



Instituto Tecnológico y de Estudios Superiores de Monterrey  
Campus Ciudad de México

Departamento de Computación  
Diseño y Arquitectura de Software  
Profesora Marlene O. Sánchez Escobar

“Proyecto Arquitectura de Software”

Javier Arturo Flores Zavala

A01651678

Ángel Heredia Vázquez

A01650574

Samuel Kareem Cueto González

A01656120

Carlos Andrés Conde Besil

A01650549

José Carlos Acosta García

A01650306

Mathilde Tournemire

A01760232

Smart Pay Co

Documento de Arquitectura de Software

Versión 1.0

### Historial de Revisiones

| Fecha         | Versión | Descripción  | Autor             |
|---------------|---------|--|-------------------|
| 18 abril 2022 | 1.0     | Corrección de requerimientos   | Equipo Condenados |
| 26 abril 2022 | 1.1     | Corrección de diagramas  | Equipo Condenados |
| 28 abril 2022 | 1.2     | Corrección interfaces, notación diagramas, especificidad casos de uso. | Equipo Condenados |
| 11 mayo 2022  | 1.3     | Vista microservicios añadida   | Equipo Condenados |

Elaborado por: .

## Contenido

|  |    |
|--|----|
| <b>1. Introducción</b>   | 5  |
| <b>1.1 Propósito</b>   | 5  |
| <b>1.2 Alcance</b>   | 5  |
| <b>1.3 Definiciones, Siglas y Abreviaturas</b>   | 8  |
| <b>1.4 Referencias</b>   | 8  |
| <b>1.5 Resumen</b>   | 9  |
| <b>2. Representación Arquitectónica</b>  | 10 |
| <b>2.1 Objetivos Arquitectónicos</b>   | 10 |
| <b>2.2 Restricciones arquitectónicas</b>   | 14 |
| <b>2.3 Estilo Arquitectónico y rationale</b>   | 17 |
| <i>Describe el estilo arquitectónico seleccionado y la razón por la cual lo seleccionó</i> |    |
| <b>2.4 Vista de casos de uso</b>   | 18 |
| <b>2.5 Escenarios Arquitectónicos y Requerimientos No Funcionales Asociados</b>            | 33 |
| <b>2.5.1 Escenarios Arquitectónicos</b>  | 33 |
| <i>Incluir los escenarios arquitectónicos que se espera cumplir</i>                        |    |
| <b>2.5.2 Requerimientos Funcionales Asociados</b>  | 35 |
| <b>Tamaño y Rendimiento</b>  |    |
| <b>2.6 Vista lógica</b>  | 39 |
| <b>2.7 Vista de proceso</b>  | 41 |
| <b>2.8 Vista de implementación</b>   | 44 |
| <b>2.9 Vista de datos</b>  | 42 |
| <b>2.9.1 Diagrama de Interfaz API's</b>  | 45 |
| <b>2.9.2 Base de datos</b>   | 46 |
| <b>3 Lineamientos Arquitectónicos</b>  | 49 |
| <b>3.1 Codificación</b>  | 49 |
| <b>3.2 Diseño</b>  | 55 |
| <b>3.3 Reutilización del Software</b>  | 59 |

|   |    |
|---|----|
| <b>4 Architecture Tradeoff Analysis Method</b>                | 60 |
| <b>4.1 Drivers Arquitectónicos</b>                            | 60 |
| <b>4.2 Restricciones técnicas, administrativas</b>            | 60 |
| <b>4.3 Stakeholders</b>                                       | 62 |
| <b>4.4 Vistas Arquitectónicas</b>                             | 63 |
| <b>4.4.1 Vista Lógica Front-End</b>                           | 63 |
| <b>4.4.2 Vista Lógica Back-End</b>                            | 64 |
| <b>4.4.3 Vista de Proceso</b>                                 | 65 |
| <b>4.4.4 Vista de Implementación</b>                          | 66 |
| <b>4.4.5 Vista de Datos</b>                                   | 67 |
| <b>4.5 Descripción de Arquitectura Microservicios ATAM</b>    | 67 |
| <b>4.6 Tradeoffs Específicos de la Arquitectura</b>           | 68 |
| <b>4.7 Atributos de calidad ATAM - Quality Attribute Tree</b> | 70 |
| <b>4.8 Análisis de Atributos y Arquitectura</b>               | 71 |
| <b>4.9 Commercial Off-The-Shelf (COTS)</b>                    | 91 |
| <b>4.10 Pruebas unitarias</b>                                 | 92 |

# Documento de Arquitectura de Software

## 1. Introducción

Este documento es la descripción formal y minuciosa del proyecto y arquitectura del sistema a desarrollar para la institución financiera Smart Pay Co. Dicho documento será implementado con base en los criterios arquitectónicos y rationale que más se adecúen a las necesidades del cliente, de los stakeholders, los requerimientos funcionales y no funcionales.

### 1.1 Propósito

El documento actual tiene como objetivo la definición y especificación de las características y requerimientos del sistema desarrollado para Smart Pay Co, especificando claramente las decisiones arquitectónicas para lograr desarrollar un software mantenible que cumpla con altos estándares de calidad

En términos generales, se desarrollará un sistema optimizado para la gestión de pagos. Esto le permitirá a los empleados agilizar su trabajo mediante un sistema jerarquizado con funcionalidad de búsqueda y registro de transacciones.

El presente documento está destinado a aquellos usuarios del sistema Smart Pay Co. que cuenten con cierto conocimiento técnico que deseen conocer a detalle el funcionamiento del sistema. Asimismo, está dirigido para aquellos analistas y programadores que deseen en un futuro, modificar y añadir funcionalidades del sistema. Más aún, este documento será particularmente útil para los arquitectos de software que lleguen a involucrarse con el proyecto pues podrán comprender las decisiones y justificaciones del estilo arquitectónico.

### 1.2 Alcance

El sistema de gestión de pagos de Smart Pay Co permitirá la gestión de pagos de los usuarios de la compañía. El sistema debe permitir registrar pagos a través de una interfaz de usuario (web), la cual le permitirá a los usuarios ingresar al sistema (a través de validaciones de inicio de sesión) y realizar la

gestión de los pagos correspondientes de forma adecuada, conforme a los privilegios de su tipo de cuenta manejado mediante roles de usuarios.

El alcance del proyecto incluye el desarrollo de un sistema web utilizando las tecnologías de React, Node JS, Express y MySQL, para la gestión y procesamiento de pagos considerando las siguientes funcionalidades:

- CRUD de Usuarios en la base de datos relacional (MySQL)
- CRUD de Pagos en la base de datos relacional (MySQL)
- Autenticación de usuarios a través de *OAuth*
- Conexión con las siguientes API's:
  - Banxico (Consulta de tipo de cambio al día)
  - Google Cloud Platform API
- Interfaz web interactiva con distintas pantallas para desplegar y gestionar lo siguiente:
  - Pagos Recibidos (Con opciones de edición)
  - Creación de órdenes de pago
  - Visualización y edición de cuentas bancarias
  - Creación de cuentas bancarias
  - Visualización y edición de usuarios (Únicamente para administradores)
  - Creación de usuarios (Únicamente para administradores)

En primer lugar, el sistema debe permitir al usuario ingresar a su cuenta a través de un proceso de autenticación en la ventana de inicio de sesión. Por lo que, se debe de validar las credenciales del usuario directamente en la base de datos para su ingreso exitoso.

En segundo lugar, el sistema debe permitir a los usuarios crear órdenes de pago.

El alcance del proyecto incluye la modificación de los formularios de llenado de información para las órdenes de pago y la base de datos para agregar nuevos campos. Esto con la intención de estar actualizados con lo que tanto la ley, como Smart Pay Co requieran solicitar y almacenar respectivamente para la creación de pagos.

En tercer lugar, el sistema debe permitir al usuario consultar y visualizar el historial de pagos recibidos, permitiendo filtrar los pagos y la edición de los mismos.

En cuarto lugar, el sistema debe incluir un apartado para que el usuario pueda gestionar sus propias cuentas bancarias, permitiendo la creación de nuevas cuentas.

En quinto lugar, el sistema debe tener un apartado para que los usuarios con rol de administrador puedan gestionar las cuentas existentes y poder crear nuevas cuentas, esto únicamente disponible para este tipo de usuarios y restringido para el resto de usuarios.

Finalmente, podrá existir una posible modificación del front end para tener una visualización correcta en dispositivos móviles para que la funcionalidad de geolocalización pueda ser subida desde un celular.

En términos generales, esta aplicación permitirá a los empleados Smart Pay Co y usuarios del sistema realizar su trabajo con mayor velocidad y efectividad, mejorando significativamente la calidad de sus servicios.

### 1.3 Definiciones, Siglas y Abreviaturas

Back End: Lógica de la aplicación que se caracteriza por las transiciones, operaciones u obtención de datos que el servicio web solicite por parte del usuario. No todas las interacciones del usuario requieren del Back end, pero otras funcionalidades como buscar un producto, crear un nuevo usuario u otra operación similar es necesario su uso.

CRUD: define las siglas de Create, Read, Update y Delete de datos.

DB: Database o base de datos.

Divisas: Moneda extranjera manejada de manera internacional.

Endpoint: Dispositivo informático remoto que se comunica con una red a la que está conectado. Punto que es la parte final de una red.

Front End: Lógica de una aplicación que se caracteriza por las funcionalidades que suceden por delante de la aplicación. Estas pueden ser el despliegue de pantalla, validaciones frontales, o interacciones del usuario con la interfaz gráfica.

Rationale: Razón fundamental por la cual se toman decisiones dentro de un esquema de acciones.

React: Librería de JavaScript open source para el desarrollo de interfaces de usuario.

Ripple Effect: Que exista una cadena de efectos al momento de realizar una modificación

Smart Pay Co: Institución de control inteligente de transacciones bancarias.

### 1.4 Referencias

BBVA. (2022) ¿Qué es una Divisa? bbva. Recuperado el 30 de marzo del 2022 de: <https://www.bbva.mx/educacion-financiera/d/divisa.html>

Cambridge. (2021). Significado de RATIONALE. Dictionary Cambridge. Recuperado el 30 de marzo del 2022 de: <https://dictionary.cambridge.org/es/diccionario/ingles/rationale>

Cervantes, H., Velasco, P. y Castro, L. (2016) Arquitectura de Software, Conceptos y ciclo de desarrollo. Universidad Autónoma de Zacatecas. Recuperado el 30 de marzo del 2022 de: [https://www.researchgate.net/profile/Perla-Velasco-Elizondo/publication/281137715\\_Arquitectura\\_de\\_Software\\_Conceptos\\_y\\_Ciclo\\_de Desarrrollo/links/57144e1408aebe07c0641ab/Arquitectura-de-Software-Conceptos-y-Ciclo-de-Desarrrollo.pdf](https://www.researchgate.net/profile/Perla-Velasco-Elizondo/publication/281137715_Arquitectura_de_Software_Conceptos_y_Ciclo_de Desarrrollo/links/57144e1408aebe07c0641ab/Arquitectura-de-Software-Conceptos-y-Ciclo-de-Desarrrollo.pdf)

Desarrolloweb. (2019). Que es React. Para que se usa React. Desarrollo-web. Recuperado el 30 de marzo del 2022 de: <https://desarrolloweb.com/articulos/que-es-react-motivos-uso.html>



Okhravi, C. (2017). Adapter Pattern. Recuperado el 23 de abril de 2022 de:  
<https://www.youtube.com/watch?v=2PKQtcJjYvc>

Okhravi, C. (2017). Facade Pattern. Recuperado el 23 de abril de 2022 de:  
<https://www.youtube.com/watch?v=K4FkHVO5iac>

García, J. (2012) SOLID y GRASP: Buenas prácticas hacia el éxito en el desarrollo de software. Escuela Técnica Superior de Ingeniería Informática del a Universidad de Sevilla. Recuperado el 30 de marzo del 2022 de:  
<https://jbravomontero.files.wordpress.com/2012/12/solid-y-grasp-buenas-practicas-hacia-el-exito-en-el-desarrollo-de-software.pdf>

## 1.5 Resumen

El sistema Smart Pay Co está enfocado en la operación de transacciones entre dos cuentas bancarias, el propósito del documento es mostrar las características del sistema, así como sus distintas capacidades y restricciones que tendrá durante la operación. Por último se describe a nivel lógico los componentes que debe incluir dicho proyecto para su fase de desarrollo posterior. Dicho esto, el trabajo a realizar se compone de dos bloques principales:

En primer lugar, se encuentra una gestión de usuarios, en esta sección será capaz de realizar operaciones básicas de cuentas que serían la creación, lectura, actualización y borrado de registros, o mejor conocido como CRUD. De esta manera, con los datos ingresados, estos podrán hacer un inicio de sesión dentro de la plataforma y tener acceso al sistema con los criterios de autenticación requeridos. Por otro lado, cada perfil de usuario podrá ser distinto dependiendo de sus permisos que cuentan dentro de la aplicación, que para este proyecto existen tres tipos de usuarios, cuyas funciones se explican a detalle en los casos de uso.

En segundo lugar, las operaciones transaccionales funcionarán por medio de un formulario llenando la información solicitada, esté realizará una validación para que no introduzcan datos incongruentes con lo requerido. Una vez pasada esta etapa se agregara a un estado de pre aprobación donde únicamente los usuarios determinados podrán confirmar la operación y se realice la transacción bancaria.

Todas estas operaciones mencionadas se visualizarán dentro de un sistema web, lugar donde se podrá acceder dentro de los distintos dispositivos que los usuarios utilicen. Este servicio web se distribuirá por medio de un servidor que será el manejador de las peticiones realizadas por los usuarios.

## 2. Representación Arquitectónica

### 2.1 Objetivos Arquitectónicos

Acorde al entendimiento del negocio y las normas que este debe seguir, mismo análisis que se ha estipulado en los puntos anteriores, podemos denotar que el eje principal para la empresa contratante, Smart Pay Co, es la gestión de pagos/transacciones bancarias con flexibilidad de divisas mediante un sistema de autenticación de usuarios con jerarquía requerida. Teniendo en mente dicho objetivo principal, el equipo desarrollador ha optado por definir los siguientes objetivos arquitectónicos (drivers arquitectónicos):

#### *Drivers Funcionales*

Acorde a lo estipulado por *Cervantes, H., Velasco, P. y Castro, L.* en *Arquitectura de Software, Conceptos y ciclo de desarrollo* en 2016, los drivers funcionales se definen como un subconjunto de los requerimientos funcionales, buscando proveer información relevante para la descomposición funcional del sistema, acotando aquellos que resulten más relevantes para la satisfacción de los objetivos del negocio del sistema.

Para este tipo de drivers/objetivos, se tomaron como base aquellos requerimientos funcionales descritos con alta prioridad para el cumplimiento del eje principal de la empresa (definido en el párrafo anterior). De este modo, se prevé que los siguientes drivers permitan el desarrollo de los requerimientos funcionales del sistema, sirviendo como guía para el desarrollo funcional del sistema. Cabe mencionar que se encuentran ordenados por prioridad y que los que tengan prioridad alta fueron tomados en cuenta para el diseño de la arquitectura del sistema.

| Driver funcional     | Descripción                             | Complejidad | Prioridad |
|----------------------|---|-------------|-----------|
| Inicio de sesión por | Con la finalidad de otorgar un nivel de | Media       | Alta      |

|   |   |       |      |
|---|---|-------|------|
| medio de autenticación de usuarios.   | seguridad, así como de jerarquía en permisos a los usuarios del sistema, se plantea el inicio de sesión con correo electrónico y contraseña con alto nivel de seguridad.  |       |      |
| Registro de pagos/transacciones.  | Con la finalidad de mantener una interacción de creación, lectura, edición y eliminación (CRUD) sobre los pagos que se manejen en el sistema, se plantea la interacción de usuarios con pagos acorde al tipo de usuario, así como el permiso que este posea.  | Alta  | Alta |
| Gestión de usuarios del sistema a través de usuarios activos con nivel máximo jerárquico.                   | Con la finalidad de evitar alteración de información de los usuarios activos, así como un control adecuado del manejo de nuevos o existentes miembros del sistema, se plantea que únicamente los usuarios con el mayor rango de permisos otorgados serán quienes mantengan un control sobre las acciones de interacción con los usuarios miembros del sistema, mismas que comprenden su creación, lectura, edición y eliminación (CRUD).        | Media | Alta |
| Gestión de cuentas bancarias del sistema a través de usuarios activos con nivel máximo jerárquico.          | Con la finalidad de evitar alteración de información de las cuentas bancarias activas, así como un control adecuado del manejo de nuevas o existentes cuentas o el manejo de sus fondos, únicamente los usuarios con el mayor rango de permisos otorgados serán quienes mantengan un control sobre las acciones de interacción con las cuentas bancarias del sistema, mismas que comprenden su creación, lectura, edición y eliminación (CRUD). | Media | Alta |
| Validaciones de campos en torno a la información introducida para creación de entidades dentro del sistema. | Con la finalidad de cumplir con todas las estructuras de información que requiere cada una de las entidades del sistema (pagos/transacciones, cuentas bancarias, usuarios, solicitudes, etc.), se plantea la validación de cada campo por medio de revisión de texto introducido, así como su posible validación con datos almacenados en base de datos, esto para garantizar que la información requerida será proporcionada y                 | Media | Alta |

|   |  |      |       |
|---|--|------|-------|
|   | no sea duplicada en ningún momento.  |      |       |
| Solicitar como requisito para creación de nuevos pagos/transacciones la geolocalización del solicitante al momento de su interacción. | Con la finalidad de evitar movimientos bancarios que pudiesen resultar en fraude y/o actividades ilícitas, se plantea la activación de geolocalización por parte del dispositivo del solicitante al momento de generar interacción de solicitud de creación. Esta será almacenada en la base de datos como campo obligatorio, en caso de no aceptar la activación de la misma, la solicitud de generación de movimiento bancario será rechazada. | Baja | Alta  |
| Aceptación de divisas extranjeras en manejo de pagos/transacciones.   | Con la finalidad de otorgar un alcance internacional, se plantea la posibilidad de manejo de movimientos bancarios con divisas extranjeras, permitiendo a los usuarios la visualización del monto en moneda internacional y su conversión nacional acorde al tipo de cambio actual.  | Baja | Media |
| Inicio de sesión por medio de redes sociales.   | Con la finalidad de otorgar flexibilidad a los usuarios registrados en el sistema, se plantea el inicio de sesión por medio de redes sociales vinculadas a la cuenta registrada.   | Alta | Baja  |

### ***Drivers de atributos de calidad***

Acorde a lo estipulado por *Cervantes, H., Velasco, P. y Castro, L.* en *Arquitectura de Software, Conceptos y ciclo de desarrollo* en 2016, los drivers de atributos de calidad se definen como un subconjunto de los requerimientos de atributos de calidad, acotando aquellos que resulten más relevantes para la satisfacción de los objetivos del negocio del sistema.

| <b>Driver de atributo de calidad</b>                                 | <b>Descripción</b>  | <b>Complejidad</b> | <b>Prioridad</b> |
|--|---|--------------------|------------------|
| Rechazo de cualquier usuario no registrado o inactivo en el sistema. | Atributo de calidad: <b>Security</b><br>Con la finalidad de evitar brechas de seguridad en el sistema y comprometer la información almacenada en el mismo, se | Media              | Alta             |

|   |   |       |      |
|---|---|-------|------|
|   | plantea el rechazo de todo aquel usuario no registrado en la base de datos o que se encuentre inactivo, pues las credenciales de acceso serían inválidas.   |       |      |
| Manejo de datos sensibles con hash en base de datos.  | Atributo de calidad: <b>Security</b><br>Con la finalidad de evitar comprometer la información de carácter sensible (contraseñas, fondos de cuenta, etc.) se plantea el resguardo de información en la base de datos con un hash de tipo BCrypt con un factor de costo 10.   | Baja  | Alta |
| Las transacciones deberán poderse efectuar aún si el monto de la divisa extranjera no puede ser calculado al momento de su generación.  | Atributo de calidad: <b>Availability</b><br>Con la finalidad de garantizar la disponibilidad de la generación de pagos/transacciones en todo momento aún cuando no sea posible contactar la API encargada de calcular el monto de la divisa nacional a su equivalente en moneda internacional, se plantea el despliegue de posibles acciones ejecutables por el usuario en caso de detonación de error, permitiendo así continuar con la operación sin su equivalente internacional o reintentar la operación con una nueva petición al servicio externo. | Media | Alta |
| Preparar el sistema para incorporar o editar módulos a la estructura base del manejo de pagos/transacciones bancarias de manera óptima. | Atributo de calidad: <b>Modifiability</b><br>Con la finalidad de optimizar los costos en producción para la generación de modificaciones o aditamentos de módulos externos al módulo principal de procesamiento de pagos, se plantea utilizar un estilo arquitectónico que permita realizar dichas modificaciones reutilizando código, evitando en todo momento alterar los módulos que no requieran modificaciones y respetando la funcionalidad establecida para el módulo base de procesamiento de movimientos bancarios.                              | Alta  | Alta |
| Las operaciones bancarias no deberán de tomar más de 5 segundos para ser  | Atributo de calidad: <b>Performance</b><br>Con la finalidad de garantizar la exactitud de la información bancaria manejada en cada transacción, se plantea que al crear un nuevo  | Alta  | Alta |

|   |  |      |       |
|---|--|------|-------|
| creadas o continuar con el flujo establecido de estados conforme a la acción efectuada. | movimiento y en cada actualización de estado que este pueda tener (conforme a la interacción de los usuarios autorizados), el tiempo de actualización y/o respuesta no sea mayor a 5 segundos.   |      |       |
| Uso intuitivo y práctico de la interfaz de usuario para realización de operaciones.     | Atributo de calidad: <b>Usability</b><br>Con la finalidad de que cualquier usuario con conocimientos de las operaciones que efectúa Smart Pay Co, así como los servicios que brinda pueda hacer uso del sistema sin capacitación previa, se plantea el diseño de una interfaz intuitiva que le permita a cualquier miembro del mismo navegar y hacer uso de los servicios del sistema únicamente mediante la información desplegada en la interfaz de usuario.             | Alta | Media |
| Optimizar tiempos de ejecución de pruebas para el mantenimiento del sistema.            | Atributo de calidad: <b>Testability</b><br>Con la finalidad de optimizar los tiempos de ejecución de pruebas para el mantenimiento del sistema y/o el aseguramiento de calidad en modificaciones entrantes al mismo, se plantea el aislamiento de funcionalidad del código por módulos que cumplan con las buenas prácticas de POO y principios SOLID, garantizando así una correcta separación de funcionalidad en módulos, haciendo las pruebas lo más óptimas posibles. | Alta | Baja  |

## 2.2 Restricciones arquitectónicas

Acorde a lo estipulado por *Cervantes, H., Velasco, P. y Castro, L.* en *Arquitectura de Software, Conceptos y ciclo de desarrollo* en 2016, las restricciones arquitectónicas son aquellos aspectos que limitan el proceso del desarrollo del sistema.

