Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención Ingeniería de Software)

Discovery2

Autor:
D. Guillermo Anta Alonso

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
(Mención Ingeniería de Software)

Discovery2

Autor:
D. Guillermo Anta Alonso

Tutores:
D. Benjamín Sahelices Fernández
D. Guillermo Santos Melgar
D. Roberto Torío Pastor
D. Marcos Orive Izarra

Resumen

El objetivo de este proyecto es la actualización de la herramienta web utilizada por los ingenieros de software de HP inc., la cual facilita la tarea de mantenimiento y gestión de los recursos conectados a las redes locales de sus oficinas. A lo largo de esta memoria se analizará y desglosará la estructura y funcionamiento de la misma, así como la interacción entre cada una de las partes. El objetivo de dicha herramienta es el de listar los recursos conectados a la red local y su estado, así como el de proporcionar al usuario una manera de interactuar con los mismos, añadiendo o eliminando información, creando eventos o modificando sus parámetros.

Abstract

The objective of this project is to update the web tool used by HP inc software engineers, which facilitates the task of maintenance and management of resources connected to the local networks of their offices. Throughout this document, the structure and operation of the tool will be analyzed and broken down, as well as the interaction between each of the two parts. The objective of this tool is to list the resources connected to the local networks and their status, as well as to provide the user with a way to interact with them, adding or removing information, creating events or modifying their parameters.

Tabla de contenidos

Resumen 1

Abstract 1

Tabla de contenidos 2

1. Introducción y objetivos 5

1.1. Motivación 5

1.2. Objetivos 5

1.3. Metodología 5

1.4. Resumen 5

2. Contexto tecnológico 6

2.1. MongoDB 6

2.1.1. Mongoose 6

2.2. Express 6

2.2.1. Connection-history-api-fallback 7

2.2.2. CORS 7

2.2.3. Helmet 7

2.2.4. Role-acl 7

2.3. Vue 8

2.3.1. Vuetify 8

2.3.2. Vuex 8

2.3.3. Vue-moment 9

2.3.4. Vue-router 9

2.3.5. Vue-socket.io 9

2.3.6. Vuetify-datetime-picker 9

2.4. Node 9

2.4.1. Bcrypt 10

2.4.2. ff-napi 10

2.4.3. ref-napi 10

2.4.4. Json-diff 10

2.4.5. Jsonwebtoken 10

2.4.6. LdapJS 10

2.4.7. Winston 10

2.4.8. Xml-JS 10

2.5. Socket.io 10

2.6. Babel 11

2.7. Nginx 11

3. Análisis y diseño 12
3.1. Issues 12
3.2. Requisitos 12
3.3. Whishlist 13
3.4. Current issues 13
3.5. Base de datos 13
3.5.1. Colecciones 14
3.5.2. Esquemas 14
3.6. Estructura de la aplicación 16
3.7. Front-end 16
3.7.1. Vistas y componentes 17
3.7.2. Componente Login 17
3.7.3. Componente Alert 17
3.7.4. Componente CreateReservation 17
3.7.5. Componente Navigation 18
3.7.6. Componente PrinterComponent 18
3.7.7. Componente PrinterDetails 18
3.7.8. Componente PrinterLog 19
3.7.9. Componente ReservationComponent 19
3.7.10. Componente UserComponent 19
3.7.11. Aplicación 20
3.7.12. Vista Calendar 20
3.7.13. Vista Configuration 21
3.7.14. Vista PrinterList 21
3.7.15. Vista ReservationList 22
3.7.16. Vista UserList 23
3.7.17. Gestión de estado y conexión con la API 23
3.8. Back-end 24
3.8.1. Servidor Express 24
3.8.2. HPDiscovery 25
3.9. Comunicaciones 26
4. Implementación 28
4.1. Inicialización del servidor 28
4.2. Discovery Adapter 28
4.3. Implementación de roles 29
4.4. Auto actualización y gestión de recursos 29
4.5. Eventos desde el servidor 30
4.6. Solicitud y descarga del log de una impresora 31

4.7. Jsonwebtoken 31
4.8. Logger 31
4.9. Registro de cambios de impresora 32
5. Conclusiones 33
Bibliografía y referencias 34
Anexos 37
Equipo 37
Manual de instalación 37
MongoDB 37
Node 38
Discovery2 38
Configuración 39
Nginx 39
Manual de usuario 40
1. Ejemplo 1 - Gestión de usuarios y configuración de la aplicación: 40
2. Ejemplo 2 - Búsqueda de recursos, creación y modificación de reservas propias: 42
3. Ejemplo 3 - Modificación de recursos y reservas: 44
4. Ejemplo 4 - Eliminación de recursos, reservas y usuarios: 44

1. Introducción y objetivos

1.1. Motivación

Este proyecto surge como propuesta del observatorio tecnológico de HP en colaboración con la Universidad de Valladolid, motivado por la necesidad de actualización de la herramienta existente.

Como se ha comentado anteriormente, esta herramienta web utilizada por los ingenieros de software de HP requiere de una nueva versión en la que se amplíe la funcionalidad y se resuelvan los problemas existentes en la versión actual.

La herramienta en cuestión pretende simplificar el flujo de trabajo de los ingenieros de HP, escaneando la red local de la oficina y recuperando los recursos conectados a esta, permitiéndoles interactuar con ellos, así como modificar, añadir o eliminar información.

1.2. Objetivos

El objetivo de esta herramienta es mostrar y actualizar la información necesaria de los recursos utilizados y proporcionar de manera sencilla una forma de interacción.

Para ello la herramienta deberá descubrir los dispositivos conectados a una cierta red y mostrar la información detallada de los mismos, también deberá proporcionar una manera de registrar que un cierto usuario va a trabajar con un recurso de la red, para evitar que se originen posibles conflictos al intentar dos usuarios trabajar sobre el mismo recurso. Para ello se deberá mantener la información actualizada en todo momento, esto quiere decir que el estado reflejado en la herramienta debe de ser el mismo que presente el recurso.

Adicionalmente se proporcionará un control de acceso basado en roles para poder separar la funcionalidad y los datos mostrados en función del tipo de usuario actual.

1.3. Metodología

Se ha elegido SCRUM como forma de trabajo, por lo que la planificación ha quedado dividida en sprints, cada uno de ellos con ciertas tareas asignadas y con una duración aproximada de dos o tres semanas. Al final de cada uno de estos sprints se ha agendado una reunión para revisar las tareas completadas y debatir cuáles eran las tareas a completar en el siguiente sprint y de qué forma se tenían que llevar a cabo.

1.4. Resumen

En los capítulos siguientes se detalla en profundidad los requisitos y las correcciones necesarias que se han realizado y la manera en la que estos se han llevado a cabo. También se realiza un desglose de la estructura de la herramienta, así como de las tecnologías utilizadas para implementarla. Como apartado final se proporciona un manual de instalación con las instrucciones necesarias para que la aplicación esté operativa y funcionando de una manera correcta en cualquier sistema y un manual de usuario en el que se muestra el comportamiento y como se debe interactuar con la herramienta. También se listan una serie de flujos de trabajo ejemplificando cómo se realizan las operaciones más comunes.

2. Contexto tecnológico

En este apartado se detallan las tecnologías elegidas para el desarrollo del proyecto, como se ha comentado anteriormente, se ha decidido usar el stack tecnológico MEVN. Este conjunto de tecnologías está formado por MongoDB como base de datos, Express que junto a Node nos permitirá crear una API y Vue que es el framework con el que construiremos la parte del cliente.

Esta herramienta será una single page application o SPA, lo que quiere decir que no será necesario navegar a través de distintas rutas para acceder a las distintas vistas de la misma, aunque sí que se proporciona un sistema de rutas e historial.

Las ventajas de las SPA frente a las páginas web tradicionales es que no es necesario cargar distintos recursos web para utilizar la aplicación, es decir la aplicación se genera bajo una única dirección web y solo es necesario cargarla al inicio de la misma, lo que resulta en una sustancial mejora de rendimiento frente a las webs tradicionales.

2.1. MongoDB

MongoDB es un sistema de base de datos, de código abierto y orientado a documentos, a diferencia de las bases de datos tradicionales, MongoDB guarda los datos en documentos que a su vez están agrupados en colecciones. Estos documentos se almacenan mediante esquemas dinámicos utilizando un tipo de documento especial llamado BSON. Un grupo de colecciones constituirá una base de datos.

Esto se traduce de forma directa en una gran flexibilidad, rapidez y facilidad a la hora de realizar operaciones. Aunque la base de datos proporciona esta flexibilidad es necesario definir un modelo para cada tipo de documento a almacenar, para esto utilizaremos los esquemas proporcionados por Mongoose, herramienta de la que hablaremos más adelante.

El uso de este sistema de base de datos fue el único requisito tecnológico impuesto, ya que la anterior aplicación también lo utilizaba.

2.1.1. Mongoose

Es una biblioteca que nos permite crear esquemas, para definir los documentos que vamos a almacenar en la base de datos, las instancias de estos esquemas serán modelos, la unidad de datos con la que trabajaremos. En resumen, Mongoose nos permite definir los datos que guardaremos en la base de datos y facilita las tareas de comprobación y validación de los mismos.

2.2. Express

Es un framework de aplicaciones web para Node, que nos permitirá construir una API que será la encargada de exponer la funcionalidad desarrollada en el back-end para poder utilizarla desde la parte del cliente.

También proporciona una serie de funcionalidades que nos han sido especialmente útiles a la hora de desarrollar el back-end de la aplicación, como un router que nos permitirá asociar una funcionalidad con una determinada ruta y operación http.

2.2.1. Connection-history-api-fallback

Aunque nuestra aplicación sea una SPA y por lo tanto no sea necesario navegar mediante rutas entre las distintas vistas, se han definido relaciones entre rutas y componentes, para permitir su acceso directo. Con este middleware de express, mantenemos un historial de las rutas accedidas, permitiendo al usuario utilizar la funcionalidad de retroceder a la vista anterior o avanzar a una vista ya accedida.

2.2.2. CORS

Es un middleware que permite una comunicación segura entre cliente y servidor, trabaja con las peticiones HTTP de clientes que solicitan recursos de un dominio distinto al que pertenecen. En resumen, CORS añade cabeceras a las peticiones HTTP, detallando desde que orígenes se permite el acceso a los recursos del servidor.

2.2.3. Helmet

Es otro middleware de express que añade cabeceras a las peticiones HTTP, incrementando la seguridad de la aplicación. Morgan
Es un middleware de express que nos permitirá loguear las peticiones HTTP recibidas, los parámetros y los resultados de las mismas.

2.2.4. Role-acl

Es una biblioteca que de manera sencilla permite definir un sistema de roles para nuestra aplicación. Su funcionamiento es el siguiente, se definen los tipos de recursos, los roles y las operaciones disponibles, luego estos tres parámetros se relacionan para crear el acceso basado en roles. Cuando un usuario quiere realizar una operación contra un recurso, se consulta si su rol está relacionado con esa operación y ese recurso, permitiéndole o no realizar la operación. En la siguiente tabla se detallan estas relaciones.

		ROLES		
RESOURCES	OPERATIONS	BASIC	MAINTAINER	ADMIN
PRINTERS	CREATE	X	v	v
	READ	v	v	v
	UPDATE	X	v	v
	DELETE	X	X	v
RESERVATIONS	CREATE	v	v	v
	READ	v	v	v
	UPDATE	OWN	v	v
	DELETE	OWN	OWN	v
USERS	CREATE	X	X	v
	READ	X	v	v
	UPDATE	X	X	v
	DELETE	X	X	v
CONFIG	CREATE	X	X	v
	READ	v	v	v
	UPDATE	X	X	v
	DELETE	X	X	v

2.3. Vue

Vue es un framework de JavaScript de código abierto modelo-vista-vista-modelo que nos permite construir interfaces de usuario y SPAs, se caracteriza por el uso de un DOM virtual, por su alta reactividad, es decir cuando se produce un cambio en el estado de la aplicación este se refleja rápidamente en la vista.

Vue permite desarrollar componentes, los cuales cuentan con tres partes, una plantilla o template, escrita en HTML que será la que describa la estructura del componente, un script que será el encargado de aportar la funcionalidad del componente, y una parte de estilo que será donde se apliquen los estilos mediante CSS u otros lenguajes de estilos al componente.

También se proporcionan ciertas directivas, como condicionales o bucles, que aplican directamente a la plantilla de los componentes, facilitando la tarea de modificar la vista en función del estado de la aplicación.

Junto a Vue se han utilizado otras tecnologías que se detallan más adelante.

2.3.1. Vuetify

Es un framework de material design para Vue, es decir Vuetify proporciona componentes Vue siguiendo los estilos de diseño de material design, estos componentes son totalmente personalizables y permiten una adaptación sencilla y rápida para crear una aplicación. Se ha elegido Vuetify frente a otros frameworks como BoostrapVue o ElementUI, por la gran variedad de componentes y la abundante documentación que se proporciona en su página web. Y ha sido una pieza verdaderamente importante en el desarrollo de la aplicación ya que ha agilizado el proceso al no tener que invertir tanto tiempo en dar estilo a los componentes creados.

