

Projet Cassiopée 2021 n°36
Outil de gestion de base de donnée
de QCM

Rapport
—
Documentation

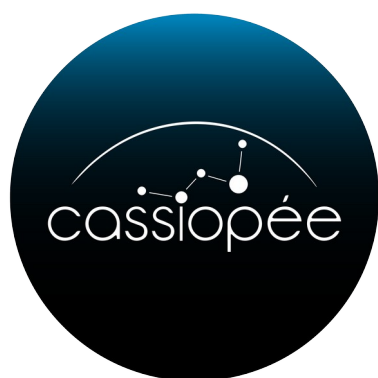


Table des matières

1 - Introduction.....	3
2.1 Contexte du projet.....	3
2.2 Description de l'application.....	3
2 - Modélisation.....	4
2.1 Diagramme de classes.....	4
2.2 Modèle de données.....	4
2.3 Diagramme utilisateur.....	5
2.4 Diagramme du flux de données.....	6
3 - Côté utilisateur.....	7
3.1 Import.....	7
3.1 Show.....	7
3.1 Search.....	7
3.1 Export.....	8
3.2 Tag.....	8
4 - Côté développeur.....	9
4.1 QCM.....	9
4.1 DB.....	10
4.1 Parser.....	12
4.1 Gestion.....	13
4.2 GQCM.....	15
5 - Modifications notables du projet.....	16
6 - Propositions d'amélioration.....	17
7 - Exemples d'utilisation.....	18

Introduction

Contexte du projet

Nous reprenons ce projet Cassiopée pour sa deuxième année. Il s'agit d'un logiciel de gestion de questions et réponses de QCM, couplé à une base de donnée au format .json, il doit permettre l'import de questions dans la base de donnée ainsi que l'export le tout sur des documents au format .tex.

Les consignes du professeur encadrant étaient donc de reprendre le projet et de continuer son développement, en insistant notamment sur l'ergonomie et son fonctionnement complet de l'import à l'export.

Nous avons donc ignoré la partie d'interface visuel développée l'année dernière pour nous concentrer sur la partie fonctionnelle et la rendre aussi compréhensible que possible via le code mais aussi via une documentation.

Description de l'application

L'application permet de gérer une base de donnée de questions afin de générer facilement des questionnaires à choix multiples au format LaTeX.

Elle s'utilise principalement en ligne de commande. Un interface graphique a été développée par l'équipe précédente, mais elle est incomplète.

Voici les fonctionnalités de l'application :

Import : importe et analyse des fichiers LaTeX afin d'en extraire des questions vers la base de données.

Export : génère un fichier LaTeX à partir de questions choisies dans la base de données.

Tag : applique des tags à certaines questions afin de pouvoir les associer à une catégorie.

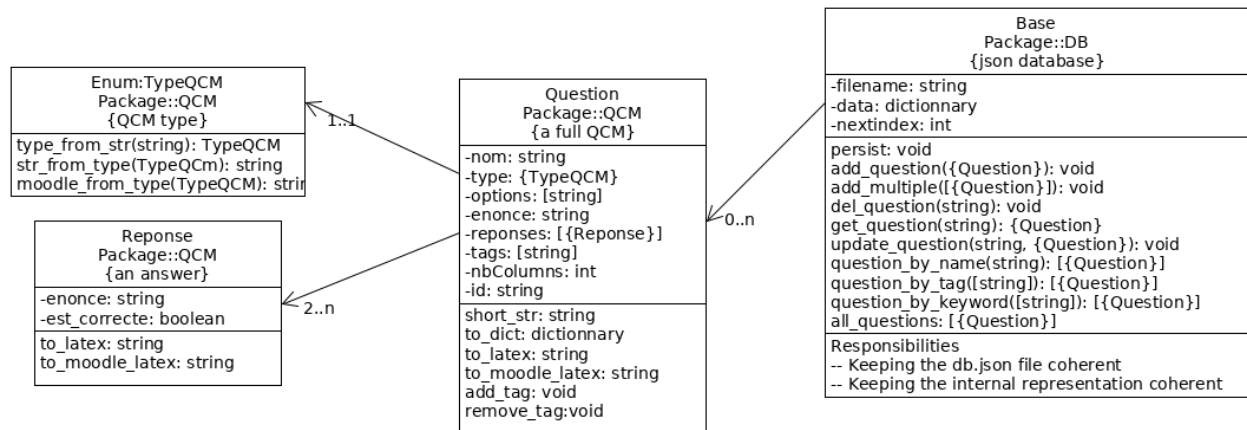
Recherche : cherche et affiche les questions dans la base selon un critère donné.

L'import de questions s'effectuant depuis des fichiers LaTeX, l'application a pour vocation d'être utilisée par un utilisateur ayant

l'habitude d'écrire des questionnaires LaTeX, et qui souhaiterait rassembler toutes les question déjà écrites en LaTeX dans une base de données.

