Práctica 3. Aplicación Web de control de gastos

En esta práctica crearemos una aplicación web progresiva (PWA), que por lo tanto se podrá instalar desde el navegador en nuestro dispositivo móvil, ordenador, etc., que no va a requerir crear un servidor, si no que utilizará lo que se denomina una arquitectura Serverless.

Los navegadores actuales son capaces de almacenar una gran cantidad de información de forma local, e incluso pueden realizar funciones de base de datos o gestión de archivos. Aprovecharemos esta capacidad para crear una aplicación que guarde todo la información localmente en el navegador, y que ni tan siquiera necesite una conexión a internet para funcionar, una vez que se ha instalado.

La arquitectura Serverless sí que requiere de servidores, sin embargo estos son genéricos, y no es necesario que nuestro código se ejecute en los mismos. Esto lo conseguiremos gracias a que podremos crear una versión estática de nuestra aplicación, y solo necesitaremos publicar los archivos generados en algún repositorio de archivos.

Existen muchas opciones para guardar y publicar los archivos estáticos de nuestra aplicación. En esta práctica utilizaremos el servicio GitHub Pages, incluido en tu cuenta GitHub de estudiante de la UAM, y que podrás usar en cualquier proyecto, simplemente usando la rama gh-pages de tu repositorio.

Como framework para nuestra aplicación, utilizaremos VUE, junto con otras librerías, como NUXT, VUEX, Vuetify, etc., de las que hablaremos en los siguientes apartados.

Para el desarrollo de esta práctica, se recomienda instalar el plugin Vetur en Visual Studio Code, así como la extensión para Chrome o Firefox Vue Dev Tools.

Requisitos de la aplicación

Aunque dispones de gran libertad creativa a la hora de desarrollar esta aplicación, se recomienda comenzar por los requisitos mínimos, antes de incorporar una mayor funcionalidad.

Los requisitos mínimos son los siguientes:

- Disponer de al menos una página, aunque es preferible crear varias páginas más sencillas. También se recomienda dividir la funcionalidad en componentes más pequeños, si las páginas son muy complejas.
    - Para estructurar la aplicación, en páginas y componentes, usaremos NUXT, que también proporciona una base para incorporar el resto de librerías necesarias, un servidor para las pruebas locales, etc.
- Poder introducir gastos, con una fecha, cantidad, y categoría.

    - La fecha debería ser correcta, para lo cual se recomienda usar algún mecanismo como elegir una fecha de un calendario, como por ejemplo el componente DatePicker
    - Para las categorías se recomienda usar un desplegable con una serie de categorías predefinidas. Puedes usar por ejemplo el componente Select.
- Todos los gastos introducidos se mostrarán en una tabla.
    - Para las tablas puedes usar el componente DataTable de vuetify
- Mostrar algún tipo de estadística o gráfico, que nos ayude a ver los gastos por categorías.
    - Puedes por ejemplo mostrar los gastos totales para una categoría entre dos fechas, o en el último mes, mostrar una gráfica con la evolución de los gastos para una categoría, etc.
    - Si quieres usar gráficas, puedes usar el componente Sparklines de Vuetify.
- Separar el modelo de datos del resto de la aplicación, lo cual facilitará que en un futuro la aplicación pueda guardar los datos en la nube o en otro tipo de almacenamiento persistente. Las vistas pueden seguir teniendo sus datos propios, por ejemplo datos temporales, sin embargo los datos más importantes, que no queremos perder, se deberían gestionar aparte. Para esto se recomienda utilizar Vuex, que es el modelo de datos que incorpora Vue.

Además de los requisitos mínimos, también son deseables los siguientes:

- Persistencia. En una aplicación como esta, normalmente guardaríamos los datos en un servicio cloud. Para simplificar el desarrollo, utilizaremos en su lugar el almacenamiento que proporciona el propio navegador. Esto lo haremos desde las acciones de Vuex, de forma que no sea necesario modificar los componentes o las páginas de la aplicación. Existen multitud de librerías para facilitar esta tarea, y la principal limitación es que no podremos crear una aplicación universal, ya que solo podremos usar los datos desde el lado cliente (el navegador).
- Alojar la aplicación en la nube, por ejemplo en Github pages. Puedes encontrar la información sobre cómo hacerlo con nuxt en : https://nuxtjs.org/faq/github-pages
- Una vez que has conseguido generar de forma estática la aplicación, y funciona correctamente en el proveedor de alojamiento estático, puedes convertirla en una PWA. La forma recomendada de hacerlo es con el módulo nuxt-pwa. Prueba a instalarla en distintos dispositivos.
    - Importante: No la instales si la ejecutas en localhost, solo cuando accedas a la versión alojada en la nube, ya que se asocia la dirección con la aplicación instalada, y te podría dar problemas si pruebas otras aplicaciones en localhost.

