Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas

Trabajo Fin de Grado de Ingeniería Informática

PLATAFORMA WEB para la GESTIÓN de RESERVAS en RESTAURANTES

Autor: Rubén Mazo Jiménez

Tutor(a): Alejandro Rodríguez González

Madrid, junio 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas
Título: PLATAFORMA WEB para la GESTIÓN de RESERVAS en RESTAURANTES
Junio 2022

Autor: Rubén Mazo Jiménez

Tutor:
Alejandro Rodríguez González
Lenguajes y Sistemas Informáticos e Ingeniería del Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Con el paso del tiempo, la transformación digital en las empresas se ha ido estableciendo de forma que, a día de hoy, está a la orden del día para muchos procesos cotidianos de las mismas.

Un claro ejemplo de esta transformación digital es el surgimiento de plataformas web que habilitan a los usuarios realizar acciones que, anteriormente, eran realizadas por otros medios.

En concreto, una acción muy común, que hasta hace poco tiempo se ha realizado de forma telefónica o presencialmente en algunos establecimientos, es la de realizar una reserva. Se pueden realizar reservas de videojuegos, determinados productos en tiendas, habitaciones de hotel, actividades en un gimnasio, salas de estudio y otro tipo de servicios.

Muchas de estas reservas han modificado su forma de realizarse, proporcionando actualmente un entorno tecnológico moderno que facilita a los usuarios finales su realización en cualquier momento y evitando la dependencia directa de otras personas, dejando atrás los obsoletos métodos de reserva.

Este trabajo consiste en el desarrollo de una aplicación web que posibilita la gestión de reserva de mesas en un restaurante y que está construida en base a una interfaz usable y simple para los usuarios finales.

Los usuarios pueden añadir, ver, modificar o borrar sus reservas de mesa. Por otro lado, los administradores de la plataforma, que son un tipo de usuario concreto desde la perspectiva del propio restaurante, son capaces de añadir, ver, modificar o borrar mesas disponibles para los demás usuarios. Esto es posible gracias a la interfaz de la plataforma que favorece la usabilidad de la aplicación, así como de los procesos que actúan de forma invisible para el usuario y que garantizan la integridad de sus acciones. Todo ello, acaba resultando en un sistema final disponible para cualquier usuario.

Abstract

Over time, digital transformation in companies has been presented on a daily basis being very important for many of their processes.

A clear example of this digital transformation is the emergence of web platforms that enable users to carry out actions that were previously carried out by other means.

Specifically, a very common action, which until recent days has been carried out by telephone or in person at some establishments, is to make a reservation. Making a reservation includes videogames, certain products in stores, hotel rooms, activities in a gym, study rooms in libraries and other types of services.

Many of these reservations have changed the way they are made, they currently require a modern technological environment that makes it easier for end users to make them at any time and avoid direct dependency on other people, leaving behind obsolete reservation methods.

This project consists of the development of a web application that enables the management of table reservations in a restaurant and that is built on the basis of a usable and simple interface for end users.

Users can add, view, modify or delete their table reservations. On the other hand, the platform administrators, who are a specific type of user from the perspective of the restaurant itself, are capable of adding, viewing, modifying or deleting tables available to other users. This is possible thanks to the platform interface that favours the usability of the application, as well as the processes that act in an invisible way for the user and that guarantee the integrity of their actions. All this ends up resulting in a final functional system for any user.

Tabla de contenidos

Tabla de ilustraciones

Índice de tablas

1 Introducción

A lo largo de los años, el mundo empresarial ha ido evolucionado constantemente con el fin de ofrecer a los clientes los mejores productos y servicios. Se persigue la optimización de la calidad y los procesos, resultando en el aumento de la satisfacción del cliente.

Actualmente, muchas empresas optimizan estos procesos optando por incorporar las últimas mejoras tecnológicas, que consideran diferenciales en la elección final de los individuos. Siguen la denominada *transformación digital*, a través de la cual, los procesos que seguían metodologías obsoletas, quedan actualizados a la nueva era digital [1].

En concreto, un proceso frecuente antiguamente era realizar una reserva de forma telefónica, donde el individuo interesado quedaba limitado a llamar dentro del horario fijado por el establecimiento, incluso generando a veces demoras de tiempo. Otra alternativa era acudir presencialmente, lo que obligaba al sujeto a desplazarse al lugar del establecimiento. En cualquier caso, acababa resultando una tarea no demasiado agradable para la persona interesada.

Sin embargo, este proceso se ha adaptado tecnológicamente en algunas empresas facilitando a los usuarios realizar una reserva sin demoras de tiempo y sin necesidad de desplazarse. Otras empresas han comprobado la mejora que esto supone y han decidido poner en marcha esta transformación digital.

Este documento describe un proyecto que consiste en la creación y adaptación tecnológica del proceso de realizar reservas en un restaurante mediante la creación de una aplicación web, apoyada en una interfaz amigable y en procesos invisibles a los usuarios que garantizan la integridad de la plataforma.

La aplicación en cuestión facilita a los individuos crearse un perfil y autentificarse para gestionar sus reservas desde su dispositivo, evitando la dependencia de terceras personas y haciendo la tarea más amena que en anteriores metodologías.

1.1 Lista de objetivos del proyecto

El **objetivo principal** del proyecto se basará en desarrollar el sistema especificado. Será necesario cumplir una serie de objetivos parciales para la consecución del objetivo principal.

Los **objetivos parciales** del proyecto serán:

- El análisis de los requisitos generales del proyecto. Es decir, la realización de una investigación previa de las capacidades y limitaciones de las tecnologías usadas y como enlazarlas entre sí, así como la adquisición de conocimientos previos incluyendo el estudio de los lenguajes de programación a utilizar: HTML, CSS, JavaScript, Java.
- El diseño de las interfaces, APIs y bases de datos. Esto es, de manera secuencial al aprendizaje, la puesta en práctica del desarrollo individual e implementación de los distintos módulos que formarán parte de la plataforma final.
- La depuración de dichos módulos y el enlace entre ellos para el correcto soporte de la aplicación.

- La realización de casos de prueba para la comprobación del correcto funcionamiento de la plataforma y en caso de fallo, su corrección en los módulos.
- La documentación general del proyecto a través de la cual se actualizarán los avances conseguidos hasta la fecha.

1.2 Planificación: Diagrama de Gantt

Para el cumplimiento de las tareas principales y secundarias del proyecto, es necesario seguir una planificación que sirva como punto de partida para organizar el diseño y la implementación de la plataforma.

En la *Ilustración 1*, se muestra el diagrama de Gantt que plasma las diferentes tareas y plazos que se seguirán para cumplirlas. Este diagrama sirve para esquematizar las diferentes labores que se van a realizar y ayuda a la comprensión de las diferentes partes que conformarán la aplicación.

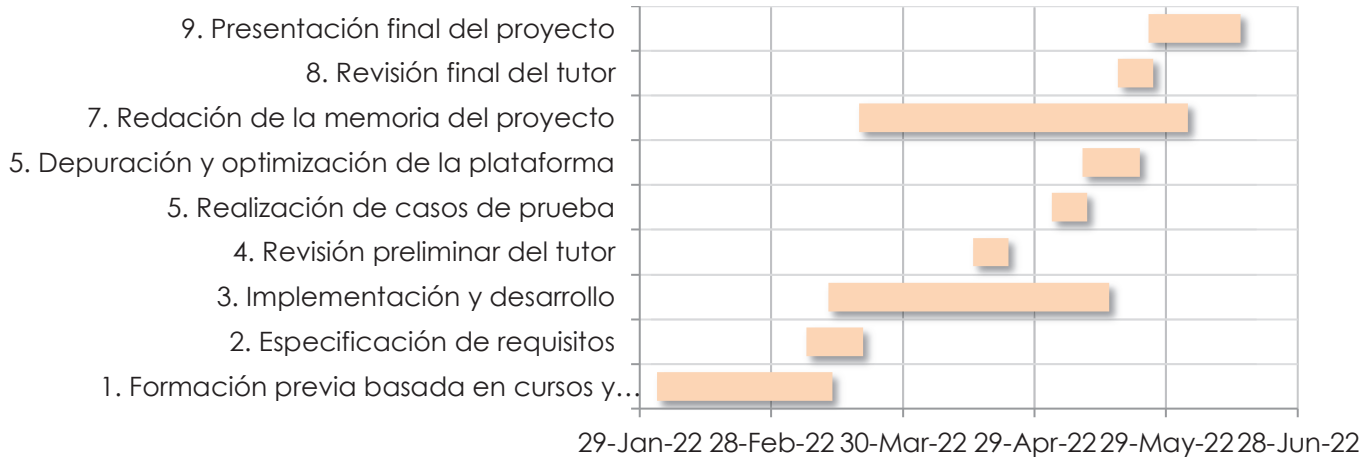

Ilustración 1: Diagrama de Gantt del proyecto.

1.3 Estado del arte: Aplicaciones similares

Con el fin de establecer un contexto que sirva como punto de partida para la realización del proyecto, se ha realizado una investigación de sistemas similares que han dado lugar a una plataforma interactiva de reservas.

Estos sistemas han perseguido unos objetivos muy similares a los establecidos en nuestro caso en el apartado *'1.2 Lista de objetivos del proyecto'*. Por tanto, resulta conveniente tomar como referencia los resultados obtenidos para las diferentes aplicaciones comentadas.

Existe una gran variedad de aplicaciones que tratan de abordar diferentes funcionalidades para gestionar reservas en diferentes ámbitos de uso. Estas aplicaciones han servido para gestionar reuniones y citas, o para organizar formaciones y eventos, entre otras cosas. Se exponen a continuación:

- **Bookitit:** Es un sistema para gestionar reservas online relacionadas con todo tipo de servicios. Se pueden realizar reservas en centros deportivos,

de belleza, organizar formaciones, cursos o eventos y demás funcionalidades. Las reservas se realizan a través de la web, página de Facebook o perfil de Instagram [2].

- **Super Saas:** Es una aplicación destinada a pequeñas y medianas empresas a través de la cual se gestionan reservas relacionadas con los negocios. Permite sincronizarse con el calendario de Google o crear eventos grupales [3].
- **Calendly:** Es un software destinado a gestionar reuniones y citas. Fue creada en el año 2013, siendo una de las primeras herramientas en abordar funcionalidades relacionadas con la gestión de reservas. A través de ella, se puede compartir un enlace de la reunión o cita a los demás invitados asistentes [4].

2 Análisis de requisitos funcionales

A continuación, se realiza un análisis previo de requisitos funcionales de la aplicación que, desde la perspectiva de la Ingeniería del Software, se consideran importantes para el posterior diseño y desarrollo de la plataforma.

A través de estos requisitos, será más fácil diferenciar las entidades que formarán parte del diseño de la aplicación y que evitarán posibles errores a la hora de enfocar la solución al problema. Por tanto, los requisitos funcionales del proyecto serán:

1. *Registro de un usuario*

   Un usuario podrá crear un perfil introduciendo una serie de datos que lo identifiquen. Gracias a este perfil, podrá acceder posteriormente a la gestión de sus reservas. Es la primera acción a realizar en la aplicación. Sin haber creado un perfil, no se puede acceder al resto de acciones del sistema.

2. *Inicio de sesión de un usuario*

   Un usuario podrá iniciar sesión posteriormente a la creación de su perfil. Deberá introducir el nombre de usuario y la contraseña para poder realizar esta acción correctamente.

3. *Fin de sesión de un usuario*

   Un usuario podrá cerrar sesión si tiene una sesión activa.

4. *Consulta de usuarios del sistema*

   Un usuario administrador podrá consultar todos los usuarios que existan en el sistema, incluyendo usuarios básicos y administradores.

5. *Consulta de datos de un usuario*

   Un usuario básico podrá consultar los datos concretos de sí mismo. Un usuario administrador podrá consultar los datos concretos de cualquier usuario.

6. *Modificación de datos de un usuario*

   Un usuario podrá editar datos de su cuenta como su nombre o contraseña. Sin embargo, no podrá modificar el nombre de usuario y contraseña utilizados para iniciar sesión.

7. *Borrado del perfil de un usuario*

   Un usuario podrá eliminar su perfil de forma que no exista más en el sistema y que no se pueda volver a iniciar sesión con él.

8. *Añadido de mesas*

Un usuario administrador podrá añadir una mesa al sistema indicando la extensión o capacidad de la mesa y la sala en la que se encontrará. Por defecto, cuando un administrador añade una mesa al sistema, ésta obtiene un estado *disponible* para su reserva.

9. Consulta de mesas del sistema

Un usuario podrá consultar todas las mesas que existan en el sistema.

10. Consulta de datos de una mesa

Un usuario podrá consultar los datos concretos de una mesa que exista en el sistema, para conocer su extensión y la sala donde se encuentra.

11. Modificación de datos de una mesa

Un usuario administrador podrá editar los datos (extensión de la mesa y sala donde se encuentra) de una mesa.

12. Eliminación de una mesa

Un usuario administrador podrá eliminar una mesa existente en el sistema.

13. Habilitación de una mesa

Un usuario administrador podrá habilitar una mesa existente en el sistema para su reserva, en caso de que la mesa no estuviera disponible previamente.

14. Deshabilitación de una mesa

Un usuario administrador podrá deshabilitar una mesa existente en el sistema, en caso de que la mesa estuviera disponible previamente. Esta acción se llevará a cabo, por ejemplo, cuando el restaurante esté reorganizando sus mesas en las diferentes salas.

15. Realización de reservas

Un usuario podrá realizar una reserva de una mesa existente en el sistema indicando el nombre de la reserva, la fecha y la hora para la que se va a reservar una determinada mesa. Una reserva de mesa en una determinada fecha y hora conlleva que los demás usuarios no puedan hacer una reserva en esa misma fecha y hora.

16. Consulta de reservas del sistema

Un usuario administrador podrá consultar todas las reservas que existan en el sistema.

17. Consulta de reservas de un usuario concreto

Un usuario básico podrá consultar todas las reservas que haya realizado él mismo para poder gestionarlas.

18. Consulta de reservas de una mesa concreta

Un usuario administrador podrá consultar todas las reservas para una mesa determinada del sistema y de esta manera saber su ocupación para los próximos días.

19. Consulta de datos de una reserva

Un usuario podrá consultar los datos de una reserva concreta para comprobar el nombre, fecha y hora para los que está realizada la reserva.

20. Modificación de una reserva

Un usuario podrá editar una reserva de forma que pueda cambiar el nombre, la fecha y la hora. De esta manera, la mesa quedaría libre para la fecha y hora antiguas y ocupada para la fecha y hora nuevas.

21. Eliminación de una reserva

Un usuario podrá eliminar una reserva de mesa si no va a acudir por la razón que sea. Al eliminar la reserva, la mesa queda libre para la fecha y hora para la que estaba reservada previamente.

3 Diseño

El diseño es el paso previo a la implementación y desarrollo de la plataforma, a través del cual se establecen las tecnologías que se van a utilizar, los elementos que van a interactuar dentro del sistema y de qué manera se van a relacionar, garantizando una integridad total para el usuario. Dentro de esta sección, se han establecido las siguientes subdivisiones: el Modelo Entidad-Relación y la Base de Datos, la API REST, el Prototipo de Interfaz y el Diagrama de Flujo.

3.1 Modelo Entidad-Relación

Es la base principal del diseño, ya que sirve para establecer qué entidades o elementos formarán las tablas de la base de datos de la aplicación y la relación entre las mismas [5]. La función principal de la aplicación es permitir que los usuarios realicen reservas de mesas en un restaurante, por tanto, las entidades del sistema serán:

- Usuario
- Mesa
- Reserva
- Ocupación de la mesa

Estas entidades y sus atributos, que definen sus relaciones, son definidos a continuación.

3.1.1 Usuario

El usuario simboliza a la persona que da uso a la aplicación y quiere gestionar sus reservas. Las acciones posibles a realizar sobre los usuarios son: registrarse, iniciar sesión, cerrar sesión, consultar los usuarios del sistema (si el usuario es administrador), consultar los datos de usuario, editar los datos de usuario y eliminar el perfil del usuario.

Teniendo en cuenta los requisitos estudiados previamente y las acciones necesarias para la entidad *usuario*, se establecen los atributos que se muestran en la *Ilustración 2* y que formarán las columnas de la tabla *usuario* posteriormente en la base de datos.

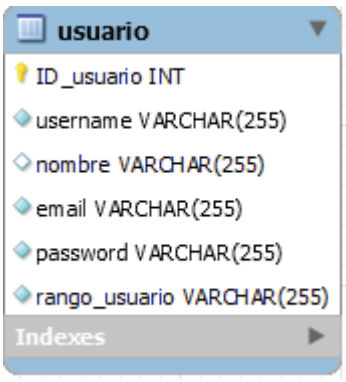

Ilustración 2: Entidad 'usuario' con sus atributos.

