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

Elaborado por: .

Contenido

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 registrando la siguiente información:

- Cuenta de ordenante
- Cuenta de beneficiario
- Divisa
- Monto Total
- Monto Total en MXN

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. Los registros de pagos deberán contener los siguientes indicadores para que el usuario pueda ver la información necesaria:

- Id de pago
- Estado
- Fecha
- Cuenta origen
- Cuenta destino

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 ingresando la siguiente información:

- Nombre(s)
- Apellido(s)
- Número de cuenta
- Fondos

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: Dirección final de una URL utilizada en enrutamientos de Node.

Front End: Lógica de una aplicación que se caracteriza por las funcionalidades que suceden por delante de la aplicación. Estás 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_Desarrollo/links/57144e1408aeebe07c0641ab/Arquitectura-de-Software-Conceptos-y-Ciclo-de-Desarrollo.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

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 con el que va a contar, así como sus distintas capacidades y restricciones que tendrá 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.

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 cuentas de usuarios, editar cierta información, visualizar y borrar. De igual forma determina qué categoría de las 3 mencionadas la cuenta se encuentra.

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 preveé que los siguientes drivers permitan el desarrollo de los requerimientos funcionales del sistema, sirviendo como guía para el desarrollo funcional del sistema.

Driver funcional	**Descripción**	**Complejidad**	**Prioridad**
Inicio de sesión por medio de autenticación	Con la finalidad de otorgar un nivel de seguridad, así como de jerarquía en permisos	Media	Alta

de usuarios.	a los usuarios del sistema, se plantea el inicio de sesión con correo electrónico y contraseña con alto nivel de seguridad.		
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
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
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
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,	Media	Alta

	lectura, edición y eliminación (CRUD).		
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
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 plentea 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 no sea duplicada en ningún momento.	Media	Alta

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.

Driver de atributo de calidad	**Descripción**	**Complejidad**	**Prioridad**
Rechazo de cualquier usuario no registrado o inactivo en el sistema.	Atributo de calidad: **Security** Con la finalidad de evitar brechas de seguridad en el sistema y comprometer la información almacenada en el mismo, se plantea el rechazo de todo aquel usuario no	Media	Alta

	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: **Security** 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
Uso intuitivo y práctico de la interfaz de usuario para realización de operaciones.	Atributo de calidad: **Usability** 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
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: **Availability** 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
Optimizar tiempos de ejecución de pruebas para el mantenimiento del sistema.	Atributo de calidad: **Testability** 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	Alta	Baja

	prácticas de POO y principios SOLID, garantizando así una correcta separación de funcionalidad en módulos, haciendo las pruebas lo más óptimas posibles.		
Preparar el sistema para incorporar o editar módulos a la estructura base del manejo de pagos/transacciones bancarias de manera óptima.	Atributo de calidad: **Modifiability** 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 creadas o continuar con el flujo establecido de estados conforme a la acción efectuada.	Atributo de calidad: **Performance** Con la finalidad de garantizar la exactitud de la información bancaria manejada en cada transacción, se plantea que al crear un nuevo 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.	Alta	Alta

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 hara 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: - Google Chrome (a partir de versión 100.0.4896.127). - Microsoft Edge (a partir de versión 100.0.1185.44). - Mozilla Firefox (a partir de versión 99.0.1).	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.

Restricción técnica	**Descripción**
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

Describa 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.

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.

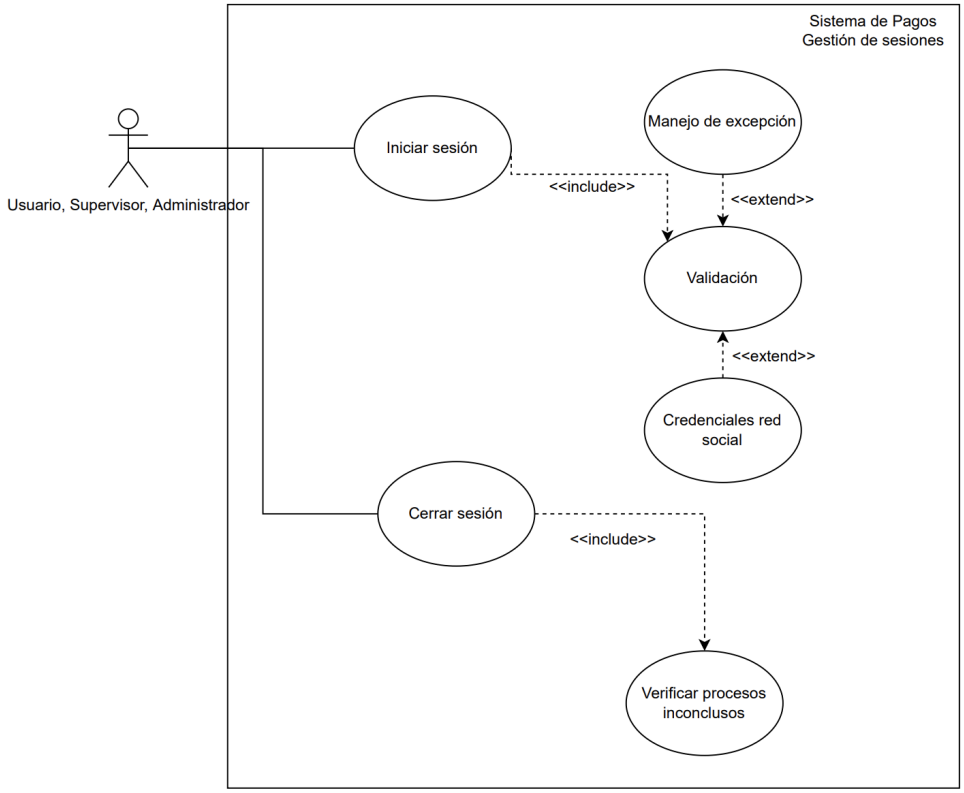
En segundo lugar, resultan atractivos en este patrón tanto su facilidad de desarrollo como de despliegue. Debido a que la funcionalidad está aislada en unidades, el desarrollo se mantiene sencillo pues su alcance es pequeño. Esto implica que la asignación de actividades y el trabajo en equipo es eficiente 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.