Modélisation

Diagramme de classe



Modèle de données

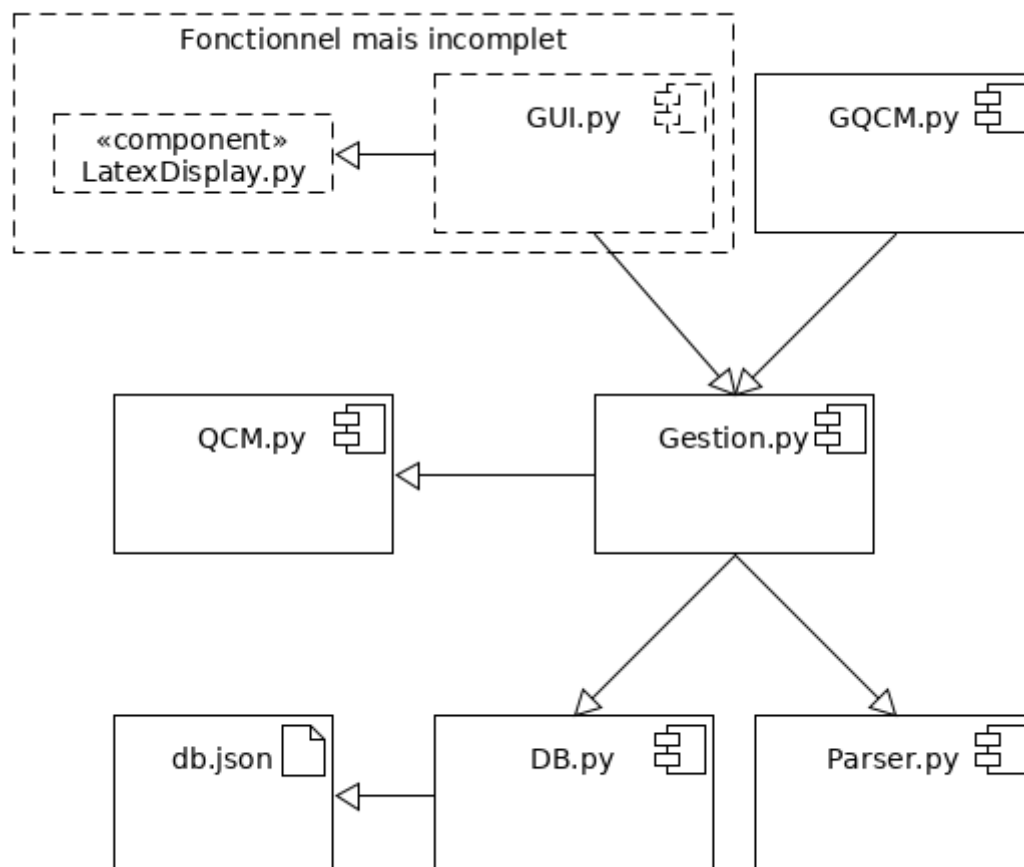


Diagramme utilisateur

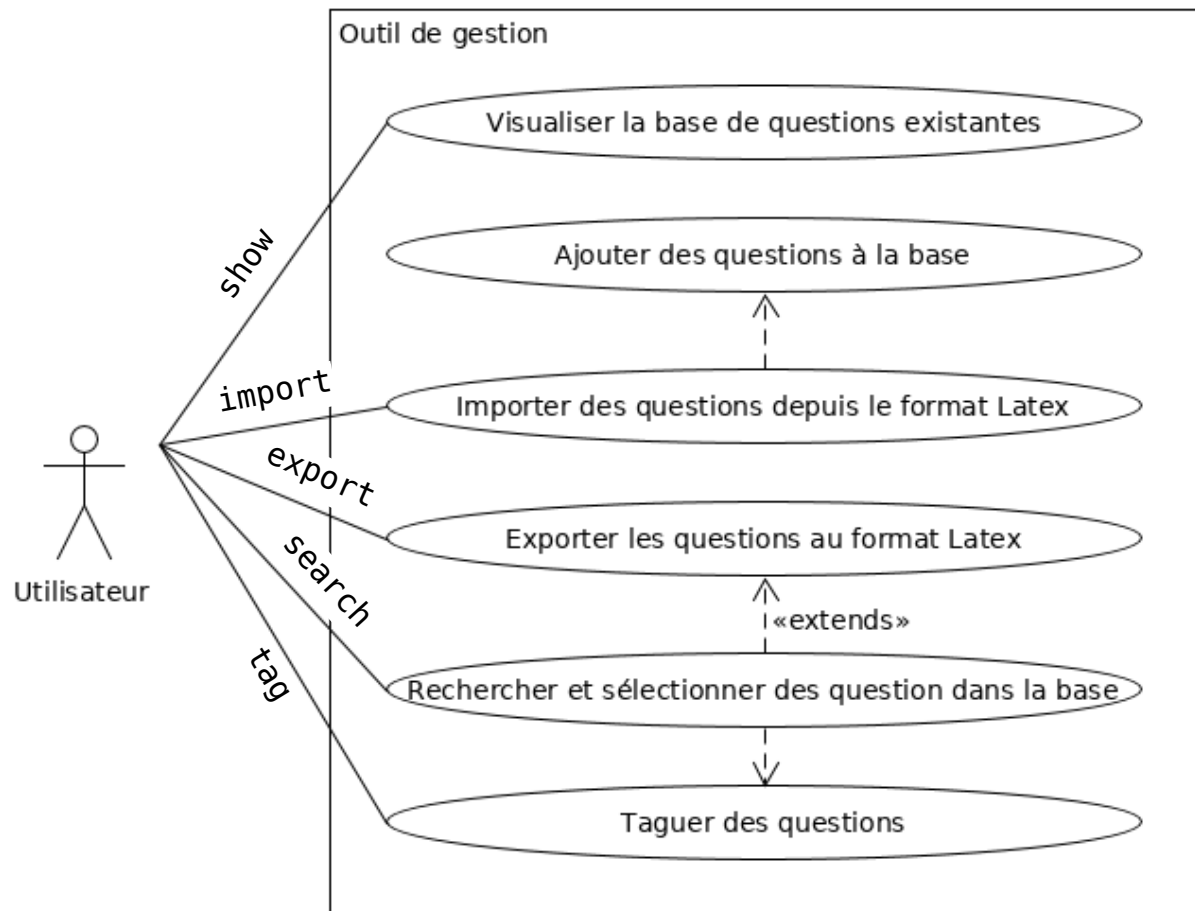
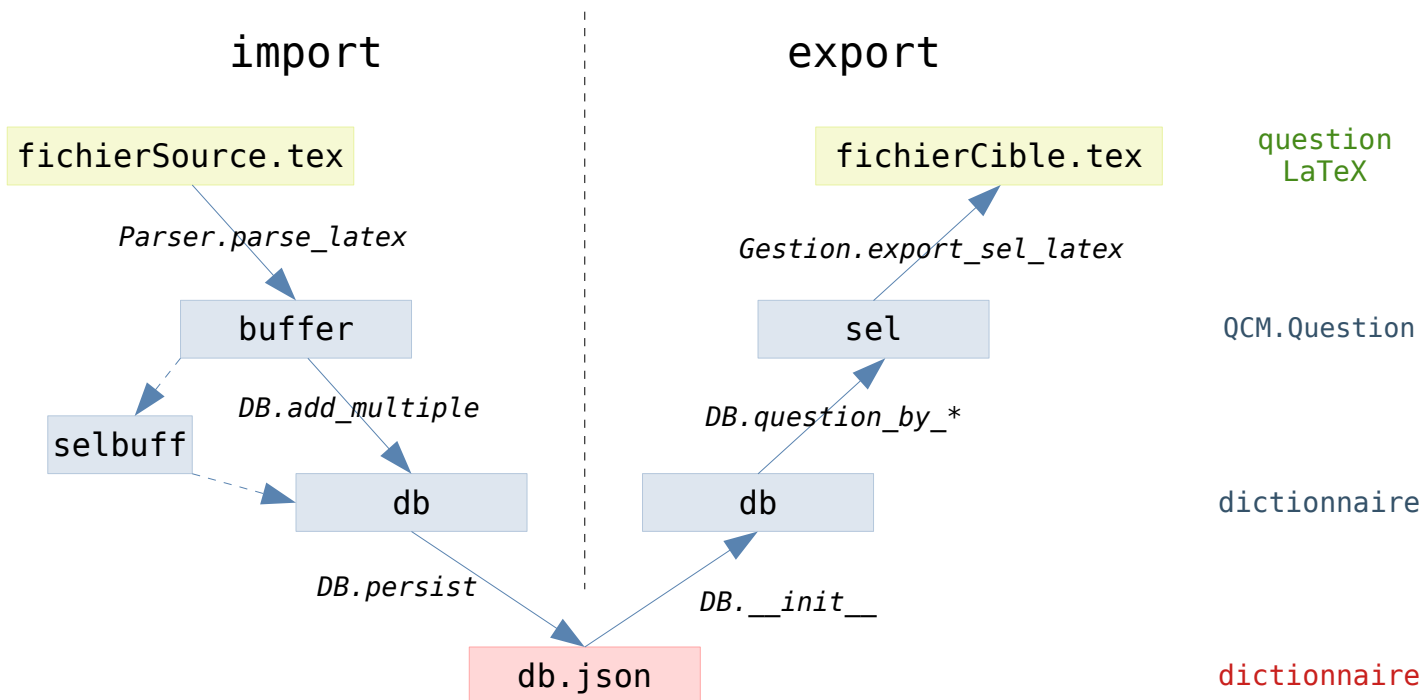


