Rodríguez, Gonzalo Leandro

SRT Hub Cloud Based Media Routing

2020

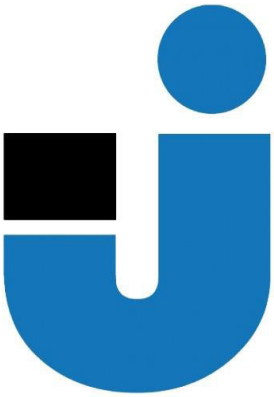
Instituto: Ingeniería y Agronomía

Carrera: Ingeniería en Informática

Universidad Nacional
Arturo Jauretche

Instituto de Ingeniería y Agronomía

Ingeniería en Informática



TRABAJO FINAL DE LA PRACTICA PROFISIONAL SUPERVISADA

SRT Hub Cloud Based Media Routing

Estudiante:

Rodríguez Gonzalo Leandro

Tutores:

Prof. Lía Lavigna

Dr. Ing. Martin Morales

Ing. Javier Rebagliatti

2020-12-11

PRÁCTICA PROFESIONAL SUPERVISADA (PPS)
SRT Hub Cloud Based Media Routing

DATOS DEL ESTUDIANTE

Apellido y Nombres: Rodríguez Gonzalo Leandro

DNI: 39115529

N.º de Legajo: 14418

Correo electrónico: pgonzalorodriguez@gmail.com

Cantidad de materias aprobadas al comienzo de la PPS: 42 (cuarenta y dos)

PPS enmarcada en artículo (4 o 7) de la Resolución (CS) 103/16: PPS enmarcada en el artículo 7. El estudiante se encuentra en el ejercicio regular y permanente de un empleo formal con encuadramiento temático propio de la carrera en la entidad para la que solicita la excepción.

DOCENTE SUPERVISOR

Apellido y Nombres: Dr. Ing. Martin Morales

Correo electrónico: martin.unaj@gmail.com

DOCENTE TUTOR DEL TALLER DE APOYO PARA LA PRODUCCIÓN DE TEXTOS ACADÉMICOS DE LA UNAJ

Apellido y Nombres: Prof. Lía Lavigna

Correo electrónico: lialavigna@gmail.com

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA LA PPS

Nombre o Razón Social: SOUTHWORKS S.R.L

Dirección: Perú 375 4º, (C1067AAG), Ciudad de Buenos Aires

Teléfono: 5411-4331-4331

Sector: Desarrollo de software

TUTOR DE LA ORGANIZACIONAL

Apellido y Nombres: Ing. Rebagliatti Javier

Correo electrónico: javier.rebagliatti@southworks.com

FIRMA DEL COORDINADOR DE LA CARRERA

Imágenes y gráficos

Introducción

Presentación

Haivision SRT Hub es un servicio que funciona a través de la nube para enrutar medios de baja latencia, seguros y confiables (SRT) para los flujos de trabajo de contribución, producción y distribución. Esta aplicación será orientada para las emisoras que buscan alternativas a los costosos enlaces satelitales, redes de fibra o soluciones de transporte patentadas. Además, les permitirá transmitir en vivo desde cualquier lugar del mundo.

SRT Hub es adaptable a las necesidades específicas del flujo de trabajo. Basado en un marco abierto y documentado, los usuarios pueden crear Hublets (concepto que se desarrollará con posterioridad) para proporcionar traducción de medios de entrada y salida, alineación de metadatos, acceso de usuarios y control. Con los Hublets, las emisoras pueden conectar múltiples proveedores de soluciones en un flujo de trabajo unificado.

Como se mencionó anteriormente, SRT Hub utiliza el protocolo SRT (Secure Reliable Transport Protocol) para enviar el contenido a través de Internet utilizando la columna vertebral de Microsoft Azure. La función de cada una de las siglas SRT son las siguientes:

- Secure: encripta el contenido que será enviado por la transmisión.
- Reliable: recupera todos los paquetes perdidos desde el servidor.
- Transport: se adapta constantemente a las condiciones cambiantes de red.

A continuación, se observará en altos niveles cómo se plantea la arquitectura de SRT Hub (figura #1).

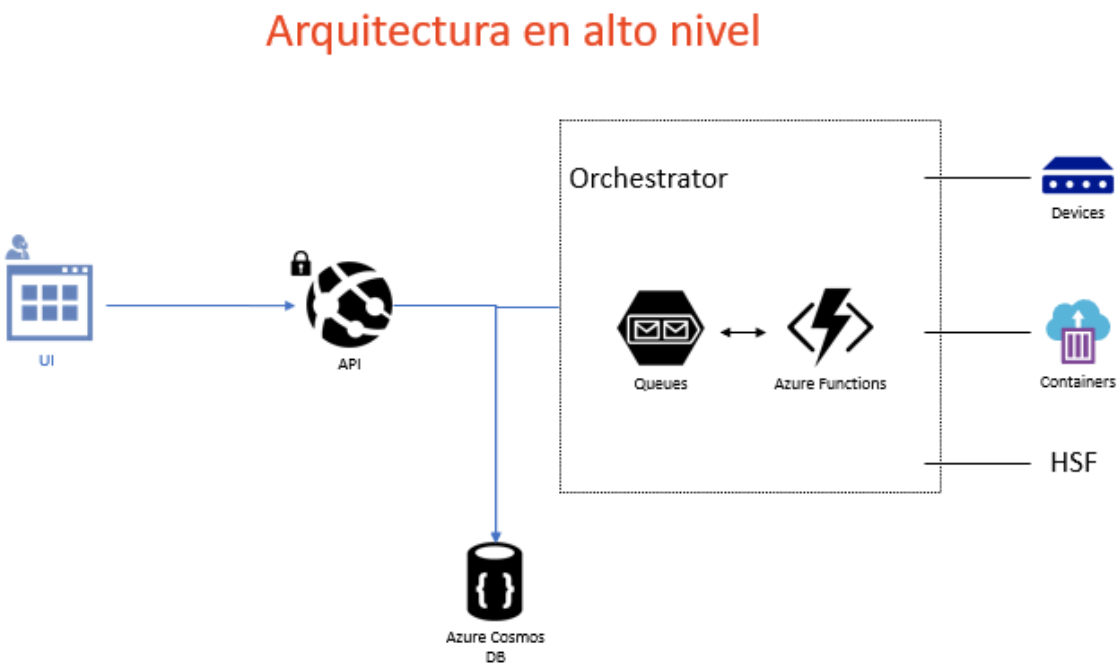


Figura #1. *Arquitectura básica y servicios esenciales.*
Fuente: Elaboración propia, basada en la práctica.

SRT Hub posee diversos componentes que serán desarrollados en este apartado, y pueden resumirse en los siguientes:

- UI (Vue JS SPA).
- API (ASP.NET Core).
- Base de datos (CosmosDB: Servicio de base de datos multimodal, SQL).
- Orquestador (Motor del núcleo del sistema de enrutamiento de SRT Hub).
    - Define cuándo, qué y cómo se manejan los recursos.
    - Abastece y/o gestiona recursos (contenedores, dispositivos, fábricas de servicio de Hublets externos).

En la figura #1 se observa el concepto de “Containers”, este concepto hace referencia a un servicio de Microsoft Azure, el cual se utiliza para darle vida a los llamados “Hublets”.

Como servicio, Azure Container Instancies ofrece la forma más rápida y sencilla de ejecutar un contenedor en Azure, sin tener que gestionar ninguna

máquina virtual y sin tener que adoptar un servicio de nivel superior. Además, permite exponer grupos de contenedores directamente a Internet con una dirección IP y un nombre de dominio (FQDN). Al crear una instancia de contenedor, se podrá especificar una etiqueta de nombre de DNS personalizada la que será de utilidad para el objetivo principal. La figura #2 indica el funcionamiento principal de Azure Container Instances.

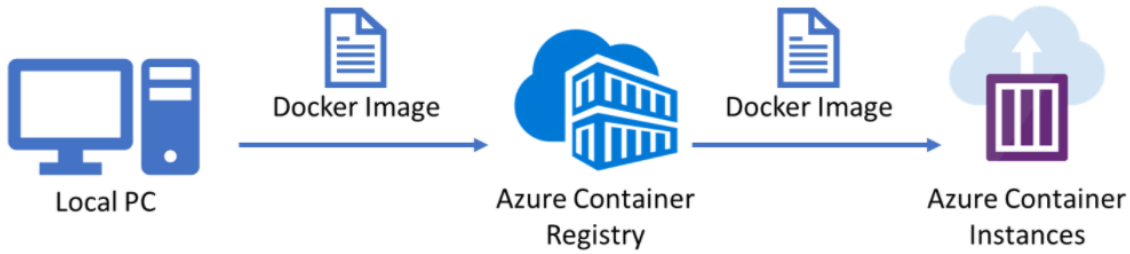


***Figura #2**. Funcionamiento principal de Azure Container Instances.*
Fuente: Recuperado de https://samcogan.com/windows-containers-and-azure/ (2019)

Con motivo de entender mejor los conceptos principales, se presentarán de manera más detallada las ideas mencionadas con anterioridad y, además, se incorporarán nuevos conceptos tales como:

- Ruta: una ruta se crea a partir de dos o más Hublets.
- Hublet Definition: son “plantillas” utilizadas para la configuración de cualquier tipo de Hublet en el uso de la aplicación.

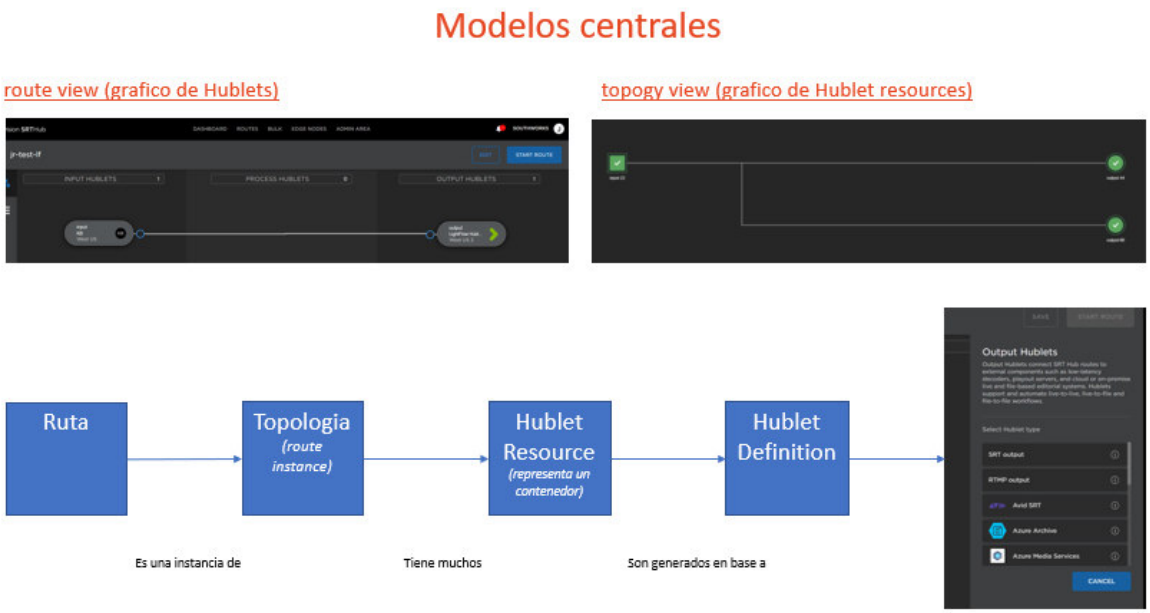


***Figura #3**. Creación de ruta y conceptos clave.*
Fuente: Elaboración propia, basada en la práctica.

Al momento de iniciar esta ruta, como se muestra en la figura #3, se construirá una topología de los recursos utilizados por los Hublets, es decir, se podrá apreciar la cantidad de contenedores que se inician por cada Hublet.

Además, muchos servicios se pondrán en marcha utilizando el orquestador, lo que se explicará en profundidad más adelante.

Propuesta

La problemática general que intenta resolver SRT Hub es la de eliminar la complejidad de hacer deambular los diferentes recursos en nubes regionales y determinar el camino más viable a través de Internet. A tal fin, se espera utilizar la última arquitectura de nubes de Microsoft Azure.

Por consiguiente, SRT Hub provee automática e inteligentemente los recursos de procesamiento requeridos en cualquier centro de datos Azure y, de este modo, escalar y enrutar inteligentemente los medios de comunicación desde la fuente hasta el destino.

Por ende, esta aplicación será orientada para las empresas o personas que buscan alternativas a los costosos enlaces satelitales, redes de fibra o soluciones de transporte patentadas. Se propondrá desarrollar una aplicación junto a un equipo de desarrolladores para implementar la integración de todos los servicios descriptos con anterioridad. Además, utilizando las últimas tecnologías disponibles del mercado se buscará competir con otras empresas cuyos productos y funcionabilidades son idénticas a SRT Hub.

Por otra parte, se buscará desarrollar novedosas pantallas WEB obteniendo una experiencia de usuario única y llamativa para cualquier tipo de persona interesada en utilizarlo.

Objetivos

Los objetivos específicos que se pretenden alcanzar en la PPS serán los siguientes:

- Presentar y detallar la arquitectura.
- Explicar y exhibir la implementación y servicios.
- Detallar principalmente el desarrollo realizado en el área de Front-End.
- Exponer el funcionamiento básico de la aplicación.

Tareas a ejecutar

Las tareas que se pretenden ejecutar en la PPS serán las siguientes:

- Explicar y desarrollar la arquitectura.
    - Explicar los conceptos claves
    - Exponer la arquitectura en alto nivel.
    - Modelar el esquema principal de la aplicación.
    - Detallar los servicios utilizados.
    - Introducir los conceptos claves del sistema de notificaciones.
- Exponer el modelo utilizado en el Front-End:
    - Detallar la arquitectura.
    - Comentar las tecnologías utilizadas.
    - Explicar la configuración de Hublets.
    - Mostrar los frameworks utilizados.

    - Presentar el sistema de manejador de errores.
    - Exhibir el manejo de telemetría de los contenedores.
- Desarrollar el concepto de telemetría:
    - Explicar el concepto clave.
    - Exponer la salud de los contenedores.
    - Detallar el manejo de los estados.
- Exponer Hublets externos:
    - Desarrollar y presentar la creación.
    - Exponer el funcionamiento.

Desarrollo

Definición y desarrollo de la arquitectura

Tecnologías utilizadas

SRT Hub contará con su propia API (Application programming interface) y las tecnologías que se utilizarán para la implementación del lado de back-end, serán las siguientes:

- **API:** ASP.NET Core versión 4.7.2.
- **Orquestador:** motor del núcleo del sistema de enrutamiento de SRT Hub. Define cuándo, qué y cómo abastece o gestiona recursos (contenedores, dispositivos, etc.)
- **Base de datos:** se utilizará CosmosDB el cual es un servicio de base de datos multi-modelo.

Modelación en alto nivel

A continuación, en la figura #4, se muestra un diagrama de la modelación de la arquitectura en alto nivel:

.

