Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Sistema De Planificación De Recursos Para Restaurantes
Resource lanning System For Restaurants

Manuel Jesús Peraza Alonso

La Laguna, 6 de julio de 2020

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Sistema De Planificación De Recursos Para Restaurantes*"

ha sido realizada bajo su dirección por D.**Manuel Jesús Peraza Alonso**,con N.I.F. 78.645.518-P.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de julio de 2020

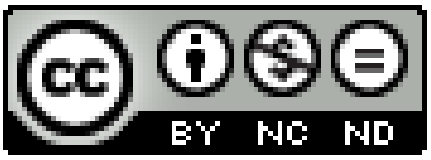
Agradecimientos

Quiero aprovechar la ocasión para agradecer a mis compañeros, en especial a Jesús Ramos Álvarez y Sergio del Pino Hernández, por su apoyo en todo lo que he necesitado a lo largo de estos años. También me gustaría agradecer a mi tutor Vicente José Blanco Pérez por su atención, así como de su constante ayuda proporcionando tanto las herramientas como los consejos necesarios para llevar a cabo este trabajo de manera mucho más sencilla.

Por último darle las gracias a mis padres, pareja y amigos por ser ese pilar fundamental en momentos buenos y no tan buenos.

Gracias por todo.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional.

Resumen

Este trabajo se ha basado en la realización de un ERP (Sistema de planificación de recursos empresariales), orientado al sector de la restauración. Está destinado a un cliente real el cual quiere mejorar la eficacia de su negocio, tras estudiar sus métodos de trabajo se ha seleccionado esta aplicación como primer paso a la automatización de los procesos de los gerentes, usuario objetivo del aplicativo.

Este proyecto trata de suplir el problema de tiempo que tienen los gerentes a la hora de programar pedidos, revisarlos y administrar a sus proveedores. Para ello se ha creado esta herramienta que mediante una interfaz sencilla e intuitiva diseñada poco a poco gracias al feedback de los propios gerentes se ajuste a sus necesidades y les permita generar un trabajo más eficiente.

Las tecnologías utilizadas para el desarrollo han sido: NodeJS, Express, MongoDBAtlas, Firebase, DigitalOcean, Travis, Jest, Cypress, Vue, Vue-CLI, Vuetify.

Se ha externalizado la base de datos con el servicio de MongoDBAtlas para evitar problemas de escalado así como de seguridad. Como ventaja de esta aplicación tenemos el pequeño número de usuarios. El único uso que iría en aumento claro sería el almacenamiento. Por motivos parecidos se ha externalizado la autenticación con el módulo de Firebase Authentication.

Palabras clave: Interfaz, Diseño, Gestión, Tratamiento de datos, Tecnologías Web, Administración Empresarial.

Abstract

This work has been based on the implementation of an ERP (Enterprise Resource Planning System), oriented to the catering sector. It is intended for a real client who wants to improve the efficiency of their business. After studying their working methods, this application has been selected as the first step towards automating the processes of managers, the target user of the application.

The project tries to fill the time problem that managers have when scheduling orders, reviewing them and managing their suppliers. For this purpose, this tool has been created that, through a simple and intuitive interface designed little by little thanks to the feedback of the managers themselves, adjusts to their needs and allows them to generate a more efficient job.

The technologies used for development have been: NodeJS, Express, MongoDBAtlas, Firebase, DigitalOcean, Travis, Jest, Cypress, Vue, Vue-CLI, Vuetify.

The database has been outsourced with the MongoDBAtlas service to avoid escalation as well as security problems. As an advantage of this application we have the small number of users. The only use that would be increasing clearly would be storage. For similar reasons, authentication with the Firebase Authentication module has been outsourced.

Keywords: Interface, Design, Management, Data Processing, Web Technologies, Business Administration.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Esta aplicación web pretende dar soporte así como mejorar la gestión de establecimientos en el área de la restauración mediante la automatización de tareas repetitivas.

La idea principal es generar una aplicación ERP que permita a los encargados de los establecimientos poder gestionar los pedidos y el personal de manera efectiva y sencilla. También se incluirá un gestor de artículos y proveedores.

El usuario podrá gestionar los artículos de cada proveedor para, a la hora de crear un pedido, pueda seleccionar aquel que más se acomode a lo requerido (precio, proveedor). Otra funcionalidad de la que disponen los usuarios es la muestra de todos los pedidos realizados, por cuenta, así como ver aquellos que han sido completados.

En cuanto al manejo del personal, dispondrá un control de todos los turnos de los empleados así como un control de las horas de entrada y salidas, todo esto dividido por establecimiento.

La aplicación tendrá dos roles, el rol de administrador, el cual maneja la parte de artículos y proveedores, por otro lado tenemos el rol de usuario que se dedicará a crear pedidos y manejar a la plantilla.

1.1. Antecedentes y estado actual del sector

Desde la revolución industrial nace la producción a gran escala, con esto aparecen los primeros sistemas de gestión con el único objetivo de aumentar la producción y automatizar procesos. Han ido evolucionando desde simples controles de inventario a gestores completos de todo el funcionamiento de la empresa. Estos pueden ser hechos a medida para una empresa concreta o estandarizados y usados de manera modular.

Gracias al cloud computing la inmediatez, accesibilidad y escabilidad de estos proyectos es mucho más simple y efectiva, la clave y el éxito de este tipo de software es su gran adaptabilidad y flexibilidad.

Hoy en día podemos encontrar gigantes de los ERP como Odoo [14] , SAP ERP [16] y Microsoft Dynamics [10] , en el caso de Odoo dispone de una parte de código abierto que permite a aquellos desarrolladores que van mejorar un ERP ahorrarse gran parte del

trabajo que ya ha sido estandarizado. La elección de no usar la opción estandarizada para el desarrollo de este proyecto se debe a que lo que buscaba el cliente no era demasiado complejo y se le podía desarrollar a medida para la forma de operar en su negocio.

1.2. Actividades realizadas para la finalización del proyecto

Este proyecto consistirá en la creación de una aplicación web mediante las tecnologías MEVN (MongoDB[14], Express[5], Vue.js[20], NodeJS[12]) y con un acercamiento a la creación de *Single Page Application* (SPA) y uso del desarrollo *Mobile First*.

Para la conclusión satisfactorio del proyecto se realizarán diferentes actividades, tales descritas a continuación:

- Prototipo de la versión móvil para tener una idea de como se quiere estructurar la aplicación web.
- *Setup* de la infraestructura de desarrollo mediante diferentes herramientas y técnicas de desarrollo:
  - Utilización de herramientas como Trello[1] para definir bien las tareas a realizar y llevar una buena planificación y seguimiento del proyecto.
  - Repositorio en la herramienta web GitHub[9] para llevar el proyecto bajo control de versiones.
  - Uso Travis CI[3] para realizar el testing.
  - Método TDD para el testing.
  - Despliegue continuo y una integración continua de la aplicación.
- *Backend*, cuya base de datos está alojada en la nube.
- *Frontend* desglosado en diferentes funcionalidades:
  - Base de la aplicación *frontend* (navegación entre las páginas disponibles)
  - *Login* para mostrar una página u otra dependiendo del usuario ingresado.
  - Diseño *frontend* de los pedidos realizados así como los recibidos.
  - Una vez conseguido lo anterior, se generará un archivo de texto con la información de los pedidos para facilitar la migración a las API de proveedores.
- Aunque la aplicación web está principalmente pensada para dispositivos móviles, se ha realizado una versión *Desktop*

1.3. Plan de Trabajo seguido

En la siguiente figura se procede a ver la organización que se hizo en el diseño previo de la aplicación. Esos fueron los tiempo estipulados en un principio para el desarrollo de cada parte del aplicativo. Dentro del transcurso del proyecto se fueron modificando según las demandas del cliente así como de las necesidades.

Nombre de la tarea	Fecha Inicio	Fecha Final	Duración(días)
Prototipo	20/02/2020	24/02/2020	4
Setup de la infraestructura de desarrollo	25/02/2020	10/03/2020	17
Desarrollo del *Backend*	11/03/2020	22/03/2020	11
Versión inicial del *Frontend*	23/03/2020	31/03/2020	8
Desarrollo *Frontend* Pedidos	01/04/2020	09/04/2020	9
Desarrollo *Frontend* Artículos	10/04/2020	17/04/2020	7
Desarrollo *Frontend* Empleados	18/04/2020	25/04/2020	7
Despliegue automatizado	26/04/2020	07/05/2020	12

Tabla 1.1: Plan de trabajo

Capítulo 2

Entorno de Desarrollo

Tras haber introducido un poco el trabajo en el capítulo anterior así como sus antecedentes y situación actual, en este capítulo nos centraremos en las tecnologías y herramientas utilizadas para llevar a cabo este proyecto.