Diagramme du flux de données

Ce diagramme illustre le flux de données représentant une question importée vers la base de données puis exportée vers un fichier LaTeX. A droite se trouve le type de donnée représentant la question, et dans les rectangles le fichier ou la variable où elle est stockée, et sur les flèches la fonction au cœur de l'opération.



: fichiers LaTeX

: variables python : créées puis détruites pendant l'exécution du programme

: base de données au format JSON

db : DB.base → objet créant le dictionnaire python représentant la base de données au sein du programme ; variable est créée et détruite à chaque appel de GQCM

buffer : List[QCM.Question] → questions résultant du parsing d'un fichier LaTeX ; elles ne sont pas encore dans la base de données.

selbuff: List[QCM.Question] → sélection de questions provenant du buffer.

sel: List[Tuple[str, QCM.Question]] → sélection de questions
provenant de la base de donnée

Côté utilisateur

Cette partie de la documentation regroupe les commandes de l'utilisateur.

*Note : toutes les commandes décrites ici doivent être précédées de GQCM pour fonctionner (exemple : **GQCM** import fichier.tex).*

Import :

Permet d'importer les questions contenues dans un fichier .tex dans la base de donnée du logiciel. On a plusieurs cas d'utilisation :

« **GQCM import Cheminversfichier/fichier.tex** »

Les questions sont transformées en objet Question utilisables par le système et stockées dans une sélection temporaire, l'utilisateur peut ensuite choisir celles qu'il désire sauvegarder tel qu'indiqué par la commande.

« **GQCM import** »

« **Cheminversfichier/fichier.tex** »

Même principe mais entrée de la commande en plusieurs lignes.

Show :

« **GQCM show** »

Affiche toutes les questions sauvegardées dans la base de donnée, sous le format :

id : Nom {TYPE} [tags]

Début de l'énoncé...

Search :

« **GQCM search** »

« **critèreDeRecherche** »

« **critèreA critèreB ...** »

Affiche les questions de la base de données correspondant aux critères données (nom, tags ou id), puis permet d'effectuer certaines opérations sur les questions affichées. Une utilisation en une seule ligne se fait de la sorte :

« **GQCM search name Question1 Question2** »

« **tag tagA tagB** »

On cherche ici les questions de nom Question1 et Question2, et une fois trouvées on décide de leur appliquer les tags tagA et tagB.

Export :

« **GQCM export** »
« **critère** »
« **nomCritère** »
« **quit** »
« **finish** »
« **fichierDestination** »

Permet d'exporter des questions dans un fichier destination.

On choisit un critère d'export (tag, nom ou id) puis on entre le critère (exemple : on tape nom puis « nomQuestion »), on tape ensuite quit puis on peut choisir de sélectionner plus de question de la même manière ou passer directement à l'export en tapant finish. On tape ensuite un nom de fichier (exemple fichierExport) qui doit être différent de « », ensuite les questions sélectionnées seront sauvegardées dans le fichier **fichierExport.tex**.

Tag :

« **GQCM tag** »
« **critère** »
« **nomCritère** »
« **quit** »
« **finish** »
« **tagA tagB** »

Permet d'ajouter des tags à une sélection de questions, avec un fonctionnement similaire à export.

On choisit un critère de sélection des questions à taguer (tag, nom ou id) puis on entre le critère (exemple : on tape nom puis « nomQuestion »), on tape ensuite quit puis on peut choisir de sélectionner plus de question de la même manière ou passer directement au tag en tapant finish. On entre ensuite tous les tags à ajouter aux questions (par exemple « tagA tagB »).

Côté développeur

Cette partie de la documentation explique les différents modules utilisés par le système, et donne le rôle des différentes fonctions. Les fonctions simples (getters, setters, etc.) sont volontairement omises par soucis de clareté.

