Communauté Française de Belgique

Institut des Carrières Commerciales

Ville de Bruxelles

Rue de la Fontaine, 4

1000 Bruxelles

Rapport d'épreuve intégrée:

Volume 2

Application de gestion des inscriptions et des ressources scolaires

Épreuve intégrée réalisée en vue de l'obtention du titre de « Bachelier en Informatique de gestion »

RAMSAMY Bryan

2019 – 2020

Remerciements

Je tiens à remercier mes proches et mes amis qui m'ont soutenu durant ces trois années difficiles.
Tout d'abord mes frères Andy C et Alvin C, qui m'apportent leur soutien moral et leur amour inconditionnel en permanence. C'est grâce à eux que je n'ai pas sombré dans isolationnisme durant cette période de crise. Je remercie également mes parents.

Ensuite mes cousins et cousines tel que Shawn R, Kevin DR, Frederic DR, Adrien S qui m'ont aidé à me construire et à trouver un but dans la vie sans jamais rien n'avoir demandé en retour. Ils ont grandement contribué à mon évolution en tant que personne.

Je remercie également mes amis Maxime C et Mounia S qui m'ont toujours soutenu depuis des années, en particulier en cette période de crise, et qui n'hésitent jamais à me bousculer lorsque ceux-ci estiment que je m'apprête à m'engager sur une mauvaise voie.

Je n'oublie pas mes collègues de travail qui ont toujours été d'une grande bienveillance envers ma personne. En particulier mes collaborateurs directs, soit Mohand B et Shean M, qui m'ont fortement fait progresser au niveau de mes connaissances et de mes compétences personnelles et professionnelles. Je souhaite également citer mes supérieurs, Carolyn S, Theodora G et Benoît H, pour l'immense opportunité professionnelle qui me fût accordée ainsi que la confiance placée en moi. Petite mention également pour mes collègues m'ayant soutenu en dehors du monde professionnel, à savoir Gia Q, Andreja K, Maite L, Joelle L et surtout Diana M qui m'a aidé à peaufiner l'aspect juridique de mon projet.

Pour finir, je remercie mes collègues de l'Institut des Carrières Commerciales pour leur aide et leur collaboration, entre autres Lionel S, Cedric N, Caleb N, Fernando M, Dieudonné T, Nicolas D, Abdellah E, Cristian M, Najim E, Adel Z, Ayoub H pour ne citer que ceux-là, sans oublier Coralie O.

Table des matières

1. Glossaire

Conteneurisation

Structure de données, d'une classe, ou d'un type de données abstrait, dont les instances représentent des collections d'autres objets. Autrement dit, les conteneurs sont utilisés pour stocker des objets sous une forme organisée qui suit des règles d'accès spécifiques. On peut implémenter un conteneur de différentes façons, qui conduisent à des complexités en temps et en espace différentes. On choisira donc l'implémentation selon les besoins.[1]

DevOps

Combinant développement (Dev) et opérations (Ops), DevOps est l'union des personnes, des processus et des technologies destinés à fournir continuellement de la valeur aux clients.

DevOps permet la coordination et la collaboration des rôles autrefois cloisonnés (développement, opérations informatiques, ingénierie qualité et sécurité) pour créer des produits plus performants et plus fiables. En adoptant une culture DevOps ainsi que des pratiques et outils DevOps, les équipes peuvent mieux répondre aux besoins des clients, accroître la confiance suscitée par les applications qu'elles développent, et atteindre plus rapidement les objectifs de leur entreprise.[2]

Framework

Ensemble de composants logiciels qui permettent de créer le squelette d'un logiciel ou d'une application. Un framework est comparable à une boite à outils dans laquelle le développeur vient chercher les composantes dont il a besoin. C'est en fait un cadre de travail qui simplifie le travail des développeurs en leur offrant une structure d'ensemble.
Les frameworks fonctionnent par langage de programmation et permettent de développer tous types de supports : sites web, jeux, applications mobiles etc.[3]

[1] ROUSE, Margaret. Mise à jour : le 24-02-2016. « Conteneur (container) » sur *LeMagIT*. Site Web sur INTERNET. <https://www.lemagit.fr/definition/Conteneurs>. Dernière consultation : le 04-01-2020.
[2] MICROSOFT AZURE. 2020. *Qu'est-ce que le DevOps ?*. Site Web sur INTERNET. <https://azure.microsoft.com/fr-fr/overview/what-is-devops/>. Dernière consultation : le 06-08-2020.
[3] C., Florian. Mise à jour: le 10-06-2019. « Qu'est-ce qu'un framework ? » sur Wild Code School. Site Web sur INTERNET. <https://www.wildcodeschool.com/fr-FR/blog/quest-ce-quun-framework>. Dernière consultation : le 19-01-2020.

GitLab

Plateforme permettant d'héberger et de gérer des projets web de A à Z. Présentée comme la plateforme des développeurs modernes, elle offre la possibilité de gérer ses dépôts Git et ainsi de mieux appréhender la gestion des versions de vos codes sources.

Initialement connu pour sa capacité de gestion de versions des codes sources, Gitlab s'est développé au cours des dernières années pour devenir aujourd'hui un outil incontournable de gestion de projet web.[4]

GraphQL

GraphQL est un langage de requêtes pour API ainsi qu'un environnement pour exécuter ces requêtes. Il est défini par une spécification indépendante des langages de programmation et des protocoles de transport, dans le but de s'inscrire comme un nouveau standard dans le développement d'API.[5]

Object-Relational Mapping ou ORM

Technique de programmation informatique qui permet de simplifier l'accès à une base de données en proposant à l'informaticien des « objets » plutôt que d'accéder directement à des données relationnelles. Ce niveau d'abstraction supplémentaire fait correspondre le monde objet (programmation orientée objet) et le monde relationnel (les bases de données relationnelles classiques et massivement utilisées aujourd'hui).[6]

Pipeline

Permet de faire transiter un code entre plusieurs intermédiaires (des fonctions) pour le compléter ou le modifier.[7]

[4] REGNAULT, Camille. Mise à jour : le 24-02-2017. « GitLab, c'est quoi ? » sur *AXOPEN*. Site Web sur INTERNET. <https://blog.axopen.com/2017/02/gitlab-cest-quoi/>. Dernière consultation : le 14-08-2020.

