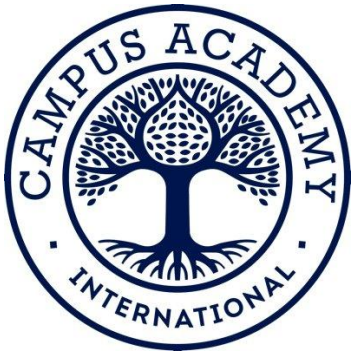
Jean-Marin MONNIER
Bachelor 1 / Switch IT
Campus Academy - Angers

La Graine Informatique / iD Systèmes
1 avenue du Bois l'Abbé - 49070 Beaucouzé

R A P P O R T D E S T A G E

-._@.

Stage du 31 mai au 27 août 2021
Maître de stage : Kévin DELARUE



SOMMAIRE

Présentation de l'entreprise

La Graine Informatique a été rachetée il y a 3 ans par iD Systèmes.
Basé à Bordeaux, Reims, Nîmes, Angers et Lyon, le **Groupe iD SYSTEMES** (150 collaborateurs), est devenu, en 25 ans, la référence en France des logiciels spécialisés pour les vins et spiritueux et a acquis dernièrement une position sur l'horticulture et la distribution de boissons. Avec plus de 17M€ de chiffres d'affaires et plus de 2000 clients, le **Groupe iD SYSTEMES** commercialise une gamme logicielle et des services métiers couvrant l'ensemble des besoins des clients : de l'amont (culture, vendanges, œnologie, production horticole), aux outils de gestion et de commerce (solutions de gestion métier, ERP généraliste, solutions mobiles de commerce, plateforme e-commerce), en passant par les solutions règlementaires (comptabilité matières, déclaration aux douanes) et des solutions de Data & Analytics (plateforme BI & Analytics, solutions de reporting, base de données métiers et support d'impressions spécialisés).

Présentation du projet

C'est à Angers que j'ai eu l'opportunité d'effectuer mon stage pratique de 1ère année.
J'ai travaillé sur le **projet NextGC** qui se compose de 2 applications qui sont en co-construction depuis 5 ans (**iDViniFlow** et **iDHortiFlow**).
Il s'agit d'un projet qui développe une application web de type SAAS (Software As A Service) pour les domaines de la viticulture et de l'horticulture.
Le but étant de proposer une solution pour permettre aux professionnels de ces métiers de pouvoir gérer leurs stocks, réaliser des ventes/commandes, avoir un suivi statistique, etc…
En bref, c'est un outil qui offre un accompagnement et une aide très poussés afin de faciliter la vie des professionnels.

L’équipe de développement

Sur ce projet nous sommes 9. Nous sollicitons une équipe de sous-traitants située en Tunisie (travaillant pour la société Nerium Software). Cette équipe qui constitue un renfort de 5 personnes est composée de : Skander, Amine, Abdessalem, Sahar et Hamdi.
À Angers, nous sommes 4 : Émilie et moi-même, ainsi que 2 chefs de projet : Kévin et Mahdi.
Mais nous travaillons également en lien avec Mathieu (sur Angers) qui s’occupe de la partie UX UI et aussi Gaël (sur Bordeaux), qui en spécialiste des métiers, connaît bien les besoins et les attentes des clients.

L’environnement de développement

Le projet se divise en 3 sous-projets : Le Front-end, le Back-end et l’Update. Tous les développeurs du projet travaillent aussi bien côté Back-end que côté Front-end et sur l’Update.

Pour le Front nous utilisons Angular comme Framework avec DevExtreme, une suite de composants JavaScript et HTML5 (langages qui proposent de nombreux composants UI), simplifiant le développement.

Pour le Back nous utilisons du C Sharp (C#).

L’Update est lui aussi en C#. Dans celui-ci, on va créer des migrations pour modifier la base de données qui est en SQL.

Nous travaillons en local mais il existe un Url sur lequel l’application est hébergée. Cet environnement s’appelle « **preprod** ».

Les logiciels et applications utilisés

Comme logiciel pour le développement nous utilisons :

- **Visual Studio Code** pour le Front.
- **Visual Studio** pour le Back et l'Update.
- **Microsoft SQL Server Management Studio** pour la base de données.
- **Redis Server** pour faire communiquer les différents services du Back.
- **Git** pour apporter nos modifications au code.
- **Postman** pour tester le Back sans appel du Front.

Comme logiciel de communication :

- **Microsoft Teams** qui est un outil essentiel pour l'échange d'informations, pour s'entraider,…

Comme sites :

- **Zoho** pour l'organisation du travail à faire.
- **Figma** pour récupérer les maquettes des pages pour reproduire les styles.
- **TFS** pour voir le code modifié.