Note : La plupart des fonctions expliquées ici sont également expliquées directement en commentaire dans le fichier DB.py

QCM :

Ce module définit les classes `Question` et `Reponse` qui servent à représenter les questions à choix multiple qui seront stockées dans la base de données.

class TypeQCM(Enum): Cette classe énumération représente le type de question (à réponses multiple ou non) sans utiliser une chaîne de caractère.

type_from_str(string): Permet d'obtenir l'objet `TypeQCM` à partir d'une chaîne de caractère.

str_from_type(type_qcm): Permet d'obtenir la chaîne de caractère (format LaTeX) associée à l'objet `TypeQCM`.

moodle_from_type(type_qcm): Permet d'obtenir la chaîne de caractère (format LaTeX Moodle) associée à l'objet `TypeQCM`.

class Reponse: Cette classe représente la réponse à une question.

`self.est_correcte` : indique si la réponse est bonne ou mauvaise

`self.enonce` : l'énoncé de la réponse.

`to_latex(self):` retourne un string au format LaTeX représentant la réponse

`to_moodle_latex(self):` retourne un string au format LaTeX Moodle représentant la réponse

class Question: Cette classe représente une question à choix multiples.

`self.type:` le type (réponses multiples ou non) de la question

`self.nom:` le nom (court, ce n'est pas l'énoncé)

`self.amc_options:` options de Auto Multiple Choice

`self.enonce:` l'énoncé complet

`self.reponses:` la liste des réponses

self.tags: la liste des tags
self.numberColumn: le nombre de colonnes
self.id: l'id de la question. Il est unique dans la base de données, et n'est défini que lorsque la question est bien insérée dans la base.
short_str(self): renvoie une string décrivant la question. Elle est au format :
 id : Nom {TYPE} [tags]
 Début de l'énoncé...
to_dict(self, index): renvoie un dictionnaire représentant l'objet Question. Chaque attribut correspond à une clé. C'est ce dictionnaire qui est inséré dans la base de données.
to_latex(self): renvoie une string contenant le code LaTeX représentant la question
to_moodle_latex(self): renvoie une string contenant le code LaTeX Moodle représentant la question

DB :

Ce module implémente une base de donnée stockant des objets provenant du module QCM. Cette base de donnée est sauvegardée dans un fichier au format json en tant que dictionnaire.

tag_check(tags, required): retourne vrai si tags contient tous les tags contenus dans required, faux sinon. (tags et required sont des listes de variables de type string)

keyword_check(text, keywords): retourne vrai si text contient tous les mots contenus dans keywords, faux sinon.

reponse_from_dict(rdict): retourne une variable de type QCM.Reponse créée à partir du dictionnaire rdict passé en paramètre.

question_from_dict(qdict): retourne une variable de type QCM.Question créée à partir du dictionnaire qdict passé en paramètre.

class Base: Il s'agit de la classe utilisée pour représenter la base de donnée dans le système. Elle crée un dictionnaire python à partir de la base de données au format JSON.

__init__(self, input_file): stocke input_file dans self.filename, puis stocke les données récupérées dans le fichier

ouvert via `input_file` dans le dictionnaire `self.data`.

`self.nextindex` : l'indice du prochain élément à insérer dans la base

`persist(self)`: Sauvegarde data dans le fichier ouvert via `filename`. Tant que `persist` n'est pas appelée, les changements ont lieu dans le dictionnaire python mais pas dans le fichier JSON.

`add_question(self, question)`: Ajout de l'objet `QCM.Question` à data après conversion au format dictionnaire.

`add_multiple(self, questions)`: Appelle `add_question` pour chaque `QCM.Question` contenue dans la liste `questions`

`del_question(self, index)`: Supprime la question d'index correspondant de data

`get_question(self, index)`: Retourne la question d'index correspondant de data

`update_question(self, index, question)`: Remplace la question d'index correspondant dans data par la nouvelle question, ou la crée si aucune question d'index correspondant n'existe

`question_by_*(self, *)`: Retourne la liste des questions correspondant au critère * (nom, id ou tag) de la base de données, après conversion en objet `QCM.Question`.

`all_questions(self)`: Retourne une liste de toutes les questions contenues dans data, après conversion en objet `QCM.Question`.

Parser:

Ce module permet de parser les fichiers .tex et d'en obtenir les questions, c'est à dire de construire des objets de classe Question (qui ont des attributs de classe Reponse) à partir d'un fichier .tex.

Les différentes fonctions décomposent le fichier en lignes et utilisent les différentes marques de latex pour reconnaître les éléments lus.

get_block(text, block_open_char, block_close_char):

Cherche et renvoie dans text le premier bloc de texte compris entre les caractères block_open_char et block_close_char. Par exemple, `get_block("test {content {sub} content}", '{', '})` renvoie 'content {sub} content'. Les caractères ouvrant et fermant sont omis.

pattern_at(text, index, pattern): Renvoie un booléen indiquant si le motif pattern commence à l'index index dans la chaîne de caractères text.

parse_latex(latex): Lit le fichier latex ligne par ligne et fait appel à `parse_qcm` pour créer des objets QCM.Question à partir des données récupérées. La fonction détecte les balises `\\begin{response}` et `\\end{response}` et fait appel à `parse_qcm` pour créer un objet QCM.question à partir des lignes (`q_lines`) comprises entre ces balises.

parse_qcm(q_lines, nb_questions): Crée un objet QCM.Question à partir des lignes `q_lines` comprises entre `\\begin{response}` et `\\end{response}`. De manière similaire, la fonction fait appel à `parse_reponses` quand elle détecte des lignes (`r_lines`) correspondant à une réponse.

Cette fonction donne également un nom générique à la question si cette dernière n'en a aucun indiqué dans le fichier LaTeX. Ce nom est « QuestionN » avec N le rang de la question dans la liste des questions extraites. Le nom généré peut donc ne pas être unique dans la base de données.

parse_reponses(r_lines): Crée un objet de classe QCM.Reponse à partir des lignes `r_lines` correspondant à une réponse.

Gestion :

Ce module est le cœur de l'application. C'est le module qui fait le lien entre les quatre autres modules. Tous les autres modules devraient éviter autant que possible d'utiliser d'autres modules que celui-ci.

Note 1 : la variable `view` ainsi que les fonctions associées sont utilisées par l'interface graphique `GUI.py`. Elles sont laissées dans le code si une prochaine équipe souhaite continuer le développement de l'interface graphique, mais cette documentation se concentre sur l'utilisation en ligne de commande. Nous n'en parlerons donc pas.

