DEVELOPPEMENT LOGICIEL

Projet SunTracking – Schneider Electric

BREVET DE **T**ECHNICIEN
SUPERIEUR –
SERVICE **I**NFORMATIQUE AUX
ORGANISATIONS –
BAC +2

Option SLAM (Solution Logicielle et Application Métier)

Thibaut MESLIN

18 Janvier 2021 – 26 Février 2021

ÉTABLISSEMENT SAINT-ADJUTOR

1. Remerciements

Je voudrais tout d'abord adresser tous mes remerciements à mon maître de stage, **M. Thomas GAUDRIOT**, pour sa confiance, sa disponibilité et surtout l'autonomie qu'il m'a offerte pendant ce stage.

J'aimerais également remercier **M. Claude UZAN** et **M. Bertrand MUNIER**, membres de l'équipe dans laquelle j'ai été intégré, pour leurs précieux conseils, leur disponibilité et le partage de leurs expériences afin de m'aider au mieux à réaliser la mission qui m'avait été confiée.

Je saisis cette occasion pour adresser mes profonds remerciements aux responsables et aux professeurs du BTS SIO du lycée Saint Adjutor à VERNON (27200), qui m'ont fourni les outils nécessaires au bon déroulement de mon stage. Je tiens à remercier spécialement **Mme. Marise HELLARD**, directrice de notre BTS, qui m'aida lors de ma recherche de stage.

Un grand merci à ma famille, pour ses conseils ainsi que son soutien inconditionnel, à la fois moral et économique.

Table des matières

2. Présentation

1. Présentation de l'entreprise

Schneider Electric est un groupe industriel français créé en décembre 1836 par Eugène Schneider. Cette entreprise fabrique et propose des produits de gestion d'électricités, automates et solutions métiers. Cette entreprise est aujourd'hui largement présente à l'international et l'un des leaders mondiaux dans les secteurs cités plus haut. Ce groupe réuni plus de 100 marques parmi lesquelles se trouvent par exemple *APC* ou *Télémécanique*.

Son siège social se situe à Rueil-Malmaison dans les Hauts-de-Seine (35 rue Joseph-Monier). La direction est répartie en 3 pôles : Rueil-Malmaison en France, Boston aux États-Unis et Hong Kong en Asie. Son président-directeur général à ce jour est Jean-Pascal Tricoire.

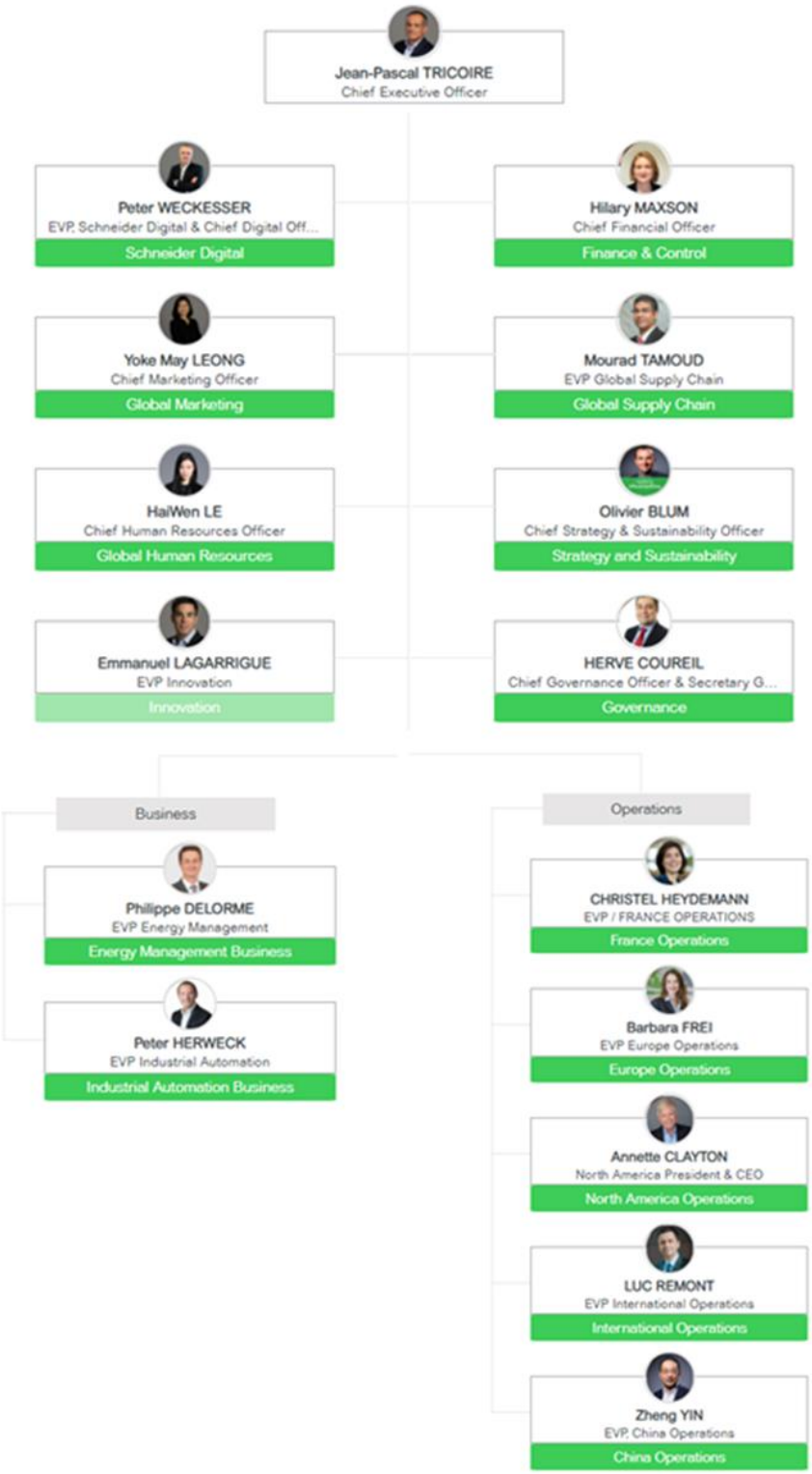
Le groupe compte plus de 130 000 salariés à travers le monde réparti dans 250 sites de production. Le chiffre d'affaires de la société est de 27.2 milliards d'euros, en hausse par rapport à l'année précédente. Le groupe est actuellement numéro un mondial de la distribution électrique (gestion d'énergie, bâtiment, distribution, optimisation de l'énergie, sécurité électrique…). Schneider Electric est également impliqué dans le développement environnemental et durable.

En France, on compte près de 20 000 collaborateurs et 100 sites pour un chiffre d'affaires d'environ 2 milliards d'euros. La moitié de ces sites sont des usines de fabrication du matériel pour 4 centres logistiques et 28 agences commerciales.

La France est au premier plan pour l'innovation et compte 2500 ingénieurs et techniciens R&D (Recherche & Développement). On compte 3 sites majeurs : Electropole à Eybens, Technopole à Grenoble et Horizon à Carros.