[5] CALAMIER, Romain. Mise à jour : le 09-08-2018. « GraphQL: Et pour quoi faire ? » sur *OCTO talks*. Site Web sur INTERNET. <https://blog.octo.com/graphql-et-pourquoi-faire/>. Dernière consultation : le 16-08-2020.

[6] Christophe. Mise à jour : le 08-03-2018. « ORM » sur *Base de données*. Site Web sur INTERNET. <https://www.base-de-donnees.com/orm/>. Dernière consultation : le 13-08-2020.

[7] MARCILLAUD, Matthieu. Mise à jour : le 29-12-2008. « Qu'est-ce qu'un pipeline ? » sur *Programmer.Spip.net*. Site Web sur INTERNET. <https://programmer.spip.net/Qu-est-ce-qu-un-pipeline>. Dernière consultation : le 06-08-2020.

Runner

L'intégration continue va permettre de lancer les tests et les compilations directement sur le serveur via des pipelines. Les pipelines sont des groupes de jobs qui vont définir les scripts à exécuter sur le serveur.

Pour gérer des pipelines, il faut mettre en place un runner qui va gérer les jobs et les lancer automatiquement quand une branche sera envoyée sur le dépôt ou lorsqu'elle sera mergée, par exemple. Il est également possible de lancer les processus à la main ou changer complètement la configuration.[8]

VCS ou Version Control System (Système de gestion de version)

Système qui enregistre l'évolution d'un fichier ou d'un ensemble de fichiers au cours du temps de manière qu'on puisse rappeler une version antérieure d'un fichier à tout moment.

Dans le cas d'un dessinateur ou un développeur web souhaitant conserver toutes les versions d'une image ou d'une mise en page, un système de gestion de version est un outil qu'il est très sage d'utiliser. Il permet de ramener un fichier à un état précédent, de ramener le projet complet à un état précédent, de visualiser les changements au cours du temps, de voir qui a modifié quelque chose qui pourrait causer un problème, qui a introduit un problème et quand, et plus encore. Utiliser un VCS signifie aussi généralement qu'en cas d'erreur ou de perte des fichiers, il est facilement possible revenir à un état stable.[9]

[8] BRIDAY, Guillaume. Mise à jour : le 24-02-2018. « Installer et utiliser les GitLab Runners » sur *GuillaumeBriday.fr*. Site Web sur INTERNET. <https://guillaumebriday.fr/installer-et-utiliser-les-gitlab-runners>. Dernière consultation : le 06-08-2020.

[9] GIT. 2020. *1.1 Démarrage rapide - À propos de la gestion de version*. Site Web sur INTERNET. <https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version>. Dernière consultation : le 06-08-2020.

2. Introduction

L'épreuve intégrée est l'épreuve finale de tout étudiant de l'ICC. Le dernier test à passer avant d'être officiellement qualifié pour le monde professionnel de l'informatique.
Ayant pour ma part fait mon entré dans le milieu professionnel depuis plusieurs années tout en ayant acquis une expérience non négligeable. Je perçois l'épreuve comme étant un défi plus qu'une évaluation. Une façon de pousser un peu plus mes limites et de remettre en question mes connaissances.

Ce second volume du rapport de l'épreuve couvrira l'aspect développement plus en détails. Certains paragraphes ayant déjà abordé la question dans le premier volume seront repris pour y être développé plus en détail.

Contrairement au premier rapport s'adressant à n'importe quel lecteur, celui-ci s'adresse principalement aux développeurs avertis car il contient des détails bien plus techniques qui ne seront pas expliqué, conformément aux exigences imposées lors de l'évaluation du travail dans le but de ne pas rendre ce rapport trop indigeste.

Tout d'abord, les différentes technologies utilisées dans le cadre de ce projet seront énumérées et rapidement passée en revue.

Ensuite, le plan de programmation, déjà abordé dans le premier volume, sera également abordé de manière plus complète.

Ce rapport se conclura par le contenu du code source de l'application. L'application ayant beaucoup trop de lignes de codes, seules les parties les plus importantes seront explicitement reprises.

3. Technologies utilisées

Ce chapitre rend compte des différentes technologies utilisées dans le cadre de l'épreuve intégrée. Conformément aux consignes de l'épreuve, l'aspect théorique ne sera abordé que brièvement. Le contenu se focalisera sur leur application.

3.1. Python 3.8

Le langage utilisé dans le cadre de l'épreuve est le langage Python 3.8.
Au début, il fût envisagé de développer le projet dans un langage moderne, populaire et ayant espérance de vie. Les deux meilleures options étaient le langage Ruby, qui permet une grande souplesse et surtout de créer une syntaxe unique au sein même du code, et le langage Python, d'une souplesse tout aussi grande mais ayant l'atout de disposer d'une grande communauté et donc d'un large éventail d'extensions.[10]

Le projet n'ayant pas besoin d'un niveau de complexité allant jusqu'à exprimer le besoin de concevoir une syntaxe unique, ainsi que l'immense gamme d'extensions proposé par l'option alternative permettant de faciliter le travail du développeur, fit basculer le choix vers le langage Python 3.[11] Le langage Python 2 n'étant plus supporté depuis le 1er janvier 2020, il n'était cependant pas question d'utiliser cette version.[12]

[10] SAUNIER, Sébastien. Mise à jour : le 06-09-2018. « Pourquoi apprendre Ruby on Rails ? » sur *le wagon*. Site Web sur INTERNET. <https://www.lewagon.com/fr/blog/apprendre-ruby-on-rails>. Dernière consultation : le 24-08-2020.