*Note 2 : Certaines fonctions du module ne sont jamais utilisées, mais pourraient l'être si de nouvelles fonctionnalités sont ajoutées à l'avenir. Nous avons donc décidé de les laisser dans le code et de les inclure à la documentation. Elles sont précédées d'une *.*

sel: List[Tuple[str, QCM.Question]] = []: La liste des questions actuellement sélectionnées : ce sont celles auxquelles on va appliquer une opération (export, tag, affichage...). La sélection s'effectue depuis la base de données JSON. La chaîne de caractères de la paire contient l'id de la Question.

buffer: List[QCM.Question] = []: La liste des questions résultant du parsing d'un fichier LaTeX. Elles ne sont pas encore dans la base de données.

selbuff: List[QCM.Question] = []: Une sélection de questions du buffer. Elle est différente de la sélection `sel` car les questions de `selbuff` ne sont pas encore dans la base de données (on les sélectionne afin de pouvoir éventuellement les y ajouter).

db: DB.Base: le dictionnaire python représentant la base de données.

init(): Cette fonction doit toujours être appelée afin de pouvoir utiliser les autres fonctions du module.

Elle initialise la base de données (c'est à dire initialise le dictionnaire python à partir du fichier JSON, cf. classe `DB.Base`) ou bien la crée si elle n'existe pas.

export_sel_latex(filename): Exporte les questions sélectionnées depuis la base de données dans le fichier `filename` au format LaTeX.

export_sel_moodle(filename): Exporte les questions sélectionnées depuis la base de données dans le fichier filename au format LaTeX Moodle.

***export_buffer_latex(filename):** Exporte les questions du buffer dans le fichier filename au format LaTeX.

***export_buffer_moodle(filename):** Exporte les questions du buffer dans le fichier filename au format LaTeX Moodle.

parse_file(filename): Parse le fichier LaTeX filename et stocke les questions extraites dans le buffer. Renvoie le nombre de questions trouvées.

save_buffer(): Insère le contenu du buffer dans la base db. Attention, cela modifie le dictionnaire python, pas le fichier JSON : un appel à `persist()` reste nécessaire pour que les modifications soient conservées après la fin du programme.

save_sel_buffer(): Insère le contenu du selbuff dans la base db.

***remove_buffer/sel(index):** Supprime la question d'index index du buffer/sel.

***save_sel():** Sauvegarde les modifications effectuées sur les questions de la sélection. Ne les enregistre pas à nouveau dans la base, mais modifie leur entrée.

***update_index(index, update):** Met à jour la question à d'id index de la base, en la remplaçant par update.

persist_db(): Appelle `db.persist` pour répercuter les modifications apportées au dictionnaire python db sur la base de données JSON. Sans appel à cette fonction, les modifications ne sont pas enregistrées après la fin du programme.

apply_tag(index, tag): Ajoute le tag tag à la liste des tags de la question de la base d'id index. Des variantes permettent d'appliquer le tag à : toute la sélection (`apply_tag_all`), tout le buffer (`apply_all_buffer`), ou la question d'id index du buffer (`apply_tag_all_buffer`).

remove_duplicates(): Supprime les entrée en double de la sélection.

select_attribute: Ajoute à la sélection les questions dont l'attribut (tags, name, keywords) correspond. Pour tags et keywords, l'entrée est une liste de tags/keywords et les questions renvoyées doivent posséder *tous* les tags/keywords de la liste. Les keywords sont cherchés dans l'énoncé de la question, pas dans son nom ni ses réponses.

select_id(db_id): Ajoute à la sélection la question d'id db_id, et renvoie true si elle a bien été trouvée.

select_buffer_name(name): Ajoute à la sélection selbuff les question du buffer nommées name.

select_all(): Ajoute toutes les questions de la base de données à la sélection.

***refresh_sel():** Rafraîchit la sélection, c'est-à-dire remplace chaque question de la sélection par la question de même id dans la base de données. Cela permet de voir les changements qui viennent d'être appliqués aux questions de la sélection.

GQCM :

C'est l'application en ligne de commande utilisée par l'utilisateur. Elle n'est constituée que d'un seul script qui interprète les attributs entrés par l'utilisateur et effectue divers opérations au besoin. Le module Gestion est écrit de sorte à ce que GQCM.py n'utilise que des fonctions du module Gestion.

Le fonction mesure le nombre d'arguments et prévient l'utilisateur s'il manque des arguments : ce dernier est alors amené à entrer les attributs manquants à chaque étape.

Pour plus d'informations concernant les différentes utilisations de GQCM, se référer à la documentation utilisateur.

Modifications notables du projet

Concernant le système lui-même nous avons modifié la manière d'entrer des commandes afin de se rapprocher d'un fonctionnement en ligne de commande similaire à Git. L'ancienne version demandait de lancer le programme avant d'enchaîner les opérations puis de sauvegarder la base. Désormais, les opérations peuvent être effectuées individuellement et beaucoup plus rapidement.

Nous avons modifié les systèmes d'import et d'export afin de corriger certaines erreurs et les adapter au nouveau fonctionnement.

Concernant l'ergonomie, nous avons rajouté des instructions et des aides afin de guider l'utilisateur, comme un affichage de toutes les commandes lorsque GQCM est entré seul. Si l'utilisateur ne donne pas assez d'arguments pour une fonction donnée, le programme le détecte sans s'interrompre et demande les arguments manquants avant de continuer.

Propositions d'amélioration

Dans un cas de reprise du projet, nous proposons la reprise de l'interface graphique que nous avons laissé de côté cette année. Celle-ci pourrait fonctionner en parallèle du script GQCM, en utilisant les modules de manière similaire. Celle-ci devra permettre l'import et l'export de questions mais surtout une visualisation travaillée du contenu de la base de donnée. Le but devra être de pouvoir visualiser en temps réel les modifications apportées à la base de donnée.

