

Desarrollo Web en Entorno Cliente

UD 15. Ampliando Vue

2. Quasar, Firebase

(Cloud Firestore) y Nuxt

Actualizado Enero 2021

Licencia



Reconocimiento – NoComercial - CompartirIgual (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:



Importante



Atención




Interesante

ÍNDICE DE CONTENIDO

1. Introducción	3
2. Framework Quasar	3
2.1. Empezando con Quasar	3
2.2. Probando Quasar desde un editor online	3
2.3. Instalando Quasar	4
2.4. Creando un proyecto con Quasar	4
2.5. Escribiendo código Vue con Quasar	5
2.6. Ejemplo código Quasar: ValenBisi Checker	7
3. AMPLIACIÓN: Vue + Firebase (Cloud Firestore) + Vuefire	8
3.1. Firebase (Cloud Firestore) y Vuefire	8
3.2. Ejemplo: Lista de tareas con Cloud Firestore y Vuefire	8
4. AMPLIACIÓN: NodeJs + Express + Nuxt (Vue)	9
4.1. Ejemplo de aplicación	9
4.2. Bases del ejemplo visto	10
5. Enlaces interesantes	10
6. Bibliografía	10
7. Autores (en orden alfabético)	11

UD15. AMPLIANDO VUE 2. QUASAR, FIREBASE Y NUXT.

1. INTRODUCCIÓN

 **Importante:** esta unidad trata las herramientas expuestas por encima, ya que su fin es el de difundir su existencia y permitir los primeros pasos con ellas y a efectos académicos sólo se valorarán a ese nivel. No obstante, animo a todos a ampliar conocimientos de forma autodidacta.

En esta unidad vamos a presentar mediante ejemplos prácticos otras tecnologías que puede complementar a nuestras aplicaciones Vue. Entre ellas vamos a ver:

- Ejemplo de aplicación que utiliza Vue y la base de datos de Google “Firebase” <https://firebase.google.com> uniéndolos mediante la biblioteca “Vuefire” <https://github.com/vuejs/vuefire>
- Uso del Framework de Vue llamado Quasar <https://quasar-framework.org/> para realizar aplicaciones cliente Vue y poder exportarlas directamente a SPA, PWA, Electron y Cordova.
- Ejemplo de aplicación servidor usando NodeJS + Express + Nuxt (Vue).
 - Nuxt <https://nuxtjs.org/> básicamente tiene un funcionamiento similar a Vue en cuanto al desarrollo de la aplicación, pero el cliente solo recibe los cambios en el renderizado, trasladando la carga computacional al servidor (y protegiendo el acceso al código fuente)
 - Express: <http://expressjs.com/> incluye un conjunto de funciones y métodos para desarrollar aplicaciones Web en NodeJs.

2. FRAMEWORK QUASAR

2.1 Empezando con Quasar

Quasar Framework <https://quasar.dev/> es un framework para crear aplicaciones usando Vue y otras bibliotecas de NodeJS y desplegarla de inmediato en 4 tipos de aplicaciones:

- **SPA:** Single Page Application (Aplicación de una página)
- **PWA:** Progressive Web Application (Similar a la anterior, pero con disponibilidad offline).
- **Cordova:** Aplicaciones móviles, estudiada en unidades anteriores.
- **Electron:** Aplicaciones de escritorio, estudiadas en unidades anteriores.

Esto nos facilita el desarrollo de aplicaciones que queramos desplegar en distintos tipos de lugares (Web, móvil, escritorio, etc.).

Además **Quasar incluye componentes Vue** propios (los que empiezan por “q-algo”, tipo “q-btn” o “q-select”) La lista completa está disponible en <https://quasar.dev/vue-components/>

2.2 Probando Quasar desde un editor online

Es posible hacer pruebas utilizando Quasar desde editores online. Hay varios ejemplos de los cuales se puede partir en la url <https://quasar.dev/start/playground>

El enlace anterior nos lleva a los siguientes ejemplos:

- <https://jsfiddle.quasar.dev/>

- <https://codepen.quasar.dev/>
- <https://codesandbox.quasar.dev/>

2.3 Instalando Quasar

Si deseamos instalar Quasar, debemos seguir los pasos indicados en <https://quasar.dev/quasar-cli/installation>

Si tenemos NodeJS en marcha, simplemente con “npm install -g @quasar/cli” instalaremos Quasar CLI que nos facilitará la creación de proyectos Quasar.

Si además queremos generar aplicaciones para móviles, para disponer de lo necesario lo más práctico y rápido es primero instalar Android Studio (que auto configura el Android SDK) y tras ello instalar Cordova con “npm install -g cordova”.