### ***Restricciones técnicas***

Acorde a lo estipulado por *Cervantes, H., Velasco, P. y Castro, L.* en *Arquitectura de Software, Conceptos y ciclo de desarrollo* en 2016, las restricciones técnicas son aquellas que a menudo

expresan solicitudes sobre el uso de productos de software provistos por terceros, métodos de implementación o diseño, productos de hardware o lenguajes de programación.

| Restricción técnica   | Descripción  |
|---|--|
| El hosting del programa no debe rebasar los \$4,000.00 pesos por año.   | Con la finalidad de optimizar los costos en su totalidad, se plantea el uso de un servicio de hosting que permita desplegar el sistema y acceder al mismo en cualquier momento con un precio anual menor a \$4,000.00 pesos.   |
| El lenguaje de programación deberá ser JavaScript y el framework elegido para el desarrollo deberá ser Express.js en conjunto con React.js  | Con la finalidad de que el sistema mantenga una vida útil duradera, considerando las tendencias tecnológicas actuales, se plantea el uso del entorno Node.js, mismo que funciona con el lenguaje de programación JavaScript. Del mismo modo, se usará el framework Express.js para el trabajo del back-end, mientras que se hará uso de React.js para el trabajo del front-end del sistema. Debido a que ambas herramientas son consideradas emergentes y a su amplia comunidad en internet, el mantenimiento de las mismas será bastante ameno.         |
| El sistema deberá ser desplegable desde cualquiera de los siguientes navegadores: <ul style="list-style-type: none"> <li>- Google Chrome (a partir de versión 100.0.4896.127).</li> <li>- Microsoft Edge (a partir de versión 100.0.1185.44).</li> <li>- Mozilla Firefox (a partir de versión 99.0.1).</li> </ul> | Con la finalidad de que cualquier usuario miembro de la empresa Smart Pay Co. pueda acceder al sistema desde cualquier ordenador de la misma, utilizando los navegadores aprobados por las normas de la compañía, se plantea la compatibilidad del programa con cualquiera de los navegadores establecidos a partir de su última versión estable al momento de creación de este documento. De este modo, se prevee un funcionamiento adecuado y un despliegue visual correcto de la interfaz de usuario con independencia de los navegadores utilizados. |
| El despliegue deberá ser únicamente para ordenadores u ordenadores portátiles.  | Con la finalidad de utilizar únicamente los dispositivos aprobados y utilizados en las instalaciones de Smart Pay Co. se plantea el despliegue del sistema vía web con un diseño de interfaz pensado únicamente para ordenadores y ordenadores portátiles, dejando fuera de alcance de desarrollo el diseño para dispositivos móviles y/o posible implementación de aplicación para las tiendas virtuales de los mismos.   |

***Restricciones administrativas***

Acorde a lo estipulado por *Cervantes, H., Velasco, P. y Castro, L.* en *Arquitectura de Software, Conceptos y ciclo de desarrollo* en 2016, las restricciones administrativas son aquellas que a menudo expresan restricciones sobre el costo y tiempo de desarrollo, al igual que las que comprenden al equipo de desarrollo.

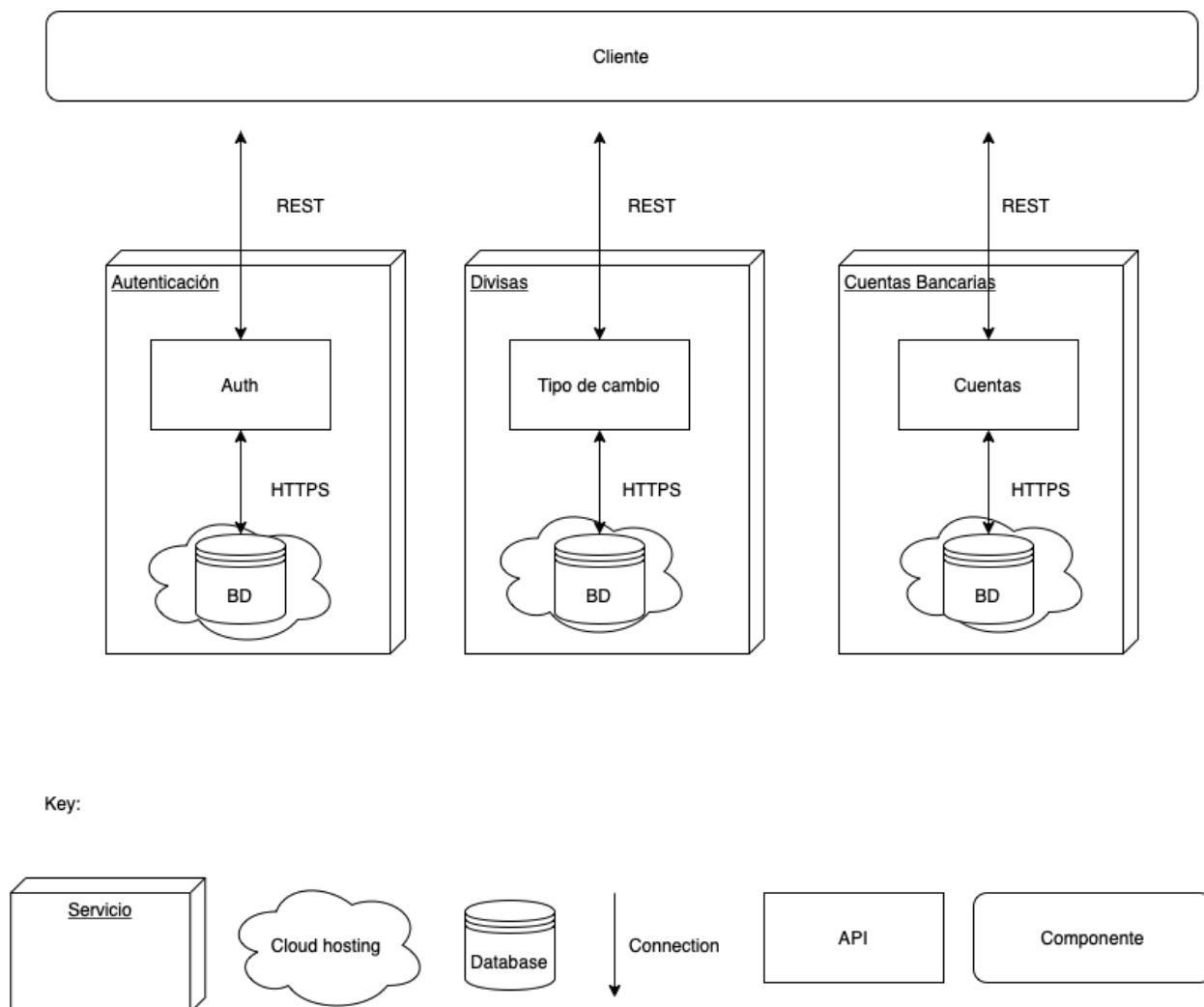
| <b>Restricción administrativa</b>  | <b>Descripción</b>  |
|--|---|
| El programa deberá entregarse en un plazo no mayor a 4 meses.  | Con la finalidad de entregar el programa dentro del tiempo límite para su despliegue en producción (antes del término del semestre) se plantea que el proyecto no tenga una implementación mayor a 4 meses en desarrollo para su entrega final.   |
| El equipo de desarrollo tendrá un máximo de 3 accesos simultáneos a la VPN empresarial de Smart Pay Co.  | Con la finalidad de no sobrepasar el límite de accesos permitidos por el departamento de TI de la empresa, se deberá trabajar en el desarrollo del programa con un máximo de 3 accesos simultáneos a la VPN empresarial para el despliegue del producto o modificaciones a las herramientas requeridas para el mismo en el tiempo de producción.      |
| Los servicios de desarrollo deberán ser brindados únicamente por el equipo de desarrollo, sin posibilidad de contratación externa de terceros. | Con la finalidad de garantizar la seguridad e integridad de la información autorizada por Smart Pay Co. se plantea que los ingenieros involucrados en el desarrollo del sistema sean provenientes en su totalidad por la empresa desarrolladora, sin posibilidad de autorizar a laborar en el proyecto a terceros ajenos a la empresa desarrolladora. |



## 2.3 Estilo Arquitectónico y rationale

*Describe el estilo arquitectónico seleccionado y la razón por la cual lo seleccionó*

El estilo arquitectónico seleccionado para la implementación de Smart Pay Co, basado en el análisis realizado es el patrón de microservicios. A continuación se muestra un diagrama que ejemplifica la estructura genérica de este patrón, así como el diagrama específico de este proyecto.



Las características de este estilo que son de especial interés para el desarrollo de este proyecto se explican a continuación.

En primer lugar, su alto nivel de agilidad. La aplicación se encuentra separada en múltiples unidades desplegadas o componentes de servicio que pueden ser desarrolladas, probadas y desplegadas de forma independiente a otras unidades. Esta es una característica valiosa pues tener estos componentes débilmente acoplados implica que podemos desarrollarlos rápidamente y poder entregarlo en tiempo y forma. Asimismo, implica que en caso de recibir retroalimentación por parte de los clientes, podremos realizar los cambios pertinentes de forma efectiva.

En segundo lugar, resultan atractivos en este patrón tanto su facilidad de desarrollo como de despliegue. Debido a que la funcionalidad está aislada en unidades, el desarrollo se mantiene sencillo pues su alcance es pequeño. Esto implica que la asignación de actividades y el trabajo en equipo es eficiente pues requiere menos coordinación y existe una probabilidad muy baja de que un cambio en un módulo afecte a otro, pues la funcionalidad está aislada. Lo anterior también implica que los despliegues se puede hacer de forma progresiva con un riesgo menor, pues el diseño modular implica la rápida identificación y corrección de errores, así como un alcance de los errores reducido pues afectarán únicamente al servicio desplegado.

En tercer lugar, este diseño para pruebas resulta ideal. Eso se debe a la separación de funcionalidad en aplicaciones independientes por lo que el alcance de las pruebas puede ser ajustado permitiendo así alta especificidad en lo que se desea probar. Asimismo el desacoplamiento de este patrón implica que hay bajas probabilidades de que una funcionalidad afecte a otra, por lo que las pruebas se pueden acotar a módulos específicos, sin tener que probar nuevamente toda la aplicación por un cambio menor.

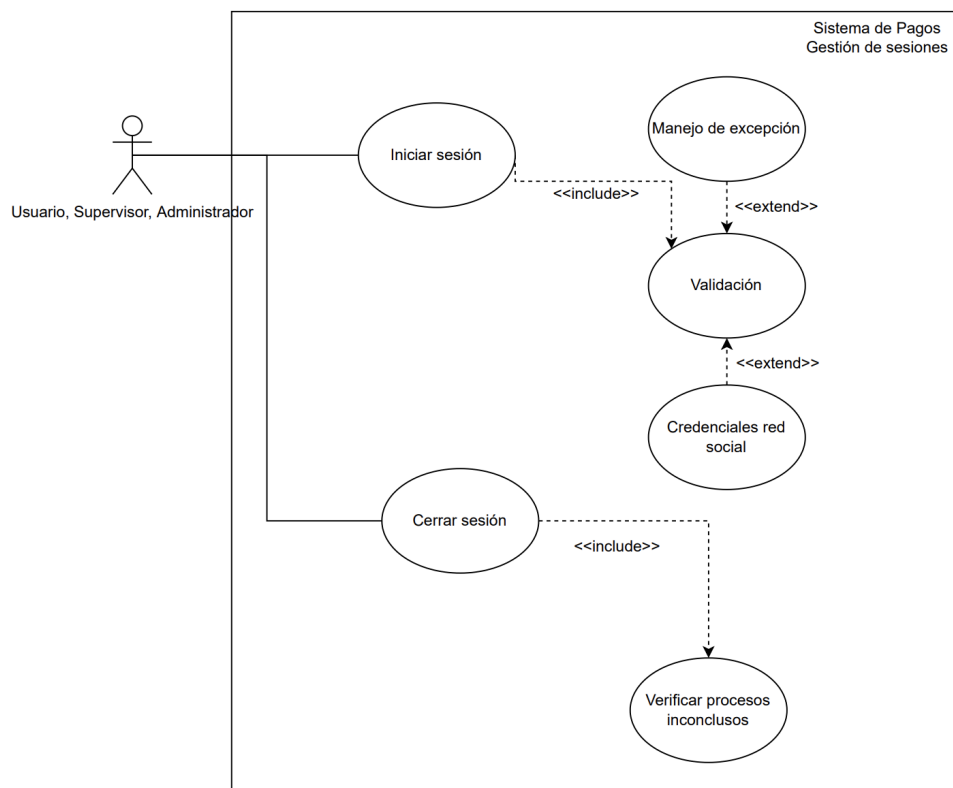
## **2.4 Vista de casos de uso**

Smart Pay Co, una institución financiera, ha enfocado su departamento de desarrollo de software en la creación de un nuevo sistema de pagos con soporte de divisas y aprobaciones de pagos. Existen tres

tipos de usuario del sistema, el usuario básico, el supervisor y el administrador. A continuación se describe cada uno de ellos.

- Usuario: categoría más baja, acceso a operaciones de creación de transacciones entre cuentas (explicado el sistema de cuentas más abajo).
- Supervisor: Incluyendo los permisos del Usuario, este usuario es capaz de aprobar dichas transacciones determinadas, este tipo de usuario se utiliza como filtro extra para corroborar que la transacción pueda realizarse con los fondos necesarios y/o que la información sea íntegra.
- Administrador: Tipo de usuario con todos los permisos disponibles, agregando la capacidad de agregar, editar, visualizar y borrar cuentas de usuario.

## Gestión de sesiones



Key: UML

|  |
|--|
| <b>ESCENARIO I: Gestión de sesiones</b>  |
| <b>Casos de uso 1:</b> Inicio de sesión  |
| <b>Descripción:</b> Permitir al usuario iniciar sesión para acceder al sistema con usuario y contraseña, o bien autenticación por medio de una red social.   |
| <b>Precondiciones:</b> El repositorio de información de usuarios debe estar actualizada y disponible.  |
| <b>Condición de éxito:</b> Usuario ingresa al sistema de forma exitosa.  |
| <b>Datos mostrados en pantalla:</b> Previo al inicio de sesión se muestra la interfaz de inicio de sesión con los campos nombre de usuario, contraseña y el botón de iniciar sesión. Tras el inicio de sesión exitoso, se muestra en pantalla la interfaz principal del sistema.   |
| <b>Stakeholders e interesados:</b> <ul style="list-style-type: none"> <li>• Compañía: Desea dar acceso únicamente a empleados por medio del inicio de sesión.</li> <li>• Empleados: Desean acceder al sistema para desempeñar su trabajo.</li> </ul>   |
| <b>Actores primarios:</b> Usuario base, Supervisor, Administrador  |
| <b>Disparador:</b> El usuario ha seleccionado la opción de inicio de sesión.   |
| <b>Eventos normales (Narrativa detallada):</b> <ol style="list-style-type: none"> <li>1.1 El usuario ingresa al sistema para iniciar sesión.</li> <li>1.2 El usuario ingresa su nombre de usuario, contraseña y selecciona iniciar sesión.</li> <li>1.3 El sistema valida que el usuario esté registrado en el repositorio de información y sus credenciales son correctas.</li> <li>1.4 El sistema redirige al usuario a la interfaz principal del software Smart Pay Co.</li> </ol>  |
| <b>Extensiones o flujos alternativos:</b> <ol style="list-style-type: none"> <li>1.2a Campo de usuario o contraseña no fueron llenados.<br/>El sistema muestra un mensaje de error en pantalla que pide al usuario verificar los campos.</li> <li>1.3a Datos incorrectos.<br/>Se despliega un mensaje de error en pantalla indicando al usuario que no es posible iniciar sesión pues su nombre de usuario o contraseña son incorrectos.</li> <li>1.3b Datos del usuario no encontrados en repositorio de información.</li> <li>1.2a El usuario desea iniciar sesión por medio de cuenta de Google<br/>El usuario da click en el botón de inicio de sesión con Google. El sistema despliega la interfaz de inicio de sesión. El usuario coloca su cuenta y contraseña de Google y da click en el botón de iniciar</li> </ol> |

sesión.

## ESCENARIO I: Gestión de sesiones

### Casos de uso 2: Cierre de sesión

**Descripción:** Permitir al usuario cerrar sesión.

**Precondiciones:** El usuario debe haber iniciado sesión previamente.

**Condición de éxito:** El sistema cierra correctamente la sesión del usuario y redirige a la interfaz de inicio de sesión.

**Datos mostrados en pantalla:** Previo al cierre de sesión debe estar visible el botón correspondiente. Una vez cerrada la sesión, el usuario es redirigido a la interfaz de inicio de sesión.

### Stakeholders e interesados:

- Compañía: Desea terminar la sesión de empleados por turno.
- Empleado: Desea terminar su sesión por término de turno.

**Actores primarios:** Usuario base, Supervisor, Administrador

**Disparador:** El usuario ha seleccionado la opción de salir.

### Eventos normales (Narrativa detallada):

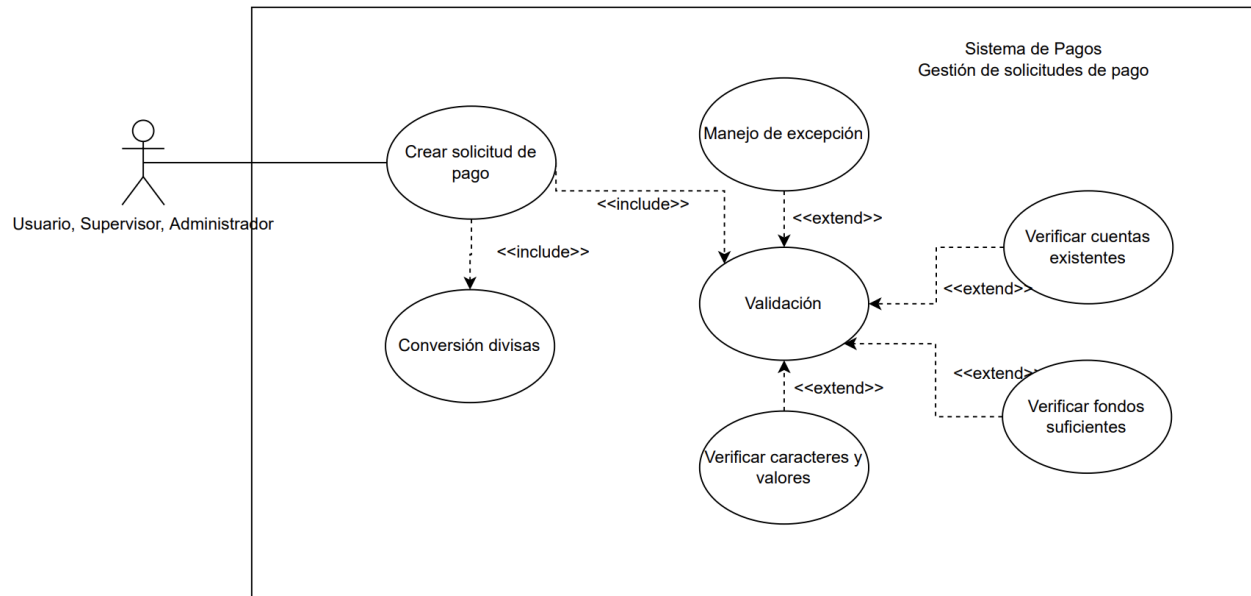
- 1.1 El usuario ha iniciado sesión exitosamente.
- 1.2 El usuario navega hasta la interfaz principal.
- 1.3 El usuario selecciona la opción de salir.
- 1.4 El sistema verifica que el usuario no cuente con procesos inconclusos.
- 1.5 El sistema cierra la sesión y redirige al usuario a la interfaz de inicio de sesión.

### Extensiones o flujos alternativos:

1.4 a El usuario cuenta con un proceso inconcluso.

El sistema desplegará un mensaje de error alertando al usuario que debe concluir con el proceso para poder cerrar sesión.

## Gestión de solicitudes de pago

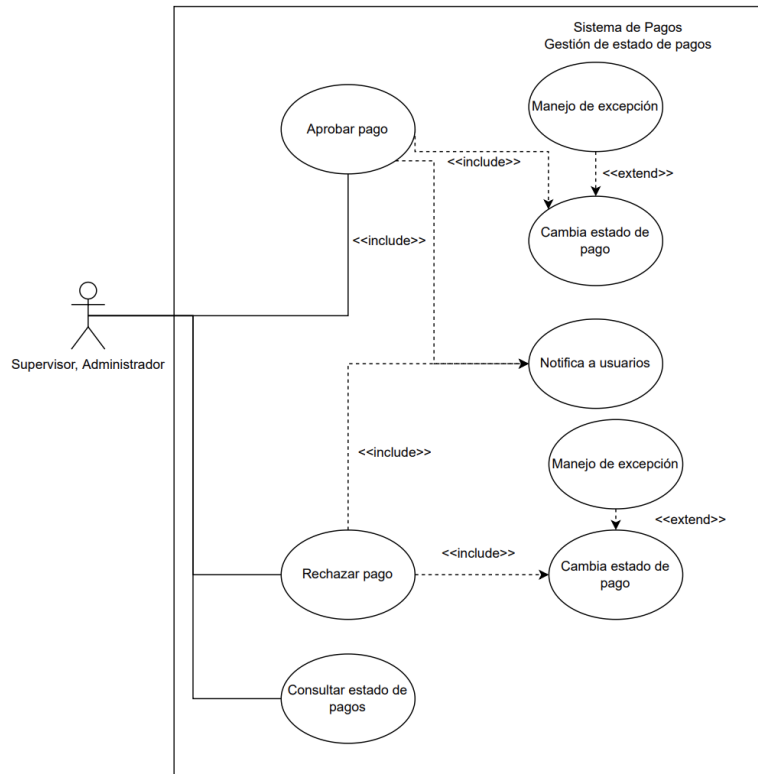


Key: UML

|  |
|--|
| <b>ESCENARIO II: Gestión de solicitudes de pago</b>  |
| <b>Casos de uso 1:</b> Crear solicitud de pago   |
| <b>Descripción:</b> Permitir al usuario crear una solicitud de pago.   |
| <b>Precondiciones:</b> El usuario debe haber iniciado sesión previamente.  |
| <b>Condición de éxito:</b> Se genera correctamente una solicitud de pago que incluye cuenta destino, cuenta ordenante, monto y divisa.   |
| <b>Datos mostrados en pantalla:</b> Al seleccionar la opción de crear solicitud de pago se despliega un formulario de pago que contiene la cuenta ordenante, cuenta destino, divisa, monto, monto total en MXN y un botón para confirmar la solicitud.                                     |
| <b>Stakeholders e interesados:</b> <ul style="list-style-type: none"> <li>• Compañía: Desea ser capaz de procesar transacciones internacionales.</li> <li>• Empleados: Desea solicitar los pagos de sus clientes y ganar comisiones.</li> <li>• Clientes: Desea efectuar pagos.</li> </ul> |

|  |
|--|
| <b>Actores primarios:</b> Usuario base, Supervisor, Administrador  |
| <b>Disparador:</b> El usuario ha seleccionado la opción de nueva solicitud de pago.  |
| <b>Eventos normales (Narrativa detallada):</b><br>1.1 El usuario ha iniciado sesión exitosamente.<br>1.2 El usuario navega hasta la interfaz principal.<br>1.3 El usuario selecciona la opción de nueva solicitud de pago.<br>1.4 El usuario llena el formulario con el monto, la cuenta destino y la divisa del pago.<br>1.5 El sistema verifica que tanto la cuenta destino como el monto sean valores numéricos.<br>1.6 El sistema verifica en tiempo real el valor de las divisas, realiza la conversión a moneda nacional.<br>1.7 El sistema valida que las cuentas origen y destino sean existentes.<br>1.8 El sistema valida que la cuenta de origen cuenta con los fondos suficientes para efectuar la transacción.<br>1.9 El sistema registra la solicitud de pago.   |
| <b>Extensiones o flujos alternativos:</b><br>1.5a Sistema encuentra caracteres inválidos (no numéricos), o los campos no fueron llenados.<br>El sistema despliega un mensaje de error indicando al usuario que debe revisar los campos del formulario.<br>1.6a Sistema no se comunica con el sistema de divisas.<br>El sistema despliega un mensaje de error indicando que existe un problema de comunicación que se intente realizar la operación más tarde.<br>1.7a Cuenta de origen o destino inexistente.<br>El sistema despliega un mensaje de error que indica que la cuenta de origen o destino no es válida.<br>1.8a Cuenta de origen no cuenta con fondos insuficientes.<br>El sistema despliega un mensaje de error indicando que la cuenta de origen no cuenta con fondos suficientes para efectuar la transacción. |

## Gestión de estado de pagos



Key: UML

|  |
|--|
| <b>ESCENARIO III: Gestión de estado de pagos</b>   |
| <b>Casos de uso 1:</b> Consultar estado de pagos   |
| <b>Descripción:</b> Permitir al usuario consultar el pago y su estado.   |
| <b>Precondiciones:</b> El usuario debe haber iniciado sesión previamente y contar con privilegios de supervisor o administrador.   |
| <b>Condición de éxito:</b> El sistema despliega correctamente el pago consultado, mostrando cuenta destino, cuenta origen, monto, divisa y estado (rechazado, aprobado, completado). |



|  |
|--|
| <b>Datos mostrados en pantalla:</b> Una vez que se consulta el estado de un pago se despliega en la pantalla el identificador del pago, la cuenta ordenante, cuenta destino, monto, divisa, monto total en MXN y un estado.  |
| <b>Stakeholders e interesados:</b> <ul style="list-style-type: none"> <li>• Compañía: Desea contar con estadísticas y un registro histórico de transacciones.</li> <li>• Empleados: Desea consultar transacciones de sus clientes.</li> <li>• Clientes: Desea saber el estado de sus pagos realizados.</li> </ul>  |
| <b>Actores primarios:</b> Supervisor, Administrador  |
| <b>Disparador:</b> El usuario ha seleccionado la opción de consultar pago.   |
| <b>Eventos normales (Narrativa detallada):</b> <ol style="list-style-type: none"> <li>1.1 El usuario ha iniciado sesión exitosamente.</li> <li>1.2 El usuario navega hasta la interfaz principal.</li> <li>1.3 El usuario selecciona la opción de consultas y selecciona un pago.</li> <li>1.4 El sistema despliega en pantalla el estado del pago (rechazado, aprobado, completado).</li> </ol> |
| <b>Extensiones o flujos alternativos:</b> <ol style="list-style-type: none"> <li>1.3a Sin conexión con el repositorio de información.<br/>El sistema despliega un mensaje que indica que no hay comunicación con el repositorio de información, que se intente realizar la operación más tarde.</li> </ol>   |

|  |
|--|
| <b>ESCENARIO III: Gestión de estado de pagos</b>   |
| <b>Casos de uso 2:</b> Aprobar pago  |
| <b>Descripción:</b> Permitir al usuario aprobar pagos de sus clientes.   |
| <b>Precondiciones:</b> El usuario debe haber iniciado sesión previamente y contar con privilegios de supervisor o administrador.   |
| <b>Condición de éxito:</b> El estado del pago es actualizado correctamente a aprobado.   |
| <b>Datos mostrados en pantalla:</b> Previo a la aprobación del pago se despliega en pantalla el identificador del pago, la cuenta ordenante, la cuenta destino, el monto, la divisa, el monto total en |

MXN, así como un botón para aprobar el pago. Una vez efectuada la operación se debe mostrar el estado del pago como aprobado..

