

## Préambule

### 1 - Le ticket

### 2 - Analyse du ticket

#### 2.1 - Contexte de l'implémentation

#### 2.2 - Proposition d'implémentation

##### 2.2.1 - Implémentation graphique

##### 2.2.2 - Implémenter la mise à jour des notes

### 3 - Problèmes rencontrés

#### 3.1 - Problème d'API

#### 3.2 - Lecture et réutilisation du code

### 4 - Conclusion

# Préambule

Le dépôt Git utilisé lors de cet exercice est disponible à [cette adresse](#).

Il a été réalisé par Armand SADY et Romain DEGEZ.

## 1 - Le ticket

Ticket décrit sur [cette page](#).

### Formulaire de saisie de note centré sur l'étudiant

*Labels : Enhancement*

Liste tous les modules où est inscrit l'étudiant avec possibilité de saisie de note (vérifier compatibilité avec l'historique des saisies)

#### Commentaire 1

- Lister les évaluations par ordre de modules / évaluation
- Rappeler note(s) existante(s)
- Indiquer clairement le barème (notes sur 10, sur N...)
- Sauvegarder au fur et à mesure (utilisation API notes)
- Gérer correctement les permissions (le(s) resp. de semestre(s) et admins)

#### Commentaire 2

En option:

Afficher les notes des modules correspondants semestres précédents (au cas où l'utilisateur voudrait conserver d'anciennes notes)

## 2 - Analyse du ticket

**Classification** : On peut classer le ticket comme une maintenance évolutive.

En effet, on peut tout d'abord voir que le ticket contient le label `Enhancement`, qui signifie qu'il s'agit d'une nouvelle fonctionnalité. De plus, le ticket énumère différents critères pour cette fonctionnalité, ce qui nous permet de déduire qu'il s'agit bien d'un ajout à effectuer en suivant ces critères.

## 2.1 - Contexte de l'implémentation

Tout d'abord, nous cherchons à quel endroit sur Scodoc sera ajoutée la fonctionnalité.

Vu que le ticket nous demande d'ajouter une liste de modules où l'étudiant est inscrit, avec un formulaire de saisie de notes, il nous semble convenable d'implémenter cette fonctionnalité sur la fiche étudiante.

Accès à la page d'un étudiant :

The screenshot shows the Scodoc interface for a student profile. On the left is a navigation menu with options like 'jean.dupont déconnexion', 'Dépt. CAS', 'Semestres', 'Formations', 'Assiduité', 'Utilisateurs', 'Paramétrage', 'Chercher étudiant:', 'M. Pablo ECHEQUE', 'Absences', 'À propos Aide', and 'DEBUG'. The main content area displays the student's name 'M. Pablo ECHEQUE (ancien)', a profile picture with a question mark, and their status 'Situation : ancien élève'. Below this, the address is listed as 'Adresse : inconnue' with a 'modifier adresse' link. The 'Cursus' section is a table with columns for course number, semester, course name, status, and grades. It lists four semesters of 'Bas de cas de jury' with various grades and ECTS values. At the bottom, there are buttons for 'Transporter', 'Organiser', and 'Manager' across different semesters (BUT 1, BUT 2, BUT 3).

Cursus	Semestre	Etat	Abs	UE	UE	UE	UE	UE
1	Sept 2021	Bas de cas de jury semestre 1 FI 2021-2022	BUT					
2	Fév 2022	Bas de cas de jury semestre 2 FI 2022	BUT					
3	Sept 2022	Bas de cas de jury semestre 3 FI 2022-2023	BUT	UE	UE	UE	UE	UE
	Jan 2023			3.1	3.2	3.3	3.4	3.5
		08.73 0	08.50	11.00	11.00	10.20		
		ECTS: 18 / 34	0	7	7	4	0	
4	Fév 2023	Bas de cas de jury semestre 4 FI 2023	BUT	UE	UE	UE	UE	UE
	Juin 2023			4.1	4.2	4.3	4.4	4.5
		08.82 0	08.50	11.00	11.00	11.00		
		ECTS: 18 / 34	0	7	7	4	0	

Ensuite, nous avons besoin de savoir où cette page est générée dans le code.