2.1. Elección de las tecnologías

2.1.1. FrontEnd

Para el desarrollo del *frontend* de esta aplicación se usó el *framework* para JavaScript conocido como Vue, con su herramienta de creación de proyectos Vue-CLI [17]. Para la parte de diseño se ha utilizado el *framework* Vuetify [21], pensado principalmente para ahorrar tiempo de desarrollo aprovechando ya sus componentes con estilo de *Material Design*. La autenticación de la aplicación está externalizada bajo el servicio de *Firebase Authentication* [6].

2.1.2. BackEnd

Para el *backend* de la aplicación se usó NodeJS, para la creación de la API-Rest se usó Express. De cara a la base de datos se decidió externalizarla por temas de seguridad y escalabilidad, el servicio elegido fue MongoDBAtlas [11].

2.1.3. Gestión Proyecto

El proyecto empezó a gestionarse las tareas con la herramienta Trello, aunque con el tiempo fue acabando por no usarse, debido a que al ser una única persona la encargada del proyecto y tampoco tener mucha experiencia a la hora de dividir tareas sólo alargaba el proceso de desarrollo, al final sólo se usaba a modo de documentación. Para el control de versiones de la aplicación se ha usado Github.

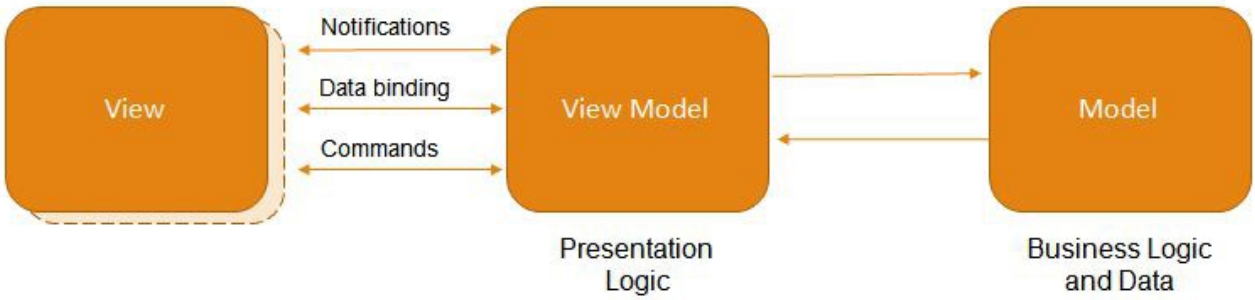


Figura 2.1: MVVM

2.1.4. Metodología

Para llevar a cabo la aplicación se usó una metodología TDD ya que lo que se buscaba principalmente con este trabajo era generar experiencia en el *testing* y despliegue. Sin embargo, el uso de esta metodología no duró demasiado, esto se debía a que no se tenían conocimientos de *testing* dentro de estos *frameworks* como para llevar un correcto control de las pruebas. Otro motivo que lo complicó fue el trato con el cliente, ya que a pesar de haber realizado un diseño inicial, este fue cambiando enormemente en aspectos no tan simples como diseño o apariencia, durante el desarrollo se pidieron varios cambios que exigían un cambio estructural fuerte en la aplicación, lo que supuso tener que desechar muchos test así como tiempo.

2.2. Arquitectura Software

2.2.1. Patrón de Diseño seleccionado

El patrón seleccionado para llevar a cabo la aplicación fue MVVM (Modelo - Vista, Vista - Modelo) ya que se buscaba que las partes del aplicativo estuvieran lo más desacopladas posibles para llevar a cabo un buen *testing* así como disponer de una buena capacidad para escalar. Las partes que lo componen son:

- El Modelo:

  Donde se encuentra todos los datos o lógica de la aplicación y debe ser completamente independiente de la vista.

- La Vista:

  Se encarga de mostrar la información a través de sus elementos visuales conteniendo sus propias funciones así como enlaces al modelo.

- Modelo De Vista:

  Coordina la vista y el modelo, la conexión que maneja con la vista se basa en los *"data bindings"*

Con este patrón y sus relaciones tenemos una interfaz reactiva y desacoplada que es junto lo que buscan *frameworks* como Vue 2.1.

2.2.2. Vue

Vue es un *framework* de JavaScript progresivo para construir interfaces de usuario basada en el patrón MVVM. Se basa en el diseño de componentes para permitir una arquitectura adaptable e incremental.

2.2.3. Vue CLI

Para crear el proyecto se usó VueCLI, que como indica su nombre es la interfaz de comandos de Vue , pensado para la creación rápida de aplicaciones con dicho *framework*, permitiéndote elegir las librerías que vas a usar y que quieres integrar en la propia creación.

2.2.4. WebPack

WebPack[22] es un empaquetador de módulos, que además de eso se encarga de la gestión de dependencias, conversión de formatos, servidor de desarrollo entre otras cosas. Se ha nombrado en este punto ya que Vue-CLI trabaja con esta tecnología.

2.3. Herramientas de Desarrollo

En este apartado comentaremos aquellas herramientas usadas en el desarrollo de la aplicación para su correcto funcionamiento.

2.3.1. MongoDB Atlas

MongoDBAtlas es un servicio en la nube que te proporciona una base de datos MongoDB, existen varios planes dependiendo de las necesidades, en este caso nos valdrá el gratuito.

Se ha decidido el uso de esta herramienta ya que de esta forma la base de datos estaría desacoplada de la aplicación, además de estar protegida por los estándares de seguridad de la propia empresa de MongoDB.

2.3.2. Firebase Authentication

Firebase es una plataforma para el desarrollo de aplicaciones web que ofrece varios servicios como:

- Storage
- Authentication
- Database
- Hosting
- Machine Learning

Para este proyecto se ha utilizado únicamente la parte de autenticación por motivos parecidos a lo anterior. Usando esta tecnología conseguimos desacoplar la autenticación del sistema, así como mejorar la seguridad de la aplicación.

2.3.3. Despliegue continuo: Digital Ocean

DigitalOcean[13] es un proveedor estadounidense de servidores, el cual usa el concepto de *droplet* para designar a cada uno de los servidores virtuales unos servidores virtuales privados que luego son los que alquilan.

El despliegue se ha decidió hacer en esta plataforma a través de la integración continua de TravisCI, a pesar de no ser compatibles, por comodidad con el entorno en proyectos anteriores.

2.3.4. Integración continua: TravisCI

TravisCI es un servicio de integración continua alojado. Se encarga de construir y probar proyectos alojados en plataformas como GitHub o BitBucket.

En esta imagen está representado el proceso de trabajo en el que participa TravisCI 2.2. Primero se realizaría un *push* en la rama *deploy* o *master*, dependiendo de la configuración del equipo(en el caso de este proyecto será *master*), TravisCi se descarga ese proyecto desde github y lo construye, tras esto ejecuta los test y *scripts* que se le hayan indicado, si todo ha salido bien procede a desplegarlo en la nube. En este caso se usó DigitalOcean como servicio de alojamiento, el cual no tiene integración con TravisCI por lo que hubo que realizar un *script* que se ejecutara en el proceso de contrucción de TravisCI.

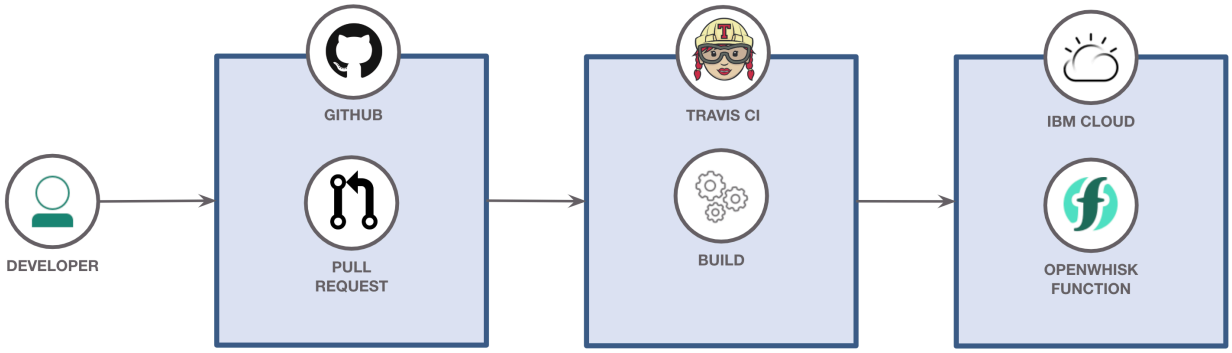


Figura 2.2: TravisCI

2.3.5. Control de versiones: GitHub

Para el control de versiones del proyecto se ha usado GitHub por comodidad, para ello se ha creado un repositorio privado para el *frontend* del aplicativo y otro para el *backend*. Justo después de la imagen se puede consultar ambos enlaces.

