Instituto Tecnológico de Costa Rica I Semestre 2019
Est. Randy Martínez Sandí
Curso: SOA4ID
Prof. Raúl Madrigal Acuña
Fecha: 15/5/19

Proyecto III: Smart Hotel Services

1. **Smart Phone App:**
    a. **React-Native (Expo CLI - Mobile Tools):**

La utilización de la biblioteca de React-Native muy rápido y ligero. En esta ocasión se decidió usar otros sistemas de navegación para las pantallas: pestañas abajos, diferentes pilas para una serie de pantallas en una pestaña dada. También usaron los métodos y sistemas de *async storage* de React Native. Probé nuevos elementos de interfaz nuevos, que no conocía antes, dando un vista más agradable y más fácil de manejar la información que se esté trabajando con.

Esta vez produje con código más depurado y limpio, más ordenado en carpetas a pesar de que son más pantallas y más cosas que controlar. Utilizó e importó librerías, funciones y métodos que sólo usó en este lugar y momento. Aplique mucho más la premisa de React de crear componentes que reacciones entre ellos y sean más ligeros. Me queda la duda de la nomenclatura para las carpetas donde tengo el código. Investigando en internet, casi que toda las respuestas quedan a autor (programador), por lo que decidí nombrar esta vez con lo nombres más representativos para cada parte del código.

Algunas cosas que podrían mejorar el código: (1) reutilización de componentes, (2) funciones fechas y funciones normales, (3*) utilización de librerías o implementación personales y (4) ordenamiento lógico y secuencial de la información. (1) Hice varios componentes específicos que pude haber hecho para usos múltiples, por lo que tengo una porción de código repetido que me pudo haber ahorrado, esto lo noté cuando ya todo está funcionamiento y estaba haciendo pruebas. (2) En algunos casos utilizó funciones flechas y en otros funciones normales que tengo que declarar en cl constructor con el *.bind(),* me queda la duda de cuándo se debe usar cada cual y por qué motivo. (3*) Este punto lo exponen en el la sección de Sanity y (4) queda como a cargo del UX saber cuál es la mejor manera de localizar visualmente las cosas (información) y cómo debería interactuar con eso, mi despliegue se basó en mi manera de pensar.

No obstante, es extraño que saquen una versión nueva de CLI de manera tan seguida. Se empezó el proyecto con la versión 2.17.4 y se actualizó a la 2.18.5 y después a las 2.19.5. Al menos en Mojave 10.14.5 cada actualización hace que falle npm y node.js. Se caen los permisos de EACCES de npm y se tiene que estar corrigiendo eso en cada actualización. React-Native y su SDK en Expo hacen el proyecto muy fácil de utilizar, es portable, sus pruebas y visualización en dispositivos es rápido, seguro, compacto y fiable, pero las configuraciones pueden hacer que sea complicado.

2. **Backend**:
    a. **AWS Services (AWS Amplify - Cloud Services):**
    b. **Sanity (Node.js - Content Manager):**

Sanity resultó ser una herramienta muy simple, ligera y rápida para crear manejadores de contenido. El balance entre el uso de interfaz e implementación de código fue la correcta. Es decir, al crear el proyecto se hacen los esquemas que se desean y se ajustan con los datos que se quieren trabajar más adelante. Una vez estos declarados en el .js principal, se ejecuta el proyecto y se levanta la parte de interfaz donde uno va viendo el trabajo realizado. Todo cambio que se haga en el código ya queda implementado en tiempo real. Al finalizar se hace un *deploy* en la nube de Sanity.io y todo el manejador y contenido queda listo para acceso en todo lado y en todo momento.

Todo el contenido se puede consumir a través de los clientes para la aplicación que se esté desarrollando, sea Node.js, Java, PHP, React, React-Native, etc. La librería que ofrece para React-Native la investigué y revisé, pero no la usé por cuestiones de tiempo. Decidí usar la función *fetch()* para consumir el JSON que se genera a partir de la extensión de consultas que ofrece Sanity desarrollados por ellos llamado GROQ. Este GROQ es un sistema de consultas que permite conseguir la información que uno quiere o necesita de todo el contenido de los esquemas. *Todo esto se devuelve en formato JSON. Así que, hago una consulta ya predeterminada, esa consulta me devuelve un enlance todo está el JSON, es decir, es un servicio. Ya en la app, con *fetch()* consumo este JSON. No sé cuáles es la ventaja de hacer esto con respecto a instalar la librería de para React-Native, realizas las consultas desde ahí y utilizar los métodos de consulta. Como dije antes, no probé por tiempo. Lo considero más elaborado y un deber más de lógica a la aplicación, pero en términos de seguridad no sé si es eficiente y seguro.

Por cuestiones del proyecto, todos los accesos quedan permitidos, no hay restricción de consumo ni de permisos de pago por consumir algo. Esta herramienta es muy buena y adaptable para proyectos simples y pequeños como este, no obstante, me queda la duda de cómo será su comportamiento en gran escala. Sanity tiene buena documentación, vídeos interactivos.

    c. **Apixu API (Climate Weather):**

El API utilizado para consultar el clima, es relativamente sencillo. Con la cuenta que uno asocia al sistema se pueden obtener datos deseados. Quizá, por el hecho que sea gratis, hacer que tenga ciertos detalles que podrían ajustarse. Por ejemplo, las funciones y parámetros ofrece para consultar son variadas y eso permite flexibilidad, pero toda consulta (al menos hasta donde se probó) siempre retorna muchos datos que no todos se van usar. Sería genial que se pudiera hacer uno consultas sea con GraphQL o algún lenguaje/herramienta para obtener únicamente los datos que uno desea trabajar. Datos como humedad, velocidad del viento, porcentaje de nubes (sin considerar lluvias) o presión atmosférica son demasiado específicos y de poco interés, al menos para este caso. Tener que 'parsear' o trabajar con estos datos sin utilizarlos del todo, hace un gasto innecesario.

3. App: Capturas de Pantalla