[11] TAIEB, John. 2020. « Pourquoi apprendre Python ? » sur *apprendre-a-coder.com*. Site Web sur INTERNET. <https://apprendre-a-coder.com/pourquoi-apprendre-python/>. Dernière consultation : le 24-08-2020.

[12] BEKY, Ariane. Mise à jour : le 29-08-2019. « Python 2 : le clap de fin à haut risque » sur *Silicon.fr*. Site Web sur INTERNET. <https://www.silicon.fr/python-2-clap-fin-risque-259807.html>. Dernière consultation : le 24-08-2020.

3.2. Django 3.0

Lorsqu'un projet de cette envergure fût abordé, c'est naturellement vers Django 3.0 que le choix du framework se porta. Django étant le framework le plus populaire, le plus complet et le plus simple à utiliser parmi les framework Python.[13] Celui-ci s'accord parfaitement avec la version 3 de Python en plus d'être facilement extensible aux framework frontend.

Au sein de Django, et implicitement du containeur Python faisant tourner les services de Django, se trouvent plusieurs dépendances installées afin d'étendre les libraires disponibles et utilisées dans le cadre du projet. Voici une liste des principales dépendances installées :

- **django-extensions 3.0** : Permet la mise en place d'un serveur de débogage plus avancé, facilitant la phase de développement du projet.
- **django-paypal 1.0** : Permet l'utilisation d'une API PayPal homologuée, directement intégrée côté serveur sur l'application dans le but de permettre l'exécution de paiement depuis la plateforme.
- **gunicorn 20.0.4** : Permet la liaison entre le framework et le serveur web de production.
- **ipython 7.16** : Utilisé conjointement avec django-extensions et werkzeug dans le but de faciliter les pratiques de débogage lors de la phase de développement de l'application. Cette extension permet entre autres, l'utilisation d'un interpréteur Python plus avancé.
- **Pillow 7.2** : Nécessaire à Django afin d'effectuer le traitement d'image. Cette fonctionnalité n'est malheureusement pas intégrée dans le framework.
- **psycopg2 2.8** : Dépendance utilisée par l'ORM de Django afin de traduire les instructions Python en requêtes SQL. Cette dépendance est utilisée pour les bases de données Postgres utilisé dans ce cas-ci, dans un autre conteneur.
- **pytest 3.9** : Cette librairie ainsi que ces dépendances sont la base des tests effectuées au sein de l'application.
- **werkzeug 1.0** : Une ensemble de modules permettant à un développement facilité par des techniques de débogage avancé par rapport aux moyens standard mis à disposition par Django. Cette extension fonctionne conjointement avec ipython et django-extensions.

[13] Malick. Mise à jour : le 17-10-2018. « Quels sont vos frameworks Web Python préférés en 2018 ? Pourquoi ? » sur *Developpez.com*. Site Web sur INTERNET. <https://www.developpez.com/actu/229336/Quels-sont-vos-frameworks-Web-Python-preferes-en-2018-Pourquoi-Partagez-vos-avis/>. Dernière consultation : le 24-08-2020.

3.3. Docker 19.03 et docker-compose 1.26

Afin de rendre l'application entièrement portable et indépendante de son environnement, le principe de conteneurisation des services est d'application.[14]
Le moteur Docker 19.03 est utilisé pour la construction, la gestion et l'exécution des conteneurs.
Pour ce qui est de docker-compose, c'est la version 1.26 qui a été favorisé par sa syntaxe plus explicite et facile d'utilisation.

3.4. PostgreSQL 12.3-alpine

La base de données conteneurisée utilisée dans le cadre du projet est une base de données SQL de type Postgres. La version actuellement utilisée est la version allégée 12.3. Cette version permet un bon compromis entre une base de données robuste, fiable et adaptée à Django, tout étant moins gourmande en ressources, notamment au niveau de l'espace de stockage du service Postgres en lui-même.[15]

3.5. Graphene 2.12

Pour la mise en place de l'API, c'est l'extension Django Graphene-Django 2.12 qui fut choisie.[16]
Graphene offre une API GraphQL parfaitement intégrée à Django par le billet de requetés et de mutations simple à initialisée. Il est également possible de rendre chaque action de l'API documentée afin de laisser une API interactive, s'adaptant aux besoins de l'utilisateur, en un minimum de temps et d'effort de développement. La maintenance de celle-ci s'en retrouve également facilitée.

[14] ROUSE, Margaret. Mise à jour : le 24-02-2016. « Conteneur (container) » sur *LeMagIT*. Site Web sur INTERNET. <https://www.lemagit.fr/definition/Conteneurs>. Dernière consultation : le 04-01-2020.
[15] D-BOOKER. 2012-2020. *PostgreSQL : Robuste, performant, stable et open-source*. Site Web sur INTERNET. <https://www.d-booker.fr/content/72-postgresql>. Dernière consultation : le 24-08-2020.
[16] CALAMIER, Romain. Mise à jour : le 09-08-2018. « GraphQL: Et pour quoi faire ? » sur *OCTO talks*. Site Web sur INTERNET. <https://blog.octo.com/graphql-et-pourquoi-faire/>. Dernière consultation : le 16-08-2020.

3.6. Nginx 1.19-alpine

Pour l'environnement de déploiement, c'est le service NGINX conteneurisé qui se charge de la gestion des requêtes émises et reçues par le projet au lieu du serveur de développement intégrée dans Django. Celui-ci pourvoit des performances de traitement bien supérieurs en plus d'être le serveur web le plus populaire du moment.[17] De plus, sa syntaxe et sa modularité en font un service web parfait pour le projet et sa nature hautement portable.

3.7. GitLab 8.0

Pour terminer, c'est le dépôt GitLab.com et sa version 8.0 du logiciel qui fût utilisé pour le VCS du projet, mais pas seulement.[18]

GitLeb également de registre permettant de stocker les différentes versions des images des conteneurs de l'application. Ces images sont générées lors du processus CI/CD, plus particulièrement lors de la phase de compilation et de la phase de tests du projet.

