Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Frontend para la gestión de las Máquinas Virtuales en el IaaS de la ULL

Frontend for the management of Virtual Machines in the IaaS of the ULL

Míriam Núñez García

La Laguna, 2 de julio de 2021

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Estadística, Investigación Operativa y Computación de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"Frontend para la gestión de las Máquinas Virtuales en el IaaS de la ULL"

ha sido realizada bajo su dirección por Dña. **Míriam Núñez García**, con N.I.F. 43.841.018-M.

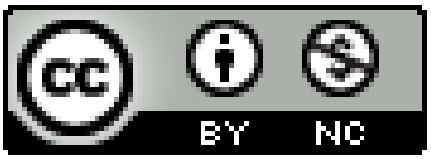
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 2 de julio de 2021

Agradecimientos

En primer lugar, agradecer mi tutor, Vicente Blaco Pérez, por su ayuda, orientación y apoyo durante la realización de este proyecto.

También agradecer a mi familia, amigos y compañeros que me han ayudado y apoyado a lo largo de, no solo de la realización de este trabajo, sino también a lo largo de estos años en la Universidad de La Laguna.

Licencia



Resumen

El objetivo de este proyecto es crear una aplicación web que permita la gestión de máquinas virtuales en el IaaS de la Universidad de La Laguna. En específico, diseñar el frontend de esta con una interfaz sencilla y funcional.

El IaaS de la ULL es un servicio proporcionado a la comunidad universitaria como soporte para la docencia, de forma que tanto profesores como alumnos pueden acceder a ella. Se utiliza la plataforma oVirt para albergar esta funcionalidad.

Actualmente, el IaaS se ha estado utilizando como se nombró anteriormente, principalmente para la docencia. Sin embargo, no solo se reduce a eso, sino que también llega a ser usado por los profesores, para sus investigaciones, o el personal de administración y servicios de la universidad, para cualquier prueba que necesiten. Además, es útil para aulas de ordenadores, ya que los alumnos no se ven condicionados por el hardware de los equipos físicos, que normalmente son menos potentes y están más desfasados que los de la nube.

Se desarrollará una aplicación web Single Page Application (SPA), basada en el stack MEVN (MongoDB, Express, Vue.js, Node.js) con un diseño centrado en Mobile First, aunque también dispondrá de una versión web.

Palabras clave: Aplicación web, Máquinas virtuales, IaaS, Frontend, oVirt, SPA, MEVN, MongoDB, Express, Vue, Node, Mobile First

Abstract

The goal of this project is to create a web application that allows the management of virtual machines in the IaaS of the University of La Laguna. Specifically, the design and implementation of the frontend of this application with a simple and functional interface.

The IaaS of the ULL is a service provided to the university community as a support for teaching, so that both teachers and students can access it. The oVirt platform is used to host this functionality.

Currently, IaaS has been used as named above, primarily for teaching. However, it is not only reduced to that, but it also comes to be used by the professors, for their research, or the administration and services staff of the university, for any test they need. In addition, it is useful for computer classrooms, since students are not conditioned by the hardware of the physical equipment, which is usually less powerful and more outdated than those in the cloud.

A Single Page Application (SPA) web application will be developed, based on the MEVN stack (MongoDB, Express, Vue.js, Node.js) with a design focused on Mobile First, although it will also have a web version.

Keywords: Web application, Virtual machines, IaaS, Frontend, oVirt, SPA, MEVN, MongoDB, Express, Vue, Node, Mobile First

Índice general

Índice de Figuras

Índice de Tablas

Índice de Códigos

Capítulo 1

Introducción

En este apartado se describirán los antecedentes y el estado actual del tema, las tecnologías utilizadas en el proyecto y la planificación del desarrollo del mismo.

1.1. Antecedentes y estado actual del tema

La infraestructura como servicio (*IaaS*) es una de las estrategias para ofrecer servicios en la nube, junto con Plataforma como Servicio (*PaaS*) y Software como Servicio (*SaaS*). Esta proporciona acceso a recursos informáticos situados en el entorno virtualizado que constituye la nube a través de una conexión pública.

En la Universidad de La Laguna se ha implementado una infraestructura de cloud computing para dar servicios de cómputo a PDI y alumnado. En el caso de la docencia se utilizan principalmente para ofrecer máquinas virtuales a los alumnos en los que pueden realizar sus prácticas. Dispone de una interfaz web[7] de gestión muy completa pero quizás complicada para las funciones básicas que los alumnos utilizan.

En este proyecto pretendemos desarrollar una interfaz más simple para las operaciones más habituales como crear desde cero máquinas virtuales, eliminarlas, encenderlas y apagarlas, reiniciarlas desde plantillas o hacer una búsqueda con diferentes filtros. Además, nuestra intención es añadir otras funcionalidades como la de compartir máquinas entre compañeros y permitir una búsqueda por estado de la máquina.

Actualmente, la comunidad universitaria también tiene acceso a una web[10] que permite ver información de tus máquinas virtuales (nombre, sistema operativo, estado y direcciones IP) sin necesidad de entrar a la plataforma de *oVirt*. Sin embargo, esta es solo de lectura. Por ello, queremos darle funcionalidad, para que si necesitamos modificar algo de esa información o, simplemente, encender una máquina no tengamos que acceder al portal.

Estos recursos tienen un acceso limitado a la red interna de la universidad, por

lo que, en el caso de que no se esté físicamente en ella, se tendrá que configurar una red privada virtual (*VPN*). Esta *VPN* permitirá disponer de conexión a la red de la ULL desde cualquier parte, y así poder acceder a tus máquinas virtuales.

1.2. Tecnologías usadas en el Proyecto

El proyecto consistirá en el desarrollo de una aplicación web *Single Page Application (SPA)*, basada en el *stack MEVN (MongoDB, Express, Vue.js, Node.js)* con un diseño centrado en *Mobile First*, aunque también dispondrá de una versión web. Además, el *frontend* se ha desarrollado con *Quasar*, un framework basado en *Vue*.

Las tecnologías que se han utilizado en el proyecto han sido:

- *Vue.js*[18]
- *Quasar*[11]
- *Vuex*[22]

Asimismo, se han hecho uso de varias herramientas como:

- *Smartmock*[12]
- *Git*[4]
- *Github*[5]
- *Travis CI*[17]
- *Surge*[13]

1.3. Planificación del Desarrollo

En la siguiente tabla (Tabla 1.1) encontramos las tareas principales en las que se divide el proyecto.

Además de estas, para poder llevar un orden en el día a día se han ido definiendo tareas más específicas que se han adaptado al rumbo que ha ido llevando el proyecto. Por ello, decimos que hemos utilizado las metodologías ágiles.

Para visualizar esas tareas he utilizado una aplicación llamada *Todoist*[16], un gestor de tareas multiplataforma, característica que aumentaba su facilidad de uso.

Tarea	Descripción
Prototipado	Diseño de la Interfaz de Usuario
Setup de la infraestructura	Preparación del repositorio y las herramientas
Desarrollo	Implementación de las funcionalidades
Testing e integración continua	Comprobar que la aplicación no tiene fallos
Despliegue	Despliegue de la aplicación

Tabla 1.1: Tareas generales del Desarrollo del Proyecto

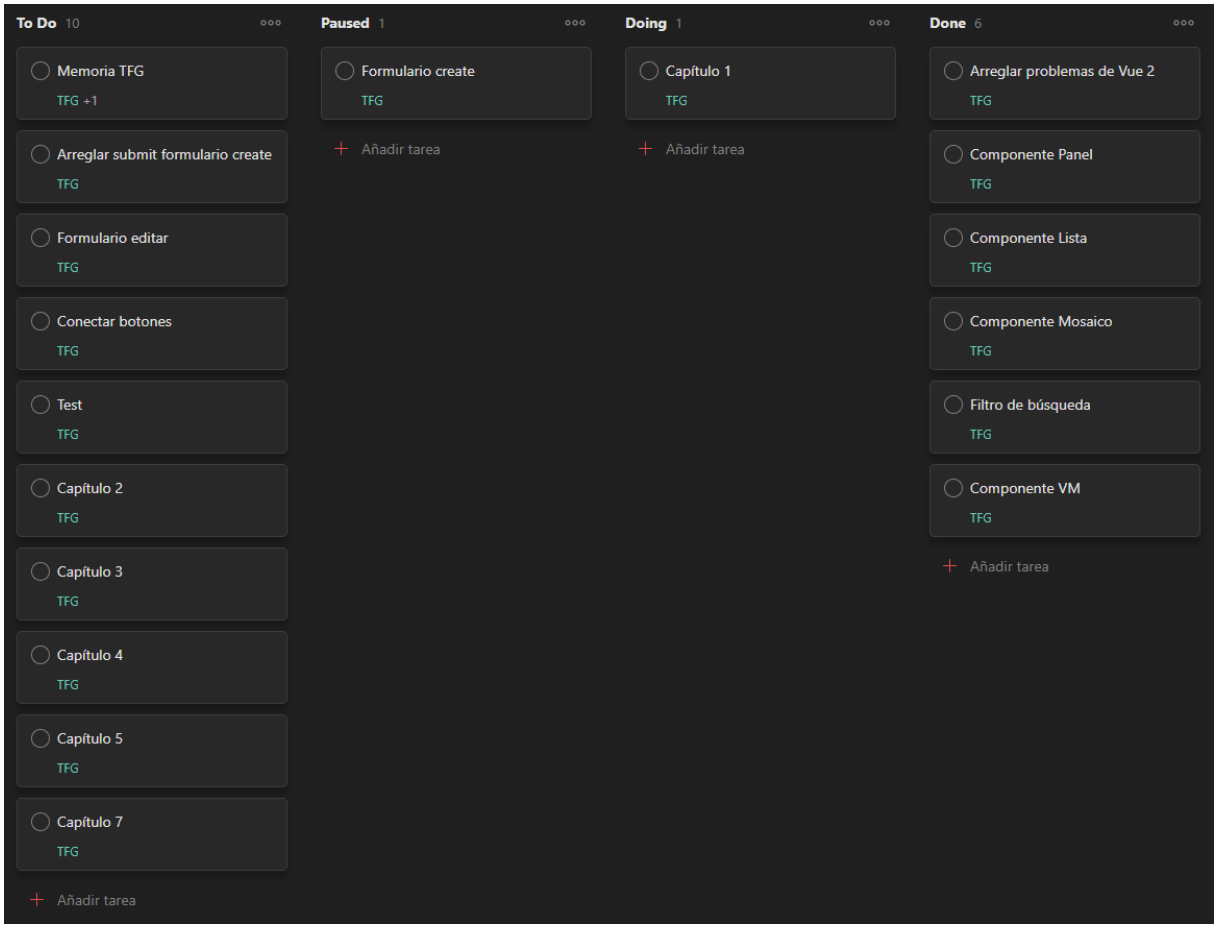


Figura 1.1: Subtareas del Desarrollo del Proyecto

Capítulo 2

Tecnologías y Herramientas

Tras las introducción, en esta sección detallaremos las herramientas y las tecnologías que se han utilizado durante el desarrollo de la aplicación, clasificadas diferenciando en si era herramientas generales y las específicas del *frontend*.

2.1. Herramientas de Desarrollo

2.1.1. Smartmock

Smartmock[12] es un servicio *Mock-As-A-Service* de virtualización de servicios alojados en la nube, es decir, una herramienta que te simula una *API* en la nube.

Como el objetivo proyecto es realizar el diseño del *frontend* no dispongo de *API*, por ello, es necesaria esta web, para poder hacer peticiones *HTTP* y obtener los resultados sin necesidad de un *backend*.

2.1.2. Git y Github

Para llevar el control de versiones de este proyecto se ha usado *Git*[4] y un repositorio remoto en la plataforma *Github*[5]. En él, se han creado dos ramas a parte de la rama principal (*main*), una para la parte del panel y el menú principal, y otra para todo lo que tenía relación con las máquinas virtuales en sí.

El repositorio de *Github* es *TFG-2021-Miriam-ULLIaaSFrontend*[15].

2.1.3. ESLint y Babel

Al crear un proyecto con *Vue*, te da la opción de elegir herramientas que te permitan ver la calidad del código, en este caso, se trata de *ESLint*[3] y de *Babel*[1].

ESLint es una herramienta que te permite analizar el código de forma que pueda encontrar patrones que no concuerden con las reglas que tiene predeter-

minadas o que se hayan configurado.

Por otro lado, *Babel* es un compilador de *JavaScript* utilizado para convertir el código *ECMAScript 2015* en versiones anteriores de *JavaScript* para que sea compatible con navegadores antiguos.