- **ID_usuario**: Corresponde al identificador de usuario, que es único para cada uno y forma la clave primaria de la tabla *usuario.*
- **username**: Corresponde al nombre de usuario con el que se puede iniciar sesión. Es único para cada usuario, por lo que no puede haber dos usuarios con el mismo *username.*
- **nombre**: Corresponde al nombre de la persona que se crea su perfil de usuario y realiza las demás acciones de la aplicación. No es único, por lo que puede haber varios usuarios con el mismo nombre.
- **email**: Corresponde al correo electrónico elegido por el usuario para crear su perfil. Es único para cada usuario, por lo que no puede haber dos usuarios con el mismo *email.*
- **password**: Corresponde a la contraseña del usuario con la que se puede iniciar sesión. Cada usuario debe recordar su contraseña en el proceso de autenticación, por lo que es un atributo de relativa importancia.
- **rango_usuario**: Corresponde al tipo de usuario registrado en el sistema, si es administrador o usuario básico. Con este atributo, se pueden distinguir las funciones que puede realizar el usuario que inicia sesión.

3.1.2 Mesa

La mesa simboliza al objeto principal del restaurante, sobre el que realiza las reservas el usuario. Las acciones posibles a realizar sobre las mesas son: añadir mesas, consultar las mesas del sistema, consultar los datos de una mesa, editar mesas, eliminar mesas y habilitar o deshabilitar mesas para su reserva.

Teniendo en cuenta los requisitos estudiados previamente y las acciones necesarias para la entidad *mesa*, se establecen los atributos que se muestran en la *Ilustración 3* y que formarán las columnas de la tabla *mesa* posteriormente en la base de datos.

Ilustración 3: Entidad 'mesa' con sus atributos.

- **ID_mesa**: Corresponde al identificador de mesa, que es único para cada una y forma la clave primaria de la tabla *mesa.*
- **extension**: Corresponde a la extensión o capacidad de la mesa, es decir, cuántos comensales pueden sentarse en una determinada mesa.
- **sala**: Corresponde al número de sala donde se encuentra la mesa. Sirve para poder situar cada mesa de forma correcta y para que los comensales puedan decidir en qué sala estar.
- **situacion**: Corresponde a la disponibilidad de la mesa para poder realizar una reserva sobre ella. No debe confundirse con su disponibilidad para una determinada fecha a una determinada hora, que se refiere a la ocupación de la mesa por otro usuario y que viene

recogido en la tabla *ocupacion_mesa*. Este campo muestra la habilitación de mesas de una forma más genérica, desde el punto de vista del administrador.

3.1.3 Reserva

La reserva simboliza la disposición de una mesa por parte de un usuario durante una fecha y una hora determinadas. Las reservas realizadas por los usuarios son de una hora, ya que es el tiempo estándar en el que los comensales terminan de comer o de cenar. Si un usuario quisiera emplear más de una hora en una mesa, deberá realizar dos reservas en horas consecutivas. Las acciones posibles a realizar sobre las reservas son: realizar reservas, consultar todas las reservas del sistema, consultar las reservas de un usuario determinado, consultar las reservas de una mesa determinada, consultar los datos de una reserva, editar reservas y eliminar reservas.

Teniendo en cuenta los requisitos estudiados previamente y las acciones necesarias para la entidad *reserva*, se establecen los atributos que se muestran en la *Ilustración 4* y que formarán las columnas de la tabla *mesa* posteriormente en la base de datos.

Ilustración 4: Entidad 'reserva' con sus atributos.

- **ID_reserva**: Corresponde al identificador de reserva, que es único para cada una y forma la clave primaria de la tabla *reserva*.
- **nombre**: Corresponde al nombre o descripción de la reserva que se añade al realizarla y sirve para describir de qué trata una reserva en cuestión. No es único, por lo que puede haber varias reservas con el mismo nombre. Un ejemplo de este campo podría ser 'Cena con amigos'.
- **fecha**: Corresponde al día concreto en que se realiza una reserva.
- **hora**: Corresponde a la hora concreta en que se realiza una reserva.
- **ID_usuario**: Corresponde al identificador de usuario que está realizando una determinada reserva. Actúa como clave foránea de la tabla *usuario*.
- **ID_mesa**: Corresponde al identificador de mesa para la que se está realizando una determinada reserva. Actúa como clave foránea de la tabla *mesa*.

3.1.4 Ocupación de la mesa

La ocupación de la mesa simboliza a aquellas mesas ocupadas para una fecha y hora determinadas. Sirve para controlar la disponibilidad de las mesas en el tiempo de forma que los usuarios puedan realizar sólo las reservas de mesa disponibles.

Este control de la disponibilidad es posible gracias a los atributos que ofrece la tabla *ocupacion_mesa,* que pueden ser observados en la *Ilustración 5*.

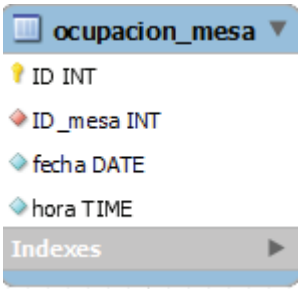

Ilustración 5: Entidad 'ocupacion_mesa' con sus atributos.

- **ID**: Corresponde al identificador de la ocupación de mesa, que es único para cada una y forma la clave primaria de la tabla *ocupacion_mesa.*
- **ID_mesa**: Corresponde al identificador de mesa, que es único para cada una y forma una clave foránea de la tabla *mesa.* Cada mesa que quede ocupada un determinado tiempo, será identificada con el *ID_mesa.*
- **fecha**: Corresponde al día concreto en que una mesa está ocupada.
- **hora**: Corresponde a la hora concreta en que una mesa está ocupada.

En conclusión, la entidad referente a ocupación de la mesa es capaz de ofrecer una tabla que filtre directamente sobre aquellas mesas ocupadas, proporcionando a los usuarios administradores un acceso instantáneo a aquellas mesas que no se pueden reservar para un tiempo concreto y un control exhaustivo de la disponibilidad de mesas, para que no existan solapamientos entre reservas de distintos usuarios básicos.

3.1.5 Modelo final y relaciones

Las entidades mencionadas anteriormente se relacionan entre sí garantizando una total integridad del sistema. Esto es posible gracias a los atributos que actúan como claves foráneas posteriormente en la base de datos.

En la *Ilustración 6* se muestra el modelo entidad-relación final que surge de las relaciones existentes entre las entidades *usuario, reserva, mesa y ocupacion_mesa.*

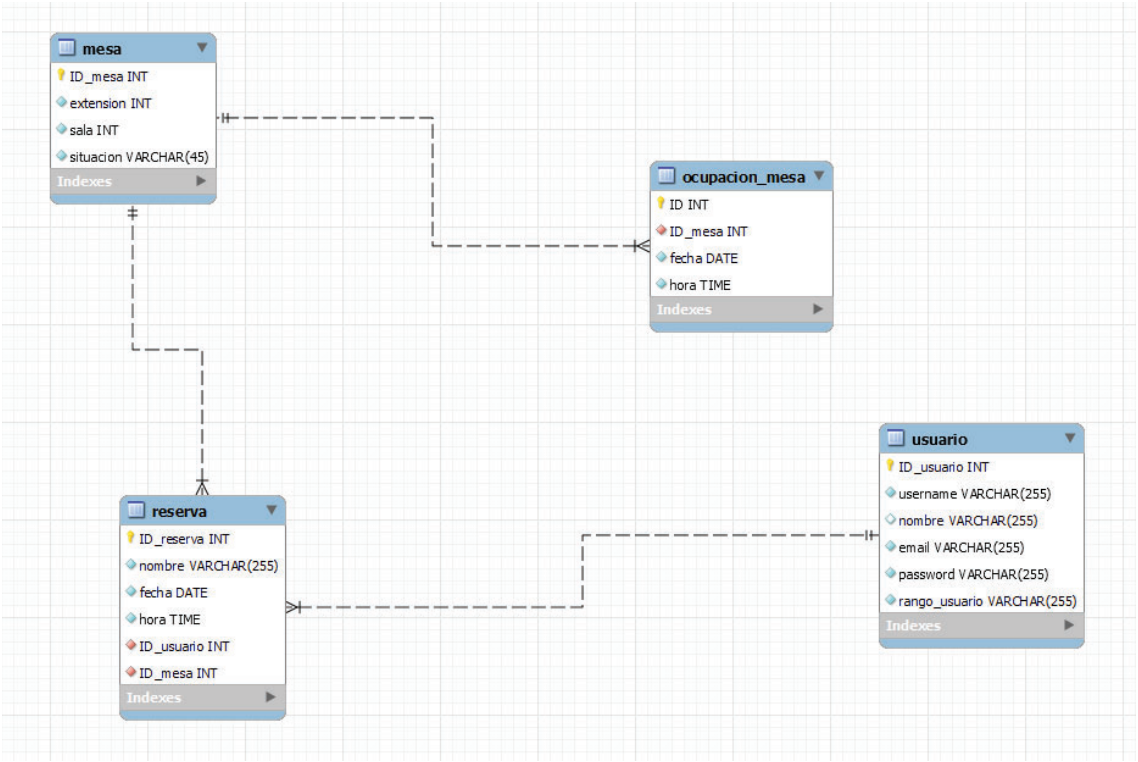

Ilustración 6: Modelo entidad-relación para la base de datos.

Básicamente, este modelo entidad-relación representa de forma sencilla el principal requisito de la aplicación: un usuario podrá realizar tantas reservas como desee de mesas de restaurante para fechas y horas concretas, dejando estas mesas ocupadas para el resto de usuarios.

El modelo final es la base del diseño de la aplicación, siendo el paso inmediatamente previo a la creación de las tablas en la base de datos de MySQL [6].

3.2 Diseño de la API REST

Una API REST es una interfaz de programación que proporciona un servicio que permite el lanzamiento de peticiones HTTP: GET, POST, PUT o DELETE. Estas peticiones devuelven un 'HTTP Status Code' que informa al cliente sobre el estado de la solicitud enviada, orientándole sobre cómo actuar ante dicha respuesta [7].

En el caso de la plataforma web para la gestión de reservas, sirve como mecanismo de conexión entre el cliente y la base de datos, de forma que éste lance peticiones que, a priori son invisibles para el usuario, consultando o modificando filas de la base de datos.

De esta manera, se cumple con la especificación de requisitos, pudiendo consultar reservas, añadir mesas, registrar usuarios y demás acciones. En definitiva, es la principal responsable del manejo de los datos de la aplicación.

La API desarrollada para la plataforma deberá estar apoyada en:

1. Una URI que sirva de identificador único para cada recurso. El *entry point* de la URI será por defecto el siguiente:

http://localhost:8080/MiRestaurante/api/restaurante

2. Un formato de datos para los recursos representado en JSON.
3. Un método GET, POST PUT o DELETE que intervenga en los recursos.
4. Un código de respuesta, o 'HTTP Status Code', que informe sobre el estado de la petición.

Los recursos definidos para nuestro sistema serán *usuario, mesa* y *reserva.*

A continuación, se describen las diferentes estructuras de cada petición que dan soporte a los requisitos del sistema, para cada recurso.

3.2.1 Recurso usuario

Los métodos mostrados a continuación operarán sobre el recurso usuario. En el campo URI se especifica el *end point* que completará el *entry point* mencionado anteriormente. Los campos método, descripción y código de respuesta describirán el método en cuestión.

- Registro de un usuario

URI	/registro	
Método	POST	
Descripción	Crea un usuario en el sistema introduciendo un formulario que incluya el username, nombre, email y password. Si el username o el email ya están en uso por otro usuario, devuelve un error. A los usuarios registrados se les da un rango de usuario básico.	
Código de respuesta	200: OK.	Registro de usuario correctamente.
	400: Bad Request.	Username o email ya en uso.

Tabla 1: POST - Registro de usuario.

- Inicio de sesión de un usuario

URI	/login	
Método	POST	
Descripción	Realiza la autenticación de un usuario, comprobando que exista en el sistema y que su username y password son correctas. En caso de no existir el usuario en el sistema o de introducir credenciales incorrectas, devuelve un error.	
Código de respuesta	200: OK.	Autenticación de usuario correcta.
	400: Bad Request.	Autenticación de usuario fallida.
	403: Forbidden.	Username o password incorrectos.
	404: Not found.	Usuario no encontrado.

Tabla 2: POST - Login de usuario.

- Consulta de todos los usuarios del sistema

URI	/usuarios	
Método	GET	
Descripción	Realiza una consulta de todos los usuarios existentes en el sistema. En caso de no poder realizar la consulta devuelve un error.	
Código de respuesta	200: OK.	Consulta realizada correctamente.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 3: GET - Consulta de usuarios del sistema.

- Consulta de los datos de un usuario

URI	/usuarios/{ID_usuario}	
Método	GET	
Descripción	Realiza una consulta sobre los datos de un usuario concreto, su identificador, username, nombre, email, password y rango de usuario. En caso de no encontrar al usuario o de no poder realizar la consulta, devuelve un error.	
Código de respuesta	200: OK.	Consulta realizada correctamente.
	404: Not found.	Usuario no encontrado.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 4: GET - Consulta de los datos de un usuario.

- Modificación de los datos de un usuario

URI	/usuarios/{ID_usuario}	
Método	PUT	
Descripción	Modifica los datos del usuario que no sean el identificador, el username o el email, que no se pueden modificar. En caso de no encontrar al usuario o de no poder realizar la modificación, devuelve un error.	
Código de respuesta	200: OK.	Datos modificados correctamente.
	404: Not found.	Usuario no encontrado.
	500: Internal Server Error.	Error realizando la modificación.

Tabla 5: PUT - Modificación de los datos de un usuario.

- Modificación de la contraseña de un usuario

URI	/usuarios/changepassword/{ID_usuario}

Método	PUT	
Descripción	Modifica la contraseña de un usuario introduciendo la contraseña antigua. En caso de no encontrar al usuario, que la contraseña antigua no coincida o de no poder realizar la modificación, devuelve un error.	
Código de respuesta	200: OK.	Password editada correctamente.
	404: Not found.	Usuario no encontrado.
	500: Internal Server Error.	Error realizando la modificación.

Tabla 6: PUT - Modificación de la contraseña de un usuario.

- Borrado del perfil de un usuario

URI	/usuarios/{ID_usuario}	
Método	DELETE	
Descripción	Elimina un perfil de usuario del sistema. En caso de no encontrar al usuario o de no poder eliminar el perfil, devuelve un error.	
Código de respuesta	204: No Content.	Usuario eliminado correctamente
	404: Not found.	Usuario no encontrado.
	500: Internal Server Error.	Error realizando el borrado.

Tabla 7: DELETE - Borrado del perfil de un usuario.

3.2.2 Recurso mesa

Los métodos mostrados a continuación operarán sobre el recurso *mesa*.

- Consulta de todas las mesas del sistema

URI	/mesas	
Método	GET	
Descripción	Realiza una consulta de todas aquellas mesas existentes en el sistema. En caso de no poder realizar la consulta devuelve un error.	
Código de respuesta	200: OK.	Consulta realizada correctamente.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 8: GET - Consulta de mesas del sistema.

- Consulta de los datos de una mesa

URI	/mesas/{ID_mesa}	
Método	GET	
Descripción	Realiza una consulta sobre los datos de una mesa concreta, su identificador, extensión, sala y situación. En caso de no encontrar la mesa o de no poder realizar la consulta, devuelve un error.	
Código de respuesta	200: OK.	Consulta realizada correctamente.
	404: Not found.	Mesa no encontrada.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 9: GET - Consulta de los datos de una mesa.

- Añadido de una mesa

URI	/mesas	
Método	POST	
Descripción	Añade una mesa en el sistema introduciendo un formulario que incluya la extensión y la sala. En caso de no poder añadir la mesa, devuelve un error.	
Código de respuesta	200: OK.	Añadido de mesa correcto.
	500: Internal Server Error.	Error añadiendo la mesa.

Tabla 10: POST - Añadido de una mesa.

- Modificación de los datos de una mesa

URI	/mesas/{ID_mesa}	
Método	PUT	
Descripción	Modifica los datos de la mesa. En caso de no encontrar la mesa o de no poder realizar la modificación, devuelve un error.	
Código de respuesta	200: OK.	Datos modificados correctamente.
	404: Not found.	Mesa no encontrada.
	500: Internal Server Error.	Error realizando la modificación.

Tabla 11: PUT - Modificación de los datos de una mesa.

- Borrado de los datos de una mesa

URI	/mesas/{ID_mesa}

Método	DELETE	
Descripción	Elimina una mesa del sistema. En caso de no encontrar la mesa o de no poder eliminarla, devuelve un error.	
Código de respuesta	204: No Content.	Mesa eliminada correctamente.
	404: Not found.	Mesa no encontrada.
	500: Internal Server Error.	Error realizando el borrado.