- ErpFrontend GitHub
- ErpBackend GitHub

2.3.6. Vue Router

Vue Router[18] es una dependencia de vue que permite crear *single page application* de manera simple. Permite crear rutas dinámicas y reactivas. En el siguiente trozo de código se puede ver la declaración de las rutas así como darle la información a Vue Router de qué componente es el que tiene que renderizar en cada momento.

```
import Vue from "vue";
import VueRouter from "vue-router";
import firebase from "firebase";
Vue.use(VueRouter);
const routes = [
  {
    path: "/",
    redirect: "/login"
  },
  {
    path: "/login",
    name: "login",
    component: () =>
      import("../views/login.vue")
  }
];
const router = new VueRouter({
  mode: "history",
  routes
});
```

Listing 2.1: index.js

```
    <v-app-bar app clipped-left>
      <v-app-bar-nav-icon @click.stop="drawer = !drawer" />
      <v-toolbar-title>ERP</v-toolbar-title>
    </v-app-bar>
    <v-content>
      <router-view />
    </v-content>
    <v-footer app>
      <span>&copy; 2020</span>
    </v-footer>

```

Listing 2.2: App.vue

En este trozo de código se puede ver dónde se le llama a vue-router para que renderize el componente oportuno.

2.3.7. Vuex

Vuex [19] es una herramienta para aplicaciones Vue.js que controla los estados y modificaciones de los componentes. Sirve como una tienda centralizada para todos los componentes de una aplicación, con reglas que aseguran que el estado solo pueda

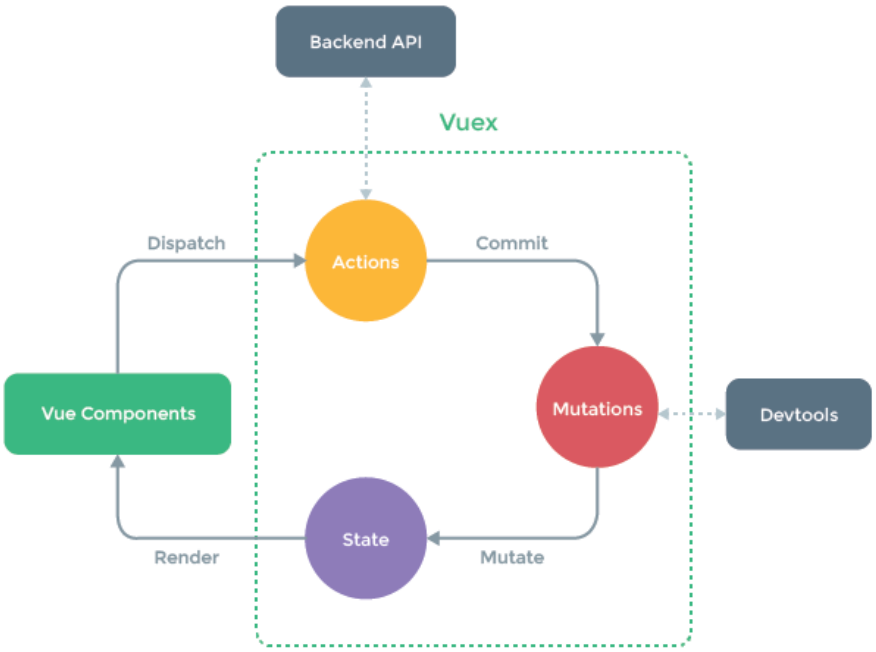


Figura 2.3: Vuex

mutarse de manera predecible. Vuex se compone de tres partes principales que se pueden observar en 2.3:

- State: Este apartado es donde se guardan los estados de las variables que serán renderizadas por parte de los componentes de Vue, funcionan como *data-binding*.
- Mutations: Estas son las funciones que trabajan y cambian el valor del *state*, si quieres modificar un valor del *state* es necesario ejecutar una mutación.
- Actions: Se encargan de las acciones asíncronas así como peticiones a la base de datos o a una API. También son capaces de ejecutar mutaciones siempre y cuando se ejecuten con la palabra reservada *Commit*.

2.3.8. Vuetify

Para la apariencia del proyecto se ha utilizado la librería Vuetify donde se pueden utilizar sus componentes y cambiar su estética mediante al modificación de sus parámetros.

2.4. Herramientas de calidad de código

2.4.1. EsLint-Prettier

ESLint[23] es una herramienta de análisis de código para identificar problemas de patrón, sintácticos o de calidad de código. Puede configurarse con algunas normas estándar o generar tus propias normas así como modificar las que existen.

Prettier[15] es un formateador de código, orientado a una mayor estructuración y legibilidad. Vue-CLI trae una opción para trabajar con ambas a la vez, que es lo que se ha llevado a cabo en este proyecto.

2.4.2. Babel

Babel[8] es un compilador JavaScript gratuito y de código abierto integrado también en Vue-CLI que permite convertir el código ECMAScript 2015+ en una versión de JavaScript compatible con versiones anteriores en los entornos donde sea necesario.

2.4.3. Jest

Jest[7] es una plataforma de prueba universal, con la capacidad de adaptarse a cualquier biblioteca o marco de JavaScript. En este proyecto se ha usado para la creación de las pruebas unitarias de la aplicación.

Para comprender mejor esta herramienta pasemos a hablar un poco sobre los conceptos que la definen, que son los siguientes:

- Test: Se encarga de dar una descripción de lo que se va a probar, así como marcar el espacio que ocupará el desarrollo de dicho *test*.
- Expect: Teniendo lo anterior es lógico pensar que nos falta un apartado que sirva para comprobar que los valores que obtenemos de nuestros componentes tengan como resultado lo que esperamos, en este punto es donde entra *expect* a comprobarlo con múltiples formas explicadas en la propia *wiki* de la herramienta.
- Mock: Cuando se tienen elementos que están acoplados entre sí o funciones que no pueden ejecutarse para realizar un test nace el concepto de *mock*. Se basa en generar una pseudofunción o pseudocomponente con el mismo comportamiento que el que se encuentra en el aplicativo original. Esto sirve por ejemplo, para no llenar de basura la base de datos a la hora de hacer los test. Para realizar esto, Jest hace uso de jest.fn, lo que le indica a Jest que se trata de una función *mocked*.
- SpyOn: Como último punto tenemos los SpyOn, este punto va justo al final de la realización del *test* y lo que busca es regenerar la misma situación de la que se partió. Esto se hace para que independientemente de cuántas veces se ejecute el *test*, el resultado sea siempre el mismo así como que empiece siempre desde el mismo punto.

2.4.4. Cypress

Cypress[4] es un *framework* de *testing end to end* de código abierto pensado para un desarrollo rápido y eficaz de estos *tests*, los cuales suelen conllevar un gran coste.

Después de haber hablado previamente de Jest, explicar Cypress se hace mucho más simple. Partimos de la misma base que con Jest, la diferencia es que ahora nuestro objetivo es probar el comportamiento de nuestra aplicación frente a un usuario. Para poder realizar esta tarea, Cypress nos aporta una serie de funcionalidades como poder acceder a elementos de DOM de manera sencilla, así como ejecutar funciones o introducir datos en ellos. Una vez haces esto el modo de proceder es igual que con Jest, primero se describe el *test* y luego se hace la prueba como por ejemplo: Obtener un *input* de un formulario, escribir dentro una palabra como hola, tras esto compruebas que el valor que se obtiene en el campo de texto al coger ese elemento sea igual a hola. Con esto conseguirías comprobar que el usuario puede introducir los datos en la aplicación sin error.

2.4.5. Codecov

Codecov[2] es una herramienta utilizada para saber el grado de *testing* que tiene tu aplicación. Calcula el ratio de cobertura viendo qué líneas son ejecutadas mientras corren los *tests* unitarios.