Pour cela, nous recherchons dans le code la chaîne de caractères Adresse : , visible sur la page. Cela nous permet de trouver le fichier /app/scodoc/sco\_page\_etud.py, dans lequel est généré tout le HTML de la page.

```
adresse_template = (  
    ""  
    if restrict_etud_data  
    else ""  
    <!-- Adresse -->  
    <div class="ficheadresse" id="ficheadresse">
```

```

<table>
<tr>
  <td class="fichetitre2">Adresse :</td>
  <td> %(domicile)s %(codepostal domicile)s %(ville domicile)s %(pays domicile)s
  %(modif adresse)s
  %(telephones)s
  </td>
</tr>
</table>
</div>
"""
)

```

Après une analyse du code, on observe que dans `/app/scodoc/sco_page_etud.py`, la fiche étudiante est créée dans une méthode `fiche_etud(etudid)`, où différentes parties du HTML sont mises dans un tableau `info` :

```

if has_debouche:
    info[
        "debouche_html"
    ] = f"""<div id="fichedebouche"
        data-readonly="{suivi_readonly}"
        data-etudid="{info['etudid']}">
<span class="debouche_tit">Devenir:</span>
<div><form>
<ul class="listdebouches">
{link_add_suivi}
</ul>
</form></div>
</div>"""

```

La méthode renvoie un `render_template` de Flask, auquel nous donnons :

- Une template Jinja2 définissant la structure de base de la page.
- Une variable `tmpl` organisant le cœur des informations de la page, qui proviennent du tableau `info`.
- Le tableau `info`, les classes CSS et le JavaScript de la page.

```

return render_template(
    "sco_page_dept.j2",
    content=tmpl % info,
    title=f"Fiche étudiant {etud.nomprenom}",
    cssstyles=[
        "libjs/jquery-tagEditor/jquery.tag-editor.css",
        "css/jury_but.css",
        "css/cursus_but.css",
    ],
    javascripts=[
        "libjs/jinplace-1.2.1.min.js",
        "js/ue_list.js",
        "libjs/jquery-tagEditor/jquery.tag-editor.min.js",
        "libjs/jquery-tagEditor/jquery.caret.min.js",
        "js/recap_parours.js",
        "js/etud_debouche.js",
    ],
)

```

## 2.2 - Proposition d'implémentation

Maintenant que nous savons où l'implémentation doit être faite, il faut déterminer comment rendre accessibles les informations nécessaires pour résoudre le ticket. Pour cela, nous avons envisagé deux approches permettant la visualisation des modules et des notes :

- Ajouter un menu déroulant sur la page de l'étudiant, similaire à la rubrique Devenir déjà présente.
- Créer une nouvelle page, accessible via un bouton Modifier notes, qui redirigerait vers une page externe. Ce bouton pourrait visuellement ressembler à celui permettant l'inscription à un autre semestre.

Nous avons réfléchi à la manière d'afficher efficacement toutes les notes sur la page actuelle. Étant donné que nous travaillons sur une base existante, il serait plus simple d'ajouter une nouvelle section plutôt que de disperser les informations, ce qui faciliterait la lecture et la navigation.

Après avoir analysé ce qui s'intégrerait le mieux à l'application, nous avons décidé d'opter pour un menu déroulant.

L'idée est de reproduire le comportement que l'on retrouve sur la page Bas de cas de jury d'un semestre de étudiant, avec l'affichage de toutes les notes de chaque module. Chaque note disposerait d'un champ modifiable :

Comme indiqué dans [plus tôt](#), l'HTML de l'application est partiellement gardée dans le dictionnaire "info", donc afin de garder nos évolutions cohérentes dans le projet, nous créons une nouvelle valeur dans le dictionnaire, nommée "module\_html", que nous ajoutons à l'endroit voulu dans la template renvoyée par la fonction:

```
tmpl = (  
  \"""<div class="menus_etud">%(menus_etud)s</div>  
<div class="fiche_etud" id="fiche_etud"><table>  
<tr><td>  
<h2>%(nomprenom)s %(inscription)s</h2>  
  \"""  
)
```

.....

```
%(module_html)s
```

```
%(debouche_html)s
```

Et nous allons créer une fonction, appelée dans la variable module\_html afin de mettre les informations unique a chaque étudiant:

```
get_ressources().
```

Ressources		Liste ^
<b>R2.01 - Transporter</b>	^	
Moy		10.10 Coef. 01.00
<b>R2.02 - Organiser</b>	^	
Moy		03.00 Coef. 01.00
<b>R2.03 - Manager</b>	^	
Moy		00.00 Coef. 01.00

Ce ticket implique une implémentation à la fois en front-end et en back-end. En effet, il s'agit non seulement d'afficher les notes des évaluations pour chaque module sur la fiche étudiante, mais aussi de permettre leur ajout et leur modification.

