Microkernel (plug-in)
Travelix

Daniel Toledo Daniel Machado Osvaldo Moreno José Antonio Concepción
C-311 - Agencia de Viajes. Equipo 3 Semigrupo 2.

1 de marzo de 2024

Índice

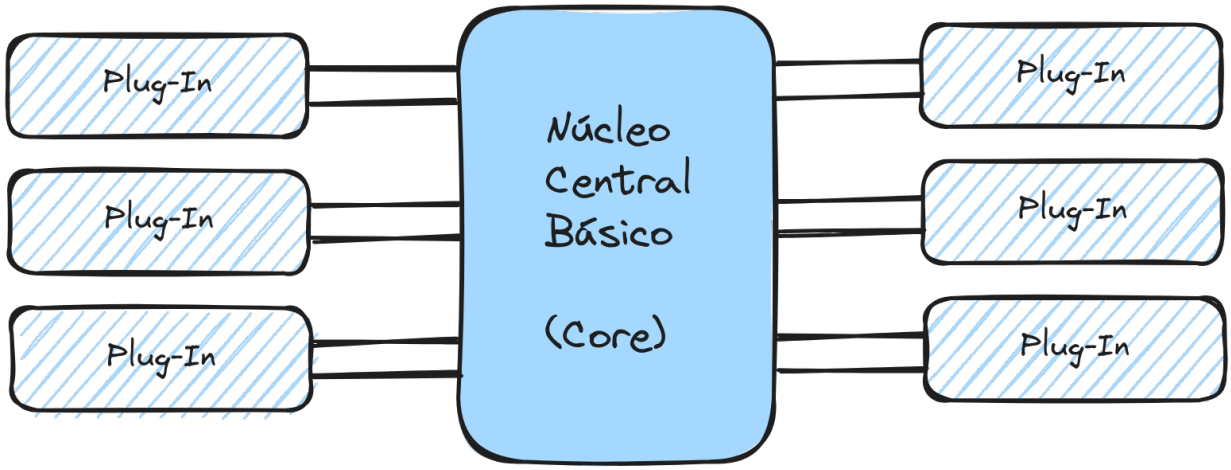
1. Microkernel o Plug-in

1.1. Información Teórica sobre la Arquitectura

El concepto de arquitectura de microkernel, también conocida como arquitectura de complementos, ha sido un término reconocido desde hace varias décadas y sigue siendo relevante en la actualidad. Este enfoque arquitectónico se integra de manera efectiva en aplicaciones basadas en productos, las cuales se distribuyen y se instalan como una sola implementación monolítica, generalmente en el entorno del cliente como un producto de terceros. Además, es ampliamente empleado en diversas aplicaciones comerciales personalizadas que no se consideran productos.

1.1.1. Estructura

La arquitectura de microkernel es una estructura monolítica relativamente sencilla que se compone de dos partes principales: un núcleo central y módulos complementarios. La lógica de la aplicación se distribuye entre los módulos complementarios independientes y el núcleo central básico. Esta distribución permite una mayor extensibilidad, adaptabilidad y aislamiento de las características específicas de la aplicación y la lógica de procesamiento personalizada.



1.1.2. Núcleo Central o Core

El sistema central, también conocido como núcleo, se define formalmente como la funcionalidad mínima necesaria para ejecutar el sistema. Un buen ejemplo de esto es el IDE de VSCode. El sistema central de VSCode es solo un editor de texto básico: abrir un archivo, cambiar texto y guardar el archivo. No es hasta que se agregan extensiones (complementos) que el IDE comienza a convertirse en un producto más robusto y utilizable.

Sin embargo, otra definición del sistema central es ´el camino feliz´ (flujo de procesamiento general) a través de la aplicación, con poco o ningún procesamiento personalizado. Mover la complejidad ciclomática del sistema central y colocarla en componentes complementarios separados permite una mejor extensibilidad, mantenibilidad y capacidad de prueba.

Por ejemplo, supongamos que una aplicación de reciclaje de dispositivos electrónicos debe realizar reglas de evaluación personalizadas específicas para cada dispositivo electrónico recibido. En este caso, las reglas de evaluación personalizadas se implementarían como componentes complementarios separados, mientras que el sistema central se encargaría de la funcionalidad básica como la gestión de dispositivos, el flujo general de la aplicación y la interacción con el usuario.

En lugar de integrar toda la personalización específica del cliente en el núcleo central, lo cual puede generar una alta complejidad ciclomática, es más eficiente desarrollar un módulo complementario independiente para cada dispositivo electrónico que se desea evaluar. Estos módulos específicos del cliente no solo aíslan la lógica del dispositivo del resto del proceso, sino que también facilitan la expansión. La inclusión de un nuevo dispositivo electrónico para su evaluación se reduce a añadir un nuevo módulo y actualizar el registro correspondiente. Con la arquitectura de microkernel, la evaluación de un dispositivo electrónico se simplifica al requerir que el núcleo central identifique e invoque los complementos de dispositivo apropiados.