**Stakeholders e interesados:**

- Compañía: Desea autorizar los pagos provenientes de cuentas con fondos suficientes.
- Empleados: Desea aprobar pagos de sus clientes para hacerse acreedor a una comisión.
- Clientes: Desea la pronta aprobación de las transacciones realizadas.

**Actores primarios:** Supervisor, Administrador

**Disparador:** El usuario ha seleccionado la opción de gestionar el estado de un pago.

**Eventos normales (Narrativa detallada):**

- 1.1 El usuario ha iniciado sesión exitosamente.
- 1.2 El usuario navega hasta la interfaz principal.
- 1.3 El usuario selecciona la opción de gestionar pagos.
- 1.4 El usuario verifica el monto, cuenta destino, divisa y selecciona aprobar el pago.
- 1.5 El sistema cambia el estado del pago a completado.
- 1.6 El sistema envía una notificación a los usuarios vía correo electrónico indicándoles que el pago con la cuenta destino, monto y divisa ha sido completado correctamente.

**Extensiones o flujos alternativos:**

- 1.5a Sin conexión con el repositorio de información.  
El sistema despliega un mensaje que indica que no hay comunicación con el repositorio de información, que se intente realizar la operación más tarde.
- 1.6a Sin conexión con el módulo de notificaciones.  
El sistema despliega un mensaje que indica que no hay comunicación con el módulo de notificaciones.

**ESCENARIO III: Gestión de estado de pagos**

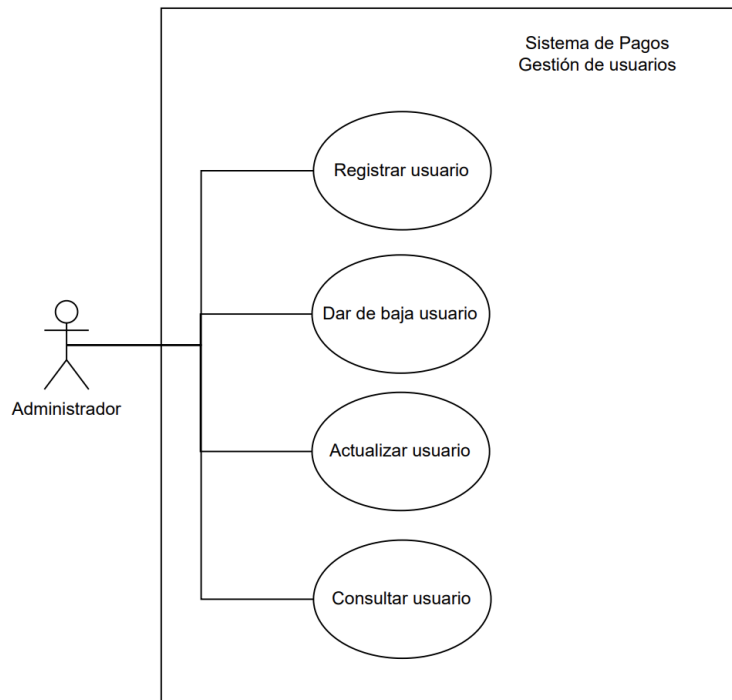
**Casos de uso 3:** Rechazar pago

**Descripción:** Permitir al usuario rechazar pagos de sus clientes.

**Precondiciones:** El usuario debe haber iniciado sesión previamente y contar con privilegios de supervisor o administrador.

|  |
|--|
| <b>Condición de éxito:</b> El estado del pago ha sido actualizado correctamente a rechazado.   |
| <b>Datos mostrados en pantalla:</b> Previo al rechazo del pago se despliega en pantalla el identificador del pago, la cuenta ordenante, la cuenta destino, el monto, la divisa, el monto total en MXN, así como un botón para rechazar el pago. Una vez efectuada la operación se debe mostrar el estado del pago como rechazado.  |
| <b>Stakeholders e interesados:</b> <ul style="list-style-type: none"><li>• Compañía: Desea rechazar operaciones fraudulentas.</li><li>• Empleado: Desea rechazar operaciones sin fondos suficientes en la cuenta de origen.</li><li>• Cliente: Desea cancelar operaciones hechas por fraude, robo o clonación.</li></ul>   |
| <b>Actores primarios:</b> Supervisor, Administrador  |
| <b>Disparador:</b> El usuario ha seleccionado la opción de gestionar el estado de un pago.   |
| <b>Eventos normales (Narrativa detallada):</b> <ol style="list-style-type: none"><li>1.1 El usuario ha iniciado sesión exitosamente.</li><li>1.2 El usuario navega hasta la interfaz principal.</li><li>1.3 El usuario selecciona la opción de gestionar pagos.</li><li>1.4 El usuario verifica la cuenta destino, el monto, la divisa y selecciona rechazar el pago.</li><li>1.5 El sistema cambia el estado del pago a rechazado</li><li>1.6 El sistema envía una notificación a los usuarios vía correo electrónico indicándoles que el pago con la cuenta destino, monto y divisa ha sido rechazado.</li></ol> |
| <b>Extensiones o flujos alternativos:</b> <ol style="list-style-type: none"><li>1.5a Sin conexión con el repositorio de información.<br/>El sistema despliega un mensaje que indica que no hay comunicación con el repositorio de información, que se intente realizar la operación más tarde.</li><li>1.6a Sin conexión con el módulo de notificaciones.<br/>El sistema despliega un mensaje que indica que no hay comunicación con el módulo de notificaciones.</li></ol>  |

## Gestión de usuarios



Key: UML

|  |
|--|
| <b>ESCENARIO IV: Gestión de usuarios</b>   |
| <b>Casos de uso 1:</b> Crear usuarios  |
| <b>Descripción:</b> Permitir a administradores crear nuevos usuarios   |
| <b>Precondiciones:</b> El usuario debe haber iniciado sesión previamente y contar con privilegios de administrador. La base de datos de usuarios debe estar actualizada y disponible.  |
| <b>Condición de éxito:</b> Un nuevo usuario ha sido registrado correctamente.  |
| <b>Datos mostrados en pantalla:</b> Para crear un usuario se despliega en pantalla un formulario que contiene el nombre de usuario, la contraseña y además un campo donde se selecciona la categoría del usuario a crear es decir: usuario, supervisor, administrador. |
| <b>Stakeholders e interesados:</b> <ul style="list-style-type: none"> <li>• Compañía (RH): Desea crear las credenciales de usuario para comenzar sus labores.</li> </ul>   |
| <b>Actores primarios:</b> Administrador  |

**Disparador:** El administrador ha seleccionado la opción de alta de usuarios.

**Eventos normales (Narrativa detallada):**

- 1.1 El usuario ha iniciado sesión exitosamente.
- 1.2 El usuario navega hasta la interfaz principal.
- 1.3 El usuario ingresa a la opción para dar de alta un usuario.
- 1.4 El usuario ingresa el nombre de usuario, contraseña y tipo de cuenta a crear (usuario base, supervisor, administrador)
- 1.5 El sistema valida que el usuario no esté previamente registrado en el repositorio de información.
- 1.6 El sistema da de alta el usuario.

**Extensiones o flujos alternativos:**

- 1.5a Usuario registrado previamente.  
El sistema despliega un mensaje en pantalla que indica que ya existe un usuario con ese nombre de usuario.
- 1.6a Campo de nombre de usuario o contraseña vacíos.  
El sistema despliega un mensaje que le indica al usuario que debe llenar los campos faltantes.

**ESCENARIO IV: Gestión de usuarios**

**Casos de uso 2:** Baja de usuarios.

**Descripción:** Permitir a administradores dar de baja usuarios.

**Precondiciones:** El usuario debe haber iniciado sesión previamente y contar con privilegios de administrador. La base de datos de usuarios debe estar actualizada y disponible.

**Condición de éxito:** El usuario seleccionado ha sido dado de baja correctamente.

**Datos mostrados en pantalla:** Previo a la eliminación de un usuario se despliega en pantalla su nombre de usuario, su contraseña así como el tipo de cuenta, es decir: usuario, supervisor, o administrador, así como un botón para dar de baja al usuario.

**Stakeholders e interesados:**

- Compañía (RH): Desea desactivar empleados que ya no se encuentran activos en la empresa.

**Actores primarios:** Administrador

**Disparador:** El administrador ha seleccionado la opción de baja de usuarios.

**Eventos normales (Narrativa detallada):**

- 1.1 El usuario ha iniciado sesión exitosamente.
- 1.2 El usuario navega hasta la interfaz principal.
- 1.3 El usuario ingresa a la opción para dar de baja a un usuario.
- 1.4 El sistema envía un mensaje de confirmación de baja.
- 1.5 El usuario envía su confirmación de baja.
- 1.6 El sistema da de baja al usuario.

**Extensiones o flujos alternativos:**

- 1.5a Usuario cancela el proceso de baja  
El sistema cancela el proceso de baja y envía al usuario a la interfaz anterior.

**ESCENARIO IV: Gestión de usuarios****Casos de uso 3:** Cambio de usuarios.

**Descripción:** Permitir a administradores modificar usuarios.

**Precondiciones:** El usuario debe haber iniciado sesión previamente y contar con privilegios de administrador. El repositorio de información de usuarios debe estar actualizado y disponible.

**Condición de éxito:** El nombre de usuario y/o contraseña del usuario seleccionado ha sido actualizado correctamente.

**Datos mostrados en pantalla:** Previo al cambio del usuario se despliega el nombre de usuario, la contraseña y el tipo de cuenta que posee, en campos editables que pueden ser editados. Asimismo, se muestra un botón para guardar el cambio.

**Stakeholders e interesados:**

- Compañía (RH): Desea modificar los permisos del usuario de acuerdo con su puesto y rendimiento.
- Empleado: Desea modificar su contraseña de acceso al sistema.

**Actores primarios:** Administrador

**Disparador:** El administrador ha seleccionado la opción de actualización de usuarios.

**Eventos normales (Narrativa detallada):**

- 1.1 El usuario ha iniciado sesión exitosamente.
- 1.2 El usuario navega hasta la interfaz principal.
- 1.3 El usuario ingresa a la opción para actualizar a un usuario.
- 1.4 El sistema envía un mensaje de confirmación de actualización.
- 1.5 El usuario envía su confirmación de actualización.
- 1.6 El sistema verifica en el repositorio de información que no esté duplicada.
- 1.7 El sistema actualiza al usuario.

**Extensiones o flujos alternativos:**

- 1.5a Usuario cancela proceso de actualización.  
El sistema cancela el proceso de actualización y redirige al usuario a la interfaz anterior.
- 1.6a Información duplicada en repositorio de información.  
El sistema cancela el proceso de actualización y envía un mensaje notificando al usuario que no es posible realizar el cambio deseado pues su información está duplicada.

**ESCENARIO IV: Gestión de usuarios**

**Casos de uso 4:** Consulta de usuarios.

**Descripción:** Permitir a administradores consultar usuarios.

**Precondiciones:** El usuario debe haber iniciado sesión previamente y contar con privilegios de administrador. La base de datos de usuarios debe estar actualizada y disponible.

**Condición de éxito:** El sistema despliega correctamente la lista de usuarios registrados en el sistema.

**Datos mostrados en pantalla:** La consulta de usuarios muestra en pantalla el nombre de usuario, la contraseña y el tipo de usuario.

**Stakeholders e interesados:**

- Compañía: Desea consultar a sus empleados y sus facultades dentro del sistema.
- Empleados Administradores: Desean llevar un control de usuarios de sistema.

**Actores primarios:** Administrador

**Disparador:** El administrador ha seleccionado la opción de consulta de usuarios.

**Eventos normales (Narrativa detallada):**

- 1.1 El usuario ha iniciado sesión exitosamente.
- 1.2 El usuario navega hasta la interfaz principal.
- 1.3 El usuario ingresa a la opción para consultar usuarios.
- 1.4 El sistema despliega una lista de los usuarios registrados mostrando su nombre de usuario.

**Extensiones o flujos alternativos:**

- 1.4a Sin comunicación con el repositorio de información.

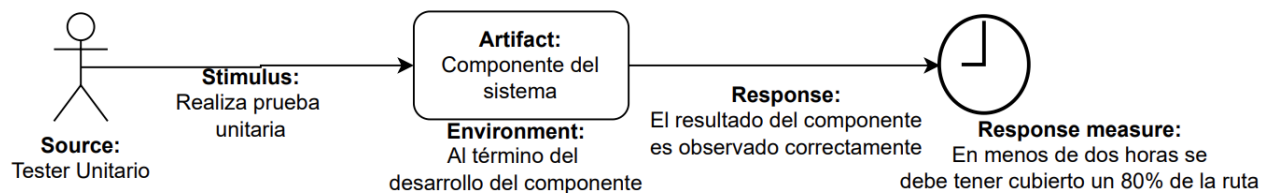
El sistema despliega un mensaje indicando al usuario que no hay comunicación, que intente la operación más tarde.



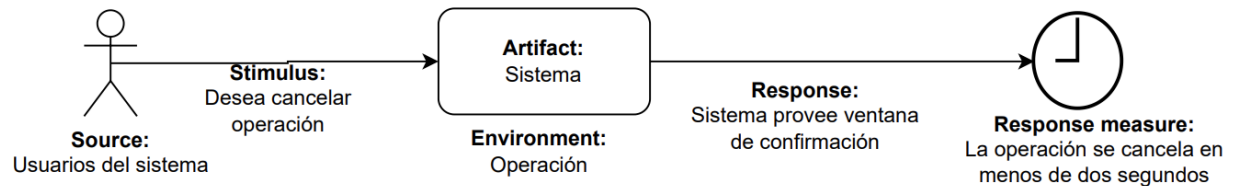
## 2.5 Escenarios Arquitectónicos y Requerimientos No Funcionales Asociados

### 2.5.1 Escenarios Arquitectónicos

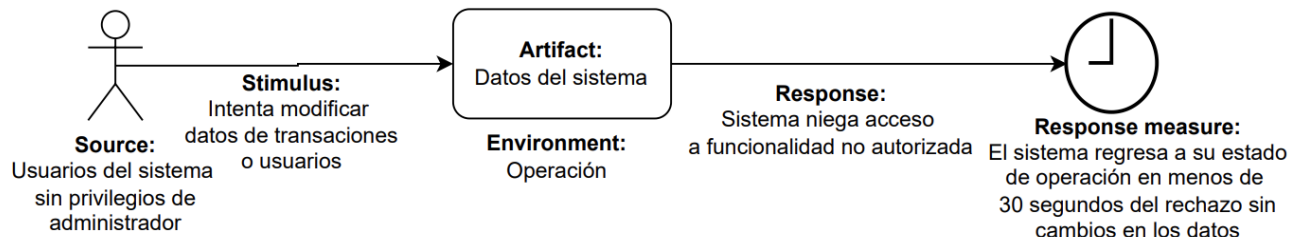
#### Testability



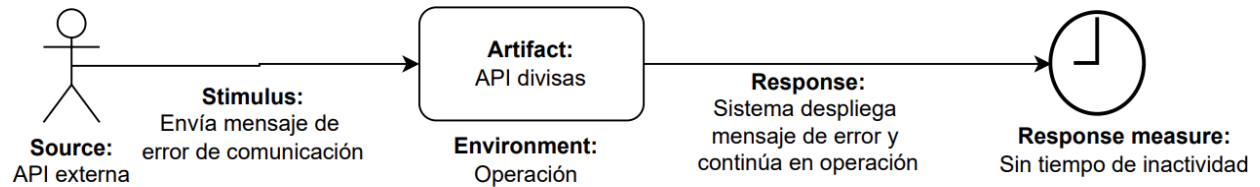
#### Usability



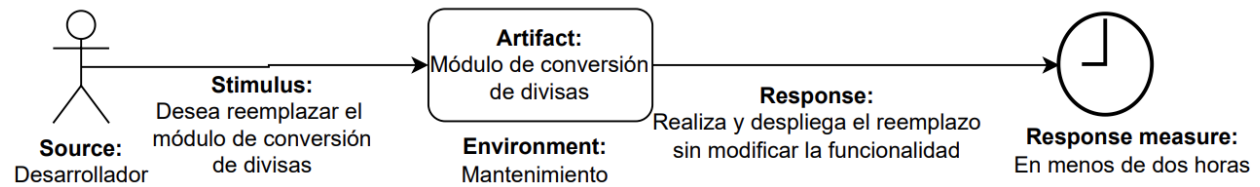
#### Security



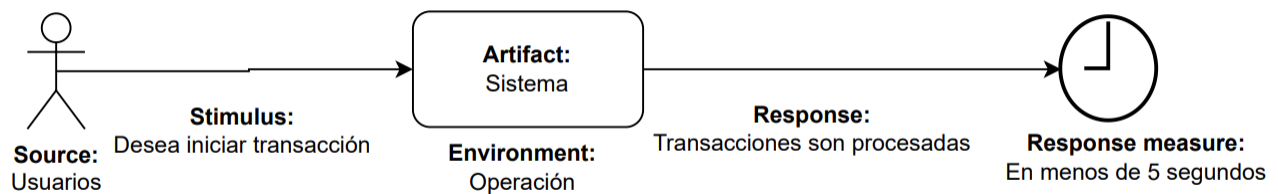
### Availability



### Modifiability



### Performance



## 2.5.2 Requerimientos No Funcionales Asociados

### Testability:

|   |  |
|---|--|
| RNF 0001  | El sistema podrá grabar qué operaciones se realizaron durante un tiempo para después ser consultada y verificar las fallas o errores que sucedan en la plataforma. |
| Registro de operaciones   |  |
| Durante la operaciones transaccionales, el sistema realizará la solicitud y registrará el sistema como se realizó la operación y en caso de existir una falla, se conozca que sucedió durante la operación y en un futuro se valide esta funcionalidad y pueda ser corregido. |  |

|   |   |
|---|---|
| <b>RNF 0002</b>   | <b>El tester realiza una prueba unitaria sobre un componente del sistema.</b> |
| <b>Registro de operaciones</b>  |   |
| Esta prueba debe ser realizada al término del desarrollo del componente para comprobar su funcionalidad. El resultado del componente debe ser observado correctamente y se debe tener cubierto el 80% de la ruta en menos de dos horas. |   |

### Usability:

|                                   |   |
|-----------------------------------|---|
| <b>RNF 0003</b>                   | <b>Los usuarios del sistema podrán cancelar operaciones que hayan iniciado por error o ya no deseen efectuar.</b> |
| <b>Operación de transacciones</b> |   |

Esto se realizará durante la operación del sistema donde antes de cada operación el sistema provee una ventana de confirmación donde el usuario selecciona registrar la operación o salir sin cambios y cancelar la operación. Una vez seleccionada la opción de cancelar o salir sin cambios, la operación es cancelada en menos de dos segundos.

**Availability:**

|  |  |
|--|--|
| <b>RNF 0004</b>  | <b>La aplicación solo será disponible durante las horas de trabajo del staff</b> |
| <b>Horas de servicio</b>   |  |
| Al realizar un registro de alguna transacción. Esta operación únicamente podrá ser realizada durante las horas de servicio del usuario. Fuera de las horas laborales, el sistema no permitirá el registro de pre aprobación o aprobación de la transacción hasta que las horas laborales reanuden. |  |

**Modifiability:**

|   |  |
|---|--|
| RNF 0005  | El sistema deberá estar comentado y documentado las partes desarrolladas dentro del proyecto. Ya sea de parte lógica o visual. |
| Documentación de código y métodos   |  |
| El código del sistema tendrá especificado que realiza cada documento y dentro del mismo una explicación de los métodos, funciones, etc. De acuerdo con el lenguaje utilizado. Esto para evitar que se realicen cambios (en caso de ser necesario) así evitando un <i>ripple effect</i> en las funcionalidades desarrolladas ya sea que afecte directa o indirectamente un comportamiento del sistema. |  |

**Performance:**

|   |   |
|---|---|
| <b>RNF 0006</b>   | <b>El sistema debe responder con velocidad la transacción cuando el usuario indique que se realice una operación del sistema.</b> |
| <b>Velocidad de carga de transacciones</b>  |   |
| Cuando el usuario decida que se empiece a realizar la transacción y los campos necesarios estén correctamente llenados, la aplicación será capaz de completar la solicitud en un lapso menor a 5 segundos. Durante ese tiempo, se desplegará una carga para dar a conocer que la operación está en un proceso de completarse. |   |

**Scalability:**

|   |   |
|---|---|
| <b>RNF 0007</b>   | <b>El sistema debe soportar cantidades considerables y simultáneas de operaciones dentro de la misma plataforma</b> |
| <b>Cantidad de transacciones</b>  |   |
| Durante un tiempo de operación, el sistema debe ser capaz de soportar alrededor de 50 mil operaciones y ser capaz de escalar sus operaciones cuando se requiera que más registros se realicen. De manera que si incrementa un 50% a 70% las transacciones, se pueda seguir realizando sin que falle el sistema. |   |

|   |   |
|---|---|
| RNF 0008  | El sistema debe ser capaz de registrar y contener grandes cantidades de usuarios independientemente del tipo de usuario que persiste. |
| Registro de usuarios  |   |
| Al momento que un administrador registre un nuevo usuario o supervisor, éste persistirá en la base de datos que contendrá hasta 50 GB de información de manera correcta sin importar la cantidad de usuarios existentes y sin colisiones. Al momento de autenticarse deberá identificar el usuario dentro del registro. |   |

