UNIVERSIDAD DE SONORA

DIVISION DE INGENIERIA

Reporte de Prácticas Profesionales

Alumno: Joshua Nathanael Saucedo Uriarte

Expediente: 215205720

Carrera: Ingeniería en Sistemas de Información

Asesor: Dr. Alonso Pérez Soltero

Hermosillo, Sonora. 03 de diciembre de 2019

Índice General

Índice de Figuras

Introducción

En la reglamentación de la Universidad de Sonora se tiene como requisito que en todo plan de estudio se incluyan actividades de vinculación con el sector social o productivo, el propósito de estas actividades es complementar la formación académica de los estudiantes a través situaciones reales donde necesita aplicar los conocimientos obtenidos en las diversas materias de las carreras. Así, la carrera de Ingeniería en Sistemas de Información, del Departamento de Ingeniería Industrial, incluye en su plan de estudios las prácticas profesionales con valor 20 créditos, que son equivalentes a 340 horas de servicio.

Las prácticas serán llevadas a cabo en AD Sistemas Soluciones en TI S.A. de C.V., empresa en la cual el alumno ya se tiene experiencia laboral, ya que el alumno primero fue trabajador en dicha empresa antes que practicante. La empresa se dedica al desarrollo de software, el software desarrollado es para la misma organización aunque también se desarrollan proyectos de otras empresas.

La idea del proyecto es de la misma empresa, se basaron en el proyecto SIG el cual es de la misma empresa, el proyecto PyME que es en el cual se harán las prácticas y es el mismo proyecto en el cual el alumno comenzó a trabajar desde que entró a dicha empresa; son similares pero tienen una gran diferencia y esa diferencia radica en que SIG es para empresas grandes y se suele modificar por cliente, cada cliente tiene su propia versión de SIG en su empresa, en cambio PyME es mucho más genérico para empresas pequeñas y medianas y no se modificará bajo pedido, solo se puede acceder mediante la página web.

El proyecto PyME se comenzó a desarrollar desde el primero de Junio del 2019, desde el día del inicio del desarrollo se había seleccionado las tecnologías a utilizar por decisión de todo el equipo y experiencia previa, para FrontEnd Angular con un template llamado Metronic, BackEnd NetCore y de base de datos SQL Server. La Base de datos ya tenía el esquema de algunas tablas y relaciones ya que es similar a SIG se utilizó de la lógica de la base de datos de SIG cosas como: relaciones, campos, tablas. aunque se han realizado cambios durante el desarrollo de PyME.

Explicación Del Proyecto

La intención del proyecto es crear técnicamente un ERP para pequeñas y medianas empresas, control de inventarios, validaciones del SAT, ventas, compras. Estos son algunos de los puntos más importantes que tiene que cubrir la aplicación para cumplir satisfactoriamente con los requerimientos del sistema.

La aplicación se separa en dos partes importantes, backend y frontend, el backend se desarrolla con NetCore y ASP, se desarrolla una API Restful y desde el frontend el cual está hecho en Angular consume esa API Restful para mostrar información y hacer operaciones, se decidió hacer esa separación para tener libertad al momento de hacer cambios y sea más fácil de escalar.

Objetivos

El objetivo general es desarrollar una aplicación que ofrezca una solución para el control de inventario, empleados, socios y todo el proceso contable relacionado con el SAT para las pequeñas y medianas empresa.

Entre los principales objetivos específicos se tiene:

- Concluir con el proyecto para poder comerciar con él.
- Adquirir experiencia para desarrollo y crecimiento de una empresa.
- Adquirir experiencia desarrollando software para control de inventario.

Metodología

La metodología aplicada durante las prácticas desarrolladas por el estudiante consistió en tres etapas generales que se explican brevemente a continuación.

La primera etapa consistió en el análisis y comprensión del sistema previo, como el nuevo sistema a desarrollar iba a ser una pequeña implementación del sistema llamado SIG, quitando algunos módulos y agregando otros, durante esta etapa se hicieron bosquejos de comunicación y lógica, eventualmente estos bosquejos se volvieron recurrente en el desarrollo del sistema.

La segunda etapa consistió en el desarrollo del backend, la cual es la parte lógica de la aplicación, en esta etapa se desarrollaron storage procedure para guardar, modificar, eliminar y obtener datos de la base de datos, endpoint con una api RestFul y una arquitectura basada en componentes, para una rápida escalabilidad y sencillo, mantenimiento, así no se genera un hell callback

dependency, y el desarrollo de nuevos endpoints se vuelve mucho más rápida y ágil.

La tercera etapa consistió en el análisis y desarrollo del frontend basado en angular, como el backend está basado en componentes, el front se desarrolló de la misma manera, con componentes genéricos, utilizando enums, listas, clases genéricas y peticiones por http. Esta fue la parte donde se integra todo el front con el back para que el usuario pueda utilizar el sistema a través de nuestro web client. aunque también se puede utilizar a través api RestFul.

Entorno donde se ubica la unidad receptora

La empresa se encuentra en una zona urbana pero cerca de zonas comerciales, haciendo que a futuro la empresa o local aumente en su plusvalía,la colonia donde se encuentra es Nueva Galicia, calle Paseo Florido número 62. En la Figura 1 se muestra una foto de la empresa.

Figura 1: Ubicación de la empresa AD Sistemas

Problemas planteados para resolverlos

- Que los clientes se puedan hacer cargo de sus propios reportes, qué campos mostrar, cómo, imágenes, información y finalmente el diseño.
- Seguridad, que haya seguridad tanto de la parte de la API como en el web client.
- Fácil de escalar, que en el proyecto no existan complicaciones mayores al momento de querer integrar nuevos endpoints o nueva lógica de negocios.
- Alto rendimiento, que el procesamiento de datos no tome tanto tiempo, ya que los clientes se prevé que sean miles.
- Eventos o ejecuciones de tareas en cierto momento del día, como actualizaciones del SAT.

Alcances y limitaciones en la solución de los problemas

La principal limitación del desarrollo del software es la limitada cantidad de recursos con las que se cuenta para el desarrollo del sistema, actualmente sólo el estudiante es quien desarrolla el sistema PyME, por lo que el tiempo disponible para la planeación y elaboración del sistema es bastante limitado. Y no solo el tiempo, ya que los recursos, es decir, el servidor es algo viejo, cuenta con un procesador i5 de 7ma generación, pero no es el único sistema que corre en ese servidor, y los servicios como base de datos o vpns.

El alcance del sistema se planteó para ir cambiando conforme pase el tiempo, planeando una versión inicial solo para clientes que se considere su operación lo suficientemente simple para funcionar con pocos módulos del sistema, conforme diferentes clientes adopten el sistema. El alcance de este seguirá creciendo con el desarrollo de nuevos módulos y nuevas funcionalidades para abarcar la operación de nuevos clientes y hacer del sistema una opción más atractiva.

Fundamento teórico de las herramientas y conocimientos aplicados

Para el desarrollo principal del sistema (algunas otras herramientas serán discutidas más adelante) se optó por trabajar con tecnologías ya manejadas por

otros sistemas de AD Sistemas, aprovechando la experiencia ya obtenida por los desarrolladores para agilizar el desarrollo del sistema.

Se utilizó C# con el engine de netcore2.0 en la parte de backend, de frontend se utilizó Angular 6 aunque se actualizo a la versión 8 durante el desarrollo del sistema, la razón para utilizar netcore2.0 es por su funcionalidad de multiplataforma, y así poder mover el sistema a cualquier servidor sin importar el sistema operativo, y que cualquier desarrollador, ya sea con linux, macos o windows se pueda integrar fácilmente al desarrollo del sistema.

Un conocimiento teórico que se utilizó durante todo el desarrollo del sistema es el uso de genéricos, de clases y listas genéricas para ahorrar bloques de código o procesamiento para castear datos y convertirlos al tipo de dato que necesitas.

código para generar un objeto genérico

```
public T CreateObject<T>(SqlDataReader dataReader) where T : class, new()
{
    var item = (T)null;
    while (dataReader.Read())
    {
        item = Activator.CreateInstance<T>();
        foreach (var property in typeof(T).GetProperties())
        {
            if (!HasColumn(dataReader, property.Name))
            {
                continue;
            }

            if (dataReader[property.Name] != DBNull.Value)
            {
                Type convertTo = Nullable.GetUnderlyingType(property.PropertyType) ?? property.PropertyType;
                property.SetValue(item, Convert.ChangeType(dataReader[property.Name], convertTo), null);
            }
        }

    }

    return item;
}
```

Figura 2: Código para generar un objeto genérico

código para generar una lista generica

```
public List<T> CreateList<T>(SqlDataReader dataReader) where T : class, new()
{
    var results = new List<T>();
    while (dataReader.Read())
    {
        var item = Activator.CreateInstance<T>();
        foreach (var property in typeof(T).GetProperties())
        {
            if (!HasColumn(dataReader, property.Name))
            {
                continue;
            }

            if (dataReader[property.Name] != DBNull.Value)
            {
                Type convertTo = Nullable.GetUnderlyingType(property.PropertyType) ?? property.PropertyType;
                property.SetValue(item, Convert.ChangeType(dataReader[property.Name], convertTo), null);
            }
        }

        results.Add(item);
    }

    return results;
}
```

Figura 3: Código para generar una lista generica

De esta manera, sin importar qué clase o modelo es el que se utiliza o se manda a llamar, siempre utilizará el mismo código, de esa manera se ahorra el andar desarrollando un reader para cada clase o modelo.

Procedimientos empleados y actividades realizadas

A continuación, se enlistan los diferentes módulos o funcionalidades del sistema desarrollados, descripciones de su funcionamiento y capturas de pantalla de las ventanas involucradas.

BackEnd

En primer lugar, la estructura basa en componentes es como se muestra en la Figura 4.

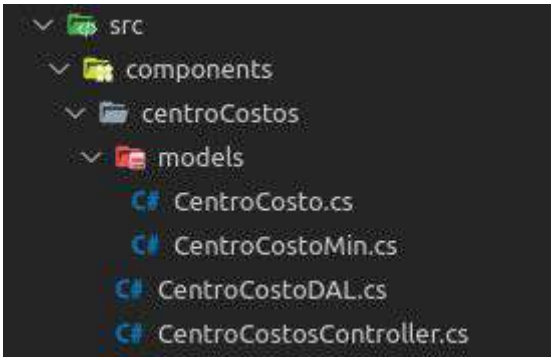


Figura 4: Estructura basada en componentes

El controlador es donde se encuentra la lógica necesaria para responder y tratar las peticiones solicitadas por el cliente, los DAL que es Data Access Layer, se utilizan para comunicarse con la base de datos, hacer las consultas necesarias, ya sea para insertar, actualizar, eliminar o solo pedir información, los modelos, en cada componente se pueden tener varios modelos, el modelos es dinámico, en un endpoint puedes solicitar o responder con un modelo en especifico y en otro endpoint con cualquier otro modelo, de esta manera se pueden tener varios modelos que se adecuen a las necesidades del endpoint.

Como la lectura de datos es de manera dinámica, no se necesita tener que volver a programar algo, o insertar mas código del que se necesita, si los datos o campos están o son regresados por el Storage Procedure, el método es capaz de mapearlo.

En cuanto a la seguridad, se utilizó JWT, json web token, se guarda la información necesaria para las consultas en la base de datos, como idUsuario, username, idEmpresa, etc, de esta manera, cada vez que se hace alguna petición a algun endpoint, necesita pasar en las cabeceras el a campo *Authorization* con el token de jwt, ejemplo: Authorization:'Bearer jwt-token'.

De esta manera, se verifica al usuario y se tienen los valores necesarios para limitar su alcance de escritura y lectura, si no se coloca el jwt en la cabecera es imposible leer o escribir en la api. ejemplo de un endpoint validado con jwt. Se muestra en la Figura 5.

```
// GET api/centrocostos
[HttpGet]
[Authorize]
public ActionResult Get()
{
    var USER = HttpContext.User;
    int CveEmpresa = Int32.Parse(USER.FindFirst("CveEmpresa").Value);

    return Ok(_centroCostoDAL.AllCentroCostosByCveEmpresa(CveEmpresa));
}
```

Figura 5: Código para tomar el JWT

Ya en este punto se llevan dos problemas resueltos, seguridad y escalabilidad, ahora se necesita resolver el primer problema, que los clientes se hagan cargo de sus propios reportes. Para ello se implementó *Stimulsoft*, es una librería para la creación de reportes, tiene un render web, así que cada cliente puede editar sus propios reportes y adecuarlos a sus necesidades, logos, campos, colores, etc. (ver Figura 6).

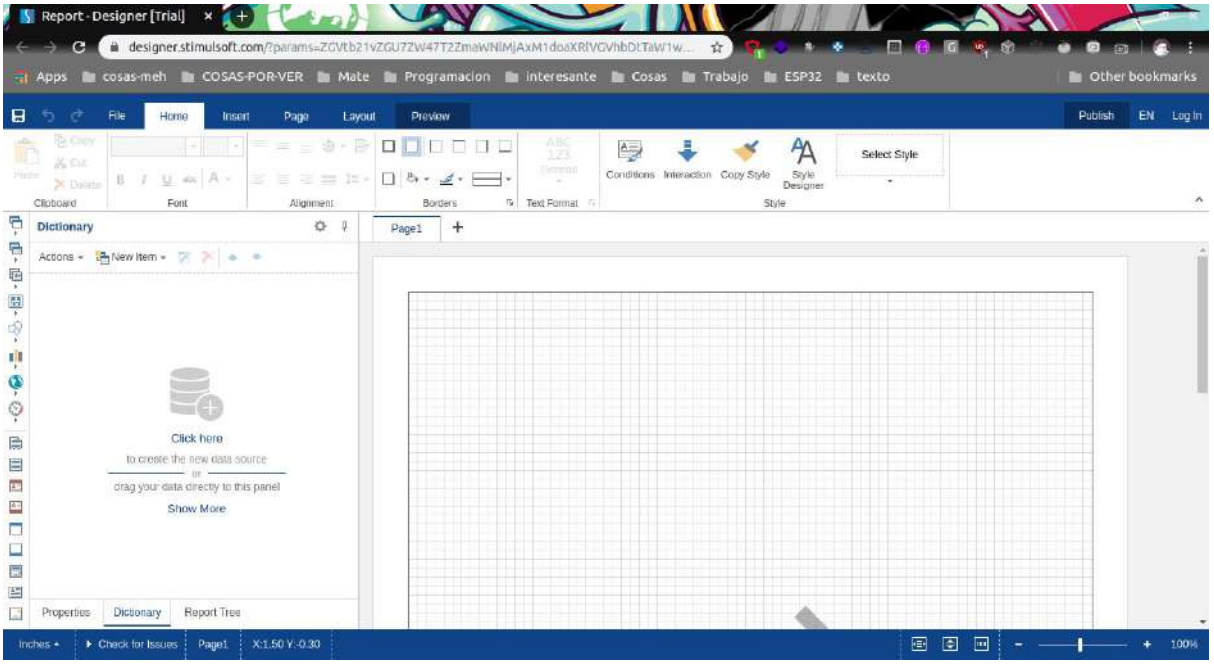


Figura 6: Stimulsoft web client

En este momento solo faltan dos puntos para lograr resolver: alto rendimiento en el procesamiento de datos y la actualización periódica del SAT.