2.1.4. Travis CI

Travis CI[17] es un servicio de integración continua alojado. Con él es posible enlazar un proyecto de *Github* para comprobar los test realizados.

Como funciona con repositorios remotos, la forma de activarlo es hacer un *push* a la rama que hayas definido, subiendo así los cambios de tu repositorio local al remoto.

2.1.5. Surge

Surge[13] es una plataforma en la que es posible publicar páginas web estáticas. Esta orientada a desarrolladores *frontend*.

2.2. FrontEnd

2.2.1. Vue

Vue.js[18] es un framework de *JavaScript* de código abierto para la construcción de interfaces de usuario y aplicaciones de una sola página (*SPA*).

Como primera opción se había decidido utilizar la versión *Vue 3*, sin embargo, no todas las librerías o *frameworks* de *ui component* tiene soporte para esta o si lo tienen no son estables del todo. Por ello, finalmente, se decidió cambiar a la versión *Vue 2*.

2.2.2. Vue CLI

Para crear el proyecto ha sido necesario el uso de la interfaz de comandos de *Vue*, *Vue CLI*[19].

2.2.3. Vue Router

Como *Vue* se basa en crear *single page aplications (SPAs)*, necesita de una librería que haga el enrutamiento. Aquí es donde entra *Vue Router*[20], la librería con soporte oficial de *Vue* con la que puedes crear rutas y realizar un mapeo de estas con sus respectivas vistas.

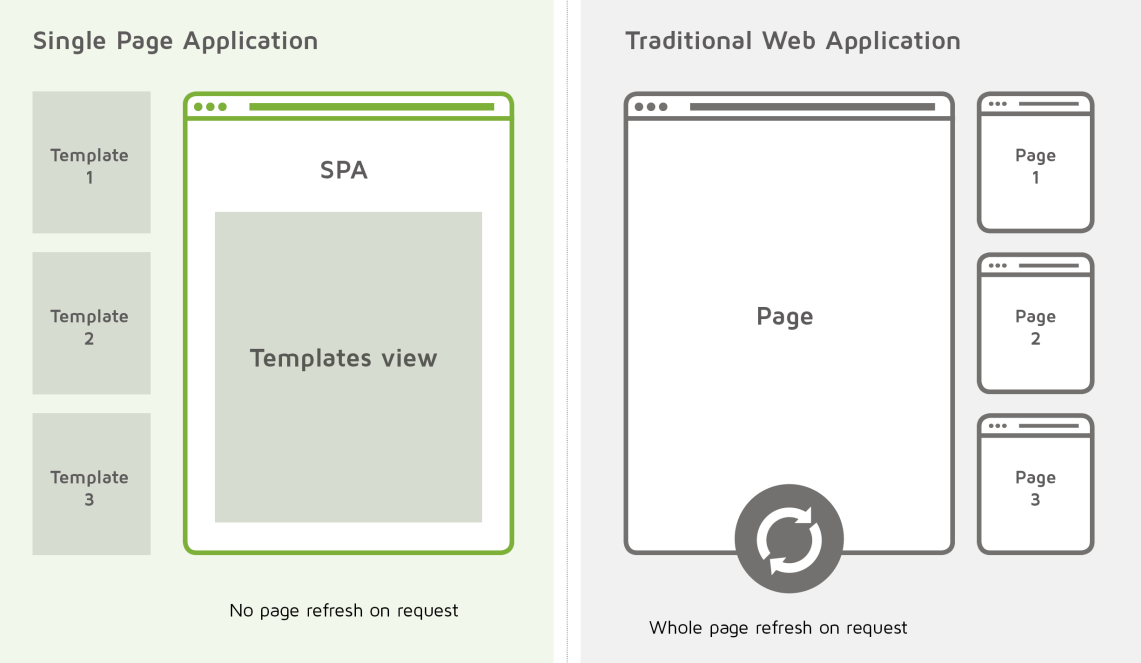


Figura 2.1: *Single Page Aplication* (Aplicación de una sola página)

2.2.4. Vuex

Flux es una arquitectura para el manejo y el flujo de datos, principalmente de *frontend*, en una aplicación web. Para poder usar este patrón y poder obtener los mismos datos desde todas las vistas del proyecto en *Vue* existe *Vuex*[22].

Vuex es una biblioteca de patrones de administración de estado. Como se comentó, sirve como almacén centralizado para todos las vistas y componentes de la aplicación.

Como vemos ver a continuación, este tiene cuatro grandes módulos:

- Estado (*State*): Estado o valor de las variables que están guardadas.
- Mutaciones (*Mutations*): Funciones que modifican el valor del *State*, tienen el inconveniente de que no se pueden llamar directamente.
- Acciones (*Actions*): Funciones necesarias para poder acceder a las *Mutations*.
- Adquiridores (*Getters*): Funciones que te permiten obtener el *State*.

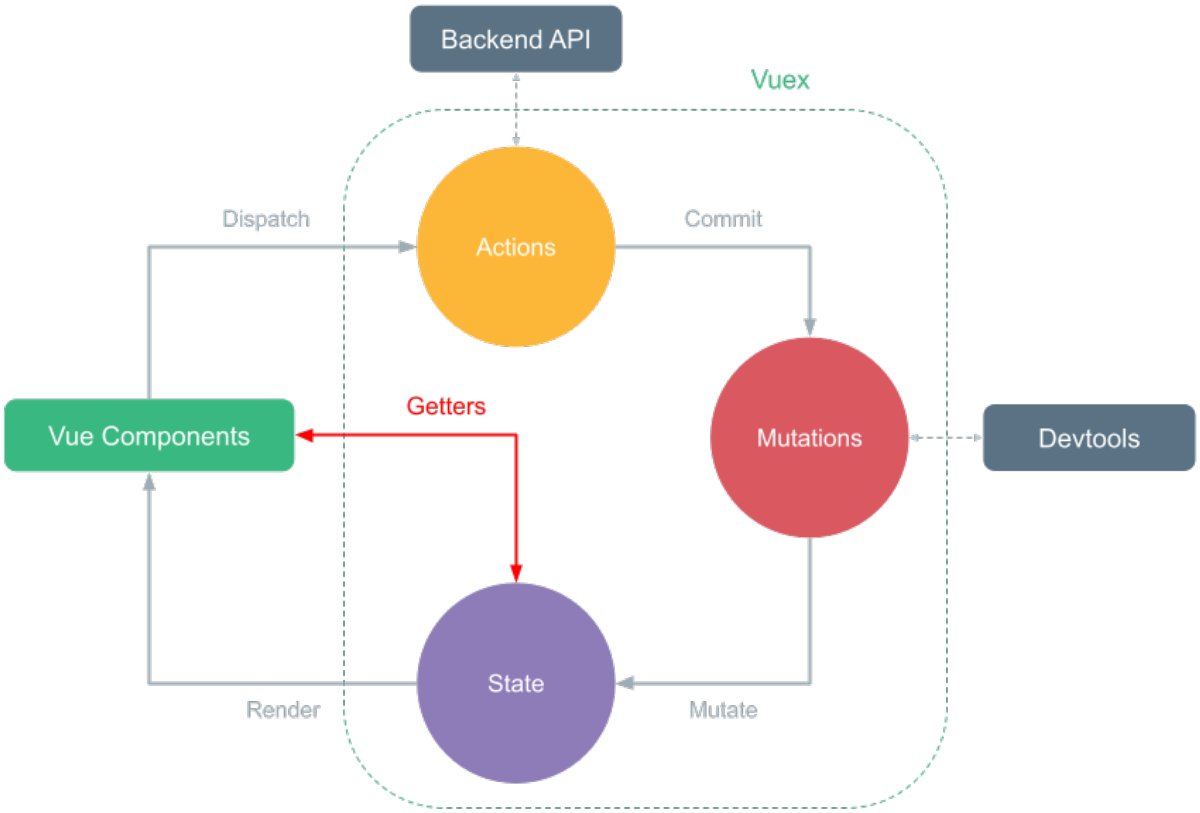


Figura 2.2: Esquema de *Vuex*

2.2.5. Quasar

Quasar[11] es un *framework* utilizado tanto en aplicaciones móviles como aplicaciones web basado de *Vue*, por lo que también se crean aplicaciones *SPA*.

Utilizamos este *framework* por ser uno de los *frameworks ui component* mas completos, con una gran variedad de componentes.

A pesar de que *Quasar* tiene una interfaz de comandos que simplifica crear proyecto, nos decidimos por este *framework* una vez el proyecto estaba iniciado, así que en vez de utilizarla lo añadimos a este.

Capítulo 3

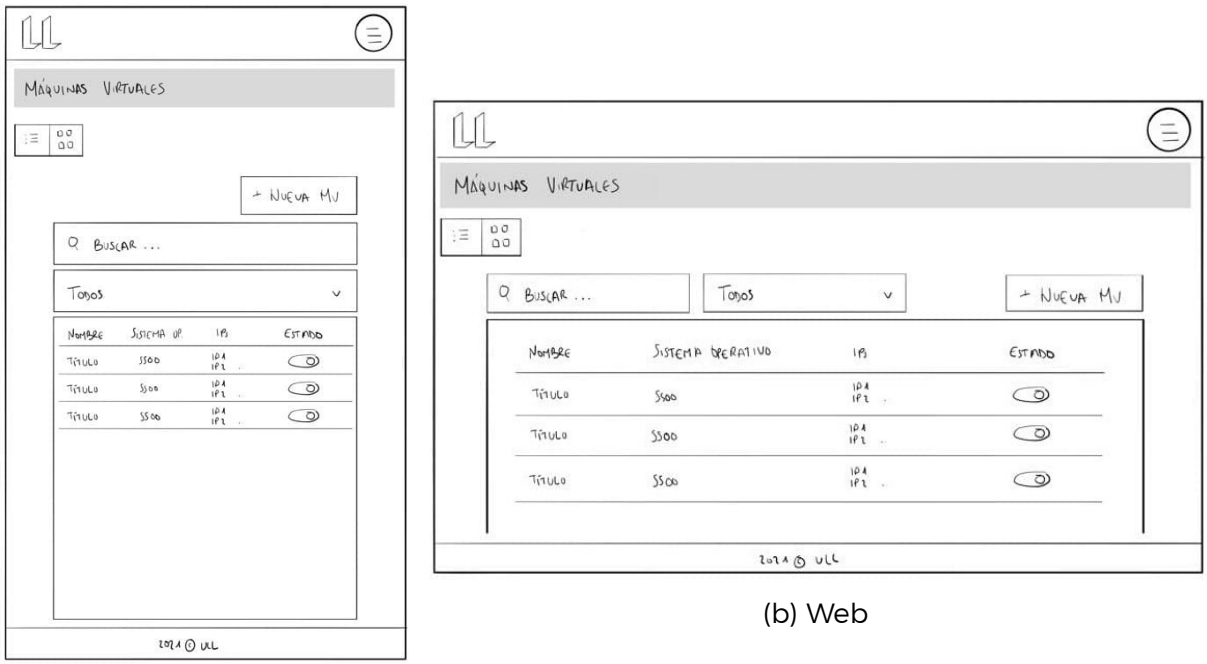
Desarrollo de la Aplicación

A continuación se procederá a explicar los pasos que se han seguido para el desarrollo del proyecto.

3.1. Prototipado

Antes de comenzar a desarrollar cualquier aplicación es necesario saber como quieres que estructure, para ello, se prototipa, es decir, se diseña la interfaz de usuario de esta.

