PONTIFICIA UNIVERSIDAD CATÓLICA DEL

ECUADOR

FACULTAD DE INGENIERÍA

CARRERA DE: SISTEMAS DE LA INFORMACIÓN

Trabajo de Titulación

Tema: Desarrollo de un sistema web prototipo para la gestión del servicio a la habitación y optimizar la facturación en la “Hostería Vista Hermosa”.

AUTOR:

DIEGO ALEXANDER CERVANTES AVILES

QUITO 09, JUNIO DE 2024

ÍNDICE

FIGURAS

TABLAS

ANEXOS

CAPÍTULO I - INTRODUCCIÓN

1.1. TEMA

Desarrollo de un sistema web prototipo para la gestión del servicio a la habitación y optimizar la facturación en la “Hostería Vista Hermosa”.

1.2. JUSTIFICACIÓN

Es importante tener claro que el servicio de habitaciones en una hostería consiste en atender las necesidades de los huéspedes sin que estos tengan que desplazarse hasta la recepción, siendo una razón suficiente para que estos servicios sean parámetros de medida a la hora de elegir un buen lugar.

En la actualidad la gestión del servicio a la habitación es un aspecto fundamental para el éxito y la competitividad en la industria hotelera, que para brindar un excelente servicio requiere del apoyo tecnológico a través de un sistema de información que pueda apoyar a la eficiencia operativa, reducción de costos, minimizar errores y disminución de tiempos de espera, lo que permitirá mejorar la precisión en base a las necesidades de cada cliente y a su vez se traduce en un aumento de ingresos para este negocio.

El presente proyecto de titulación pretende cubrir la necesidad que tiene la “Hostería Vista Hermosa” para maximizar la experiencia del huésped y diferenciarse de otros, a fin de atraer un público más amplio.

1.3. PLANTEAMIENTO DEL PROBLEMA

La industria hotelera se enfrenta a un entorno competitivo en constante evolución, en el caso de las hosterías, la calidad del servicio al cliente desempeña un papel fundamental en la elección de hospedaje por parte de los consumidores. Por ende, el servicio a la habitación representa una parte esencial de la experiencia del cliente, lo que es vital para la reputación y progreso de una hostería.

Sin embargo, el desarrollo eficaz de este tipo de negocios se ha visto sujeto a la gestión eficiente de los servicios a la habitación, que normalmente se ven afectados por dificultades como retrasos, comunicación tardía, cobros indebidos, falta de organización y escasez de un sistema que permita un servicio personalizado. Estos problemas recaen cuando un mal servicio a la habitación puede llevar a una experiencia insatisfactoria para el turista, lo que, a su vez, puede resultar en comentarios negativos, pérdida de clientes y una disminución en los ingresos.

Por lo mencionado anteriormente, es crucial desarrollar un sistema web eficiente y personalizado que permita a las hosterías, en este caso especialmente a la "Hostería Vista Hermosa" la gestión del servicio a la habitación y optimizar la facturación. Este sistema deberá abordar los desafíos mencionados y proporcionar una experiencia extraordinaria, lo que a su vez contribuirá al crecimiento y la competitividad de esta hostería en el mercado hotelero.

1.4. OBJETIVOS

1.4.1. GENERAL

Desarrollar un sistema web prototipo para optimizar la gestión de servicio a la habitación en la "Hostería Vista Hermosa", mediante el uso de una metodología ágil que permita diseñarlo a partir de los requerimientos reales del caso de estudio.

1.4.2. ESPECIFICOS

- Analizar el proceso relacionado con la gestión de habitaciones en la hostería e identificar los requisitos esenciales para desarrollar el sistema.
- Seleccionar e implementar una metodología ágil de desarrollo para asegurar la entrega iterativa y continua de funcionalidades en el sistema web de gestión de servicio a la habitación, garantizando una documentación clara y un mejor desarrollo.
- Diseñar una interfaz de usuario amigable que facilite la gestión del servicio a la habitación y que cumpla los requisitos funcionales.

- Desarrollar el sistema web para la gestión de servicio a la habitación en la "Hostería Vista Hermosa", acorde al diseño realizado.
- Crear la documentación técnica y de usuario para el manejo adecuado del sistema desarrollado.

1.5. ALCANCE

El proyecto, denominado "Desarrollo de un sistema web prototipo para la gestión del servicio a la habitación y optimizar la facturación en la Hostería Vista Hermosa" se enfoca en la creación de un sistema de información prototipo para mejorar la eficiencia y calidad de los servicios de atención al cliente en la hostería. Las funcionalidades clave incluyen: un módulo de gestión de habitaciones, módulo de gestión de cuentas, módulo que calculará y mostrará el saldo a deber de cada cliente y un módulo de gestión de pedidos. Para enriquecer el proyecto, se desea realizar un análisis de requisitos a través de encuestas o entrevistas con el personal y los huéspedes, además de diseñar una interfaz amigable y accesible. Es importante destacar que se excluyen deliberadamente las siguientes funcionalidades: historial detallado de cada habitación y registro de clientes, para garantizar un desarrollo enfocado y evitar desviarse de lo definido.

CAPÍTULO II - MARCO TEÓRICO

En este capítulo II se concentra en las metodologías ágiles, de qué manera son útiles al momento de gestionar un proyecto, además se presenta las herramientas que se utilizarán dentro del desarrollo del sistema. Comenzando con las metodologías ágiles, se especifica en detalle las más comunes como, por ejemplo: Scrum, XP y Kanban, y se mostrará rápidamente cómo funcionan y sus conceptos básicos.

Posteriormente, se aborda el tema de las herramientas necesarias para el desarrollo, incluyendo frontend, backend, bases de datos y la plataforma 'JIRA' para la gestión de proyectos ágiles. Estas herramientas se presentan como principales para el éxito de este plan, proporcionando una buena base teórica para comprender y aplicar eficazmente metodologías ágiles.

2.1 METODOLOGÍAS ÁGILES

A partir de los años noventa del siglo anterior, aparecieron las denominadas metodologías ágiles para el desarrollo del software, en un inicio tomaron el nombre de "ligeras" pero posteriormente adoptaron la nomenclatura actual. Su finalidad fue restar las probabilidades de fracasar debido a aspectos como: el costo, la funcionalidad o el tiempo que toma desarrollar proyectos de software. Esta alternativa suplantó a los métodos tradicionales y se orientó a optimizar los procesos (Navarro Cadavid, 2013).

La **Figura 1** presenta una comparación de las metodologías ágiles y tradicionales, en donde podemos ver que la primera sigue su ciclo de forma lineal, lleva una documentación rigurosa y es menos eficiente al momento de generar cambios, por otro lado, las ágiles son flexibles, rápidas e innovadoras, lo que ayudará a desarrollar este plan de titulación de manera eficiente.

Figura 1

Comparación Metodologías Ágiles Vs Tradicionales

Nota. El gráfico fue tomando de Metodología tradicional vs. Metodología scrum: ¿cuál es la mejor opción para tu empresa – SAC Panamá. (s. f.). https://sacpma.com/metodologia-tradicional-vs-metodologia-agil-2/

Dado que la gestión de proyectos dentro de organizaciones se encarga especialmente del desarrollo del software y optimización, fue necesaria la implantación de métodos que conduzcan a mejores resultados. Por lo tanto, vemos las metodologías ágiles como la unión de técnicas aplicadas en periodos breves, que facilitan el trabajo. En este sentido, se ve viable utilizarla para generar una propuesta de tesis exitosa, debido a que son mejores que los tradicionales (Figueroa, Solís, & Cabrera, 2008).

2.1.2 SCRUM

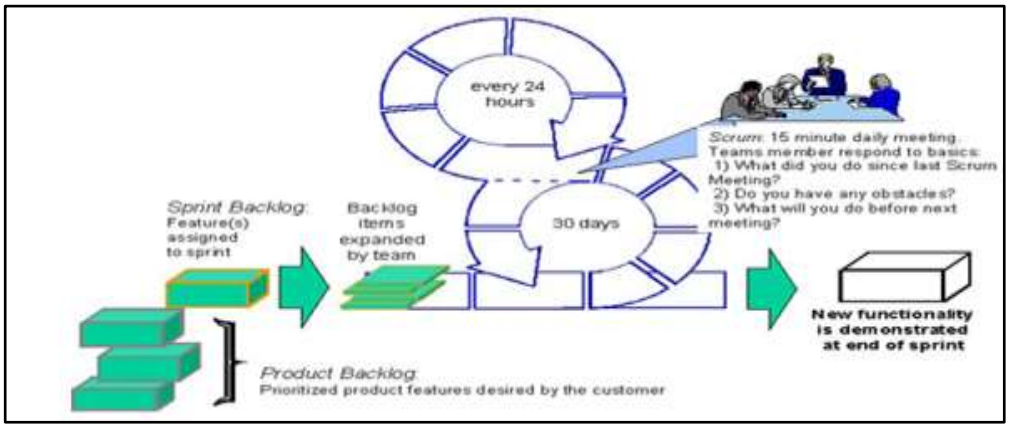
SCRUM es un instrumento muy utilizado dentro del conjunto de las metodologías ágiles. Es así que (Islam & Ferworn, 2020) la describen como "un esqueleto de proceso con prácticas y roles pre-definidos" (p. 13). Lo expresado significa que esta herramienta proporciona múltiples pautas al momento de desarrollar un sistema, permitiendo dejar la fase de planificación y generar el desarrollo incremental, buscando ofrecer el mejor producto posible en la menor cantidad de tiempo.

En la **Figura 2** se puede ver el ciclo de vida de SCRUM y todas las partes que lo componen de forma concisa y clara. En la parte inferior izquierda, se ve el Product Backlog, que es una lista que representa tareas pendientes y mejoras. Encima a él, vemos la parte del Sprint Backlog, que

selecciona los trabajos más importantes para el próximo Sprint. A lado vemos al Backlog que básicamente, recopila todas las labores y asigna roles. En el medio se encuentra el Sprint, en donde se realizan tareas como: codificar, diseñar y mejorar el producto final. En otro rincón tiene lugar el Daily Scrum, que una breve reunión diaria para coordinar al equipo. Al final del Sprint, se logra obtener una versión del producto, la cual se seguirá mejorando para el próximo ciclo.

Figura 2

Proceso Scrum



Nota. El gráfico fue tomando de Islam, A. K., & Ferworn, D. A. (2020). A Comparison between Agile and Traditional Software Development Methodologies. Obtenido de Global Journal of Computer Science and Technology: https://doi.org/10.34257/GJCSTCVOL20IS2PG7.

Estos beneficios llevan a considerar que la flexibilidad y adaptabilidad de SCRUM son la mejor manera de dirigir un proyecto en base a nuevas prioridades o cambios. Al tener a todos los miembros del proyecto informados desde el inicio y trabajando en conjunto con el equipo, promueve una mayor productividad y calidad al simplificar y mejorar continuamente los procesos de trabajo (Tymkiw, Bournissen, & Tumino, 2020). En efecto, vemos que el enfoque iterativo e incremental de esta metodología es la mejor vía para identificar y mitigar los riesgos desde las primeras etapas del proyecto, lo que contribuye a tener mejores resultados.

2.1.3 XP

En 1999, salió el primer libro sobre Programación Extrema (XP) escrito por Kent Beck, tenía un enfoque de desarrollo de software conocido por sus prácticas ágiles y su énfasis en la sencillez. XP, es una metodología que funciona mediante la implementación de entregas incrementales y la simplificación de los requisitos. Se basa en principios ágiles y promueve la

comunicación a través de reuniones periódicas (Sánchez-Hernández, Lizano-Madriz, & Sandoval-Carvajal, 2020). XP también enfatiza el manejo de los requisitos a alto nivel y la ejecución de pruebas de usuario y desarrollo dirigido. Sin embargo, se enfrenta a desafíos en la integración de la usabilidad y la interacción humano-computadora, lo que dificulta su adaptación en entornos ágiles y la ejecución efectiva de pruebas de usabilidad.

La **Figura 3** muestra el esquema de Programación Extrema (XP), una metodología ágil de desarrollo de software. En la imagen vemos las fases que cuenta esta metodología y porque se caracteriza XP. Este método se compone de 5 etapas interconectadas:

- Planificación: En esta etapa, se definen las historias de usuario y se prioriza lo más importante. Finalmente se revisa lo expuesto dentro de un determinado tiempo.
- Diseño: Una vez planificado, se procede a trabajar en lo principal, ya que se busca obtener un prototipo inicial.
- Codificación: Con el diseño en marcha, se inicia la codificación. Los programadores trabajan conjuntamente, ya que se busca generar buenas prácticas.
- Pruebas: A medida que se desarrolla el código, se realizan pruebas continuas para verificar su funcionamiento y errores.
- Lanzamiento: Finalmente, una vez que se superó todo lo expuesto se procede al lanzamiento o despliegue del producto (Canive, 2020).

Figura 3

Esquema Metodología XP

Nota. La imagen fue obtenida de Canive, T. (27 de mayo de 2020). Metodología XP o Programación Extrema: ¿Qué es y cómo aplicarla? Obtenido de Sinnaps: https://www.sinnaps.com/blog-gestion-proyectos/metodologia-xp

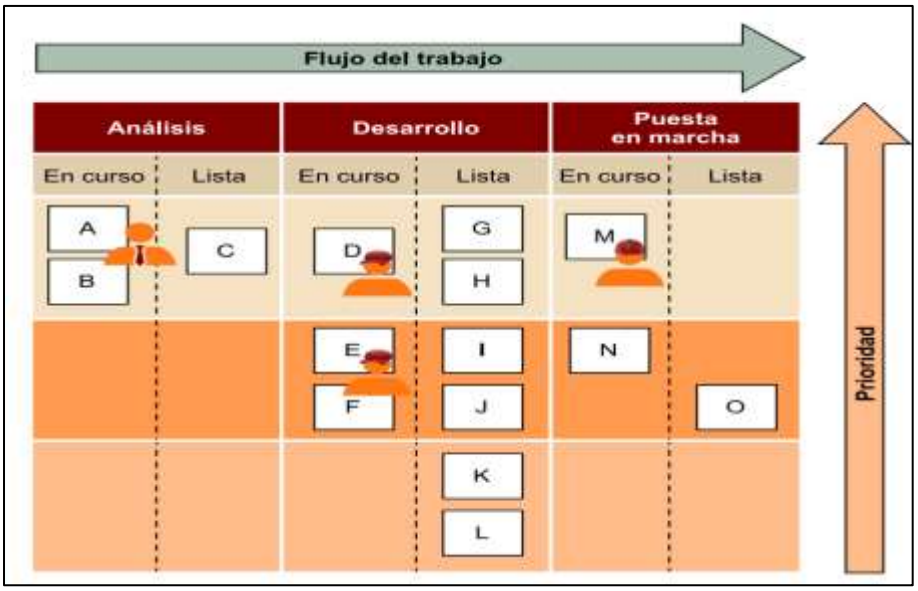
2.1.4 KANBAN

Kanban es un sistema de gestión del trabajo en curso que busca producir la cantidad justa de trabajo que el sistema puede manejar sin sobrecargar a las personas o al equipo en general. Se enfoca en limitar el trabajo en curso para mejorar la calidad y reducir el tiempo de servicio de las tareas. Es una aproximación flexible a la gestión del cambio organizativo, adaptándose a la situación actual de la empresa sin necesidad de empezar de cero (Bermejo, 2011).

La **Figura 4** muestra un ejemplo de la aplicación práctica de Kanban, donde se considera un escenario común, en que un equipo se dedica al desarrollo evolutivo de tareas pequeñas. En la imagen podemos diferenciar a diversos actores (programadores, técnicos expertos y analistas), los cuales se encargan de partes específicas del tablero. Las flechas situadas en la parte superior y al final de la ilustración indican claramente el flujo de trabajo y la prioridad dentro del sistema Kanban. Finalmente, Kanban divide todo por secciones, de las cuales nos dice lo que se encuentra listo y lo que no.

Figura 4

Tablero Trabajo Kanban



Nota. La figura tiene como fuente: Bermejo, M. (2011). El Kanban. Barcelona, España: UOC.

Al analizar el panel Kanban, se obtiene una comprensión detallada del progreso de las tareas en curso, los tiempos de ejecución y las oportunidades de mejora en la gestión del trabajo. Esto se convierte en una herramienta esencial para perfeccionar el proceso de producción y elevar la calidad del producto final.

2.1.6 CASOS DE USO

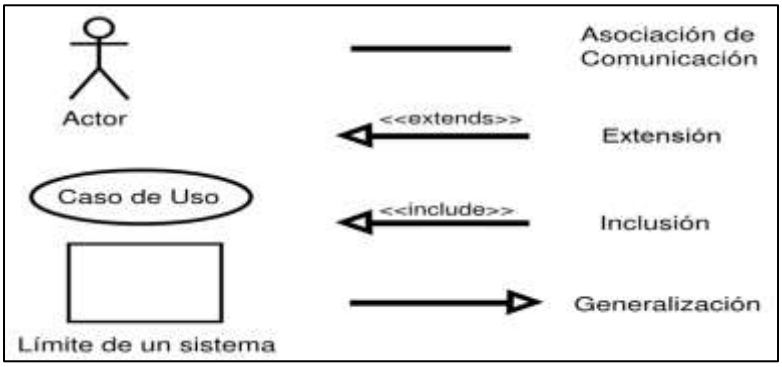
Jacobson introdujo el concepto de casos de uso en 1992 (Jayaraman & Whittle, 2007), su naturaleza implica describir los requisitos funcionales derivados de las necesidades del cliente. Este enfoque fue adoptado rápidamente por los especialistas en métodos orientados a objetos para puntualizar comportamientos externos de los sistemas, así como su notación. (Jayaraman & Whittle, 2007). El éxito de los casos de uso radica en comprender un sistema a través de las secuencias o interacciones con los actores, para ello, es preciso describir la funcionalidad del sistema.

La **Figura 5** muestra las partes que representa un modelo de casos de uso. En la parte superior izquierda se encuentran los actores, que son entidades externas al sistema que interactúan con él. Estos actores pueden ser humanos, sistemas externos o entidades abstractas como el tiempo. Justo debajo de los actores se encuentran un óvalo (casos de uso), que son las acciones o procesos que el sistema puede realizar. En la parte inferior izquierda de los casos de uso se representa un cuadrado (límite del sistema), que establece las fronteras y el alcance del sistema. En la parte derecha de la imagen, se observan una línea (asociación) que conectan los actores con los casos de uso correspondientes.