Para el alto rendimiento, se crearon pools en el servidor de sql, así poder atender más de una petición al mismo tiempo, y todo es tratado con Storage procedures, y se siguieron todas las normas de calidad y rendimiento recomendado en SQLServer.

Para las actualizaciones periódicas del SAT, inicialmente se había utilizado cron jobs in windows, que es el task scheduler, el problema que resultó con esta solución, es que a veces algunas tareas no eran ejecutadas, así que, se optó por desarrollar un subsistema para estas tareas periódicas, ya que eventualmente iban a ser más, backups, actualizaciones, validaciones, etc. El subsistema desarrollado fue nombrado *Automat*, en este subsistema se utilizó *Quartz* la cual es una librería para ejecutar tareas de manera periódica, ya sea con la estructura de cron jobs o con un archivo de configuración. Se puede observar el ejemplo en la Figura 7.

```
// Lanzamos el job a las 22:10:00 y se repite cada 24 hrs
appStartTrigger = TriggerBuilder.Create()
    .WithIdentity("solicitudYdescarga", "sat")
    .StartAt(DateBuilder.DateOf(22, 10, 00)) //22:10:00
    .WithSimpleSchedule(s => s
        .WithIntervalInHours(24)
        .RepeatForever())
    .Build();
```

Figura 7: Archivo de configuración para Quartz

En este momento, se resolvieron los cinco problemas planteado para resolver en cuanto al BackEnd, ya se tiene la seguridad, escalabilidad y fácil mantenimiento, los usuarios se hacen cargo de sus propios reportes, tareas que se ejecutan de manera periódica, y alto rendimiento en el procesamiento de datos. Solo nos falta el frontend, el web client donde el usuario interactuara con el sistema.

FrontEnd

Como todo el backend se basa en componentes, el frontend no es diferente, se crearon tablas y páginas genéricas para poder ser reutilizados y ahorrar código y tiempo de desarrollo, mientras menos código, menos cosas que mantener y actualizar y es más fácil de comprender.

Primero se verán las tablas. La primera tabla se llama *tableInf*, esta tabla es una tabla infinita, es decir, tiene implementación de infinite scroll, al llegar al final de

la lista hace una petición *http* solicitando más ítems, los *inputs* para esta tabla se ven en la Figura 8.

```
@Input() displayedColumns: any[];
@Input() dataSource;
@Input() Titulo: string;
@Input() uniqueID: any;
@Input() actions: string[];

@Output() eventos = new EventEmitter();
```

Figura 8: Inputs para la tabla tableInf

El *displayedColumns*, es el nombre de los atributos que se desea mostrar, digamos que se tiene el siguiente objeto Figura 9.


```
{
  "name": "Yavin IV",
  "rotation_period": "24",
  "orbital_period": "4818",
  "diameter": "10200",
  "climate": "temperate, tropical",
  "gravity": "1 standard",
  "terrain": "jungle, rainforests",
  "surface_water": "8",
  "population": "1000",
  "residents": [],
  "films": [
    "https://swapi.co/api/films/1/"
  ],
  "created": "2014-12-10T11:37:19.144000Z",
  "edited": "2014-12-20T20:58:18.421000Z",
  "url": "https://swapi.co/api/planets/3/"
}
```


Figura 9: Primer objeto de ejemplo

Y de ese objeto solo se quiere mostrar *name* y *climate*, entonces *displayedColumns* se pasa en un string esos dos atributos, para ser mostrado en la tabla, *dataSource*, es la lista inicial, *Titulo*, es el título que se quiere mostrar en la tabla, uniqueID, es la clave única del objeto, como sería idUsuario, actions, es la lista de acciones, ahí pasa qué acciones puede hacer, crear, editar, eliminar. Ejemplo de la tabla con un ítem se muestra en la Figura 10.

Figura 10: Ejemplo de la tabla tableInf

Para esta tabla, estos fueron los *inputs* utilizados, excepto por *actions*, se agregó *delete* en la tabla utilizada para la foto (ver Figura 11).

```
displayedColumns = ['ps', 'cveProductoServicio']
dataSource = null
Titulo = 'Catálogo de Productos y Servicios'
uniqueID = 'cveProductoServicio'
actions = ['edit']
```

Figura 11: Inputs utilizados en tableInf

El *dataSoruce* se llena después con una petición *http* (ver Figura 12).

```
private getProductosYServicios(clear: boolean) {
  this.productosYServicios.getAllProductosYServicios()
    .subscribe(
      dataRes => {
        if (clear) {
          this.dataSource.data = []
        }

        if (this.dataSource === null) {
          this.dataSource = new MatTableDataSource(dataRes)
        } else {
          const data = this.dataSource.data
          data.push(...dataRes)
          this.dataSource.data = data
        }

        this._ref.detectChanges()
      },
      err => {
        Utils.errorMessage('Error al Buscar', err)
      }
    )
}
```

Figura 12: El dataSoruce utilizado en tableInf

Ahora se verá la *tablePage.* Esta tabla es muy similar a la tabla *tableInf*, la diferencia es que esta página cuenta con paginado y no hace nuevas peticiones *http*, o no de manera automática, cuenta exactamente con los mismos *inputs* que *tableInf*, mirar Figura 8, una demostración de este tipo de tabla se observa en la Figura 13.

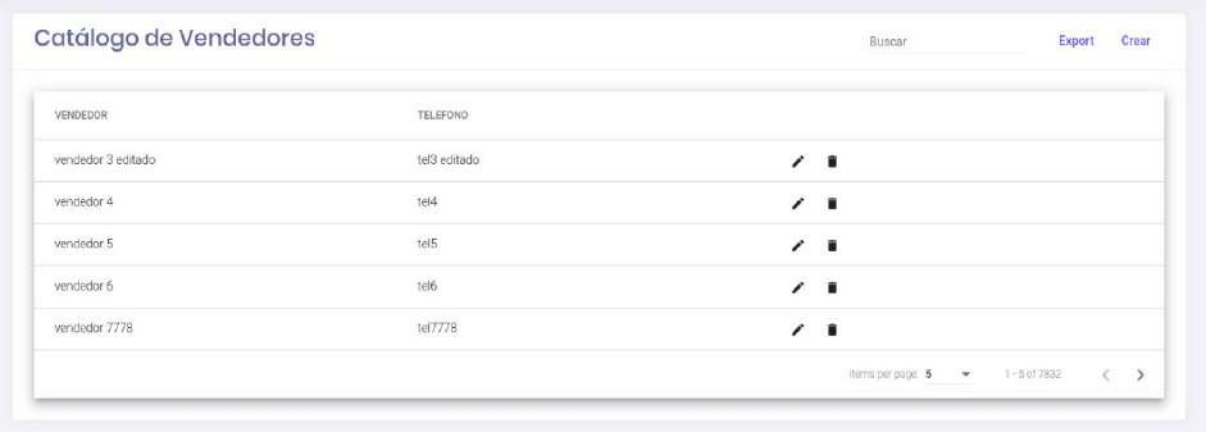


Figura 13: Ejemplo de la tabla tablePage

Los datos utilizados para la tabla de la Figura 13 se pueden ver en la Figura 14.

```
displayedColumns2 = ['vendedor'];
dataSource2 = null;
Titulo2 = 'Catálogo de Vendedores';
uniqueID2 = 'cveVendedor';
actions = ['edit', 'delete'];
```

Figura 14: Inputs utilizados en tablePage

El *dataSoruce* se llena después con una petición *http* (ver Figura 15).

```
private getVendedores(clear: boolean) {
  this.vendedores.getAllVendedores()
    .subscribe(
      dataRes => {
        if (clear) {
          this.dataSource2.data = [];
        }

        if (this.dataSource2 === null) {
          this.dataSource2 = new MatTableDataSource(dataRes);
        } else {
          const data = this.dataSource2.data;
          data.push(...dataRes);
          this.dataSource2.data = data;
        }
        this._ref.detectChanges();
      },
      err => {
        Utils.errorMessage('Error al Buscar', err)
      }
    )
}
```

Figura 15: El dataSoruce utilizado en tablePage

Como se puede ver, la forma de implementar esas tablas genéricas es completamente igual, gracias a eso, si se necesita hacer un cambio de una página con infinite scroll a una con paginado es completamente facil, solo cambia el *import* del documento.

En el *html* del componente de angular donde se desea utilizar la tabla se cambia de Figura 16 a Figura 17

```
<table-page [Titulo]="Titulo2" [actions]="actions" [dataSource]="dataSource2" [displayedColumns]="displayedColumns2"
    [uniqueID]="uniqueID2" (eventos)="tableScroll($event)"></table-page>
```

Figura 16: Como usar tablePage en html

```
<table-inf [Titulo]="Titulo" [actions]="actions" [dataSource]="dataSource" [displayedColumns]="displayedColumns"
    [uniqueID]="uniqueID" (eventos)="tableScroll($event)"></table-inf>
```

Figura 17: Como usar tableInf en html

De esa manera, se puede hacer cualquier cambio de tabla en cuestión de segundos sin problema alguno. Todos los componentes funcionan de la misma

manera, genéricos, solo los forms de cada ítem es necesario desarrollarlo manualmente.

Ahora se verá *tableCheck*. Esta tabla genérica se utiliza cuando se necesita regresar los ítems seleccionados de una lista, solo los items seleccionados, así ya se puede hacer lo que se necesita con esa lista de objetos seleccionados, en este caso, se utilizó para relacionar los módulos con los perfiles o con los usuarios.

Un usuario tiene un perfil designado, de esa manera se puede crear el perfil vendedor, y se le agregan los módulos que se necesitan, como ventas e inventarios, y se comienza a crear los usuarios que serán vendedores y se les selecciona ese perfil. Los *inputs* de esta tabla se pueden ver en la Figura 18.

```
@Input() displayedColumns: any[];
@Input() dataSource;
@Input() Titulo: string;
@Input() selects: string[];

@Output() eventos = new EventEmitter();
```

Figura 18: Inputs para la tabla tableCheck

Un ejemplo de la implementación de esta tabla se muestra en la Figura 19.

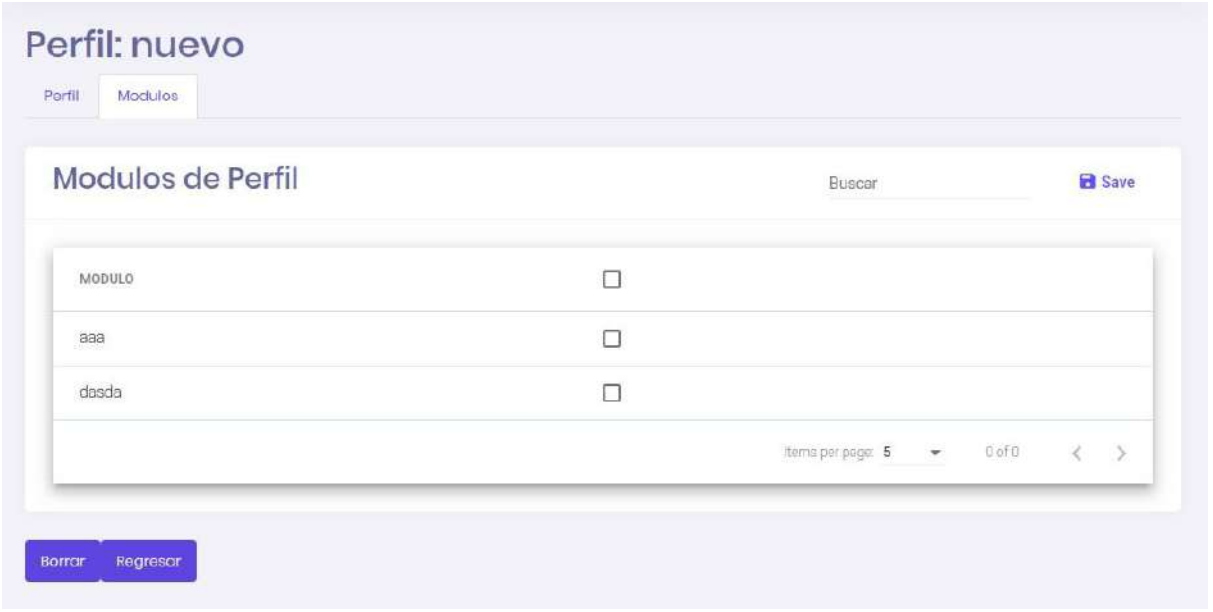


Figura 19: Ejemplo de la tabla tableCheck

Los inputs de esta tabla se pueden observar en la Figura 20.

```
displayedColumns = ['modulo', 'select'];
Titulo = 'Modulos de Perfil';
dataSource = null;
selects = ['select'];
```

Figura 20: Inputs utilizados en tableCheck

La última tabla desarrollada de manera genérica fue *tableInfModal*. Esta tabla fue desarrollada para poder crear objetos, editarlos y eliminarlos de manera genérica, los inputs que recibe esta tabla se ven en la Figura 21.

```
@Input() displayedColumns: any[]
dataSource
@Input()
set setDataSource(dataSource: any) {
  if (dataSource === null) {
    return
  }
  this.dataSource = new MatTableDataSource(dataSource)
}
@Input() Titulo: string
@Input() actions: string[]
@Input() form
@Input() checkList: []
@Input() ModalType: ModalList

@Output() eventos = new EventEmitter()
```

Figura 21: Inputs para la tabla tableInfModal

Un ejemplo de implementación se muestra en la Figura 22.

```
dataSource = null
displayedColumns = ['cantidad', 'ps', 'pu', 'descuento', 'ivaPorcentaje', 'total']
Titulo = 'Detalles de Venta'
actions = ['edit', 'delete']
ModalType = ModalListVentas
form = {
  cantidad: 0.0,
  observacion: '',
  pu: 0.0,
  descuento: 0.0
}
```

Figura 22: Inputs utilizados en tableInfModal

Como se puede ver, los nuevos atributos que se agregaron son *form* y *ModalType*, form es el tipo de objeto que se desea mostrar en el formulario, de ahí su nombre form. *ModalType* es el tipo de modal, un modal puede heredar al *Modal* padre y luego implementar la lógica que necesite de manera separada, en este caso instanciamos *ModalListVentas*, que hereda de *ModalList*.

