Résumé de Stage
BTS SIO - SLAM

Nom : Dylan Balmigere
Année scolaire : 2023 - 2024
Lycée : Jean Lurçat

Entreprise d'accueil : Euresto
Période du stage : Du 21 mai au 21 juin 2024

Introduction

Ce document présente un résumé des différentes missions effectuées lors de mon stage. Il met en avant les tâches réalisées, les compétences acquises ainsi que les problématiques rencontrées et les solutions apportées.

J'ai effectué mon stage de deuxième année au sein de l'entreprise Euresto, une société spécialisée dans la programmation informatique, située à Thuir, dans les Pyrénées-Orientales. Euresto conçoit, développe et diffuse des logiciels ainsi que des outils de gestion, tout en proposant des services de conseil, d'installation, de formation et de maintenance informatique. L'entreprise utilise principalement le langage de développement **WINDEV** pour la création de ses applications.

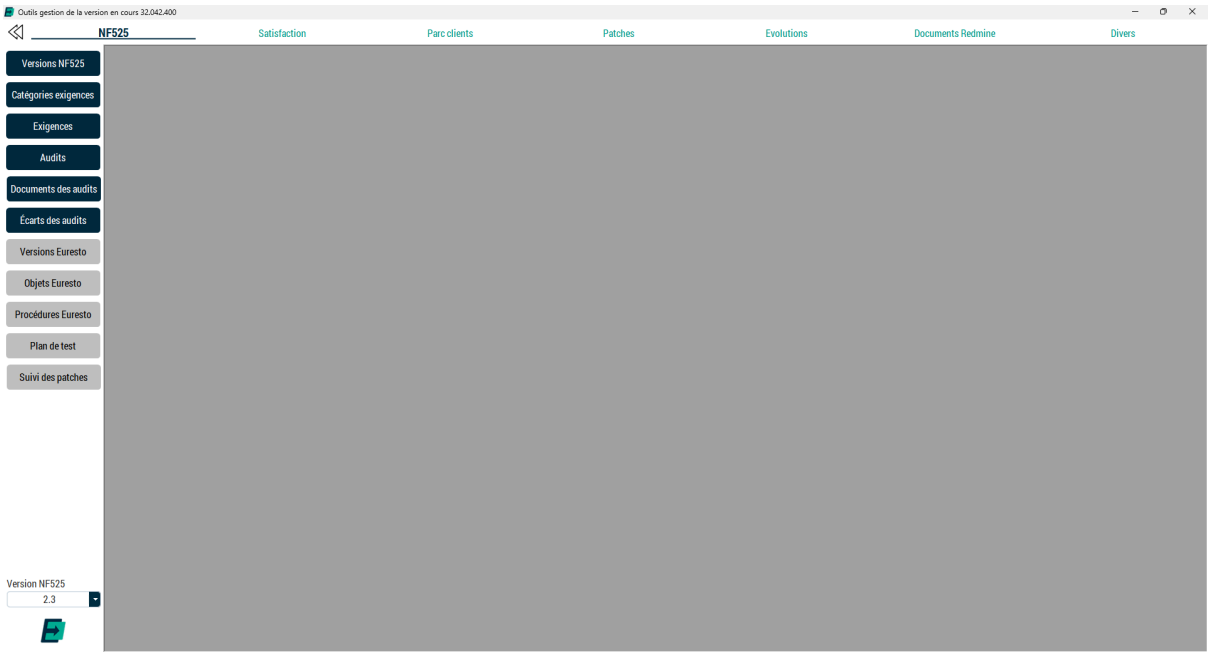
Mon stage s'est déroulé sur une durée de 5 semaines durant lesquelles j'ai pu m'investir pleinement dans différentes missions en lien avec le développement logiciel. J'ai été intégré à l'équipe de développement de l'entreprise Euresto, où j'ai participé à plusieurs projets d'envergure.

Ce stage m'a permis de renforcer mes compétences en développement informatique et de mieux comprendre les enjeux d'un environnement professionnel, notamment en matière de gestion de projet, de communication en équipe et d'adaptation aux besoins des utilisateurs.

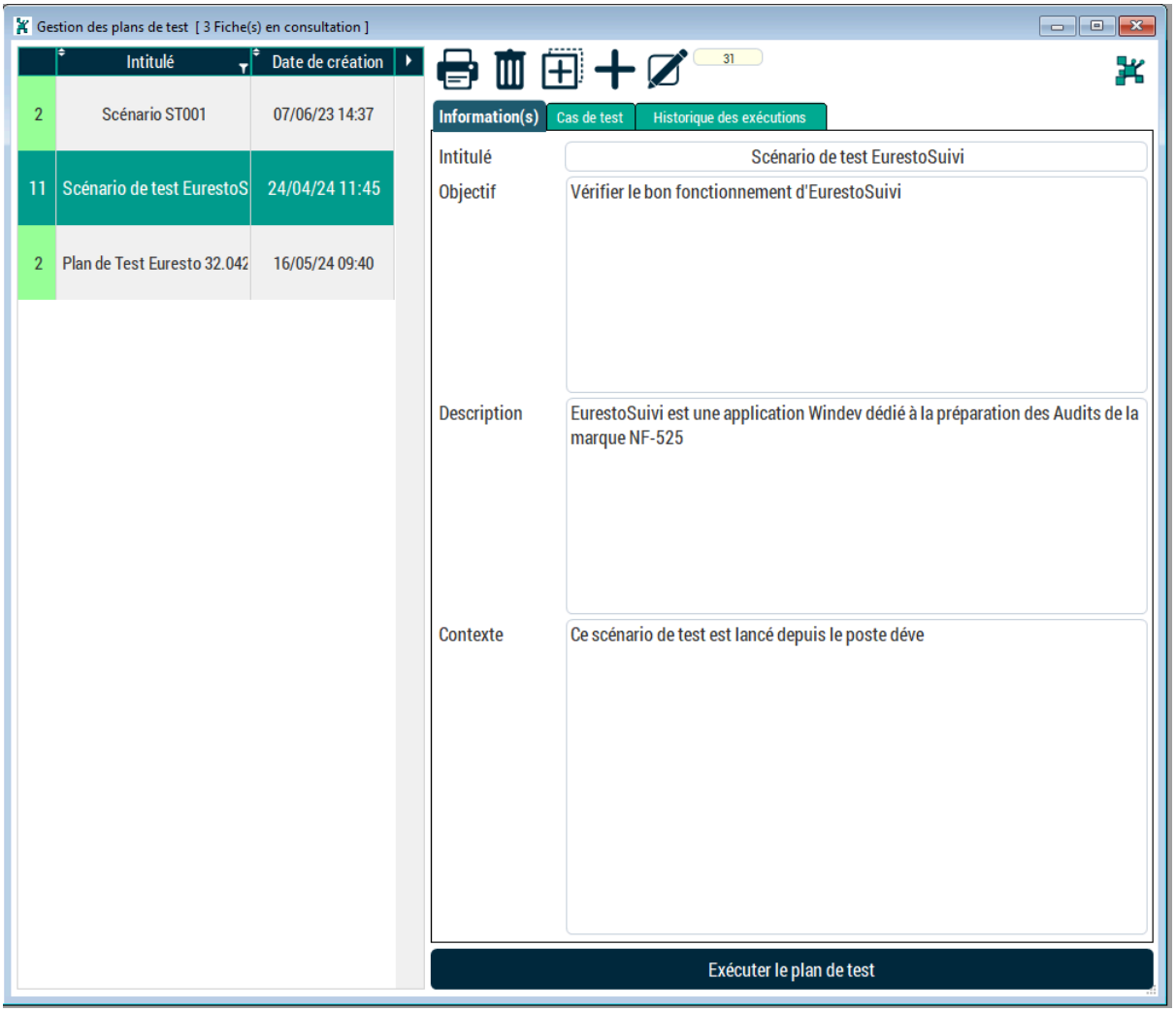
Ce que j’ai fait sur le projet Windev (Niveau Visuel UI)

Je vais vous présenter ce que j’ai fait sur le projet windev, on se trouve donc sur le projet EurestoSuivi qui permet aux employés d’Euresto de respecter les normes pour son logiciel de caisse (Facturation/Billetterie/Gestion de stock) Euresto. En premier temps, je vais vous montrer à quoi ressemble la fenêtre principale lorsqu’on lance l’application EurestoSuivi, on peut remarquer qu’il y a divers options mais pour mon projet, l’option qui m'intéresse est celle des plans de tests comme on peut le voir ci dessous.

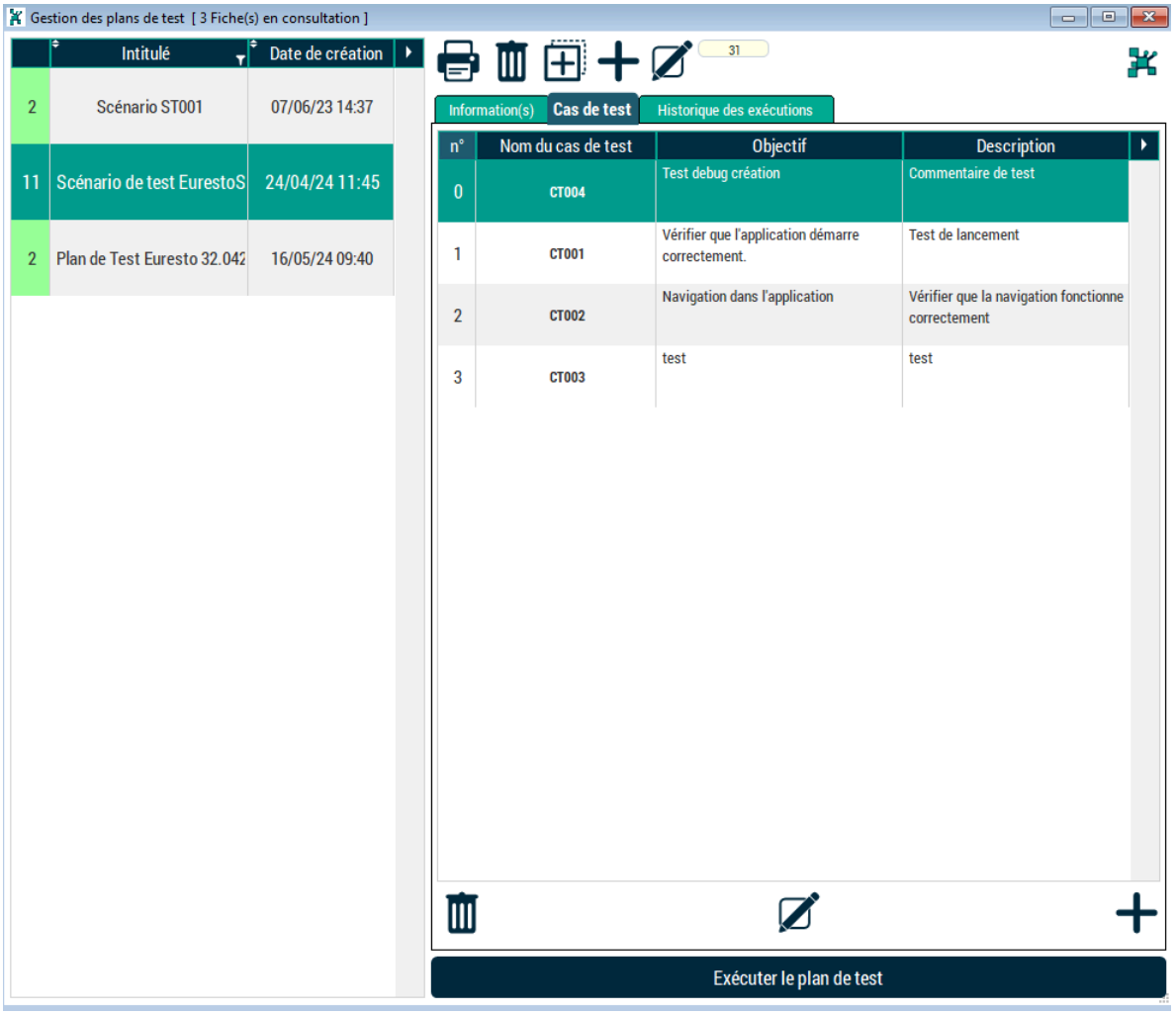
Un plan de test est un document détaillé qui définit une stratégie pour tester un produit logiciel ou un système, en l'occurrence ici le logiciel principal Euresto. Il fournit une structure pour organiser et exécuter les tests comme on l’aura ici avec les différents cas de test que nous verrons juste après :



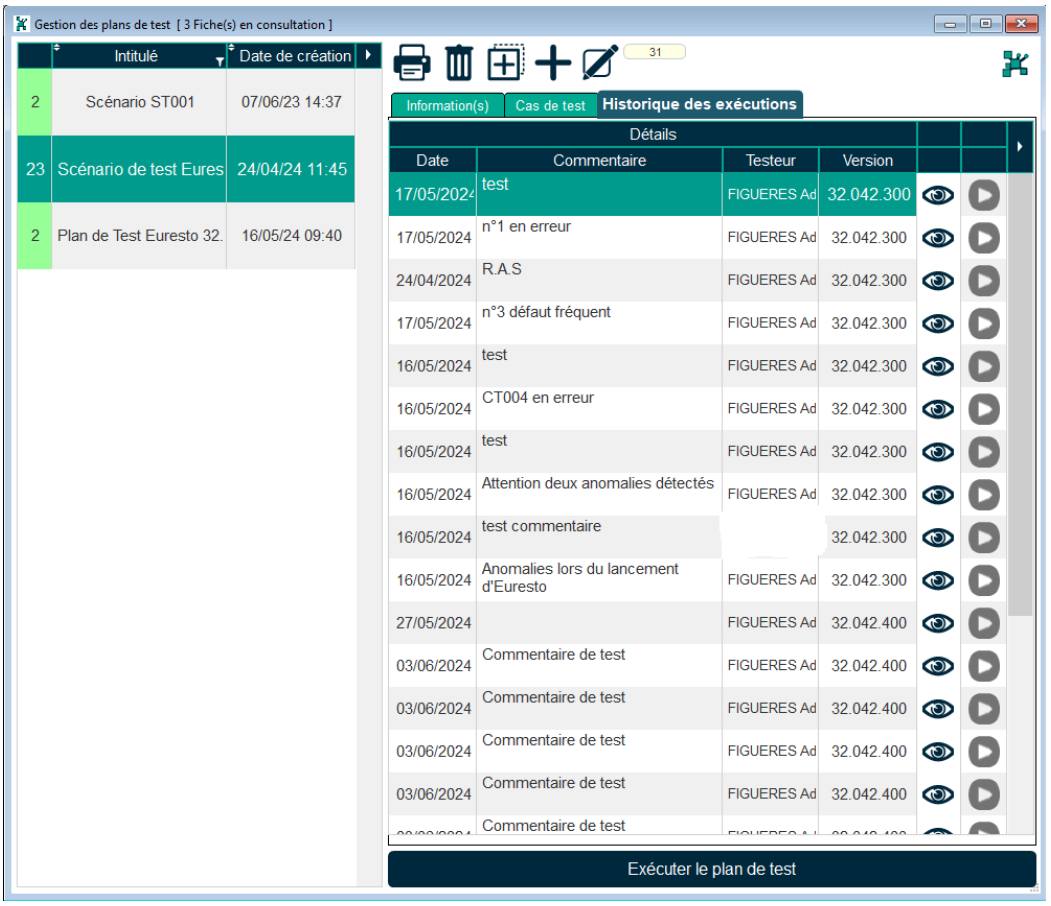
Une fois après avoir cliqué dessus, on se retrouve sur cette fenêtre ci dessous ou l'on peut y retrouver les différents plans de tests sur la gauche de l'écran avec leur intitulé et leur date de création et, du côté droit, on peut voir plusieurs onglets comme les informations du plan de test sélectionné :