Además, en la parte inferior derecha se encuentran otras líneas que representan relaciones de inclusión y extensión entre diferentes casos de uso. La inclusión denota la integración de comportamientos de escenarios en otros casos de uso, mientras que la extensión permite agregar funcionalidades adicionales de manera controlada a un caso de uso base (WIKIPEDIA, 2012).

Figura 5

Elementos Casos de Uso



Nota. La imagen fue obtenida de: WIKIPEDIA. (2012). Caso de uso. Obtenido de WIKIPEDIA La Enciclopedia Libre: https://es.wikipedia.org/wiki/Caso_de_uso.

En conjunto, la imagen ofrece una representación visual clara de los elementos que tenemos en los casos de uso, y cómo existen flechas o líneas que definen el comportamiento y la funcionalidad del sistema.

2.2 HERRAMIENTAS

Una sociedad pluricultural, que se desarrolla en un mundo globalizado como sucede en la actualidad, requiere de contenidos que resulten amigables de interés y de fácil uso. Justamente por esta razón aparece múltiples herramientas para generar conocimiento, algunas de ellas se caracterizan por ser de uso libre, mientras que otras, cuenta con propietarios. A fin de poder crear cualquier tipo de contenido o diseñar eventos de forma sofisticada es importante tener conocimientos de lenguajes de programación como HTML y JavaScript (Díaz & Medina, 2020). Por tanto, las herramientas de desarrollo de software consisten en programas aplicaciones o utilidades que son manejadas por los desarrolladores para crear, probar, mantener o depurar programas informáticos.

2.2.1 REACT

React es una herramienta muy útil para el desarrollo web y móvil. Fue creada por Facebook y está desarrollada por un conjunto de bloques de código escritos en JavaScript. La principal función que tiene es renderizar los componentes de la capa de vista de una aplicación. React se presenta como una alternativa a otros frameworks como Angular o Vue.js, que permiten desarrollar aplicaciones con funcionalidades más avanzadas (Deyimar, 2023) .

Una de las características fundamentales es su especial modo para la creación de componentes independientes, proporcionando una reutilización eficiente y construyendo interfaces de usuario complejas. Este enfoque está basado en un modelo de programación de componentes, en el que las partes individuales de los componentes realizan las funciones individuales con las que ejecuta el usuario (Hunabku.mx., s.f.).

Al analizar React podemos decir que es ventajoso utilizarlo, ya que la programación es más fácil, debido a la reutilización de componentes. Esta guía modular convierte a React en una herramienta potente y flexible.

2.2.2 BACKEND

El backend se refiere a la parte del software que no interactúa directamente con el usuario, pero sí ejecuta la lógica en la aplicación y manejo de las bases de datos, la autenticación, las autorizaciones, y la lógica de negocio. Las tecnologías del backend incluyen lenguajes de programación como Python (y frameworks como Django o Flask), JavaScript (Node.js, Express), Ruby (Ruby on Rails), PHP, Java (Spring Boot), entre otros. También implica el manejo de servidores, bases de datos, y la arquitectura de las aplicaciones (AWS, 2023).

Por otro lado, el backend, también llamado lado del servidor gestiona los datos y la infraestructura que permite el funcionamiento de la aplicación. Almacena y procesa datos de aplicaciones en nombre de los usuarios. Cuando un usuario interactúa con el front-end, dichas interacciones desencadenan solicitudes que se envían al back-end a través de HTTP. El backend procesa estas solicitudes y devuelve respuestas al frontend.

El backend normalmente interactúa con varios componentes, incluidos:

- Servidor de base de datos para recuperar o modificar datos relevantes.
- Microservicios que realizan tareas específicas solicitadas por los usuarios.
- API de terceros para recopilar información adicional o realizar funciones adicionales.

El desarrollo backend a menudo utiliza lenguajes de programación como Python, Java o Ruby para escribir lógica del lado del servidor, administrar bases de datos, tecnologías de almacenamiento y comunicaciones API. Por consiguiente, el front-end se centra en los aspectos de una aplicación relacionados con el usuario, incluida la interfaz de usuario y la interacción del usuario, mientras que el back-end gestiona la funcionalidad, la informática y la infraestructura de la aplicación. Ambos son esenciales para crear una aplicación coherente y funcional y, a menudo, trabajan juntos para brindar una experiencia de usuario perfecta.

2.2.3 FRONTEND

Frontend se representa como una parte de la aplicación a través de la cual interactúan los usuarios, también conocido como la “interfaz de usuario”. Incluye todo lo que el usuario ve y con lo que interactúa en el navegador o aplicación móvil, como formularios, botones, imágenes y otros elementos gráficos. Las tecnologías fundamentales del frontend incluyen HTML, CSS y JavaScript, y se complementan con frameworks/librerías como React, Angular, y Vue.js para crear

interfaces ricas y dinámicas.

El término “frontend” al ser la parte del sistema informático con las que interactúan los usuarios finales se refiere a la interfaz de usuario y la lógica de presentación que permite a los usuarios interactuar con la aplicación. Esto podría incluir la interfaz gráfica de usuario (GUI) en una aplicación web o móvil, así como la lógica de interacción del usuario. Un estudio realizado por (Parks & Hall, 2016) ha buscado simplificar la conexión entre el frontend de la aplicación y la base de datos, permitiendo a los desarrolladores centrarse en el diseño y la funcionalidad del frontend sin preocuparse por la implementación del backend. Esto se logra mediante el uso de tecnologías como Node.js y MongoDB para crear un backend eficiente y escalable que pueda manejar múltiples solicitudes de usuarios simultáneamente, garantizando al mismo tiempo la privacidad y seguridad de los datos.

El frontend, también llamado lado del cliente es con lo que los usuarios interactúan directamente. Incluye los elementos visuales de una aplicación, como botones, casillas de verificación, gráficos y mensajes de texto, que permiten a los usuarios interactuar con la aplicación. Técnicamente, la interfaz incluye los elementos de la interfaz de usuario (UI) que los usuarios ven e interactúan, a menudo representados por el Modelo de objetos de documento (DOM). Los principales lenguajes que impactan la interfaz incluyen HTML para definir la estructura, CSS para diseñar y JavaScript para agregar funcionalidad dinámica a la interfaz de usuario (AWS, 2023).

2.2.4 BASE DE DATOS

Las bases de datos se crean con la necesidad de guardar datos, para luego procesarlos en información valiosa. Hace algunos años, la información se guardaba en papel, pero con la llegada de la tecnología, se puede almacenar grandes cantidades de datos en dispositivos electrónicos.

Para manejar estas bases de datos, usamos sistemas llamados DBMS (Sistemas de Gestión de Bases de Datos), encontramos los siguientes: MySQL, Microsoft SQL Server, Oracle, etc. (Etecé, 2023).

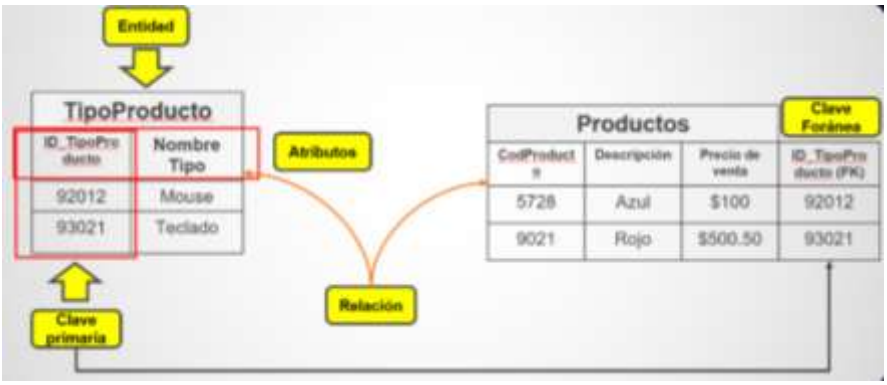
En la **Figura 6** podemos ver las partes del modelado de base de datos, en la cual encontramos varios elementos, como, por ejemplo:

- Entidad: Son el tema que aborda una tabla en la base de datos.
- Atributos: Son todos los elementos de la entidad, se lo puede traducir como el encabezado.

- Clave primaria: Es un atributo de la tabla, pero este permite identificar cada uno de los registros o campos de nuestra tabla con un número o código.
- Clave foránea: Se da cuando una tabla le presta su clave primaria a otra tabla.
- Relación: Se da cuando existe una relación entre tablas. (Ramírez, Pintado, & Contreras, 2023)

Figura 6

Elementos Base de datos



Nota. La imagen fue obtenida de: Ramírez, C., Pintado, D., & Contreras, E. (09 de 01 de 2023). Elementos básicos de una base de datos. Obtenido de UMAEE: https://www.umaee.edu.mx/blog/elementos-basicos-de-una-base-de-datos

2.2.5 JIRA

Jira es una herramienta de gestión de proyectos muy popular que contribuye al seguimiento de problemas y errores en el desarrollo del software. Su evolución tecnológica le ha permitido convertirse en una versátil solución que utiliza amplia gama de sectores y contextos. Su funcionalidad en la planificación, seguimiento y gestión de proyectos permite a los equipos organizarse y priorizar el trabajo, creando tableros personalizados, así como asignando tareas, estableciendo fechas límites y realizando el seguimiento y progreso de los proyectos (Mercado & Elizabeth, 2022).

Los planes en Jira también tienen otras características. Por ejemplo, los proyectos de la plataforma a menudo contienen elementos de trabajo similares. Para entender el resto de la estructura interna de Jira, tenemos que observar de qué forma lo trabaja en la aplicación y como tiene su organización jerárquica (Arnautović, 2022):

- Categorías de proyectos: el primer nivel, que representa los conceptos de proyectos abarcadores y cómo se relaciona el trabajo en proyectos con otros proyectos.
- Proyectos: este nivel incluye componentes y versiones de proyectos, que también pueden considerarse hitos de proyectos.
- Elementos de trabajo: también llamados tareas de trabajo en métodos ágiles, ayudan a cumplir con los requisitos de las historias de usuario, clasificados por tipos de elementos.
- Subtareas: esta característica te permite dividir elementos de trabajo complejos en unidades más pequeñas y manejables (Arnautović, 2022).

Las mejores prácticas de gestión de proyectos en Jira incluyen configurar un proyecto de manera adecuada, asignar roles apropiados, establecer comunicación efectiva y diseñar flujos de trabajo eficientes, con lo dicho, se desea implementar el uso de esta aplicación para lograr llevar un mejor control del progreso, priorizar tareas y sacar el mejor producto posible.

CAPÍTULO III – PLANIFICACIÓN

En la fase de planificación para el desarrollo de un sistema web prototipo para la gestión del servicio a la habitación y optimizar la facturación en la “Hostería Vista Hermosa”, se utiliza el Product Backlog para crear lista de tareas, mejoras y funcionalidades pendientes, priorizadas según su importancia. Esta lista proporciona una visión clara de los objetivos del proyecto. El Kanban Board es una herramienta visual que facilita la gestión del flujo de trabajo,

Utilizar varias técnicas en un proyecto puede ayudar a distribuir mejor la carga de trabajo, por lo tanto, se decidió utilizar la metodología Scrum con Kanban, ya que es una manera mejorada para la colaboración y la transparencia en el proceso.

La planificación se lleva a cabo en sprints los cuales se abordan tareas seleccionadas del Product Backlog. Cada sprint tiene objetivos claros y entregables definidos, lo que permite un enfoque ágil y eficiente en el desarrollo del sistema. Al final de cada sprint, se revisa el progreso y se ajusta la planificación según las necesidades del proyecto, garantizando un proceso iterativo y adaptable.

3.1 PRODUCT BACKLOG

El producto backlog de un proyecto siempre se lo debe ver como una herramienta indispensable para la gestión, ya que se lo define con un listado de todas las tareas que se desean realizar durante el tiempo de desarrollo, permitiendo mantener informados a los actores del plan y dar las limitaciones del proyecto (Molina, 2023).

Con el fin de entender de manera más fácil las necesidades principales del proyecto, se asignaron lo roles que se van a cumplir.

Product Owner: Dueño de la Hostería, el cual nos ayudará a generar el backlog.

Scrum Master y Desarrollador: Diego Cervantes, es la persona encargada de administrar y realizar el proyecto.

Al momento de tener los actores principales del proyecto, se debe obtener el listado de tareas (requerimientos) que alimentarán al backlog, en el **Anexo A** se muestra un link para ingresar a un audio, en el que se evidencia una entrevista con el dueño del producto, la cual es indispensable para generar y priorizar los requisitos.

En la **Tabla 1**, **2, 3, 4, 5, 6, 7** se ve los requerimientos del sistema, en donde encontramos parámetros como: nombre, identificador, prioridad, tipo y descripción.

Tabla 1

Panel Administrador del Sistema

Identificador**: S1**	Nombre**: Panel Administrador**	
Tipo**: Necesario**	**Crítico**: Si	**Prioridad**: Alta
Descripción**: Se desea que exista una parte que divida al dueño del negocio con lo demás, en donde se puede ver la información general y administrar permisos.**		

Nota. En esta tabla se describe uno de los requerimientos del sistema.

Tabla 2

Gestión de Productos

Identificador**: S2**	Nombre**: Gestión de Productos**	
Tipo**: Necesario**	**Crítico**: Si	**Prioridad**: Alta
Descripción**: En esta parte queremos que en el panel administrador pueda hacer un CRUD de los productos disponibles.**		

Nota. En esta tabla se describe uno de los requerimientos del sistema para administrar productos.

Tabla 3

Gestión Categorías

Identificador**: S3**	Nombre**: Gestión de Categoría**	
Tipo**: Deseable**	**Crítico**: Si	**Prioridad**: Media
Descripción**: Para la gestión y organización de los productos queremos ubicar categorías que los diferencien.**		

Nota. En esta tabla se describe uno de los requerimientos del sistema.

Tabla 4

Historial de Pagos General

Identificador**: S4**	Nombre**: Historial de Pagos General**	
Tipo**: Necesario**	**Crítico**: Si	**Prioridad**: Media
Descripción**: Se quiere tener un control de desempeño y las ventas generales que se tiene**		

en el servicio a la habitación.

Nota. En esta tabla se describe uno de los requerimientos del sistema.

Tabla 5

Gestión de habitaciones

Identificador**: S5**	Nombre**: Gestión de habitaciones**	
Tipo**: Necesario**	**Crítico**: Si	**Prioridad**: Alta
Descripción**: Este parte es importante, porque se desea que se muestre el estado de cada habitación (Existe pedido, No existe y En marcha).**		

Nota. En esta tabla se describe uno de los requerimientos del sistema.

Tabla 6

Panel de Cliente

Identificador**: S6**	Nombre**: Panel de Cliente**	
Tipo**: Necesario**	**Crítico**: Si	**Prioridad**: Alta
Descripción**: Se desea que del lado del huésped pueda generar ordenes de comida o cualquier servicio extra.**		

Nota. En esta tabla se describe uno de los requerimientos del sistema.

Tabla 7

Generación de Códigos QR

Identificador**: S7**	Nombre**: Generación de Códigos QR**	
Tipo**: Necesario**	**Crítico**: Si	**Prioridad**: Media
Descripción**: Esta sección es para que pueda ingresar a la app fácil.**		

Nota. En esta tabla se describe uno de los requerimientos del sistema.

Dentro de la metodología de scrum debemos asignar una estimación de las prioridades de cada requerimiento, escogemos dar el número 4 como la prioridad más alta y el número 1 como prioridad baja.

En la **Tabla 8** se puede ver el identificador de cada requerimiento y el número según la prioridad.

Tabla 8

Prioridades de Cada Requerimiento

Prioridad Alta	*Requerimiento*	*Estimación*	*Prioridad*
	S1	4	4
	S6	4	4
	S5	3	4
	S2	3	3
	S4	2	2
	S3	2	2
Prioridad Baja	S7	2	3

Nota. Esta tabla describe las prioridades de cada requerimiento.

3.2 KANBAN BOARD

En la **Figura 7** nos da una demostración gráfica de las tareas que se deben realizar, los actores que van a participar y la actividad asignada. Dentro de Kanban nos pide asignar una clave única para cada tarea y vemos que se mantiene la misma numeración, pero la clave cambia.

Figura 7

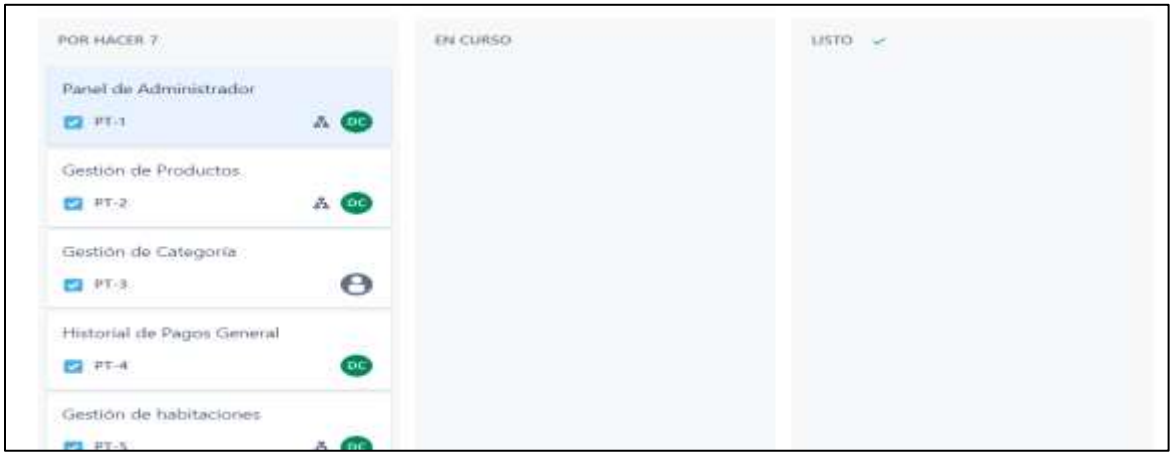
Lista Tareas Por Hacer

Nota. Esta lista fue creada en la plataforma Jira, especializada en la gestión de proyectos.

En la **Figura 8** nos muestra nuestro tablero inicial en Kanban, podemos ver que no existen cambios y que todos requerimientos se encuentran en la sección “Por Hacer”.

Figura 8

Kanban Board



Nota. Este tablero fue creado en la plataforma Jira, ya que se especializa en la gestión de proyectos.

La **Figura 9** es una de las partes más importante del Kanban, ya que nos da una descripción del requerimiento, el estado, fecha de finalización, subtareas y la carga que tiene la persona a la que se le va a asignar.

Figura 9

Referencia Descripción Requerimiento

Nota. Descripción individual de un requerimiento en la plataforma Jira.