Tabla 12: DELETE - Borrado de los datos de una mesa.

- Habilitación de una mesa

URI	/mesas/{ID_mesa}/enable	
Método	PUT	
Descripción	Habilita una mesa, desde el punto de vista del administrador, para ser reservada. No debe confundirse con la disponibilidad de una mesa para un determinado tiempo. En caso de no encontrar la mesa o no poder habilitarla devuelve un error.	
Código de respuesta	204: No Content.	Mesa habilitada correctamente.
	404: Not found.	Mesa no encontrada.
	500: Internal Server Error.	Error habilitando la mesa.

Tabla 13: PUT - Habilitación de una mesa.

- Deshabilitación de una mesa

URI	/mesas/{ID_mesa}/disable	
Método	PUT	
Descripción	Deshabilita una mesa, desde el punto de vista del administrador, de modo que no puede ser reservada. En caso de no encontrar la mesa o no poder deshabilitarla devuelve un error.	
Código de respuesta	204: No Content.	Mesa deshabilitada.
	404: Not found.	Mesa no encontrada.
	500: Internal Server Error.	Error deshabilitando la mesa.

Tabla 14: PUT - Deshabilitación de una mesa.

3.2.3 Recurso reserva

Los métodos mostrados a continuación operarán sobre el recurso *reserva*.

- Consulta de todas las reservas del sistema

URI	/reservas	
Método	GET	
Descripción	Realiza una consulta de todas aquellas reservas existentes en el sistema. En caso de no poder realizar la consulta devuelve un error.	
Código de respuesta	200: OK.	Consulta realizada correctamente.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 15: GET - Consulta de todas las reservas del sistema.

- Consulta de los datos de una reserva

URI	/reservas/{ID_reserva}	
Método	GET	
Descripción	Realiza una consulta sobre los datos de una reserva concreta, su identificador, nombre, fecha, identificador del usuario que reserva e identificador de la mesa reservada. En caso de no encontrar la reserva o de no poder realizar la consulta, devuelve un error.	
Código de respuesta	200: OK.	Consulta realizada correctamente.
	404: Not found.	Reserva no encontrada.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 16: GET - Consulta de los datos de una reserva.

- Consulta de las reservas de un usuario

URI	/usuarios/{ID_usuario}/reservas	
Método	GET	
Descripción	Realiza una consulta de las reservas realizadas por un usuario. En caso de no encontrar el usuario o de no poder realizar la consulta, devuelve un error.	
Código de respuesta	200: OK.	Consulta realizada correctamente.
	404: Not found.	Usuario no encontrado.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 17: GET - Consulta de las reservas de un usuario.

- Consulta de las reservas para una mesa

URI	/mesas/{ID_mesa}/reservas
Método	GET
Descripción	Realiza una consulta de las reservas realizadas para una mesa. En caso de no encontrar la mesa o de no poder realizar la consulta, devuelve un error.

Código de respuesta	200: OK.	Consulta realizada correctamente.
	404: Not found.	Mesa no encontrada.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 18: GET - Consulta de las reservas para una mesa.

- Realización de una reserva

URI	/usuarios/{ID_usuario}/mesas/{ID_mesa}/reservas	
Método	POST	
Descripción	Realiza una reserva introduciendo un formulario que incluya la el nombre, la fecha y la hora de la reserva. En caso de que la mesa ya esté reservada para una determinada fecha y hora o de no poder realizar la reserva, devuelve un error. Además añade una mesa ocupada a la tabla *ocupacion_mesa.*	
Código de respuesta	200: OK.	Reserva realizada correctamente.
	409: Conflict.	Mesa ocupada para esa fecha.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 19: POST - Realización de una reserva.

- Modificación de una reserva

URI	/reservas/{ID_reserva}	
Método	PUT	
Descripción	Modifica los datos de la reserva. En caso de no encontrar la reserva, de intentar cambiar una reserva con una mesa ya reservada para una determinada fecha y hora o de no poder realizar la modificación, devuelve un error. También incorpora los cambios en la tabla *ocupacion_mesa.*	
Código de respuesta	200: OK.	Reserva editada correctamente.
	404: Not found.	Reserva no encontrada.
	409: Conflict.	Mesa ocupada para esa fecha.
	500: Internal Server Error.	Error realizando la consulta.

Tabla 20: PUT – Modificación de una reserva.

- Borrado de una reserva

URI	/reservas/{ID_reserva}	
Método	DELETE	
Descripción	Elimina una reserva del sistema. En caso de no encontrar la reserva o de no poder eliminarla, devuelve un error.	
	204: No Content.	Reserva eliminada correctamente.

Código de respuesta	404: Not found.	Reserva no encontrada.
	500: Internal Server Error.	Error realizando el borrado.

Tabla 21: DELETE – Borrado de una reserva.

3.3 Prototipo de la interfaz

Para la implementación de la interfaz web de usuario, es imprescindible la realización de un prototipo que represente el modelo a seguir, con el fin de plasmar el diseño y las acciones de usuario que posteriormente serán desarrollados en código.

Un ejemplo de este prototipo, es la pantalla de inicio, representada en la *Ilustración 7.* El resto de prototipo de pantallas puede encontrarse en el apartado *'Anexo 1 – Prototipo de la interfaz',* a través del cual se ha propuesto un modelo básico donde el usuario pueda elegir entre máximo 3 o 4 acciones por pantalla, con el objetivo de dar forma a un sistema usable y trivial.

Ilustración 7: Prototipo de pantalla inicio.

3.4 Flujograma

A continuación, en la *Ilustración 8*, se muestra el modelo que representa el flujograma o diagrama de flujo del sistema, que posteriormente acabará derivando en las pantallas de la interfaz de usuario.

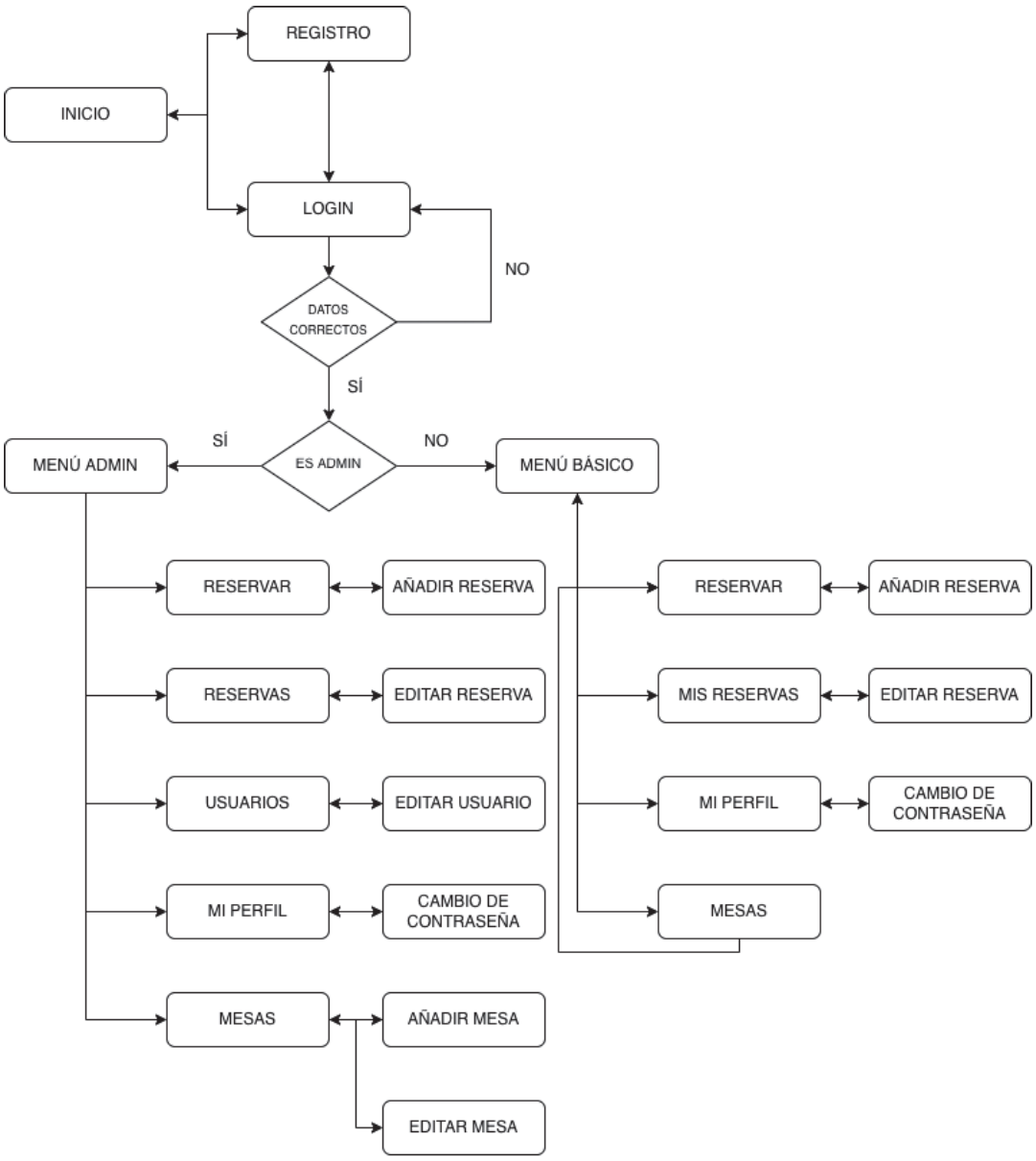

Ilustración 8: Flujograma del sistema.

4 Implementación de la aplicación

Siguiendo el diseño y modelos presentados en el apartado anterior, a continuación, se expone el desarrollo en código de la plataforma. Dentro de este desarrollo, se distingue entre dos piezas claves de código con finalidades y funciones diferentes: el frontend y el backend.

El **frontend** corresponde a la parte de desarrollo visible para el usuario final. Dentro del mismo, se garantiza una navegación segura, fluida y fiable a través de la interfaz. En nuestro caso concreto, se corresponde al desarrollo de la interfaz principal, tratando de adaptar el diseño de la misma al prototipo representado en el apartado anterior y dibujando las pantallas, a través de las cuales se pueda navegar, en función del diagrama de flujo mencionado previamente.

El **backend** corresponde a la parte de desarrollo que es invisible para el usuario final. Su finalidad es garantizar la integridad de la plataforma y cumplir con las acciones solicitadas por el usuario. Esto se realiza gracias a la implementación de la API REST, la cual actúa como servidor para cumplimentar las acciones solicitadas por el usuario; y gracias a la incorporación de una base de datos, que sirva como herramienta de almacenamiento y consulta de aquellas instancias manejadas por el usuario en la interfaz.

Para el correcto funcionamiento de la aplicación, debe existir una adecuada comunicación entre ambos, por lo que deben estar de alguna manera conectados entre sí. Además, se debe de tener en cuenta cada caso concreto aplicable a las acciones que pueda realizar el usuario y tratarse cada uno de los resultados posibles dentro del marco de actuación de este usuario.

A continuación, se exponen los procesos cumplidos para la implementación de la plataforma de gestión de reservas de mesa en restaurantes, donde se hace especial hincapié en las tecnologías utilizadas y los puntos clave que optimizan el funcionamiento del sistema.

4.1 Transcurso del desarrollo

Para poder entender el proceso, se explican cronológicamente los pasos seguidos que han dado lugar a la aplicación final. Estos pasos incluyen, para la parte del backend, la configuración de la base de datos y de la API REST; y para la parte del frontend, la implementación de la interfaz y las llamadas al backend.

4.1.1 Configuración de la base de datos

En primer lugar, se ha configurado la base de datos, cumpliendo con las entidades y relaciones representadas en el apartado *'3.1 Modelo Entidad-Relación'.*

Para completar esta configuración se ha establecido la conexión en *MySQL Workbench,* una herramienta visual de diseño de bases de datos que integra desarrollo de software y administración de bases de datos [8].

El lenguaje utilizado en esta herramienta es *MySQL*. Un ejemplo que muestra el uso de este lenguaje y que confirma la correcta configuración de la base de datos es la siguiente consulta:

*SELECT * FROM USUARIO;*

A través de ella, es posible consultar los usuarios almacenados en el sistema. Esto supone un punto clave, ya que mediante la ejecución de la sentencia *'INSERT'* será posible almacenar los usuarios en el momento de registro y consultarlos con la sentencia anterior para poder realizar un inicio de sesión. El resultado devuelto para la sentencia *'SELECT'* anterior se muestra en la *Ilustración 9* que, al no haber registrado aún ningún usuario, devuelve una tabla vacía.

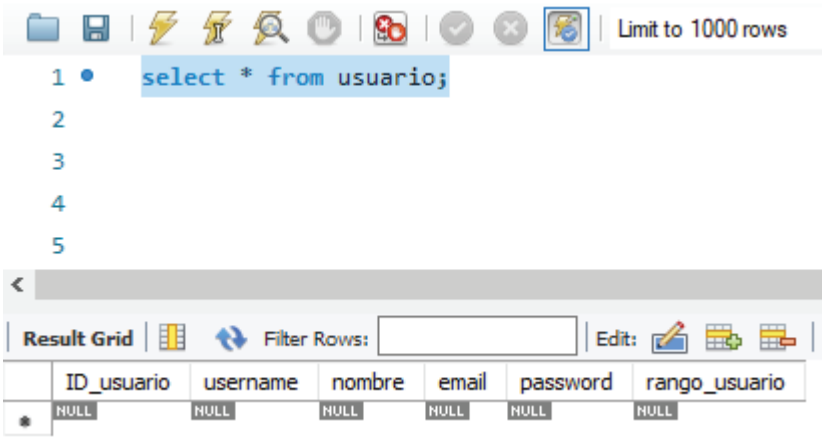

Ilustración 9: Resultado devuelto por sentencia 'SELECT'.

4.1.2 Desarrollo de la API REST

En segundo lugar, se ha desarrollado la API REST. Esta ha consistido en hacer efectivas las funcionalidades previstas de la especificación de requisitos funcionales. Para la implementación de esta parte del código, se ha tomado como referencia el apartado *'3.2 Diseño de la API REST'*.

La API REST se ha implementado en base al *'entry point'* y los *'end point'* establecidos en el apartado tomado como referencia. Se ha desarrollado en el lenguaje *Java*, utilizando *Eclipse* como entorno de desarrollo [9].

Para el correcto lanzamiento de la API, se ha utilizado el servidor *Apache Tomcat,* que es un servidor web de código abierto y contenedor de Servlet para código Java [10]. Al tratarse de un servicio web, han sido necesarias las librerías correspondientes de JAX-WS, que es una interfaz de programación de IBM de aplicaciones de Java para la creación de servicios web [11].

En la *Ilustración 10*, se muestra un ejemplo de la implementación del método correspondiente a *'Consulta de los datos de un usuario'*.

```java

    //Obtener información de un usuario

    @GET
    @Path("usuarios/{ID_usuario}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response obtenerUsuario(@PathParam("ID_usuario") String ID_usuario) {
        try {
            int int_id = Integer.parseInt(ID_usuario);
            String sql = "SELECT * FROM Usuario where ID_usuario = " + int_id + ";";
            PreparedStatement ps = conn.prepareStatement(sql);
            ResultSet rs = ps.executeQuery();
            if(rs.next()) { //existe usuario ?
                Usuario usuario = usuarioFromRS(rs); //guardamos en usuario los datos devueltos por rs
                return Response.status(Response.Status.OK).entity(usuario).build();
            }
            return Response.status(Response.Status.NOT_FOUND).entity("Usuario no encontrado \n").build();
        }
        catch(NumberFormatException e) {
            return Response.status(Response.Status.BAD_REQUEST).entity("No se pudo parsear entero \n").build();
        }
        catch(SQLException e) {
            return Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity("Error de acceso a la BBDD \n" + e.getMessage()).build();
        }
    }

```

Ilustración 10: Desarrollo del método que obtiene los datos de un usuario.

Lo primero que se observa del método es que corresponde a una petición GET de HTTP, seguida del *'end point'* donde se asigna como parámetro el *ID_usuario*, necesario para poder realizar la petición. Este método devuelve una respuesta en formato *JSON*, que corresponde a un formato de texto sencillo para el intercambio de datos [12].

La API establece una conexión con la base de datos configurada anteriormente y lanza la siguiente consulta a través de un *PreparedStatement*, dependiente del *ID_usuario* pasado como parámetro:

*"SELECT * FROM Usuario where ID_usuario = ? ";*

Si el usuario existe en la base de datos, se lanza un método auxiliar que crea un objeto *Usuario* a partir del *ResultSet* generado, y este es devuelto con el código *'200 OK'*.