Gestión de sesiones



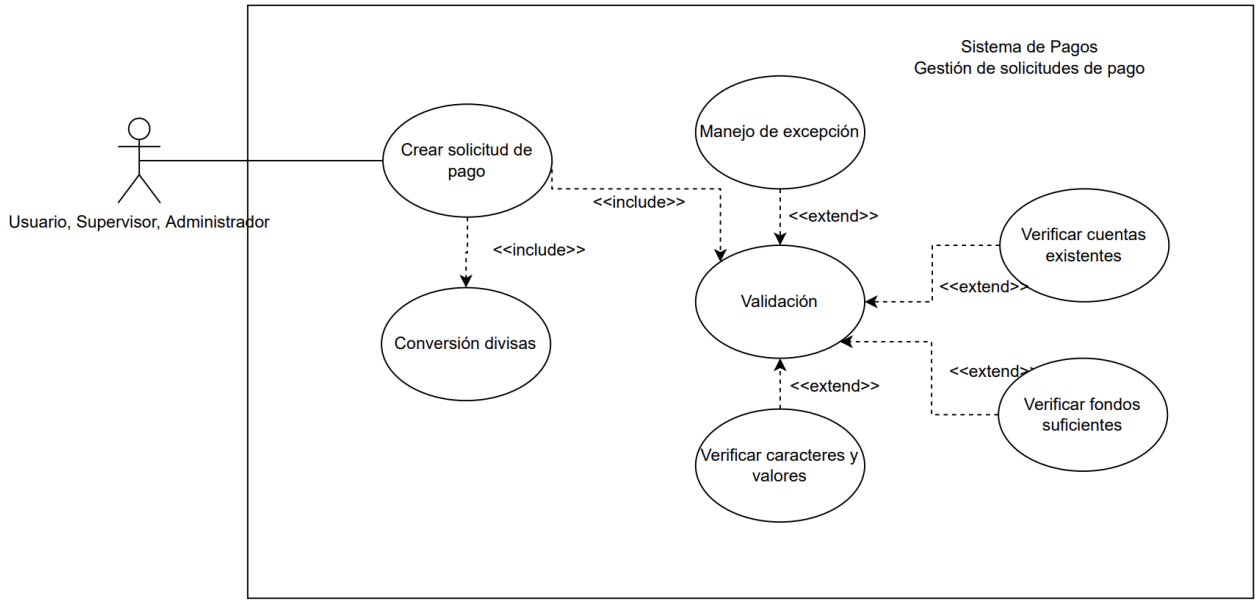
ESCENARIO I: Gestión de sesiones
Casos de uso 1: Inicio de sesión
Descripción: Permitir al usuario iniciar sesión para acceder al sistema con usuario y contraseña, o

bien autentificación por medio de una red social.
Precondiciones: El repositorio de información de usuarios debe estar actualizada y disponible.
Condición de éxito: Usuario ingresa al sistema de forma exitosa.
Stakeholders e interesados: ● Compañía: Desea dar acceso únicamente a empleados por medio del inicio de sesión. ● Empleados: Desean acceder al sistema para desempeñar su trabajo.
Actores primarios: Usuario base, Supervisor, Administrador
Disparador: El usuario ha seleccionado la opción de inicio de sesión.
Eventos normales (Narrativa detallada): 1.1 El usuario ingresa al sistema para iniciar sesión. 1.2 El usuario ingresa su nombre de usuario, contraseña y selecciona iniciar sesión. 1.3 El sistema valida que el usuario esté registrado en el repositorio de información y sus credenciales son correctas. 1.4 El sistema redirige al usuario a la interfaz principal del software Smart Pay Co.
Extensiones o flujos alternativos: 1.2a Campo de usuario o contraseña no fueron llenados. El sistema muestra un mensaje de error en pantalla que pide al usuario verificar los campos. 1.3a Datos incorrectos. 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. 1.3b Datos del usuario no encontrados en repositorio de información. Se realiza una consulta para iniciar sesión por medio de una red social.

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.
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.