L'affectation du travail

Régulièrement nous organisons des réunions sur Teams, dans lesquelles nous faisons un tour de table pour évoquer le travail en cours, l'état d'avancement des tâches et les problèmes rencontrés. En fin de réunion, les chefs de projets nous affectent différentes missions qui seront à réaliser dans la semaine.
Ces missions se divisent en 3 catégories :

- Les tâches
- Les bugs
- Les bugs freins

Les tâches concernent l'ajout de nouvelles fonctionnalités.
Les bugs concernent la résolution d'erreurs, la correction de fautes d'orthographes, l'amélioration de fonctionnalités qui dysfonctionnent.
Les bugs freins sont des bugs qu'il faut résoudre en priorité.

Le fonctionnement du Git

Le Git se divise en 2 branches principales : la « **Master** » et la « **Develop** ».

Lorsque l'on commence une nouvelle tâche, il faut créer une nouvelle branche qui sera basée sur Develop. Elle devra être appelée comme suit :
« feature/NGC-Tnuméro_tâche »
ex : NGC-T1060

Pour les bugs, on fait pareil mais le « feature » devient bug-fix et le T devient un I
ex : bug-fix/NGC-I1260

Pour les bugs freins, c'est différent. Il faut créer une branche basée sur la branche Master qui sera nommée hot-fix/NGC-Inuméro_tâche
ex : hot-fix/I1250

Il existe 3 projets Git : Front, Back et Update. Si ma mission me fait faire appel au Front et au Back je devrais créer une branche portant le même nom sur chacun de ces deux projets Git.

Les modifications apportées à la branche Develop sont ensuite envoyées, après vérifications, à la branche Master et inversement (voir fonctionnement Zoho plus loin).

Le fonctionnement de Postman

Postman est un logiciel très pratique qui nous permet d'appeler les méthodes que nous développons en Back et de les tester, sans même utiliser les appels du Front.
On y renseigne le type de la méthode (1), le path avec le port du micro service (2), le nom du web service (3), le nom de la méthode (4) et ses paramètres (5).
Voici un exemple :

Le fonctionnement de Zoho

Zoho est un très bon outil pour s'organiser dans le travail à faire. Il nous permet d'avoir un calendrier avec les différentes tâches ou bugs à traiter, selon les dates d'échéances fixées.

Voici son fonctionnement :

- Pour les tâches

Une fois créée, la tâche se retrouve dans un tableau divisé en 5 catégories qui correspondent à différents statuts d'avancement :

- "Ouvert"
- "En cours"
- "Cross Testing"
- "À tester"
- "Clôturé"

- **Ouvert** signifie que la tâche est créée mais que pour l'instant on ne travaille pas encore dessus.
- **En cours** signifie que l'on est actuellement en train de travailler dessus.
- **Cross Testing** signifie que l'on a terminé la tâche. À ce moment-là, une fois le code push sur la branche du git correspondant au nom de la tâche, un autre développeur va venir tester notre code. Il va s'assurer que la fonctionnalité correspond aux attentes décrites dans la tâche. Si le travail n'est pas conforme, le développeur crée un bug qui sera nommé sous la forme : « RCT-nom_tâche
  ex : RCT-NGC-T1060
  Dans ce bug, il décrit alors tous les problèmes et donne le plus d'informations possibles pour aider à la résolution.
- **À tester** signifie que le Cross Testing est fini et que tous les problèmes possiblement signalés ont été réglés. À ce stade, ce sont les chefs de projets qui reviennent de nouveau tester le code.
- **Clôturé** signifie que toutes les vérifications ont été faites sur le code mais qu'il est fusionné sur la branche Develop.

- Pour les bugs :

C'est comme pour les tâches,sauf qu'ici le Cross Testing n'existe pas. En effet, on passe directement de « En cours » à « À tester ».

- Pour les bugs freins :

C'est le même fonctionnement que pour les bugs, sauf qu'à la clôture, la fusion se fait avec la branche Master.

Le fonctionnement de TFS

TFS (Team Foundation Server) est un logiciel qui nous permet, une fois un bug traité ou une tâche terminée, de récupérer la branche du Git créée ainsi que les modifications apportées au code -afin de les comparer avec ce qu'il y avait avant. On appelle ça faire une « **pull request** » -ou requête de tirage en français. Lors d'un Cross Testing, cet outil s'avère très utile pour retrouver où le code a été modifié et pour, déjà, repérer de possibles erreurs.

Exemple de réalisation d'une tâche

J'ai une tâche affectée, imaginons la NGC-T1035.
Je commence à travailler dessus, donc je passe la tâche de « Ouvert » à « En cours ».
Je crée une branche sur Develop qui s'appelle « feature/NGC-T1035.
J'ai fini, de mon côté tout fonctionne.
Je passe la tâche de « En cours » à « Cross Testing » et je crée une pull request.
Si le développeur qui fait le Cross Testing n'a pas de retour à me faire, la tâche passe alors en « À tester ». Et une fois que le chef de projet a revérifié le code, il fusionne ce code sur Develop puis de Develop sur Master.

Ma plus grosse tâche réalisée

Semaine 6 Jour 3

Mahdi m'a donné à réaliser un web service pour les imprimantes (NGC-T1073).