Para poner en funcionamiento dicha herramienta en este proyecto ha sido necesario configurar tanto el travis.yml como la configuración de Jest. En los siguientes fragmentos de código se mostrará los cambios realizados. Aquí se puede observar como se instala la herramienta codecov y se ejecuta mediante el comando de bash. Debido a la naturaleza de los repositorios es necesario aportarle un token identificativo.

Para configurar codecov con Jest sólo es necesario incluir las dos últimas en el archivo de configuración.

```
language: node_js
node_js: '12'
services:
- docker
install:
- npm install
- npm install -g codecov
before_install:
- bash scripts/install.sh
script:
- npm run test:unit:travis
- npm run build
after_success:
- bash <(curl -s https://codecov.io/bash)
- bash scripts/deploy.sh
env:
- global:
  - REMOTE_PATH=/data/www
  - REMOTE_USER=deploy_user
  - REMOTE_HOST=178.62.8.6
- CODECOV_TOKEN="83d8b72d-1645-4184-990c-fe18f85a6314"
```

Listing 2.3: .travis.yml

```
module.exports = {
  preset: "@vue/cli-plugin-unit-jest",
  coverageDirectory: "./coverage/",
  collectCoverage: true
};
```

Listing 2.4: jest.config.js

Capítulo 3

Descripción de la aplicación

En este capítulo se hará mención a todas las funcionalidades que se han desarrollado para esta aplicación así como unas imágenes de cómo se ve.

3.1. Usuarios

La aplicación realizada está orientada únicamente a los gerentes de cada restaurante, así que el número de usuarios no sería grande. Lo que si es importante es que cada usuario disponga de su propia tabla de artículos y pedidos, ya que no en todos los restaurantes/bares se sirve lo mismo o se trabaja con los mismos proveedores.

Se ha trabajado con el *feedback* directo de los empleados para crear una interfaz de usuario clara e intuitiva con la que se sintieran cómodos.

3.2. Flujo de la aplicación

Para entender mejor cada una de las figuras que se irán poniendo a continuación es necesario entender el flujo que llevará la aplicación.

Pongamos el caso de un gerente que acaba de entrar a trabajar y se da cuenta de que no dispone de suficientes refrescos para la semana que viene, en ese momento acudiría a la aplicación y generaría un pedido donde se vería reflejado la fecha de la creación y un número identificativo del mismo de forma automática, tras esto insertaría el número de refrescos del tipo x que necesita y cierra el pedido. Va pasando el día y se da cuenta de otros dos productos que necesita, volvería a abrir el mismo pedido y modificar las cantidades de los productos necesarios apoyándose en los filtros y buscadores que explicaremos posteriormente.

El turno del empleado acabó y se dispone a generar el pedido final para el proveedor. Primero vuelve a acceder a la aplicación, comprueba que todo está en orden y presiona el botón de enviar.

En este punto ya el pedido se habrá colocado en la colección de pedidos recibidos donde esperará ser marcado con un *checkbox* en el momento en el que se reciba.

3.3. Login

La aplicación partirá desde una página de *login* como la que se muestra en la figura 3.1 cualquier intento de acceso a otro dominio de la aplicación o de usuarios no autenticados será direccionado a esta página.

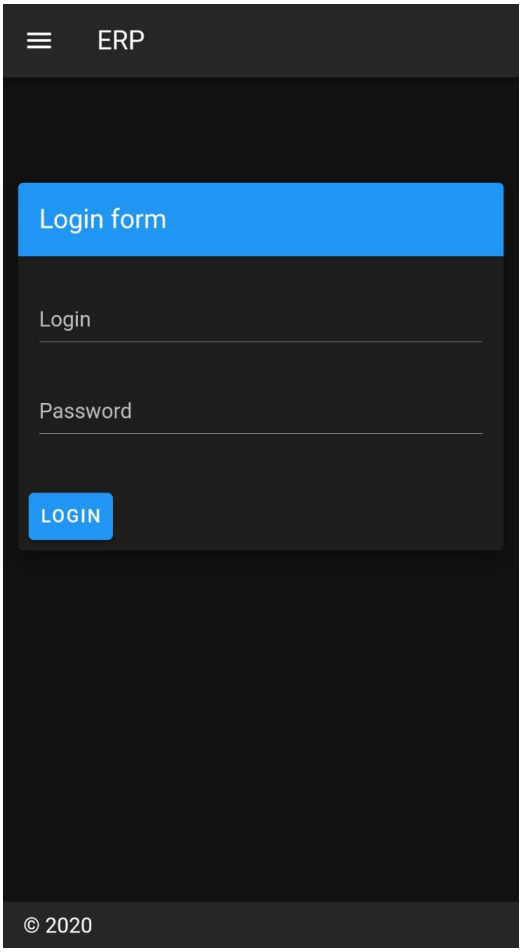


Figura 3.1: Pantalla De Login

3.4. Pantalla de Pedidos

Tras iniciar sesión en la aplicación, el usuario será redireccionado a la pantalla de pedidos donde se podrá ver, editar, borrar y enviar los pedidos a otra vista de la aplicación que se encarga de verificar la correcta llegada del pedido al gerente.Este proceso se hará al finalizar la jornada, debido a que ya se habrán apuntado los artículos necesarios.En la figura 3.2 se puede ver cómo sería el diseño de la pantalla. En las figura 3.3 se puede ver la interfaz de la edición y en 3.5 se puede ver la vista de la verificación del pedido.

Figura 3.2: Pantalla De Pedidos

3.5. Pantalla de Edición y Creación de Pedidos

Como se mencionó antes, en la figura 3.2 se ve la pantalla de edición de pedidos, si nos fijamos en la tabla desde la parte superior a la inferior podemos ver varias funcionalidades que han sido implementadas por petición del cliente. En la parte superior derecha disponemos de un buscador donde se puede buscar una cadena de texto cualquiera, Se quedarán en la tabla sólo aquellos artículos que contengan la cadena de texto buscada en alguno de sus componentes.

Pasando a las cabezas de cada categoría podemos ver cómo existe un número a su derecha, este número indica el orden por el que está siendo ordenados, permitiéndote generar una búsqueda dinámica. Hay que mencionar que esta vista ha sido reutilizada para la creación de pedidas, cambiando únicamente su comportamiento. En el caso de la edición se rescatan los datos previamente insertados en el pedido, en el caso de crearlo obviamente no.

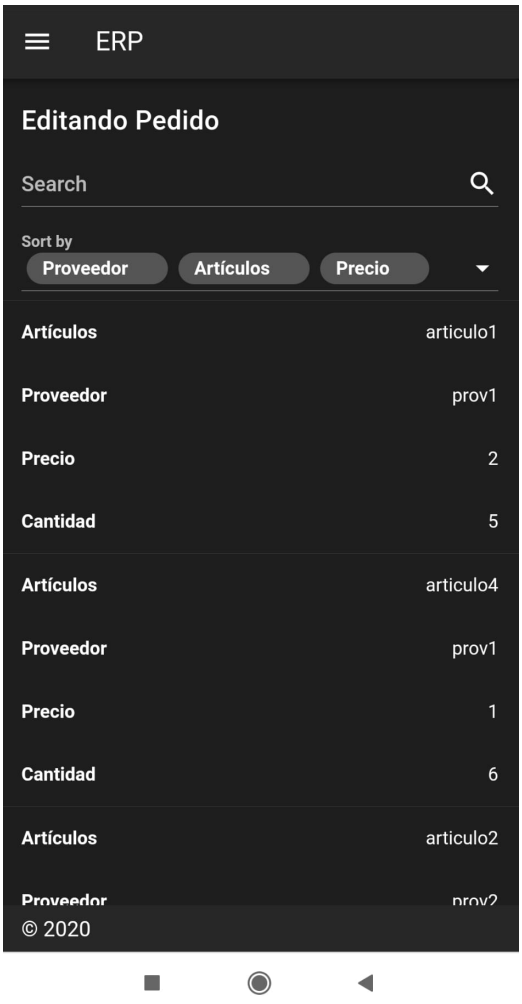


Figura 3.3: Pantalla De Editar Pedidos

3.6. Pantalla de Creación y Edición de Artículos

Pasando a explicar la última vista de la aplicación podemos ver aquí 3.4 el mismo sistema de tablas que en la anterior vista, con el filtrado y paginación. En esta vista podemos crear nuevos artículos presionando el botón, donde se nos abrirá un pequeño modal en el que incluiremos los datos, en caso de querer editarlo se abriría el mismo modal pero con los datos previamente elegidos, obviamente.