Finalement, GitLab gère également le processus CI/CD en lui-même grâce son intégration des pipelines et de ses runners partagées utilisées pour l'exécution des étapes.

[17] Bruno. Mise à jour : le 11-06-2019. « Nginx est maintenant le serveur web le plus utilisé par les sites les plus fréquentés au monde » sur *Developpez.com*. Site Web sur INTERNET. <https://web.developpez.com/actu/265652/Nginx-est-maintenant-le-serveur-web-le-plus-utilise-par-les-sites-les-plus-frequentes-au-monde-devant-Apache-et-Microsoft-IIS-selon-W3Tech/>. Dernière consultation : le 24-08-2020.

[18] REGNAULT, Camille. Mise à jour : le 24-02-2017. « GitLab, c'est quoi ? » sur *AXOPEN*. Site Web sur INTERNET. <https://blog.axopen.com/2017/02/gitlab-cest-quoi/>. Dernière consultation : le 14-08-2020.

4. Plan de programmation

4.1. Principes CI/CD

Le projet fut abordé selon les principes DevOps CI/CD.

> *« L'approche CI/CD permet d'augmenter la fréquence de distribution des applications grâce à l'introduction de l'automatisation au niveau des étapes de développement des applications. Les principaux concepts liés à l'approche CI/CD sont l'intégration continue, la distribution continue et le déploiement continu. L'approche CI/CD représente une solution aux problèmes posés par l'intégration de nouveaux segments de code pour les équipes de développement et d'exploitation (ce qu'on appelle en anglais « integration hell », ou l'enfer de l'intégration).*
> *Plus précisément, l'approche CI/CD garantit une automatisation et une surveillance continues tout au long du cycle de vie des applications, des phases d'intégration et de test jusqu'à la distribution et au déploiement. Ensemble, ces pratiques sont souvent désignées par l'expression « pipeline CI/CD » et elles reposent sur une collaboration agile entre les équipes de développement et d'exploitation. »* [19]

Ces principes fussent appliqués grâce au service repository en ligne mise à disposition par GitLab qui permet non seulement de stocker le code source de l'application et de son VCS, les images conteneurs source, mais également la mise en place de pipeline, s'activant à l'aide de runners et qui permettent de pousser le nouveau code fraichement développé. Mais également de lui faire passer une série de tests unitaires et fonctionnels automatisés (qui ont été écrit à l'avance) et de fusionner à la version principale en cas de succès. Ainsi, le développement en équipe se retrouve grandement facilité en plus d'être plus rapide.

Dans un contexte réel de production, un serveur local de repository comme le logiciel serveur de GitLab serait mis en place au lieu d'envoyer le code sur la plateforme en ligne. Il en va de même pour les runners qui devraient tourner depuis des machines locales, voir idéalement depuis des conteneurs. Mais toutes ces démarches demandent une infrastructure avancée et onéreuse.

[19] REDHAT. 2020. *Qu'est-ce que l'approche CI/CD ?*. Site Web sur INTERNET. <https://www.redhat.com/fr/topics/devops/what-is-ci-cd>. Dernière consultation : le 06-08-2020.

Dans le cas de ce projet, c'est donc la plateforme GitLab.com et ses runners publiques partagées qui sont utilisées, ce qui peut causer des délais importants dans le processus CI.
Une fois le code ayant passé toute la phase CI, la phase CD s'enclenche, provoquant une succession d'actions effectuées par le runner sur le serveur de production afin d'importer et de déployer le résultat du code, ayant déjà passé tous les tests dans la phase CI. Il est ainsi possible de passer du développement à la production en seulement quelques minutes sans entrer la moindre ligne de code supplémentaire.

4.2. Principes SOLID

Le projet fût bâti selon les principes SOLID.[20]

- Le principe de responsabilité unique fût appliqué dans la mesure du possible afin de rendre les différentes fonctions et les classes plus indépendantes les unes des autres. Cela permet des tests ciblés sur des fonctions élémentaires et de construire le projet selon un schéma pyramidale. Ainsi, si une fonction au niveau fonctionne correctement, toutes les fonctions n-1 fonctionnent également et les bugs devront ainsi se trouver aux fonctions n ou n+1. Ce principe vient par ailleurs de pair avec le principe DRY (voir Chapitre 4.1.4. Le principe DRY).
  Ce principe est toutefois limité à certains points. Le projet ayant un niveau de complexité variable, certaines situations font exception à cette règle afin de gagner du temps.
- Le principe d'ouverture et de fermeture est également appliqué. L'application est complètement réfractaire à la modification mais ouverte à l'extension. L'héritage multiple des classes que permet le langage Python est ainsi mit à profit afin de pouvoir aisément ajouter une classe dotée d'un niveau d'abstraction supérieur en réutilisant d'autres classes sans devoir modifier ces dernières.
- Le principe de substitution n'est pas appliqué pour la bonne et simple raison qu'il n'existe aucune classe concrète héritant d'une autre. L'objet B ne peut donc pas remplacer l'objet A étant donné que ce premier n'existe pas. Si c'était le cas, ce principe doit impérativement être respecté dans l'optique de maintenir le code le plus modulable possible.
- Le principe de ségrégation des interfaces est également appliqué afin d'éviter tout risque de restriction d'accès croisé. Un étudiant ne doit en aucun cas utiliser la même interface qu'un manager ou qu'un administrateur. Les vues furent toutefois exportées dans des fonctions indépendantes afin de réutiliser le code et éviter la redondance, en accord avec le premier

[20] VIALATTE, Phillipe. Mise à jour : le 21-10-2008. « Bonnes pratiques objet en .net : Introduction aux principes SOLID » sur *Developpez .com*. Site Web sur INTERNET. <https://philippe.developpez.com/articles/SOLIDdotNet/>. Dernière consultation : le 10-08-2020.

principe SOLID et le principe DRY expliqué dans le prochain paragraphe (voir Chapitre 4.1.4. Le principe DRY).