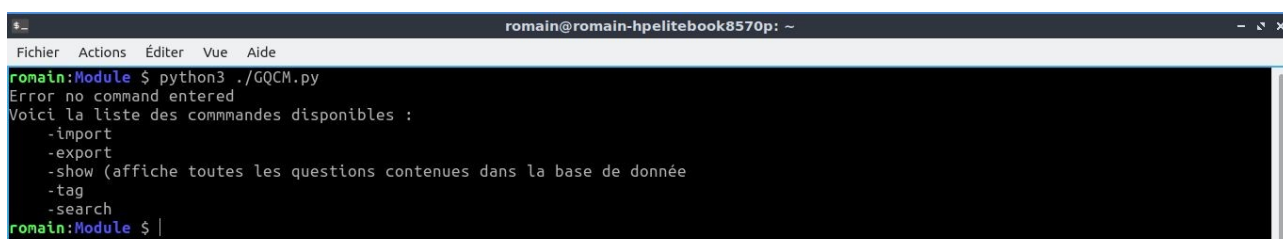
On pourra également modifier les commandes présentes dans GQCM pour les rendre plus efficaces et faciles d'utilisation et possiblement en rajouter si le besoin se fait sentir. Il semble notamment utile d'ajouter des fonctions permettant la modification des questions dans la base. Si en modifier l'énoncé ou les réponses semble peu pratique en ligne de commande, une commande de suppression de tags et de modification du nom d'une questions peut être envisagée.

Exemples d'utilisation

Afin de faciliter la prise en main par l'utilisateur, voici un guide illustré pas-à-pas d'une utilisation typique de l'application.

Étape 0 - État initial de la base de données

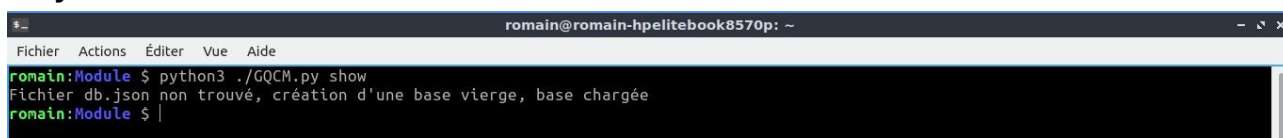
Je viens d'installer l'application depuis le dépôt git. Lançons GQCM et voyons ce qu'il se passe :



```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py  
Error no command entered  
Voici la liste des commandes disponibles :  
-import  
-export  
-show (affiche toutes les questions contenues dans la base de donnée  
-tag  
-search  
romain:Module $ |
```

GQCM ne peut être appelé sans argument. Le premier argument est la fonction utilisée. Il en existe actuellement 5 : import, export, show, tag et search. Les arguments suivant sont les arguments de la fonction.

Voyons l'état de la base de données avec la fonction show.



```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py show  
Fichier db.json non trouvé, création d'une base vierge, base chargée  
romain:Module $ |
```

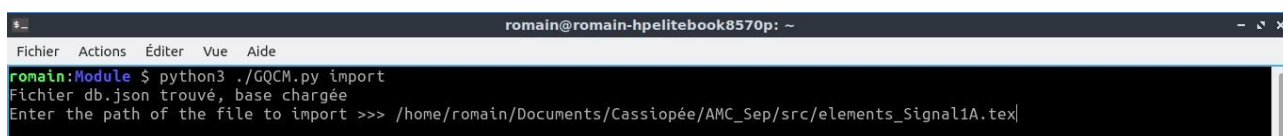
Le programme n'ayant pas trouvé la base (car elle n'existe pas), il en crée une vierge.

Remplissons là.

Étape 1 - Import de questions dans la base

Le fichier `elements_Signal1A.tex` contient des questions en LaTeX déjà écrites pour un ancien examen. Ajoutons-les à la base de données avec la fonction import.

Le programme nous demande d'entrer le chemin vers le fichier en question.



```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py import  
Fichier db.json trouvé, base chargée  
Enter the path of the file to import >>> /home/romain/Documents/Cassiopée/AMC_Sep/src/elements_Signal1A.tex
```

On aurait pu faire cela en une seule étape en entrant directement :
`import [cheminVersLeFichier]`.

Le programme a parsé le fichier et en a extrait 46 questions, qui sont alors affichées sur le terminal. Parmi elles, seule certaines nous intéressent. On entre alors leurs noms.

```
QUESTION40 {QUESTION} []  
\label \ref{fig:filtreNum020609bis} représente schématiquement le module de la réponse en fré...  
:  
QUESTION41 {QUESTION_MULT} []  
Soit  $\gamma_x(f)$  la densité spectrale d'énergie (ou respectivement de puissance) d'un signal ...  
:  
QUESTION42 {QUESTION_MULT} []  
Un filtre numérique rationnel défini par sa fonction de transfert en  $z$   $H(z)$  ou par sa réponse ...  
:  
QUESTION43 {QUESTION} []  
Pour les signaux alé...  
:  
QUESTION44 {QUESTION} []  
Soit un filtre temps continu de réponse impulsionnelle  $h(t)$  et de réponse en fréquence  $H(f)$ . ...  
:  
QUESTION45 {QUESTION} []  
Deux séances de travaux pratiques (TP) ont eu ...  
:  
QUESTION46 {QUESTION} []  
\label page \pageref{fig:MatlabSpec1} a été obtenue à l'aide du logiciel \textsc{Matlab}. ...  
Input names of questions to save, or press enter to select all >>> Question1 Question2 QUESTION41 QUESTION42]
```

On pourrait sélectionner toutes les questions en appuyant sur la touche ↵.

Les questions sont alors sauvegardées dans la base. On peut les voir avec la commande `show`.

```
Input names of questions to save, or press enter to select all >>> Question1 Question2 QUESTION41 QUESTION42  
Base de donnée sauvegardée  
romain:Module $ python3 ./GQCM.py show  
Fichier db.json trouvé, base chargée  
1: Question1 {QUESTION} []  
La fonction d'autocorrélation  $\gamma_x(\tau)$ ,  $\tau \in \mathbb{R}$  (en énergie ou puissance) d'un signal à ...  
2: Question2 {QUESTION} []  
On peut observer un phénomène d'élargissement des raies ...  
3: QUESTION41 {QUESTION_MULT} []  
Soit  $\gamma_x(f)$  la densité spectrale d'énergie (ou respectivement de puissance) d'un signal ...  
4: QUESTION42 {QUESTION_MULT} []  
Un filtre numérique rationnel défini par sa fonction de transfert en  $z$   $H(z)$  ou par sa réponse ...  
romain:Module $ |
```

Maintenant que notre base est remplie, agissons sur les questions présentes.

