 26-30
<http://cotana.informatica.edu.bo/downloads/Id-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF>
- [15] Scrum: qué es, cómo funciona y por qué es excelente. Atlassian.
<https://www.atlassian.com/es/agile/scrum> [Consulta: 27 de junio de 2022]
- [16] Gestión predictiva.
https://www.scrummanager.net/bok/index.php?title=Gesti%C3%B3n_predictiva [Consulta: 27 de junio de 2022]
- [17] Indeed, Sueldo Promedio Programador Junior España
https://es.indeed.com/career/programador-junior/salaries?from=top_sb [Consulta: 13 de junio de 2022]
- [18] Indeed, Sueldo Promedio Programador Senior España
https://es.indeed.com/career/desarrollador-senior/salaries?from=top_sb [Consulta: 13 de junio de 2022]
- [19] What is UML?, UML.org <https://www.uml.org/what-is-uml.htm> [Consulta: 31 de mayo de 2022]
- [20] Diagramas de clases. Diagramas UML. <https://diagramasuml.com/diagrama-de-clases/>
- [21] Flux, Facebook <https://facebook.github.io/flux/docs/in-depth-overview/> [Consulta: 7 de junio de 2022]
- [22] npm, Wikipedia <https://es.wikipedia.org/wiki/Npm> [Consulta: 13 de junio de 2022]