Description de la tâche :

L'objectif est la création de gestion des imprimantes
Pour cela:
1/Back:
-Crée une nouvelle entity "model/Configuration/printers" avec les attributs designation(string) et internalName(string) qui hérite de BaseEntity2
-Crée printers.hbm.xml

2/Update
-Crée la migration pour "printers"

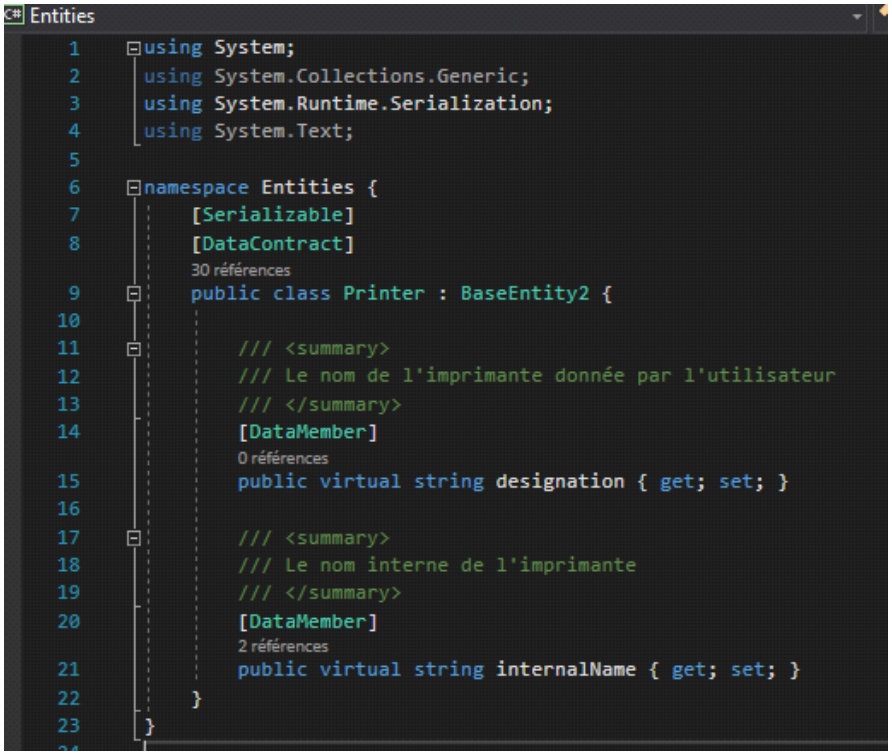
3/Back (CRUD): préparation des web services (MicroService Configuration) (voir EtiquetteBaseController)
-Api/Controller/PrinterController =>
    1-getAll: qui retourne tous les données du table printer
    2-post
    3-put
    4-delete
-Service
-Repository: doit hériter GenericRepo et implémenter l'interface

4/ Front
Dans les paramètres administrateur ,menu société :
Ajouter une entrée :
Titre : "Gestion des imprimantes"

Actions : Redirige vers un écran qui listes les imprimantes enregistrées permettant d'afficher la listes des imprimantes.

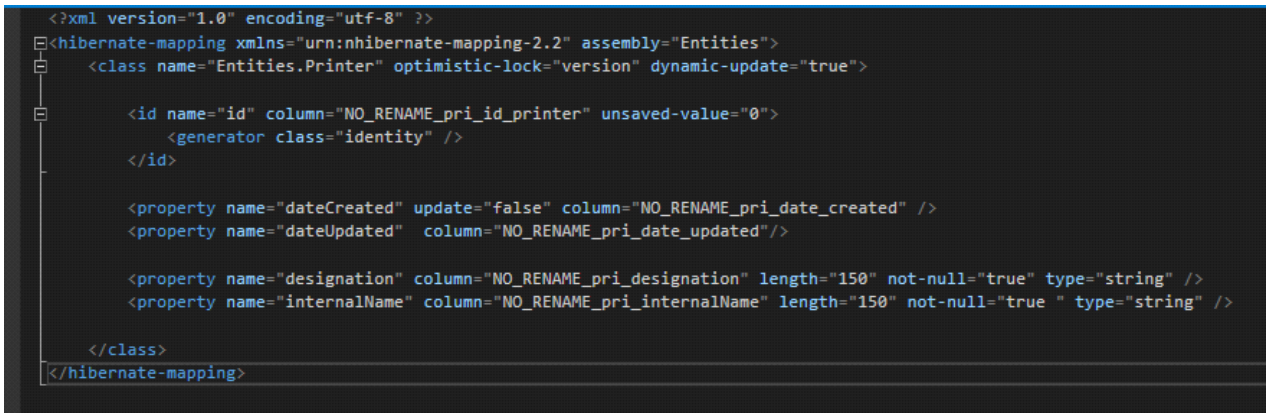
Créer une nouvelle entrée "Gestion des imprimantes".
Cette écran doit afficher la liste des imprimantes retournée par le back

- Création du fichier Printer.cs dans Entities et ajout des attributs :



Pour l’instant rien de difficile.