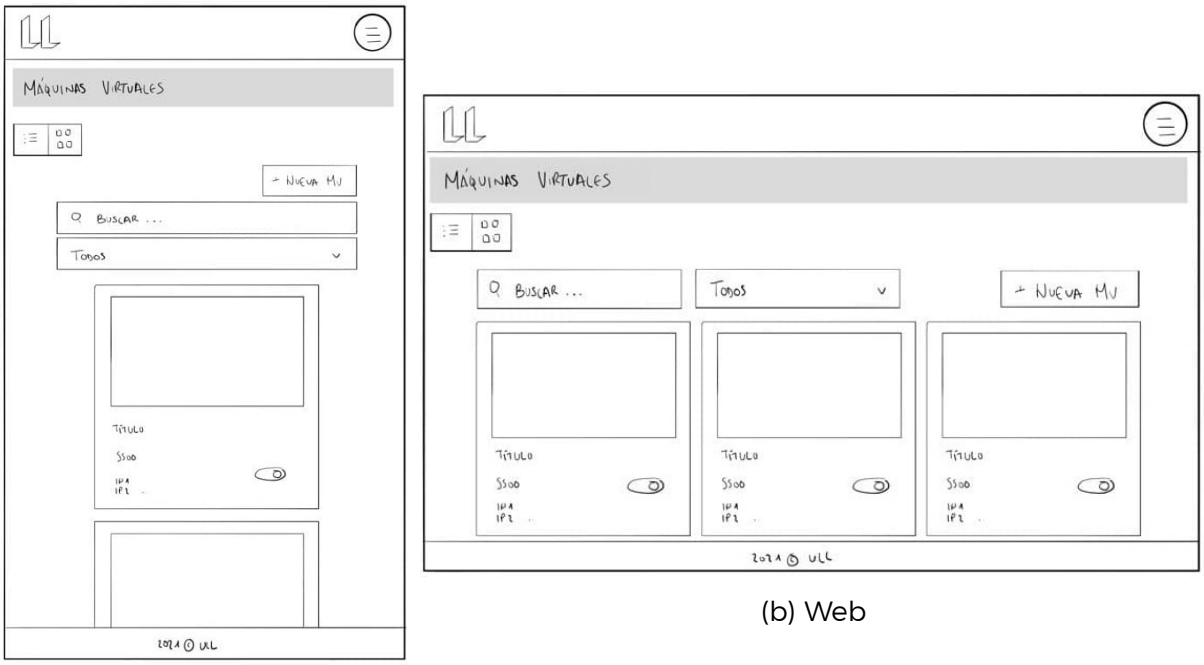
En las siguientes imágenes se pueden ver esos diseños, el de la izquierda para móviles y el de la derecha para web.



(a) Móvil

(b) Web

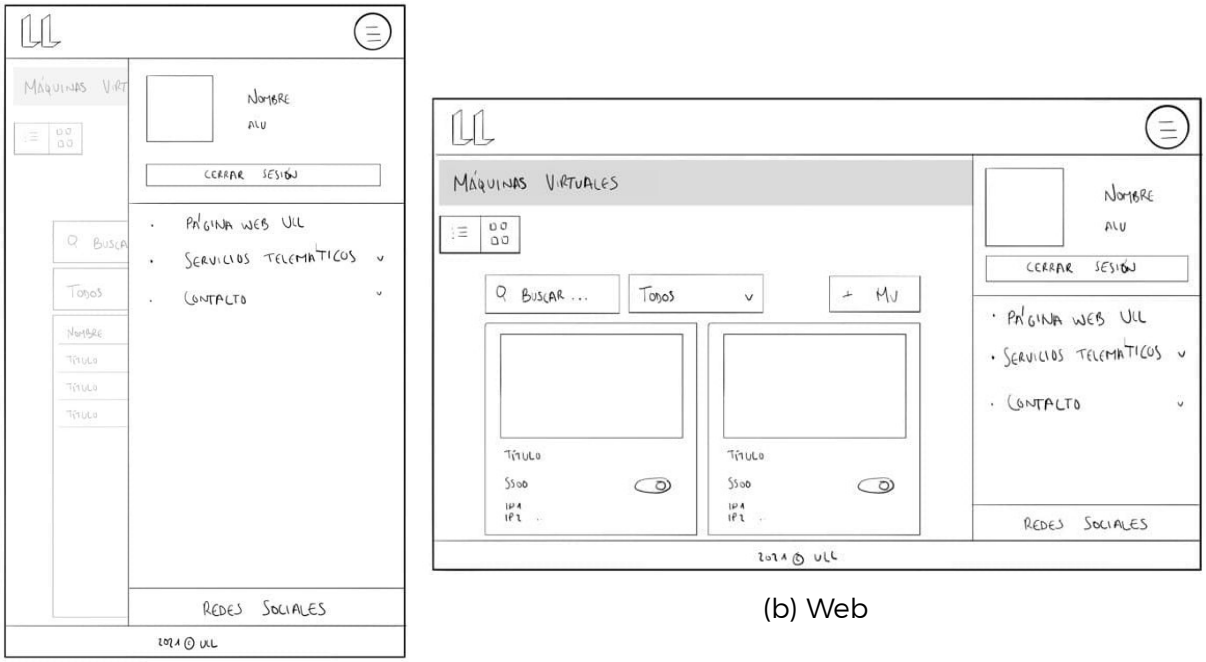
Figura 3.1: Prototipo del panel en vista de lista



(a) Móvil

(b) Web

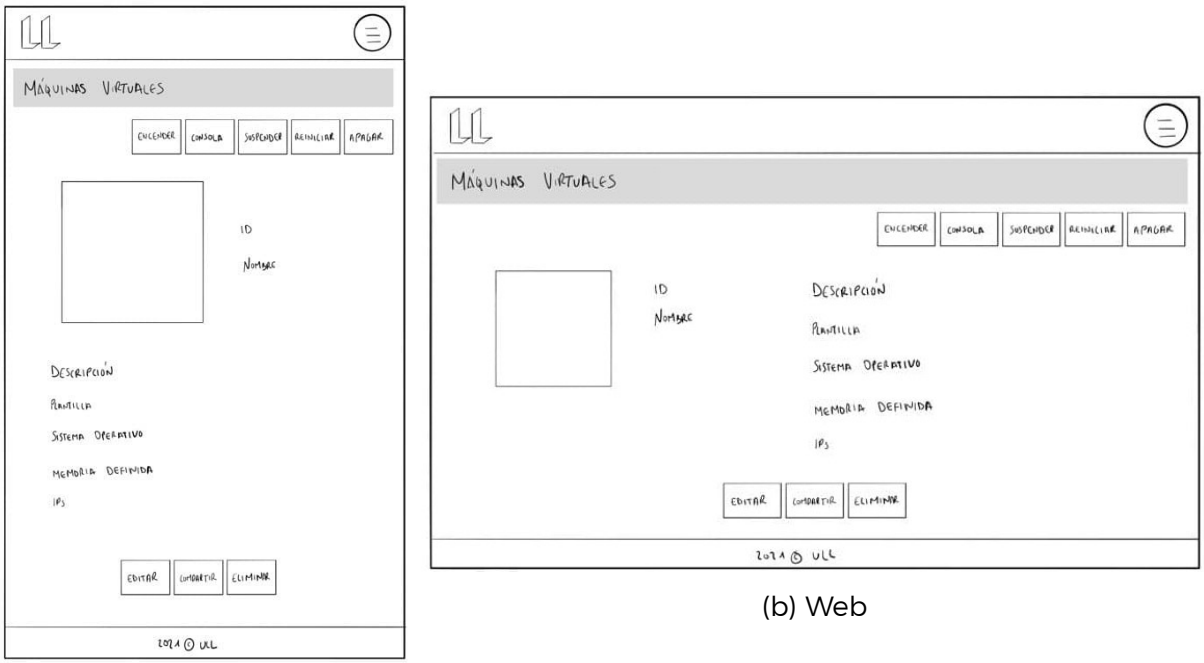
Figura 3.2: Prototipo del panel en vista de mosaico



(a) Móvil

(b) Web

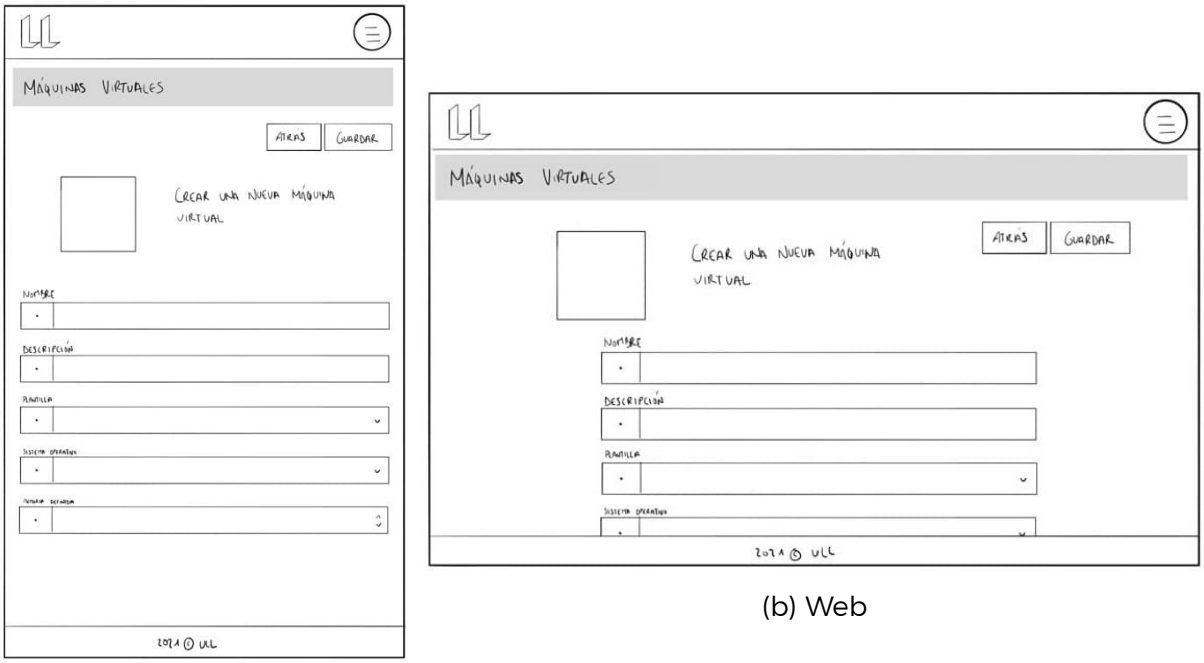
Figura 3.3: Prototipo del menú derecho



(a) Móvil

(b) Web

Figura 3.4: Prototipo de la vista de la máquina virtual



(a) Móvil

(b) Web

Figura 3.5: Prototipo del formulario

3.2. Setup de la Infraestructura

Como se había mencionado anteriormente, el proyecto se desarrolló en Vue, utilizando la herramienta Vue CLI.

```
1 npm install -g @vue/cli
2 vue create ull-iaas-frontend
```

Código 3.1: Crear proyecto

Al ejecutar lo anterior, se te abre una ventana en la que puedes elegir que el proyecto se cree con las dependencias que sepas que vas a utilizar de antemano.

En una primera instancia se eligieron las siguientes dependencias:

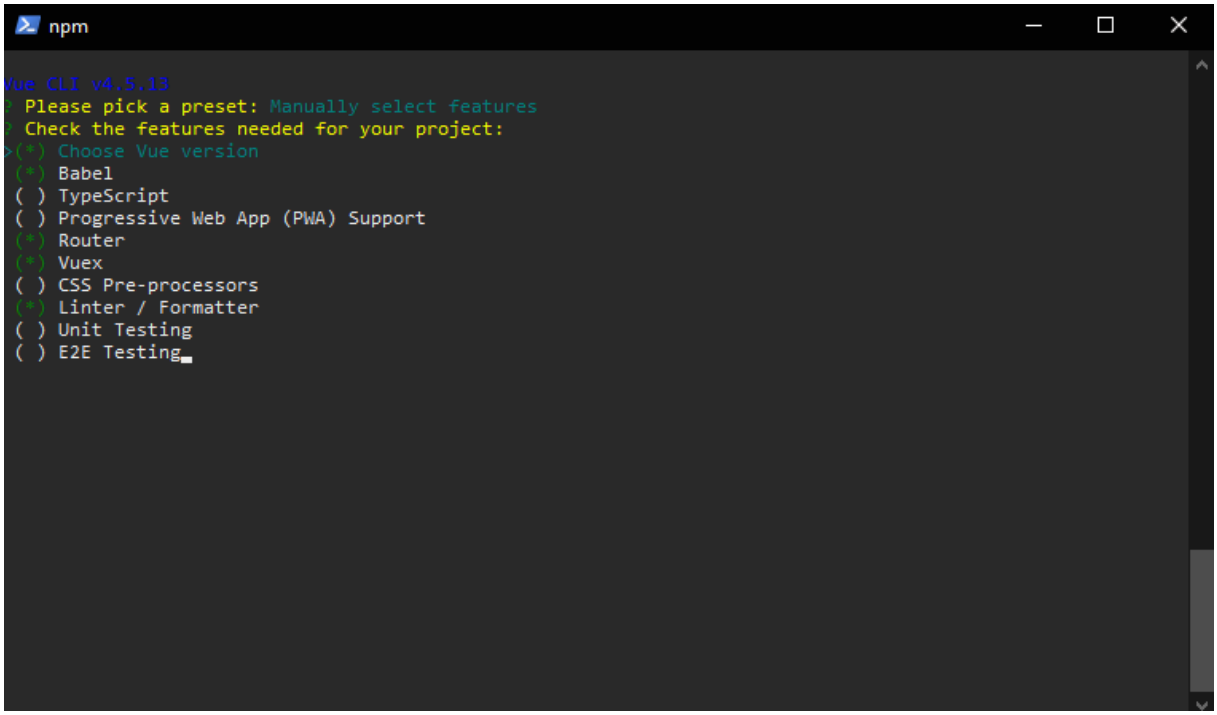


Figura 3.6: Selección de dependencias

Seguidamente, elegimos la versión de *Vue* que, como se nombró, es *Vue 2*.