Si el usuario no existe en la base de datos, se devuelve un texto plano *'Usuario no encontrado'*, con el código *'404 Not Found'*.

Si en la petición se pasa como parámetro un ID de usuario que no sea entero, se devuelve un texto plano *'No se pudo parsear a entero'*, con el código *'405 Bad Request'*.

Si ocurre cualquier error relacionado con la conexión a la base de datos, se devuelve un texto plano *'Error de acceso a la BBDD'*, con el código *'500 Internal Server Error'*.

Todas estas posibilidades cumplen con lo establecido en la *'Tabla 4: GET – Consulta de los datos de un usuario'*. Los demás métodos de la API se han desarrollado siguiendo la misma estructura empleada en este ejemplo propuesto, posteriormente se comprueba su funcionamiento en el apartado *'5 Casos de prueba'*.

4.1.2.1 Uso de la herramienta Postman

Con el fin de comprobar el funcionamiento de la API REST para los diferentes casos posibles en el ámbito de las peticiones HTTP que el usuario puede realizar, se ha utilizado la herramienta *Postman*.

Postman es una plataforma de API para que el desarrollador diseñe, construya, pruebe e itere sus API [13]. Aunque se analizará más a fondo en el apartado correspondiente a los casos de prueba, a continuación, se muestra un ejemplo de funcionamiento de esta plataforma para el método analizado anteriormente, en la *Ilustración 11*.

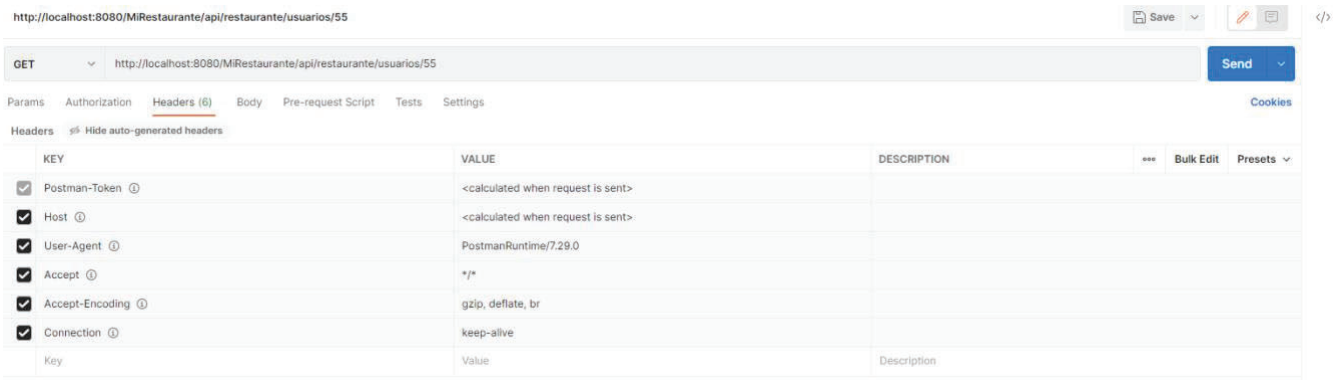

Ilustración 11: Ejemplo de uso de la herramienta Postman.

En el ejemplo, se puede observar como a través de la herramienta *Postman*, se lanza una petición GET de consulta de los datos de un usuario, aportando la URI completa que utiliza este método.

En la URI, se pasa como parámetro el *ID_Usuario,* que para este caso se pretende consultar la información sobre el usuario con identificador 55. Además, se incluyen diversas *Headers* o cabeceras que son indispensables para el manejo de la petición.

4.1.3 Implementación de la interfaz web

En tercer lugar, y ya habiendo completado y comprobado el funcionamiento de la parte backend de la plataforma, ha tomado parte el desarrollo de la interfaz en diferentes lenguajes frontend. Para la implementación de esta parte del código, se han tomado como referencia los apartados *'3.3 Prototipo de la interfaz'* y *'3.4 Flujograma'*.

Los diferentes lenguajes utilizados para este desarrollo han sido:

- **HTML**: es el código que se utiliza para estructurar y desplegar una página web y sus contenidos [14]. En la plataforma, ha ido directamente relacionado con la estructura de la interfaz web, dejando el diseño y la dinámica de la misma para los demás lenguajes.
- **CSS**: en español 'Hojas de estilo en cascada', es un lenguaje de diseño gráfico para definir y crear la presentación de un documento previamente estructurado, por ejemplo, en HTML [15].
- **JavaScript**: es un lenguaje de programación interpretado, orientado a objetos, basado en prototipos, imperativo y dinámico [16]. En nuestra

aplicación, sirve para añadir características interactivas, como la adaptación del HTML y su estructura en función de la petición realizada.

Estos lenguajes han sido desarrollados en el editor de texto *Sublime Text,* que se puede emplear como editor de código fuente y que se adapta en función del lenguaje específico de desarrollo facilitando la implementación [17].

A continuación, se expone un ejemplo de cómo se ha desarrollado la pantalla que sirve para que los usuarios añadan reservas. Esta pantalla se encuentra en el apartado *'Anexos – Interfaz web para usuarios'*, en la ilustración *'Pantalla de realización de reservas'.*

En la *Ilustración 12*, se muestra el código HTML empleado para estructurar dicha pantalla.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" type="text/css" href="../CSS/crear_reserva.css">
    <title>Mi Restaurante</title>
</head>
<body>
    <nav>
        <a id="atras"><img src="../IMG/back.png">Atrás</a>
        <a id="menu"><img src="../IMG/menu.png">Menú principal</a>
    </nav>
    <div>
        <h4>REALIZAR UNA RESERVA</h4>
        <br>
        <form id="formulario">
            <p>Nombre de la reserva: </p>
            <input class="input"type="text" name="nombre">
            <br>
            <br>
            <p>Mesa: </p>
            <input class="input"type="number" name="mesa">
            <br>
            <br>
            <p>Día: </p>
            <input class="input"type="date" name="dia">
            <br>
            <br>
            <p>Hora: </p>
            <input class="input"type="time" name="hora">
            <br>
            <br>
            <br>
            <button id="boton1" type="submit" class="boton">Confirmar reserva</button>
        </form>
    </div>
</body>
    <script type="text/javascript" src="../JS/crear_reserva.js"></script>
</html>

```

Ilustración 12: Código HTML de la pantalla de realización de reservas.

El código estructura al comienzo una barra de navegación con dos botones que sirven para volver a atrás y volver al menú principal. Posteriormente, establece un contenedor principal donde se encuentra el título de la pantalla, el formulario específico de realización de una reserva y el botón para confirmar dicha reserva.

Este código HTML coge de referencia el estilo proporcionado por la clase *crear_reserva.css*, que se expondrá a continuación, y el script de la clase *crear_reserva.js*, que establece las diferentes interacciones dentro de la pantalla.

En la *Ilustración 13*, se muestra una parte del código CSS empleado para dar estilo a la pantalla.

```css
*{
  font-family: 'Lobster', cursive;
  font-weight: 100;
  font-size: 14,5px;
  background-image: url(../IMG/restaurante-etxaniz-015176.jpg);
  background-size: cover;
}


nav {
  overflow: hidden;
  background: #CD853F;
  border: 2px solid black;
  border-color: black;
  position: fixed;
  top: 0;
  left: 0;
  width: 6000px;
}

nav a:hover { /* Green */
  color: black;
}

h4 {
  display: block;
  background: #E9967A;
  text-align: center;
  margin-top:  0px;
}
p{
  display: block;
  background: #E9967A;
  text-align: left;
  margin-top:  0px;
  margin-bottom: 0px;
  font-size: 12,5px;

}
.input{
  background: #E9967A;
  width: 300px;
  border-bottom:1px solid lightgrey;
  color:  lightgrey;
  text-align: left;
  border:none;

}
```

Ilustración 13: Código CSS de la pantalla de realización de reservas.

Este código es capaz de determinar el formato de la fuente del texto, su tamaño, el fondo y su color, o los márgenes existentes entre las estructuras HTML. La parte correspondiente a JavaScript se muestra en el siguiente apartado, aprovechando que este actúa de forma directa para la comunicación con el backend.

4.1.4 Comunicación entre el frontend y el backend

Una vez desarrolladas las partes independientes de frontend y backend, se ha llevado a cabo la comunicación entre ambas. Antes de analizar el mecanismo utilizado para dicha comunicación, es conveniente entender como interactúan las diferentes partes de la plataforma entre ellas.

La representación del sistema en conjunto se basa en un modelo donde el cliente se apoya en un servidor para hacer peticiones, a lo que este último contesta proporcionándole una respuesta que el cliente recibe. En la *Ilustración 14*, se muestra dicha representación.

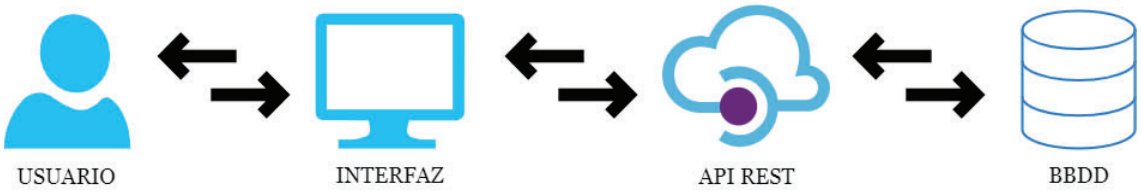

Ilustración 14: Representación de interacciones en el sistema.

Como se puede observar, la interfaz es la encargada de comunicarse con la API REST, que ya forma parte del backend. Para esta comunicación, ha sido preciso utilizar un mecanismo de llamadas *fetch en JavaScript*, que realiza peticiones HTTP para obtener recursos de forma asíncrona por la red, sirviendo de alternativa a *XMLHttpRequest,* otro tipo de funcionalidad obsoleta utilizada antiguamente [18].

Además, al tratarse de una API privada, ha sido necesaria una extensión de Google Chrome, que permitiese el Intercambio de Recursos de Origen Cruzado (*CORS*, en inglés) para este tipo de llamadas *fetch* [19]. Esta extensión se habilita según se muestra en la *Ilustración 15.*

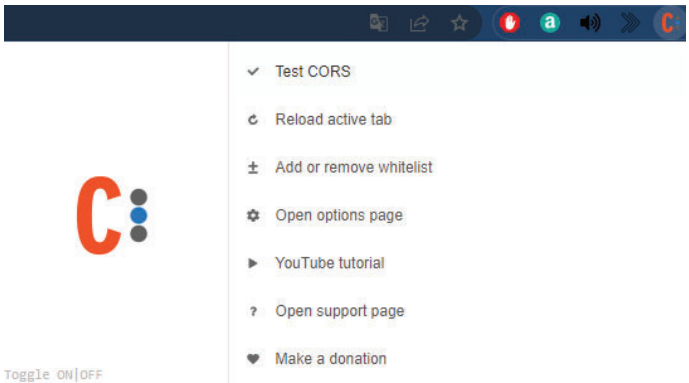

Ilustración 15: Extensión de Google para habilitar CORS.

Una vez
habilitada la extensión, era posible realizar llamadas *fetch* sobre la API REST privada. En la Ilustración 16, se muestra un ejemplo del código empleado en JavaScript para realizar la llamada *fetch* correspondiente a la realización de una reserva, vista previamente.