**Interoperability:.**

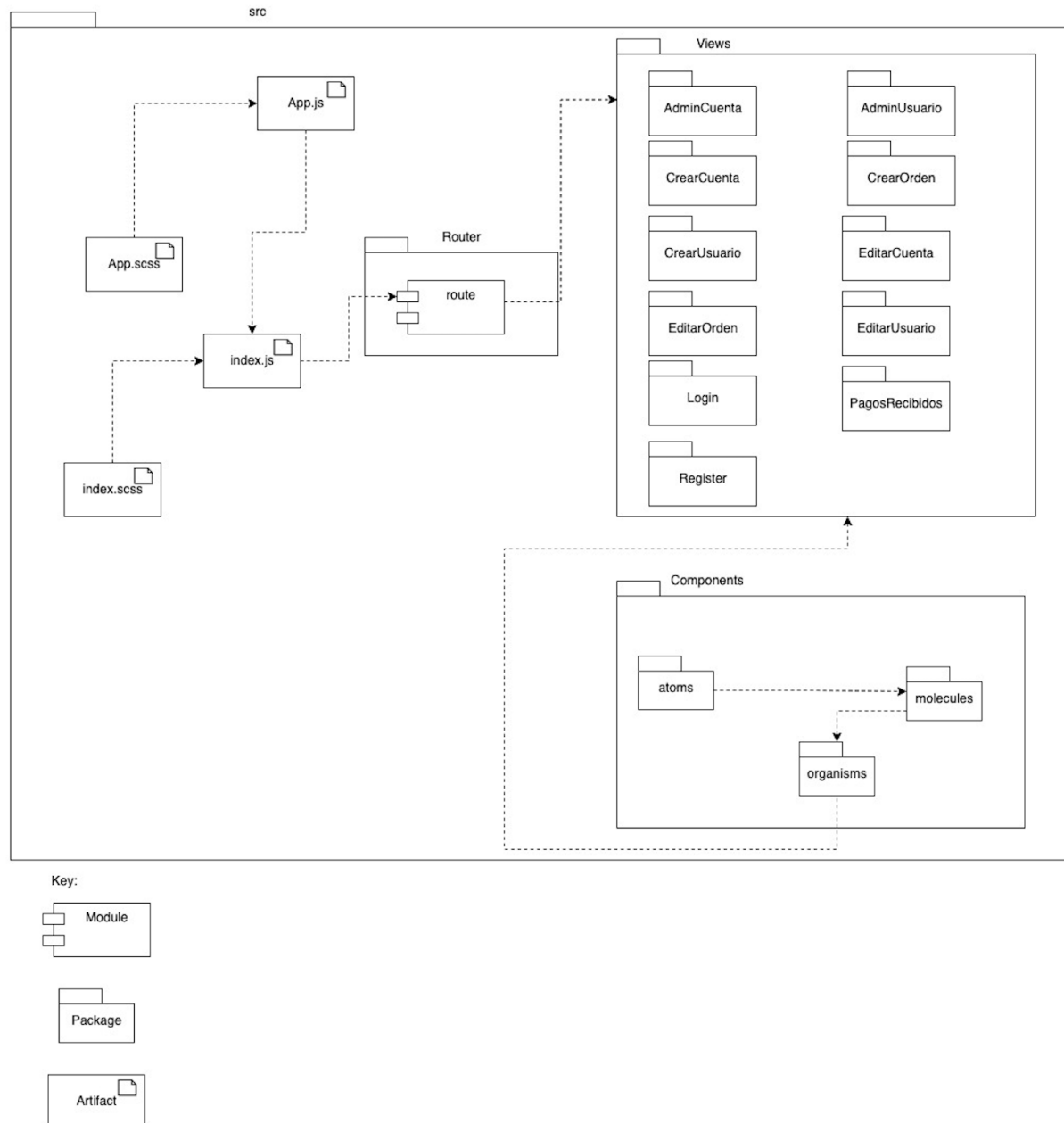
|  |   |
|--|---|
| <b>RNF 0009</b>  | <b>El sistema se podrá cargar desde distintos motores de navegación existente persistiendo con sus componentes funcionales y visuales</b> |
| <b>Visualización de dispositivos</b>   |   |
| El sistema será capaz de poder visualizarse en distintos navegadores como Chrome (versión 100.0.4896.127 y posteriores), Safari (versión 15.4 y posteriores) . Y cargará de manera intencionada en el servicio web e independientemente del sistema operativo que se utilice. El sistema podrá realizar todas las operaciones necesarias de manera correcta. |   |

**Security:**

|   |   |
|---|---|
| <b>RNF 0010</b>   | <b>El sistema debe detectar la autenticación de los usuarios en cada sesión abierta del sistema</b> |
| <b>Autenticación del Usuario</b>  |   |
| El usuario que no se encuentre registrado dentro de la base de datos no podrá acceder a ninguna funcionalidad del sistema. Esté deberá mostrar la aceptación o rechazo del registro en la pantalla de manera íntegra de su información.   |   |
| <b>RNF 0011</b>   | <b>El sistema debe mostrar los componentes que el usuario está autorizado visualizar.</b>           |
| <b>Autorización del Usuario</b>   |   |
| Un individuo identificado correctamente debe estar autorizado a las operaciones correspondientes y solo a esas del sistema. Si el usuario realiza alguna operación dentro del sistema, se validará con sus permisos de cuenta si está autorizado a realizarlo. El sistema realizará esto detectando el 100% de los casos. |   |

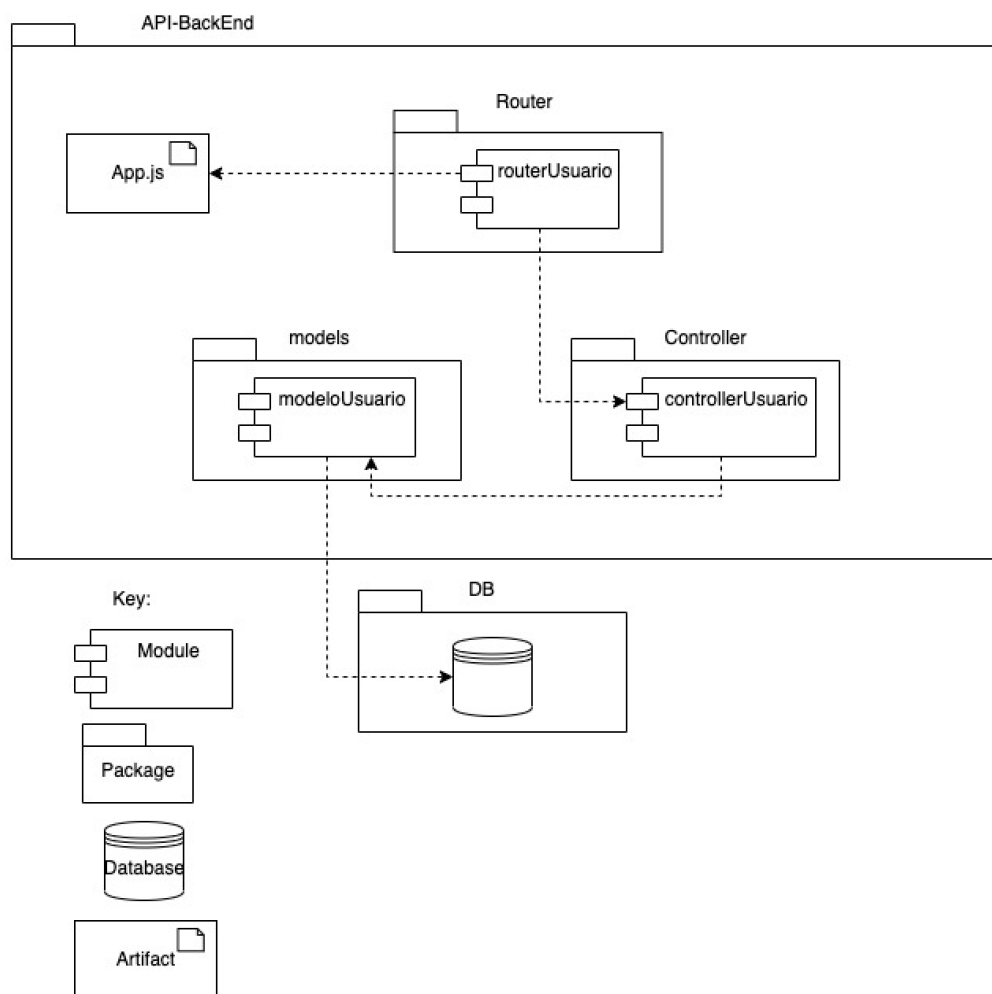
## 2.6 Vista lógica

### Vista lógica front end



El sistema lógico se encuentra estructurado por varias secciones, la parte de src que es como la vista principal, es lo primero que se llama cuando React corre el servicio web, de ahí se conecta a un enrutador que es el encargado de llamar las vistas de cada bloque de pantalla y accede dependiendo cual sea el requerido por el usuario. Cada vista cuenta con componentes llamados organismos. Estos se distribuyen dependiendo si la vista lo requiere o no, depende de la vista que lo llame para que este sea entregado por el componente.

### Vista lógica del back end



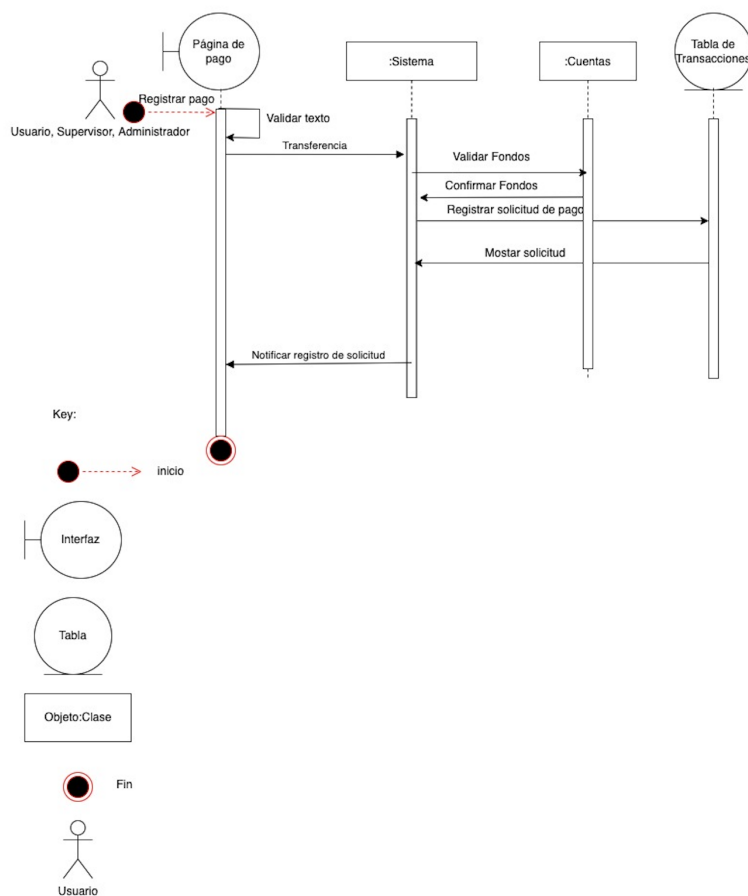


Existe un src que es el encargado de correr la estructura principal del sistema que después llama a unos enrutadores que son los determinantes de los endPoints y el redireccionamiento de información. Una vez realizado esto, el router accede al controlador que es el encargado de utilizar el modelo cuando se llega a un endPoint establecido. El modelo es únicamente la estructura que requiere el sistema y que por lo regular accede a una base de datos.

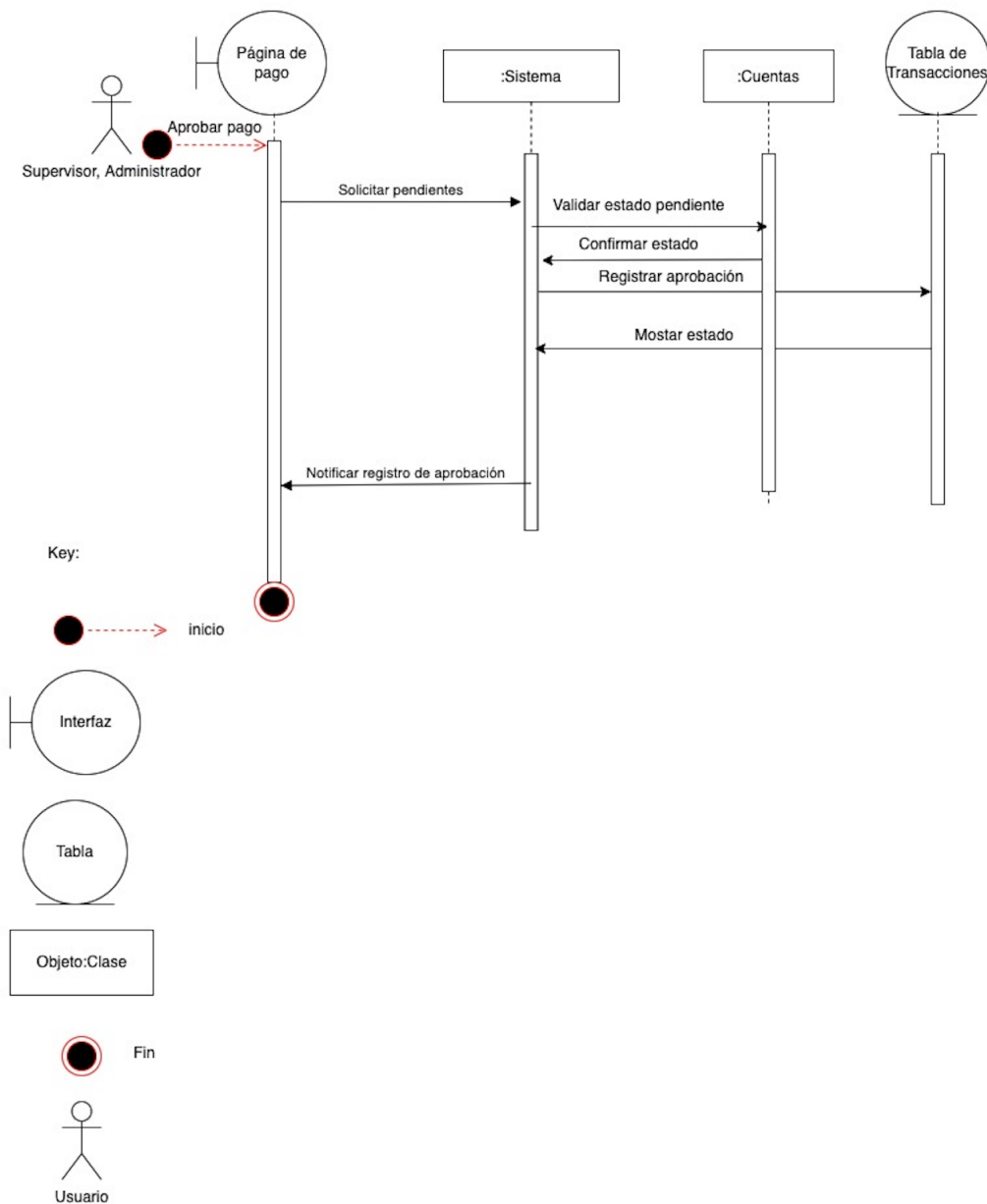
## 2.7 Vista de proceso

Utilizar un diagrama de secuencia o de actividad para mostrar las interacciones

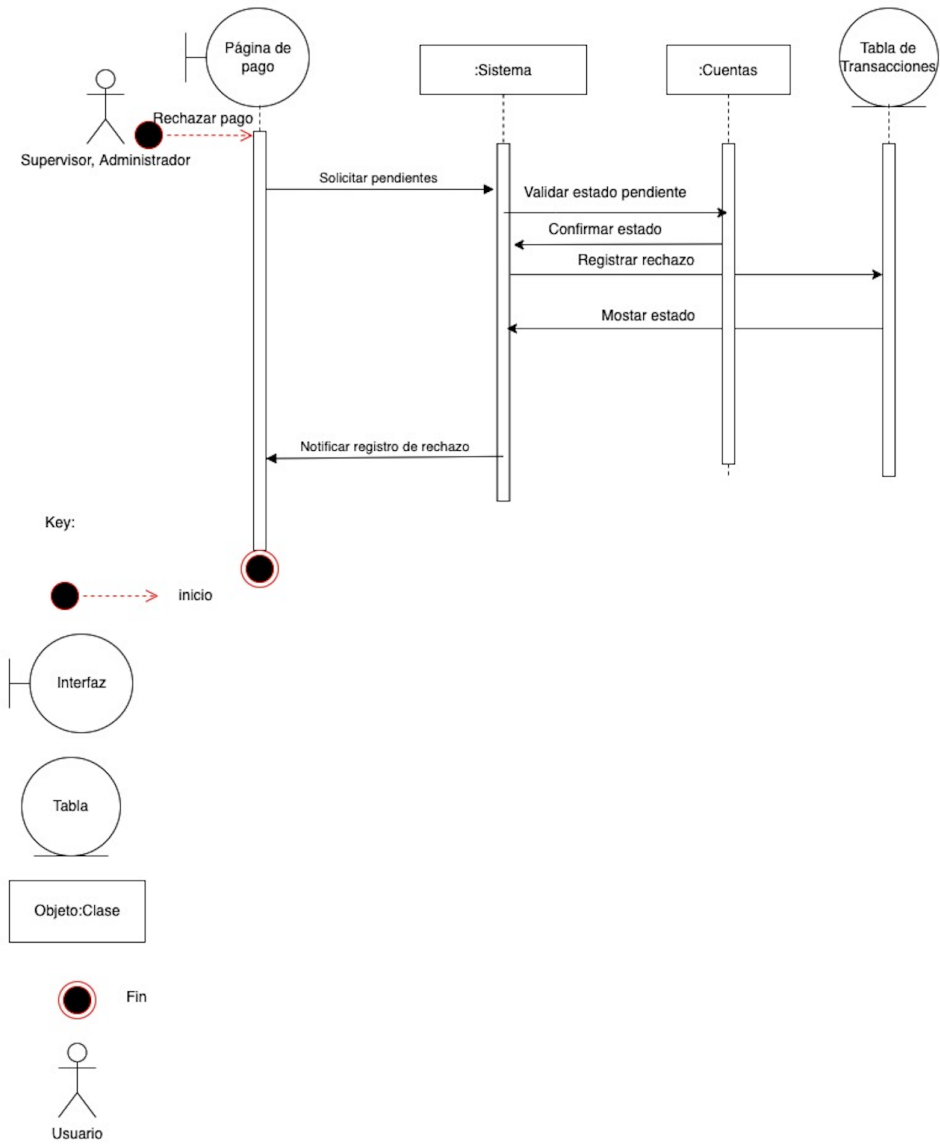
### Crear solicitud de pago



## Aprobar pago

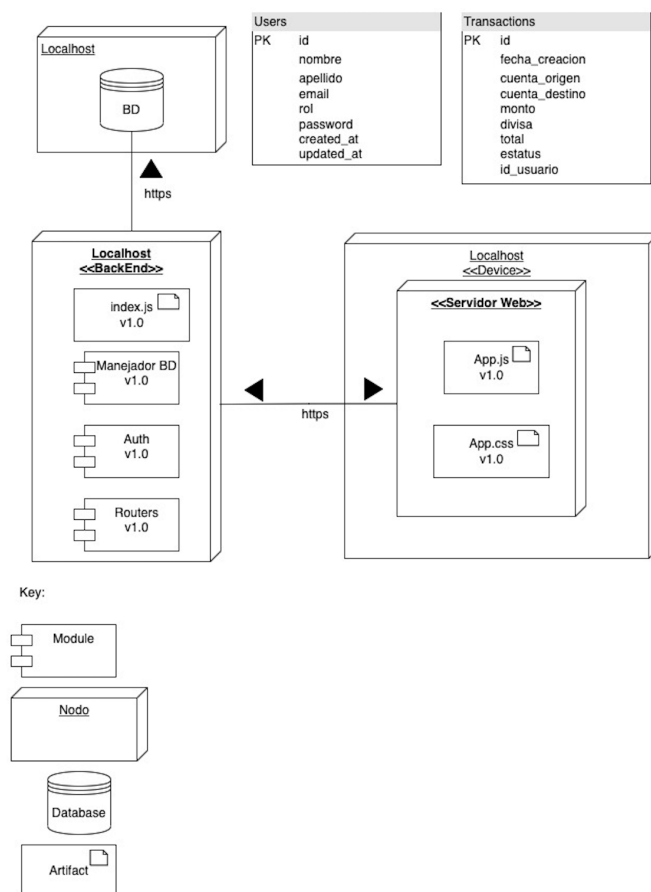


## Rechazar pago



## 2.8 Vista de implementación

Descripción del diagrama de despliegue

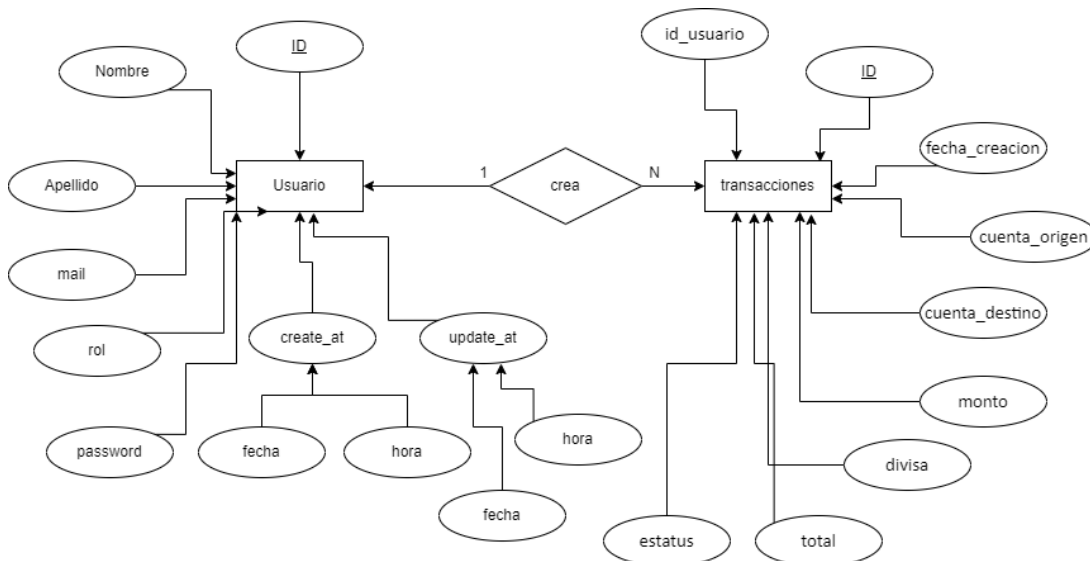


- Existe una vista de implementación del usuario, que son aquellos componentes visuales y físicos que se requieren para poder visualizar el sistema.
- El dispositivo del usuario (equipo de cómputo, tablet) que desee acceder al sistema será el encargado de acceder al servidor web para mostrar los formularios, la página de inicio de sesión o alguna otra funcionalidad que el usuario requiera.

- Dependiendo del uso, el servidor web valida si la petición es por parte de las vistas o de back End y realiza peticiones dependiendo que se solicitó.
- En caso que sea la vista, este traerá los componentes necesarios para construir y crear la vista solicitada.
- En caso que requiera alguna operación más lógica accederá a la implementación del back end y aquí se realiza la operación necesaria y en caso que sea necesario acceder alguna información de la base de datos, este lo realizará de manera automática.