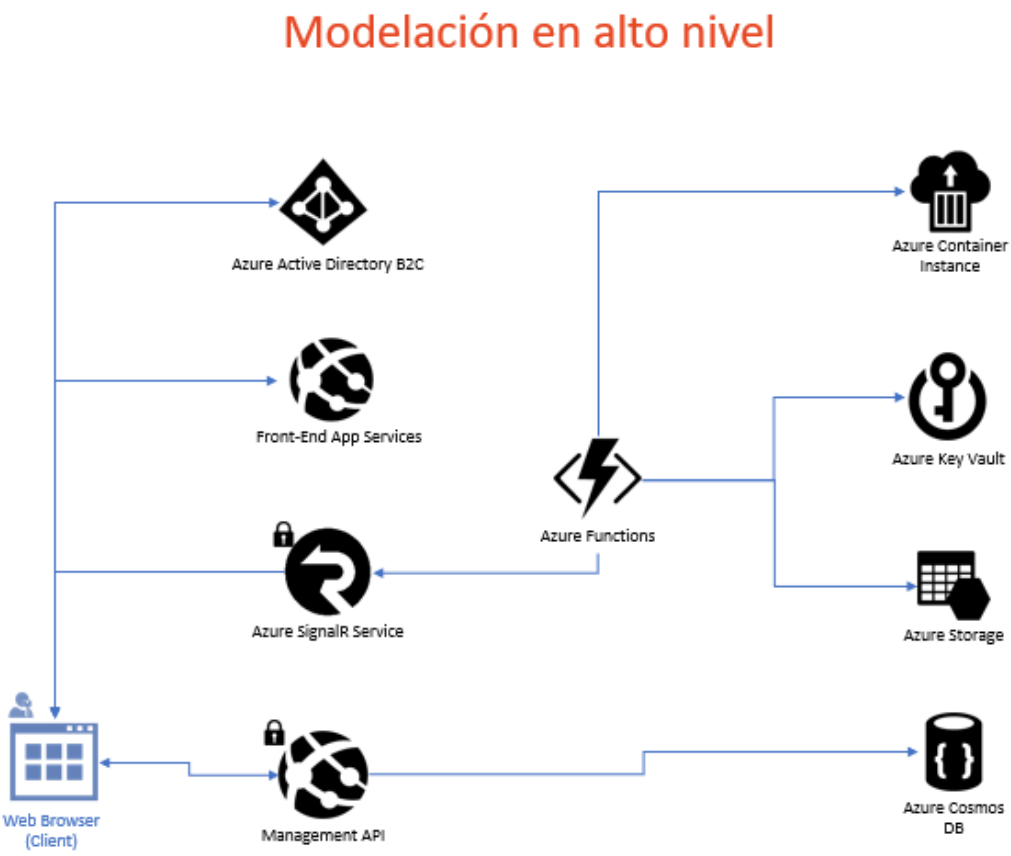


***Figura #4**. Modelación en alto nivel.*
Fuente: Elaboración propia, basada en la práctica.

Servicios utilizados

Los servicios que serán utilizados por SRT Hub se expresan a continuación:

- Directorio Activo de Azure B2B.
- Registro de contenedores de Azure.
- Manejador de API.
- Servicio de SignalR de Azure.
- App secrets.
- Azure Cosmos DB.
- Orquestador.
- Azure Key Vault.
- Azure IoT Hub.

- Azure Event Hub.
- App Insights.
- Azure Storage.
- Telemetry Handling.

Con motivo de desarrollar mejor los conceptos, a continuación, se detallará cada servicio mencionado:

Directorio Activo de Azure B2B: es una característica dentro de las Identidades Externas que permite invitar a usuarios a colaborar con la organización creada. Con la colaboración B2B, se podrá compartir de forma segura las aplicaciones y servicios de la empresa con usuarios invitados de cualquier otra organización, mientras se mantendrá el control sobre los propios datos corporativos.

Registro de contenedores de Azure: permitirá construir, almacenar, asegurar, escanear y gestionar imágenes y artefactos de contenedores con una instancia de distribución de OCI (Open Container Initiative) totalmente gestionada y geo-replicada. Además, facilitará la conexión entre entornos.

Manejador de API: aplicación que se desarrollará en ASP.NET Core en la que se crearán todos los puntos finales.

Servicio de SignalR de Azure: con este servicio se realizará la incorporación de comunicación en tiempo real a una aplicación web. Es decir, se utilizará para enviar notificaciones desde el back-end y mostrarlas en tiempo real desde el front-end.

App secrets: se utilizará para almacenar y recuperar información sensible de la API como, por ejemplo, claves de producción de alto riesgo.

Azure Cosmos DB: es un servicio de base de datos NoSQL totalmente administrado para el desarrollo de aplicaciones modernas.

Orquestador: será el núcleo principal de la aplicación, allí se alojarán las funciones de Azure las cuales realizan todos los procesos para levantar/detener contenedores.

Azure Key Vault: aumenta la seguridad y el control sobre claves y contraseñas. Además, se utilizará para reducir la latencia con la escala de nubes y la redundancia global.

Azure IoT Hub: proporciona comunicaciones bidireccionales, se podrá conectar cualquier dispositivo a este servicio y así asegurar una comunicación segura y confiable.

Azure Event Hub: es un servicio de ingesta de datos en tiempo real. Se podrán realizar streaming de millones de eventos por segundo desde cualquier origen para crear canalizaciones de datos dinámicos.

App Insights: se utilizará para monitorear la aplicación en tiempo real. Detecta automáticamente las anomalías de rendimiento e incluye potentes herramientas analíticas para ayudar a diagnosticar los problemas y a comprender lo que los usuarios hacen realmente con la aplicación.

Azure Storage: almacenamiento en la nube seguro. Permite almacenar datos sin la necesidad de preocuparse por un servidor.

Telemetry Handling: manejador de telemetría de los contenedores es un servicio que permite conocer la salud de cada contenedor y así tomar esa información e indicarla en la aplicación.

Concepto de Hublets y su arquitectura

Definición general

Un Hublet es un componente de una ruta de transmisión en vivo de SRT Hub. Hay tres tipos de Hublets, dependiendo de la función que el Hublet realiza en la ruta:

- **Input Hublets:** son el punto de entrada de las rutas y, generalmente, reciben la transmisión de un codificador. Adicionalmente, pueden manejar uno de los codificadores administrados y soportados (SRT Connector, KB, etc.)
- **Output Hublets**: son el borde externo de la ruta, y pueden o no tener un dispositivo administrado, o un servicio externo asociado. Por ejemplo, un servicio de Video Streaming (MS Stream, AMS, Youtube), un decodificador administrado (Decoder, STB, IP TV), o cualquier otro servicio externo (como Azure Storage)
- **Processing Hublets**: se utilizarán para hacer transformaciones en el contenido, como la transcodificación, el sub-corte, el compostaje, etc.

En consecuencia, se detallará una lista con los tipos de Hublet que la aplicación SRT Hub podría soportar:

- Hublets de entrada:
  - Entrada SRT genérica
  - Entrada de dispositivos administrados
- Hublets de salida:
  - Salida genérica de RTMP.
  - Salida genérica de TS/UDP.
  - Salida de SRT genérica.
  - Salida de Microsoft Stream.
  - Salida de los Servicios de Medios de Comunicación de Azure.
  - Salida Blob Storage.
  - Salida Stream Analysis.
- Procesamiento de Hublets:
  - Transcodificación.
  - Composición.
  - Superposición.

De esta manera, cada Hublet contiene un conjunto de conectores, que se utilizan para unir los Hublets entre sí. Mientras que los Hublets de procesamiento tendrán dos tipos diferentes de conectores (conectores de entrada y salida), los Hublets de entrada y salida tendrán sólo uno.

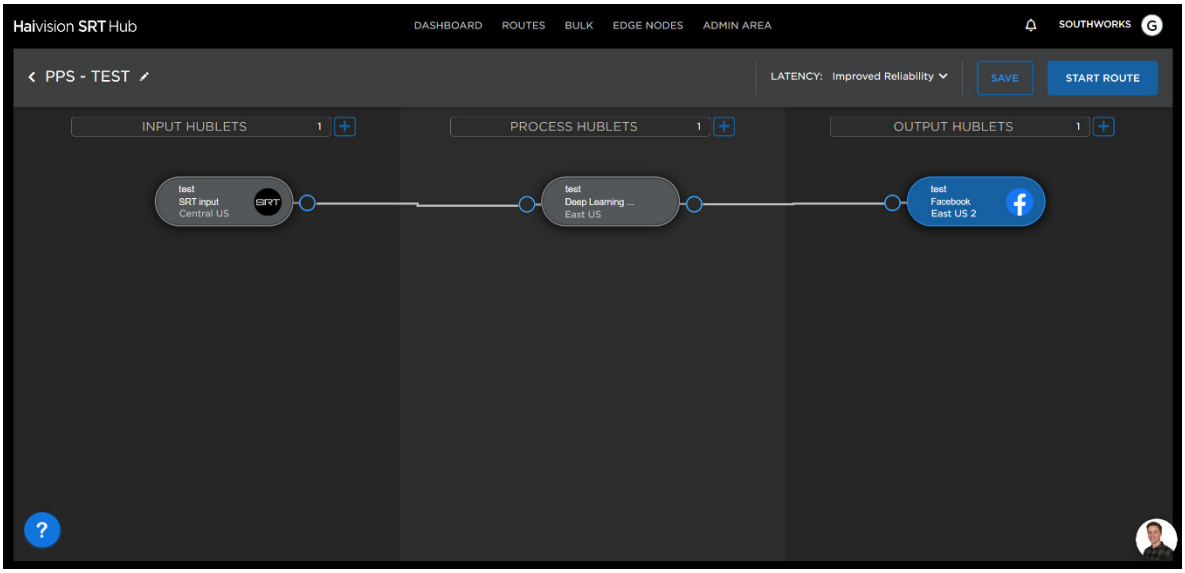


***Figura #5**. Simple representación de Hublets.*
Fuente: Elaboración propia, basada en la práctica.

Los enlaces entre Hublets representan el flujo de la transmisión de SRT, desde la(s) entrada(s) a la(s) salida(s). La Figura #5 es un ejemplo de una configuración de ruta simple.

Especificaciones

Los Hublets se definen por un conjunto de propiedades:

- **Nombre:** será utilizado para identificar al Hublet.
- **Descripción:** contendrá una breve explicación de cuál es el propósito del Hublet. Este campo también debe ser localizado en los diferentes idiomas soportados por la aplicación.
- **Tipo:** los tipos disponibles serán:
  - Input (entrada).
  - Output (salida).
  - Processing (procesador).
- **Metadatos requeridos:** son los campos que el Hublet requiere para trabajar, que se clasifican según sean proporcionados por el usuario o por el sistema. Aunque cada uno tiene su propio conjunto, algunas propiedades son comunes a todos los Hublets:

- Parámetros: esquema JSON que contiene cualquier parámetro que el Hublet necesite para funcionar. Estos serán preguntados en la interfaz de usuario de SRT Hub.
- Estilos:
  - Logotipo: Imagen PNG de un logo.
  - Color: Color primario en Hexadecimal. Se pretende que represente al socio que desarrolló el Hublet.
- Nombre: un nombre para identificar el componente en la ruta.
- Región (sólo para los Hublets de entrada y salida): Es el Centro de Datos de Azure donde el Hublet será desplegado.

- **Mostrar los metadatos:** son los campos que el Hublet proveerá al usuario, a través de la interfaz de usuario, una vez que sea provisto.
- **Especificación de entrada (sólo para los tipos de Hublet de Procesamiento y Salida):** incluye los siguientes campos:
  - Número de entradas: es el número de entradas soportadas por el Hublet. Si este número es mayor que el número de entradas soportadas por el contenedor, el orquestador debe girar un contenedor para cada conexión de entrada.
- **Especificación del contenedor:** contiene los datos necesarios para hacer girar el contenedor asociado al Hublet, incluidas las siguientes propiedades:
  - Especificación de entrada: incluye los siguientes campos:
    - Número de entradas: el número de entradas que soporta el contenedor debe ser igual al número de entradas admitidas por el Hublet, o 1.
    - Modos SRT soportados: Caller o Listener.
  - Especificación de salida (sólo para los tipos de Hublet de procesamiento y de entrada): incluye los siguientes campos:
    - Número de salidas que soporta el contenedor.
    - Modos SRT soportados: Caller o Listener.

  - Servidor de imágenes: remite al dominio del Registro de Contenedores de Azure.
  - Nombre de la imagen: corresponde al nombre de una imagen ya registrada.
  - Versión de la imagen: incluye la versión de la imagen que contiene un identificador único para el manejo interno del contendor.
  - Nombre de usuario: nombre de usuario para acceder al contenedor.
  - Contraseña: keyvault donde se almacena la contraseña para acceder al contenedor.
- **Colas de configuración personalizadas**: hace referencia al nombre de las colas utilizadas para realizar la lógica personalizada.
- **Tipos de codificador admitidos:** se encuentras sólo disponible para Hublets de entrada y salida, contiene la lista de tipos de codificador admitidos por el Hublet (SRTConnector, KB, Makito, STB, etc.).

El único protocolo de transporte soportado entre los Hublets es el SRT, por lo que todos los Hublets deben tener una implementación de ese protocolo, y deben recibir y devolver los flujos en el protocolo SRT.

Además, el modo SRT a utilizar a cada lado de un enlace entre los Hublets será determinado automáticamente por la aplicación, basándose en los modos SRT soportados por los Hublets participantes, utilizando las siguientes reglas:

- Si se dispone de más de una configuración (si ambas partes soportan los modos de Caller y Listener), se preferirá el modo de Listener-Caller, en el que el Caller es la parte más cercana a la fuente y el Listener es la parte más cercana al destino.
- Si el enlace une dos conexiones incompatibles (ambas partes sólo soportan los modos de Listener o Caller), la aplicación lanzará una excepción, impidiendo que se guarde la ruta.

En definitiva, los Hublets de entrada y salida pueden o no tener asociado un dispositivo administrado. La lógica para gestionar los dispositivos, será proporcionada por la propia aplicación (no es necesario escribir la lógica para manejar los dispositivos en los Hublets). Se admitirán nuevos tipos de dispositivos a medida que cumplan las especificaciones de los dispositivos administrados.

Especificaciones del contenedor

Cada Hublet podrá levantar una imagen de un Docker en Azure Container Instancies para cada ejecución, es decir, se permitirá a los Hublets realizar cuatro operaciones de preparación, antes y después de aprovisionar el contenedor, y antes y después de des-provisionar el contenedor.

Por lo tanto, la información devuelta por la función de configuración previa al aprovisionamiento se pasará al contenedor como parte de la carga útil de configuración. Toda la información relacionada con el contenedor (parámetros públicos de IP, SRT, etc.) creados se pasarán a la función de configuración posterior al aprovisionamiento.

Para reducir la dependencia y la disponibilidad entre los componentes, la comunicación entre el orquestador SRT Hub y los procesos de configuración personalizada el motor SRT Hub utilizará el procesamiento en cola, lo que significa que el orquestador realizará una solicitud a la función de configuración personalizada poniendo un mensaje en la cola y esperará una respuesta en la cola de respuestas de la función de configuración personalizada. La Figura #6 muestra el procesamiento en cola del orquestador.