1.1.3. Componentes Complementarios o Plug-Ins

Los componentes complementarios son componentes independientes y autónomos que contienen procesamiento especializado, funciones adicionales y código personalizado para mejorar o extender el sistema central. Además, se pueden usar para aislar código altamente volátil, creando una mejor mantenibilidad y capacidad de prueba dentro de la aplicación. Idealmente, los componentes complementarios deben ser independientes entre sí y no tener dependencias entre ellos.

La comunicación entre los componentes complementarios y el sistema central generalmente es punto a punto, lo que significa que la "tubería"que conecta el complemento al sistema central suele ser una invocación de método o una llamada de función a la clase de punto de entrada del componente complemento.

1.1.4. Beneficios y desventajas del acceso remoto a componentes complementarios

El acceso remoto a componentes complementarios implementados como servicios individuales presenta **ventajas**, como una mejor separación general de componentes, mayor escalabilidad y rendimiento, y la posibilidad de realizar cambios en tiempo de ejecución. También permite la comunicación asincrónica con complementos, lo que, según el escenario, podría mejorar significativamente la capacidad de respuesta general del usuario.

Sin embargo, estos beneficios también tienen **desventajas**. El acceso remoto a complementos convierte la arquitectura de microkernel en una arquitectura distribuida en lugar de monolítica, lo que dificulta su implementación y despliegue para la mayoría de los productos locales de terceros. Además, crea una mayor complejidad y costo general.

La elección de si hacer que la comunicación con los componentes complementarios desde el sistema central sea punto a punto o remota debe basarse en requisitos específicos y, por lo tanto, requiere un análisis cuidadoso de las ventajas y desventajas de dicho enfoque.

1.1.5. Registros

El sistema central necesita saber qué módulos complementarios están disponibles y cómo acceder a ellos. Una forma común de implementar esto es a través de un registro de componentes complementarios. Este registro contiene información sobre cada módulo complementario, incluyendo aspectos como:

- **Nombre**: Identifica de manera única al componente complementario.

- **Contrato de datos**: Define el formato y el significado de los datos intercambiados entre el sistema central y el componente complementario.
- **Detalles del protocolo de acceso remoto (si aplica)**: Solo en el caso de componentes que se conectan remotamente al sistema central, esta información especifica el protocolo utilizado para la comunicación.

El registro de componentes complementarios puede ser tan simple como:

- **Una estructura de mapa interna**: Propiedad del sistema central, que contiene una clave y la referencia al componente del complemento.
- **O una herramienta de registro y descubrimiento más compleja**: Puede estar integrada dentro del sistema central o implementada externamente.

La complejidad del registro dependerá de las necesidades específicas del sistema y del entorno de implementación.

1.1.6. Contratos

Los contratos entre los componentes complementarios y el sistema central generalmente son estándar dentro de un dominio de componentes complementarios e incluyen:

- **Comportamiento**: Describe las acciones que puede realizar el componente complementario.
- **Datos de entrada**: Define el formato y la estructura de los datos que el sistema central proporciona al componente complementario.
- **Datos de salida**: Define el formato y la estructura de los datos que el componente complementario devuelve al sistema central.

Los contratos personalizados se encuentran típicamente en situaciones donde los plug-ins son desarrollados por terceros y no se tiene control sobre el contrato utilizado. En tales casos, es común crear un adaptador entre el contrato del complemento y el contrato estándar para que el sistema central no necesite código especializado para cada complemento.

Al utilizar contratos estándar, se promueve la interoperabilidad entre componentes complementarios, lo que facilita la adición y eliminación de componentes sin necesidad de modificar el sistema central de manera significativa.

1.2. Análisis crítico sobre la arquitectura. Ventajas y Desventajas

1.2.1. Ventajas

- **Particionamiento por dominio y técnico**: La arquitectura Microkernel permite una partición tanto por dominio como técnicamente, lo que la hace única. Esto es especialmente útil para aplicaciones que requieren diferentes configuraciones para cada ubicación o cliente, o para productos que enfatizan la personalización del usuario y la extensibilidad de funciones.
- **Capacidad de prueba, implementación y confiabilidad**: Tiene una capacidad de prueba, implementación y confiabilidad ligeramente superior al promedio, gracias a la aislación de la funcionalidad en componentes complementarios independientes. Esto reduce el alcance general de las pruebas de cambios y disminuye el riesgo de implementación, especialmente si los componentes complementarios se implementan en tiempo de ejecución.
- **Modularidad y extensibilidad**: Ofrece una modularidad y extensibilidad ligeramente superior al promedio. La funcionalidad adicional se puede agregar, eliminar y modificar a través de componentes complementarios independientes y autónomos, facilitando la extensión y mejora de las aplicaciones creadas con este estilo de arquitectura y permitiendo a los equipos responder a los cambios de manera más rápida.
- **Rendimiento**: Las aplicaciones Microkernel generalmente son pequeñas y no crecen tanto como la mayoría de las arquitecturas en capas, lo que contribuye a un mejor rendimiento. Además, no sufren tanto del antipatrón de sumidero de arquitectura y pueden simplificarse desconectando la funcionalidad innecesaria, lo que hace que la aplicación se ejecute más rápido.