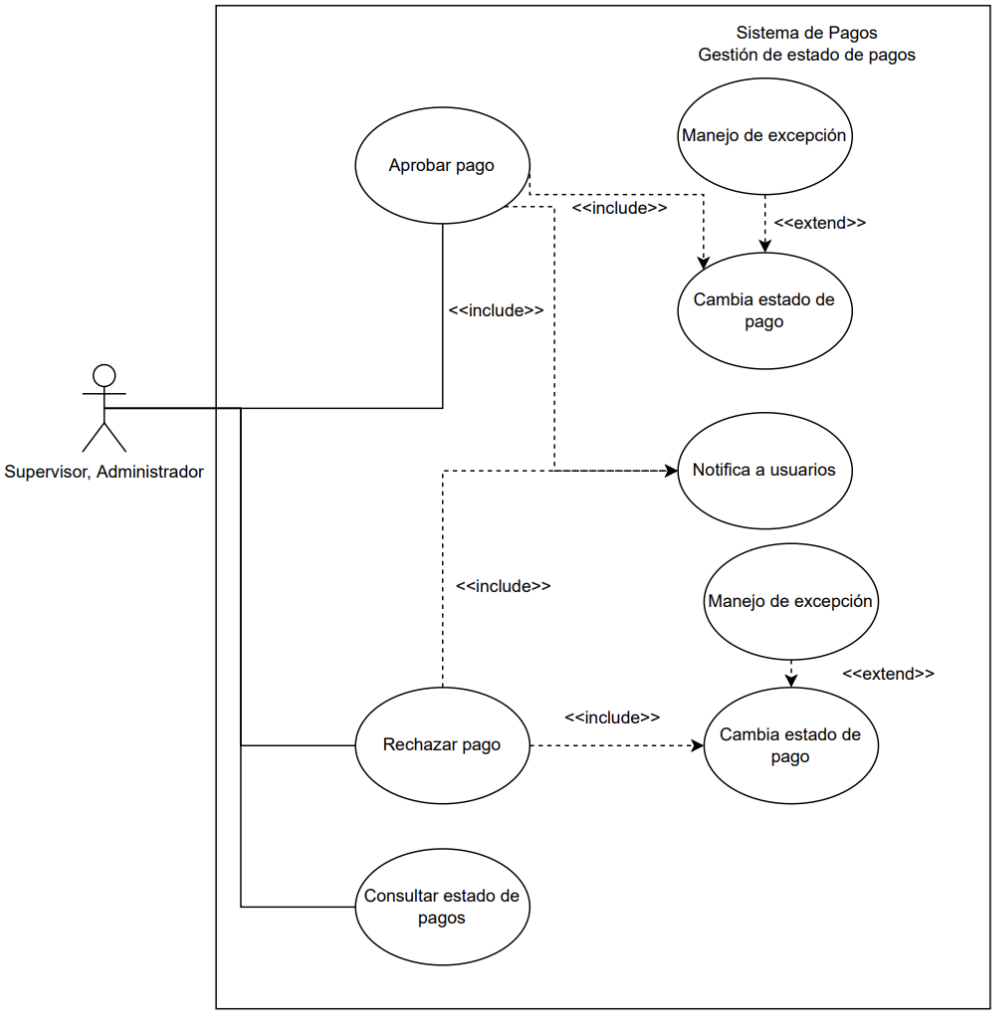
Gestión de solicitudes de pago



ESCENARIO II: Gestión de solicitudes de pago
Casos de uso 1: Crear solicitud de pago
Descripción: Permitir al usuario crear una solicitud de pago.
Precondiciones: El usuario debe haber iniciado sesión previamente.
Condición de éxito: Se genera correctamente una solicitud de pago que incluye cuenta destino, monto y divisa.
Stakeholders e interesados: ● Compañía: Desea ser capaz de procesar transacciones internacionales. ● Empleados: Desea solicitar los pagos de sus clientes y ganar comisiones. ● Clientes: Desea efectuar pagos.
Actores primarios: Usuario base, Supervisor, Administrador
Disparador: El usuario ha seleccionado la opción de nueva solicitud de 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 nueva solicitud de pago. 1.4 El usuario llena el formulario con el monto, la cuenta destino y la divisa del pago. 1.5 El sistema verifica que tanto la cuenta destino como el monto sean valores numéricos. 1.6 El sistema verifica en tiempo real el valor de las divisas, realiza la conversión a moneda nacional. 1.7 El sistema valida que las cuentas origen y destino sean existentes. 1.8 El sistema valida que la cuenta de origen cuenta con los fondos suficientes para efectuar la transacción. 1.9 El sistema registra la solicitud de pago.
Extensiones o flujos alternativos: 1.5a Sistema encuentra caracteres inválidos (no numéricos), o los campos no fueron llenados. El sistema despliega un mensaje de error indicando al usuario que debe revisar los campos del formulario. 1.6a Sistema no se comunica con el sistema de divisas. El sistema despliega un mensaje de error indicando que existe un problema de comunicación que se intente realizar la operación más tarde. 1.7a Cuenta de origen o destino inexistente. El sistema despliega un mensaje de error que indica que la cuenta de origen o destino no es válida. 1.8a Cuenta de origen no cuenta con fondos insuficientes. 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



ESCENARIO III: Gestión de estado de pagos
Casos de uso 1: Consultar estado de pagos
Descripción: Permitir al usuario consultar el pago y su estado.
Precondiciones: El usuario debe haber iniciado sesión previamente y contar con privilegios de

supervisor o administrador.
Condición de éxito: El sistema despliega correctamente el pago consultado, mostrando cuenta destino, cuenta origen, monto, divisa y estado (rechazado, aprobado, completado).
Stakeholders e interesados: ● Compañía: Desea contar con estadísticas y un registro histórico de transacciones. ● Empleados: Desea consultar transacciones de sus clientes. ● Clientes: Desea saber el estado de sus pagos realizados.
Actores primarios: Supervisor, Administrador
Disparador: El usuario ha seleccionado la opción de consultar 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 consultas y selecciona un pago. 1.4 El sistema despliega en pantalla el estado del pago (rechazado, aprobado, completado).
Extensiones o flujos alternativos: 1.3a 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.

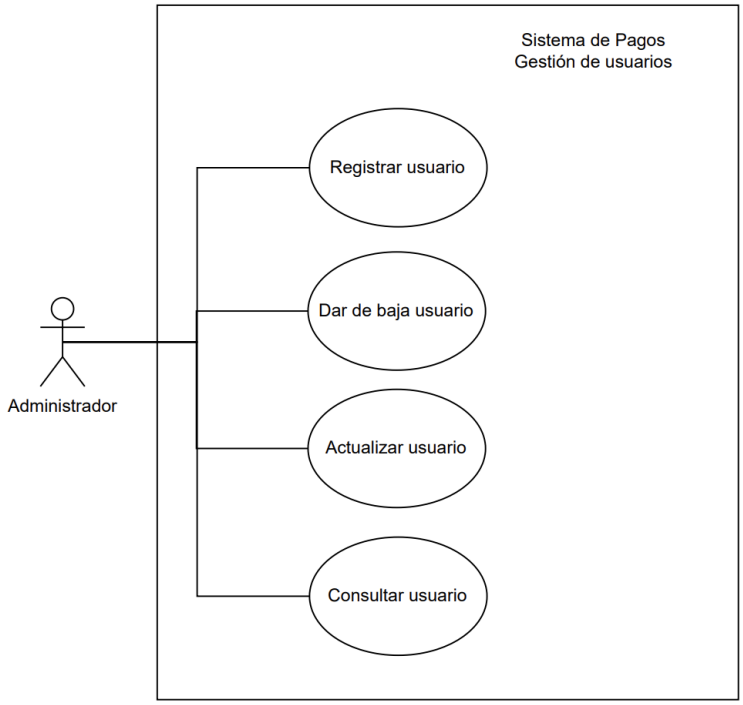
ESCENARIO III: Gestión de estado de pagos
Casos de uso 2: Aprobar pago
Descripción: Permitir al usuario aprobar pagos de sus clientes.
Precondiciones: El usuario debe haber iniciado sesión previamente y contar con privilegios de supervisor o administrador.
Condición de éxito: El estado del pago es actualizado correctamente a 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.
Condición de éxito: El estado del pago ha sido actualizado correctamente a rechazado.
Stakeholders e interesados: ● Compañía: Desea rechazar operaciones fraudulentas.