Site Schneider Electric: https://www.se.com/fr/fr/

2. Organigramme



3. Présentation de l'entité et du service d'accueil

Lors de mon stage, j'ai été intégré à l'entité **Digital Buildings**. Cette entité traite différentes offres en relation avec le domaine des bâtiments tertiaires :

- La gestion du confort (Gestion de la climatisation, l'éclairage, les stores…)
- La sécurité incendie
- La sécurité (Contrôle d'accès)
- La gestion des espaces (Gestion des réservations des salles, géolocalisation, comptage, occupation des salles…)

J'ai été intégré au service **Recherche & Développement** (**R&D**) de l'entité **Digital Buildings**. Ce service regroupe environ 200 personnes réparties sur différents sites partout dans le monde comme en Angleterre, en France ou au Canada. Les principaux sites se situent à **Lund** en Suède et **Andover** aux États-Unis. Ce service est découpé en plusieurs équipes travaillant sur les différentes offres listées ci-dessus et un domaine parmi : **Software**, **Firmware**, **Test Système**, et **Hardware** (électronique et mécanique).

L'équipe dans laquelle j'ai travaillé est une équipe de R&D basée en France travaillant sur la gestion du confort et qui a pour missions principales :

- Assurer la maintenance d'une offre locale en fin de vie
- Développement des nouvelles offres sur les parties :
    - Software (Le projet principal du moment étant le projet SunTracking)
    - Logiciel embarqué (Télécommandes Bluetooth, gestion de l'éclairage)
- Support technique de la partie commerce France

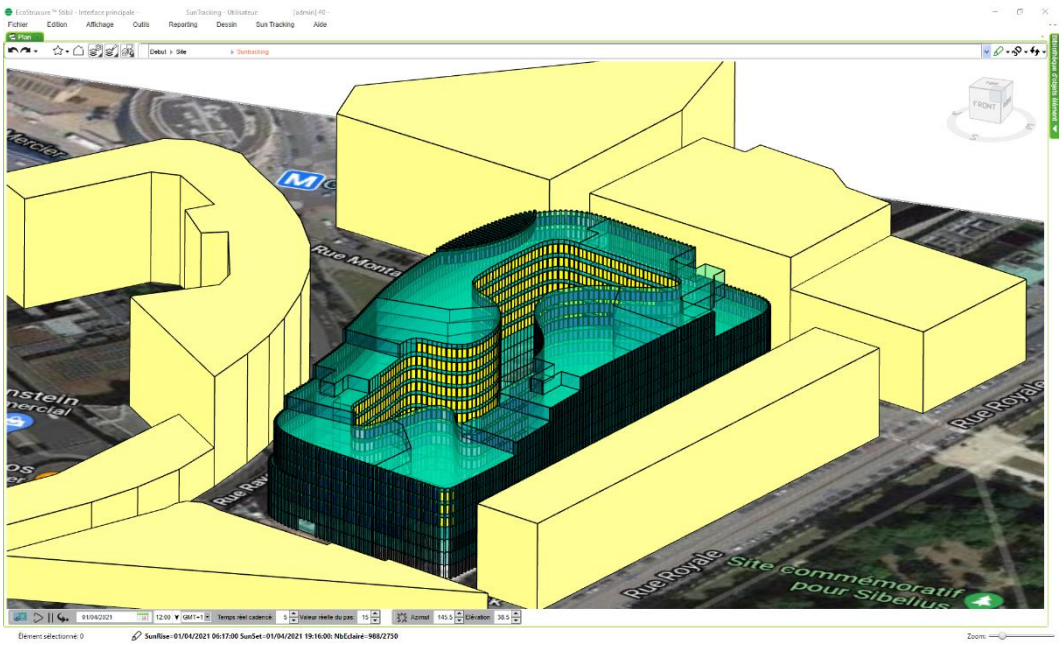
3. Présentation du projet

1. Description des sujets

Il m'a été confié deux sujets liés au projet **SunTracking**.

Tout d'abord, le projet SunTracking a pour but de rendre un bâtiment autonome et propre, c'est-à-dire qui limite au minimum les consommations électriques et permet de garder la même température dans le bâtiment. Ce projet s'articule principalement autour du soleil, de la chaleur et la puissance que celui-ci produit. Au lieu d'augmenter la puissance de la climatisation pour maintenir la température, on baissera les stores afin de limiter les rayons lumineux entrants.

La solution SunTracking est basé sur une modélisation en trois dimensions du bâtiment et de son environnement (bâtiments aux alentours, reliefs…). Cette modélisation 3D permet alors de déterminer théoriquement l'exposition des stores au cours d'une journée, d'un mois ou d'une année donnée. Cette modélisation permettant d'obtenir des fichiers avec des courbes d'expositions solaires et des valeurs permettant ensuite un traitement dans une situation réelle.



Exemple de modélisation 3D (Jaune = store exposé, Bleu = store non exposé)

Le premier sujet consistait à trouver une formule mathématique de l'absorption de la puissance solaire d'un vitrage. Ce sujet était basé sur beaucoup de recherches dans le domaine de la physique. De nombreux paramètres rentrent en compte comme la position du soleil par rapport au vitrage ou encore le type de vitrage. Le but de ce sujet était d'éviter que de l'énergie soit gâchée lorsque la climatisation tourne alors qu'elle pourrait ne pas tourner.

Le second sujet était le sujet principal de mon stage : La réalisation d'une interface d'analyse de traces envoyées par le service SunTracking. Le service SunTracking sauvegarde les différentes actions effectuées au cours de la journée et en garde une trace dans un historique. Cette interface sera utile aux techniciens afin de savoir s'il y a un problème ou une anomalie et le cas échéant pouvoir le localiser pour pouvoir le régler. Elle permet également de savoir si le service fonctionne correctement et d'expliquer au client pourquoi un évènement a eu lieu ou non. Cette interface devra être capable de lire des données et de les analyser pour pouvoir obtenir un historique des actions sur les stores et les commandes.

2. Besoin du projet

Ce projet a pour but d'éviter le gâchis énergétique et donc une pollution due à une surconsommation d'énergie. C'est important d'éviter la surconsommation pour limiter au maximum l'impact sur la planète et le réchauffement climatique. C'est également un moyen de limiter le coût électrique du bâtiment.

Un deuxième besoin était de transférer la fonctionnalité existante afin de la rendre fonctionnelle sur des matériels de nouvelle génération.

3. Client

Ce projet est destiné à un service interne de Schneider Electric lui-même client d'une banque. Par souci de confidentialité, le nom du client sera tu dans ce rapport.

4. Réalisation du projet

1. Outils utilisés

Lors de mon stage j'ai été amené à utiliser différents outils.