Luego, añadimos la configuración *Standard* al *linter*, si queremos que se ejecute al guardar o al hacer un *commit* y si se guarda la configuración de este en un fichero dedicado o en el *package.json*.

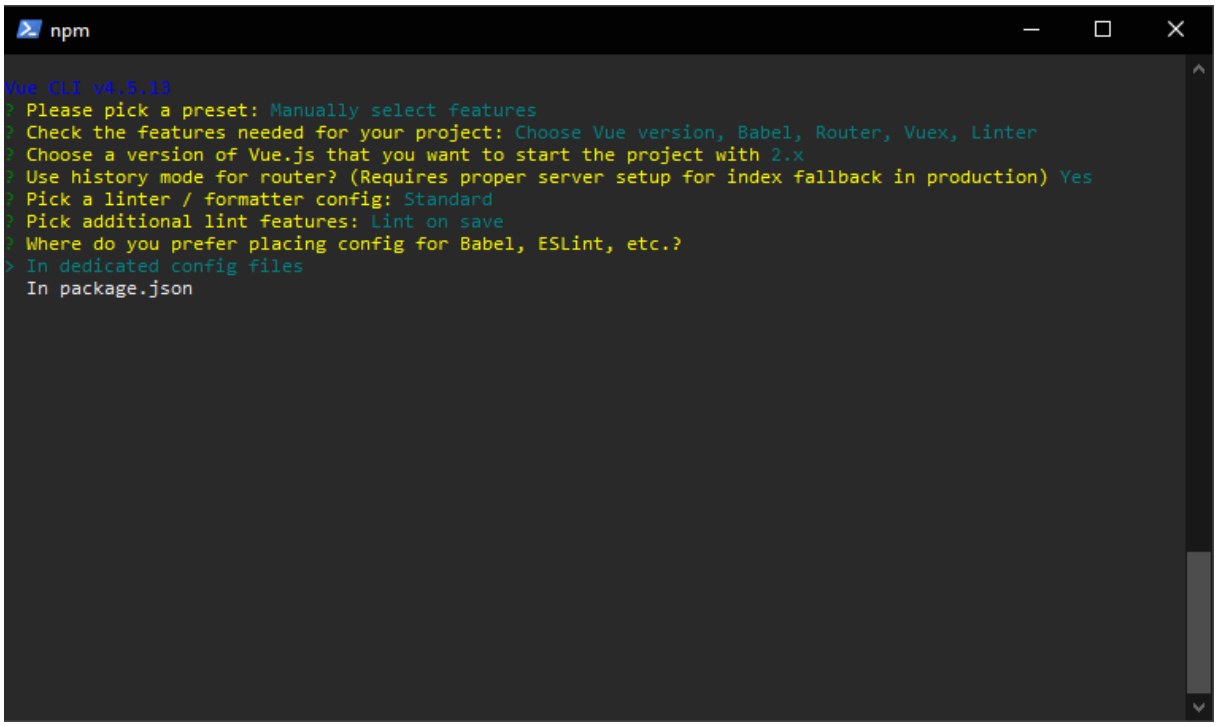


Figura 3.7: Selección de la configuración del *linter*

Una vez hecho esto, añadimos Quasar al proyecto con `vue add quasar`.

En el proceso, también deberemos seleccionar las opciones de preprocesadores de CSS o los conjuntos de iconos que queremos en nuestro proyecto.

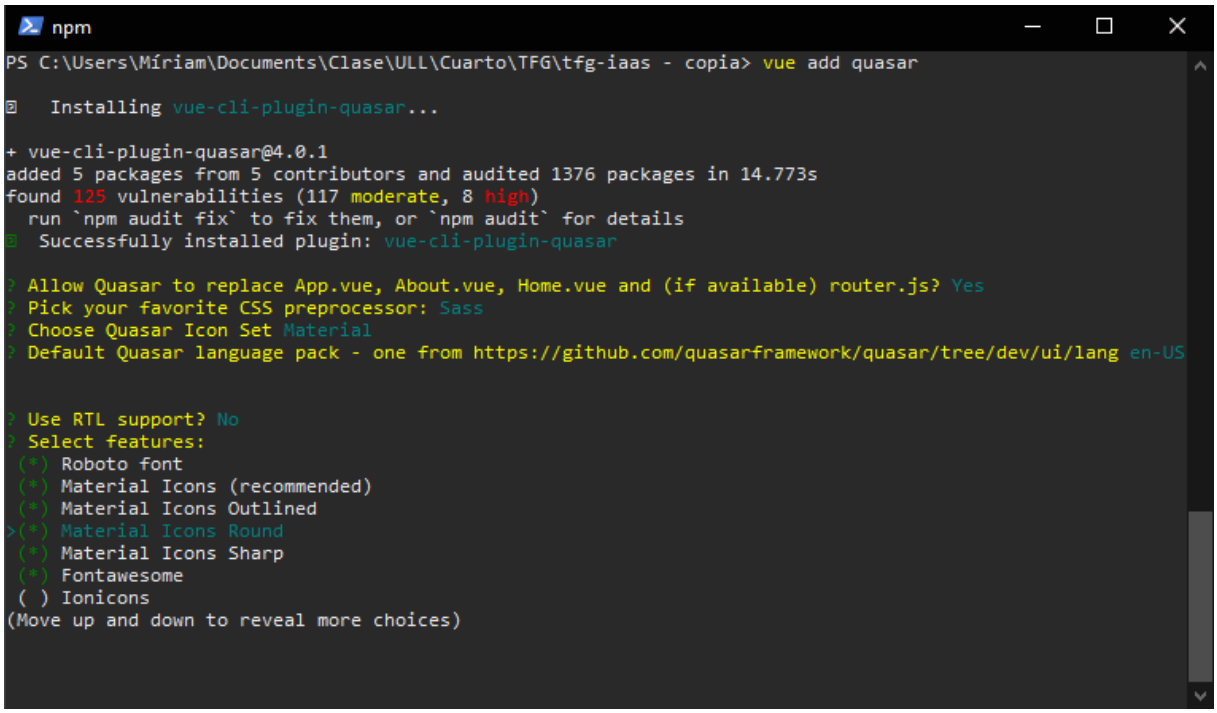


Figura 3.8: Selección de la configuración de *Quasar*

3.2.1. Travis CI

Para añadir *Travis CI*[17] al proyecto, tenemos que dar permiso al repositorio y crear un fichero *.travis.yml* que contendrá la configuración de este, aquí vemos un ejemplo:

```yaml
language: node_js
node_js:
  - stable
cache:
  directories:
    - node_modules
script:
  - npm install -g @vue/cli
  - npm install
  - npm run test:unit
```

Código 3.2: *.travis.yml*

Una vez hecho esto, *Travis* comenzará a ejecutar los *tests* al hacer un *push*.

3.3. Smartmock

Como comenté anteriormente, no dispongo de *backend*, por ello, para conseguir los datos de las posibles máquinas virtuales tengo que hacer peticiones a las simulaciones (*mocks*) de *API* que cree en la página *Smartmock*[12].

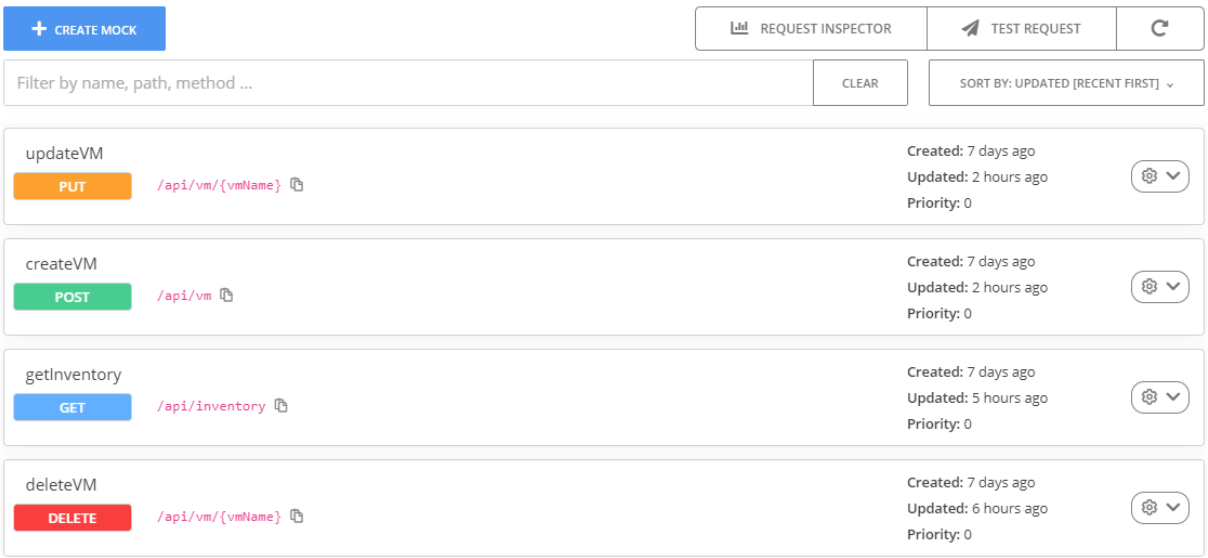


Figura 3.9: Mocks en la página *Smartmocks*

Por ejemplos, al realizar una petición a `https://tfg-iaas-vm.app.smartmock.io/inventory` se envía la siguiente respuesta:

```
{
  'method': '{{ req.method }}',
  'description': 'get the inventory of an user',
  'status': 200,
  'response': 'items returned',
  'allVirtualMachines': [
    {
      'id': '{{randomString length=4 type='NUMERIC'}}',
      'name': 'SyTW-BackEND',
      'description': 'Lorem ipsum dolor sit amet consectetur ...',
      'status': 'ON',
      'os': 'linux',
      'ram': '4 GiB',
      'memory': '20 GiB',
      'template': 'ubuntu-1804',
      'ips': ['172.{{ randomString length=2 type='NUMERIC' }}.{{ ...'],
      'created': 'YYYY-MM-DD'
    },
    ...
```

Código 3.3: Parte de respuesta de hacer la petición al *path /inventory*

Para crear esas simulaciones el compañero que esta realizando el backend de la aplicación realizó un descripción de lo que sería su *API* para que tuviera concordancia mis simulaciones, definiendo el *path*, los parámetros necesarios y las respuestas que devolvería la petición *HTTP*. Para visualizarla tenía la opción del navegador en un archivo *.html* o en un *.json*. El diseño de la API se ha realizado siguiendo el estandar OpenAPI 3.0[9] y utilizando la herrameinta Swagger.io[14].

```
"paths": {
  "/inventory": {
    "get": {
      "tags": [ "all" ],
      "summary": "get the inventory of an user",
      "description": "It returns the invenotry of virtual machines that
      are assigned to an user",
      "operationId": "getInventory",
      "responses": {
        "200": {
          "description": "items returned",
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Inventory"
              }
            }
```

```
18            }
19          },
20          "400": {
21            "description": "invalid input, object invalid"
22          }
23        }
24      }
25    }
```