● Empleado: Desea rechazar operaciones sin fondos suficientes en la cuenta de origen. ● Cliente: Desea cancelar operaciones hechas por fraude, robo o clonación.
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 la cuenta destino, el monto, la divisa y selecciona rechazar el pago. 1.5 El sistema cambia el estado del pago a rechazado 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.
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.

Gestión de usuarios



ESCENARIO IV: Gestión de usuarios
Casos de uso 1: Crear usuarios
Descripción: Permitir a administradores crear nuevos 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: Un nuevo usuario ha sido registrado correctamente.
Stakeholders e interesados: ● Compañía (RH): Desea crear las credenciales de usuario para comenzar sus labores.
Actores primarios: 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.
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.
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.

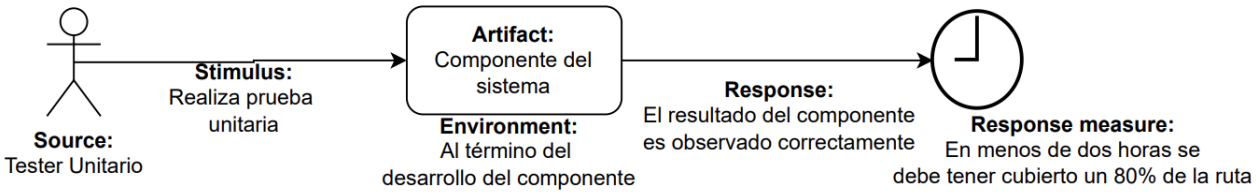
Extensiones o flujos alternativos: 1.5a Usuario cancela proceso de actualización. El sistema cancela el proceso de actualización y redirige al usuario a la interfaz anterior. 1.6a Información duplicada en repositorio de información. El sistema cancela el proceso de actualización y envía un mensaje notificando 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.
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.

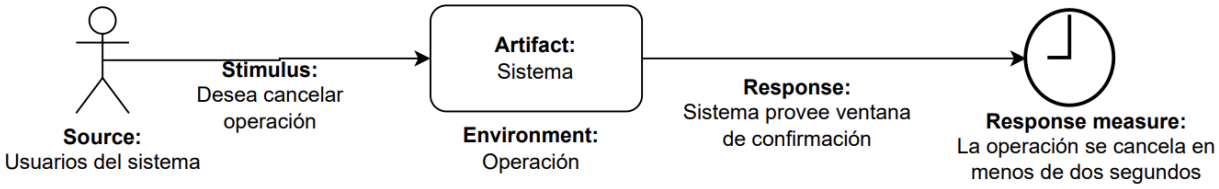
2.5 Escenarios Arquitectónicos y Requerimientos No Funcionales Asociados

2.5.1 Escenarios Arquitectónicos

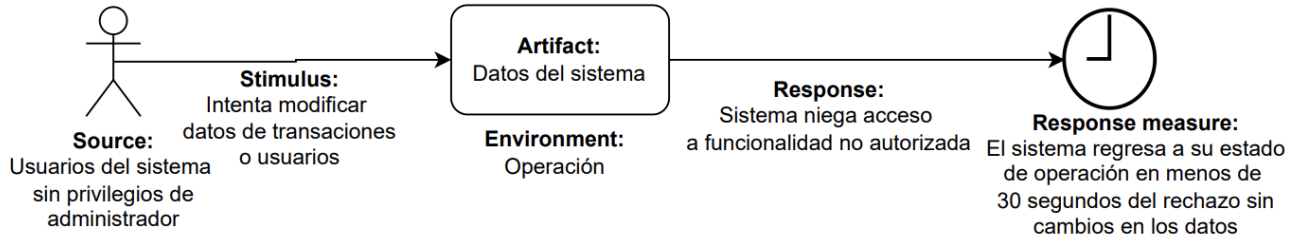
Testability



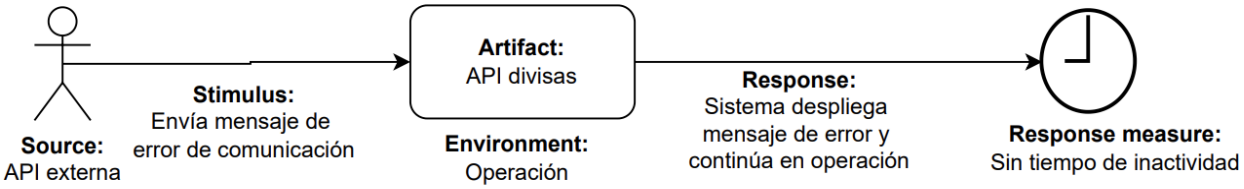
Usability



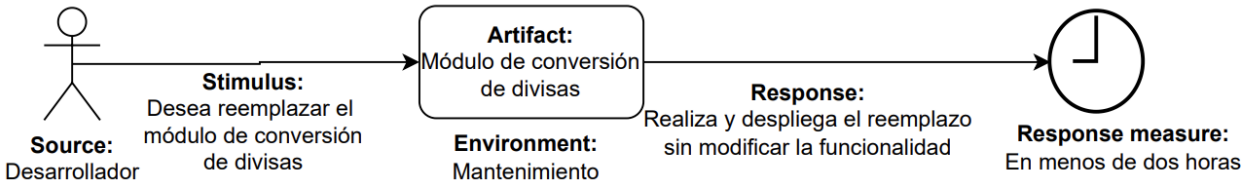
Security



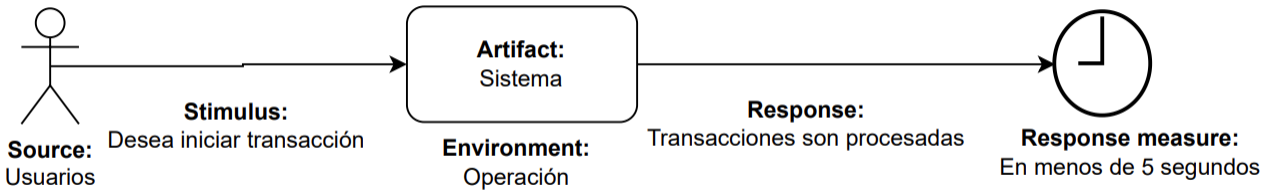
Availability



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.	