- Finalement, le principe d'inversion des dépendances fût également appliqué en accord avec le premier et le quatrième principe SOLID. En outre, la règle d'abstraction permet également un code plus modulaire en plus d'une optimisation du code ainsi que de la base de données.

4.3. Principe DRY

Finalement, n'oublions pas le principe DRY.[21]

Le principe DRY en corrélation avec le premier principe SOLID (voir Chapitre 4.1.3. Les principes SOLID) est basé sur le fait d'éviter la redondance au sein du code. Ce principe fut appliqué lorsqu'une fonctionnalité se voit utilisée à plusieurs endroits du code et si celle-ci contient plus de 2 lignes. Si une instruction conserve un degré de complexité simple et une syntaxe relativement courte, l'application du principe DRY n'a pas cours afin d'éviter du travail inutile. Ce principe permet également de tester les fonctions les plus élémentaires afin de respecter le schéma pyramidal expliqué dans le paragraphe précédent.

Le framework Django est d'ailleurs bâti selon ce principe en exploitant au maximum les différents niveaux d'abstraction qu'offre le langage Python.[22]

4.4. Standards PEP 8

Les standards PEP 8 ont rigoureusement été respectés à la lettre lors de l'écriture du code[23]. Ceci s'est notamment fait grâce à un correcteur sémantique python intégrée dans l'éditeur de code utilisé lors de l'élaboration du code source. Ces standards permettent une lecture et une compréhension simple et efficace du code source par n'importe quel développeur python averti.

Ceci va de concert avec la philosophie selon laquelle le code doit être modulable, ouvert à l'extension et portable.

[21] ORSIER, Bruno. Mise à jour : le 03-04-2008. « Comment éviter les duplications de code : le principe DRY (Do not Repeat Yourself) » sur *Developppez.com*. Site Web sur INTERNET. <https://bruno-orsier.developpez.com/principes/dry/>. Dernière consultation : le 10-08-2020.

[22] DJANGO PROJECT. 2020. « Ne vous répétez pas (DRY) » sur *Django Documentation*. Site Web sur INTERNET. <https://docs.djangoproject.com/fr/3.0/misc/design-philosophies/#don-t-repeat-yourself-dry>. Dernière consultation : le 10-08-2020.

[23] PYTHON. 2020. *PEP 8 -- Style Guide for Python Code*. Site Web sur INTERNET. <https://www.python.org/dev/peps/pep-0008/>. Dernière consultation : le 24-08-2020.

5. Code source

Le code source de l'application ne contient pas moins de 15'000 lignes au moment de la rédaction de ce rapport. Celle-ci n'étant par ailleurs pas encore terminé, il se peut que de nombreuses lignes viennent s'y ajouter.
Afin de ne pas rendre ce rapport indigeste, seules les parties les plus critiques du code seront reprises.

Ce chapitre est consacré au code pur. Aucune explication supplémentaire ne sera fournie. Par ailleurs, le code est entièrement commenté en anglais.
Il est également possible que le code final ne corresponde pas entièrement au code reprit dans les prochains paragraphes.

5.1. Conteneurisation

docker-compose.yml

```yaml
version: '3.8'

services:
  web:
    container_name: serina_web
    build:
      context: ./serina-project
      dockerfile: Dockerfile
    image: serina_django

    networks:
      - serina_network
    depends_on:
      - db
    restart: "no"
    ports:
      - 8000:8000

    volumes:
      - type: bind
        source: ./serina-project
        target: /usr/src/app

    env_file:
      - .env.dev
```

```
    command: bash scripts/runserver_plus.sh


  db:
    container_name: serina_db
    image: postgres:12.3-alpine
    networks:
      - serina_network
    restart: unless-stopped

    volumes:
      - type: volume
        source: postgres_database_dev
        target: /var/lib/postgresql/data

    environment:
      POSTGRES_DB: 'postgres_database_dev'
      POSTGRES_USER: 'postgres_user_dev'
      POSTGRES_PASSWORD: 'postgres_password_dev'


networks:
  serina_network:


volumes:
  postgres_database_dev:
```

docker-compose.prod.yml

```
version: '3.8'

services:
  web:
    container_name: serina_web
    build:
      context: ./serina-project
      dockerfile: Dockerfile.prod
    image: serina_django

    networks:
      - serina_network
    depends_on:
      - db
    restart: always
```

```
    expose:
      - 8000

    volumes:
      - type: volume
        source: web_media_volume
        target: /home/app/web/media
      - type: volume
        source: web_static_volume
        target: /home/app/web/static

    env_file:
      - .env.prod

    command: gunicorn serina.wsgi:application --bind 0.0.0.0:8000


  db:
    container_name: serina_db
    image: postgres:12.3

    networks:
      - serina_network
    restart: always

    volumes:
      - type: volume
        source: postgres_database_prod
        target: /var/lib/postgresql/data

    env_file:
      - .env.prod.db

  nginx:
    container_name: serina_nginx
    build:
      context: ./nginx
      dockerfile: Dockerfile

    networks:
      - serina_network
    depends_on:
      - web
    restart: always
    ports:
      - 80:80

    volumes:
      - type: volume
```

```
        source: web_media_volume
        target: /home/app/web/media
      - type: volume
        source: web_static_volume
        target: /home/app/web/static


networks:
  serina_network:


volumes:
  web_media_volume:
  web_static_volume:
  postgres_database_prod:
```

serina-project/Deockerfile

```
# Pull official python image

FROM python:3.8


# Set environnement variables

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1


# Create and set work directory

WORKDIR /usr/src/app


# Update apt and install apt dependencies

RUN apt update -y
RUN apt install gettext netcat -y


# Upgrade pip and install pip dependencies

RUN pip install --upgrade pip
COPY requirements.txt .
RUN pip install -r requirements.txt
```

```
# Copy source code

COPY . .


# Run entrypoint.sh

ENTRYPOINT ["/usr/src/app/scripts/entrypoint.sh"]
```

