Grado en Ingeniería Informática

Trabajo de Final de Grado

Gestor de reservas para excursiones en barco a las Islas Columbretes

Autor:
Alejandro Guinot Pérez

Supervisor:
Antonio Vera Segarra
Tutor académico:
Jorge Sales Gil

Fecha de lectura: 12 de julio de 2023
Curso académico 2022/2023

Resumen

La siguiente memoria expone el proceso de creación y desarrollo de una aplicación web a medida, hecha durante la estancia de prácticas en la empresa Angal Informática S.L. Forma parte del itinerario de Ingeniería de Software del grado en Ingeniería Informática de la Universitat Jaume I.

A través de cada capítulo se va exponiendo cuáles han sido los procedimientos necesarios para desarrollar una pasarela de reservas de barcos, con el objetivo de realizar excursiones a las Islas Columbretes. La aplicación permite gestionar días de salida y reservas, con el añadido de ofrecer información al usuario mediante notificaciones personalizadas.

Mediante el uso de tecnologías web, se detalla paso a paso, cómo se ha planificado el proyecto, cómo se ha modelado y, finalmente, cómo se ha implementado, haciendo hincapié en detalles y problemáticas que han ido surgiendo durante el desarrollo del mismo.

Palabras clave

Barcos, reservas, Laravel, desarrollo web, pasarela de pago, aplicación *full-stack*

Keywords

Boats, reservations, Laravel, web development, gateway payment, full-stack application

Índice general

Índice de figuras

Índice de cuadros

Listings

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto que se presenta en esta memoria, se ha desarrollado durante la estancia de prácticas en Angal Informática, empresa creada en Castellón con el objetivo de ofrecer soluciones informáticas a nivel empresarial. Angal ofrece una gran variedad de servicios, entre ellos, el posicionamiento en buscadores, programas de gestión a medida o el desarrollo de aplicaciones web. El principal objetivo de la empresa es, a través de las soluciones que proporcionan, satisfacer las necesidades de sus clientes mediante el seguimiento de sus proyectos, esto implica que la comunicación e implicación con el cliente sea vital para el correcto desarrollo de cualquier proyecto [1].

La colaboración con Angal que ocupará la siguiente memoria técnica será con una empresa encargada de realizar excursiones en barco a las Islas Columbretes [2], que no tiene a disposición un servicio de reservas similar al de *Booking* [3] o *Airbnb* [4]. La principal motivación del proyecto es ofrecer una solución informática que automatice el proceso de la gestión de reservas de barcos, controlando diferentes posibilidades de embarque y limitando el control de plazas para los barcos.

1.2. Objetivos del proyecto

El principal objetivo del proyecto es desarrollar una pasarela de reservas de barcos, con el fin de realizar actividades de ocio como excursiones o actividades marítimas por destacar las más importantes. Este objetivo se puede concretar en los siguientes objetivos secundarios:

- Controlar el número de plazas cuando se realice una reserva.
- Notificar a los usuarios cuando se haya realizado alguna reserva u otra acción relevante.
- Comunicar la aplicación con una pasarela de pago externa.
- Gestionar la sobreventa de plazas.

- Definir una política y sistema de cancelaciones.

Algunos de los objetivos secundarios que ocupan al proyecto son:

- Lograr que la utilización del producto final sea sencilla.
- Favorecer la satisfacción del usuario mediante un servicio eficaz.
- Realizar un diseño agradable y cómodo de la interfaz de usuario.

1.3. Alcance del producto

El alcance de la aplicación que se va a desarrollar se puede subdividir en alcance funcional, organizativo e informático.

- **Alcance funcional**. Son las funcionalidades que va a cubrir la aplicación final. En concreto son las siguientes:
  - Permitir la obtención de datos de los clientes para su registro en la aplicación.
  - Permitir la gestión de plazas de un barco.
  - Permitir la creación de una reserva para un barco.
  - Permitir la cancelación de reservas.
  - Mantener un sistema de tarifas dependiendo del tipo de pasajero.
  - Realizar descuentos a las reservas cuando sea necesario.
  - Enviar notificaciones personalizadas a los clientes.
  - Permitir distintas formas de pago.
  - Controlar el listado de reservas por día.
  - Controlar la ocupación de reservas.
  - Realizar una reserva sin realizar un pago.
  - Modificar fechas de las reservas.

  Algunas de estas funciones serán exclusivas de los administradores, como por ejemplo, la posibilidad de ver un listado de reservas por día, fechas de las reservas o realizar una reserva sin realizar un pago previo.
- **Alcance organizativo**. La aplicación comprende los siguientes tipos de usuario que van a hacer uso de la aplicación:
  - Administrador. Es el usuario *(o usuarios designados por la empresa)* que se encargan de realizar tareas administrativas y realizar cambios importantes. Las funciones que cubre este rol son: ver el listado de reservas completo, ver la ocupación completa de las reservas, realizar una reserva sin pasar por la pasarela de pago, controlar las plazas disponibles y visibles que tiene un barco, y realizar cambios de fechas en las reservas.

- Cliente. Representa a cualquier otro usuario final que va a hacer el uso real de la aplicación. Será el tipo de usuario encargado de realizar el proceso completo de reserva y pago de un barco.

- **Alcance informático**. El sistema necesitará estar conectado a una base de datos con el fin de mantener la gestión de las reservas de barcos en todo momento. También se incluirán los datos de los usuarios para poder comprobar a qué funciones tienen acceso y a cuáles no. La parte del *back-office*[1] solo estará disponible para los administradores.

1.4. Descripción detallada del desarrollo del proyecto

La aplicación será utilizada internamente por el administrador designado por el cliente para controlar el estado de las plazas de los días de salida, con posibilidad de realizar cambios de días si, por ejemplo, un día de salida un barco no puede salir de excursión.

El cliente dispone actualmente de una página web que ofrece información sobre viajes a las Islas Columbretes. En la misma se indican los puertos de salida, los barcos que actualmente están en vigor para realizar dichas excursiones y, lo más importante, una sección donde se comprueba la disponibilidad de los barcos. Actualmente, esta sección se compone de un formulario de contacto a través de correo electrónico o vía llamada telefónica.

Este proceso tiene muchos inconvenientes, que pueden darse cuando haya una solicitud grande de reservas en épocas clave como verano u otras franjas de tiempo que favorecen la realización de este tipo de actividades. Los potenciales usuarios tendrían que estar a la espera de si las fechas que han solicitado se corresponden a un día que puedan cubrir sus necesidades de ocio o buscar otras fechas disponibles que puedan entrar en conflicto con sus intereses. En definitiva, el proceso es lento y tedioso.

Por esta razón, la finalidad del proyecto es automatizar esta sección mediante una aplicación tipo pasarela de reservas que será accesible desde la página web principal. El cliente tiene una serie de requisitos que se deben cumplir en cuanto al proceso de realizar la reserva, así que se realizará lo más simple posible y guiada para que el usuario final no tenga problemas al respecto.

1.5. Estructura de la memoria

La presente memoria técnica se extiende a lo largo de seis capítulos.

El primer y actual capítulo sirve de introducción, presentando el contexto y motivación del proyecto, los objetivos que comprenden el proyecto en cuestión, el alcance del producto que se quiere desarrollar y una descripción más en profundidad de los recursos utilizados.

El segundo capítulo expone la planificación del proyecto, desglosado en el tipo de metodología empleada, cómo se ha planificado temporalmente, el seguimiento del proyecto y un análisis de

[1] Gestión interna de la aplicación.

costes y riesgos.

El tercer capítulo desarrolla el análisis del sistema, el cual se puede dividir en la recolección de requisitos por parte del cliente y un posterior análisis de los mismos.

El cuarto capítulo ilustra el diseño del sistema, mostrando cómo se ha diseñado la base de datos, el diseño de software, el diseño de la arquitectura del sistema y el diseño de las interfaces de usuario.

El quinto capítulo explica en mayor detalle cómo se ha implementado la aplicación y las pruebas que se han realizado. Aquí se comentará cómo está estructurado el código, decisiones tomadas a nivel técnico de la implementación y la verificación y validación de pruebas.

Para terminar, en el sexto capítulo se comentarán reflexiones personales sobre el desarrollo del proyecto, además de posibles funcionalidades extra que se pueden implementar en un futuro.

Capítulo 2

Planificación del proyecto

2.1. Metodología

Debido a que el servicio que se quiere desarrollar es propio de actividades de ocio y culturales, se ofrece la posibilidad de desarrollar el proyecto en un periodo de 3 meses con el fin de que el servicio esté disponible y sea funcional en verano, siendo una época idónea para ofrecer el servicio de excursiones. Como es un espacio de tiempo delimitado, se propone utilizar una metodología predictiva basada en PMBOK[1] [5].

Es una metodología en cascada dividida por fases que delimita las diferentes fases del desarrollo. Tradicionalmente, la metodología en cascada sigue las siguientes iteraciones: análisis del sistema, diseño del sistema, implementación, desarrollo de pruebas y mantenimiento. Algunas de estas secciones pueden requerir de un desglose en más apartados en caso de tratar aspectos concretos del proyecto. La Figura 2.1 muestra cuál es el modelo a seguir para este proyecto:

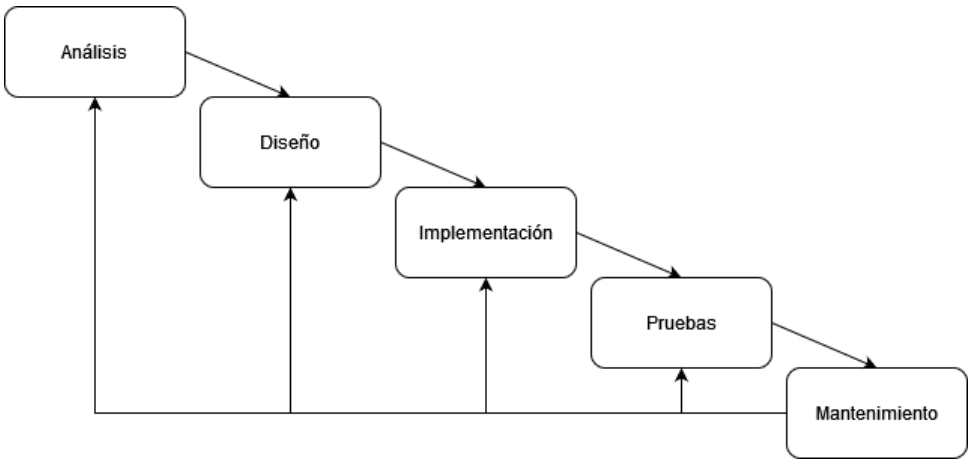


Figura 2.1: Modelo en cascada tradicional

La fase de análisis, como se puede ver en la Figura 2.2, tendrá un desglose para remarcar

[1]Cuerpo de conocimientos de la gestión de proyectos *(Project Management Body of Knowledge)*.

cómo se ha iniciado el proyecto y su planificación. Esto hará que el proceso de desarrollo para este proyecto será prácticamente el mismo a la metodología tradicional, pero con mejores especificaciones:

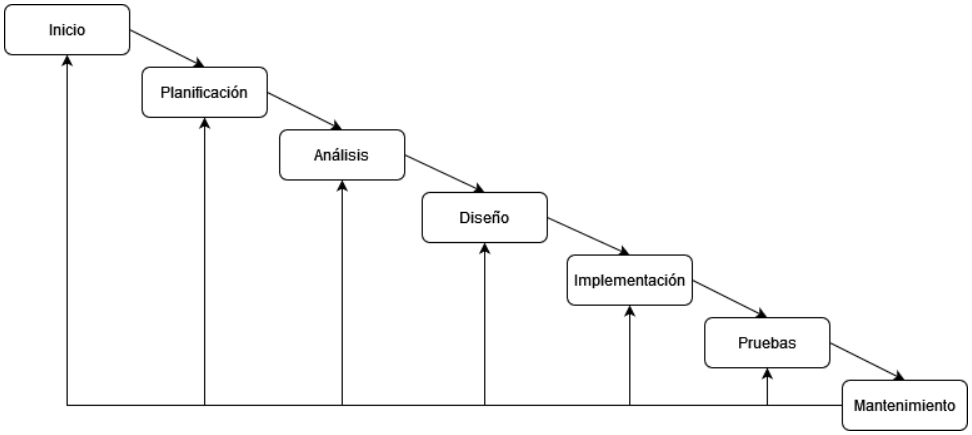


Figura 2.2: Modelo en cascada detallado

1. **Inicio**. Es la fase donde se toma un primer contacto básico con el proyecto y se realiza un proceso de entendimiento en el que se explica con un lenguaje menos formal qué es lo que se quiere conseguir con el desarrollo. Se debe definir un documento que recoja los aspectos más relevantes que se vayan a llevar a cabo y cuáles son los que no, es decir, definir cuál será el alcance del proyecto y los objetivos de este.

2. **Planificación**. Tras saber cómo va a funcionar la aplicación a desarrollar, es necesario establecer un estudio de viabilidad de recursos del proyecto. Algunos de los que se van a tratar serán:

   - Realizar una estimación temporal.
   - Realizar una estimación de recursos humanos, materiales y de *software/hardware*.
   - Anticipar qué riesgos pueden poner en peligro el desarrollo.
   - Definir un plan de prevención para los riesgos del proyecto.
   - Calcular el coste de desarrollo del proyecto.

3. **Análisis**. En esta etapa se realiza una especificación más concreta de los requisitos de la aplicación mediante diagramas de casos de uso. Estos ayudan a entender cómo es el sistema y cómo van a interactuar sus componentes.

4. **Diseño**. Aquí se realizará el modelado de la aplicación mediante diagramas *UML*[2], además de realizar modelos de interfaz para el usuario y, no menos importante, el diseño de la base de datos.

5. **Implementación**. Durante esta etapa es donde se realiza la codificación de toda la información recogida y definida anteriormente. Dependiendo de las tecnologías de desarrollo, es recomendable realizar un periodo de formación para que el estudiante se acostumbre a su entorno y a las buenas prácticas. Es la etapa del proyecto a la que se le dedica la mayor cantidad de tiempo.

[2]Lenguaje unificado de modelado *(Unified Modeling Language)*.

6. **Pruebas**. Antes de desplegar la aplicación, es necesario codificar también una batería de pruebas para comprobar si lo desarrollado en la fase anterior es funcional o tiene algún fallo que suponga la parada completa de la aplicación.

7. **Despliegue y mantenimiento**. A partir de este punto, la aplicación ya es funcional y realiza cada acción como debe acorde a los requisitos que se establecieron durante el inicio del proyecto, por tanto, se procede al despliegue y al mantenimiento en caso de errores durante la ejecución, o la incorporación de módulos adicionales en un futuro.

Toda esta sucesión de fases engloba tareas más específicas con diferentes propósitos que ayudan a entender cómo va a ser el proceso de desarrollo paso a paso. Como consecuencia, es necesario crear un diagrama *WBS*[3] [6] para delimitar qué tareas va a tener cada fase. La Figura 2.3 muestra cómo se han dividido cada una de las tareas que van a componer el proyecto.

2.2. Planificación temporal del proyecto

La duración prevista de la estancia del estudiante en la empresa es de 300 horas, acordadas en una distribución de 5 horas al día, dando lugar a aproximadamente los 3 meses que se han planteado inicialmente para la estancia de prácticas. Las Figuras 2.4 y 2.5 muestran la planificación temporal del proyecto mediante el diagrama de Gantt.

[3]Estructura de desglose del trabajo *(Work Breakdown Structure)*.

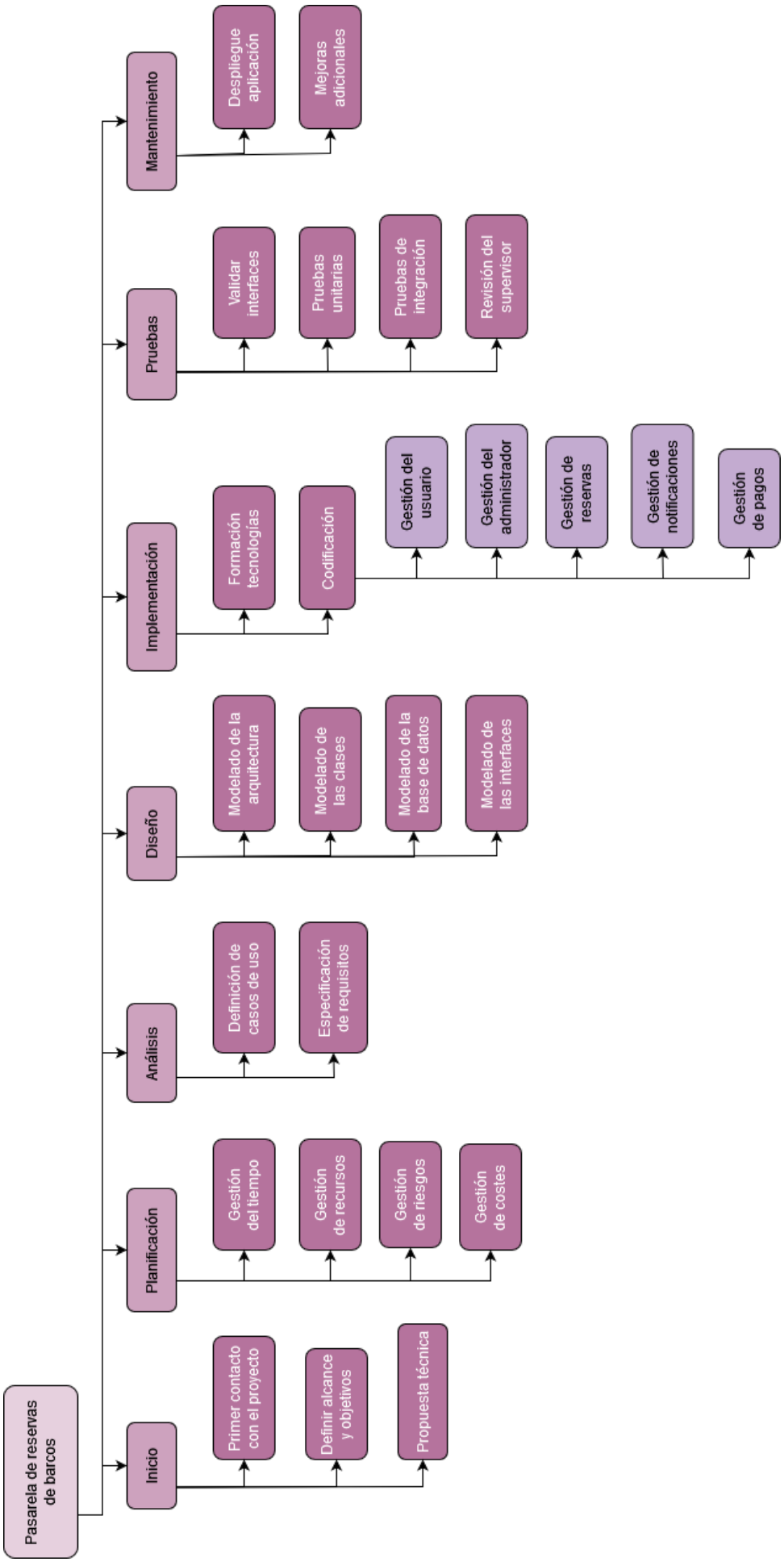


Figura 2.3: Diagrama WBS del proyecto

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Pasarela de reservas de barcos**	**58 días**	**vie 03/03/23**	**mar 23/05/23**	
2		**Reunión de seguimiento**	**51 días**	**lun 06/03/23**	**lun 22/05/23**	
15		**Inicio**	**6 días**	**vie 03/03/23**	**vie 10/03/23**	
16		Primer contacto con el proyecto	1 día	vie 03/03/23	vie 03/03/23	
17		Definir alcance y objetivos	2 días	vie 03/03/23	lun 06/03/23	
18		Redacción propuesta técnica	4 días	mar 07/03/23	vie 10/03/23	16;17
19		**Planificación**	**3 días**	**lun 13/03/23**	**mié 15/03/23**	**15**
20		Gestión del tiempo	2 días	lun 13/03/23	mar 14/03/23	
21		Gestión de recursos	2 días	lun 13/03/23	mar 14/03/23	
22		Gestión de costes	2 días	lun 13/03/23	mar 14/03/23	
23		Gestión de riesgos	1 día	mié 15/03/23	mié 15/03/23	20;21;22
24		**Análisis**	**6 días**	**jue 16/03/23**	**jue 23/03/23**	**19**
25		Definición de casos de uso	3 días	jue 16/03/23	lun 20/03/23	
26		Especificación de requisitos	3 días	mar 21/03/23	jue 23/03/23	25
27		**Diseño**	**7 días**	**vie 24/03/23**	**lun 03/04/23**	**24**
28		Modelado de la arquitectura	2 días	vie 24/03/23	lun 27/03/23	
29		Modelado de las clases	2 días	vie 24/03/23	lun 27/03/23	
30		Modelado de la base de datos	3 días	vie 24/03/23	mar 28/03/23	
31		Modelado de las interfaces	4 días	mié 29/03/23	lun 03/04/23	29;30
32		**Implementación**	**30 días**	**mar 04/04/23**	**lun 15/05/23**	**27**
33		Formación de las tecnologías	5 días	mar 04/04/23	lun 10/04/23	
34		**Codificación**	**25 días**	**mar 11/04/23**	**lun 15/05/23**	**33**
35		Gestión del usuario	25 días	mar 11/04/23	lun 15/05/23	
36		Gestión del administrador	25 días	mar 11/04/23	lun 15/05/23	
37		Gestión de reservas	25 días	mar 11/04/23	lun 15/05/23	
38		Gestión de notificaciones	25 días	mar 11/04/23	lun 15/05/23	
39		Gestión de pagos	25 días	mar 11/04/23	lun 15/05/23	
40		**Pruebas**	**5 días**	**mar 16/05/23**	**lun 22/05/23**	**32;34**
41		Validar interfaces	3 días	mar 16/05/23	jue 18/05/23	
42		Pruebas unitarias	5 días	mar 16/05/23	lun 22/05/23	
43		Pruebas de intergación	5 días	mar 16/05/23	lun 22/05/23	
44		<Revisión con el supervisor>	0 días	lun 22/05/23	lun 22/05/23	41;42;43
45		**Mantenimiento**	**1 día**	**mar 23/05/23**	**mar 23/05/23**	**40**
46		Propuesta de mejoras adicionales	1 día	mar 23/05/23	mar 23/05/23	
47		Despliegue de la aplicación	1 día	mar 23/05/23	mar 23/05/23	
48		<Fin y cierre del proyecto>	0 días	mar 23/05/23	mar 23/05/23	45

Figura 2.4: Desglose de tareas del proyecto en Microsoft Project

Figura 2.5: Diagrama de Gantt del proyecto

2.3. Costes

Para computar el coste total del desarrollo del proyecto, se van a tener en cuenta los siguientes tipos de recursos: *software*, *hardware*, humanos e indirectos.

- **Software**. Los recursos lógicos escogidos para el desarrollo comprenden las herramientas que utilizará el estudiante para poder conseguir el producto final. Por su parte, ha trabajado con multitud de herramientas para generar *software* de calidad. La gran mayoría de *software* de pago con la que ha trabajado ha sido gratuita debido a las licencias de estudiante que proporciona el centro universitario, computando una mejor optimización de costes que puedan repercutir en el proyecto. Mencionar también que para el sistema operativo se ha escogido una licencia de tipo OEM[4] debido a que el ordenador va a ser usado únicamente para el desarrollo. Para la gestión del dominio mencionar que la duración de la licencia es anual, con lo que habrá que tener en cuenta la duración del proyecto. En el Cuadro 2.1 se recogen los recursos a utilizar en el proyecto.