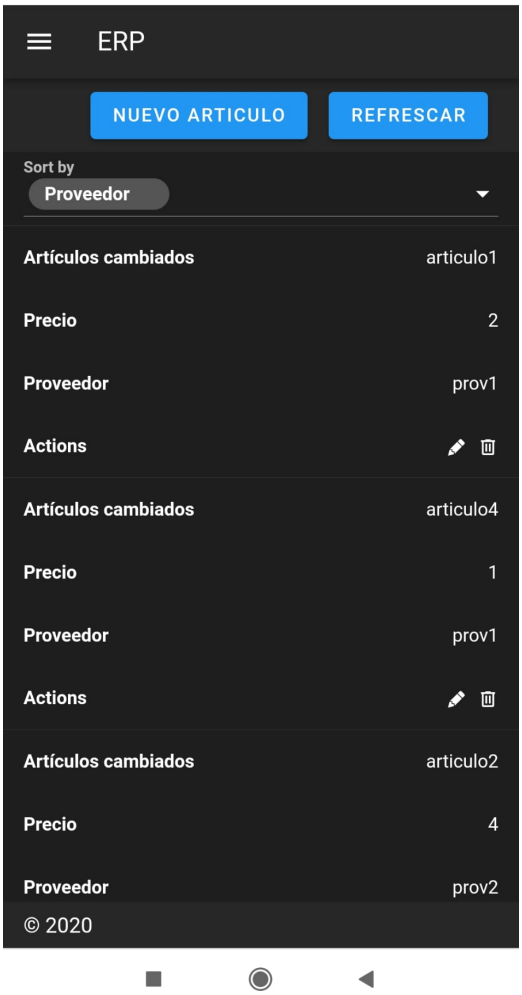


Figura 3.4: Pantalla De artículos

3.7. Pantalla Recibo de Pedidos

En esta vista 3.5 es donde se realizará el último proceso de la aplicación. El gerente podrá desplegar el pedido y revisar que se le ha entregado todos los artículos de manera satisfactoria. Tras esto se realizará un *click* en el *checkbox* de Recibido y el pedido se archivará en la base de datos, pero no estará activo a la vista para los gerentes.

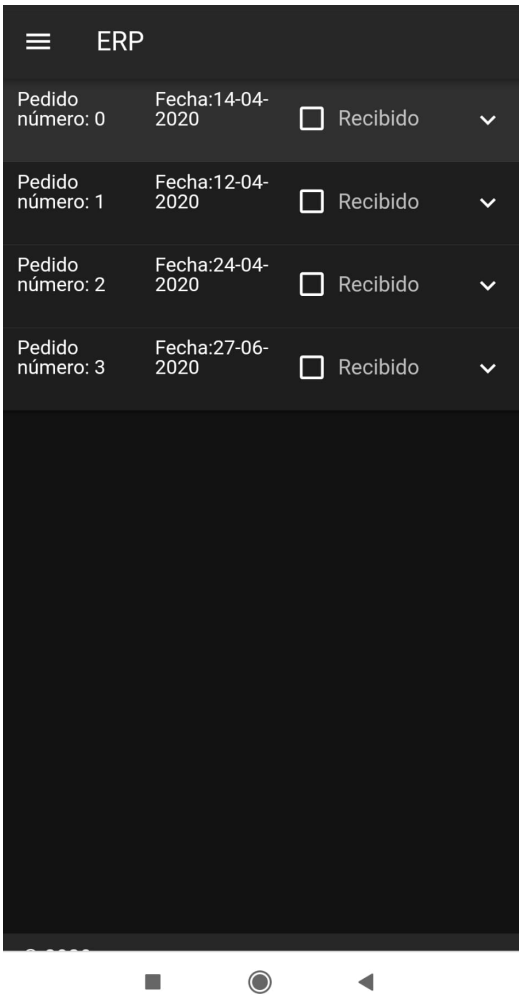


Figura 3.5: Pantalla de Pedidos Recibidos

Capítulo 4

Desarrollo del Proyecto

En el anterior capítulo se enseñó el objetivo y funcionalidades de la aplicación, en este nos centraremos más en el porqué se ha desarrollado de una forma u otra.

4.1. Primeros pasos

Como se había mencionado anteriormente el proyecto se inicializó con la herramienta de VueCLI, para iniciar la aplicación se instalaron las dependecias de babel, Linter/formatter, Vuex, Router y Unit Testing.

Después de eso se inicializó un repositorio en github para llevar un control de versiones de la aplicación. Se decidió comenzar por el desarrollo del *frontend*, en un principio se iba a utilizar el *framework* "Quasar"pero por falta de documentación se decidió el uso de Vuetify.

4.2. Desarrollo del FrontEnd

Durante el proceso de desarrollo del *frontend* se intentó llevar una construcción lo más desacoplada posible de los componentes, aunque más tarde se tuvo que refactorizar bastante el código. Para explicar la idea que se perseguía pondré el ejemplo de la vista del *login* ya que es la más simple del proyecto.

La ruta del *login* llama a este componente que contiene el componente encargado del login en sí se puede apreciar en este *listing* 4.1. En caso de querer añadir algo más en la vista del *login* se haría en esta sección.

En este trozo de código 4.2 se puede ver cómo se hace mención a dos componentes internos de la propia **"InterfazLogin"** llamados **"errorUserLog"** utilizado para notificar los errores al introducir las credenciales y **"botonInicioSesion"** encargado de lo llevar a cabo de iniciar sesión en la propia aplicación.

En la parte de *script* del mismo archivo 4.3 Se puede observar que se incluye los componentes previamente nombrados así como el uso de Vuex para la comunicación entre componentes. Al trabajar con componentes es necesario ,tarde o temprano, de

```
1
2 <template>
3   <InterfazLogin />
4 </template>
5
6 <script>
7 import InterfazLogin from "@/components/Login/InterfazLogin.vue";
8
9 export default {
10   components: {
11     InterfazLogin
12   }
13 };
14 </script>
15
```

Listing 4.1: login.vue

hacer uso de vuex junto a sus módulos ya que la comunicación de los componentes es necesaria desacoplarla primero antes de poder desacoplar la vista en sí.

Pasando a hablar de uno de los subcomponentes tenemos a **"errorUserLog"** 4.4. Se puede apreciar que es muy poco código debido a que lo único que tiene que hacer es controlar un estado en Vuex dentro del módulo de *login*.

Pasando a hablar del último de los subcomponentes tenemos a **"InterfazLogin"** 4.5. Como en el caso anterior podemos apreciar que el código no es demasiado extenso ya que sólo controla el sistema de sesión por firebase.

4.3. Testing

Para el *testing* se usaron dos tecnologías: **"Jest"** y **"Cypress"**, la primera para los test unitarios, la segunda para los test *end to end*, buscando la simpleza como en el anterior caso, mostraré el ejemplo del *login* para dar una muestra del desarrollo.

Como vemos en el listing 4.6 al usar vuex es necesario mockear parte del módulo para testear un cambio en el estado del *store* de vuex. Otras comprobaciones más simples son las del texto o la inicialización de las funciones.

La diferencia más notable entre este archivo y el anterior es la necesidad de incluir a vuetify en el *testing* ya que jest no es capaz de entender los componentes incorporados de Vuetify.

Para seguir con la filosofía de los ejemplos anteriores, veremos el ejemplo del uso de Cypress para simular a un usuario iniciando sesión en la plataforma 4.8. Podemos ver que las pruebas realizadas son de mayor nivel, se busca comprobar que los componentes sean visibles cuando deberían serlo, así como que no se puede acceder donde no se debe. También se comprueba la correcta ejecución de las funciones del componente.