```
    const url1 = 'http://localhost:8080/MiRestaurante/api/restaurante/usuarios'
    var datos = new FormData(formulario)
    var date = formatearFecha(datos.get('dia'))
    var datos2 = {nombre: datos.get('nombre'), fecha: date, hora: datos.get('hora')};
    id_usuario = data.ID_usuario
    id_mesa = datos.get('mesa')
    fetch(`${url1}/${id_usuario}/mesas/${id_mesa}/reservas`, {
      method: 'POST',
      body: JSON.stringify(datos2),
      headers:{
        'Content-Type': 'application/json'
        }
    })
    .then(res => {
    if(res.status == 201){
      window.alert("Reserva realizada con éxito!")
    }
    else if (res.status == 409){
      window.alert("Lo sentimos, ya existe una reserva para esa mesa en la fecha y hora indicadas")
    }
    else if(res.status == 404){
      window.alert("Error realizando la reserva: la mesa indicada no existe")
    }
    else{
      window.alert("Error realizando la reserva")
    }
    })
    .then(data => console.log(data))
    .catch(err => console.log(err))
  })
```

Ilustración 16: Código JavaScript para llamada fetch a API.

5 Casos de prueba

El desarrollo del código de la aplicación y configuración de los diferentes agentes que interactúan sobre ella, ha supuesto la parte preliminar sobre la que está apoyada la plataforma y que comprende los diferentes casos a los que está expuesto el usuario final al realizar sus acciones.

Sin embargo, con el fin de garantizar la integridad total de la plataforma, es necesario realizar los diferentes **casos de prueba** que comprueban si realmente los resultados son los esperados al realizar diversas acciones.

Estos casos de prueba aseguran al usuario final el correcto funcionamiento del sistema, tanto para la realización de solicitudes que resultan exitosas, como para aquellas que terminan notificando al usuario algún impedimento que ha imposibilitado realizar su acción.

En este apartado, se distingue la comprobación del funcionamiento por separado, para la parte backend y para la parte frontend. Posteriormente a esta comprobación, se verifica si la comunicación entre ambas se realiza correctamente y si funcionan simultáneamente en el sistema.

5.1 Backend

En primer lugar, y siguiendo con el orden cronológico de implementación visto en el apartado *'4.1 Transcurso del desarrollo'*, se han realizado las pruebas pertinentes en la parte backend del proyecto, la cual engloba a la API REST y a la base de datos.

Como se mencionó previamente, para la realización de estas pruebas se ha utilizado la herramienta *Postman*, que facilita la llamada a peticiones HTTP sin necesidad de un cliente similar a la interfaz final del usuario.

Primeramente, es necesario lanzar el servidor *Apache Tomcat* en *Eclipse* para que estas llamadas surtan efecto en *Postman*. Para el supuesto concreto que se expondrá, en la *Ilustración 17*, se muestran una serie de mesas almacenadas en la base de datos con diferentes características.

	ID_mesa	extension	sala	situacion
▶	9	2	1	no disponible
	10	3	2	disponible
	11	1	3	disponible
	12	6	4	no disponible
	13	5	1	disponible
	14	8	1	disponible
	15	3	1	no disponible
	16	1	2	disponible
*	NULL	NULL	NULL	NULL

Ilustración 17: Supuesto de mesas almacenadas en base de datos.

La petición que queremos comprobar en Postman es la que consulta todas las mesas del sistema. Para ello, se prepara una petición *GET*, incluyendo la URI que trata esa consulta. En la *Ilustración 18*, se comprueba que, efectivamente, se devuelve el código *'200 OK'* y la URI de cada mesa almacenada en base de datos, con su ID de mesa concreto.

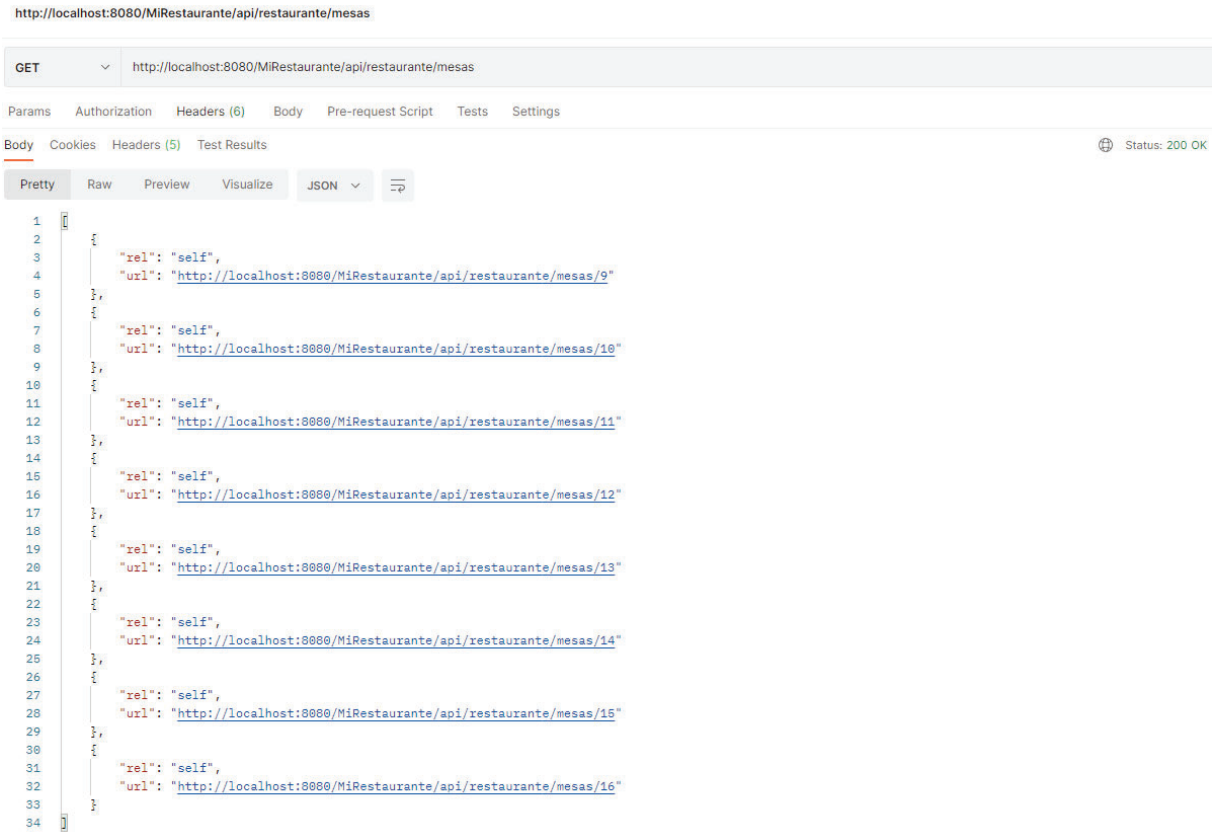

Ilustración 18: Comprobación de consulta de mesas en Postman.

Para consultas de este tipo, donde se espera más de una instancia, se ha considerado conveniente obtener la URI de cada una, que ha hecho accesible la consulta de cada mesa específica en este caso.

Por ejemplo, accediendo a la primera URI, se abre una nueva pestaña en *Postman* para realizar la petición *GET* de la mesa con identificador 9. En la *Ilustración 19*, se muestra dicha solicitud y su resultado, con código *'200 OK'*.

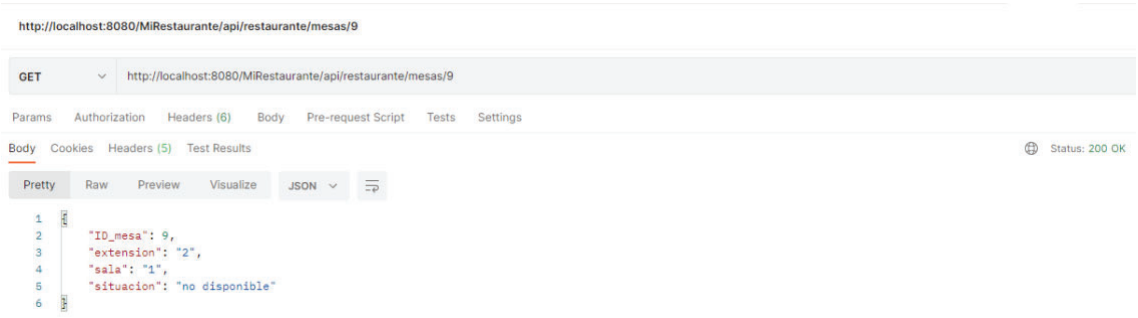

Ilustración 19: Comprobación de consulta de la mesa 9 en Postman.

Los datos mostrados de la mesa 9, que incluyen la extensión, sala y situación; coinciden con los consultados en base de datos en la *Ilustración 17.*

Sin embargo, con el fin de comprobar un caso erróneo, si en la URI presentada en lugar del identificador 9 se introduce el identificador 8, se devuelve que la mesa no ha sido encontrada en el sistema. Esto comprueba la integridad de la base de datos, que nos mostraba desde la mesa 9 hasta la mesa 16, no existiendo la mesa 8. La *Ilustración 20* muestra este resultado, con el código *'404 Not Found'*.

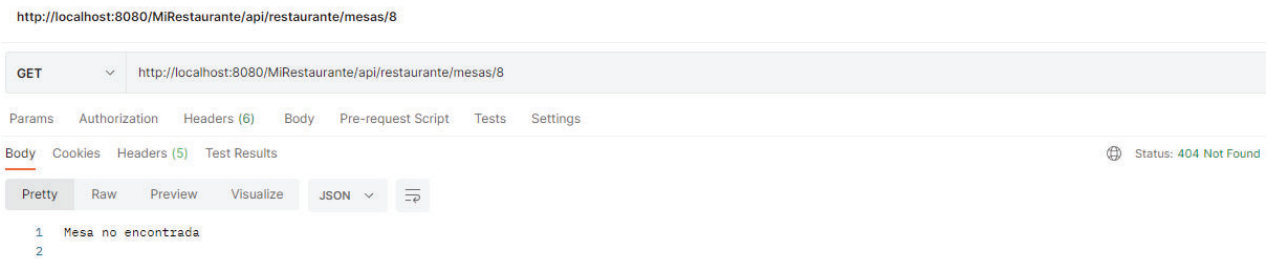

Ilustración 20: Comprobación de consulta de mesa no existente.