Usability:

RNF 0002	**El usuario que realice la operación debe identificar con facilidad los componentes necesarios para realizarlo.**
Operación de transacciones	
Durante la operación del sistema, se deberá identificar con gran facilidad las funcionalidades necesarias para lograr el objetivo del sistema. No deberá existir más de 3 saltos en la navegación de la página para poder lograr la operación deseada.	

Availability:

RNF 0003	**La aplicación solo será disponible durante las horas de trabajo del staff**
Horas de servicio	

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 0004	**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 *ripple effect* en las funcionalidades desarrolladas ya sea que afecte directa o indirectamente un comportamiento del sistema.	

Performance:

RNF 0005	**El sistema debe responder con velocidad la transacción cuando el usuario indique que se realice una operación del sistema.**
Velocidad de carga de transacciones	
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:

RNF 0006	**El sistema debe soportar cantidades considerables y simultáneas de operaciones dentro de la misma plataforma**

Cantidad de transacciones	
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 0007	**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:.

RNF 0008	**El sistema se podrá cargar desde distintos motores de navegación existente persistiendo con sus componentes funcionales y visuales**
Visualización de dispositivos	
El sistema será capaz de poder visualizarse en distintos navegadores como Chrome, Safari, Microsoft Edge, etc. 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.	

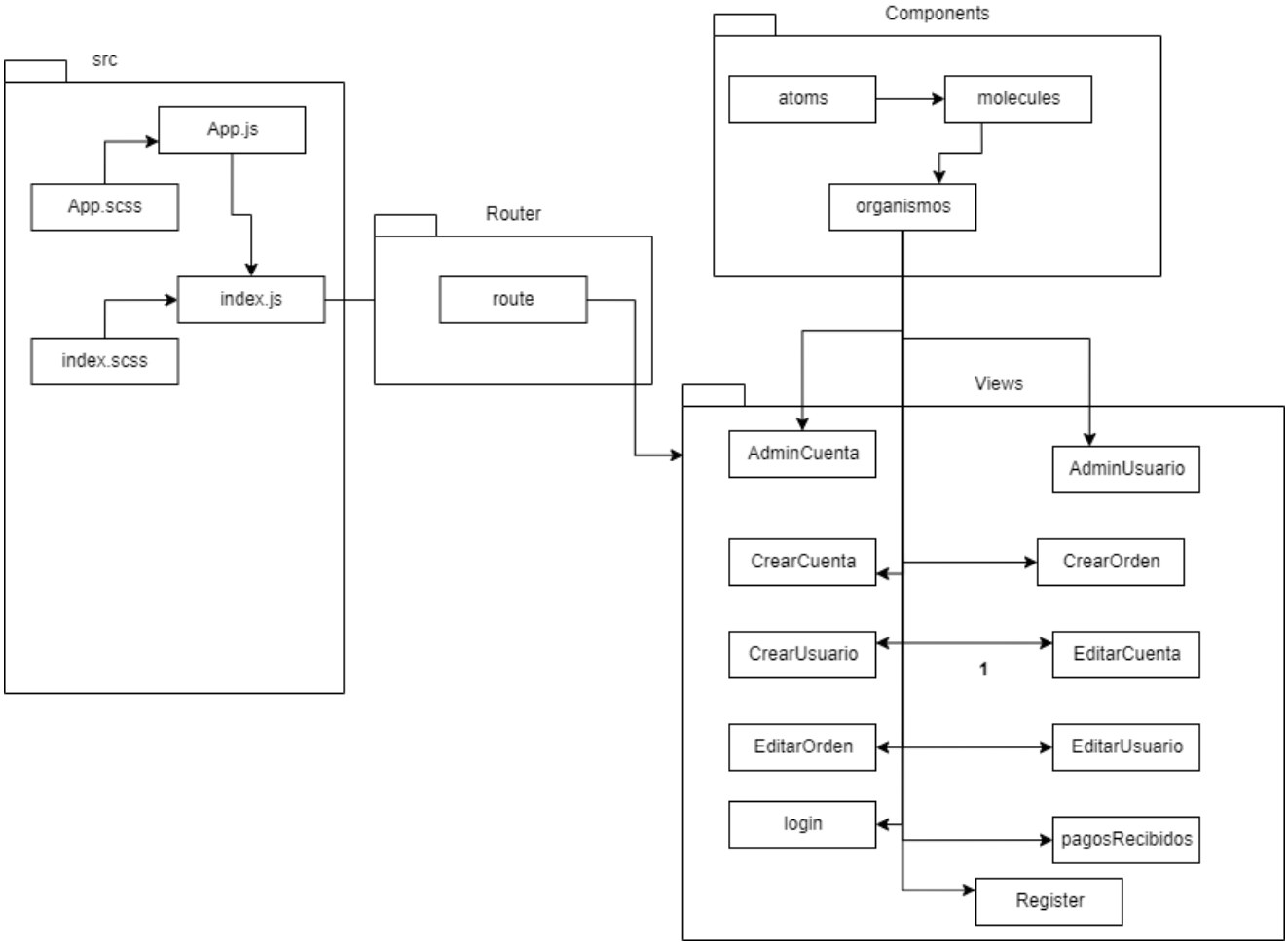
Security:

RNF 0009	**El sistema debe detectar la autentificación de los usuarios en cada sesión abierta del sistema**

Autentificación del Usuario	
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.	
RNF 0010	**El sistema debe mostrar los componentes que el usuario está autorizado hacer.**
Autorización del Usuario	
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.	