## 2.9 Vista de datos

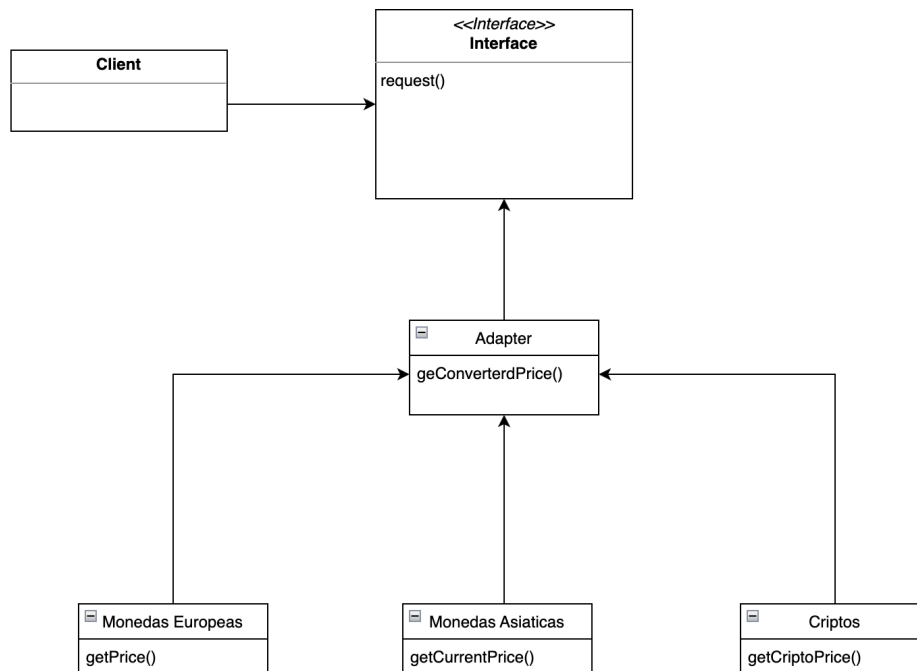
Incluir el diagrama entidad-relación



Key: UML

### 2.9.1 Diagrama de Interfaz API's

Interfaz de conversión de monedas a través del consumo de API's



Key: UML

## 2.9.2 Base de datos

Incluir la información de las tablas

| Nombre de columna | Tipo de dato | Permite Nulo |
|-------------------|--------------|--------------|
| id                | int unsigned | x            |
| Nombre            | varchar(255) | x            |
| Apellido          | varchar(255) | x            |
| email             | varchar(255) | x            |
| rol               | varchar(255) | x            |

|            |              |   |
|------------|--------------|---|
| password   | varchar(512) | x |
| create_at  | TIMESTAMP    | x |
| updated_at | TIMESTAMP    | x |

| Nombre de columna | Significado  |
|-------------------|--|
| id                | Identificador único de cada usuario  |
| Nombre            | Nombre registrado de cada individuo  |
| Apellido          | Apellido registrado de cada individuo  |
| email             | correo electrónico como un identificador de comunicación de cada individuo y de inicio de sesión   |
| rol               | Su categoría dentro de la organización, este se puede clasificar en tres tipos determinado como un usuario staff, un supervisor y un administrador |
| password          | Contraseña encriptada como autenticación de usuario  |
| create_at         | Fecha donde el usuario fue creado  |
| updated_at        | Fecha donde el usuario fue actualizado de alguno de sus campos anteriores.   |

| Nombre de columna | Tipo de dato | Permite Nulo |
|-------------------|--------------|--------------|
| id                | unsigned int | x            |
| fecha_creacion    | TIMESTAMP    | x            |

|                |              |   |
|----------------|--------------|---|
| cuenta_origen  | unsigned int | x |
| cuenta_destino | unsigned int | x |
| monto          | double       | x |
| divisa         | double       |   |
| total          | double       | x |
| estatus        | varchar(255) | x |
| id_usuario     | unsigned int | x |

| Nombre de columna | Significado  |
|-------------------|--|
| id                | identificador único de transacción                           |
| fecha_creacion    | Fecha y hora que se realiza la transferencia                 |
| cuenta_origen     | cuenta bancaria del origen de la transacción                 |
| cuenta_destino    | cuenta bancaria del beneficiario que recibirá la transacción |
| monto             | Cantidad monetaria de la transacción                         |
| divisa            | su cambio de moneda (en caso que sea necesario)              |
| total             | el total de la transferencia contando divisas                |
| estatus           | el estado que se encuentra la transacción                    |
| id_usuario        | El usuario que realizó dicha transacción                     |



### 3. Lineamientos Arquitectónicos

*Describe los lineamientos que aplicará así como los objetivos de los mismos*

#### 3.1 Codificación

(principios solid, documentar decisiones arquitectónicas, tamaño de métodos. Organización de paquetes del diseño por funcionalidad, por deployment )

Acorde a lo estipulado por Cervantes, H., Velasco, P. y Castro, L. en Arquitectura de Software, Conceptos y ciclo de desarrollo en 2016, la utilización de estándares de codificación fomenta la institucionalización de las buenas prácticas y recomendaciones en el planeamiento del diseño, desencadenando así en el alcance de mayores niveles de calidad en el desarrollo de productos de software.

Por otra parte, siguiendo uno de los objetivos arquitectónicos en cuanto a atributos de calidad de mantenimiento, el seguimiento de los estándares antes mencionados llevaría al producto a un estado de mantenimiento óptimo acorde al estilo de codificación estandarizado, permitiendo que los posibles cambios futuros al producto puedan realizarse sin la alteración de los módulos base, además de optimizar costos en el proceso de mantenimiento del sistema.

Enfocándonos en otros objetivos arquitectónicos en cuanto a atributos de calidad de rendimiento y *testability*, al establecer una estructuración e incluso estandarización del código que se utilizará en el producto, se podrán alcanzar menores índices en errores de codificación, permitiendo evitar fallas, bugs, defectos, entre otros posibles escenarios en que el programa se comporte de manera no deseada. Dicho escenario desencadenaría una fase de pruebas óptima, en la cuál, debido al diseño arquitectónico elegido (Microservicios), la ejecución de pruebas por módulo implicaría un ahorro en tiempo y esfuerzo. Así mismo, debido a el ahorro de ejecución de procesos innecesarios y utilizando algoritmos de búsqueda optimizados, como es el caso de búsqueda indexada por medio de HashSets, se preveé que el rendimiento del programa tenga resultados satisfactorios.

#### Principios SOLID

Para cumplir los objetivos establecidos para el código en la sección anterior, será necesario implementar los principios SOLID a lo largo del desarrollo del producto final. Para ejemplificar y establecer la guía que deberá seguir el equipo de desarrollo con los principios antes mencionados, se optará por plasmar en esta sección del documento, dónde, cómo y el por qué se deberá usar cada uno de los principios.

##### *Single Responsibility Principle*

Acorde a lo estipulado por García, J. en *SOLID y GRASP: Buenas prácticas hacia el éxito en el desarrollo de software*, 2012, este principio nos habla de que cada componente del programa, deberá concentrarse en sólo desempeñar una función única, buscando que, en caso de requerir alguna modificación de funcionalidad, las modificaciones a cada componente no alteren el funcionamiento entero del sistema o de componentes ajenos.

Considerando que nuestro producto contempla el manejo de clases y componentes únicos de *React.js*, se plantea el manejo de utilización de principio, así como de estandarización de la siguiente manera:

- Cada clase que sea requerida para el funcionamiento del programa deberá ser pensada con la finalidad de cumplir un requerimiento específico.
- Cada uno de los métodos dentro de las clases deberá realizar una tarea específica dentro de los requerimientos de la clase.
- En caso que múltiples funciones requieran de operaciones, algoritmos, interacciones con el usuario o cualquier otro caso de acción, se deberá aislar en una función a parte para su reutilización por parte de las funciones externas, sólo si es parte del mismo requerimiento de la clase. En caso de que el requerimiento salga del establecido por la clase, deberá plantearse la creación de una clase nueva.
- Todo componente utilizado dentro de la interfaz de usuario deberá seguir el modelo atómico, conformando grandes organismos/componentes, por otros más pequeños, siguiendo la siguiente estructura: átomos, moléculas y organismos. Cada uno de ellos deberá ser separado o conformado acorde a la función desempeñada con la interacción deseada por parte del usuario.
- Al momento de ejecutar pruebas a cada uno de los módulos, estas deberán ser de tipo funcional, probando cada módulo por separado y garantizando en cada modificación a versión del programa que la funcionalidad del mismo no se vea afectada aún cuando un componente presente cambios.

### ***Open / closed principle***

Acorde a lo estipulado por García, J. en *SOLID y GRASP: Buenas prácticas hacia el éxito en el desarrollo de software*, 2012, este principio nos habla de cómo cada componente deberá permanecer disponible para extensión en cuanto a funcionalidad empleando conceptos de *POO* como herencia, polimorfismo y composición. Sin embargo, los mismos deberán quedar cerrados o no disponibles a modificaciones de funcionalidad.

Considerando que nuestro producto contempla el manejo de clases y componentes únicos de *React.js*, se plantea el manejo de utilización de principio, así como de estandarización de la siguiente manera:

- Cada clase deberá ser separada y/o catalogada acorde al requerimiento para el cuál fue diseñada. La representación de dicha norma se hará presente mediante la utilización de interfaces o clases abstractas, de las cuales se crearán clases dependientes que puedan heredar y/o sobrescribir cada una de las características y/o funcionalidades de los padres.
- En caso de requerir un componente extra en el programa, se podrán crear las clases pertinentes siempre y cuando estas estén asociadas a la clase abstracta o interfaz padre correspondiente.
- Siempre que se requiera una funcionalidad nueva, se deberá añadir a la clase abstracta o interfaz padre, validando en todo momento que se encuentre dentro del segmento de requerimiento correspondiente. En caso de que la funcionalidad salga de dichos elementos padres, se deberá contemplar la creación de nuevos elementos para el requerimiento solicitado.
- Todo organismo dentro de la interfaz de usuario deberá asociarse como una clase hija, que importará (heredará) a las moléculas, mismas que importarán (heredarán) a los átomos. En caso de que alguna extensión visual sea requerida, dichos componentes podrán importar componentes con menor jerarquía. Para comprender mejor la implementación de este principio en *React.js* se puede pensar en los organismos como las clases hijas y en las moléculas y/o átomos (según sea el caso del diseño) como en las clases abstractas o interfaces padres.
- El único caso en que sea permitida la modificación de algún componente será en caso de que las normas bancarias o normas legislativas de la nación en que Smart Pay Co. pueda ejercer sus servicios sea modificada, obligando a la empresa a realizar modificaciones en componentes que comprometan información de los usuarios y/o clientes, o alguna otra interacción con usuarios que pueda poner en riesgo la integridad de la empresa desarrolladora y de la contratante (Smart Pay Co).

### ***Liskov Substitution Principle***

Acorde a lo estipulado por García, J. en *SOLID y GRASP: Buenas prácticas hacia el éxito en el desarrollo de software*, 2012, este principio nos habla de cómo cada componente que se desempeñe como subclase deberá poder comportarse adecuadamente cuando sean requeridas en lugar de sus clases padre/base. Así mismo, se considera que el empleo de este principio podría violar el principio de *Open / Closed*, según sea el caso. Por ello, para implementación en el sistema a desarrollar, se considerará este principio como una medida de contingencia para imprevistos, mas no se buscará sustituir una clase por la otra si este comportamiento no es requerido, evitando así incurrir en amenazas de fallo en la funcionalidad del programa.

Considerando que nuestro producto contempla el manejo de clases y componentes únicos de *React.js*, se plantea el manejo de utilización de principio, así como de estandarización de la siguiente manera:

- Cada clase que actúe como subclase o extensión de otra clase/clase abstracta deberá ser diseñada para poder ser sustituida por la clase base, cuidando en todo momento que la funcionalidad de la clase base se siga cumpliendo.
- Bajo un esquema de funcionamiento normal, no se pretende el uso de una subclase por otra que sea base, sin embargo, si la necesidad esporádica surge, cualquier componente que funcione como subconjunto deberá estar listo para su reemplazo.
- Todo componente de la interfaz gráfica quedará exento de este principio, pues cada uno de los mismos contará con un diseño visual diferente, pensar en la sustitución de subcomponentes visuales por otros tendría repercusiones en cuanto al diseño de interfaz.

### ***Interface Segregation Principle***

Acorde a lo estipulado por García, J. en *SOLID y GRASP: Buenas prácticas hacia el éxito en el desarrollo de software*, 2012, este principio nos habla de cómo cada interfaz aplicada a cada componente deberán ser implementadas adecuadamente, cuidando que aquellos clientes que resulten dependientes de las mismas requieran en su totalidad de la funcionalidad de la misma.

Considerando que nuestro producto contempla el manejo de clases y componentes únicos de *React.js*, se plantea el manejo de utilización de principio, así como de estandarización de la siguiente manera:

- Cada interfaz deberá cumplir con el planteamiento de un requerimiento específico, cuidando que los sub requerimientos que puedan presentarse se hagan presentes de toda aquella clase que sea cliente de la misma, implementando la funcionalidad del requerimiento base.
- Si algún sub requerimiento por parte de la clase cliente se presenta fuera del primer requerimiento base, se deberá agregar una interfaz que subsecuentemente cumpla con el requerimiento base adicional, teniendo una clase cliente de ambas interfaces.
- Una clase podrá tener cuantas interfaces sean necesarias, siempre y cuando su implementación no incurra en una falla de los puntos anteriormente establecidos.
- Todo componente de interfaz gráfica deberá consumir únicamente los componentes con menor jerarquía, con base al diseño atómico, que requiera para su conformación final. En caso de que se consuma algún componente de menor jerarquía obstruyendo alguna característica por no ser requerida, se deberá plantear la creación de un nuevo componente de dicha jerarquía con las características que requiera el organismo que busque implementarlo.
- Todo componente de interfaz gráfica podrá consumir la cantidad de componentes con menor jerarquía necesarios, siempre y cuando su uso sea total.

### ***Dependency Inversion Principle***

Acorde a lo estipulado por García, J. en *SOLID y GRASP: Buenas prácticas hacia el éxito en el desarrollo de software*, 2012, este principio nos habla de cómo los módulos de alto nivel no deberán depender de módulos de nivel bajo. En otras palabras, con tal de garantizar flexibilidad, bases estables, así como la posibilidad de reutilización en el código, se deberá vincular la dependencia del código hacia abstracciones, no de concreciones.

Considerando que nuestro producto contempla el manejo de clases y componentes únicos de *React.js*, se plantea el manejo de utilización de principio, así como de estandarización de la siguiente manera:

- Toda clase planteada en el código deberá ser cliente de alguna interfaz o en su defecto, heredar de una clase abstracta, todo bajo el esquema de tener a dicha clase como un sub requerimiento y a la entidad padre como la funcionalidad de un requerimiento base.
- Cada interfaz de la que dependa una clase deberá englobar únicamente la funcionalidad de la misma, permitiendo a cada clase tener una personalización única de las características de cada objeto.
- En caso de que una clase se vea en la necesidad de heredar atributos de otra clase no abstracta, se deberá plantear la creación de una entidad abstracta que englobe la funcionalidad requerida por parte de la clase base y posteriormente ligar ambas clases como clientes a la entidad abstracta, respetando así en todo momento el principio estipulado.
- Toda conexión con servicios exteriores de terceros fuera del código como lo pueden ser: bases de datos, API 's, conectividad con servidor, etc. deberán ser aisladas en componentes individuales para realizar las conexiones mediante la implementación de métodos en los componentes necesarios sin tener que establecer nuevamente la configuración con cada petición a los mismos.
- Todo componente de interfaz gráfica deberá cumplir con el diseño atómico, es decir que ningún organismo visual deberá ser codificado de manera única, sino que deberá consistir de cada uno de los componentes de menor jerarquía para cumplir con este principio.

### **Normas de estructuración durante la programación del programa**

Con la finalidad de mantener una línea de seguimiento durante todo el desarrollo del sistema, se estipula como norma el seguimiento e integración de las prácticas de los principios SOLID (acordados en la sección anterior) en conjunto con los siguientes lineamientos para la programación del producto final.

### ***Estructuración de paquetes/módulos***

Dentro del lenguaje de programación JavaScript, la existencia de paquetes o *packages* dentro del código es nula, pues esta propiedad de lenguaje es propia de otros, como lo es Java. Sin embargo, buscando tener una organización en cuanto a los archivos de programación, se fomenta el uso de ficheros (folders) dentro de los archivos de programación.

De este modo, cada fichero deberá corresponder a cada tarea, anteriormente estructurado en el análisis de requerimientos, en donde cada uno de estos representa una gran cantidad de funcionalidad enfocada a una meta del programa mismo. De este modo, dentro de cada fichero, se encontrarán interfaces o clases abstractas, que abarcarán la funcionalidad de cada uno de los casos de uso correspondientes. Finalmente, las clases que busquen implementar o heredar de las entidades abstractas serán los sub requerimientos que se establecen a partir de los requerimientos base.

Al seguir esta estructura, cada uno de los requerimientos definidos con el cliente serán cubiertos en su totalidad, permitiendo su fácil localización dentro de la estructura del programa y cumpliendo en todo momento con la funcionalidad del sistema solicitada en la toma de requerimientos.

### ***Normas de calidad***

Con la finalidad de alcanzar los más altos estándares de calidad en la entrega del sistema, así como garantizar el entendimiento total del funcionamiento del código dentro del programa, se plantean las siguientes normas como lineamientos de calidad al codificar.

- Ningún tipo de error de compilación o *warning* será aceptable para la entrega de módulos del sistema.
- Se deberá seguir en todo momento el estilo de nombramiento *Cammel case* para variables, archivos, ficheros y/o cualquier otro tipo de componentes dentro del sistema.
- Todo archivo y fichero deberán comenzar con mayúscula en el nombre.
- Todo archivo, función (considerando parámetros, return y excepciones), así como consulta de servicios externos al sistema deberán estar documentados internamente siendo concretos con la funcionalidad de cada componente.
- El idioma que se usará para documentación interna del código y nombramiento de componentes dentro del mismo será el inglés.
- El límite de líneas de código dentro de un método será de 20 (sin considerar la declaración del mismo, saltos de línea o la separación por llaves).
- El límite de líneas de código dentro de una clase será de 500 (sin considerar saltos de línea).
- El uso de recursión estará fuera del margen de programación dentro del sistema.

- En el peor de los casos al aplicar un algoritmo, la complejidad algorítmica en cuanto a pasos requeridos para lograr un resultado deberá de ser lineal,  $O(n)$ .
- Ningún módulo deberá ser subido a ambiente de producción sin antes haber pasado las pruebas identificadas en el ambiente de QA.

Debido a que se está utilizando React, es importante definir las mejores prácticas para garantizar un desarrollo de calidad y facilitar el cumplimiento de los objetivos arquitectónicos.

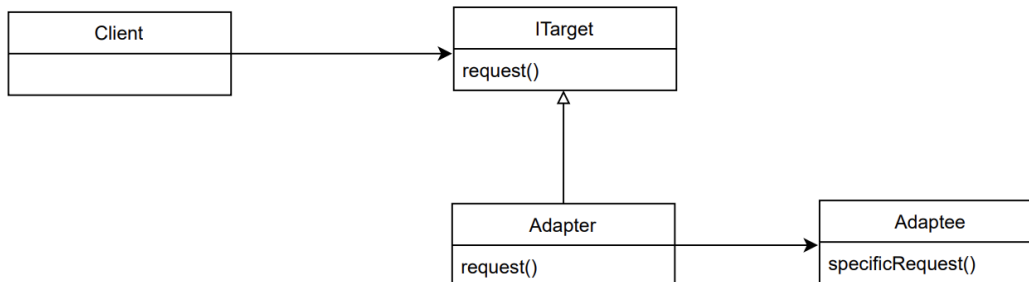
En React, existe un objeto llamado estado, que guarda los valores que pertenecen a un componente. Cada que cambia este objeto, se hace un render del componente en la interfaz de usuario. Teniendo esto en cuenta, es importante cuidar que no sea excesiva la cantidad de estados que se utilizan en el desarrollo para prevenir que la aplicación consuma demasiados recursos en los dispositivos de los usuarios.

Adicionalmente, es importante mencionar que aunque el desarrollo se haga en JavaScript, deberán definirse los tipos de propiedades que se pasarán a cada componente, de esta manera se define una estructura clara que nos permite conocer las dependencias de los componentes y por ende cuidar fácilmente el cumplimiento de los principios SOLID.

## 3.2 Diseño

### *Adapter*

Este patrón convierte la interfaz de una clase en otra interfaz que un cliente espera. Este patrón permite que las clases funcionen en conjunto cuando sus interfaces son incompatibles. Visualmente se ve de la siguiente manera.



Key: UML

En este caso, el cliente desea llamar a *specificRequest* pero usando la firma *request*. Es por eso que se utiliza un adaptador entre ellas. El adaptador en este caso es la implementación concreta de ITarget.

Este patrón se utiliza para alcanzar cierto nivel de interoperabilidad. Es decir, cuando deseas cierto comportamiento, pero no puedes tenerlo utilizando la interfaz que tienes. Un ejemplo de su uso puede ser que cierta librería va a cambiar el orden de los parámetros que recibe una función. Si no deseamos cambiar cada implementación, podemos recurrir a este patrón para crear un adaptador y utilizarlo.

Su implementación en código sería de la siguiente manera.

Cliente

```
ITarget target = new Adapter(new Adaptee());
target.request();
```

Interface

```
interface ITarget {
    void request();
}
```

Adapter



```
class Adapter: ITarget {  
    Adaptee adaptee;  
  
    public Adapter(Adaptee a) {  
        this.adaptee = a;  
    }  
  
    public void request(){  
        this.adaptee.specificRequest()  
    }  
}
```

### Adaptee

```
class Adaptee {  
    public void specificRequest(){  
    }  
}
```

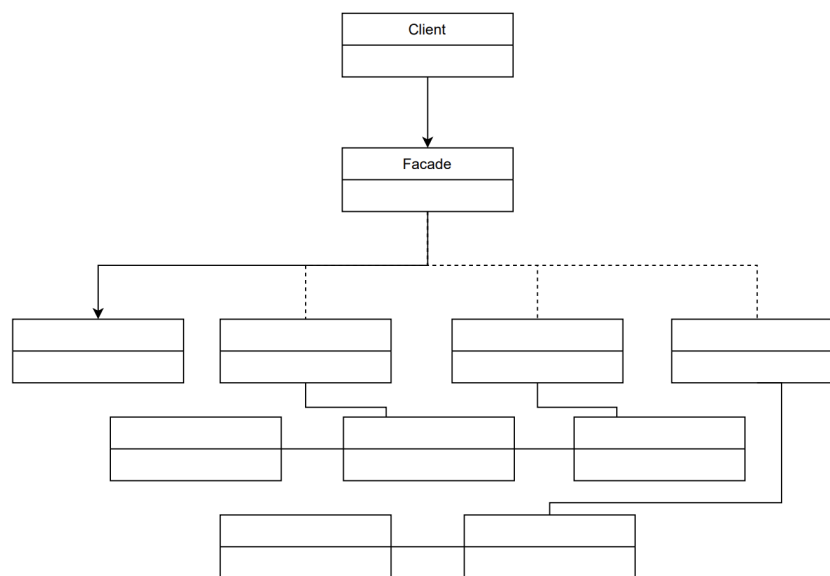
La razón por la que se desea implementar este patrón en el desarrollo del proyecto es que utilizaremos interfaces para la funcionalidad principal del mismo, pero también haremos uso de librerías, APIS y componentes externos que nos ayudarán a complementarlo. Específicamente, se requiere el uso de conversión de divisas a la moneda local. Cabe mencionar que utilizaremos divisas americanas, europeas, asiáticas e incluso criptomonedas, lo que implica que requerimos acceso a APIS para cada una de las categorías mencionadas previamente.

Debido a que cada una de estas proviene de desarrolladores y fuentes de información distintas, los métodos y en general el uso es distinto. Por lo que se desea ocultar toda la complejidad de la consulta del precio en tiempo real. Es decir, se utilizará una interfaz que contiene un método llamado consultarPrecio() genérico, que a su vez estará conectado a un adaptador que desarrollaremos que llamará a los métodos específicos que correspondan dentro de cada API externa que utilicemos en el desarrollo del proyecto.

### Facade Pattern

Este patrón provee una interfaz unificada a un conjunto de interfaces de un subsistema. En otras palabras, provee una interfaz de más alto nivel que hace que el subsistema sea más fácil de usar.

Su implementación consiste en que existe un cliente que desea realizar una función significativa, pero para poder hacerlo, requiere de la interacción compleja de diversos objetos. De forma gráfica se ve como en la siguiente figura.



Key: UML

Existen razones significativas por las cuales la implementación de este patrón es de utilidad en el desarrollo del proyecto. Es importante subrayar que el proyecto está apegado a los principios SOLID, específicamente al principio de responsabilidad única. Esto implica que contaremos con diversos módulos que se encargan de realizar una función específica. Por lo que, para realizar una transacción, se requiere utilizar la interacción entre cada uno de los módulos desarrollados.

En particular, para realizar una transacción, de forma general se requiere realizar la conversión de la divisa, la validación de fondos suficientes, la aprobación del pago y finalmente el envío del correo de confirmación. Si consideramos que cada funcionalidad es llevada a cabo por un módulo independiente, el proceso de realizar la transacción implica la interacción. Con el objetivo de facilitar

este proceso, se implementará este patrón con una fachada que se encarga de manejar estas interacciones complejas.

### 3.3 Reutilización de Software

Para este desarrollo se utilizará el framework de React. Esto nos permitirá programar componentes que sean reutilizables y cumplir con el objetivo de interoperabilidad en distintos navegadores. Adicionalmente su diseño orientado a la actualización de datos en tiempo real nos facilitará la conversión de divisas y la actualización de esta información en la interfaz de usuario.

Adicionalmente, deberán crearse distintos componentes para cumplir con la vista lógica de la aplicación.

- Router: Este será encargado de registrar las rutas disponibles en la aplicación y será llamado desde los otros componentes del front-end. De esta manera si se quieren agregar rutas solo se modifica este componente.
- Wrapper: Este componente servirá como una capa de abstracción que llame a los otros servicios pero que exponga funcionalidad de una manera clara a todo el front end. De esta manera, si se agrega un nuevo servicio, solo deberá agregarse la interfaz para interactuar con él en este componente y con esto el front end podrá usar el servicio.
- Database: Este componente realizará todas las interacciones con la base de datos y proveerá métodos para creación, lectura, actualización y eliminación de datos. De esta manera solo este componente depende de la implementación de la base de datos y se pueden generalizar las acciones más comunes.
- Auth: Este componente se encargará de toda la autenticación y autorización de los usuarios. De esta manera podremos acceder a él y conocer el estado de la autenticación así como el nivel de permisos que tiene el usuario actual.

