

Memoria de Trabajo

Proyecto de Fin de Ciclo

Petland Resort



Autora: Dalieska Andrea Vieira Hernández

Tutor: Prof. Antonio Gabriel González Casado,

Centro: IES Alixar

Ciclo: Desarrollo de Aplicaciones Web

Curso académico: 2022-2023



Índice

| | |
|---|----|
| 1. Introducción | 2 |
| 2. Objetivos y Motivación | 2 |
| 3. Análisis de las alternativas del mercado | 3 |
| 4. Tecnologías empleadas | 4 |
| 4.1. Desarrollo | 5 |
| 4.1.1. Back | 5 |
| 4.1.2. Front | 5 |
| 4.2. Despliegue | 6 |
| 4.3. Útiles | 6 |
| 4.3.1. Construcción | 6 |
| 4.3.2. Control | 6 |
| 4.3.3. Diseño | 6 |
| 4.3.4. Pruebas | 7 |
| 4.3.5. Otros | 7 |
| 5. Diseño de la aplicación | 7 |
| 5.1. Funcionalidades | 7 |
| 5.2. Diseño de la base de datos | 11 |
| 5.2.1. Esquema | 11 |
| 5.2.2. Modelado de datos | 12 |
| 5.3. Vistas | 19 |
| 5.3.1. Wireframe | 19 |
| 5.3.2. Mockup | 20 |
| 6. Manual de despliegue | 26 |
| 7. Postmortem y conclusiones | 26 |



1. Introducción

En el documento que se presenta a continuación, se muestran las fases que componen el proyecto de fin de curso para el ciclo de Desarrollo de Aplicaciones Web. El proyecto pretende aunar en una pequeña aplicación todos los conocimientos adquiridos a lo largo de este ciclo, haciendo especial énfasis en los módulos del último año.

Para la realización de este proyecto se han tenido en cuenta distintos aspectos como que sea factible en tiempo, forma y contenido; que sea una muestra significativa de los conocimientos; y que responda a una necesidad real.

Se puede consultar el repositorio con toda la información y el proyecto a través de la siguiente url: <https://github.com/dandreavh/PetlandResort-ProyectoFinal>

2. Objetivos y Motivación

Son muchos los motivos por los que se ha realizado este proyecto. Por una parte, la motivación tecnológica, por el uso de un stack completo en JavaScript, un lenguaje de programación con gran impacto en la actualidad, una base de datos no relacional y un sistema, en general, de creación de aplicaciones muy rápido, moderno y con mucha flexibilidad.

Por otra parte, la idea de negocio ha sido el motor principal para trabajarlo con mimo y detalle, pues mi sueño siempre ha sido tener un espacio para el cuidado de animales; por lo tanto, este proyecto reúne pasión y profesión.

Como objetivo, además de la satisfacción personal de realizar un aplicativo web completo y funcional, se quiere responder a una necesidad de nuestro cliente, un complejo hostelero, Petland Resort, que se encarga de alojar mascotas y ofrecer una serie de servicios durante la estancia de estas.

El hotel quiere contar con una aplicación all-in-one propia que permita tener una gestión interna de sus recursos y que también sirva para que sus clientes puedan contratar estancias para sus mascotas. Este software debe contar con un sistema de almacenamiento en la nube, debe usar tecnologías de vanguardia, debe ser ligero en cuanto a su uso y fácil de escalar. Esta aplicación, además, debe

proyectar la imagen del hotel y ofrecer una experiencia de usuario muy satisfactoria.

En esta primera fase del proyecto, se quieren cubrir los requisitos mínimos para que el hotel pueda usar ya la aplicación en su día a día, es decir, se podrán registrar los usuarios, hacer reservas, contratar servicios, listas, modificar y eliminar datos, contar con un entorno que proporcione una correcta experiencia de usuario y disponer de un sistema para el almacenamiento de datos.

3. Análisis de las alternativas del mercado

Nuestra aplicación pretende ser una alternativa de PMS (Property Management System) o software de gestión hotelera destinado exclusivamente a hoteles cuyos huéspedes sean animales. Actualmente existen muchas alternativas de PMS que se utilizan en los hoteles convencionales, cada uno aporta distintas ventajas, que se adaptan a las necesidades de cada negocio. A continuación, se muestran los software más destacados que servirán de inspiración para nuestro proyecto.