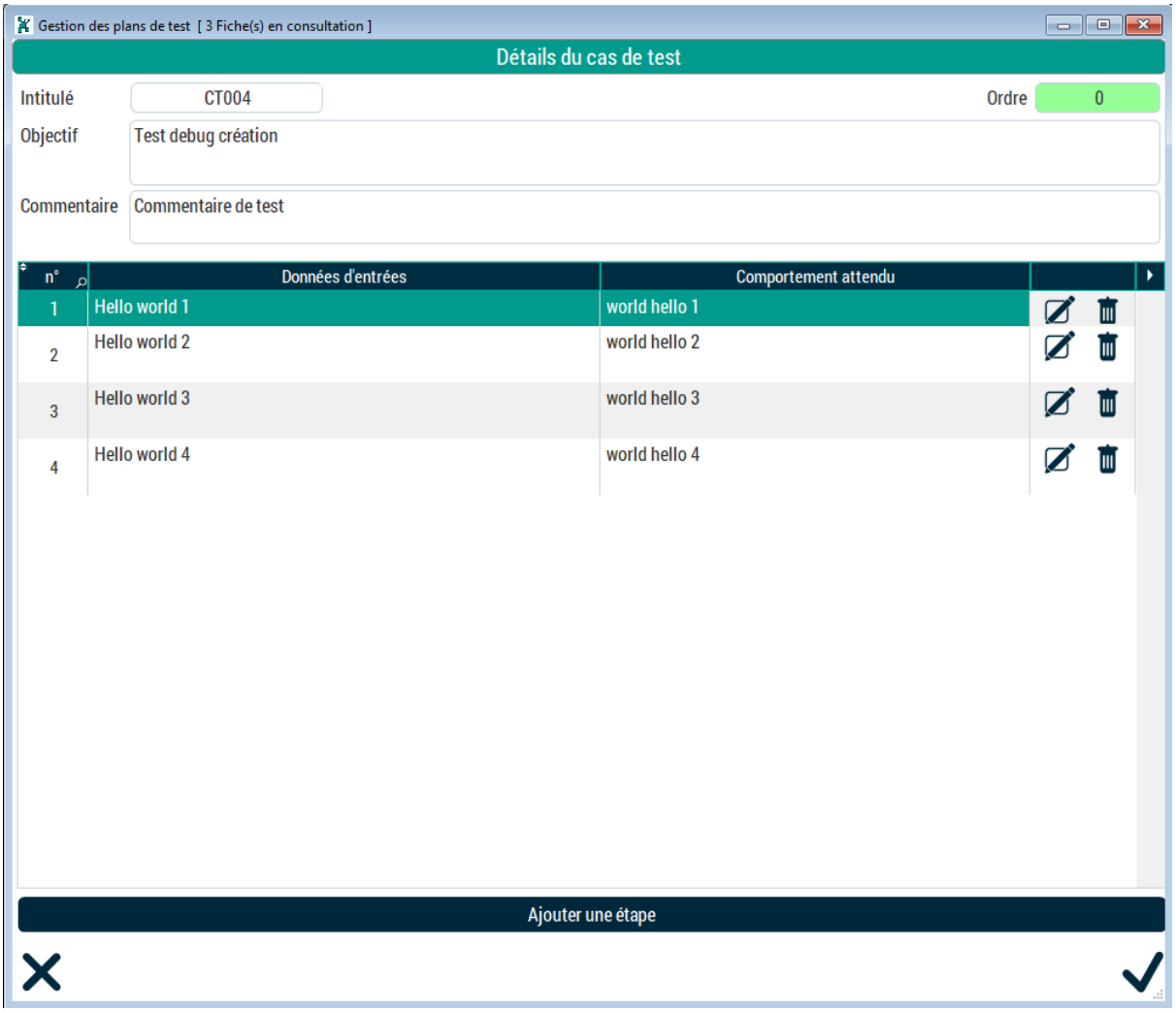
On a aussi l' onglet cas de test ou il y est représenté les différents cas de test que l'on peut retrouver dans un plan de test et il y a un onglet historique des exécutions :



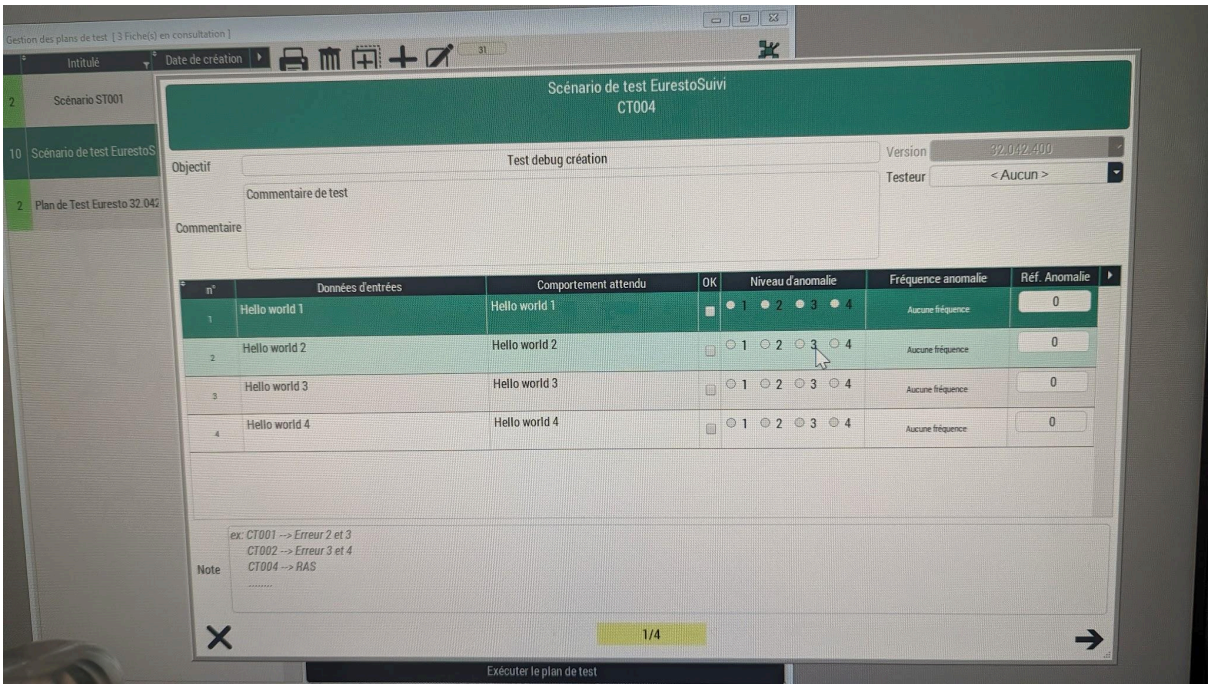
En dernier onglet, on a l'historique des exécutions ou l'on peut voir un historique de qui a effectué un cas de test, a quelle date, sur quelle version il a fait son test et il y a un commentaire à chaque fois :



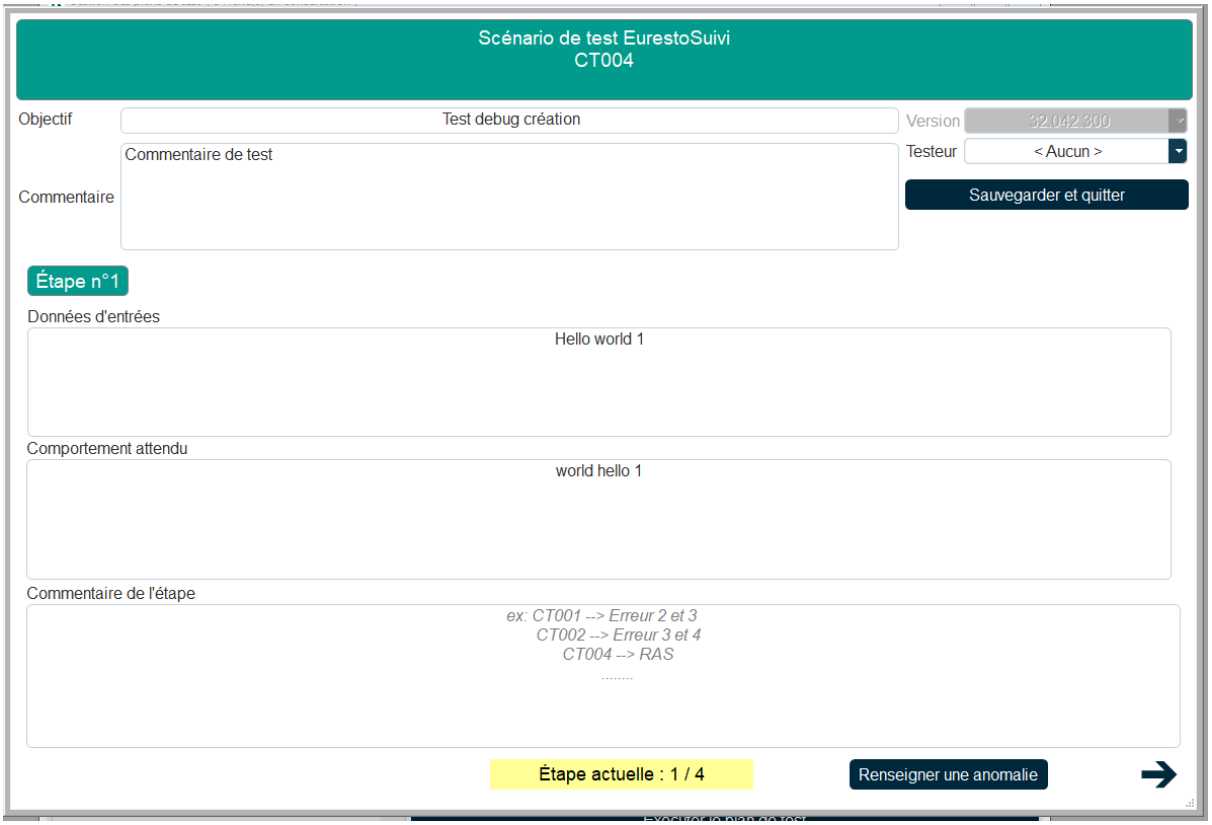
Si on appuie sur le bouton de modification du cas de test en bas de la page, on se retrouve sur cette fenêtre-ci ou l'on peut voir l'intitulé, l'objectif et un commentaire sur le cas de test et en dessous, on peut voir les différentes étapes présentes dans le cas de test que l'on peut modifier à notre guise et même ajouter de nouvelles étapes:



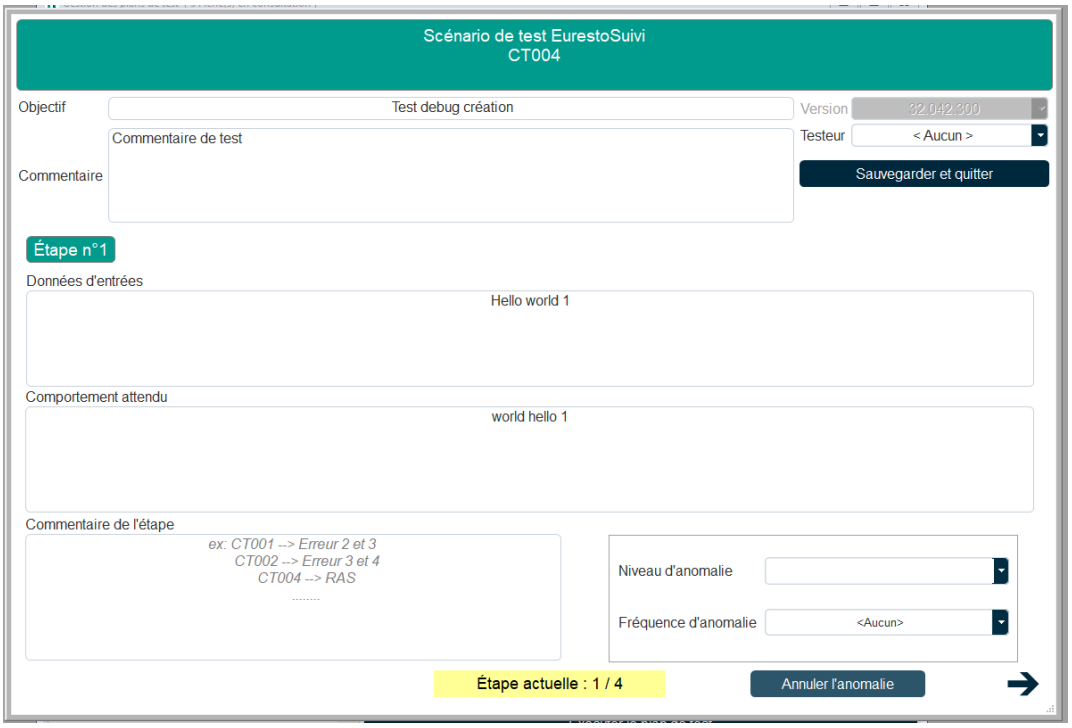
Et là, on peut enfin en venir au projet.
Lorsque l’on appuyait sur le bouton exécuter le plan de test, on arrivait sur cette fenêtre qui présente le cas de test avec ses différentes étapes dans un seul et unique tableau avec très peu de détails sur les différentes colonnes ce qui pouvait rendre la compréhension difficile pour les développeurs d’Euresto:



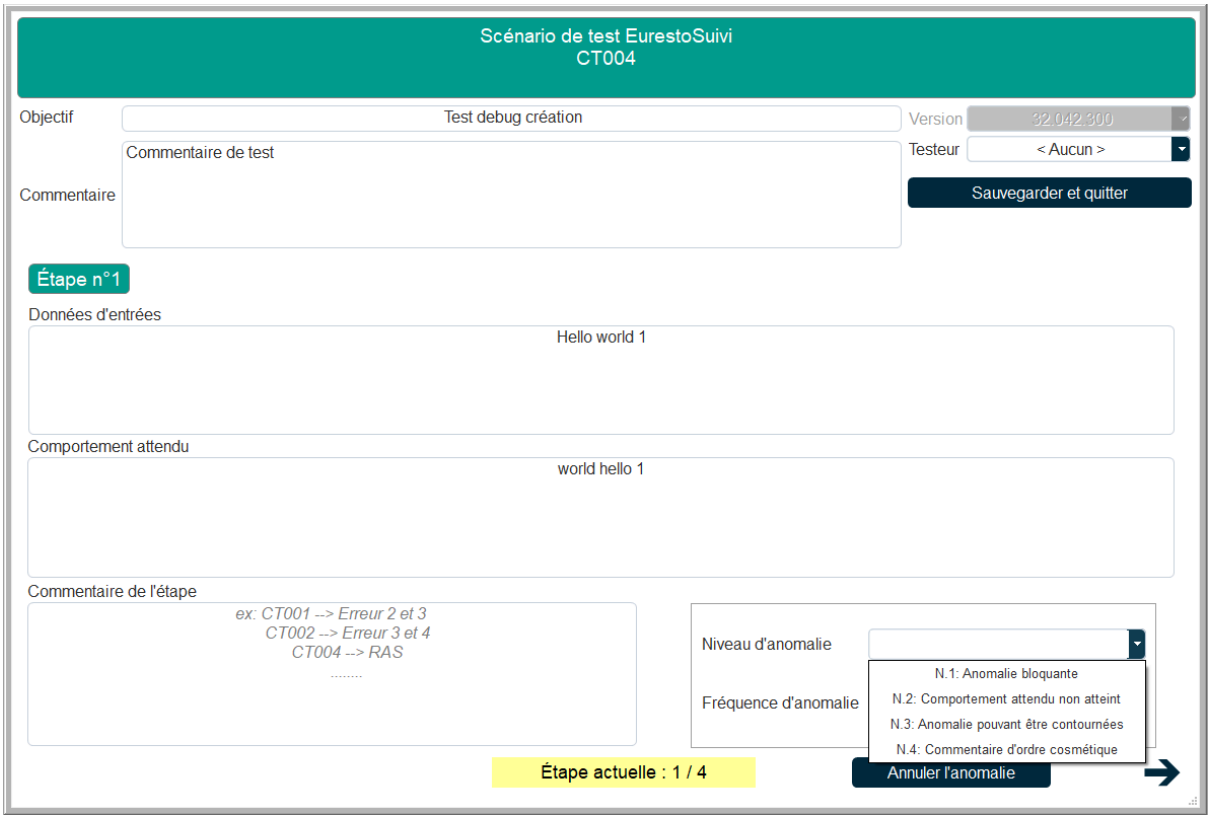
Mon travail a donc consisté à refaire et améliorer cette fenêtre en supprimant le tableau et en restructurant tous les éléments dans des parties distinctes pour une meilleure compréhension, on peut donc voir 3 champs de texte ou sont affichés les données d'entrées, le comportement attendu et le commentaire de l'étape.
On a un bouton "Sauvegarder et quitter" qui à la même utilité que le bouton X de l'ancienne page mais qui est plus jolie a voir que celui-ci, on a un libellé en bas de la page permettant d'afficher en clair à quelle étape l'on se situe et combien il en reste, on a un bouton de Validation étant la flèche en bas à droite qui permet d'actualiser les informations afin d'afficher les infos de la seconde étape.
On a une flèche qui se situe en bas à gauche mais que l'on ne voit pas sur l'image car elle apparaît uniquement lorsque l'on est à une autre étape que la première et elle permet de revenir à l'étape précédente si besoin de faire des modifications.
Il y a un bouton Renseigner une anomalie en bas de la fenêtre qui permet d'afficher les autres information ayant un rapport avec les anomalies quand on appuie dessus tels que la fréquence d'anomalie et le niveau de l'anomalie et le libellé du bouton change en "Annuler l'anomalie" quand il est appuyé :