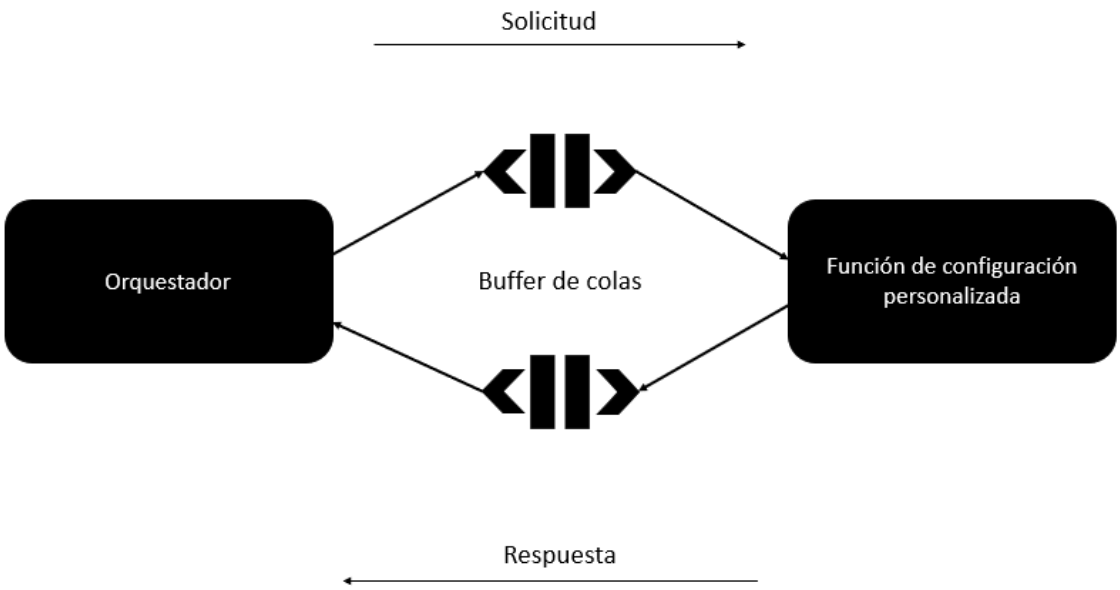


***Figura #6**. Procesamiento en cola del orquestador.*
Fuente: Elaboración propia, basada en la práctica.

En el marco de la integración, el Hublet debe proporcionar el nombre de las colas que escuchará, para cada una de las operaciones que soporta (preinstalación de provisión, post-instalación de provisión, preinstalación de desprovisión, postinstalación de desprovisión). Para esto habrá una única y conocida cola para enviar todas las respuestas. Cada una de estas respuestas debe incluir el identificador de la operación, que es proporcionado por la orquestación en el mensaje de solicitud.

Por razones de seguridad, la cuenta, las claves, las frases de contraseña, etc. no se pasan a las funciones. En su lugar, se almacenan en la bóveda de claves, y sólo se pasa la clave de la bóveda. Debido a esto es responsabilidad de la función recuperar la bóveda del servicio de Key Vault.

Definición y desarrollo del front-end

Tecnologías y frameworks utilizados

Actualmente se utilizan las siguientes tecnologías:

- Vue js & Vuex.

- Typescript.
- Bootstrap & SCSS.

A continuación, se detallará un breve resumen de cada una de ellas:

- **Vue js:** es un framework progresivo para la construcción de interfaces de usuario. A diferencia de otros frameworks monolíticos, Vue está diseñado desde el principio para ser adaptable de forma incremental. La biblioteca central se enfoca sólo en la capa de vista y es fácil de recoger e integrar con otras bibliotecas o proyectos existentes. En la Figura #7 se ejemplifica un componente de Vue JS utilizando HTML 5.

```
    <div class="d-flex justify-content-end last-items">
        <div class="navbar-mobile my-auto">
            <div class="text-white my-auto box-bars cursor-pointer px-3 py-2" @click="toggleCollapse()">
                <i class="fa fa-bars"></i>
            </div>
            <div class="topnav text-white">
                <div v-show="showTabsMobile">
                    <li v-for="tab in featureFlaggedTabs" :key="tab.key" class="nav-item"
                        :class="{'font-active active' : isActive(tab.key)}">
                        <a class="text-uppercase navigation-text nav-link"
                            :class="{'font-active' : isActive(tab.key)}"
                            @click="navigate(tab.key)">{{ $t(tab.key) }}</a>
                    </li>
                </div>
            </div>
        </div>
        <NotificationsBell />
        <div id="home-button-user" @mousedown.prevent="toggleMenu()" class="box-user-name py-1 my-auto d-flex flex-row px-3">
            <h6 class="my-auto username-button">{{ subscription }}</h6>
            <div class="form-inline">
                <div class="pl-3">
                    <UserLogo />
                </div>
            </div>
        </div>
    </div>
    <UserMenu :display.sync="display" />
</header>
```

***Figura #7**. Ejemplo de código utilizando Vue js.*
Fuente: Elaboración propia, basada en la práctica.

- **Vuex:** es un patrón de gestión de estado y biblioteca para aplicaciones Vue.js. Su función es la de servir como un almacén centralizado para todos los componentes de una aplicación, con reglas que aseguran que el estado sólo puede ser mutado de manera predecible. También, se integra con la extensión oficial de devtools de Vue para proporcionar características avanzadas como la depuración en el tiempo de configuración cero y la exportación/importación de estado instantáneas. La Figura #8 muestra la arquitectura principal de Vuex, mientras que la Figura #9 muestra un ejemplo de su utilización en la práctica.

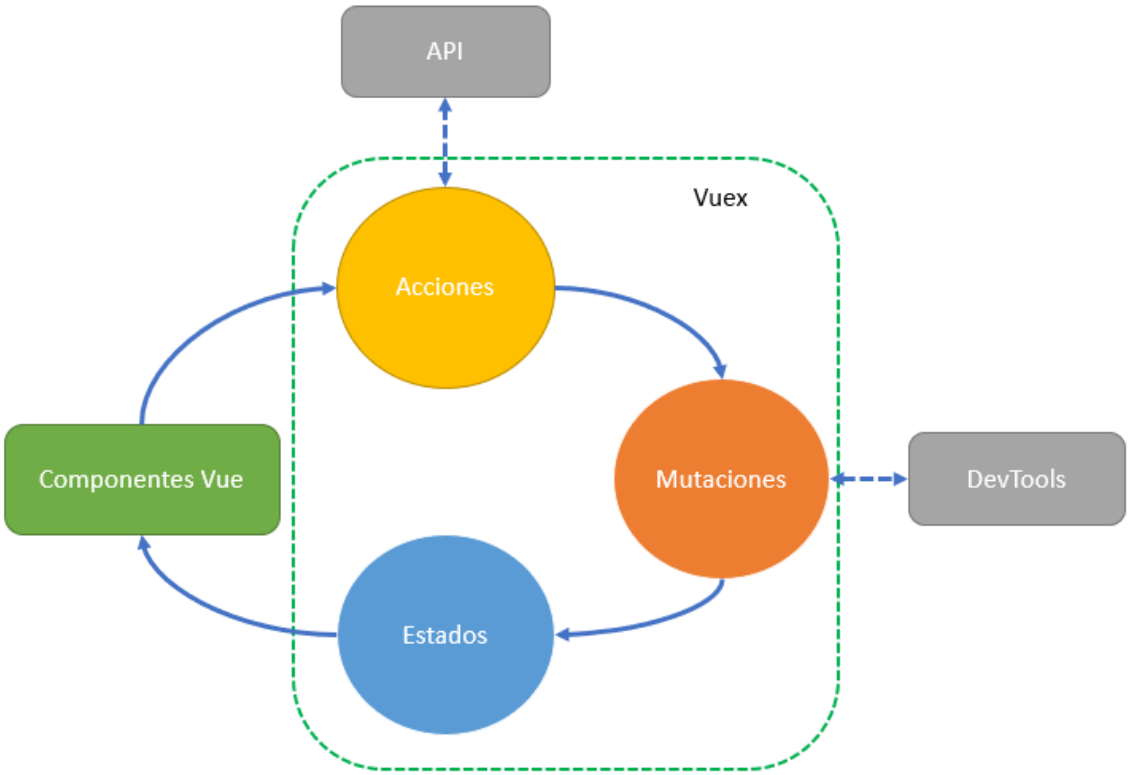


***Figura #8**. Concepto principal de Vuex.*
Fuente: Elaboración propia, basada en la práctica.

```
export default function actionsFactory(
    api: UserService,
): ActionTree<UserState, RootState> {
    return {
        Complexity is 4 Everything is cool!
        [actionTypes.CLOUD_LOGIN](context) {
            const promise = api.cloudLogin();
            promise.then((user) => {
                context.commit(mutationTypes.SAVE_USER, user);
                configOrganizationData(context.state.organizationData);
                setUserRoleData(context.state.user!.preferredSubscription.role);
                return user;
            });
            return promise;
        },
        Complexity is 3 Everything is cool!
        [actionTypes.UPDATE_USER](context, updatedUser: UpdateUserRequest) {
            const promise = api.updateUser(updatedUser);
            promise.then((user: User) => {
                context.commit(mutationTypes.SAVE_USER, user);
                configOrganizationData(context.state.organizationData);
                setUserRoleData(context.state.user!.preferredSubscription.role);
            });
            return promise;
```

***Figura #9**. Ejemplo de código utilizando Vuex.*
Fuente: Elaboración propia, basada en la práctica.

- **Typescript:** es un superconjunto de JavaScript que proporciona principalmente escritura estática opcional, clases e interfaces. Uno de los grandes beneficios es permitir que los IDEs (Integrated Development Environment) proporcionen un entorno más utilizable para detectar errores comunes mientras se escribe el código.

Para un proyecto de magnitud de JavaScript, la adopción de TypeScript podría resultar de un software más robusto y útil. La Figura #10 muestra la sintaxis de typescript en la práctica.

```
import {Component, Vue, Watch} from 'vue-property-decorator';  71.4K (gzipped: 25.2K)
import DashboardMap from '@/modules/dashboard/components/DashboardMap/DashboardMap';
import StatisticsWrapper from '@/common/components/StatisticsWrapper/StatisticsWrapper';
import RoutesNavBar from '@/common/components/RoutesNavBar/RoutesNavBar';
import {PagedResult, HubletRoute} from 'srthub-api';  19.7K (gzipped: 6.3K)
import {namespace} from 'vuex-class';
import {name as routeModuleName} from '@/modules/routes/store/index';
import {actionTypes as routesActionType} from '@/modules/routes/store/routes.types';
import Notifications from '@/common/utilities/notifications/notifications';

const routesModule = namespace(routeModuleName);

Fernando Martin Besteiro, 8 days ago | 8 authors (Julian Schiffer and others) | Complexity is 6 It's time to do something...
@Component({
    components: {
        DashboardMap,
        StatisticsWrapper,
        RoutesNavBar,
    },
})
export default class Home extends Vue {
    @routesModule.Getter('getActiveRoutes')
    public activeRoutes!: HubletRoute[];

    @routesModule.Action(routesActionType.GET_ROUTES)
    public getRoutes!: () => Promise<PagedResult<HubletRoute>>;

    public display: boolean = true;
    public displayRegion: boolean = false;

    public totalActiveRoutes: number = 5;
    public activeFilteredRoutes: HubletRoute[] = [];
    public search: string = '';
    private toastNotifications: Notifications = new Notifications();
```

***Figura #10**. Ejemplo de código utilizando typescript.*
Fuente: Elaboración propia, basada en la práctica.

- **SCSS (Sassy CSS):** es una extensión de la sintaxis de CSS (Cascading Style Sheets). Esto significa que cada hoja de estilo CSS es un archivo SCSS válido con el mismo significado. Esta sintaxis se mejora con las características de Sass (Syntactically Awesome Stylesheets). Es por ello que los archivos que usan esta sintaxis tienen la extensión SCSS. La Figura #11 muestra un ejemplo de código SCSS aplicado en la práctica.

```scss
.unauthorized-page {
    height: 100%;
    background: linear-gradient(180deg, $main-blue 50%, $main-white 50%);

    .card {
        width: 33%;
        min-height: auto;
        border: 1px solid $main-white;
        margin: auto;
        padding: 35px;
        background: $main-white;
        box-shadow: 0px 5px 9px 3px rgba(0, 0, 0, 0.3);
        border-radius: 0;

        @media screen and (max-width:1024px) {
            min-width: 50%;
        }

        @media screen and (max-width:445px) {
            min-width: 100%;
        }

        .card-body {
            a:link {
                text-decoration: none;
                color: $main-sky-blue;
                align-content: center;
                display: grid;
            }

            .btn.btn-sx {
                min-height: 40px;
                min-width: 50%;
```

Figura #11. *Ejemplo de código utilizando scss.*
Fuente: Elaboración propia, basada en la práctica.

Framework principal

La pantalla principal de SRT Hub se basará en una creación de ruta (concepto desarrollado anteriormente), lo que requiere que esta pantalla sea visualmente llamativa y poderosa.

Para unir ambos contenedores y permitir el desplazamiento de los Hublets en la pantalla principal se utilizará un framework denominado Rete.

Además, Rete es un framework para la programación visual y permite crear un editor basado en nodos directamente en el navegador. Se podrán definir nodos y trabajadores que permitirá crear instrucciones para procesar datos en el editor sin una sola línea de código.

Algunas ventajas:

- El framework no está vinculado a ningún dominio, sino que sólo visualiza y procesa el editor de nodos.
- La arquitectura basada en eventos permite añadir nuevas funcionalidades en forma de plugins.
- Es libre de elegir los componentes necesarios. Se conectarán los plugins que mejor se adapten al propósito, como se representa en la Figura #12.

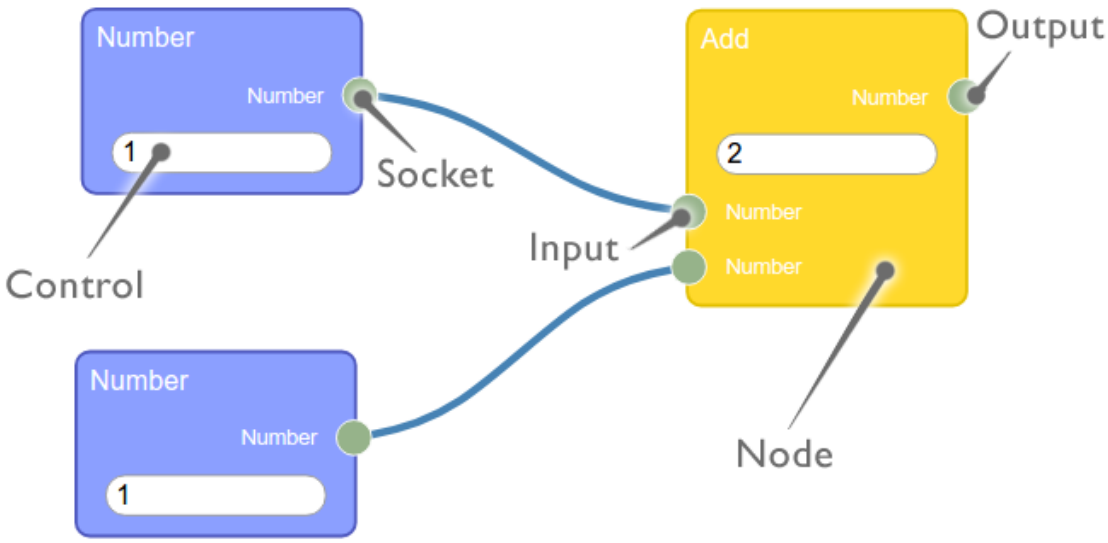


***Figura #12**. Rete framework, concepto principal.*
Fuente: Recuperado de https://rete.js.org/#/docs (2016)

¿Cómo funciona con SRT Hub?

Basándose en las definiciones de los Hublets de la base de datos, las diferentes píldoras se representan en Hublets de entrada, proceso y salida. Básicamente, se detecta cuando se crea un nodo y se conecta a otro. Por lo tanto, dependiendo del tipo de nodo y sus propiedades, se configuran los datos y estilos necesarios. La figura #13 registra la utilización del framework.