Il faut ensuite créer le fichier Printer.hbm.xml qui doit être mappé sur ce qu’il y aura en base de données c’est-à-dire un id en Primary Key, "designation", "internalName" ainsi que la date de création et de modification.



En s'inspirant de ce qui existe déjà, ce n'est encore pas trop difficile.

- Maintenant il faut insérer la table printer et ses colonnes en BDD. Pour cela il faut créer une migration dans le projet Update.

Création de la migration : _202107081150_CreateTablePrinter.

On met dedans :

- la classe qui hérite de "Migration"

- Un constructeur par défaut

- La requête SQL

```
[Migration(202107081150, TransactionBehavior.Default, "1.0.13")]
1 référence
public class _202107081150_CreateTablePrinter : Migration
{
    0 références
    public _202107081150_CreateTablePrinter()
    {
    }

    0 références
    public override void Up()
    {
        Execute.Sql(@"
            CREATE TABLE printer (
                pri_id_printer INT IDENTITY NOT NULL,
                pri_date_created DATETIME2 DEFAULT(getdate()) NULL,
                pri_date_updated DATETIME2 DEFAULT(getdate()) NULL,
                pri_designation NVARCHAR(150) NOT NULL,
                pri_internalName NVARCHAR(150) NOT NULL,
                CONSTRAINT PK_id_printer PRIMARY KEY (pri_id_printer)
            )
        ");
    }

    0 références
    public override void Down()
    {
    }
```

Je passe ensuite ma migration, sans erreur.

- Il faut maintenant créer les web services.

Pour ce faire, il me faut créer 3 fichiers :

- PrinterController.cs

- PrinterService.cs

- PrinterRepository.cs

Chacun de ces fichiers occupe un emplacement différent. Il existe 3 bibliothèques de classes par micro service :

- API qui contiendra le Controller
- Business qui contiendra le Service
- DAL qui contiendra le Repository

Dans le PrinterRepository, la classe hérite de "GenericRepository<Printer>" et de la future interface pour récupérer les CRUD. Dans le constructeur, on crée la connexion avec la BDD.

On crée l'interface IPrinterRepository qui hérite de "GenericRepository<Printer>".

```
namespace Configuration_DAL {
    3 références
    public class PrinterRepository : GenericRepository<Printer>, IPrinterRepository  {

        0 références
        public PrinterRepository(IHibernateConnexion hibernateConnexion, SharedLocalizer sharedLocalizer,
            INewConnectionRepository newConnectionRepository,
            LogBaseService logBaseService) : base(hibernateConnexion, sharedLocalizer, newConnectionRepository, logBaseService) {

        }
    }

    10 références
    public interface IPrinterRepository : IGenericRepository<Printer> {
    }
}
```

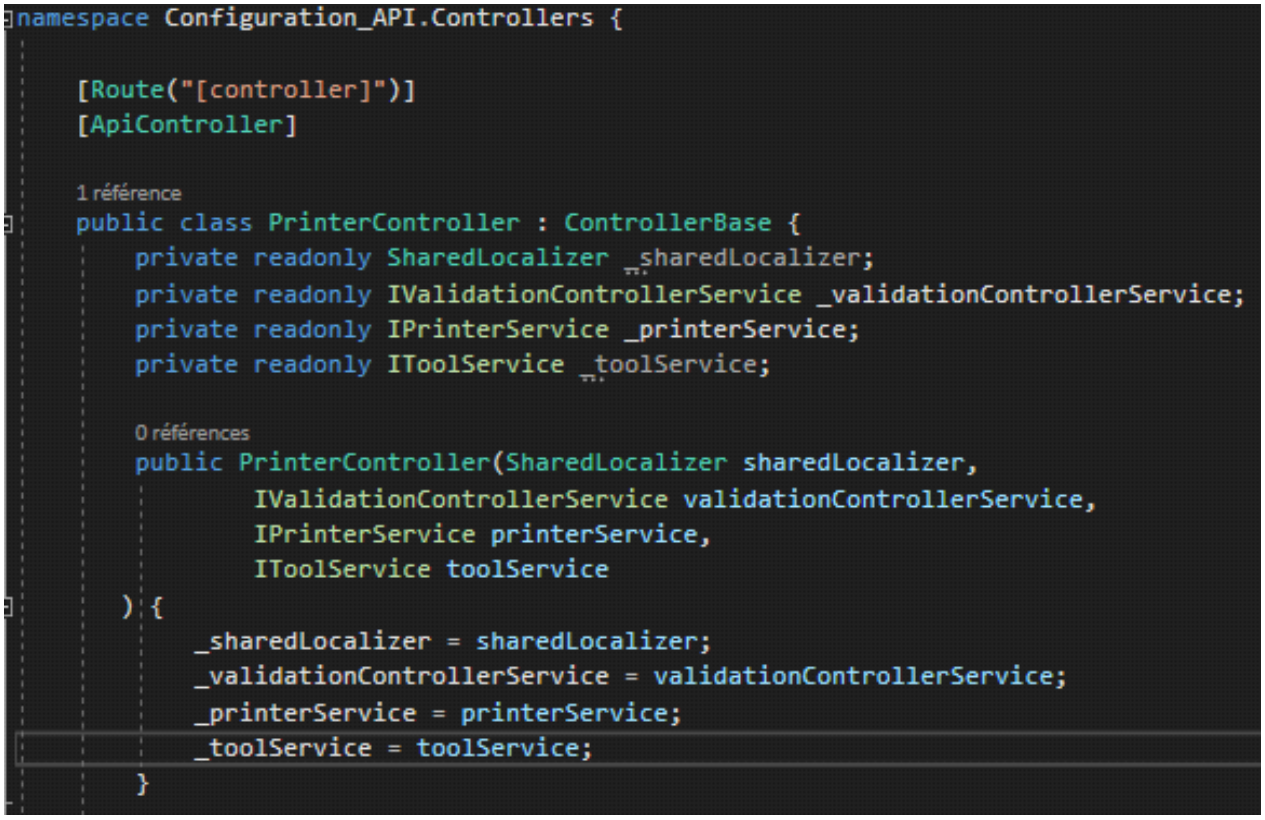
Dans le PrinterService, on fait appel au CRUD du IPrinterRepository tout en faisant des vérifications -pour jeter des exceptions, si besoin.