serina-project/scripts/entrypoint.sh

```
#!/bin/sh

if [ "$DATABASE" = "postgres" ]
then
    echo "Waiting for postgres database..."

    while ! nc -z $DATABASE_HOST $DATABASE_PORT; do
      sleep 1
    done

    echo "PostgreSQL started"
fi


exec "$@"
```

serina-project/Dockerfile.prod

```
############
# BUILDER #
############

# Pull official python image

FROM python:3.8 as builder

# Create and set work directory

WORKDIR /usr/src/app
```

```
# Set environnement variables

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1


# Update apt and install apt dependencies

RUN apt update -y
RUN apt install netcat -y


# Upgrade pip and lint

RUN pip install --upgrade pip
# RUN pip install flake8
# COPY . .
# RUN flake8 --ignore=E501,F401 .


# Install pip dependencies

COPY requirements.txt .
RUN pip wheel --no-cache-dir --no-deps --wheel-dir /usr/src/app/wheels -
r requirements.txt


#########
# FINAL #
#########


# Pull official python image

FROM python:3.8


# Create directory for the app user

RUN mkdir -p /home/app


# Create the app user

# RUN addgroup --system app && adduser --system app --group app
RUN groupadd app && useradd -g app -s /bin/bash app


# Create the appropriate directories
```

```
ENV HOME=/home/app
ENV APP_HOME=/home/app/web
RUN mkdir $APP_HOME
RUN mkdir $APP_HOME/media
RUN mkdir $APP_HOME/static
WORKDIR $APP_HOME


# Install dependencies

RUN apt update && apt install gettext libpq-dev netcat -y
COPY --from=builder /usr/src/app/wheels /wheels
COPY --from=builder /usr/src/app/requirements.txt .
RUN pip install --upgrade pip
RUN pip install --no-cache /wheels/*


# Copy project

COPY . $APP_HOME


# Chown all the files to the app user

RUN chown -R app:app $APP_HOME


# Change to the app user

USER app


# Run entrypoint.prod.sh

ENTRYPOINT ["/home/app/web/scripts/entrypoint.prod.sh"]
```

serina-project/entrypoint.prod.sh

```
#!/bin/sh

if [ "$DATABASE" = "postgres" ]
then
    echo "Waiting for postgres database..."

    while ! nc -z $DATABASE_HOST $DATABASE_PORT; do
      sleep 1
```

```
    done

    echo "PostgreSQL started"
fi


# Migrate all migrations

python manage.py migrate


# Collect all static files

python manage.py collectstatic --no-input --clear


exec "$@"
```

serina-project/scripts/runserver_plus.sh

```
#!/bin/bash

# Automatically restart the runserver_plus process if it stops with an error c
ode
# This is for development purposes

while true; do
    if ! python3 manage.py runserver_plus 0.0.0.0:8000; then
        printf "_____________________________________________________\n>>> RUNSERV
ER_PLUS EXITED WITH A NON-
0 STATUS <<<\n>>>          Restarting in 3 seconds          <<<\n"
        sleep 3;
    else
        break
    fi
done
```

serina-project/scripts/createsuperadmin.sh

```
#!/bin/bash

# Set variables for coloured text
```

```
RED='\033[0;31m'
GREEN='\033[0;32m'
ORANGE='\033[0;33m'
NC='\033[0m'


# Create default superadmin account

echo "from django.contrib.auth.models import User; User.objects.create_superus
er('superadmin', 'superadmin@serina.com', 'superadmin')" | docker-
compose run --rm web python3 manage.py shell_plus


# Print confirmation message and exit with 1 if failed

if [ $? == 0 ]
then
    echo "Superadmin account successfully created"
    echo "${ORANGE}Please change the default superadmin username and password
immediately !${NC}"
else
    echo "${RED}Error: superadmin creation failed. Try with the shell_plus uti
lity.${NC}"
    echo -e "\tdocker-compose run --rm web python3 manage.py shell_plus"
    exit 1
fi


# Add superadmin to Administrator group

echo "from django.contrib.auth.models import User; from registration.utils.gro
ups import promote_to_administrator; superadmin = User.objects.get(username='s
uperadmin'); promote_to_administrator(superadmin); superadmin.save()" | docker
-compose run --rm web python3 manage.py shell_plus


# Print confirmation message and exit with 2 if failed

if [ $? == 0 ]
then
    echo "Superadmin successfully added to Administrator group"
    exit 0
else
    echo "${RED}Error: superadmin could not be added to the Administrator grou
p. Try with the shell_plus utility.${NC}"
    echo -e "\tdocker-compose run --rm web python3 manage.py shell_plus"
    exit 2
fi
```

serina-project/scripts/deployment.sh

```
#!/bin/bash

# Deployment script


# Go to projet root

cd ICC-EpreuveIntegree_2019-2020


# Pull source code from repository

git pull


# Rebuild and restart the production docker containers

docker-compose -f docker-compose.prod.yml down
docker-compose -f docker-compose.prod.yml up -d --build
```

nginx/Dockerfile

```
FROM nginx:1.19.0-alpine

RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d
```

nginx/nginx.conf

```
upstream serina {
    server web:8000;
}

server {

    listen 80;

    location / {
```

```
        proxy_pass http://serina;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_redirect off;
    }

    location /static/ {
        alias /home/app/web/static/;
    }

    location /media/ {
        alias /home/app/web/media/;
    }
}
```