Código 3.4: Parte de definición de *API*

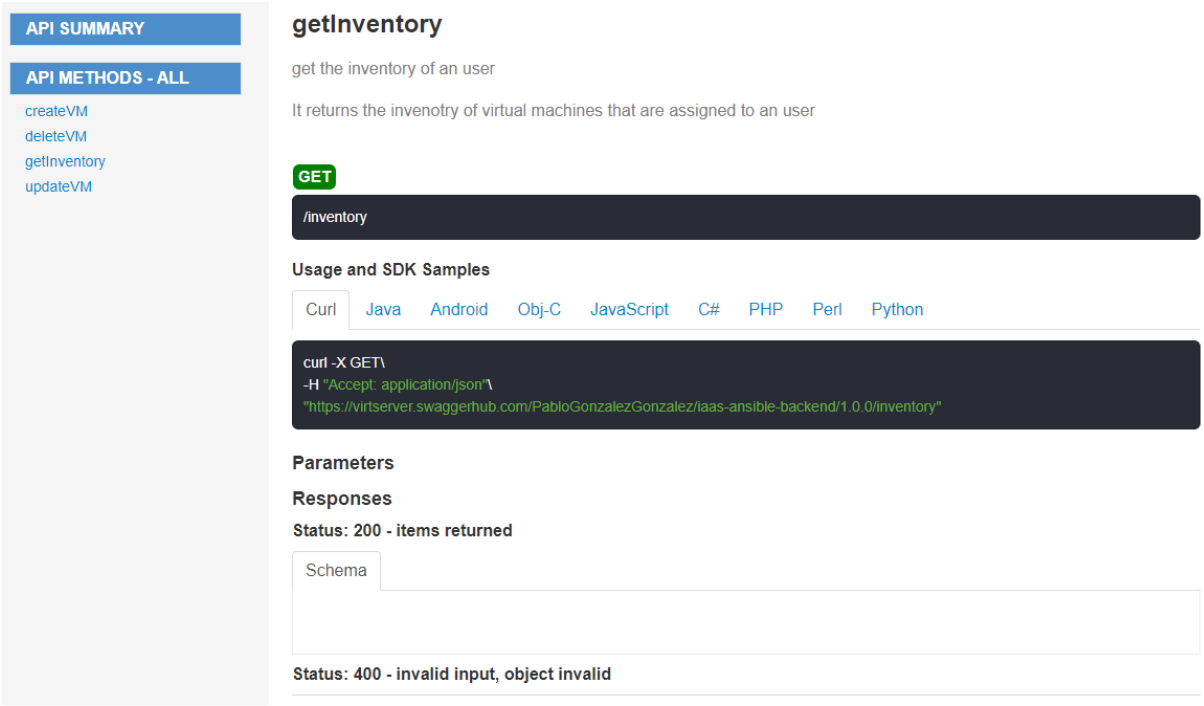


Figura 3.10: Definición de *API* en el navegador

3.4. Desarrollo del Front-End

Como ya sabemos, una aplicación desarrollada en *Vue*[18] se compone de vistas y de componentes. Estos dos tipos tiene la misma organización:

```
1 <template></template>
2 <script></script>
3 <style lang="sass" scoped></style>
```

Código 3.5: *views/Home.vue*

3.4.1. Vista

En este proyecto nos encontramos tres vistas en el directorio */src/views*, *Home.vue*, *VirtualMachine.vue* y *Form.vue*.

La primera se utiliza para tener el componente *Panel*, la segunda para el componente *VirtualMachineInfo* y en la tercera tenemos los dos componentes formularios, *CreateForm* y *EditForm*.

3.4.2. Componente

Los componentes se guardan en la carpeta */src/components*.

En ella, vemos los componentes básicos de *Header*, *Footer* y *RightMenu*.

También vemos el componente *Panel*, que contiene una tabla y dependiendo de la forma en la que el usuario quiera la información se cambiará entre el componente *VirtualMachineList* o *VirtualMachineCard*.

Asimismo, encontramos el componente *VirtualMachineInfo* con la información más detallada de la máquina virtual.

Por último, tenemos los dos formularios *CreateForm* y *EditForm* para crear y editar las máquinas virtuales.

3.4.3. Vue Router

Para esta aplicación hay tres vistas creadas, por ello, se han definido las siguientes rutas, una por cada vista.

Este fichero *index.js* se encuentra dentro de */src/router*.

```
1  import Vue from 'vue'
2  import VueRouter from 'vue-router'
3
4  Vue.use(VueRouter)
5
6  const routes = [
7    {
8      path: '/',
9      name: 'Home',
10     component: () => import('../views/Home')
11   },
12   {
13     path: '/vm',
14     name: 'VirtualMachine',
15     component: () => import('../views/VirtualMachine')
16   },
```

```
17   {
18     path: '/form',
19     name: 'Form',
20     component: () => import('../views/Form')
21   }
22 ]
23
24 const router = new VueRouter({
25   mode: 'history',
26   base: process.env.BASE_URL,
27   routes
28 })
29
30 export default router
```

Código 3.6: *router/index.js*

3.4.4. Vuex

En el almacén de *Vuex*, decidimos guardar el conjunto de las máquinas virtuales, la máquina virtual que se estuviera visualizando en el momento y una variable que te determina el tipo de formulario, para saber si se muestra el componente *CreateForm.vue* o *EditForm.vue*.

Este fichero *index.js* se encuentra en el directorio */src/store*.

```
1  import Vue from 'vue'
2  import Vuex from 'vuex'
3
4  Vue.use(Vuex)
5
6  export default new Vuex.Store({
7    state: {
8      virtualMachines: null,
9      virtualMachine: null,
10     formType: null
11   },
12   mutations: {
13     setVirtualMachines (state, data) {
14       state.virtualMachines = data
15     },
16     setVirtualMachine (state, data) {
17       state.virtualMachine = data
18     },
19     setFormType (state, data) {
20       state.formType = data
```

```
21      }
22    },
23    actions: {
24      setVirtualMachinesAction (context, data) {
25        context.commit('setVirtualMachines', data)
26      },
27      setVirtualMachineAction (context, data) {
28        context.commit('setVirtualMachine', data)
29      },
30      setFormTypeAction (context, value) {
31        context.commit('setFormType', value)
32      }
33    },
34    getters: {
35      virtualMachines (state) {
36        return state.virtualMachines
37      },
38      virtualMachine (state) {
39        return state.virtualMachine
40      },
41      formType (state) {
42        return state.formType
43      }
44    }
45  })
```

Código 3.7: *store/index.js*

3.5. Testing

La idea principal del desarrollo de este proyecto fue que se desarrollara mediante la metodología de desarrollo guiado por pruebas o *Test-Driven Development* (*TDD*). Sin embargo, esto no se ha podido cumplir y, finalmente, los *test* se han realizado prácticamente en el último momento.

Se han llevado a cabo dos tipos de *tests*, los test unitarios y las pruebas *end-to-end*.

3.5.1. Unit Testing

Para realizar *tests* en la aplicación se ha utilizado principalmente la librería de *Vue*, *Vue Test Utils*[21] y *Jest*[8] para los test unitarios (*unit testing*).

Podemos ver a continuación un ejemplo de ellos dentro del directorio */tests/unit*:

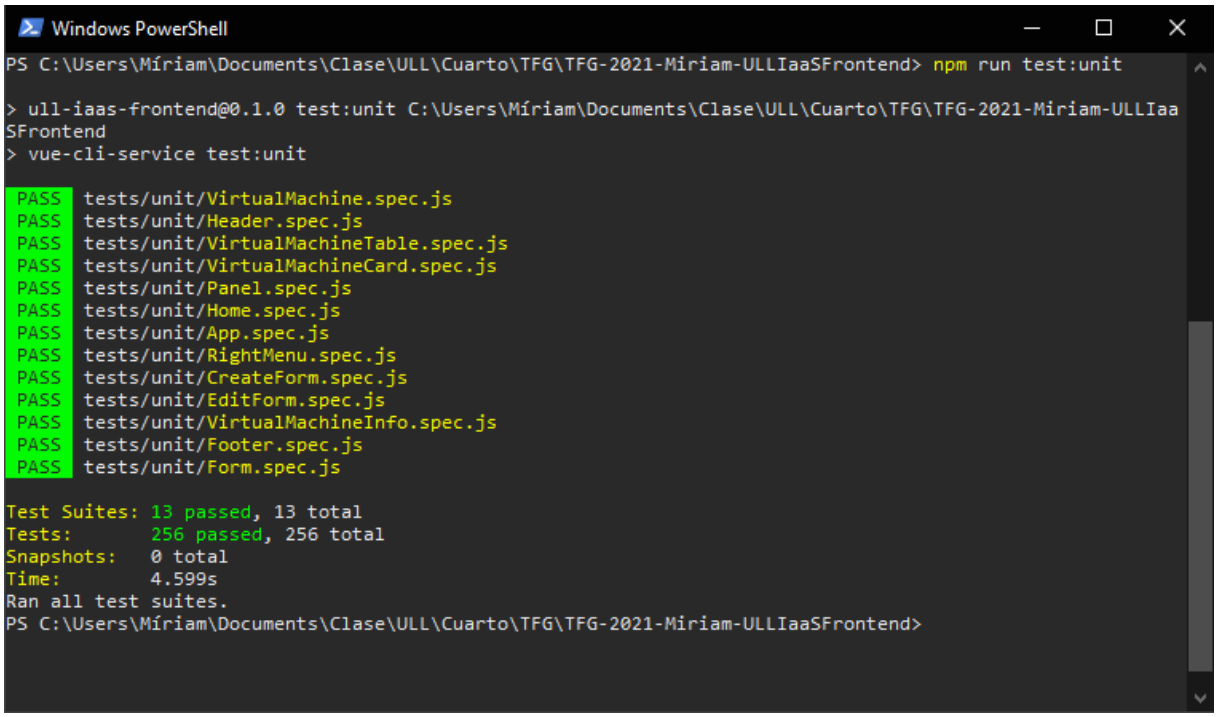


Figura 3.11: Pruebas unitarias correctas

```
describe('Form', () => {
  const wrapper = shallowMount(Form, {
    localVue,
    store
  })

  it('mounts without errors', async () => {
    expect(wrapper).toBeTruthy()
  })

  it('contains CreateForm component', async () => {
    await wrapper.setData({
      formType: 'create'
    })
    expect(wrapper.findComponent(CreateForm).exists()).toBe(true)
  })
})
```

Código 3.8: *unit/index.js*

3.5.2. End-To-End Testing

En el caso de este tipo de pruebas, hemos utilizado el *framework Cypress*[2]. Este facilita la ejecución de los *tests* ya que lo realiza desde el mismo navegador,

eso sí, es necesario que tu aplicación este corriendo en el fondo para su correcto funcionamiento.

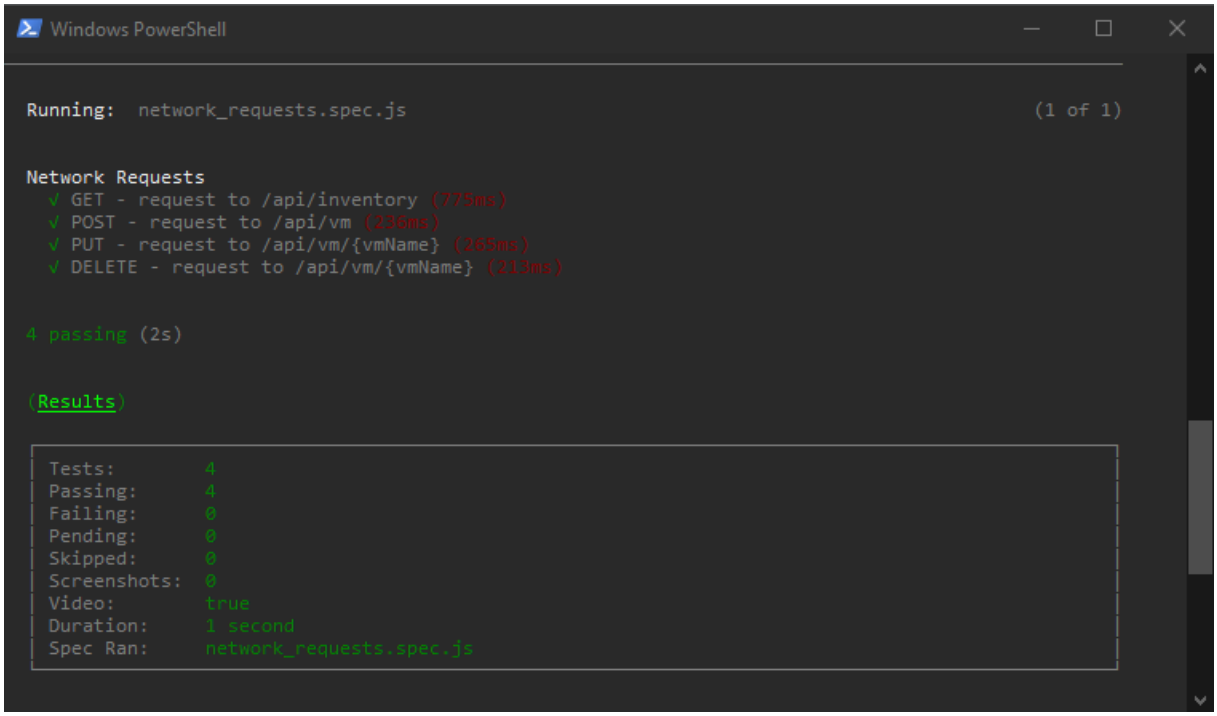


Figura 3.12: Pruebas e2e correctas

Podemos ver un ejemplo de estas pruebas que se encuentran en la carpeta */tests/cypress/integration*:

```
context('Network Requests', () => {
  it('DELETE - request to /api/vm/{vmName}', () => {
    cy.request('DELETE', 'https://tfg-iaas-vm.app.smartmock.io/api/vm/
    SyTW-BackEND')
      .then((response) => {
        expect(response.status).to.eq(200)

        const data = JSON.parse(response.body.replace(/'/g, '"'))

        expect(data).to.have.property('method')
        expect(data).property('method').to.equal('DELETE')

        expect(data).to.have.property('description')
        expect(data).property('description').to.equal('delete a virtual
        machine')

        expect(data).to.have.property('name')
        expect(data).property('name').to.equal('SyTW-BackEND')

        expect(data).to.have.property('response')
```

```
21          expect(data).property('response').to.equal('item SyTW-BackEND
22          deleted')
23        })
24    })
25 }
```

Código 3.9: *integration/network_request.spec.js*

3.6. Despliegue

Para desplegar la aplicación pensamos en *Surge*[13].