Cloudbeds combina herramientas de operaciones, ingresos, distribución, marketing y crecimiento, con un marketplace de integraciones de terceros para ayudar a hoteleros y anfitriones a aumentar sus ingresos y agilizar operaciones, con sistemas de almacenamiento en la nube.

Little Hotelier ofrece distintos software para la gestión de complejos hoteleros entre los que destacan el de administración y el de reservas.

Hotelquest que es un software multiplataforma serverless que ofrece sistema de identificación sin usuarios con una aplicación web, por lo que no se requiere descarga y distintos procesos automatizados.

Existen muchos más y todos cuentan con características similares, pero ninguno destinado a el tipo de negocio que tiene el cliente de este proyecto, aunque los distintos módulos o complementos que se pueden contratar se pueden solventar las necesidades.

4. Tecnologías empleadas

Para este proyecto, se emplearán distintas tecnologías y aplicaciones en cada uno de los aspectos que forman parte de su desarrollo y explotación. La imagen que se muestra a continuación representa el uso de las mismas y tras dicha imagen, las explicaciones y razonamientos para cada una, añadiendo otros detalles empleados en el desarrollo.



mongoDB

Base de datos



Stack Back-End

express JS

HTML



<%= EJS

Stack Front-End

CSS



Bootstrap



Servidor de despliegue

Otras herramientas empleadas



4.1. Desarrollo

4.1.1. Back

- **MongoDB** como sistema de base de datos no relacional, por seguir la estructura de documentos JSON, ser de código abierto y la compatibilidad con JavaScript e ir en sintonía con las tecnologías punteras actuales.
- **MongoDB Atlas** como servicio de base de datos multi-cloud, que permite desplegar y manejar bases de datos de una forma sencilla y de última tecnología.
- **Mongoose** como librería que permite la conexión de la base de datos con Node.js.
- **JavaScript** como lenguaje de programación por su flexibilidad de uso tanto en el lado cliente como en servidor, por ser un lenguaje puntero con popularidad en proyectos modernos.
- **Node.js** como entorno en tiempo de ejecución para la capa del servidor, pues nos permite conectar las peticiones del cliente con la base de datos de una forma sencilla gracias a las librerías que nos permite incorporar, además de ser una tecnología moderna con mucha presencia actual.
- **Express.js** como entorno de trabajo en sintonía con Node, que nos permite la creación de nuestro aplicativo web de una forma sencilla y muy potente.

4.1.2. Front

- **EJS** como sistema de vistas que permite embeber JavaScript en los diseños de las páginas, de manera que resulta mucho más fácil controlar la información del servidor y la experiencia de usuario a nivel visual, muy integrado en HTML y de muy fácil aprendizaje, además, permite tener un proyecto monolítico con las tecnologías punteras.
- **HTML5** como lenguaje de marcado, por ser el estándar de referencia para todos los navegadores, siendo además fácil de usar, con extensiones para VSC y rápida implementación con las plantillas de EJS.



- **CSS3** como hoja de estilo para el diseño de la interfaz por su alta aceptación, comunidad y documentación. Además por su facilidad para integrarse en el HTML y contar con librerías como Bootstrap.
- **Bootstrap** como framework para el diseño de vistas de la aplicación, por su fácil integración, uso y comunidad, que respalda y facilita la integración de componentes y estilos accesibles.

4.2. Despliegue

- **Railway** como servidor de despliegue del proyecto, por su facilidad de uso, compatibilidad con las tecnologías empleadas y la disponibilidad de 500 horas de uso gratuitas.

4.3. Útiles

4.3.1. Construcción

- **Sistema operativo Windows 10**, que cuenta con una alta compatibilidad con las herramientas que se emplean, gran aceptación y respaldo, y facilidad de uso.
- **Navegador Chrome**, por su facilidad de uso y las extensiones que permite incorporar, siendo, además, el navegador más utilizado tanto en ordenadores como en móviles.
- **Visual Studio Code** como editor de código por su ligereza, facilidad de uso y las extensiones que ayudan enormemente al desarrollo como son: GitLens, JavaScript Code Snippets, Tabnine, Auto Close Tag, Auto Rename Tag, Prettier y Color Highlight.