Con eso se termina todas las tablas, pero aún falta otros componentes genéricos, como los select. Existen dos tipos de select genéricos, uno solo regresa el valor del item seleccionado y también una query url. El segundo select hace peticiones http mientras se escribe, y no solo regresa el valor del item seleccionado, también muestra información de dicho ítem, como sería el RFC o nombre de algún usuario dentro del selec detallado. Ejemplo de select genérico se encuentra en la Figura 23.

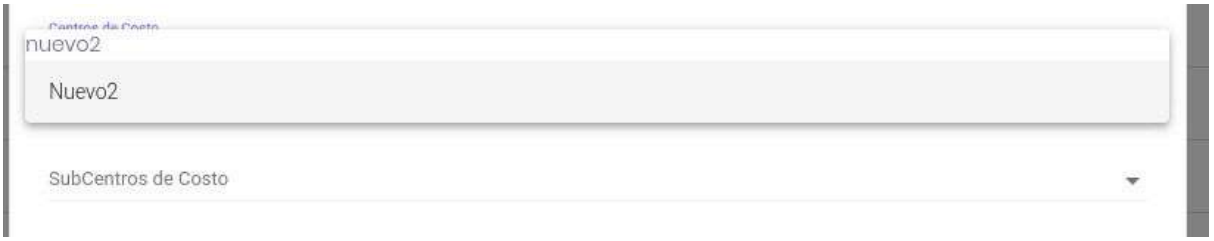


Figura 23: Ejemplo de select genérico

Esto es lo que se hizo de manera superficial, también se utilizaron mensajes de notificación, rxjs, sweetalert, entre más herramientas y librerías, un template llamado metronic del cual se tomó la base para desarrollar los componentes del sistema.

Resultados obtenidos

Los resultados obtenidos son exactamente lo que se esperaba. Tiene seguridad gracias a jwt, es fácil de escalar y mantener gracias a la arquitectura basada en componentes. La api tiene múltiples tipos de integraciones gracias al tipo de api que se utilizó RestFul, así que las respuestas y peticiones son en formato json. Tiene tareas programadas para estar actualizando la información del SAT y crear backups. En cuanto al frontend sigue la arquitectura basada en componentes, así que igualmente es fácil de escalar y mantener, gracias a los componentes genéricos es fácil cambiar de componente al que se desea.

Conocimientos aplicados

Se aplicó en su mayoría todo el rango de conocimientos adquiridos en la carrera, el principal fue el de la clase de base de datos para el diseño de esta en la aplicación. Los cursos de programación fueron esenciales en plantear las bases para el desarrollo en netcore2.0 y Angular, además de un refuerzo en tópicos avanzados de computación fue crucial para mi aprendizaje del lenguaje con el que fue desarrollado el sistema, como la arquitectura basada en componentes y la creación de una API.

Sin embargo, los conocimientos más valiosos fueron los de las clases donde el profesor se enfocó en el diagramado de sistemas para su desarrollo (UML), hacer uso oportuno de estos diagramas dio lugar a muchas soluciones al interactuar con el diagrama creado del sistema, ya sea por piezas faltantes o redundancias que causan problemas en la consistencia de datos.

La transición de los conocimientos a lo práctico no fue algo sencillo, ya que cada empresa tiene sus propias ideologías en cómo desarrollar el Software. En el caso de AD Systemas se empeña más en un enfoque al cliente para que sienta que tiene el soporte adecuado, ya que hacen cambios solicitados por clientes, pero ese es en el caso del SIG, este sistema PyME, funciona de manera mucho más general, lo cual no es común en dicha empresa. Para maximizar la eficiencia de este progreso, existen rigurosos estándares para el “Log” de errores inesperados, esto ayuda a que el proceso de corregir errores en el sistema sea más adecuado, pero requiere cierto cambio por parte de cada desarrollador adaptar estas políticas.

Conclusiones y Recomendaciones

El desarrollo de un sistema ERP para PyME me ayudó mucho a internalizar muchos de los conocimientos que obtuve de diferentes clases a través de la práctica. El hecho de haber tenido la oportunidad de participar en el desarrollo desde su inicio me dio la oportunidad de aplicar mis conocimientos en la recaudación de información para la planeación adecuada del alcance del sistema antes de comenzar su desarrollo. Además del uso de la programación orientada a objetos para mantener el sistema limpio, coherente y preparado para continuar creciendo de una manera sustentable para su desarrollo continuo y agregando una arquitectura basada en componentes junto con el uso de una API RestFul.

Para futuras personas que buscan realizar sus prácticas profesionales en una empresa trabajando como desarrolladores de software, recomiendo tomar su tiempo

para buscar alguna empresa donde puedan participar en un proyecto de este carácter, donde puedan formar parte del proceso completo del desarrollo de software y tengan la oportunidad de expresar sus ideas y obtener retroalimentación inmediata de las personas trabajando en el proyecto.

Referencias

https://docs.microsoft.com/es-es/dotnet/

https://angular.io/guide/releases

https://update.angular.io/

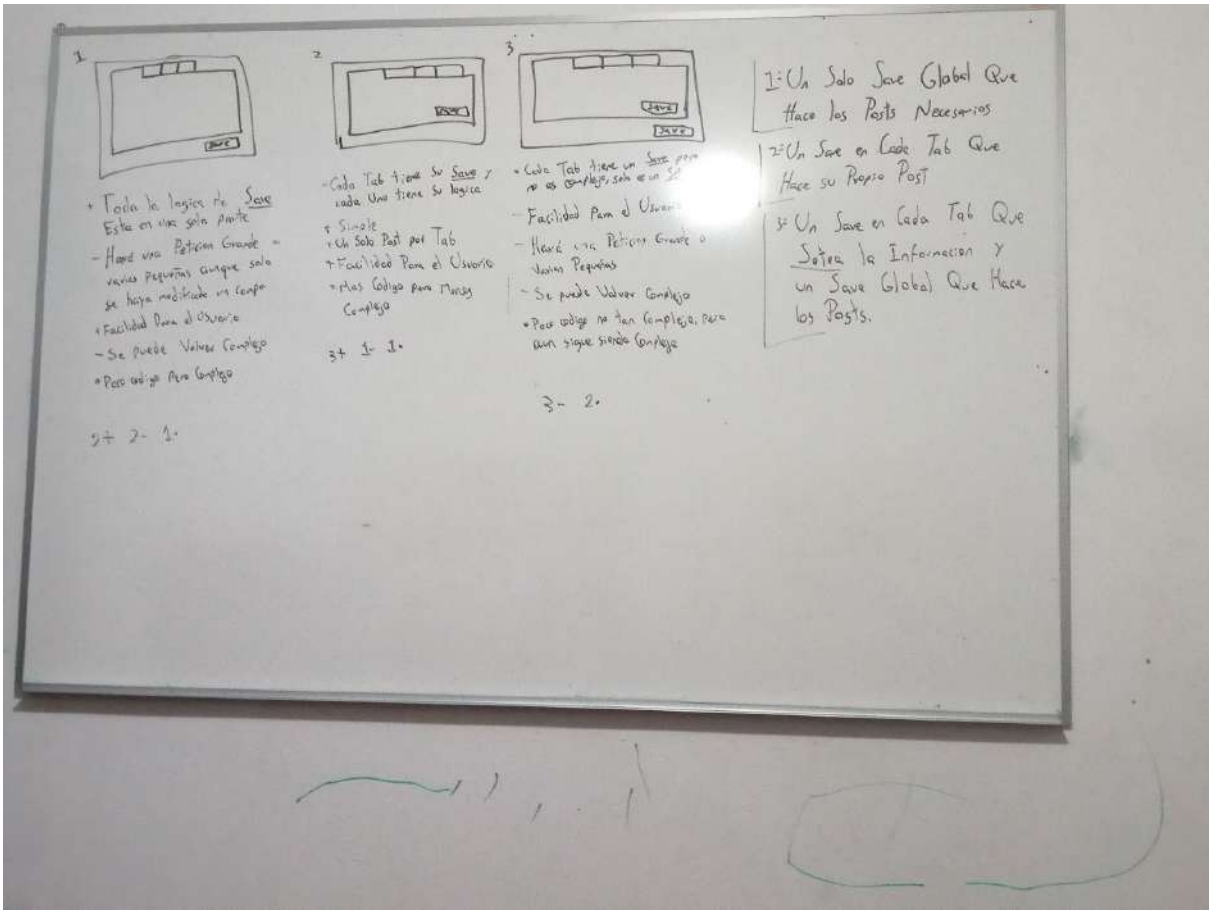
https://medium.com/omarelgabrys-blog/component-based-architecture-3c3c23c7e348

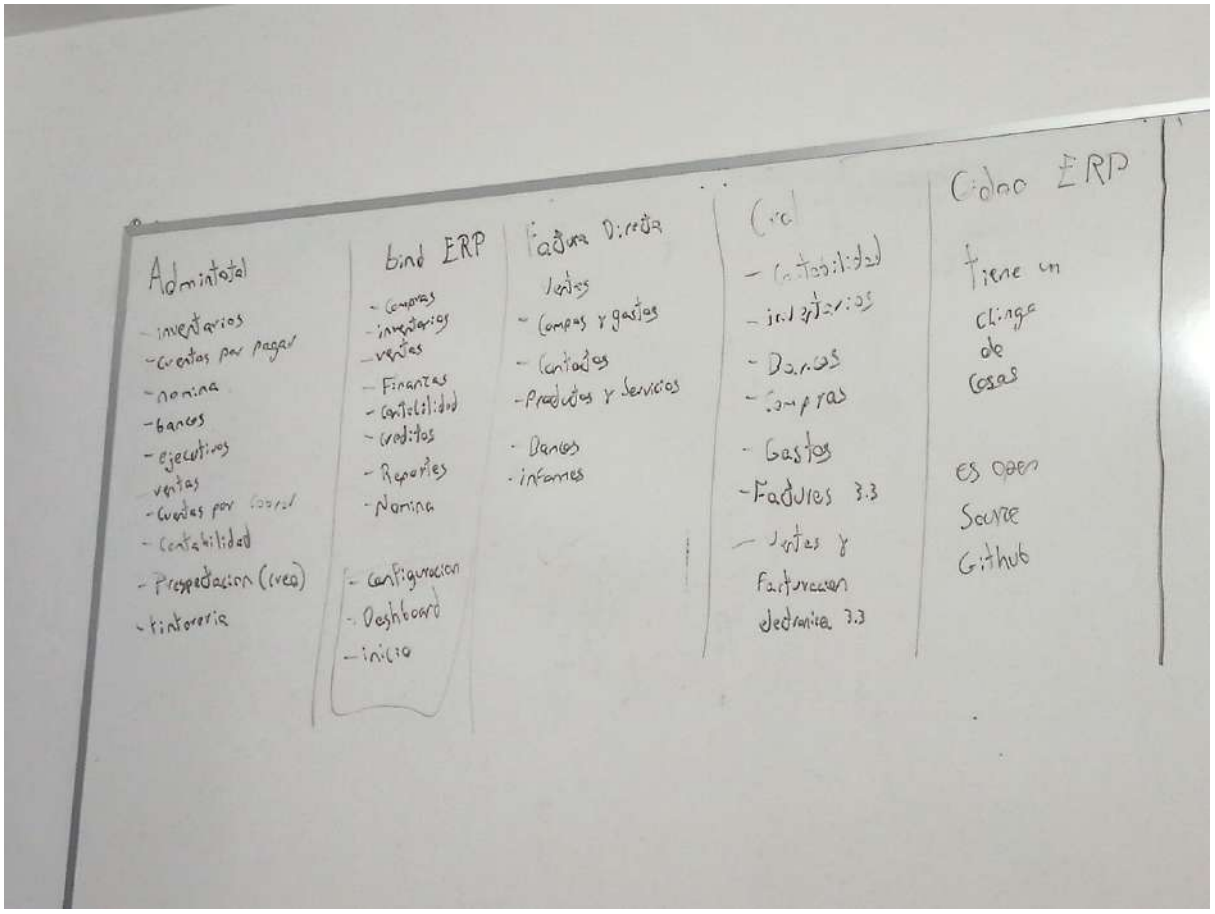
https://www.tutorialspoint.com/software_architecture_design/component_based_architecture.htm

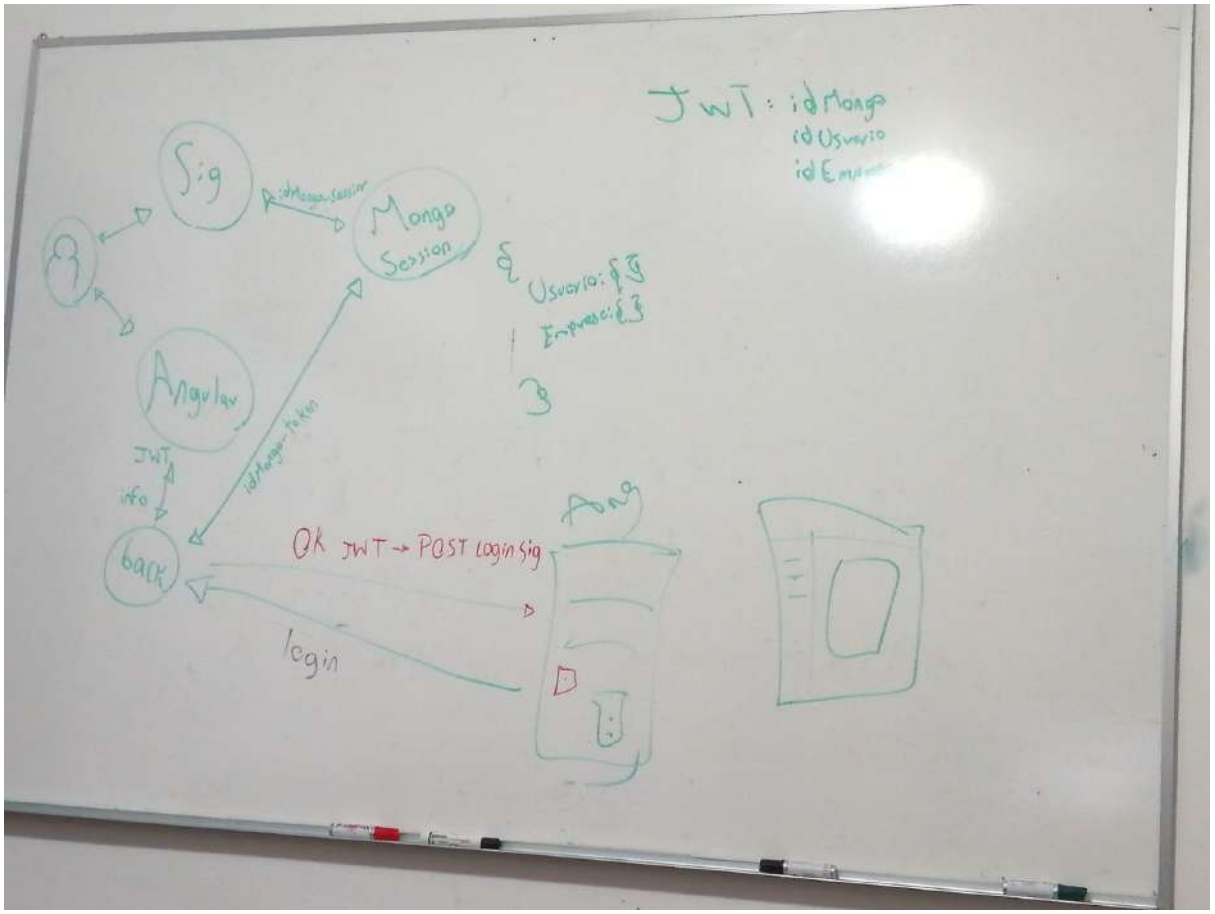
https://medium.com/@lazlojuly/what-is-a-restful-api-fabb8dc2afeb