De igual manera se utilizarán componentes disponibles de librerías como MaterialUI o AntDesign para minimizar el tiempo de desarrollo al generar las distintas vistas de los usuarios. Con esto ya tenemos componentes listos para text fields, dropdowns, entre otros componentes de diseño que podrían ser sumamente útiles durante el desarrollo. Adicionalmente se utilizarán herramientas como Prettier para llevar el control de la mayoría del estilo descrito en el punto anterior de manera automática.

## 4 Architecture Tradeoff Analysis Method

### 4.1 Drivers Arquitectónicos

#### Ambiente de negocio

- **Cliente y usuarios**
  - No flexible: El cliente tiene una visión y un producto de software requerido específico.
- **Organización del desarrollo**
  - Flexible: Los desarrolladores tienen el control sobre la gestión y estructura del equipo de trabajo, así como del proceso de desarrollo.
- **Ambiente técnico**
  - Flexible: Las herramientas que pueden ser utilizadas para el desarrollo del software están limitadas a librerías de NodeJS.
  - NodeJS provee librerías extensas que ayudan a los desarrolladores a construir aplicaciones.
- **Experiencia de arquitectura**
  - No flexible: El equipo está integrado por desarrolladores arquitectónicamente inexperimentados. Cuentan con una formación tradicional basada en programación orientada a objetos. Sin embargo, son asesorados por la profesora Marlene O. Sánchez, quien cuenta con basta experiencia en el tema.

### 4.2 Restricciones técnicas, administrativas

|   |
|---|
| El hosting del programa no debe rebasar los \$4,000.00 pesos por año.   |
| El lenguaje de programación deberá ser JavaScript y el framework elegido para el desarrollo deberá ser Express.js en conjunto con React.js  |
| El sistema deberá ser desplegable desde cualquiera de los siguientes navegadores: <ul style="list-style-type: none"><li>- Google Chrome (a partir de versión 100.0.4896.127).</li><li>- Microsoft Edge (a partir de</li></ul> |

|  |
|--|
| versión 100.0.1185.44).<br>- Mozilla Firefox (a partir de<br>versión 99.0.1).        |
| El despliegue deberá ser<br>únicamente para ordenadores u<br>ordenadores portátiles. |

|   |
|---|
| El programa deberá entregarse en<br>un plazo no mayor a 4 meses.  |
| Los servicios de desarrollo deberán<br>ser brindados únicamente por el<br>equipo de desarrollo, sin posibilidad<br>de contratación externa de terceros. |

### 4.3 Stakeholders

Compañía Smart Pay Co

Ejecutivo de cuenta

Clientes

Personal de RH

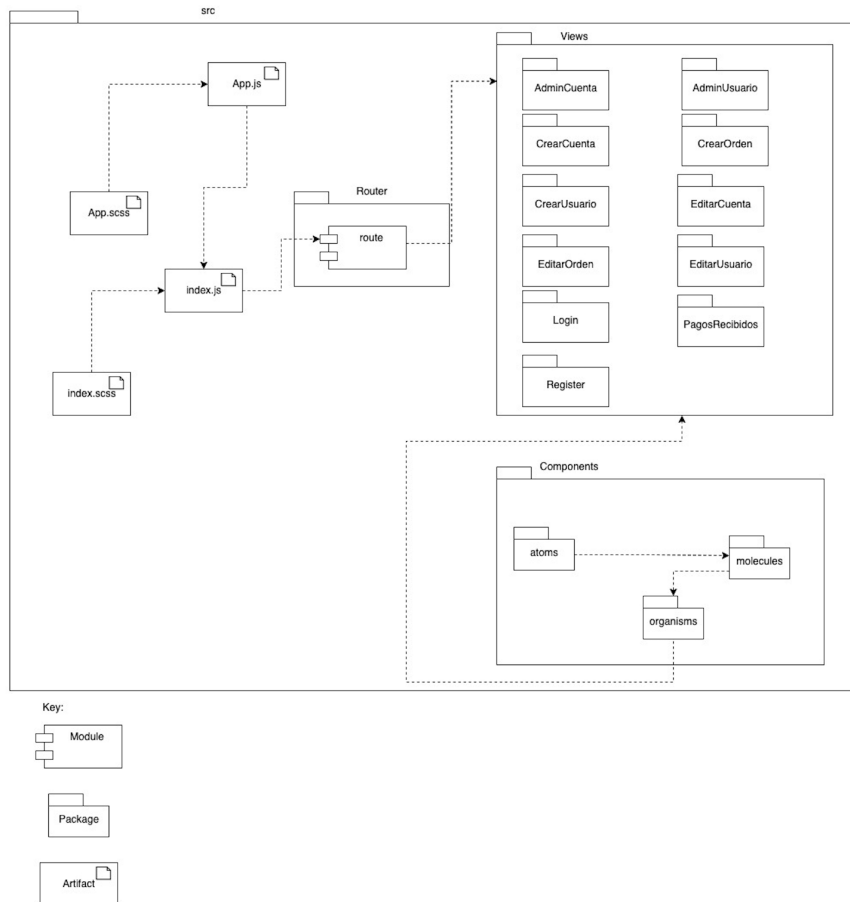
Gerentes y supervisores

Profesora Marlene O. Sánchez Escobar (Arquitectura de Software)

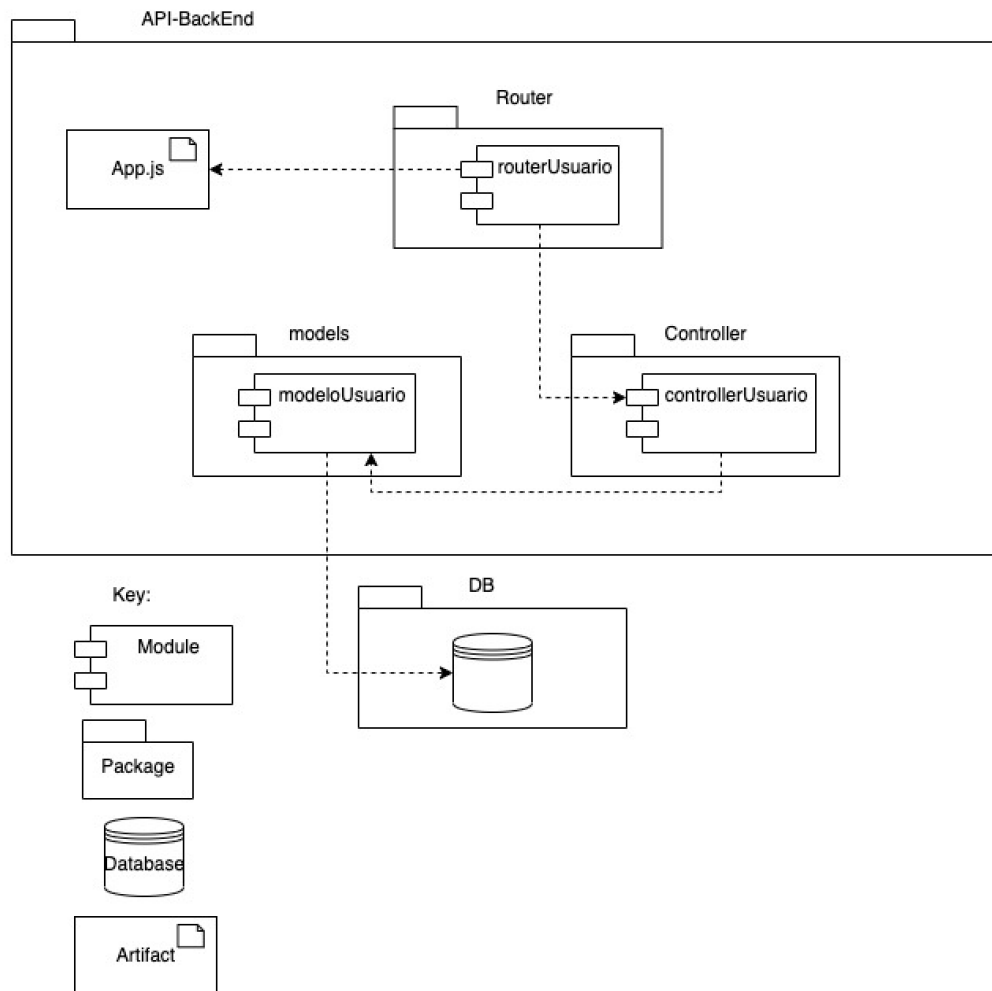
Profesor Luis José González Gómez (Laboratorio de Desarrollo Web)

## 4.4 Vistas Arquitectónicas

### 4.4.1 Vista Lógica Front-End

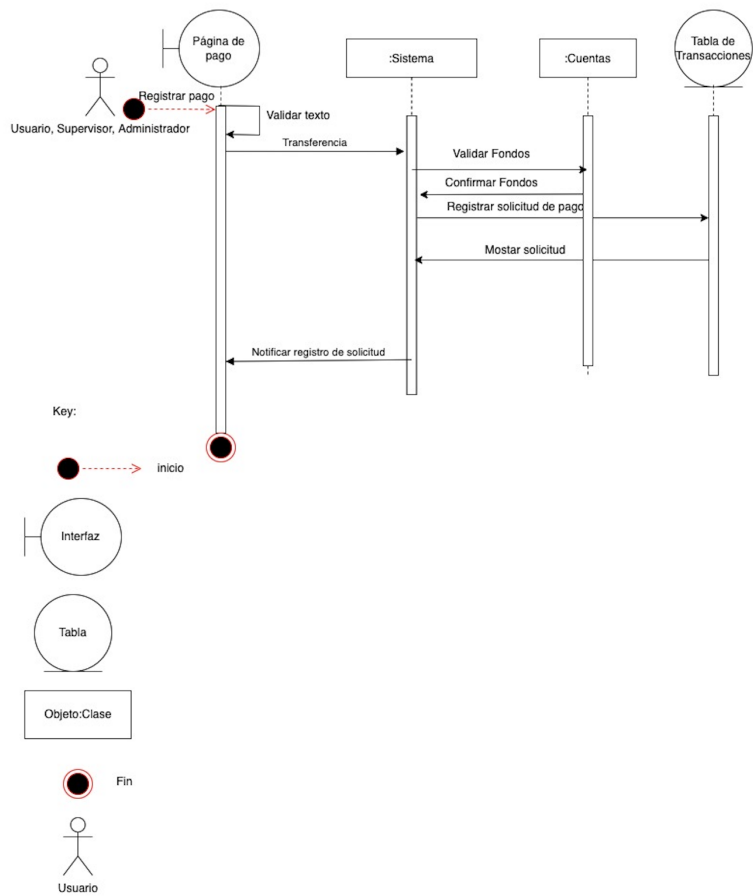


#### 4.4.2 Vista Lógica Backend

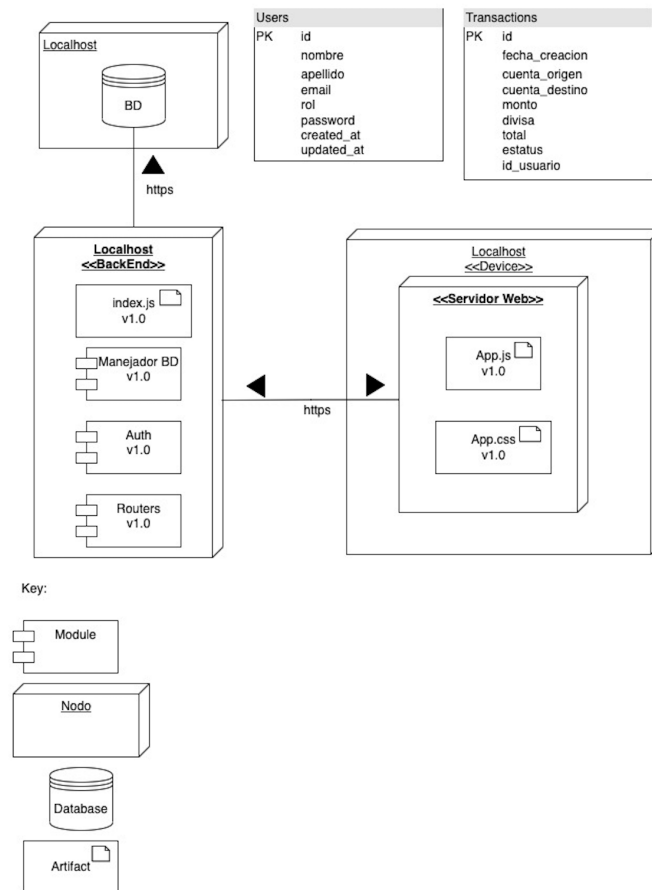




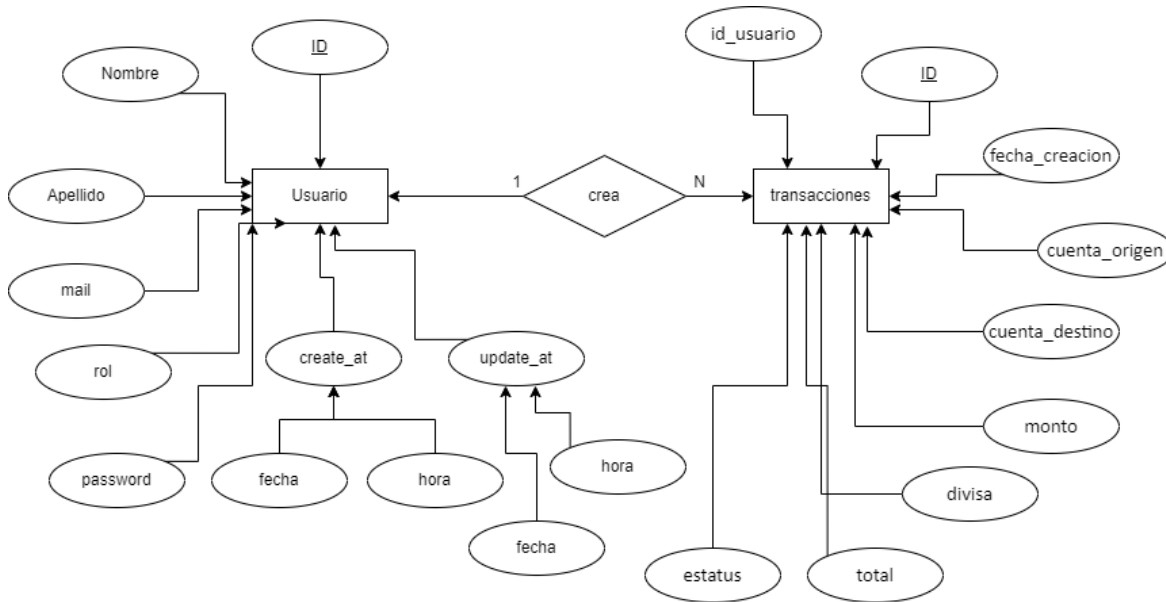
#### 4.4.3 Vista de Proceso



#### 4.4.4 Vista de Implementación



#### 4.4.5 Vista de Datos



Key: UML

#### 4.5 Descripción de Arquitectura Microservicios ATAM

La arquitectura de microservicios está cada vez más presente en la industria como una alternativa viable a las aplicaciones monolíticas.

##### Descripción General de la Arquitectura

El primer concepto asociado con esta arquitectura son los componentes desplegados como unidades separadas. El segundo concepto es el de componentes de servicio, que se refiere a uno o varios módulos que representan la misma funcionalidad y propósito. Bajo este aspecto se presenta la dificultad de escoger el nivel correcto de granularidad. El tercer concepto clave de este patrón es una arquitectura distribuida, es decir, que todos los componentes estén desacoplados.

Comparando este tipo de arquitectura con sus sucesores, como la arquitectura monolítica, cuenta con componentes estrechamente acoplados que forman parte de la misma unidad de despliegue, lo que

provoca que sea difícil de probar, de cambiar y de desplegar. Estos factores provocan aplicaciones frágiles que dejan de funcionar cada vez que un nuevo cambio se despliega.

Debido a lo anterior, se optó por implementar la arquitectura de microservicios debido a que resuelve estos inconvenientes separando la aplicación en múltiples componentes que pueden ser desarrollados, probados y desplegados independientemente. Requerimos que sea modificable debido a que al tratarse de un proyecto desarrollado para una materia, anticipamos cambios significativos y agregación de funcionalidades en cada entrega del parcial, es decir, entregas continuas. Asimismo, las pruebas resultan relevantes para identificar las capacidades del modelo, para dicho propósito se tiene contemplado utilizar Jasmine.

#### **4.6 Tradeoffs Específicos de la Arquitectura**

Agilidad: Alta.

Con este patrón es muy fácil adaptarse al entorno cambiante lo que permite un desarrollo sencillo y rápido.

Facilidad de despliegue: Alta.

Debido a su granularidad y a su independencia en los servicios, el riesgo al momento del despliegue es significativamente menor y en caso de tener un error, es más fácil restaurar la versión.

Facilidad para pruebas: Alta.

Debido a la separación de funcionalidad y aislamiento en módulos independientes, todos los esfuerzos de pruebas pueden ser dirigidos de forma que sean muy específicos. Asimismo, se reduce la posibilidad de tener que hacer pruebas sobre toda la aplicación por un cambio en un solo módulo.

Rendimiento: Bajo.

Debido a la naturaleza de distribución de servicios del patrón, no se recomienda su uso para aplicaciones que requieran un rendimiento muy alto.

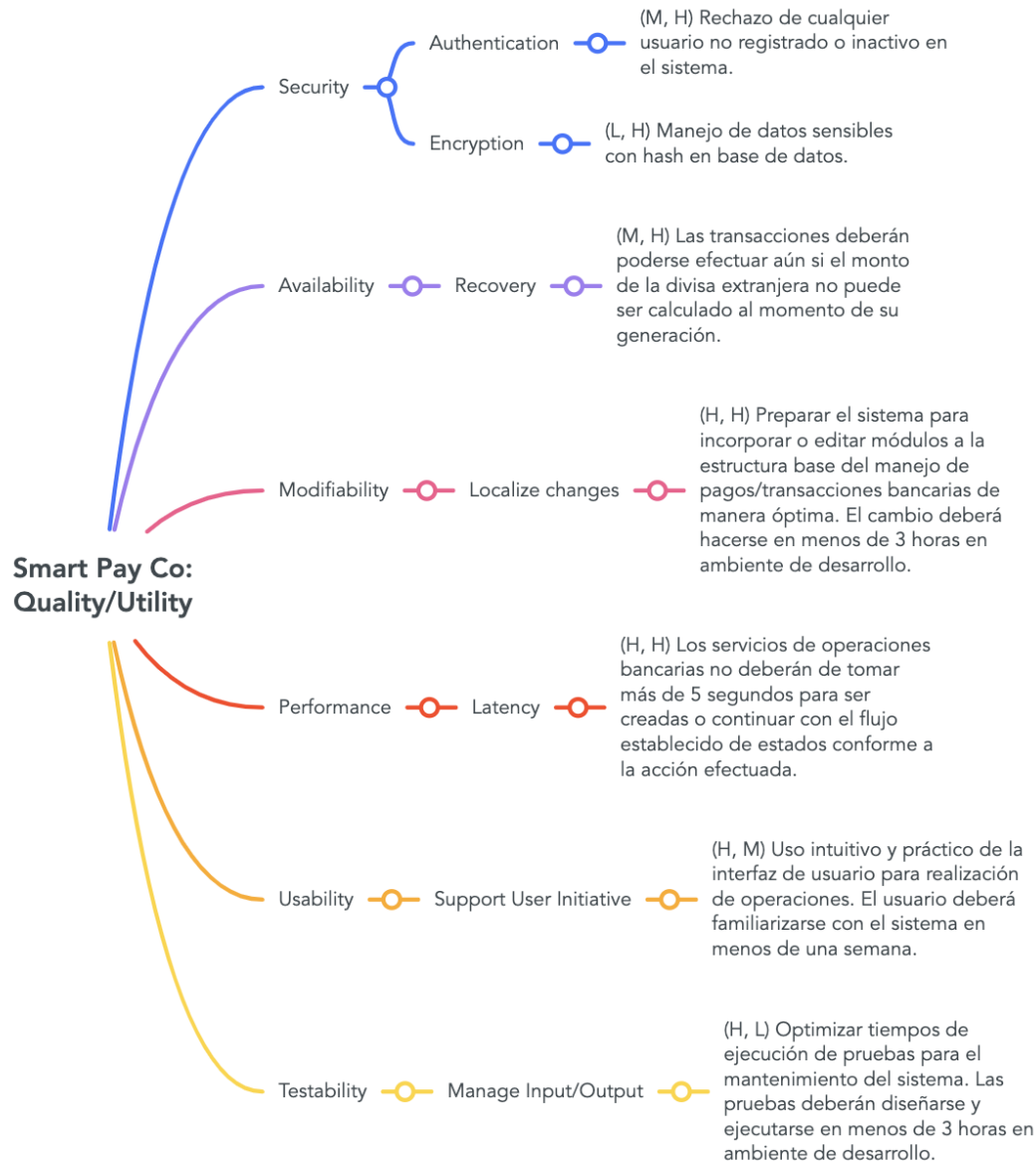
Escalabilidad: Alta.

El hecho de que están separadas las funcionalidades en servicios, implica que se puede escalar aquellos servicios más concurridos o con mayor volumen de usuarios.

Facilidad de desarrollo: Alta.

Cada componente puede ser asignado a un desarrollador o equipo, reduciendo así significativamente las tareas de coordinación.

#### 4.7 Atributos de calidad ATAM - Quality Attribute Tree



## 4.8 Análisis de Atributos y Arquitectura

**Atributo 1:** Security - Authentication - Rechazo de cualquier usuario no registrado o inactivo en el sistema.

### ¿De qué necesidad de negocio se deriva?

Esa necesidad se deriva de permitir el acceso al sistema solamente al personal de Smart Pay Co.

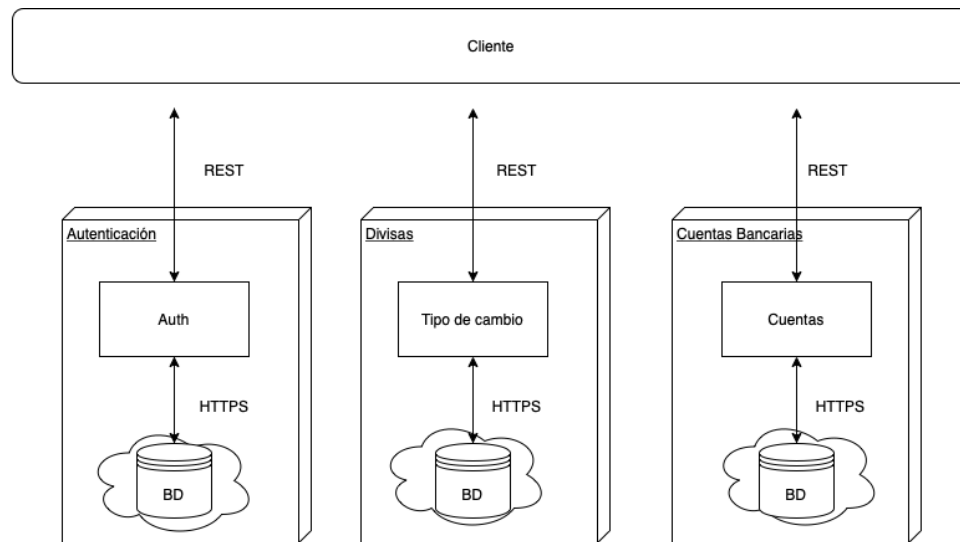
### Preguntas de refinamiento

¿Qué interés tendría un tercero en acceder al sistema?

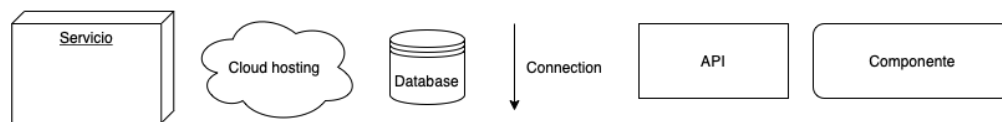
Al tratarse de un sistema de servicios bancarios y transacciones, el interés recae ya sea en robar la información financiera de los usuarios o de despojarlos de sus saldos.

¿Qué pasa con la cuenta si un empleado deja su puesto de trabajo?