On crée l'interface IPrinterService qu'on implémente avec les méthodes du CRUD.

```
public class PrinterService  : IPrinterService{

    /// <summary>
    /// Gets the SharedLocalizer
    /// </summary>
    private readonly SharedLocalizer _sharedLocalizer;

    /// <summary>
    /// Gets the IPrinterRepository
    /// </summary>
    private readonly IPrinterRepository _printerRepository;

    private readonly INewConnectionRepository _newConnectionRepository;
    private readonly LogBaseService _logBaseService;
    private readonly IRedisUserManagementService _redisUserManagementService;
    /// <summary>
    /// PrinterService constructor
    /// </summary>
    /// <param name="sharedLocalizer"></param>
    0 références
    public PrinterService(SharedLocalizer sharedLocalizer,
        IPrinterRepository etiquetteBaseRepository,
        INewConnectionRepository newConnectionRepository,
        LogBaseService logBaseService,
        IRedisUserManagementService redisUserManagementService) {
        _sharedLocalizer = sharedLocalizer;
        _printerRepository = etiquetteBaseRepository;
        _newConnectionRepository = newConnectionRepository;
        _redisUserManagementService = redisUserManagementService;
        _logBaseService = logBaseService;
    }

    /// <summary>
    /// GetAll Printer
    /// </summary>
    /// <returns>ISet<EtiquetteBase></returns>
    2 références
    public async Task<ISet<Printer>> Get() {
        return await _printerRepository.Get<Printer>();

    }
```

```
    /// <summary>
    /// méthode appelé lors de la  creation d'une imprimante
    /// </summary>
    /// <param name="printer"></param>
    /// <returns>EtiquetteBase</returns>
    2 références
    public async Task<Printer> Post(Printer printer) {
        return await _printerRepository.Post(printer);
    }

    /// <summary>
    /// Modifier l'imprimante
    /// </summary>
    /// <param name="id"></param>
    /// <param name="printer"></param>
    /// <returns>EtiquetteBase</returns>
    2 références
    public async Task<Printer> Put(int id, Printer printer) {
        Printer oldPrinter = await _printerRepository.Get(id);

        if (oldPrinter.id != printer.id) {
            await _newConnectionRepository.InsertStatelessObject(await _logBaseService.GenerateLogBaseUpdateForUser_API(oldPrinter, printer, true));
            throw new MyException(_sharedLocalizer.GetLocalizedString("ChangedRecord"), System.Net.HttpStatusCode.Conflict);
        }
        printer = await _printerRepository.Put(printer);
        return printer;
    }

    /// <summary>
    /// Supprimer l'imprimante
    /// </summary>
    /// <param name="id"></param>
    2 références
    public async Task Delete(int id) {
        Printer printer = await _printerRepository.Get(id);
        if (printer == null) {
            throw new MyException(_sharedLocalizer.GetLocalizedString("UnknownObject"), System.Net.HttpStatusCode.NotFound);
        }
        await _printerRepository.Delete(printer);
    }
}
```

```
4 références
public interface IPrinterService {

    /// <summary>
    /// GetAll Printers
    /// </summary>
    /// <returns>ISet<Printer></returns>
    2 références
    Task<ISet<Printer>> Get();

    /// <summary>
    /// méthode appelé lors de la  creation d'une imprimante
    /// </summary>
    /// <param name="printer"></param>
    /// <returns>EtiquetteBase</returns>
    2 références
    Task<Printer> Post(Printer printer);

    /// <summary>
    /// Modifier l'imprimante
    /// </summary>
    /// <param name="id"></param>
    /// <param name="printer"></param>
    /// <returns>EtiquetteBase</returns>
    2 références
    Task<Printer> Put(int id, Printer printer);

    /// <summary>
    /// Supprimer l'imprimante
    /// </summary>
    /// <param name="id"></param>
    2 références
    Task Delete(int id);

}
```



```
namespace Configuration_API.Controllers {

    [Route("[controller]")]
    [ApiController]

    1 référence
    public class PrinterController : ControllerBase {
        private readonly SharedLocalizer _sharedLocalizer;
        private readonly IValidationControllerService _validationControllerService;
        private readonly IPrinterService _printerService;
        private readonly IToolService _toolService;

        0 références
        public PrinterController(SharedLocalizer sharedLocalizer,
                IValidationControllerService validationControllerService,
                IPrinterService printerService,
                IToolService toolService
        ) {
            _sharedLocalizer = sharedLocalizer;
            _validationControllerService = validationControllerService;
            _printerService = printerService;
            _toolService = toolService;
        }
```