Al igual que para el método de consulta de mesas, se realizaron pruebas para todos los casos de todos los métodos dispuestos en el sistema. Para el caso concreto de las solicitudes *POST* y *PUT*, era necesario incluir en la pestaña *Body,* una estructura JSON que sirviese como entrada de los parámetros que se querían añadir o modificar. Por ejemplo, la *Ilustración 21* muestra el *Body* incluido para la solicitud *POST* de registro de un usuario.

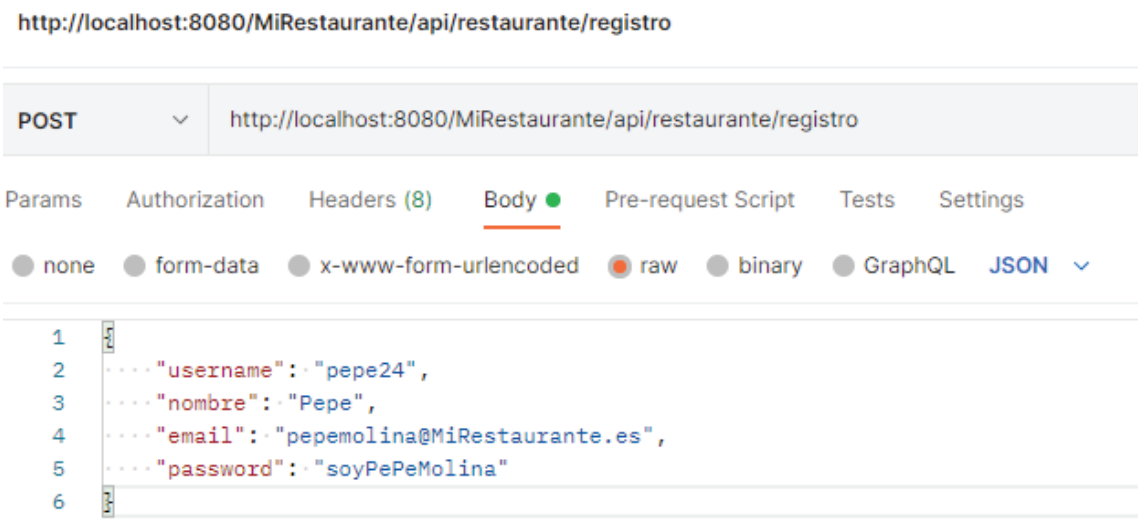

Ilustración 21: Body para el método POST de registro de usuario.

5.2 Frontend

En segundo lugar, se han realizado las pruebas pertinentes en la parte frontend del proyecto, asociada a la interfaz de usuario. En vistas a que esta interfaz ha sido lanzada directamente en el navegador de *Google Chrome*, que disponía de la extensión que habilitaba *CORS* vista en la *Ilustración 15*, se ha utilizado la consola para comprobar el funcionamiento del frontend.

En concreto, gracias al método *console.log* de *JavaScript*, era posible imprimir por consola cualquier tipo de contenido que debiera ser depurado. De esta manera, al producirse un evento tal como 'clickar' en un botón, los resultados eran mostrados por consola.

Aprovechando lo visto en el apartado anterior, relacionado con el *Body* que debían presentar las peticiones *POST* y *PUT*, se ha utilizado la consola para imprimir el contenido de los *inputs* a introducir al iniciar sesión. En la *Ilustración 22*, se muestra el formulario introducido iniciando sesión.

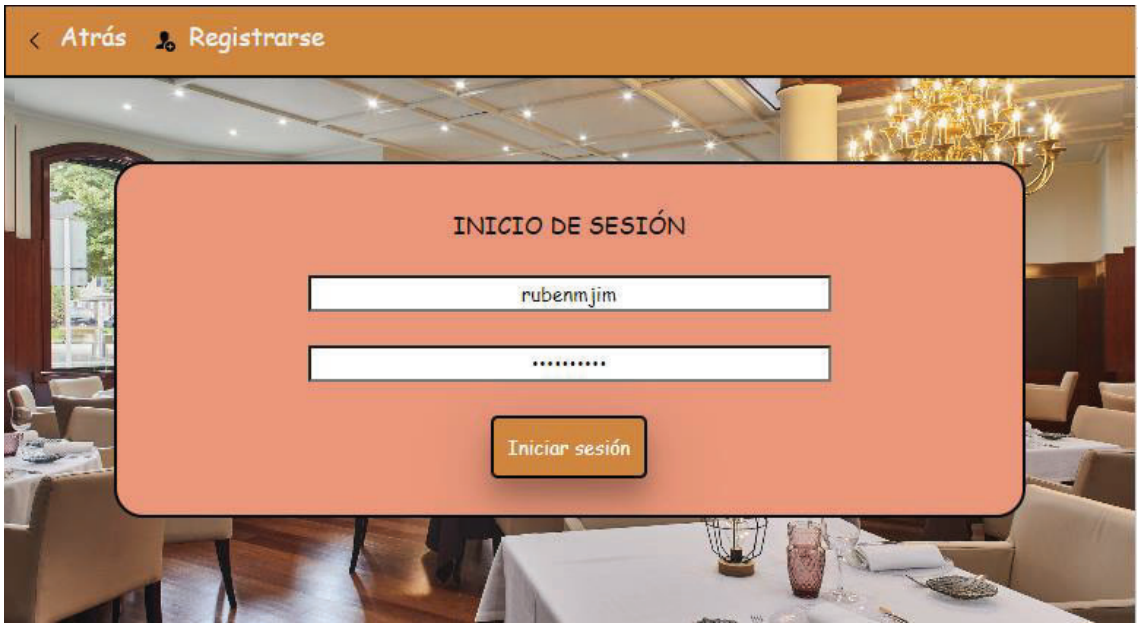

Ilustración 22: Formulario introducido al iniciar sesión.

En la *Ilustración 23*, se muestra el contenido impreso por consola para el formulario introducido iniciando sesión.

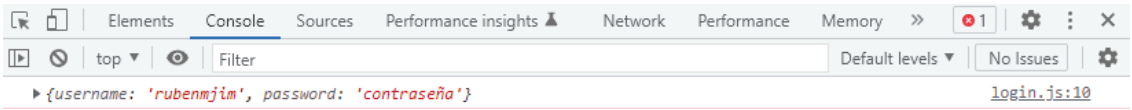

Ilustración 23: Comprobación del contenido impreso por consola.

Efectivamente, se recogen los datos introducidos para el formulario. Esta herramienta ha sido utilizada para imprimir el contenido de los cuerpos, para observar la construcción de las celdas de una tabla en consultas con muchas instancias o para recoger mensajes de error cuando la acción lo requería.

5.3 Funcionamiento simultáneo

Una vez se ha comprobado que el backend y el frontend funcionan de manera independiente, es momento de realizar la comprobación de ambos en el conjunto de la aplicación. O dicho de otra manera, este apartado muestra de qué manera se han realizado las pruebas para las llamadas a la API REST por parte de la interfaz web.

Como ya se comentó en el apartado *'4.1.4 Comunicación entre el frontend y el backend',* esta comunicación de las partes se realiza a través de llamadas *fetch.* Pues bien, esta metodología es capaz de recoger los códigos de estado devueltos en las peticiones HTTP. Esto se realiza a través del método *.status,* para comprobar si una acción en su totalidad se ha realizado de forma correcta por el usuario o no.

Por tanto, cuando la llamada *fetch* correspondiente recupera un código de error, se muestra en la interfaz un *pop-up* o ventana emergente que avisa al usuario que su acción no se ha podido completar.

Para el caso concreto de realizar una reserva, existen tres posibilidades. Que la reserva se realice con éxito, que no se pueda reservar una mesa en una fecha y hora porque ya esté ocupada o que la mesa que se quiera reservar no exista.

En la *Ilustración 24*, se muestra un ejemplo de la ventana emergente en la interfaz web al realizar una reserva correctamente, tras haber llamado a la API para que almacene dicha reserva en la base de datos.

Ilustración 24: Comprobación de reserva realizada correctamente con fetch.

En la *Ilustración 25*, se muestra un ejemplo de la ventana emergente en la interfaz web al realizar una reserva de una mesa que ya está ocupada a una determinada fecha y hora del calendario.

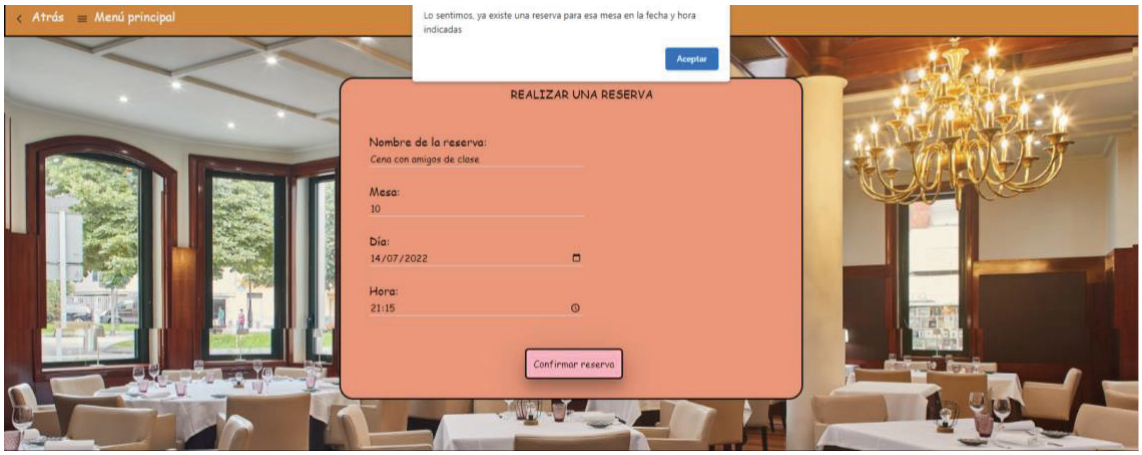

Ilustración 25: Comprobación de caso erróneo 1 para reserva de mesa.

Este mensaje ha resultado tras haber realizado la reserva anterior *'Cena con Irene'* y haber querido realizar otra reserva de la misma mesa el mismo día y a la misma hora.

En la *Ilustración 26*, se muestra un ejemplo de la ventana emergente en la interfaz web al realizar una reserva de una mesa que no existe en el sistema.

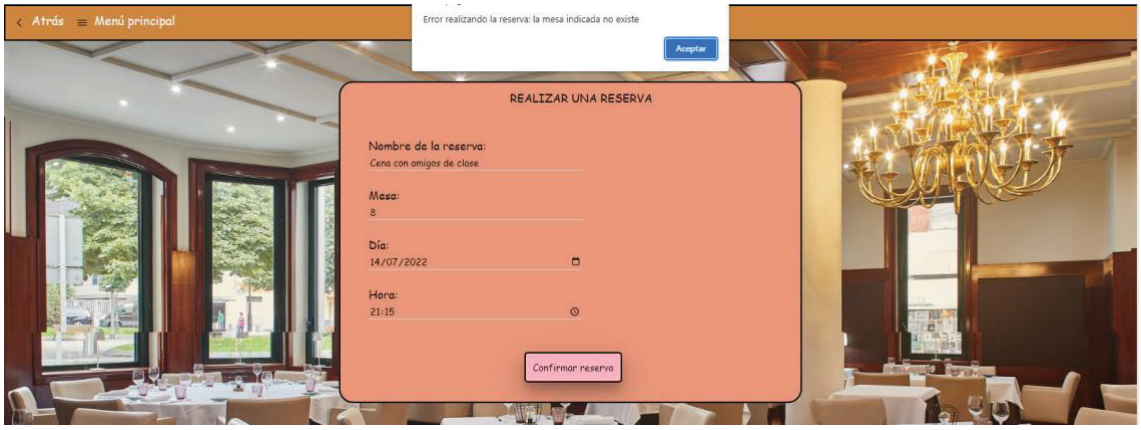

Ilustración 26: Comprobación de caso erróneo 2 para reserva de mesa.

Efectivamente, como ya se comprobó en el apartado de *'5.1 Backend'*, la mesa con identificador 8 no existe y por tanto, no se puede realizar la reserva.

En la base de datos, que corresponde a la parte última del backend, ha quedado registrado solamente el primer intento de reserva *'Cena con Irene'*. Se puede comprobar en la *Ilustración 27*.

	ID_reserva	nombre	fecha	hora	ID_usuario	ID_mesa
▶	75	Cena con Irene	2022-07-14	21:15:00	65	10
*	NULL	NULL	NULL	NULL	NULL	NULL

Ilustración 27: Única reserva registrada en base de datos.

6 Resultados y conclusiones

El resultado final acaba siendo una plataforma web interactiva en la que los usuarios pueden realizar reservas de mesas de restaurante con total seguridad e integridad. Esta plataforma ha sido construida en base a diferentes tecnologías y lenguajes de programación que aportan cada pieza fundamental de la misma para lograr los resultados esperados.