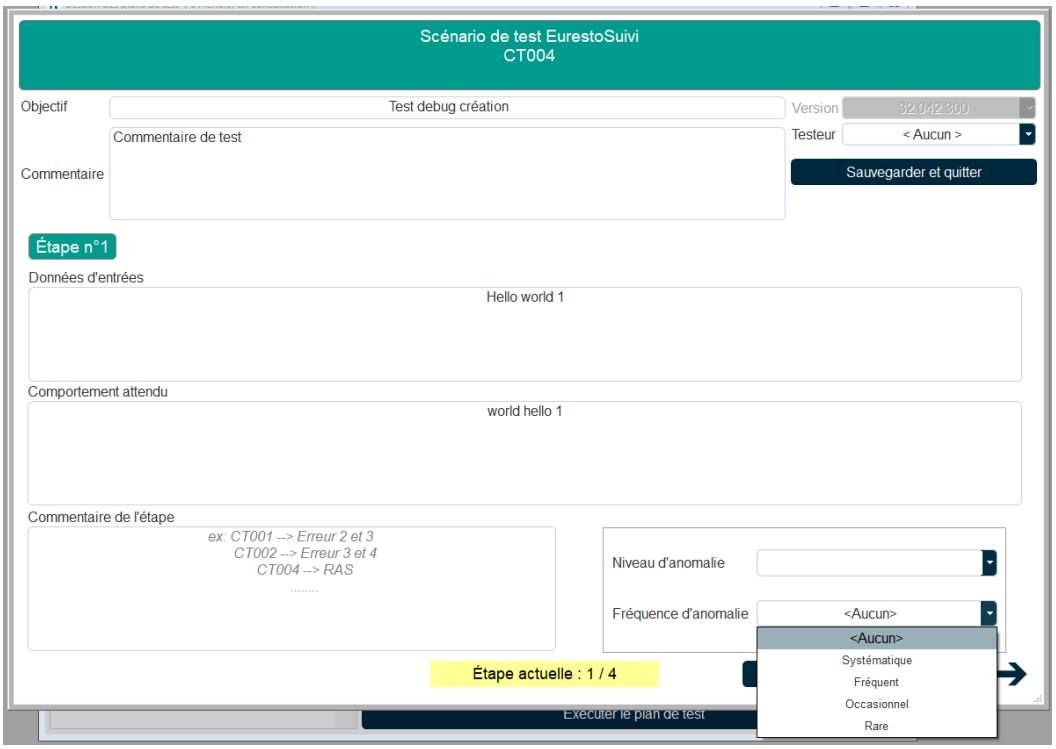


Voici donc la fenêtre lorsque l'on appuie sur le bouton Renseigner une anomalie, le champ du commentaire a laissé de la place au champ de l'anomalie :

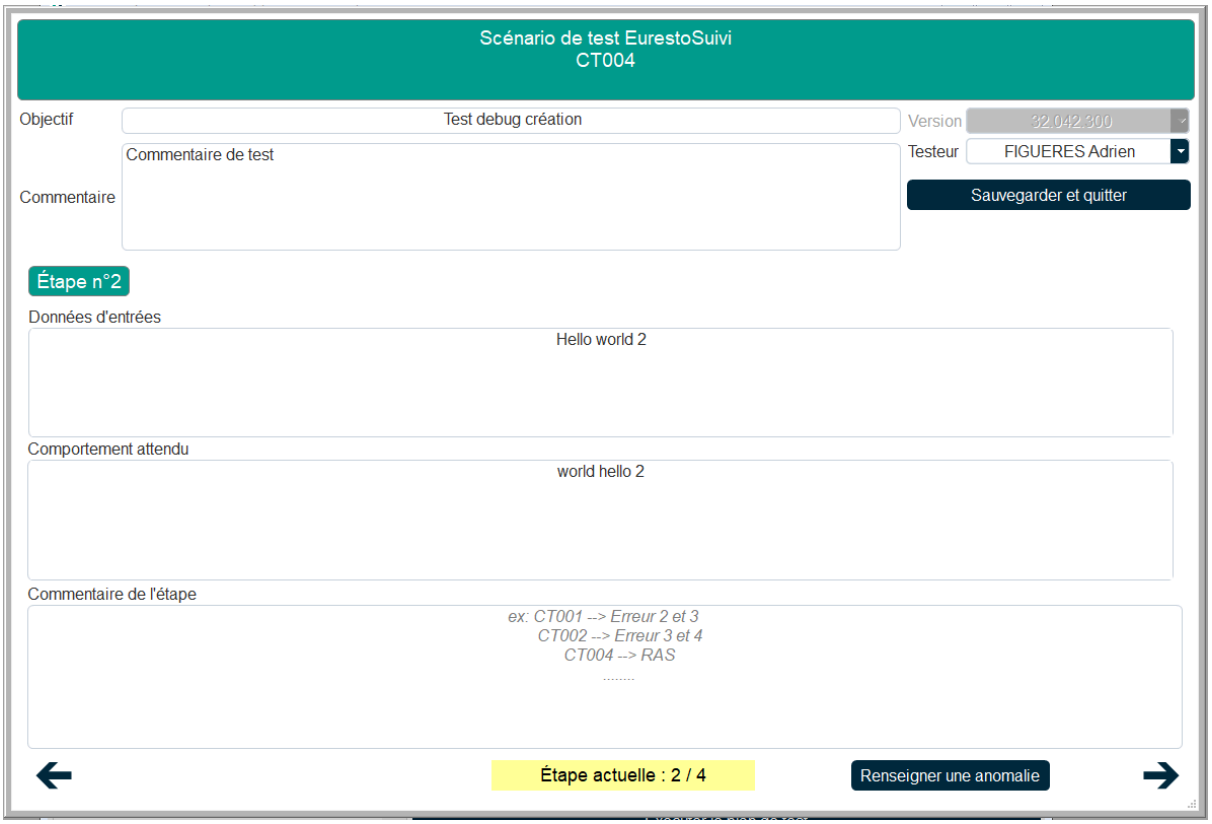


Voici les différentes options que l'on peut retrouver pour le niveau d'anomalie et la fréquence d'anomalie :



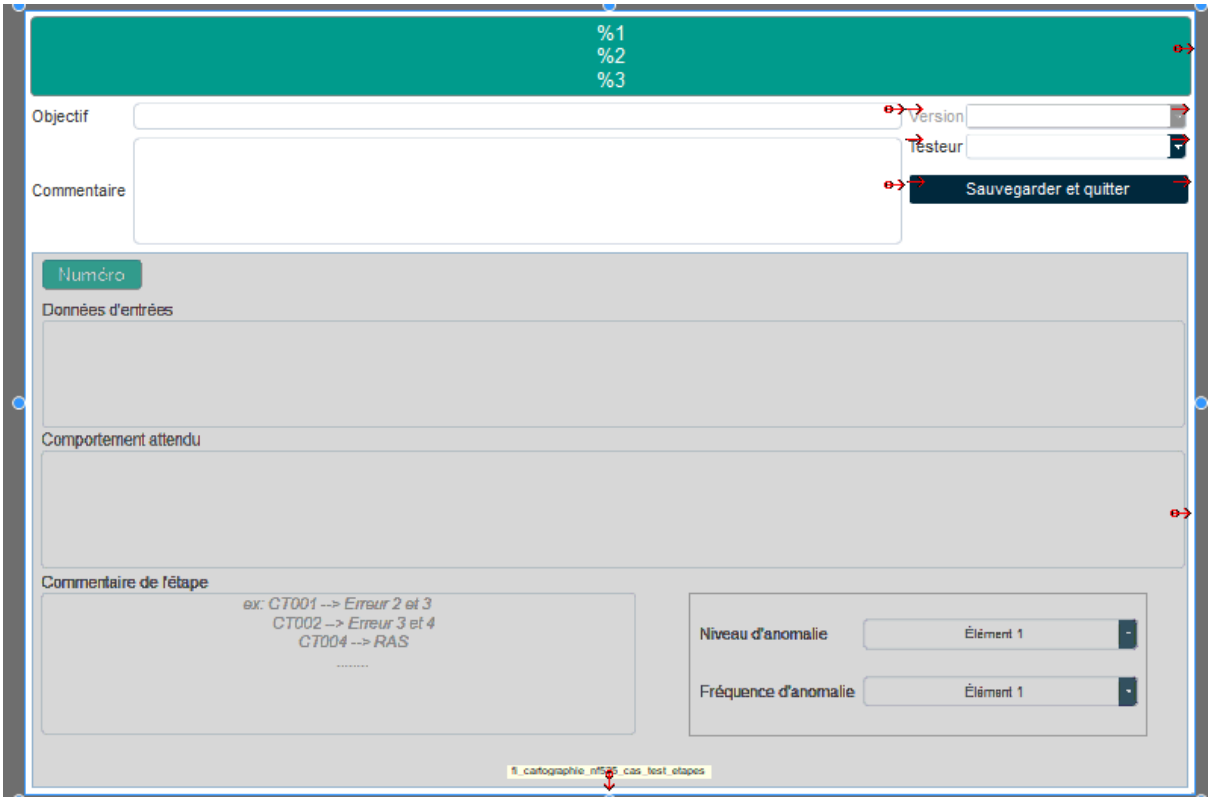


Et voici ce qui se passe quand on appuie sur le bouton Valider , on peut voir que l'on est passé à l'étape 2, les différentes informations dans les champs de texte ont changées et la flèche situé en bas à gauche est bien présente :

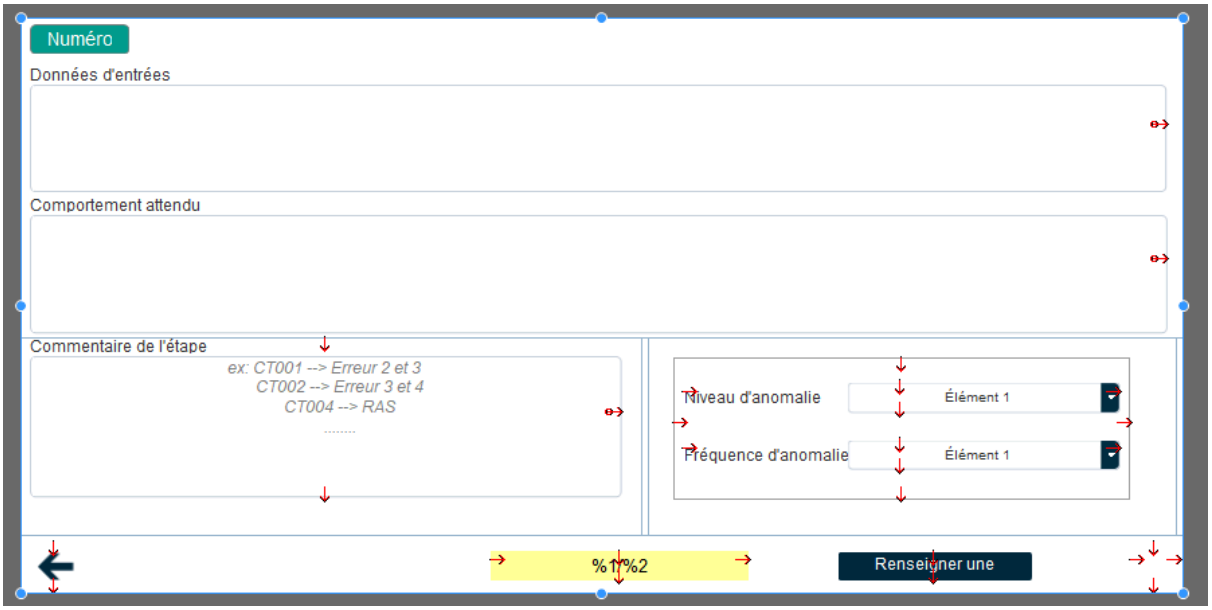


Ce que j'ai fait sur le projet Windev (Niveau Programmation)

Au niveau du codage, comme on peut le voir au titre, j'ai utilisé Windev qui utilise le système de WLangage permettant la programmation impérative et la programmation orientée objet.
Ci dessous, voici la fenêtre principale ff_carographie_nf525_plan_test_execution sur laquelle j'ai travaillé et, comme vous pouvez le remarquer il y a une partie grisé qui correspond au champ fenêtre interne que j'ai intégré dans la fenêtre principale ou j'ai surtout fait des modifications à l'intérieur de celle-ci :



Vous pouvez donc voir juste ici la fenêtre interne fi_cartographie_nf525_cas_test_etapes que j'ai créé à la place du tableau et c'est là que j'ai fait les plus grosses modifications, on peut voir plusieurs champs de texte au format RTF qui permettent de transformer les codes JSON en seulement les données souhaités avec des commentaires pour chacune des étapes, on a un champ de texte SaEtapeCommentaire qui est lié aux différents éléments de l'anomalie grâce à un champ Disposition qui me permet d'adapter la taille des différents champs en fonction de s' ils sont visibles ou non.
Il y a un libellé LiEtapeInfo qui dit a quel étape du cas de test on se trouve, on a un champ bouton "Renseigner une anomalie" BuActionEtat qui affichera toutes les infos concernant les anomalies tels que la fréquence d'anomalie qui est une comboBox / liste déroulante CbFréquenceAnomalie, et le niveau de l'anomalie qui est une combobox CbNiveauAnomalie :



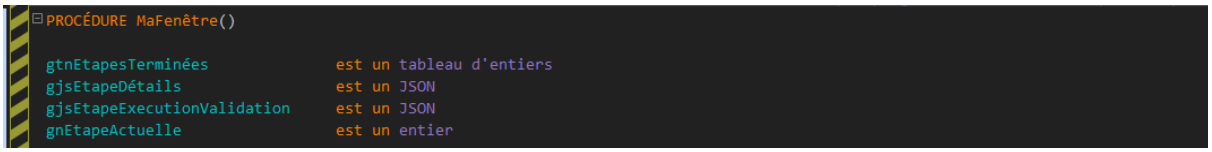
Pour faire cette page j’ai dû coder plusieurs choses :

En premier temps, dans ma déclaration globale, je définis ma fenêtre interne grâce à une procédure qui l'initialise et je définis aussi la variable gtnEtapesTerminées qui est un tableau d'entiers me permettant de stocker la valeur de chaque étape que j’affiche
(Par exemple, quand je vais passer à l’étape 2, je vais alors stocker la valeur 1 dans mon tableau).