Dans le PrinterController, on récupère le type d'appel et les infos contenues.
On fait appel au CRUD ciblé du PrinterService.

```
        /// Get All
        /// </summary>
        /// <returns></returns>
        [HttpGet]
        0 références
        public async Task<ISet<Printer>> Get() {
            return await _printerService.Get();
        }

        /// <summary>
        /// méthode appelé lors de la  creation d'une imprimante
        /// </summary>
        /// <param name="printer"></param>
        /// <returns>NGCResults</returns>
        [HttpPost]
        0 références
        public async Task<NGCResults<Printer>> Post([FromBody] Printer printer) {
            printer = await _printerService.Post(printer);
            NGCResults<Printer> result = new NGCResults<Printer>(printer);
            return result;

        }

        /// <summary>
        /// Modifier l'imprimante
        /// </summary>
        /// <param name="id"></param>
        /// <param name="printer"></param>
        /// <returns></returns>
        [HttpPut("{id}")]
        0 références
        public async Task<NGCResults<Printer>> Put(int id, [FromBody] Printer printer) {
            await _validationControllerService.ValidationPut(id, printer);
            printer = await _printerService.Put(id, printer);
            NGCResults<Printer> result = new NGCResults<Printer>(printer);
            return result;
        }

        /// <summary>
        /// Supprimer l'imprimante
        /// </summary>
        /// <param name="id"></param>
        [HttpDelete("{id}")]
        0 références
        public async Task<ActionResult> Delete(int id) {
            await _printerService.Delete(id);
            return Ok();
        }
    }
```

Le fonctionnement des web services est le suivant :

Le Controller récupère les infos et les communique au Service qui va les traiter/modifier, puis les communique au Repository qui, lui, va les envoyer en base de données.

Avant de partir, sur le Front, je crée le bouton qui servira à accéder à la page des Printers dans administration > Société > Imprimantes.

Semaine 6 Jour 4

Je fais un point avec Mahdi pour qu'il m'explique comment tester mon Back car je n'étais pas encore à l'aise sur Postman. Il m'aide aussi pour créer les components pour le Front car il y a une erreur.

- Il faut déjà ajouter PrinterService.cs dans le startup.cs
- Sur Postman, j'indique localhost:44310/Printer/2 pour le type Delete
- Pour la création de component, cela marchait il y a une semaine mais ça ne fonctionne plus. Mahdi n'en connaît pas la raison.
- On teste de Delete dans Postman mais erreur => "Objet inconnu"
    - C'est un problème en BDD donc ma migration est mal passée. J'en recrée une nouvelle et je supprime l'ancienne. Cette fois, c'est bon : elle est bien passée et ma table Printer apparaît dans ma base perso et admin. J'insère des Printer avec INSERT INTO dans ma table pour faire mes tests.

Retour à mon ordi, tout seul.

Je fais mes tests mais ça ne fonctionne pas. J'effectue des recherches sur différents sites et trouve qu'il me faut faire un changement dans les propriétés de Printer.hbm.xml : l'action de génération doit passer de « défaut » à « ressource incorporée ».

Ça fonctionne pour les méthodes Get, Post et Delete.

Le put ne fonctionne pas => DifferentID Error.

Après ma pause déjeuner, je laisse le Put de côté pour m'attaquer au Front.

Après des recherches, j'identifie le problème de la génération de component.

En fait, c'est à cause de la migration vers Angular 12, effectuée quelques jours avant. Dans Angular.json il me faut supprimer "styleext" : "scss".

Je génère donc mes 2 components : printer-grid et printer-detail. Comme les fichiers de styles sont en css, je les transforme en scss.

Je relie le printer-grid au bouton.

Je crée Printer.ts pour faire le model.