1.2.2. Desventajas

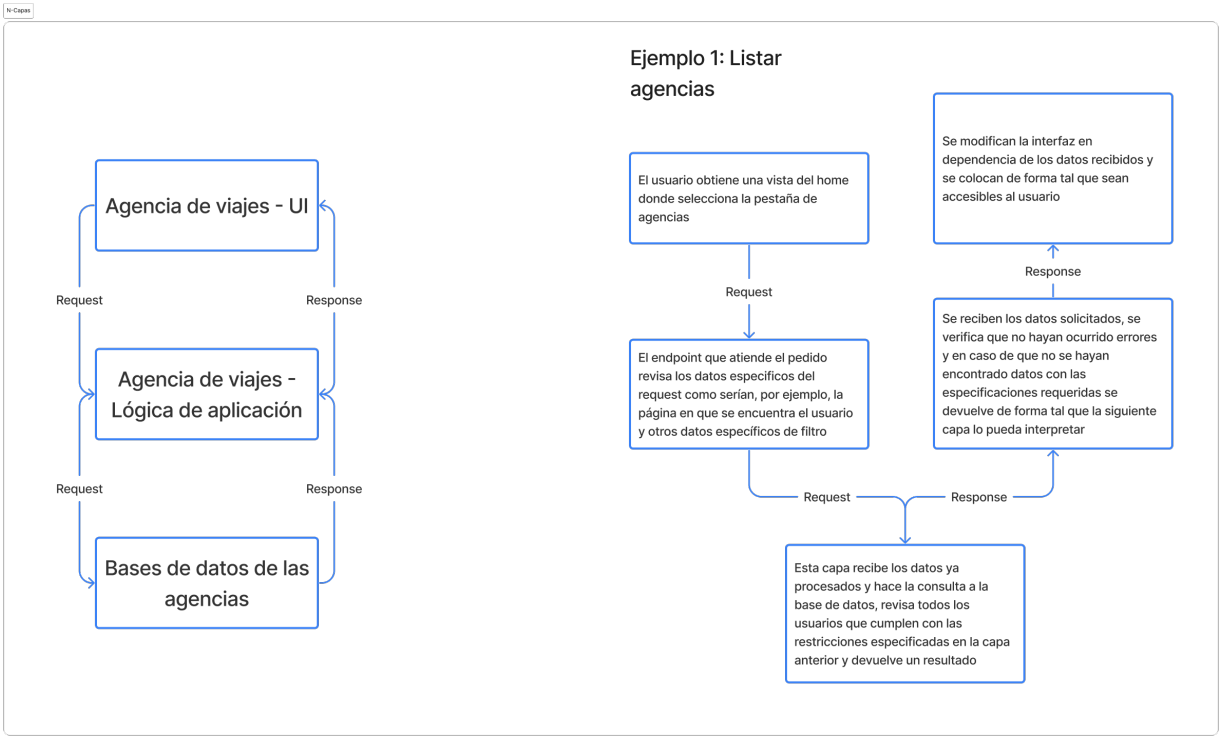
- **Escalabilidad**: Las aplicaciones basadas en Microkernel son generalmente desarrolladas para ser ejecutadas en modo Standalone. Si bien existen aplicaciones que implementan el estilo de Microkernel que son altamente escalables como algunos servidores de aplicaciones, la realidad es que este no es un estilo arquitectónico que se distinga por crear aplicaciones altamente escalables.
- **Alta complejidad**: Las aplicaciones basadas en Microkernel son difíciles de desarrollar, no solo por la habilidad técnica para soportar la agregación de funcionalidad adicional por medio de Plugins, si no que requiere un análisis muy elaborado para identificar hasta qué punto puede ser extendida la aplicación sin afectar la esencia del sistema Core.

1.3. ¿Es aplicable esta arquitectura al problema de la agencia de viajes?