La configuración de Vuetify se encuentra en la ruta /client/src/plugins/vuetify.js, es aquí donde se definen los temas de la aplicación, así como el idioma a utilizar.

2.3.2. Vuex

Es una biblioteca que facilita la tarea de gestionar el estado de la aplicación centralizando y proporcionando una manera de acceder al mismo de manera reactiva. Sin Vuex los datos de la aplicación tendrían que ir viajando a través de los componentes de la aplicación, lo que dificulta el detectar los cambios y reflejarlos en la vista.

Vuex proporciona una forma de acceso a los datos reactiva, es decir cada componente accede solo a la parte del estado de la aplicación que necesita, este estado está definido en la carpeta store/, en él encontramos diferentes módulos que separan los modelos de datos en archivos distintos. Cada módulo está compuesto por:

- State: En esta parte del módulo se registra la información que se va a manejar, a esta parte del módulo se accede desde los componentes a través de las propiedades computadas definidas mediante mapState.

- Mutations: Las mutations registran operaciones síncronas contra el estado del módulo, estas operaciones reciben como parámetro el estado y lo modifican. Estas se utilizan en los componentes de la aplicación a través de los métodos definidos mediante mapMutations.

- Actions: Las acciones nos permiten realizar operaciones asíncronas como recuperar datos consumiendo una API, cuando estas operaciones han sido completadas se modifica el estado del módulo haciendo uso de las mutaciones.

- Getters: Los getters proporcionan una forma de acceso al estado mediante una operación, es decir si se necesita recuperar una lista y luego filtrarla, en vez de realizar esta operación a nivel de componente, lo hacemos a nivel de módulo, estos getters, al igual que el state, son propiedades computadas que reaccionan a los cambios de estado, se utilizan en los componentes a través de las propiedades computadas definidas en mapGetters.

2.3.3. Vue-moment

Es una adaptación de la conocida biblioteca momentJS, que proporciona una manera sencilla de manejar fechas y tiempos.

2.3.4. Vue-router

Aunque la aplicación sea una SPA se ha utilizado un router, para proporcionar un acceso directo mediante una dirección web a ciertas vistas de la aplicación. Encontramos la configuración de este módulo y las relaciones existentes entre rutas y componentes en el archivo /client/src/routes/index.js

2.3.5. Vue-socket.io

Proporciona una adaptación del cliente de socket.io para Vue, el cual permite establecer una conexión con el servidor y escuchar los eventos que este emite, para realizar las operaciones necesarias y mantener en todo momento el estado de la aplicación sincronizado con la base de datos.

La configuración de socket.io en la parte del cliente se encuentra bajo la ruta /client/src/plugins/socket.js, es importante actualizar el parámetro de la conexión que registra la dirección y el puerto en el que se encuentra el servidor.

2.3.6. Vuetify-datetime-picker

Es una composición de componentes de Vuetify que proporciona una manera sencilla de escoger una fecha y hora determinada.

2.4. Node

Node es un entorno de ejecución multiplataforma que nos permite ejecutar código JavaScript fuera del navegador, dándonos la posibilidad no sólo de desarrollar la parte front-end si no también el back-end. Hace uso del entorno de ejecución V8 que fue creado para Google Chrome.

También proporciona funcionalidades como un sistema de archivos o la posibilidad de utilizar bibliotecas que hayan sido escritas en otros lenguajes de programación, que nos serán especialmente útiles para el desarrollo de esta aplicación.

Junto a Node se utiliza su gestor de paquetes, npm que nos permitirá utilizar e integrar en nuestra aplicación funcionalidades desarrolladas por otras personas de una manera sencilla y rápida.

2.4.1. Bcrypt

Bcrypt se ha utilizado para encriptar las contraseñas de los usuarios, esto nos permite almacenar una cadena generada encriptando la contraseña en vez de almacenar la contraseña como texto plano.

2.4.2. ff-napi

Ha sido una pieza fundamental de la parte back-end, ya que ha permitido integrar la funcionalidad HPDiscovery, de la que hablaremos más adelante, que ha sido proporcionada por HP, con el servidor de nuestra aplicación haciendo posible utilizar una biblioteca C++ junto a un servidor JavaScript.

2.4.3. ref-napi

Se ha usado como complemento a la comentada anteriormente y ha permitido crear las definiciones de tipos necesarias para definir en JavaScript la funcionalidad que ejecuta la biblioteca HPDiscovery.

2.4.4. Json-diff

Este paquete para Node permite realizar de una manera sencilla una comparación entre dos objetos JavaScript, ha sido útil para registrar los cambios realizados en los recursos del sistema.

2.4.5. Jsonwebtoken

También conocido como jwt, se ha utilizado para implementar un acceso basado en tokens, es decir cuando un usuario inicia sesión se genera un token único que contiene entre otras cosas los permisos del usuario. Cuando este usuario realice una petición desde el cliente hacia el servidor, adjuntará este token en la cabecera “x-access-token”, que será comprobado para determinar si la operación es o no permitida.

2.4.6. LdapJS

LdapJS ha facilitado la tarea de implementar un inicio de sesión contra un servidor LDAP, como parte de la configuración del servidor se definen, la dirección del servidor LDAP, el nodo del usuario administrador y la contraseña del mismo.

2.4.7. Winston

Para esta aplicación se ha implementado un sistema de logs personalizado, el cual se ha basado en el popular logger Winston.

2.4.8. Xml-JS

Esta funcionalidad ha servido para parsear los documentos xml devueltos por la biblioteca HPDiscovery y convertirlos en objetos JavaScript.

2.5. Socket.io

Aunque no se ha mencionado hasta ahora, Socket.io ha sido una pieza fundamental, ya que ha permitido implementar de una forma sencilla un sistema de SSE (server sent event), para que el back-end pueda emitir eventos hacia el cliente avisando de que los datos han cambiado en la base de datos, facilitando la tarea de desarrollar una aplicación en tiempo real.

Cuando los datos cambian en la base de datos se emite un evento, para que el front-end realice una petición y recupere los datos con las nuevas modificaciones, permitiendo reflejar en todo momento el estado real en el cliente.

2.6. Babel

Nos permite utilizar la sintaxis de JavaScript ES6, que aún no es compatible con todos los navegadores. Realiza una traducción de la sintaxis ES6 a código JavaScript interpretable por el navegador. Se ha utilizado tanto en la parte de cliente como en la de servidor.

2.7. Nginx

Como servidor web se ha utilizado Nginx, es un servidor web, gratuito, de código abierto y multiplataforma. Su arquitectura es asíncrona y está basada en eventos, es fácilmente escalable y proporciona un gran rendimiento.

3. Análisis y diseño

3.1. Issues

A continuación, se listan los problemas existentes con la versión actual y su prioridad, los cuales han sido solucionados como se detalla posteriormente.

1. El algoritmo de auto actualización de la aplicación parece no funcionar correctamente, la mayoría de las veces que se actualizan los recursos manualmente se perciben cambios en el estado de la aplicación. El problema que registra este issue es que el estado mostrado de los recursos de la herramienta no se corresponde con el estado real del recurso. Es decir, la tarea encargada de recoger y actualizar el estado de los recursos mostrados no estaba funcionando, por lo que al realizar una actualización manual se percibía un cambio de estado. (Prioridad alta).
2. El filtro seleccionado se borraba cuando se recargaba la página, este issue no ha sido solucionado de manera implícita, es decir el problema ha dejado de existir gracias a la elección de las tecnologías utilizadas. (Prioridad media).
3. El campo hostname hace distinción entre mayúsculas y minúsculas, este campo es muy relevante a la hora de identificar los recursos de la red, por lo tanto, es importante que no se haga distinción entre “HOSTNAME” y “hostname”. (Prioridad media).
4. Los recursos con la misma dirección IP deben ser resaltados para facilitar tareas de mantenimiento. Este problema surge a causa de la asignación dinámica de IP a los recursos de la red, es decir, la dirección IP de un recurso que permanece inactivo o está apagado durante un cierto tiempo, puede ser asignada a un nuevo recurso al conectarse a la red. Aunque esto no supone ningún malfuncionamiento es necesario informar de ello. (Prioridad baja).
5. Problemas de rendimiento y fiabilidad de la aplicación. Aunque este issue en principio es muy abstracto se ha encontrado la causa y se ha proporcionado una solución a la misma. (Prioridad media).
6. Al actualizar la página en la vista de calendario se redirige a la página principal. Al igual que el issue número 2 este problema se ha solucionado gracias a la tecnología utilizada. (Prioridad baja).
7. Las alertas y mensajes en el lado del cliente ocultan el campo de búsqueda de la aplicación. Cuando aparece un mensaje informativo, este tapa el campo de búsqueda imposibilitando la tarea de buscar un cierto recurso en la aplicación. (Prioridad baja).

Más adelante se detalla cuáles han sido las soluciones a cada uno de estos issues.

3.2. Requisitos

A continuación, se listan los nuevos requisitos que se han de cumplir en la nueva versión de la aplicación y su prioridad.

1. Nueva vista en la que se muestre los recursos ya filtrados con las reservas existentes para el día actual, esta vista debe proporcionar una forma de reservar un recurso de manera rápida y con pocos clicks. (Prioridad alta).
2. Posibilidad de añadir recursos manualmente. (Prioridad media).
3. Mostrar y resaltar en gris los recursos que han permanecido en estado “non-connected” durante un largo periodo de tiempo. (Prioridad baja).
4. Resaltar los recursos seleccionados. (Prioridad baja).
5. Proporcionar una forma de identificación utilizando la cuenta de HP y validando contra su servidor LDAP. (Prioridad alta).

6. Los usuarios sin rol especial deben ser capaces de actualizar sus reservas, pero no las de otros usuarios. (Prioridad alta).
7. Creación de un rol de administrador para configurar los parámetros de funcionamiento de la herramienta. Este rol debe tener permisos para poder eliminar reservas y recursos de la herramienta. (Prioridad media).
8. Creación de un rol de mantenedor que será el encargado de actualizar los diferentes recursos y reservas, independientemente de si son o no suyas. (Prioridad media).
9. Mostrar las alertas de los recursos. (Prioridad media).
10. Recuperar el log con los errores de cada recurso. (Prioridad baja).
11. Recuperar el “part-number” si está disponible. (Prioridad media).
12. Auto actualizar la vista de usuario si una nueva reserva se ha creado. (Prioridad alta).
13. Avisar al usuario si la reserva actual no se puede crear. (Prioridad alta).
14. El nombre del recurso reservado debe aparecer en la vista del calendario. (Prioridad baja).
15. Registrar las operaciones y el usuario que las ha realizado de cada recurso. (Prioridad alta).
16. El formulario de reserva debe auto rellenarse con la fecha actual para agilizar el proceso de reserva. (Prioridad baja).
17. Guardar y mostrar el tipo de recurso. (Prioridad media).

3.3. Whishlist

Los requisitos que se listan a continuación no son imprescindibles, pero sí deseables:

1. Mapa de la oficina con los recursos mostrados según su ubicación física
2. Despertar un recurso desde la aplicación.
3. Actualizar el firmware de un recurso desde la aplicación.

Ninguno de ellos se ha podido llevar a cabo debido a restricciones temporales y la complejidad de los mismos.

3.4. Current issues

Estos son los issues conocidos en el momento de finalizar el proyecto:

1. La columna updated y created de la vista “List” no muestra la reactividad a los cambios que debería, aunque el valor cambia internamente, no se renderiza el campo al detectar el cambio. Una posible solución sería cambiar la manera en la que se añade esta columna a la tabla, actualmente se hace definiendo un campo personalizado <template/> dentro de ella.
2. Al cambiar de sesión con un usuario con rol “admin” o “maintainer” a un usuario con rol “basic”, aparece la columna “actions” en la vista “List”, esto se puede deber a cómo se mantiene el DOM virtual en memoria cache.

3.5. Base de datos

Como MongoDB es una base de datos no relacional y orientada a documentos, para explicar su diseño debemos explicar las colecciones que lo componen y las estructuras de los documentos que maneja cada colección.

3.5.1. Colecciones

La base de datos cuenta con cinco colecciones, las cuales se describen a continuación:

- Configurations: Almacenará la configuración relativa a la aplicación.
- Reservations: Almacenará las reservas.
- Printers: Almacenará las impresoras.
- Updates: Almacenará las operaciones realizadas contra las impresoras.
- Users: Almacenará los usuarios del sistema.

3.5.2. Esquemas

A continuación, se encuentran los esquemas proporcionados por Mongoose, para facilitar la tarea de validar los datos y realizar operaciones en las colecciones de la base de datos.