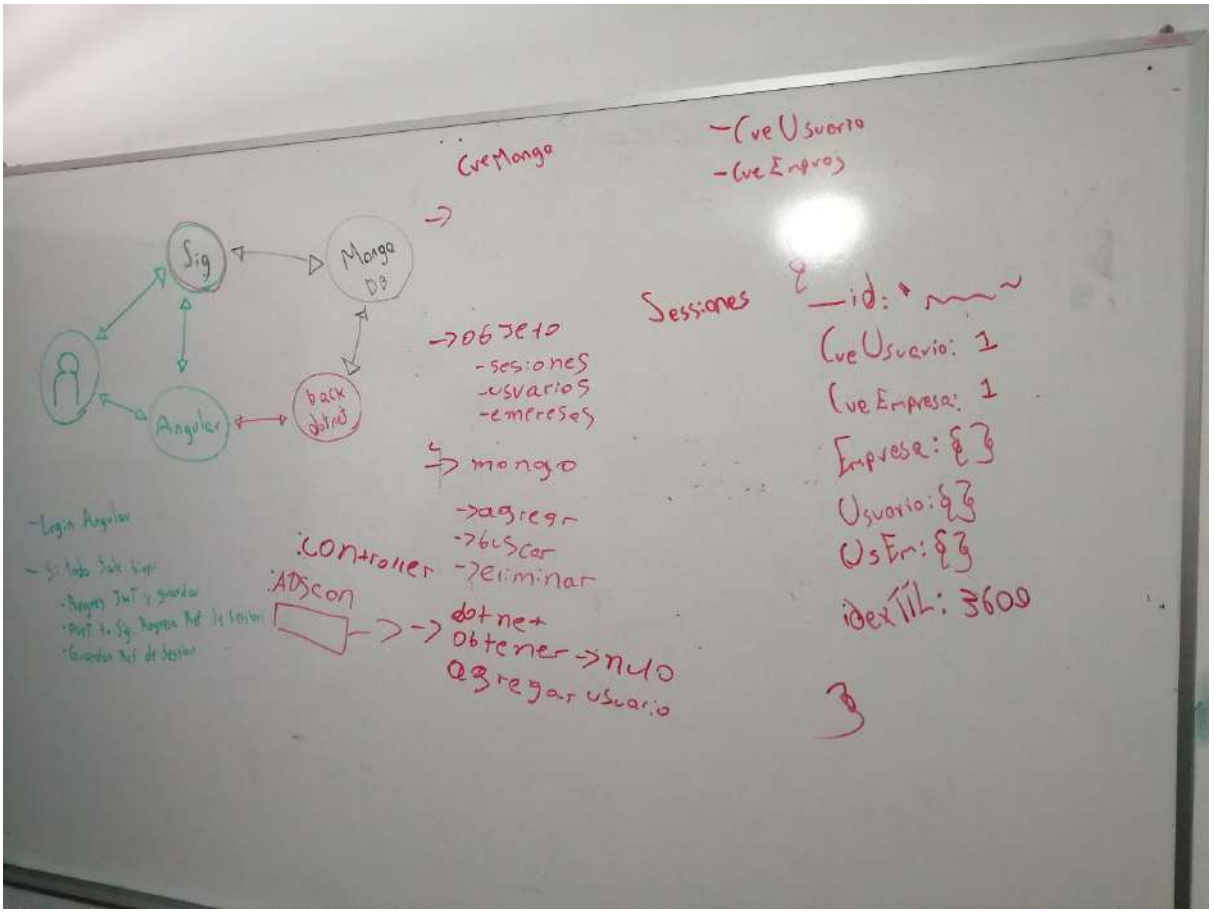
Anexos

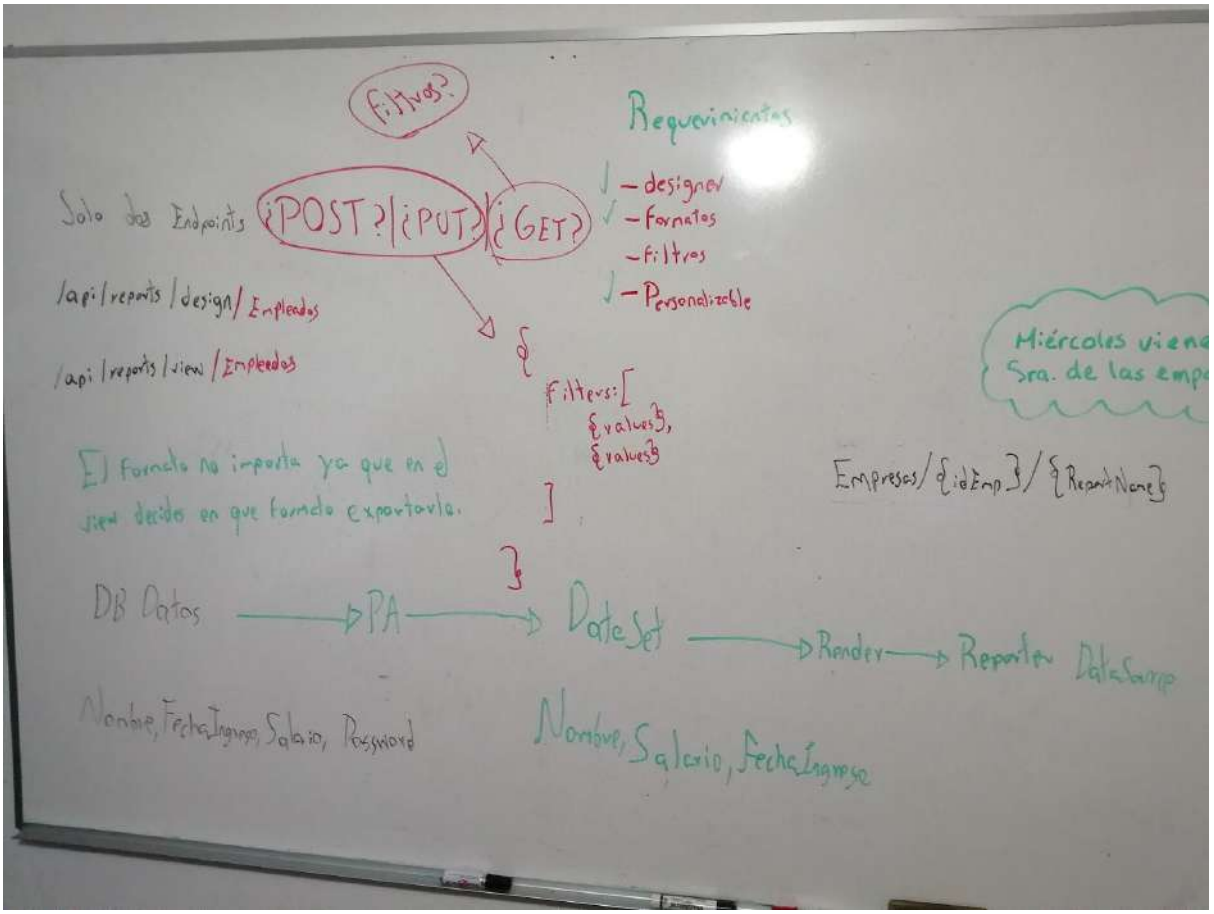
Los siguientes anexos son los diagramas realizados al momento de desarrollar el sistema tanto backend como frontend.

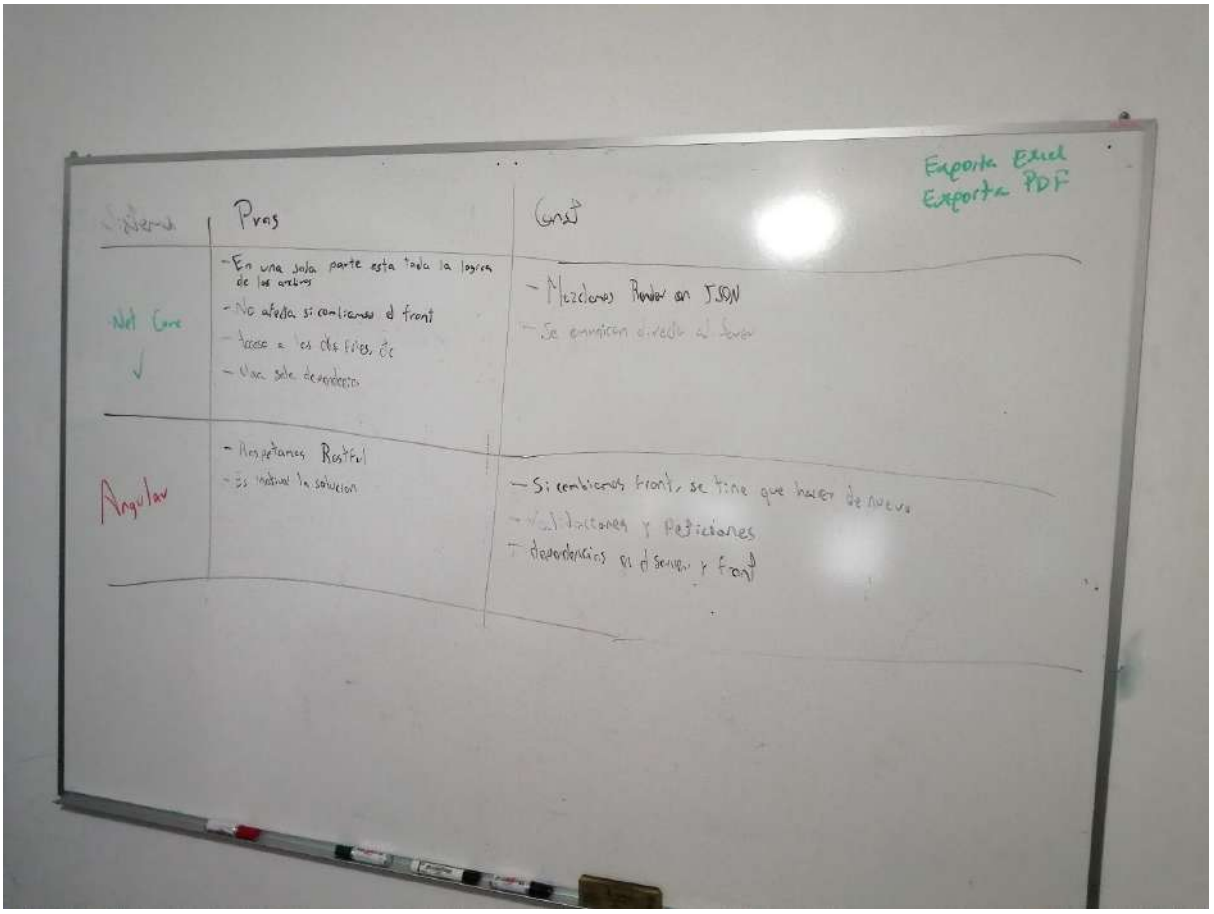


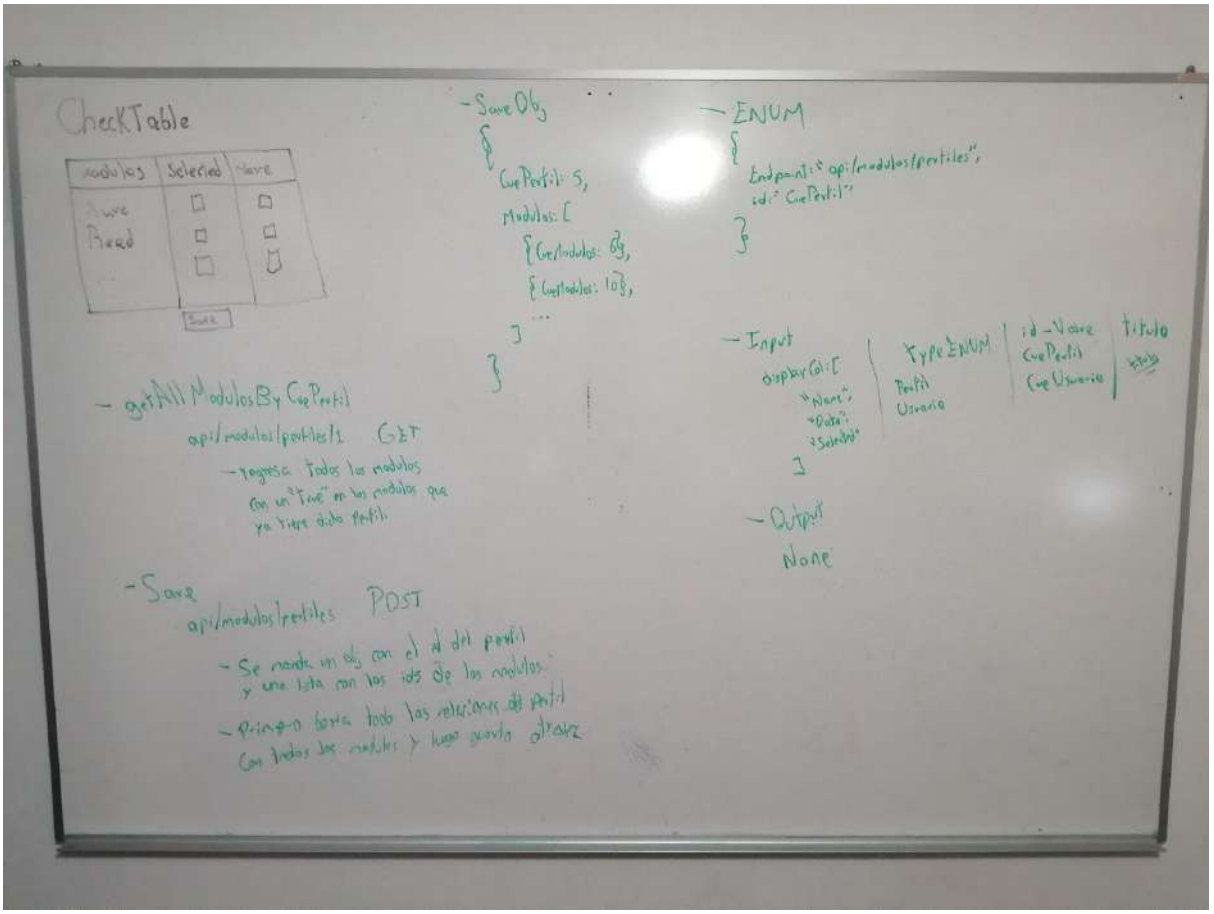

Administral
- inventarios
- cuentas por pagar
- nomina
- bancos
- ejecutivos ventas
- cuentas por cobrar
- contabilidad
- Preparacion (crea)
- tintoreria
bind ERP
- compras
- inventarios
- ventas
- Finanzas
- contabilidad
- creditos
- Reportes
- Nomina
- configuracion
- Dashboard
- inicio
factura Directa
Ventas
- Compras y gastos
- Contactos
- Productos y Servicios
- Bancos
- Informes
Cfdi
- Contabilidad
- Inventarios
- Bancos
- Compras
- Gastos
- Facturas 3.3
- Ventas y Facturacion declarada 3.3
Codoo ERP
Tiene un chingo de cosas
es open Source Github