Recurso	**Coste**	**Meses**	**Coste total**
Windows 10 OEM	15,00 €	-	15,00 €
Visual Studio Code	0,00 €	3	0,00 €
MagicDraw Personal Edition	0,00 €	3	0,00 €
Laravel	0,00 €	3	0,00 €
Github	0,00 €	3	0,00 €
Balsamiq Wireframes	0,00 €	3	0,00 €
Adobe Color	0,00 €	3	0,00 €
Gestión de dominio	20,00 €	3	5,00 €
Total	35,00 €	-	20,00 €

Cuadro 2.1: Desglose de costes de los recursos *software* del proyecto

- **Hardware**. Los recursos físicos que se van a proporcionar al estudiante durante su estancia será un equipo bien preparado para el desarrollo de una aplicación web acompañado de una apropiada colección de periféricos. Se va a incluir un ordenador de sobremesa con un procesador de última generación, un monitor de 27 pulgadas y un combo de teclado y ratón. Por otro lado, se va a escoger un servidor externo para desplegar la aplicación. Debido a su cómodo precio, la empresa lo ha seleccionado expresamente para alojar una gran cantidad de servicios web. El Cuadro 2.2 desglosa el coste estimado para estos componentes.

  El equipo tiene un coste inicial fijo para cada componente, a partir de ahí se va a suponer que, para un equipo informático, el tiempo de amortización es de 5 años. Finalmente, hay que tener en cuenta la duración del desarrollo, que, como se ha mencionado anteriormente, va a ser de 300 horas. El servidor va a tener un tratamiento especial ya que su precio es anual, entonces se calculará la proporción a los meses de desarrollo.

- **Humanos**. Los trabajadores que van a estar a cargo del proyecto van a ser el estudiante con contrato de prácticas y su supervisor. El estudiante tendrá asignadas las 300 horas que dura la estancia de prácticas, mientras que el supervisor estará disponible para la

[4]Fabricante de equipo original *(Original Equipment Manufacturer)*.

Recurso	Coste	Meses	Coste total amortizado
Ordenador	450,00 €	3	3,13 €
Monitor	120,00 €	3	0,83 €
Teclado	10,00 €	3	0,07 €
Ratón	10,00 €	3	0,07 €
Servidor	200,00 €	12	50,00 €
Total	790,00 €	-	54,10 €

Cuadro 2.2: Desglose de costes de los recursos *hardware* del proyecto

resolución de dudas y problemáticas que puedan surgir durante el desarrollo, todo para que el estudiante aprenda mientras está trabajando en un entorno real. El estudiante no percibe ninguna remuneración, así que, tras consultar con la empresa y consultando referencias externas, se ha supuesto un salario bruto promedio para un desarrollador junior de 22.513 € [7]. Además, también se ha consultado mediante fuentes externas el salario bruto promedio para un desarrollador senior, siendo de 35.377 € [8]. El Cuadro 2.3 refleja el coste en base al tiempo dedicado para los 3 meses de duración del proyecto.

Recurso	Salario mensual bruto	Horas	Coste total
Desarrollador junior	1.876,08 €	300	7.035,00 €
Desarrollador senior (supervisor)	2.948,08 €	30	1.105,53 €
Total	-	-	8.140,53 €

Cuadro 2.3: Desglose de costes de los recursos humanos del proyecto

- **Indirectos**. El salario de los trabajadores y el uso del equipo informático de la oficina requieren de un uso de recursos variados que proporciona cualquier empresa como es el uso de electricidad, internet o el alquiler, entre otros. Así, a cada trabajador se le aplicará una reducción acordada por la empresa de un 5 % para poder realizar esos gastos que repercuten en el entorno de trabajo. El Cuadro 2.4 representa brevemente esa reducción:

Recurso	Salario mensual bruto	Coste total
Desarrollador junior	1.876,08 €	93,80 €
Desarrollador senior (supervisor)	2.948,08 €	147,40 €
Total	-	241,20 €

Cuadro 2.4: Desglose de costes de los recursos indirectos del proyecto

Con todos los costes ya calculados se procede a combinarlos para calcular el coste total del proyecto (véase el Cuadro 2.5). Cabe destacar que los recursos indirectos van incluidos con los recursos humanos.

Tipo de recurso	Costes totales
Software	20,00 €
Hardware	54,10 €
Humanos	8.140,53 €
Indirectos	241,20 €
Total	8.455,83 €

Cuadro 2.5: Costes totales de los recursos del proyecto

2.4. Riesgos

Debido a la envergadura del proyecto, es muy probable que surjan dificultades de diversos tipos durante el desarrollo. Por esto, es necesario definir una gestión de riesgos que puedan poner en peligro tanto el proyecto como el producto final, con el fin de reducir cualquier problema. Se proponen una serie de riesgos comunes como los del Cuadro 2.6 que, al estar en el departamento de desarrollo, son los más susceptibles de aparecer. En el Cuadro 2.7 se detallan adecuadamente cada uno de los riesgos. Además, se incluye un plan de contingencia en el Cuadro 2.8 para prevenir y solucionar cualquiera de estos riesgos en caso de manifestarse.

Riesgo	Descripción del riesgo	Tipo de riesgo
R1	Desconocimientos tecnologías y herramientas de desarrollo	Riesgo del proyecto y producto
R2	Mala distribución temporal de las tareas del proyecto	Riesgo del proyecto
R3	Ambigüedad en la definición de requisitos	Riesgo del producto
R4	Dificultad del desarrollo en el entorno de trabajo	Riesgo del proyecto

Cuadro 2.6: Lista de riesgos del proyecto

Riesgo	Análisis
R1	*Magnitud:* Media
	Impacto: Puede provocar retrasos en la fase de implementación del proyecto
	Descripción: El desarrollador principal cuenta con una experiencia nula o deficiente en cuanto al uso de las tecnologías de desarrollo
	Indicadores: El desarrollo puede colapsar o generar una tasa de errores elevada durante la fase de implementación
R2	*Magnitud:* Alta
	Impacto: Puede generar conflicto con el cumplimiento de plazos, haciendo que el proyecto sea inviable
	Descripción: La estimación inicial no representa la realidad del proyecto, ya que la duración real de las tareas difiere bastante de la duración estimada. Puede dar lugar a aplazamientos forzados
	Indicadores: La definición temporal no ha sido revisada exhaustivamente. No se ha planificado con claridad del proyecto
R3	*Magnitud:* Alta
	Impacto: Puede provocar retrasos en la planificación. Sería necesario hacer un reajuste de las fases hasta el punto actual
	Descripción: Los requisitos se han definido incorrectamente o presentan alguna ambigüedad
	Indicadores: El cliente no ha expresado de manera concisa las funcionalidades y los servicios que ofrece su producto. Surgen dudas durante el desarrollo sobre funcionalidades
R4	*Magnitud:* Variable
	Impacto: Un ambiente deficiente en el entorno de trabajo puede dar lugar al abandono del proyecto o un mal producto final
	Descripción: Por cualquier motivo o razón, un desarrollador decide abandonar el proyecto
	Indicadores: Desmotivación, urgencia imprevista o ambiente hostil

Cuadro 2.7: Descripción detallada de los riesgos

Riesgo	Plan de prevención	Plan de contingencia
R1	Dedicar un periodo de formación para las tecnologías con las que no haya experiencia de uso	Si alguna de las tecnologías es conocida por el supervisor, establecer una reunión con un listado de dudas que puedan surgir de dichas tecnologías
R2	Será necesario consultar con el supervisor la viabilidad temporal de la estimación del proyecto	Si se da el caso de una estimación incorrecta, puede haber cambios en la planificación dependiendo de la fase donde se haya detectado. Si ya es muy tarde, se propondrá al supervisor realizar un aplazamiento forzoso
R3	Especificar cada uno de los requisitos que generen dudas o información equívoca que el cliente desea. Mantener una comunicación constante con el cliente	Dependiendo de la fase donde se generen dudas sobre los requisitos, será más sencillo hacer una revisión de la planificación o comprobar, si el tiempo lo permite, solventarlo lo más rápido posible
R4	Dependiendo del contexto y la situación de cada persona implicada, establecer un marco comunicativo para realizar una mejora sustancial del entorno	Si un imprevisto o acción desencadena una situación similar, valorar posibles cambios del entorno de trabajo. Si se da la opción de teletrabajo, cambiar el entorno para adecuarse a la nueva situación

Cuadro 2.8: Definición de los planes de contingencia de los riesgos del proyecto

2.5. Seguimiento del proyecto

En esta sección se expone en detalle cómo se ha llevado a cabo el seguimiento del desarrollo de la aplicación. Debido que al inicio se ha definido una metodología tradicional, se sugirió realizar puntos de control del estado al final de cada fase y, además, al finalizar las funcionalidades más críticas.

Cabe destacar que desde la empresa ofrecieron completa libertad en cuanto a la gestión de cada una de las fases, así que me dediqué a usarlas como referencia temporal junto a los informes quincenales. Los informes recogían qué tareas se iban completando y ciertos matices que surgían a medida que se estudiaba en profundidad los requisitos y las secciones de código que se iban implementando.

El cliente también quería saber en algunos momentos cuál era el progreso actual del desarrollo, punto que he considerado de gran valor para seguir realizando pequeños ajustes cuando correspondía. Mientras tanto, el supervisor también estaba atento a cambios que consideré importantes o ligeras desviaciones para ajustar el desarrollo a las necesidades del cliente.

2.5.1. Fase de Inicio y Planificación

Las dos primeras fases estuvieron predominadas por la familiarización con la empresa y donde se recogió toda la información relacionada con la Propuesta Técnica. Por mi parte, investigué sobre distintos servicios de reservas para tener una idea de cómo se podría adaptar los requisitos del cliente teniendo como referencia aplicaciones populares. También, tracé un breve plan de estudio sobre las tecnologías que se proponían y me dispuse a realizar pequeños *spikes*[5] para valorar cuáles serían las elegidas.

2.5.2. Fase de Análisis y Diseño

Las siguientes fases se centraron en convertir los requisitos del cliente en casos de uso funcionales, por los que se definen sus comportamientos con el sistema. Tras esto, me dispuse a realizar varios esquemas para representar la base de datos. Gracias al supervisor y a una reunión que hubo con el cliente, matizamos ciertas ambigüedades que surgieron con el modelado. A partir de este punto, ya tuve claro cuáles fueron las tecnologías que se usarían para el desarrollo, las cuales se comentarán más adelante.

2.5.3. Fase de Implementación

La fase más extensa del proyecto engloba todo el desarrollo de la aplicación a nivel de código. Los seguimientos estuvieron más limitados a los informes quincenales y a los recursos que se

[5]En desarrollo de software, concretamente en la metodología de desarrollo dirigida por tests, es un pequeño proyecto cuyo objetivo es estudiar e investigar tecnologías con el fin de entender de clarificar requisitos, estimaciones o tecnologías en sí.

iban implementando. Durante el desarrollo hubo que realizar varios ajustes al modelo de la base de datos por incompatibilidades. Aun así, dichos ajustes no suponían un gran esfuerzo porque no desestabilizaba lo que ya había desarrollado. Para acabar esta fase, también se incluyó el despliegue de la aplicación, el cual dio varios problemas que fueron previstos y solucionados.

2.5.4. Fase de Pruebas y Mantenimiento

Para terminar, se dedicaron unas sesiones para realizar pruebas con el supervisor, con el fin de corroborar que las acciones que realiza un usuario son las adecuadas. Para añadir, también se dedicaron varias sesiones para hacer refactorización en el código, debido a ciertos apartados donde quedaban muy cargados con mucha lógica, como en los controladores. Como extra, realicé también tests de usuario que supervisé en cada momento y que fueron bastante útiles para realizar algunas mejoras.

Capítulo 3

Análisis del sistema

3.1. Definición de requisitos

La definición de requisitos comprende analizar las funcionalidades que se definieron con el cliente y crear una colección de requisitos que recoja todas las necesidades de la aplicación. Como ya se definió anteriormente, el alcance funcional y los objetivos establecen un buen punto de partida para concretar las funcionalidades. Mediante un diagrama de casos de uso, se representa el sistema a desarrollar, junto con los actores que van a interactuar con el mismo, y con las funcionalidades en sí, siendo los casos de uso. La Figura 3.1 muestra cómo se ha modelado la aplicación.

Se puede apreciar la presencia de los actores que van a interactuar con el sistema: el usuario no registrado, el usuario registrado (representado en el diagrama como Cliente), el administrador y la pasarela externa de pago. Además, el sistema está dividido en dos apartados. El *back-office* representa la sección donde se hará la gestión interna, como la creación de barcos y tarifas, las devoluciones o la gestión de las fechas de salida. El *front-office*[1] recoge todas las funcionalidades básicas que un usuario registrado puede realizar, junto a la interacción con el TPV[2] externo.

A grandes rasgos, la mayoría de casos de uso se encuentran agrupados bajo la nomenclatura "Gestionar X". Normalmente se integran las operaciones de creación, modificación, consulta y eliminación, a no ser que haya que especificar concretamente alguna otra operación o clarificar algunas de las mencionadas operaciones. Teniendo esto en cuenta, el Cuadro 3.1 recoge los casos de uso del sistema.

[1] Sección de la aplicación que va a emplear el usuario final.
[2] Terminal Punto de Venta.

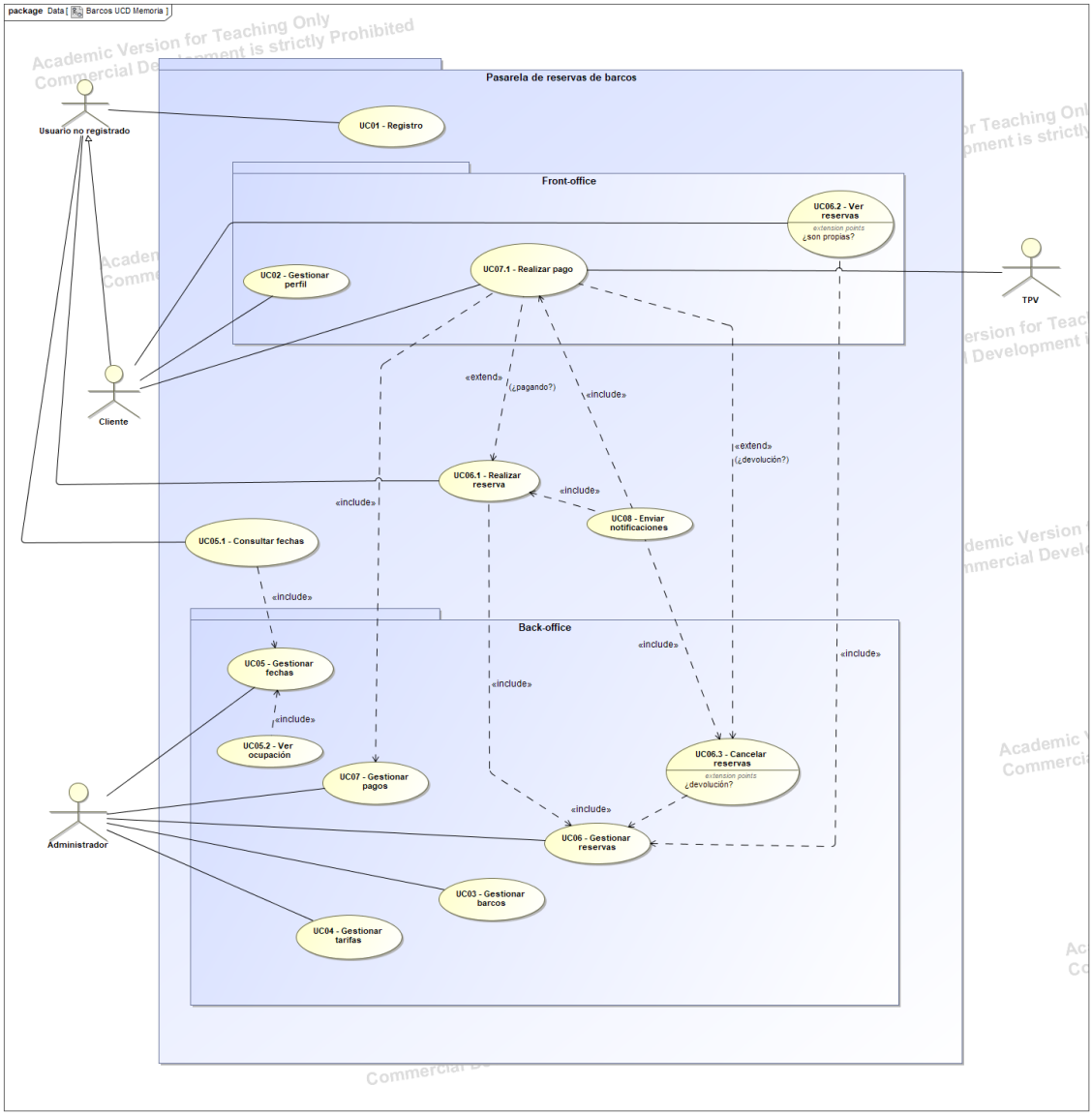


Figura 3.1: Diagrama de casos de uso

Identificador	**Caso de uso**
UC01	Registro
UC02	Gestionar perfil
UC03	Gestionar barcos
UC04	Gestionar tarifas
UC05	Gestionar fechas
UC05.1	Consultar fechas
UC05.2	Ver ocupación
UC06	Gestionar reservas
UC06.1	Realizar reserva
UC06.2	Ver reservas
UC06.3	Cancelar reservas
UC07	Gestionar pagos
UC07.1	Realizar pago
UC08	Enviar notificaciones

Cuadro 3.1: Desglose de los casos de uso del sistema

3.2. Análisis de requisitos

Cada caso de uso necesita ser definido con más claridad mediante una especificación concisa sobre cómo se va a comportar en el sistema. Cada caso de uso ha sido especificado mediante un modelo estandarizado que recoge la identificación del caso de uso en cuestión, la descripción de encadenamientos y, si procede, comentarios adicionales que se quieran mencionar.

La primera colección de cuadros, desde el Cuadro 3.2 hasta el Cuadro 3.15 recoge todas las especificaciones de los casos de uso comentados anteriormente. Posteriormente, también se recoge una colección de los requisitos de datos a partir del Cuadro 3.16 hasta el Cuadro 3.23, analizando en detalle cómo se van a tratar los diferentes tipos de datos.

Requisito funcional UC01	
Identificador	UC01
Nombre	Registro
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir a un usuario no registrado crear una cuenta
Actor principal	Usuario no registrado
Precondición	El usuario no está registrado en el sistema
Disparador	El usuario introduce sus datos en un formulario del sistema
Secuencia	1. Un cliente quiere realizar una nueva reserva o registrarse en el sistema 2. El cliente accede al formulario de registro 3. El cliente introduce sus datos 4. Los datos se validan y se efectúa el registro
Excepciones	Algunos de los campos son incorrectos. El usuario ya está registrado en el sistema
Prioridad	Alta

Cuadro 3.2: Requisito funcional UC01: Registro

Requisito funcional UC02	
Identificador	UC02
Nombre	Gestionar perfil
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir a un usuario modificar sus datos
Actor principal	Cliente
Actor secundario	Administrador
Precondición	El usuario está dentro del sistema
Disparador	El usuario quiere ver y/o modificar alguno de sus datos
Secuencia	1. El usuario selecciona su perfil en el sistema 2. El usuario accede a su menú 3. El usuario decide qué acciones realizar en su perfil 4. Modificar *a*) El usuario introduce nuevos datos en los campos *b*) El sistema valida los datos y efectúa los cambios 5. Eliminar cuenta *a*) El usuario selecciona eliminar su cuenta *b*) El usuario confirma su eliminación
Excepciones	Algún campo a modificar es inválido
Prioridad	Media

Cuadro 3.3: Requisito funcional UC02: Gestionar perfil

Requisito funcional UC03	
Identificador	UC03
Nombre	Gestionar barcos
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir crear, consultar, modificar y borrar barcos
Actor principal	Administrador
Actor secundario	Cliente
Precondición	-
Disparador	Un usuario quiere consultar información de los barcos. Un administrador quiere realizar modificaciones a los barcos
Secuencia	1. El administrador accede al menú de los barcos 2. El administrador decide qué acciones realizar con los barcos 3. Crear *a*) El administrador introduce los datos en los campos del barco *b*) El sistema valida los datos y efectúa la inserción 4. Modificar *a*) El administrador introduce nuevos datos en los campos del barco *b*) El sistema valida los datos y efectúa los cambios 5. Eliminar *a*) El administrador selecciona eliminar un barco *b*) El administrador confirma la eliminación
Excepciones	El sistema está en mantenimiento
Prioridad	Alta

Cuadro 3.4: Requisito funcional UC03: Gestionar barcos

Requisito funcional UC04	
Identificador	UC04
Nombre	Gestionar reservas
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir crear, consultar, modificar y borrar tarifas
Actor principal	Administrador
Actor secundario	Cliente
Precondición	El usuario está dentro del sistema
Disparador	Un usuario quiere consultar información de las tarifas. Un administrador quiere realizar modificaciones a las tarifas
Secuencia	1. El administrador accede al menú de las tarifas 2. El administrador decide qué acciones realizar con las tarifas 3. Crear *a*) El administrador introduce los datos en los campos de la tarifa *b*) El sistema valida los datos y efectúa la inserción 4. Modificar *a*) El administrador introduce nuevos datos en los campos de la tarifa *b*) El sistema valida los datos y efectúa los cambios 5. Eliminar *a*) El administrador selecciona eliminar una tarifa *b*) El administrador confirma la eliminación
Excepciones	El sistema está en mantenimiento
Prioridad	Alta
Comentarios	-

Cuadro 3.5: Requisito funcional UC04: Gestionar tarifas

Requisito funcional UC05	
Identificador	UC05
Nombre	Gestionar fechas
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir crear, modificar y borrar fechas de salida
Actor principal	Administrador
Precondición	El usuario está dentro del sistema
Disparador	Un administrador quiere realizar modificaciones a las fechas de salida
Secuencia	1. El administrador accede al menú del calendario 2. El administrador decide qué acciones realizar con las fechas de salida 3. Crear *a*) El administrador selecciona un día en el calendario *b*) El administrador introduce los datos en los campos de la fecha de salida *c*) El sistema valida los datos y efectúa la inserción 4. Modificar *a*) El administrador introduce nuevos datos en los campos de la fecha de salida *b*) El sistema valida los datos y efectúa los cambios 5. Eliminar *a*) El administrador selecciona eliminar una fecha de salida *b*) El administrador confirma la eliminación
Excepciones	Día inválido por algún motivo. Se asigna más capacidad al día que la del barco.
Prioridad	Alta

Cuadro 3.6: Requisito funcional UC05: Gestionar fechas

Requisito funcional UC05.1	
Identificador	UC05.1
Nombre	Consultar fechas
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir consultar las fechas de salida
Actor principal	Visitante y Cliente
Actor secundario	Administrador
Precondición	El usuario está dentro del sistema
Disparador	Un usuario quiere consultar los días en los que hay excursiones programadas
Secuencia	1. El usuario accede al menú del calendario 2. El usuario puede ver qué días hay excursiones programadas
Excepciones	Un día no tiene fecha de salida
Prioridad	Alta

Cuadro 3.7: Requisito funcional UC05.1: Consultar fechas

Requisito funcional UC05.2	
Identificador	UC05.2
Nombre	Ver ocupación
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir mostrar la ocupación actual de una fecha de salida
Actor principal	Administrador
Precondición	El usuario está dentro del sistema
Disparador	Un administrador quiere consultar la ocupación de un día con el fin de evaluar una fecha de salida
Secuencia	1. El administrador accede al menú del calendario 2. El administrador selecciona una fecha de salida 3. El administrador puede ver información relativa a la capacidad actual
Excepciones	Un día no tiene fecha de salida
Prioridad	Alta