On a aussi une variable gjsEtapeDétails qui est un JSON que je récupère depuis la fenêtre principale.

On a une variable jsEtapeExecutionValidation qui est un JSON qui me permet une fois que j’ai vidé le tableau, de passer au cas de test suivant si le testeur a bien été choisi.

Et on a la variable gnEtapeActuelle qui est un entier que j’utilise pour pouvoir enlever la flèche en bas à gauche quand je suis à l’étape 1 et qui récupère la valeur de l’étape actuelle pour revenir aux étapes précédentes:



Ici je déclare ma procédure _Init() dans laquelle je définis pjsEtapeDétails (Paramètre d'une fonction) qui est un JSON puis, je dis a pjsEtapeDétails d'affecter son contenu a gtnEtapesDétails comme ça il récupère bien le JSON de la fenêtre principale qui contient les informations principales de l'étape en cours.

Ensuite, j'initialise LiEtapeInfo qui est un libellé de ma fenêtre me permettant d'afficher sur quel étape l'on se trouve grâce à gtnEtapesDétails, je fais ensuite appel à la procédure EtapesAction() que j'ai défini plus bas qui me permettra d'afficher les différentes étapes des cas de test, je lui indique CTE_PROC_ACTION_AFFICHAGE_COMPLET qui est juste une constante (Une constante est une variable que l'on déclare comme toute variable cependant la valeur de cette variable figé lors de sa déclaration et elle est immuable dans la plupart des cas c'est à dire qu'il n'est pas possible de lui affecter une nouvelle valeur plus tard dans le code) que l'on donne pour mieux comprendre le code et dans lequel on peut lui donner des instructions dans ce qu'on appelle des cas.

Je fais appel à la procédure ComboNiveauAnomalieRemplissage() que j'ai définie plus bas et elle me permet d'initialiser les différentes informations que j'ai dans mon Combo des niveaux d'anomalie et pareil pour ComboFrequenceAnomalieRemplissage() qui me permet d'initialiser les différentes informations que j'ai dans mon Combo des fréquences d'anomalie :

```
Procédure locale _Init *    Si Erreur : par programme    Quand Exception : par programme
// Déclaration de la procédure _Init avec un paramètre pjsEtapeDétails de type JSON
// Syntaxe :
//_Init (<pjsEtapeDétails> est JSON)
//
// Paramètres :
//  pjsEtapeDétails (JSON) : <indiquez ici le rôle de pjsEtapeDétails>
// Valeur de retour :
//  Aucune
//

PROCÉDURE _Init(pjsEtapeDétails est un JSON)

//--- Affecte le contenu du paramètre pjsEtapeDétails à la variable globale gjsEtapeDétails

gjsEtapeDétails = pjsEtapeDétails

//--- Initialise le libellé LiEtapeInfo avec des informations d'étape

LiEtapeInfo = "Étape actuelle : 0 / " + gjsEtapeDétails.etapes..Occurrence

//--- Appelle la procédure AfficheEtapes pour afficher les étapes

EtapesAction(CTE_PROC_ACTION_AFFICHAGE_COMPLET)

ComboNiveauAnomalieRemplissage()
ComboFrequenceAnomalieRemplissage()
```

On se retrouve dans la procédure locale ComboNiveauAnomalieRemplissage() qui, comme je l'ai dit au dessus, sert a afficher les différentes valeurs du combo (sauf le 0 car il n'est pas dans la limite) des niveaux d'anomalie, j'utilise juste un ListeSupprimeTout() pour que cela ne m'ajoute pas constamment dès que je change d'étapes ces différentes valeurs.Vous pouvez voir tout ça sur les photos juste en dessous :

```
Procédure locale ComboNiveauAnomalieRemplissage
// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
//ComboNiveauAnomalieRemplissage ()
//
// Paramètres :
//  Aucun
// Valeur de retour :
//  Aucune
//
// Exemple :
// <Indiquez ici un exemple d'utilisation>


PROCÉDURE ComboNiveauAnomalieRemplissage()

ListeSupprimeTout(CbNiveauAnomalie)

POUR nLigne= 1 _À_ 4
    ListeAjoute(CbNiveauAnomalie, gLien(nLigne) + CesPlanTestAnomalieNiveauLibellé(nLigne))
FIN
```

```
PROCÉDURE CesPlanTestAnomalieNiveauLibellé(LOCAL pnNiveauAnomalie       est un entier)

SELON pnNiveauAnomalie
    CAS 0 : RENVOYER "N.0: Aucune anomalie"
    CAS 1 : RENVOYER "N.1: Anomalie bloquante"
    CAS 2 : RENVOYER "N.2: Comportement attendu non atteint"
    CAS 3 : RENVOYER "N.3: Anomalie pouvant être contournées"
    CAS 4 : RENVOYER "N.4: Commentaire d'ordre cosmétique"
    AUTRE CAS : RENVOYER "Aucune (Pas d'anomalie)"
FIN
```

On se retrouve dans la procédure locale ComboFrequenceAnomalieRemplissage() qui, comme je l'ai dit au dessus, sert a afficher les différentes valeurs du combo des fréquences d'anomalie, j'utilise juste un ListeSupprimeTout() pour que cela ne m'ajoute pas constamment dès que je change d'étapes ces différentes valeurs.Vous pouvez voir tout ça sur les photos juste en dessous et juste après, j'initialise un à un les différents éléments du combo avec un gLien qui me permet de récupérer dans le code, la valeur logique de l'élément que j'ai sélectionné et je récupère les différentes valeurs que je veux afficher dans CesPlanTestAnomalieFrequenceApparitionLibellé() :

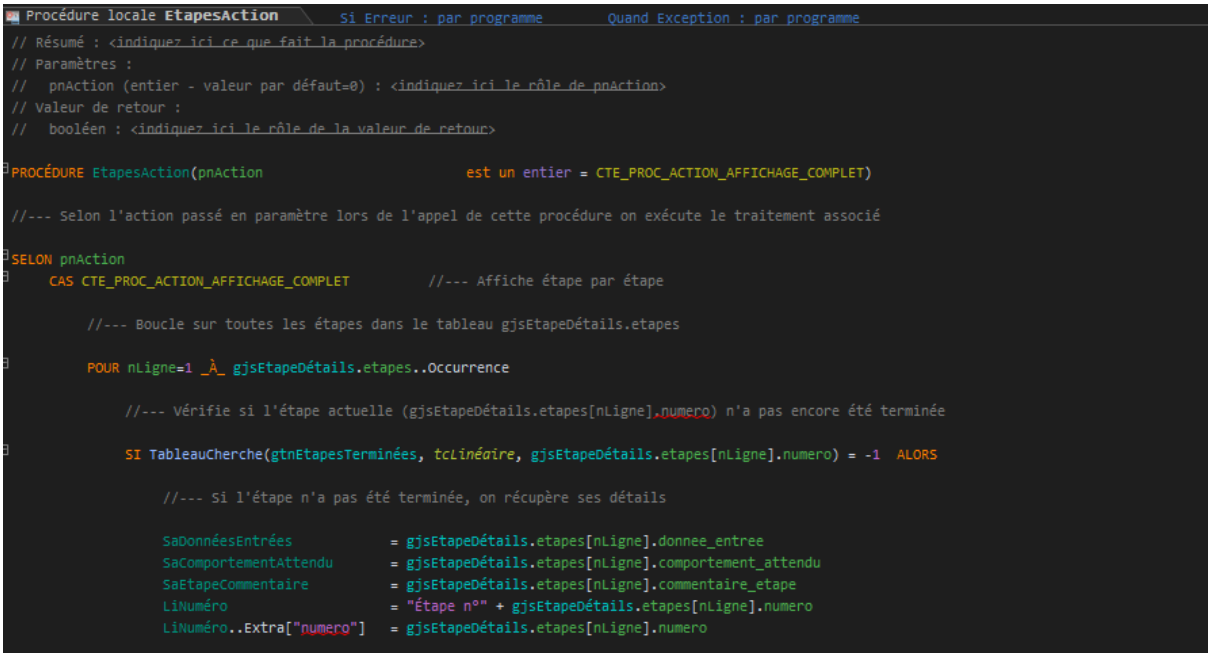
```
Procédure locale ComboFrequenceAnomalieRemplissage
// Résumé : <Indiquez ici ce que fait la procédure>
// Syntaxe :
// ComboFrequenceAnomalieRemplissage ()
//
// Paramètres :
//   Aucun
// Valeur de retour :
//   Aucune
//
// Exemple :
// <Indiquez ici un exemple d'utilisation>


PROCÉDURE ComboFrequenceAnomalieRemplissage()

ListeSupprimeTout(CbFréquenceAnomalie)

ListeAjoute(CbFréquenceAnomalie, gLien(CTE_PDT_FREQ_ANOMALIE_AUCUN) +  "<Aucun>" )
ListeAjoute(CbFréquenceAnomalie, gLien(CTE_PDT_FREQ_ANOMALIE_SYSTEMATIQUE ) + CesPlanTestAnomalieFrequenceApparitionLibellé(CTE_PDT_FREQ_ANOMALIE_SYSTEMATIQUE))
ListeAjoute(CbFréquenceAnomalie, gLien(CTE_PDT_FREQ_ANOMALIE_FREQUENT)     + CesPlanTestAnomalieFrequenceApparitionLibellé(CTE_PDT_FREQ_ANOMALIE_FREQUENT ))
ListeAjoute(CbFréquenceAnomalie, gLien(CTE_PDT_FREQ_ANOMALIE_OCCASIONNEL)  + CesPlanTestAnomalieFrequenceApparitionLibellé(CTE_PDT_FREQ_ANOMALIE_OCCASIONNEL))
ListeAjoute(CbFréquenceAnomalie, gLien(CTE_PDT_FREQ_ANOMALIE_RARE )       + CesPlanTestAnomalieFrequenceApparitionLibellé(CTE_PDT_FREQ_ANOMALIE_RARE  ))
```

```
CTE_PDT_FREQ_ANOMALIE_SYSTEMATIQUE  = 17
CTE_PDT_FREQ_ANOMALIE_FREQUENT      = 18
CTE_PDT_FREQ_ANOMALIE_OCCASIONNEL   = 19
CTE_PDT_FREQ_ANOMALIE_RARE          = 20
CTE_PDT_FREQ_ANOMALIE_AUCUN         = -1
```

```
Procédure globale CesPlanTestAnomalieFrequenceApparitionLibellé
// Procédure qui retourne le libellé de fréquence d'apparition d'une anomalie
// Paramètres :
//  pnCteFrequence (entier) : Constante CTE_PDT_FREQ_ANOMALIE_XXXXXXXXXXXXXXXXX
// Valeur de retour :
//  chaîne ANSI : Libellé
//
PROCÉDURE CesPlanTestAnomalieFrequenceApparitionLibellé(LOCAL pnCteFrequence          est un entier)

SELON pnCteFrequence
    CAS  CTE_PDT_FREQ_ANOMALIE_SYSTEMATIQUE  :  RENVOYER "Systématique"
    CAS  CTE_PDT_FREQ_ANOMALIE_FREQUENT      :  RENVOYER "Fréquent"
    CAS  CTE_PDT_FREQ_ANOMALIE_OCCASIONNEL   :  RENVOYER "Occasionnel"
    CAS  CTE_PDT_FREQ_ANOMALIE_RARE          :  RENVOYER "Rare"
    AUTRE CAS                                :  RENVOYER "Aucune erreur"
FIN
```

Nous voici maintenant dans la procédure locale EtapesAction() ou se passera généralement toutes les actions principales de la fenêtre, on commence par faire appel à pnAction qui est un paramètre de type entier correspondant à la constante action CTE_PROC_ACTION_AFFICHAGE_COMPLET, puis on dit que selon pnAction, on appelle l'action CTE_PROC_ACTION_AFFICHAGE_COMPLET et on définit cette action en disant que Pour toutes les étapes allant de la première à la dernière occurrence, on dit que SI le numéro de l'étape en cours n'est pas encore dans le tableau ALORS on l'affiche à l'écran avec toutes les informations correspondante que l'on va récupérer dans le code JSON, en partant de la 1 ère ligne jusqu'à la dernière comme vous pouvez le voir sur les 2 photos en dessous :