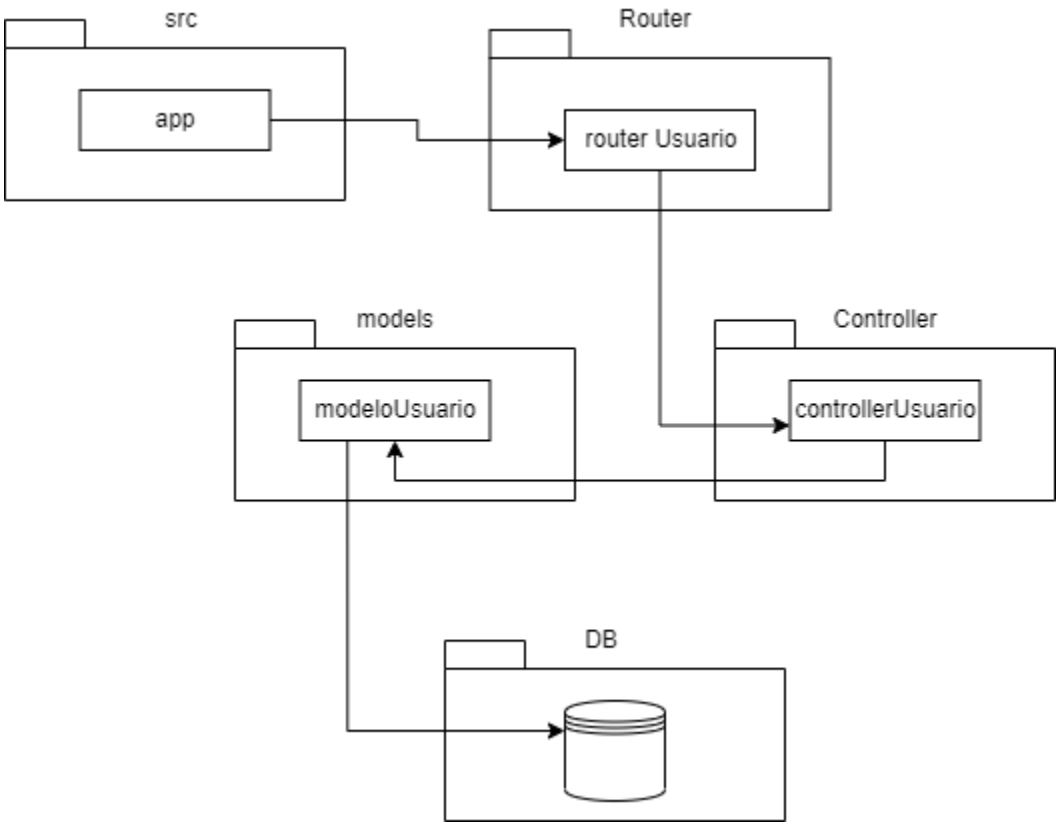
2.6 Vista lógica

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

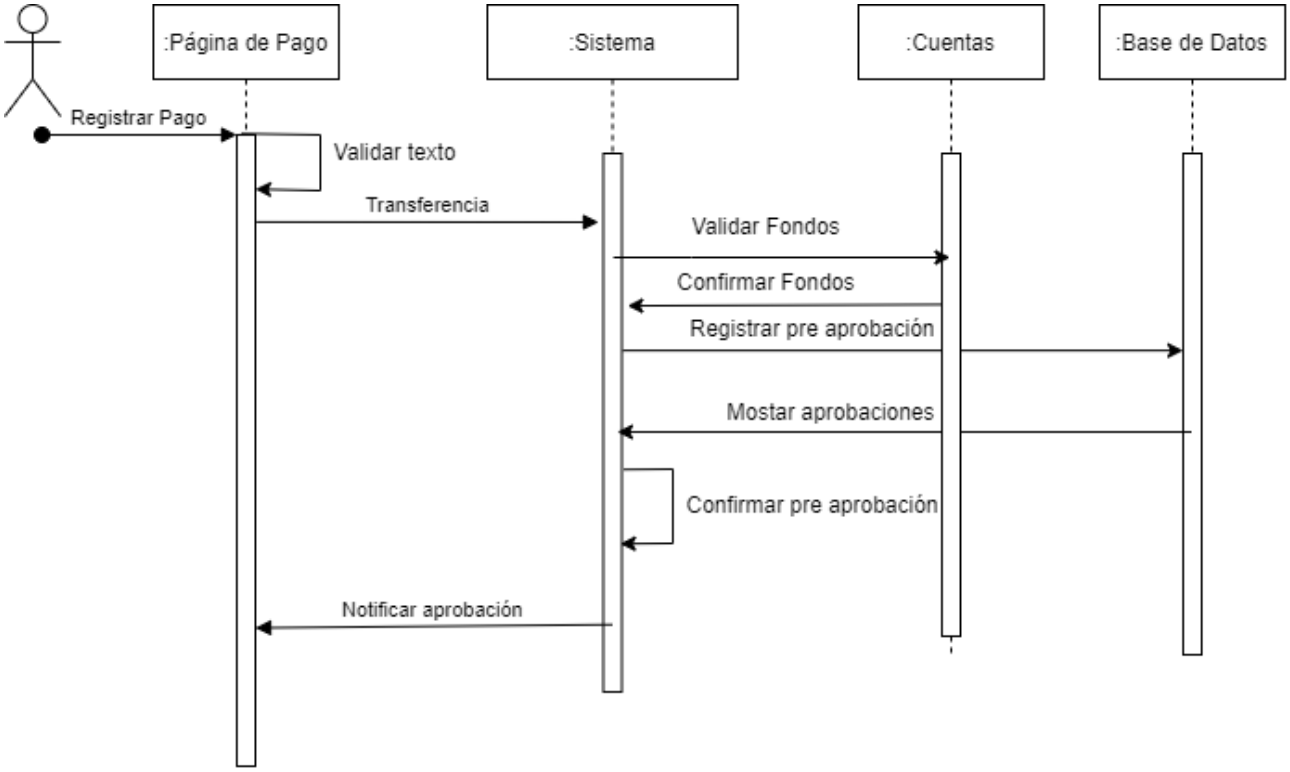


Existe un src que es el encargado de correr la estructura principal del sistema que después llama a unos enrutadores que son los determinantes de los endPoints y el redireccionamiento de información. Una vez realizado esto, el router accede al controlador que es el encargado de utilizar el modelo cuando 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.