Android studio lo podéis descargar en <https://developer.android.com/studio>

2.4 Creando un proyecto con Quasar

Una vez instalado podemos crear un proyecto Quasar con “**quasar create <nombre_directorio>**”.

Con dicho comando, os creará un proyecto Quasar con código de ejemplo basado en esta plantilla <https://github.com/quasarframework/quasar-starter-kit>

Tras ello os pedirá varios datos (nombre del proyecto, autor, nombre Cordova, si utilizas algunas herramientas extra , etc.).

Una vez contestadas las preguntas, te creará la estructura necesaria en el directorio indicado.

Con el proyecto ya creado, podrás probar/construir tu aplicación usando comandos como los que aparecen en <https://quasar.dev/quasar-cli/build-commands>.

El detalle de todos los comandos soportados por Quasar CLI está disponible en la siguiente dirección <https://quasar.dev/quasar-cli/commands-list>

A continuación ejemplos de comandos para probar o construir una aplicación:

- “**quasar dev -m spa**”
 - Lanzar Quasar en modo desarrollo en plataforma SPA.
- “**quasar dev -m electron**”
 - Lanzar Quasar en modo desarrollo y para plataforma Electron.
- “**quasar dev -m cordova -T android**”
 - Lanza Quasar en modo desarrollo y para Android.
- “**quasar build -m pwa**”
 - Lanza Quasar para crear la aplicación de tipo PWA en modo producción.

Veamos un ejemplo concreto: con el siguiente comando “**quasar dev -m spa**”, creamos una SPA en modo desarrollo. Nos aparece la siguiente información en consola

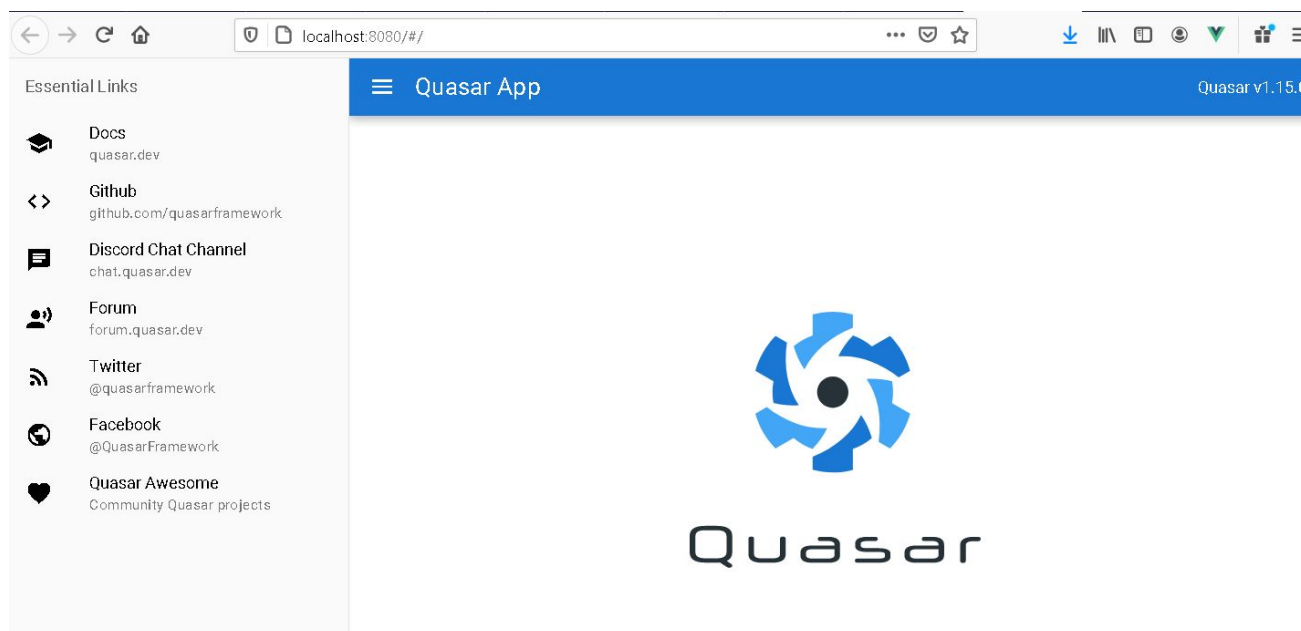
```
DONE Compiled successfully in 6420ms

N App dir..... C:\Users\Sergi\pruebaquasar
  App URL..... http://localhost:8080
  Dev mode..... spa
  Pkg quasar..... v1.15.0
  Pkg @quasar/app... v2.1.14
  Transpiled JS..... yes (Babel)

i @wds@: Project is running at http://0.0.0.0:8080/
i @wds@: webpack output is served from
App - Opening default browser at http://localhost:8080
```

Así tenemos cargada en modo “dev” nuestra aplicación Quasar, de forma que cada cambio que hagamos en el código, se recargará automáticamente.