Tout d'abord, j'ai principalement utilisé **Visual Studio 2019** comme IDE (Integrated Development Environment). C'est sur cet IDE que j'ai développé mon application, je connaissais cette interface et ce logiciel mais j'ai été plus loin dans l'utilisation de celui-ci en utilisant de nombreuses références comme *Linq* ou *Text.JSON*. Nous avons également développé en utilisant le framework **.NET CORE 3.0** et nous avons également utilisé le **WPF** (Windows Presentation Foundation) qui est la spécification graphique du .NET CORE 3.0 afin de réaliser l'interface graphique de ma partie. Pour les traces, le langage utilisé était le JSON (JavaScript Object Notation) que je connaissais mais n'avais jamais utilisé.

Nous avons été amenés à utiliser **Github** afin de mettre en commun le projet. Pour envoyer ou recevoir les données au dépôt nous utilisions **Fork** ou github directement intégré dans Visual Studio.

2. Démarche de travail

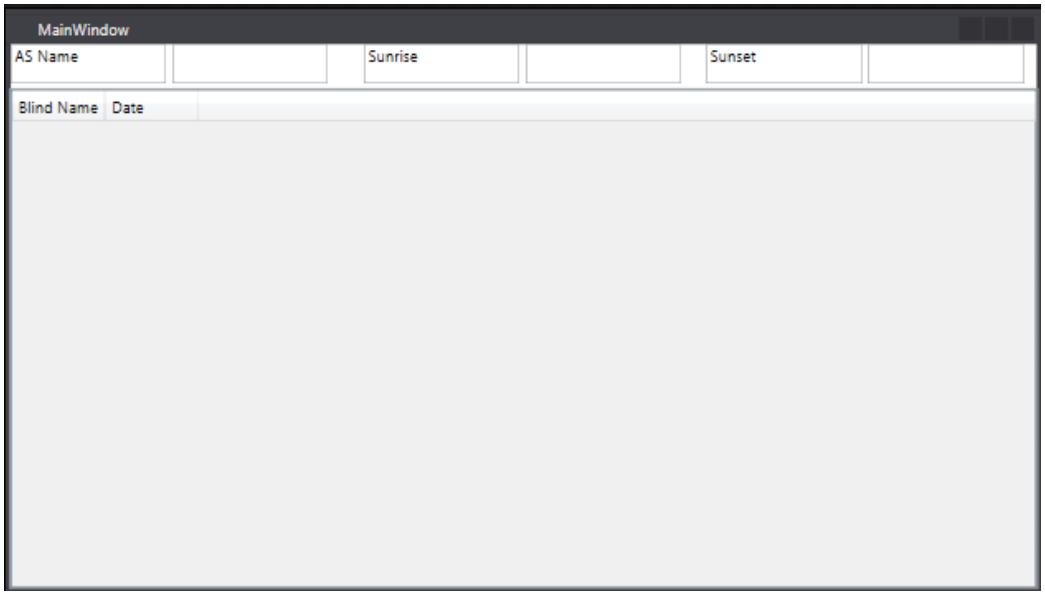
Nous avons travaillé en méthode AGILE, de ce fait nous avions deux réunions dans la journée pour parler de l'avancement de notre travail et des éventuels problèmes rencontrés. Nous en avions une le matin à 9h30 avec **Thomas GAUDRIOT**, **Claude UZAN** et **Bertrand MUNIER** et une à 13h30 avec **William QUIERZY** et **Stéphane BERNARD**, d'autres membres de l'équipe avec lesquels je n'ai pas été amené à travailler.

Je travaillais généralement en autonomie mais j'ai également travaillé avec **Claude UZAN** et **Bertrand MUNIER** qui m'ont apporté de l'aide pour mon sujet ou pour m'apprendre de nouvelles choses comme le WPF que je ne connaissais pas ou encore les requêtes *Linq*. J'ai également vu comment bien organiser mon code et comment faire pour le réduire pour utiliser le moins de performance possible et qu'il puisse tourner sur une machine moins performante. J'apprenais également comment gérer les erreurs, débugger et tester facilement.

J'ai commencé par les recherches sur la formule. J'ai majoritairement fait des recherches sur internet pour de la documentation, des formules existantes ou des schémas pour mieux visualiser la chose.

Ensuite, j'ai enchainé avec la partie de développement. J'ai commencé par me familiariser avec la notion de sérialisation / désérialisation des fichiers **JSON** en **C#**. Une partie importante de mon projet étant de désérialiser des fichiers afin de les lires et de récupérer des informations qui me serviront par la suite, c'était important que cette notion me soit familière et que je sache la mettre en œuvre. J'ai ensuite intégré mon travail au projet existant avec l'aide de **Claude UZAN**. Une fois l'intégration terminée, je me suis mis sur la réalisation de l'interface graphique. J'ai réalisé plusieurs maquettes dont j'ai automatisé quelques éléments pour me familiariser avec le XAML et le WPF en général.

La maquette finale s'est formé au fur et à mesure des *mornings* avec l'équipe SunTracking et des besoins dont nous prenions conscience. La réunion à laquelle j'ai assisté avec le client a apporté beaucoup d'idées pour l'interface comme une vue d'une vitre et de son store en fonction de l'heure et des conditions climatiques extérieures (soleil sans obstacle, nuages qui dure, …).



Exemple de maquette réalisée

3. Modèle de développement

Pour ce projet, les interfaces graphiques sont développées en WPF et le XAML que celui-ci intègre. Nous avons également utilisé un modèle de développement appelé modèle MVVM (**M**odel **V**iew **V**iew**M**odel). Le MVVM est un modèle d'organisation d'architecture logicielle qui a pour but de faciliter le développement d'une interface UI (**U**ser **I**nterface). On compte dans l'architecture du projet 3 dossiers représentatifs de ce modèle :

- Model : la couche de données métier, il n'est lié à aucune représentation graphique
- View : le dossier qui contient les vues qu'on pourra appeler depuis l'application pour les afficher.
- ViewModel : on y retrouve les fichiers permettant de faire le lien entre la vue et le modèle. C'est dans ces fichiers que sont gérées les liaisons de données, conversions et autres. C'est ici que l'on retrouve une autre notion importante : le *Binding*.

Le *Binding*, ou plus précisément *DataBinding* dans notre cas, est le nom donné à la liaison de plusieurs objets afin de les faire communiquer entre eux. C'est grâce à ce procédé que l'on va être capable d'envoyer des variables à la vue qui pourront, si un changement se produit dans l'interface, être modifiées par le code du ViewModel et réafficher dynamiquement. Dans le projet SunTracking, on retrouve par exemple des *bindings* de liste de données afin de les afficher.

L'avantage de ce modèle est qu'il permet de limiter le nombre de fichiers à charger mais surtout de gagner en lisibilité dans le code grâce à une architecture précise et à des codifications pour le nommage des différents fichiers ou variables. On évite également la duplication d'un code car il est utilisé dans plusieurs vues, on réduit donc la taille finale du projet.