4.3.2. Control

- **GitHub** para alojar el proyecto y tener un control de versiones, y acceso al mismo desde Internet.
- **Trello** como herramienta para administrar el desarrollo ágil del proyecto.

4.3.3. Diseño

- **Figma** como editor gráfico para el diseño visual de las páginas tanto para el wireframe como el mockup, pues facilita el diseño por sus distintas funcionalidades como la creación de componentes, grupos y las extensiones que se le pueden añadir como Iconify, el plugin de los iconos.



4.3.4. Pruebas

- **Postman** como cliente para usar las API antes de relacionarlas con vistas, permite organizar las distintas peticiones y tratar JSON, es fácil de instalar y usar.

4.3.5. Otros

- **Google Documents** para la creación del manual de despliegue y la memoria del proyecto. Esta herramienta permite tener toda la información almacenada en la nube de manera inmediata, nos aporta un sistema de indexado automático, personalización de estilos, realización de comentarios y la conversión a PDF, entre muchas otras opciones, además de un sistema seguro de disponibilidad, acceso y facilidad de uso.
- **Google fonts** como directorio de fuentes para la web, por ser gratuito y ser propiedad de una empresa con reconocimiento mundial.

5. Diseño de la aplicación

Para poner en marcha el proyecto es necesario definir una serie de elementos previamente.

5.1. Funcionalidades

En primer lugar, se deben tener en cuenta qué requisitos funcionales debe cumplir el sistema. Estos requisitos están identificados y priorizados, por lo que el desarrollo se realiza, preferiblemente, por orden de prioridad.

| ID | Prioridad | Título | Descripción |
|----|-----------|----------|--|
| F1 | 100 | Acceso | Un usuario anónimo debe poder acceder a la página de inicio, ver los servicios, las instalaciones, la ubicación de la empresa e interactuar con el sistema |
| F2 | 100 | Registro | Un usuario anónimo se debe poder registrar |



| | | | |
|-----|-----|---------------------|--|
| F3 | 100 | Inicio sesión | Un usuario registrado con cualquier rol debe poder iniciar sesión |
| F4 | 90 | Cierre sesión | Un usuario registrado con cualquier rol debe poder cerrar sus sesión sin perder sus datos |
| F5 | 90 | Editar perfil | Un usuario registrado con cualquier rol debe poder modificar algunos datos de su perfil |
| F6 | 100 | Resetear contraseña | Un usuario registrado con cualquier rol debe poder resetear su contraseña en caso de olvido |
| F7 | 50 | Darse de baja | Un usuario registrado con rol cliente debe poder solicitar darse de baja, perdiendo el acceso al servicio |
| F8 | 100 | Añadir mascota | Un usuario registrado con rol cliente debe poder añadir mascotas a su perfil |
| F9 | 100 | Editar mascota | Un usuario registrado con rol cliente debe poder modificar algunos datos de sus mascotas |
| F10 | 100 | Eliminar mascota | Un usuario registrado con rol cliente debe poder solicitar la eliminación de una mascota de su espacio (No debe ser una eliminación real en la bd) |
| F11 | 100 | Crear reserva | Un usuario registrado con rol cliente debe poder realizar la reserva de una habitación |
| F12 | 100 | Editar reserva | Un usuario registrado con rol cliente debe poder editar una reserva que tenga activa |
| F13 | 100 | Eliminar reserva | Un usuario registrado con rol cliente debe poder eliminar una reserva que tenga activa |
| F14 | 80 | Listar reservas | Un usuario registrado con rol cliente debe poder ver todas sus reservas, activas o no |
| F15 | 80 | Listar mascotas | Un usuario registrado con rol cliente debe poder ver todas sus mascotas |