Observamos visualmente el resultado de la siguiente forma:



2.5 Escribiendo código Vue con Quasar

En el siguiente enlace tenéis el detalle de la estructura de directorios que presenta Quasar <https://quasar.dev/quasar-cli/directory-structure>

Dentro de este árbol de directorios exista la carpeta “src” (la que contendrá vuestro código).

También conforme vayamos generando código para distintas plataformas se crearán las carpetas con el código asociado “src-electron”, “src-cordova”, etc. por si queremos exportarlo como proyectos independientes, aunque a la hora de trabajar con Quasar deberemos hacerlo siempre sobre “src”.

De manera sencilla, la forma más típica de escribir código sencillo dentro de la carpeta “src”:

- Modificar “**router/routes.js**” para incluir las rutas nuevas y asociarlas a componentes Vue y ficheros “.vue” que se requieran
- Modificar el directorio “**layouts**” donde se incluirán los “layout” creados para nuestro proyecto que irán cargando las distintas páginas Vue.
 - Mas información en <https://quasar.dev/layout/layout>
- Modificar el directorio “**pages**” donde se incluirán los componentes “.vue” que contendrán las páginas .
- Otros directorios interesantes: “**assets**” para recursos (imágenes, sonidos), “**components**” para componentes Vue, “**css**” para ficheros css, etc.

La página inicial Vue (En el raíz de “src” llamada “**App.vue**”) tiene la siguiente estructura:

```
<template>
  <div id="q-app">
    <router-view />
  </div>
</template>
<script>
export default {
  name: 'App'
}
</script>
```

Donde hace referencia que en el div con id="q-app" (definido en el fichero “**index.template.html**”) de la raíz de “src”. En el div se mostrará lo que le indique el componente **<router-view>**. Recordamos el uso de router en <https://router.vuejs.org/api/#router>

La configuración del funcionamiento de router en nuestra aplicación está definida en el fichero “**router/routes.js**” el cual tiene inicialmente una estructura como esta, que indica que se carga un componente “**MainLayout.vue**” y según la ruta, en su interior se carga el componente hijo indicado (en el ejemplo solo esta la regla para “Index.vue”). Si no se encuentra nada, se carga el componente “**Error 404.vue**”.

```
const routes = [ {
  path: '/',
  component: () => import('layouts/MainLayout.vue'),
  children: [
    { path: '', component: () => import('pages/Index.vue') } ]
}, {
  path: '*',
  component: () => import('pages/Error404.vue')
}]
export default routes
```

Vamos a ver como es un ejemplo de componente .vue con una página. Observamos “Index.vue”.

```
<template>
  <q-page class="flex flex-center">
    
  </q-page>
</template>

<script>
export default {
  name: 'PageIndex'
}
</script>
```

En este ejemplo observamos en **<template>** el código que se cargará en **<router-view>** cuando se cargue este componente.

2.6 Ejemplo código Quasar: ValenBisi Checker

En la unidad se incluye un ejemplo completo comentado de una aplicación Quasar llamado “ValenBisi Checker”. Este ejemplo, conecta con la información que proporciona el Ayuntamiento de Valencia indicando el estado de las estaciones de alquiler de bicicletas “ValenBisi”.

La información la obtiene de <http://mapas.valencia.es/lanzadera/opendata/Valenbisi/JSON>

El código fuente del ejemplo está disponible en <https://github.com/sergarb1/Quasar-Examples> como “Ejemplo01-Quasar-ValenbisiChecker” y en el aula virtual. Podéis ver la aplicación (en su versión web) en funcionamiento en <https://apuntesfpinformatica.es/DWEC/valenbisi-checker/>

La configuración de este ejemplo incluye los siguientes elementos:

- **Lintor “ESLint” deshabilitado** <https://quasar.dev/quasar-cli/linter#Disabling-Linter>
- **Módulo “Axios”** para realizar peticiones AJAX a la web de Valenbisi.