Cuadro 3.8: Requisito funcional UC05.2: Ver ocupación

Requisito funcional UC06	
Identificador	UC06
Nombre	Gestionar reservas
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir ver, crear y modificar reservas
Actor principal	Administrador
Actor secundario	Cliente y Visitante
Precondición	El usuario está dentro del sistema
Disparador	El administrador necesita realizar alguna acción con las reservas. El usuario quiere realizar alguna acción disponible con las reservas
Secuencia	▪ Si el usuario es un Administrador o Cliente: 1. El usuario accede al listado de reservas 2. El usuario selecciona una reserva para realizar una acción: • Realizar reserva → UC06.1 • Ver detalles de las reservas → UC06.2 • Cancelar una reserva → UC06.3 • Cambiar datos de una reserva ▪ Si el usuario es un Visitante procede a realizar una reserva → UC06.1
Excepciones	No hay reservas en el sistema
Prioridad	Alta
Comentarios	El cliente solo podrá realizar reservas y consultar las suyas. Las demás operaciones son exclusivas del administrador, que tiene todas las funciones disponibles

Cuadro 3.9: Requisito funcional UC06: Gestionar reservas

Requisito funcional UC06.1	
Identificador	UC06.1
Nombre	Realizar reserva
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir la creación de reservas
Actor principal	Cliente y Visitante
Actor secundario	Administrador
Precondición	-
Disparador	Un usuario quiere realizar una excursión en barco
Secuencia	1. El usuario accede al menú del calendario 2. El usuario selecciona una fecha de salida 3. El usuario introduce los pasajeros 4. Se validan los campos del formulario 5. Se crea la reserva correctamente 6. Se genera un pago asociado a la reserva creada → UC07
Excepciones	No hay plazas suficientes. Día inválido por algún motivo
Prioridad	Alta

Cuadro 3.10: Requisito funcional UC06.1: Realizar reserva

Requisito funcional UC06.2	
Identificador	UC06.2
Nombre	Ver reservas
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir mostrar un listado de las reservas hechas
Actor principal	Cliente y Administrador
Precondición	El usuario está dentro del sistema y hay reservas en el sistema
Disparador	El usuario quiere consultar sus reservas. El administrador quiere consultar las reservas de los usuarios
Secuencia	1. El usuario selecciona la sección de Reservas" 2. El sistema muestra las reservas que hay hechas 3. Si el usuario es Cliente: *a*) El sistema muestra las reservas propias del cliente *b*) El cliente decide ver información o realizar otra reserva 4. Si el usuario es Administrador: *a*) El sistema muestra todas las reservas hechas en el sistema *b*) El administrador decide qué acción realizar con las reservas
Excepciones	No hay reservas en el sistema
Prioridad	Alta

Cuadro 3.11: Requisito funcional UC06.2: Ver reservas

Requisito funcional UC06.3	
Identificador	UC06.3
Nombre	Cancelar reservas
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir la cancelación de reservas
Actor principal	Administrador
Precondición	El administrador está dentro del sistema y hay reservas en el sistema
Disparador	Por diversos motivos, la excursión asociada a una reserva no se puede realizar
Secuencia	1. El administrador valora que un viaje no se puede efectuar 2. El administrador selecciona ver las reservas 3. El administrador selecciona eliminar la reserva en concreto 4. La eliminación de la reserva se efectúa correctamente
Excepciones	No hay reservas en el sistema
Prioridad	Alta
Comentarios	La cancelación estará sujeta a las condiciones proporcionadas

Cuadro 3.12: Requisito funcional UC06.3: Cancelar reservas

Requisito funcional UC07	
Identificador	UC07
Nombre	Gestionar pagos
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir la ver, crear y modificar pagos
Actor principal	Administrador
Precondición	El usuario está dentro del sistema
Disparador	Se desencadena la creación de una reserva
Secuencia	1. El usuario accede a la visualización de sus reservas 2. El usuario decide qué acción realizar: ▪ Realizar el pago → UC07.1 ▪ Modificar el pago
Excepciones	La reserva asociada a un pago está cancelada
Prioridad	Alta

Cuadro 3.13: Requisito funcional UC07: Gestionar pagos

Requisito funcional UC07.1	
Identificador	UC07.1
Nombre	Realizar pago
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir el pago de una reserva mediante el TPV
Actor principal	Cliente y TPV
Precondición	El usuario está dentro del sistema
Disparador	El usuario ha seleccionado la opción de .Efectuar pago"
Secuencia	1. El usuario introduce sus datos de pago en el TPV 2. El TPV valida los datos y efectúa la transacción 3. El TPV confirma el pago
Excepciones	Algunos campos de los datos de pago son incorrectos
Prioridad	Alta

Cuadro 3.14: Requisito funcional UC07.1: Realizar pagos

Requisito funcional UC08	
Identificador	UC08
Nombre	Enviar notificaciones
Autor	Alejandro Guinot Pérez
Descripción	El sistema debe permitir enviar información a los usuarios sobre las reservas
Actor principal	-
Precondición	-
Disparador	Un usuario ha realizado una reserva, habiéndose pagado al completo o no. Un administrador ha realizado una devolución. Un usuario ha solicitado una cancelación
Secuencia	1. Si el usuario no paga el 100 % de la reserva: ▪ El sistema genera una notificación personalizada con el pago restante de la reserva 2. Si el usuario paga el 100 % de la reserva: ▪ El sistema genera una notificación personalizada con los datos de la reserva 3. Cuando el usuario solicita una cancelación: ▪ El sistema genera una notificación personalizada con la cantidad que va a recibir de vuelta 4. Cuando el administrador realiza una devolución: ▪ El sistema genera una notificación personalizada con la cantidad devuelta
Excepciones	-
Prioridad	Media / Alta

Cuadro 3.15: Requisito funcional UC08: Enviar notificaciones

Requisito de datos RD01	
Identificador	RD01
Nombre	Datos de registro
Versión	1.0
Autor	Alejandro Guinot Pérez
Fuentes	Ley de Protección de Datos
Requisitos asociados	UC01, UC06.1
Datos específicos	Nombre completo del usuario, número de teléfono, dirección de correo electrónico, contraseña de acceso
Ocurrencias	Muy frecuente: cada vez que un visitante quiera realizar una reserva
Importancia	Alta

Cuadro 3.16: Requisito de datos RD01: Datos de registro

Requisito de datos RD02	
Identificador	RD02
Nombre	Autenticación en el sistema
Versión	1.0
Autor	Alejandro Guinot Pérez
Fuentes	Ley de Protección de Datos
Requisitos asociados	UC02, RD03
Datos específicos	Credenciales de acceso del usuario: dirección de correo electrónico y contraseña
Ocurrencias	Variable: cada vez que un usuario haya hecho uso del servicio previamente quiera volver a utilizarlo, o consultar la aplicación
Importancia	Media-Alta

Cuadro 3.17: Requisito de datos RD02: Autenticación en el sistema

Requisito de datos RD03	
Identificador	RD03
Nombre	Gestión del usuario
Versión	1.0
Autor	Alejandro Guinot Pérez
Fuentes	Ley de Protección de Datos
Requisitos asociados	UC02, RD02
Datos específicos	Credenciales de acceso del usuario: dirección de correo electrónico y contraseña. Información personal del usuario: nombre completo y teléfono de contacto
Ocurrencias	Variable: cada vez que un usuario haya hecho uso del servicio previamente y necesite consultar el estado de algunos recursos propios como reservas o pagos
Importancia	Alta

Cuadro 3.18: Requisito de datos RD03: Gestión del usuario

Requisito de datos RD04	
Identificador	RD04
Nombre	Gestión de los barcos
Versión	1.0
Autor	Alejandro Guinot Pérez
Fuentes	Excursiones Columbretes Castellón
Requisitos asociados	UC03, UC05, UC06, UC07
Datos específicos	Información detallada de los barcos: nombre, capacidad y otras características
Ocurrencias	Media-Alta: cada vez que sea necesario cambiar información relevante del barco o añadir más barcos a la aplicación
Importancia	Alta

Cuadro 3.19: Requisito de datos RD04: Gestión de los barcos

Requisito de datos RD05	
Identificador	RD05
Nombre	Gestión de las tarifas
Versión	1.0
Autor	Alejandro Guinot Pérez
Fuentes	Excursiones Columbretes Castellón
Requisitos asociados	UC04, UC05, UC06, UC07
Datos específicos	Información detallada de los precios por persona: precio para adultos y precio para niños
Ocurrencias	Alta: cada vez que sea necesario añadir tarifas en base a ofertas, promociones o por el uso de barcos
Importancia	Alta

Cuadro 3.20: Requisito de datos RD05: Gestión de las tarifas

Requisito de datos RD06	
Identificador	RD06
Nombre	Gestión de las fechas de salida
Versión	1.0
Autor	Alejandro Guinot Pérez
Fuentes	Excursiones Columbretes Castellón
Requisitos asociados	UC05, RD04, RD05
Datos específicos	Información detallada de la fecha de salida: barco que va a ser utilizado, tarifa impuesta, cantidad de pasajeros disponibles y tipo de pago asociado a dicho día
Ocurrencias	Alta: cada vez que sea necesario añadir fechas de salida o realizar un control de plazas, procurando que no haya una sobreventa de plazas
Importancia	Alta

Cuadro 3.21: Requisito de datos RD06: Gestión de las fechas de salida

Requisito de datos RD07	
Identificador	RD07
Nombre	Gestión de reservas
Versión	1.0
Autor	Alejandro Guinot Pérez
Fuentes	Excursiones Columbretes Castellón
Requisitos asociados	UC06, RD03, RD04, RD06
Datos específicos	Información detallada de una reserva: usuario que la ha realizado, barco que va a ser utilizado, cantidad de pasajeros que harán uso de la reserva y tipo de pago que tiene asociado la fecha de salida
Ocurrencias	Alta: en caso de que hayan cancelaciones o aplazamientos de salidas, mantener la integridad de datos como capacidad o tipos de pago con el fin de notificar al usuario las acciones que se van a realizar
Importancia	Alta

Cuadro 3.22: Requisito de datos RD07: Gestión de reservas

Requisito de datos RD08	
Identificador	RD08
Nombre	Gestión de pagos
Versión	1.0
Autor	Alejandro Guinot Pérez
Fuentes	Excursiones Columbretes Castellón
Requisitos asociados	UC02, UC06, UC07, RD06, RD07
Datos específicos	Información detallada del pago: usuario asociado, reserva asociada y cantidad restante por pagar
Ocurrencias	Alta: al generar una reserva y, a su vez, dicho pago, tratar los datos del pago para generar en la pasarela de pago externa el formulario de pago
Importancia	Alta
Comentarios	La aplicación no guarda datos bancarios. Forma parte de la pasarela de pago externa

Cuadro 3.23: Requisito de datos RD08: Gestión de pagos

Capítulo 4

Diseño del sistema

4.1. Diseño de la base de datos

El diseño de la base de datos se vio influenciado por técnicas usadas durante el grado. En primer lugar, y tomando como referencia los casos de uso del Cuadro 3.1, se identificaron rápidamente cuáles serían las clases principales que formarían las entidades en la base de datos. La Figura 4.1 ilustra cómo se han estructurado las clases y las relaciones que hay entre ellas para formar la base de datos.

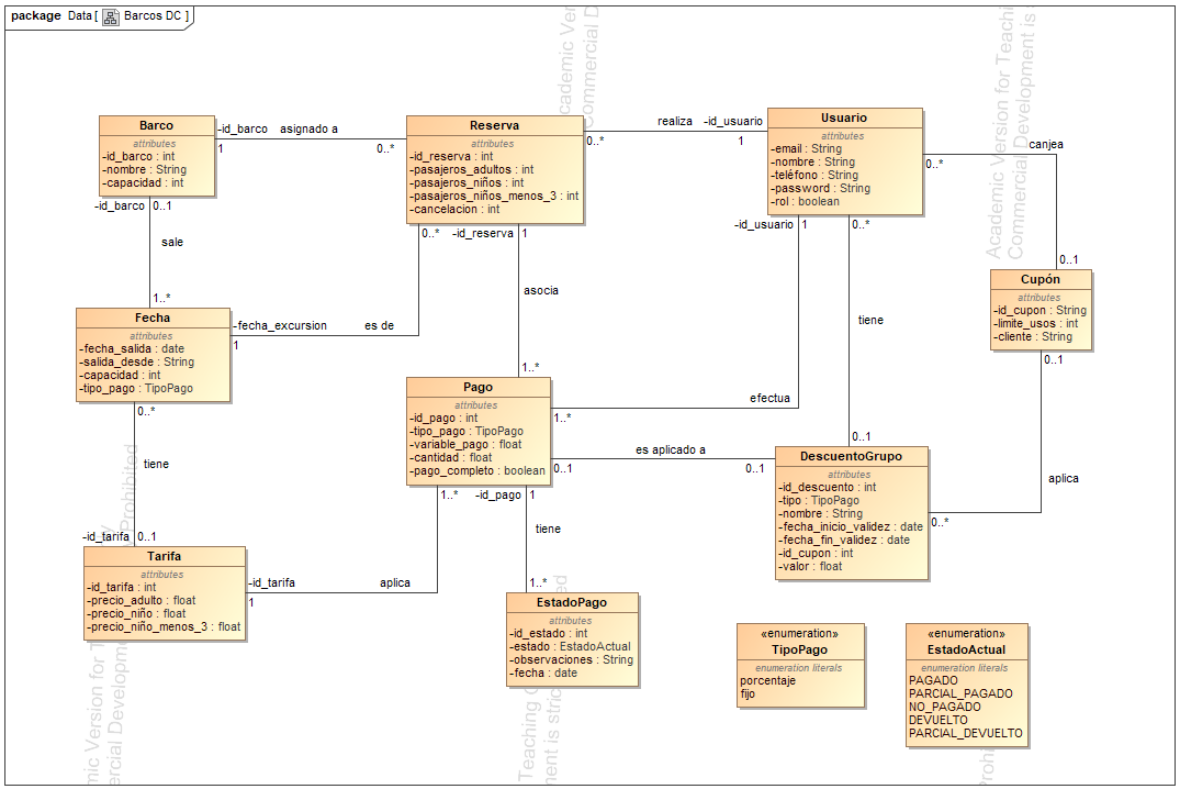


Figura 4.1: Diagrama de clases y estructura de la base de datos

La base de datos se ha implementado mediante el mapeador objeto-relacional *Eloquent* [9]. Esta librería trae funciones que hace que la interacción con las tablas de la base de datos sea muy intuitiva, ya que van ligadas a cada uno de los modelos que conforman la aplicación. El tipo de base de datos que se ha utilizado ha sido de tipo relacional con el lenguaje *MySQL* [10]. En cuanto al material proporcionado por *Eloquent*, se han empleado operaciones simples de inserción, borrado, modificación y consultas. Sin embargo, en algún apartado ha sido necesario realizar alguna consulta con algo más de complejidad.

4.2. Diseño de la arquitectura del sistema

Laravel[1] [11], el *framework* [12] escogido, trae consigo varios patrones de diseño implementados y que han ayudado a generar varias funciones vitales de la aplicación. La arquitectura del sistema viene dada en gran parte por el uso de *Laravel*, que está basado en el clásico patrón de diseño MVC[2], tal y como se muestra en la Figura 4.2. Este patrón separa la lógica de la aplicación del modelo de datos haciendo que el mantenimiento y la implementación de código sea mucho más fácil de realizar, a la vez que el proceso de desarrollo sea mucho más rápido. Las tres secciones de este patrón se divide en:

- Modelo. Es la capa que encapsula los esquemas de los objetos de la aplicación. Su función es interactuar con la base de datos, sacando y guardando información de esta dependiendo de qué operación de la aplicación la haya desencadenado.
- Vista. Es la capa responsable de la interacción con el usuario. Se encarga de enviar las peticiones a la capa de la lógica de negocio. Sí que tiene una parte de funcionalidad mínima, que se reduce al rutado y a la navegación entre los componentes del sistema.
- Controlador. Es la capa que contiene toda la lógica funcional de la aplicación. Se sitúa entre la capa del modelo y de la vista y gestiona las peticiones de la vista junto a las respuestas del modelo y la base de datos. Una de las ventajas de emplear el patrón es que ayuda con la invocación de métodos asíncronos, que se ven representados con las llamadas a servicios externos como una base de datos o incluso APIs[3] externas para consultar información de los barcos.

4.2.1. Patrón de plantillas

Laravel incluye *Blade* [13], un motor de plantillas para generar las vistas usando *HTML* [14] y *CSS* [15]. De hecho, al crear el proyecto, *Blade* ofrece de forma genérica una colección de componentes y de disposiciones de vistas básicas como punto de partida para desarrollar el resto de vistas de la aplicación.

[1] Es un framework para crear aplicaciones web basadas en el lenguaje de programación PHP.
[2] Modelo-Vista-Controlador.
[3] Interfaz de programación de aplicaciones *(Application Programming Interface)*.

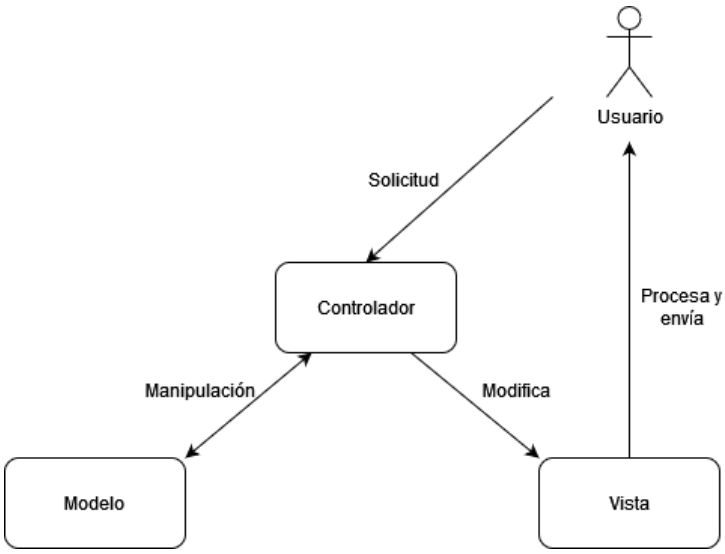


Figura 4.2: Diagrama patrón MVC

Las disposiciones de vistas están formadas por un componente el cual tiene su propio estado y comportamiento. Su función es renderizar la vista asociada a un usuario visitante o autenticado en la aplicación. Otra función útil es cómo gestiona el estado con atributos, ya que para diferenciar las distintas vistas se les asociará sus títulos correspondientes para que sea más sencillo saber en qué sección nos encontramos al navegar por la aplicación.

El uso de *slots* es otra característica importante, ya que permite usar una sección personalizada en las vistas. Por ejemplo, si una disposición la tenemos dividida en la cabecera, el cuerpo y el pie de página, se pueden crear componentes personalizados para cada uno de ellos e inyectarlos en el componente padre.

El siguiente fragmento de *HTML* muestra claramente esta organización:

```
1  <body class="font-sans antialiased">
2      <div class="min-h-screen bg-gray-100">
3          @include('layouts.navigation')
4
5          @if (isset($header))
6              <header class="bg-white shadow">
7                  <div class="max-w-7xl mx-auto py-6 px-4 sm:px-6 lg:px-8">
8                      {{ $header }}
9                  </div>
10             </header>
11         @endif
12
13         <main>
14             {{ $slot }}
15         </main>
16     </div>
17 </body>
```

Listado 4.1: Vista genérica de la disposición de vistas del usuario

En la línea 3 se emplea una directiva de *Blade* para renderizar la barra de navegación. El bloque comprendido entre las líneas 5 y 11 corresponde al uso de una cabecera en la página,

ya sea para mostrar un título u otros datos relevantes. Finalmente, la línea 14 emplea el uso de *slots* para renderizar el resto de la vista que use la disposición.

4.2.2. Patrón de observador

Aunque no haya una implementación estricta, el sistema de notificaciones está estrechamente relacionado con el patrón *Observer* junto con algunos elementos que *Laravel* trae consigo implementado. El patrón *Observer* es un patrón de diseño de comportamiento que permite definir un mecanismo de suscripción para notificar múltiples objetos sobre cualquier evento que le ocurra al objeto que observan [16].

Desde *Laravel*, el sistema de notificaciones viene dado por los eventos, los escuchadores[4] y el propio concepto de notificación. Los eventos representan cualquier acción importante que ocurra durante la ejecución, como en la realización de una reserva o la creación de un nuevo pago. Los escuchadores se encargan de gestionar estos eventos y ejecutar las acciones que gestionan las notificaciones vía correo electrónico, proporcionando así un desacople de responsabilidades.

4.3. Diseño de las interfaces

4.3.1. Prototipos iniciales

El diseño de la interfaz comprendió el proceso de cómo interpretar los requisitos del cliente y exponer de forma visual cómo se estructuraría la información de la aplicación. Para ello se recurrió a *Balsamiq Wireframes* [17], una herramienta sencilla de diseño que permite la creación de interfaces de usuario. Acto seguido se muestran dichas interfaces, las cuales no se alejan demasiado de las versiones iniciales, que tuvieron que modificarse ligeramente.

A continuación se presentan los *mockups*[5] de las páginas principales, junto al formulario de registro. El formulario de acceso sería prácticamente igual al de registro aunque solo cambian los datos a introducir. Estos diseños comprenden desde la Figura 4.3 hasta la Figura 4.5.

Los *mockups* para la gestión de los recursos de las reservas se muestran desde la Figura 4.6 hasta la Figura 4.9.

El módulo de pagos, a nivel visual, solo comprende la opción de visualizar cada uno de ellos o el que esté asociado a una reserva propia. Las Figuras 4.10 y 4.11 ejemplifican dicho módulo visual.

Para llegar a dichas interfaces es necesario remarcar la interfaz visual del calendario. En ella se ha diseñado un calendario navegable y que cada cuadro diario tenga un objeto coloreado seleccionable. Mediante esta acción se procederá con el formulario de reserva en adelante. La

[4]En *Laravel*, para los escuchadores se emplea la clase *Listener*.
[5]Modelos visuales.

Figura 4.3: Prototipo de la página de inicio

Figura 4.12 es prácticamente igual a la Figura 4.3, dejando anotado la reusabilidad de los componentes al momento de realizar la implementación.