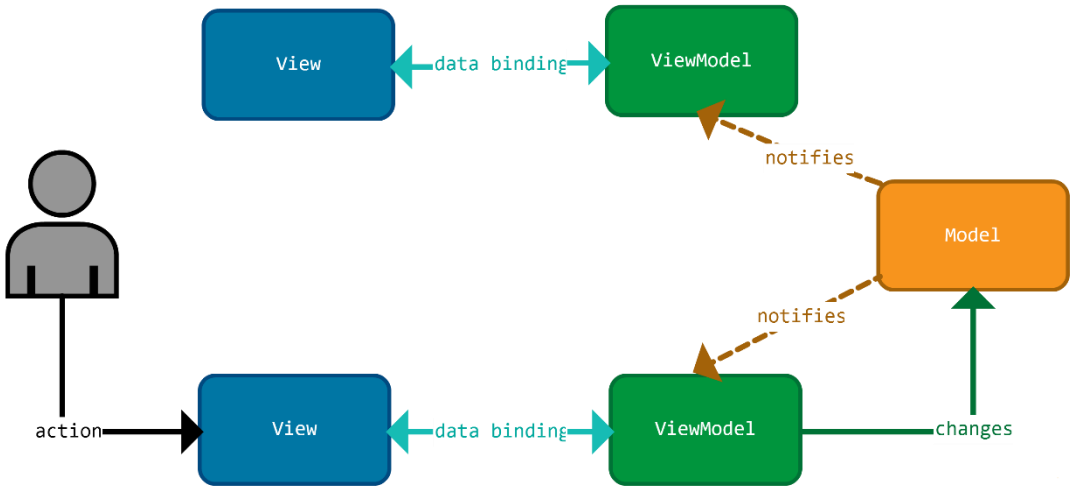


Schéma représentatif du modèle MVVM (**M**odel **V**iew **V**iew**M**odel)

4. Détails techniques : Première activité

Pour trouver une formule d'absorption de la puissance solaire d'un vitrage, j'ai fait beaucoup de recherches sur internet pour voir quels paramètres pourraient influencer la puissance solaire fournie par la sonde météo sur le toit du bâtiment.

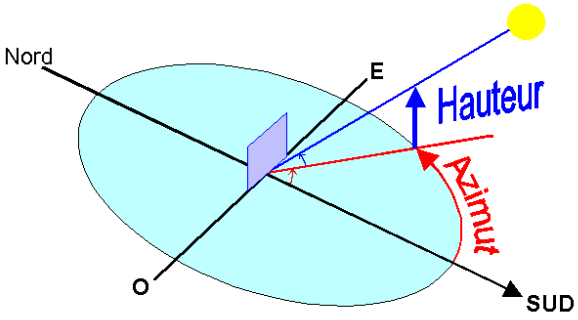
J'ai trouvé plusieurs paramètres, hormis ceux qui m'étaient donnés (azimut et élévation, vitrage à 90° par rapport au sol) qui selon moi peuvent influer sur la puissance solaire : la teinte du verre, le type de vitrage (simple, double, etc...), la capacité du vitrage à réfléchir la lumière et les gaz utilisés dans les doubles vitrages ou plus.

J'ai également trouvé le schéma ci-dessous qui m'a beaucoup aidé pour visualiser la situation et voir une base de formule.

J'ai alors déduit la formule suivante :

$$P_{abs} = P_{brut} * \sin(Azimut) * \tan(Hauteur)$$

Cette formule a ensuite été transmise à un autre service pour voir si celle-ci est correcte et utilisable dans notre contexte.



5. Détails techniques : Deuxième activité

Premièrement, j'ai donc commencé par me familiariser avec la notion de sérialisation / désérialisation en utilisant la référence **System.Text.Json**. Le choix d'utiliser des fichiers Json pour les fichiers de traces a été fait car les fichiers Json utilisent une syntaxe simple et facilement modulable en fonction des besoins mais ceux-ci sont avant tout très légers.

Nous nous sommes d'abord mis d'accord sur la structure que devait avoir le fichier Json et donc les propriétés que notre objet devait avoir car, lors de la désérialisation d'un fichier Json, le contenu devra être associé à un objet qui sera créé avec les données du fichier. J'ai donc créé plusieurs classes contenant les propriétés des traces comme une date et un contexte.

J'ai ensuite créé la méthode de sérialisation qui accepte en argument un objet et un chemin où l'on écriera le fichier, puis la méthode de désérialisation qui admet uniquement le chemin d'un fichier Json à désérialiser. Mais lors de la phase de test un problème est survenu, la bibliothèque **System.Text.Json** ne supporte pas la désérialisation d'une liste contenant des objets de plusieurs classes. Avec Claude, nous avons fait des recherches et trouvé une solution.

La classe de base qui contient la date de création de la trace et le contexte doit être une interface et non une classe, cette même interface doit hériter d'une autre interface qui contiendra un *TypeDiscriminator* qui plus tard permettra à la désérialisation de savoir à quelle classe doit se référer le Json. Les classes des différentes traces doivent toutes dériver de l'interface de base et la méthode de désérialisation doit contenir en option le paramètre *TypeDiscriminator*.

```
public static void Serialize(STTraces obj, string pathfile) {
    JsonSerializerOptions jsonSerializerOptions = new JsonSerializerOptions() {
        Converters = { new TypeDiscriminatorConverter<ISTTrace>() },
        WriteIndented = true
    };
    byte[] jsonUtf8Bytes;
    jsonSerializerOptions.Converters.Add(new Converter.TimeSpanConverter());
    jsonUtf8Bytes = JsonSerializer.SerializeToUtf8Bytes(obj, jsonSerializerOptions);

    File.WriteAllBytes(pathfile, jsonUtf8Bytes);
}
```

Méthode de sérialisation

```
public static STTraces Deserialize(string pathfile) {
    JsonSerializerOptions jsonSerializerOptions = new JsonSerializerOptions() {
        Converters = { new TypeDiscriminatorConverter<ISTTrace>() }
    };
    byte[] jsonUtf8Bytes = File.ReadAllBytes(pathfile);
    ReadOnlySpan<byte> readOnlySpan = new ReadOnlySpan<byte>(jsonUtf8Bytes);
    STTraces obj = JsonSerializer.Deserialize<STTraces>(readOnlySpan, jsonSerializerOptions);
    return obj;
}
```