La aplicación al problema en cuestión se haría compleja en su conjunto debido al vínculo que existe desde el planteamiento del problema con las numerosas interacciones que existen entre los distintos componentes, sin embargo, si podemos aplicarla a un área específica del problema, que sería la relacionada con la visualización de las estadísticas, en este ámbito podemos tener un módulo central encargado de manejar las solicitudes (core) y cada uno de los distintos análisis se pueden establecer como plugins, que serían todos los objetos que implementen la interfaz solicitada por el módulo estadístico, teniendo, por ejemplo, la cantidad de turistas que solicitan servicios de una agencia en un período de tiempo dado, el promedio de turistas que realizan excursiones en un período de tiempo dado, etc.

1.4. Propuesta de arquitectura para aplicar al problema

Para nuestro problema hemos decidido utilizar la arquitectura N-capas, basado en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división de los problemas a resolver.



En la capa de presentación (Agencia de viajes - UI) se muestra al usuario las diferentes opciones con las que puede interactuar, para ello estaremos utilizando como tecnología React.

En la capa de aplicación (Agencia de viajes - Lógica de aplicación) se estará realizando todo lo relacionado con el procesamiento de datos, como tecnología se estará utilizando FastAPI de python. En esta capa se realizan el procesamiento de información y las solicitudes a la capa de acceso a datos.

En la capa de acceso a datos (Base de datos de las agencias) se realizan las operaciones CRUD sobre los

datos, para esto estaremos utilizando PostgreSQL y SQLAlchemy

2. Planificación

Planificación del primer release:

- **Qué:** Como primera tarea vamos a estar desarrollando el módulo de autenticación de usuario, enfocado, en principio, a la creación de usuario, el inicio de sesión, la verificación de que el usuario esté autenticado, la eliminación de usuarios, el acceso a la información personal propia y la edición de usuarios.
- **Cómo:** En esta tarea, nuestro objetivo es implementar dos ventanas emergentes o modales que faciliten el registro y la autenticación en la aplicación. La información personal del usuario se alojará en una sección separada, a la que solo se podrá acceder una vez que el usuario esté autenticado. Nuestro enfoque principal será establecer un sistema de autenticación robusto que asegure la protección de los datos del usuario y de su contraseña. Para lograr esto, utilizaremos los módulos de autenticación OAuth2 de FastAPI y bibliotecas de cifrado como JWT. Es esencial que verifiquemos las credenciales del usuario antes de permitirle acceder a su información o gestionar su cuenta. Además, crearemos una tabla para almacenar los datos del turista y realizaremos operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en dicha tabla.
- **Cuándo:** Estas tareas deberán terminarse antes del día 5 de marzo
- **Quién:** La organización de las tareas es clara y directa: cada miembro del equipo se encargará de una operación específica relacionada con la base de datos, ya sea Crear, Leer, Actualizar o Eliminar. Posteriormente, el integrante será responsable de construir el componente correspondiente y de gestionar su interacción en la interfaz de usuario. Para facilitar el desarrollo paralelo y evitar conflictos, el esqueleto de la aplicación -incluyendo las rutas, las páginas y las tablas de la base de datos- ya ha sido establecido previamente. Esto asegura que todas las tareas son autónomas y pueden ser desarrolladas de manera independiente. Una vez que las tareas estén completas, el líder del equipo será el encargado de integrarlas en el proyecto principal, asegurando una fusión sin problemas.

Tareas:

- **Registrar usuario:**
  - crear modal para el registro del usuario a nivel visual
  - crear endpoint que reciba los datos del registro
  - verificar que los datos insertados cumplan con los requisitos para crear la cuenta
  - hashear la contraseña para garantizar la seguridad del usuario
  - agregar la cuenta a la base de datos
  - devolver al usuario información referente al estado de la operación realizada
- **Iniciar sesión:**
  - crear modal para el inicio de sesión
  - crear endpoint para el inicio de sesión
  - verificar los datos introducidos
  - solicitar información a la base de datos
  - comparar que el hash contraseña insertada sea igual al hash de la contraseña guardada

    - devolver error o un token de autenticación
    - almacenar el token de autenticación en el frontend
    - mostrar al usuario si se completó con éxito la autenticación o no
- **Solicitar información del usuario:**
    - crear interfaz con botón para acceder a la información personal
    - crear endpoint para acceder a la información personal
    - validar con token de autenticación
    - solicitar a la base de datos la información del usuario
    - mostrar al usuario los datos solicitados o un error en caso de que no esté autenticado
- **Editar usuario:**
    - crear modal para la edición de usuario
    - solicitar información del usuario
    - permitir al usuario modificar los datos
    - crear el endpoint para la edición de usuario
    - enviar nuevos datos al endpoint
    - modificar los datos a nivel de base de datos
    - mostrar al usuario si fue realizada satisfactoriamente la operación, y los datos actuales
- **Eliminar usuario:**
    - crear modal para la eliminación del usuario
    - crear endpoint para la eliminación del usuario
    - validar utilizando token de autenticación
    - eliminar usuario de la base de datos
    - invalidar token de autenticación
    - informar al usuario del resultado de la operación