3.3 PLANIFICACIÓN SPRINT

La planificación de un sprint es uno de los momentos más importantes, ya que se desea mantener un ritmo estable y equilibrado de trabajo, tomar decisiones sobre qué aspectos priorizar, como correcciones de errores o nuevas funciones.

Antes de cada sprint, se debe establecer el objetivo del sprint y crear un Sprint Backlog con las tareas necesarias para alcanzar ese objetivo. Esta planificación se desea lleva a cabo con la finalización de cada sprint (Eby, 2018).

3.2.1 SPRINT 1

Sprint 1: Configuración Inicial

Tiempo Estimado: 8 horas

Objetivo: Determinar la base del sistema y configurar las funciones esenciales mediante el uso de las herramientas mencionadas en los capítulos anteriores para poder definir el aumento o disminución de actividades en los otros Sprints.

Tareas incluidas: S1, S2 y S3.

3.2.2 SPRINT 2

Sprint 2: Funcionalidades Críticas

Tiempo Estimado: 10 horas

Objetivo: Desarrollar las funcionalidades críticas necesarias para el funcionamiento básico del sistema mediante la ayuda de los actores del proyecto.

Tareas incluidas: S4 y S5.

3.2.3 SPRINT 3

Sprint 3: Optimización y Mejoras

Tiempo Estimado: 5 horas

Objetivo: Mejorar la experiencia del usuario y realizar ajustes finales en el sistema.

Tareas incluidas: S6, S7 y correcciones.

CAPÍTULO IV - DESARROLLO DE LA APLICACIÓN

Este Capítulo IV del desarrollo de la aplicación describe el proceso completo de creación del "Sistema web prototipo para la gestión del servicio a la habitación y optimizar la facturación en la Hostería Vista Hermosa", utilizando la metodología SCRUM y KANBAN, organizado en tres sprints iterativos.

En el Sprint 1, se establecen las funcionalidades básicas del sistema, incluyendo la configuración inicial y la documentación. Utilizando un tablero Kanban, se monitoriza el progreso de las tareas, desde "Tareas por hacer" hasta "Finalizada". Los diagramas de casos de uso ilustran las interacciones clave entre todos los actores del sistema, mientras que el diseño abarca la interfaz de usuario. La codificación se enfoca en implementar estas funcionalidades utilizando las herramientas y lenguajes adecuados.

En el Sprint 2, se introducen mejoras y nuevas funcionalidades basadas en el feedback del Sprint 1. El Sprint 3 se centra en mejorar la experiencia del usuario y realizar ajustes finales. Los tableros Kanban continúan reflejando el avance de las tareas, mientras que los diagramas de casos de uso aseguran la integración completa de las funcionalidades, para finalmente obtener un producto funcional.

En el **Anexo F** podemos ver el diagrama de casos de uso general de la aplicación, donde interactúan los diferentes actores y en que procesos se encuentra. En el **Anexo G**, **Anexo H** se encuentra el link del repositorio de toda la aplicación.

4.1 **DESARROLLO SPRINT 1**

4.1.1 SPRINT 1 BACKLOG

Las Figuras **Figura 10**, **Figura 11** y **Figura 12** presentan el Sprint Backlog 1, generado en el entorno de Jira, desde su inicio hasta su finalización. Este sprint abarca el período de una semana, con un total de diez incidencias que deben ser abordadas, considerando que al ser el Sprint 1 se decidió dar las tareas de mayor prioridad.

La **Figura 10** muestra el estado inicial del Sprint Backlog 1, donde cada ítem está identificado por un código único (SCRUM-ID) y una breve descripción de la tarea. Todas las tareas están en estado “Tareas por hacer”. A continuación, se detalla la información de cada uno:

- SCRUM-1: Panel Administrador.
- SCRUM-8: Retraso en la configuración.
- SCRUM-12: Crear cuenta de administrador con permisos que se puedan dar.
- SCRUM-9: Creación de documentación para iniciar el entorno.
- SCRUM-13: Agregar pestañas teniendo en cuenta los demás Sprints.
- SCRUM-2: Panel Cliente.
- SCRUM-10: Interfaz amigable y fácil de usar.
- SCRUM-11: Conexión entre las dos partes (Administrador y Cliente).
- SCRUM-3: Gestión de Producto.
- SCRUM-14: Se debe asignar primero las categorías para seguir.

La **Figura 12** muestra el estado del Sprint Backlog 1 una vez finalizado. En esta etapa, todas las tareas han sido completadas. Esto se refleja en el cambio de estado de cada ítem a "Finalizada", indicando que todos los objetivos del sprint han sido alcanzados dentro del período establecido.

Figura 10

Sprint Backlog 1

Nota. Visualización del Sprint 1 con la asignación de cada requerimiento e incidencia.

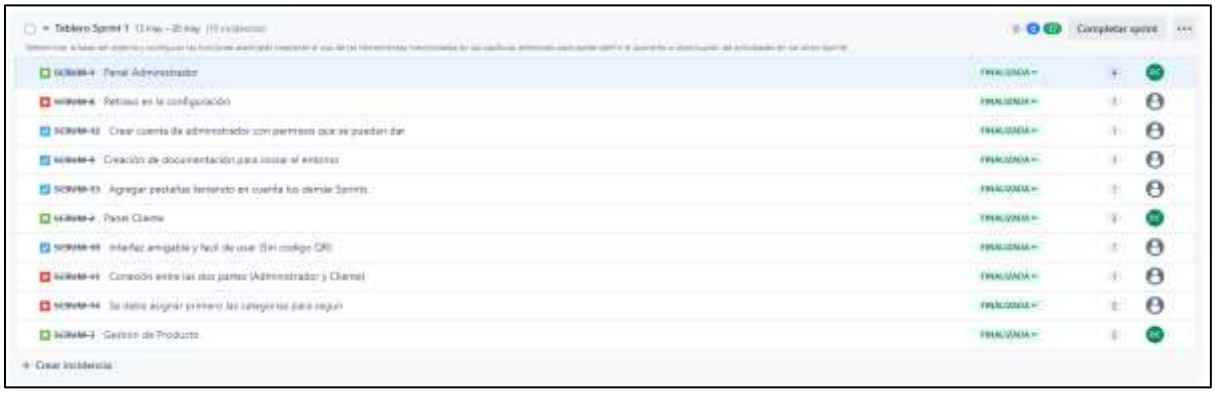
Figura 11

Sprint Backlog 1 Estado Avances

Nota. Avances del Sprint 1 con el estado de cada incidencia.

Figura 12

Sprint Backlog 1 Finalizado



Nota. Finalización del Sprint 1.

4.1.2 KANBAN BOARD 1

La **Figura 13, Figura 14 y Figura 15** muestra las iteraciones del Kanban Board para el Sprint 1, organizado en tres columnas: “Por Hacer”, “En Curso” y “Listo”. Esta visualización permite gestionar y visualizar el flujo de trabajo de manera eficiente durante el sprint. En la **Figura 13** nos ayuda a visualizar las tareas prioritarias, ya que son las están “En Curso”.

La **Figura 14** muestra el Kanban Board actualizado para el proyecto, destacando las tareas que han alcanzado la fase de finalización. Esta vista incluye las tareas completadas y permite visualizar el progreso general del proyecto, dentro de las tareas completadas tenemos: SCRUM-1,

SCRUM-8, SCRUM-12, SCRUM-9 y SCRUM-13. Finalmente, en la **Figura 15** tenemos que todas las tareas fueron finalizadas.

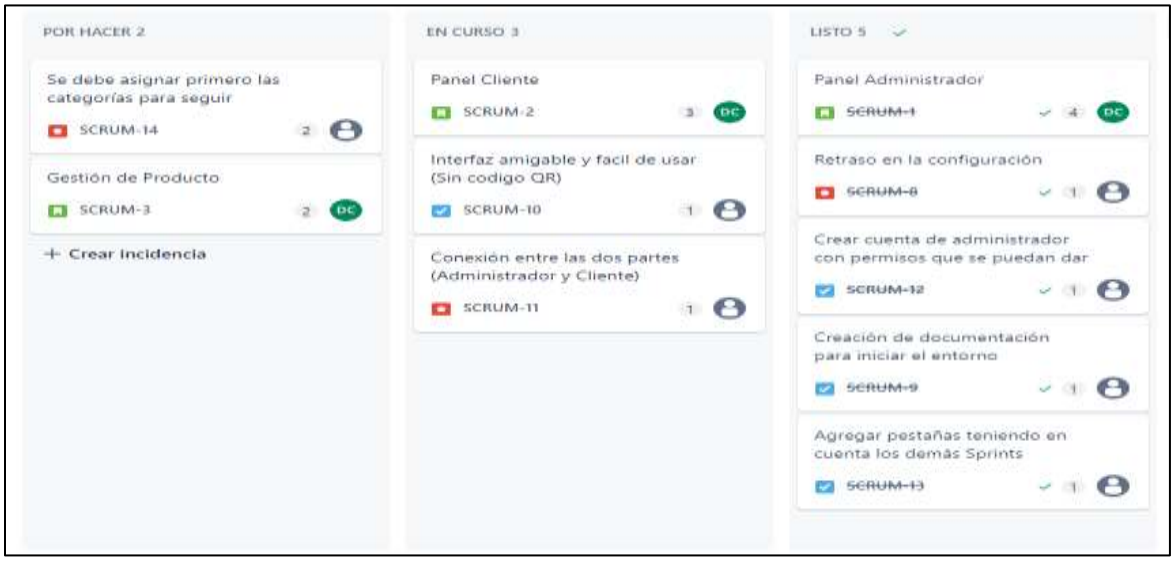
Figura 13

Kanban Board Iteración 1.1

Nota. Primera iteración del Kanban Board con las acciones “Por Hacer”, “En Curso” y “Listo”.

Figura 14

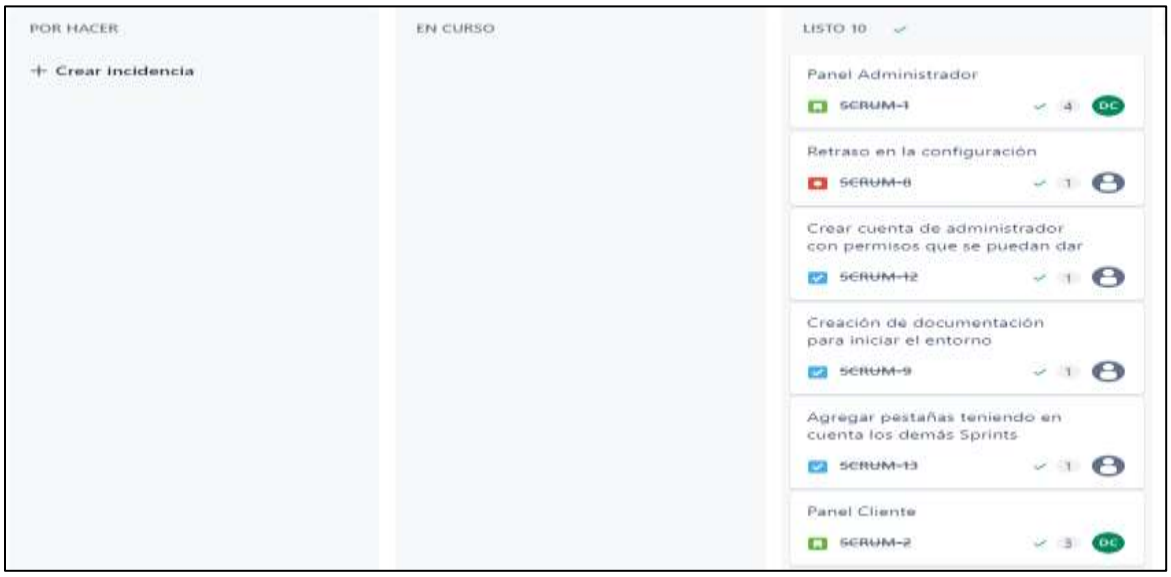
Kanban Board Iteración 1.2



Nota. Kanban Board con los avances en curso e iteraciones finalizadas.

Figura 15

Kanban Board Iteración 1.3



Nota. Kanban Board con la finalización de todas las iteraciones.

4.1.3 DIAGRAMAS CASOS DE USOS

La **Figura 16** es un diagrama de casos de uso. A continuación, se describen los elementos presentes en el diagrama:

Actores que interactúan:

- Gerente, Administrador y Empleado

Casos de Uso:

- Ver Productos
- Eliminar Productos
- Editar Productos
- Incluye el caso de uso “Cambiar Imagen”
- Crear Productos
- Incluye el caso de uso “Seleccionar Categoría”
- Incluye el caso de uso “Producto Activo”

Relaciones:

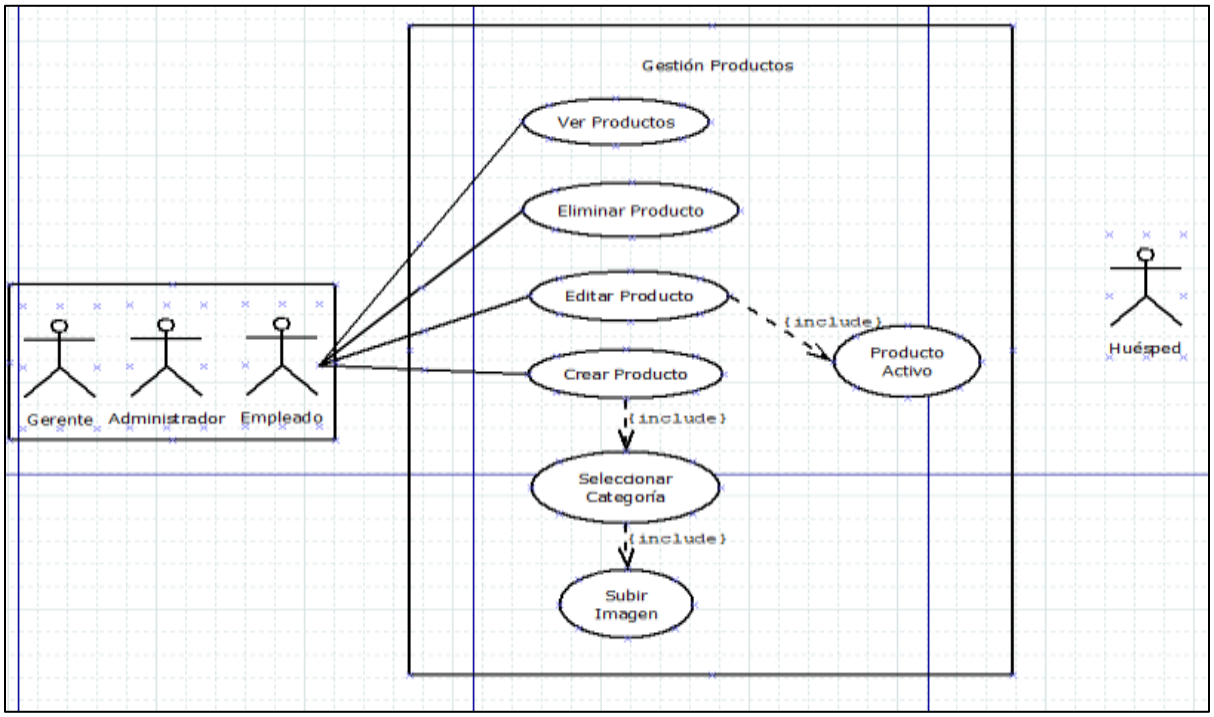
- Los actores “Gerente”, “Administrador” y “Empleado” están agrupados en una relación de generalización con una línea que apunta a una relación común de “Gestión Productos”

(CRUD).

- El caso de uso “Producto Activo” es incluido en el caso de uso “Editar Producto”.
- El caso de uso “Seleccionar Categoría” y “Subir Imagen” es incluido en el caso de uso “Crear Producto”.

Figura 16

Diagrama Casos de Uso Gestión Productos



Nota. La figura muestra el diagrama de caso de uso de productos realizado en el entorno “Dia”.

La **Figura 17** es un diagrama de casos de uso. A continuación, se describen los elementos presentes en el diagrama:

Actores que interactúan:

- Gerente, Administrador y Empleado

Casos de Uso:

- Ver Usuarios
- Eliminar Usuarios
- Editar Usuarios
- Crear Usuarios

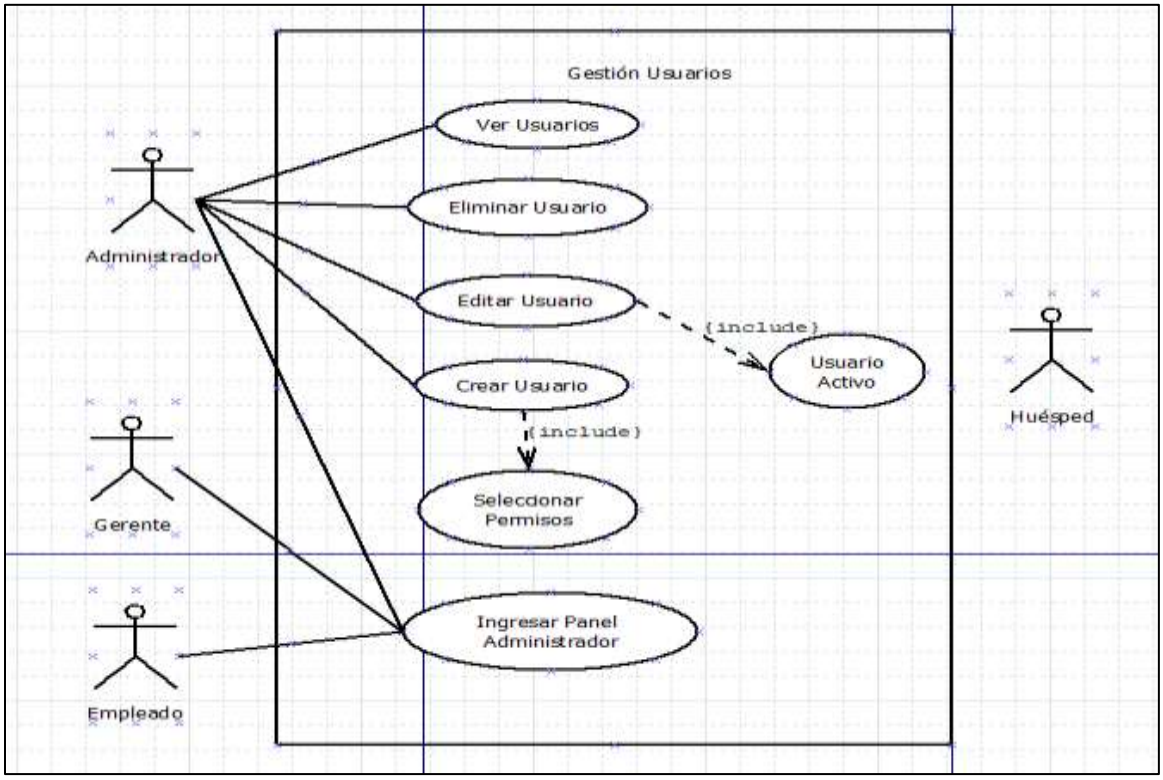
- Incluye el caso de uso “Seleccionar Permisos”
- Incluye el caso de uso “Usuario Activo”
- Ingresar al Panel administrador

Relaciones:

- Los actores “Gerente”, “Administrador” y “Empleado” están agrupados en una relación de generalización con una línea que apunta a una relación común de “Ingresar al Panel Administrador”.
- El actor administrador está agrupado con la relación de generalización que apunta a la “Gestión de Usuarios” (CRUD).
- El caso de uso “Seleccionar Permisos” es incluido en el caso de uso “Crear Usuario”.
- El caso de uso “Usuario Activo” es incluido en el caso de uso “Editar Usuario”.