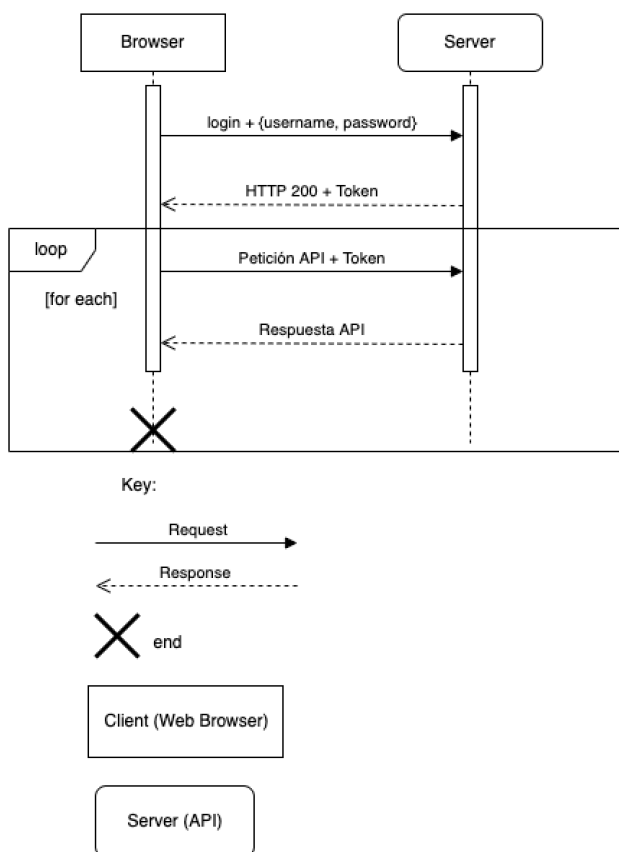
En ese caso, un administrador puede acceder al sistema y cambiar el estado de la cuenta a inactivo. Con lo cual, si dicho usuario intenta acceder, le será negado el acceso.

**Estilo arquitectónico, patrón o mecanismo implementado**

Key:







JSON Web Token o JWT es un estándar abierto que define una manera compacta y autocontenida para transmitir información de forma segura a través de objetos JSON.

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
"name": "John Doe",
"iat": 1516239022
}
```

```
HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    your-256-bit-secret
) ☐ secret base64 encoded
```

## Riesgos

## Puntos de sensibilidad

**Atributo 2 :** Security - Encryption - Manejo de datos sensibles con hash en base de datos.

74

Al tratarse de un sistema bancario, es necesario proteger la información guardada para que en caso de que se vulnere la base de datos, no le sea posible al usuario malintencionado hacer lectura o uso de los datos recabados. Asimismo, los mismos desarrolladores y gestores de bases de datos tampoco podrán leer los datos almacenados.

### Preguntas de refinamiento

¿La información que el sistema almacena es sensible?

Sí debido a que se trata de datos financieros de clientes, por lo que sí se requiere hashearlos.

¿Los desarrolladores y gestores de bases de datos tendrán acceso a los datos?

Los datos almacenados pasarán por un proceso de hashing previo a su almacenamiento de forma que no puedan visualizar la información confidencial.

### Estilo arquitectónico, patrón o mecanismo implementado

Se le conoce como hashing al proceso de asegurar texto plano, usando una función criptográfica de una vía. Esto quiere decir que resulta inviable invertir.

Para el caso particular del proyecto se utilizó una librería llamada bcrypt que contiene una función de hashing que es capaz de proteger contra ataques de tabla arcoiris, de forma que se mantiene protegida contra ataques de fuerza bruta.



### Riesgos

Cifrado SHA-512 fue diseñado para ser un algoritmo rápido, por lo que no debe ser utilizado para cifrar contraseñas. La solución óptima es un algoritmo de cifrado lento ya que estos tienen el propósito, se les conoce también como PBKDF (Password-based key derivation function).

### **Puntos de sensibilidad**

El uso de bcrypt tiene una característica que permite seleccionar el número de iteraciones en las que se le agrega más complejidad. Esto es un punto de sensibilidad debido a que seleccionar muchas iteraciones disminuye significativamente el rendimiento de la aplicación, mientras que seleccionar pocas hace más vulnerable al sistema.

**Atributo 3:** Availability - Recovery and Redundancy - Las transacciones deberán poderse efectuar aún si el monto de la divisa extranjera no puede ser calculado al momento de su generación.

#### **¿De qué necesidad de negocio se deriva?**

Este atributo se deriva de la necesidad de utilizar divisas extranjeras en las transacciones. Se utilizarán APIs externas para obtener el tipo de cambio en tiempo real.

### **Preguntas de refinamiento**

¿Qué tipo de fallas podrían interrumpir la operación del sistema?

Las principales fallas que podrían presentarse durante la operación del sistema son la caída de los servicios externos de obtención del tipo de cambio.

¿Estas fallas podrían generar otras fallas?

En caso de no desarrollar acciones contra las fallas de las APIs externas (cachar el error y actuar en consecuencia), la falla de conversión podría tener un efecto dominó donde el sistema de transacciones dejaría de funcionar correctamente.

¿Qué tan frecuente se puede predecir que la falla ocurra?

Se debe poner en operación el sistema y guardar estadísticas propias sobre las fallas presentadas. Esto debido a que si bien estas APIs ya llevan funcionando mucho tiempo, no han puesto como información pública su historial de fallas.

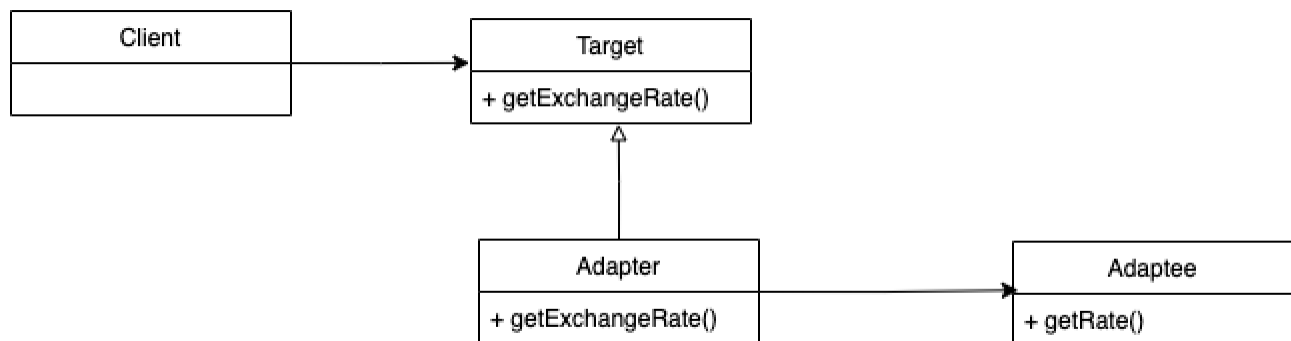
¿Cuánto tiempo toma la detección de la falla?

Debido a que se realiza una solicitud para obtener el tipo de cambio, la respuesta a esta solicitud contiene ya sea el valor solicitado o un error. Por lo que se puede detectar la falla inmediatamente en cuanto se detecte el error.

### Estilo arquitectónico, patrón o mecanismo implementado

Para prevenir este inconveniente se utilizará la redundancia. Es decir, la implementación de múltiples servicios de obtención del tipo de cambio en tiempo real. De forma que si uno de ellos no se encuentra disponible, se puede utilizar el otro, como se explica a continuación.

Para esto se utiliza el patrón adapter que visualmente se ve de la siguiente manera.



Key: UML

Debido a que cada una de las APIs proviene de desarrolladores y fuentes de información distintas, los métodos y en general el uso es distinto. Por lo que se desea ocultar toda la complejidad de la consulta del precio en tiempo real. Es decir, se utilizará una interfaz que contiene un método genérico para consultar el precio, que a su vez estará conectado a un adaptador que desarrollaremos que llamará a los métodos específicos que correspondan dentro de cada API externa que utilicemos en el desarrollo del proyecto.

El uso de este patrón permitirá sustituir fácilmente la API que utilizamos, ya que los métodos de la misma serán llamados a través de su genérico en el adaptador.

### Riesgos

Cabe mencionar que el versionamiento de las APIs externas presenta un riesgo debido a que en caso de una actualización que modifique la funcionalidad, el nombre de los métodos, el orden de los parámetros, entre otros. Nuestro sistema podría dejar de funcionar.

## Puntos de sensibilidad

La expansión de las empresas de tecnología, representa un punto de sensibilidad para este atributo en particular. Esto se debe a que es muy común que una empresa haga una adquisición de otra, fusionando sus líneas de producto y unificándolas como parte de una solución. Por lo que nos deja sin opciones alternativas sin correlación.

## Escenario de crecimiento

La aplicación actualmente soporta entre 100 y 150 usuarios, debido a la infraestructura que estamos usando. La aplicación esta desplegada con Heroku, sin embargo en el plan que estamos usando no cuenta con una disponibilidad tan alta pues no se paga ningún servicio. Considerando un plan de crecimiento para soportar 500 usuarios, deberíamos pagar entre \$25 y \$50 dólares al mes para obtener una disponibilidad de 99.95%, así como poder procesar más rápido los datos pues incrementaríamos de 512MB de RAM a 1GB y mejoraríamos el poder de procesamiento de los servidores. Adicionalmente, este nivel provee crecimiento horizontal automático. Por lo que en caso de que tengamos aún más usuarios en el futuro, de igual manera, deberíamos poder soportarlos gracias a este producto.

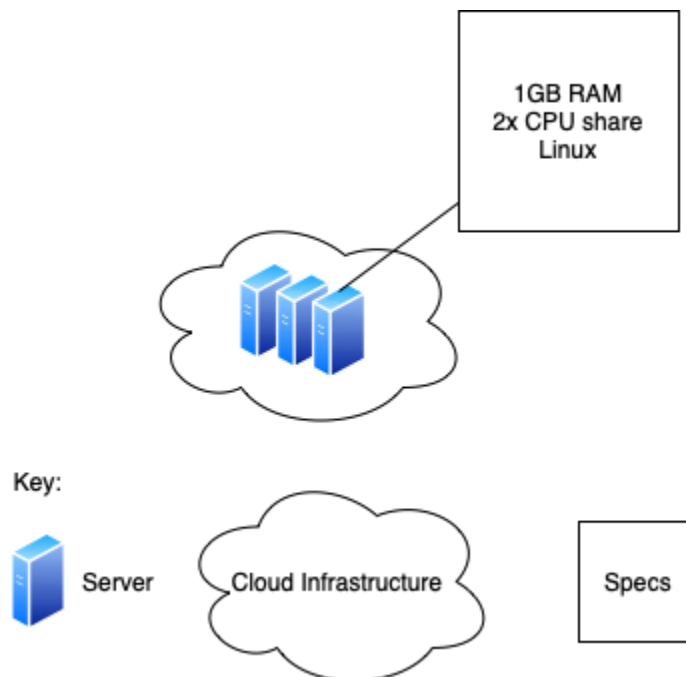


Diagrama de crecimiento de la arquitectura

**Atributo 4:** Modifiability - Localize changes - Preparar el sistema para incorporar o editar módulos a la estructura base del manejo de pagos/transacciones bancarias de manera óptima. El cambio deberá hacerse en menos de 3 horas en ambiente de desarrollo.

#### **¿De qué necesidad de negocio se deriva?**

Esto se deriva de la necesidad de un desarrollo incremental. Se desarrolló en conjunto con la materia de Laboratorio de Desarrollo Web, por lo que en cada parcial se entrega un avance con más funcionalidades. Es decir, más módulos o los mismos módulos modificados para cumplir con la retroalimentación proporcionada.

#### **Preguntas de refinamiento**

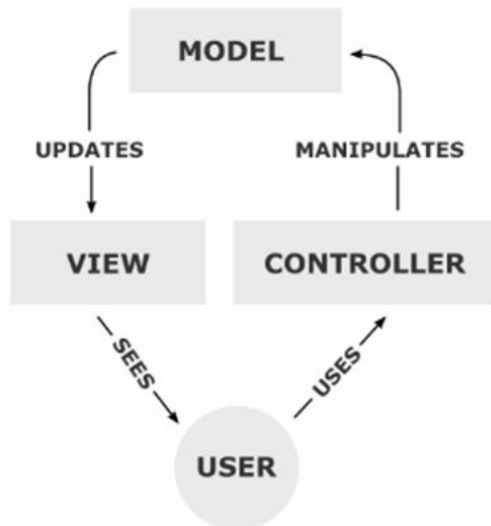
¿Qué tipo de cambios puede anticipar que van a realizar?

Al tratarse de un desarrollo incremental, en cada entrega anticipamos añadir nuevos módulos y funcionalidades tales como: inicio de sesión, inicio de sesión con redes sociales, gestión de pagos, soporte para divisas extranjeras, gestión de usuarios, roles de usuario.

Por cada cambio, ¿cuántos componentes requiere añadir, cambiar o borrar?

Se desarrolló teniendo en cuenta que se debe minimizar el número de componentes afectados por cada cambio. Para esto se siguió el principio de responsabilidad única que nos habla de que cada componente del programa, deberá concentrarse en sólo desempeñar una función única, buscando que, en caso de requerir alguna modificación de funcionalidad, las modificaciones a cada componente no alteren el funcionamiento entero del sistema o de componentes ajenos.

#### **Estilo arquitectónico, patrón o mecanismo implementado**



Además del principio de responsabilidad única, se utilizó en el desarrollo modelo-vista-controlador. En términos generales, esto es una extensión del principio de responsabilidad única pues el modelo es el que se encarga de los datos, la vista de cómo se ven estos datos y el controlador es el que manipula el modelo.

Seguir el principio de responsabilidad única ayuda significativamente a cambiar la funcionalidad del sistema debido a que los cambios no se propagan, es decir, se tiene que modificar únicamente el módulo donde se implementan los cambios. En otras palabras, la restricción de la comunicación o la reducción del número de módulos que consumen los datos de otros módulos, agiliza los cambios.

| Componente Modificado       | Campos que afecta  |
|-----------------------------|--|
| 1. Modificación de la Vista | En ocasiones:<br>El controlador, base de datos<br>Cambios frecuentes:<br>Únicamente la vista |



|   |   |
|---|---|
| <b>2. Modificación del Modelo</b>             | En ocasiones:<br>Act. de APIS<br>Frecuente:<br>El controlador, modelo     |
| <b>3. Modificación del Controlador</b>        | En ocasiones:<br>La vista<br>Frecuente:<br>La vista, Controlador          |
| <b>4. Modificación de base de datos</b>       | En ocasiones:<br>cambios en vista<br>Frecuente:<br>El modelo, controlador |
| <b>5. Sustitución o Actualización de APIS</b> | En ocasiones:<br>la vista<br>Frecuente:<br>El modelo, base de datos.      |

### Matriz de dependencias

| No. de dependencia | 1 | 2 | 3 | 4 | 5 | <b>Keys:</b><br><b>1. Vista</b><br><b>2. Modelo</b><br><b>3. Controlador</b><br><b>4. Base de datos</b><br><b>5. API</b> |
|--------------------|---|---|---|---|---|--|
| 1                  | x |   |   |   |   |  |
| 2                  |   | x | x |   |   |  |
| 3                  | x |   | x |   |   |  |
| 4                  |   | x | x | x |   |  |
| 5                  |   | x |   | x | x |  |

## Riesgos

En caso de no seguir este atributo de calidad, se presenta un riesgo muy grande que es el no poder cumplir con los requerimientos y funcionalidades para la siguiente etapa de entrega del proyectos. Esto puede deberse a que se provoque una propagación de cambios donde para añadir cierta funcionalidad se tengan que modificar la mayor parte de los módulos del proyecto.

Otro de los riesgos es que el sistema no ha sido probado en producción, solamente en el equipo local donde se hicieron los cambios.

## Puntos de sensibilidad

En esta sección se habló de mantener la funcionalidad separada en módulos. Sin embargo, hay que mencionar que esto no debe hacerse en exceso, ya que esto podría provocar malos olores, en específico, tener clases que no cumplen con una funcionalidad base para el sistema y que deban ser reagrupadas en otra clase.

**Atributo 5:** Performance - Latency - Los servicios de operaciones bancarias no deberán de tomar más de 5 segundos para ser creadas o continuar con el flujo establecido de estados conforme a la acción efectuada, en ambiente de desarrollo.

## ¿De qué necesidad de negocio se deriva?

Smart Pay Co se quiere posicionar como una de las empresas de transacciones bancarias de clase mundial. Y para ello requiere el rendimiento necesario para efectuar de forma ágil las operaciones.

## Preguntas de refinamiento

¿Se utiliza uno o varios hilos?

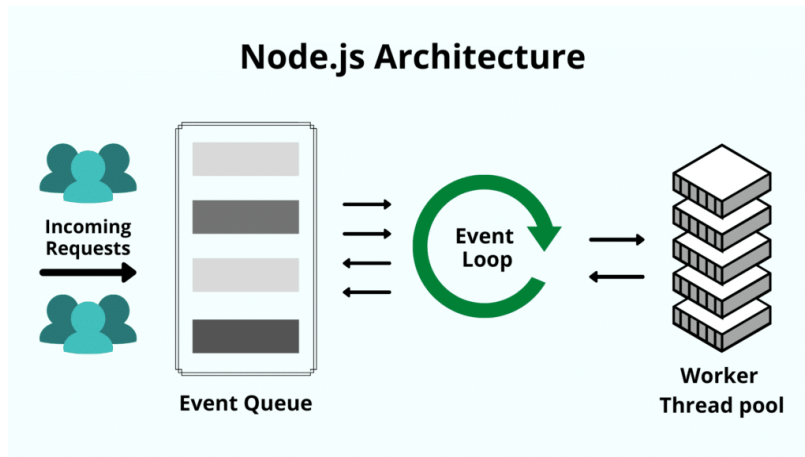
NodeJS utiliza un solo hilo.

¿Los hilos se bloquean hasta completar una petición?

El hilo de NodeJS no se bloquea, permite atender varias peticiones al mismo tiempo.

## Estilo arquitectónico, patrón o mecanismo implementado

El uso de NodeJS fue la respuesta a este atributo de calidad. Esto se debe a que es escalable, permite uso de datos intensivos, y es ideal para aplicaciones en tiempo real. Tiene una arquitectura asincrónica, no bloqueante. Es decir, hay un hilo único que se encarga de atender todas las solicitudes y mientras la base de datos ejecuta el query, el hilo se utiliza para atender a otra solicitud. Esto permite que podamos servir a más clientes con un rendimiento alto, sin necesidad de más hardware.



## Riesgos

Uno de los principales riesgos es el versionamiento del aplicativo. Esto se refiere específicamente a que cierta versión no funcione en todas las plataformas o que nos encontremos con inconvenientes en las actualizaciones y que afecte la funcionalidad del sistema.

Un número muy elevado de usuarios también podría disminuir significativamente el rendimiento, véase la sección de escenario de crecimiento.

## Puntos de sensibilidad

NodeJS no es ideal para su uso en aplicaciones que sean intensivas en su procesamiento como software de manipulación de imágenes, entre otros.

**Atributo 6:** Usability - Support User Initiative - Uso intuitivo y práctico de la interfaz de usuario para realización de operaciones. El usuario deberá familiarizarse con el sistema en menos de una semana.

## ¿De qué necesidad de negocio se deriva?

Se trata de un sistema el cual cuenta con mucha interacción directa con el usuario final, por lo que Smart Pay Co desea proporcionar a sus clientes un sistema fácil de usar e intuitivo.

## Preguntas de refinamiento

¿Quién es el usuario final?

El usuario final del sistema son los empleados de Smart Pay Co, en específico, los gestores de cuenta, y administradores.

¿El sistema tiene una alta interacción con el usuario?

El sistema fue diseñado con el propósito de facilitar el trabajo al personal de Smart Pay Co y por lo tanto, interactuar de diferentes maneras con el usuario.

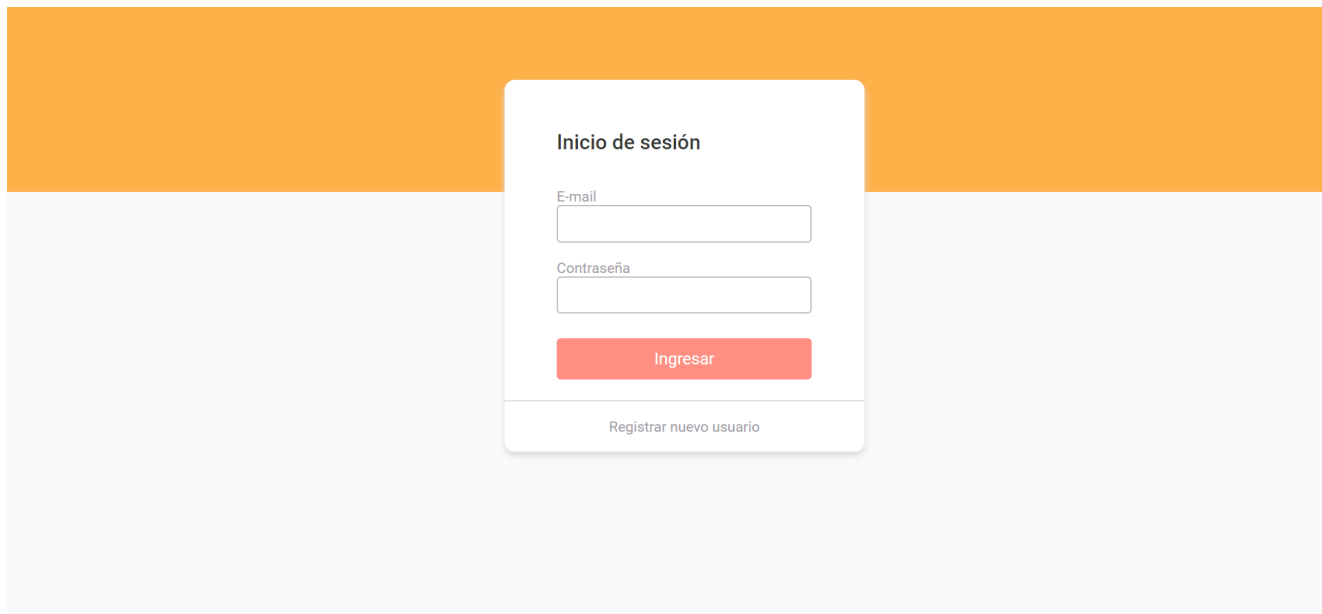
¿El sistema requiere manual de usuario?

Debido al alcance del proyecto no se contempló el desarrollo de un manual de usuario, por lo que la interfaz de usuario del sistema debe ser lo suficientemente autodescriptiva para que cualquier persona que conozca el contexto de la aplicación pueda descubrir su funcionalidad y acostumbrarse rápidamente a su uso.

### **Estilo arquitectónico, patrón o mecanismo implementado**

El sistema ayuda al usuario proporcionándole feedback sobre lo que está realizando. Esto es, mostrarle el nombre de la pantalla, textos claros en las etiquetas y en los botones. De igual forma, se le da al usuario la posibilidad de cancelar, deshacer tareas, volver a la pantalla anterior.

Esto se logra teniendo una interfaz de usuario, que contiene elementos y sigue las mejores prácticas de User Experience como uniformidad, herramientas de navegación y simplicidad.



The login form is centered on a page with an orange header and a light gray background. The form itself is white with rounded corners and a subtle shadow. It features a title 'Inicio de sesión' at the top. Below the title are two input fields: 'E-mail' and 'Contraseña'. A red 'Ingresar' button is positioned below the password field. At the bottom of the form, there is a link 'Registrar nuevo usuario'.

Inicio de sesión

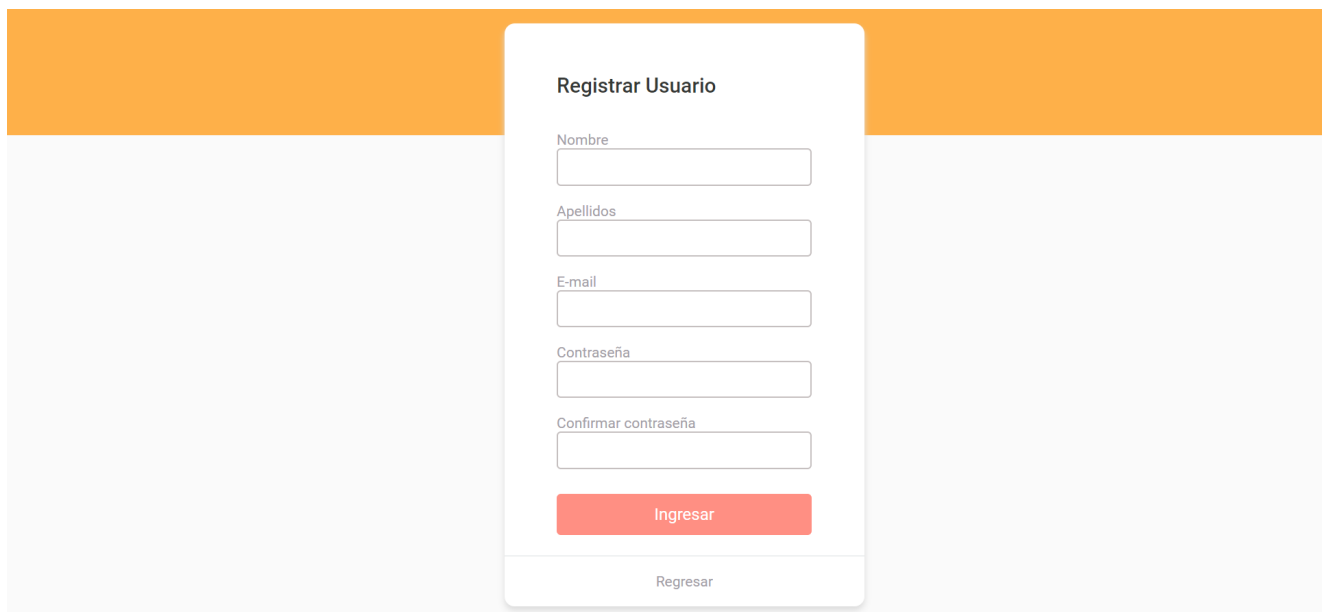
E-mail

Contraseña

Ingresar

Registrar nuevo usuario

Fig. Interfaz de inicio de sesión



The registration form is centered on a page with an orange header and a light gray background. The form is white with rounded corners and a subtle shadow. It has a title 'Registrar Usuario' at the top. Below the title are five input fields: 'Nombre', 'Apellidos', 'E-mail', 'Contraseña', and 'Confirmar contraseña'. A red 'Ingresar' button is located below the password fields. At the bottom of the form, there is a link 'Regresar'.

Registrar Usuario

Nombre

Apellidos

E-mail

Contraseña

Confirmar contraseña

Ingresar

Regresar

Fig. Registro de usuario

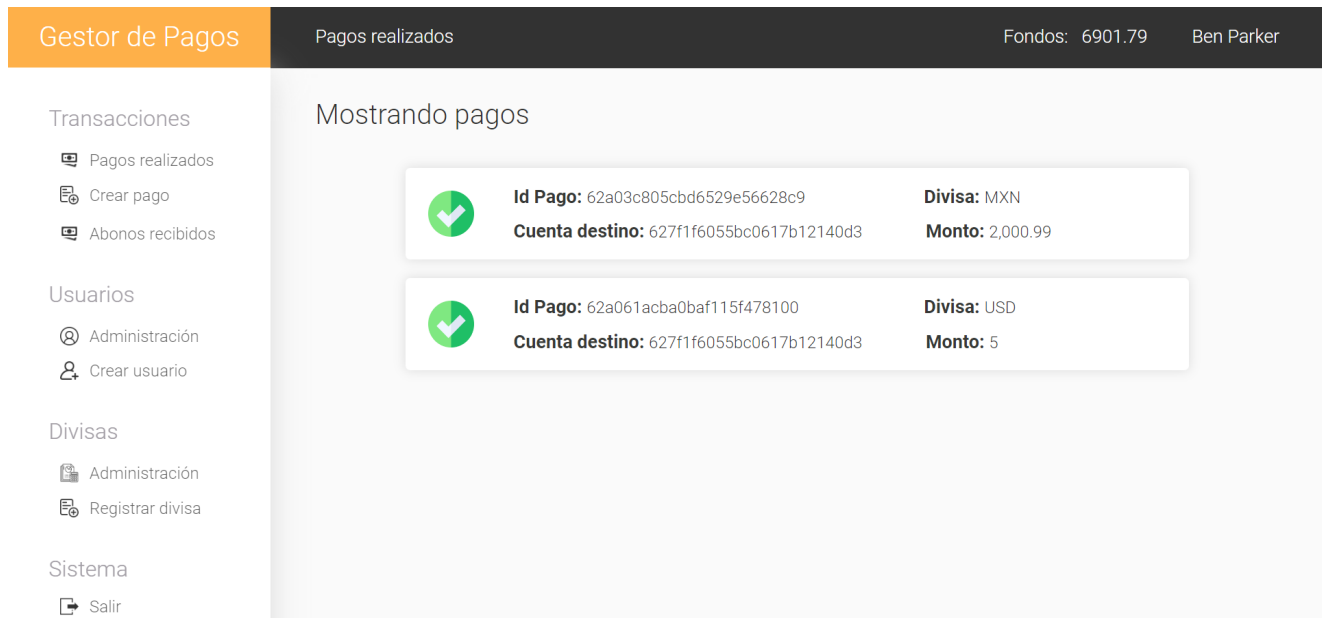


Fig. Interfaz de pagos realizados

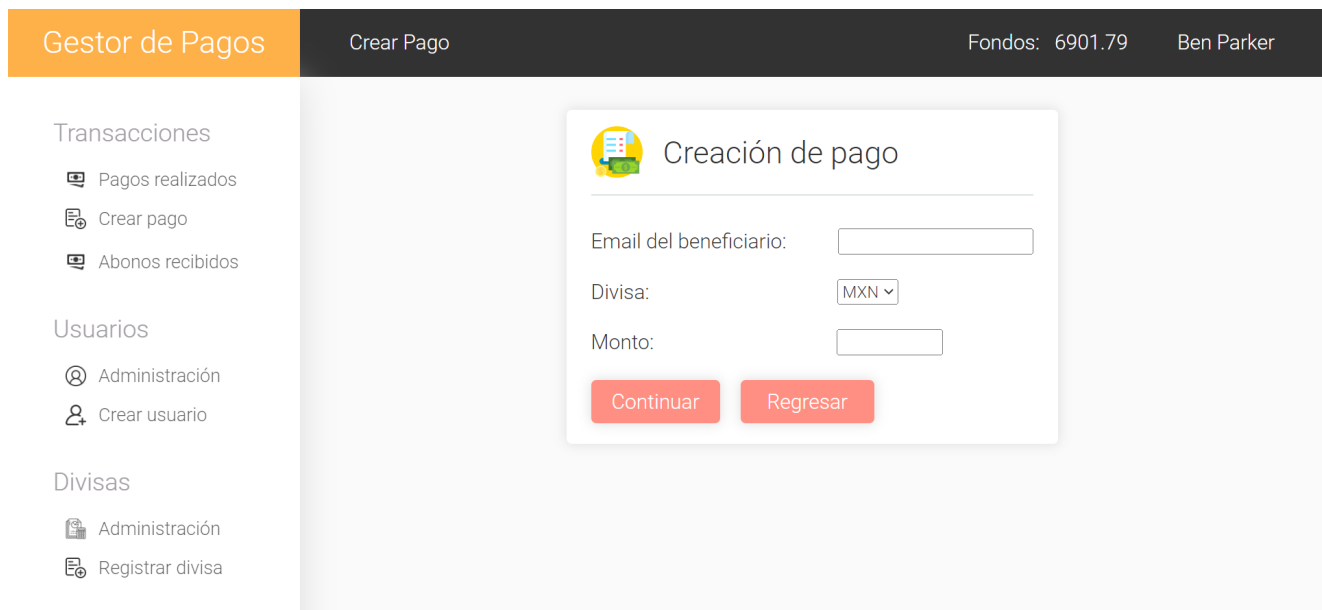


Fig. Interfaz de creación de pago

Gestor de Pagos

Administración de usuarios

Fondos: 6901.79 Ben Parker

Transacciones

Pagos realizados

Crear pago

Abonos recibidos

Usuarios

Administración


Crear usuario

Divisas

Administración

Registrar divisa


Mostrando usuarios



**Id Usuario:** 627aef977677b43f1eef8596 **Rol:** Superadmin

**Email:** jose@gmail.com


**Nombre(s):** Jose **Apellido(s):** Heredia



**Id Usuario:** 627aefb47677b43f1eef859c **Rol:** Admin

**Email:** carlos@gmail.com

**Nombre(s):** Carlos **Apellido(s):** Conde Besil



**Id Usuario:** 627df578e01d3c46729f466f **Rol:** client

**Email:** samuel@gmail.com

**Nombre(s):** Samuel **Apellido(s):** Kessner

Fig. Interfaz de administración de usuarios

Gestor de Pagos

Divisas extranjeras

Fondos: 6901.79 Ben Parker

Transacciones

Pagos realizados

Crear pago

Abonos recibidos

Usuarios

Administración

Crear usuario

Divisas


Administración

Registrar divisa

Sistema


Salir

Divisas extranjeras




**Siglas:** USD **Estado:** Activada

**Descripción:** Dolar americano



**Siglas:** EUR **Estado:** Activada

**Descripción:** Euro



**Siglas:** CAD **Estado:** Desactivada

**Descripción:** Dolar Canadiense

Fig. Interfaz de divisas extranjeras

The screenshot displays the 'Gestor de Pagos' application. The top navigation bar is orange with the title 'Gestor de Pagos'. The main header is dark grey, showing 'Registrar Divisa' on the left, and 'Fondos: 6901.79' and 'Ben Parker' on the right. A left sidebar contains three sections: 'Transacciones' (with 'Pagos realizados', 'Crear pago', and 'Abonos recibidos'), 'Usuarios' (with 'Administración' and 'Crear usuario'), and 'Divisas' (with 'Administración' and 'Registrar divisa'). The main content area features a modal window titled 'Formato registro de divisa' with a yellow icon. Inside the modal, there are three input fields: 'Siglas (3 dígitos):', 'Descripción:', and 'Estado:' (a dropdown menu currently showing 'Activada'). A red 'Crear' button is at the bottom of the modal.

Fig. Registro de divisa

Como se puede observar, en cada una de las pantallas de la interfaz principal, se cuenta con un menú del lado izquierdo en el que se puede navegar a través de las secciones del proyecto. Principalmente existen tres secciones las cuales son las transacciones, los usuarios y las divisas.

Cada una de las pantallas están diseñadas para tener una funcionalidad en específico solamente. Para darle al usuario una noción sobre lo que está haciendo, se despliega en la parte superior de la pantalla una etiqueta que describe la funcionalidad.

Finalmente, para proporcionar la mejor experiencia de usuario y una alta usabilidad, se presenta una interfaz sencilla con pocos elementos distribuidos en toda la pantalla que no abruman al usuario y que lejos de eso, lo invitan a interactuar con el sistema.

## Riesgos

Interfaz que resulta tan autodescriptiva que llega a saturar al usuario de información.

## Puntos de sensibilidad



Demoras en el tiempo de entrega debido al nivel de detalle de la interfaz.

**Atributo 7:** Testability - Manage Input/Output - Optimizar tiempos de ejecución de pruebas para el mantenimiento del sistema. Las pruebas deberán diseñarse y ejecutarse en menos de 3 horas en ambiente de desarrollo.

#### ¿De qué necesidad de negocio se deriva?

Durante el levantamiento de requerimientos y revisión de la rúbrica, la presencia de pruebas unitarias con Jasmine se hicieron presentes.

#### Preguntas de refinamiento

¿Qué tipos de pruebas se desea realizar?

Se realizarán pruebas unitarias sobre las principales funcionalidades del sistema.

¿Con qué herramienta se realizarán las pruebas?

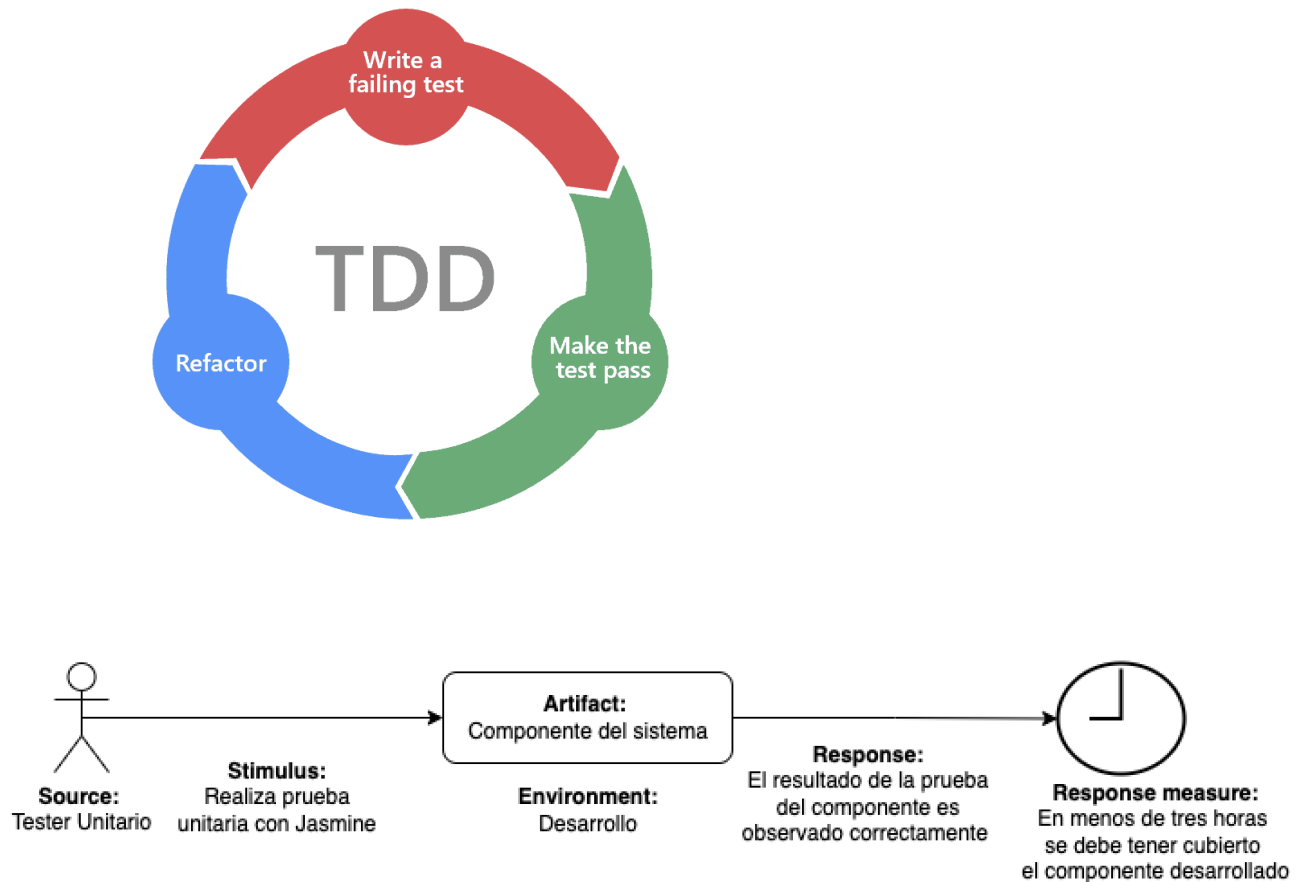
La principal herramienta donde se realizan las pruebas es Jasmine. Asimismo, existe la posibilidad de utilizar mocha-chai el cual cuenta con un funcionamiento muy similar, con únicamente ligeros cambios en la sintaxis.

¿Las pruebas interactúan con la base de datos?

La mayor parte de las pruebas cuentan con funciones que se ejecutan antes de comenzar y al finalizar el conjunto de pruebas (beforeAll, afterAll). Aparte de eso, existen funciones que se ejecutan antes y después de cada una de las pruebas (beforeEach, afterEach). Dentro de la ejecución de las pruebas, la base de datos es modificada por lo que cualquier prueba debe realizarse en ambiente de desarrollo.

#### Estilo arquitectónico, patrón o mecanismo implementado

Se siguió la metodología Test-Driven Development la cual consiste en escribir primero las pruebas y posteriormente desarrollar el código que cumple con las mismas. Esto da como resultado un código bastante más robusto, más seguro, mantenible y una mayor rapidez de desarrollo.



El uso del principio de responsabilidad única también facilita significativamente las pruebas, debido a que tras realizar un cambio no es necesario testar toda la aplicación, sino solamente el módulo que se cambió.

### Riesgos

Cambios en la base de datos: Debido a que en sistemas con base de datos, la ejecución de las pruebas suele cambiar los datos almacenados. En el caso particular de nuestro proyecto, hay pruebas que vacían las base de datos antes de comenzar.

Otro riesgo presente en este atributo es la versión de las herramientas de testing. En este caso en particular utilizamos la versión de Jasmine v4.1.0 que soporta las versiones 14 y 16 de Node. En caso de que se presente una actualización, podría dejar de funcionar temporalmente hasta que se libere el parche.

## **Puntos de sensibilidad**

Disminución de productividad: Debido a que este enfoque de desarrollo es relativamente nuevo para el equipo, se podría presentar una baja de productividad sobre todo durante las primeras entregas, debido a la curva de aprendizaje.

## **4.9 Commercial Off-The-Shelf (COTS)**

### **COTS de la interfaz de usuario (front-end)**

React v18.1.0 es una librería open-source de Javascript que se utiliza para desarrollar interfaces de usuario. Actualmente es mantenida por Facebook así como desarrolladores independientes. Actualmente es una de las tecnologías front-end más utilizadas.

React-router-dom 6.0.2 es el manejador de rutas para el front-end y el cambio de vistas.

material UI 5.1.0 es una librería de react que apoya para el uso de componentes, íconos o cualquier apoyo visual de la página de manera más sencilla

axios 0.24.0 es una librería de promesas de node.js de tal manera que el browser está basado en los protocolos de HTTP.

Styled-components 11.3.0 es una librería de react que nos ayuda al estilo de la página. (css) de manera más directa y entendible.

### **COTS del lado del servidor MVC (back-end)**

NodeJS v16.14 es un ambiente de ejecución de Javascript fuera del navegador, de código abierto. Utiliza el motor V8 de Google y además agrega módulos que proveen capacidades que no están en los navegadores tales como un sistema de archivos, red, entre otros.

Sendgrid 9.28.0 es un gestor de correo para desarrolladores, que provee una API por medio de la cual se pueden enviar correos electrónicos.

Mongoose 6.3.6 es un gestor de conexión para el manejo de una base de datos de tipo no relacional MongoDB.

Bcrypt 5.0.1 es una dependencia de Node que nos apoya para hashear la información sensible de manera que viaje segura dentro de cada requisito

Express 4.16.1 es el encargado de apoyar la gestión del modelo vista controlador, facilita la funcionalidad de operación en el back-end.

Passport 0.5.2 es el manejador de sesiones para uso de terceros como inicio de sesión con redes sociales. ayuda al manejo de la información.

nodemon 2.0.15 es el encargado de mantener un servidor corriendo y se actualice en cada cambio de datos en algún documento actualizando el servidor de manera automática.

#### 4.10 Pruebas unitarias

Para el apartado de realización de pruebas en el sistema, se optó por ejecutar pruebas unitarias. Dicha decisión favorece el correcto funcionamiento de cada uno de los componentes involucrados en la composición del producto final de software.

En cuanto a las herramientas utilizadas para dicha tarea, se contempló utilizar el framework *Mocha* y *Chai*, mismos que están desarrollados con el lenguaje de programación Javascript. De este modo, se plasmaron las *test suites* en archivos de javascript independientes, cada uno con acceso a las dependencias necesarias para ejecutar los casos de prueba, tales como acceso a la base de datos, peticiones a las rutas correspondientes de los microservicios, componentes visuales, etc.

Cabe destacar que el total de pruebas unitarias varía acorde al módulo en cuestión. Para este caso de documentación, nos permitimos ejemplificar una prueba unitaria como ejemplo, denotando en todo momento el funcionamiento del código y la representación de los resultados finales.