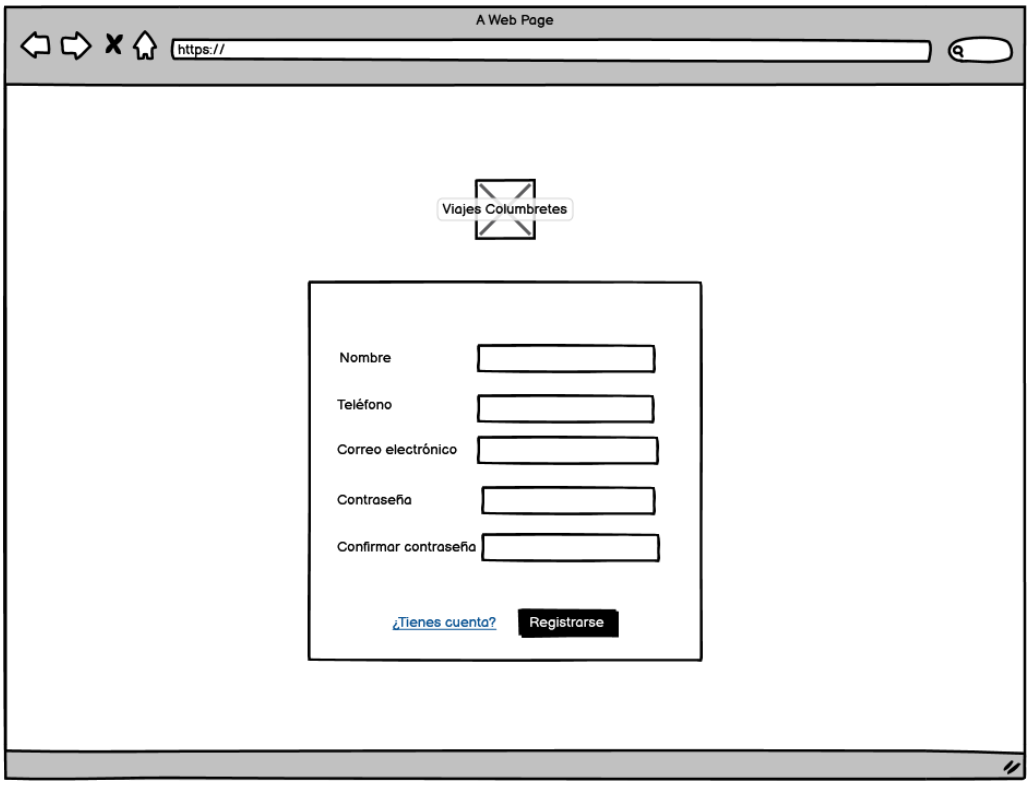


Figura 4.4: Prototipo del formulario de registro

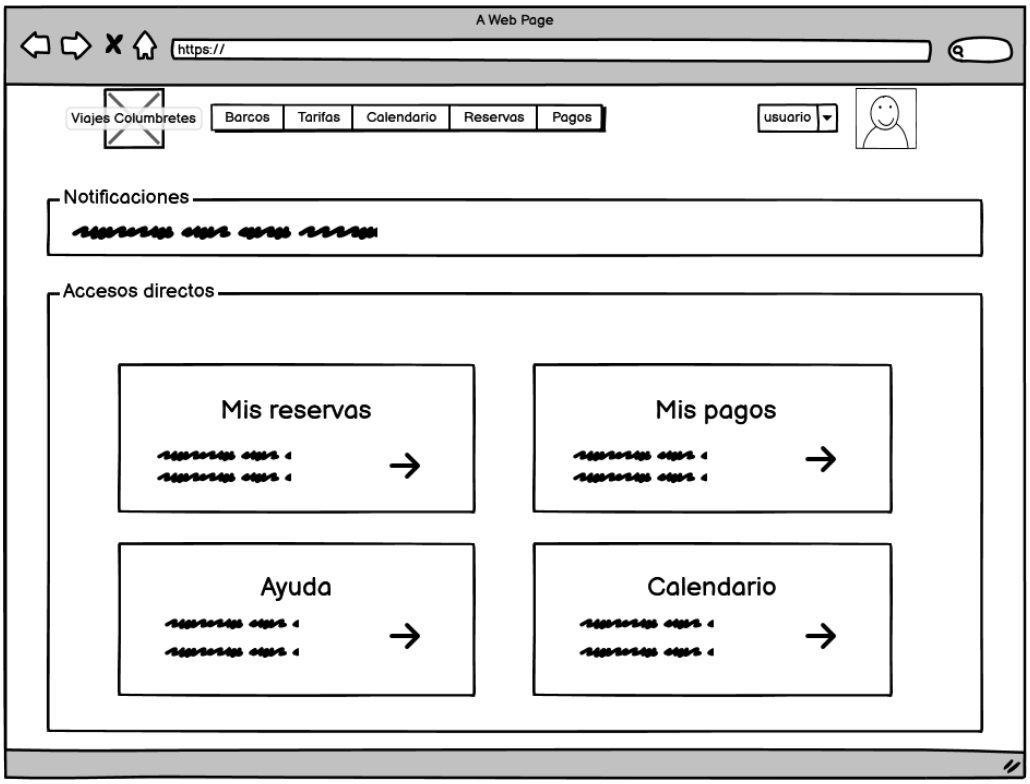


Figura 4.5: Prototipo de la página de inicio para usuario autenticado (*Dashboard*)

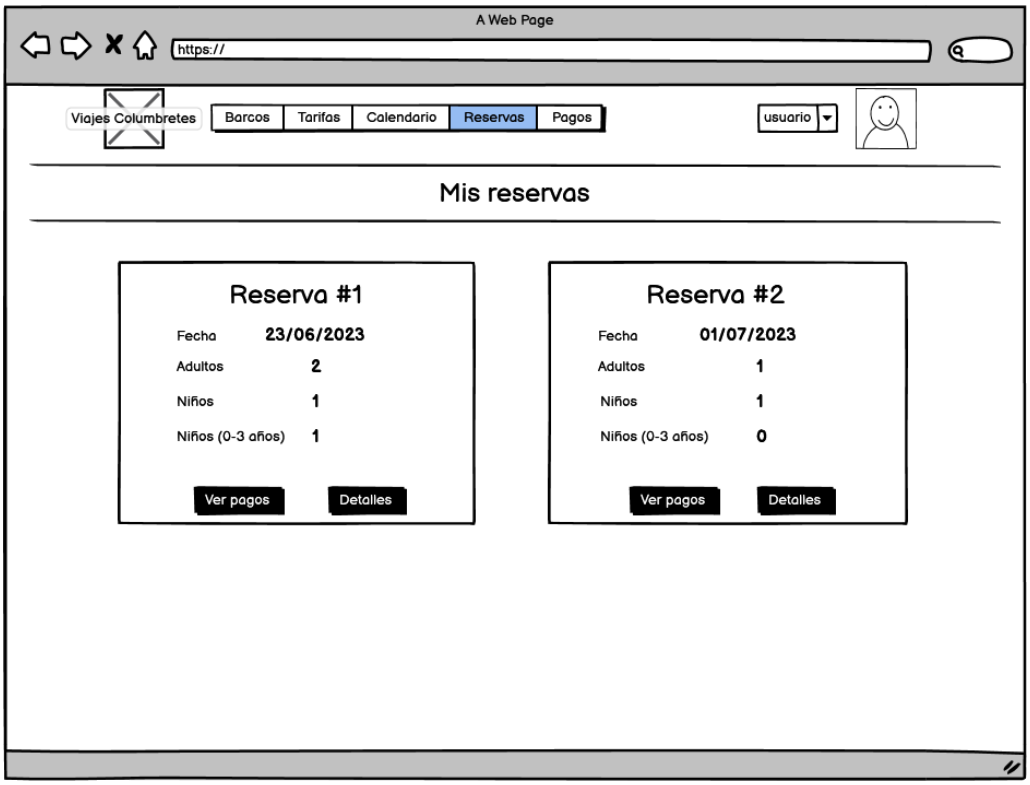


Figura 4.6: Prototipo del listado de reservas

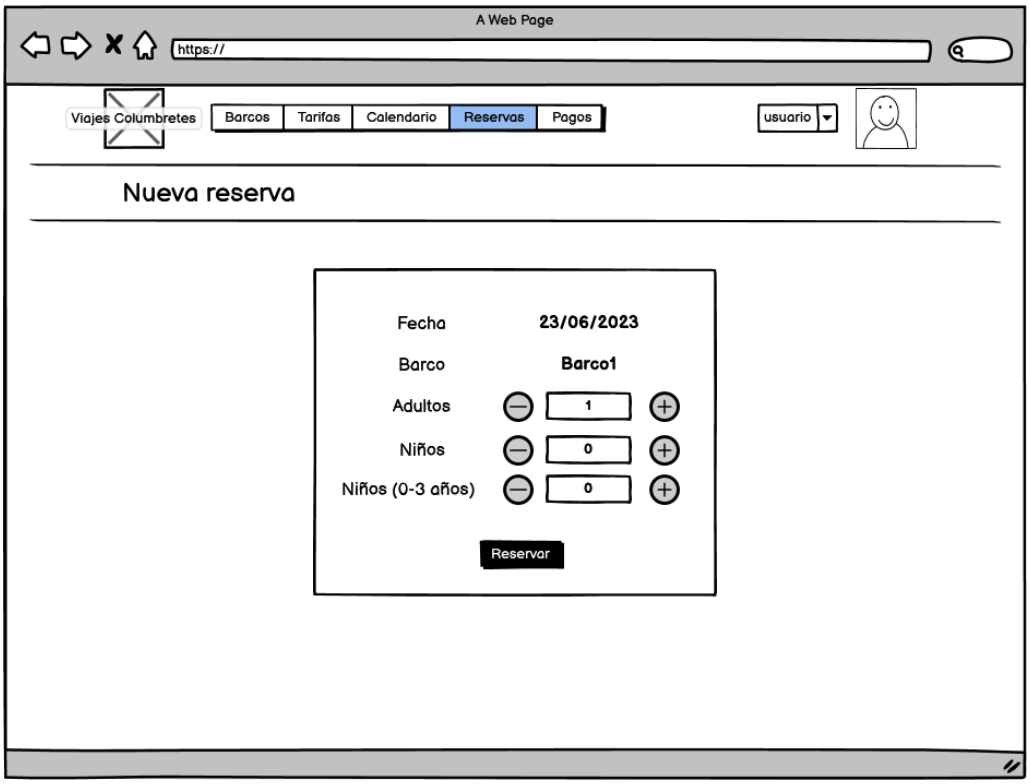


Figura 4.7: Prototipo del formulario de creación de reservas

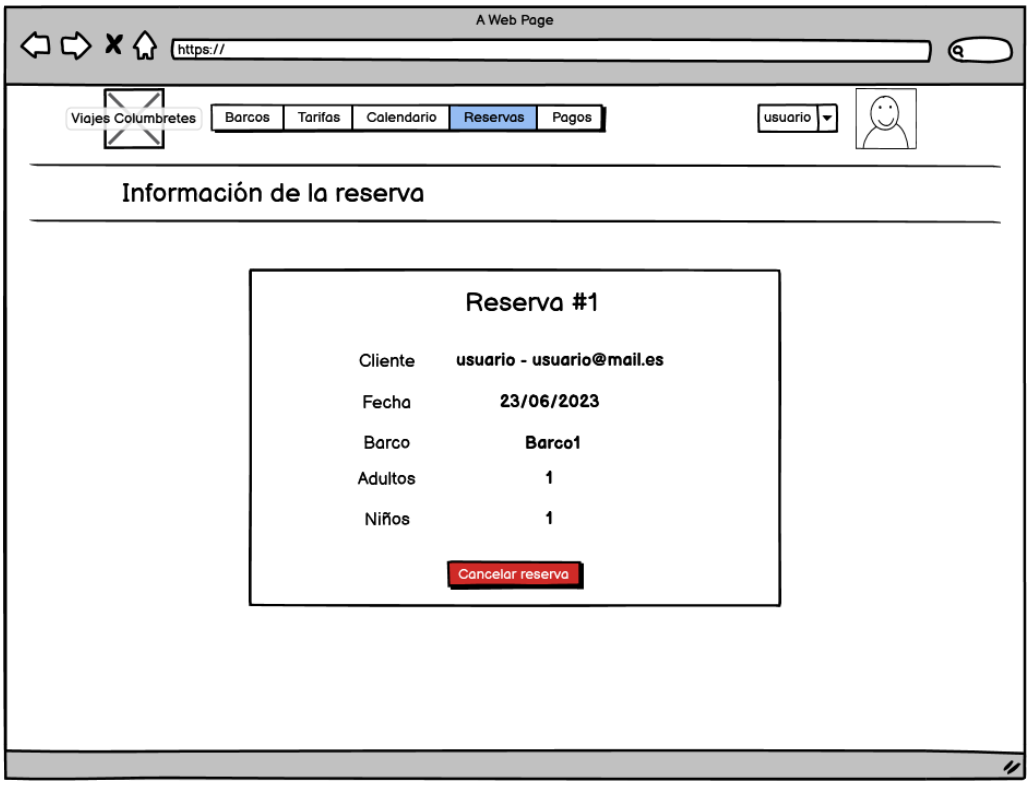


Figura 4.8: Prototipo de información de una reserva concreta

Figura 4.9: Prototipo de confirmación de solicitud de cancelación de reserva

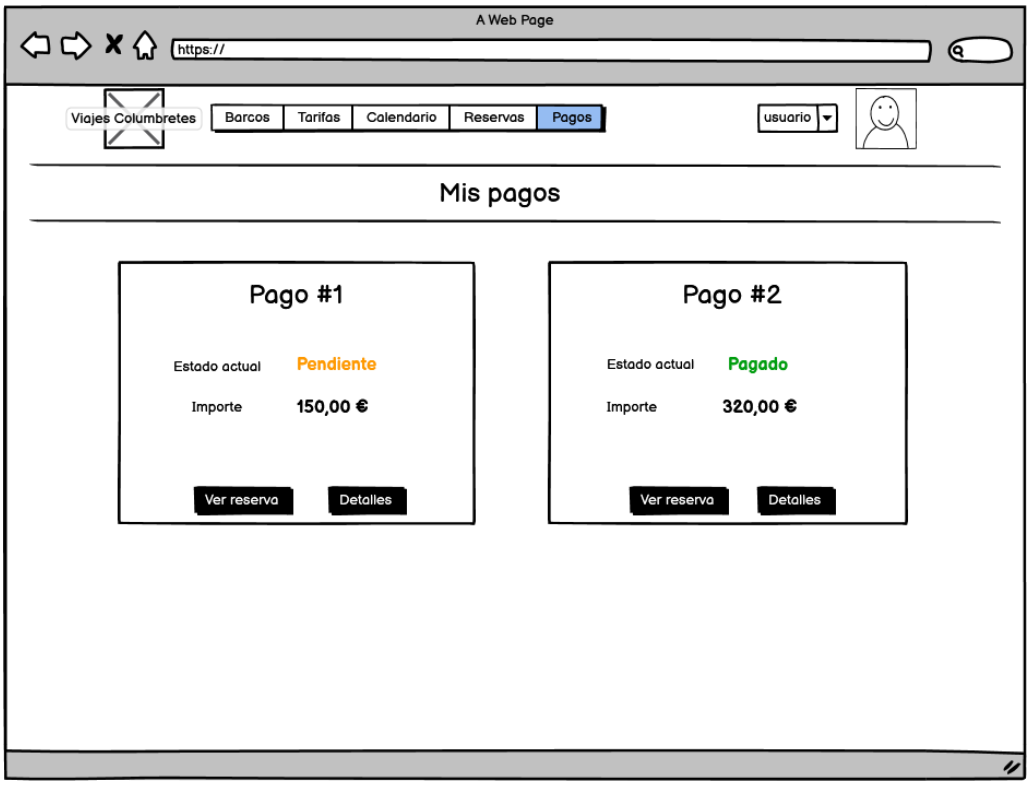


Figura 4.10: Prototipo del listado de pagos

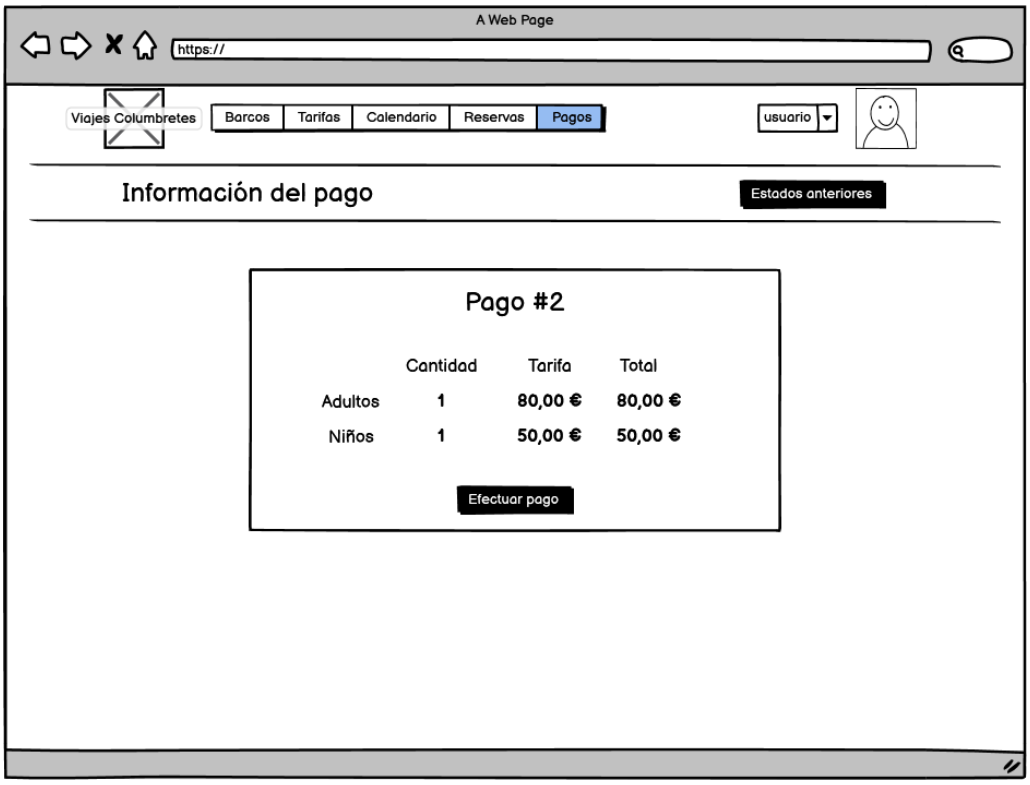


Figura 4.11: Prototipo de información de un pago concreto

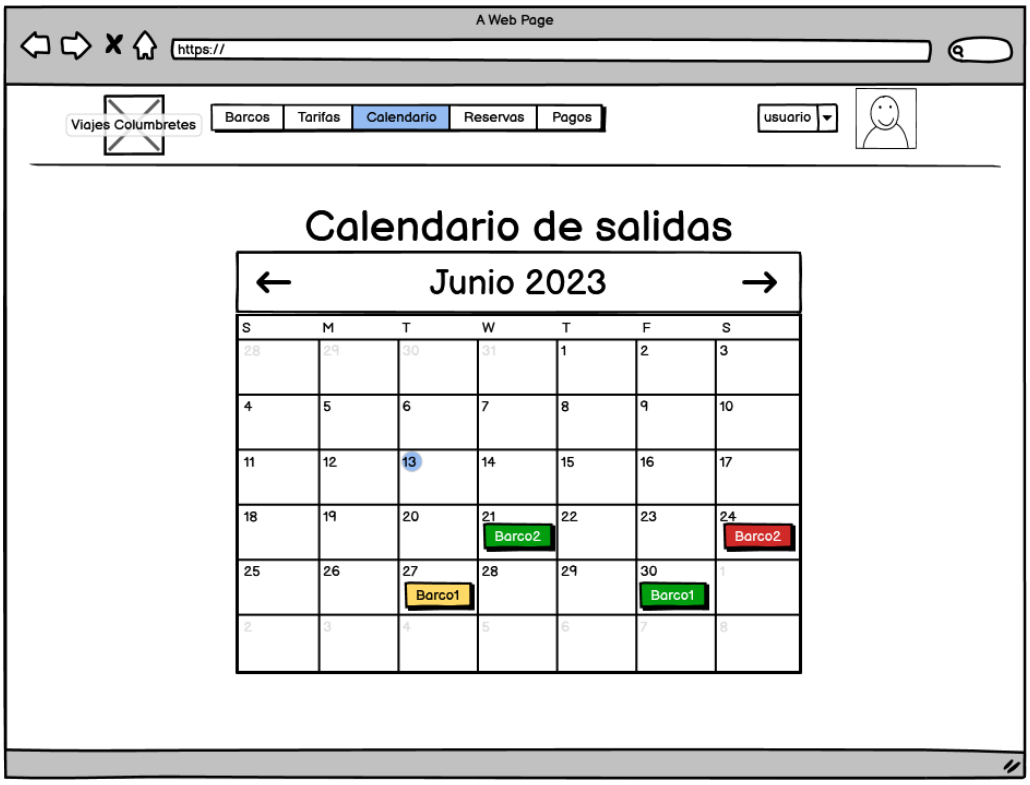


Figura 4.12: Prototipo del calendario de salidas para el usuario autenticado

4.3.2. Colores

La elección de colores se realizó tomando como referencia la página principal de la cual se podrá acceder al gestor. También se emplearon algunos colores a partir de un cartel de horarios que el cliente tenía a mano. Para no cargar demasiado la interfaz visual, se escogieron los colores expuestos en la Figura 4.13. Otros elementos como los botones, igual que la capacidad actual de una fecha de salida, han sido ligeramente modificados para tener una paleta propia de colores (ver Figura 4.14) que contrasta bastante bien con el tono azulado principal.

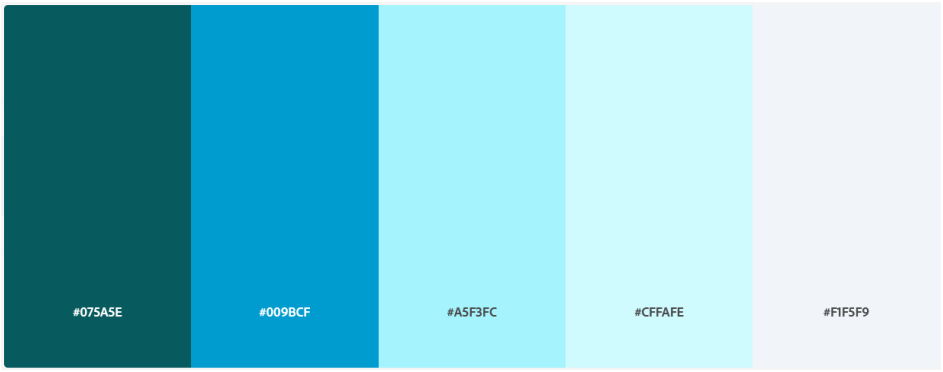


Figura 4.13: Paleta de colores principal

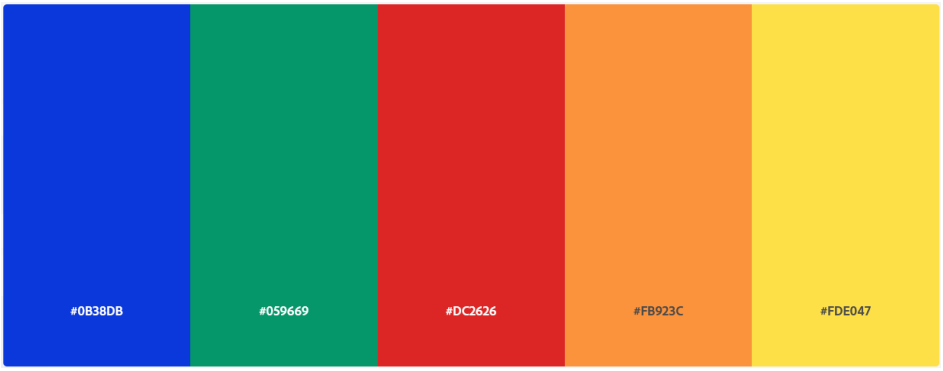


Figura 4.14: Paleta de colores de los botones

Capítulo 5

Implementación y pruebas

5.1. Estructura del código

La implementación del proyecto se ha realizado mediante el uso del lenguaje *PHP* [18], junto al *framework Laravel* y *Laravel Breeze* [19]. El estilado se ha hecho mediante el motor de plantillas *Blade* junto a *Tailwind CSS* [20]. Para el entorno de desarrollo se ha utilizado *Visual Studio Code* [21] con extensiones concretas para ayudar con la implementación. Además, para controlar cómo se van integrando los cambios en el repositorio, se ha utilizado la combinación de Git [22] y Github [23] para el alojamiento del repositorio, junto a *Conventional Commits*, un estándar para describir con claridad los cambios en el histórico del repositorio [24].

Por parte de la empresa no hay una plantilla definida para la estructura del proyecto, ya que me han dado vía libre para explorar las funcionalidades que ofrece *Laravel*. Como punto de referencia, he seguido la guía que ofrece *Laravel* para desarrollar un pequeño proyecto web y luego adaptarlo a las necesidades de este. Como ya se ha mencionado, la librería empleada ha sido *Laravel Breeze*. Esta librería tiene implementada prácticamente en su totalidad los componentes de autenticación, incluyendo el registro y el *login* del usuario. A partir de este punto se ha implementado el resto del código [25].