Los ficheros importantes a revisar del ejemplo son los siguientes:

- **Directorio “assets”**: incluye imágenes usadas en la aplicación.
- **Fichero “router/routes.js”**: incluye las rutas de la aplicación, indicando cual es el layout principal y que rutas indican que elementos deben cargarse como hijos.
- **Fichero “layouts/LayoutPrincipal.vue”**: fichero que define el layout principal. Carga entre otras cosas la barra superior y lateral (esta última con un componente) y tiene el componente **<router-view>** donde se irán cargando las distintas páginas según la ruta utilizada.
- **Fichero “components/EstacionesLateral.vue”**: componente en formato SFC. Este es registrado automáticamente por Quasar(al estar en ese directorio lo auto-registra) y lo podemos usar en la aplicación. El componente enlaces que vemos en el lateral y es utilizado dentro de “layouts/LayoutPrincipal.vue”.
- **Directorio “pages”**: contiene páginas, definidas con ficheros “.vue” en formato SFC.

3. AMPLIACIÓN: VUE + FIREBASE (CLOUD FIRESTORE) + VUEFIRE

3.1 Firebase (Cloud Firestore) y Vuefire

Firebase es una plataforma de apoyo al desarrollo Web en la nube creada por Google. Su dirección de acceso es <https://firebase.google.com>. Entre otras funciones, Firebase facilita el alojamiento de base de datos no relacional (que es lo que usaremos en nuestra aplicación) y cuya característica más importante es que permite la sincronización en tiempo real. Además Firebase ofrece otras funciones (login, estadísticas, etc.). Más información en wikipedia <https://es.wikipedia.org/wiki/Firebase>. Aquí tenéis información de como añadir Firebase a un proyecto Javascript cualquiera <https://firebase.google.com/docs/web/setup?hl=es-419>

Firebase dentro de sus herramientas incluye una muy interesante llamada “Cloud Firestore”. Esta herramienta nos permite tener una base de datos no relacional y reactiva que sincroniza datos con todos los clientes conectados en tiempo real. Mas información en <https://firebase.google.com/docs/firestore>

En esta unidad presentamos un ejemplo de como crear una aplicación Vue que utilice la herramienta Cloud Firestore de Firebase, con la ayuda del plugin Vuefire. Vuefire es un plugin que nos facilita la integración de Firebase con Vue manteniendo la reactividad.

El plugin está disponible en <https://github.com/vuejs/vuefire> y la guía de uso en <https://vuefire.vuejs.org/>.

Para ampliar información en este tutorial de integrar Cloud Firestore y Vue con Vuefire <https://www.positronx.io/vue-firebase-tutorial-integrate-cloud-firestore-with-vuefire/>

3.2 Ejemplo: Lista de tareas con Cloud Firestore y Vuefire

El ejemplo presentado es una modificación de la lista de tareas que creamos con Vue en la unidad anterior. Este nos presenta una lista de tareas, pero con la información guardada y sincronizada en la nube usando Cloud Firestore. Además, si mientras usas la lista de tareas, alguien realiza algún cambio (añade una tarea, elimina una tarea, etc.), la lista de tareas se actualizará de forma automática en tu dispositivo (podéis hacer la prueba abriendo dos instancias del navegador).

Tenéis el ejemplo de la “Lista de tareas” con Cloud Firestore en funcionamiento disponible en <https://apuntesfpinformatica.es/DWEC/lista-tareas-firebase/>

En el aula virtual y en <https://github.com/sergarb1/Vuejs2-Examples/> con el nombre de “Ejemplo06BIS-lista-de-tareas-firebase”, tenéis disponible también el código fuente.

Para hacer funcionar ese código fuente deberéis de configurar vuestra propia cuenta en Firebase (Cloud Firestore) y colocar vuestros datos en el fichero “**firestore.js**”.

4. AMPLIACIÓN: NODEJS + EXPRESS + NUXT (VUE)

En este punto observaremos un ejemplo de aplicación servidor usando NodeJS + Express + Nuxt

(Vue). Procedemos a definir estos elementos:

- **Nuxt** <https://nuxtjs.org> básicamente tiene un funcionamiento similar a Vue en cuanto al desarrollo de la aplicación, pero el cliente solo recibe los cambios en el renderizado, trasladando la carga computacional al servidor (y protegiendo el acceso al código fuente).
- **Express** <http://expressjs.com/> incluye un conjunto de funciones y métodos para desarrollar aplicaciones Web en NodeJS. En este ejemplo se utiliza como apoyo.