| | | | |
|-----|-----|---|---|
| F16 | 60 | Contratar servicio | Un usuario registrado con rol cliente debe poder añadir servicios para cada mascota en una reserva |
| F17 | 50 | Editar servicio contratado | Un usuario registrado con rol cliente debe poder editar alguno de los servicios contratados para cada mascota en una reserva |
| F18 | 40 | Eliminar servicio contratado | Un usuario registrado con rol cliente debe poder eliminar alguno de los servicios contratados para cada mascota en una reserva. |
| F19 | 80 | Listar servicios contratados | Un usuario registrado con rol cliente debe poder ver todos los servicios que ha contratado durante la estancia de sus mascotas |
| F20 | 100 | Listar usuarios | Un usuario registrado con rol administrador puede ver todos los usuarios del sistema y filtrar si están activos o no |
| F21 | 90 | Listar clientes | Un usuario registrado con rol gerente o recepcionista puede listar los usuarios con rol cliente y filtrar si están activos o no |
| F22 | 50 | Ver ganancias | Un usuario registrado con rol gerente puede visualizar una gráfica con las ganancias mensuales obtenidas por las estancias |
| F23 | 30 | Ver estadística de servicios | Un usuario registrado con rol gerente puede visualizar una gráfica con los servicios más demandados |
| F24 | 30 | Ver estadística de tipo de habitaciones | Un usuario registrado con rol gerente puede visualizar una gráfica con el tipo de habitaciones más demandado |
| F25 | 50 | Listar empleados | Un usuario registrado con rol gerente puede visualizar las fichas de los empleados |
| F26 | 60 | Dar de baja empleado | Un usuario registrado con rol gerente puede dar de baja a un empleado |





| | | | |
|-----|----|------------------------------|---|
| F27 | 90 | Dar de alta empleado | Un usuario registrado con rol gerente puede dar de alta a un empleado |
| F28 | 90 | Crear servicio | Un usuario registrado con rol gerente puede crear servicios ofrecidos por el resort |
| F29 | 90 | Eliminar servicio | Un usuario registrado con rol gerente puede eliminar servicios ofrecidos por el resort. (No debe ser una eliminación real en la bd) |
| F30 | 90 | Editar servicio | Un usuario registrado con rol gerente puede modificar datos de los servicios ofrecidos por el resort |
| F31 | 90 | Ver servicios | Todos los usuarios pueden ver los servicios disponibles |
| F32 | 50 | Ver habitaciones | Un usuario registrado con rol gerente o recepcionista puede ver las habitaciones disponibles por día |
| F33 | 90 | Asignar habitación | Un usuario registrado con rol gerente o recepcionista debe poder asignar una habitación a una reserva |
| F34 | 40 | Ver mascotas asignadas | Un usuario registrado con rol cuidador debe poder ver las mascotas que le han sido asignadas para su cuidado |
| F35 | 30 | Generar informe | Un usuario registrado con rol cuidador debe poder generar un informe diario de cada mascota que tiene asignada |
| F36 | 10 | Ver informes | Un usuario registrado con rol cuidador o recepcionista debe poder listar los informes diarios de las mascotas |
| F37 | 10 | Enviar informe | Un usuario registrado con rol recepcionista debe poder enviar por correo a los clientes los informes de sus mascotas generados por los cuidadores |
| F38 | 10 | Añadir reserva al calendario | Un usuario registrado con rol cliente puede asociar su reserva a su |



| | | | |
|-----|----|-------------------|--|
| | | | calendario |
| F39 | 10 | Añadir reseña | Un usuario registrado con rol cliente puede añadir una reseña sobre su estancia |
| F40 | 10 | Añadir puntuación | Un usuario registrado con rol cliente puede añadir una puntuación de su estancia |