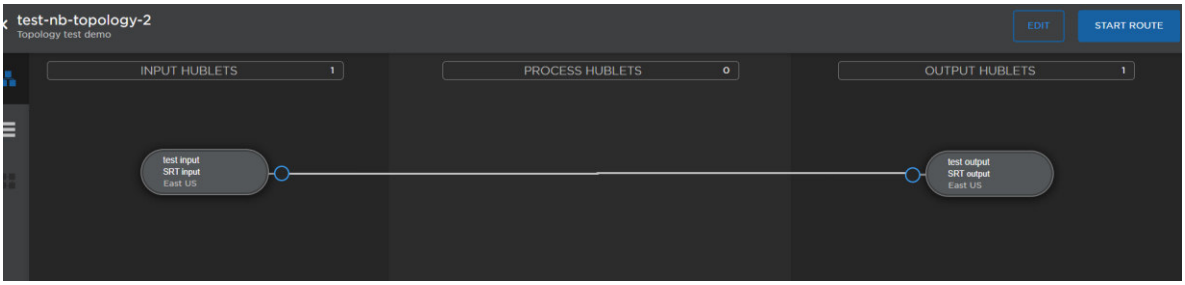


***Figura #13**. Rete framework en SRT Hub.*
Fuente: Elaboración propia, basada en la práctica.

Detalle de la arquitectura

La arquitectura del lado del front-end de la aplicación será de tipo SPA (Single Page Application), es decir, que se engloba en una sola página con el propósito de dar una experiencia más fluida a los usuarios, como si se tratara de una aplicación de escritorio.

En un SPA todos los códigos de HTML, JavaScript y CSS se cargan una sola vez y los recursos necesarios se llamarán dinámicamente a medida que lo requiera la página, normalmente como respuesta a las acciones del usuario. La página no tiene que cargarse de nuevo en ningún punto del proceso y tampoco es necesario transferir a otra página, se permite la navegabilidad en páginas lógicas dentro de la aplicación.

A continuación, en la Figura #14 se observarán las diferencias entre un ciclo de vida tradicional de una página web con una aplicación de página única.

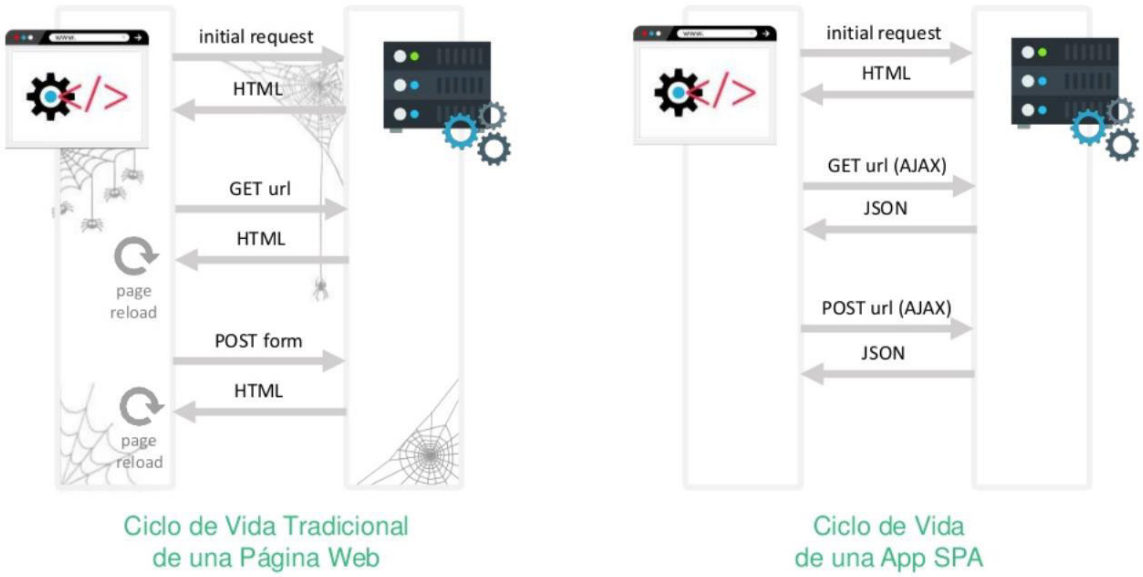


Figura #14. *Single page application*.
Fuente: Recuperado de https://www.slideshare.net/JavierAbada/django-vue-javascript-de-3-generacin-para-modernizar-django (2017)

Árbol de archivos

A continuación, en la Figura #15 se mostrarán capturas de pantalla de cómo se organizará el entorno.

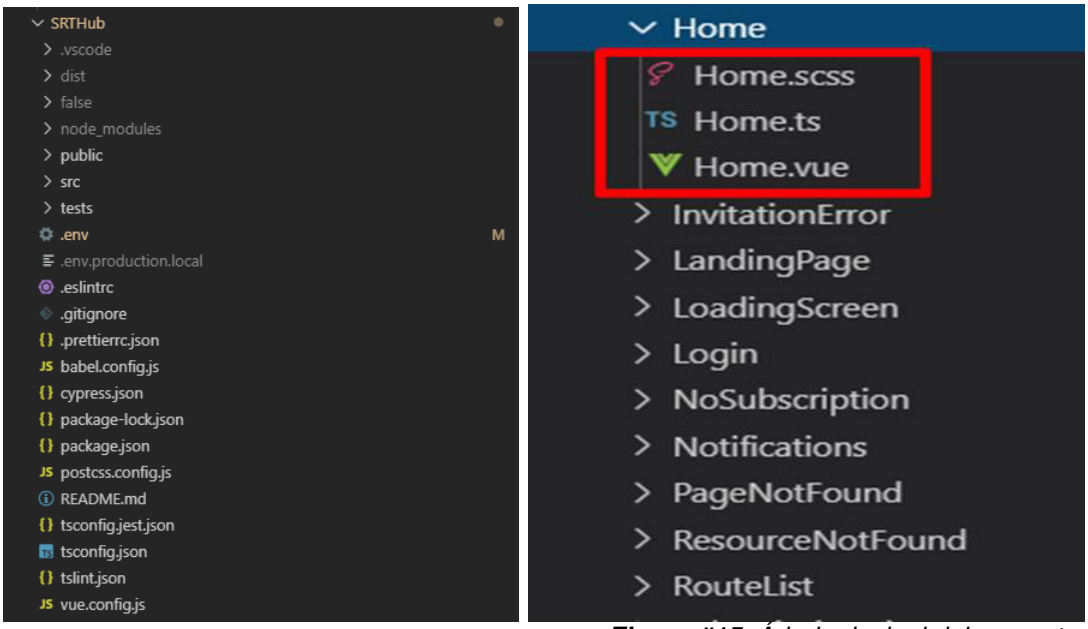


Figura #15. *Árbol principal del proyecto.*
Fuente: Elaboración propia, basada en la práctica.

En la Figura #12 se puede apreciar la configuración y organización principal del proyecto. Vue JS cuenta con un concepto llamado Componentes, de los cuales se podrán utilizar para tener un mayor orden en el proyecto. En la columna derecha, se observa que un componente consta de una carpeta, la que siempre contendrá tres archivos con diferentes extensiones, y éstas serán las detalladas a continuación:

- ***Archivo.vue***: archivo principal el cual se utiliza para creación, en este archivo se inserta código HTML y lógica propia del framework de Vue.
- ***Archivo*.scss:** este archivo se utiliza para otorgarle estilo a los elementos HTML.
- ***Archivo*.ts:** en este archivo se define toda la lógica del componente, utilizando typescript.

Componentes compartidos

Se utilizarán componentes compartidos entre ambos proyectos (API-Portal), esto se realizará con el uso de node.js para la conexión con el back-end. Los componentes compartidos se detallan a continuación:

- Componente de API.
- Componente AUTH.
- Componente notificaciones.

Componente API: este componente se utilizará para realizar la conexión con el back-end y así poder crear los servicios correspondientes para la creación de puntos finales. La Figura #16 muestra el servicio principal correspondiente al componente de API.

```
import {ApiService} from '../apiService';       Tomas Escobar, 2 months ago •
import {OrganizationServiceConfig} from '../organizationData';

Tomas Escobar, 2 months ago | 5 authors (Nahuel Berg and others) | Complexity is 3 Everything is cool!
export default class DeviceService extends ApiService {
    constructor() {
        const config: OrganizationServiceConfig = {
            organization: true,
            subscription: true,
            group: true,
        };
        super(config);
    }

    public async createBulk(file: File): Promise<string> {
        return this.sendFileOnRequest('/bulk/hublets/routes/create', file);
    }
```

***Figura #16**. Ejemplo de un servicio correspondiente al componente API.*
Fuente: Elaboración propia, basada en la práctica.

Componente AUTH: este componente se utilizará para hacer el log-in con Microsoft. Para ello, se emplea un SDK de JavaScript para autentificar con OAuth2. La Figura #17 muestra la implementación del SDK utilizando el componente de AUTH.

```
import * as Hello from 'hellojs';  65.7K (gzipped: 20K)
import {IAuth, AuthConfig} from './interfaces';

const loginDisplayType = {
    PopUp: 'popup',
    None: 'none',
    Page: 'page',
};

const network = 'adB2CSignInSignUp';

southworks.sgomez, 2 months ago | 4 authors (Tomas Escobar and others) | Complexity is 14 You must be kidding
export default class Auth implements IAuth {
    Complexity is 13 You must be kidding
    constructor(applicationConfig: AuthConfig) {
        const helloJsSignInSignUpPolicy = 'adB2CSignInSignUp';

        Hello.init({
            adB2CSignInSignUp: {
                name: 'Azure Active Directory B2C',
                oauth: {
                    version: 2,
                    auth:
                        `https://${applicationConfig.tenantName}.b2clogin.com/tfp/` +
                        applicationConfig.tenantName +
                        '.onmicrosoft.com' +
    public login(displayType?: any, state?: string): PromiseLike<string | undefined> {
        if (!displayType) {
            displayType = loginDisplayType.Page;
        }

        if (!state) {
            state = '';
        }

        if (!this.isLogged() && displayType === loginDisplayType.None) {
            return this.login(loginDisplayType.Page);
        }

        Complexity is 4 Everything is cool!
        return Hello.login(network, {display: displayType, state}).then((auth) => {
            return auth.authResponse && auth.authResponse.access_token || undefined;
        },
        (err) => {
            return undefined;
        },
        );
    }
```

Figura #17. *Utilización del SDK para realizar el log-in en la aplicación.*
Fuente: Elaboración propia, basada en la práctica.

<u>Componente Notificaciones:</u> este componente se utilizará para hacer la conexión con el servicio SignalR y enviar notificaciones desde el back-end al front-end. La Figura #18 representa la conexión con el servicio de SignalR.

```
import {default as Vue} from 'vue';
import axios from 'axios';  13.5K (gzipped: 4.7K)
import * as signalR from '@aspnet/signalr';  67.6K (gzipped: 17.3K)

Andres Rodriguez Schiller, a year ago | 2 authors (Andres Rodriguez Schiller and others)
interface PendingListener {
    target: string;
    callback: NotificationCallback;
}

export type NotificationCallback = (...args: any[]) => void;

Complexity is 23 You must be kidding
export function notificationPlugin(
    vue: typeof Vue,
    options: NotificationOptions,
): void {
    let connection: signalR.HubConnection;
    let pendingListeners: PendingListener[] = [];
    let isReady = false;
    let timeoutCounter = 1000;
    function ready() {
        flushPendingSubscriptions();
        isReady = true;
    }
```

```
function flushPendingSubscriptions() {
    for (const listener of pendingListeners) {
        connection.on(listener.target, listener.callback);
    }
    pendingListeners = [];
}

Complexity is 12 You must be kidding
options.loginEvent(() => {
    const token = options.getAuthTokenFunction();
    const getConnectionInfo = axios
        .post(`${options.baseUrl}/notifications/negotiate`, null, {
            headers: {
                'Authorization': `bearer ${token}`,
                'X-MS-CLIENT-PRINCIPAL-ID': 'Debugging User',
            },
        })
        .then((resp) => {
            start(resp.data);
        });
```

Figura #18*. Integración del servicio SignalR, componente notificaciones.*
Fuente: Elaboración propia, basada en la práctica.

Sistema de manejador de errores

El sistema de manejo de errores será responsable de administrar las solicitudes realizadas a la API. Éste ejecuta acciones específicas dependiendo de si la solicitud tuvo éxito o no. El mismo está construido como un envoltorio alrededor de la verdadera fábrica de acciones de Vuex.

Además, será el responsable de la gestión de los observadores y sus suscripciones a los eventos. Cuando se produce un evento, el encargado del mismo enviará notificaciones a los observadores suscritos a ese evento.

Dado que el "layout" es el componente raíz de todos los demás componentes, creará y registrará en el gestor de eventos dos observadores: uno para cuando una petición tenga éxito y otro para cuando fracase. De esta manera, las notificaciones serán globales. En la Figura #19 se representa la arquitectura del sistema de manejador de errores.

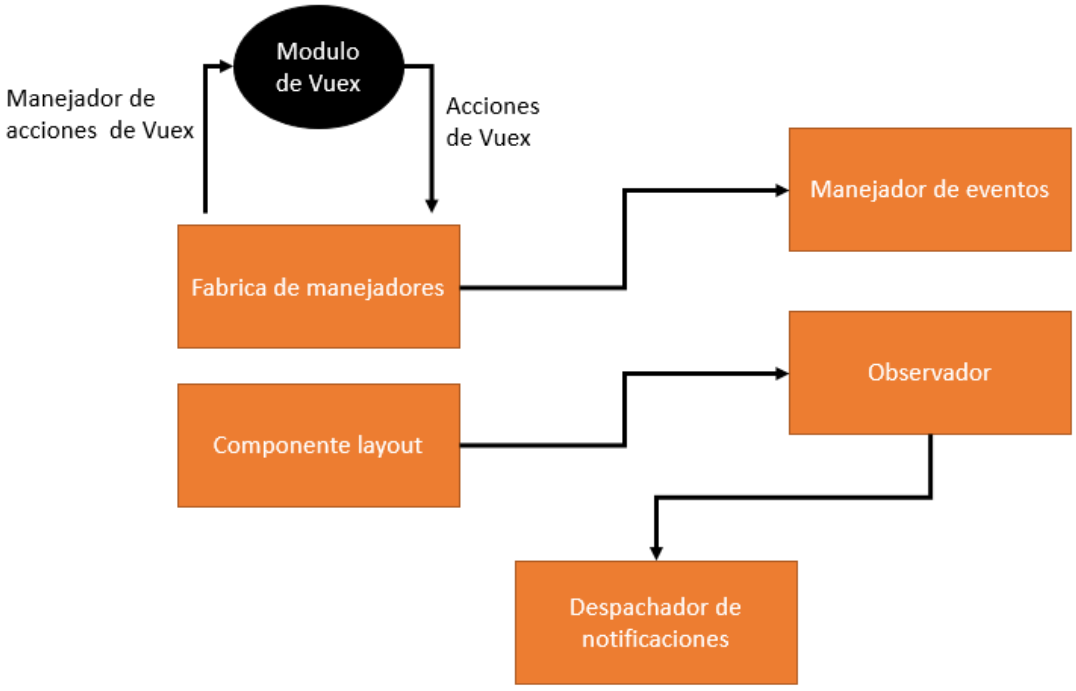


***Figura #19**. Sistema manejador de errores.*

Fuente: Elaboración propia, basada en la práctica.