3.6.1. Surge

La plataforma *Surge* hace realmente sencillo poder desplegar una aplicación.

En primer lugar, es necesario ejecutar el comando `npm run build` para que se genere el directorio */dist* que es el que vamos a subir a la plataforma. Podemos ver los pasos a seguir a continuación.

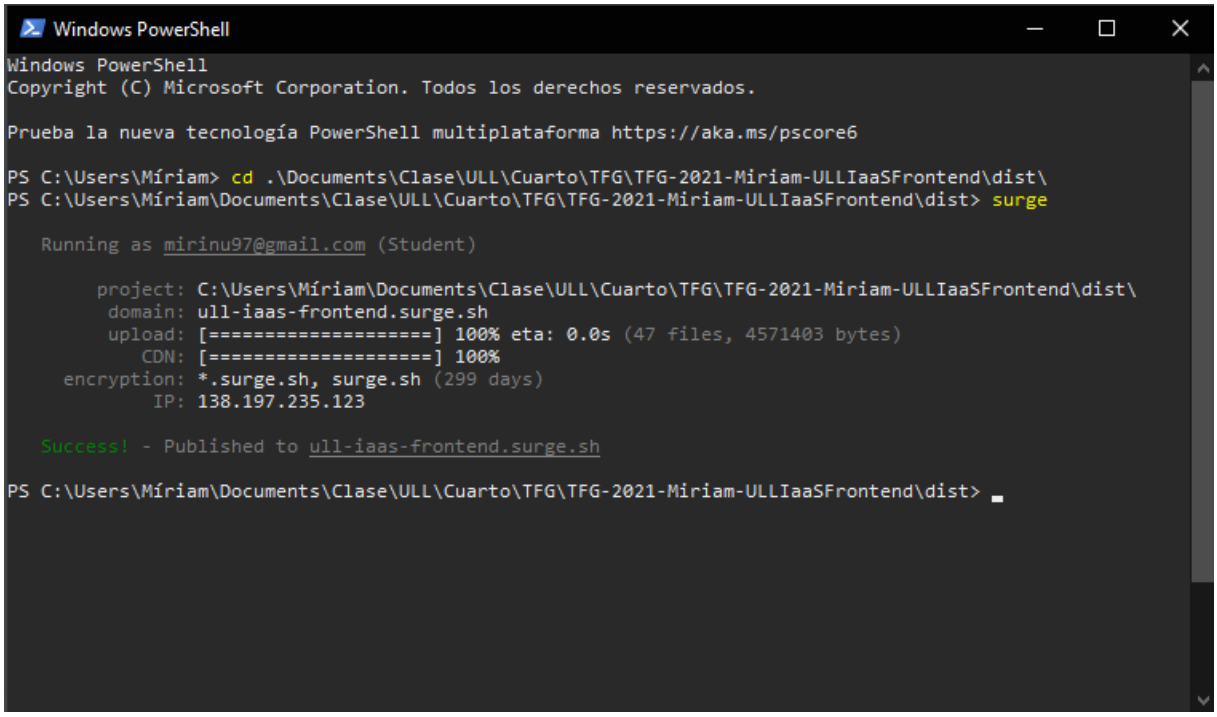


Figura 3.13: Pasos para desplegar con *Surge*

El enlace es: `https://ull-iaas-frontend.surge.sh/`

Capítulo 4

Descripción de la Aplicación

Una vez hablado de como se desarrolló el proyecto, en este capítulo se hará una descripción de la interfaz de usuario que encontraremos en la aplicación acompañado de capturas de pantallas de la versión web y de la versión móvil.

4.1. Usuarios

Esta aplicación se encuentra orientada a todo persona asociada a la Universidad de La Laguna que pueda tener acceso al servicio *IaaS* que es proporcionado, desde alumnos, profesores o personal de administración y servicios.

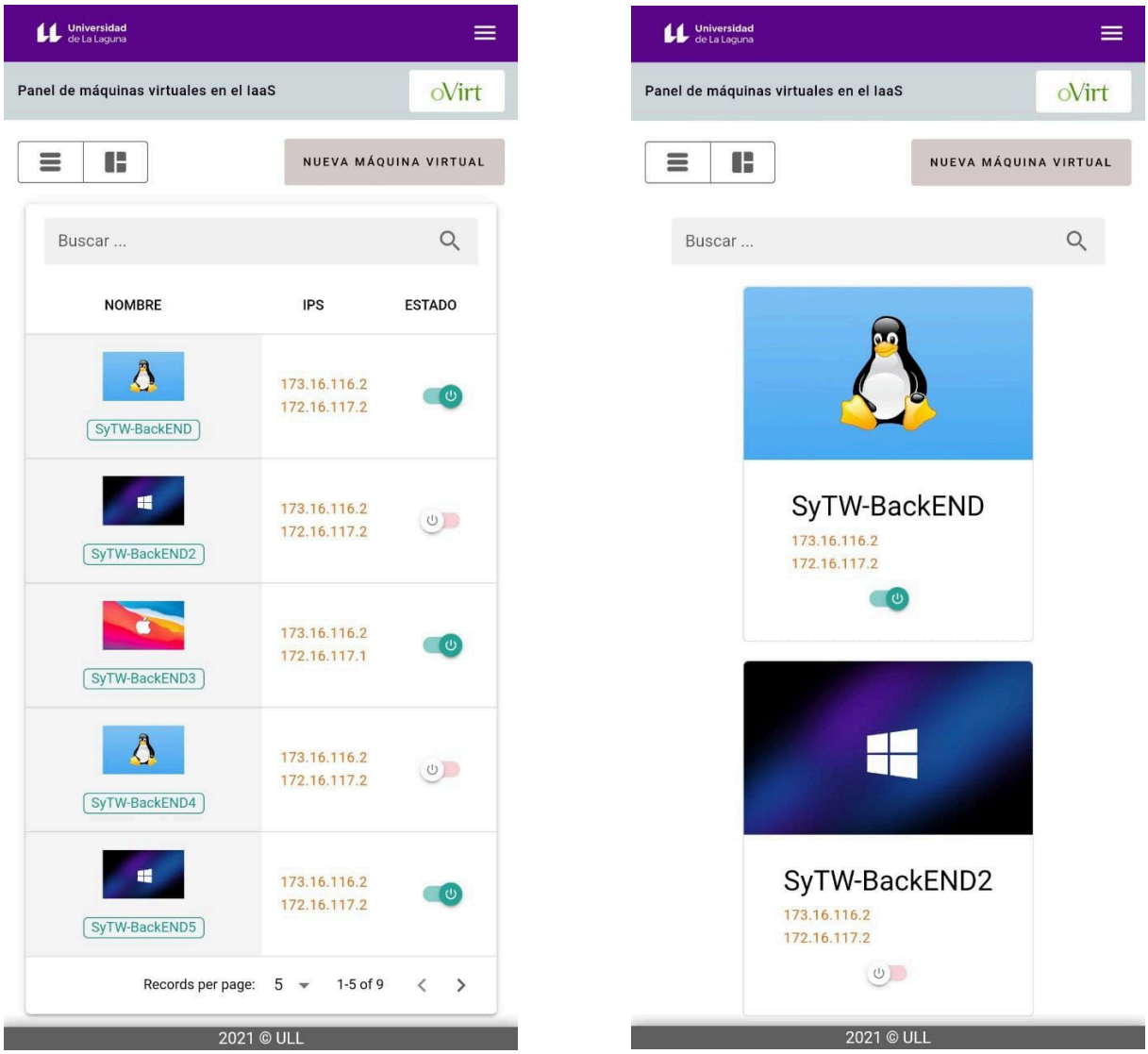
4.2. Sección de Panel

Según se entra a la aplicación, encontramos una vista con un panel donde aparecerán las máquinas virtuales a las que se tiene acceso, ya sea porque la ha creado el usuario o porque se le ha creado automáticamente desde alguna asignatura.

En esta pantalla tendremos dos formas de visualizar el contenido, en forma de lista y en forma de mosaico, como podemos ver en las imágenes a continuación.

En la tabla podremos ordenar las máquinas virtuales dependiendo del nombre de la columna en la que pulsemos, se ordenará en orden ascendente o descendente.

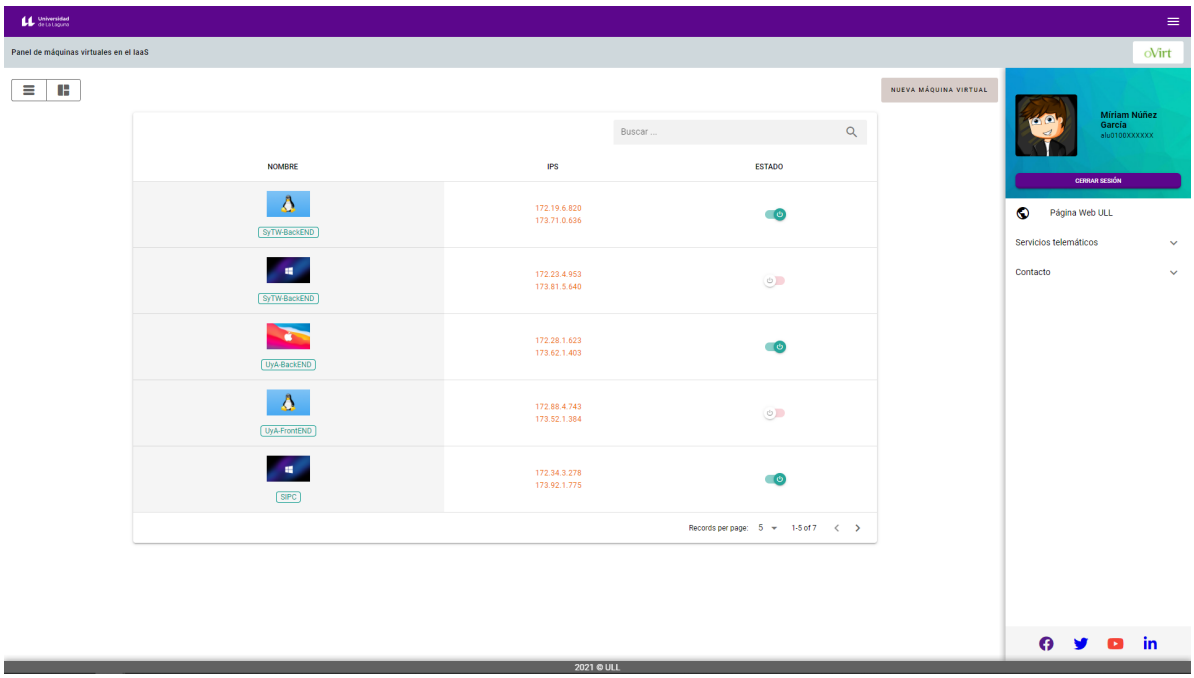
Además, disponemos de un filtro para buscar por el nombre, las IPs o el estado de la máquina.