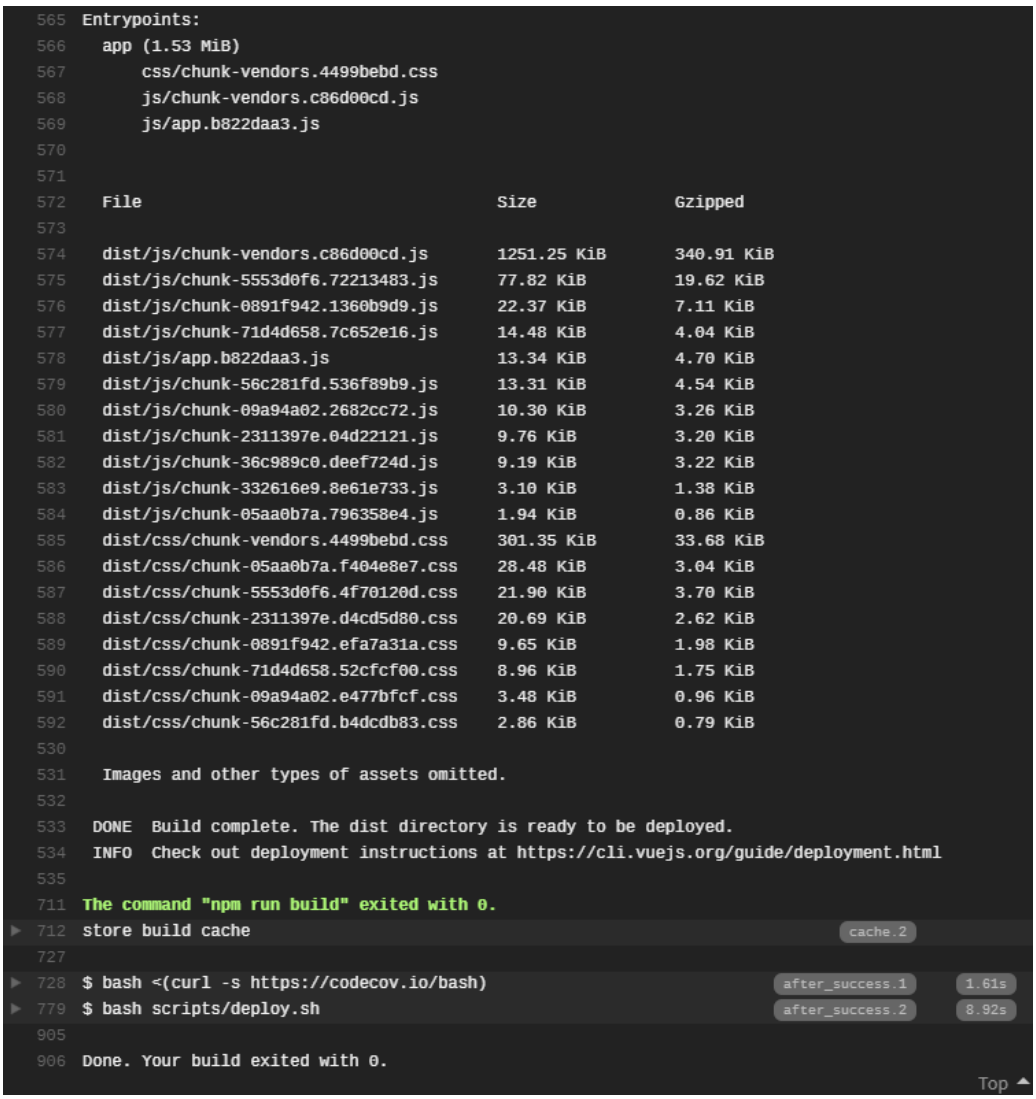


Figura 4.1: TravisCI Log

En la figura4.1 podemos observar una parte de los *logs* generados por TravisCI al construir y probar nuestra aplicación.

```
1
2  <template>
3    <v-app id="inspire">
4      <v-content>
5        <v-container fluid>
6          <v-row align="start" justify="center">
7            <v-col cols="12" sm="8" md="4">
8              <v-card class="elevation-12">
9                <errorUserLog />
10               <v-toolbar color="primary" dark flat>
11                 <v-toolbar-title>Login form</v-toolbar-title>
12                 <v-spacer />
13               </v-toolbar>
14               <v-card-text>
15                 <v-form @submit.prevent="login">
16                   <v-text-field
17                     v-on:input="updateUser"
18                     v-model="usuarioAux"
19                     label="Login"
20                     name="login"
21                     type="text"
22                   />
23
24                   <v-text-field
25                     v-on:input="updatePassword"
26                     v-model="contrasenaAux"
27                     id="password"
28                     label="Password"
29                     name="password"
30                     type="password"
31                   />
32                 </v-form>
33               </v-card-text>
34               <v-card-actions>
35                 <botonInicioSesion />
36               </v-card-actions>
37             </v-card>
38           </v-col>
39         </v-row>
40       </v-container>
41     </v-content>
42   </v-app>
43 </template>
44
```

Listing 4.2: InterfazLogin.vue

```
1
2  <script>
3  import Vue from "vue";
4  import Vuex, { mapState } from "vuex";
5  import errorUserLog from "./ErrorUserLog.vue";
6  import botonInicioSesion from "./BotonInicioSesion.vue";
7  Vue.use(Vuex);
8
9  export default {
10   components: {
11     errorUserLog,
12     botonInicioSesion
13   },
14   data() {
15     return {
16       usuarioAux: "",
17       contrasenaAux: ""
18     };
19   },
20   methods: {
21     ...Vuex.mapMutations("loginModule", ["SetUsuario", "SetContrasena"]),
22     updateUser() {
23       this.SetUsuario(this.usuarioAux);
24     },
25     updatePassword() {
26       this.SetContrasena(this.contrasenaAux);
27     }
28   },
29   computed: {
30     ...mapState("loginModule", ["usuario", "contrasena"])
31   }
32 };
33 </script>
34
```

Listing 4.3: InterfazLogin.vue

```vue
<template>
  <v-row v-if="this.error" align="center" justify="center">
    <v-overlay :absolute="absolute" :value="overlay" :opacity="opacity"
               :z-index="zIndex">
      <v-card class="d-flex justify-center align-center flex-column">
        <v-card-text class="d-flex justify-center align-center flex-column">
          <h1 class="red--text text--lighten-2">Error</h1>
          <p
            class="text-center mt-4"
          >El usuario....Por favor, inténtelo de nuevo.</p>
        </v-card-text>
        <v-btn color="blue" @click="changeError">Cerrar</v-btn>
      </v-card>
    </v-overlay>
  </v-row>
</template>
<script>
import Vue from "vue";
import Vuex, { mapState } from "vuex";
Vue.use(Vuex);
export default {
  data: () => ({
    absolute: true,
    overlay: true,
    zIndex: 1
  }),
  methods: {
    ...Vuex.mapMutations("loginModule", ["SetError"]),
    changeError() {
      this.overlay = false;
      this.SetError(false);
      this.overlay = true;
    }
  },
  computed: {
    ...mapState("loginModule", ["error"])
  }
};
</script>
```

Listing 4.4: errorUserLog.vue

```
1
2
3  <template>
4    <v-btn data-test="loginButtonTest" type="submit" @click="login">Login</v-btn>
5  </template>
6
7  <script>
8  import firebase from "firebase";
9  import Vue from "vue";
10 import Vuex, { mapState } from "vuex";
11
12 Vue.use(Vuex);
13
14 export default {
15   data: () => ({
16     absolute: true,
17     overlay: false
18   }),
19   methods: {
20     ...Vuex.mapMutations("loginModule", ["SetLogued", "SetError"]),
21     login() {
22       firebase
23         .auth()
24         .signInWithEmailAndPassword(this.usuario, this.contrasena)
25         .then(
26           () => this.$router.replace("MostrarPedidos"),
27           () => this.SetError(true)
28         )
29         .then(() => this.SetLogued(true));
30     }
31   },
32   computed: {
33     ...mapState("loginModule", ["logued", "usuario", "contrasena"])
34   }
35 };
36 </script>
37
38
39
```

Listing 4.5: InterfazLogin.vue

```
import { shallowMount, createLocalVue } from "@vue/test-utils";
import Vuex from "vuex";
import Vue from "vue";
import errorUserLog from "@/components/Login/ErrorUserLog.vue";
import loginModule from "../../../src/store/module/loginModule";
import Vuetify from "vuetify";
Vue.use(Vuetify);
const localVue = createLocalVue();
describe("ErrorUserLog.vue", () => {
  it("establece los datos correctos por defecto", () => {
    expect(typeof errorUserLog.data).toBe("function");
    const defaultData = errorUserLog.data();
    expect(defaultData.absolute).toBe(true);
    expect(defaultData.overlay).toBe(true);
    expect(defaultData.zIndex).toBe(1);
  });
});
describe("ErrorUserLog.vue render", () => {
  let actions;
  let store;
  beforeEach(() => {
    localVue.use(Vuex);
    localVue.use(Vuetify);
    store = new Vuex.Store({
      modules: {
        loginModule: {
          namespaced: loginModule.namespaced,
          state: {
            usuario: "",
            contrasena: "",
            error: true
          },
          actions,
          getters: loginModule.getters
        }
      }
    });
  });
  it("se ha montado correctamente", () => {
    const wrapper = shallowMount(errorUserLog, { store, localVue });
    const texto = wrapper.find("p");
    expect(texto.text()).toBe("El usuario...Por favor, inténtelo de nuevo.");
  });
  it("función changeError bien inicializada", () => {
    const wrapper = shallowMount(errorUserLog, { store, localVue });
    expect(typeof errorUserLog.methods.changeError).toBe('function')
  });
});
```