```
const mongoose = require('mongoose')
const Pago = require('../models/pago')
const Usuario = require('../models/usuario')
const { expect } = require('chai')
```

Fig. Importación de dependencias en archivo de pruebas unitarias

En primera instancia, requerimos de la importación de las librerías correspondientes para acceder a la base de datos, los modelos/clases de los objetos a instanciar y finalmente el método que nos permitirá comparar los resultados finales.

```
describe('Test Suite "pagos"', function () {
  beforeEach(function (done) {
    var mongoDB = 'mongodb://localhost:27017/payment_processor'
    mongoose.connect(mongoDB, { useNewUrlParser: true })

    const db = mongoose.connection
    db.on('error', console.error.bind(console, 'connection error'))
    db.once('open', function () {
      done()
    })
  })

  afterEach(function (done) {
    Usuario.deleteMany({}, function (err, success) {
      if (err) console.log(err)
      Pago.deleteMany({}, function (err, success) {
        if (err) console.log(err)
        if (err) console.log(err)
        const db = mongoose.connection
        db.close()
        done()
      })
    })
  })
})
```

Fig. definición de *suite* y acciones previas, posteriores a cada caso de prueba

Como segundo punto, tenemos la definición de nuestra *suite*, en este caso será la que contenga casos de prueba con relación a la operación de los pagos. Así mismo, se definen las acciones a realizar antes de cada prueba, teniendo en este caso la conexión con la base de datos para futuras consultas. Además, podemos ver la definición de acciones al término de cada prueba, siendo la eliminación de cualquier

instancia almacenada en base de datos durante las pruebas para evitar la alteración de los datos previamente registrados. También se considera el cierre de la conexión con la misma base de datos.

```
//Tests...
describe('TC:1 - Es posible crear un objeto de la clase Pago', () => {
  it('Paso 1 - Crear usuario ordenante', (done) => {
    let usuario = new Usuario({ nombre: 'Angel', apellidos: 'Heredia', email: 'angel@gmail.com', password: 'asdf', rol: 'Cliente', activo: true })
    Usuario.add(usuario, function (err, success) {
      Usuario.find({}, function (err, usuarios) {
        expect(usuarios.length).toBe.eq(1)
        done()
      })
    })
  })
  it('Paso 2 - Crear usuario beneficiario', (done) => {
    let usuario = new Usuario({ nombre: 'Carlos', apellidos: 'Conde', email: 'carlos@gmail.com', password: 'asdf', rol: 'Cliente', activo: true })
    Usuario.add(usuario, function (err, success) {
      Usuario.find({}, function (err, usuarios) {
        expect(usuarios.length).toBe.eq(1)
        done()
      })
    })
  })
  it('Paso 3 - Crear pago con relación a los usuarios creados', (done) => {
    Usuario.findOne({ email: "angel@gmail.com" }, function (err, ordenante) {
      Usuario.findOne({ email: "carlos@gmail.com" }, function (err, beneficiario) {
        let pago = new Pago({ _ordenanteId: ordenante._id, _beneficiarioId: beneficiario._id, monto: 100, divisa: "MXN", precioPorDivisa: 1, divisaApesos: 100 })
        Pago.add(pago, function (err, success) {
          Pago.find({}, function (err, pagos) {
            expect(pagos.length).toBe.eq(1)
          })
        })
      })
    })
  })
});
```

Fig. Definición del caso de prueba a ejecutar con pasos, descripción y validaciones

Por otra parte, tenemos la definición del caso de prueba, mismo que contará con pasos a ejecutar, teniendo en cada uno un punto a validar para definir el éxito o fracaso del proceso. En primera instancia, consideramos como caso de prueba el almacenamiento de una instancia de pago, creada previamente por un usuario creado y con un destinatario también creado, guardando en todo momento los registros en base de datos.

En otras palabras, verificamos en el primer paso que se pueda crear y almacenar un usuario ordenante del pago. En el segundo paso rectificamos que se pueda crear y almacenar un usuario beneficiario del pago. Finalmente, validamos que podamos crear un pago con solicitante al primer usuario creado y como beneficiario al segundo usuario creado, a la par de guardarlo en la base de datos.

```
> mocha spec/pruebas_unitarias

Test Suite "pagos"
  TC:1 - Es posible crear un objeto de la clase Pago
    ✓ Paso 1 - Crear usuario ordenante (201ms)
    ✓ Paso 2 - Crear usuario beneficiario (100ms)
    ✓ Paso 3 - Crear pago con relación a los usuarios creados (82ms)

3 passing (489ms)
```

Fig. Resultados de ejecución de pruebas unitarias en *suite* de pagos

Finalmente, podemos observar que al ejecutar los pasos en cada prueba unitaria de la *suite* correspondiente, pasamos todos los puntos de comparación-evaluación, resultando en éxito el proceso de pruebas del sistema (se cuentan con más pruebas en el sistema).