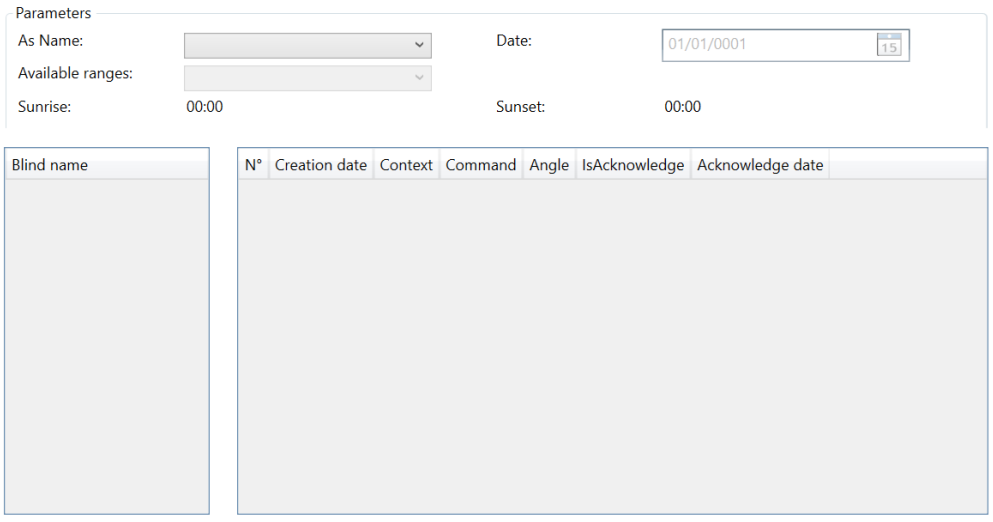
Méthode de désérialisation

Par la suite, j'ai réalisé avec **Claude UZAN** l'intégration de ces méthodes, classes et interfaces dans la solution en utilisant **Github** pour récupérer la solution existante. Nous revenions souvent sur ces classes pour les modifier ou encore pour en ajouter de nouvelles car nous nous étions rendu compte qu'un nouveau type de trace existait et qu'il devait pouvoir être lu, sérialisé / désérialisé et bien reconnu et identifié comme objet de la classe correcte dans le code.

Ensuite je me suis lancé dans la partie graphique. J'ai appris le WPF en créant la première maquette d'UI de ma partie. Je me suis aidé de mes connaissances en Windows Form, de tutoriels internet et d'exemples en tous genres. J'ai également reçu de l'aide de **Bertrand MUNIER** qui avait déjà réalisé d'autres interfaces pour le projet. J'ai appris beaucoup sur les nouveaux composants graphiques disponibles, le modèle MVVM, l'organisation du code avec les vues / ViewModel et surtout le *binding* de propriétés afin de les afficher / modifier dans l'interface. Cette maquette m'a surtout servi à me familiariser avec le WPF en utilisant le XAML pour le côté graphique et l'utilisation des *UserControl* pour les vues, le rôle et le fonctionnement des ViewModel ainsi que l'interface *INotifyPropertyChanged* qui sert à informer des différents changements dans les valeurs des propriétés pour effectuer les mises à jour qui s'imposent.

Avant de fixer l'interface, il fallait réfléchir aux informations dont nous allions avoir besoin pour pouvoir afficher nos traces. Pour cela, j'ai regardé le contenu des fichiers de traces et de configurations et en sont ressorties trois informations : le nom de l'AS (**A**utomation **S**erver) qui est un serveur qui pilote différents équipements comme ceux qui contrôlent les stores, la date du jour pour lequel on veut obtenir l'historique et enfin la plage horaire. Plusieurs fichiers de traces peuvent être émis donc il peut y avoir plusieurs plages. On pourra ensuite choisir le store au cas par cas.

Après quelques interfaces, l'interface que j'ai présenté a été acceptée. Je me suis alors mis à rendre cette interface « réactive », c'est-à-dire que l'affichage puisse s'adapter à la taille de l'écran. Lors de mes recherches, j'ai découvert le système de *Grid* que j'ai appliqué à mon interface. Je découvrais souvent des petits détails qui ne me plaisaient pas sur l'interface comme un espace trop grand lorsque l'on met la fenêtre en plein écran. Je faisais dans ce cas des recherches pour régler mon problème et améliorer mon interface. Les possibilités de personnalisation sont très nombreuses, j'essayais donc de chercher la solution la plus légère et/ou performante.



Interface répondant aux nouveaux besoins

Je suis passé au codage du ViewModel de cette vue. Il fallait d'abord lire les fichiers de traces présents afin de récupérer les informations dont nous avions besoin mais sans récupérer les traces car celles-ci sont nombreuses et prennent du temps à charger et à être lue or elles ne vont pas servir. Il fallait donc s'arrêter une fois la liste des traces ouverte. Une fois ceci fait il fallait ajouter les caractères « **]** » et « **}** » pour fermer la liste puis l'objet et donc avoir une syntaxe correcte. Le but d'avoir une syntaxe correcte était de permettre une sérialisation de cette chaine de caractères afin de récupérer le même objet que dans le fichier mais sans les traces lors de la désérialisation de celui-ci. On aura alors un moyen sûr et performant de récupérer ces données. Une autre information à récupérer était la date de la dernière trace émise afin de « fermer » l'intervalle de temps. Mais le problème était qu'on ne lit pas les traces donc je devais m'y prendre autrement. J'ai alors utilisé la méthode **.Seek** afin de me déplacer dans le fichier et de me positionner **X caractères** avant sa fermeture. Cette méthode m'évitait de devoir lire les traces une à une pour récupérer cette date, le cas où le fichier ne faisait pas 1500 caractères devait alors être pris en compte.

```
private static STTraces LoadFileTrace(string pathfile) {

  string buffer = "";
  string line;

  // on recupere que le debut de la trace (on ne veut pas la liste des traces)
  using (StreamReader sr = new StreamReader(pathfile)) {
    while ((line = sr.ReadLine()) != null) {
      buffer += line;
      if (line.Contains("Traces")) {
        buffer += "]}";
        break;
      }
    }
  }
  STTraces rtn = JsonSerializer.Deserialize<STTraces>(buffer);


  // il faut rechercher dans la liste des traces le dernier DtCreate qui va etre la fin de la trace!
  using (FileStream fs = new FileStream(pathfile, FileMode.Open, FileAccess.Read)) {
    using (StreamReader sr = new StreamReader(fs)) {

      long selectionSize = 1500;
      long posMin = fs.Length - buffer.Length;

      if (posMin < selectionSize)
        fs.Seek(posMin, SeekOrigin.Begin);
      else
        fs.Seek(-selectionSize, SeekOrigin.End);

      while ((line = sr.ReadLine()) != null) {
        if (line.Contains("DtCreate")) {
          // [      "DtCreate": "2021-02-02T17:36:08.4288087+01:00",]
          int posDebut = line.IndexOf("\"DtCreate\": \"");
          if (posDebut >= 0) {
            string s = line.Substring((posDebut + "\"DtCreate\": \"".Length), line.Length - (posDebut + "\"DtCreate\": \"".Length));
            int posFin = s.IndexOf("\"");
            if (posFin >= 0) {
              rtn.DtEOF = Convert.ToDateTime(s.Substring(0, posFin));
            }
          }
        }
      }

      if (rtn.DtEOF == DateTime.MinValue) {
        log.Warn($"the file [{pathfile}] contains no traces");
        return null;
      }
      else
        return rtn;
    }
  }
}
```