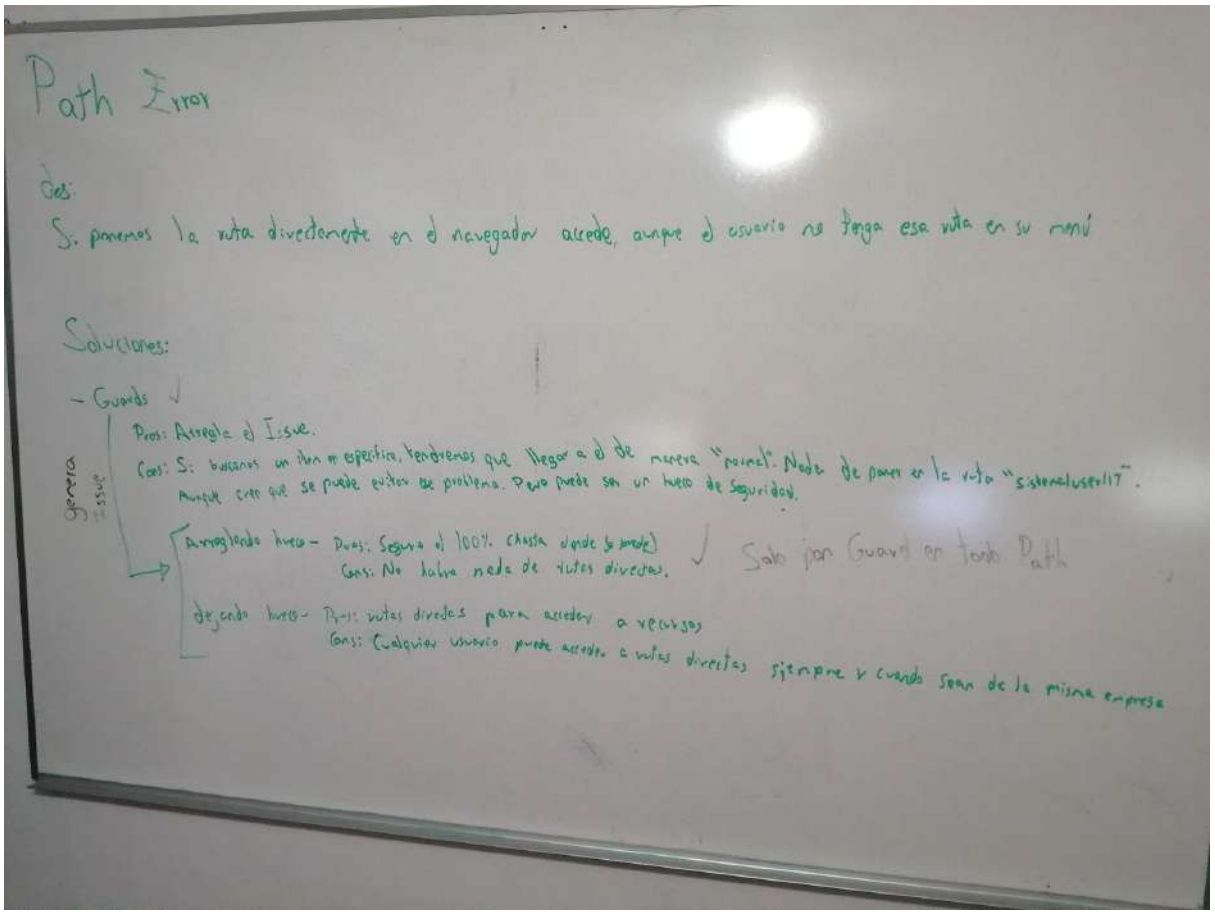

JWT: id Mongo
id Usuario
id Empresa
Sig
Mongo Session
Angular
JWT
info
back
id Mongo Token
OK JWT → POST Login Sig
login
Ang

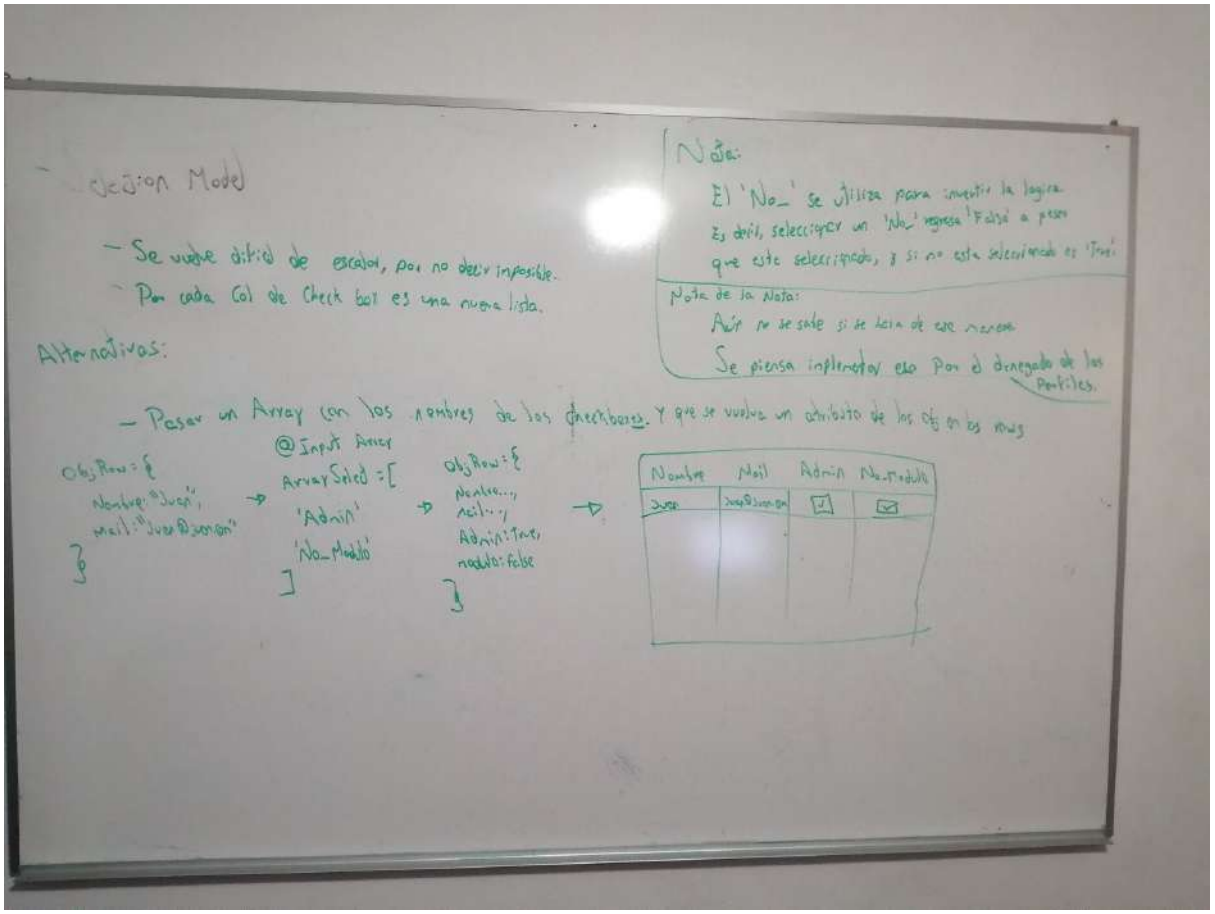

Sig
Mongo DB
Angular
back datos
CveMongo
-CveUsuario
-CveEmpres
->Objeto
-sesiones
-usuarios
-empresas
-> mongo
->agregar
->buscar
->eliminar
.controller
ADScon
->-> dotnet
Obtener -> nulo
Agregar usuario
Sessiones
_id:
CveUsuario: 1
CveEmpresa: 1
Empresa: {}
Usuario: {}
UsEm: {}
idexTIL: 3600

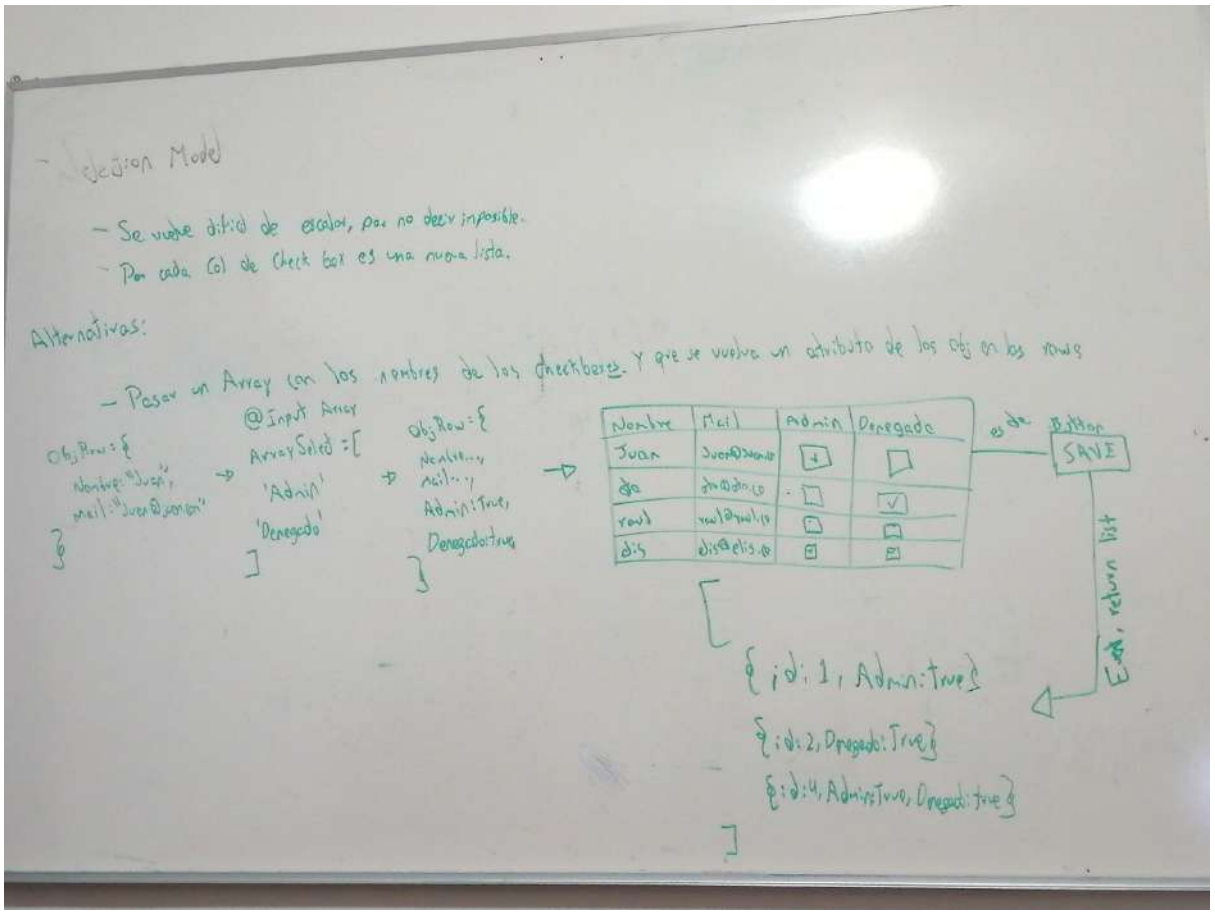

Filtros?
Requerimientos
Solo dos Endpoints ¿POST? ¿PUT? ¿GET?
- designer
- formatos
- filtros
- Personalizable
/api/reports/design/Empleados
/api/reports/view/Empleados
filters: [
{values},
{values}
]
El formato no importa ya que en el view decido en que formato exportarlo.
Miércoles viene Sra. de las empa
Empresas/{idEmp}/{ReportName}
DB Datos → PA → DataSet → Render → Reporte DataSamp
Nombre, FechaIngreso, Salario, Password
Nombre, Salario, FechaIngreso