5.2. Pipeline

.gitlab-ci.yml

```
image: docker:stable


services:
  - docker:19.03.0-dind


stages:
  - build
  - test
  - deploy


Build and push stage:
  stage: build
  script:
    - docker login --username $CI_REGISTRY_USER --
password "$CI_BUILD_TOKEN" $CI_REGISTRY
    - docker pull $CI_REGISTRY_IMAGE:latest || true
    - docker build --cache-from $CI_REGISTRY_IMAGE:latest -
t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA ./serina-project
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
  only:
    - master
    - tests


Test stage:
  stage: test
  variables:
    SECRET_KEY: '2(o^nzfzonpqr*5d-6-trf1-l+s4zlvq-
(v1u*f)acfb&h6$#!'  # Dummy key for pipeline
    DJANGO_ALLOWED_HOSTS: 'localhost 127.0.0.1 [::1]'
  script:
    - docker login --username $CI_REGISTRY_USER --
password "$CI_BUILD_TOKEN" $CI_REGISTRY
    - docker pull $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
    - docker run --rm -e SECRET_KEY='$SECRET_KEY' -
e DJANGO_ALLOWED_HOSTS='$DJANGO_ALLOWED_HOSTS' $CI_REGISTRY_IMAGE:$CI_COMMIT_S
HA pytest
    - docker tag $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA $CI_REGISTRY_IMAGE:latest
    - docker push $CI_REGISTRY_IMAGE:latest
  only:
    - master
```

```
    - tests


Deploy stage:
  stage: deploy
  script:
    - chmod 600 $DEPLOYMENT_KEY_FILE
    - ssh -i "$DEPLOYMENT_KEY_FILE" -o "StrictHostKeyChecking=no" -
o "IdentitiesOnly=yes" bryan@78.46.198.120 "bash ICC-EpreuveIntegree_2019-
2020/serina-project/scripts/deployment.sh"
  only:
    - master
```

5.3. Django

serina-project/requirements.txt

```
asgiref==3.2.10
Django==3.0.8
django-crispy-forms==1.9.2
django-extensions==3.0.2
django-import-export==2.3.0
django-paypal==1.0.0
graphene-django==2.12.1
gunicorn==20.0.4
ipython==7.16.1
mixer==6.1.3
Pillow==7.2.0
psycopg2-binary==2.8.5
pytest==6.0.1
pytest-cov==2.10.1
pytest-django==3.9.0
pytz==2020.1
werkzeug==1.0.1
```

serina-project/serina/settings.py

```
"""
Django settings for serina project.

Generated by 'django-admin startproject' using Django 3.0.8.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.0/ref/settings/
"""

import os

from django.contrib.messages import constants as messages


# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

```
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.environ.get('SECRET_KEY')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = int(os.environ.get("DEBUG", default=0))


ALLOWED_HOSTS = os.environ.get("DJANGO_ALLOWED_HOSTS").split(" ")


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Third-parties apps
    'crispy_forms',
    'django_extensions',
    'graphene_django',
    'import_export',
    'paypal.standard.ipn',

    # Project apps
    'api',
    'management',
    'rating',
    'registration',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'serina.urls'
```

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'serina.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': os.environ.get('DATABASE_ENGINE', "django.db.backends.sqlite
3"),
        'NAME': os.environ.get('DATABASE_NAME', os.path.join(BASE_DIR, "db.sql
ite3")),
        'USER': os.environ.get('DATABASE_USER', "user"),
        'PASSWORD': os.environ.get('DATABASE_PASSWORD', "password"),
        'HOST': os.environ.get('DATABASE_HOST', "localhost"),
        'PORT': os.environ.get('DATABASE_PORT', "5432"),
    }
}


# Graphene

GRAPHENE = {
    'SCHEMA': 'api.schema.schema'
}

# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
```

```
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilari
tyValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidato
r',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidat
or',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValida
tor',
    },
]


# URLs

LOGIN_URL = '/registration/login/'
LOGIN_REDIRECT_URL = '/'

# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en'

TIME_ZONE = 'Europe/Brussels'

USE_I18N = True

USE_L10N = True

USE_TZ = True

LOCALE_PATHS = [
    os.path.join(BASE_DIR, 'locale')
]

LANGUAGES = [
    ('en', 'English'),
    ('fr', 'French'),
    ('nl', 'Dutch'),
]


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/
```

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')


# Paypal settings

PAYPAL_RECEIVER_EMAIL = os.environ.get('PAYPAL_RECEIVER_EMAIL')

PAYPAL_TEST = True


# SMTP configuration

EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
EMAIL_HOST = os.environ.get('EMAIL_HOST')
EMAIL_USE_TLS = True
EMAIL_PORT = os.environ.get('EMAIL_PORT')
EMAIL_HOST_USER = os.environ.get('EMAIL_HOST_USER')
EMAIL_HOST_PASSWORD = os.environ.get('EMAIL_HOST_PASSWORD')


# Mails

CONTACT_MAILS = {
    "administrators": "administrators@serina.com",
    "support": "support@serina.com",
}


# App settings

SUCCESS_SCORE_THRESHOLD = 50


# Messages

MESSAGE_TAGS = {messages.ERROR: 'danger'}
```

serina-project/serina/urls.py

```
from django.conf import settings
from django.conf.urls import url
```

```
from django.conf.urls.i18n import i18n_patterns
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import include


urlpatterns = [
    url(r"^i18n/", include("django.conf.urls.i18n")),
]

urlpatterns += i18n_patterns(
    url(r"^", include('registration.urls')),
    url(r"^admin/", admin.site.urls),
    url(r"^api/", include('api.urls')),
    url(r"^management/", include('management.urls')),
    url(r"^paypal/", include('paypal.standard.ipn.urls')),
    url(r"^rating/", include('rating.urls')),
    prefix_default_language=True,
) + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT) \
  + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)


admin.site.site_header = "SERINA Back-Office"
admin.site.site_title = "Administration"
admin.site.index_title = "Superintent application for Educational " \
                         "Resources, Inscriptions and Network Administration"
```

5.3.1. Module d'inscription

serina-project/registration/models/degree_rr.py

```
from django.core.exceptions import ValidationError
from django.db import models
from django.shortcuts import reverse
from django.utils.translation import ugettext as _

from . import resource
from .student_rr import StudentRegistrationReport

from management.models import Degree


class DegreeRegistrationReport(resource.FrontOfficeResource):
    """Model definition for DegreeRegistrationReport.
```