Pantallas principales

Creación de una ruta y funcionamiento principal

A continuación, se detallará paso a paso el funcionamiento principal de la aplicación siguiendo un flujo normal que realizaría cualquier usuario al ingresar a SRT Hub.

Como se mencionó anteriormente, SRT Hub realiza el flujo de autenticación de usuarios a través de OAuth2, los cuales estarán integrados con la base de datos para dar de alta un usuario. Actualmente soportará el ingreso utilizando una cuenta corporativa o individual de Microsoft o Gmail. En la Figura #20 se muestra la pantalla de log-in de SRT Hub.

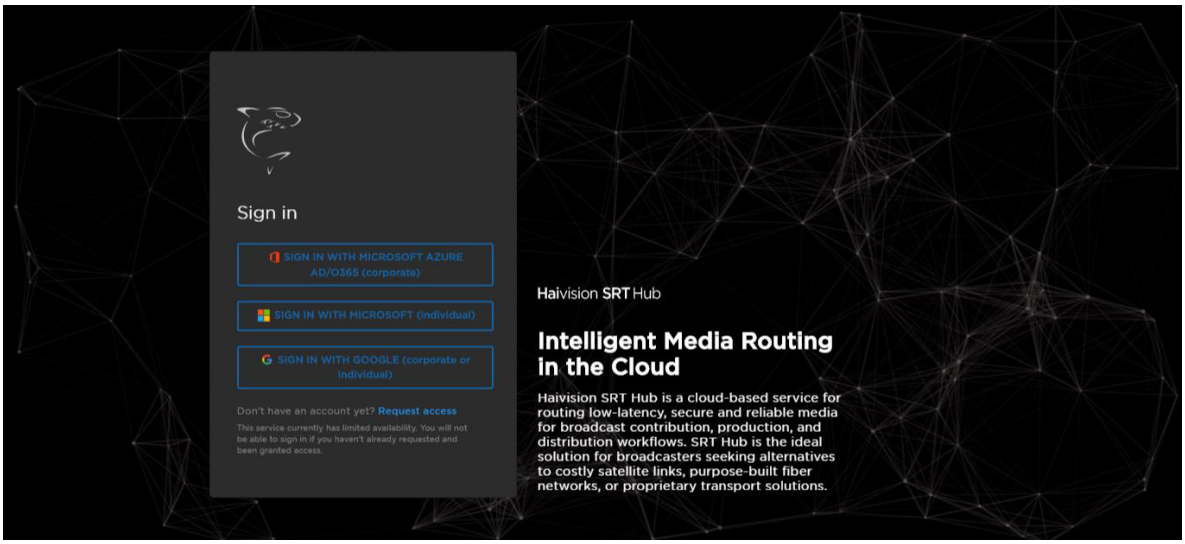


***Figura #20**. Pantalla de log-in de SRT Hub.*
Fuente: Elaboración propia, basada en la práctica.

Una vez que se ingresa a la aplicación, se visualizará la pantalla principal y en la que se observa la información de las rutas creadas, datos sobre los contenedores, y un mapa con la ubicación en donde fue creada cada una de ellas como se presenta en la Figura #21.

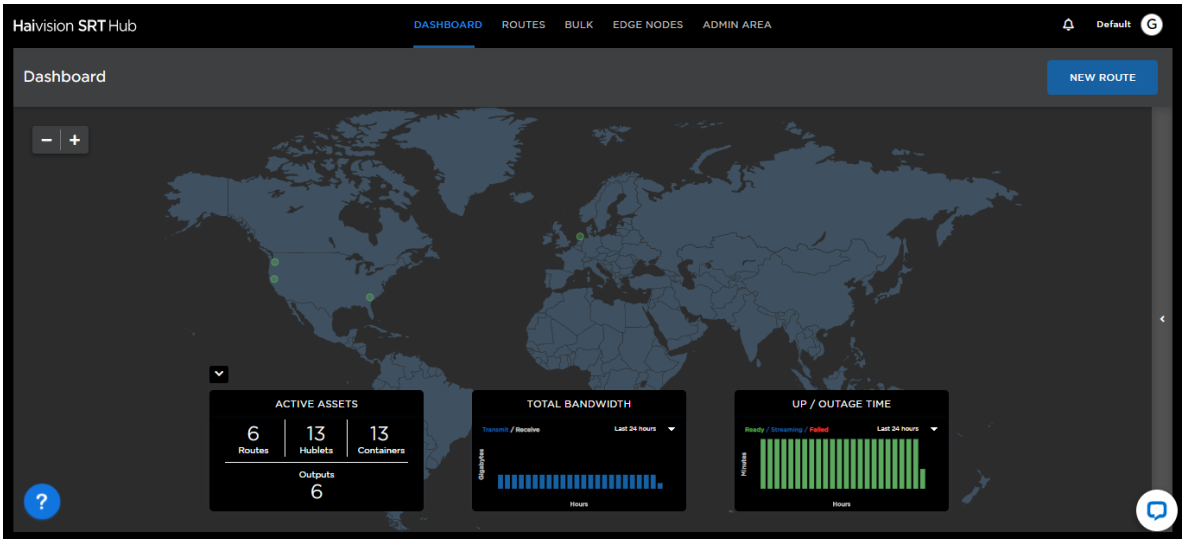


***Figura #21**. Pantalla principal de SRT Hub.*
Fuente: Elaboración propia, basada en la práctica.

Para crear una nueva ruta, se tendrá que clickear en el botón de “New route”, el que conduce a una nueva pantalla. En este espacio se incluirá un nombre, una descripción y se seleccionará la latencia en la que se quiere configurar la ruta.

Además, podrá ser configurada para que se detenga automáticamente en cierto periodo de tiempo. Esto se debe a que cada ruta corriendo genera un gasto por minuto, y si una persona olvida detenerla se seguirá cobrando por su uso.

A continuación, las Figuras #22, #23 y #24 reflejarán los pasos necesarios para crear una ruta.

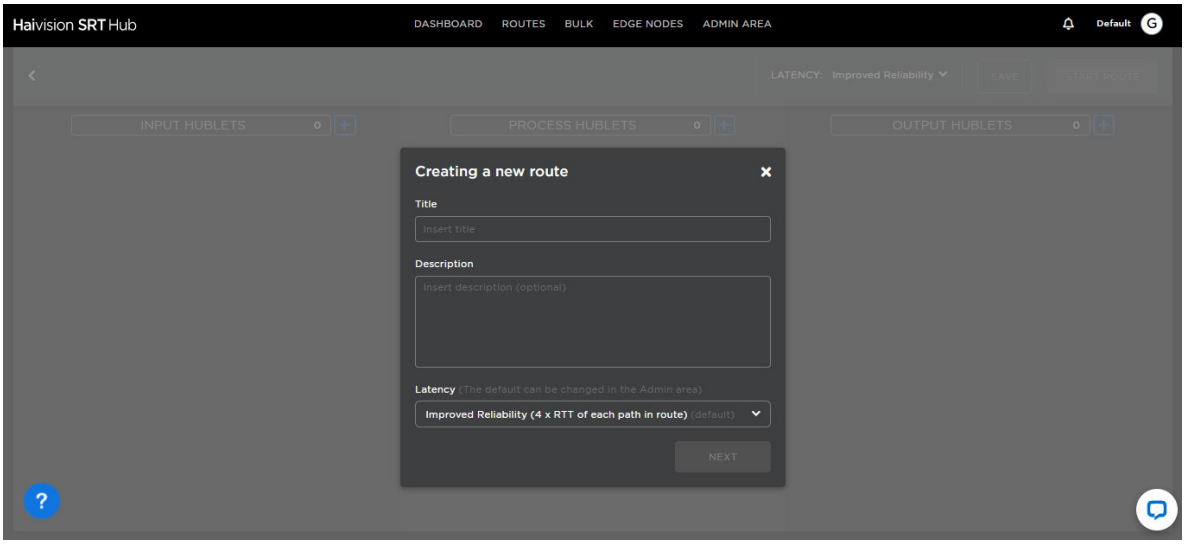


Figura #22. *Creación de ruta, paso 1.*
Fuente: Elaboración propia, basada en la práctica.

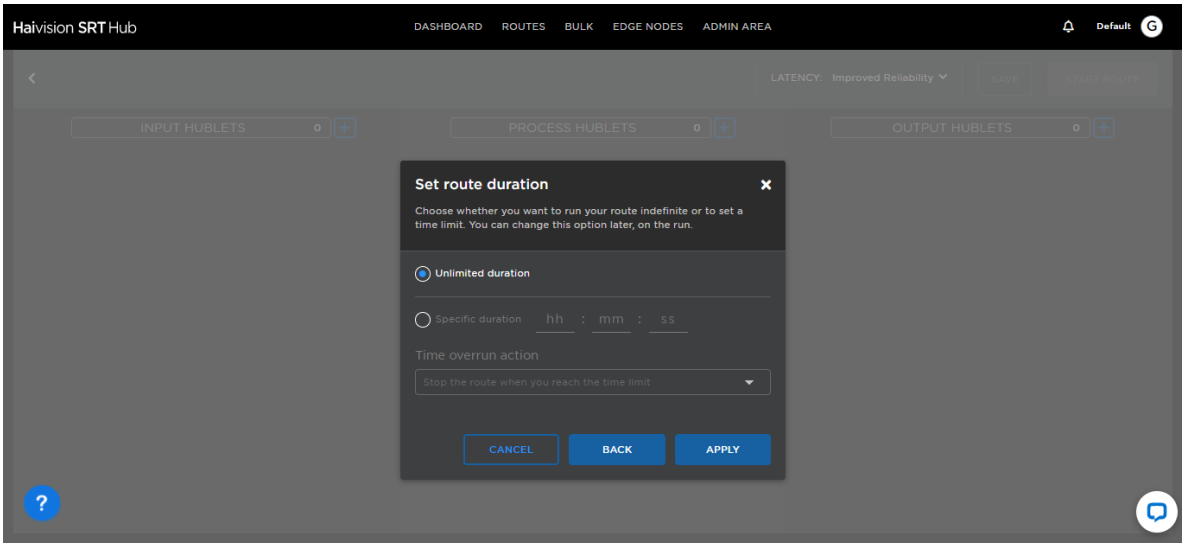


Figura #23. *Creación de ruta, paso 2.*
Fuente: Elaboración propia, basada en la práctica.

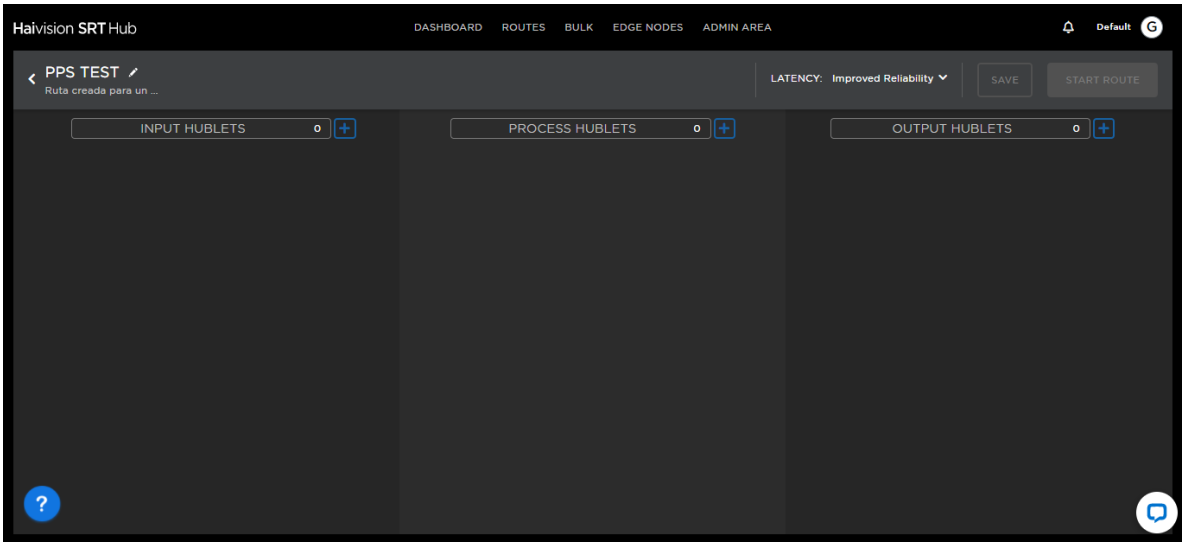


***Figura #24**. Creación de ruta paso 3.*
Fuente: Elaboración propia, basada en la práctica.

Una vez en esta pantalla, se continuará con la selección del Hublet y la creación del mismo. La Figura #25 muestra la selección de un Hublet.

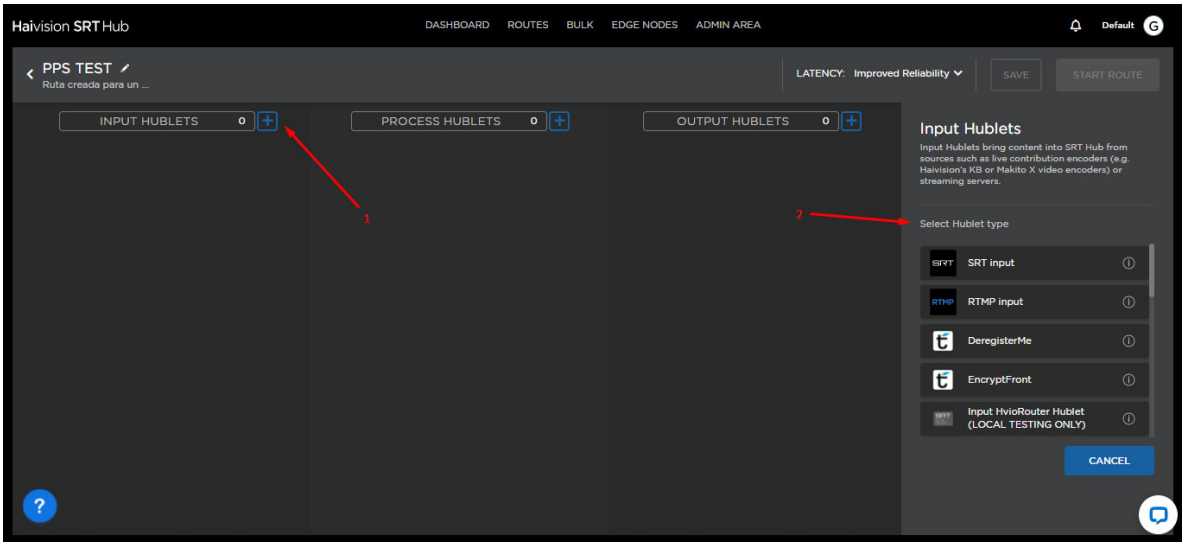


***Figura #25**. Creación de ruta, selección de Hublet.*
Fuente: Elaboración propia, basada en la práctica.

Al seleccionar un Hublet, se deben ingresar los parámetros correspondientes (éstos varían según el Hublet seleccionado). Para fines prácticos, se crearán dos Hublets de tipos genéricos (Figura #26).

Los principales campos serán:

- Nombre: nombre del Hublet.
- Región: región en donde Azure creará el Hublet.
- Modo: escuchante o llamador.
- Latencia: latencia con el cual se creará
- Frase clave: requerida para comenzar la transmisión.