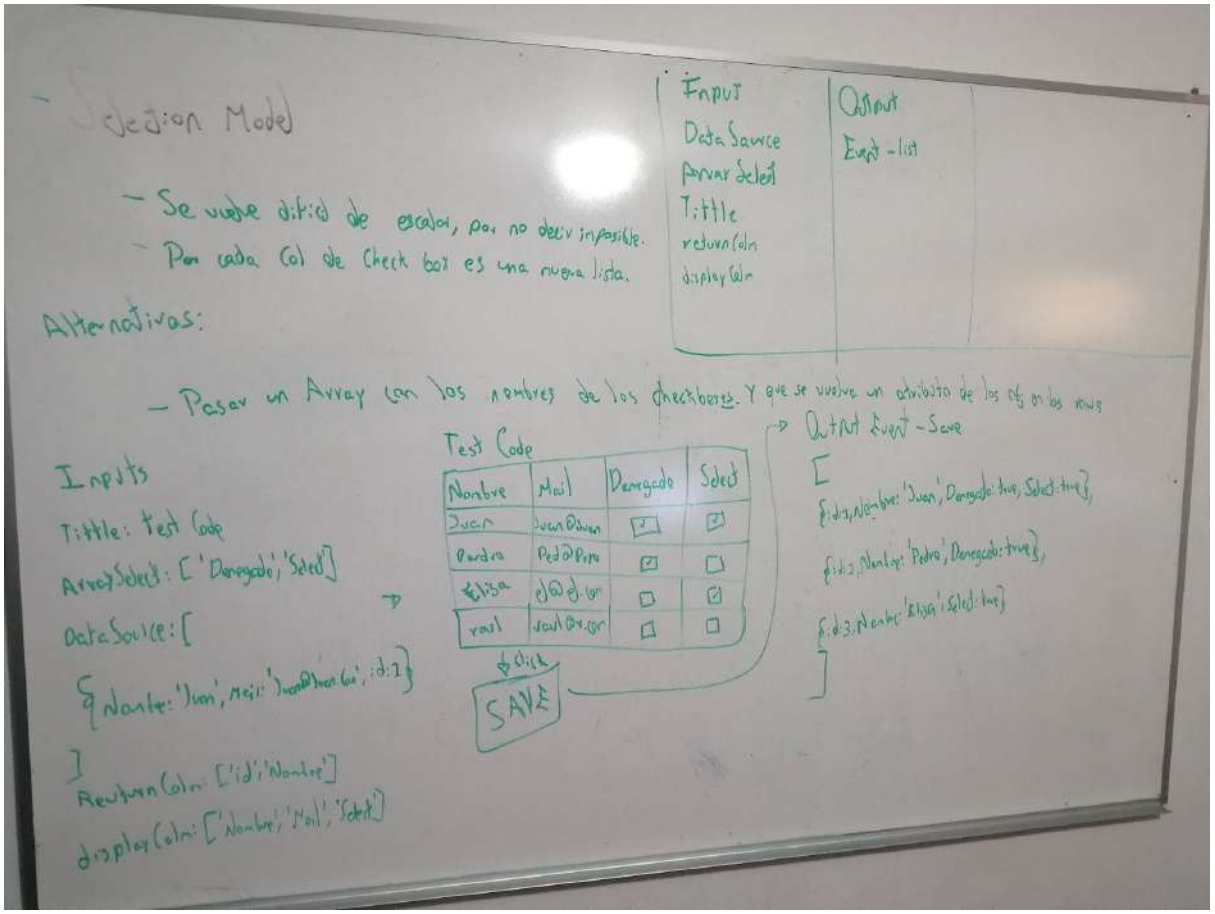

Exporta Excel
Exporta PDF
Pros
Cons
Sistema
Net Core
Angular
- En una sola parte esta toda la logica de los archivos
- No afecta si cambiamos el front
- Acceso a los cls files, etc
- Usa solo dependencias
- Respetamos RestFul
- Es intuitivo la solucion
- Mezclamos Razor con JSON
- Se envian directo al Server
- Si cambiamos front, se tiene que hacer de nuevo
- Validaciones y Peticiones
- dependencias en el server y front

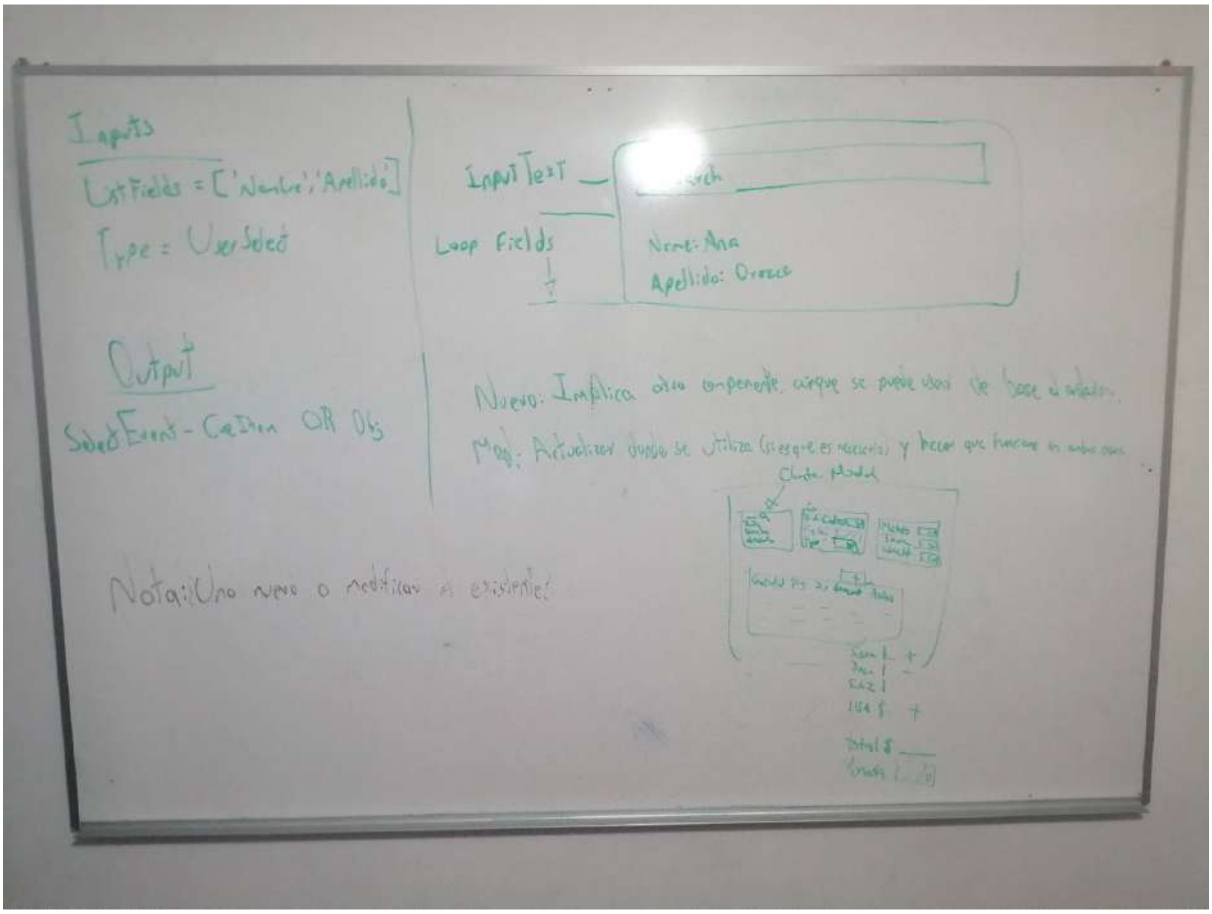

CheckTable
Modulos
Selected
Save
Read
Save
- SaveObj
{
CvePerfil: 5,
Modulos: [
{CveModulos: 69,
{CveModulos: 10},
...
]
}
- ENUM
{
Endpoint: "api/modulos/perfiles",
id: "CvePerfil"
}
- Input
displayCol: [
"Name",
"Data",
"Selected"
]
TypeENUM
Perfil
Usuario
id - Name
CvePerfil
CveUsuario
Titulo
- Output
None
- getAllModulosByCvePerfil
api/modulos/perfiles/1 GET
- regresa todos los modulos con un "True" en los modulos que ya tiene dicho perfil
- Save
api/modulos/perfiles POST
- Se manda un obj con el id del perfil y una lista con los ids de los modulos
- Primero borra todo las relaciones del perfil con todos los modulos y luego guarda otra vez

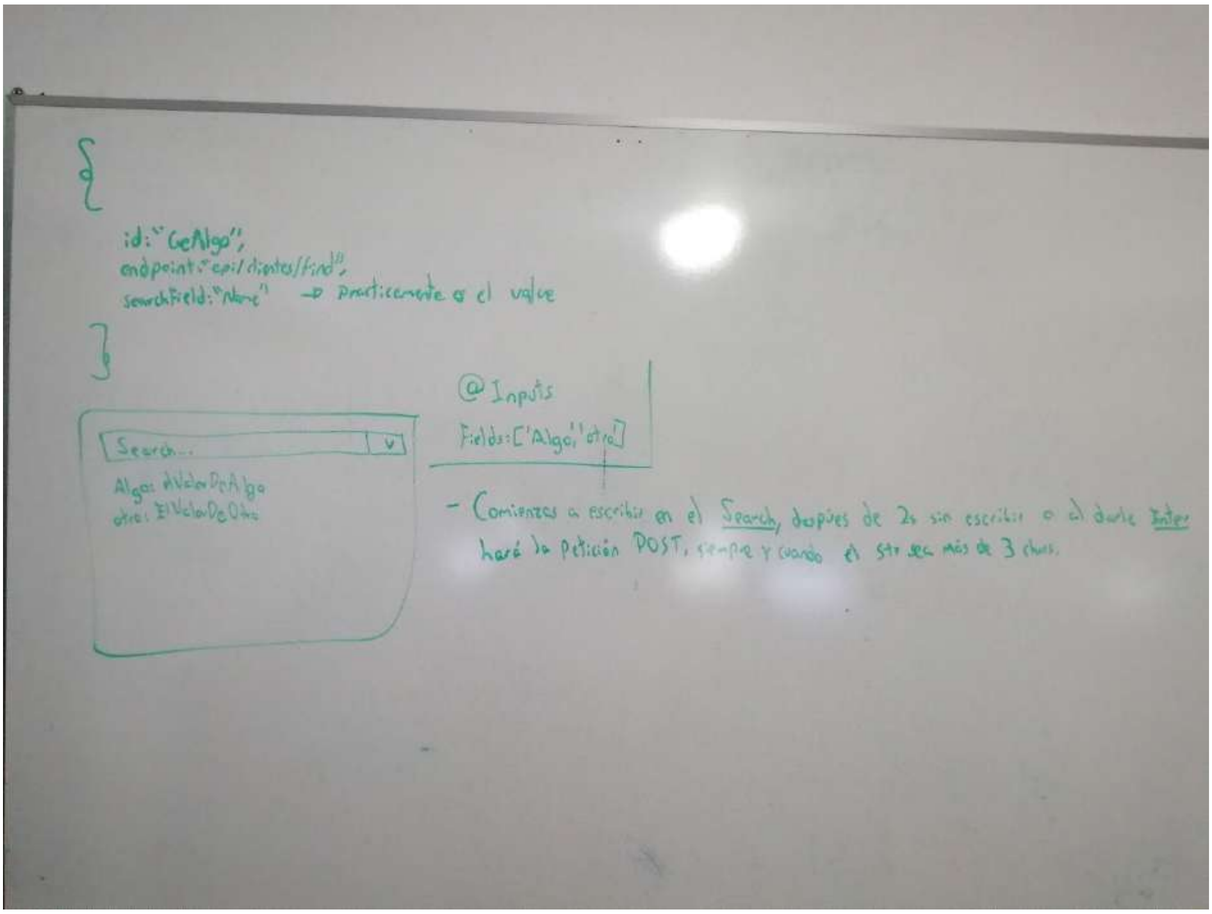

Path Error
Si ponemos la ruta directamente en el navegador accede, aunque el usuario no tenga esa ruta en su menú
Soluciones:
- Guards ✓
genera Issue
Pros: Arregla el Issue.
Cons: Si buscamos un ban específico, tendremos que llegar a él de manera "normal". Nada de poner en la ruta "sistema/usuario11".
Aunque creo que se puede evitar de problema. Pero puede ser un hueco de seguridad.
Arreglando hueco - Pros: Seguro al 100% (hasta donde se puede) ✓ Solo por Guard en todo Path
Cons: No habría nada de rutas directas.
dejando hueco - Pros: rutas directas para acceder a recursos
Cons: Cualquier usuario puede acceder a rutas directas siempre y cuando sean de la misma empresa

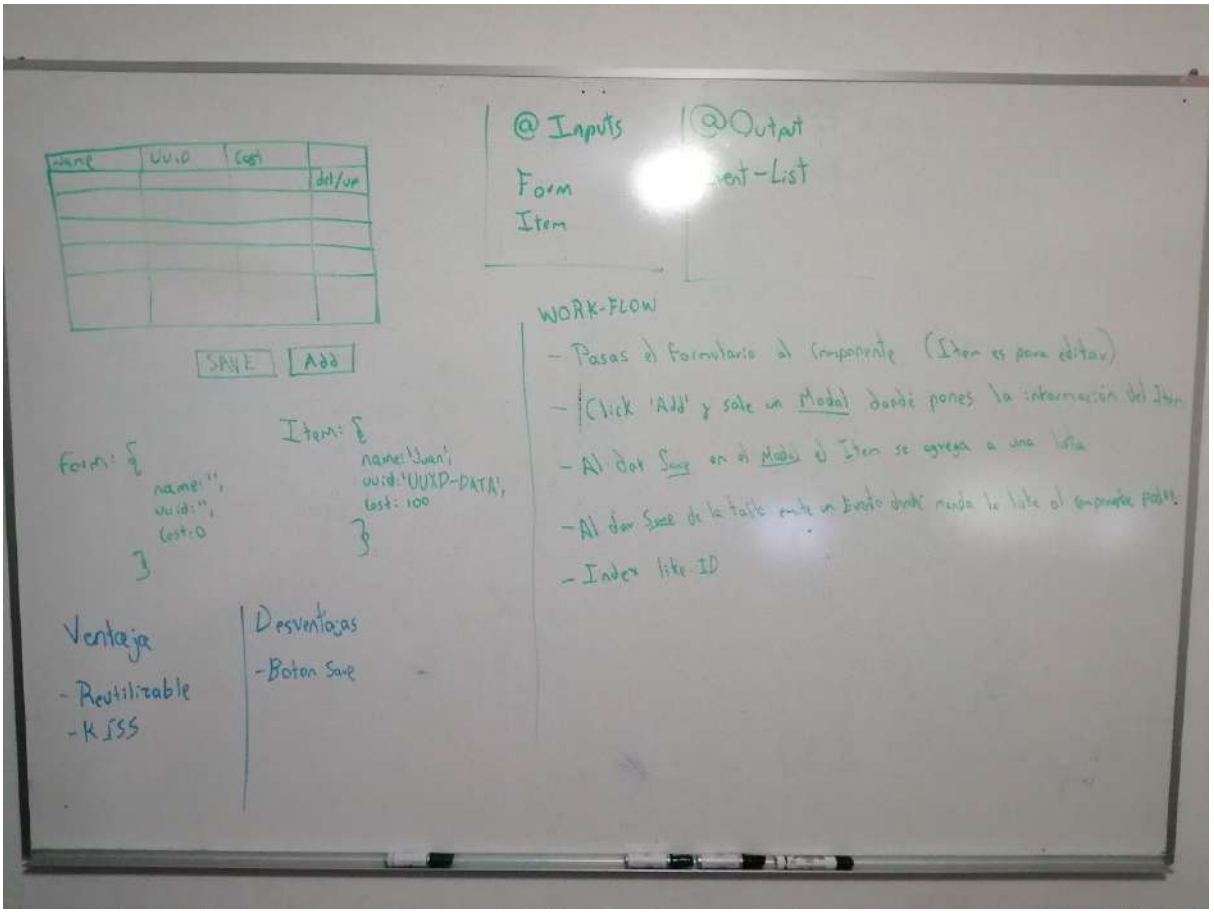

Decision Model
- Se vuelve difícil de escalar, por no decir imposible.
- Por cada Col de Check box es una nueva lista.
Alternativas:
- Pasar un Array con los nombres de los checkboxes. Y que se vuelva un atributo de los obj en los rows
ObjRow: { Nombre: "Juan", mail: "juan@juan.com" }
@Input Array
ArraySelect: ['Admin' 'No_Modulo']
ObjRow: { Nombre..., mail..., Admin: true, noModulo: false }
Nombre | Mail | Admin | No_Modulo
Juan | juan@juan.com
Nota:
El 'No_' se utiliza para invertir la lógica.
Es decir, seleccionar un 'No_' regresa 'False' a pesar que este seleccionado, y si no está seleccionado es 'True'.
Nota de la Nota:
Aún no se sabe si se hará de este manera
Se piensa implementar esto Por el desarrollo de los Perfiles.