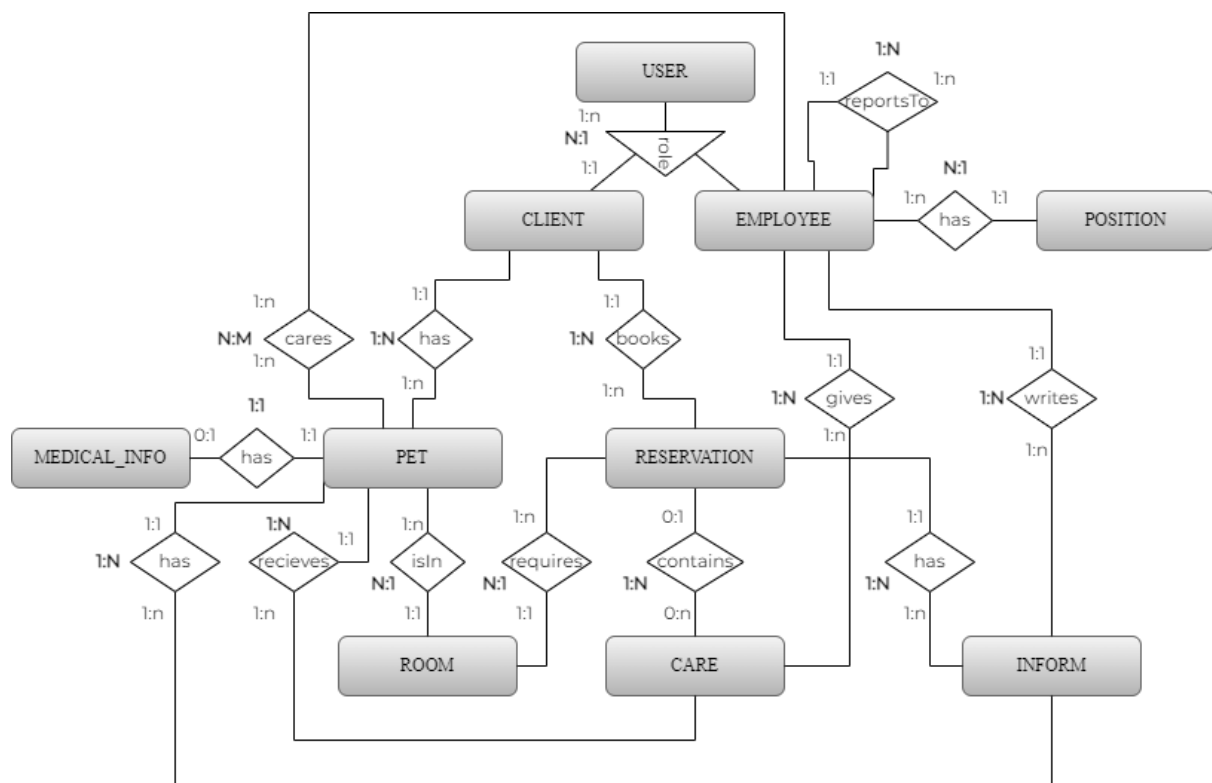
5.2. Diseño de la base de datos

Por otra parte, en cualquier proyecto es fundamental contar con un sistema que ayude a gestionar los datos contenidos, para ello, es fundamental disponer de un modelo de datos y la relación que existe entre las diferentes colecciones. Tal y como se ha mencionado en el apartado del stack tecnológico, para este proyecto se utilizará una base de datos no relacional, MongoDB, que nos permitirá tener una gran flexibilidad en cuanto a la información contenida.

5.2.1. Esquema

En el siguiente gráfico se muestra la relación de las distintas entidades que componen la base de datos del proyecto.

Encontramos la entidad User, de la nacen los tipos Client y Employee, pues tienen algunos atributos distintos. Entre los empleados encontramos el responsable de cada trabajador y su puesto en la empresa. Algunos empleados se encargan de las mascotas, presentan informes y prestan servicios. Los clientes, por otro lado, tienen mascotas y realizan reservas para las mismas. Cada mascota tiene una información médica y se aloja en una habitación, la cual es asignada en la reserva. Si observamos, las entidades con más impacto en nuestra base de datos son User, Pet y Reservation.



5.2.2. Modelado de datos

En este proyecto contamos con colecciones que se estructuran de la siguiente forma:

User:

username: {type: String, required: true, index: {unique: true}}, // identificador de tipo cadena con un nombre elegido por el usuario, el cual no puede estar registrado en la base de datos, es de ingreso obligatorio y se le crea un index para una búsqueda más eficiente

name: {type: String, required: true}, // nombre del usuario como campo obligatorio de tipo cadena

surnames: {type: String, required: true}, // apellidos del usuario como campo obligatorio de tipo cadena

password: {type: String, required: true}, // contraseña del usuario, que cumplirá con unas validaciones y se almacenará encriptada. Este campo es obligatorio y de tipo cadena



idnumber: {type: String, required: true, index: {unique: true}}, // número de identidad del usuario como campo obligatorio, único e indexado para facilitar la búsqueda y de tipo cadena

email: {type: String, required: true, index: {unique: true}}, // correo electrónico del usuario como campo obligatorio, único e indexado para facilitar la búsqueda y de tipo cadena