Figura 17

Diagrama Casos de Uso Gestión Usuarios



Nota. La figura muestra el diagrama de caso de uso de usuarios realizado en el entorno “Dia”.

4.1.4 DISEÑO S1

La **Figura 18** muestra la interfaz de gestión de usuarios del sistema "Hostería Vista Hermosa". Esta pantalla admite la creación de nuevos usuarios y la asignación de permisos. En la tabla se presentan los siguientes campos para cada usuario:

- Username: Nombre de usuario
- Email: Correo electrónico
- Nombre: Primer nombre
- Apellido: Apellido
- Activo: Indica si el usuario está activo o no
- Staff: Indica si el usuario es parte del personal o no

Cada fila de la tabla incluye botones de acciones que permiten actualizar y eliminar usuarios. Además, se encuentra disponible un botón de "Nuevo usuario" que ayuda la adición de nuevos usuarios al sistema.

Figura 18

Interfaz Gestión Usuarios

Nota. La figura muestra el sistema realizado.

La **Figura 19** ilustra la interfaz de gestión de productos, también del sistema "Hostería Vista Hermosa". Esta pantalla permite la administración de los productos ofrecidos, mostrando una tabla con los siguientes campos:

- Imagen: Foto del producto
- Producto: Nombre del producto
- Precio: Costo del producto

- Categoría: Clasificación del producto
- Activo: Indica si el producto está disponible

Al igual que en la gestión de usuarios, cada producto puede ser actualizado o eliminado. Estas interfaces son parte integral del sistema, permitiendo una administración eficiente.

Figura 19

Interfaz Gestión Productos

Nota. La figura muestra el sistema realizado.

4.1.5 CODIFICACIÓN S1

La codificación en el Sprint 1 se enfoca en implementar las funcionalidades básicas del sistema, asegurando que las tareas prioritarias sean operativas. Se muestran los lenguajes y herramientas utilizados en la sección **Anexo B**, **Anexo D** y **Anexo I**.

El **Anexo I** muestra la distribución de las páginas del sistema y las partes cruciales del código que forman parte del Sprint 1, podemos observar: Código de gestión de usuarios y de productos.

4.2 DESARROLLO SPRINT 2

4.2.1 SPRINT 2 BACKLOG

Las **Figura 20** y **Figura 21** presentan el Sprint Backlog 2, generado en el entorno de Jira, desde el inicio hasta su fin. Este sprint abarca el período de una semana, con un total de cinco incidencias que deben ser abordadas, considerando que al ser el Sprint 2 se decidió dar continuidad a las tareas que requieren mejorar la funcionalidad del sistema.

La **Figura 20** muestra el estado inicial del Sprint Backlog 2, donde cada ítem está identificado por un código y una breve descripción. A continuación, se detalla la información de cada uno:

- SCRUM-4: Gestión de Categoría.
- SCRUM-16: Poder unir todo con la sección de productos.
- SCRUM-5: Historial de Pagos General.
- SCRUM-15: Poder visualizar el detalle de cada pedido.
- SCRUM-17: Quitar botón de eliminación y actualización de historial de pago terminado.

La **Figura 21** muestra el estado del Sprint Backlog 2 una vez finalizado. En esta etapa, todas las tareas han sido completadas. Esto se refleja en el cambio de estado de cada ítem a "Finalizada", indicando que todos los objetivos del sprint han sido alcanzados dentro del período establecido.

Figura 20

Sprint Backlog 2

Nota. Visualización del Sprint 2 con los errores, incidencias e historias de usuario.

Figura 21

Sprint Backlog 2 Finalizado

Nota. Finalización del Sprint 2

4.2.2 KANBAN BOARD 2

La **Figura 22** y **Figura 23** muestra las iteraciones del Kanban Board para el Sprint 2, desde el inicio del Sprint hasta su finalización. La **Figura 22** muestra el Kanban Board actualizado para el proyecto, destacando una tarea finalizada, la cual está relacionada con el Sprint anterior. Esta vista permite visualizar el progreso general del proyecto, la única tarea completada en el momento es: SCRUM-16. Finalmente, en la **Figura 23** tenemos que todas las tareas fueron finalizadas, las cuales podemos observar cómo: SCRUM-5, SCRUM-15, SCRUM-17, SCRUM-4 y SCRUM-16.

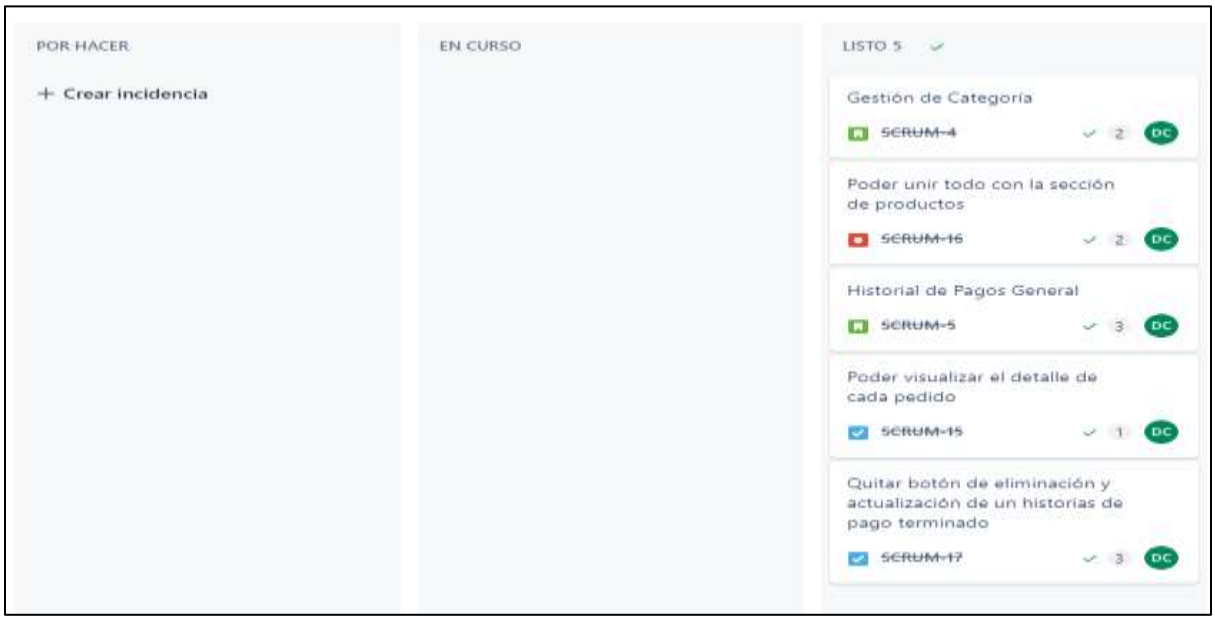
Figura 22

Kanban Board Iteración 2.1

Nota. Segunda iteración del Kanban Board con todas acciones.

Figura 23

Kanban Board Iteración 2.2



Nota. Kanban Board con la finalización de todas las iteraciones.

4.2.3 DIAGRAMAS CASOS DE USOS

La **Figura 24** es un diagrama de casos de uso. A continuación, se describen los elementos presentes en el diagrama:

Actores que interactúan:

- Gerente, Administrador y Empleado

Casos de Uso:

- Ver Categoría
- Eliminar Categoría
- Editar Categoría
- Crear Categoría
- Incluye el caso de uso “Cambiar Imagen”

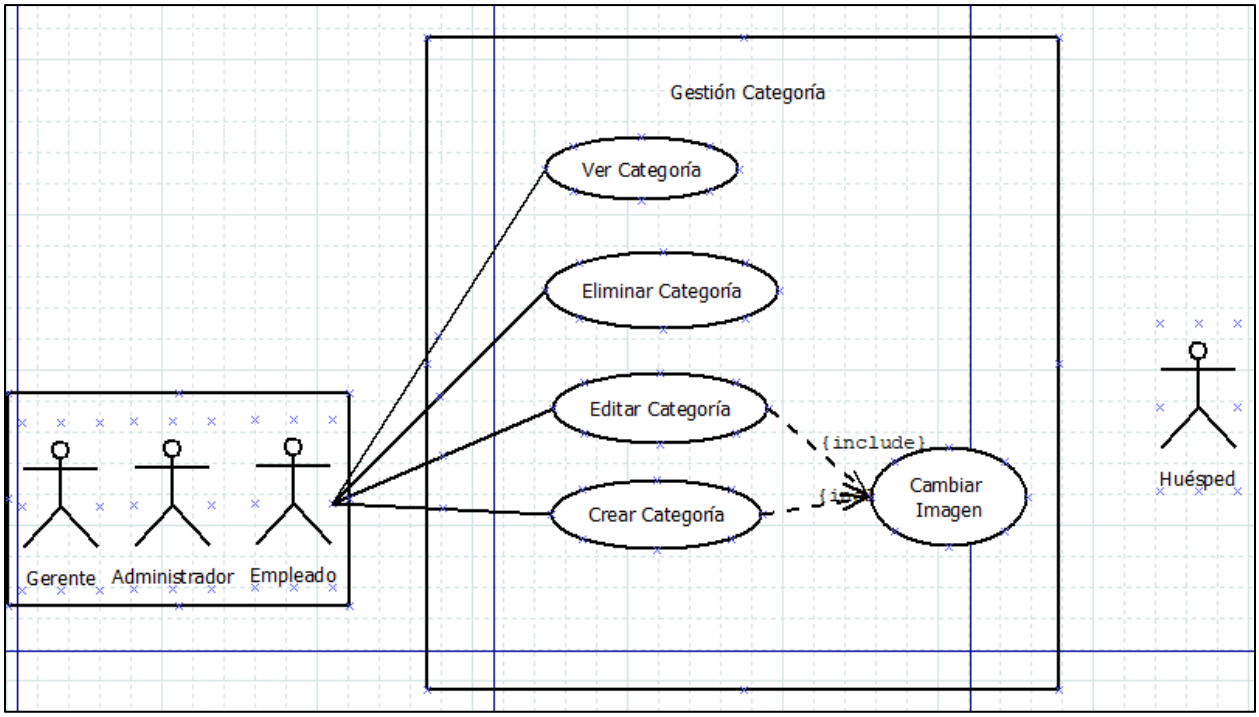
Relaciones:

- Los actores “Gerente”, “Administrador” y “Empleado” están agrupados en una relación de generalización con una línea de relación común de “Gestión Categoría”(CRUD).

- El caso de uso “Cambiar Imagen” es incluido por los casos de uso “Editar Categoría” y “Crear Categoría” .

Figura 24

Diagrama Casos de Uso Gestión Categoría



Nota. La figura muestra el diagrama de caso de uso de categoría realizado en el entorno “Dia”.

La **Figura 25** es un diagrama de casos de uso. A continuación, se describen los elementos presentes en el diagrama:

Actores que interactúan:

- Gerente, Administrador, Empleado y Huésped

Casos de Uso:

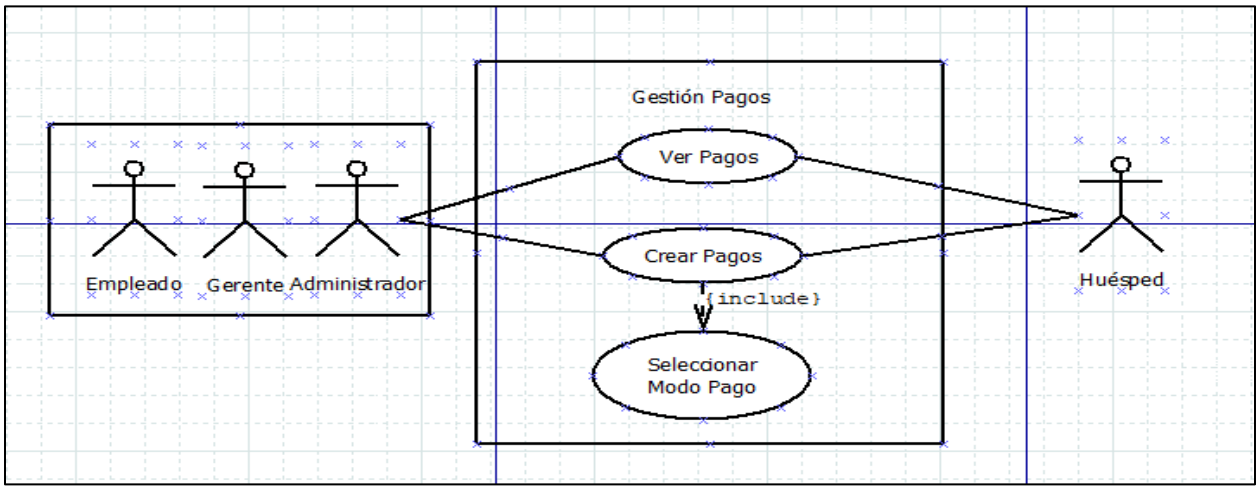
- Ver Pagos
- Crear Pagos
- Incluye el caso de uso “Seleccionar Modo Pago”

Relaciones:

- Los actores “Gerente”, “Administrador”, “Huésped” y “Empleado” cuenta con una línea de relación común de “Ver Pagos” y “Crear Pagos”.

Figura 25

Diagrama Casos de Uso Gestión Pagos



Nota. La figura muestra el diagrama de caso de uso de pagos realizado en el entorno “Dia”.

4.2.4 DISEÑO S2

La **Figura 26** ilustra la interfaz de gestión de categorías del sistema "Hostería Vista Hermosa". Esta pantalla permite la administración de las categorías que se deben asignar a cada producto, se muestra una tabla con los siguientes campos:

- Imagen: Foto de la categoría
- Categoría: Nombre de la categoría

Al igual que las anteriores interfaces, de gestión de usuarios y producto, las categorías pueden ser actualizadas o eliminadas mediante los íconos. También se incluye un botón verde "Nuevo Categoría" para agregar categorías nuevas al sistema.

Figura 26

Interfaz Gestión Categoría

Nota. La figura muestra el sistema realizado.

La **Figura 27** muestra la interfaz del historial de pagos del sistema "Hostería Vista Hermosa". Esta pantalla permite al administrador ver los pagos realizados, se muestra una tabla con los siguientes campos:

- ID: Identificación pago
- Total: Monto total
- Tipo de pago: Escoger forma de pago

Esta interfaz de historial de pagos solamente puede ver los pagos, no sé puede realizar una creación, actualización, ni eliminación de estos.

Figura 27

Interfaz Historial De Pagos

Nota. La figura muestra el sistema realizado.

4.2.5 CODIFICACIÓN S2

La codificación en el Sprint 2 se centra en implementar las nuevas funcionalidades, mejoras identificadas y corrección de errores del anterior Sprint. El **Anexo I** tiene el código que forman parte del Sprint 2, podemos observar: Código de gestión de categoría y de historial de pagos.

4.3 DESARROLLO SPRINT 3

4.3.1 SPRINT 3 BACKLOG

Las **Figura 28** y **Figura 29** presenta el Sprint Backlog 3, generado en el entorno de Jira, desde su inicio hasta su finalización. Este sprint abarca el período del 26 de mayo al 30 de mayo, con un total de cuatro incidencias que deben ser abordadas, considerando que al ser el Sprint 3 se decidió enfocar en mejorar la experiencia del usuario y realizar ajustes finales en el sistema.

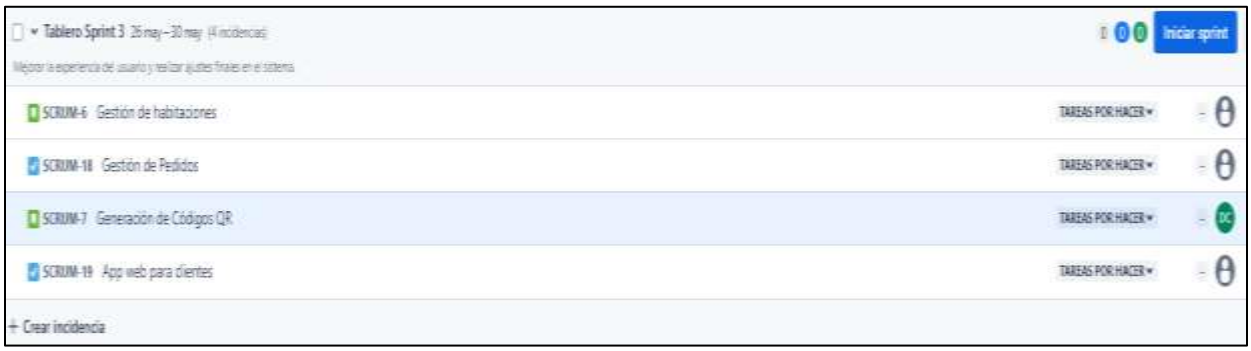
La **Figura 28** muestra el estado inicial del Sprint Backlog 3, donde cada ítem está identificado por un código único (SCRUM-ID) y una breve descripción de la tarea. Todas las tareas están en estado “Tareas por hacer”. A continuación, se detalla la información de cada uno:

- SCRUM-6: Gestión de habitaciones
- SCRUM-18: Gestión de Pedidos
- SCRUM-7: Generación de Códigos QR
- SCRUM-19: App web para clientes

La **Figura 29** muestra el estado del Sprint Backlog 3 una vez finalizado. En esta etapa, todas las tareas han sido completadas.

Figura 28

Sprint Backlog 3



Nota. Visualización del Sprint 3 con las incidencias e historias de usuario.