En las Figuras 5.1 y 5.2 se muestra la estructura final del proyecto. Debido a la organización modular [26] que tiene *Laravel*, se ha seguido, en la mayor medida de lo posible, la estructuración de los ficheros donde les corresponde. Los controladores tienen su directorio específico, igual que la definición de las rutas, los modelos y las vistas, entre otros. A continuación se van a explicar los directorios en los que se ha hecho la implementación:

- Directorio App: Es el directorio donde se encapsula la gran mayoría de la aplicación. Este directorio contiene a su vez otros directorios, que conforman el esqueleto lógico de una aplicación de *Laravel*. Los más relevantes son:
    - Directorio Http: Aquí se almacenan los controladores de los recursos y los middlewares.

- Directorio Models: Los modelos de Eloquent se almacenan en este directorio. Cada recurso tiene un modelo asociado a una tabla de la base de datos.
- Directorio View: Aquí se encuentran los componentes principales encargados de gestionar las disposiciones de vistas.
- Directorio Services: Este directorio no existe por defecto, sino que existe para almacenar clases separadas que contienen lógica adicional que utilizan los controladores [27].
- Directorios Events/Listeners/Notifications: Este trío de directorios encapsula el sistema de notificaciones de la aplicación. Los eventos se utilizan para avisar otras partes de la aplicación cuando se ha realizado una acción concreta. Los escuchadores se encargan de, mediante un evento cualquiera, realizar las acciones pertinentes cuando se desencadenan tales eventos. Las notificaciones son mensajes usados para informar al usuario que algo ha ocurrido durante la ejecución.
- Directorio Policies: Este directorio contiene las políticas de autorización de los recursos, en los que se determina si un usuario puede realizar operaciones o no sobre dichos recursos.
- Directorio Providers: El directorio de proveedores contiene los proveedores de servicios. Aquí se han registrado los comportamientos de los eventos.

- Directorio Database: Este directorio contiene los ficheros de las migraciones de la base de datos, las factorías y los seeders[1].
- Directorio Resources: Es el directorio que contiene todas las vistas (ver Figura 5.2b). El directorio *Layout* contiene las disposiciones de vistas. El directorio *Components* contiene los componentes de *Blade* por defecto. Algunos han sufrido ligeras modificaciones en cuanto al estilo que tienen definido. El resto de directorios son los distintos recursos que ofrece la aplicación, cada uno con su vista asociada las operaciones *CRUD*[2] que le corresponda.
- Directorio Routes: Aquí se recogen las definiciones de los *endpoints* de la aplicación. Las rutas implementadas se han hecho en los ficheros *auth.php* y en *web.php*.
- Directorio node_modules, vendor y public: Los dos primeros directorios almacenan las dependencias de *JavaScript* [28] y *PHP* respectivamente. En *public* se encuentra el punto de entrada a la aplicación, además de otros ficheros de utilidad como iconos e imágenes.

[1] Un seeder es una clase especializada en poblar la base de datos.
[2] Crear-Leer-Actualizar-Eliminar *(Create-Read-Update-Delete)*.

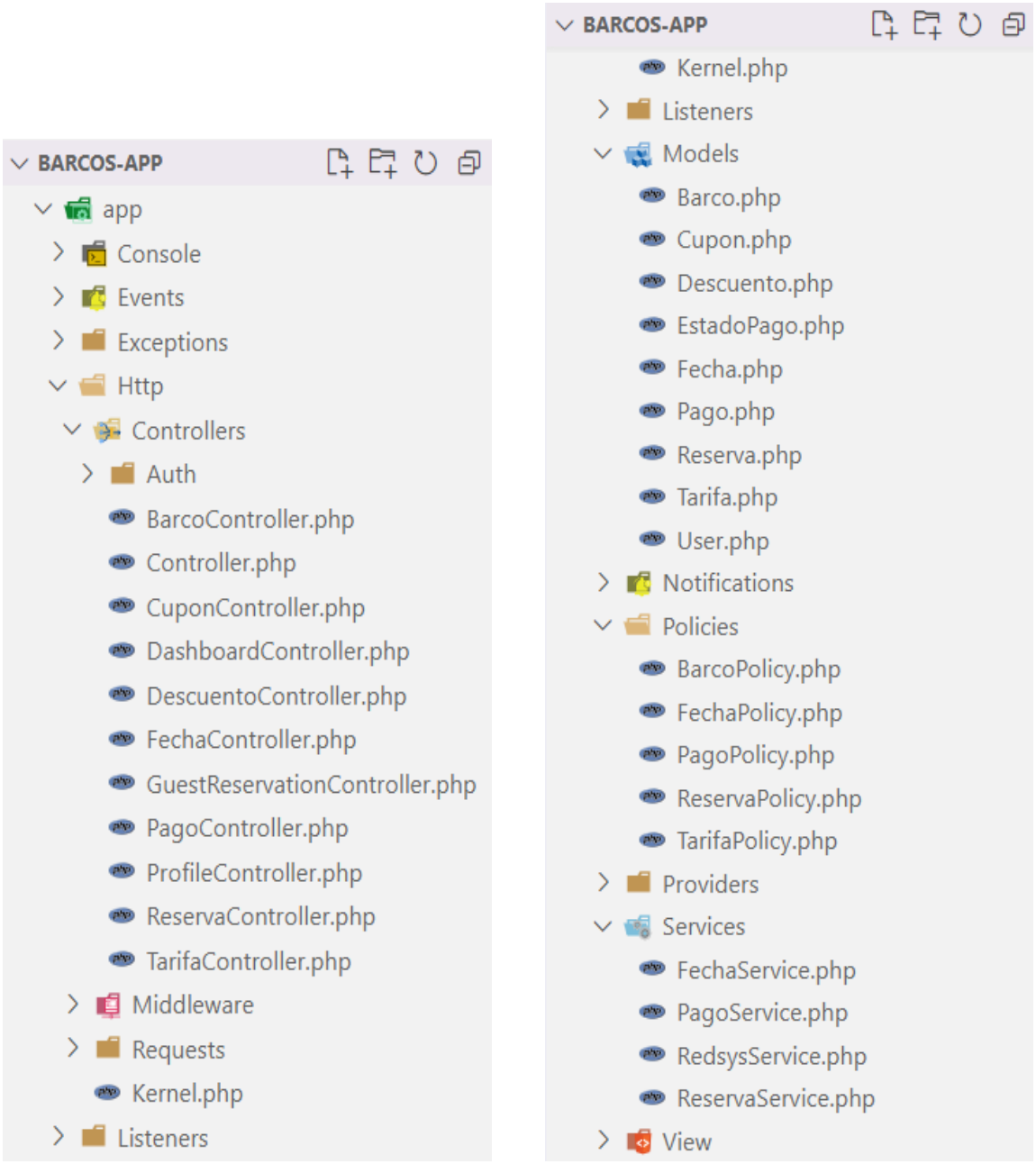
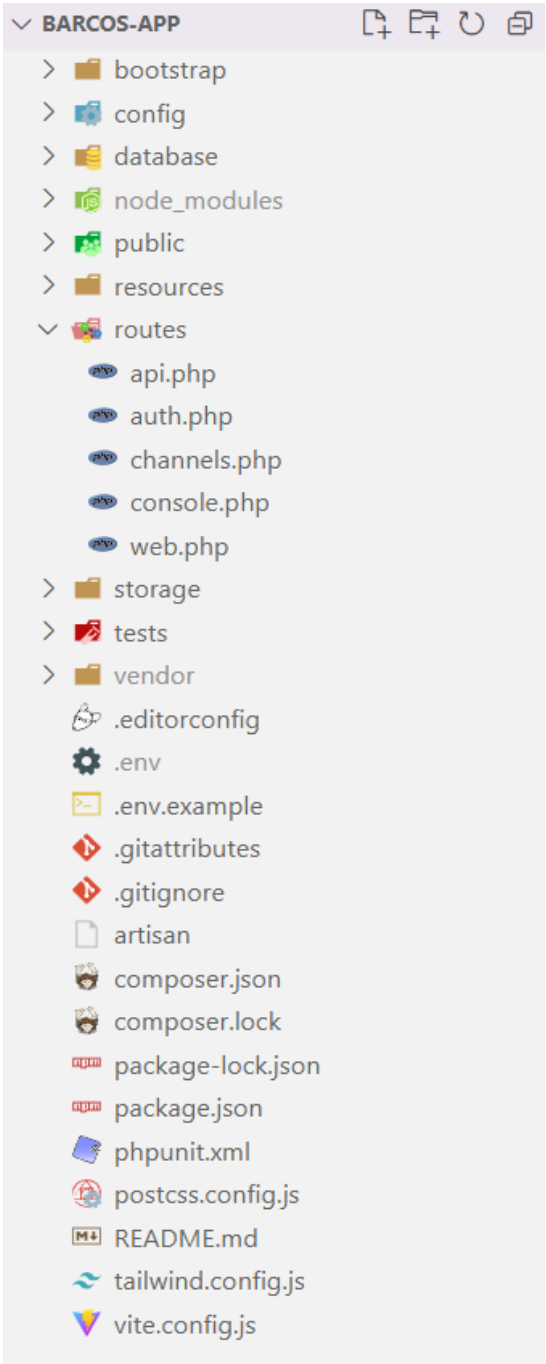
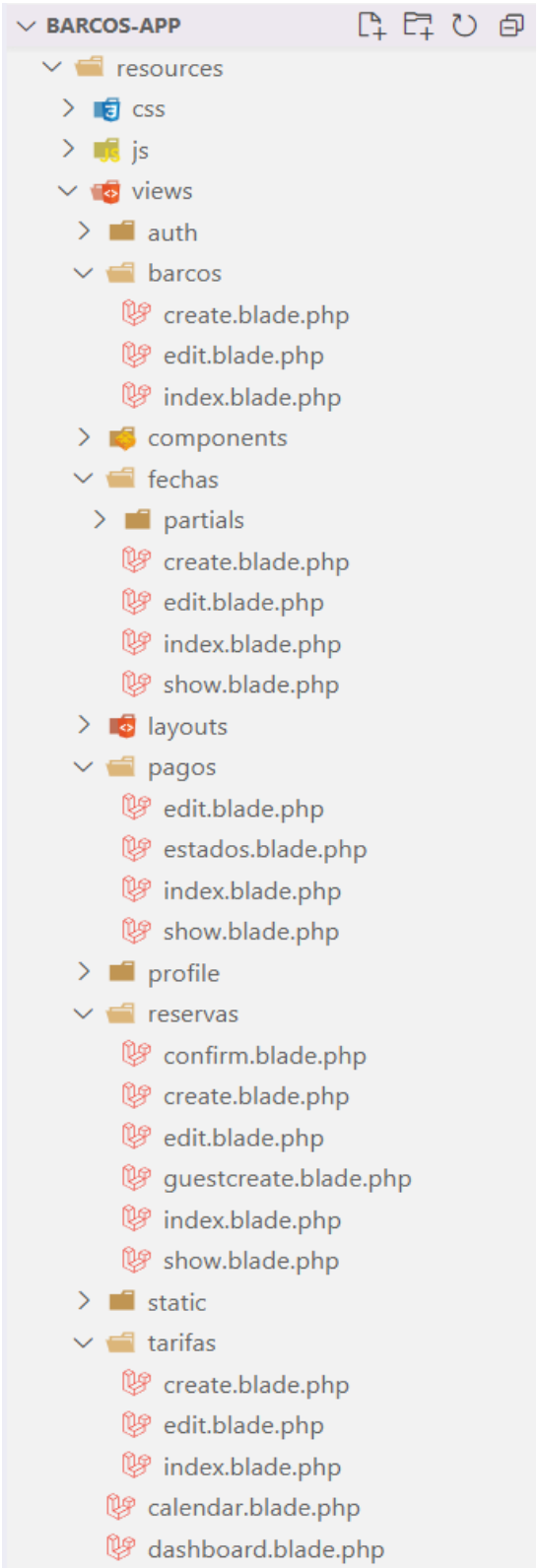


Figura 5.1: Estructura final de los directorios del proyecto (1)



(a) Resto de directorios



(b) Organización de las vistas

Figura 5.2: Estructura final de los directorios del proyecto (2)

5.2. Descripción técnica de la implementación

Las siguientes secciones muestran cómo ha evolucionado el proceso de desarrollo junto a las decisiones tomadas en los distintos módulos implementados.

5.2.1. Gestión de usuarios

Gracias a *Laravel Breeze*, la implementación de la gestión de usuarios venía hecha casi en su totalidad, pero todavía quedaba la gestión de los roles. Como ya se definió durante el análisis, el usuario y el administrador comparten ciertas funcionalidades pero éste último va a tener algunas exclusivas. En el diagrama de clases que se planteó al inicio, se propuso una jerarquía de clases para diferenciar al usuario y al administrador por las funcionalidades que pueden o no hacer. Esto dio lugar a que se crearan clases específicas para ambos, con la única diferencia en que ambas clases tenían un método para comprobar si son administradores o no.

Se vio rápidamente que esta gestión específica era más tediosa de lo que parece. Cada clase tendría su controlador, la mayoría de los casos con código repetido. Entonces, se optó por modificar el diagrama de clases para que solo la clase base del usuario tuviera un atributo que lo identificaría si es administrador o no.

Así, se hizo la implementación de un *middleware* que maneja en todo momento el estado del rol de usuario. La implementación se basa en crear una clase llamada *AdminMiddleware* (ver Listado 5.1) y que en su método *'handle'* se compruebe si el rol del usuario autenticado es el adecuado. Si no lo es, redirige a la pantalla principal o a la de inicio de sesión.

```
class AdminMiddleware {

    public function handle(Request $request, Closure $next): Response {
        if (Auth::check()) {
            if (Auth::user()->role == '1') {
                return $next($request);
            } else {
                return redirect('/dashboard')
                    ->with('message', 'Acceso denegado.');
            }
        }
        return redirect('/login')
            ->with('message', 'Inicia sesion para acceder.');
    }
}
```

Listado 5.1: Fragmento de código de AdminMiddleare

5.2.2. Gestión de barcos y tarifas

Los primeros recursos que se implementaron fueron los que no tienen dependencias con otros. Estos son los barcos y las tarifas. Para empezar, como ya se mencionó anteriormente, *Laravel* ofrece un tutorial para implementar recursos de una manera muy intuitiva [29]. Con *Compo-*

ser[3] [30] se crearon los modelos de barcos y tarifas, además de sus respectivos controladores y migraciones. Al asignar las rutas, se tuvo en cuenta el *middleware* del administrador con el fin de proteger, principalmente, las operaciones de creación, modificación y borrado.

```
1  // Operaciones protegidas para el administrador
2  Route::middleware(['auth', 'isAdmin', 'verified'])->group(function() {
3
4      Route::resource('barcos', BarcoController::class)
5      ->only(['create', 'store', 'edit', 'update', 'destroy']);
6
7      Route::resource('tarifas', TarifaController::class)
8      ->only(['create', 'store', 'edit', 'update', 'destroy']);
9
10 }
11
12 // Operaciones del usuario
13 Route::middleware(['auth', 'verified'])->group(function() {
14
15     Route::resource('barcos', BarcoController::class)
16     ->only(['index']);
17
18     Route::resource('tarifas', TarifaController::class)
19     ->only(['index']);
20
21 }
```

Listado 5.2: Gestión de la protección de las rutas con AdminMiddleware

En este caso se puede ver como se han declarado dos recursos para la gestión de barcos. Por parte del administrador, como se puede verificar en la línea 2, se aplica el *middleware 'isAdmin'* a las operaciones pertinentes. Fuera de *AdminMiddleware* se aprecia que el listado de los barcos y las tarifas no tienen este *middleware*, siendo una operación común para ambos usuarios.

5.2.3. Gestión de fechas de salida

El siguiente módulo a desarrollar fue el de las fechas de salida, que conectaría en primera instancia con los barcos y las tarifas, haciendo que para una fecha de salida solo haya un barco asignado y una tarifa aplicada a todas las reservas. El cliente quería que las fechas de salida aparecieran en una disposición de calendario, para que el usuario pudiera elegir qué día realizar una excursión y proceder al posterior proceso de reserva. Como *Laravel* no trae una implementación nativa de un calendario, elegí la librería *FullCalendar* [31] que ofrece una plantilla visual muy cómoda y la visualización de eventos, la cual es perfecta para declarar las fechas de salida.

Como el usuario y el cliente van a ver información distinta, implementé dos calendarios. El calendario del usuario solo tendría la opción de seleccionar los eventos para guiarle al proceso de reserva, mientras que el administrador tendrá las opciones de ver información más detallada de las fechas de salida tales como reservas, cancelaciones u ocupación real, además de la posibilidad de seleccionar día para asignar fechas de salida. El método *'index'* del recurso controla cuál es el calendario que hay que mostrar en base al rol del usuario:

[3]Gestor de dependencias de PHP.

```
<div class="container">
    @if (Auth::user()->role == '1')
        @include('fechas.partials.calendar-admin')
    @else
        @include('fechas.partials.calendar-user')
    @endif
</div>
```

Listado 5.3: Control de visualizado del calendario

Eloquent integra por defecto que sus modelos tengan un identificador numérico único. Para las fechas de salida se decidió cambiar este identificador tras la definición de requisitos con el cliente, que estipuló que un día tuviera una salida. En el modelo de la fecha se declaró cambiar la clave primaria tal y como se ilustran en los siguientes *listings*:

```
class Fecha extends Model {
    use HasFactory;

    protected $primaryKey = 'fecha_salida';
    public $incrementing = false;
    protected $keyType = 'date';

    protected $fillable = [
        'fecha_salida',
        'salida_desde',
        'variable_pago',
        'capacidad',
        'tarifa_id',
        'barco_id'
    ];

    // ...

}
```

Listado 5.4: Modelo de las fechas de salida

```
public function up(): void {
    Schema::create('fechas', function (Blueprint $table) {
        $table->date('fecha_salida')->primary();
        $table->string('salida_desde');
        $table->integer('variable_pago');
        $table->integer('capacidad');
        $table->foreignId('tarifa_id')->constrained()->cascadeOnDelete();
        $table->foreignId('barco_id')->constrained()->cascadeOnDelete();
        $table->timestamps();
    });
}
```

Listado 5.5: Método de la migración que gestiona la creación de las fechas de salida

5.2.4. Gestión de reservas

El módulo de las reservas fue el siguiente en ser implementado, conectando las fechas de salida con el usuario mediante la asignación de plazas. La implementación de los recursos de las reservas ha sido lineal. Principalmente, la metodología de implementación ha sido la misma

que el tutorial excepto en dos casos: qué puede ver el administrador y el cliente, y cómo se valida la creación de una reserva. El primer caso es bastante sencillo, en el método *'index'* se introduce una condición sobre qué tipo de usuario está accediendo al listado de reservas. Esto se hace porque el administrador querrá ver todas las reservas de todos los clientes, mientras que el cliente solo podrá ver las suyas.

La validación de la creación de la reserva se ha hecho en dos partes. La primera es validando los campos del formulario de creación con las reglas de *Eloquent*. La segunda y, de las que más complicaciones ha dado, es comprobar el estado de las plazas que hay reservadas y actualizar o no, dependiendo de la capacidad.

Conectando con el módulo anterior, se consiguió implementar en la vista del calendario los eventos asociados a las fechas de salida con la información de las reservas actualizadas al momento. El resultado ha sido la generación de las rutas que ejecutan la lógica de consulta a la base de datos.

```
Route::get('/fetch-events', function() {
    $salidas = FechaService::fetchEvents();
    header('Content-Type: application/json');
    echo json_encode($salidas);
});
```

Listado 5.6: Ruta asociada a la consulta de fechas de salida

```
events: function(start, end, timezone) {
    return $.ajax({
        url: 'fetch-events',
        dataType: 'json',
    });
}
```

Listado 5.7: Función encargada de extraer eventos de la base de datos

Desde este punto, un usuario podía crear una reserva y tener unas plazas asignadas pero, ¿y si no estoy registrado? ¿Cómo puedo realizar el proceso si soy un visitante y no tengo una cuenta?

La solución fue generar otro controlador llamado *GuestReservationController* y asignar ahí todo el proceso de creación de reserva. Dicha implementación tuvo bastantes cambios, sobre todo a nivel visual, porque en el mismo formulario de reserva se quería tener una sección donde el visitante tuviera un resumen de la reserva que va a realizar, junto a la tarifa aplicada y el precio a pagar. Por mi parte, decidí mantenerlo separado en vistas distintas ya que el intercambio de variables entre pantallas me resultaba más sencillo de implementar. Hubo dos métodos de los controladores de la autenticación del usuario que hubo que modificar para controlar la reserva del visitante. Los ficheros *AuthenticatedSessionController* y *RegisteredUserController* que *Laravel* traía por defecto se les añadieron en sus funciones *'store'* la validación de la existencia de las variables de sesión creadas por el formulario, para que cuando el usuario efectúe la operación de registro o de acceso a la aplicación, internamente se cree la reserva correctamente. Esta decisión de implementación conllevó a la generación de dos vistas prácticamente idénticas a las de registro y acceso, con la diferencia de que las que venían por defecto no tienen en cuenta las variables de sesión del formulario de reserva, evitando así la creación de reservas no deseadas.

5.2.5. Gestión de pagos

Una de las funcionalidades más críticas es el módulo de los pagos, el cual está asociado a una pasarela de pagos externa. Cabe destacar que durante la planificación inicial se definieron los tipos de pago porcentuales y fijos. El de tipo fijo fue el primero que se implementó, ya que solo es necesario generar un único pago cuando se cree una reserva. Una vez más, se siguió el mismo esquema de implementación: creación de los modelos, controladores y vistas de la misma forma que los módulos anteriores.

Por parte de la empresa se aportó la documentación relacionada con la instalación de los módulos de *Redsys* [32] para gestionar la pasarela de pago con la aplicación. El tipo de integración elegido para crear la pasarela es el de conexión por redirección, con el que se abre una nueva pestaña con los datos del pago en *Redsys* para que el usuario introduzca sus datos de pago y se procesen adecuadamente. Cabe destacar que la gestión de anulaciones y devoluciones propia de *Redsys* no entran en el alcance de esta funcionalidad, solo la aceptación y denegación de pagos. Adicionalmente, también se puede utilizar el módulo de administración que proporciona el servicio, pero solo con las dos funciones básicas son más que suficientes para la aplicación.

El fichero *RedsysService.php* es el encargado de procesar las peticiones al TPV externo. Se han definido dos métodos principales: el primero crea la petición con la información necesaria del comercio más la cantidad del pago a cobrar y el segundo recibe la respuesta de *Redsys* después de que el usuario introduzca sus datos de pago. Con un bloque *if-else* se comprueba el código de respuesta y se realizan las acciones pertinentes en caso de que el pago sea válido o *Redsys* lo haya rechazado. En el fichero *PagoController.php* la llamada al servicio se declara en el método *'show'*, para efectuar la encriptación de los datos de un pago.

Hubo una reunión con el supervisor para aclarar ciertas dudas que surgieron con la implementación de *Redsys* y los pagos. Se comentó que los pagos deberían tener información almacenada de distintos estados, cosa que es relevante para pagos parciales e incluso devoluciones. Esto hizo extender al modelo de pagos con un modelo más que define el estado de un pago. La acción resultó en añadir una clase más a los pagos y una relación.