Pour rappel une étape qui est dans le tableau, c'est une étape qui a été validé

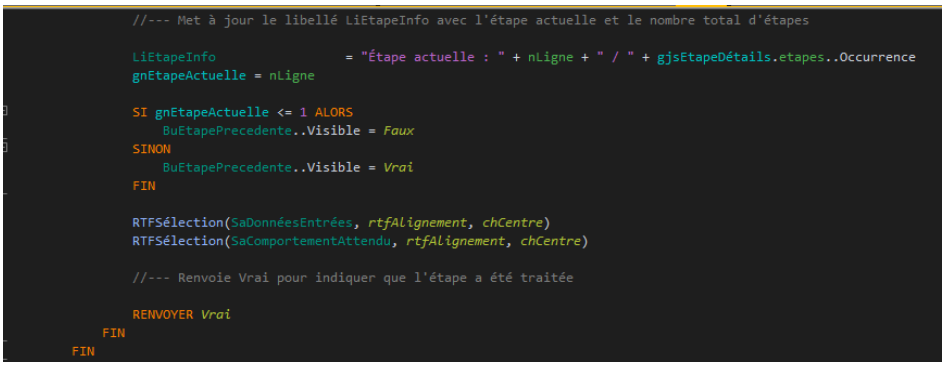


```
Procédure locale EtapesAction        Si Erreur : par programme        Quand Exception : par programme
// Résumé : <indiquez ici ce que fait la procédure>
// Paramètres :
//   pnAction (entier - valeur par défaut=0) : <indiquez ici le rôle de pnAction>
// Valeur de retour :
//   booléen : <indiquez ici le rôle de la valeur de retour>

PROCÉDURE EtapesAction(pnAction                    est un entier = CTE_PROC_ACTION_AFFICHAGE_COMPLET)

//--- Selon l'action passé en paramètre lors de l'appel de cette procédure on exécute le traitement associé

SELON pnAction
    CAS CTE_PROC_ACTION_AFFICHAGE_COMPLET         //--- Affiche étape par étape

        //--- Boucle sur toutes les étapes dans le tableau gjsEtapeDétails.etapes

        POUR nLigne=1 _À_ gjsEtapeDétails.etapes..Occurrence

            //--- Vérifie si l'étape actuelle (gjsEtapeDétails.etapes[nLigne].numero) n'a pas encore été terminée

            SI TableauCherche(gtnEtapesTerminées, tcLinéaire, gjsEtapeDétails.etapes[nLigne].numero) = -1  ALORS

                //--- Si l'étape n'a pas été terminée, on récupère ses détails

                SaDonnéesEntrées            = gjsEtapeDétails.etapes[nLigne].donnee_entree
                SaComportementAttendu       = gjsEtapeDétails.etapes[nLigne].comportement_attendu
                SaEtapeCommentaire          = gjsEtapeDétails.etapes[nLigne].commentaire_etape
                LiNuméro                    = "Étape n°" + gjsEtapeDétails.etapes[nLigne].numero
                LiNuméro..Extra["numero"]   = gjsEtapeDétails.etapes[nLigne].numero
```


```
{
  "objectif":"Test debug cr\u00e9ation",
  "commentaire":"Commentaire de test",
  "etapes":[
    {
      "numero":"1",
      "comportement_attendu":"{\\rtf1\\ansi\\ansicpg1252\\deff0\\nouicompat\\deflang1036{\\fonttbl{\\f0\\fnil\\fprq2\\fcharset0 Roboto Condensed;}}\r\n{\\colortbl ;\\red51\\green51\\blue51;}\r\n{\\*\\generator Richec
      "donnee_entree":"{\\rtf1\\ansi\\ansicpg1252\\deff0\\nouicompat\\deflang1036{\\fonttbl{\\f0\\fnil\\fprq2\\fcharset0 Roboto Condensed;}}\r\n{\\colortbl ;\\red51\\green51\\blue51;}\r\n{\\*\\generator Riched20 10.0
      "commentaire_etape":"Bonjour",
      "anomalie_etat":true,
      "anomalie_niveau":2,
      "anomalie_frequence":"18",
      "anomalie_ref":0
    },
    {
      "numero":"2",
      "comportement_attendu":"{\\rtf1\\ansi\\ansicpg1252\\deff0\\nouicompat\\deflang1036{\\fonttbl{\\f0\\fnil\\fprq2\\fcharset0 Roboto Condensed;}}\r\n{\\colortbl ;\\red51\\green51\\blue51;}\r\n{\\*\\generator Richec
      "donnee_entree":"{\\rtf1\\ansi\\ansicpg1252\\deff0\\nouicompat\\deflang1036{\\fonttbl{\\f0\\fnil\\fprq2\\fcharset0 Roboto Condensed;}}\r\n{\\colortbl ;\\red51\\green51\\blue51;}\r\n{\\*\\generator Riched20 10.0
      "commentaire_etape":"",
      "anomalie_etat":false,
      "anomalie_niveau":-1,
      "anomalie_frequence":-1,
      "anomalie_ref":-1
    },
    {
      "numero":"3",
      "comportement_attendu":"{\\rtf1\\ansi\\ansicpg1252\\deff0\\nouicompat\\deflang1036{\\fonttbl{\\f0\\fnil\\fprq2\\fcharset0 Roboto Condensed;}}\r\n{\\colortbl ;\\red51\\green51\\blue51;}\r\n{\\*\\generator Richec
      "donnee_entree":"{\\rtf1\\ansi\\ansicpg1252\\deff0\\nouicompat\\deflang1036{\\fonttbl{\\f0\\fnil\\fprq2\\fcharset0 Roboto Condensed;}}\r\n{\\colortbl ;\\red51\\green51\\blue51;}\r\n{\\*\\generator Riched20 10.0
      "commentaire_etape":"",
      "anomalie_etat":false,
      "anomalie_niveau":-1,
      "anomalie_frequence":-1,
      "anomalie_ref":-1
    },
    {
      "numero":"4",
      "comportement_attendu":"{\\rtf1\\ansi\\ansicpg1252\\deff0\\nouicompat\\deflang1036{\\fonttbl{\\f0\\fnil\\fprq2\\fcharset0 Roboto Condensed;}}\r\n{\\colortbl ;\\red51\\green51\\blue51;}\r\n{\\*\\generator Richec
      "donnee_entree":"{\\rtf1\\ansi\\ansicpg1252\\deff0\\nouicompat\\deflang1036{\\fonttbl{\\f0\\fnil\\fprq2\\fcharset0 Roboto Condensed;}}\r\n{\\colortbl ;\\red51\\green51\\blue51;}\r\n{\\*\\generator Riched20 10.0
      "commentaire_etape":"",
      "anomalie_etat":false,
      "anomalie_niveau":-1,
      "anomalie_frequence":-1,
      "anomalie_ref":-1
    }
  ]
}
```


On met à jour le libellé pour qu'il nous indique bien sur quelle étape l'on se trouve, on dit que gnEtapeActuelle correspond a nLigne => l'indice de l'étape présente dans le tableau json de gjsEtapeDétails.etapes[], on établit une condition qui dit que SI l'on se trouve à une étape inférieur ou égal à 1 ALORS on

n’affiche pas la flèche de retour en bas à gauche et SINON, on l’affiche, on utilise RTFSélection pour récupérer les éléments RTF de la fenêtre et on les aligne au centre de leur champ et on retourne Vrai pour indiquer que l’étape à bien été effectué :



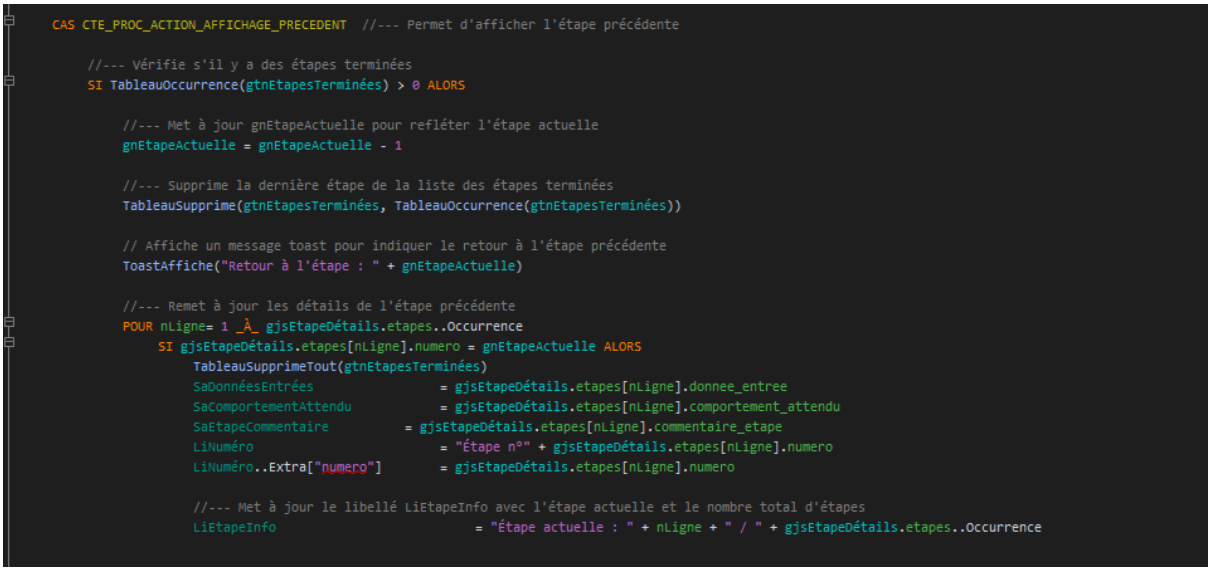
```
            //--- Met à jour le libellé LiEtapeInfo avec l'étape actuelle et le nombre total d'étapes

            LiEtapeInfo                = "Étape actuelle : " + nLigne + " / " + gjsEtapeDétails.etapes..Occurrence
            gnEtapeActuelle = nLigne

            SI gnEtapeActuelle <= 1 ALORS
                BuEtapePrecedente..Visible = Faux
            SINON
                BuEtapePrecedente..Visible = Vrai
            FIN

            RTFSélection(SaDonnéesEntrées, rtfAlignement, chCentre)
            RTFSélection(SaComportementAttendu, rtfAlignement, chCentre)

            //--- Renvoie Vrai pour indiquer que l'étape a été traitée

            RENVOYER Vrai
        FIN
    FIN
```

Après, on va vérifier si toutes les étapes d’un cas de test ont été effectuées, si oui alors on vide avec RAZ entièrement le tableau et on passe au cas de test suivant, sinon on affiche la fenêtre demandant de choisir un testeur.
On passe au cas CTE_PROC_ACTION_AFFICHE_ACTUALISATION ou l’on va réinitialiser toutes les informations du groupe GrChangementEtape grâce à RAZ, ce groupe représente toutes les informations sur l’anomalie et, à chaque étape, elles se mettent à leur valeur qu’elles avaient au départ, donc à l’initial, on effectue une condition qui dit que si le contenue de la cellule 2 de mon champ disposition est visible, alors on le rend invisible et on remet son libellé initial étant “Renseigner une anomalie”. On fait appel à la procédure ComboNiveauAnomalieRemplissage() et ComboFrequenceAnomalieRemplissage() pour remplir les différents combos, on met la valeur “<Aucun>” par défaut, on appelle dans la procédure EtapeAction le cas précédemment défini étant CTE_PROC_ACTION_AFFICHAGE_COMPLET et on fait appel, grâce à ExécuteTraitement, a la combo de la fréquence

d’anomalie se situant dans cette fenêtre pour lui ajouter les bonnes informations :

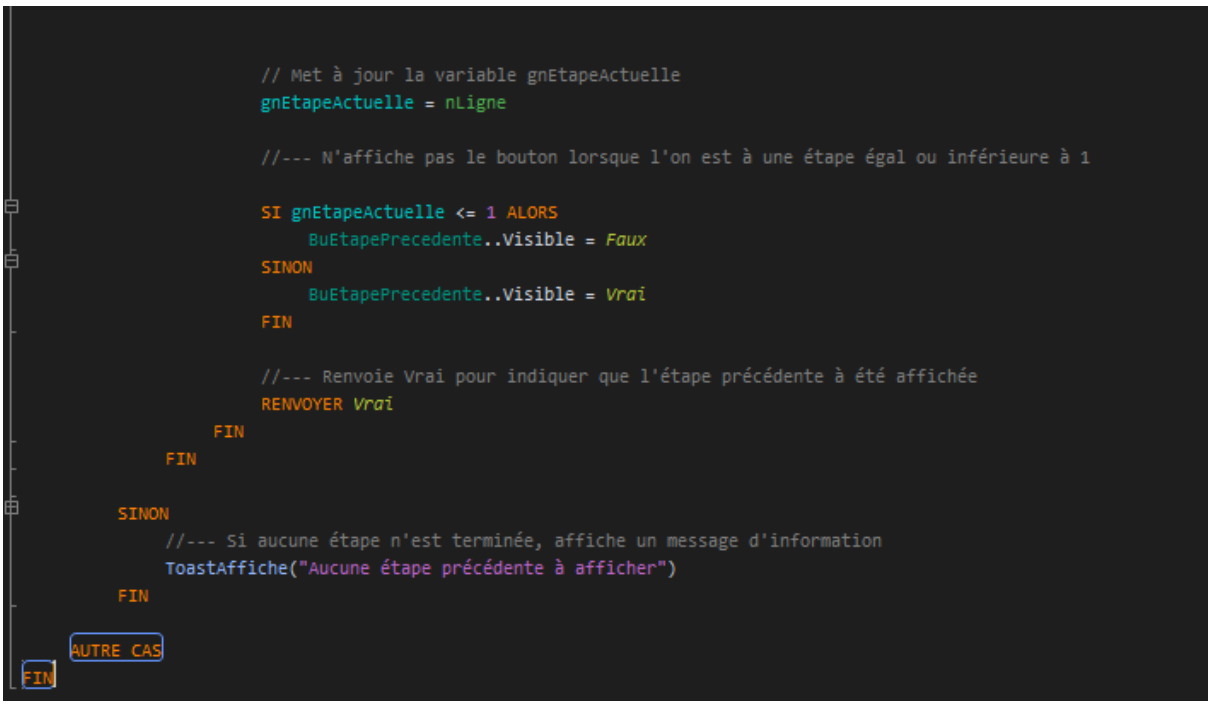
```
        //--- Toutes les étapes ont été affichées, on vide le tableau gtnEtapesTerminées

        SI FenEtat(ff_cartographie_nf525_plan_test_execution)<>Inexistant ALORS
            SI ff_cartographie_nf525_plan_test_execution.CbTesteur<>0 ALORS
                TableauSupprimeTout(gtnEtapesTerminées)
                ff_cartographie_nf525_plan_test_execution._ExecutionPlanTest(CTE_PROC_ACTION_AFFICHE_ACTUALISATION, gjsEtapeExecutionValidation)
                gjsEtapeExecutionValidation = ""
            SINON
                MessageFormat("Veuillez sélectionner un testeur", <pnFenType>:MF_ERREUR)
            FIN
        FIN

    CAS CTE_PROC_ACTION_AFFICHE_ACTUALISATION      //--- Actualise l'étape suivante avec les nouvelles données

        RAZ(GrChangementEtape)

        ComboNiveauAnomalieRemplissage()
        ComboFrequenceAnomalieRemplissage()

        SI DispContenue[2]..Visible = Vrai  ALORS
           DispContenue[2]..Visible = Faux
           BuActionEtat..Libellé        = "Renseigner une anomalie"
        FIN

        // On affiche <Aucun> par défaut dés que le bouton est appuyé

        ListeSelectPlus(CbFréquenceAnomalie, 1)

        EtapesAction(CTE_PROC_ACTION_AFFICHAGE_COMPLET)

        ExécuteTraitement(fi_cartographie_nf525_cas_test_etapes.CbFréquenceAnomalie, trtInit)
```

Passons au cas CTE_PROC_ACTION_AFFICHAGE_PRECEDENT ou je définis le code pour revenir aux étapes précédentes.
Pour cela, je fait une condition qui dit que SI, dans le tableau, il y a des étapes terminées, ALORS on va mettre à jour gnEtapeActuelle pour bien afficher l’étape actuelle, on va supprimer la dernière étape du tableau des étapes terminées et on affiche un petit message sous forme de ToastAffiche pour dire que l’on est bien de retour a cette étape.
On établit ensuite une boucle Pour qui va remettre à jour les détails de l’étape précédente, c’est quasiment la même chose que lorsque l’on passe à l’étape suivante sauf que l’on va se baser sur gnEtapeActuelle et on va vérifier si l’étape que l’on veut afficher est déjà dans le tableau et si oui, alors on va supprimer toutes les étapes du tableau sur lesquelles on vient de passer afin de ne pas les sauter quand on va vouloir passer aux étapes suivantes et on met à jour le libellé avec l’étape actuelle que l’on affiche :



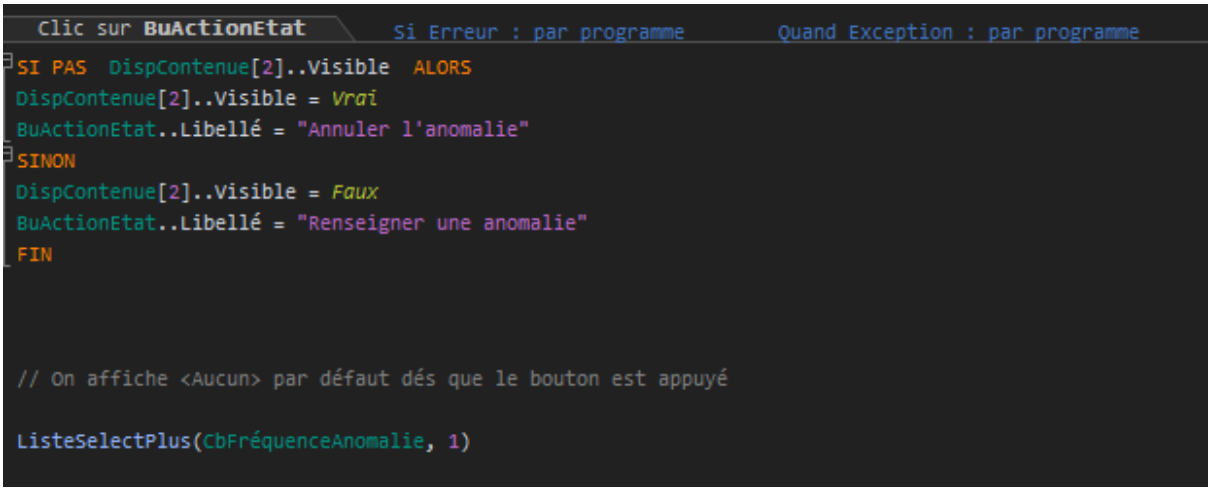
```
CAS CTE_PROC_ACTION_AFFICHAGE_PRECEDENT  //--- Permet d'afficher l'étape précédente

    //--- Vérifie s'il y a des étapes terminées
    SI TableauOccurrence(gtnEtapesTerminées) > 0 ALORS

        //--- Met à jour gnEtapeActuelle pour refléter l'étape actuelle
        gnEtapeActuelle = gnEtapeActuelle - 1

        //--- Supprime la dernière étape de la liste des étapes terminées
        TableauSupprime(gtnEtapesTerminées, TableauOccurrence(gtnEtapesTerminées))

        // Affiche un message toast pour indiquer le retour à l'étape précédente
        ToastAffiche("Retour à l'étape : " + gnEtapeActuelle)

        //--- Remet à jour les détails de l'étape précédente
        POUR nLigne= 1 _À_ gjsEtapeDétails.etapes..Occurrence
            SI gjsEtapeDétails.etapes[nLigne].numero = gnEtapeActuelle ALORS
                TableauSupprimeTout(gtnEtapesTerminées)
                SaDonnéesEntrées                = gjsEtapeDétails.etapes[nLigne].donnee_entree
                SaComportementAttendu           = gjsEtapeDétails.etapes[nLigne].comportement_attendu
                SaEtapeCommentaire          = gjsEtapeDétails.etapes[nLigne].commentaire_etape
                LiNuméro                        = "Étape n°" + gjsEtapeDétails.etapes[nLigne].numero
                LiNuméro..Extra["numero"]       = gjsEtapeDétails.etapes[nLigne].numero

                //--- Met à jour le libellé LiEtapeInfo avec l'étape actuelle et le nombre total d'étapes
                LiEtapeInfo                         = "Étape actuelle : " + nLigne + " / " + gjsEtapeDétails.etapes..Occurrence
```