Figura 29

Sprint Backlog 3 Finalizado

Nota. Finalización del Sprint 3

4.3.2 KANBAN BOARD 3

La **Figura 30** y **Figura 31** las iteraciones del Kanban Board para el Sprint 3, desde el inicio del Sprint hasta su finalización. La **Figura 30** muestra el Kanban Board inicial para el proyecto, destacando a todas las tareas "En Curso", las cuales está relacionada con el Sprint 2. Esta vista permite visualizar el progreso general del proyecto. Finalmente, en la **Figura 31** tenemos que todas las tareas fueron finalizadas, las cuales podemos observar cómo: SCRUM-6, SCRUM-18, SCRUM-7 y SCRUM-19.

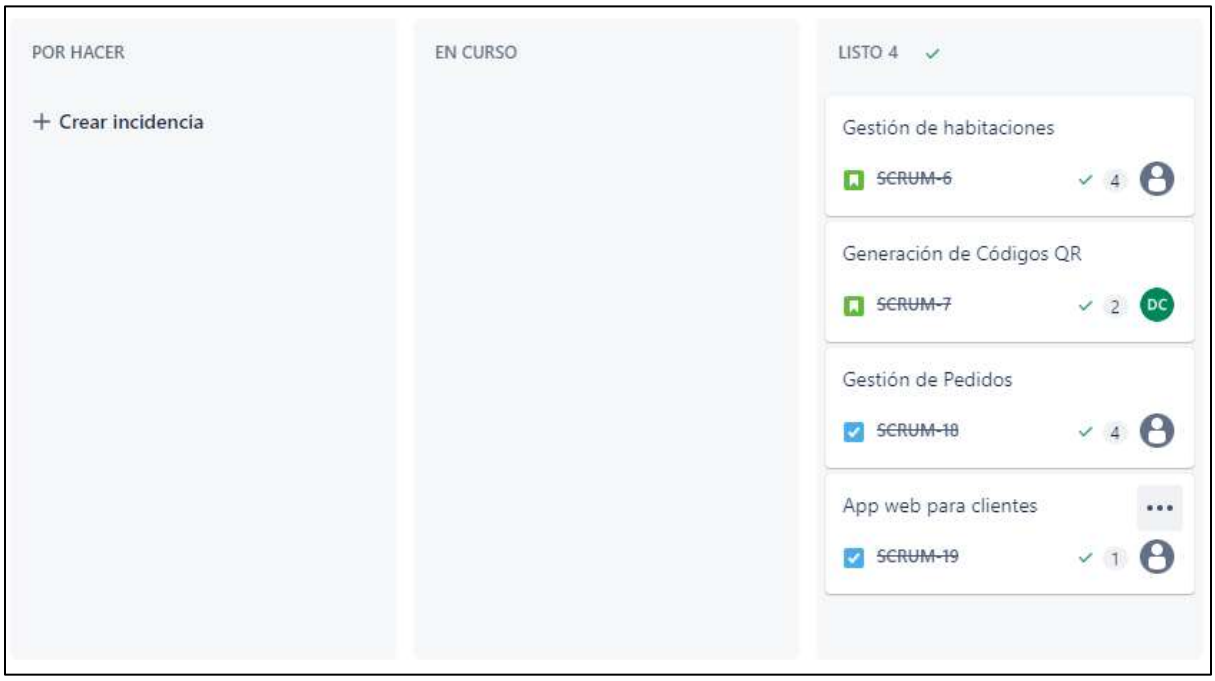
Figura 30

Kanban Board Iteración 3.1

Nota. Tercera iteración del Kanban Board con todas acciones "En Curso".

Figura 31

Kanban Board Iteración 3.2



Nota. Kanban Board con la finalización de todas las iteraciones.

4.3.3 DIAGRAMAS CASOS DE USOS

La **Figura 32** es un diagrama de casos de uso. A continuación, se describen los elementos presentes en el diagrama:

Actores que interactúan:

- Gerente, Administrador y Empleado

Casos de Uso:

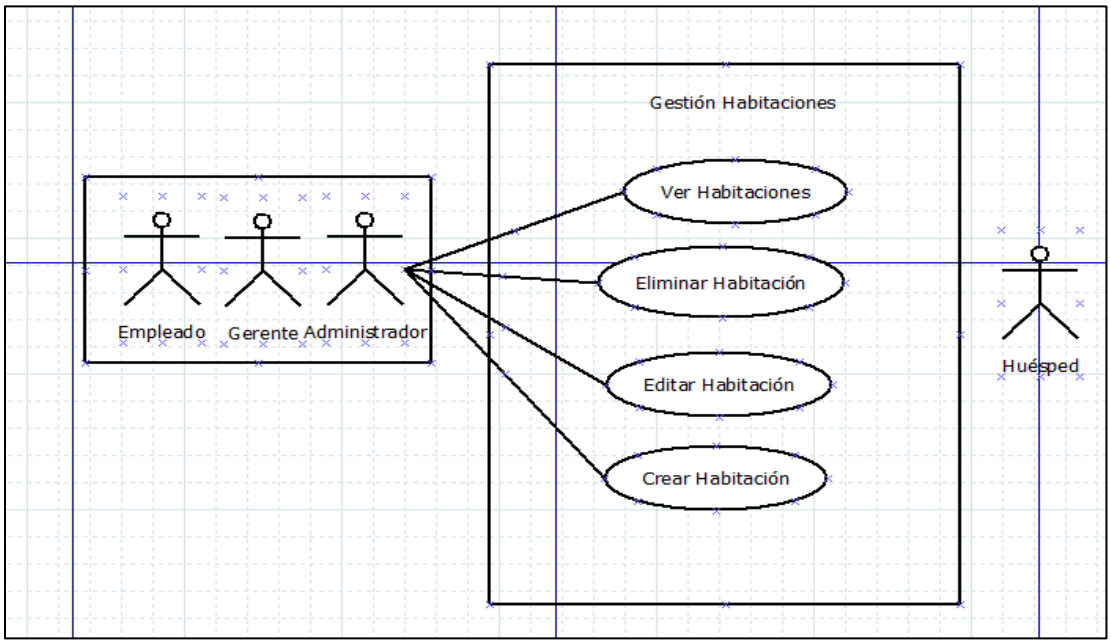
- Ver Habitaciones
- Eliminar Habitaciones
- Editar Habitaciones
- Crear Habitaciones

Relaciones:

- Los actores “Gerente”, “Administrador” y “Empleado” están agrupados en una relación de generalización con una línea de relación común de “Gestión Habitaciones”(CRUD).

Figura 32

Diagrama Casos de Uso Gestión Habitaciones



Nota. La figura muestra el diagrama de caso de uso de habitaciones realizado en el entorno "Dia".

La **Figura 33** es un diagrama de casos de uso. A continuación, se describen los elementos presentes en el diagrama:

Actores que interactúan:

- Gerente, Administrador, Empleado y Huésped

Casos de Uso:

- Ver Pedidos
- Eliminar Pedidos
- Editar Pedidos
- Crear Pedidos
- Incluye el caso de uso "Seleccionar Categoría"
- Incluye el caso de uso "Seleccionar Producto"
- Incluye el caso de uso "Estado Pedido"

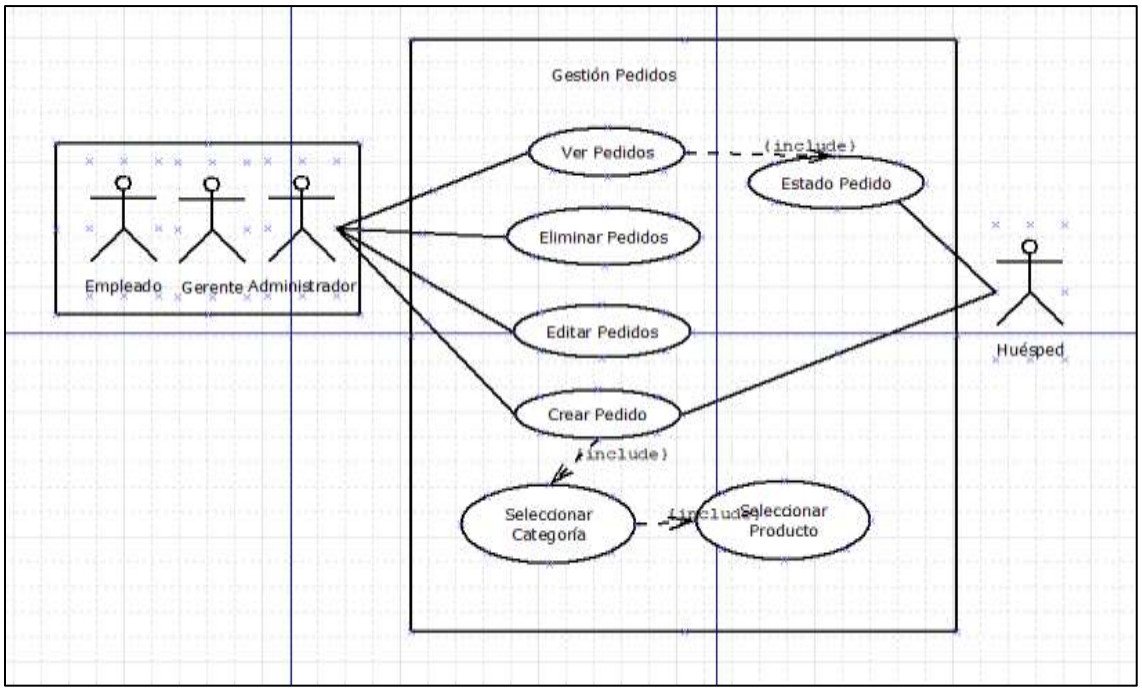
Relaciones:

- Los actores "Gerente", "Administrador" y "Empleado" están agrupados en una relación de generalización con una línea que apunta a una relación común de "Gestión Pedidos".

- El actor “Huésped” está agrupado con la relación de generalización que apunta a “Crear Pedido” y ver “Estado Pedido”.
- El caso de uso “Seleccionar Categoría” es incluido en el caso de uso “Crear Pedido”.
- El caso de uso “Seleccionar Producto” es incluido en el caso de uso “Crear Pedido”.

Figura 33

Diagrama Casos de Uso Gestión Pedidos



Nota. La figura muestra el diagrama de caso de uso de pedido realizado en el entorno “Dia”.

4.3.4 DISEÑO S3

La **Figura 34** ilustra la interfaz de gestión de habitaciones del sistema "Hostería Vista Hermosa". Esta pantalla permite la administración de las habitaciones, mostrando una tabla con el siguiente campo:

- Habitación: Número de habitación

Al igual que en las anteriores páginas, cada habitación puede ser actualizada o eliminada. También se incluye un botón verde "Crear Nueva Habitación" para agregar habitaciones adicionales al sistema.

Esta interfaz es una parte importante del sistema, ya que es en donde se debe asignar los pedidos que realicen los clientes de la hostería.

Figura 34

Interfaz Gestión Habitaciones

Nota. La figura muestra el sistema realizado.

La **Figura 35** presenta la interfaz de gestión de pedidos del sistema "Hostería Vista Hermosa". En esta pantalla se puede visualizar el estado de las habitaciones y gestionar los pedidos asignados a cada una de ellas. La distribución y los colores de las habitaciones indican su estado actual:

- Habitaciones con número: Indican que tienen pedidos asignados.
- Habitaciones en color negro: Indican que están disponibles.
- Habitaciones en color rojo: Indican que están ocupadas.
- Habitaciones con cuenta: Indican habitaciones que pidieron la cuenta.

En la parte superior derecha de la interfaz, se encuentra la opción de "Reload automático" que se puede activar o desactivar, junto a un botón azul de recargar manualmente la página.

En el menú lateral izquierdo, se puede navegar a otras secciones del sistema, tales como Pedidos, Habitaciones, Historial de pagos, Categorías, Productos y Usuarios.

La interfaz está diseñada para facilitar la gestión eficiente de los pedidos, permitiendo al usuario ver rápidamente el estado de las habitaciones y actuar en consecuencia.

Figura 35

Interfaz Gestión Pedidos

Nota. La figura muestra el sistema realizado.

4.3.5 CODIFICACIÓN S3

La codificación en el Sprint 3 finaliza el desarrollo de la aplicación, implementando todas las funcionalidades y realizando ajustes finales. Se describen las técnicas y herramientas utilizadas para completar el proyecto.

El **Anexo I** tiene código perteneciente al Sprint 3, podemos observar: Código de gestión de habitaciones y de pedidos.

4.4 PRUEBAS DE ACEPTACIÓN DEL SISTEMA

Las pruebas de aceptación son esenciales para asegurar que el software desarrollado cumple con las expectativas y necesidades de los usuarios finales y clientes. Son una parte crucial para detectar discrepancias entre el software y lo esperado. Estas pruebas se integran en el proceso, ayudando a verificar que el software cumple con los criterios de aceptación desde el inicio hasta el final, lo que minimiza sorpresas desagradables y asegura que el producto esté siempre en un estado aceptable (Venema, 2024).

Existen varios tipos de pruebas de aceptación, las cuales intentan verificar que el software satisface las necesidades del usuario final, evalúan la ejecución y gestión eficiente del software y aseguran el cumplimiento de estándares y regulaciones aplicables. Estas pruebas son fundamentales para garantizar que el software no solo cumple con los requisitos funcionales, sino que también es seguro, confiable y apto para su uso en un entorno de producción (Venema, 2024).

La **Tabla 9** muestra las pruebas que se realizaron en los requerimientos del Sprint 1, cuyo objetivo es mejorar el sistema y dar a conocer al cliente el estado del software. A continuación, se presenta una descripción de cada campo de la tabla de pruebas:

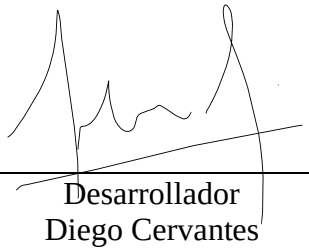
- Requerimiento Funcional: Un identificador único para cada prueba de aceptación.
- Nombre de Funcionalidad: Una breve descripción de la funcionalidad o característica del sistema que se está probando.
- Status: El resultado actual de la prueba, que puede ser "Aprobado" o "Fallido". Esto ayuda a identificar el progreso y las áreas que necesitan atención adicional.
- Observaciones: Se pone cualquier comentario del cliente o desarrollador.

Tabla 9
Pruebas Aceptación Sprint 1

Requerimiento Funcional	**Nombre de la Funcionalidad**	**Status**	**Observaciones**
SPRINT 1 (13 May – 20 May)			
RF 1	Crear Nueva Cuenta Usuario	✓	Se debe crear permisos de los usuarios y que se pueda activar o desactivar un usuario.
RF 2	Editar Cuenta Usuario	✓	Al momento de editar se desea que se pueda quitar permisos o activar la cuenta del usuario.
RF 3	Ver Cuentas Existentes	✓	Se debe ver las cuentas que tenemos en una tabla.
RF 4	Eliminar Cuenta	✓	Solo el panel de administrador puede eliminar una cuenta.
RF 5	Conexión entre las dos partes (Administrador y Cliente)	✓	Se debe tener en cuenta que solo los empleados y administradores van a tener permisos y huéspedes no van a tener cuenta.
RF 6	Crear Productos	✓	Se tiene que escoger una categoría al crear un producto.
RF 7	Ver Productos	✓	Debemos poder ver los productos de forma ordenada y con una foto.
RF 8	Eliminar Productos	✓	Cualquier usuario con permisos puede eliminar un producto.
RF 9	Editar Productos	✓	Agregar funcionalidad si el producto está activo o no.
RF 10	Inicio Sesión Panel Administrador	✓	Ninguno.
RF 11	Cerrar Sesión	✓	Ninguno.
RF 12	Navegación Entre Pantallas	X	El panel administrador debe navegar por cada pantalla de manera libre y sin errores.
Desarrollador Diego Cervantes			Copropietario Hostería Vista Hermosa Dora Avilés Castillo

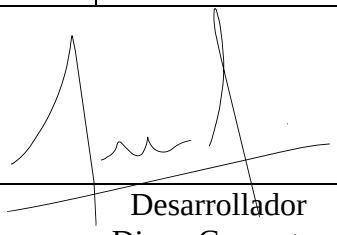
Nota. La tabla muestra las pruebas de aceptación del sistema. Elaboración propia.

Tabla 10
Pruebas de Aceptación Sprint 2

Requerimiento Funcional	**Nombre de la Funcionalidad**	**Status**	**Observaciones**
SPRINT 2 (20 May – 25 May)			
RF 13	Crear Nueva Categoría	✓	Se debe crear categorías nuevas con los permisos de administrador.
RF 14	Editar Categoría	✓	Al momento de editar se desea que igual se actualice la tabla de productos con los datos actualizados.
RF 15	Ver Categorías Existentes	✓	Se debe ver todas las categorías que tenemos en una tabla.
RF 16	Eliminar Categoría	✓	Solo personas con permisos pueden eliminar una categoría.
RF 17	Crear Pagos	✓	El sistema debe poder poner el estado para pagar una cuenta
RF 18	Ver Pagos	✓	Se debe poder ver los pagos totales realizados.
			
Desarrollador Diego Cervantes			Copropietario Hostería Vista Hermosa Dora Avilés Castillo

Nota. La tabla muestra las pruebas de aceptación del sistema. Elaboración propia.

Tabla 11
Pruebas de Aceptación Sprint 3

Requerimiento Funcional	**Nombre de la Funcionalidad**	**Status**	**Observaciones**
SPRINT 3 (25 May – 31 May)			
RF 19	Crear Nueva Habitación	✓	Para que cada cliente pueda ordenar se tiene que crear una habitación.
RF 2	Editar Habitación	✓	Ninguno.
RF 3	Ver Habitación Existentes	✓	Se debe ver las habitaciones que tenemos en una tabla.
RF 4	Eliminar Habitación	✓	Solo el panel de administrador puede eliminar.
RF 6	Crear Pedidos	✓	Se tiene que poder crear pedidos de la parte de administrador y del cliente.
RF 7	Ver Pedidos	✓	Se debe poder ver el estado de cada pedido.
RF 8	Eliminar Pedidos	✓	Ninguno.
RF 9	Editar Pedidos	✓	Ninguno
			
Desarrollador Diego Cervantes		Copropietario Hostería Vista Hermosa Dora Avilés Castillo	

Nota. La tabla muestra las pruebas de aceptación del sistema. Elaboración propia.