Étape 2 - Recherche dans la base

On cherche une question en particulier dans la base, car on ne sait plus si elle est bien présente. On utilise cela la fonction `search`.

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py search  
Fichier db.json trouvé, base chargée  
Select searching method ('tag', 'name' or 'id') or press enter to abort >>> |
```

Il faut alors préciser le critère de recherche. On cherche une question avec un nom particulier.

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py search  
Fichier db.json trouvé, base chargée  
Select searching method ('tag', 'name' or 'id') or press enter to abort >>> name  
Input the names of the questions you're searching for, or press enter to abort >>> |
```

On entre alors le ou les nom(s) recherchés : ici, on veut la Question2.

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py search  
Fichier db.json trouvé, base chargée  
Select searching method ('tag', 'name' or 'id') or press enter to abort >>> name  
Input the names of the questions you're searching for, or press enter to abort >>> Question2  
2: Question2 {QUESTION} []  
On peut observer un phénomène d'élargissement des raies ...  
Select what to do with the selection ('tag', 'export') or press enter to quit >>> |
```

La questions trouvée est alors affichée.

On aurait pu effectuer cette étape en une seule ligne, et chercher plusieurs noms à la fois, de la manière suivante :

```
search name Question1 Question2 QUESTION41
```

Si on veut juste voir s'afficher la question cherchée, on peut alors quitter en appuyant sur la touche ↵. On décide cependant d'appliquer un tag à la Question2. On entre alors tag, puis le nom du tag à appliquer.

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py search  
Fichier db.json trouvé, base chargée  
Select searching method ('tag', 'name' or 'id') or press enter to abort >>> name  
Input the names of the questions you're searching for, or press enter to abort >>> Question2  
2: Question2 {QUESTION} []  
On peut observer un phénomène d'élargissement des raies ...  
Select what to do with the selection ('tag', 'export') or press enter to quit >>> tag  
Input the tags to apply to the selection >>> spectre|
```

On peut vérifier que le tag a bien été appliqué avec la fonction show, ou avec la fonction search si la base est bien remplie.

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py search  
Fichier db.json trouvé, base chargée  
Select searching method ('tag', 'name' or 'id') or press enter to abort >>> name  
Input the names of the questions you're searching for, or press enter to abort >>> Question2  
2: Question2 {QUESTION} []  
On peut observer un phénomène d'élargissement des raies ...  
Select what to do with the selection ('tag', 'export') or press enter to quit >>> tag  
Input the tags to apply to the selection >>> spectre  
romain:Module $ python3 ./GQCM.py show  
Fichier db.json trouvé, base chargée  
1: Question1 {QUESTION} []  
La fonction d'autocorrélation  $\gamma_x(\tau)$  (en énergie ou puissance) d'un signal à ...  
2: Question2 {QUESTION} ['spectre']  
On peut observer un phénomène d'élargissement des raies ...  
3: QUESTION41 {QUESTION_MULT} []  
Soit  $\gamma_x(f)$  la densité spectrale d'énergie (ou respectivement de puissance) d'un signal ...  
4: QUESTION42 {QUESTION_MULT} []  
Un filtre numérique rationnel défini par sa fonction de transfert en  $z$   $H(z)$  ou par sa réponse ...  
romain:Module $ |
```

Étape 3 - Appliquer des tags à une grande sélection

La fonction search sert à faire une recherche rapide de questions, puis à leur appliquer une opération tant qu'on les a sous la main. Mais pour taguer un ensemble plus complexe de questions, il existe la fonction tag.

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py tag  
Fichier db.json trouvé, base chargée  
Select how to choose what you want to tag : 'tag';'name';'id';   enter finish to finalize tag >>> |
```

On peut alors ajouter des questions à la sélection, selon les 3 critères tag, name ou id. Cela suppose que l'on connaisse au préalable quelles questions correspondent aux critères que l'on va entrer. On souhaite taguer la question d'id égal à 3 :

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py tag  
Fichier db.json trouvé, base chargée  
Select how to choose what you want to tag : 'tag';'name';'id';   enter finish to finalize tag >>> id  
Enter an id or enter quit to go back to selection back >>> 3|
```

On ne veut pas ajouter d'autres questions selon leur id, on entre alors quit pour sélectionner un autre critère d'ajout.

Elle est alors ajoutée à la sélection, et les questions de la sélection sont affichées.

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py tag  
Fichier db.json trouvé, base chargée  
Select how to choose what you want to tag : 'tag';'name';'id';   enter finish to finalize tag >>> id  
Enter an id or enter quit to go back to selection back >>> 3  
Enter an id or enter quit to go back to selection back >>> quit  
  
Questions currently selected to tag :  
  
3: QUESTION41 {QUESTION_MULT} []  
Soit  $\gamma_x(f)$  la densité spectrale d'énergie (ou respectivement de puissance) d'un signal ...  
  
Select how to choose what you want to tag : 'tag';'name';'id' :  
Or enter finish to finalize tag >>>|
```

On veut également taguer la question de nom Question1 : on entre name, puis Question1, puis quit.


```

romain@romain-hpelitebook8570p: ~
Fichier Actions Éditer Vue Aide

romain:Module $ python3 ./GQCM.py tag
Fichier db.json trouvé, base chargée
Select how to choose what you want to tag : 'tag';'name';'id';   enter finish to finalize tag >>> id
Enter an id or enter quit to go back to selection back >>>3
Enter an id or enter quit to go back to selection back >>>quit

Questions currently selected to tag :

3: QUESTION41 {QUESTION_MULT} []
Soit  $\gamma_x(f)$  la densité spectrale d'énergie (ou respectivement de puissance) d'un signal ...

Select how to choose what you want to tag : 'tag';'name';'id' :
Or enter finish to finalize tag >>>name
Enter a name or enter quit to go back to selection type >>> Question1
Enter a name or enter quit to go back to selection type >>> quit

Questions currently selected to tag :

3: QUESTION41 {QUESTION_MULT} []
Soit  $\gamma_x(f)$  la densité spectrale d'énergie (ou respectivement de puissance) d'un signal ...

1: Question1 {QUESTION} []
La fonction d'autocorrélation  $\gamma_x(\tau)$ ,  $\tau \in \mathbb{R}$  (en énergie ou puissance) d'un signal à ...

Select how to choose what you want to tag : 'tag';'name';'id' :
Or enter finish to finalize tag >>>|