Con esta ligera extensión, el fichero *PagoService.php* tendría la responsabilidad de crear los estados de los pagos, omitiendo su respectivo controlador al ser una acción específica de la clase principal. Las otras localizaciones donde este método es utilizado es en los método *'store'* y *'update'* del controlador de pagos y el método encargado de las cancelaciones de las reservas. El siguiente fragmento de código muestra, sin entrar en detalle, cómo se procesa la respuesta de *Redsys* con los estados de los pagos:

```
public function recibeRespuestaRedsys() {
    $miObj = new RedsysAPI;

    $datos = $_GET["Ds_MerchantParameters"];
    $signatureRecibida = $_GET["Ds_Signature"];
    $kc = env('keyTest');
    $firma = $miObj->createMerchantSignatureNotif($kc,$datos);

    if ($firma === $signatureRecibida) {

        // Codigo de respuesta
        $codigo = $miObj->getParameter("Ds_Response");
```

```
13
14          // Procesar la respuesta
15          if ($codigo == '0000') {
16              // Autorizada
17
18              // Actualizar el pago en la base de datos
19              // Si es tipo fijo:
20              //      Actualizar el pago a completado
21              //      Insertar estado pago completado
22              // Si es tipo parcial:
23              //      Mirar ultimo estado
24              //      Si no estaba pagado:
25              //          Insertar estado pago parcial
26              //      Si estaba pagado parcialmente:
27              //          Insertar estado pago completado
28              //          Actualizar el pago a completado
29
30              return redirect(route('pagos.index'));
31          }
32          elseif ($codigo == '0190') {
33              // Denegado. No se ha podido realizar el pago
34              return redirect(route('pagos.index'));
35          }
36          elseif ($codigo == '9915') {
37              // Denegado. El usuario ha cancelado el pago
38              return redirect(route('pagos.index'));
39          }
40      }
41      else {
42          // Firma invalida
43          return;
44      }
45  }
```

Listado 5.8: Función encargada de procesar las respuestas de Redsys

Tanto si el pago que se ha realizado es de tipo fijo o parcial, hay casuísticas que fuerzan a que se dé el caso de realizar devoluciones de pagos de una reserva. El ejemplo más claro es si un día en el que hay una fecha programada hace muy mal tiempo, haciendo que sea peligroso realizar una excursión.

```
1  class EstadoPago extends Model {
2      use HasFactory;
3
4      protected $fillable = [
5          'estado',
6          'observaciones'
7      ];
8
9      public function pago(): BelongsTo {
10         return $this->belongsTo(Pago::class);
11     }
12 }
```

Listado 5.9: Modelo del estado de los pagos

5.2.6. Gestión de notificaciones

El último módulo por implementar fue completar el resto con el envío de notificaciones al usuario. Primeramente se identificaron en qué casos hay que notificar al usuario, con el fin de conectar dichas acciones de los controladores con los eventos correspondientes. Se consultó con el supervisor todas las posibilidades pero para el alcance que hubo definido, con hacer la reserva y la gestión de los pagos es más que suficiente.

Mediante la guía de *Laravel* para crear eventos y notificaciones, se implementó la lógica necesaria. Se creó la notificación con el cuerpo del mensaje personalizado en la función *'toMail'*. Luego se creó el evento y, como se enviarán notificaciones cada vez que se haga una reserva o se cree el estado de un pago, se inyectará en el método constructor del evento. El modelo tendrá que tener definido qué eventos se activan. Para este caso, asignar el evento al modelo en la propiedad de *'dispatchesEvents'*[4]. Más adelante hubo que crear una clase escuchadora, que se encarga de esperar a que haya algún cambio durante la ejecución para lanzar el evento correspondiente. El proceso se hizo de la misma forma con la creación de un nuevo estado del pago, ya sea cuando se genera el primero, se efectúa un pago o una devolución.

```
class EnviarNuevaReservaCreadaNotification implements ShouldQueue {

    // Constructor por defecto...

    // Funcion encargada de notificar al usuario que hace la reserva
    public function handle(ReservaCreada $event): void {
        $user = User::find($event->reserva->user_id);
        $user->notify(new NuevaReserva($event->reserva));
    }
}
```

Listado 5.10: Listener encargado de notificar la reserva creada

5.3. Verificación y validación

Durante la implementación se iban haciendo pequeñas pruebas funcionales para que cada recurso se gestionara correctamente en la base de datos, aunque más adelante, a medida que la complejidad de la aplicación iba subiendo muy rápido. Casi al final de la sección de implementación, mientras se efectuaban mejoras visuales, se procedió a realizar pruebas con el fin de comprobar que las operaciones se efectuaban correctamente y que no provocaran inconsistencias en la base de datos.

Por parte de *Eloquent*, cada vez que se ejecuta un método del controlador se efectúa una operación de validación de campos. Así, se verifica correctamente que los datos que recibe el método desde el formulario son congruentes. Estas validaciones hacían que la mayoría de requisitos se cumplieran correctamente, dejando así en buen lugar las pruebas unitarias.

Finalmente, se procedió a efectuar pruebas de integración manualmente, comprobando los casos más extremos donde podía fallar la aplicación. Solo hubo unos pocos casos límite donde

[4]Propiedad del modelo encargada en relacionar qué eventos dispara el modelo cuando se realice una acción del controlador.

la aplicación generaba inconsistencias, pero fueron fácilmente solucionadas con una ligera modificación en las reglas de validación y las funciones auxiliares que efectúan las consultas a la base de datos.

5.4. Despliegue de la aplicación

Antes de realizar los cambios finales, se procedió a realizar el despliegue de la aplicación en un servidor externo. Angal tiene adquirido el sistema de alojamiento de *Plesk Obsidian* [33] [34], siendo una herramienta con muy buena configuración y compatibilidad para aplicaciones desarrolladas en *PHP*. Sin embargo, el uso de *Laravel* requería ciertas configuraciones adicionales para realizar el despliegue.

Surgieron varios problemas en los que *Plesk* no informaba correctamente el estado del despliegue y, por tanto, no se podía acceder a la web. Junto con el supervisor y dos compañeros más, quienes tenían más experiencia con el uso de *Plesk*, hubo que realizar ciertos ajustes en la configuración interna de la cuenta asociada al despliegue y también en la conexión del repositorio con el servidor.

Finalmente, la aplicación quedó compilada correctamente y se definieron dos subdominios: el primero es donde está alojada la aplicación en modo de producción, y el segundo es donde se aloja una copia del primero, pero empleando el modo de desarrollo, haciendo que los cambios se vean reflejados correctamente, para luego pasarlos a producción.

5.5. Resultados finales

La siguiente colección de figuras muestra el resultado de la implementación del proyecto, incluyendo la evolución de las plantillas iniciales mostradas en la sección 4.3.1, desde la Figura 5.3 hasta la 5.12. Desde la Figura 5.13 hasta la Figura 5.17 comprende la funcionalidad de las devoluciones de los pagos.

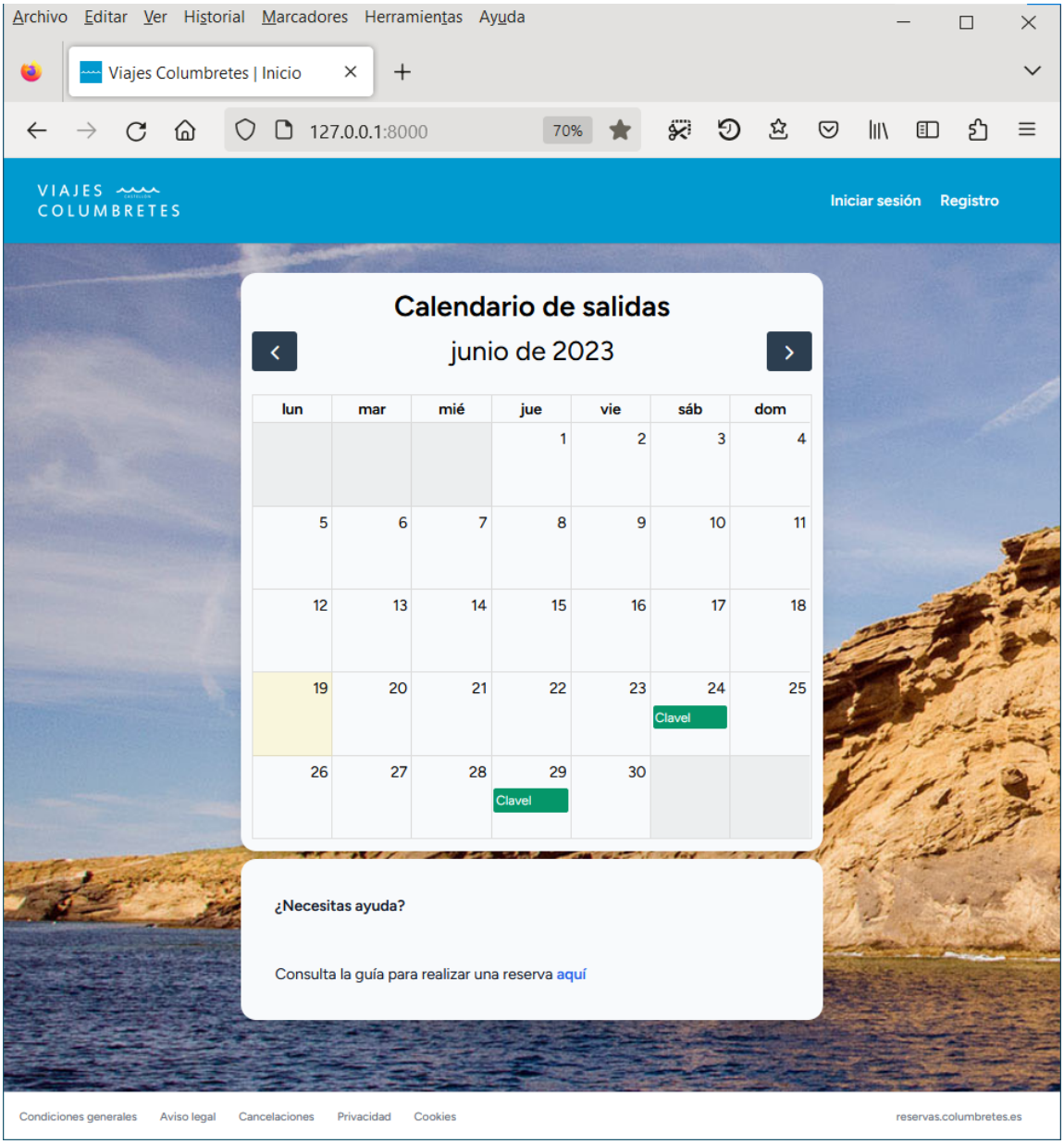


Figura 5.3: Interfaz final de la página de inicio

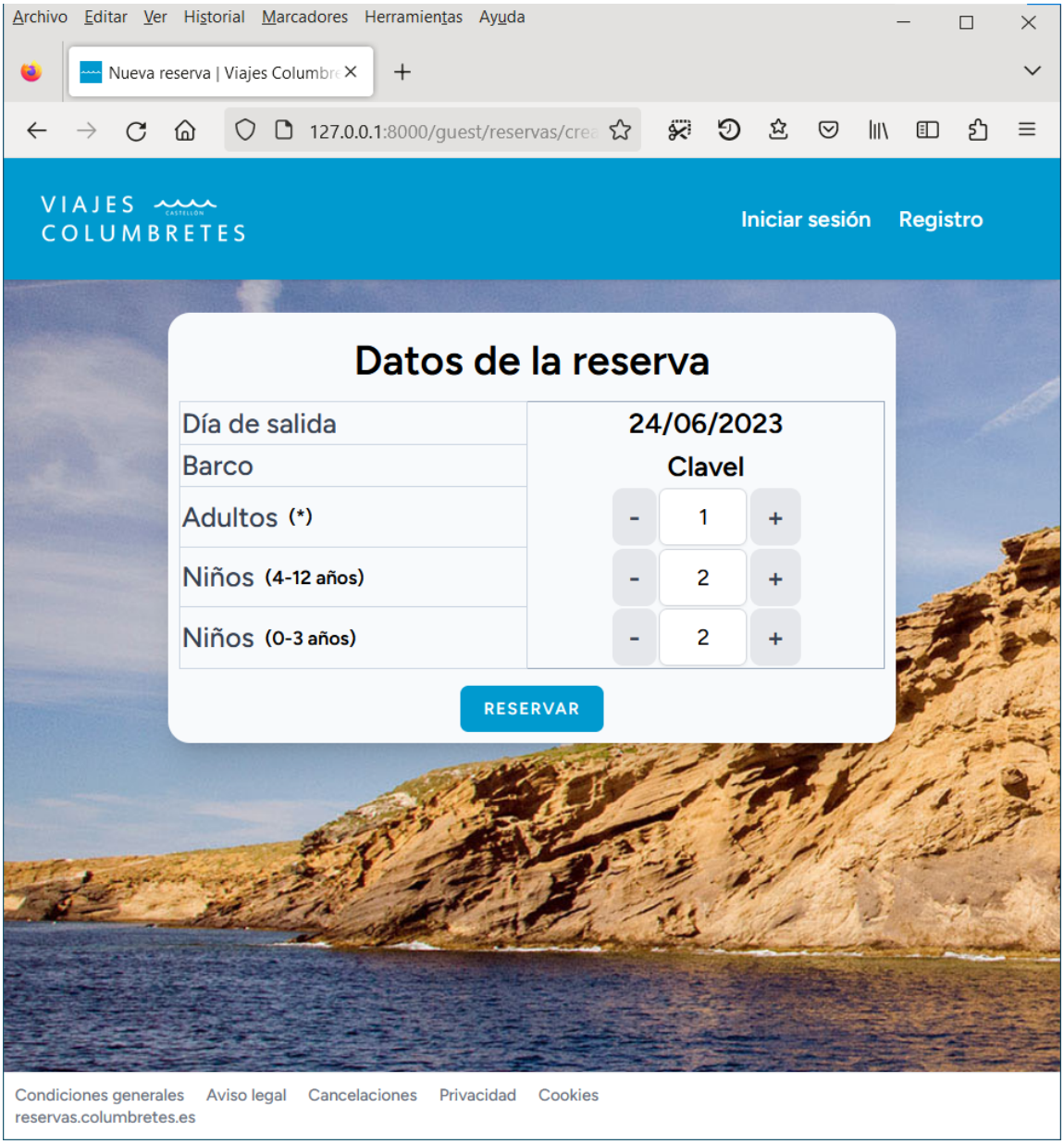


Figura 5.4: Interfaz final del formulación de creación de reservas (visitante)