On met à jour gnEtapeActuelle en lui donnant la valeur de nLigne qui représente en clair la valeur du numéro de l'étape sur laquelle on se trouve.
On établit ensuite une petite condition qui dit que SI gnEtapeActuelle (le numéro de l'étape) à une valeur inférieure ou égale à 1, ALORS on n'affiche pas le bouton pour revenir à l'étape précédente et sinon, on l'affiche dans tous les autres cas.
On renvoie Vrai pour dire que l'étape à bien été affichée et SINON, si on ne trouve aucune étape à afficher, alors on va afficher un message disant qu'il n'y a aucune autre étape précédente à afficher et cette action est enfin terminée.
Et c'est enfin la fin du code principale de la fenêtre interne :

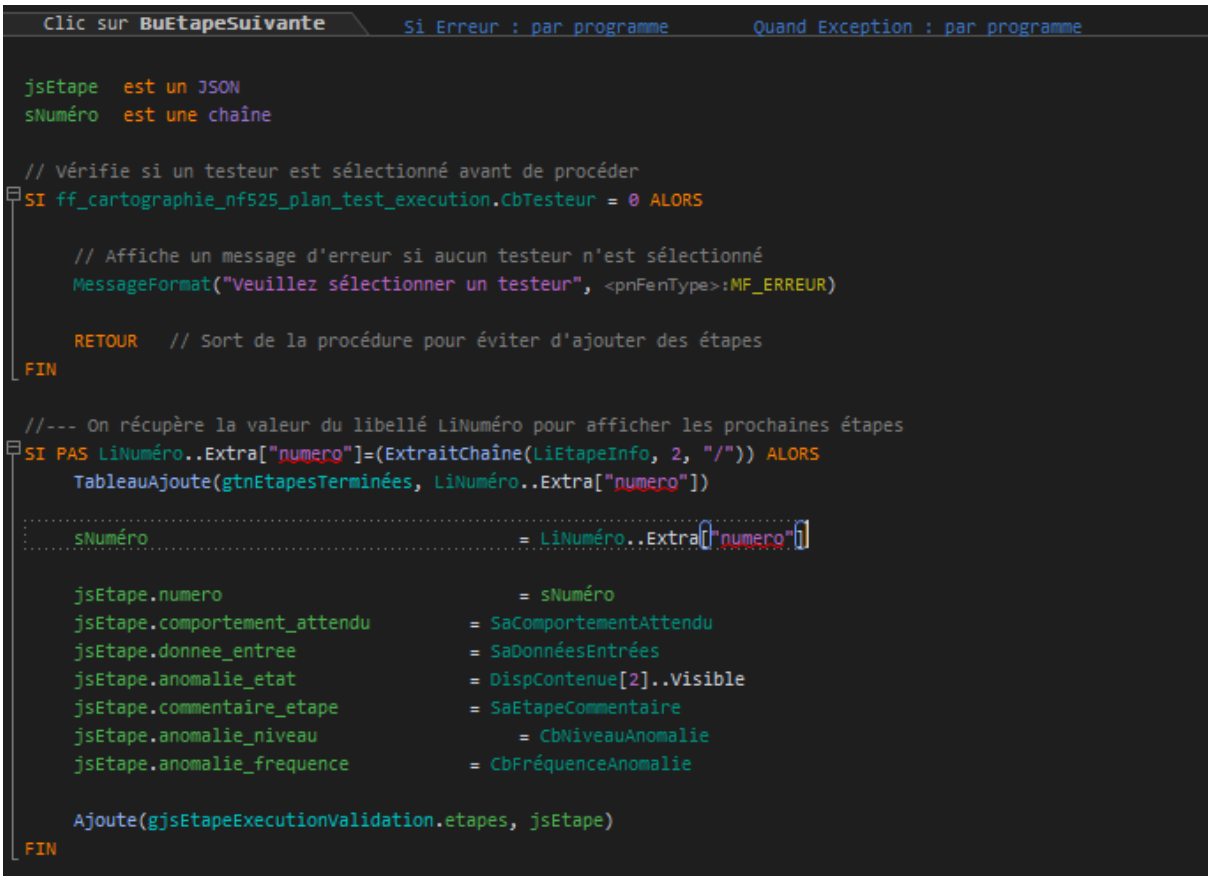


```
                    // Met à jour la variable gnEtapeActuelle
                    gnEtapeActuelle = nLigne

                    //--- N'affiche pas le bouton lorsque l'on est à une étape égal ou inférieure à 1

                    SI gnEtapeActuelle <= 1 ALORS
                        BuEtapePrecedente..Visible = Faux
                    SINON
                        BuEtapePrecedente..Visible = Vrai
                    FIN

                    //--- Renvoie Vrai pour indiquer que l'étape précédente à été affichée
                    RENVOYER Vrai
                FIN
            FIN

        SINON
            //--- Si aucune étape n'est terminée, affiche un message d'information
            ToastAffiche("Aucune étape précédente à afficher")
        FIN

    AUTRE CAS
FIN
```

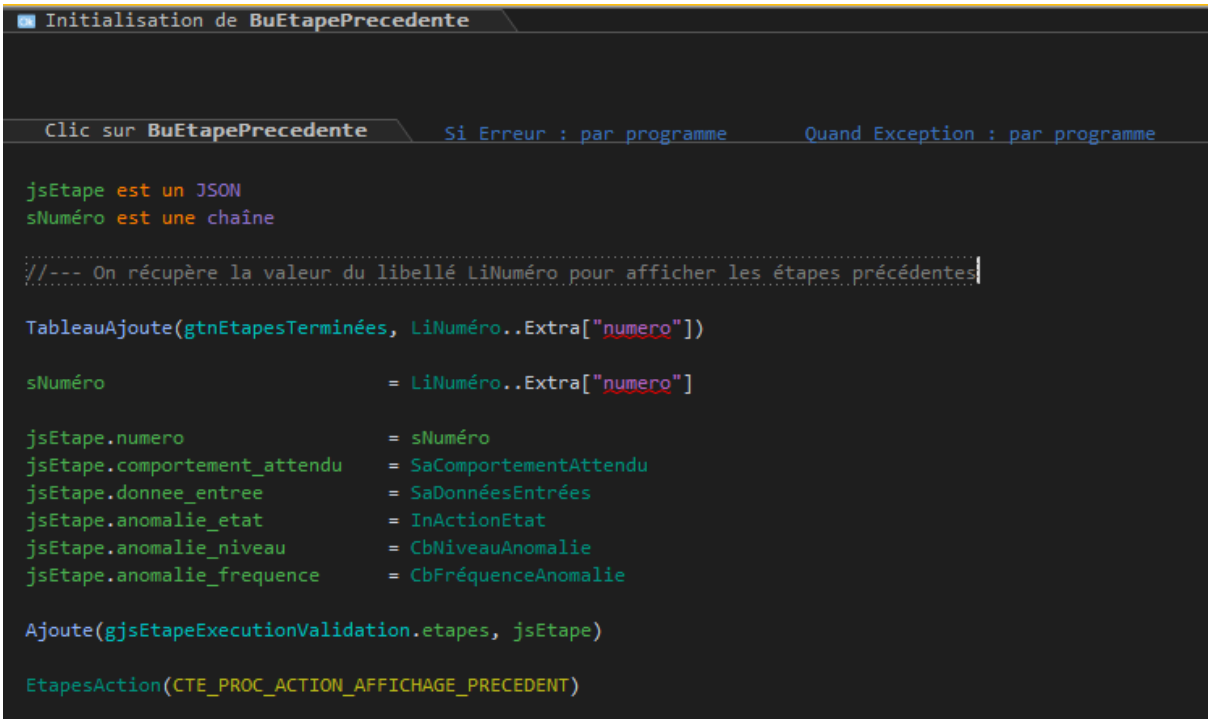
Maintenant, on est dans le code du bouton BuActionEtat qui va afficher toutes les informations sur l'anomalie en fonction de la condition étant que, SI la cellule 2 de mon champ disposition n'est pas visible, ALORS on la rend visible et on met à jour le libellé du bouton et sinon,on met le contenue invisible et on met à jour le libellé du bouton.
On affiche de nouveau <Aucun> par défaut dès que le bouton est appuyé sinon, dès que l'on va changer d'étape ou réappuyer sur le bouton , il ne sera plus affiché directement :



On se retrouve maintenant dans le bouton pour afficher les éléments de la prochaine étape et on définit 2 variables, jsEtape étant un JSON qui va représenter les différentes informations des étapes et sNuméro qui est une chaîne qui va récupérer la valeur du libellé Linuméro.
Après avoir définis ces variables, on effectue une condition disant que SI un testeur n'a pas été choisi ALORS on affiche le message d'erreur approprié et on utilise un RETOUR permettant de sortir de la procédure pour éviter d'ajouter des étapes en trop.
Si la valeur de LiNuméro n'est pas égal au deuxième segment de chaîne extrait de LiEtapeInfo alors on ajoute au tableau d'entier la valeur du libellé Linuméro de l'étape ou l'on était pour pouvoir, par la suite, afficher les prochains étapes, sNuméro va donc récupérer la valeur du libellé, jsEtape va donc récupérer toutes les valeurs des différents champs de la fenêtre pour les ajouter à gjsEtapeExecutionValidation qui représente le JSON du code de la fenêtre interne et on fait finalement appel à la procédure locale EtapesAction avec en paramètre l'action CTE_PROC_ACTION_AFFICHE_ACTUALISATION :



Et pour finir, dans le code du bouton permettant de passer à l'étape précédente, ce sera strictement le même code mais il y aura juste une petite subtilité qui est, à la dernière ligne, lorsque l'on fait appel à la procédure locale EtapesAction, on ne va pas mettre en paramètre l'action CTE_PROC_ACTION_AFFICHE_ACTUALISATION mais plutôt l'action CTE_PROC_ACTION_AFFICHAGE_PRECEDENT :



En résumé, le code est plutôt bien construit, il est clair, net et précis pour que normalement n'importe qui puisse le comprendre même s'il est toujours possible de faire de l'optimisation ou des modifications afin d'améliorer encore plus le code.

Ce que j'ai fait sur le projet Windev (Niveau des bugs du Codage)

Au niveau des problèmes que l'on a eu vis à vis du code, il y en a eu plusieurs comme le fait que, lorsque je voulais passer au cas de test suivant, le problème étant que dans mon premier cas de test, j'avais 4 étapes différentes, et lorsque je voulais

passer au prochain cas de test, cela me le sautait car il ne possédait qu'une seule étape et le code voulait a tout pris afficher une étape 5.
Ce qu'il faisait, c'est qu'il cherchait le prochain cas de test mais qui contenait une étape 5 et il l'affiche à partir de cette étape là.
Le problème était que, quand je finissais un cas de test, je ne supprimait pas le contenue de ma table, du coup il a fallu que je rajoute une ligne de code étant :

```
TableauSupprimeTout(gtnEtapesTerminées)
```

Cette ligne me permet de supprimer tout le contenu actuel de la table gtnEtapesTerminées, donc il peut se réinitialiser à la fin de chaque étape.

Une autre erreur que l'on a eu au début du projet est le fait que lorsque l'on voulait afficher les étapes d'un cas de test, on commençait toujours à afficher la dernière étape du premier cas de test car dans le code je n'avais pas créé de boucle me permettant de d'afficher étape par étapes les différents éléments ce qu'il fait qu'il commençait par la dernière étapes et non par la première.
Pour se faire, on a donc due créer une boucle qui permet d'afficher les étapes unes par unes et cette boucle, je vous l'ai déjà montré juste avant et c'est celle de la procédure EtapesAction :

```
PROCÉDURE EtapesAction(pnAction                est un entier = CTE_PROC_ACTION_AFFICHAGE_COMPLET)

 //--- Selon l'action passé en paramètre lors de l'appel de cette procédure on exécute le traitement associé

SELON pnAction
    CAS CTE_PROC_ACTION_AFFICHAGE_COMPLET       //--- Affiche étape par étape

        //--- Boucle sur toutes les étapes dans le tableau gjsEtapeDétails.etapes

        POUR nLigne=1 _À_ gjsEtapeDétails.etapes..Occurrence

            //--- Vérifie si l'étape actuelle (gjsEtapeDétails.etapes[nLigne].numero) n'a pas encore été terminée

            SI TableauCherche(gtnEtapesTerminées, tcLinéaire, gjsEtapeDétails.etapes[nLigne].numero) = -1  ALORS

                //--- Si l'étape n'a pas été terminée, on récupère ses détails

                SaDonnéesEntrées            = gjsEtapeDétails.etapes[nLigne].donnee_entree
                SaComportementAttendu       = gjsEtapeDétails.etapes[nLigne].comportement_attendu
                LiNuméro                    = "Étape n°" + gjsEtapeDétails.etapes[nLigne].numero
                LiNuméro..Extra["numero"]   = gjsEtapeDétails.etapes[nLigne].numero

                //--- Met à jour le libellé LiEtapeInfo avec l'étape actuelle et le nombre total d'étapes

                LiEtapeInfo                 = "Étape actuelle : " + nLigne + " / " + gjsEtapeDétails.etapes..Occurrence
                gnEtapeActuelle = nLigne
```

gjsEtapeDétails : JSON

Encore une autre erreur que j'ai eu est le fait que, quand on arrivait à l'étape 4 du premier cas de test et que l'on voulait passer au cas de test suivant, si on avait pas choisi de testeurs au préalable, alors, quand on appuyait sur le bouton pour passer à la prochaine étape, alors cela nous renvoyait à la première étape du premier cas de test et il fallait revenir jusqu'à l'étape 3 pour passer au prochain cas de test.
Pour corriger cela, on à utilisé une condition qui vérifie si la fenêtre ff_cartographie_nf525_plan_test_execution existe et si un testeur est sélectionné, puis, elle vide le tableau des étapes terminées et exécute un nouveau plan de test, sinon il affiche un message d'erreur demandant de sélectionner un testeur :

```
SI FenEtat(ff_cartographie_nf525_plan_test_execution)<>Inexistant ALORS
    SI ff_cartographie_nf525_plan_test_execution.CbTesteur<>0 ALORS
        TableauSupprimeTout(gtnEtapesTerminées)
        ff_cartographie_nf525_plan_test_execution._ExecutionPlanTest(CTE_PROC_ACTION_AFFICHE_ACTUALISATION, gjsEtapeExecutionValidation)
    SINON
        MessageFormat("Veuillez sélectionner un testeur", <pnFenType>:MF_ERREUR)
    FIN
FIN
```

Un autre problème que j’ai rencontré au tout début du projet est le fait que je n’arrivais pas à récupérer les données de la fenêtre principale pour les afficher dans la fenêtre interne car je ne savais pas comment fonctionnait exactement le principe des JSON, comment les utiliser dans le code et comment faire des liens entre deux fenêtres différentes.