Méthode *LoadFileTrace* décrite ci-dessus

Une fois la méthode créée, je l'appelle dans le constructeur de mon ViewModel et je l'assigne à une variable. Cette variable est présente dans le constructeur car je vais avoir besoin de celle-ci tout au long de mon processus d'affichage soit 4 évènements. Il fallait d'abord penser à comment faire en sorte qu'un utilisateur lambda puisse utiliser l'application sans créer de bug ou de crash. Les bugs les plus visibles sur trouvent dans l'ordre de sélection des éléments. Si la date est sélectionnée avant l'AS alors l'application va se fermer car pour poursuivre le traitement on a besoin du nom de l'AS et ainsi de suite pour les autres composants. Il fallait donc dans un premier temps bloquer le **DatePicker** et la **ComboBox** des intervalles. Pour faire cela dynamiquement, j'ai utilisé deux propriétés : *IsDpEnable* et *IsCbRangeEnable* toutes deux liées à la propriété **IsEnabled** de leur contrôle graphique respectif. J'avais ensuite simplement à *binder* mes propriétés dans celle-ci pour les rendre ou non accessible.

```
private bool _isDpEnable;
/// <summary>
/// déblocage du DatePicker
/// </summary>
2 références | claude uzan, Il y a 22 heures | 2 auteurs, 3 modifications
public bool IsDpEnable {
  get { return this._isDpEnable; }
  set {
    if (this._isDpEnable == value)
      return;

    this._isDpEnable = value;
    this.OnPropertyChanged();
  }
}
```

```
<DatePicker IsEnabled="{Binding IsDpEnable}"
```

Exemple pour la propriété *IsDpEnable*

Ensuite, j'ai commencé à coder les évènements *CurrentChanged*. Cet évènement permet d'informer l'interface lorsqu'une modification se produit sur l'élément actif. On passe alors dans une méthode dans laquelle on peut coder des changements, débloquer des éléments graphiques, etc… Le ViewModel comprend trois de ces évènements ainsi qu'un *setter* reprenant ce principe.

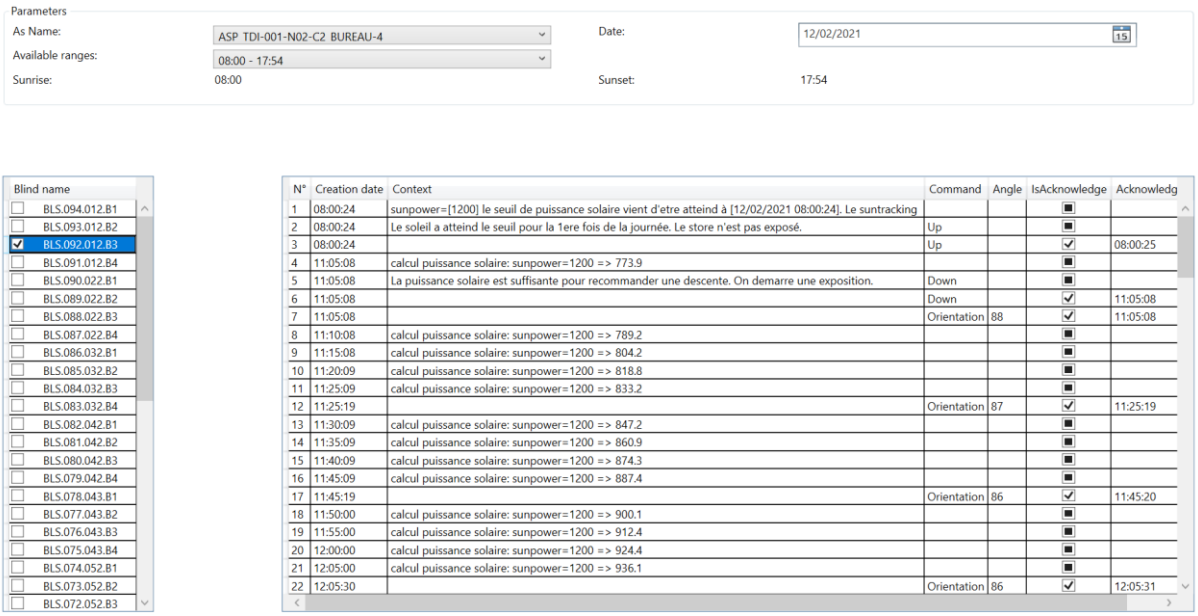
Avant de commencer, les différentes variables et collections devaient être instanciés dans le constructeur. C'est dans celui-ci que l'on charge dans un dictionnaire appelé « _dicoTraceFile » tous les fichiers de traces sans la liste des traces grâce à la méthode *GetAllSTTraces*. On récupère tous les fichiers dans le dossier dont le chemin est spécifié puis on appelle la méthode *LoadFileTrace* ci-dessus. On récupère en sortie un dictionnaire ayant pour clé le chemin du fichier sélectionné et en valeur l'objet **STTraces**.

Le premier évènement se trouve sur la collection des **AS** (**A**utomation **S**erver) chargée dans le premier comboBox. Tout d'abord, on remet à zéro les différentes collections dans le cas où l'utilisateur voudrait recommencer. On récupère d'abord le nom de l'AS sélectionné qu'on met dans une variable pour simplifier l'écriture et la lecture du code. On va alors charger dans une liste les dates pour lesquelles des traces ont été émises. Pour cela, on effectue simplement une requête *Linq* dans laquelle se situe une condition sur le nom de l'AS et une sélection sur les dates avec un *Distinct* pour ne pas avoir de doublons. On récupère également la date la plus récente et la plus lointaine afin de les assigner dans les propriétés *DtDisplayStart* et *DtDisplayEnd* qui définiront les limites de l'intervalle du DatePicker. On débloquera enfin le DatePicker en mettant à jour la propriété *IsDpEnable* pour permettre à l'utilisateur de continuer la sélection.

Le second évènement est le *setter* de la propriété *selectedDate*. On va ici définir les intervalles correspondant aux différents fichiers de traces pour l'AS et la journée sélectionnée. Pour cela, on va à nouveau utiliser une requête *Linq* sur le dictionnaire « _dicoTraceFile » avec comme conditions le nom de l'AS et la journée et en triant par ordre croissant. On ajoute ensuite à l'**ObservableCollection** des intervalles une chaine de caractères du temps de création du fichier et du temps de la dernière trace du fichier. Cet intervalle correspond à la clé du dictionnaire et la valeur au chemin d'accès du fichier. On débloque enfin l'accès au comboBox des intervalles.

Le troisième évènement se situe donc sur le comboBox des intervalles. On va simplement récupérer le chemin du fichier dont l'intervalle est sélectionné et le désérialiser. On va alors récupérer un objet **STTraces** complet (avec les traces) et on va afficher la liste des stores contenus dans cet objet. On va ensuite faire un *foreach* sur la liste de *Blinds* de cet objet et les ajouter à l'**ObservableCollection** des stores.

Enfin, le dernier évènement se situe sur la liste des stores. Une fois qu'un store est sélectionné, on récupère le chemin de celui-ci et on l'assigne à une variable. On va également boucler à l'aide d'un *foreach* sur la liste des traces contenue dans l'objet précédemment obtenu et on teste si le store est le même et dans ce cas on l'ajoute à l'**ObservableCollection** des traces. Comme dit plus tôt, lors de la désérialisation, tous les objets sont des objets de la classe **ISTTraces** et on utilise donc un switch afin de permettre la différentiation des objets et donc d'obtenir un affichage correct.