phone: {type: String, required: true}, // número de teléfono del usuario como campo obligatorio y de tipo cadena

address: {type: String, required: true}, // dirección del usuario como campo obligatorio y de tipo cadena

birthday: {type: Date, required: true}, // fecha de nacimiento del usuario como campo obligatorio y de tipo fecha

register_date: {type: Date, default: Date.now, required: true}, // fecha en la que se realiza el registro del usuario (se crea), que tiene por defecto la fecha en la que se realiza, siendo de almacenamiento obligatorio y de tipo fecha

avatar: {type: String, default: 'XXX'}, // url de la foto de perfil del usuario, que será de tipo cadena y que tendrá un valor por defecto en caso de no introducir ninguno

status: {type: String, enum: ['inactive', 'active'], default: 'active'}, // estado del usuario para el sistema, de manera que si se encuentra activo podrá realizar todas las acciones correspondientes y si está inactivo, solo queda registrado en la base de datos, pero no tiene acceso a nada. Este campo es de tipo cadena y sólo admite dos valores, siendo 'active' el valor por defecto

role: {type: String, enum: ['client', 'employee'], default: 'client'}, // rol del usuario para el sistema, solo se podrán crear 'client' desde el registro "normal", siendo el administrador la única persona que puede asignar el rol de empleado. Este campo es de tipo cadena y sólo admite dos valores, siendo 'client' el valor por defecto

----- Si el 'role' es 'client': -----

pets: {type: Array, [{type: mongoose.Schema.Types.ObjectId, ref: Pet, default: null}] }, // array de mascotas del usuario, en el que se referencian los identificadores de las mismas, por defecto se carga en vacío. Este campo sólo está disponible si el usuario es de tipo 'client'





----- Si el 'role' es 'employee': -----

start_date: {type: Date, default: Date.now}, // fecha de incorporación en la empresa, que por defecto es la fecha en la que se crea el usuario

end_date: {type: Date}, // fecha de baja en la empresa, que se rellenará cuando el empleado se dé de baja

studies: [type: Array, {type: String}], // array con los estudios que tenga el empleado

```
position: {type: Object, {
  title: {type: String},
  salary: {type: Number},
  reportsTo: {type: mongoose.Schema.Types.ObjectId},
}
```

}, // este elemento hace se trata de un documento embebido donde se almacenan los datos del puesto que desempeña el empleado en cuestión, teniendo un 'title' donde se almacena el nombre del puesto, el 'salary' con la cantidad que cobra, y 'reportsTo' donde se indica el identificador del encargado de esa persona (el único empleado que no tendrá este campo es la persona propietaria de la empresa)

**Pet:**

_id: (autogenerado), // identificador autogenerado por Mongo único y eficiente

register_date: {type: Date, default: Date.now, required: true}, // fecha en la que se registra a la mascota que, por defecto, será la actual y es de tipo obligatoria

name: {type: String, required: true}, // nombre de la mascota obligatorio

birthday: {type: Date, required: true}, // fecha de nacimiento de la mascota obligatoria

chip: {type: String}, // el código del chip que tenga integrado, si lo tiene

specie: {type: String, enum: ['dog', 'cat', 'bird', 'rodent', 'fish', 'other'], default: 'dog', required: true}, // tipo de mascota, que solo podrá ser solo de alguno de los tipos que admite el hotel, siendo el perro el valor por defecto. Este campo es obligatorio de rellenar

breed: {type: String}, // detalle de la raza de la mascota

size: {type: String, enum: ['small', 'medium', 'large'], default: 'medium'}, // tamaño de la mascota, que estará en un rango, siendo por defecto mediano el valor asignado

color: {type: String}, // color de la mascota

friendly: {type: Boolean, required: true, default: true}, // campo boolean para indicar si la mascota es sociable con otras mascotas de su misma especie u otras

avatar: {type: String, default: 'XXX'}, // url de la foto de la mascota, que será de tipo cadena y que tendrá un valor por defecto en caso de no introducir ninguno

comments: {type: String}, // campo de tipo cadena para introducir comentarios de la mascota

status: {type: String, enum: ['active', 'inactive'], default: 'active'}, // campo para indicar el estado de la mascota, si está activa o no, en el caso de no estar activo, no se mostrará pero seguirá almacenado en la base de datos