Para facilitar la tarea de validación de datos y gestión de los documentos dentro de las colecciones, se ha utilizado Mongoose, que nos permite definir los esquemas de los documentos y aplicar restricciones y validaciones. Estos son los esquemas utilizados que se encuentran en la carpeta /models del servidor:

- ConfigurationSchema:

```
{
  server: {
    discovery: {
      updateFrecuency: { type: Number },
      printerExpires: { type: Number },
      maxPrinterLogs: { type: Number },
    },
    ldap: {
      url: { type: String },
      adminUsername: { type: String },
      adminPassword: { type: String },
    },
  },
  client: {
    colors: [{ type: String }],
    status: [{ text: { type: String }, color: { type: String } }],
    defaultHeaders: [{ type: String }],
    alertTimeout: { type: Number },
    defaultHours: { type: Number },
    calendarStartHour: { type: Number },
    calendarEndHour: { type: Number },
  },
}
```

- ReservationSchema:

```
{
  resourceid: { type: mongoose.Schema.Types.ObjectId, required: true },
  start: { type: Date, required: true },
  end: { type: Date, required: true },
  reservedby: { type: String, required: true },
  name: { type: String, required: true },
  active: { type: Boolean, default: true },
  color: { type: String },
}
```

Como se puede observar, el atributo resourceid es del tipo ObjectId, que representa otro documento del sistema, en este caso una impresora.

- PrinterSchema:

```
{
  hostname: {
    type: String,
    trim: true,
    unique: true,
    required: true,
    uppercase: true,
  },
  ip: { type: String, trim: true, required: true },
  modelname: { type: String, trim: true, required: true },
  updated: { type: DateRes },
  created: { type: Date },
  fromdiscovery: { type: Boolean, default: false },
  color: { type: String },
  information: {
    status: { type: String },
    firmwareversion: { type: String },
    partnumber: { type: String },
    printertype: { type: String },
    alerts: [{ type: String }],
  },
  metadata: {
    alias: { type: String, default: '' },
    location: { type: String, default: '' },
    workteam: { type: String, default: '' },
    reservedby: { type: String, default: '' },
    reserveduntil: { type: String, default: '' },
    notes: { type: String, default: '' },
  },
  log: [{
  status: { type: String },
  path: { type: String },
  filename: { type: String },
  timestamp: { type: Date },
  }],
}
```

- UpdateSchema

```
{
  printer: { type: mongoose.Schema.Types.ObjectId },
  user: { type: String },
  description: { type: String },
  timestamp: { type: Date },
}
```

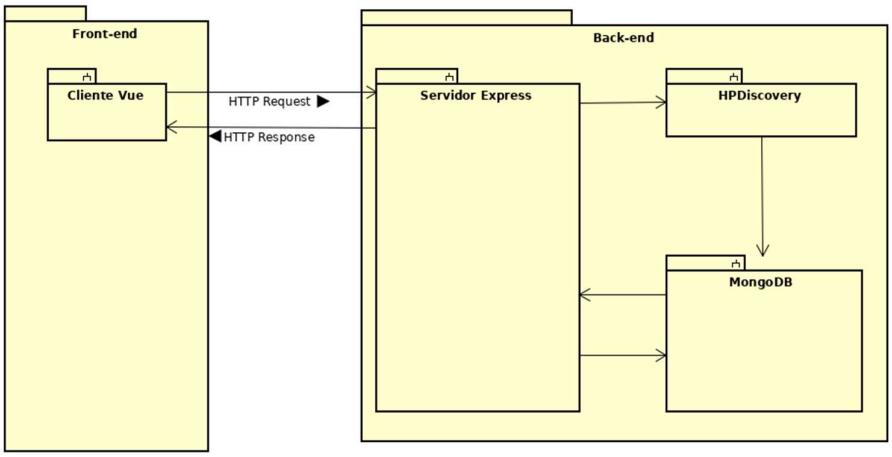
- UserSchema:

```
{
  name: { type: String },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  lastlogin: { type: Date, default: Date.now },
  active: { type: Boolean, default: true },
  role: { type: String, required: true,
        enum: ['basic', 'maintainer', 'admin'] },
  accesstoken: { type: String },
}
```

En el esquema de usuario encontramos definidos los tipos de rol que se permiten en la aplicación “basic”, “maintainer” y “admin”.

3.6. Estructura de la aplicación

En esta SPA podemos diferenciar dos partes claras, la parte del cliente o front-end y la parte del servidor o back-end, a continuación, se analiza más en profundidad la composición y funcionamiento de cada una de las partes.



Para facilitar la tarea de detallar el diseño de la aplicación, se ha decidido tratar estas dos partes de manera independiente, y luego explicar de qué manera se relacionan

3.7. Front-end

Una vez se ha elegido la tecnología a utilizar en el front-end, el siguiente paso fue diseñar tanto visual como funcionalmente como iba a ser esta parte. A

continuación, se explica el diseño de los componentes desarrollados, su funcionalidad y su interacción con otras partes de la aplicación.

Antes de empezar, es necesario recordar que la gestión de estado de la aplicación se ha realizado con Vuex, lo que nos ha facilitado mucho la tarea de mantener el estado de la vista sincronizado con el estado del servidor. También hemos hablado de Vuetify, que nos ha proporcionado componentes altamente personalizables siguiendo las líneas de diseño de Material Design y proporcionando una manera muy sencilla de implementar una interfaz responsiva.

Otro punto que hay que tener en cuenta es el uso que se ha hecho de Vue-router, que ha permitido implementar un sistema de rutas en una SPA. A diferencia de una ruta HTTP clásica, la única función realizada por estas rutas es la de relacionar una vista con una determinada ruta.

3.7.1. Vistas y componentes

Para empezar a describir el diseño de esta aplicación necesitamos hacer una distinción entre vista y componente, las vistas están compuestas de uno o más componentes y a ellas se accede a través de una ruta en el navegador.

Los componentes están formados de uno o más componentes y son las piezas que se renderizan en las vistas, estos componentes están formados por tres partes, la estructura HTML, que se define en la etiqueta <template/>, la funcionalidad que se define en la etiqueta <scritpt/> y los estilos que se definen en la etiqueta <style/> aunque normalmente se utilizan los proporcionados por Vuetify.

3.7.2. Componente Login

Este componente será el encargado de recoger los datos de inicio de sesión del usuario, validarlos y realizar una petición al servidor para comprobarlos.

- Estructura: Formulario con dos campos y un botón.
- Funcionalidad: Recoge el email y contraseña introducido por el usuario y realiza una petición POST con estos datos hacia la ruta “/login” de la API del servidor. Solo es visible si no se ha iniciado sesión.

3.7.3. Componente Alert

Este componente será el encargado de mostrar los mensajes al usuario en la interfaz, aparecerá un banner de distinto color según el mensaje y el tipo.

- Estructura: Banner en la parte superior en el que se mostrará un mensaje con un color elegido.
- Funcionalidad: Recibe un mensaje y un tipo, y lo muestra en la parte superior de la vista durante un tiempo determinado.

3.7.4. Componente CreateReservation

Este componente será el encargado de mostrar un botón con el mensaje “New Reservation”, que una vez pulsado lanzará un pop-up con un formulario,

donde se introducirá el usuario y el rango en el que se desea reservar la impresora.

- Estructura: Botón, Pop-Up, Formulario
- Funcionalidad: Recogerá el usuario y el rango en el que se desea crear la reserva para una impresora, realizará una petición POST hacia la ruta "/new-reservation"

3.7.5. Componente Navigation

Este componente mostrará una barra lateral donde se encontrará la información relativa al usuario, como nombre y email, también mostrará en función del usuario las rutas accesibles a este. Por último y en la parte inferior encontramos el botón de Logout para cerrar sesión.
Normalmente minimizado para no ocupar espacio en la interfaz, se despliega al pasar el ratón sobre él.

- Estructura: Barra lateral con título y subtitulo, lista con el nombre de las rutas accesibles y botón de Logout.
- Funcionalidad: Mostrará los datos del usuario que está actualmente identificado en la aplicación, también mostrará la lista de rutas accesibles y navegará hacia la misma al clicar en ella. Al hacer click en el botón Logout, pedirá confirmación al usuario y en función de la respuesta cerrará su sesión o no.

3.7.6. Componente PrinterComponent

Este componente es el encargado de mostrar un icono de edición o un botón con el mensaje "New Printer" en función de si se va a crear una nueva impresora o se desea editar una ya existente, está compuesto por otros dos componentes, PrinterDetails y PrinterLog. Al hacer click en el icono o el botón, lanzará un pop-up con estos dos componentes.

- Estructura: Muestra un icono o un botón en función de si recibe un objeto printer existente o no, Pop-up, PrinterDetails y PrinterLog.
- Funcionalidad: En función de si el objeto que recibe es una impresora o no, muestra un icono de edición o un botón con "New printer", al pulsarlo aparece un pop-up con los componentes mencionados anteriormente.

3.7.7. Componente PrinterDetails

Es el encargado de mostrar los datos de una impresora o recoger los datos de una nueva impresora, también mostrará los botones para realizar acciones en función del rol del usuario actual. Detallaremos este componente en sus dos variantes:

- Nueva impresora:
  - Estructura: Botón "New printer", formulario con los campos hostname, ip, modelname, alias, location, workteam y notes. Botón "Close" y "Create Printer".
  - Funcionalidad: Al clicar el botón se despliega un pop-up, mostrando un formulario, recogiendo los campos mencionados anteriormente y validándolos. Al hacer click en "Create Printer" realizará una petición

POST hacia la ruta “/new-printer” de la API del servidor. Al hacer clic en “Close” limpiará el formulario y cerrará el pop-up.

- Editar impresora:
  - Estructura: Icono de edición, formulario con los campos hostname, ip, modelname, status, part-number, printer-type, firmware-version, alias, location, workteam, notes, alerts, updated, created, log y botón “Download log”. Botones “Close”, “Get new log”, “Get Info”, “Update Printer”
  - Funcionalidad: Al clicar en el icono se despliega un pop-up, mostrando el formulario con los datos de la impresora existente. Al hacer click en el botón “Get new log” se realiza una petición GET hacía la ruta “/printers/:id/log” para solicitar el log de la impresora. Al hacer click en “Get info” se realiza una petición GET hacía la ruta “/printers/:id/update” para solicitar una actualización de la información de la impresora. “Update Printer” realizará una petición PUT hacía la ruta “/printers/:id” para actualizar los datos de la impresora. En caso de que exista un log disponible y que se haya seleccionado

3.7.8. Componente PrinterLog

Se encarga de mostrar el historial de cambios de una impresora y quien ha realizado estos cambios. Solo está visible en el caso de que exista la impresora.

- Estructura: Lista con título.
- Funcionalidad: Recoge las operaciones realizadas sobre la impresora actual y las muestra en una lista.
- Estilo: Se ha ocultado la barra de desplazamiento lateral de la lista.

3.7.9. Componente ReservationComponent

Se encarga de mostrar la información relativa a una reserva, también muestra los botones de acción en función del rol del usuario actual.

- Estructura: Formulario con los campos hostname, ip, modelname pertenecientes a la impresora a reservar, reserved by, start y end. Botones “Update Reservation”, “Delete reservation”
- Funcionalidad: Recoge los datos de una reserva actual y los muestra en el formulario, en función del rol del usuario permite modificarlo o no. El botón “Delete Reservation solo se muestra a usuarios con el rol admin y realiza una petición DELETE hacia la ruta del servidor ”/reservation/:id”. El botón “Update Printer”, realiza una petición PUT hacía la ruta “/reservation/:id” para actualizar los datos de la reserva.

3.7.10. Componente UserComponent

En función de si recibe un objeto usuario o no, muestra el botón “New User” o un icono de edición, contiene un formulario en el que muestra o recoge la información relativa a un usuario. Detallaremos este componente en sus dos variantes:

- Nuevo usuario:

- Estructura: Muestra el botón “New User”. Formulario con los campos email password, name y rol. Botones “Close” y “Create User”.
- Funcionalidad: Al hacer click en “New user” muestra un pop-up con el formulario del usuario. El botón “Close” limpia los campos del formulario y cierra el pop-up y el botón “Create User” realiza una petición POST hacia la ruta “/new-user” de la API del servidor.

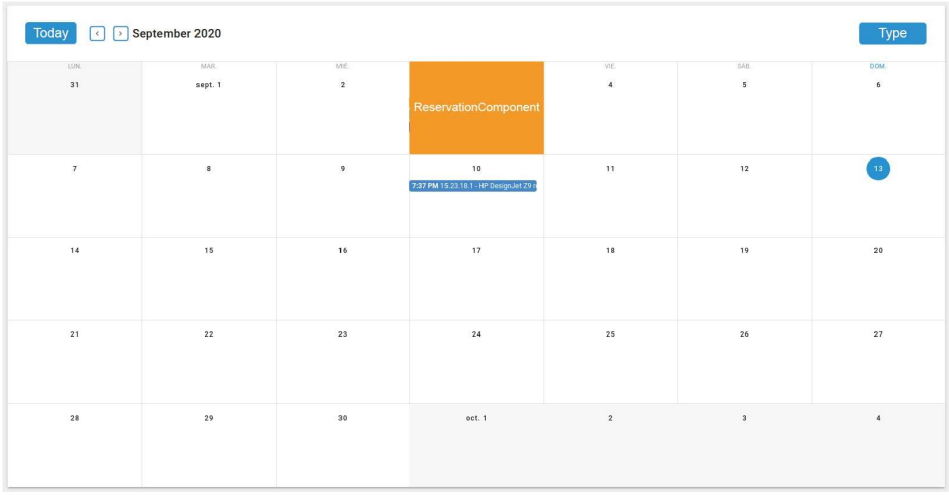
- Usuario existente:
  - Estructura: Muestra un icono de edición. Formulario con los campos email password, name y rol. Botones “Close” y “Update User”.
  - Funcionalidad: El botón “Update User” realiza una petición PUT hacía la ruta “/users/:id” para actualizar los datos del usuario.