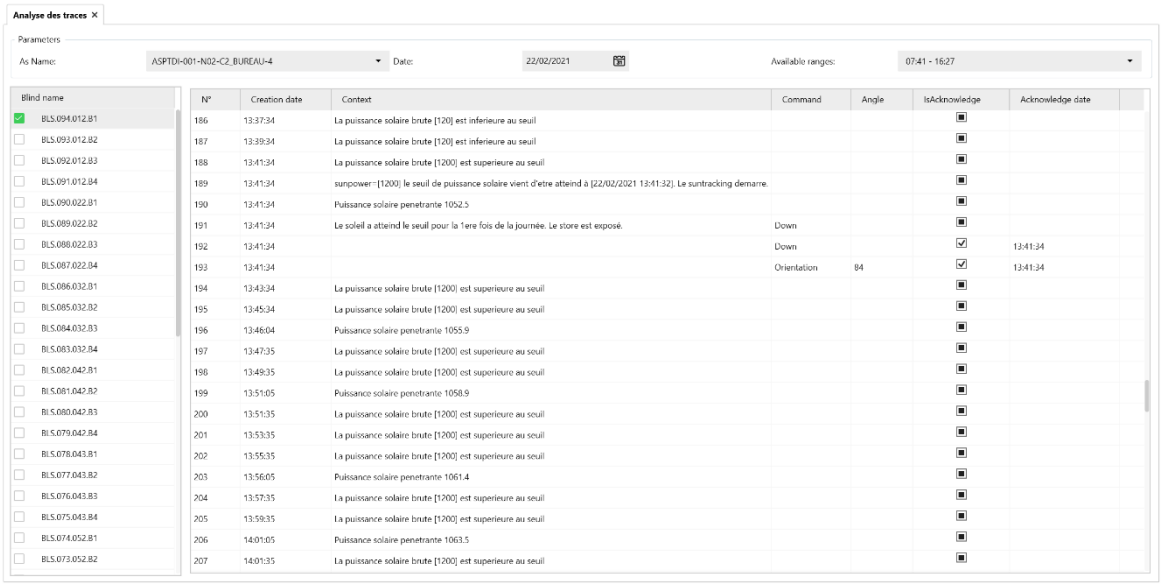


Résultat final

5. Intégration de la fonctionnalité

1. Intégration à l'existant

Une fois la fonctionnalité terminée dans sa première version, elle a été intégrée à la solution existante du projet SunTracking. Jusqu'à lors la fonctionnalité était sur un projet à part et ce projet a été envoyé sur GitHub afin d'être mis en commun. **Bertrand MUNIER** s'est chargé de remanier le style pour qu'il corresponde à l'existant et de corriger les erreurs d'espacement pour rendre l'interface complètement adaptative à la taille de l'écran. Cette fonctionnalité est donc accessible depuis le menu de l'application et fonctionnelle dans le projet.



Visuel de l'application remanié par **Bertrand MUNIER**

2. Ajout d'un nouvel élément

Un nouveau besoin a été exprimé : La visualisation des traces dans l'interface afin de comprendre ce qu'il s'est passé sans devoir lire les traces. Pour réaliser ce nouveau besoin, j'ai travaillé avec **Bertrand MUNIER**. Celui-ci m'a montré comment il faisait pour personnaliser le style d'un contrôle graphique et comment nous pourrions représenter ces traces sur notre interface. Nous avons repris un **Slider** personnalisé qu'il avait déjà utilisé pour une autre interface et nous l'avons personnalisé. Il m'a ensuite fourni du code brut que je devais automatiser et rendre dynamique en fonction de nos traces et des actions présentes dans les fichiers.

Pour ajouter ces éléments, il fallait analyser la trace. L'affichage comprenant quatre niveaux dont le premier déjà codé par **Bertrand MUNIER** et trois autres que j'ai réalisés. J'ai créé une méthode par niveau.

Le second niveau concerne les variations de la puissance solaire brute en fonction du seuil de déclenchement du service de recommandation. Il devait apparaître un soleil lorsque la puissance solaire brute dépasse le seuil et un nuage lorsque celle-ci passe en dessous du seuil. On ne souhaite afficher que le premier élément de chaque série pour éviter une suite d'icône peu compréhensible. J'ai alors créé une méthode qui permet de comparer le dernier élément ajouté avec celui qui doit être ajouté. On teste ensuite toutes les traces concernant la puissance solaire (objet de la classe **STTraceSunPower**) de la collection des traces précédemment filtrée.

```
public IEnumerable<TimeAction> SetLevel1Icons(IEnumerable<TraceViewModel> lstTVM) {

  List<TimeAction> rtn = new List<TimeAction>();

  STTraceSunPower prevTsp = null;
  TimeAction ta;
  foreach (STTraceSunPower tsp in lstTVM.Where(x => x._stTrace is STTraceSunPower).Select(x => x._stTrace as STTraceSunPower)) {
    ta = this.CreateSolarPowerTimeAction(prevTsp, tsp);
    if (ta != null)
      rtn.Add(ta);
    prevTsp = tsp;
  }
  return rtn;
}

1 référence | claude uzan, Il y a 1 jour | 2 auteurs, 2 modifications
TimeAction CreateSolarPowerTimeAction(STTraceSunPower prev, STTraceSunPower curr) {
  if (prev == null || prev.IsThresholdGreater != curr.IsThresholdGreater) {
    return new TimeAction {
      Name = curr.IsThresholdGreater ? "weather_sunny" : "weather_cloudy",
      Tooltip = $"La puissance solaire brute {(curr.IsThresholdGreater ? "a dépassée" : "est passée sous")} le seuil à {curr.DtCreate:HH\\:mm}",
      Time = curr.DtCreate.TimeOfDay,
      Width = 1800,
      Height = 24,
      Color = Application.Current.MainWindow.TryFindResource($"{(curr.IsThresholdGreater ? "Accent-6" : "Accent-16")}") as Brush,
      Level = 1
    };
  }
  else
    return null;
}
```

Méthodes *SetLevel1Icons* et *CreateSolarPowerTimeAction*

Le troisième niveau doit contenir les différentes actions de descente et/ou montée sur les stores. Pour cela, on va chercher dans la liste fournie en paramètre (toujours la liste des traces triée selon le store sélectionné) toutes les traces de commande pour lesquelles on retrouve la propriété *Command* égale à **Up** ou **Down**. On récupère alors les informations et on ajoute les bonnes informations : icône correcte, message correct et couleur : vert si l'accusé de réception est à VRAI sinon rouge.

```
public IEnumerable<TimeAction> SetLevel2Icons(IEnumerable<TraceViewModel> lstTVM) {

  List<TimeAction> rtn = new List<TimeAction>();

  foreach (STTraceBlindCommandSended tsp in lstTVM.Where(x => x._stTrace is STTraceBlindCommandSended).Select(x => x._stTrace as STTraceBlindCommandSended)
    .Where(x => x.Command == BlindCommand.EnumCommand.Up || x.Command == BlindCommand.EnumCommand.Down))
    rtn.Add(new TimeAction {
      Name = tsp.Command == BlindCommand.EnumCommand.Down ? "house_roller_blind_down" : "house_roller_blind_up",
      Tooltip = $"Recommandation de {(tsp.Command == BlindCommand.EnumCommand.Down ? "descente" : "montée")} envoyée à { tsp.DtCreate:HH\\:mm}",
      Time = tsp.DtCreate.TimeOfDay,
      Width = 1800,
      Height = 24,
      Color = Application.Current.MainWindow.TryFindResource(tsp.IsAcknowledge ? "Accent-12" : "Accent-18") as Brush,
      Level = 2
    });
  return rtn;
}
```

Méthode *SetLevel2Icons*

Enfin, on affiche dans le dernier niveau les différentes recommandations d'orientations effectuées par le service. On retrouve le même principe que pour le troisième niveau mais on ne récupère que les traces avec la propriété *Command* égale à **Orientation**. On crée ensuite l'objet sur la même base et on rajoute l'angle dans le **Tooltip**.