```

On ne veut rien taguer d'autre, on entre alors finish.
On entre alors le tag que l'on veut appliquer à la sélection (énergie, car c'est un thème commun à toutes les questions sélectionnées). Le tag est appliqué.

```

romain@romain-hpelitebook8570p: ~
Fichier Actions Éditer Vue Aide

romain:Module $ python3 ./GQCM.py tag
Fichier db.json trouvé, base chargée
Select how to choose what you want to tag : 'tag';'name';'id';   enter finish to finalize tag >>> id
Enter an id or enter quit to go back to selection back >>>3
Enter an id or enter quit to go back to selection back >>>quit

Questions currently selected to tag :

3: QUESTION41 {QUESTION_MULT} []
Soit  $\gamma_x(f)$  la densité spectrale d'énergie (ou respectivement de puissance) d'un signal ...

Select how to choose what you want to tag : 'tag';'name';'id' :
Or enter finish to finalize tag >>>name
Enter a name or enter quit to go back to selection type >>> Question1
Enter a name or enter quit to go back to selection type >>> quit

Questions currently selected to tag :

3: QUESTION41 {QUESTION_MULT} []
Soit  $\gamma_x(f)$  la densité spectrale d'énergie (ou respectivement de puissance) d'un signal ...

1: Question1 {QUESTION} []
La fonction d'autocorrélation  $\gamma_x(\tau)$ ,  $\tau \in \mathbb{R}$  (en énergie ou puissance) d'un signal à ...

Select how to choose what you want to tag : 'tag';'name';'id' :
Or enter finish to finalize tag >>>finish
Entrez un tag :energie
Tag energie appliqué aux question sélectionnées.
Base de donnée sauvegardée
romain:Module $ |

```

On peut voir résultat avec show : certaines question ont le tag energie, une autre le tag spectre et la dernière aucun tag.

```

romain:Module $ python3 ./GQCM.py show
Fichier db.json trouvé, base chargée
1: Question1 {QUESTION} ['energie']
La fonction d'autocorrélation  $\gamma_x(\tau)$ ,  $\tau \in \mathbb{R}$  (en énergie ou puissance) d'un signal à ...

2: Question2 {QUESTION} ['spectre']
On peut observer un phénomène d'élargissement des raies ...

3: QUESTION41 {QUESTION_MULT} ['energie']
Soit  $\gamma_x(f)$  la densité spectrale d'énergie (ou respectivement de puissance) d'un signal ...

4: QUESTION42 {QUESTION_MULT} []
Un filtre numérique rationnel défini par sa fonction de transfert en  $z$   $H(z)$  ou par sa réponse ...

romain:Module $ |

```


Étape 4 - Export de questions vers un fichier LaTeX

Exportons des questions vers un fichier LaTeX, qui servira de base au questionnaire d'un examen.

La fonction export fonctionne comme la fonction tag : on sélectionne plusieurs questions selon différents critères. Ici, on veut exporter les questions de nom Question1 et celles de tag spectre.

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py export  
Fichier db.json trouvé, base chargée  
EXPORTING  
Select how to choose what you want to export : 'tag';'name';'id'; enter finish to finalize export >>> name  
Enter a name or enter quit to go back to selection type >>> Question1  
Enter a name or enter quit to go back to selection type >>> quit  
  
Questions currently selected for export :  
1: Question1 {QUESTION} ['energie']  
La fonction d'autocorrélation  $\gamma_x(\tau)$  (en énergie ou puissance) d'un signal à ...  
  
Select how to choose what you want to export : 'tag';'name';'id' :  
Or enter finish to finalize export >>>tag  
Enter a tag or enter quit to go back to selection type >>> spectre  
Enter a tag or enter quit to go back to selection type >>> quit  
  
Questions currently selected for export :  
1: Question1 {QUESTION} ['energie']  
La fonction d'autocorrélation  $\gamma_x(\tau)$  (en énergie ou puissance) d'un signal à ...  
2: Question2 {QUESTION} ['spectre']  
On peut observer un phénomène d'élargissement des raies ...  
  
Select how to choose what you want to export : 'tag';'name';'id' :  
Or enter finish to finalize export >>>
```

Une fois cela fait, on entre finish puis le nom du fichier de destination (pas besoin d'écrire l'extension .tex, elle est ajoutée automatiquement).

```
romain@romain-hpelitebook8570p: ~  
Fichier Actions Éditer Vue Aide  
romain:Module $ python3 ./GQCM.py export  
Fichier db.json trouvé, base chargée  
EXPORTING  
Select how to choose what you want to export : 'tag';'name';'id'; enter finish to finalize export >>> name  
Enter a name or enter quit to go back to selection type >>> Question1  
Enter a name or enter quit to go back to selection type >>> quit  
  
Questions currently selected for export :  
1: Question1 {QUESTION} ['energie']  
La fonction d'autocorrélation  $\gamma_x(\tau)$  (en énergie ou puissance) d'un signal à ...  
  
Select how to choose what you want to export : 'tag';'name';'id' :  
Or enter finish to finalize export >>>tag  
Enter a tag or enter quit to go back to selection type >>> spectre  
Enter a tag or enter quit to go back to selection type >>> quit  
  
Questions currently selected for export :  
1: Question1 {QUESTION} ['energie']  
La fonction d'autocorrélation  $\gamma_x(\tau)$  (en énergie ou puissance) d'un signal à ...  
2: Question2 {QUESTION} ['spectre']  
On peut observer un phénomène d'élargissement des raies ...  
  
Select how to choose what you want to export : 'tag';'name';'id' :  
Or enter finish to finalize export >>>finish  
Entrez un nom de fichier :Exam  
Fichier Exam.tex exporté  
romain:Module $
```

Le fichier Exam.tex est alors créé et apparaît dans le dossier Module de l'application.