Desarrollo

Para comenzar el proyecto partiremos de una plantilla, que nos permitirá tener una aplicación básica que luego iremos modificando. En esta fase se recomienda desarrollar de forma

iterativa, con pequeños cambios, que se prueban continuamente. Nuxt incorpora un mecanismo que nos permite ver los cambios en la aplicación conforme los hacemos en el código, compilando de nuevo la aplicación cada vez que detecte cambios en un archivo.

La plantilla la podemos crear con el comando: npx create-nuxt-app <project-name>

Al ejecutar este comando, nos hará una serie de preguntas, donde seleccionaremos como queremos que sea nuestra aplicación. Elegiremos el nombre del proyecto, el lenguaje JavaScript, el gestor de paquetes NPM, el framework de componentes UI Vuetify.js, el módulo PWA, Prettier (para formatear el código), renderizado Single Page App (SPA), hosting estático, y jsconfig.json. Para el resto de opciones podemos dejar los valores sugeridos.

Podemos probar la aplicación ejecutando el comando "npm run dev". Esto creará un servidor para los archivos estáticos de nuestra aplicación, en http://localhost:3000/. También se quedará escuchando cambios en los archivos para actualizar la aplicación y poder verlos en el navegador sin tener que refrescar la página.

Plan de trabajo

Durante las clases de laboratorio se explicarán los detalles para abordar cada una de las fases de desarrollo de la aplicación, por lo que en la medida de lo posible se deberían haber completado las fases anteriores antes de cada clase.

1. La primera semana se instalará todo lo necesario para el desarrollo, se creará la plantilla de la aplicación, y se desarrollará una página con dos componentes. El primer componente será un pequeño formulario que permitirá añadir gastos, y el segundo componente es el que mostrará la tabla con los gastos. La página será la encargada de recibir un nuevo gasto introducido con el primer componente, y añadirlo a un array con todos los gastos, que se mostrará en el segundo componente. En esta primera fase el array con todos los gastos estará en la página, y más adelante lo migraremos al modelo de datos de la aplicación.
2. Crearemos un módulo Vuex, en el directorio /store, que será el encargado de guardar la lista de gastos, y tendrá los métodos (mutaciones), que modifican la lista, y los métodos (acciones) que sincronizan los datos con el sistema que se utilice para almacenarlos de forma persistente. Puesto que la persistencia la implementaremos más adelante, las acciones simplemente llamarán a las mutaciones para modificar los datos. Crearemos también la página para mostrar las estadísticas de gastos, y ambas páginas, esta y la que habíamos creado la primera semana, se comunicarán con Vuex para usar el mismo modelo de datos (generales) en toda la aplicación.
3. Una vez nos hemos asegurado que funcionan los requisitos mínimos, añadiremos el resto de requisitos adicionales.
    a. Modificaremos el modelo general de datos (Vuex) para que las acciones, que antes simplemente llamaban a las mutaciones para modificar los datos, guarden de forma persistente los cambios en el almacenamiento que proporciona el browser. Al iniciar la aplicación se deben leer todos los datos almacenados.

   b. Compilaremos la versión de producción de la aplicación, usando npm run generate, y añadiremos el script en package.json que publica la aplicación generada en nuestro proyecto GitHub, en la rama hg-pages, para que se pueda acceder desde Github Pages.
   c. Terminaremos de configurar el módulo nuxt-pwa para añadir el icono de nuestra aplicación, y probaremos a instalarla desde varios dispositivos, conectando a la versión alojada en Github Pages.
4. Si hemos seguido el plan previsto, podremos dedicar la última semana a personalizar nuestra aplicación, completar las pruebas finales, y preparar la memoria.

Duración y entrega de la práctica

La duración de esta práctica es de 4 semanas. La práctica se entregará el 17 de diciembre.

Crea un documento PDF incluyendo pantallazos con distintos ejemplos de ejecución, de los pasos principales para ejecutar y probar la práctica, comentando los problemas encontrando y cómo se han solucionado.

La entrega se realizará en un único fichero zip, que incluya el pdf de la memoria, y todos los archivos necesarios para generar la aplicación y ejecutar la práctica. No se incluirán archivos innecesarios, como los que se crean automáticamente en los directorios dist, .nuxt, o node_modules.