CAPÍTULO V - CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- La realización de entrevistas con las personas que van a utilizar el sistema me permitió analizar de manera más completa el proceso de gestión del servicio a la habitación en la Hostería Vista Hermosa. Este enfoque no solo facilitó una comprensión integral de las operaciones actuales, sino que también me ayudó a identificar y clarificar los requisitos esenciales del sistema. Gracias a estas entrevistas, pude asegurarme de que el diseño y desarrollo del sistema respondieran adecuadamente a las necesidades y expectativas reales de los usuarios, mejorando así su efectividad y usabilidad.
- La metodología Scrum y Kanban utilizada para el desarrollo de la aplicación permitió una mejor gestión de las tareas, ya que nos aseguró una entrega incremental en cada iteración. La estructura en sprints y el Kanban Board facilitó la identificación de prioridades y la adaptación continua a incidencias del proyecto, asegurando que se alcanzaran los objetivos dentro de los plazos establecidos.
- La creación de documentación técnica es un componente esencial en el desarrollo del sistema de gestión del servicio a la habitación para la Hostería Vista Hermosa. Una documentación completa y bien estructurada proporciona múltiples beneficios que son cruciales para el éxito y la sostenibilidad del sistema a largo plazo, ya que nos ayuda a obtener un entendimiento claro del funcionamiento del sistema.
- Se logró diseñar una interfaz que no solo cumple con todos los requisitos funcionales, sino que también es amigable y fácil de utilizar. Este enfoque garantiza que los usuarios puedan interactuar con el sistema de manera eficiente y efectiva, mejorando su experiencia general y facilitando la gestión del servicio a la habitación en la Hostería Vista Hermosa.

5.2 RECOMENDACIONES

- Al utilizar la metodología Scrum, es crucial mantener un ciclo de retroalimentación constante al finalizar cada sprint. Esto debe involucrar a todos los interesados del proyecto, incluyendo usuarios finales, directores del proyecto y desarrolladores. Este enfoque asegurará que el sistema se mantenga alineado con las expectativas y necesidades reales, permitiendo obtener el mejor producto final.
- Al analizar un tema para un plan de tesis, es crucial evaluar su viabilidad en relación con el tiempo disponible. Es esencial seleccionar un tema que se pueda abordar y completar dentro del plazo establecido para asegurar un desarrollo eficaz y exitoso del proyecto.
- Al momento de escoger una metodología ágil para el desarrollo de un sistema de información, es fundamental considerar la adecuación de dicha metodología en relación con aspectos clave como la documentación, el seguimiento y la comunicación. Para asegurar que la metodología seleccionada facilite un desarrollo eficiente y efectivo.
- Es fundamental que aprendas a utilizar el Kanban Board en tus proyectos, ya que permite mantener un seguimiento claro y constante del estado de las tareas, tanto si están en proceso como si han sido completadas. La ventaja de esta visibilidad detallada es que se puede enfocar en las tareas pendientes, lo cual ayuda a optimizar la gestión del flujo de trabajo y garantizar una ejecución eficiente de todas las actividades del proyecto.
- Confío utilizar React.js en el desarrollo de aplicaciones web debido a su capacidad para crear interfaces de usuario eficientes y altamente mantenibles. Al emplear React.js, es crucial dividir tu aplicación en componentes pequeños y reutilizables, ya que eso ayudará a tener mejores tiempos de respuestas.

BIBLIOGRAFÍA

Arnautović, A. (2022). *Managing project using JIRA software*. Retrieved from Serbian Journal of Engineering Management: https://doi.org/10.5937/SJEM2202040A

AWS. (2023). *Front End frente a back-end: Diferencia entre el desarrollo de aplicaciones* . Retrieved from Amazon Web Services, Inc. : https://aws.amazon.com/es/compare/the-difference-between-frontend-and-backend/

Bermejo, M. (2011). *El Kanban.* Barcelona, España: UOC.

Canive, T. (2020, Mayo 27). *Metodología XP o Programación Extrema: ¿Qué es y cómo aplicarla?* . Retrieved from Sinnaps: https://www.sinnaps.com/blog-gestion-proyectos/metodologia-xp

Deyimar, A. (2023, junio 29). *Qué es React: definición, características y funcionamiento*. Retrieved from Hostinger: https://www.hostinger.es/tutoriales/que-es-react

Díaz, J. S., & Medina, K. S. (2020). Herramientas de software libre para la creación de contenidos educativos. Ingeniare. Retrieved from https://doi.org/10.18041/1909-2458/ingeniare.28.6118

Eby, K. (2018, Junio 1). *Planificación exitosa de sprints: prácticas recomendadas, listas de verificación y plantillas*. Retrieved from smartsheet: https://es.smartsheet.com/sprint-planning

Etecé, E. (2023). *Base de datos*. Retrieved from concepto: https://concepto.de/base-de-datos/

Figueroa, R. G., Solís, C. J., & Cabrera, A. A. (2008). *Metodologías tradicionales vs. metodologías ágiles.* Loja: Universidad Técnica Particular de Loja, Escuela de Ciencias de la Computación.

Hunabku.mx. (s.f.). *React, una herramienta para el desarrollo web*. Retrieved from Hunabku: https://hunabku.mx/react-una-herramienta-para-el-desarrollo-web

Islam, A. K., & Ferworn, D. A. (2020). *A Comparison between Agile and Traditional Software Development Methodologies*. Retrieved from Global Journal of Computer Science and Technology: https://doi.org/10.34257/GJCSTCVOL20IS2PG7

Jayaraman, P., & Whittle, J. (2007). *UCSIM: A tool for simulating use case scenarios.* Retrieved from https://doi.org/10.1109/ICSECOMPANION.2007.80

Mercado, P., & Elizabeth, C. (2022). *Seguimiento y evaluación de la metodología de procesos en la gestión de requerimientos con herramienta JIRA.* Retrieved from bdigital: http://bdigital.dgse.uaa.mx:8080/xmlui/handle/11317/2327

Molina, D. (2023, Junio 27). *Qué es un product backlog y cómo hacer uno [Guía Scrum]*. Retrieved from IEBS: https://www.iebschool.com/blog/que-es-un-product-backlog-y-como-hacer-uno-guia-scrum-agile-scrum/

Navarro Cadavid, A. N. (2013, 09 20). *Revisión de metodologías ágiles para el desarrollo de software*. Retrieved from Prospectiva: https://doi.org/10.15665/rp.v11i2.36

Parks, R. F., & Hall, C. (2016). *Front-End and Back-End Database Design and Development: Scholar's Academy Case Study.* . Information Systems Education Journal.

Ramírez, C., Pintado, D., & Contreras, E. (2023, 01 09). *Elementos básicos de una base de datos*. Retrieved from UMAEE: https://www.umaee.edu.mx/blog/elementos-basicos-de-una-base-de-datos

SAC. (s.f). *Metodología tradicional vs. Metodología scrum: ¿cuál es la mejor opción para tu empresa*. Retrieved from SAC Panamá: https://sacpma.com/metodologia-tradicional-vs-metodologia-agil-2/

Sánchez-Hernández, D., Lizano-Madriz, F., & Sandoval-Carvajal, M. M. (2020). *Integración de pruebas remotas de usabilidad en Programación Extrema: Revisión de literatura.* Retrieved from Uniciencia: https://doi.org/10.15359/ru.34-1.2

Tymkiw, N., Bournissen, J. M., & Tumino, M. C. (2020). SCRUM como Herramienta Metodológica para el Aprendizaje de la Programación. *Revista Iberoamericana de Tecnología en Educación y Educación en Tecnología*, pp. 81-89.

Venema, M. (2024, Abril 26). *Pruebas de aceptación: el qué y el por qué*. Retrieved from nimblework: https://www.nimblework.com/es/agile/pruebas-de-aceptacion/

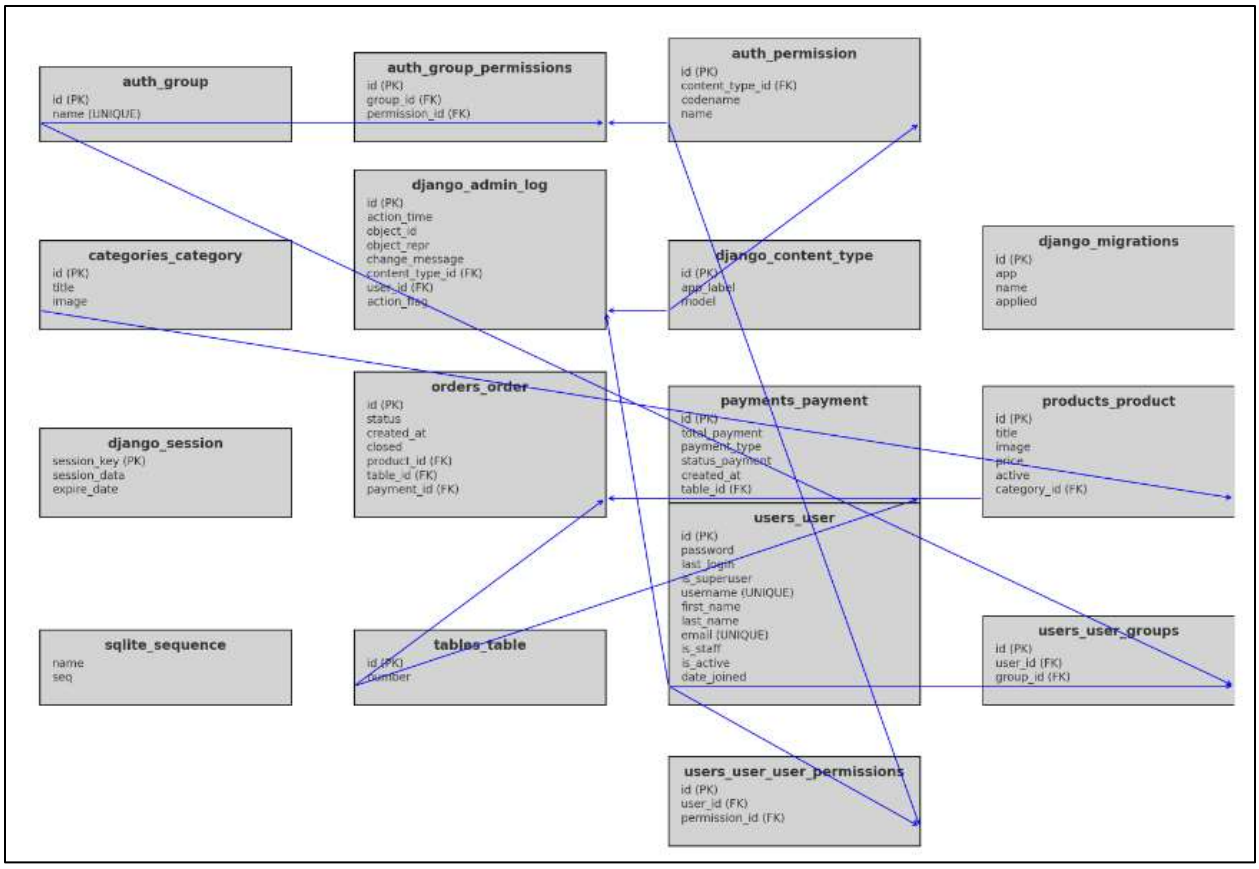
WIKIPEDIA. (2012). *Caso de uso*. Retrieved from WIKIPEDIA La Enciclopedia Libre: https://es.wikipedia.org/wiki/Caso_de_uso

ANEXOS

Anexo A *Audio Entrevista Requerimientos*

https://drive.google.com/file/d/1PKcJDyF2XIDiBSHc3NVyU02a36NPKo22/view?usp=drive_link

Anexo B *Modelo Base de Datos Relacional*



Anexo C *Tablas de Base de Datos*

Tables (16)

Name	Type	Schema
auth_group		CREATE TABLE "auth_group" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(150) NOT NULL UNIQUE)
id	integer	"id" integer NOT NULL
name	varchar(150)	"name" varchar(150) NOT NULL UNIQUE
auth_group_permissions		CREATE TABLE "auth_group_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "group_id" integer NOT NULL REFERENCES "auth_group" ("id") DEFERRABLE INITIALLY DEFERRED, "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
group_id	integer	"group_id" integer NOT NULL
permission_id	integer	"permission_id" integer NOT NULL
auth_permission		CREATE TABLE "auth_permission" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "content_type_id" integer NOT NULL REFERENCES "django_content_type" ("id") DEFERRABLE INITIALLY DEFERRED, "codename" varchar(100) NOT NULL, "name" varchar(255) NOT NULL)
id	integer	"id" integer NOT NULL
content_type_id	integer	"content_type_id" integer NOT NULL
codename	varchar(100)	"codename" varchar(100) NOT NULL
name	varchar(255)	"name" varchar(255) NOT NULL
categories_category		CREATE TABLE "categories_category" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(100) NOT NULL, "image" varchar(100) NOT NULL)
id	integer	"id" integer NOT NULL
title	varchar(100)	"title" varchar(100) NOT NULL
image	varchar(100)	"image" varchar(100) NOT NULL
django_admin_log		CREATE TABLE "django_admin_log" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "action_time" datetime NOT NULL, "object_id" text NULL, "object_repr" varchar(200) NOT NULL, "change_message" text NOT NULL, "content_type_id" integer NULL REFERENCES "django_content_type" ("id") DEFERRABLE INITIALLY DEFERRED, "user_id" bigint NOT NULL REFERENCES "users_user" ("id") DEFERRABLE INITIALLY DEFERRED, "action_flag" smallint unsigned NOT NULL CHECK ("action_flag" >= 0))
id	integer	"id" integer NOT NULL
action_time	datetime	"action_time" datetime NOT NULL
object_id	text	"object_id" text
object_repr	varchar(200)	"object_repr" varchar(200) NOT NULL
change_message	text	"change_message" text NOT NULL
content_type_id	integer	"content_type_id" integer
user_id	bigint	"user_id" bigint NOT NULL
action_flag	smallint unsigned	"action_flag" smallint unsigned NOT NULL CHECK("action_flag" >= 0)
django_content_type		CREATE TABLE "django_content_type" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app_label" varchar(100) NOT NULL, "model" varchar(100) NOT NULL)
id	integer	"id" integer NOT NULL
app_label	varchar(100)	"app_label" varchar(100) NOT NULL
model	varchar(100)	"model" varchar(100) NOT NULL
django_migrations		CREATE TABLE "django_migrations" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app" varchar(255) NOT NULL, "name" varchar(255) NOT NULL, "applied" datetime NOT NULL)
id	integer	"id" integer NOT NULL
app	varchar(255)	"app" varchar(255) NOT NULL
name	varchar(255)	"name" varchar(255) NOT NULL
applied	datetime	"applied" datetime NOT NULL
		CREATE TABLE "django_session" ("session_key" varchar(40) NOT

Name	Type	Schema
django_session		NULL PRIMARY KEY, "session_data" text NOT NULL, "expire_date" datetime NOT NULL)
session_key	varchar(40)	"session_key" varchar(40) NOT NULL
session_data	text	"session_data" text NOT NULL
expire_date	datetime	"expire_date" datetime NOT NULL
orders_order		CREATE TABLE "orders_order" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "status" varchar(255) NOT NULL, "created_at" datetime NOT NULL, "close" bool NOT NULL, "product_id" bigint NULL REFERENCES "products_product" ("id") DEFERRABLE INITIALLY DEFERRED, "table_id" bigint NULL REFERENCES "tables_table" ("id") DEFERRABLE INITIALLY DEFERRED, "payment_id" bigint NULL REFERENCES "payments_payment" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
status	varchar(255)	"status" varchar(255) NOT NULL
created_at	datetime	"created_at" datetime NOT NULL
close	bool	"close" bool NOT NULL
product_id	bigint	"product_id" bigint
table_id	bigint	"table_id" bigint
payment_id	bigint	"payment_id" bigint
payments_payment		CREATE TABLE "payments_payment" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "totalPayment" decimal NOT NULL, "paymentType" varchar(255) NOT NULL, "statusPayment" varchar(255) NOT NULL, "created_at" datetime NOT NULL, "table_id" bigint NULL REFERENCES "tables_table" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
totalPayment	decimal	"totalPayment" decimal NOT NULL
paymentType	varchar(255)	"paymentType" varchar(255) NOT NULL
statusPayment	varchar(255)	"statusPayment" varchar(255) NOT NULL
created_at	datetime	"created_at" datetime NOT NULL
table_id	bigint	"table_id" bigint
products_product		CREATE TABLE "products_product" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(255) NOT NULL, "image" varchar(100) NOT NULL, "price" decimal NOT NULL, "active" bool NOT NULL, "category_id" bigint NULL REFERENCES "categories_category" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
title	varchar(255)	"title" varchar(255) NOT NULL
image	varchar(100)	"image" varchar(100) NOT NULL
price	decimal	"price" decimal NOT NULL
active	bool	"active" bool NOT NULL
category_id	bigint	"category_id" bigint
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
name		"name"
seq		"seq"
tables_table		CREATE TABLE "tables_table" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "number" integer NOT NULL UNIQUE)
id	integer	"id" integer NOT NULL
number	integer	"number" integer NOT NULL UNIQUE
users_user		CREATE TABLE "users_user" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "password" varchar(128) NOT NULL, "last_login" datetime NULL, "is_superuser" bool NOT NULL, "username" varchar(150) NOT NULL UNIQUE, "first_name" varchar(150) NOT NULL, "last_name" varchar(150) NOT NULL, "is_staff" bool NOT NULL, "is_active" bool NOT NULL, "date_joined" datetime NOT NULL, "email" varchar(254) NOT NULL UNIQUE)
id	integer	"id" integer NOT NULL

Name	Type	Schema
564eba6		("user_id")
user_id		"user_id"
django_content_type_app_label_model_76bd3d3b_uniq		CREATE UNIQUE INDEX "django_content_type_app_label_model_76bd3d3b_uniq" ON "django_content_type" ("app_label", "model")
app_label		"app_label"
model		"model"
django_session_expire_date_a5c62663		CREATE INDEX "django_session_expire_date_a5c62663" ON "django_session" ("expire_date")
expire_date		"expire_date"
orders_order_payment_id_46928ccc		CREATE INDEX "orders_order_payment_id_46928ccc" ON "orders_order" ("payment_id")
payment_id		"payment_id"
orders_order_product_id_096244de		CREATE INDEX "orders_order_product_id_096244de" ON "orders_order" ("product_id")
product_id		"product_id"
orders_order_table_id_14015bc1		CREATE INDEX "orders_order_table_id_14015bc1" ON "orders_order" ("table_id")
table_id		"table_id"
payments_payment_table_id_76decc0f		CREATE INDEX "payments_payment_table_id_76decc0f" ON "payments_payment" ("table_id")
table_id		"table_id"
products_product_category_id_9b594869		CREATE INDEX "products_product_category_id_9b594869" ON "products_product" ("category_id")
category_id		"category_id"
users_user_groups_group_id_9afc8d0e		CREATE INDEX "users_user_groups_group_id_9afc8d0e" ON "users_user_groups" ("group_id")
group_id		"group_id"
users_user_groups_user_id_5f6f5a90		CREATE INDEX "users_user_groups_user_id_5f6f5a90" ON "users_user_groups" ("user_id")
user_id		"user_id"
users_user_groups_user_id_group_id_b88eab82_uniq		CREATE UNIQUE INDEX "users_user_groups_user_id_group_id_b88eab82_uniq" ON "users_user_groups" ("user_id", "group_id")
user_id		"user_id"
group_id		"group_id"
users_user_user_permissions_permission_id_0b93982e		CREATE INDEX "users_user_user_permissions_permission_id_0b93982e" ON "users_user_user_permissions" ("permission_id")
permission_id		"permission_id"
users_user_user_permissions_user_id_20aca447		CREATE INDEX "users_user_user_permissions_user_id_20aca447" ON "users_user_user_permissions" ("user_id")
user_id		"user_id"
users_user_user_permissions_user_id_permission_id_43338c45_uniq		CREATE UNIQUE INDEX "users_user_user_permissions_user_id_permission_id_43338c45_uniq" ON "users_user_user_permissions" ("user_id", "permission_id")
user_id		"user_id"
permission_id		"permission_id"