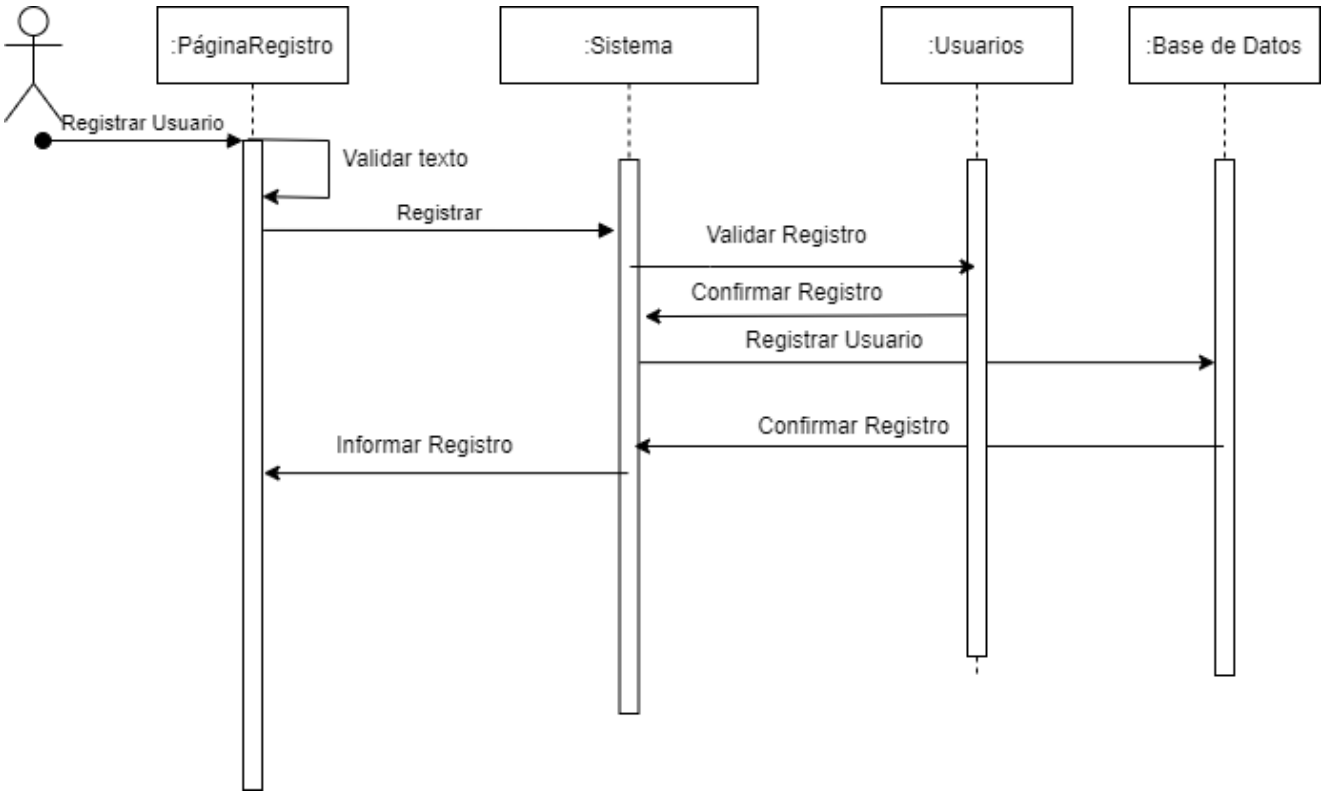
2.7 Vista de proceso

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

Realizar un registro de transacción

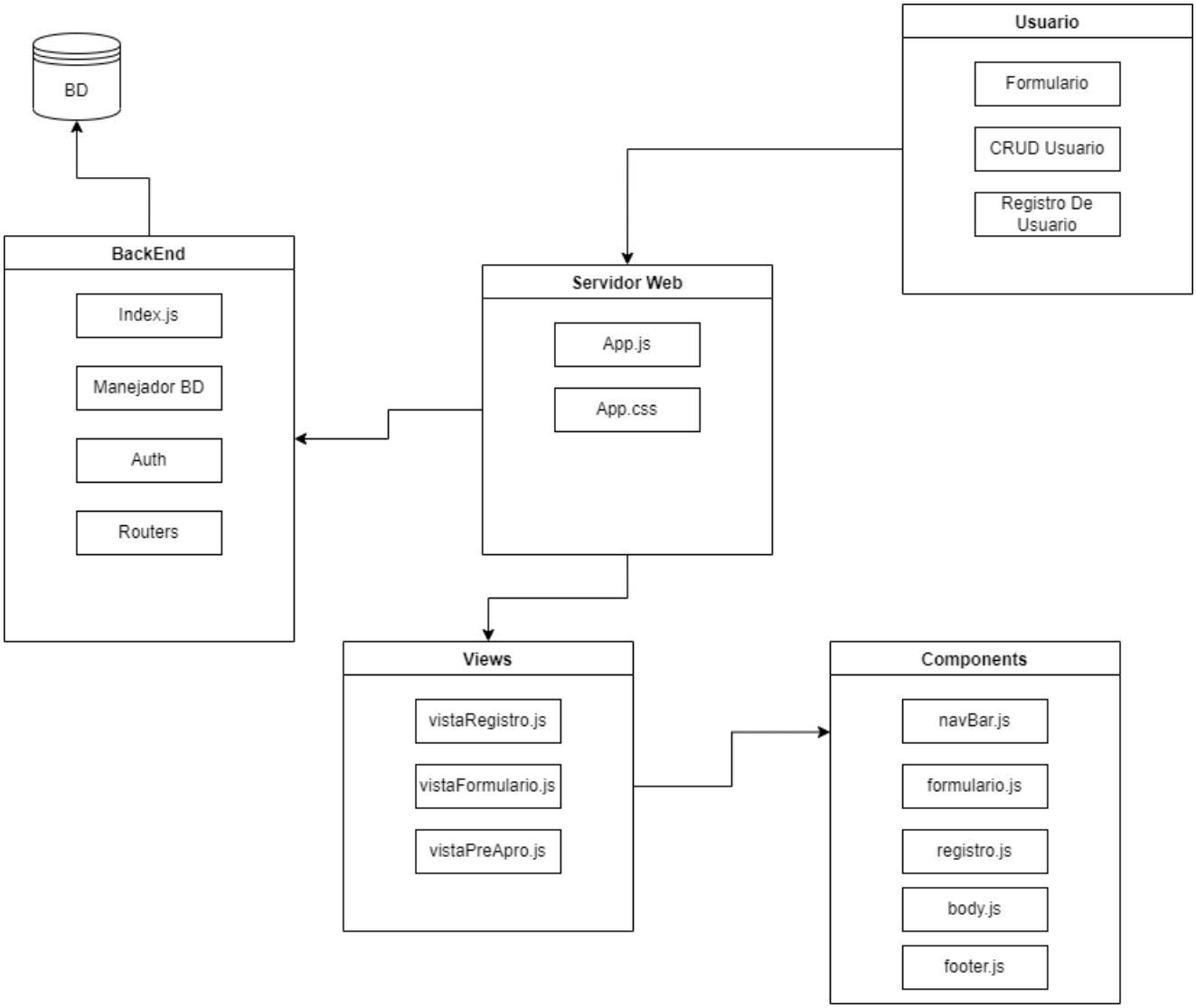


Registrar un nuevo usuario, supervisor o administrador



2.8 Vista de implementación

Descripción del diagrama de despliegue

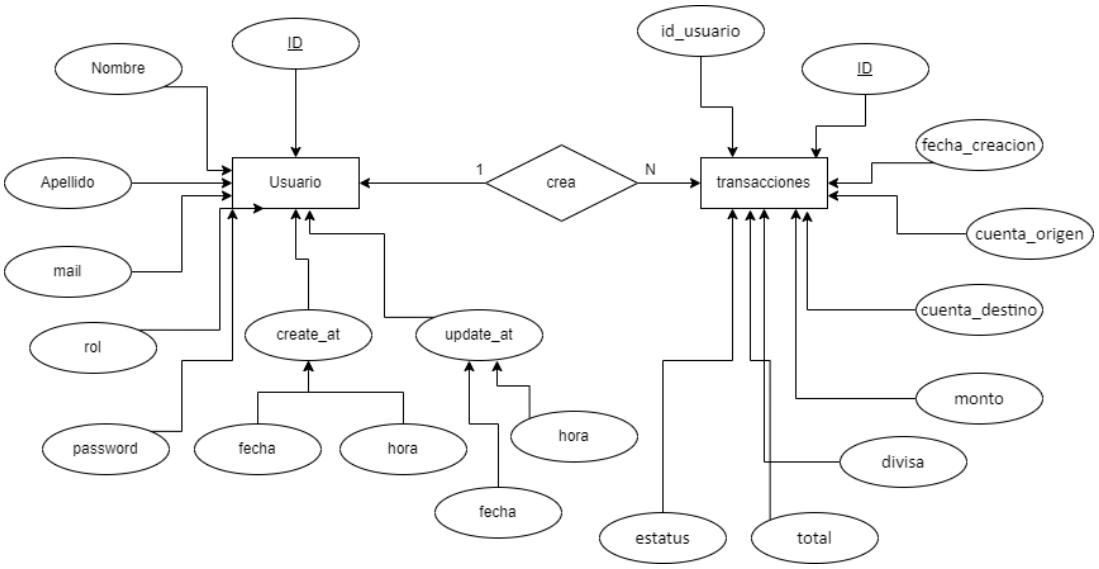


- Existe una vista de implementación del usuario, que son aquellos componentes visuales y físicos que se requieren para poder visualizar el sistema.
- El dispositivo ya será el encargado de acceder al servidor web para mostrar los formularios, la página de inicio de sesión o alguna otra funcionalidad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



2.9.1 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 autentificación 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

Describa 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* cómo 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 sobreescribir 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 sustituír 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 