Listing 4.6: errorUserLog.spec.js

```
import Vuetify from "vuetify";
import Vue from "vue";
import BotonInicioSesion from "@/components/Login/BotonInicioSesion";
import { createLocalVue, shallowMount } from "@vue/test-utils";
const localVue = createLocalVue();
Vue.use(Vuetify);
describe("CustomCard.vue", () => {
  let vuetify;
  beforeEach(() => {
    vuetify = new Vuetify({
      mocks: {
        $vuetify: {
          lang: {
            t: val => val
          }
        }
      }
    });
  });
  it("should have a custom title", () => {
    const wrapper = shallowMount(BotonInicioSesion, {
      localVue
    });
    const title = wrapper.find('[data-test="loginButtonTest"]');
    expect(title.text()).toBe("Login");
  });
  it("should be firebase login function", () => {
    const wrapper = shallowMount(BotonInicioSesion, {
      localVue
    });
    expect(typeof BotonInicioSesion.methods.login).toBe('function')
  });
  it("establece los datos correctos por defecto", () => {
    expect(typeof BotonInicioSesion.data).toBe("function");
    expect(typeof BotonInicioSesion.methods).toBe("object");
    expect(typeof BotonInicioSesion.computed).toBe("object");
    const defaultData = BotonInicioSesion.data();
    expect(defaultData.absolute).toBe(true);
    expect(defaultData.overlay).toBe(false);
  });
});
```

Listing 4.7: BotonInicioSesion.spec.js

```
describe("La página de Login", () => {
  it("Carga Correctamente", () => {
    cy.visit("/login");
    cy.get(".col-sm-8").and("be.visible");
    cy.get(".v-app-bar > .v-toolbar__content > .v-toolbar__title").and(
      "be.visible"
    );
  });
  it("Te redirecciona desde cualquier Url", () => {
    cy.visit("/pedidos");
    cy.get(".col-sm-8").and("be.visible");
    cy.visit("/urldeprueba");
    cy.get(".col-sm-8").and("be.visible");
    cy.visit("/cualquierurl");
    cy.get(".col-sm-8").and("be.visible");
  });
  it("La contraseña tiene la propiedad password", () => {
    cy.visit("/login");
    cy.get("#password").type("123456789");
    cy.get("#password").should("have.value", "123456789");
    cy.get("#password").should("have.attr", "type", "password");
  });
  it("Fallo iniciar sesión debido a una contraseña o usuario erróneo", () => {
    cy.visit("/login");
    cy.get("#input-19").type("FalseUser@gmail.com");
    cy.get("#input-19").should("have.value", "FalseUser@gmail.com");
    cy.get("#password").type("123456789");
    cy.get("#password").should("have.value", "123456789");
    cy.get("#password").should("have.attr", "type", "password");
    cy.get(".v-card__actions > .v-btn").click();
    cy.get(".v-overlay__content > .v-card").and("be.visible");
    cy.get(".text-center").contains(
      "El usuario...Por favor, inténtelo de nuevo."
    );
    cy.get(".mb-4 > .v-btn__content").click();
    cy.get(".v-overlay__content > .v-card").and("not.be.visible");
    cy.location("pathname").should("not.eq", "/MostrarPedidos");
  });
  it("Puedes iniciar sesión sin problemas", () => {
    cy.visit("/login");
    cy.get("#input-19").type("manolo@gmail.com");
    cy.get("#input-19").should("have.value", "manolo@gmail.com");
    cy.get("#password").type("123456789");
    cy.get("#password").should("have.value", "123456789");
    cy.get("#password").should("have.attr", "type", "password");
    cy.get(".v-card__actions > .v-btn").click();
    cy.location("pathname").should("eq", "/MostrarPedidos");
  });
});
```

Capítulo 5

Complicaciones en el desarrollo

Durante el desarrollo de esta aplicación se han vivido una gran cantidad de problemas, la mayoría de ellos debido a tener trato con un cliente directo. La inexperiencia en general a la hora de llevar un proyecto sumada a la inexperiencia de tratar con clientes que no tienen porqué entender nada de lo que haces, se hizo bastante complicado.

5.1. Inexperiencia con el trato con clientes

En muchas ocasiones a lo largo del desarrollo del *frontend* el propio cliente demandaba una serie de cambios que retrasaban el proyecto así como su integración con otro tipo de herramientas.

Para solucionar esto hubo que cambiar varias veces la metodología con la que se estaba trabajando ya que a veces ocasionaban un estancamiento en el desarrollo demasiado grande y un cambio de enfoque del diseño.

Hubo que llevar a cabo muchas reuniones para poder sacar algo en claro sobre las necesidades de los propios gerentes, así como dejar claro los motivos por los que les recomendaba una cosa u otra. De hecho en las figuras 3.3 y 3.4 se ve un formato diferente de botones que está causado por lo mencionado anteriormente.

El desarrollo del prototipo fue caótico, porque por culpa de mi inexperiencia no supe guiar el desarrollo del prototipo lo que ocasionó graves problemas de diseño que se podrían haber arreglado desde un inicio con una mejor comunicación.

5.2. Inexperiencia con el framework vuetify

Al trabajar por primera vez con este *framework* hubo que dedicar mucho tiempo a aprender cómo se comportaban los componentes y cuáles eran sus limitaciones, ya que si elegía un componente erróneo para la tarea, las múltiples modificaciones necesarias para adecuar su funcionamiento suponían una cantidad de tiempo y esfuerzo innecesarias.

Debido al problema comentado anteriormente, muchas veces se seleccionaba el

componente pensando en resolver un problema y tras hablar con el cliente, este cambiaba de opinión, lo que generaba un cambio en el componente usado, en la mayoría de los casos, junto a una personalización para sus limitaciones.

5.3. Inexperiencia con la metodología TDD

Uno de los mayores motivos al comenzar este tfg fue la motivación por aprender más *testing*, ya que era uno de los puntos que encontraba más débiles en mi formación.

Al comenzar a trabajar intenté usar esta metodología, pero al ver que no avanzaba o que tardaba mucho en realizarla la descarté durante gran parte del desarrollo, lo que luego ocasiono tener que hacer grandes refactorizaciones de código que no hubieran tenido que hacerse si se hubiera mantenido la metodología en el desarrollo completo.

Otro de los motivos por los que este punto se ha llevado gran parte del tiempo es debido a los *mocks* necesarios para poder testear la aplicación, los cuales eran complicados de integrar en el estado actual de la página por culpa del mal diseño al abandonar tempranamente la metodología.

5.4. Organización de los componentes y módulos

Otro de los grandes problemas que hubo fue mi inexperiencia al crear y desarrollar un buen diseño de componentes y módulos, tuve que refactorizar un gran número de veces debido a darme cuenta de dependencias que había introducido sin darme cuenta.

Los módulos de Vuex también ocasionaron grandes problemas, ya que a priori se había creado un único *store*, conforme la aplicación fue creciendo vi la necesidad de separarlo. El problema de darte cuenta tarde es que la refactorización necesaria para cambiarlo es enorme y encima complicada dado el gran acoplamiento de los componentes.

Capítulo 6

Conclusiones y líneas futuras

Recapitulando todo lo mencionado en anteriores capítulos, la aplicación que se ha desarrollado dispone a día de hoy las siguientes funcionalidades:

- Servicio de Autenticación mediante Firebase.
- Base de datos particular para cada usuario alojada en MongoDBAtlas.
- Interfaz de pedidos(creación,edición y borrado).
- Interfaz de artículos(creación, edición y borrado).
- Control del estado del pedido(En creación, en espera, recibido).
- Testing end to end de la aplicación usando Cypress.
- Parte de la aplicación testeada con test unitarios usando Jest.
- Integración continua mediante TravisCI.
- Despliegue continuo en Digital Ocean.