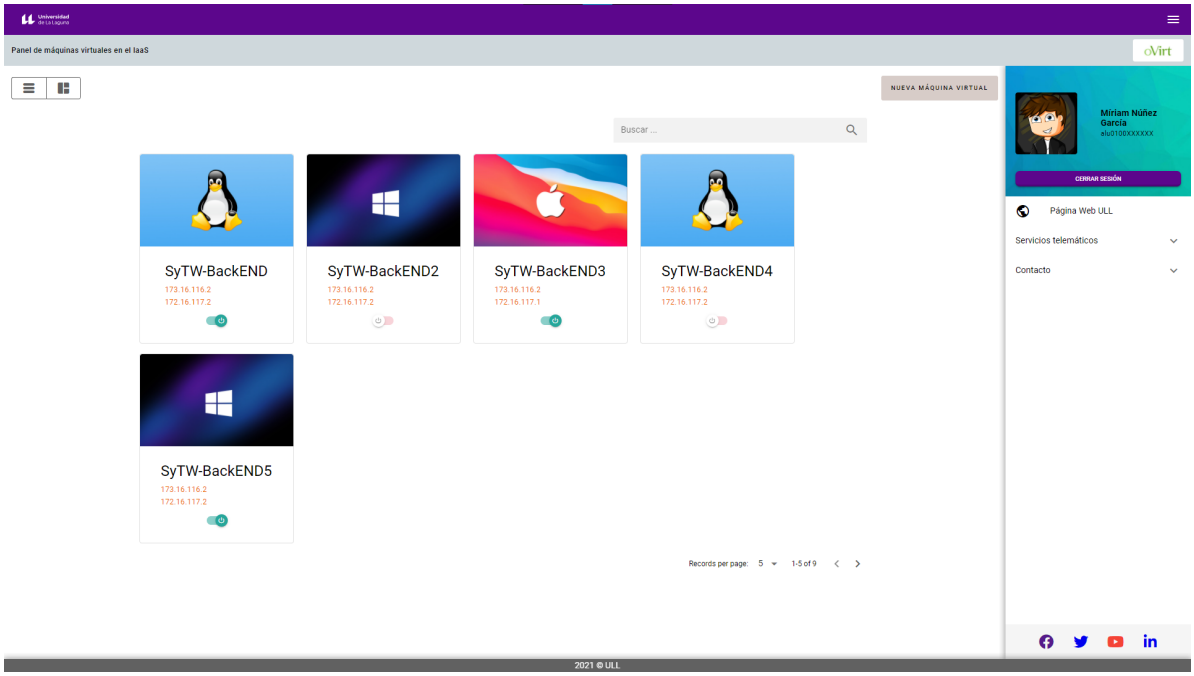
(a) Vista de lista

(b) Vista de mosaico

Figura 4.1: Pantalla de panel en el móvil



(a) Vista de lista



(b) Vista de mosaico

Figura 4.2: Pantalla de panel en la web

4.3. Sección de Menú

En la parte derecha del navegador nos encontramos un menú desplegable, que en el caso de la versión web se encuentra por defecto desplegado (se puede observar en las imágenes anteriores).

El menú contiene un enlace a la página web oficial de la ULL, los servicios telemáticos que ofrece, los datos de contacto y las redes sociales.

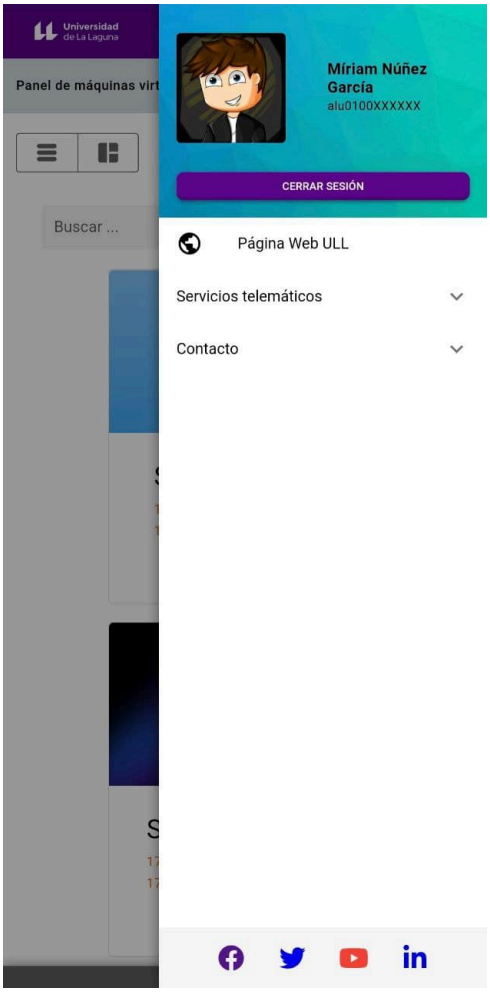


Figura 4.3: Pantalla de menú derecho en el móvil

4.4. Sección de las Máquinas Virtuales

Al seleccionar una de las máquinas virtuales del panel aparece una vista con información más detallada de esta.

Esta vista cuenta, por un lado, con una imagen que hace referencia al sistema operativo de la máquina, el nombre, el número identificador y el estado. Por otro lado, podemos ver la descripción, la plantilla que se utilizó, el sistema operativo más detallado, la memoria RAM y de almacenamiento, las IPs y la fecha de creación.

Asimismo, tenemos varios botones que hacen las funciones de encender, abrir la consola, suspender, reiniciar, apagar, editar, compartir y eliminar la máquina virtual.

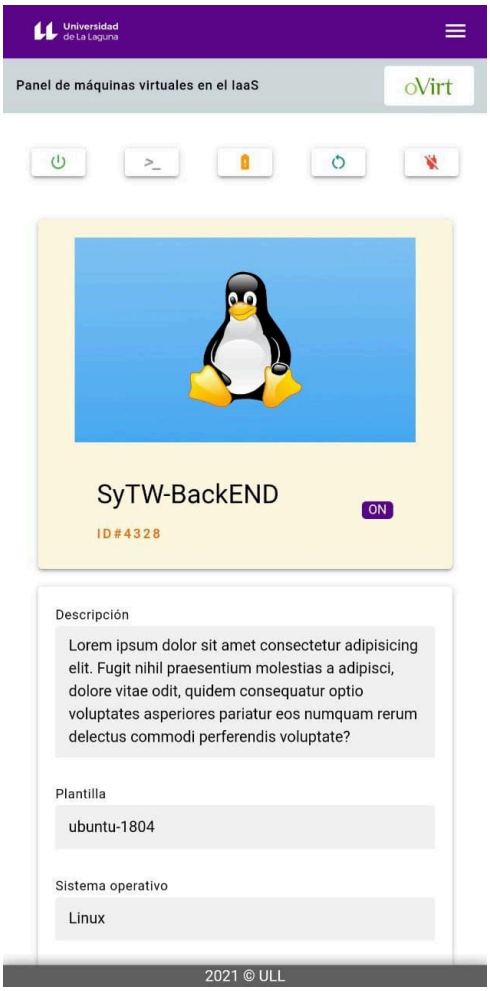


Figura 4.4: Pantalla de la información de la máquina virtual en el móvil

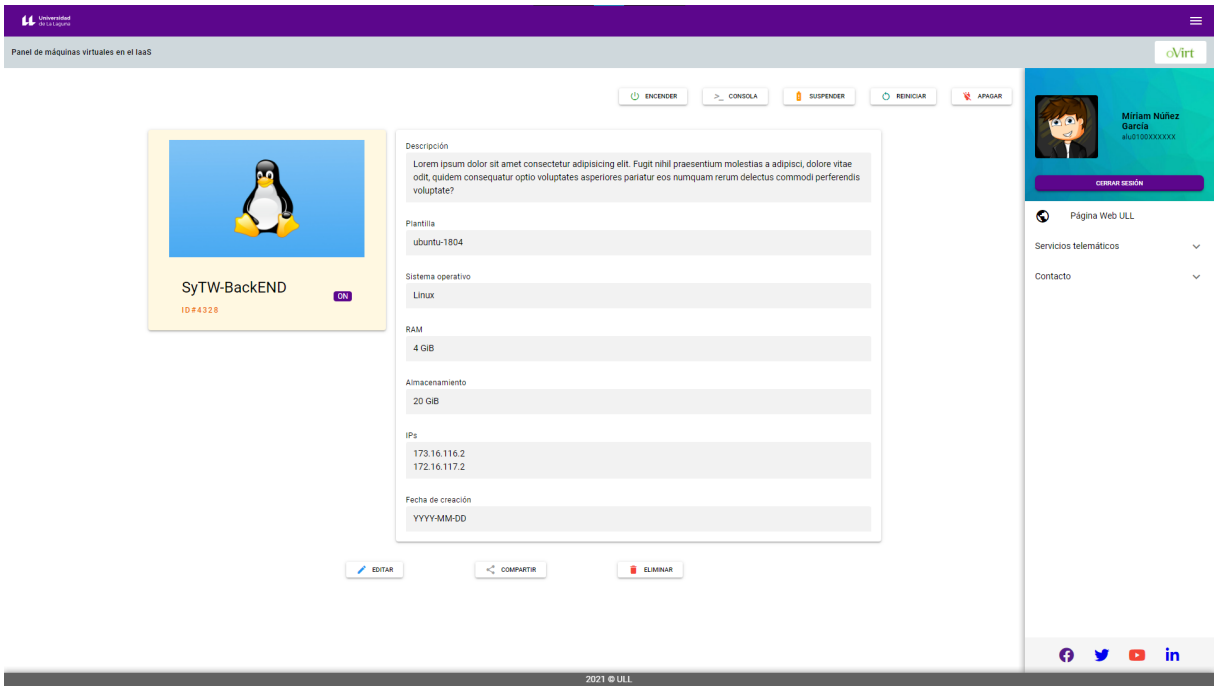


Figura 4.5: Pantalla de la información de la máquina virtual en la web

4.5. Sección de Formularios

4.5.1. Crear Máquina Virtual

En la vista del panel (Figura 4.1) también podemos observar un botón para crear un nueva máquina virtual. Este nos lleva al siguiente formulario en que se rellenarán los datos correspondientes.

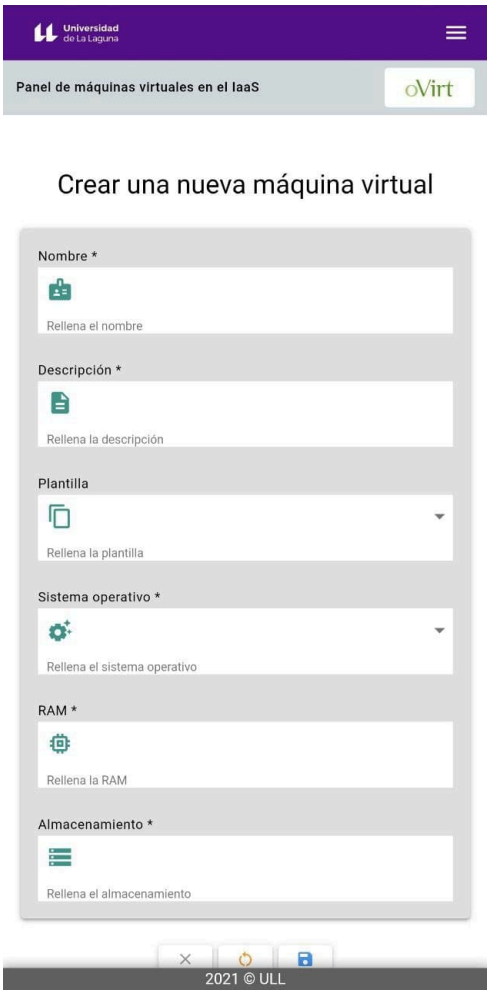


Figura 4.6: Pantalla del formulario de crear una máquina virtual en el móvil

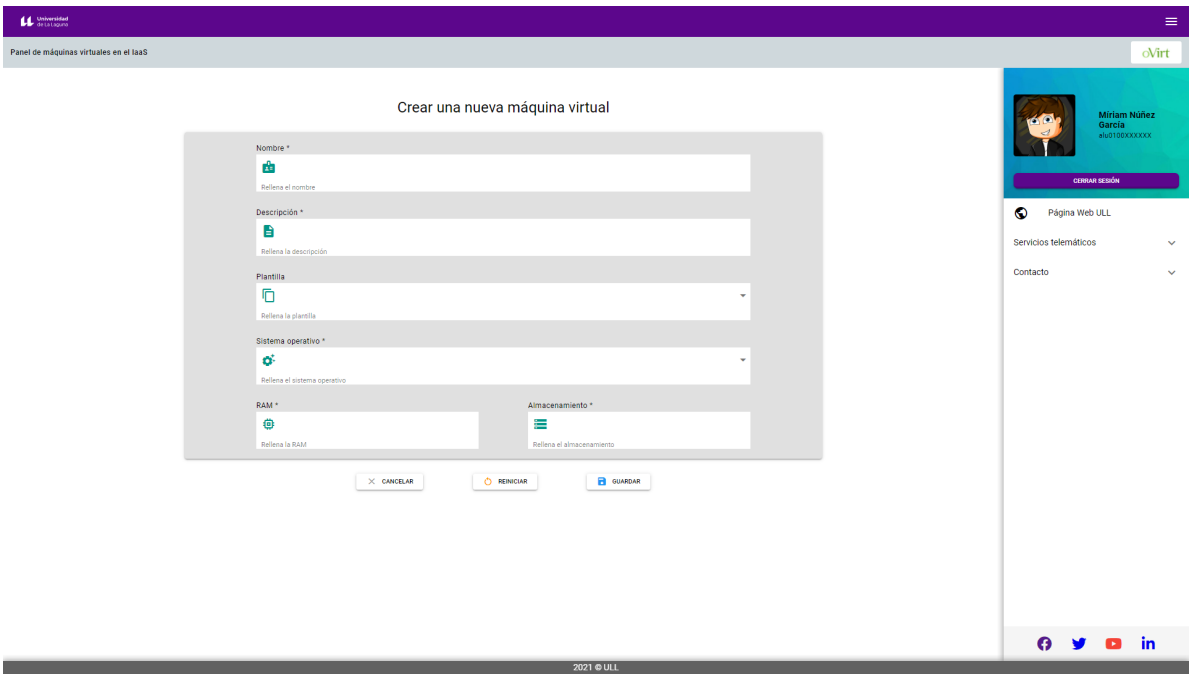


Figura 4.7: Pantalla del formulario de crear una máquina virtual en la web

4.5.2. Editar Máquina Virtual

En la parte final de la vista con la información de las máquinas virtuales (Figura 4.5 encontramos una opción para poder editarla.