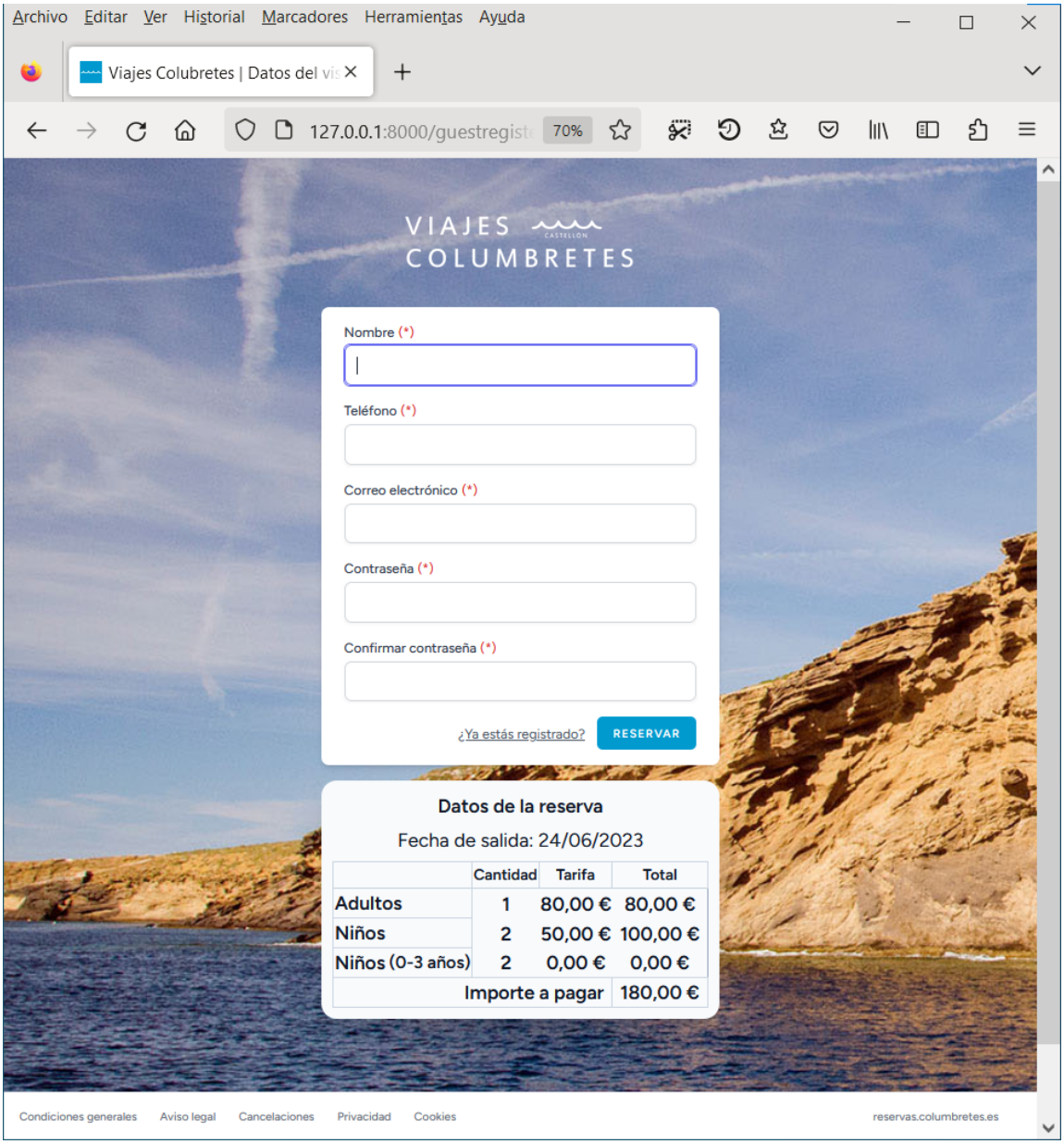


Figura 5.5: Interfaz final del formulario de registro tras generar una reserva

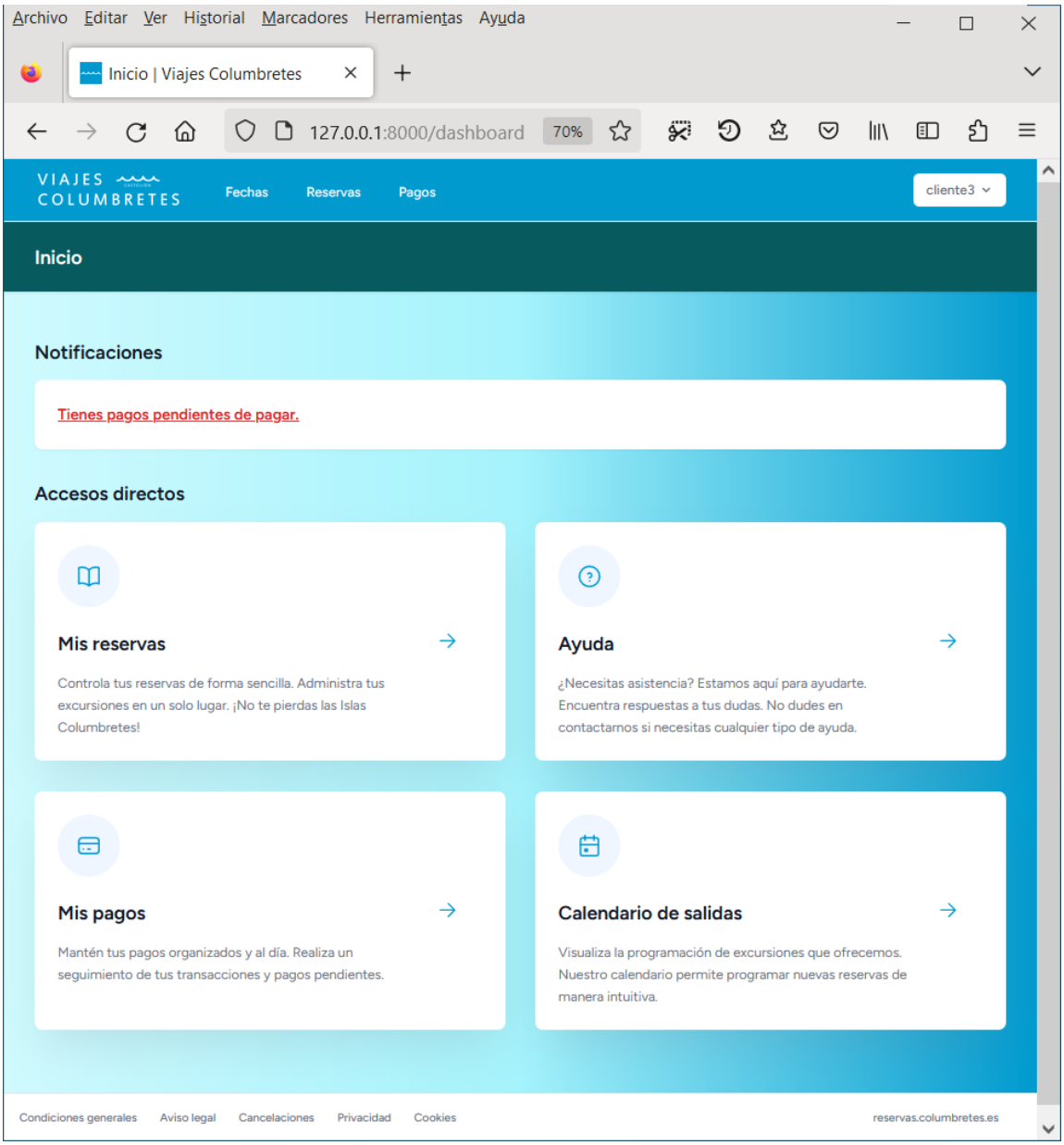


Figura 5.6: Interfaz final de la página de inicio para usuario autenticado

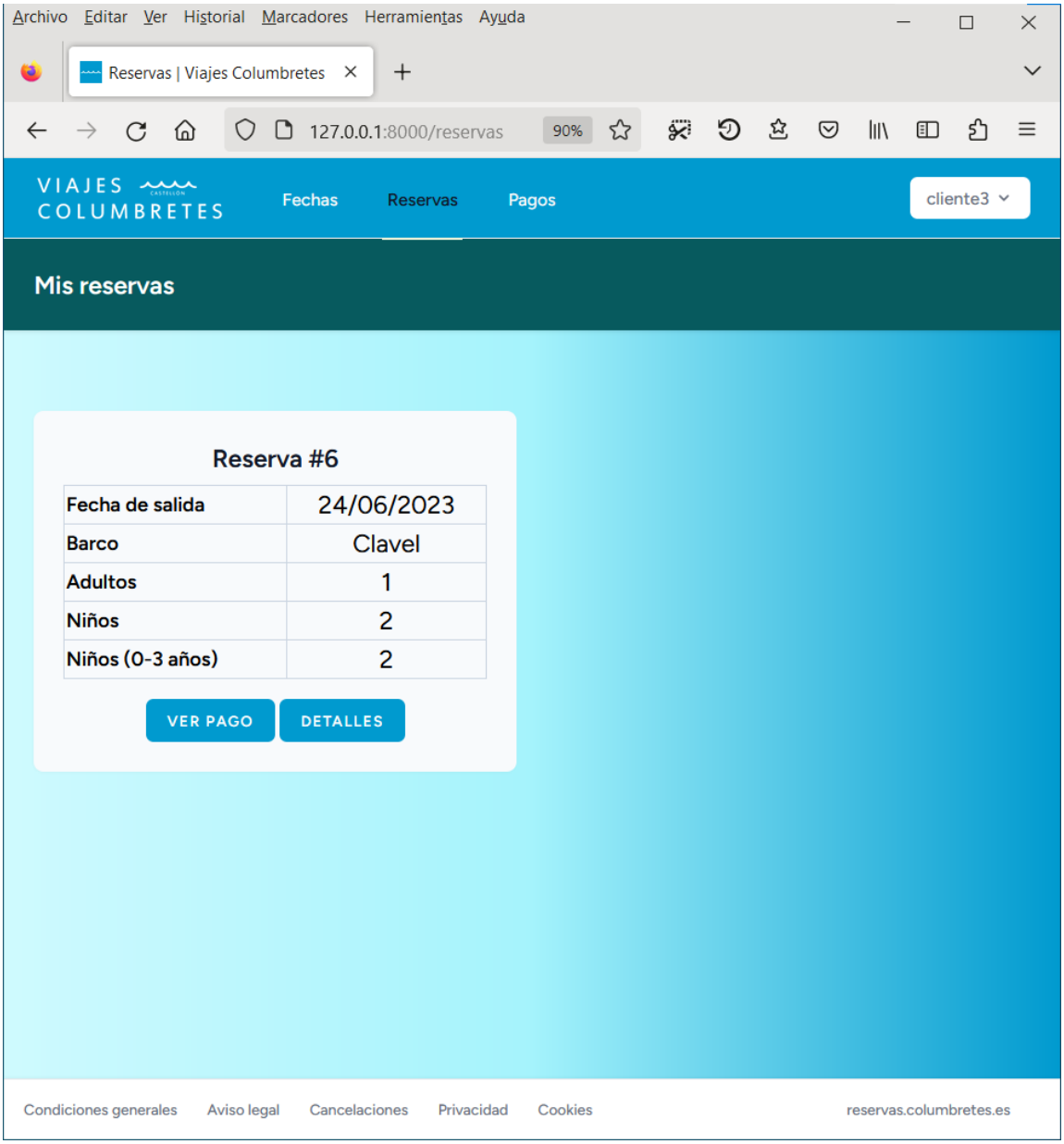


Figura 5.7: Interfaz final del listado de reservas

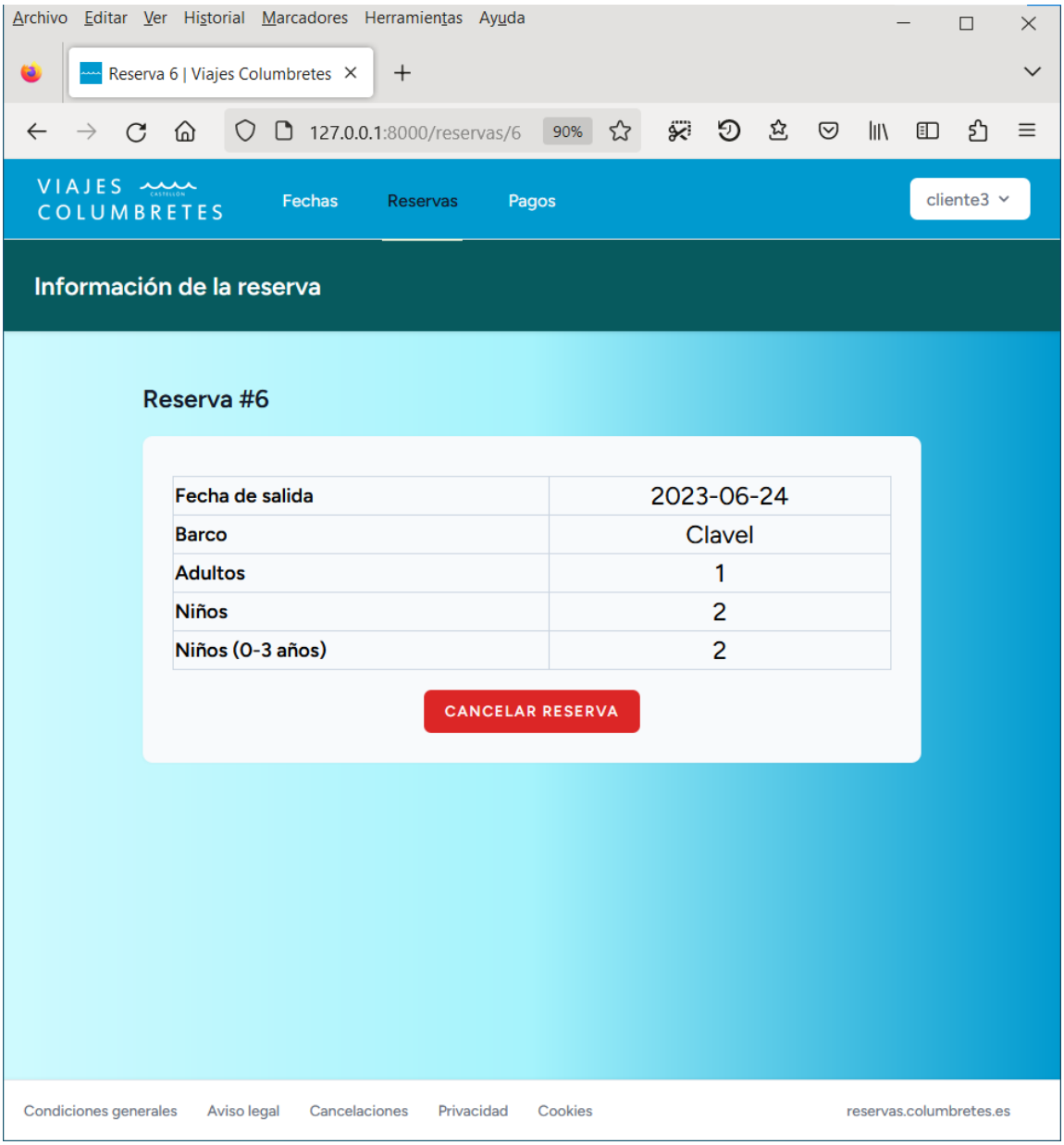


Figura 5.8: Interfaz final de información de una reserva concreta

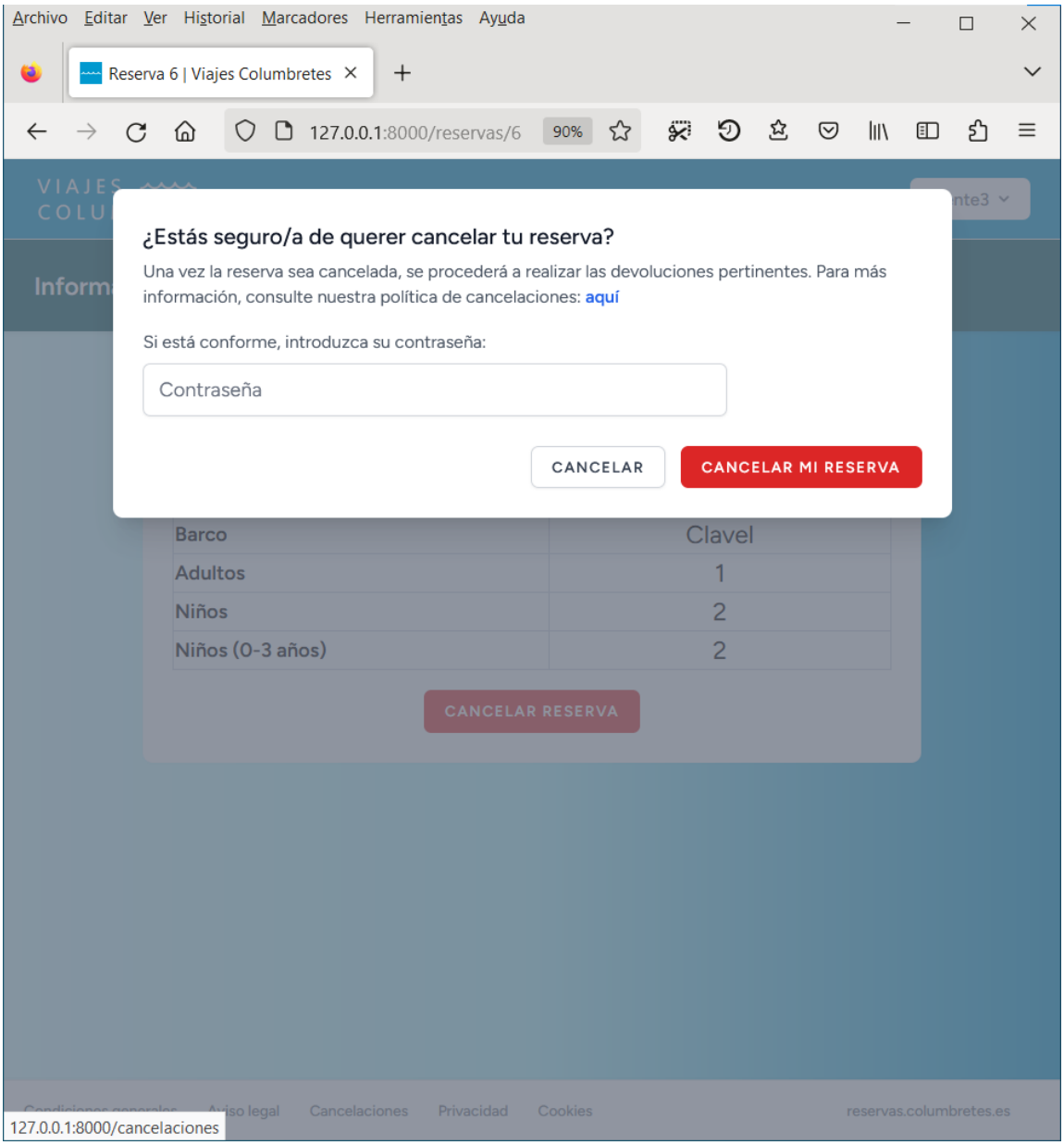


Figura 5.9: Interfaz final de cancelación de una reserva

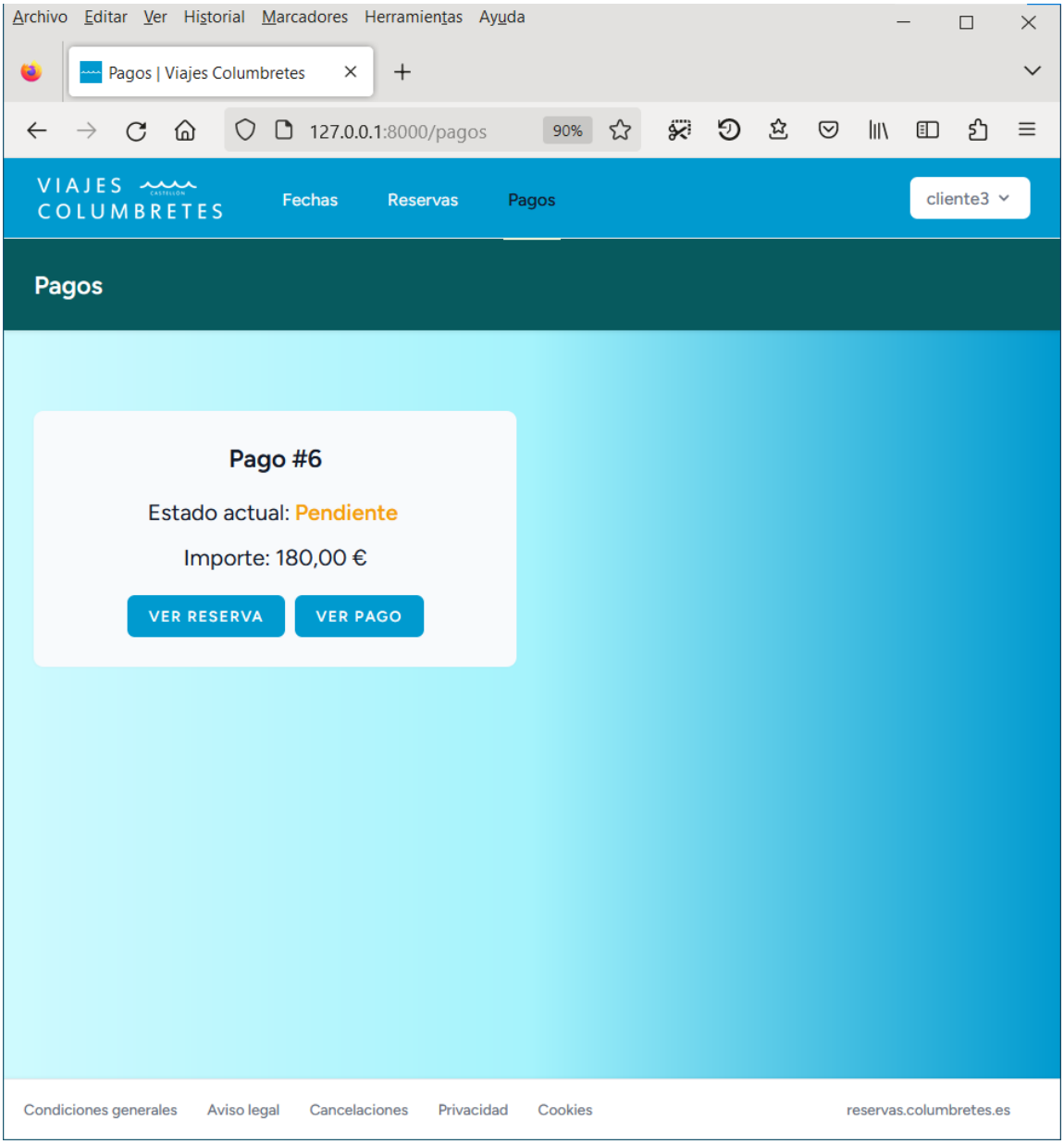


Figura 5.10: Interfaz final del listado de pagos

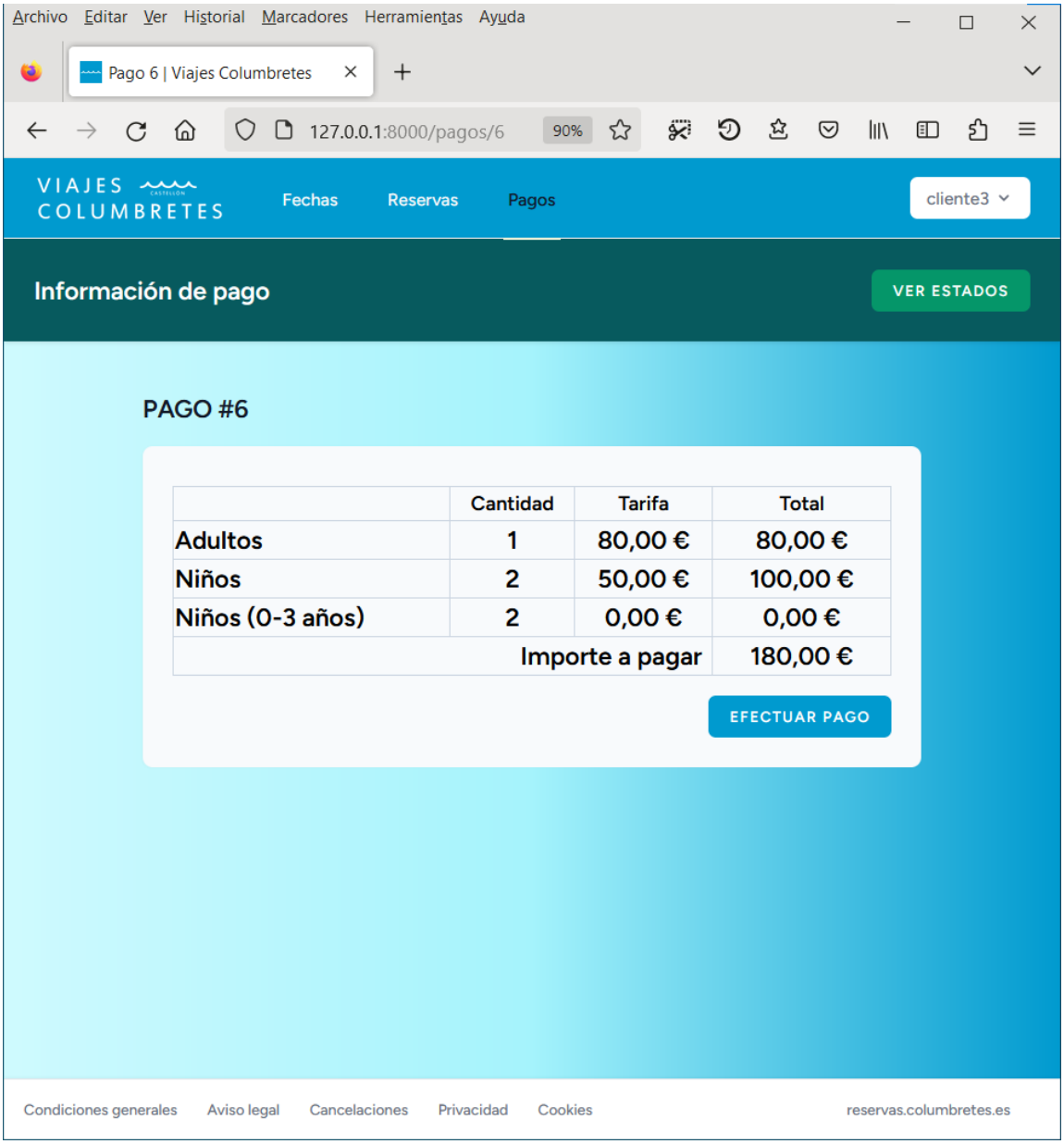


Figura 5.11: Interfaz final de información de un pago concreto

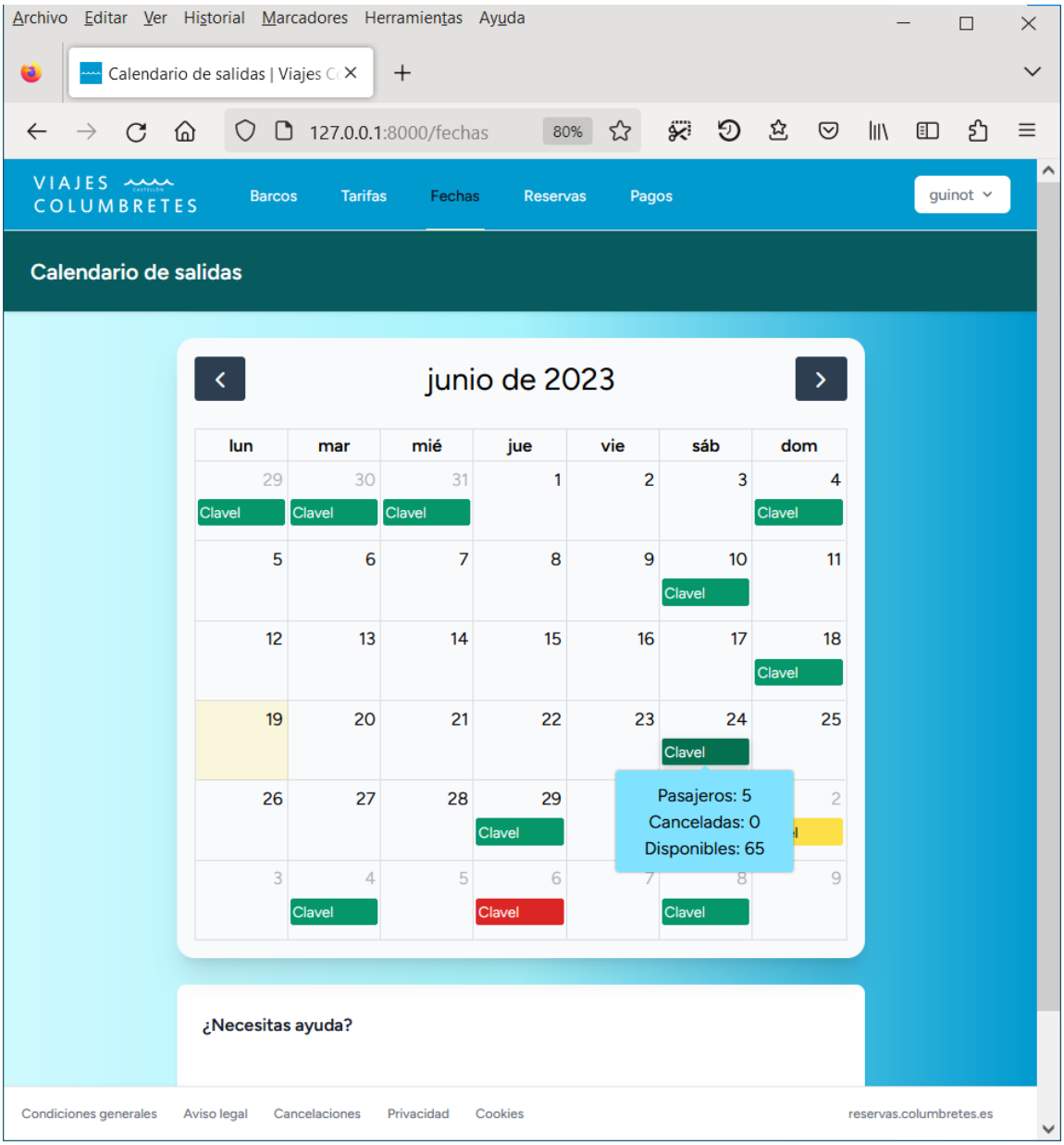


Figura 5.12: Interfaz final del calendario de salidas

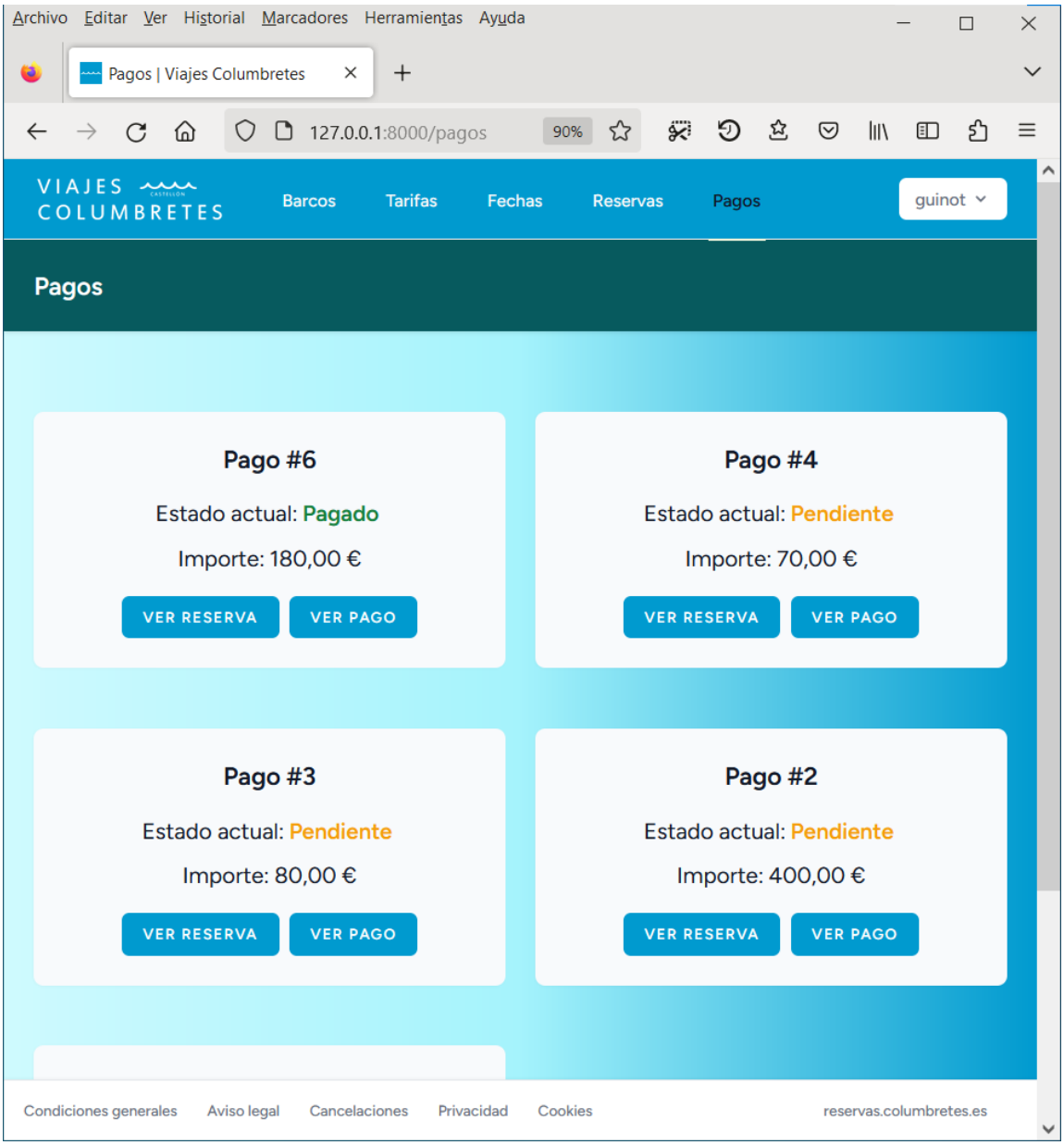


Figura 5.13: Interfaz final del listado de pagos (administrador)