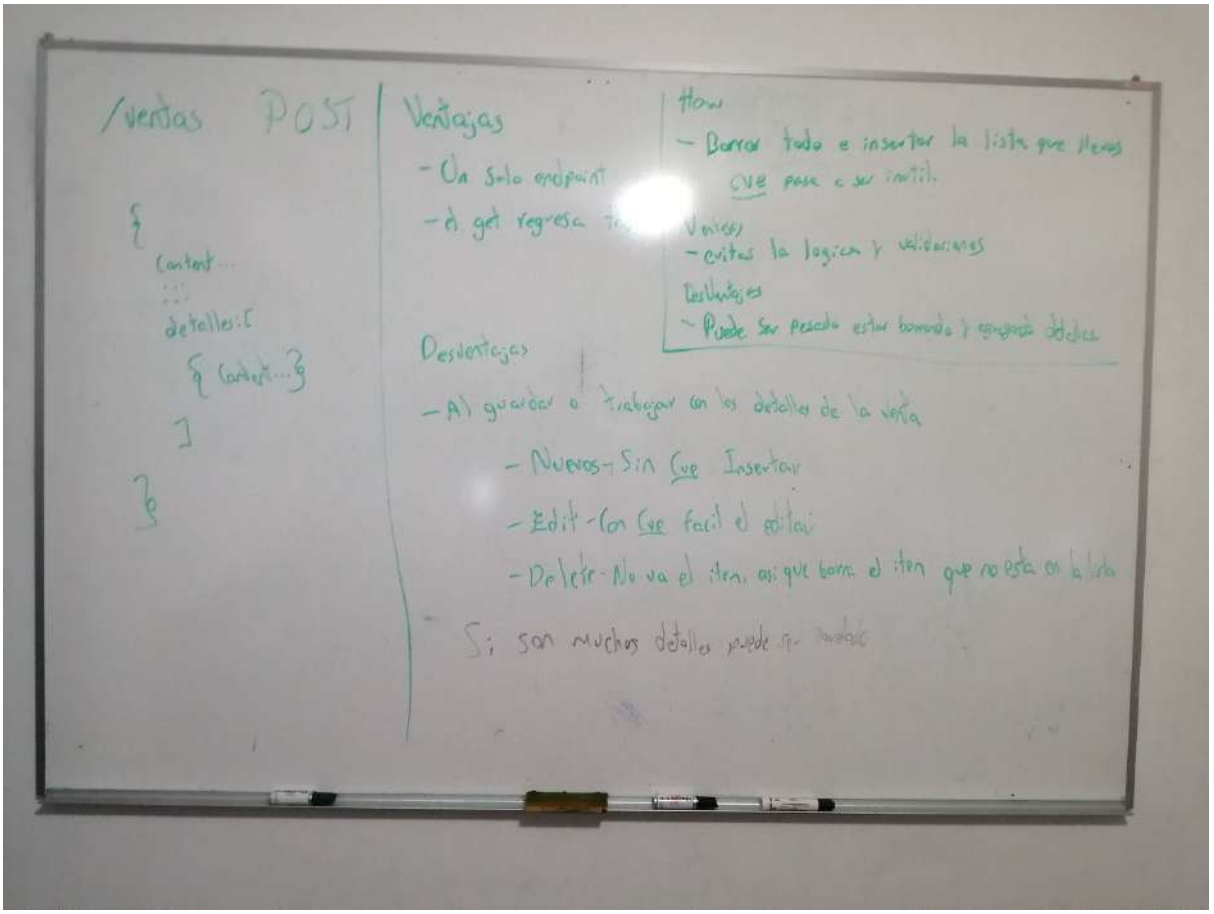

- Selection Model
- Se vuelve difícil de escalar, por no decir imposible.
- Por cada Col de Check box es una nueva lista.
Alternativas:
- Pasar un Array con los nombres de los checkboxes. Y que se vuelva un atributo de los obj en los rows
ObjRow: {
Nombre: "Juan",
mail: "juan@juan.com"
}
@Input Array
ArraySelect: [
'Admin'
'Denegado'
]
ObjRow: {
Nombre...,
mail...,
Admin: True,
Denegado: true
}
Nombre
Mail
Admin
Denegado
Juan
dja
raul
dis
Button
SAVE
Exit, return list
[
{id: 1, Admin: true}
{id: 2, Denegado: True}
{id: 4, Admin: True, Denegado: true}
]

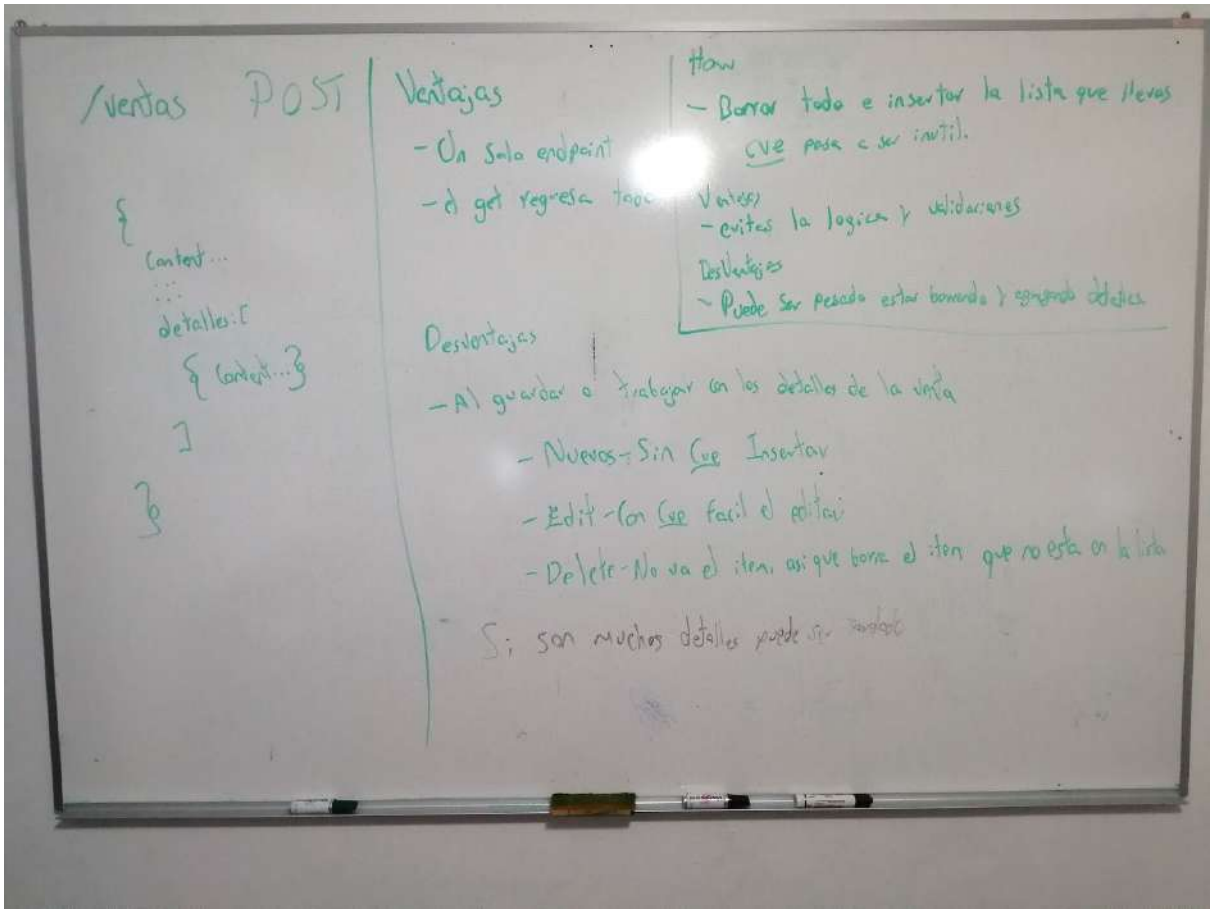

- Selection Model
- Se vuelve difícil de escalar, por no decir imposible.
- Por cada Col de Check box es una nueva lista.
Input
DataSource
ArraySelect
Tittle
returnCols
displayCols
Output
Event - list
Alternativas:
- Pasar un Array con los nombres de los checkboxes. Y que se vuelva un atributo de los obj en los rows
Inputs
Tittle: Test Code
ArraySelect: ['Denegado', 'Select']
DataSource: [
{Nombre: 'Juan', Mail: 'JuanDJuan.Co', id: 2}
]
ReturnCols: ['id', 'Nombre']
displayCols: ['Nombre', 'Mail', 'Select']
Test Code
Nombre
Mail
Denegado
Select
Juan
Pedro
Elisa
raul
Click
SAVE
Output Event - Save
[
{id: 3, Nombre: 'Juan', Denegado: true, Select: true}
{id: 2, Nombre: 'Pedro', Denegado: true}
{id: 3, Nombre: 'Elisa', Select: true}
]

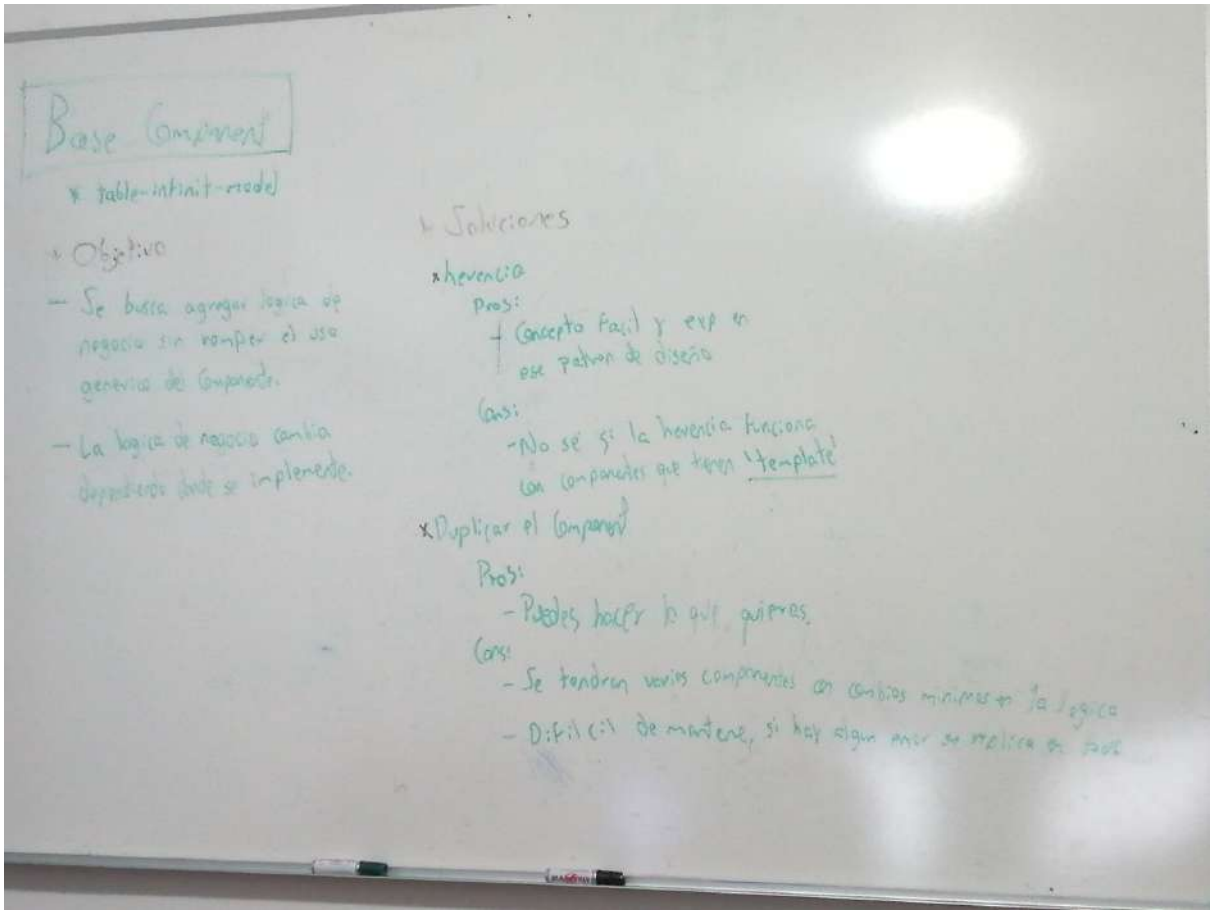

Inputs
ListFields = ['Nombre','Apellido']
Type = UserSelect
Output
Input Text
Loop Fields
Nombre: Ana
Apellido: Orozco
Nota: Uno nuevo o modificar el existente?