A modo de resumen y de forma cronológica, se ha procedido con el siguiente progreso, donde se ha llevado a cabo la toma de decisiones acerca de diferentes puntos:

- La especificación de los requisitos del sistema, que forman la base del diseño de las partes de la aplicación.
- El diseño de los diferentes puntos influyentes en la plataforma, donde el Flujograma y el Prototipo de Interfaz han acabado dando lugar a la Interfaz Web Final, mientras que el Modelo Entidad-Relación y Diseño de la API REST han resultado en la Implementación Backend del proyecto.
- La implementación de la parte Backend y Frontend, así como la configuración de la Base de Datos y la comunicación entre todos los agentes intervinientes en el sistema.
- La puesta en marcha de casos de prueba que verificaban si la totalidad de las acciones realizadas por los usuarios producían los resultados que se debían adecuar al modelo diseñado.

A modo de conclusión, la plataforma final cumple con los requisitos y diseños establecidos para una agradable experiencia de usuario. Tanto los usuarios básicos como los administradores, son capaces de acceder a la interfaz y realizar su gestión de reservas de restaurante con la integridad suficiente y dejando atrás la obsoleta forma de realizar estas reservas mediante llamada telefónica o acudiendo al local de forma personal.

Además, los usuarios administradores cuentan con funcionalidades extra, donde pueden gestionar la disponibilidad de las mesas, añadirlas o eliminarlas y gestionar a los propios usuarios básicos que se encuentren dados de alta en el sistema.

Por lo tanto, finalmente se cumple con los objetivos establecidos al comienzo del proyecto, tanto prioritarios como secundarios, dispuestos en el apartado *'1.1 Lista de objetivos del proyecto'*.

Si bien es cierto, la planificación inicial, desarrollada en el apartado *'1.2 Planificación'*, ha sufrido ciertas demoras de tiempo debido a la gran complejidad que supone, no solo diseñar e implementar una plataforma de estas características, si no también poder asegurar integridad y seguridad a los usuarios que den uso de ella.

Desde el punto de vista personal, he conseguido consolidar diversos conocimientos aprendidos durante el transcurso del Doble Grado, además de poder ampliar muchos otros gracias a diferentes inconvenientes surgidos durante el desarrollo del proyecto.

En concreto, he sido capaz de aprender desde el principio lenguajes como HTML, JavaScript o CSS. Aprender sobre estas tecnologías no solo consiste en saber

programar el código correspondiente, sino que también requiere de una gran soltura y conocimiento acerca de la manera de depurar una interfaz web.

En consecuencia, la realización de este proyecto ha resultado en una grata experiencia donde ha sido posible aplicar dichos conocimientos para la obtención de unos resultados exitosos.

7 Análisis de Impacto

193 países se comprometieron en el año 2015 a cumplimentar una serie de Objetivos de Desarrollo Sostenible propuestos por las Naciones Unidas con el fin de cumplirse para la Agenda 2030 [20].

Estos Objetivos de Desarrollo Sostenible, ODS de ahora en adelante, velan por la salud y bienestar de las personas, el fin de la pobreza, la reducción de las desigualdades o la acción por el clima, entre otros.

En nuestro caso concreto, la plataforma web para la gestión de reservas en restaurantes, vela principalmente por los siguientes:

- **ODS 9 – Industria, innovación e infraestructura**: consiste en construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible y fomentar la innovación [21]. En este caso concreto, se está velando mayormente por fomentar la innovación.

- **ODS 12 - Producción y consumo responsable**: consiste en generar más, con menos recursos. Además, promueve estilos de vida sostenibles y pretende aumentar la eficiencia de recursos, evitando efectos destructivos sobre el planeta [22]. Aunque pueda parecer que está menos relacionado, la transformación digital que supone nuestro proyecto es capaz de aumentar la eficiencia en restaurantes ahorrando las demoras de tiempo que suponían los métodos de reserva obsoletos.

Gracias a la aportación que supone velar por ambos objetivos, el desarrollo de nuestra plataforma cumple con lo dispuesto para la Agenda 2030 y se compromete directamente para seguir manteniendo este compromiso con los ODS dispuestos por la Naciones Unidas en el año 2015.

8 Bibliografía

[1] PowerData, G. (2020). Transformación digital. Qué es y su importancia y relación con los datos. PowerData. https://www.powerdata.es/transformacion-digital

[2] Bookitit. (2022, 27 enero). Sistema de reservas online, cita previa online y agendas online. https://www.bookitit.com/es/

[3] SuperSaaS Reservas Online. (2022, marzo). SuperSaaS. https://www.supersaas.es/

[4] Software de programación de reuniones en línea gratuito - Calendly. (2013). Calendly.com. https://calendly.com/es/

[5] ¿Qué es una base de datos relacional? (2020). Oracle. https://www.oracle.com/es/database/what-is-a-relational-database/