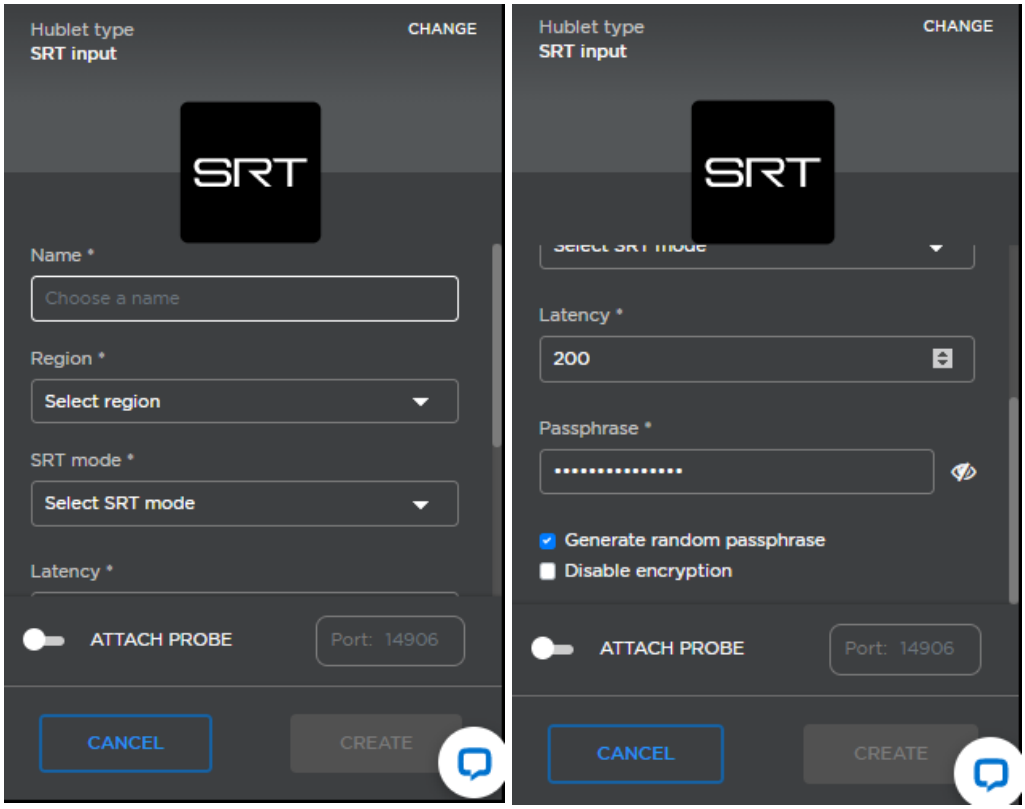


***Figura #26**. Creación de Hublet de tipo genérico.*
Fuente: Elaboración propia, basada en la práctica.

Se podrá crear la cantidad de Hublets que se deseen, sin embargo, la única condición es que un tipo de Hublet de salida, sólo puede tener 1 entrada. Es decir, la relación de entrada a salida es 1:N.

Una vez que los Hublets se encuentren conectados, se podrá guardar la ruta y comenzarla. Para este caso, se optará por comenzarla.

Por lo tanto, al iniciar la ruta, por detrás se estarán provisionando los contenedores de Azure. En las figuras #27 y #28 se visualiza una ruta lista para comenzar y su provisionamiento.

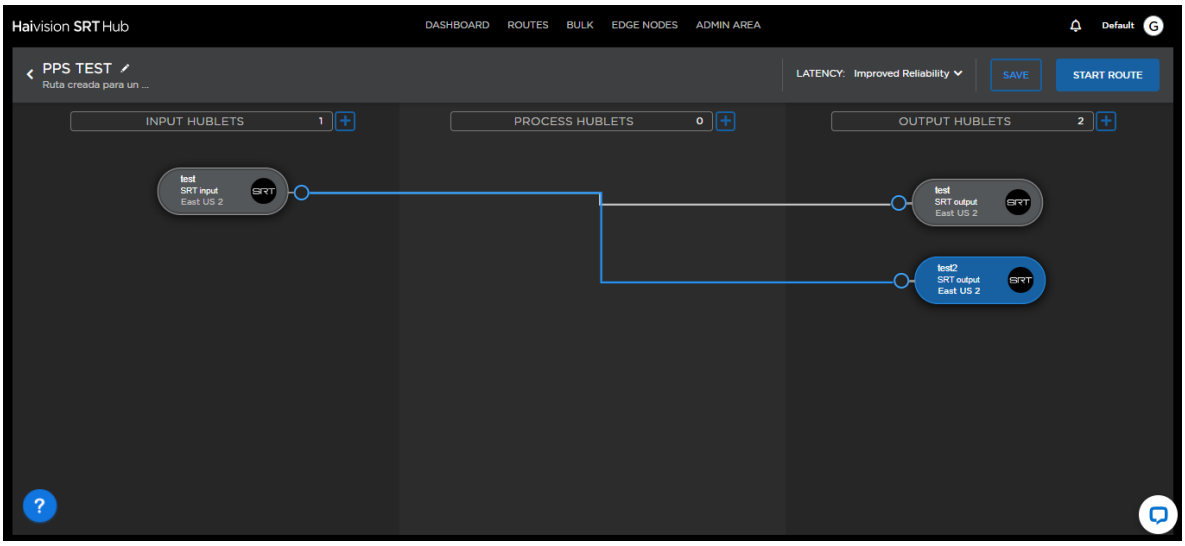


***Figura #27**. Ruta lista para comenzar.*
Fuente: Elaboración propia, basada en la práctica.

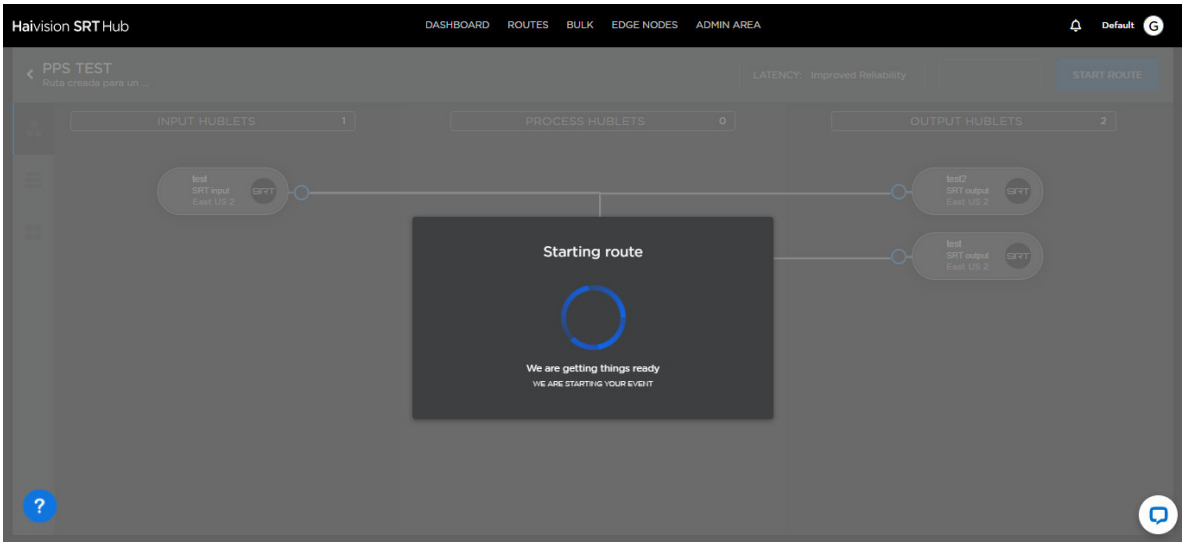


***Figura #28**. Comenzando una ruta.*
Fuente: Elaboración propia, basada en la práctica.

Una vez que se provisionan los recursos, se comenzará a recibir información de cada contenedor, lo que se denominará “información de telemetría”. La misma se indicará al hacer click en cualquier Hublet. Además, se obtendrá la IP del

contenedor levantado y la URL, que se utilizará para iniciar una transmisión en vivo. En la Figura #29 se representa la información necesaria de un Hublet para iniciar una transmisión.

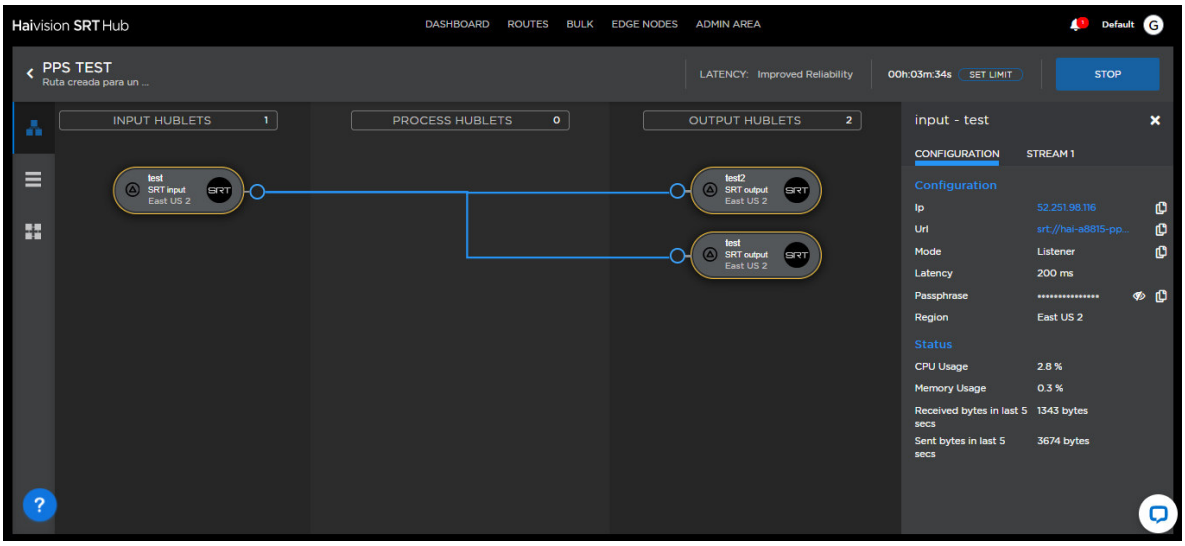


Figura #29. Información del Hublet para iniciar una transmisión en vivo.
Fuente: Elaboración propia, basada en la práctica.

Con esta información se podrá comenzar una transmisión en vivo desde cualquier lugar, tan solo se requerirá cualquier aplicación que soporte el protocolo SRT.

Telemetría en los contenedores

Cada contendor recibe permanente telemetría desde los servicios de Azure mencionados anteriormente. Esta telemetría se actualiza a cada segundo en la UI, permitiendo conocer en tiempo real el estado de cada uno.

Cabe mencionar que, si el contenedor no se encuentra transmitiendo, la información será nula como se muestra a continuación en la figura #30.

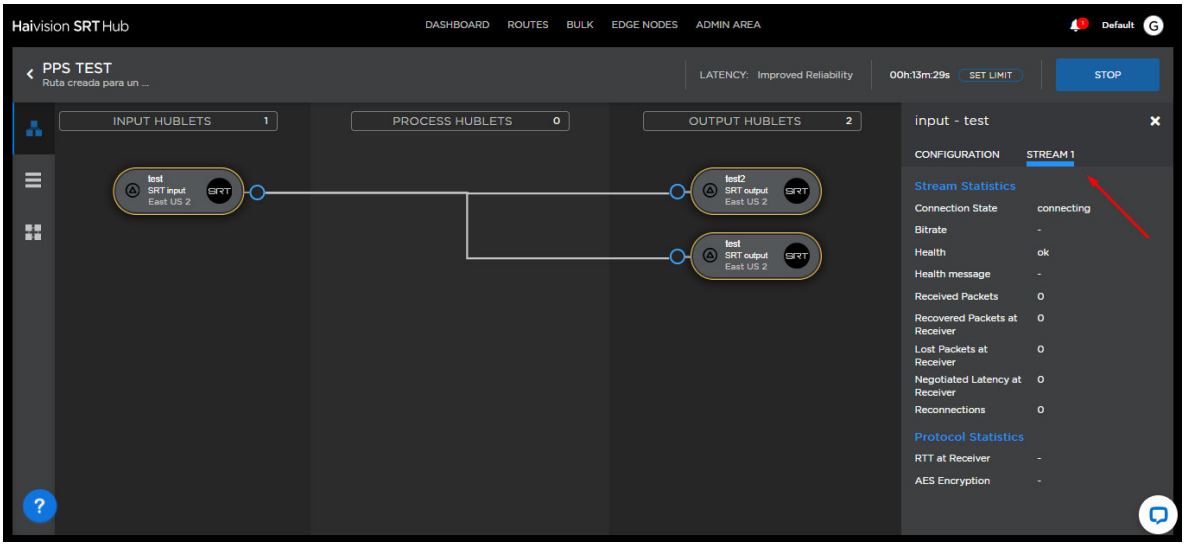


***Figura #30**. Telemetría de un contenedor.*
Fuente: Elaboración propia, basada en la práctica.

Features Flags

Estas “banderas” servirán para activar y desactivar diferentes funciones para los usuarios/organizaciones sin esperar a la ingeniería (desarrollo y despliegue).

¿Qué es una Feature?

Estos son algunos ejemplos de lo que puede ser una característica que indique la función de una Feature:

- Un Hublet.
- Un simple botón o una sección de una de nuestra página web.
- Una estrategia de acción/proceso/algoritmo realizada en el API.

¿Como se utiliza?

Implementación: Habrá un servicio de back-end expuesto como punto final de la API (api/features) para realizar operaciones CRUD en entidades de features flags almacenadas en la base de datos de Cosmos DB.

A continuación, se ejemplifica una operación de feature flag que sólo permite al usuario "juan" de la organización "southworks" utilizar el dispositivo KB. Otros usuarios de la organización "southworks" no estarían autorizados a utilizar el dispositivo de KB ya que la *flag* de la organización "southworks" estaría configurada como falsa. Sin embargo, el resto de organizaciones tienen

permitido usar el dispositivo de KB ya que la bandera está puesta en *true* para todos como se representa en la Figura #31.

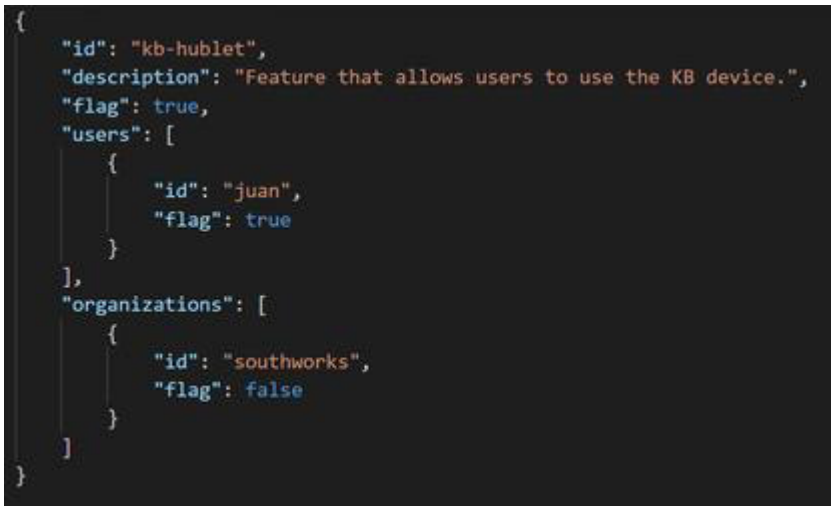


***Figura #31**. Ejemplo de Feature Flag.*
Fuente: Elaboración propia, basada en la práctica.

Uso: tanto la API como el sitio web pueden utilizar el servicio de features flags.