## 2.2.1 - Implémentation graphique

Nous avons plusieurs étapes pour implémenter l'interface comme nous le souhaitons. La première consiste à récupérer les informations nécessaires pour composer notre liste.

La première difficulté réside dans l'identification du code responsable de cette mise en page afin de réutiliser le bon style avec le HTML et les classes CSS appropriées.

On retrouve le template de l'affichage dans le fichier `app/static/js/releve-but.js`, dont le code HTML est :

```
<section>
  <div>
    <h2>Ressources</h2>
    <div class=CTA_Liste>
      Liste <svg xmlns="http://www.w3.org/2000/svg" width="26" height="26" viewBox="0 0
24 24" fill="none" stroke="#ffffff" stroke-width="2" stroke-linecap="round" stroke-
linejoin="round">
        <path d="M18 15l-6-6-6 6" />
      </svg>
    </div>
  </div>
  <div class=evaluations></div>
</section>
```

Ainsi que les méthodes `module()` et `evaluation()` qui le complètent.

Nous avons utilisé l'API de Scodoc dans Bruno pour identifier où récupérer les informations nécessaires. Cela nous a permis d'explorer plusieurs pistes, notamment l'appel de l'[endpoint bulletin](#).

Surtout, cela nous a amenés à réfléchir aux types utilisés dans le projet, car il nous semblait plus simple de récupérer ces informations de cette manière, en l'absence de fonctions dédiées trouvées pour le simplifier.

En observant le code, nous avons constaté qu'un étudiant est représenté par le type `Identite`, et que les modules que nous cherchons sont accessibles à travers un objet de type `FormSemestre`, qui représente un semestre d'une formation.

Depuis un étudiant, nous pouvons utiliser la méthode `Identite.get_formsemestres()` pour récupérer les semestres auxquels il est inscrit. Ensuite, depuis ces semestres, l'attribut `FormSemestre.modimpls_sorted` permet d'accéder aux modules d'un semestre.

Nous pouvons alors filtrer les modules selon leur type pour ne conserver que ceux qui nous intéressent, à savoir les ressources et les SAE.

Cela peut se faire ainsi :

```
m_list = {
  scu.ModuleType.RESSOURCE: [],
  scu.ModuleType.SAE: [],
  scu.ModuleType.STANDARD: [],
  scu.ModuleType.MALUS: [],
}
for modimpl in formsemestre.modimpls_sorted:
  d = modimpl.to_dict(convert_objects=True)
  m_list[modimpl.module.module_type].append(d)
```

Grâce à cette approche, nous obtenons tous les modules d'un étudiant. La logique ensuite consisterait à récupérer ses notes pour chaque module.

Faute de temps, cette partie n'a pas pu être implémentée, mais voici une piste potentielle :

Nous observons dans le code qu'un `ModuleImpl` possède un attribut `evaluations`, qui est une liste d'objets `Evaluation` triée.

À partir de ces évaluations, nous avons la méthode `Evaluation.get_etud_note()` permettant de récupérer la note de l'étudiant passé en paramètre.

Avec ces éléments, nous serions déjà capables d'afficher les notes des étudiants.

Ce qu'il faut désormais est l'affichage des notes dans un champ de saisie. Les notes existantes doivent être modifiables, tandis qu'en l'absence de note pour une évaluation, lorsque la méthode `get_etud_note()` renvoie `None`, le champ doit rester vide.

C'est dans la seconde partie que nous voyons cela.

## 2.2.2 - Implémenter la mise à jour des notes

Une fois que les notes sont affichées, l'utilisateur doit pouvoir les modifier, et elles doivent être sauvegardées en direct à chaque modification.

Tout d'abord, nous recherchons si cette fonctionnalité est déjà implémentée quelque part. Logiquement, une page permettant de saisir les notes des étudiants pour une évaluation doit exister.

Nous avons trouvé un endroit où les notes sont mises à jour comme supposé (page Semestre > Ressource > Action "Saisir notes").

En ouvrant les devtools pour inspecter un élément note, on retrouve ceci :

```
<input type="text" name="note_14514" size="5" id="formnotes_note_14514" class="note" data-last-saved-value="8.5" data-orig-value="8.5" data-etudid="14514" onkeypress="return enter_focus_next(this, event);" value="8.5">
```

Bonne nouvelle, cette page sauvegarde les notes à chaque modification, comme nous le souhaitons.