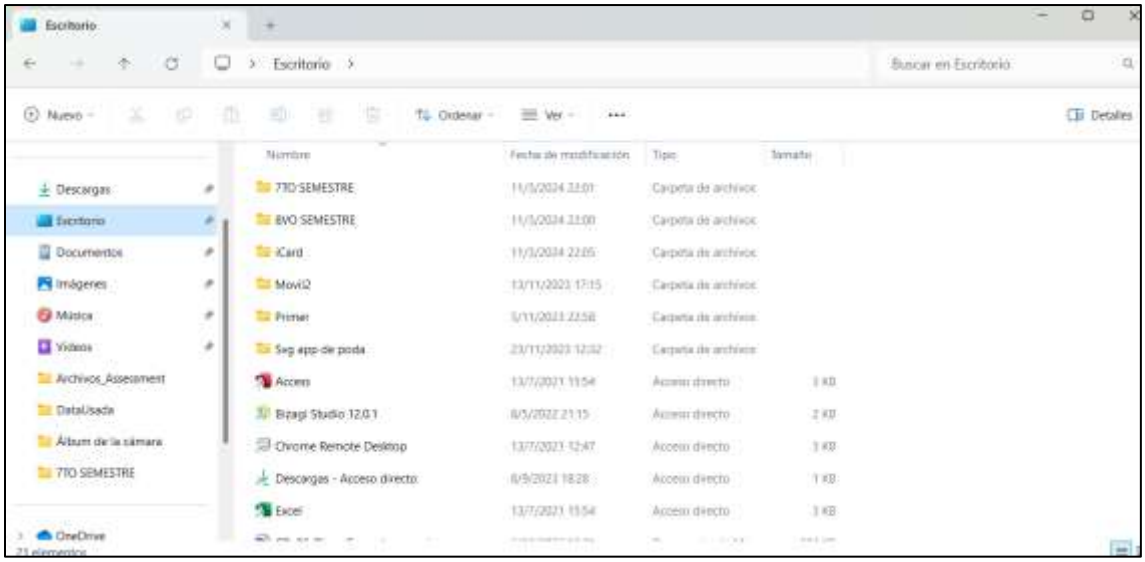
Name	Type	Schema
password	varchar(128)	"password" varchar(128) NOT NULL
last_login	datetime	"last_login" datetime
is_superuser	bool	"is_superuser" bool NOT NULL
username	varchar(150)	"username" varchar(150) NOT NULL UNIQUE
first_name	varchar(150)	"first_name" varchar(150) NOT NULL
last_name	varchar(150)	"last_name" varchar(150) NOT NULL
is_staff	bool	"is_staff" bool NOT NULL
is_active	bool	"is_active" bool NOT NULL
date_joined	datetime	"date_joined" datetime NOT NULL
email	varchar(254)	"email" varchar(254) NOT NULL UNIQUE
users_user_groups		CREATE TABLE "users_user_groups" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" bigint NOT NULL REFERENCES "users_user" ("id") DEFERRABLE INITIALLY DEFERRED, "group_id" integer NOT NULL REFERENCES "auth_group" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
user_id	bigint	"user_id" bigint NOT NULL
group_id	integer	"group_id" integer NOT NULL
users_user_user_permissions		CREATE TABLE "users_user_user_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" bigint NOT NULL REFERENCES "users_user" ("id") DEFERRABLE INITIALLY DEFERRED, "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
user_id	bigint	"user_id" bigint NOT NULL
permission_id	integer	"permission_id" integer NOT NULL

Indices (20)

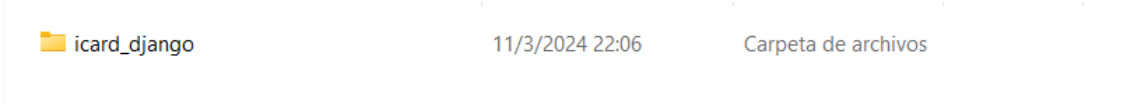
Name	Type	Schema
auth_group_permissions_group_id_b120cbf9		CREATE INDEX "auth_group_permissions_group_id_b120cbf9" ON "auth_group_permissions" ("group_id")
group_id		"group_id"
auth_group_permissions_group_id_permission_id_0cd325b0_uniq		CREATE UNIQUE INDEX "auth_group_permissions_group_id_permission_id_0cd325b0_uniq" ON "auth_group_permissions" ("group_id", "permission_id")
group_id		"group_id"
permission_id		"permission_id"
auth_group_permissions_permission_id_84c5c92e		CREATE INDEX "auth_group_permissions_permission_id_84c5c92e" ON "auth_group_permissions" ("permission_id")
permission_id		"permission_id"
auth_permission_content_type_id_2f476e4b		CREATE INDEX "auth_permission_content_type_id_2f476e4b" ON "auth_permission" ("content_type_id")
content_type_id		"content_type_id"
auth_permission_content_type_id_codename_01ab375a_uniq		CREATE UNIQUE INDEX "auth_permission_content_type_id_codename_01ab375a_uniq" ON "auth_permission" ("content_type_id", "codename")
content_type_id		"content_type_id"
codename		"codename"
django_admin_log_content_type_id_c4bce8eb		CREATE INDEX "django_admin_log_content_type_id_c4bce8eb" ON "django_admin_log" ("content_type_id")
content_type_id		"content_type_id"
django_admin_log_user_id_c		CREATE INDEX "django_admin_log_user_id_c564eba6" ON "django_admin_log"

Anexo D *Manual Activación Entorno Aplicación*

Para la creación del proyecto creamos una carpeta llamada iCard, que se encuentra en el escritorio:



Dentro de esta carpeta se encuentra otra carpeta que la creamos manualmente y se llama:



Abrimos un cmd y nos movemos a la carpeta, usando comando cd:

```
C:\Users\SISTEMAS\Desktop\iCard\icard_django>e
```

Dentro de esta carpeta descargamos los paquetes necesarios del Python y creamos otras dos carpetas dentro:

```
python3 -m venv envs/icard
```

Después iniciamos nuestro entorno virtual con este comando:

```
C:\Users\SISTEMAS\Desktop\iCard\icard_django>envs\icard\Scripts\activate

(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django>
```

Después se comienza a crear el proyecto de django, usando este comando:

```
pip3 install django
```

Luego creamos el proyecto:

```
>django-admin startproject icard
```

Y nos movemos a la carpeta creada con el cd, ponemos este código para iniciar el proyecto:

```
(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django\icard>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations f
r app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
March 11, 2024 - 22:50:03
Django version 5.0.3, using settings 'icard.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Generamos este código para que valga todo en la consola:

```
Símbolo del sistema

(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django\icard>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Luego instalamos lo que necesitamos:

```
(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django\icard>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 28, 2024 - 21:54:53
Django version 5.0.3, using settings 'icard.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

Not Found: /
[28/Apr/2024 21:54:57] "GET / HTTP/1.1" 404 2316
[28/Apr/2024 21:55:12] "GET /docs HTTP/1.1" 301 0
[28/Apr/2024 21:55:12] "GET /docs/ HTTP/1.1" 200 2158
[28/Apr/2024 21:55:12] "GET /static/drf-yasg/style.css HTTP/1.1" 200 1047
[28/Apr/2024 21:55:12] "GET /static/drf-yasg/swagger-ui-init.js HTTP/1.1" 200 14964
[28/Apr/2024 21:55:12] "GET /static/drf-yasg/swagger-ui-dist/swagger-ui-bundle.js HTTP/1.1" 200 10465
[28/Apr/2024 21:55:12] "GET /static/drf-yasg/swagger-ui-dist/swagger-ui.css HTTP/1.1" 200 145206
```

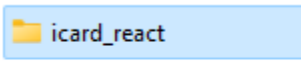
Nos movemos para crear la aplicación en React:

```
(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django>cd ..

(icard) C:\Users\SISTEMAS\Desktop\iCard>npx create-react-app icard
```

Creamos la carpeta con esto:

```
(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django>cd ..

(icard) C:\Users\SISTEMAS\Desktop\iCard>npx create-react-app icard
```

Luego cambiamos el nombre de la carpeta manualmente:



Nos movemos a la dirección de la carpeta con esto y le damos yarn para instalar dependencias:

```
(icard) C:\Users\SISTEMAS\Desktop\iCard>cd icard_react

(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_react>yarn
yarn install v1.22.21
```

Luego iniciamos con el yarn start

```
success Saved lockfile.
Done in 153.97s.

(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_react>yarn start
yarn run v1.22.21
$ react-scripts start
(node:15544) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMi
ption is deprecated. Please use the 'setupMiddlewares' option.
(Use 'node --trace-deprecation ...' to show where the warning was created)
(node:15544) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupM
 option is deprecated. Please use the 'setupMiddlewares' option.
```

Y luego tenemos que correr los dos programas al mismo tiempo con dos terminales:

```
C:\Users\SISTEMAS\Desktop\iCard\icard_django>envs\icard\Scripts\activate

(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django>python manage.py runserver
C:\Users\SISTEMAS\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\pytho
n.exe: can't open file 'C:\\Users\\SISTEMAS\\Desktop\\iCard\\icard_django\\manage.py': [Errno 2] No such file
or directory

(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django>cd icard

(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django\icard>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 29, 2024 - 21:52:04
Django version 5.0.3, using settings 'icard.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

CLIENTE = REACT

SERVIDOR = DJANGO

Para poder crear las tablas debemos crear un super usuario con django utilizando el siguiente comando:

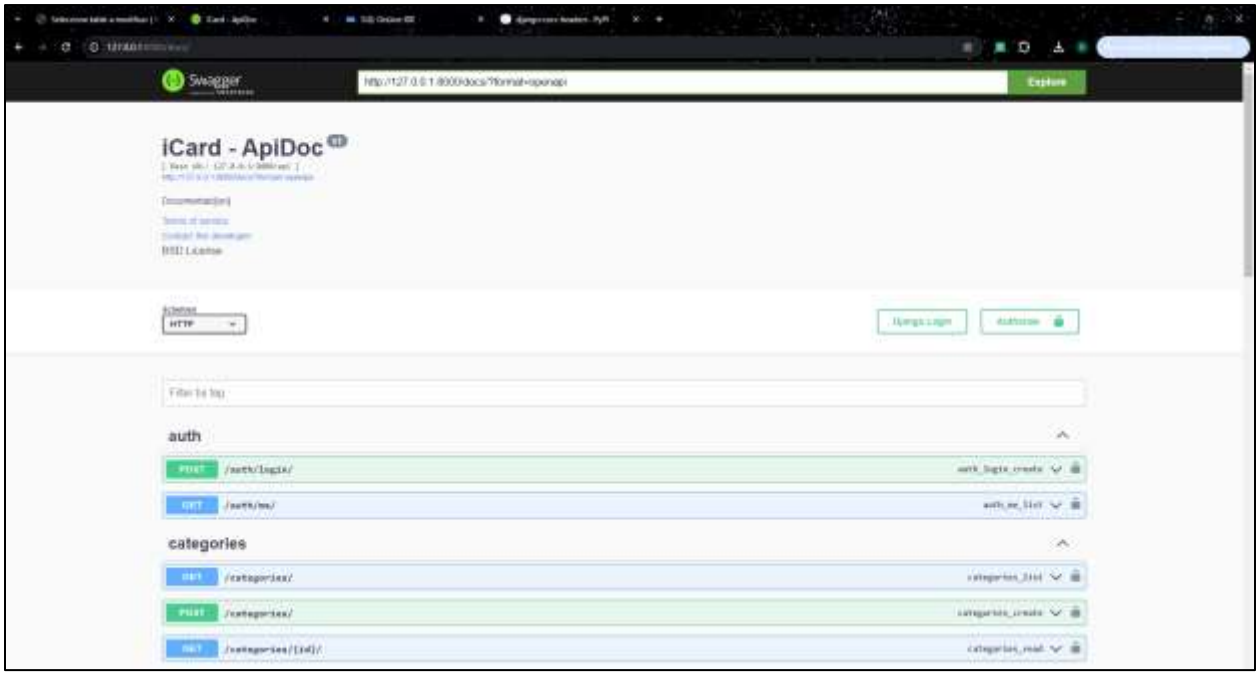
```
(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django\icard>python manage.py createsuperuser
Username (leave blank to use 'sistemas'): DiegoCer1625
```

USUARIO: DiegoCer1625

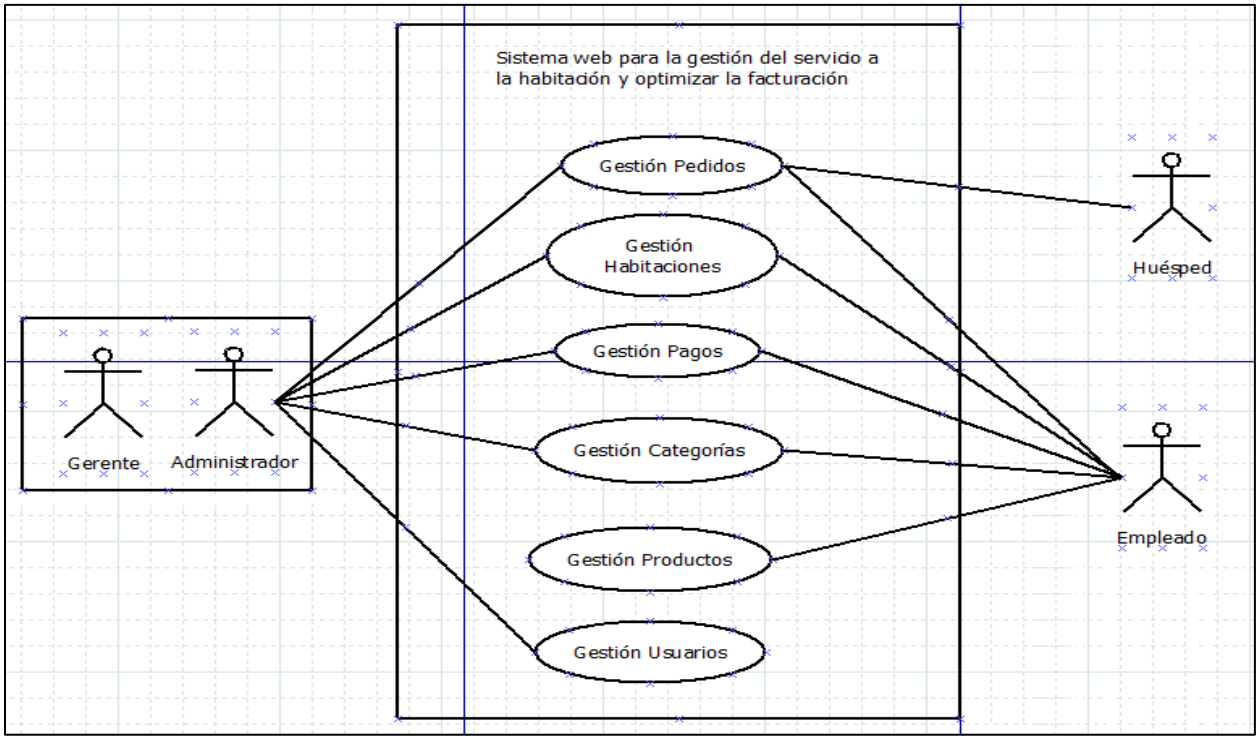
CONTRASEÑA: 123456

```
(icard) C:\Users\SISTEMAS\Desktop\iCard\icard_django\icard>python manage.py createsuperuser
Username (leave blank to use 'sistemas'): DiegoCer1625
Email address: cervantesdiego642@gmail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

Anexo E *API Aplicación*



Anexo F *Diagrama Caso Uso General Aplicación*



Anexo G *BackEnd Aplicación*

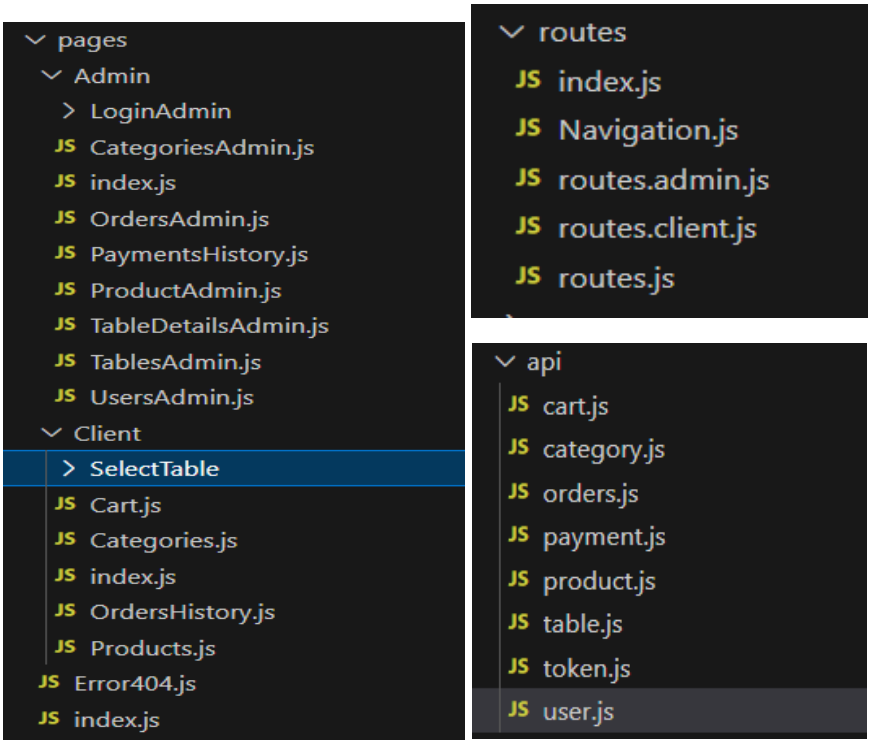
https://github.com/Dacervantes1/PlanTesisDjango.git