3.7.11. Aplicación

Es el componente raíz de la aplicación, se encargará de mostrar el componente de login en caso de que el usuario no esté identificado y de mostrar la barra de navegación y la vista actual en caso de que sí lo esté. También contiene el componente de las alertas, haciendo posible mostrar mensajes al usuario independientemente de la vista en la que nos encontremos.

3.7.12. Vista Calendar

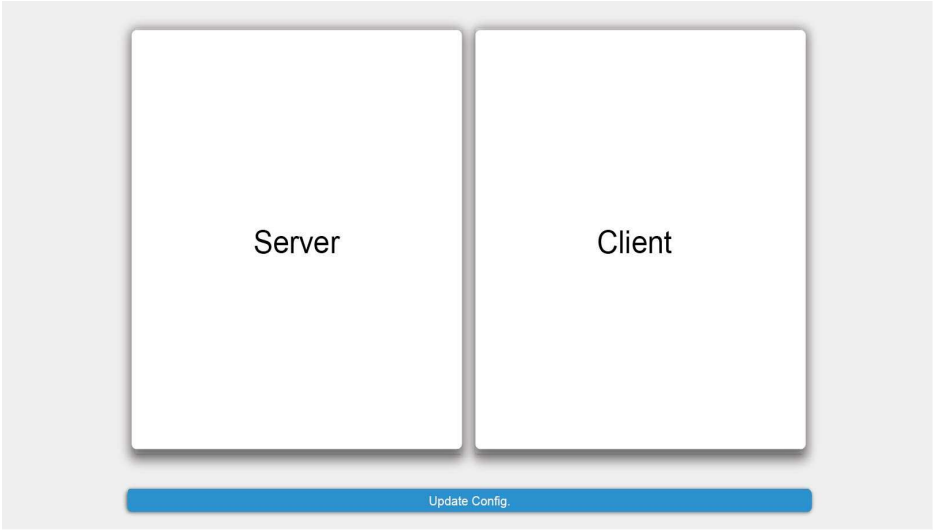
Es la vista encargada de mostrar un calendario en el que aparezcan las reservas en el rango de fechas fijado, también permitirá mostrar el calendario según un formato elegido. Si existen impresoras seleccionadas mostrará las reservas de las mismas, si no mostrará todas las reservas existentes en el sistema.



- Ruta: /calendar
- Funcionamiento: Al hacer click en cualquier reserva se abrirá el componente ReservationComponent. El botón “Today” fija la fecha en el día actual. Las flechas de navegación permiten avanzar la fecha o retroceder. El selector de tipo define como se muestra el calendario, las opciones disponibles son meses, día, 4 días, semana.

3.7.13. Vista Configuration

Muestra dos formularios y un botón, en uno de los formularios se muestra la configuración del servidor y en el otro la del cliente.

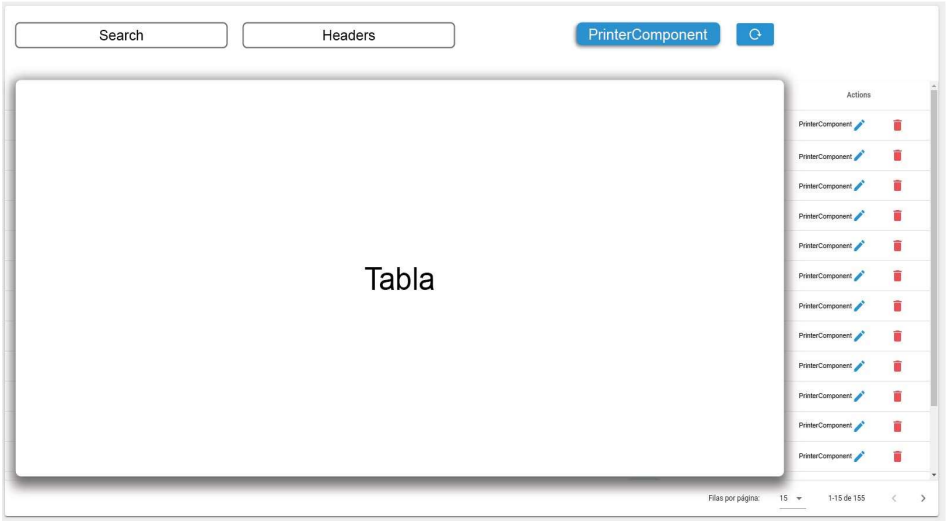


- Ruta: /config
- Funcionamiento: Los formularios muestran la configuración actual del cliente y del servidor, cuando se pulsa el botón “Update Config” se realiza una petición PUT hacía la ruta “/” de la API del servidor para actualizar la configuración existente.

3.7.14. Vista PrinterList

Muestra una tabla con las impresoras existentes, también proporciona un campo de búsqueda para filtrar entre las impresoras existentes, un campo para elegir las columnas que se muestran en la tabla, el componente

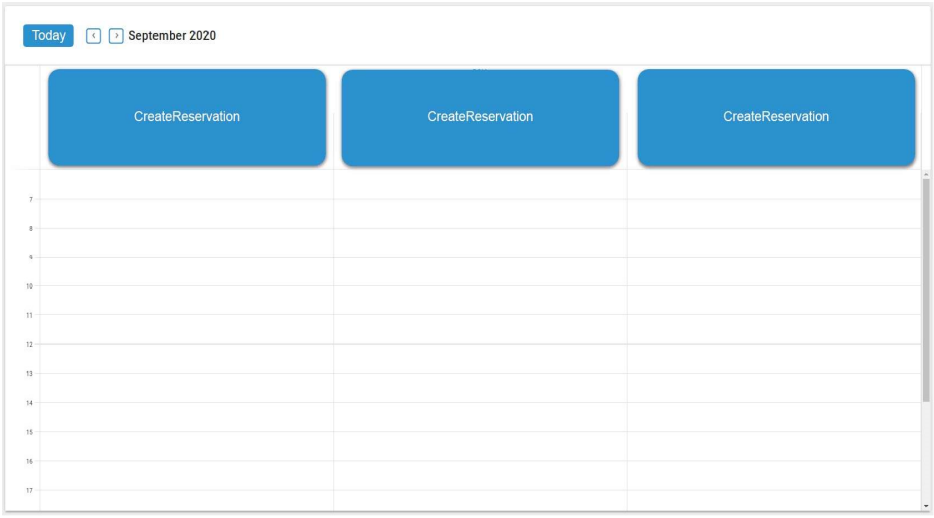
PrinterComponent, un botón para actualizar la lista de impresoras y otro para deseleccionarlas en caso de que lo estén.



- Ruta: /list
- Funcionamiento: Se muestra una tabla que muestra las impresoras existentes y permite seleccionarlas, con las columnas elegidas en el campo headers, un campo para realizar una búsqueda en la lista, el botón “New printer” y un botón para deseleccionar las impresoras en caso de que lo estén.

3.7.15. Vista ReservationList

Muestra un calendario diario por cada impresora seleccionada, el objetivo de esta vista es poder comparar las reservas de varias impresoras, previamente seleccionadas y reservar la que más se adecue a las necesidades del usuario, también es posible cambiar la fecha, volver a la fecha actual y modificar las reservas existentes.



- Ruta: /reservations
- Funcionamiento: Se genera un calendario para cada una de las impresoras seleccionadas, con el componente CreateReservation en la cabecera de cada uno. En cada calendario se muestran las reservas existentes para ese

día, haciendo click encima de una reserva se muestra el componente ReservationComponent.

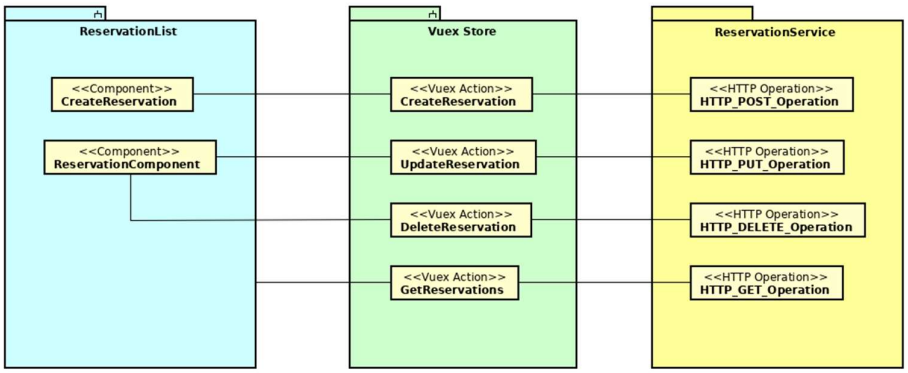
3.7.16. Vista UserList

Esta vista muestra una tabla con los datos de cada usuario, también proporciona un cuadro de búsqueda, un botón para actualizar los datos de la tabla y un botón para añadir un nuevo usuario a la aplicación.

- Ruta: /users
- Funcionamiento: En la tabla se muestran los datos de cada usuario, adicionalmente en cada fila de la tabla, se muestra el componente UserComponent en modo edición y un icono que permite eliminar el usuario de la aplicación. El botón “New User” abrirá el componente UserComponent en modo creación.

3.7.17. Gestión de estado y conexión con la API

Como hemos comentado anteriormente, se ha utilizado Vuex para la gestión del estado de la aplicación, esto además de las ventajas comentadas anteriormente, nos proporciona una forma de conectar cada componente únicamente con las peticiones necesarias para su funcionamiento. Esto se traduce en una mayor seguridad, modularidad y mantenibilidad. En la figura siguiente se detalla estas peticiones a través de un componente.



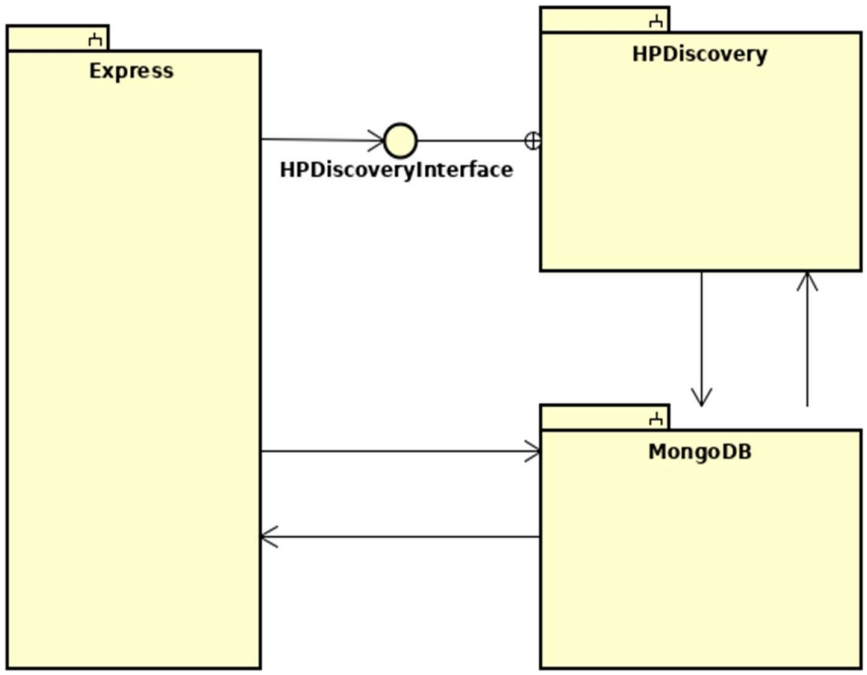
Como se puede observar la vista ReservationList ejecuta una Vuex action, encargada de utilizar ReservationService que realiza la petición HTTP, en este caso GET, para recuperar la lista de reservas del servidor. A su vez esta vista está formada por dos componentes, CreateReservation, que ejecuta una VuexAction que realiza una petición POST a través del service para crear una nueva reserva y ReservationComponent, que puede ejecutar dos actions, una para actualizar la reserva y otra para borrarla del sistema.

La vista no puede acceder a las action de los componentes que lo forman, de igual manera los componentes no pueden acceder a las action de la vista o de sus componentes padre.

De esta manera se delimita la funcionalidad a la que puede acceder cada componente, el funcionamiento es similar para el resto de componentes del cliente.