Este formulario es muy similar al de crear una máquina, sin embargo, en este no puedes modificar la plantilla ni el sistema operativo de esta, solamente es posible cambiar el nombre, la descripción, la cantidad de memoria RAM y de almacenamiento.

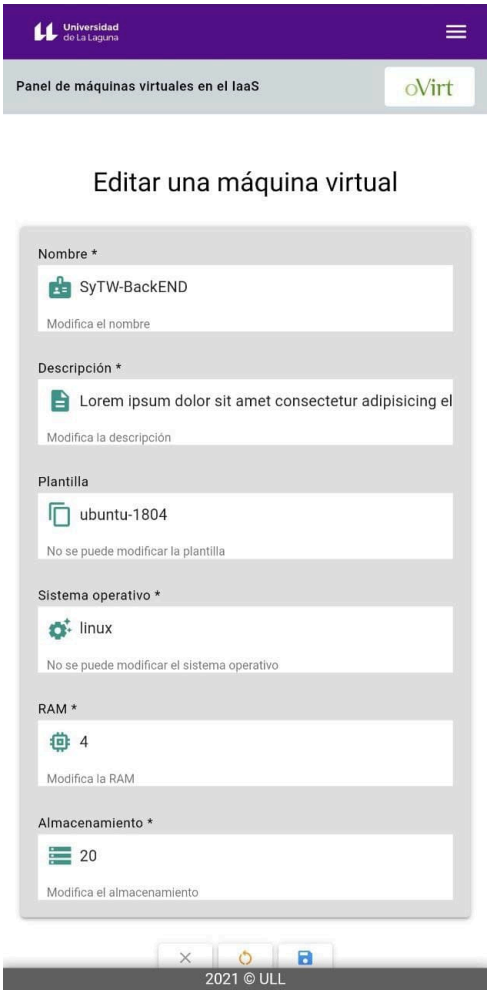


Figura 4.8: Pantalla del formulario de editar una máquina virtual en el móvil

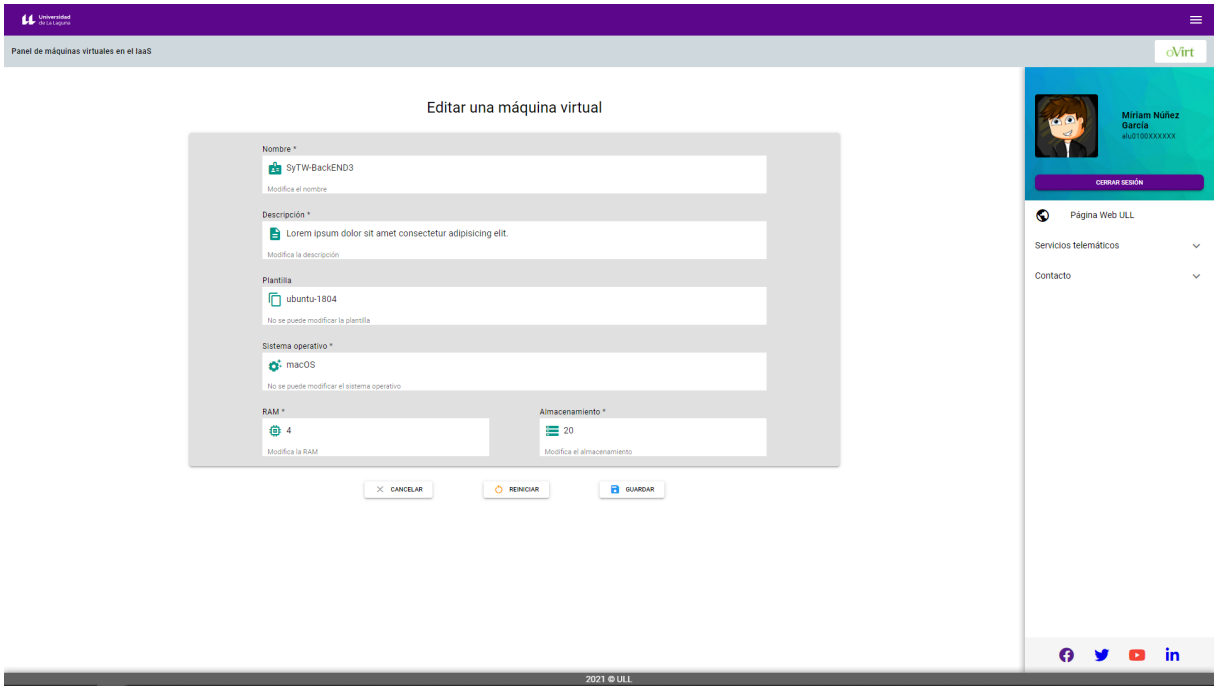


Figura 4.9: Pantalla del formulario de editar una máquina virtual en la web

Capítulo 5

Conclusiones y líneas futuras

Para concluir, este TFG ha sido un proyecto donde se ha practicado y aprendido el desarrollo de una aplicación web incompleta, es decir, sin *backend*. De forma que este ha tenido que ser simulado gracias a una plataforma cuya función es simular una *API* contra la que se puede hacer peticiones *HTTP* en estos casos. Esta API se ha diseñado siguiendo el estandar *OpenAPI 3.0* y se ha documentado con la herramienta *Swagger.io*. Esta aproximación permite el desarrollo de los proyectos de *backend* y *frontend* de forma independiente.

Como se ha comentado, este proyecto ha consistido en el diseño de un *frontend*. Para ello, hemos realizado una fase de prototipado en la que nos hemos basado a la hora de comenzar a desarrollar la aplicación web. Esta se ha desarrollado con *Vue*, un *framework* relativamente sencillo de utilizar y con una gran cantidad de *ui components* gracias a sus múltiples librerías disponibles.

Los objetivos generales propuestos del proyecto se han cumplido, incluido la realización de los *test*, que aunque se han llevado a cabo pruebas representativas de la funcionalidades implementadas, han sido bastante complicadas por mi falta de experiencia con ellos. Otra razón fue que me centré en el desarrollo de la aplicación en sí y no le di importancia a estos, por ello, cuando llegué a esa parte realmente no disponía del tiempo necesario para hacer la investigación de cómo hacer unos *tests* de calidad.

De cara al futuro, la idea sería poder integrar el proyecto con el *backend* que actualmente está desarrollando un compañero. Gracias al diseño abierto de la API, descartar la *API* simulada y conectar con el *backend* real será un proceso sencillo. Finalmente quedaría probar la aplicación real, con pruebas *end-to-end* y ser capaces de hacer una aplicación completamente funcional. Además, se plantea aumentar la cobertura de los *tests*.

Capítulo 6

Summary and Conclusions

To conclude, this TFG has been a project where the development of an incomplete web application has been practiced and learned, that is, without *backend*. So this has had to be simulated thanks to a platform whose function is to simulate an *API* against which you can make *HTTP* requests in these cases. This *API* has been designed following the *OpenAPI 3.0* standard and has been documented with the *Swagger.io* tool. This approach allows the development of the backend and frontend projects independently.

As mentioned, this project consisted of the design of a *frontend*. To do this, we have gone through a prototyping phase on which we have based on when starting to develop the web application. It has been developed with *Vue*, a relatively easy-to-use *framework* with a large number of *ui components* thanks to its multiple libraries available.

The general objectives proposed for the project have been met, including the development of the tests, which, although representative tests of the implemented functionalities have been carried out, have been quite complicated due to my lack of experience with them. Another reason was that I focused on the development of the application itself and did not give importance to these, therefore, when I got to that part I really did not have the time necessary to do the research on how to do quality tests.

Looking ahead, the idea would be to be able to integrate the project with the partner's *backend* project which is now under development. Thanks to the open design of the *API*, dropping the simulated *API* and connecting to the real *backend* will be an easy process. Finally, it would be necessary to test the real application, with *end-to-end* testing and being able to make a fully functional application. In addition, it is proposed to increase the coverage of the tests.

Capítulo 7

Presupuesto

En esta sección haremos una opción de presupuesto para lo que sería el desarrollo con una duración aproximada de 3 meses del proyecto explicado en esta memoria.

En primer lugar, podemos hablar de lo que realmente sería más costoso, el trabajo que el desarrollador web tendría que realizar. Sería necesario llevar un recuento de las horas que le dedica al proyecto y multiplicarlo por su salario por hora. Este, dependiendo de si es un Ingeniero de Software *Junior* o *Senior*, puede oscilar entre 20€/hora y 100€/hora.

En segundo lugar, tendríamos el alojamiento de la aplicación. De momento esta desplegada en la plataforma *Surge*[13], sin embargo, si en algún momento el proyecto crece había que plantearse otra plataforma como *Heroku*[6], que tiene opción gratuita y paga.

En la siguiente tabla supondremos que sí es necesaria, dado que el que se está utilizando actualmente funciona únicamente para el *frontend* una plataforma de pago.

Concepto	**Coste mensual**	**Coste total**
Desarrollo del proyecto	1.200€	3.600€
Alojamiento en Heroku	21€	63€
Total		**3.663€**

Tabla 7.1: Presupuesto para el desarrollo del proyecto

Bibliografía

[1] Babel. https://babeljs.io/. Accessed: 2021-06-25.

[2] Cypress. https://www.cypress.io/. Accessed: 2021-06-30.

[3] Eslint. https://eslint.org/. Accessed: 2021-06-25.

[4] Git. https://git-scm.com/. Accessed: 2021-05-10.

[5] Github. https://github.com/. Accessed: 2021-05-10.

[6] Heroku. http://www.heroku.com/. Accessed: 2021-06-29.

[7] IaaS de la universidad de la laguna. https://iaas.ull.es/ovirt-engine/. Accessed: 2021-03-20.

[8] Jest. https://jestjs.io/es-ES/. Accessed: 2021-06-29.

[9] Openapi 3.0. https://swagger.io/specification/. Accessed: 2021-07-02.

[10] Panel informativo de máquinas virtuales en IaaS. http://iaas.ull.es/mismaquinas/. Accessed: 2021-04-05.

[11] Quasar. https://quasar.dev/. Accessed: 2021-04-26.

[12] Smartmock. https://smartmock.io/. Accessed: 2021-04-27.

[13] Surge. https://surge.sh/. Accessed: 2021-06-29.

[14] Swagger. https://swagger.io/. Accessed: 2021-07-02.

[15] Tfg-ull-iaas-frontend. https://github.com/ULL-TFGyMs-vblanco/TFG-2021-Miriam-ULLIaaSFrontend.git.

[16] Todoist. https://todoist.com/. Accessed: 2021-06-02.

[17] Travis ci. https://www.travis-ci.com/. Accessed: 2021-06-25.

[18] Vue. https://vuejs.org/. Accessed: 2021-04-07.

[19] Vue cli. https://cli.vuejs.org/. Accessed: 2021-04-07.

[20] Vue router. https://router.vuejs.org/. Accessed: 2021-06-25.

[21] Vue test utils. https://vue-test-utils.vuejs.org/. Accessed: 2021-06-29.

[22] Vuex. https://vuex.vuejs.org/. Accessed: 2021-04-07.