- Dedans
    - Je mets mes attributs :
        - designation : string
        - internalName : string

```
export class Printer extends BaseEntity2 {

    public designation : string;
    public internalName : string;

    constructor() {
        super();
        this.designation ='';
        this.internalName ='';
    }

    validator(): ValidationRulesByFieldName {
        return {
            designation: [
                { type: 'required', trim: true, message: 'FormErrorMessages.Required', useTranslator: true },
                { type: 'stringLength', max: 150, ignoreEmptyValue: false, message: 'FormErrorMessages.StringRangeLength', useTranslato
            ],
            internalName: [
                { type: 'required', trim: true, message: 'FormErrorMessages.Required', useTranslator: true },
                { type: 'stringLength', max: 150, ignoreEmptyValue: false, message: 'FormErrorMessages.StringRangeLength', useTranslato
            ],
        }
```

Je crée les règles de validations.

Je crée printer.service.ts

Je crée le lien vers le web service Printer dans app-configuration.service.ts

```
wsPrinter: this.serviceConfiguration.concat(`Printer/`)
```

Il y a tellement de choses à faire que je ne sais pas par où commencer et surtout je ne cible pas tout ce qui est nécessaire. Je regarde sur ce qui existe déjà, pour comprendre sur quoi agir.

Semaine 6 Jour 5

J'implémente le printer.service.ts avec les méthodes get, put, post, delete, clearListCache, navigateToDetail, navigateToList et navigateToCreate.

```
export class PrinterService  extends DaoGridStateBaseService implements IDaoBaseService {

    private _list: Observable<Printer[]> = null;
    currentItem: BehaviorSubject<Printer> = new BehaviorSubject<Printer>(null);
    private _gridState: Observable<JSON> = null;

    constructor(
        private http: HttpClient,
        private router: Router,
        private _translate: TranslateService,
        private appConfiguration: AppConfiguration
    ) {
        super(http, _translate,appConfiguration);
    }

    getAll(): Observable<Printer[]> {
        const url = this.appConfiguration.UrlsConfig.wsPrinter;
        if (!this._list) {
            this._list = this.http.get<any[]>(url)
                .pipe(
                    shareReplay(1)
                );
        }
        return this._list;
    }


    /**
    * Create an article
    * @param payload request body
    */
    post(payload: Printer): Observable<NGCResults<Printer>> {
        const url = this.appConfiguration.UrlsConfig.wsPrinter;
        return this.http.post<NGCResults<Printer>>(url , payload).clearListCache(this.clearListTablesCache);
```

```
    put(id: number, payload: Printer): Observable<NGCResults<Printer>> {

        const url = this.appConfiguration.UrlsConfig.wsPrinter + id;
        return this.http.put<NGCResults<Printer>>(url, payload);
    }

    // méthode appeler par composent générique shared/module/devextreme/data-grid.component
    delete(id: number): Observable<any> {
        const url: string = this.appConfiguration.UrlsConfig.wsPrinter + id;
        return this.http.delete(url);
    }


    clearListTablesCache = () => {
        this._list = null;
    }

    // méthode appeler par composent générique shared/module/devextreme/data-grid.component
    navigateToDetail(id: number) {
        this.router.navigate(['/admin/printer/detail/', id]);
    }

    // méthode appeler par composent générique shared/module/devextreme/data-grid.component
    navigateToCreatePage(): void {
        this.router.navigate(['/admin/printer/create']);
    }

    navigateToList(): void {
        this.router.navigate(['/admin/printer/']);
}
```

J'essaye maintenant d'implémenter mon printer-grid en copiant le code d'un autre grid dans le mien. Mais rien n'apparaît.

Je travaille un peu mécaniquement, sans trop comprendre ce que je fais.

Je ne trouve pas comment les infos du grid sont récupérées.

- Erreur "Cannot read property 'mode of undefined'".
    - Je me demande si c'est ça qui me bloque ?
        - Recherche en debugger => HeaderInfo = undefined

Le problème était en fait que le path que j'avais mis sur le bouton était mauvais.

J'ai maintenant quelque chose qui apparaît mais la grille ne charge pas.

- Méthode tableStateLoad en est la cause car "undefined"
    - En debugger, je vois bien que je récupère les infos mais que ça ne s'affiche pas.
    - Erreur résolue en ajoutant dans tableState.ts une condition si on se trouve dans printer, donner l'id 1232 donc non undefined.

```
} else if (nameTable === 'printer') {
    idNumber = 1132;
```

Ça ne marche toujours pas.

Je commente tout le code sauf les déclarations pour voir ce qui s'affiche.

- J'arrive à avoir le nom des colonnes dans la grille.

J'essaye de voir dans un autre grid, ce qui affiche les infos.

- C'est simplement la déclaration du service qui affiche les infos.

Je retourne dans mon printer.service.ts et je m'aperçois d'une coquille : j'avais tout simplement oublié un "n" à "internalName".

- En corrigeant, cette fois, c'est bon : tout s'affiche dans mon grid.

Je crée un slide pannel qui s'ouvre sur le côté pour créer de nouvelles imprimantes.

Je modifie le breadcrumb :

Maintenant il me faut relier les lignes présentes dans la grille au Detail pour pouvoir les modifier ou les supprimer.

- Il faut régler un problème de path du navigateToDetail.

Semaine 7 Jour 1

Je commence la page du printer-detail.

Je change le path de printer-grid/printer-detail/id à printer/printer-detail/id.