Comme précédemment, nous recherchons cette page dans le code. En utilisant la chaîne de caractères `Mettre les notes manquantes` à, que l'on voit sur la page, nous avons pu identifier le fichier responsable de la mise à jour des notes : `/app/scodoc/sco_saisie_notes.py`.

Nous repérons plusieurs parties qui nous sont utiles : la création du formulaire et la mise à jour des notes. Nous constatons que la méthode `_record_note` permet de modifier les notes dans la base de données. Pour ce faire, elle se connecte à la BDD et exécute des requêtes SQL via un `cursor.execute()`. Cette méthode est utilisée dans `notes_add` avec d'autres méthodes afin de vérifier que tout est correct.

Nous pouvons donc utiliser la méthode `notes_add` pour ajouter ou modifier des notes à chaque fois qu'un input est modifié, en lui fournissant l'`evaluation_id` ainsi qu'une liste contenant des tuples (`etud_id`, `value`).

Un deuxième point important est qu'il faut sauvegarder l'historique des saisies de notes. Or, c'est précisément ce que fait déjà la méthode `notes_add`. Cette méthode est donc particulièrement intéressante.

Cependant, un inconvénient subsiste : le ticket demande d'utiliser l'API note pour sauvegarder les notes, et donc de ne pas modifier directement la base de données, comme le fait la méthode `notes_add`.

Ainsi, pour enregistrer les changements de notes, il faudrait effectuer une requête POST à l'API à l'adresse suivante : `/evaluation/evaluation_id/notes/set` (en remplaçant `evaluation_id` par l'ID de l'évaluation) et en envoyant comme données une liste de notes ainsi qu'un commentaire, sous la forme suivante :

```
{
  "notes": [[1, 17], [2, "SUPR"]],
  "comment": "sample test"
}
```

## 3 - Problèmes rencontrés

### 3.1 - Problème d'API

Comme mentionné dans la partie [2.2.1](#), nous avons utilisé l'API de Scodoc afin d'explorer les pistes possibles pour récupérer les informations, notamment les modules et leurs notes associées pour un étudiant.

En utilisant cette API, nous avons identifié l'endpoint `/Scodoc/api/etudiant/<string:code_type>/<string:code>/formsemestre/<int:formsemestre_id>/bulletin` comme étant prometteur car [l'exemple d'utilisation de cet endpoint](#) montre un accès aux ressources et aux notes associées à l'élève.

Cependant, lorsque nous avons testé cet appel sur notre instance de Scodoc, nous n'avons pas obtenu la liste "ressources" présente dans l'exemple.

Ce comportement nous a semblé intrigant. Après avoir exploré le code, nous avons trouvé que le fichier `app\api\etudiants.py` contient la méthode `bulletin`, qui est censée renvoyer la réponse obtenue via l'API lorsque nous effectuons cet appel. Or, dans ce code, le champ "ressources" n'est pas renseigné.

Cela pourrait signifier que l'exemple fourni n'est pas correct, ou bien que la documentation ne précise pas les conditions nécessaires pour obtenir ces informations, qui pourraient pourtant être importantes.

## 3.2 - Lecture et réutilisation du code

Afin de mener à bien le ticket, il nous a fallu comprendre ce qui existe déjà.

Cependant certaines fonctions font plus de 80 lignes, ce qui rend leur lecture et compréhension longues et difficiles. De plus, leur réutilisation est peu possible, il faut soit dupliquer le code et y apporter quelques modifications, ce qui n'est pas très propre, soit refactoriser les méthodes requises, avec le risque de les casser.

## 4 - Conclusion

Nous avons compris le besoin et analysé ce qui existe déjà en termes de fonctionnalité et de code afin de trouver l'implémentation la plus efficace et la plus proche du style de Scodoc.

Suite à nos recherches et analyses, nous avons commencé à mettre en place notre solution. Cependant, en raison du manque de temps et d'une vision plus globale du code, nous n'avons pas pu terminer le ticket.

Dans l'état actuel du ticket, nous avons donc ajouté cela dans `sco_page_etud.py`:

Une valeur `module_html` dans le dictionnaire `info[]`.

Une méthode `get_ressources()` qui permet, a partir de l'étudiant courant, de récupérer le semestre de ce dernier et les modules de ce dernier, et de les afficher.

`info[module_html]` est enfin utilisé dans `tmpl`, permettant d'afficher ce que nous avons ajouté sur la page.

Il manque actuellement les notes, malgré le fait que la logique est trouvée, et la manière de changer ces notes.