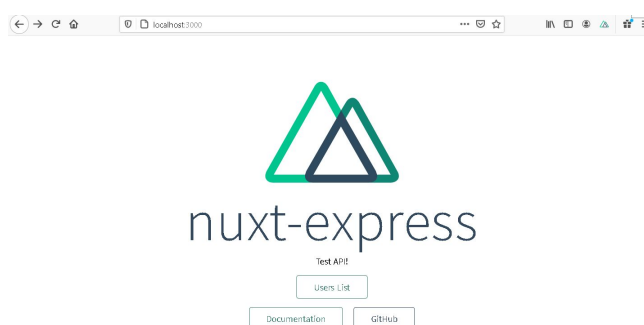
4.1 Ejemplo de aplicación

Para comprender los fundamentos básicos de Express y Nuxt, veremos un simple ejemplo basado en la plantilla Nuxt que nos crea la herramienta Vue-CLI. Dicho ejemplo está disponible en <https://github.com/sergarb1/Vuejs2-Examples/> como “Ejemplo11-miPrimerNuxt” y en el aula virtual. Para poner en marcha este ejemplo realizaremos las siguientes acciones:

- En primer lugar instalamos Vue-CLI “**npm install -g @vue/cli**” y tras ello “**npm install -g @vue/cli-init**” para instalar el addon CLI-init.
- Una vez instalado, creamos un proyecto Nuxt con Vue-CLI ejecutando “**vue init nuxt/express miPrimerNuxt**”, con lo que crea un proyecto a partir de la plantilla “**nuxt/express**” en el directorio “**miPrimerNuxt**”.
 - Este proyecto se basa en la plantilla de <https://github.com/ndarilek/express>
- Para instalar las dependencias dentro del directorio creado, usamos “**npm install**”.
- Para lanzar y depurar la aplicación Nuxt utilizamos los siguientes comandos:
 - “**npm run dev**” si queremos lanzarlo y testearlo en modo desarrollo.
 - “**npm run build**” si queremos generar el producto en modo producción.
 - “**npm start**” si queremos lanzarlo y testearlo en modo producción.

El ejemplo planteado lo lanzaremos con “**npm run dev**”. Si todo es correcto, se lanzará el proyecto para visualizarlo en **http://localhost:3000**

Aquí un ejemplo de lo que veremos.



Si observais el código HTML que recibe el cliente, veréis que no hay rastro de Vue en el cliente, ya que las operaciones de la programación reactiva son realizadas en el servidor y el cliente únicamente recibe que renderizar. Esto traslada la carga computacional al servidor, liberando a la máquina cliente. Para más información sobre Nuxt <https://github.com/i62navpm/Tutorial-Nuxt>

4.2 Bases del ejemplo visto

Dado la complejidad del ejemplo y que queremos solo explicar que es Nuxt de manera divulgativa, vemos de manera muy básica que se realiza en el código de ejemplo.

1. En “**api/index.js**” se define la aplicación Vue.

2. En el fichero **"api/routes/users.js"** se definen las constantes de los "username" que aparecen en la aplicación y se establecen las acciones a realizar según las rutas solicitadas.

```
/* GET user by ID. */
router.get('/users/:id', function (req, res, next) {
  const id = parseInt(req.params.id)
  if (id >= 0 && id < users.length) {
    res.json(users[id])
  } else {
    res.sendStatus(404)
  })
})
```

3. Si la ruta obtenida por get tiene el formato /users/0, el coge el valor "0" como valor del campo "id" en la orden **"const id = parseInt(req.params.id)"**.
4. La orden res (con sus funciones send, json, etc.) es para mostrar la respuesta HTML. En este caso con res.json, lo que hace es enviar una respuesta en formato JSON.

5. ENLACES INTERESANTES

Vuefire tutorial: Integrate Cloud Firestore con Vuefire

<https://www.positronx.io/vue-firebase-tutorial-integrate-cloud-firestore-with-vuefire/>

Vuefire tutorial: Crud application

https://www.youtube.com/watch?v=831zOI02Q_0

Quasar tutorial

<https://medium.com/@bhattjaldhi27/quasar-framework-tutorial-part-1-introduction-8628f7bf91e0>

<https://learncoding.map4b.com/2019/02/quasar-framework-for-vue-js/>

Nuxt Introduction by project

<https://www.youtube.com/watch?v=nteDXuqBfn0&t=158s>

Tutorial Nuxt

<https://github.com/i62navpm/Tutorial-Nuxt>

6. BIBLIOGRAFÍA

[1] Javascript Mozilla Developer <https://developer.mozilla.org/es/docs/Web/JavaScript>

[2] Javascript ES6 W3C https://www.w3schools.com/js/js_es6.asp

[3] Vue <https://es.vuejs.org/v2/guide/>

[4] Quasar <https://quasar.dev/>

[5] Nuxt <https://nuxtjs.org/>

7. AUTORES (EN ORDEN ALFABÉTICO)

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

- **García Barea, Sergi**