Je fais un point avec Gaël pour vérifier si je fais ce qui est attendu.

- La création d'imprimante ne doit pas se faire dans un slide pannel
- Le printer-detail doit être le même que la page FTP

- Le bouton pour accéder au Printer doit être déplacé dans "Matériel, Communication et connecteurs".

Je déplace donc le bouton.

Je restructure le printer-grid par rapport au FTP-grid.

Je fais le printer-detail comme le FTP-detail.

Je change le système de routing en créant printer.module.ts et printer-routing.module.ts

```
@NgModule({
  declarations: [PrinterGridComponent, PrinterDetailComponent],
  imports: [
    CommonModule,
    SharedModule,
    PrinterRoutingModule
  ],
  providers : [
  ]
})
export class PrinterModule { }
```

```
const routes: Routes = [

    {
        path: '', component: PrinterGridComponent,
        data: {
            breadcrumb: 'Administrator.Parameter.Printer',
            name: 'admin.parameter.printer'
        }
    },
    {
        path: 'detail/:id', component: PrinterDetailComponent, canDeactivate: [DesactivateGuard],
        data: {
            breadcrumb: 'Detail',
            name: 'printer.detail'
        }
    },
    {
        path: 'create', component: PrinterDetailComponent, canDeactivate: [DesactivateGuard],
        data: {
            breadcrumb: 'Create',
            name: 'printer.create'
        }
    },
];

@NgModule({
    imports: [RouterModule.forChild(routes)],
    exports: [RouterModule]
})
export class PrinterRoutingModule { }
```

Je remodifie le breadcrumb.

Je teste la navigation.

Je teste les méthodes pour voir si elles fonctionnent côté Front.

Les pages Create et Detail sont bien différenciées.

Dans les options du Detail, je supprime un champ blanc qui était au-dessus de "supprimer".

Je règle le problème du put : il y avait dans le printer.service.ts un paramètre, inutile et en trop, qui cassait la méthode.

Dans le footer, je gère aussi le problème de traduction.

- Il fallait rajouter une traduction du pageTitle dans la DataGridFooter.

Tout à l'air en ordre. Je fais une vérification globale du code, puis je push et fais ma pull request.

Semaine 7 Jour 2

Je reviens sur la tâche.

Je veux faire en sorte, qu'à la création ou à la modification d'un printer, on soit redirigé vers la page du grid.

- Ajout de la méthode onValidateSuccess
    - Implémenter avec un router.navigate()

```
onValidateSuccess(event : Printer){
    this.printerService.currentItem.next(event);
    this.router.navigate([this.returnButton]);
}
```

Si je fais une modification d'un printer et que je save, ça fonctionne.

Si je crée une nouvelle imprimante, je suis redirigé vers sa page Detail mais les infos des champs ne chargent pas. Il faut refresh la page.

- Il fallait juste rajouter la méthode clearListCache() que j'avais enlevée.

Tout fonctionne bien maintenant, je push et je passe la tâche en Cross Testing.

Bilan de cette mission

C'était une tâche compliquée mais dans laquelle j'ai beaucoup appris. Surtout sur le fonctionnement du Back et des web services avec le Controller, Service et Repository.

Coté Front, j'ai compris ce qu'il fallait ajouter pour relier le Front au Back, les différents fichiers à créer, les différentes infos à rajouter un peu partout. Je sais comment sont créés les Grid et Detail.

Maintenant, je saurai le refaire plus facilement, mais toujours à condition d'avoir des exemples sur lesquels copier.

En effet, comme c'est en générique, beaucoup d'infos sont juste des « copiés-collés » d'autres pages et je ne saurai pas le refaire sans exemple, pour l’instant.

Bilan de stage et remerciements

Je tiens tout d'abord à remercier particulièrement Émilie Delarue sans qui, je n'aurai pas pu rejoindre cette entreprise -car je le sais, sans ses recommandations je n'aurais sûrement jamais eu de réponse de l'entreprise.

Je souhaite également remercier son mari, Kévin Delarue, qui m'a accueilli et accompagné avec bienveillance durant ce stage.

Merci d'avoir pris du temps pour moi et de m'avoir fait confiance.

Je suis très satisfait de mon stage chez iD Systèmes, une entreprise dynamique et sympathique. J'ai tout de suite été intégré dans l'équipe et responsabilisé, au même titre que les salarié(e)s de l'entreprise.

J'ai eu la chance de bénéficier d'un stage long, d'environ 3 mois, qui en outre, m'a donné droit à mes premières rémunérations.

J'ai vraiment apprécié qu'on me confie des missions diverses et variées de développement. J'ai beaucoup appris et je remercie les collègues pour leur aide et leur soutien précieux.

Ce stage s'est révélé très enrichissant et confirme mon vif intérêt pour le développement.

Je vais également avoir la chance de pouvoir poursuivre cette collaboration chez iD Systèmes, grâce à un contrat de formation en alternance pour finaliser mes années de Bachelor 2 et 3 en Switch IT.