medical_info: {type: String}, // campo de tipo cadena que contiene información médica de la mascota, como enfermedades, el tipo y el tratamiento que tenga





```
caregiver: {type: mongoose.Schema.Types.ObjectId, required: true}, //  
campo que referencia al usuario al que pertenece esta mascota  
informs: [{type: Array, {type: Object, {  
  timestamp: {type: Date, default: Date.now},  
  author: {type: mongoose.Schema.Types.ObjectId},  
  shift: {type: String, enum: ['morning', 'afternoon', 'night']},  
  description: {type: String}  
  }}  
]}, // campo de tipo array que contiene objetos embebidos donde se  
registran los informes que se generan de esta mascota. Cada informe tiene  
la fecha en la que se realiza, el autor (empleado) que lo rellena y la  
información que se añade
```



**Reservation:**

_id: (autogenerado), // identificador autogenerado por Mongo único y eficiente

checkin: {type: Date, required: true}, // fecha de entrada obligatoria

checkout: {type: Date, required: true}, // fecha de salida obligatoria

timestamp: {type: Date, default: Date.now}, // fecha de realización de la reserva, que por defecto es la que se toma en el momento en el que se realiza la reserva

room: {type: Object, {
 type: {type: String, enum: ['suite', 'pack suite', 'deluxe suite', 'deluxe pack suite'], default: 'suite'},
 assigned: {type: String},
 details: {type: String}}

}}, // campo de tipo objeto embebido que contiene datos donde se aloja la mascota. El tipo de habitación, que la selecciona el usuario que realiza la reserva. La asignación y los detalles solo son editables por los empleados, una vez la mascota esté en el centro.

client: {type: mongoose.Schema.Types.ObjectId}, // campo que referencia al usuario titular de la reserva

pets: [{type: Array, {type: mongoose.Schema.Types.ObjectId}}], // mascotas implicadas en la reserva, que serán las que se alojen. Se almacena en este array de objetos los identificadores de dichas mascotas

petsitter: [{type: Array, {type: mongoose.Schema.Types.ObjectId, default: null}}], // array de objetos que están a cargo de los cuidados de la/s mascota/s, haciendo referencia al identificador del empleado

observations: {type: String}, // campo de tipo cadena para introducir las observaciones necesarias sobre la reserva; este campo es editable por el usuario cliente

cares: [{type: Array, {type: Object, {
 description: {type: String},
 pet: {type: mongoose.Schema.Types.ObjectId},
 observations: {type: String},
 scheduled: {type: Date},











```
giver: {type: mongoose.Schema.Types.ObjectId},  
  }  
}] // campo de tipo array que almacena objetos embebidos donde se  
indican los cuidados seleccionados para ser aplicados a las mascotas. En  
este objetos encontramos la descripción del cuidado a ofrecer, la mascota  
receptora (con su id referenciado), las observaciones que sean necesaria, la  
fecha en la que se va a ofrecer el servicio y la persona encargada de darlo  
(haciendo referencia a su identificador de usuario)
```







5.3. Vistas

5.3.1. Wireframe

| Vistas | Usos |
|---|---|
|  | <p>Página de inicio (para todos los usuarios)</p> <p>Secciones:</p> <ul style="list-style-type: none"> - Menú superior - Acción directa a reserva - Presentación - Habitaciones - Servicios - Pie de página |
|  | <p>Página de registro (para todos los usuarios)</p> <p>Página de editar perfil (para los usuarios logados)</p> <p>Página de crear o editar una reserva (para los usuarios logados)</p> |
|  | <p>Pop-up para realizar el login (para todos los usuarios)</p> |
|  | <p>Página de resetear la contraseña (para todos los usuarios)</p> |
|  | <p>Página de inicio para un usuario logado de tipo cliente</p> |
|  | <p>Página de inicio para un usuario logado de tipo empleado</p> |



| | |
|---|---|
|  | Página de inicio para un usuario logado de tipo gerente |
|  | Página de inicio para un usuario logado de tipo administrador |
|  | <p>Página para ver listado de reservas (para los usuarios logados - clientes, empleados, gerentes y admin)</p> <p>Página para ver listado de mascotas (para los usuarios logados - clientes, empleados, gerentes y admin)</p> <p>Página para ver listado de usuario (para los usuarios logados - gerentes y admin)</p> <p>Página para ver listado de informes (para los usuarios logados - empleados, gerentes y admin)</p> |
|  | <p>Página para ver los informes (para los usuarios logados - empleados, gerentes y admin)</p> <p>Página para crear los informes (para los usuarios logados - empleados, gerentes y admin)</p> <p>Página para editar los informes (para los usuarios logados - empleados, gerentes y admin)</p> |