sustituída 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 funja 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 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 *Epic*, 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 una de las historias de usuario correspondientes al *Epic* denotado. 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 (historias de usuario).

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.

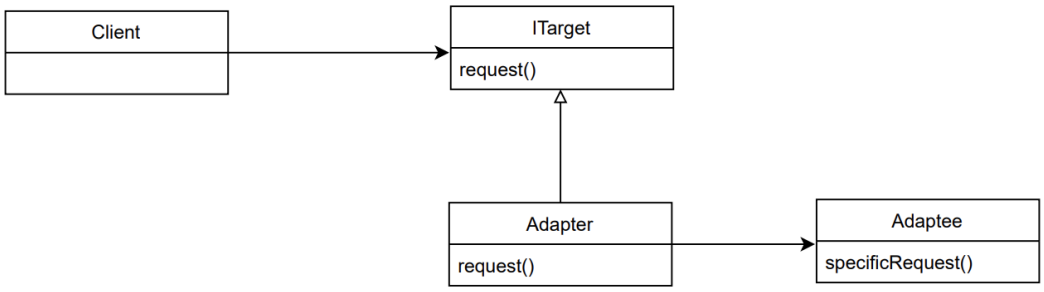
En React, existe un objeto llamado estado, que guarda los valores que pertenecen a un componente. Cada que cambia este objeto, se hace un render del componente en la interfaz de usuario. Teniendo esto en cuenta, es importante cuidar que no sea excesiva la cantidad de estados que se utilizan en el desarrollo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

Adicionalmente, es importante mencionar que aunque le 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.



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.

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.