El desarrollo de este proyecto ha sido bastante largo y complicado, ahora mismo veo el resultado que ha quedado al final y soy incapaz de entender dónde se ha ido el tiempo. Es un hecho que la mayor parte del proyecto he estado adaptándolo a las necesidades que le iban surgiendo al cliente, lo cual ha hecho que el desarrollo se haya vuelto muy caótico y estresante.

Por culpa de motivos como ese descarté la metodología TDD que estaba utilizando en un primer momento, lo que a futuro me conllevó aún más problemas. Está claro que para desarrollar un proyecto con clientes hay que tener las ideas claras y sobretodo tener claro el perfil profesional que tienes y representas. Al fin y al cabo el que quiere realizar un trabajo profesional eres tú, y quieres que esté lo mejor posible, por ello no deberías dejar que ocuparan todo lo que quisieran, aunque sean los clientes.

Todo el desarrollo mejoró en el momento en el que empezamos a tener reuniones y yo les explicaba los inconvenientes y ventajas de tener una interfaz de una forma u otra, así como el tratamiento de datos. Como conclusión diría que hay que darse mucho más

valor como desarrollador y dedicarle muchísimo más tiempo al diseño que a la propia codificación.

De cara al futuro de la aplicación, le faltan muchas cosas. Primero mejorar y ordenar el estado actual del *testing*, ya que quedan componentes sin testear y código muy acoplado; segundo punto importante sería la refactorización de todos los componentes que existen en la aplicación a día de hoy. Esos serían los puntos orientados al código, de cara a la metodología, sería seguir aplicando componentes de metodologías ágiles como las reuniones y validaciones rápidas con el cliente, así como continuar bajo la metodología TDD ya que sin duda es la más rápida.

Por último orientado a las funcionalidades que me gustaría que se llevaran a cabo la integración con los proveedores de cada establecimiento, así como una creación automática de mensajes para los proveedores ya que actualmente sólo acelera y facilita el proceso del control de pedidos así como el de su creación.

Capítulo 7

Summary and Conclusions

Summing up everything mentioned in previous chapters, the application that has been developed today has the following features:

- Authentication Service through Firebase.
- Private database for each user hosted in MongoDBAtlas.
- Order interface (creation, edition and deletion).
- Goods interface (creation, edition and deletion).
- Control of the order status (Creation, waiting, received).
- End to end testing of the application using Cypress.
- Part of the application tested with unit tests using Jest.
- Continuous integration through TravisCI.
- Continuous deployment to Digital Ocean.

The development of this project has been quite long and complicated. Right now I see the result that has been left at the end and I am unable to understand where the time has gone. It is a fact that most of the project I have been adapting it to the needs that were arising to the client, which has made the development very chaotic and stressful.

Due to reasons like that, I discarded the TDD methodology that I was using at first, which in the future led to even more problems. It is clear that to develop a project with clients you must have clear ideas and above all, be clear about the professional profile that you have and represent. At the end of the day, the one who wants to do a professional job is you, and you want it to be as good as possible, so you should not let them occupy everything they want, even if they are the clients.

The whole development improved the moment we started having meetings and I explained the disadvantages and advantages of having an interface in one way or another, as well as the data processing. In conclusion, I would say that you have to give

yourself much more value as a developer and spend a lot more time on design than on coding itself.

As future work, the application lacks many things and needs improvements. First, it needs an improvement and orderirng of the current state if tesing. There are untested components and highly coupled code. Second, a refactoring of all the components that exist in the application today needs to be made. After commenting on the points about the code. Regarding the methodology, I would say that it is necessary to continue applying agile methodologies. This type of practice increased both the speed of development and a faster validation with the client. Another necessary point would be to go back to development using the TDD methodology to save time in the future.

Finally, thinking on the functionalities that I would like to see in the applications, we can add the integration with the suppliers of each establishment, and an automatic creation of messages for the suppliers. Currently it only accelerates and facilitates the order control process as well as its creation.

Capítulo 8

Presupuesto

8.1. Coste del proyecto

A lo largo del proyecto se han llevado a cabo deferentes partes mecionadas en 1, que terminaron componiendo el cronograma.

Aunque no haya ninguna fase de diseño previamente mencionada, dado que es un proceso que se ha llevado a cabo durante todo el proyecto se unificara los puntos de: Prototipo, Setup de la infraestructura de desarrollo y versión inicial del Frontend en el punto de diseño.

Aplicando lo mismo mencionado anteriormente para le caso de análisis unificaremos los puntos de: Desarrollo Frontend Pedidos, Desarrollo Frontend Artículos, Desarrollo Frontend Empleados y Despliegue Automatizado en el punto de análisis.

Para el valor de programación por hora se ha considerado 14€ /hora. Dado que la aplicación sólo será usada por los gerentes de los propios restaurantes, no se necesita ni gran almacenamiento ni gran procesador por parte de los servicios. Esto nos permite usar el servicio gratuito de ambas plataformas sin problema.

Concepto	Tiempo	Coste Total	Coste/hora
Firebase	3 meses	0€	0€
MongoDBAtlas	3 meses	0€	0€
Análisis	210 horas	14€/hora	2940€
Diseño	150 horas	14€/hora	2100€

Tabla 8.1: Resumen de tipos

El presupuesto final para llevar a cabo este proyecto sería de un total de 5040€.

Bibliografía

[1] Atlassian. Trello: Software de administración de proyectos con interfaz web. https://trello.com/es. Accessed: 2020-03-15.

[2] CodeCov. Codecov: Herramienta de análisis de cobertura de código. https://codecov.io/. Accessed: 2020-03-15.

[3] Travis CI community. Travisci: Servicio de integración continua alojado que se utiliza para construir y probar proyectos de software alojados en github y bitbucket. https://travis-ci.org/. Accessed: 2020-03-15.

[4] Cypress. Cypress: Herramienta de testing orientados en test end to end. https://www.cypress.io/. Accessed: 2020-03-15.

[5] Express. Express: Infraestructura de aplicaciones web node.js. https://expressjs.com/es/. Accessed: 2020-03-15.

[6] Google. Firebase: Plataforma para el desarrollo de aplicaciones web y aplicaciones móviles. https://firebase.google.com/?hl=es. Accessed: 2020-03-15.

[7] Jest. Jest: Marco de pruebas de javascript. https://jestjs.io/. Accessed: 2020-03-15.

[8] Sebastian McKenzie. Babel: Trancompilador de javascript. https://babeljs.io/. Accessed: 2020-03-15.

[9] Microsoft. Github: Sistema de control de versiones. https://github.com/. Accessed: 2019-03-15.

[10] Microsoft. Microsoft dynamics: Es una línea de software erp y crm. https://dynamics.microsoft.com/es-es/. Accessed: 2020-03-15.

[11] MongoDB. Mongodbatlas: Servicio de base de datos en la nube. https://www.mongodb.com/cloud/atlas. Accessed: 2020-03-15.

[12] NodeJS. Nodejs: Entorno de ejecución para javascript. https://nodejs.org/es/. Accessed: 2020-03-15.

[13] Digital Ocean. Digitalocean: Proveedor de servidores virtuales privados. https://www.digitalocean.com/. Accessed: 2020-03-15.

[14] Odoo. Odoo: Software de gestión empresarial. https://www.odoo.com. Accessed: 2020-03-15.

[15] Prettier. Prettier: Formateador de código. https://prettier.io/. Accessed: 2020-03-15.

[16] SAP SE. Sap erp: Software de planificación de recursos. https://www.sap.com/spain/products/enterprise-management-erp.html. Accessed: 2020-03-15.

[17] Vue. Vue-cli: Sistema completo para el rápido desarrollo de vue.js,. https://cli.vuejs.org/. Accessed: 2020-03-15.

[18] Vue. Vue-router: La herramienta oficial para gestionar las rutas con vue. https://router.vuejs.org/. Accessed: 2020-03-15.

[19] Vue. Vuex: Herramienta de manejo de estados y componentes de vue. https://vuex.vuejs.org/.

[20] Vue.js. Vue.js: The progressive javascript framework. https://vuejs.org/. Accessed: 2020-03-15.

[21] Vuetify. Vuetify: Biblioteca vue ui con componentes de materiales. https://vuetifyjs.com/en/. Accessed: 2020-03-15.

[22] WebPack. Webpack: Paquete de módulos javascript de código abierto. https://webpack.js.org/. Accessed: 2020-03-15.

[23] Nicholas C. Zakas. Eslint: Herramienta de análisis de código estático. https://eslint.org/. Accessed: 2020-03-15.