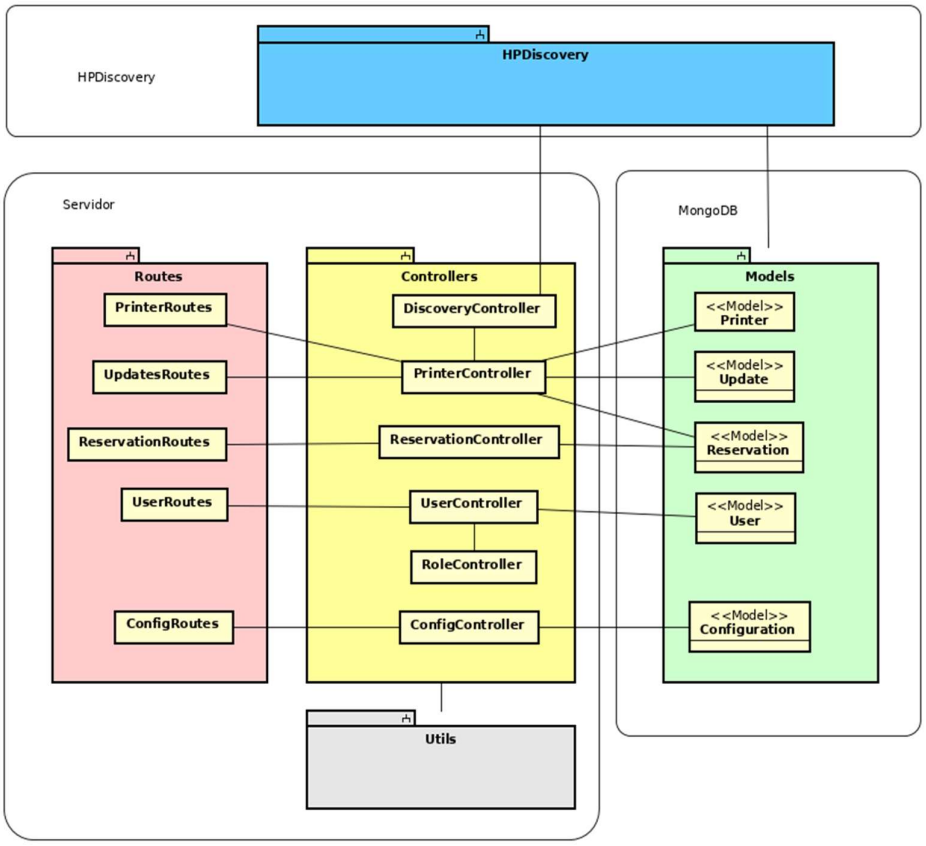
3.8. Back-end

Durante este capítulo describiremos el diseño de la parte back-end, como ya hemos comentado anteriormente, el único requisito tecnológico impuesto por HP, fue que el sistema de base de datos tenía que ser MongoDB. También explicaremos las partes que componen el servidor y la biblioteca HPDiscovery.



3.8.1. Servidor Express

El servidor será el encargado de exponer la funcionalidad desarrollada mediante una API REST, para relacionar una funcionalidad con una dirección web, utilizaremos el sistema de rutas proporcionado por express.



Como se puede observar, cada colección tiene definida una ruta y una funcionalidad asociada, que harán uso de los esquemas de la base de datos cuando sea necesario realizar operaciones contra la misma. Estas funcionalidades se apoyan en otras funcionalidades a las que se ha llamado utils, como un logger, una función de validación de tokens, un manejador de errores o un helper para la conexión LDAP entre otras.

3.8.2. HPDiscovery

En este apartado se resumirá el funcionamiento de la biblioteca, ya que no ha sido una parte desarrollada por el alumno durante este trabajo, sino que ha sido proporcionada por los ingenieros de HP.

- Funcionamiento: Esta biblioteca realiza un escaneo de la red local y recupera los dispositivos conectados a la misma, también recupera información de cada uno de estos dispositivos. Durante el desarrollo se ha trabajado, por motivos de confidencialidad, con una interfaz igual a la de la biblioteca en cuestión.

- Interfaz: Para integrar esta biblioteca escrita en C++ con nuestra aplicación JavaScript, ha sido necesario implementar un adaptador, que define la funcionalidad de la misma y los tipos de datos recibidos y devueltos, esta adaptación se realiza a través de una interfaz con las siguientes operaciones.

    - HPDiscoveryInit: Inicializa la biblioteca para poder ser utilizada, no recibe ningún dato y devuelve un int cuyo valor determinará el éxito o fracaso de la operación.

- HPDiscoveryTerminate: Elimina los recursos y la memoria asignados, es necesario invocar esta funcionalidad antes de terminar la ejecución de la aplicación. No recibe ningún parámetro y devuelve un int cuyo valor determinará el éxito o fracaso de la operación.

- HPDiscoverySuscribe: Permite relacionar la funcionalidad a ejecutar cuando se encuentre un nuevo recurso en la red, solo es posible realizar una suscripción a la vez. Recibe un puntero a una función, que será la que se ejecute cuando se dispare el evento. Esta función parsea el XML devuelto por la biblioteca y lo guarda en la base de datos.

- HPInformationSuscribe: Permite relacionar la funcionalidad a ejecutar cuando se encuentra información sobre un recurso de la red, al igual que la anterior solo permite un suscriptor a la vez. Recibe un puntero a una función que será la encargada de actualizar la información de un recurso en la base de datos.

- HPDiscoveryGetPrinterInformation: Recibe la dirección ip, el hostname y el modelname de una impresora y dispara el evento para actualizar la información de la misma. Devuelve un int cuyo valor determina el éxito o fracaso de la operación.

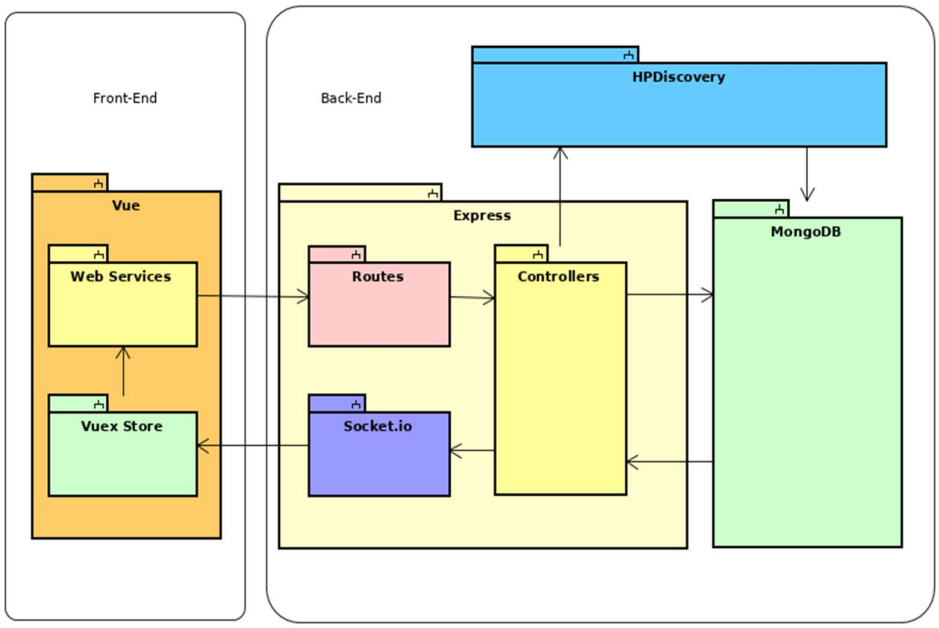
- HPPrinterLogFileSuscribe: Permite relacionar la información a ejecutar cuando se pide el log de los recursos, recibe un puntero a una funciona que será la encargada de añadir el nuevo log junto a sus detalles.

- HPDiscoveryGetPrinterLogFile: Recibe la dirección ip, el hostname y el modelname de una impresora y dispara el evento para actualizar el log de la misma. Devuelve un int cuyo valor determina el éxito o fracaso de la operación.

3.9. Comunicaciones

En este paso vamos a definir las comunicaciones existentes entre el cliente y el servidor, también detallaremos como es el funcionamiento de Soket.io que nos ha permitido mandar eventos desde el servidor hacia el cliente.

Las peticiones cliente - servidor se manejan como se ha comentado anteriormente, desde algún componente de la vista se ejecuta una Vuex action que realiza la petición web a través de un service. Esta petición web está formada de una operación HTTP válida, la ruta a la que va dirigida y el cuerpo y cabeceras de la petición. En las rutas del servidor se encuentra la relación entre la operación y la ruta y la funcionalidad a ejecutar.



Antes se ha explicado cómo se ha relacionado la funcionalidad del servidor con las operaciones HTTP a través de las rutas de express, en la imagen anterior podemos observar que también existe comunicación servidor - cliente. Esto ha sido especialmente útil para mantener la sincronía de los datos de la aplicación en todo momento. Este mecanismo solo es utilizado por el controlador del Discovery y su función es la de emitir un evento hacia el cliente avisando de que los datos han sido actualizados en la base de datos. El cliente escucha el evento en la Vuex Store y realiza una petición a través del servicio para actualizar los datos de la vista. En el siguiente capítulo detallaremos en profundidad este mecanismo.

4. Implementación

En este apartado se detallarán los mecanismos más relevantes que se han implementado, la forma en la que se ha llevado a cabo y las consideraciones a tener sobre los mismos.

4.1. Inicialización del servidor

La primera vez que se inicia la aplicación después de ser desplegada el servidor utilizará una configuración por defecto que se encuentra bajo la ruta "/server/config/config.js", en este archivo se encuentra definidos los siguientes parámetros:

- Las opciones con las que se inicia la base de datos.
- El usuario admin por defecto.
- Los parámetros de funcionamiento de la biblioteca HPDiscovery.
- Los parámetros de la conexión LDAP.
- Los colores de las reservas en el front-end.
- Los colores de los estados de las impresoras
- Los tipos de impresoras disponibles
- El tiempo que permanecen visibles las alertas al usuario.
- El número de horas que se añaden por defecto a la fecha fin al crear la reserva.
- La hora de comienzo del calendario.
- La hora de fin del calendario.

Una vez detallado el contenido del archivo de configuración, continuamos explicando cómo se realiza el primer inicio de la aplicación.

El fichero principal del servidor "index.js" se apoya en la funcionalidad "dbConnection" ubicada en la ruta "/server/utils/dbConnection.js" la cual se encarga de inicializar la conexión con la base de datos y comprobar si existe un documento con la configuración de la aplicación. Si no existe creará el documento con la configuración detallada anteriormente.

El siguiente paso es añadir a express los middlewares que utilizaremos, definir las rutas e iniciar el servidor. Una vez el servidor está instanciado, se pasa como argumento para crear un socket de tipo servidor que utilizaremos más adelante.

Lo último que haremos será inicializar la biblioteca HPDiscovery y realizar las suscripciones necesarias para su funcionamiento.

4.2. Discovery Adapter

Para poder utilizar la biblioteca proporcionada por HP, ha sido necesaria la implementación de un adaptador que permitiese el uso de la biblioteca, escrita en C++, en nuestro entorno de ejecución Node, que utiliza JavaScript.

Este adaptador se encuentra bajo la ruta "/server/HPDiscoveryV3.1/discoveryAdapter.js", en él se ha definido la interfaz con las operaciones que utilizaremos, detalladas en el apartado anterior.

La mayoría de operaciones de la interfaz se ejecutan de manera asíncrona, para respetar la asincronía de la biblioteca, se ha implementado un mecanismo de promesas, el cual devuelve un valor con estado pendiente hasta que termine la ejecución de la operación de la biblioteca, en ese momento el valor devuelto cambiará a cumplido o rechazado en función de si se ha realizado una ejecución exitosa o no.

Estas operaciones asíncronas reciben como parámetros un puntero a una función, la cual contiene la funcionalidad a ejecutar cuando la biblioteca ha recuperado los recursos de la red. Estas funcionalidades están definidas en el archivo "/servidor/controllers/discoveryController.js" en las funciones "handleSuscription", "handleInformation" y "handleLog", estás funciones se instancian cada una en una constante haciendo uso de ffi-napi para indicar que es un callback y de ref-napi para determinar los tipos devueltos y recibidos por la función.

4.3. Implementación de roles

Para implementar el sistema de permisos basados en roles se ha utilizado la biblioteca role-acl, la cual permite definir un objeto que relaciona los recursos, los roles y las operaciones, así como definir excepciones.

El objeto que contiene estas relaciones se encuentra en la ruta "/server/controllers/roleController.js", en él se detalla la excepción existente para los usuarios con rol basic, que pueden editar o eliminar sólo las reservas que ellos mismos han creado.

Cuando se va a realizar una operación, se consulta si el rol del usuario permite realizar la operación deseada contra ese tipo de recursos. Se puede encontrar un ejemplo en los ficheros de las rutas de express, que relacionan una funcionalidad con una dirección web.
Aunque la comprobación está implementada como un middleware en el archivo "/server/controller/userController.js", en las funciones "grantAcces" y "grantAccesOrOwn".

4.4. Auto actualización y gestión de recursos

Para implementar el mecanismo de auto actualización se ha utilizado un intervalo, al cual se le pasa una funcionalidad y un parámetro indicando cada cuantos milisegundos se volverá a ejecutar esta funcionalidad.

La implementación de este mecanismo se puede observar en el archivo "/server/controllers/discoveryController.js", en la función start, que será la encargada de inicializar la biblioteca y realizar las suscripciones necesarias, cuanto estas tareas terminan define el intervalo.

Es posible modificar este intervalo en tiempo de ejecución, mediante la función setUpdateInterval, que se ejecuta cuando se realiza una actualización del objeto de la configuración. Esta función comprueba si existe un intervalo ya definido, si existe lo elimina y crea uno nuevo con el nuevo tiempo de intervalo.