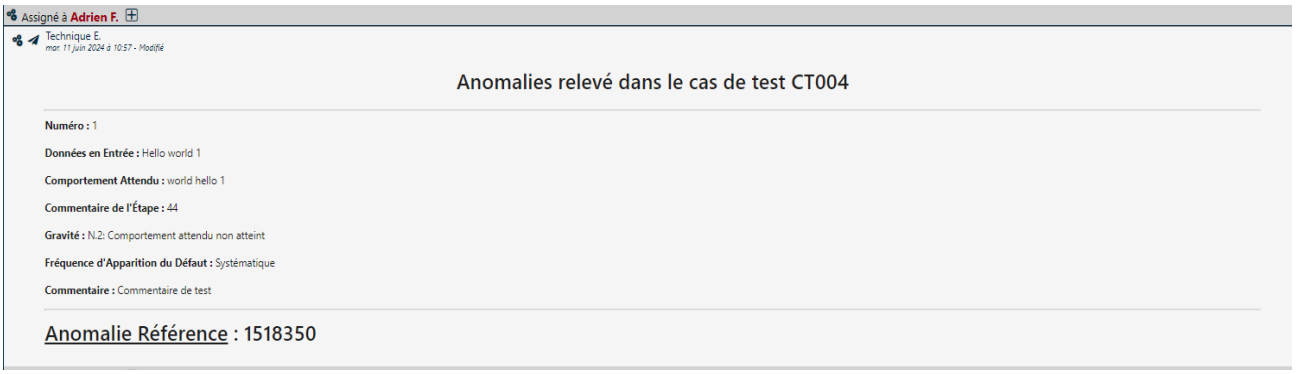
J’ai eu un souci plutôt récemment qui était que, lorsque j’avançais aux prochaines étapes et que par la suite, je voulais revenir aux étapes précédentes (si par exemple, j’avançais de l’étape 1 à l’étape 4 et que, juste après, je retourne aux étapes précédentes jusqu'à la première, si je voulais passer aux prochaines étapes, alors cela allait sauter l’étape 2 et 3 pour m’afficher directement l’étape 4), ce qu’il se passait, c’est que quand je retournais aux étapes précédentes, je ne supprimait pas dans mon tableau la valeur du numéro des étapes entre la 1 et la 4 si on reste dans ce cas, le code ne supprimait uniquement la dernière valeur de l'occurrence du tableau qui était 4, j’ai donc défini dans ma déclaration globale la variable gnEtapeActuelle que j’utilise dans l’action CTE_PROC_ACTION_AFFICHAGE_PRECEDENT comme ça je pourrais donc repasser par toutes les étapes :

```
gnEtapeActuelle                    est un entier
```

```
CAS CTE_PROC_ACTION_AFFICHAGE_PRECEDENT  //--- Permet d'afficher l'étape précédente

    //--- Vérifie s'il y a des étapes terminées
    SI TableauOccurrence(gtnEtapesTerminées) > 0 ALORS

        //--- Met à jour gnEtapeActuelle pour refléter l'étape actuelle
        gnEtapeActuelle = gnEtapeActuelle - 1

        //--- Supprime la dernière étape de la liste des étapes terminées
        TableauSupprime(gtnEtapesTerminées, TableauOccurrence(gtnEtapesTerminées))

        // Affiche un message toast pour indiquer le retour à l'étape précédente
        ToastAffiche("Retour à l'étape : " + gnEtapeActuelle)

        //--- Remet à jour les détails de l'étape précédente
        POUR nLigne= 1 _À_ gjsEtapeDétails.etapes..Occurrence
            SI gjsEtapeDétails.etapes[nLigne].numero = gnEtapeActuelle ALORS
                TableauSupprimeTout(gtnEtapesTerminées)
                SaDonnéesEntrées                = gjsEtapeDétails.etapes[nLigne].donnee_entree
                SaComportementAttendu           = gjsEtapeDétails.etapes[nLigne].comportement_attendu
                LiNuméro                        = "Étape n°" + gjsEtapeDétails.etapes[nLigne].numero
                LiNuméro..Extra["numero"]       = gjsEtapeDétails.etapes[nLigne].numero

                //--- Met à jour le libellé LiEtapeInfo avec l'étape actuelle et le nombre total d'étapes
                LiEtapeInfo                     = "Étape actuelle : " + nLigne + " / " + gjsEtapeDétails.etapes..Occurrence
```

Ce que j’ai fait sur le projet Windev (Seconde partie du projet)

Pour la seconde partie du projet, je me suis occupé de réaliser la création d’un ticket sur l’application web preprod d’Euresto qui ressemble donc à ça :



et pour ce faire, j’ai dû faire des modifications dans le code de la fenêtre principale, je vais juste vous montrer les plus grosses modifications que j’ai faites.

Les plus grosses modifications viennent de ces bouts de code ci-dessous.
Dans le premier bout de code, j'initialise les différents éléments que je récupère dans le JSON et qui se trouvent dans le ticket comme le nom du cas de test, le numéro de l'étape ou l'on se trouve, les infos principales de l'étape et de l'anomalie, j'utilise nLigne pour passer par chaque étape même si il n'y a pas d'anomalie. Détail important, je crée un ticket uniquement si une anomalie se trouve dans le cas de test sinon je n'en crée pas :

```
sMessage = "<h1><center>Anomalies relevé dans le cas de test " + NF525_Plan_Test_Détail.Libellé + "</center></h1>"

//--- Boucle sur toutes les étapes du JSON

POUR nLigne=1 _A_ jsPlanTestExecution.etapes..Occurrence

    nIdTache  est un entier

    SI jsPlanTestExecution.etapes[nLigne].anomalie_niveau=-1 ALORS CONTINUER

    //--- Pas d'anomalie alors on continue la boucle et on execute pas le code qui suit

    sMessage += "<hr>" +
    "<p><strong>Numéro :</strong> " + jsPlanTestExecution.etapes[nLigne].numero + "</p>" +
    "<p><strong>Données en Entrée :</strong> " + RTFVersTexte(jsPlanTestExecution.etapes[nLigne].donnee_entree) + "</p>" +
    "<p><strong>Comportement Attendu :</strong> " + RTFVersTexte(jsPlanTestExecution.etapes[nLigne].comportement_attendu) + "</p>" +
    "<p><strong>Commentaire de l'Etape :</strong> " + jsPlanTestExecution.etapes[nLigne].commentaire_etape + "</p>" +
    "<p><strong>Gravité :</strong> " + CesPlanTestAnomalieNiveauLibelle(jsPlanTestExecution.etapes[nLigne].anomalie_niveau) + "</p>" +
    "<p><strong>Fréquence d'Apparition du Défaut :</strong> " + CesPlanTestAnomalieFrequenceApparitionLibellé(jsPlanTestExecution.etapes[nLigne].anomalie_frequence) + "</p>" +
    "<p><strong>Commentaire :</strong> " + NF525_Plan_Test_Journal.Commentaire + "</p>" +
    "<hr>";

    //--- Valorisation du json, ajout à l'étape de la référence de l'anomalie (identifiant de la tâche créé sur le CRM)
    //nIdTache =  CesPlanTestCreationTacheAPI(pnIdPlanTestInfo, sMessage)
    jsPlanTestExecution.etapes[nLigne].anomalie_ref = nIdTache

    //CesPlanTestMajTacheApi(nIdTache, sMessage)

    //--- Valorisation de la rubrique du fichier de données en y affectant le json modifié

    NF525_Plan_Test_Journal_Détail.infoJson = jsPlanTestExecution

    //--- On modifie l'enregistrement lu précédemment

    HModifie(NF525_Plan_Test_Journal_Détail)

    //--- RAZ

    sMessage = ""
FIN
```

Le deuxième bout de code me permet de récupérer les différentes informations se trouvant dans le JSON que je vais afficher sur le ticket et si il n'y a aucune anomalie, alors j'affiche - 1 dans les différents champs du code JSON.
Le problème était que, lorsque je crée le ticket, celui-ci ne m'affichait que le titre et pas le reste, il a donc fallu que j'inverse la position de ces 2 codes afin que je récupère en premier les

informations du JSON et que par la suite, je puisse afficher les différents éléments du ticket :

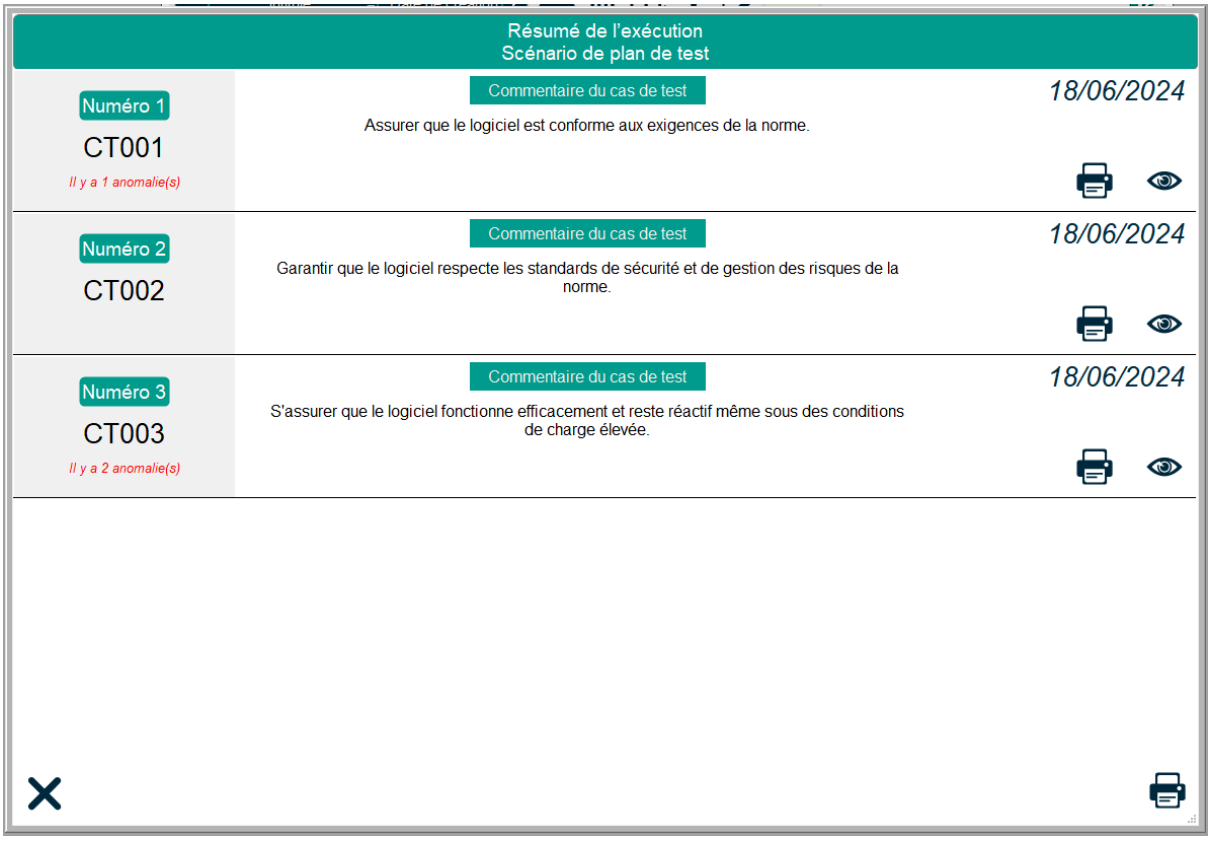
```
jsPlanTestExecution.objectif    = SaExecEtapeObjectif
jsPlanTestExecution.commentaire = SaCommentaire

POUR nLigne=1 _À_ pjsEtapesExecution.etapes..Occurrence
    jsCasTest.numero              = pjsEtapesExecution.etapes[nLigne].numero
    jsCasTest.comportement_attendu = pjsEtapesExecution.etapes[nLigne].comportement_attendu
    jsCasTest.donnee_entree       = pjsEtapesExecution.etapes[nLigne].donnee_entree
    jsCasTest.commentaire_etape   = pjsEtapesExecution.etapes[nLigne].commentaire_etape
    jsCasTest.anomalie_etat       = pjsEtapesExecution.etapes[nLigne].anomalie_etat

    //--- Vérifie si anomalie présente à l'étape en cours

    SI PAS pjsEtapesExecution.etapes[nLigne].anomalie_etat ALORS

        //--- Pas d'anomalies

        jsCasTest.anomalie_niveau      = -1
        jsCasTest.anomalie_frequence   = -1
        jsCasTest.anomalie_ref         = -1
    SINON

        //--- Anomalie présente

        jsCasTest.anomalie_niveau    = pjsEtapesExecution.etapes[nLigne].anomalie_niveau
        jsCasTest.anomalie_frequence = pjsEtapesExecution.etapes[nLigne].anomalie_frequence

    FIN

    Ajoute(jsPlanTestExecution.etapes, jsCasTest)
FIN
```

Ce que j’ai fait sur le projet Windev (Troisième partie du projet)

Pour la troisième partie du projet, ma tâche a consisté à refaire l’interface du résumé des exécutions qui était assez brouillon, il y avait 2 tableaux avec, pour le premier, les différents cas de test du plan de test et pour le second, les différentes étapes du cas de test sélectionnés avec toutes ses informations.
J’ai donc réorganisé cela en 2 fenêtres différentes.

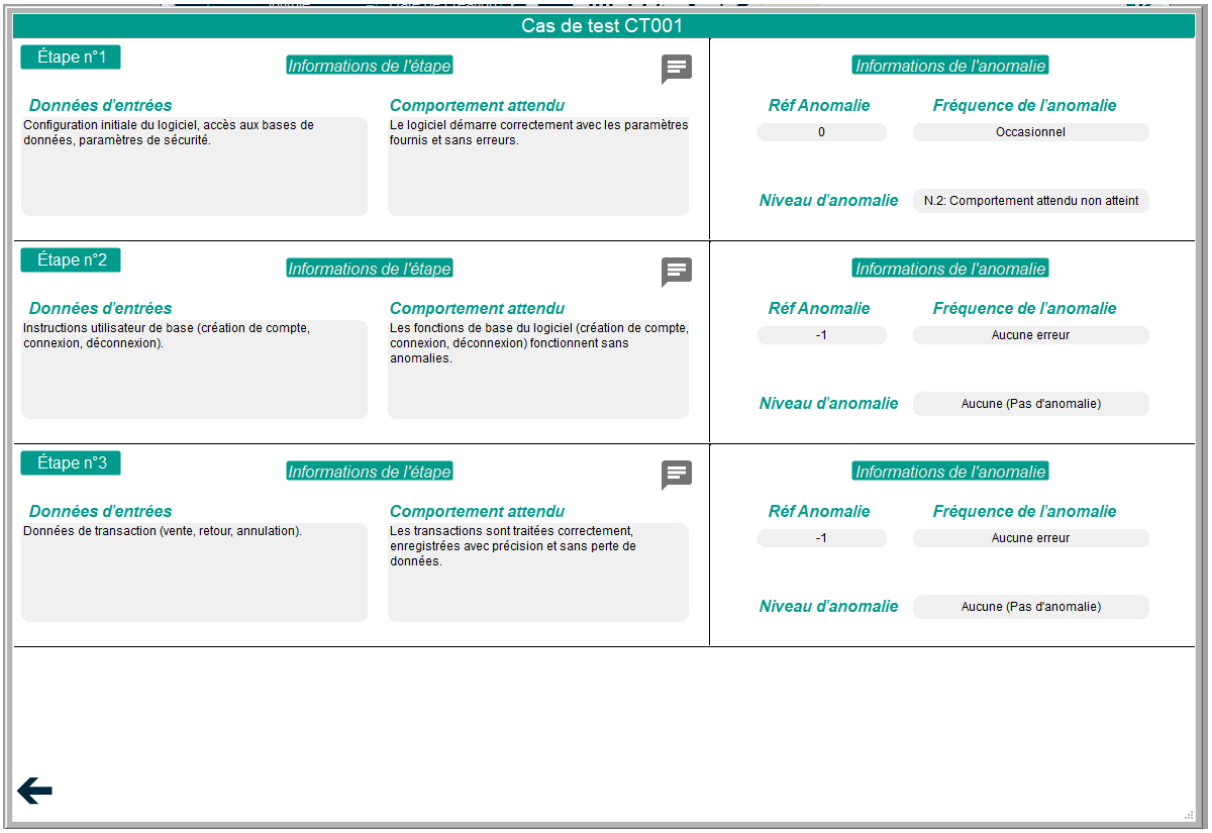
Sur la première fenêtre, on peut donc retrouver tout les cas de test se trouvant dans le plan de test actuel avec, sur le côté gauche, le numéro du cas de test avec son nom et on a un petit message qui s’affiche juste en dessous du nom qui indique combiens d’anomalies se trouvent dans le cas de test.