5.3.2. Mockup

| Vistas | Usos |
|--------|------|
|--------|------|

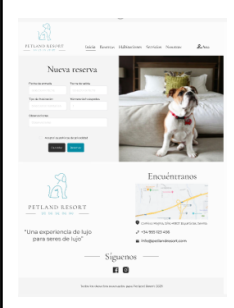


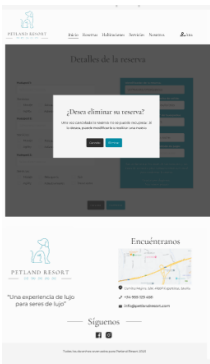
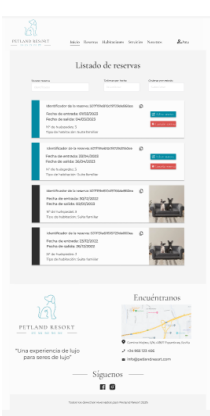
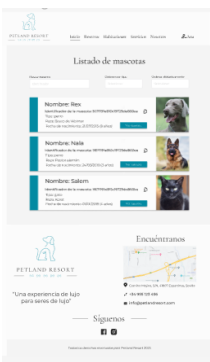
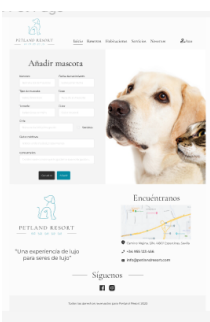


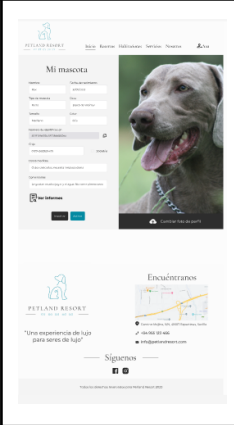
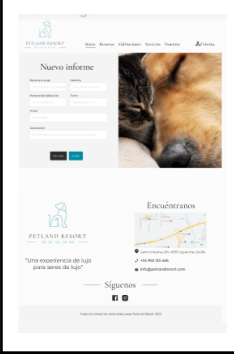
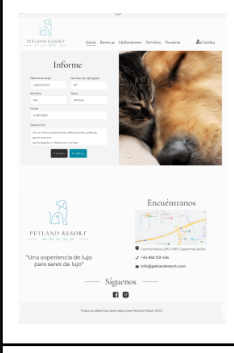

| | |
|--|---|
| | Página de inicio, donde llegan todos los usuarios |
| | Página para registrarse como cliente |
| | Pop-up para iniciar sesión |
| | Página para el restablecimiento de la contraseña |



| | |
|--|--|
| | Página de inicio al cliente tras el login |
| | Página de inicio a empleados que no son gerentes tras el login |
| | Página de inicio para los empleados gerentes tras el login |

| | |
|---|--|
|  | <p>Página de inicio del administrador de la aplicación tras el login</p> |
|  | <p>Página para iniciar una reserva</p> |
|  | <p>Página para añadir datos a la reserva</p> |
|  | <p>Pop-up para la confirmación de los datos</p> |

| | |
|---|--|
|  | <p>Pop-up para la cancelación de una reserva</p> |
|  | <p>Página para listar las reservas</p> |
|  | <p>Página para listar las mascotas</p> |
|  | <p>Página para añadir mascota</p> |

| | |
|---|---|
|  | <p>Página para ver y/o editar una mascota</p> |
|  | <p>Página para añadir informe</p> |
|  | <p>Página para ver y/o editar informe</p> |
|  | <p>Página para listar los informes</p> |

| | |
|--|---|
|  | Página para listar empleados |
|  | Página para listar clientes |
|  | Página para listar todos los usuarios del sistema |

6. Manual de despliegue

7. Postmortem y conclusiones