La funcionalidad ejecutada por el intervalo se encuentra en la función “autoUpdateAllPrinters”, que iniciará un contador para medir el tiempo que necesita cada ejecución del intervalo para poder controlar de alguna manera el rendimiento de la aplicación. Después desactivamos el flag que controla el envío de eventos desde el servidor al cliente cada vez que una impresora es actualizada para no sobrecargar la red, ya que se emitirá un evento por cada impresora y por cada usuario conectado al sistema.

El funcionamiento es el siguiente:

- Se recupera la lista de todas las impresoras.
- Se eliminan las impresoras inactivas.
- Se ejecuta la operación del adaptador solicitando la información de cada una de las impresoras.
- Una vez terminada la operación anterior se para el contador.
- Se emite un evento avisando de que todas las impresoras se han actualizado.

4.5. Eventos desde el servidor

Para implementar el envío de eventos desde el servidor nos hemos apoyado en Socket.io que nos permite definir un Socket de tipo servidor y conectarnos a este socket desde cada uno de los clientes, de manera que cada evento emitido puede ser escuchado en el cliente.

Todos los eventos se emiten desde el mismo archivo “/server/controller/discoveryController.js”, esto se hace para solucionar el problema existente de sincronía en el estado de la aplicación. El problema es que no existía una manera de avisar al cliente de que el estado de la aplicación había cambiado, por lo que se producían situaciones en el que los datos mostrados en la vista no eran los mismos que los existentes en la base de datos.

Este problema se ha solucionado con este mecanismo de eventos, a continuación, se detallan los eventos que se emiten y las operaciones realizadas por el cliente al recibirlos:

- AllPrintersUpdated: Se emite cuando se ha terminado la tarea de actualización de las impresoras, al recibir el evento, el cliente realiza una petición al servidor para actualizar esta lista.

- OnePrinterCreated: Se emite cuando se ha creado una impresora, además este evento contiene la impresora creada en cuestión, el cliente al recibir el evento añade la nueva impresora.

- OnePrinterUpdated: Se emite cuando se ha actualizado la información de una impresora, este evento contiene la impresora actualizada, el cliente al recibir el evento busca la impresora y la actualiza.

- OnePrinterLogReady: Se emite cuando el log de una impresora está listo, este evento contiene el log de la impresora, el cliente al recibir el evento actualiza el log de la impresora.

4.6. Solicitud y descarga del log de una impresora

Cuando se solicita un log de una impresora, se ejecuta la operación HPDiscoveryGetPrinterLogFile de la biblioteca HPDiscovery, una vez finalizada esta tarea, el servidor emite el evento OnePrinterLogReady, para notificar y proporcionar al cliente el nuevo log.

Cuando una impresora tiene logs disponibles el componente PrinterDetails, mostrará un desplegable para seleccionar que log es el que deseamos descargar, al pulsar el botón Download Log se realiza una petición indicando el nombre y la ruta del log a descargar, si el fichero está disponible en la ruta proporcionada, se crea la response adjuntando el archivo.

Una vez recibida esta response en el cliente, se ejecuta la función downloadResponse, que se encuentra en la ruta /client/helpers/responseHandler.js y permite descargar el archivo desde el navegador hasta nuestro equipo.

4.7. Jsonwebtoken

Para identificar a los usuarios en la aplicación se ha implementado un sistema de tokens, esto nos permite construir una cadena de caracteres única y asignarla a un usuario, esta cadena se adjunta en la cabecera de cada petición y con ella el servidor es capaz de identificar al usuario y conocer su rol.

En el archivo /server/controllers/userController.js encontramos la función login, que se encarga de recoger los datos introducidos por el usuario y comprobar si este usuario existe en el servidor LDAP o en la base de datos de la aplicación, si el usuario existe se genera un nuevo token que será válido durante las siguientes veinticuatro horas.

4.8. Logger

Se ha implementado un logger, el cual se encuentra en la ruta /server/utils/logger.js, se ha personalizado un logger Winston de la siguiente manera:

- Formato: [timestamp][label] level: mensaje.
- Log de errores: /logs/filelog-error.log.
- Todos los logs: /logs/filelog-combined.log
- Consola: si la variable NODE_ENV no es producción, los logs aparecen por consola.

4.9. Registro de cambios de impresora

El funcionamiento del registro de cambios de las impresoras es el siguiente, cuando se va a realizar una operación contra una impresora, se guarda el estado antes y el estado después, estos dos estados se comparan obteniendo las diferencias entre ellos, estas diferencias se guardan en la base de datos, estos documentos son los mostrados por el componente PrinterLog.

Esta funcionalidad se encuentra en el archivo /server/utils/printerDiff.js

5. Conclusiones

En este apartado se pretende realizar un resumen del proyecto exponiendo los resultados, los cambios en comparación con la aplicación anterior, así como los objetivos cumplidos, los puntos de mejora y las líneas de trabajo futuras.

Lo primero que debemos exponer es que se han cumplido todos los requisitos proporcionados por HP, también se han solucionado todos los issues existentes en la versión anterior, por lo que aun habiendo dejado fuera la funcionalidad detallada en la wishlist, se puede decir que el proyecto concluye de manera exitosa.

Las mejoras en comparación con la versión anterior son varias, empezando por la interfaz gráfica, un punto verdaderamente importante y que a veces cae en el olvido. Este proyecto ha seguido la guía de estilos Material Design gracias a Vuetify, que ha facilitado la tarea de implementar una interfaz limpia e intuitiva, usable y amigable para los usuarios.

Otro punto importante a destacar es el del rendimiento, ya que al implementar de manera asíncrona el adaptador de la biblioteca HPDiscovery, se han solucionado los problemas de fiabilidad y rendimiento existentes en la versión anterior. También ha mejorado el tiempo de respuesta y la reactividad de la interfaz de usuario gracias al uso de Vue.

La mantenibilidad de la aplicación también se ha mejorado, se ha separado la funcionalidad en archivos más pequeños y se han distribuido en directorios según su propósito. También se han definido los esquemas de los datos a guardar en la base de datos, haciendo que sea más sencillo localizarlos para consultarlos o modificarlos.

Los puntos de mejora futuros, ordenados por su prioridad serán, solventar los issues existentes en la versión actual, reducir el tamaño de los componentes, dividiéndolos en componentes más pequeños para aumentar la reusabilidad y limitar las acciones de los mismos. También sería conveniente revisar la reactividad de los componentes que conforman las tablas, ya que, en algunos casos, los cambios no son renderizados inmediatamente en la vista.

Si en un futuro se prevé aumentar el tamaño del proyecto, añadiendo nuevas funcionalidades, recursos, vistas u otros añadidos, no sería descabellado realizar una nueva implementación del servidor y del cliente utilizando TypeScript, que recientemente ha lanzado su versión 4. Para acompañar este cambio también sería recomendable actualizar Vue, el framework utilizado para implementar el cliente a su versión 3, que soporta de manera nativa el uso de TypeScript.

Por todo lo anterior podemos afirmar que se ha producido una mejora notable en comparación con la versión anterior, que ha sido posible gracias al uso de las tecnologías mencionadas durante el trabajo.

Bibliografía y referencias

[1] The MongoDB 3.6 Manual.
https://docs.mongodb.com/v3.6/

[2] Mongoose API Docs.
https://mongoosejs.com/docs/api.html#

[3] Express API 4.x.
https://expressjs.com/es/api.html

[4] Mejores prácticas de producción: seguridad.
https://expressjs.com/es/advanced/best-practice-security.html

[5] Mejores prácticas de producción: rendimiento y seguridad.
https://expressjs.com/es/advanced/best-practice-performance.html

[6] Node.js v12.18.3 Documentation.
https://nodejs.org/dist/latest-v12.x/docs/api/

[7] Vue.js guide 2.x
https://vuejs.org/v2/guide/

[8] Vuetify getting started
https://vuetifyjs.com/en/getting-started/quick-start/

[9] Pinheiro Malère, L. E. (18/3/2018). LDAP Linux HOWTO. Obtenido de:
https://tldp.org/HOWTO/pdf/LDAP-HOWTO.pdf

[10] The OpenLDAP project. (19/2/2008). OpenLDAP Software 2.3 Administrator's Guide. Obtenido de: https://www.openldap.org/doc/admin23/

[11] Bemenderfer, J. (28/3/2017). Integrating Vue.js and Socket.io. Obtenido de:
https://www.digitalocean.com/community/tutorials/vuejs-vue-socketio

[12] Asfo. (19/8/2019). Autenticando un API Rest con NodeJS y JWT (JSON Web Tokens). Obtenido de: https://medium.com/@asfo/autenticando-un-api-rest-con-nodejs-y-jwt-json-web-tokens-5f3674aba50e

[13] API REST con Autenticación JWT con JSON Server y jsonwebtoken.
https://desarrolloweb.com/articulos/api-rest--autenticacion-jwt.html

[14] MDN Web docs. Fetch API.
https://developer.mozilla.org/es/docs/Web/API/Fetch_API

[15] Centralized alert for Vuetify
https://techformist.com/centralize-alert-vuetify/

[16] Obielum, G. (5/9/2019) Implementing Role-Based Access Control in a Node.js application. https://blog.soshace.com/implementing-role-based-access-control-in-a-node-js-application/

[17] AccesControl API.
https://onury.io/accesscontrol/?api=ac

[18] Bcrypt.
https://github.com/kelektiv/node.bcrypt.js#readme

[19] Vue.js, Node.js, Express & MongoDB [MEVN].
https://bluuweb.github.io/mevn/

[20] Connect-history-api-fallback.
https://github.com/bripkens/connect-history-api-fallback#readme

[21] MDN Web docs. Control de acceso HTTP (CORS).
https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

[22] Cors for Express.
https://github.com/expressjs/cors#readme

[23] Node.js Foreign Function Interface for N-AP.
https://github.com/node-ffi-napi/node-ffi-napi

[24] Helmet Express.js security with HTTP headers.
https://helmetjs.github.io/

[25] JSON structural diff.
https://github.com/andreyvit/json-diff

[26] Jsonwebtoken.
https://github.com/auth0/node-jsonwebtoken#readme

[27] LdapJS Documentation.
http://ldapjs.org/

[28] Mongoose-unique-validator.
https://github.com/blakehaswell/mongoose-unique-validator#readme

[29] Morgan HTTP request logger middleware for node.js.
https://github.com/expressjs/morgan#readme

[30] Nodemon.
https://nodemon.io/

[31] Turn Buffer instances into "pointers". Ref-napi.
https://github.com/node-ffi-napi/ref-napi#readme

[32] Winston. A logger for just about everything
https://github.com/winstonjs/winston#readme

[33] Convert XML text to Javascript object / JSON text (and vice versa). Xml-js
https://github.com/nashwaan/xml-js#readme

[34] Babel is a JavaScript compiler.
https://babeljs.io/

Anexos

Equipo

El desarrollo, las pruebas y los manuales de instalación y de usuario, se han realizado en un equipo con las siguientes características:

- Sistema operativo: Ubuntu 20.04.1 LTS
- Procesador: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
- Tarjeta gráfica: GeForce GTX 1050 Mobile
- Memoria RAM: 16GB 2400Hz DDR4

Manual de instalación

En este apartado se recoge una guía explicativa de cómo poner en funcionamiento la aplicación, detallando las herramientas necesarias y sus versiones. Suponemos que partimos de una instalación limpia de Ubuntu 20.04. Antes de empezar actualizaremos la lista de repositorios del sistema mediante el comando:

```
$ sudo apt-get update
```

MongoDB

Para instalar mongodb es necesario seguir los siguientes pasos:

1. Añadir el repositorio de mongodb:
   `$ wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -`
2. Crear el archivo de lista:
   `$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list`
3. Actualizar repositorios:
   `$ sudo apt-get update`
4. Instalar MongoDB:
   `$ sudo apt-get install -y mongodb-org`
5. Iniciar el servicio:
   `$ sudo systemctl start mongod`
6. Comprobar el estado del servicio:
   `$ sudo systemctl status mongod`

Una vez realizados estos pasos, MongoDB estará listo para usarse en el sistema. Se ha conseguido realizar un despliegue exitoso utilizando la última versión hasta la fecha, la 4.4 sin embargo, si existiese algún problema relacionado con la base de datos, la versión utilizada durante todo el proyecto ha sido la 3.6.8, en este enlace se encuentran las instrucciones detalladas para la instalación de la versión en concreto, así como una guía para solucionar los problemas más habituales.

Node

Para realizar un despliegue correcto de la aplicación, es imprescindible que la versión de Node sea la v12.18.3. Para instalar esta versión se ha utilizado la herramienta nvm, la cual permite descargar, instalar y utilizar una versión concreta de Node de una manera muy sencilla.