Sur la partie centrale, on peut retrouver le commentaire qui est propre à chaque cas de test et sur le côté droit, on peut voir la date de quand le cas de test à été effectué, un bouton impression qui permet d'imprimer au format PDF ou autre le cas de test sélectionné et on a un bouton oeil qui permet de visualiser le contenue du cas de test sélectionné.
Et en bas de la page, vous pouvez voir, un bouton croix qui permet de fermer cette page et un autre bouton impression qui permet d'imprimer le plan de test avec tous les cas de test qu'il contient :



Lorsque l'on appuie sur le bouton en forme d'œil, nous tombons donc sur la deuxième fenêtre ou l'on peut voir de multiples choses.
Tout d'abord, sur le côté gauche, les différentes informations de l'étape avec le numéro de l'étape en haut à gauche, les

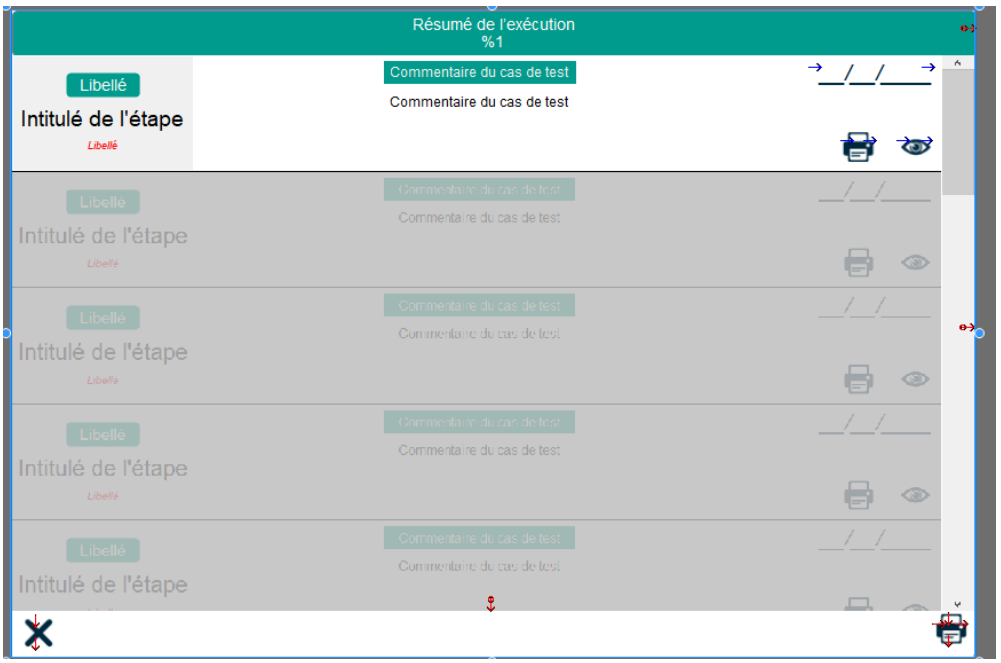
données d’entrées et le comportement attendu au centre et sur le côté droit de la partie gauche, on peut voir, un bouton en forme de message qui permet d’afficher le commentaire de l’étape sélectionné.
Sur le côté droit, on peut retrouver les différentes informations sur l’anomalie de l’étape si il y en a une ou non. Si la réf anomalie est a -1, alors cela veut dire qu’il n’y a pas d’anomalie, mais, si elle est a 0, cela veut dire qu’il y en a une (normalement, elle affiche la vraie réf anomalie mais là elle met 0 car j'ai stoppé la création de ticket). On a aussi la fréquence de l’anomalie et le niveau de l’anomalie qui s’affiche :



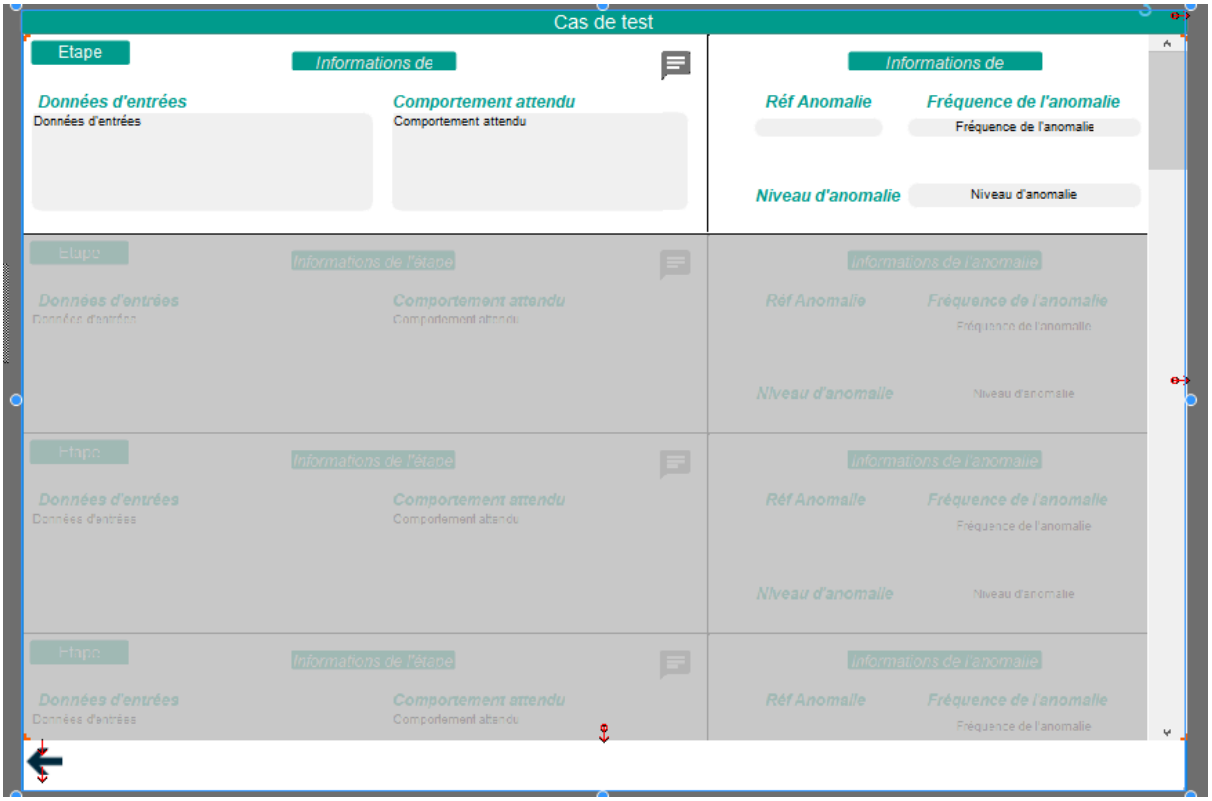
Quand on appuie sur le bouton pour voir le commentaire, on tombe sur ça, un fichier texte ou le commentaire est affiché :

Ce que j'ai fait sur le projet Windev (Codage de la Troisième partie du projet)

Au niveau du codage, la première fenêtre ressemble à ça.
On peut donc voir des libellés pour le numéro de cas de test étant LiNuméroCasTest, pour le nom du cas de test, ce sera LiIntituléEtape, pour le nombre d'anomalie LiNombreAnomalie, le petit titre en vert représente LiTitreCommentaireCasTest, le commentaire du cas de test correspond à LiCommentaireCasTest, la date LiDate et les 2 boutons, imprimer BuImprimer et l'oeil BuVisualiser.
Tout cela à été mis dans une zone répétée afin de pouvoir répéter ces champs autant que possible tant que tous les cas de test du plan de test ne sont pas affichés.
Et on a le bouton pour fermer la page BuExecValidationResumer et le bouton pour tout imprimer qui est BuImprimeTout :



La seconde ressemble donc à ça.
On a un libellé pour le numéro de l'étape LiNuméroEtape, le titre des infos de l'étape LiTitreInfosEtape, le titre Données d'entrées LiTitreDonnéesEntrées, le titre Comportement attendu LiTitreComportementAttendu, les données d'entrées LiDonnéesEntrées, le comportement attendu LiComportementAttendu, le bouton pour le commentaire BuCommentaire, le titre des infos de l'anomalie LiTitreInfosAnomalie, le titre de la réf de l'anomalie LiTitreRefAnomalie,le titre de la fréquence de l'anomalie LiTitreFrequenceAnomalie, le titre du niveau de l'anomalie LiTitreNiveauAnomalie, la réf de l'anomalie LiRefAnomalie, la fréquence de l'anomalie LiFrequenceAnomalie et le niveau de l'anomalie LiNiveauAnomalie.
Tout cela se trouve aussi dans une zone répétée afin de répéter ces champs autant que possible tant que toutes les étapes du cas de test n'ont pas été affichés :



Sur les 2 prochaines photos, vous pouvez donc voir le code réalisé pour la première fenêtre qui affiche dans les bons champs les différentes informations que l'on récupère depuis le JSON, les attribut ATT représente les différents champs de la zone répétée :

```
CAS CTE_PROC_ACTION_LISTE                                   //--- affichage résumé

    //--- Gestion fenêtre et champs spécifique pour le titre

    LiTitreRésuméExecPlanTest.Note = LiTitreRésuméExecPlanTest
    LiTitreRésuméExecPlanTest      = ChaineConstruit(LiTitreRésuméExecPlanTest.Note, NF525_Plan_Test_Infos.Libellé)

    MaFenêtre..Plan  = 2
    MaFenêtre..Titre = "Résumé de l'exécution du plan de test"

    RePlanTestExecutionDétails.pnIdJournal = gnIdJournal

    SI PAS SysExécuteRequete("RePlanTestExecutionDétails") ALORS RENVOYER Faux

    ZoneRépétéeSupprimeTout(ZoneRépétéeCasTest)

    POUR TOUT RePlanTestExecutionDétails
        ZoneRépétéeAjouteLigne(ZoneRépétéeCasTest)

        nLigne                          = TableAjouteLigne(TaResuméCasTest)

        ATT_IdJournal[nLigne]           = RePlanTestExecutionDétails.IDNF525_Plan_Test_Journal
        ATT_Date[nLigne]                = RePlanTestExecutionDétails.Horodatage
        ATT_IdJournalDétails[nLigne]    = RePlanTestExecutionDétails.IDNF525_Plan_Test_Détail


        jsPlanTestExecution             = RePlanTestExecutionDétails.infoJson

        ATT_CommentaireCasTest[nLigne]  = jsPlanTestExecution.commentaire

        SI PAS HLitRecherche(NF525_Plan_Test_Détail, IDNF525_Plan_Test_Détail, RePlanTestExecutionDétails.IDNF525_Plan_Test_Détail) ALORS CONTINUER
```

```
            ATT_IntituléEtape[nLigne]   =  NF525_Plan_Test_Détail.Libellé
            ATT_NumCasTest[nLigne]      = "Numéro " + NF525_Plan_Test_Détail.Ordre

            // Initialisation du nombre d'anomalies à 0
            LiNombreAnomalie = ""

            // Parcourez les étapes du cas de test pour compter les anomalies
            POUR nLigneJson=1 _À_ jsPlanTestExecution.etapes..Occurrence
                SI jsPlanTestExecution.etapes[nLigneJson].anomalie_niveau<>-1 ALORS
                    nTotalAnomalies += 1
                FIN
            FIN

            // Affichez le total des anomalies pour ce cas de test
            SI nTotalAnomalies > 0 ALORS
                ATT_NombreAnomalie[nLigne]          = "Il y a " + nTotalAnomalies + " anomalie(s)"
                ATT_NombreAnomalie[nLigne]..Visible = Vrai
            SINON
                ATT_NombreAnomalie[nLigne]..Visible = Faux
            FIN

            // Réinitialisez le compteur pour le prochain cas de test
            nTotalAnomalies = 0
    FIN

        _ExecutionPlanTest(CTE_PROC_ACTION_LIGNE_SELECTION)
```

Et sur cette photo, vous pouvez voir le code de la seconde fenêtre qui fait un peu la même chose que le premier code mais avec les informations de l'étape :

```
    CAS CTE_PROC_ACTION_LIGNE_SELECTION                         //--- Sélection d'un cas de test dans la zone répétée + affichage des étapes

        nLigne = ZoneRépétéeSelect(ZoneRépétéeCasTest)
        SI PAS nLigne<>-1 ALORS RENVOYER Faux

        //--- Boucle sur les étapes du cas de test sélectionné

        SI PAS HLitRecherche(NF525_Plan_Test_Journal_Détail, NF525_Plan_Test_Journal_Détail.IDNF525_Plan_Test_Détail_IDplan_test_journal, HConstruitValClé(NF525_Pla

        jsPlanTestExecution = NF525_Plan_Test_Journal_Détail.infoJson

        ZoneRépétéeSupprimeTout(ZrEtapes)

        POUR nLigneJson=1 _À_ jsPlanTestExecution.etapes..Occurrence

            nLigne =ZoneRépétéeAjouteLigne(ZrEtapes)

            ATT_NumeroEtape[nLigne]                 = " Étape n°" + jsPlanTestExecution.etapes[nLigneJson].numero
            ATT_DonnéesEntrées[nLigne]              = RTFVersTexte(jsPlanTestExecution.etapes[nLigneJson].donnee_entree)
            ATT_ComportementAttendu[nLigne]         = RTFVersTexte(jsPlanTestExecution.etapes[nLigneJson].comportement_attendu)
            ATT_Commentaire[nLigne]                 = jsPlanTestExecution.etapes[nLigneJSon].commentaire_etape
            ATT_NiveauAnomalie[nLigne]              = CesPlanTestAnomalieNiveauLibellé(jsPlanTestExecution.etapes[nLigneJson].anomalie_niveau)
            ATT_FrequenceAnomalie[nLigne]           = CesPlanTestAnomalieFrequenceApparitionLibellé(jsPlanTestExecution.etapes[nLigneJson].anomalie_frequence)
            ATT_RefAnomalie[nLigne]                 = jsPlanTestExecution.etapes[nLigneJson].anomalie_ref
        FIN

    AUTRE CAS : RENVOYER Faux
FIN
RENVOYER Vrai
```