[6] MySQL. (2022). MySQL. https://www.mysql.com/

[7] ¿Qué es una API de REST? (2022). RedHat. https://www.redhat.com/es/topics/api/what-is-a-rest-api

[8] MySQL: MySQL Workbench. (2022). MySQL. https://www.mysql.com/products/workbench/

[9] Eclipse Foundation, Inc. (2022). The Community for Open Innovation and Collaboration | The Eclipse Foundation. Eclipse IDE. https://www.eclipse.org/

[10] Project, A. T. (2022). Apache Tomcat® - Welcome! Apache Tomcat. https://tomcat.apache.org/

[11] JAX-WS. (2022). JAX-WS. https://www.ibm.com/docs/es/was/9.0.5?topic=services-jax-ws

[12] JSON. https://www.json.org/json-es.html

[13] Postman. https://www.postman.com/

[14] HTML - Aprende sobre desarrollo web. (2020, 12 diciembre). Mozilla - HTML. https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/HTML_basics

[15] CSS | MDN. (2021, 7 julio). Mozilla - CSS. https://developer.mozilla.org/es/docs/Web/CSS

[16] JavaScript | MDN. (2022, 30 mayo). Mozilla - JavaScript. https://developer.mozilla.org/es/docs/Web/JavaScript

[17] Sublime Text - the sophisticated text editor for code, markup and prose. (2022). Sublime Text. https://www.sublimetext.com/

[18] Uso de Fetch - Referencia de la API Web | MDN. (2022, 23 mayo). Mozilla - Fetch. https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Using_Fetch

[19] Control de acceso HTTP (CORS) - HTTP | MDN. (2022, 29 abril). Mozilla - CORS. https://developer.mozilla.org/es/docs/Web/HTTP/CORS

[20] Ministerio de Derechos Sociales y Agenda 2030 - ODS. (2022). Agenda 2030. https://www.mdsocialesa2030.gob.es/agenda2030/index.htm

[21] INDUSTRIA, INNOVACIÓN E INFRAESTRUCTURA. (2015, septiembre). https://www.un.org/sustainabledevelopment/es/wp-content/uploads/sites/3/2016/10/9_Spanish_Why_it_Matters.pdf

[22] Moran, M. (2020, 17 junio). Consumo y producción sostenibles. Desarrollo Sostenible. https://www.un.org/sustainabledevelopment/es/sustainable-consumption-production/

9 Anexos

9.1 Anexo 1 – Prototipo de la interfaz

9.1.1 Pantallas de inicio

Ilustración 28: Prototipo de pantalla de registro.

Ilustración 29: Prototipo de pantalla de inicio de sesión.

9.1.2 Vista de usuario básico

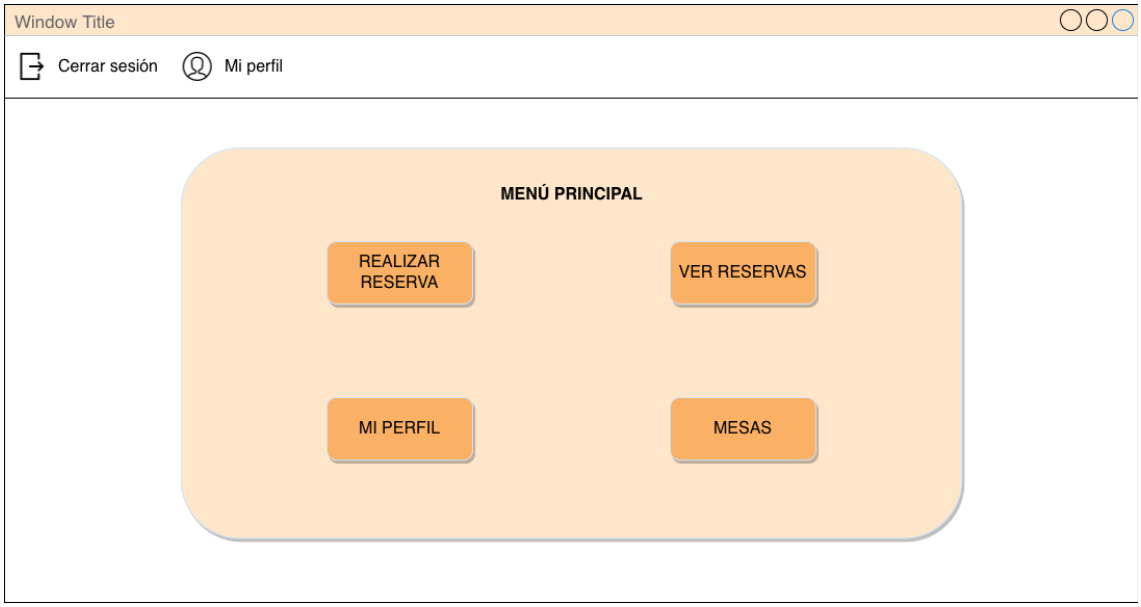

Ilustración 30: Prototipo de pantalla de menú para usuario básico.

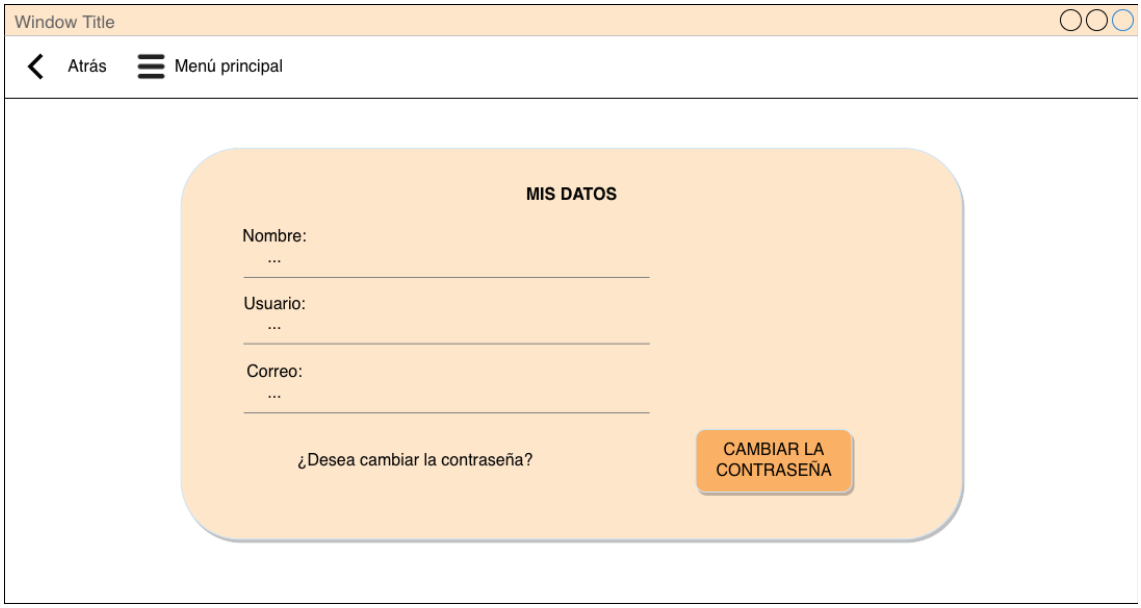

Ilustración 31: Prototipo de pantalla de perfil de usuario.

Ilustración 32: Prototipo de pantalla de cambio de contraseña.

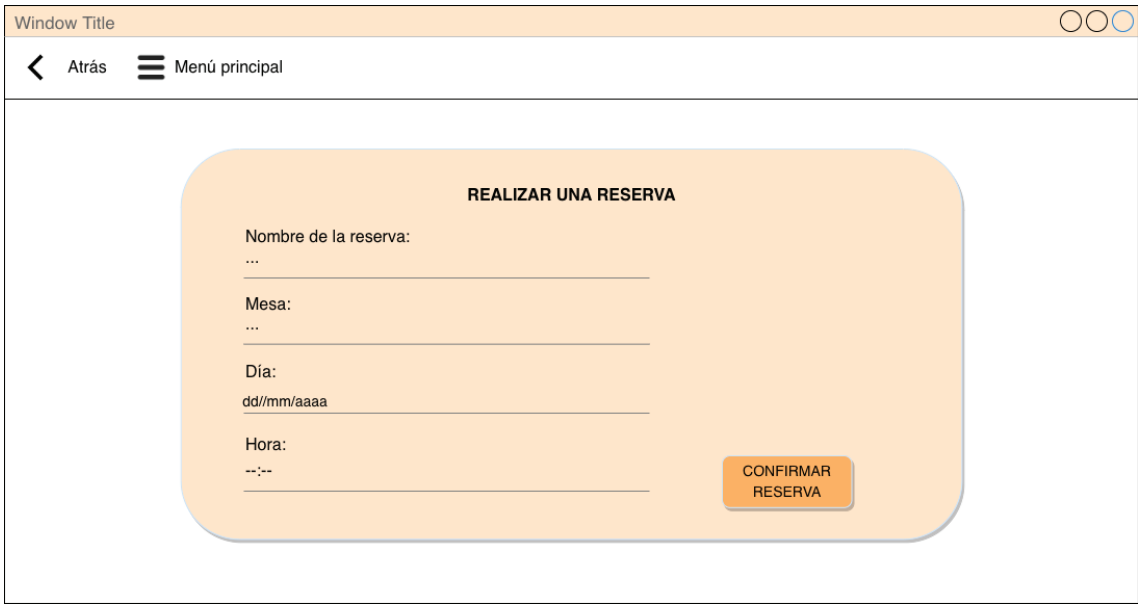

Ilustración 33: Prototipo de pantalla de añadir reserva.

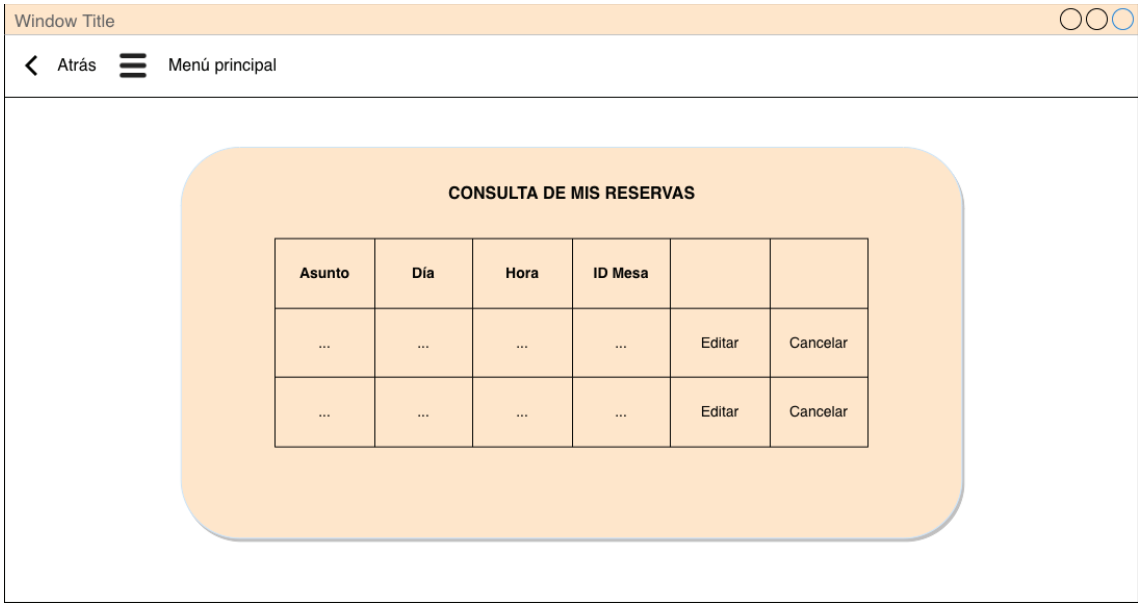

Ilustración 34: Prototipo de pantallas de reservas de usuario básico.

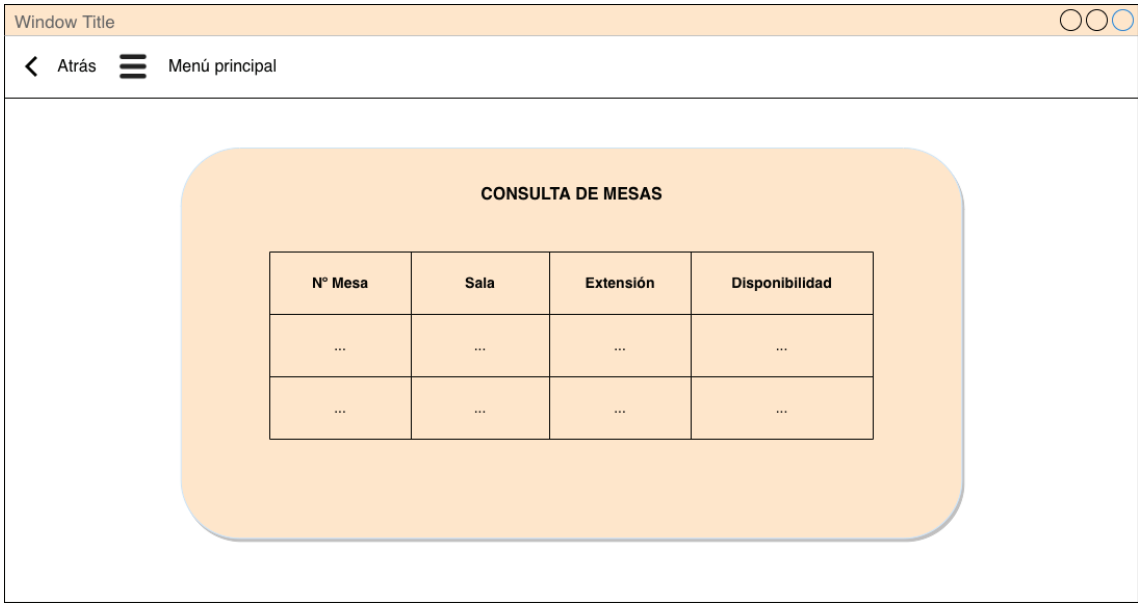

Ilustración 35: Prototipo de pantalla de mesas en usuario básico.

9.1.3 Vista de administrador

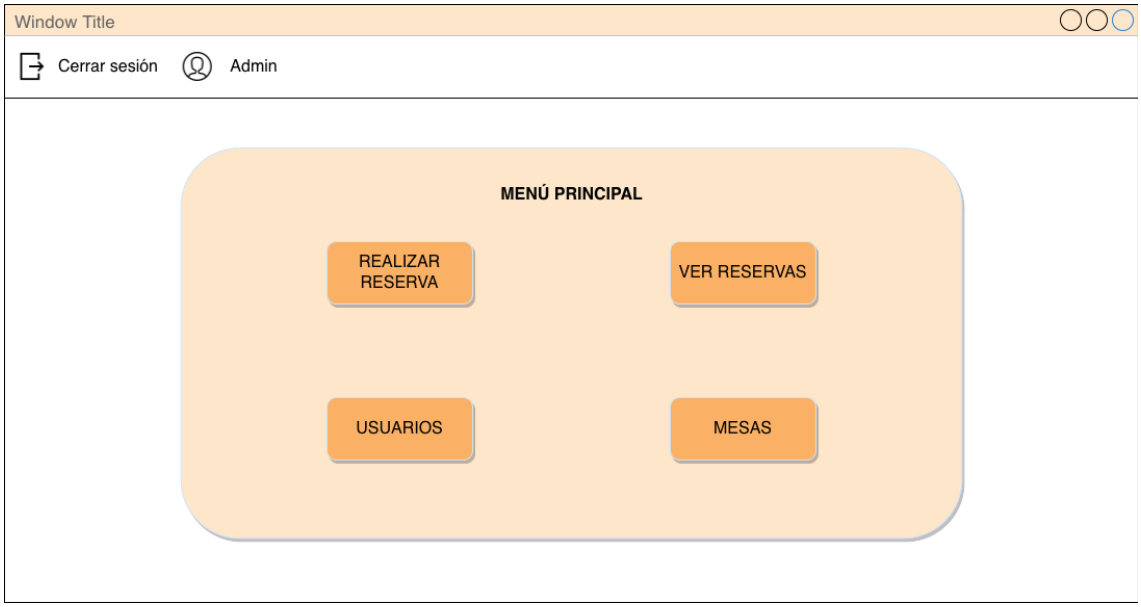

Ilustración 36: Prototipo de pantalla de menú principal para administrador.

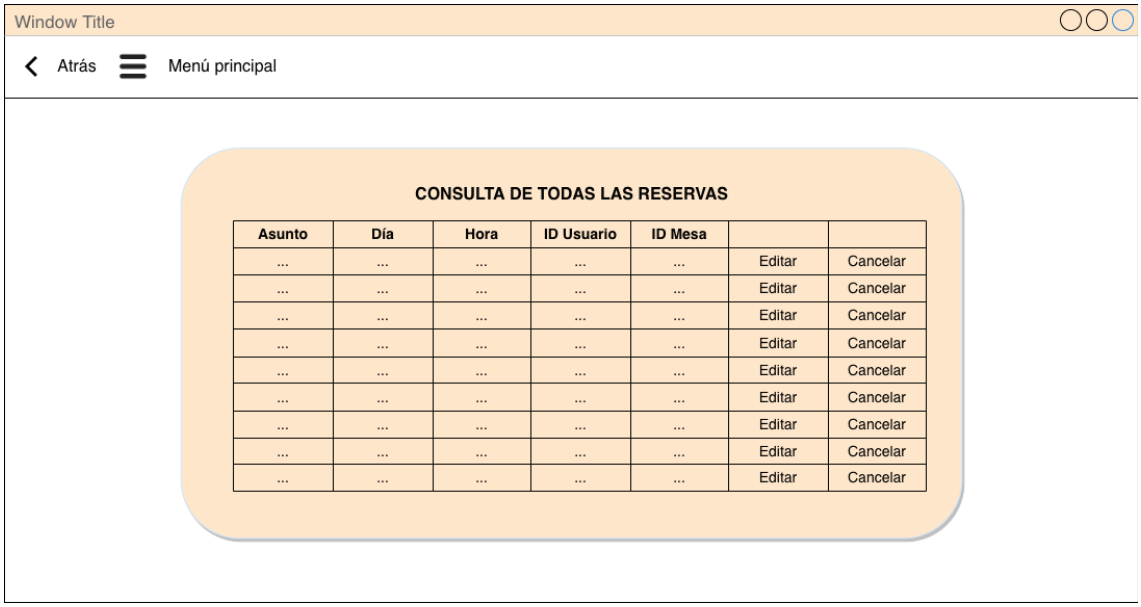

Ilustración 37: Prototipo de pantalla de todas las reservas en administrador.

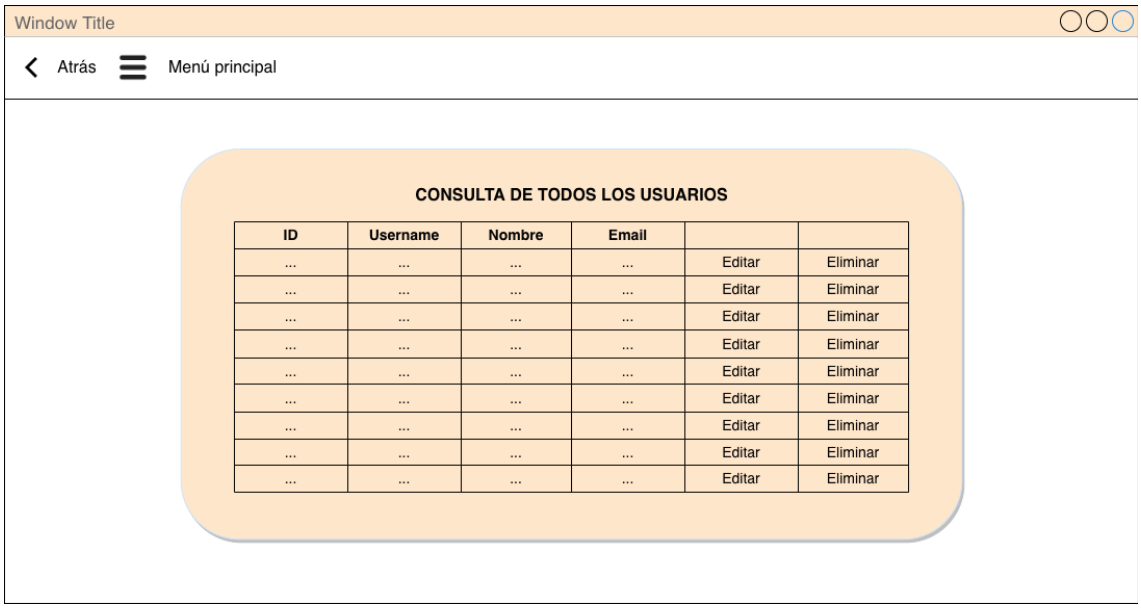

Ilustración 38: Prototipo de pantalla de usuarios en administrador.

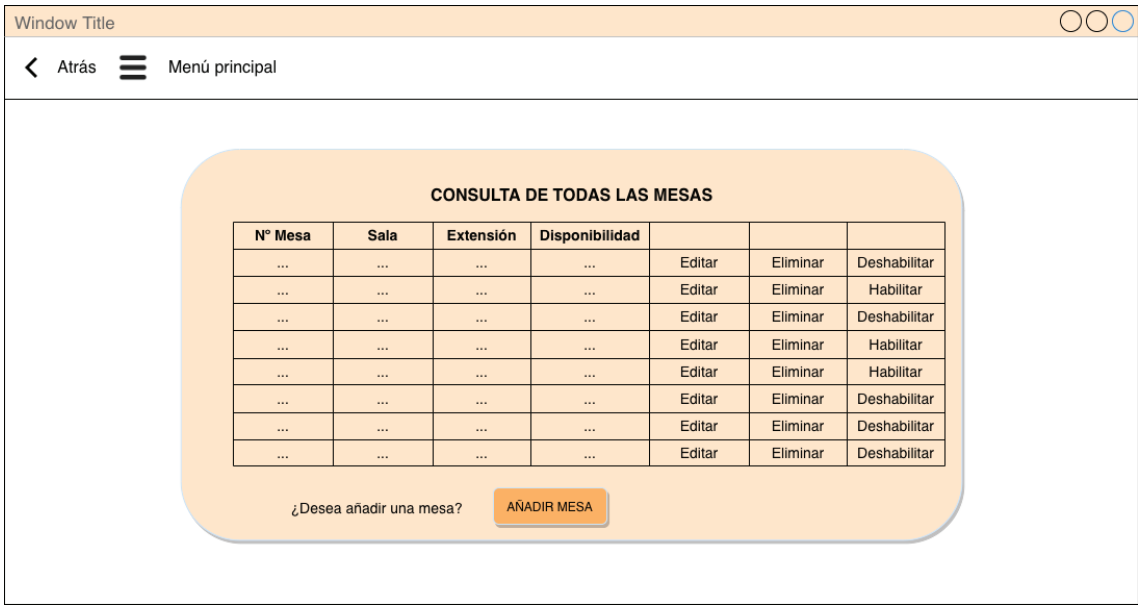

Ilustración 39: Prototipo de pantalla de mesas en administrador.

9.2 Anexo 2 – Interfaz web para usuarios

9.2.1 Pantallas de inicio

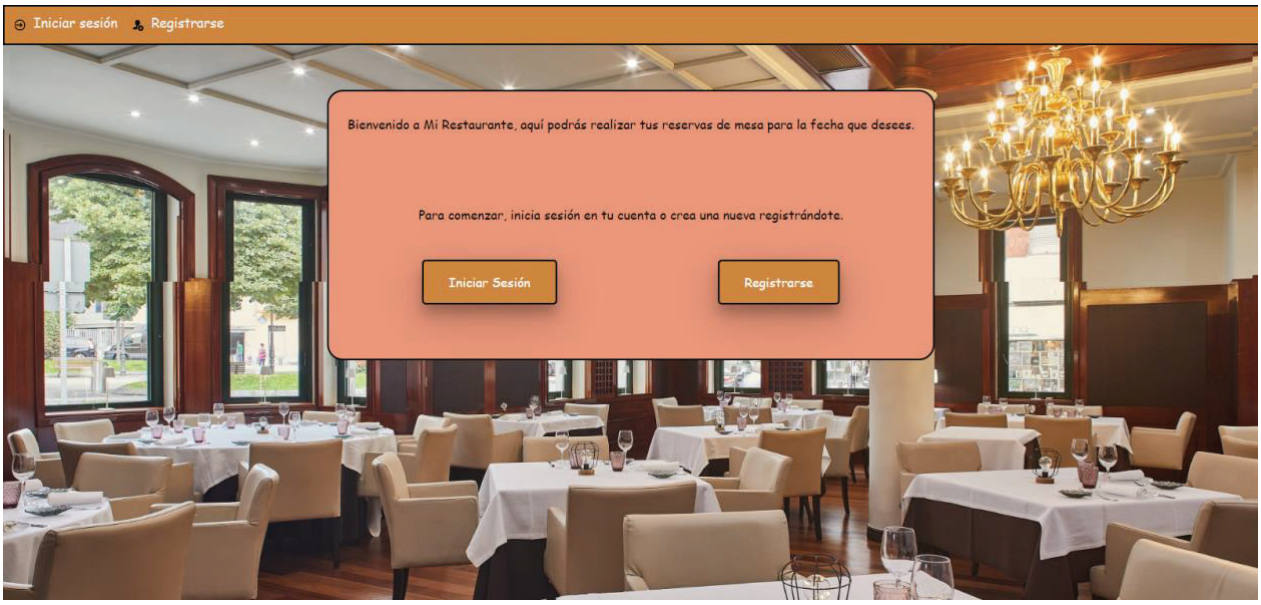

Ilustración 40: Pantalla de inicio de la plataforma web.

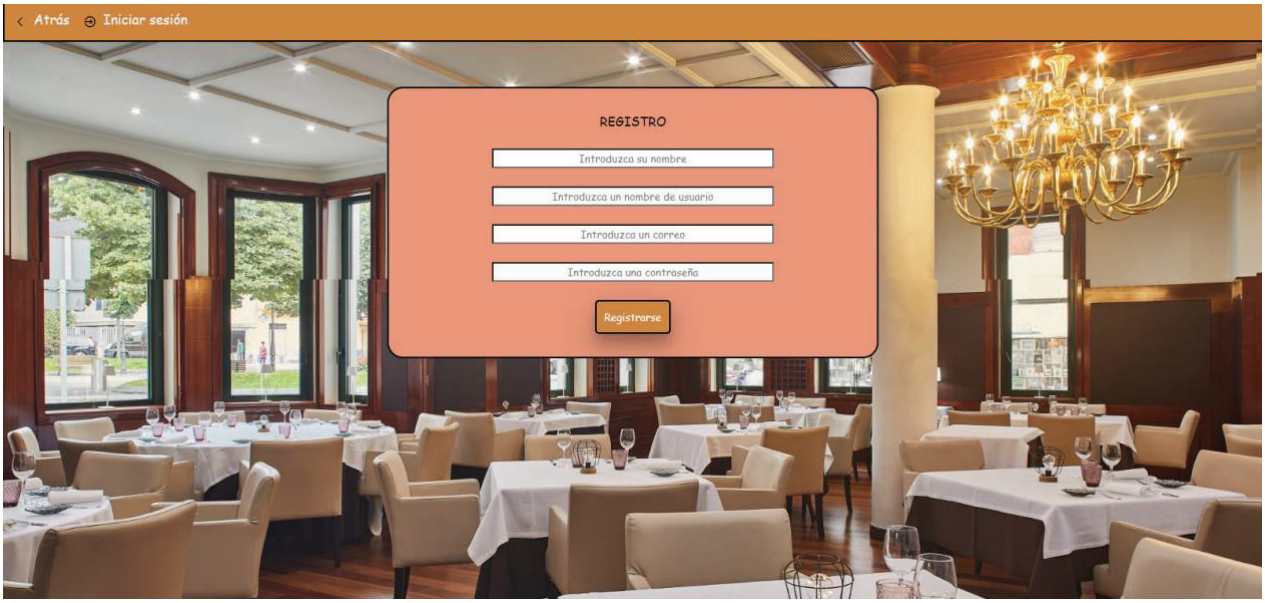

Ilustración 41: Pantalla de registro de un usuario.

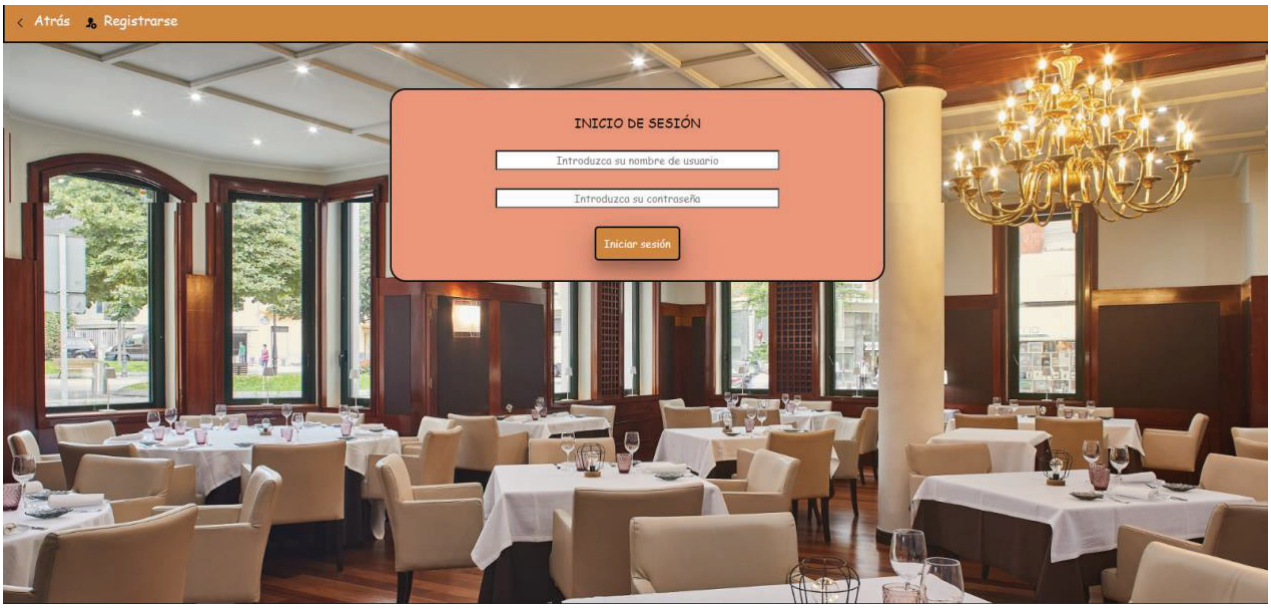

Ilustración 42: Pantalla de inicio de sesión.

9.2.2 Vista de usuario básico

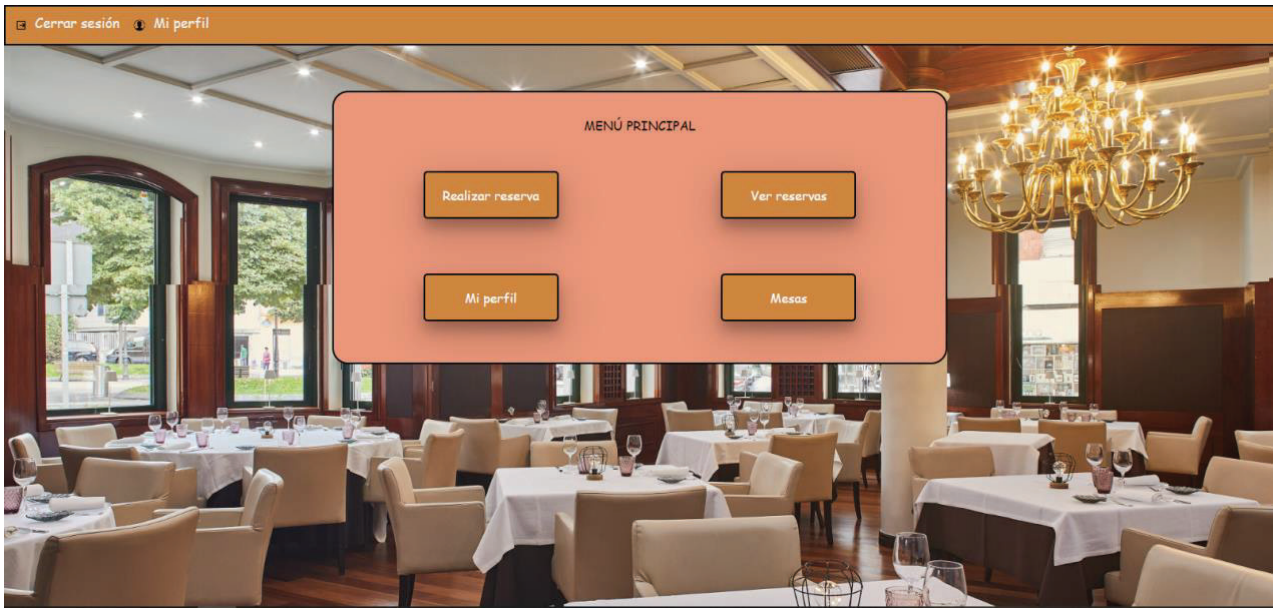

Ilustración 43: Pantalla de menú principal para usuario básico.

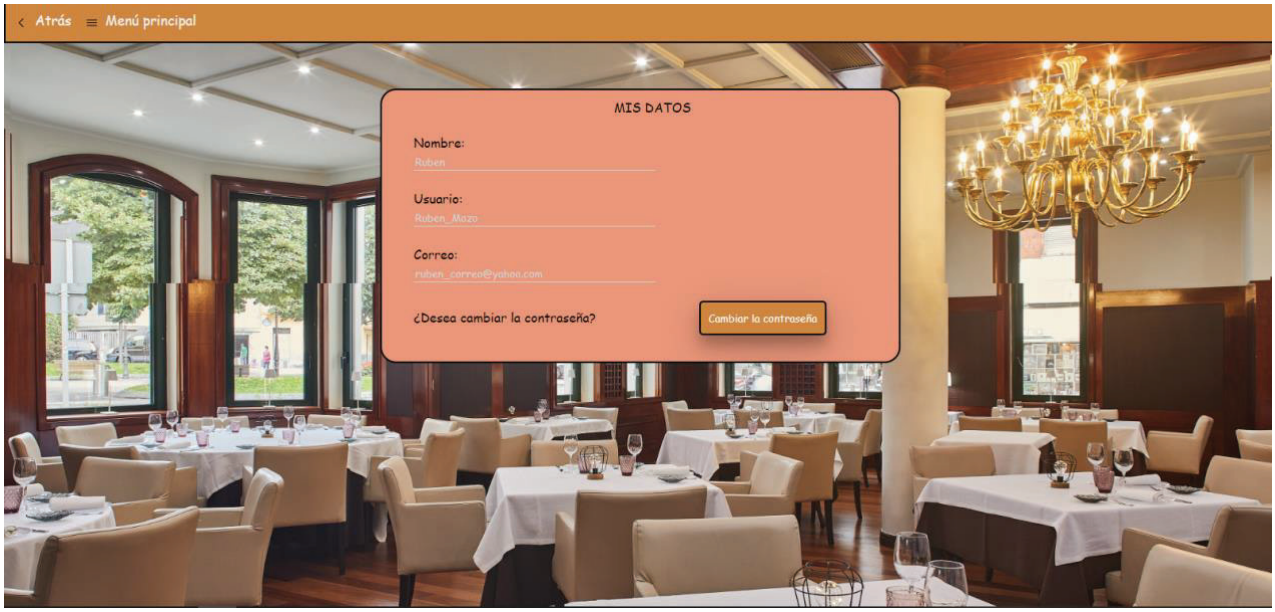

Ilustración 44: Pantalla de perfil de usuario.

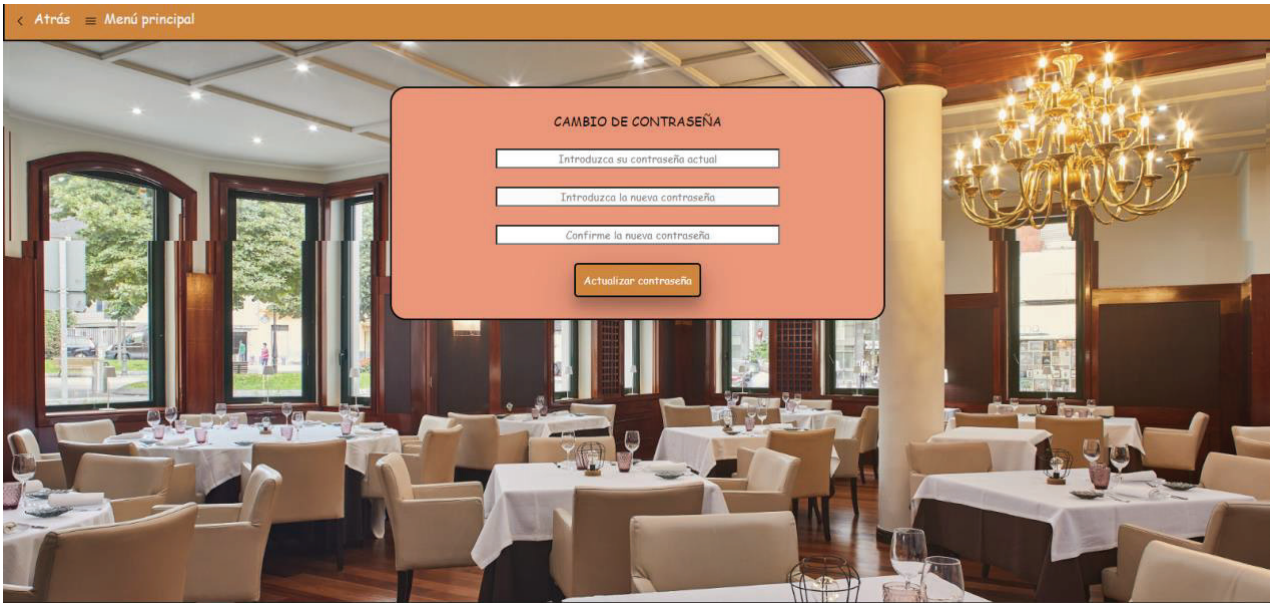

Ilustración 45: Pantalla de cambio de contraseña.

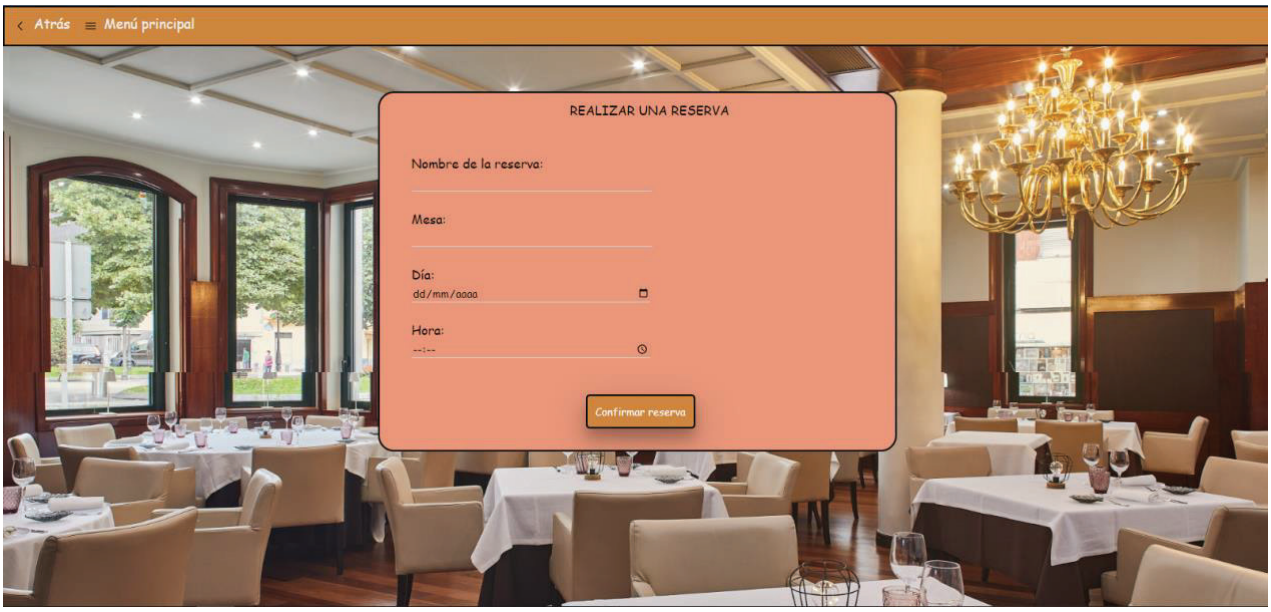

Ilustración 46: Pantalla de realización de reserva.

Ilustración 47: Pantalla de reservas de un usuario básico.

Ilustración 48: Pantalla de mesas a reservar por un usuario básico.

9.2.3 Vista de administrador

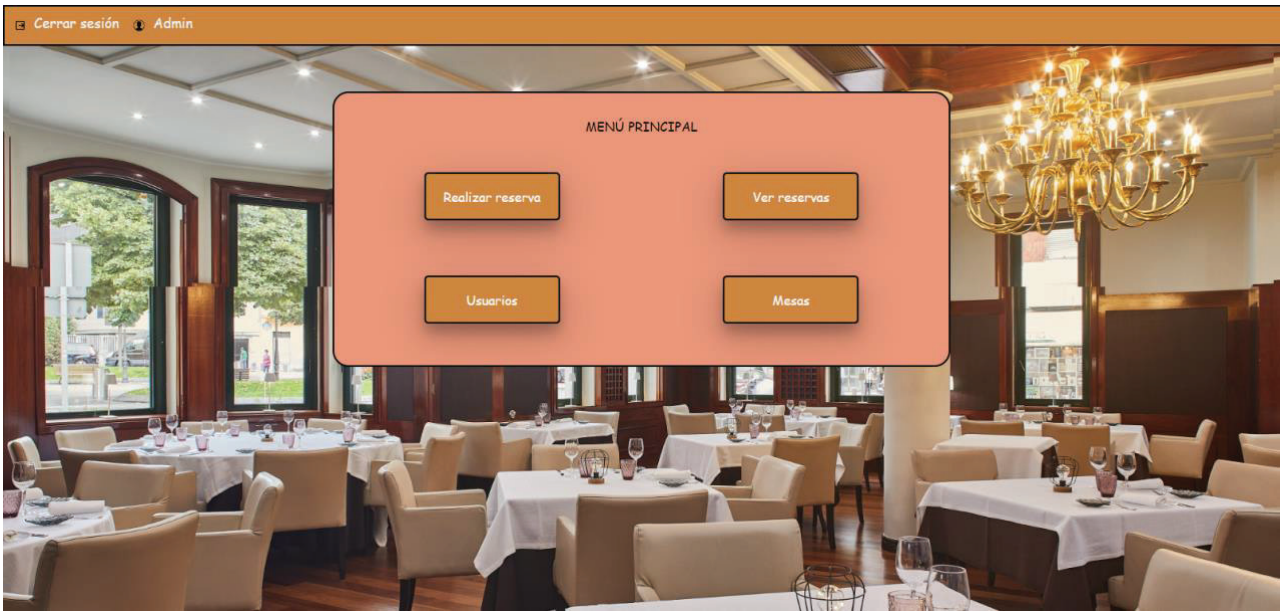

Ilustración 49: Pantalla de menú principal para un usuario administrador.

Ilustración 50: Pantalla de todas las reservas del sistema.

Ilustración 51: Pantalla de todos los usuarios del sistema.

Ilustración 52: Pantalla de todas las mesas del sistema.

** Las pantallas de realización de reserva, perfil de usuario y cambio de contraseña son prácticamente idénticas que las de un usuario básico para el caso de un usuario administrador, por ello, estas no se incluyen.*

** A través de las pantallas donde un usuario administrador accede a todos los usuarios, todas las mesas y todas las reservas; este usuario puede realizar cambios que un usuario básico no podría.*

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jun 29 16:08:49 CEST 2022
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)