1. Descargamos nvm:
   `$ wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash`
2. Permitimos el uso desde bash:
   `$ source ~/.profile`
3. Instalamos la versión deseada:
   `$ nvm install 12.18.3`
4. Comprobamos la versión con:
   `$ node -v`

Discovery2

Para obtener e instalar la aplicación hay que seguir los siguientes pasos:

1. Instalar git:
   `$ sudo apt-get install git`
2. Descargar el contenido del repositorio:
   `$ git clone https://gitlab.com/HP-SCDS/Observatorio/2019-2020/uva-discovery2.git`
3. Situarnos en la carpeta del proyecto:
   `$ cd /uva-discovery-2`
4. Cambiar a la rama donde se han realizado los desarrollos:
   `$ git checkout guianta`
5. Situarnos en la carpeta client:
   `$ cd /client`
6. Instalamos el CLI de Vue:
   `$ npm install -g @vue/cli`
7. Instalamos las dependencias:
   `$ npm install`
8. Construir la parte de cliente:
   `$ npm run build`
9. Crear la carpeta destino:
   `$ sudo mkdir -p /var/www/app`
10. Mover el cliente a la nueva carpeta:
    `$sudo mv ./dist/* /var/www/app`
11. Instalamos Qt5:
    `$ sudo apt-get install qt5-default`
12. Nos situamos en la carpeta del servidor:
    `$ cd ../server`
13. Instalamos las dependencias:
    `$ npm install`
14. Ejecutamos el servidor:
    `$ npm run serve`

Se ha observado que existe un error relacionado con bcrypt al instalar las dependencias, la solución que se ha encontrado es instalar este paquete por separado, de manera global, mediante el comando `npm install --save bcrypt` y después instalar el resto de las dependencias (punto 12).

Si todo ha ido bien, en la terminal actual deben aparecer los mensajes de inicialización de la aplicación.

Configuración

En este apartado se explica cada uno de los valores de los archivos de configuración y en que afectan al funcionamiento de la aplicación.

- Cliente: El archivo con la configuración de la parte del cliente se encuentra en la ruta /uva-discovery2/client/src/config/config.js, en él encontramos:
  - API_URL: Un string que contiene la dirección del servidor de la aplicación.
  - MESSAGE: Un objeto con los mensajes que se muestran en las alertas de la aplicación.
  - SOCKETS: Un objeto con los nombres de los eventos mandados por el servidor que escucha el cliente.

  Se recomienda no modificar este archivo de configuración, los parámetros modificables del cliente se almacenarán en BD al iniciar la aplicación y podrán ser editados desde la propia aplicación.

- Servidor: El archivo con la configuración de la parte del servidor se encuentra en la ruta /uva-discovery-2/server/config/config.js, en él encontramos:
  - SOCKETS: Un objeto con los nombres de los eventos que serán emitidos por el servidor.
  - MONGODB: Un objeto con la configuración de mongoDB
  - DEFAULT_ADMIN: El usuario que se creará la primera vez que se inicie la aplicación.
  - DISCOVERY: Los parámetros de funcionamiento de la aplicación, UPDATE_FRECUENCY determina cada cuanto ms se realizará la auto actualización de las impresoras, PRINTERS_DAY_EXPIRES determina cuánto tiempo pasará hasta que una impresora inactiva sea eliminada y MAX_PRINTER_LOGS será el número máximo de logs que se guardará para cada impresora.
  - LDAP: Un objeto con los parámetros de configuración de la conexión LDAP.
  - CLIENT: Un objeto con los parámetros configurables del cliente.

  Es necesario tener en cuenta que esta configuración está almacenada en la base de datos por lo que el archivo config.js solo se utiliza la primera vez que se arranca la aplicación, para que los cambios en el archivo tengan efecto, es necesario borrar la configuración de la base de datos y reiniciar la aplicación.

Nginx

Como servidor web se utilizará Nginx, para instalarlo es necesario realizar los siguientes pasos:

1. Instalar Nginx:
   `$ sudo apt-get install nginx`
2. Permitimos la conexión a través del firewall del sistema, como no disponemos de un certificado SSL, el perfil elegido será HTTP:
   `$ sudo ufw allow 'Nginx HTTP'`

3. Comprobamos que el servidor web está activo con:
4. `$ sudo service nginx status`
5. Modificamos el archivo /etc/nginx/sites-available/default:
   `$ sudo gedit /etc/nginx/sites-available/default` y lo dejamos de la siguiente manera:

```
server {
        listen 80;
        listen [::]:80;
        root /var/www/html;
        server_name _;
        location / {
          try_files $uri $uri/ = 404;
        }
}
```

6. Creamos el archivo /etc/nginx/sites-available/app:
   `$ sudo gedit /etc/nginx/sites-available/app` y lo dejamos de la siguiente manera:

```
server {
        listen 80;
        listen [::]:80;
        root /var/www/app;
        index index.html;
        location / {
          try_files $uri $uri/ = 404;
        }
}
```

7. Creamos un enlace simbólico a los sitios activos:
   `$ sudo ln -s /etc/nginx/sites-available/app /etc/nginx/sites-enabled/`
8. Comprobamos que todo está bien:
   `$ sudo nginx -t`
9. Reiniciamos el servicio:
   `$ sudo service nginx start`

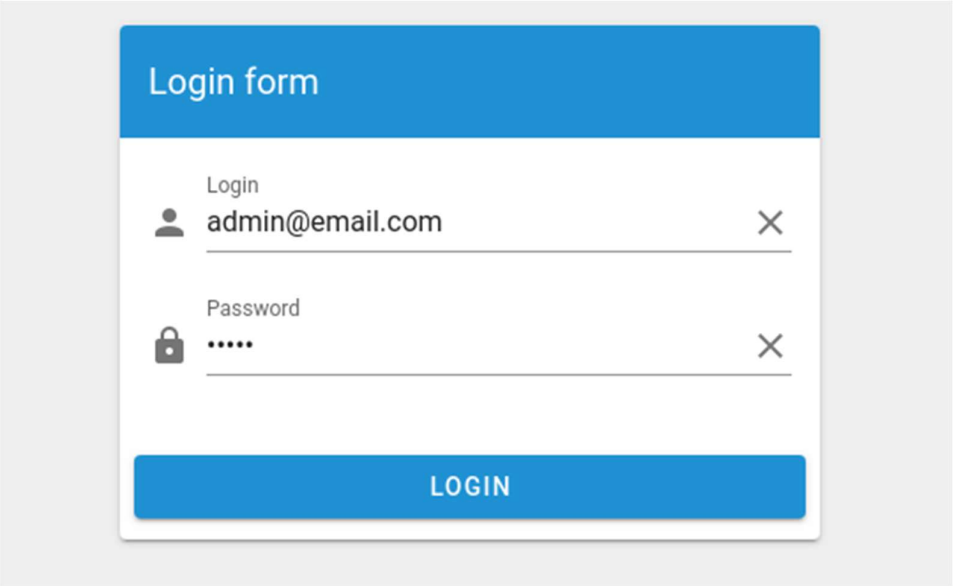
   Si se han realizado todos los pasos de manera correcta, al acceder en cualquier navegador a la dirección “localhost”, debería aparecer el formulario de login de la aplicación.

Manual de usuario

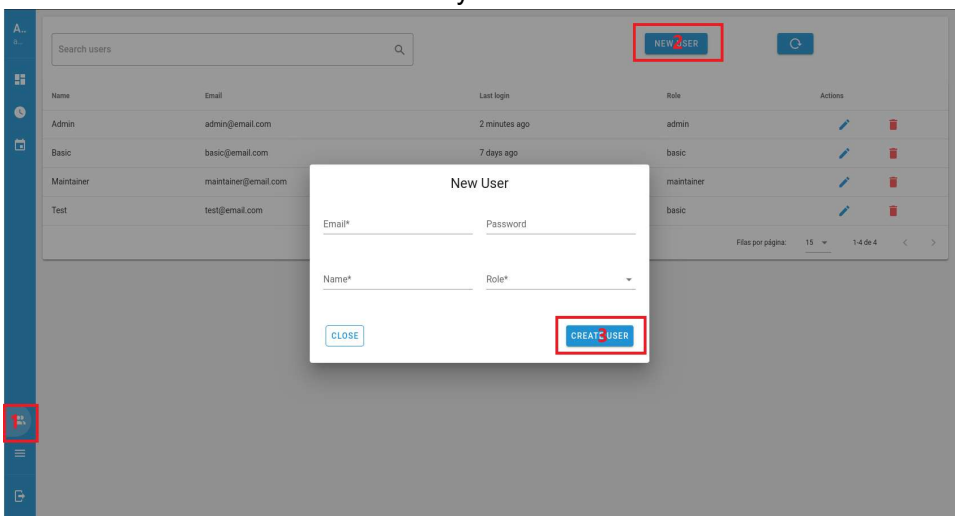
En este apartado se ejemplifica el funcionamiento de la aplicación y los flujos de trabajo más habituales en la misma. Los recursos utilizados para la realización de este manual están en el archivo printers.xml en la carpeta /server/.

1. Ejemplo 1 - Gestión de usuarios y configuración de la aplicación:

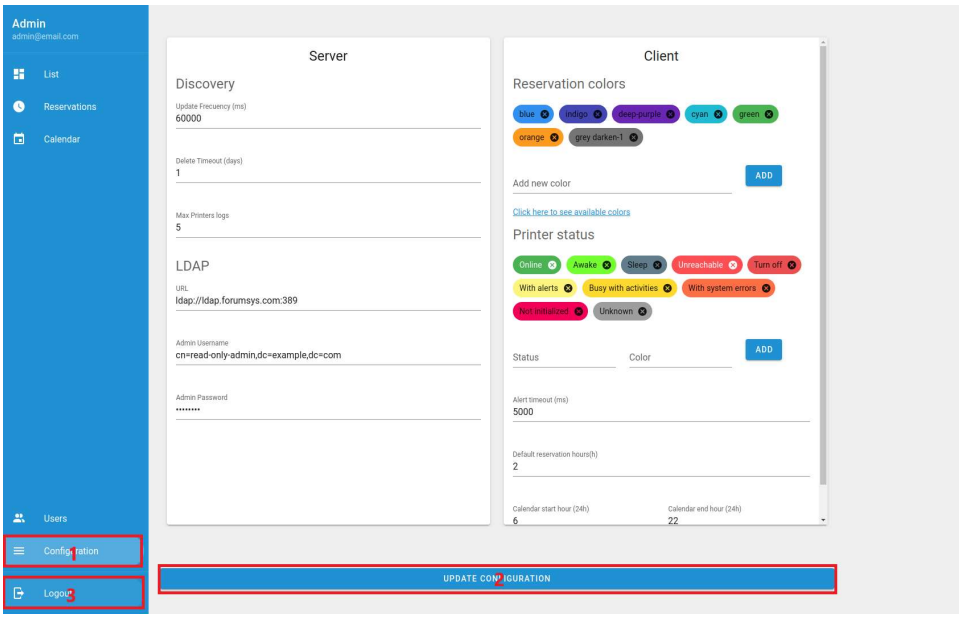
Iniciamos sesión con un usuario con rol de administrador (si es la primera vez que se inicia la aplicación email: admin@email.com, password: admin)



Después navegamos hacia la pestaña del panel izquierdo “Users” (1), hacemos click en el botón “New User” (2), rellenamos el formulario con el nombre, contraseña y mail que deseemos (3). Aprovechamos este paso para crear un usuario con rol “maintainer” y otro con rol “basic”.

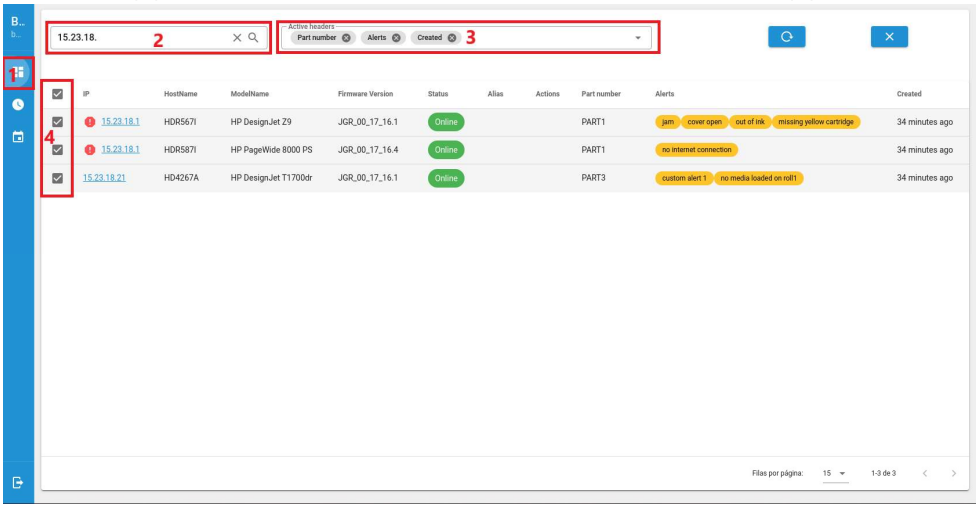


Navegamos hacia la pestaña de configuración en el panel izquierdo (1), desde esta vista podremos configurar todos los parámetros de funcionamiento de la aplicación, una vez modificados hacemos click en “Update Configuration” (2). Cerramos la sesión haciendo click en “Logout” (3), situado en la parte inferior del panel de navegación izquierdo.