- SRT Hub recuperará las flags y las almacenarán en el estado de la aplicación (esto sucede después de actualizar el navegador) de modo que cada *flag* pueda obtenerse fácilmente mientras el usuario navega por la aplicación. Luego, cada elemento del sitio web se mostrará/ocultará en función de la bandera de características.
- La API consumirá el servicio de features flags directamente, ya que la API hace referencia al conjunto de servicios. En este espacio es donde se permitirá o no que los usuarios u organizaciones utilicen los Hublets. Además, aquí se sabrá qué estrategia se utilizará para realizar algún proceso basado en el lanzamiento de una nueva característica. La Figura #32 representa el flujo de trabajo de las Features Flags.

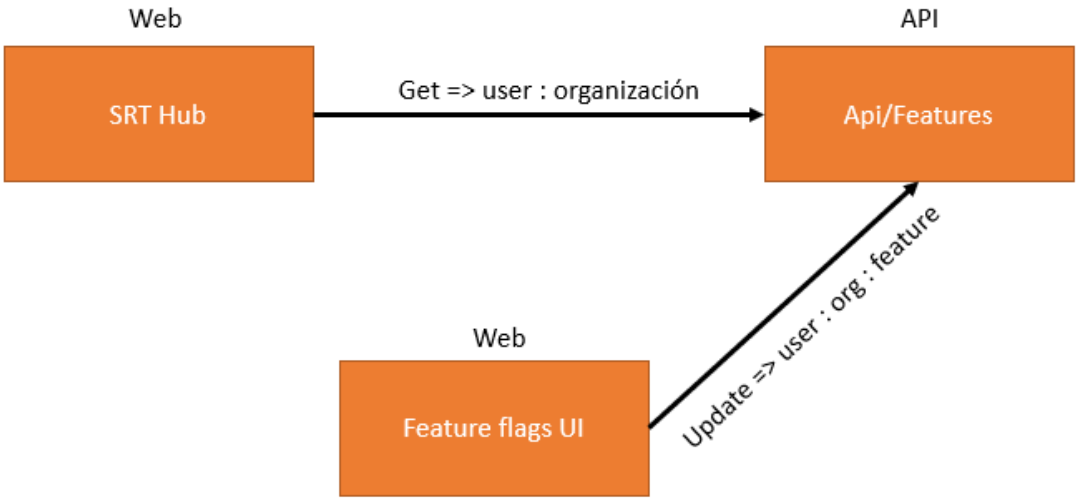


***Figura #32**. Flujo de trabajo de las features flags en alto nivel.*
Fuente: Elaboración propia, basada en la práctica.

Hublets externos

Registro y configuración

SRT Hub contará con la posibilidad de permitir a los usuarios crear sus propios tipos de Hublet. Para esto, desde la interfaz de usuario se tendrá la posibilidad de crearlos de una manera muy sencilla.

Para esto, se necesitarán permisos de administrador, en la pestaña de "manejo de Hublets" se observará que existe un botón que permite registrar un nuevo Hublet. Debido a que los Hublets son recursos administrados por Azure, se requerirá de información de una aplicación válida generada en Azure. Las Figuras #33, #34 y #35 muestran los pasos necesarios para registrar un Hublet externo.

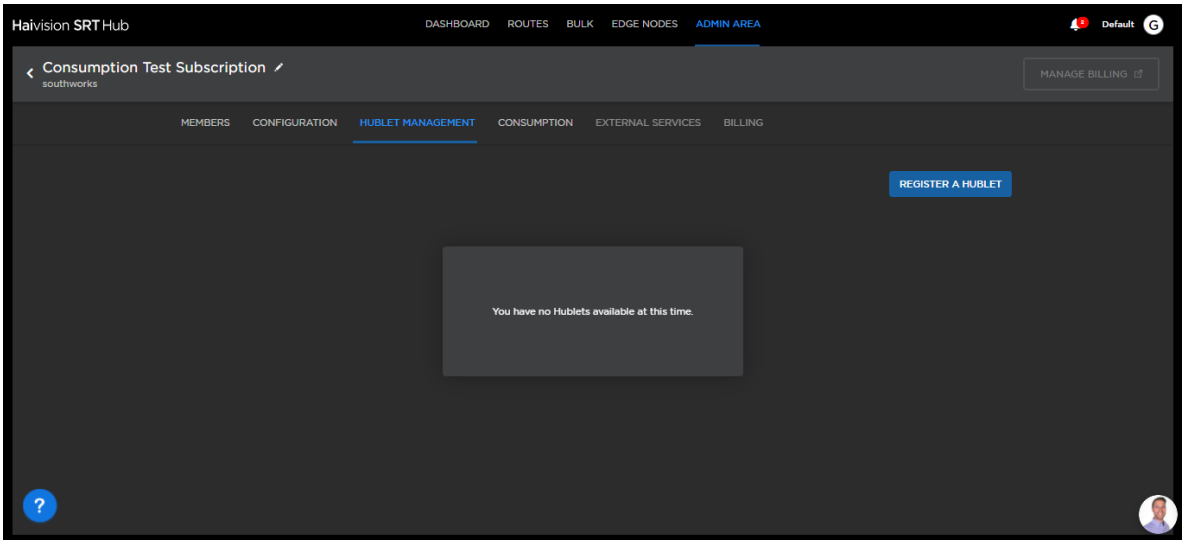


***Figura #33**. Registrando un Hublet.*
Fuente: Elaboración propia, basada en la práctica.

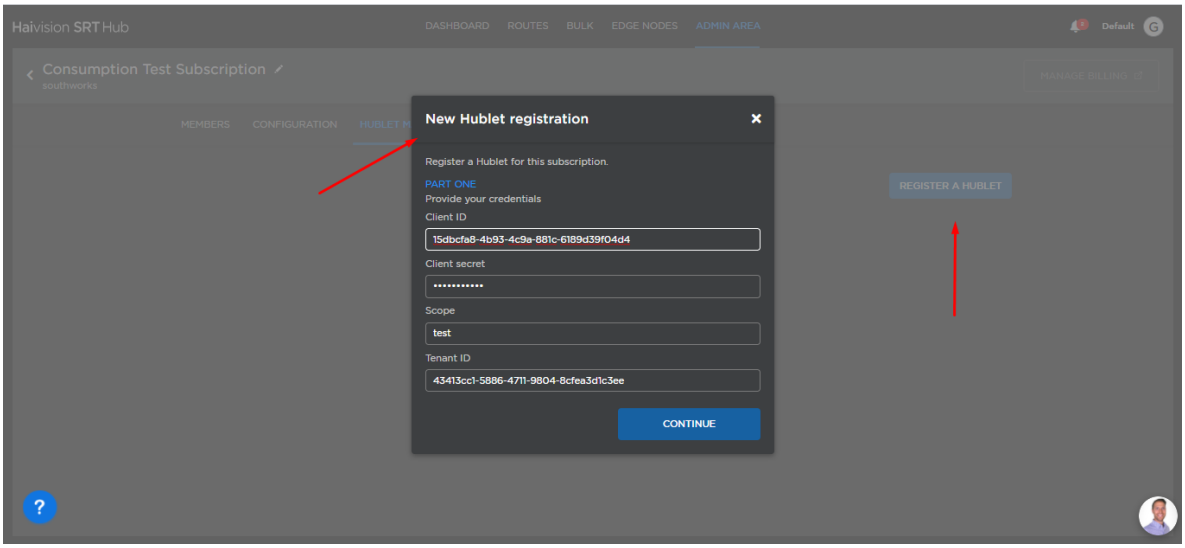


***Figura #34**. Registrando un Hublet paso 2.*
Fuente: Elaboración propia, basada en la práctica.

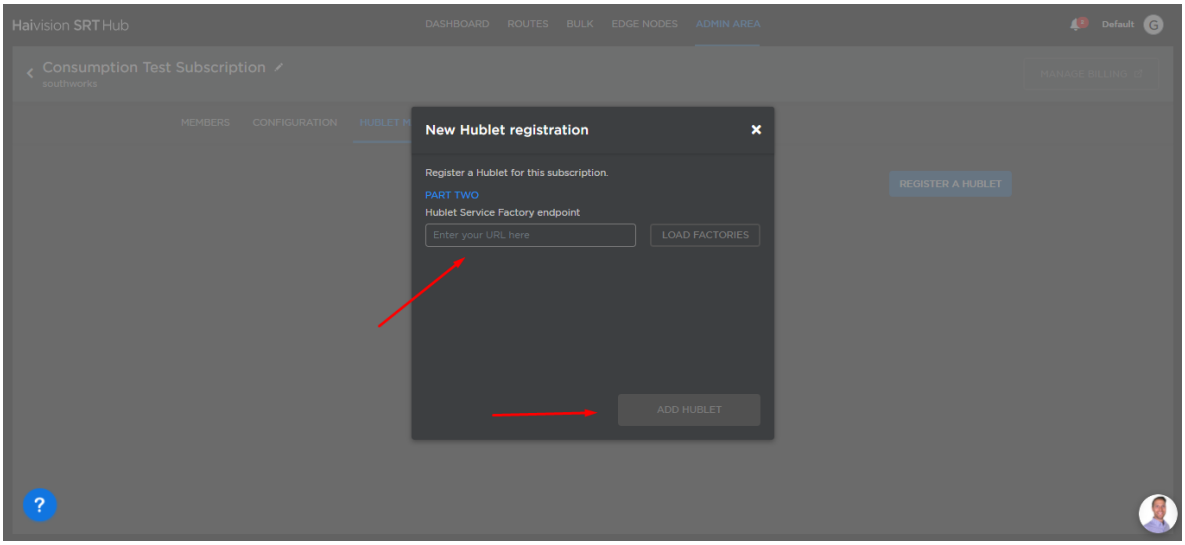


***Figura #35**. Registrando un Hublet paso 3.*
Fuente: Elaboración propia, basada en la práctica.

El objetivo principal de registrar un Hublet externo, consta de brindarle al usuario la posibilidad de crear Hublets con nuevas propiedades e información de telemetría específica para el uso de su dispositivo o recurso.

Concepto de telemetría

Concepto clave

La telemetría se utiliza para monitorear constantemente a los contenedores, además, se usa para conocer información si el tipo de Hublet es un dispositivo.

Por otra parte, la información de telemetría se recoge del Vuex Store a través de notificaciones SignalR. Inicialmente, al recuperar la ruta después de acceder a su detalle junto con varias propiedades, contiene información de telemetría que se almacena a nivel de ruta (la misma telemetría que se recibe para las notificaciones se atesoran en un almacén de blobs de Azure para finalmente ser obtenida y enviada cuando se lo solicita la ruta).

Salud en los contenedores

Para conocer la salud de los contenedores, se ha creado una nueva pantalla en la que se muestran los recursos de cada Hublet. Al hacer click en ellos, se desplegará un menú con todos los recursos asociados a ese Hublet, permitiendo

el monitoreo de una forma muy sencilla. En la Figura #36 se muestra la vista de topología, la que muestra los recursos de cada contendor.

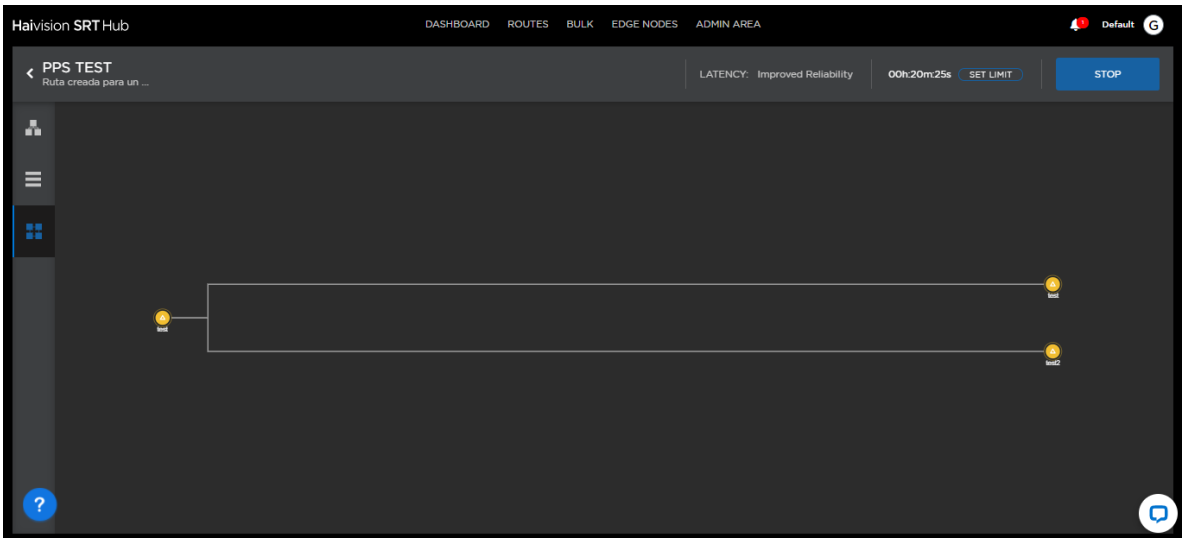


***Figura #36**. Pantalla de recursos de cada contenedor.*
Fuente: Elaboración propia, basada en la práctica.

Cabe mencionar que, si el contenedor no se encuentra transmitiendo, no se recibirá ningún tipo de información, y la misma tampoco variará en tiempo real. Las Figuras #37 y #38 muestran información más detallada de la vista de topología.