Anexo H *FrontEnd Aplicación*

https://github.com/Dacervantes1/PlanTesisReact.git

Anexo I *Documentación Y Código*

Pestañas dentro del proyecto:



Código Gestión Usuarios:

```
import React, { useState, useEffect } from "react";
import { Loader } from "semantic-ui-react";
import {
  HeaderPage,
  TableUsers,
  AddEditUserForm,
} from "../../components/Admin";
import { ModalBasic } from "../../components/Common";
import { useUser } from "../../hooks";

export function UsersAdmin() {
  const [showModal, setShowModal] = useState(false);
  const [titleModal, setTitleModal] = useState(null);
  const [contentModal, setContentModal] = useState(null);
  const [refetch, setRefetch] = useState(false);
  const { loading, users, getUsers, deleteUser } = useUser();

  useEffect(() => getUsers(), [refetch]);

  const openCloseModal = () => setShowModal((prev) => !prev);
  const onRefetch = () => setRefetch((prev) => !prev);

  const addUser = () => {
    setTitleModal("Nuevo usuario");
    setContentModal(
```

```
      <AddEditUserForm onClose={openCloseModal} onRefetch={onRefetch} />
    );
    openCloseModal();
  };

  const updateUser = (data) => {
    setTitleModal("Actualizar usuario");
    setContentModal(
      <AddEditUserForm
        onClose={openCloseModal}
        onRefetch={onRefetch}
        user={data}
      />
    );
    openCloseModal();
  };

  const onDeleteUser = async (data) => {
    const result = window.confirm(`¿Eliminar usuario ${data.email}?`);
    if (result) {
      try {
        await deleteUser(data.id);
        onRefetch();
      } catch (error) {
        console.error(error);
      }
    }
  };

  return (
    <>
      <HeaderPage
        title="Usuarios"
        btnTitle="Nuevo usuario"
        btnClick={addUser}
      />
      {loading ? (
        <Loader active inline="centered">
          Cargando...
        </Loader>
      ) : (
        <TableUsers
          users={users}
          updateUser={updateUser}
          onDeleteUser={onDeleteUser}
        />
      )}

      <ModalBasic
        show={showModal}
        onClose={openCloseModal}
        title={titleModal}
        children={contentModal}
      />
    </>
  );
}
```

Código Usuarios Conexión Api:

```
import { BASE_API } from "../utils/constants";

export async function loginApi(formValue) {
  try {
    const url = `${BASE_API}/api/auth/login/`;
    const params = {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(formValue),
    };

    const response = await fetch(url, params);

    if (response.status !== 200) {
      throw new Error("Usuario o contraseña incorrectos");
    }

    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function getMeApi(token) {
  try {
    const url = `${BASE_API}/api/auth/me/`;
    const params = {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function getUsersApi(token) {
  try {
    const url = `${BASE_API}/api/users/`;
    const params = {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    };
    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
```

```
export async function addUserApi(data, token) {
  try {
    const url = `${BASE_API}/api/users/`;
    const params = {
      method: "POST",
      headers: {
        Authorization: `Bearer ${token}`,
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function updateUserApi(id, data, token) {
  try {
    const url = `${BASE_API}/api/users/${id}/`;
    const params = {
      method: "PATCH",
      headers: {
        Authorization: `Bearer ${token}`,
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function deleteUserApi(id, token) {
  try {
    const url = `${BASE_API}/api/users/${id}/`;
    const params = {
      method: "DELETE",
      headers: {
        Authorization: `Bearer ${token}`,
      },
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
```

Código Gestión Habitación:

```
import React, { useState, useEffect } from "react";
import { Loader } from "semantic-ui-react";
import {
  HeaderPage,
  TableTablesAdmin,
  AddEditTableForm,
} from "../../components/Admin";
import { ModalBasic } from "../../components/Common";
import { useTable } from "../../hooks";

export function TablesAdmin() {
  const [showModal, setShowModal] = useState(false);
  const [titleModal, setTitleModal] = useState(null);
  const [contentModal, setContentModal] = useState(null);
  const [refetch, setRefetch] = useState(false);
  const { loading, tables, getTables, deleteTable } = useTable();

  useEffect(() => getTables(), [refetch]);

  const openCloseModal = () => setShowModal((prev) => !prev);
  const onRefetch = () => setRefetch((prev) => !prev);

  const addTable = () => {
    setTitleModal("Crear Habitación");
    setContentModal(
      <AddEditTableForm onClose={openCloseModal} onRefetch={onRefetch} />
    );
    openCloseModal();
  };

  const updateTable = (data) => {
    setTitleModal("Actualizar Habitación");
    setContentModal(
      <AddEditTableForm
        onClose={openCloseModal}
        onRefetch={onRefetch}
        table={data}
      />
    );
    openCloseModal();
  };

  const onDeleteTable = async (data) => {
    const result = window.confirm(`¿Eliminar mesa ${data.number}?`);
    if (result) {
      await deleteTable(data.id);
      onRefetch();
    }
  };

  return (
    <>
      <HeaderPage
        title="Mesas"
        btnTitle="Crear nueva mesa"
        btnClick={addTable}
      />

      {loading ? (
```

```
        <Loader active inline="centered">
          Cargando...
        </Loader>
      ) : (
        <TableTablesAdmin
          tables={tables}
          updateTable={updateTable}
          deleteTable={onDeleteTable}
        />
      )}

      <ModalBasic
        show={showModal}
        onClose={openCloseModal}
        title={titleModal}
        children={contentModal}
      />
    </>
  );
}
```

Código Habitaciones Conexión Api:

```
import { BASE_API } from "../utils/constants";

export async function getTablesApi(token) {
  try {
    const url = `${BASE_API}/api/tables/`;
    const params = {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function addTableApi(data, token) {
  try {
    const url = `${BASE_API}/api/tables/`;
    const params = {
      method: "POST",
      headers: {
        Authorization: `Bearer ${token}`,
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    };
    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
```

```
}

export async function updateTableApi(id, data, token) {
  try {
    const url = `${BASE_API}/api/tables/${id}/`;
    const params = {
      method: "PATCH",
      headers: {
        Authorization: `Bearer ${token}`,
        "Content-Type": "application/json",
      },
      body: JSON.stringify(data),
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
export async function deleteTableApi(id, token) {
  try {
    const url = `${BASE_API}/api/tables/${id}/`;
    const params = {
      method: "DELETE",
      headers: {
        Authorization: `Bearer ${token}`,
      },
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
export async function getTableApi(idTable) {
  try {
    const url = `${BASE_API}/api/tables/${idTable}/`;
    const response = await fetch(url);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
export async function getTableByNumberApi(numberTable) {
  try {
    const tableFilter = `number=${numberTable}`;

    const url = `${BASE_API}/api/tables/?${tableFilter}`;
    const response = await fetch(url);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
```

Código Gestión Productos:

```
import React, { useState, useEffect } from "react";
import { Loader } from "semantic-ui-react";
import {
  HeaderPage,
  TableProductAdmin,
  AddEditProductForm,
} from "../../components/Admin";
import { ModalBasic } from "../../components/Common";
import { useProduct } from "../../hooks";

export function ProductAdmin() {
  const [showModal, setShowModal] = useState(false);
  const [titleModal, setTitleModal] = useState(null);
  const [contentModal, setContentModal] = useState(null);
  const [refetch, setRefetch] = useState(false);
  const { loading, products, getProducts, deleteProduct } = useProduct();

  useEffect(() => getProducts(), [refetch]);

  const openCloseModal = () => setShowModal((prev) => !prev);
  const onRefetch = () => setRefetch((prev) => !prev);

  const addProduct = () => {
    setTitleModal("Nuevo producto");
    setContentModal(
      <AddEditProductForm onClose={openCloseModal} onRefetch={onRefetch} />
    );
    openCloseModal();
  };

  const updateProduct = (data) => {
    setTitleModal("Actualizar producto");
    setContentModal(
      <AddEditProductForm
        onClose={openCloseModal}
        onRefetch={onRefetch}
        product={data}
      />
    );
    openCloseModal();
  };

  const onDeleteProduct = async (data) => {
    const result = window.confirm(`¿Eliminar producto ${data.title}?`);
    if (result) {
      await deleteProduct(data.id);
      onRefetch();
    }
  };

  return (
    <>
      <HeaderPage
        title="Productos"
        btnTitle="Nuevo producto"
        btnClick={addProduct}
      />

      {loading ? (
```

```
        <Loader active inline="centered">
          Cargando...
        </Loader>
      ) : (
        <TableProductAdmin
          products={products}
          updateProduct={updateProduct}
          deleteProduct={onDeleteProduct}
        />
      )}

      <ModalBasic
        show={showModal}
        onClose={openCloseModal}
        title={titleModal}
        children={contentModal}
      />
    </>
  );
}
```

Código Productos Conexión Api:

```
import { BASE_API } from "../utils/constants";
export async function getProductsApi() {
  try {
    const url = `${BASE_API}/api/products/`;
    const response = await fetch(url);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
export async function addProductApi(data, token) {
  try {
    const formData = new FormData();
    formData.append("title", data.title);
    formData.append("price", data.price);
    formData.append("category", data.category);
    formData.append("active", data.active);
    formData.append("image", data.image);

    const url = `${BASE_API}/api/products/`;
    const params = {
      method: "POST",
      headers: {
        Authorization: `Bearer ${token}`,
      },
      body: formData,
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
```

```
export async function updateProductApi(id, data, token) {
  try {
    const formData = new FormData();
    formData.append("title", data.title);
    formData.append("price", data.price);
    formData.append("category", data.category);
    formData.append("active", data.active);
    if (data.image) formData.append("image", data.image);

    const url = `${BASE_API}/api/products/${id}/`;
    const params = {
      method: "PATCH",
      headers: {
        Authorization: `Bearer ${token}`,
      },
      body: formData,
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
export async function deleteProductApi(id, token) {
  try {
    const url = `${BASE_API}/api/products/${id}/`;
    const params = {
      method: "DELETE",
      headers: {
        Authorization: `Bearer ${token}`,
      },
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
export async function getProductByIdApi(id) {
  try {
    const url = `${BASE_API}/api/products/${id}/`;
    const response = await fetch(url);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
export async function getProductsByCategoryApi(idCategory) {
  try {
    const categoryFilter = `category=${idCategory}`;
    const url = `${BASE_API}/api/products/?${categoryFilter}`;
    const response = await fetch(url);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;  }}
```

Código Gestión Categoría:

```
import { useState } from "react";
import {
  getCategoriesApi,
  addCategoryApi,
  updateCategoryApi,
  deleteCategoryApi,
} from "../api/category";
import { useAuth } from "./";

export function useCategory() {
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(false);
  const [categories, setCategories] = useState(null);
  const { auth } = useAuth();

  const getCategories = async () => {
    try {
      setLoading(true);
      const response = await getCategoriesApi();
      setLoading(false);
      setCategories(response);
    } catch (error) {
      setLoading(false);
      setError(error);
    }
  };

  const addCategory = async (data) => {
    try {
      setLoading(true);
      await addCategoryApi(data, auth.token);
      setLoading(false);
    } catch (error) {
      setLoading(false);
      setError(error);
    }
  };

  const updateCategory = async (id, data) => {
    try {
      setLoading(true);
      await updateCategoryApi(id, data, auth.token);
      setLoading(false);
    } catch (error) {
      setLoading(false);
      setError(error);
    }
  };

  const deleteCategory = async (id) => {
    try {
      setLoading(true);
      await deleteCategoryApi(id, auth.token);
      setLoading(false);
    } catch (error) {
      setLoading(false);
      setError(error);
    }
  };
```

```
  return {
    loading,
    error,
    categories,
    getCategories,
    addCategory,
    updateCategory,
    deleteCategory,
  };
}
```

Código Categoría Conexión Api:

```
import { BASE_API } from "../utils/constants";

export async function getCategoriesApi() {
  try {
    const url = `${BASE_API}/api/categories/`;
    const response = await fetch(url);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function addCategoryApi(data, token) {
  try {
    const formData = new FormData();
    formData.append("image", data.image);
    formData.append("title", data.title);

    const url = `${BASE_API}/api/categories/`;
    const params = {
      method: "POST",
      headers: {
        Authorization: `Bearer ${token}`,
      },
      body: formData,
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function updateCategoryApi(id, data, token) {
  try {
    const formData = new FormData();
    formData.append("title", data.title);
    if (data.image) formData.append("image", data.image);

    const url = `${BASE_API}/api/categories/${id}/`;
    const params = {
      method: "PATCH",
      headers: {
```

```
        Authorization: `Bearer ${token}`,
      },
      body: formData,
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function deleteCategoryApi(id, token) {
  try {
    const url = `${BASE_API}/api/categories/${id}/`;
    const params = {
      method: "DELETE",
      headers: {
        Authorization: `Bearer ${token}`,
      },
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
```

Código Gestión Pedidos:

```
import { useState } from "react";
import {
  getOrdersByTableApi,
  checkDeliveredOrderApi,
  addOrderToTableApi,
  addPaymentToOrderApi,
  closeOrderApi,
  getOrdersByPaymentApi,
} from "../api/orders";

export function useOrder() {
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(false);
  const [orders, setOrders] = useState(null);

  const getOrdersByTable = async (idTable, status, ordering) => {
    try {
      setLoading(true);
      const response = await getOrdersByTableApi(idTable, status, ordering);
      setLoading(false);
      setOrders(response);
    } catch (error) {
      setLoading(false);
      setError(error);
    }
  };
```

```
  const checkDeliveredOrder = async (idOrder) => {
    try {
      await checkDeliveredOrderApi(idOrder);
    } catch (error) {
      setError(error);
    }
  };

  const addOrderToTable = async (idTable, idProduct) => {
    try {
      await addOrderToTableApi(idTable, idProduct);
    } catch (error) {
      setError(error);
    }
  };

  const addPaymentToOrder = async (idOrder, idPayment) => {
    try {
      await addPaymentToOrderApi(idOrder, idPayment);
    } catch (error) {
      // console.log(error)
      setError(error);
    }
  };

  const closeOrder = async (idOrder) => {
    try {
      await closeOrderApi(idOrder);
    } catch (error) {
      setError(error);
    }
  };

  const getOrdersByPayment = async (idPayment) => {
    try {
      return await getOrdersByPaymentApi(idPayment);
    } catch (error) {
      setError(error);
    }
  };

  return {
    loading,
    error,
    orders,
    getOrdersByTable,
    checkDeliveredOrder,
    addOrderToTable,
    addPaymentToOrder,
    closeOrder,
    getOrdersByPayment,
  };
}
```

Código Pedidos Conexión Api:

```
import { BASE_API, ORDER_STATUS } from "../utils/constants";

export async function getOrdersByTableApi(idTable, status = "", ordering = "") {
  try {
    const tableFilter = `table=${idTable}`;
    const statusFilter = `status=${status}`;
    const closeFilter = "close=False";

    const url =
`${BASE_API}/api/orders/?${tableFilter}&${statusFilter}&${closeFilter}&${ordering}`;
    const response = await fetch(url);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function checkDeliveredOrderApi(id) {
  try {
    const url = `${BASE_API}/api/orders/${id}/`;
    const params = {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        status: ORDER_STATUS.DELIVERED,
      }),
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function addOrderToTableApi(idTable, idProduct) {
  try {
    const url = `${BASE_API}/api/orders/`;
    const params = {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        status: ORDER_STATUS.PENDING,
        table: idTable,
        product: idProduct,
      }),
    };
    await fetch(url, params);
  } catch (error) {
    throw error;
  }
}
```

```
export async function addPaymentToOrderApi(idOrder, idPayment) {
  try {
    const url = `${BASE_API}/api/orders/${idOrder}/`;
    const params = {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        payment: idPayment,
      }),
    };
    await fetch(url, params);
  } catch (error) {
    throw error;
  }
}

export async function closeOrderApi(idOrder) {
  try {
    const url = `${BASE_API}/api/orders/${idOrder}/`;
    const params = {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        close: true,
      }),
    };
    await fetch(url, params);
  } catch (error) {
    throw error;
  }
}

export async function getOrdersByPaymentApi(idPayment) {
  try {
    const paymentFilter = `payment=${idPayment}`;

    const url = `${BASE_API}/api/orders/?${paymentFilter}`;
    const response = await fetch(url);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
```

Código Gestión Historial Pago:

```
import React, { useEffect } from "react";
import { Loader } from "semantic-ui-react";
import { HeaderPage, TablePayments } from "../../components/Admin";
import { usePayment } from "../../hooks";

export function PaymentsHistory() {
  const { loading, payments, getPayments } = usePayment();

  useEffect(() => getPayments(), []);

  return (
    <>
      <HeaderPage title="Historial de pagos" />
      {loading ? (
        <Loader active inline="centered">
          Cargando...
        </Loader>
      ) : (
        <TablePayments payments={payments} />
      )}
    </>
  );
}
```

Código Historial Pago Conexión Api:

```
import { BASE_API, PAYMENT_STATUS } from "../utils/constants";

export async function createPaymentApi(paymentData) {
  try {
    const url = `${BASE_API}/api/payments/`;
    const params = {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(paymentData),
    };

    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function getPaymentByTableApi(idTable) {
  try {
    const tableFilter = `table=${idTable}`;
    const statusFilter = `statusPayment=${PAYMENT_STATUS.PENDING}`;
```

```
    const url = `${BASE_API}/api/payments/?${tableFilter}&${statusFilter}`;
    const params = {
      headers: {
        "Content-Type": "application/json",
      },
    };
    const response = await fetch(url, params);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}

export async function closePaymentApi(idPayment) {
  try {
    const url = `${BASE_API}/api/payments/${idPayment}/`;
    const params = {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        statusPayment: PAYMENT_STATUS.PAID,
      }),
    };
    await fetch(url, params);
  } catch (error) {
    throw error;
  }
}

export async function getPaymentsApi() {
  try {
    const paymentFilter = `statusPayment=${PAYMENT_STATUS.PAID}`;
    const orderingFilter = "ordering=created_at";

    const url = `${BASE_API}/api/payments/?${paymentFilter}&${orderingFilter}`;

    const response = await fetch(url);
    const result = await response.json();
    return result;
  } catch (error) {
    throw error;
  }
}
```