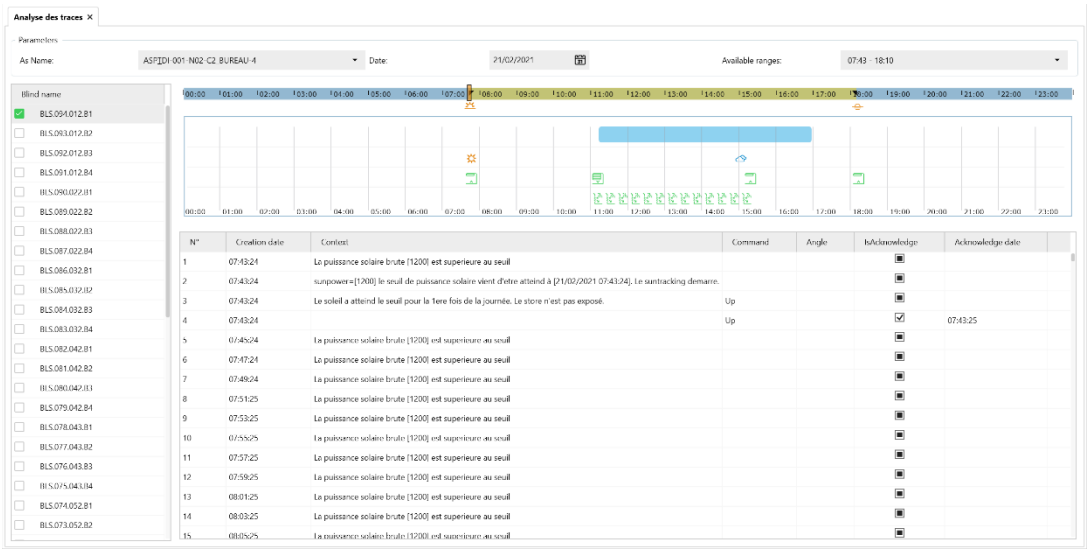
```
public IEnumerable<TimeAction> SetLevel3Icons(IEnumerable<TraceViewModel> lstTVM) {

  List<TimeAction> rtn = new List<TimeAction>();

  foreach (STTraceBlindCommandSended tsp in lstTVM.Where(x => x._stTrace is STTraceBlindCommandSended).Select(x => x._stTrace as STTraceBlindCommandSended)
    .Where(x => x.Command == BlindCommand.EnumCommand.Orientation))
    rtn.Add(new TimeAction {
      Name = "house_blindslat_close",
      Tooltip = $"Recommandation d'orientation de {tsp.Angle}° envoyée à { tsp.DtCreate:HH\\:mm}",
      Time = tsp.DtCreate.TimeOfDay,
      Width = 1800,
      Height = 24,
      Color = Application.Current.MainWindow.TryFindResource(tsp.IsAcknowledge ? "Accent-12" : "Accent-18") as Brush,
      Level = 3
    });
  return rtn;
}
```

Méthode *SetLevel3Icons*

On obtient alors l'interface ci-dessous. Celle-ci répond au besoin formulé et permet de retracer les activités de la journée notamment grâce aux différents messages affichés au survol des icônes.



Interface implémentant le nouveau besoin

3. Présentation de la fonctionnalité

À l'issue de cette fonctionnalité, j'ai présenté lors d'une réunion avec le service demandeur de l'application au sein de Schneider Electric. Ce service est notre client au sein de l'entreprise et celui qui a pour client direct le client présenté plus tôt (cf. *3.3 Client*). Cette présentation servait à présenter notre avancé sur le projet et obtenir des retours sur l'existant. Cette présentation a été réalisé à distance via visioconférence sur **Microsoft Teams**. Lors de cette présentation, j'ai dû présenter ma fonctionnalité ainsi que l'interface réalisée par **Bertrand MUNIER** avant mon arrivée. À la fin de celle-ci, le client s'est montré satisfait de ce qui avait été présenté.

6. Bilan

1. Bilan du sujet

Le sujet qui m'a été proposé était très intéressant car il fait appel à de nombreuses compétences dans le domaine informatique mais également dans le domaine des sciences. Comme me l'a montré le premier sujet, il faut être polyvalent et être capable d'apprendre ou réapprendre certaines notions pour par la suite l'utiliser dans un projet. J'ai également trouvé le sujet très intéressant car il répond à un besoin qui peut paraître simple mais qui lorsque l'on travaille sur le projet ne l'est pas du tout. Je n'imaginais pas tous les cas auxquels l'équipe a pensé afin de rendre le programme le plus complet possible. Je me suis également rendu compte qu'en tant que développeur il faut être polyvalent, connaître beaucoup de langages, non seulement des langages de programmation mais aussi des manipulations via Windows comme des manipulations dans l'invite de commande. Ces connaissances permettent d'être plus rapides dans notre travail et d'éviter des manipulations qui peuvent être longues. Le sujet m'a permis d'acquérir de nouvelles compétences et d'enrichir mes compétences actuelles dans le domaine du développement informatique mais également de voir à quoi ressemblait le travail en entreprise, le télétravail avec des collègues parfois dans d'autres pays, les problèmes que pouvait rencontrer un développeur au quotidien, etc…

2. Bilan général

Plus généralement, ce stage m'a permis de confirmer que je voulais devenir développeur. Ce projet était très intéressant sur le plan technique, humain, marketing et autres, j'en ai beaucoup appris sur tous ces sujets. L'ambiance qui régnait dans l'équipe était très agréable et j'étais content d'en faire partie. Je suis très content de ce que j'ai réussi à produire pendant ces 6 semaines de stage et j'espère que je participerai ou créerai ce genre du projet dans le futur car je les trouve très intéressants. Je me suis également rendu compte qu'il était nécessaire d'être en constant apprentissage et ne pas hésiter à demander de l'aide lorsque l'on bloque car dans ce métier l'expérience est un énorme atout selon moi et comme j'ai pu le voir au cours de mon stage.

Life Is On | Schneider Electric