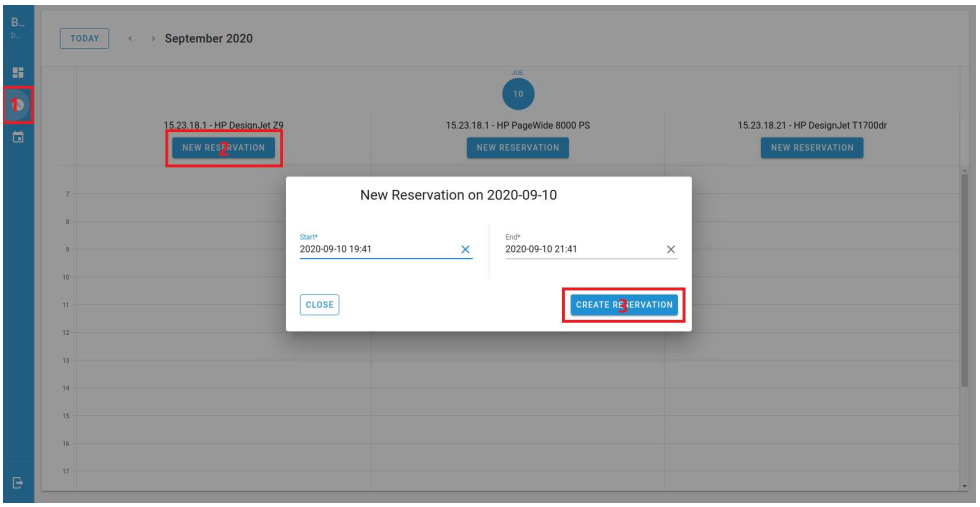


2. Ejemplo 2 - Búsqueda de recursos, creación y modificación de reservas propias:

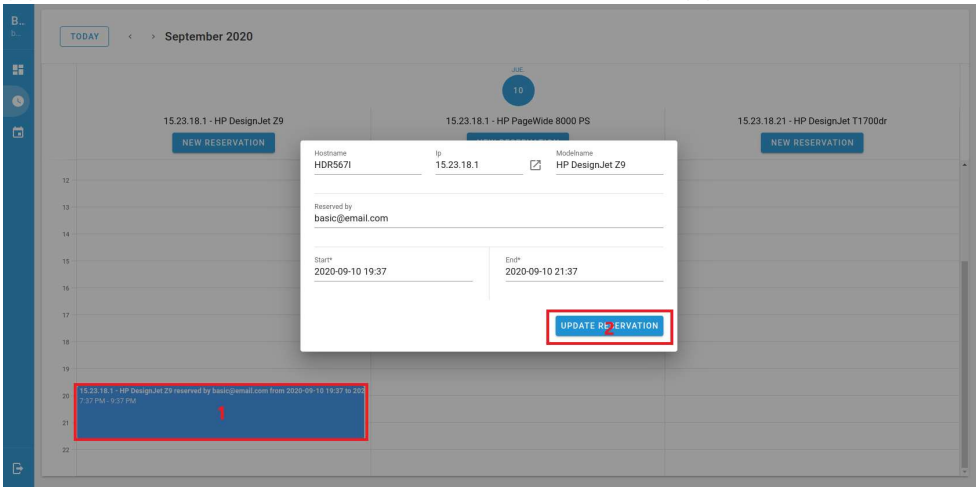
Iniciamos sesión con el usuario que hemos creado anteriormente con rol “basic”, en el panel de navegación hacemos click en “List” (1), buscamos en el cuadro de búsqueda “15.23.18.” (2), añadimos la cabecera los parámetros “alert” y “updated” (3), seleccionamos las tres impresoras que aparecen (4).



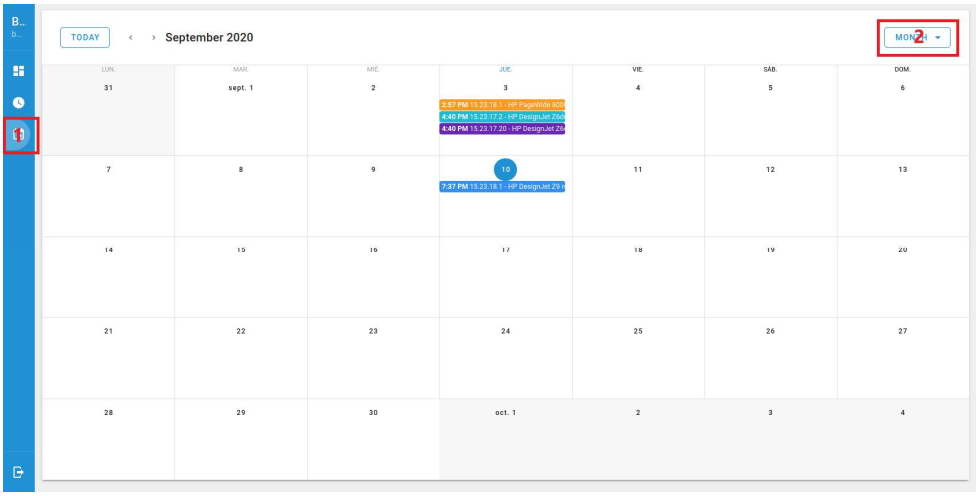
Nos desplazamos hasta la vista “Reservations” del panel de navegación izquierdo (1), creamos una nueva reserva para cualquiera de las impresoras seleccionadas mediante el botón “New Reservation” (2), rellenamos el formulario y creamos la reserva (3).



Para modificar los datos de la reserva basta con hacer click encima de la misma (1), modificar el campo deseado y actualizar la reserva (2).



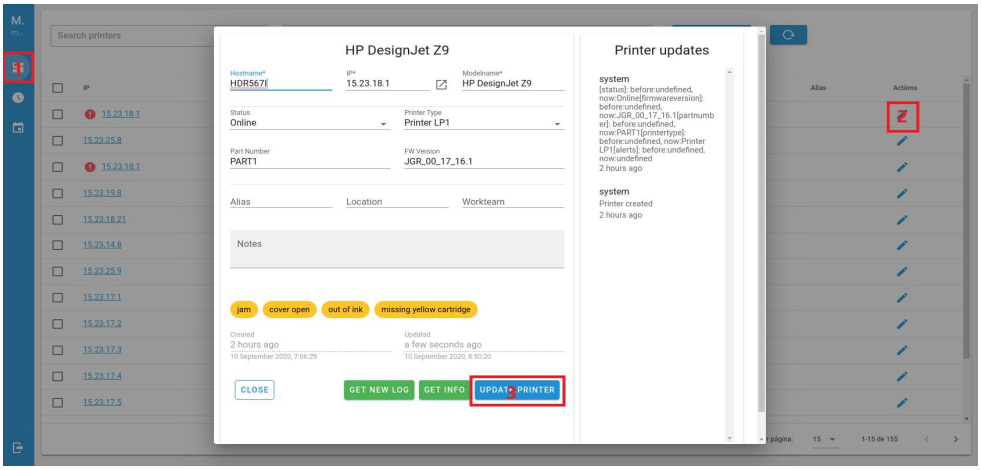
Podremos observar las reservas existentes **para las impresoras seleccionadas** desde la vista “Calendar” (1), elegimos el rango del calendario (2), desde esta vista también se pueden modificar las reservas de la misma manera.



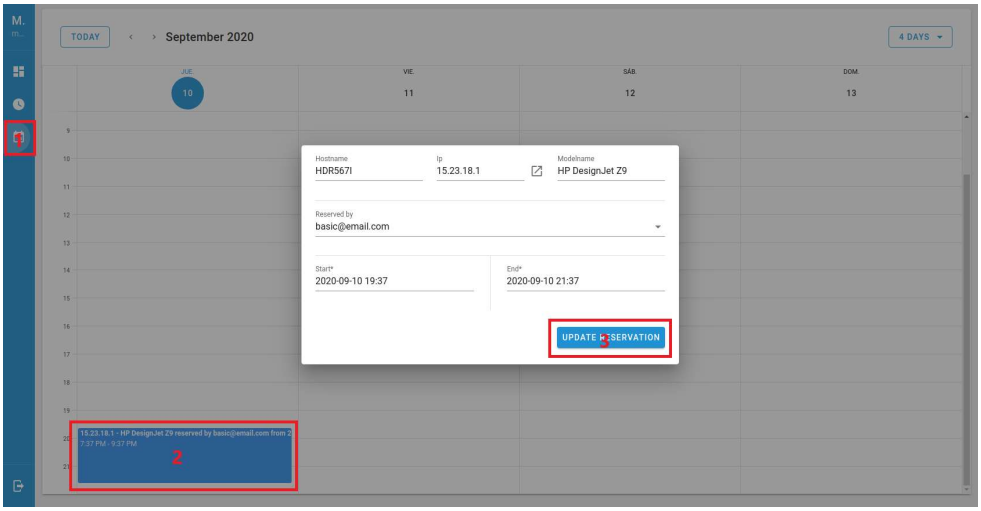
Cerramos la sesión.

3. Ejemplo 3 - Modificación de recursos y reservas:

Iniciamos sesión con el usuario con rol “maintainer” que hemos creado antes, en la vista de “List” (1), hacemos click en el icono de edición (2), actualizamos los campos deseados y hacemos click en “Update Printer” (3), en caso de que haya alguna impresora seleccionada la deseleccionamos haciendo click en el botón “X” (solo aparece si hay impresoras seleccionadas) (4).



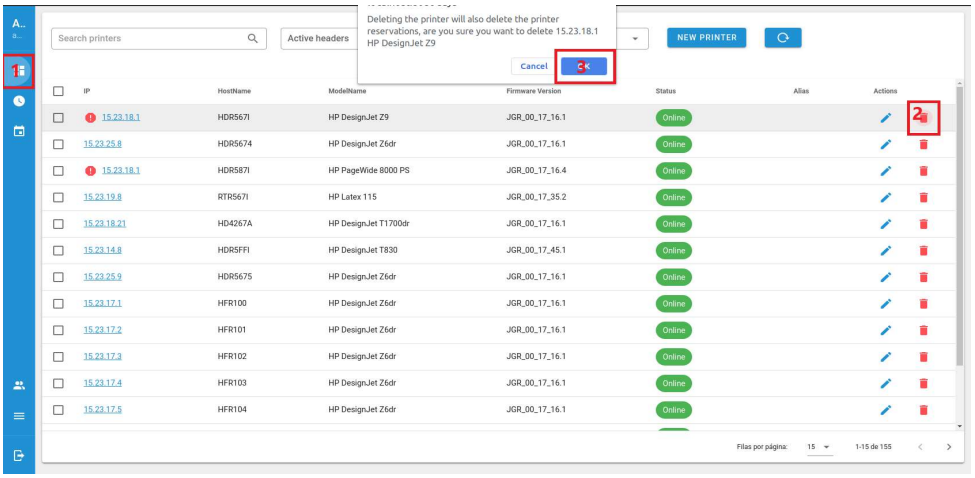
Navegamos hasta la vista “Calendar” (1), elegimos el tipo de vista (2), hacemos click encima de cualquier reserva (3) y modificamos los datos deseados y la actualizamos mediante el botón “Update Reservation” (4).



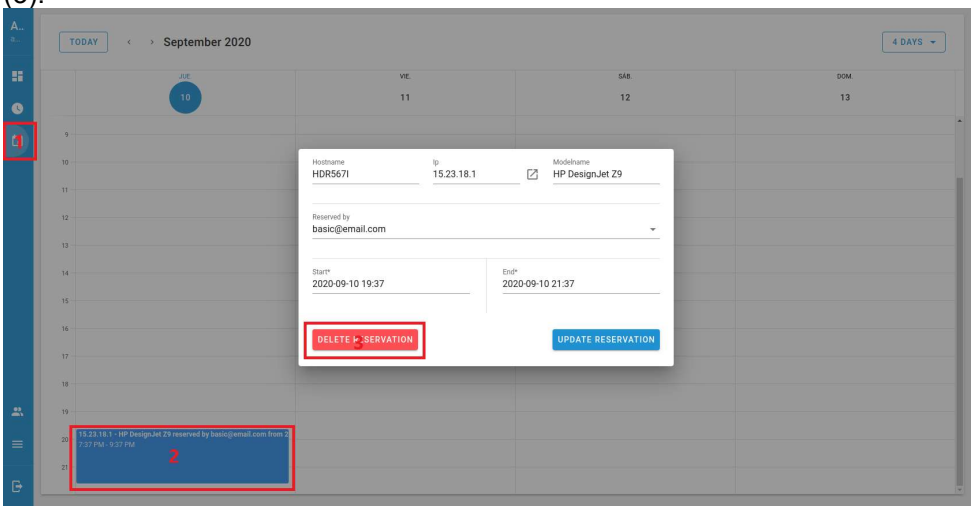
Cerramos la sesión.

4. Ejemplo 4 - Eliminación de recursos, reservas y usuarios:

Iniciamos sesión con un usuario con rol “admin”, accedemos a través del panel lateral a la vista de “List” o “Users” (1) ya que su funcionamiento es el mismo, borramos cualquier impresora o usuario haciendo click en el icono de la papelera (2), aceptamos el mensaje (3).



Si queremos borrar una reserva podremos hacerlo a través de la vista “Calendar” (1) haciendo click en la reserva deseada (2) y en el botón “Delete Reservation” (3).



Cerramos la sesión.

A tener en cuenta, las vistas de “Reservations” y “Calendar” muestran las reservas de las impresoras seleccionadas, si se desean ver todas las reservas en el calendario, primero habrá que deseleccionar en la vista “List” todas las impresoras.