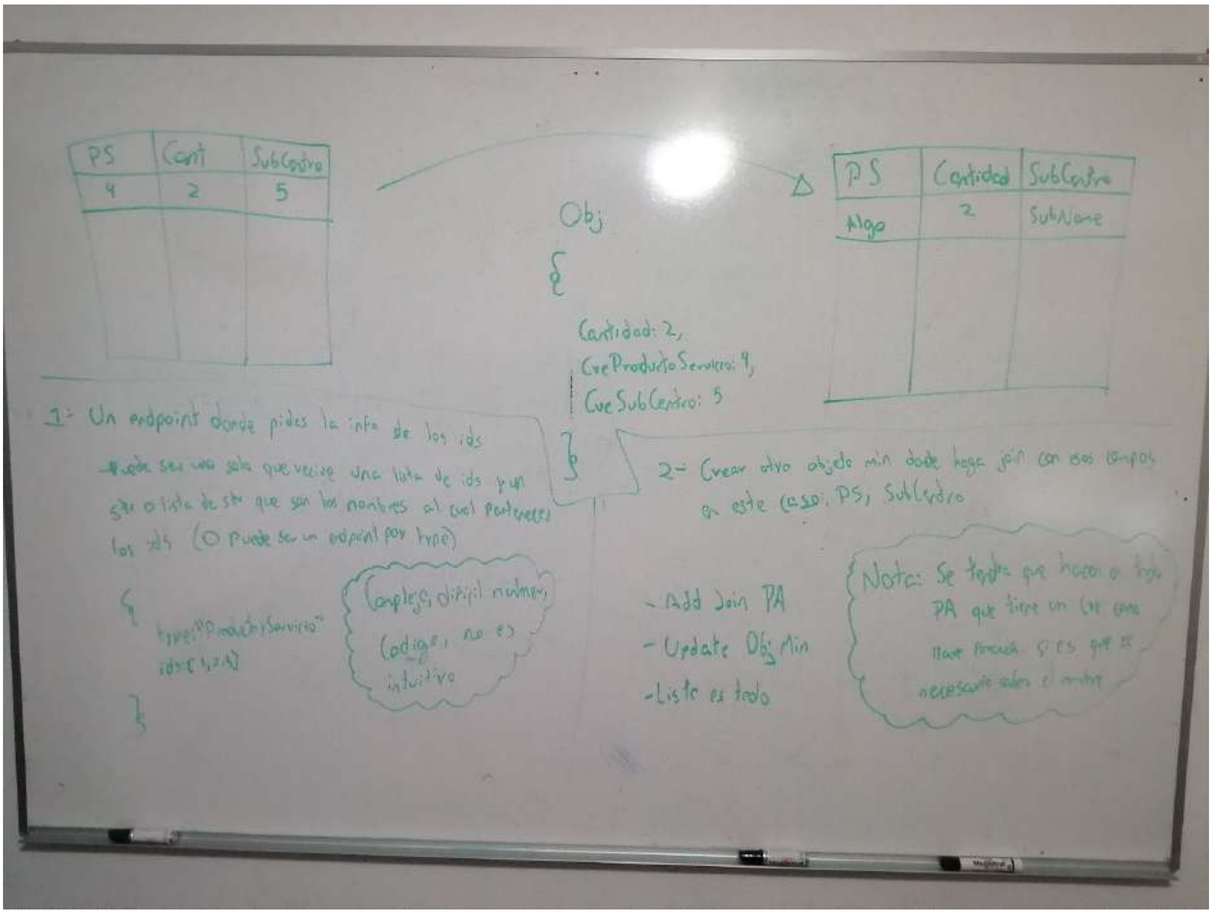

id: "GetAlgo",
endpoint: "api/clientes/find",
searchField: "Name" -> Practicamente a el value
@Inputs
Fields: ['Algo','otro']
Search...
Algo: ElValorDeAlgo
otro: ElValorDeOtro
- Comienzas a escribir en el Search, despues de 2s sin escribir o al darle Enter hará la petición POST, siempre y cuando el str sea más de 3 chars.


name
UUID
cost
del/up
SAVE
Add
@Inputs
Form
Item
@Output
ent-List
WORK-FLOW
- Pasas el formulario al componente (Item es para editar)
- Click 'Add' y sale un Modal donde pones la información del Item
- Al dar Save en el Modal el Item se agrega a una lista
- Al dar Save de la table emite un Evento donde manda la lista al componente padre.
- Index like ID
Form: {
name: '';
uuid: '';
cost: 0
}
Item: {
name: 'Juan';
uuid: 'UUID-DATA';
cost: 100
}
Ventaja
- Reutilizable
- KISS
Desventajas
- Boton Save

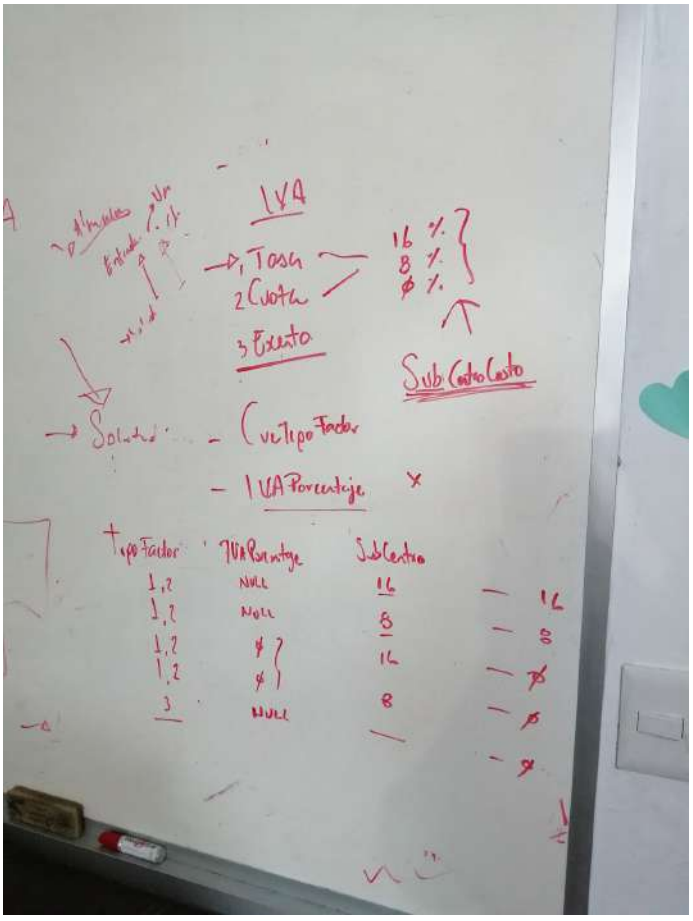

/ventas POST
{
content...
detalles: [
{ content... }
]
}
Ventajas
- Un solo endpoint
- El get regresa
Desventajas
- Al guardar o trabajar con los detalles de la venta
- Nuevos - Sin Cue Insertar
- Edit - Con Cue facil de editar
- Delete - No va el item, asi que borra el item que no esta en la lista
Si son muchos detalles puede
How
- Borrar todo e insertar la lista que llegas
Cue pasa a ser inutil.
Ventas)
- evitas la logica y validaciones
Desventajas
- Puede ser pesado estar borrando y agregando todos


/ventas POST
{
Content...
detalles: [
{ Content... }
]
}
Ventajas
- Un solo endpoint
- El get regresa todo
Desventajas
- Al guardar o trabajar con los detalles de la venta
- Nuevos - Sin Cve Insertar
- Edit - Con Cve fácil el editar
- Delete - No va el item, así que borre el item que no esta en la lista
Si son muchos detalles puede ser tardado
How
- Borrar todo e insertar la lista que llevas
Cve pasa a ser inutil.
Ventas:
- evitas la logica y validaciones
Desventajas
- Puede ser pesado estar borrando y agregando objetos


Base Component
* table-initinit-model
* Objetivo
- Se busca agregar logica de negocio sin romper el uso generico del Componente.
- La logica de negocio cambia dependiendo donde se implemente.
* Soluciones
*Herencia
Pros:
+ Concepto facil y exp en ese patron de diseño
Cons:
-No se si la herencia funciona con componentes que tienen template
*Duplicar el Componente
Pros:
- Puedes hacer lo que quieres.
Cons:
- Se tendrán varios componentes con cambios minimos en la logica
- Dificil de mantener, si hay algun error se replica en todos



PS | Cant | SubCentro
4 | 2 | 5
Obj
{
Cantidad: 2,
CveProductoServicio: 4,
CveSubCentro: 5
}
PS | Cantidad | SubCentro
Algo | 2 | SubName
1.- Un endpoint donde pides la info de los ids
2.- Crear otro objeto min donde haga join con esos campos
- Add Join PA
- Update Obj Min
- Listo es todo


IVA
1 Tasa
2 Cuota
3 Exento
16 %
8 %
0 %
Sub Costo Costo
Solicitud
- Cve Tipo Factor
- IVA Porcentaje
Tipo Factor
IVA Porcentaje
Sub Costo
1,2
1,2
1,2
1,2
3
NULL
NULL
16
8
16
8
16
8
0
0

Carta de Finalización

AD SISTEMAS
Análisis y Diseño de Sistemas

Hermosillo sonora a 13 de diciembre del 2019

Asunto: terminación de prácticas.

Por este presente se informa que el alumno **Joshua Nathanael Saucedo Uriarte** con número de expediente **215205720** termino satisfactoriamente sus prácticas profesionales en el periodo de **28 agosto a 1 diciembre de 2019,** participando en colaboración con el equipo de desarrollo de software en la elaboración del proyecto: **SIG Pymé**, cubriendo un total de 340 horas en el periodo mencionado anteriormente.

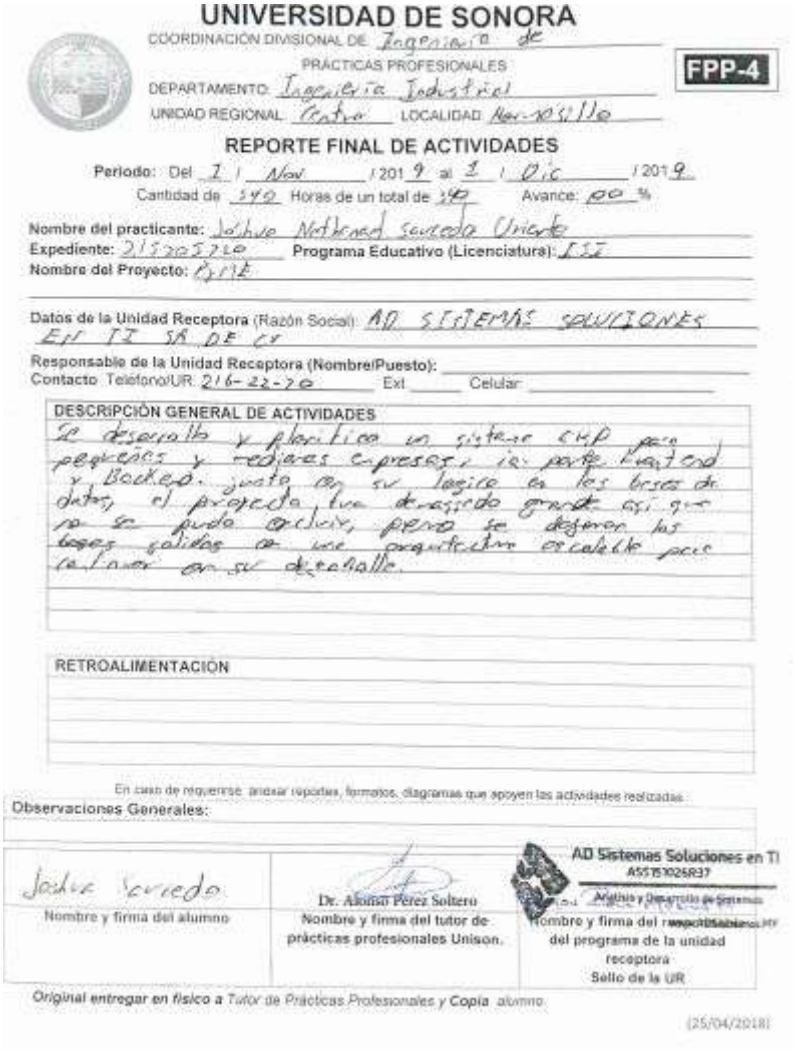
Se extiende la presente Carta de terminación de prácticas profesionales para los fines que el interesado convenga.

ATENTAMENTE

AD Sistemas Soluciones en TI
ASS151026R37
Analisis y Desarrollo de Sistemas
www.ADSismemas.MX

Ramón René Molina Figueroa
Gerente Operativo

Formato FPP-4



UNIVERSIDAD DE SONORA

COORDINACIÓN DIVISIONAL DE Ingeniería de
PRÁCTICAS PROFESIONALES

FPP-4

DEPARTAMENTO: Ingeniería Industrial
UNIDAD REGIONAL: Centro LOCALIDAD: Hermosillo

REPORTE FINAL DE ACTIVIDADES

Periodo: Del 1 / Nov / 2019 al 1 / Dic / 2019
Cantidad de 140 Horas de un total de 140 Avance: 100 %

Nombre del practicante: Joshua Nathaniel Saavedra Uriarte
Expediente: 215205210 Programa Educativo (Licenciatura): ISI
Nombre del Proyecto: Pixit

Datos de la Unidad Receptora (Razón Social): AD SISTEMAS SOLUCIONES EN IT SA DE CV
Responsable de la Unidad Receptora (Nombre/Puesto):
Contacto Teléfono/UR: 216-22-70 Ext Celular

DESCRIPCIÓN GENERAL DE ACTIVIDADES

Se desarrollo y plantico un sistema CMP para pequeñas y medianas empresas, la parte Front end y Backend, junto con su logica en las bases de datos, el proyecto fue demasiado grande asi que no se pudo acabar, pero se dejaron las bases solidas de una arquitectura escalable para seguir en su desarrollo.

RETROALIMENTACIÓN

En caso de requerirse anexar reportes, formatos, diagramas que apoyen las actividades realizadas

Observaciones Generales:

Nombre y firma del alumno	Dr. Alonso Perez Soltero Nombre y firma del tutor de prácticas profesionales Unison.	AD Sistemas Soluciones en TI ASS151026R37 Análisis y Desarrollo de Sistemas Nombre y firma del responsable del programa de la unidad receptora Sello de la UR

Original entregar en físico a Tutor de Prácticas Profesionales y Copia alumno

(25/04/2018)