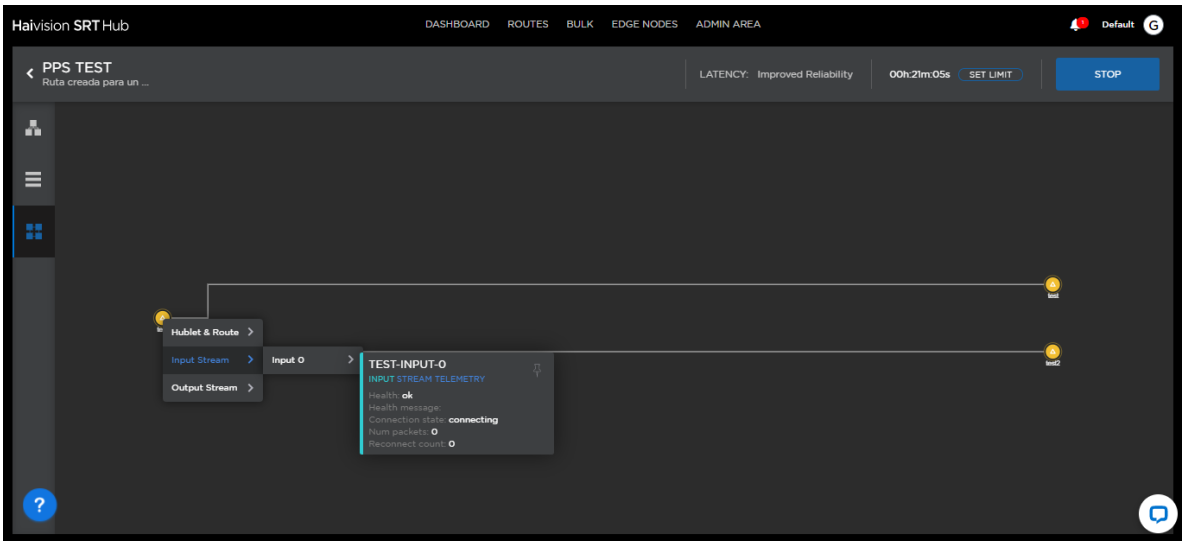


***Figura #37**. Accediendo a la información del recurso.*
Fuente: Elaboración propia, basada en la práctica.

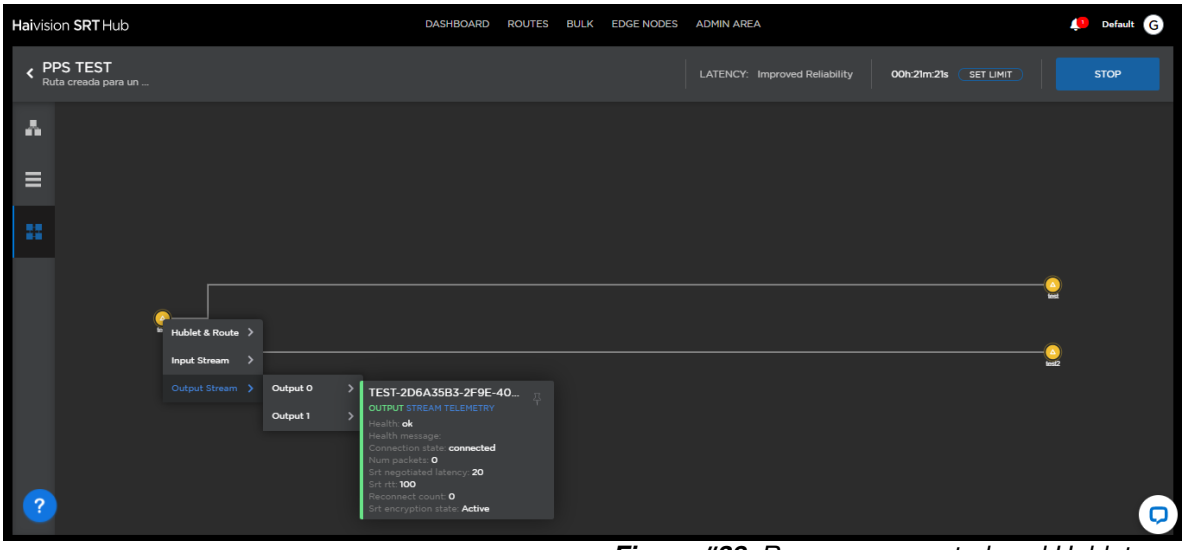


***Figura #38**. Recursos conectados al Hublet.*
Fuente: Elaboración propia, basada en la práctica.

Cada tarjeta generada por recurso, se podrá dejar fija del lado derecho de la pantalla, permitiéndole al usuario ver varias al mismo tiempo y conocer la telemetría de cada contenedor (Figura #39).

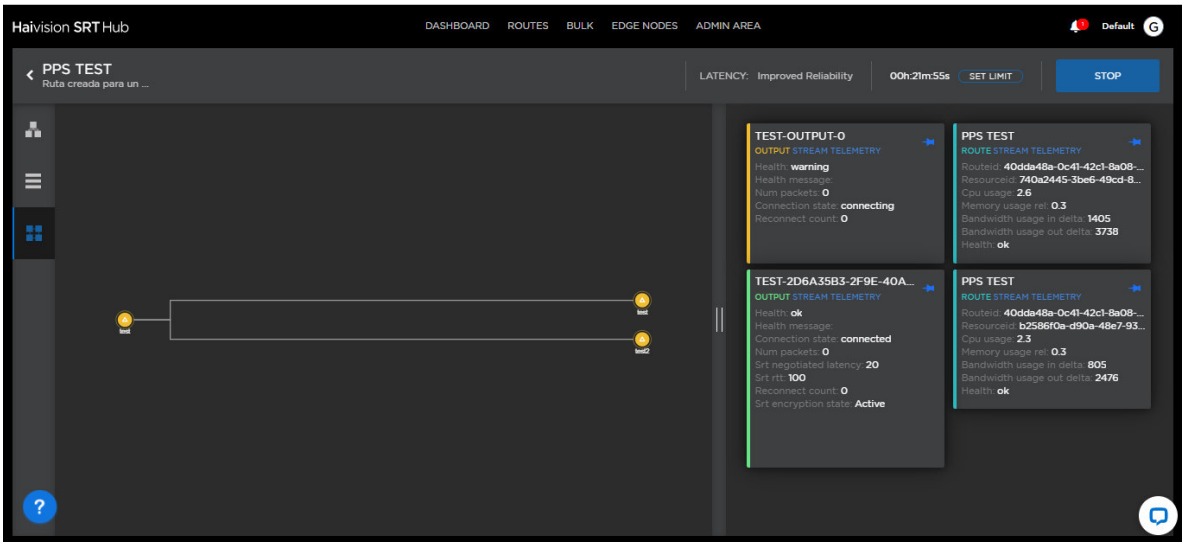


***Figura #39**. Mostrando información de varios recursos a la vez.*
Fuente: Elaboración propia, basada en la práctica.

.

Manejo de estados

Como se mencionó anteriormente, cada Hublet tiene un recurso asociado, y cada recurso tiene información sobre todos los contenedores disponibles en la ruta.

Por ejemplo, para una ruta con una entrada conectada a dos salidas, la respuesta será (Figura #40):

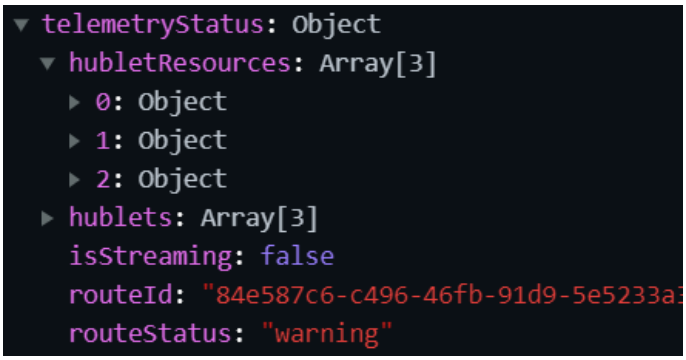


***Figura #40**. Ejemplo de objeto de estado de telemetría.*
Fuente: Elaboración propia, basada en la práctica.

- **hubletResources**: cada Hublet tiene un Recurso Hublet asociado.
- **hublets**: muestra la información de telemetría de cada Hublet.
- **isStreaming**: estado consolidado de la ruta, en caso de que un Hublet comience a transmitir, esta propiedad cambiará su valor a *true* mostrando nuevos comportamientos en la interfaz de usuario.
- **routeId**: identificación de la ruta.
- **routeStatus**: estado consolidado, brinda información general del estado de la ruta.

Dentro de cada entrada o salida se tendrá información sobre el estado y también información si se está transmitiendo (esta información vendrá a nivel de ruta, recurso de Hublet y contenedor. Ver Figura #41).

***Figura #41**. Objeto de un recurso de Hublet.*
Fuente: Elaboración propia, basada en la práctica.

Los estados de un contenedor podrán ser:

- Error: algo no funciona como se esperaba.
- Advertencia: el contenedor está esperando para transmitir.
- Ok: el contenedor tiene una salida correctamente conectada.
- Desconocido: no hay información al respecto.

Además, se cuenta con una propiedad llamada "Estado de telemetría" dentro de cada recurso de Hublet, que indica si el contenedor está levantado o caído. En el caso que estuviera caído, no se mostrará ninguna información de telemetría y la ruta y el contenedor se mostrarán en color gris como "desconectado" como lo representa la Figura #42.

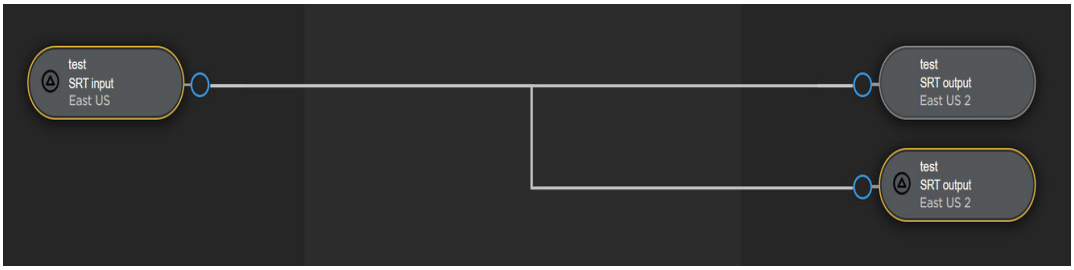


Figura #42. Hublets con dos diferentes estados.
Fuente: Elaboración propia, basada en la práctica.

Esta es toda la información requerida que un usuario necesita para entender el funcionamiento de la aplicación, la base principal de SRT Hub consta de lo detallado en el apartado de arquitectura y servicios. Con la implementación del "Orquestador", se ha logrado proveer automática e inteligentemente todos los recursos de procesamiento requeridos por el centro de datos de Azure, y de este modo escalar y enrutar los medios de comunicación desde cualquier parte del mundo.

Además, se ha logrado obtener toda esta información del centro de Azure convirtiéndola en elegantes pantallas WEB, logrando una novedosa y atractiva estadía para cualquier usuario que desea realizar una transmisión en línea.

Conclusiones

SRT Hub es un producto de alta completitud que incluye una amplia variedad de servicios, la aplicación logró destacarse rápidamente en el mercado multimedia por su novedoso servicio y pantallas WEB.

En un comienzo, SRT Hub fue solicitado para realizar una presentación virtual entre dos regiones, por lo que, la arquitectura ha sido planteada de manera básica y general sin demasiados procesos ni arquitecturas demandantes. A medida que se fue encontrando potencial en el producto, se han tomado diferentes decisiones de arquitectura y agregando así el llamado orquestador, permitiéndole a la aplicación escalar exponencialmente. Esto ha sido una complicación debido a que en su momento de origen el concepto de contenedores de Azure era nuevo en el mercado y se ha tenido que ir adaptando a los nuevos requerimientos.

Incluso en un principio no existía ninguna experiencia de usuario desarrollada para la cual crear Hublets, ha sido todo un desafío crear todas estas novedosas pantallas WEB utilizando las últimas tecnologías del mercado. Por otra parte, han existido complicaciones y cambios en cuanto a los requerimientos visuales, muchos de ellos varían constantemente con el paso de los días y realizar todas las integraciones con los servicios y arquitectura del back-end no es algo relativamente sencillo. No obstante, el resultado obtenido es el esperado ya que SRT Hub es una aplicación única por sus novedosas pantallas WEB.

También, ha sido un gran desafío comprender y utilizar todos los servicios de Azure mencionados en este informe, cada uno de ellos es sumamente requerido para el correcto funcionamiento de la aplicación, se han presentado diversos problemas en cuanto a la integración de los mismos debido a que constantemente las versiones se van modificando y con tan sólo un servicio fuera de versión basta para que la aplicación deje de cumplir con su funcionamiento principal.

Para solucionar los problemas mencionados, se han planteado y desarrollado diferentes alternativas, como, por ejemplo, el correcto uso de “GIT” como controlador de versiones, el uso de “JIRA” como organizador de trabajo y, por

último, el uso de “Pipelines” para verificar y desplegar las últimas modificaciones en el repositorio de trabajo. Cada vez que se realice algún cambio en el proyecto, se verifica cada proceso y servicio dentro de la aplicación, en caso de que se detecte alguna falla o problema a causa de esos cambios realizados, no se desplegará la última versión a la web hasta que se hayan resuelto.

Así pues, gracias a su arquitectura multinivel, cualquier persona en el mundo puede iniciar una transmisión en vivo con su celular o dispositivo portátil de manera muy sencilla y con una latencia muy baja entre las diferentes regiones del mundo.

Además, la arquitectura tanto del back-end como del front-end permite a la aplicación seguir escalando globalmente sin límite. Con los múltiples servicios utilizados, se brindará a los usuarios toda información útil y necesaria sobre el estado de la transmisión y de cada contenedor en particular, permitiéndole así mayor claridad al momento de su uso.

Reflexión sobre la PPS

En lo personal, pienso que a lo largo de este trayecto he aprendido a transmitir mucho mejor mis ideas, lo cual es sumamente importante para expresarse correctamente en nuestros empleos o en la propia vida cotidiana. La Práctica Profesional Supervisada es un trabajo necesario que como estudiantes nos hace poner a prueba ya que necesitamos transmitir todo concepto aprendido durante nuestra carrera. Quiero resaltar el trabajo constante del departamento de sistemas de la Universidad, ya que es un área en la se requiere de constante actualización puesto que las tecnologías enseñadas varían rápidamente de versión y características.

Por consiguiente, quiero hacer una mención especial a las materias “Proyecto de Software” e “Ingeniería de Software I” dictadas por el Ing. Oscar Bracho. En la primera de ellas, he aprendido lo que significa trabajar en un proyecto de programación con más de 5 personas. La modalidad de trabajo y el profesor, simulando ser un cliente, me ha ayudado a entender cómo se viven esas situaciones en un ambiente profesional. Luego de poder conseguir empleo como

desarrollador, me he dado cuenta lo útil que fue pasar por cada una de las etapas planteadas por la materia. Por otro lado, con la materia “Ingeniería de Software I” he comprendido los conceptos fundamentales que un profesional en sistemas debe tener siempre en mente, esta materia me ayudó a adaptarme rápidamente por todos los empleos que he tenido, ya que en la mayoría de las empresas de sistemas adoptan alguna metodología de trabajo en particular dictadas en esta materia.

También, quisiera destacar el contenido y la enseñanza de todas las materias relacionadas al campo de “Técnicas aplicadas a Analista Programador”, en lo personal pienso que las materias dictadas aquí son fundamentales para cualquier profesional en el ámbito de sistemas. Cada una de estas áreas nos brinda la posibilidad de conocer los muchos conceptos relacionados a Algoritmos de programación, Sistemas Operativos, Base de datos y Redes de Computadoras entre otras. Tener la posibilidad de forjar una base sólida, en cada una de estas materias, nos prepara para cualquier tipo de empleo para el cual podríamos aplicar como profesionales.

Además, con la ayuda de la Prof. Lía Lavigna y el Dr. Ing. Martin Morales, he logrado progresar en muchos aspectos académicos los cuales me ayudarán a seguir forjándome como profesional en mi trabajo o en futuros nuevos empleos.

Bibliografía consultada

- Abadía, Javier (24 de septiembre de 2017). *Django + Vue, JavaScript de 3ª generación para modernizar Django*. Slideshare. Recuperado de https://www.slideshare.net/JavierAbada/django-vue-javascript-de-3-generacin-para-modernizar-django.
- Cogan, Sam (24 de febrero de 2019). *Windows Containers and Azure*. Samcogan. Recuperado de https://samcogan.com/windows-containers-and-azure/.
- Firtman, Maximiliano (2015). *High Performance Mobile Web: Best Practices for Optimizing Mobile Web Apps.* California: O'Reilly Media.
- Gamma, Erich & Helm, Richard & Johnson, Ralph & Vlissides, John (1994). *Design Patterns: Elements of Reusable Object-Oriented Software,* Boston: Addison-Wesley.
- Gido, Jack & Clements, James (2012). *Administración exitosa de proyectos*. CABA: Cengage Learning.
- Sommerville, Ian (2005). *Ingeniería de Software*. Madrid: Pearson Educación.
- Stoliarov, Vitaliy (2017). *Rete JS*. Recuperado de https://rete.js.org/#/docs.
- Tanenbaum, Andrew S (2009). *Sistemas Operativos Modernos (3ra edición).* Madrid: Pearson Educación.
- W. West, Adrian (2016). *Practical Web Design for Absolute Beginners (1ra edición)*. New York City: Apress.