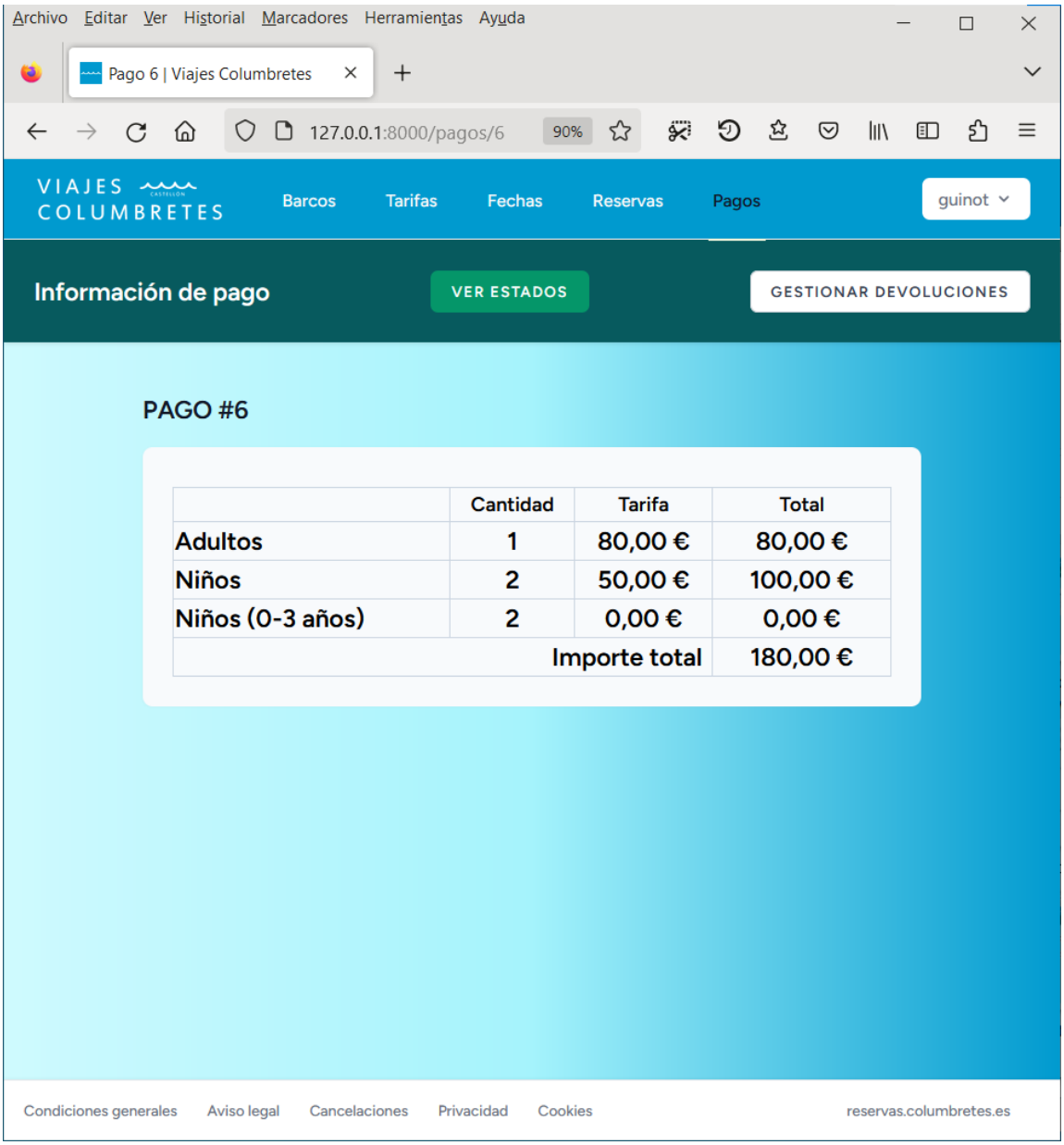


Figura 5.14: Interfaz final de información de un pago (administrador)

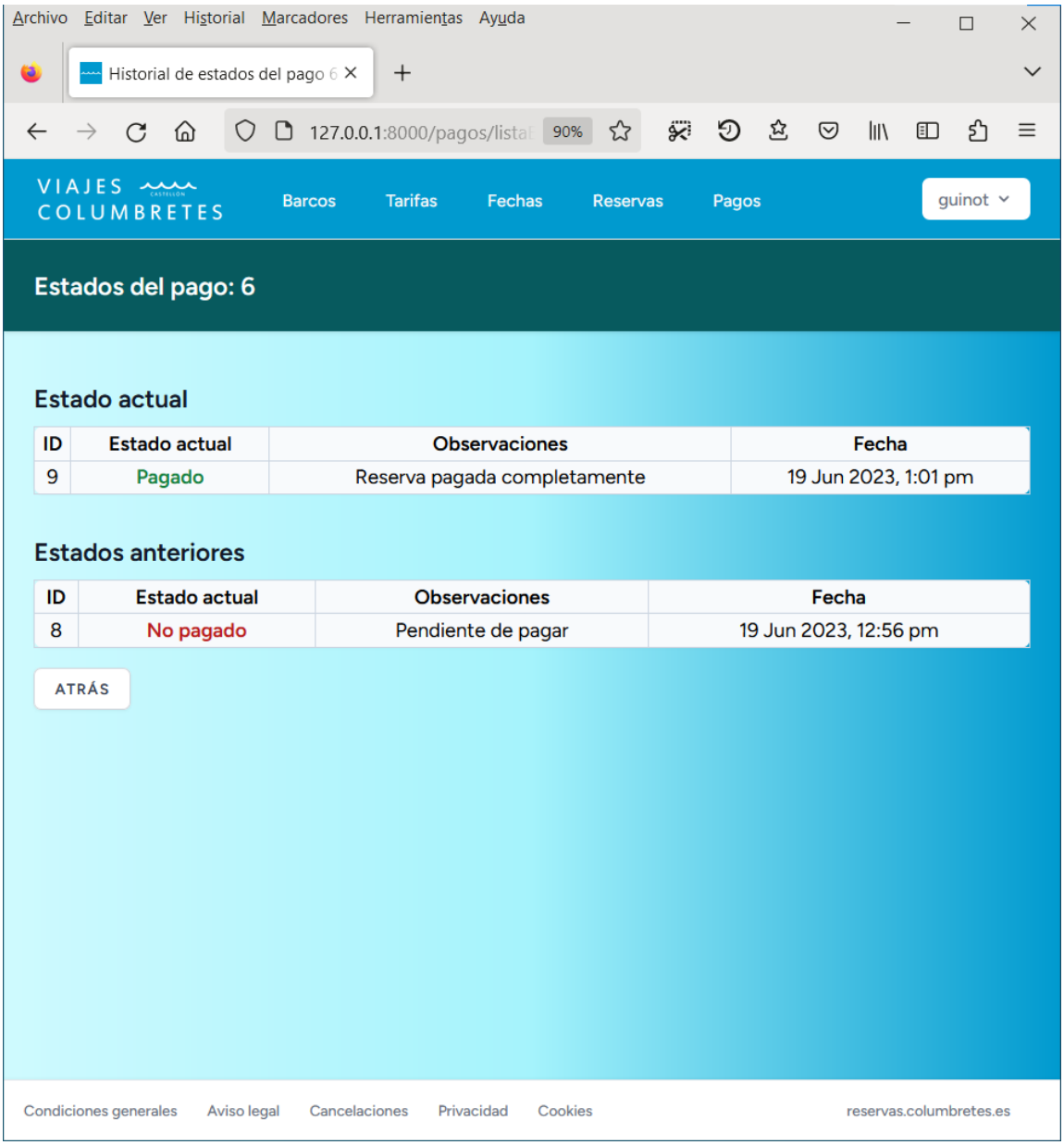


Figura 5.15: Interfaz final del histórico de un pago (tras realizar un pago)

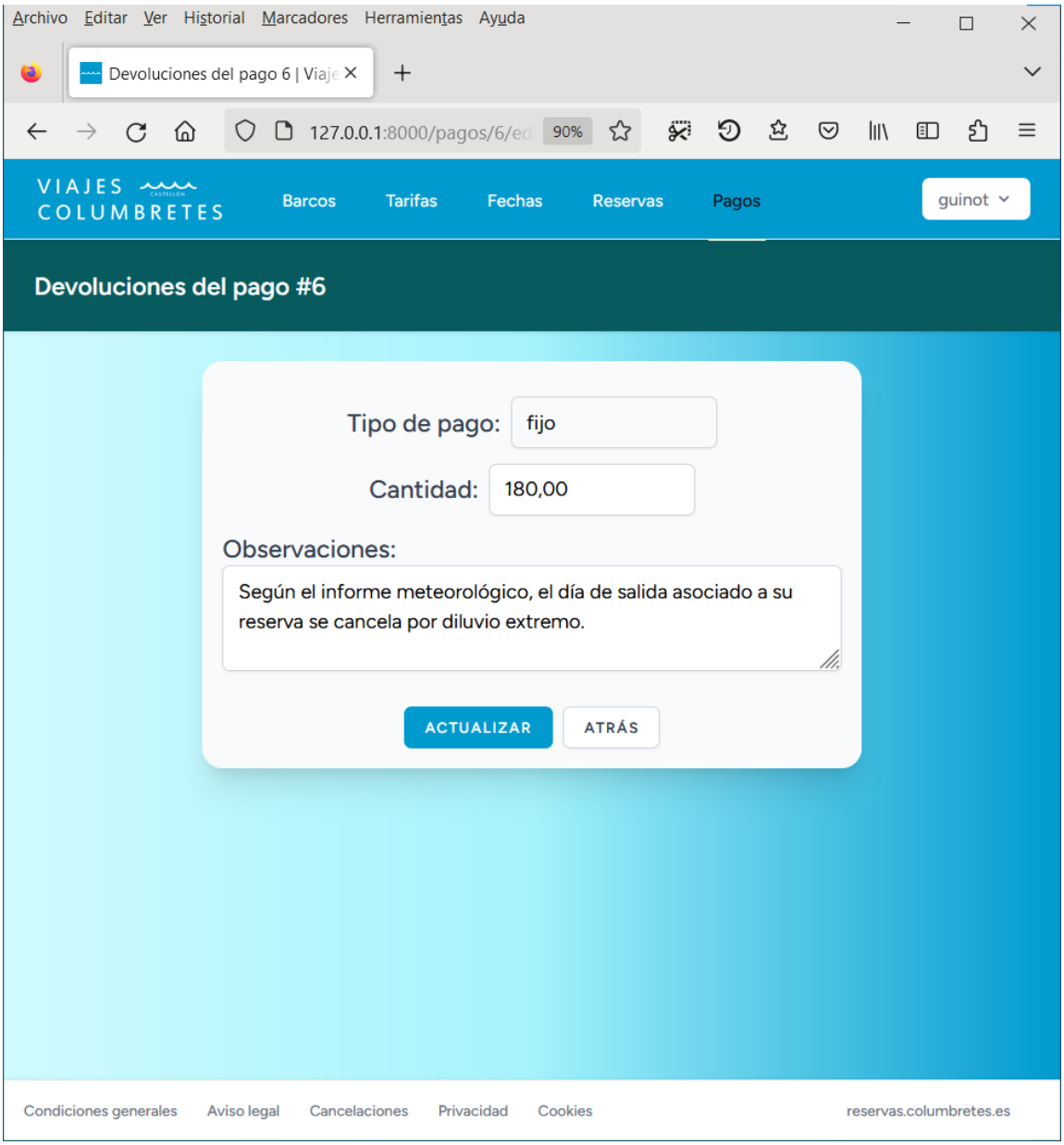


Figura 5.16: Interfaz final del formulario de las devoluciones

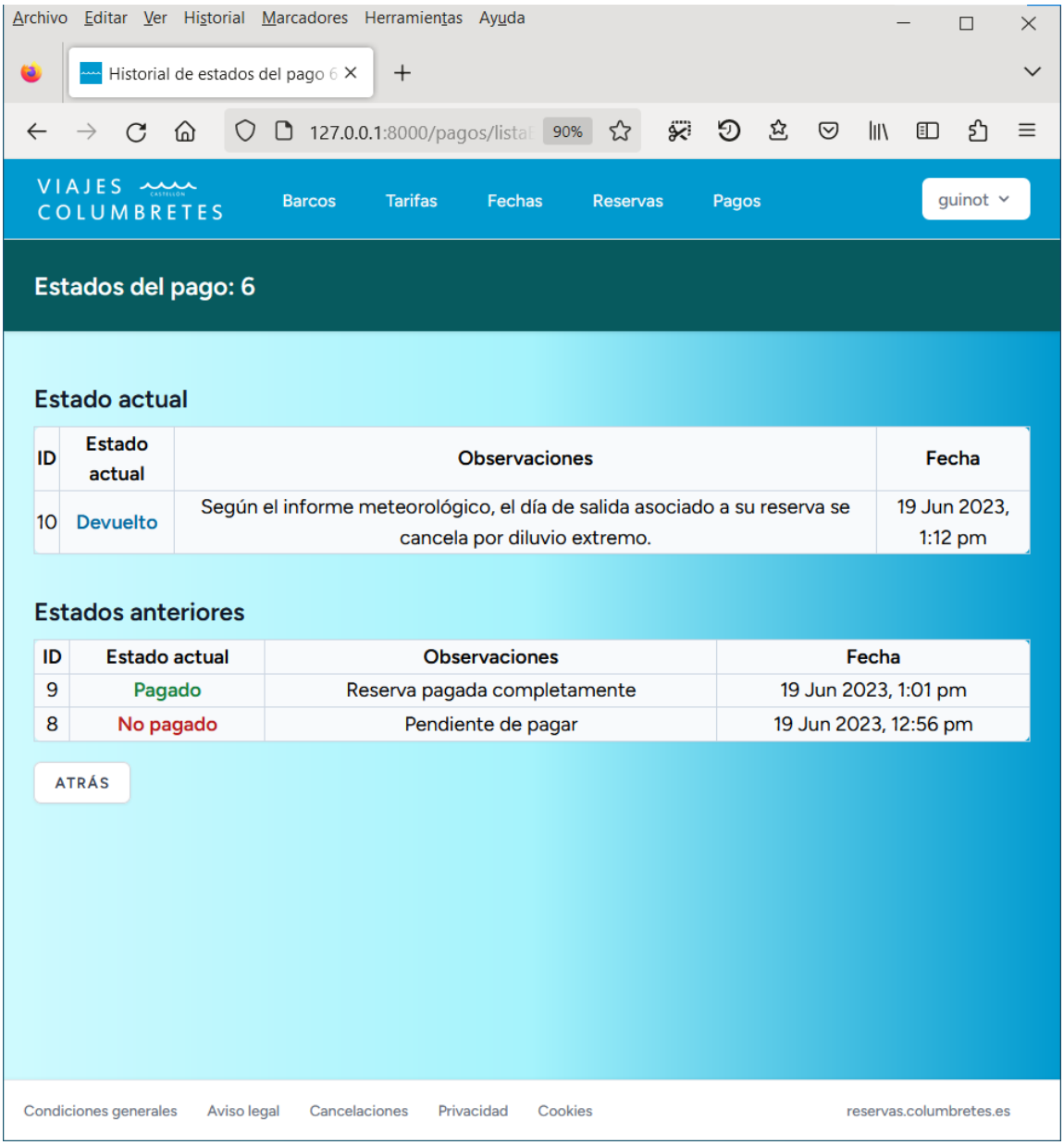


Figura 5.17: Interfaz final del histórico de un pago (tras realizar una devolución)

Capítulo 6

Conclusiones

Para concluir, he de comentar que el desarrollo del proyecto ha sido satisfactorio, a pesar de unos ligeros contratiempos que hicieron realizar una extensión adicional de una semana y que, aun así, se ha quedado muy cerca de completarse al 100 %. La flexibilidad que ha ofrecido Angal para gestionar la planificación es uno de los puntos que más he valorado a nivel personal y profesional durante la estancia, ya que ha sido el primer contacto en un entorno laboral y al principio era un terreno sin explorar en primera persona. El entorno en la empresa ha sido muy bueno y siempre ha estado a disposición para cualquier inquietud que me iba surgiendo. Es cierto, que el haber formado equipo con otras personas durante estos años en el grado, difiere en cierto modo con el mundo laboral, pero al fin y al cabo, tienen varios puntos comunes de los cuales he podido formar mi perspectiva laboral en el ámbito de ingeniería de *software*.

A nivel formativo estoy más que sorprendido por la calidad del producto. El itinerario de *Ingeniería de Software* me ha dado una base sólida de diferentes aspectos que he ido empleando como herramientas *software* y esquemas que han servido como guía inicial. En esencia, cada una de las asignaturas que he cursado anteriormente está representada en mayor o menor medida, aportando cada una de ellas su pequeño grano de arena.

Las tecnologías implementadas han sido bastante cómodas y no descarto en un futuro proponerlas como tecnologías a explorar y perfeccionar, en concreto el ecosistema de *Laravel Jetstream* [35], el cual es una extensión de *Laravel Breeze* con más funcionalidades. También he podido quitar el miedo a algunas tecnologías de las cuales no me veía bien ubicado como el lenguaje de estilos *CSS*. En cambio, *Tailwind* ha simplificado la curva de aprendizaje y ha hecho que el diseño de las vistas sea muy ameno.

Bibliografía

[1] Angal Informática S.L. - Empresa. https://www.angal.es/informatica-castellon. [Consultado el 8 de marzo de 2023].

[2] Excursiones Columbretes Castellón. https://www.excursionescolumbretes.com/. [Consultado el 6 de marzo de 2023].

[3] Booking. https://www.booking.com/index.es.html. [Consultado el 8 de marzo de 2023].

[4] Airbnb, Inc. https://www.airbnb.es. [Consultado el 8 de marzo de 2023].

[5] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute, 4th edition, 2008. [Consultado el 9 de marzo de 2023].

[6] Alicia Raeburn. EDT: cómo hacer uno para tu proyecto. https://asana.com/es/resources/work-breakdown-structure. [Consultado el 13 de marzo de 2023].

[7] Indeed.com. Sueldo de Programador/a junior en España. https://es.indeed.com/career/programador-junior/salaries?from=top_sb. [Consultado el 14 de marzo de 2023].

[8] Indeed.com. Sueldo de Programador/a senior en España. https://es.indeed.com/career/programador-senior/salaries?from=top_sb. [Consultado el 14 de marzo de 2023].

[9] Eloquent: Getting Started - Laravel - The PHP Framework For Web Artisans. https://laravel.com/docs/10.x/eloquent. [Consultado el 24 de marzo de 2023].

[10] MySQL. https://www.mysql.com. [Consultado el 8 de marzo de 2023].

[11] Laravel - The PHP Framework For Web Artisans. https://laravel.com. [Consultado el 7 de marzo de 2023].

[12] Framework - Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Framework. [Consultado el 15 de marzo de 2023].

[13] Blade Templates - The PHP Framework For Web Artisans. https://laravel.com/docs/10.x/blade. [Consultado el 8 de marzo de 2023].

[14] HTML: Lenguaje de etiquetas de hipertexto - MDN Web Docs. https://developer.mozilla.org/es/docs/Web/HTML. [Consultado el 3 de marzo de 2023].

[15] CSS - MDN Web Docs. https://developer.mozilla.org/es/docs/Web/CSS. [Consultado el 3 de marzo de 2023].

[16] Alexander Shvets (Refactoring.guru). Dive into Design Patterns - Observer pattern. https://refactoring.guru/design-patterns/observer. [Consultado el 11 de mayo de 2023].

[17] Balsamiq Wireframes - Industry Standard Low-Fidelity Wireframing Software — Balsamiq. https://balsamiq.com/wireframes/. [Consultado el 22 de marzo de 2023].

[18] PHP: Hypertext Preprocessor. https://www.php.net. [Consultado el 6 de marzo de 2023].

[19] Minimal Laravel authentication scaffolding with Blade, Vue, or React + Tailwind. https://github.com/laravel/breeze. [Consultado el 11 de junio de 2023].

[20] Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. https://tailwindcss.com/. [Consultado el 11 de junio de 2023].

[21] Visual Studio Code - Code editing. Redefined. https://code.visualstudio.com. [Consultado el 6 de marzo de 2023].

[22] Git. https://git-scm.com. [Consultado el 16 de marzo de 2023].

[23] GitHub. https://github.com. [Consultado el 3 de marzo de 2023].

[24] Alberto Chamorro. Conventional Commits - Qué es y por qué deberías empezar a utilizarlo. https://dev.to/achamorro_dev/conventional-commits-que-es-y-por-que-deberias-empezar-a-utilizarlo-23an. [Consultado el 18 de junio de 2023].

[25] Starter Kits - Laravel - The PHP Framework For Web Artisans. https://laravel.com/docs/10.x/starter-kits#laravel-breeze. [Consultado el 8 de marzo de 2023].

[26] Directory Structure - Laravel - The PHP Framework For Web Artisans. https://laravel.com/docs/10.x/structure#the-http-directory. [Consultado el 11 de junio de 2023].

[27] Povilas Korop. When to Use Dependency Injection, Services and Static Methods. https://blog.quickadminpanel.com/laravel-when-to-use-dependency-injection-services-and-static-methods/. [Consultado el 11 de junio de 2023].

[28] JavaScript - MDN Web Docs - Mozilla. https://developer.mozilla.org/es/docs/Web/JavaScript. [Consultado el 3 de marzo de 2023].

[29] Laravel Bootcamp. https://bootcamp.laravel.com/. [Consultado el 11 de junio de 2023].

[30] Composer. https://getcomposer.org/. [Consultado el 23 de marzo de 2023].

[31] FullCalendar - JavaScript Event Calendar. https://fullcalendar.io/. [Consultado el 11 de junio de 2023].

[32] Redsys — Servicios de procesamiento. http://www.redsys.es/. [Consultado el 11 de junio de 2023].

[33] Plesk - Innovative Hosting Control Panel. https://www.plesk.com/. [Consultado el 22 de mayo de 2023].

[34] Documentation and Help Portal for Plesk Obsidian. https://docs.plesk.com/es-ES/obsidian/. [Consultado el 22 de mayo de 2023].

[35] Laravel Jetstream. https://jetstream.laravel.com/3.x/introduction.html. [Consultado el 19 de junio de 2023].