```
    Degree Registration Report of a degree to which the student registered.
    Contains all the related data of the student progression in the degree.
    """

    student_rr = models.ForeignKey(
        StudentRegistrationReport,
        on_delete=models.CASCADE,
        related_name="degrees_rrs",
        verbose_name=_("Student"),
    )
    degree = models.ForeignKey(
        Degree,
        on_delete=models.CASCADE,
        related_name="students_registrations",
        verbose_name=_("Registration degree")
    )
    date_start = models.DateField(
        null=True,
        blank=True,
        verbose_name=_("Start date"),
    )
    date_end = models.DateField(
        null=True,
        blank=True,
        verbose_name=_("End date"),
    )

    class Meta:
        """Meta definition for DegreeRegistrationReport."""

        verbose_name = _('Degree Registration Report')
        verbose_name_plural = _('Degrees Registration Reports')

    @property
    def partially_approved(self):
        """Check if at least one of the related modules_rr is approved.

        Return 'None' if  there is no modules_rrs in the degree_rr.
        """

        partially_approved = None

        for module_rr in self.modules_rrs.all():
            partially_approved = False

            if module_rr.approved:
                partially_approved = True
                break
```

```
        return partially_approved

    @property
    def fully_approved(self):
        """Check if all the related modules_rr are approved.

        Return 'None' if  there is no modules_rrs in the degree_rr.
        """

        fully_approved = None

        for module_rr in self.modules_rrs.all():
            fully_approved = True

            if not module_rr.approved:
                fully_approved = False
                break

        return fully_approved

    @property
    def partially_payed(self):
        """Check if at least one of the related modules_rr is approved.

        Return 'None' if  there is no modules_rrs in the degree_rr.
        """

        partially_payed = None

        for module_rr in self.modules_rrs.all():
            partially_payed = False

            if module_rr.payed:
                partially_payed = True
                break

        return partially_payed

    @property
    def fully_payed(self):
        """Check if all the related modules_rr are payed.

        Return 'None' if  there is no modules_rrs in the degree_rr.
        """

        fully_payed = None

        for module_rr in self.modules_rrs.all():
            fully_payed = True
```

```
            if not module_rr.payed:
                fully_payed = False
                break

        return fully_payed

    @property
    def academic_years(self):
        """Display the academic years of the sutdent's degree."""

        academic_years = self.date_start.strftime("%Y")

        if self.date_end:
            academic_years += " - {}".format(self.date_end.strftime("%Y"))

        return academic_years

    @property
    def student_graduated(self):
        """Check if the student succeeded all the degree's modules."""

        graduated = True

        for module_rr in self.modules_rrs.all():
            if not module_rr.succeeded:
                graduated = False
                break

        return graduated

    @property
    def average_score(self):
        """Compute the average score of the student."""

        return resource.modules_average_score(self)

    @property
    def total_expenses(self):
        """Compute the total expenses of the student for this degree."""

        total_expenses = 0
        for module_rr in self.modules_rrs.all():
            total_expenses += module_rr.module.charge_price

        return total_expenses

    def __str__(self):
        """Unicode representation of DegreeRegistrationRappport."""
```

```
        return "[{}] {}'s degree registration for {}".format(
            self.pk,
            self.student_rr.created_by.get_full_name(),
            self.degree.title,
        )

    def get_absolute_url(self):
        """Return absolute url for DegreeRegistrationRappport."""

        return reverse("degree_rr_detailview", kwargs={"pk": self.pk})
```

serina-project/registration/models/module_rr.py

```
from django.core.exceptions import ValidationError
from django.core.validators import MaxValueValidator, MinValueValidator
from django.db import models
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.shortcuts import reverse
from django.utils.translation import ugettext as _

from .degree_rr import DegreeRegistrationReport
from .resource import FrontOfficeResource
from .student_rr import StudentRegistrationReport

from management.models import Course, Module


class ModuleRegistrationReport(FrontOfficeResource):
    """Model definition for ModuleRegistrationReport.

    Model Registration Report of a model to which the student registered.
    Contains all the related data of the student progression in the module.
    """

    student_rr = models.ForeignKey(
        StudentRegistrationReport,
        on_delete=models.CASCADE,
        related_name="modules_rrs",
        verbose_name=_("Student Registration Report"),
    )
    degree_rr = models.ForeignKey(
        DegreeRegistrationReport,
        null=True,
```

```
        blank=True,
        on_delete=models.CASCADE,
        related_name="modules_rrs",
        verbose_name=_("Degree Registration Report")
    )
    module = models.ForeignKey(
        Module,
        on_delete=models.CASCADE,
        related_name="modules_rrs",
        verbose_name=_("Registration module")
    )
    course = models.ForeignKey(
        Course,
        null=True,
        blank=True,
        on_delete=models.CASCADE,
        related_name="modules_rrs",
        verbose_name=_("Course")
    )
    date_start = models.DateField(
        null=True,
        blank=True,
        verbose_name=_("Start date"),
    )
    date_end = models.DateField(
        null=True,
        blank=True,
        verbose_name=_("End date"),
    )
    date_payed = models.DateTimeField(
        null=True,
        blank=True,
        verbose_name=_("Payment date"),
    )
    nb_attempt = models.PositiveIntegerField(
        default=0,
        verbose_name=_("Student's attempt number")
    )  # TODO: Add max_attempt value from settings
    final_score = models.DecimalField(
        null=True,
        blank=True,
        max_digits=5,
        decimal_places=2,
        validators=[MinValueValidator(0), MaxValueValidator(100)],
        verbose_name=_("Final score")
    )

    MODULE_REGISTRATION_REPORT_STATUS = [
        ("PENDING", _('Pending')),
```

```
        ("DENIED", _('Denied')),
        ("APPROVED", _('Approved')),
        ("PAYED", _('Payed')),
        ("COMPLETED", _('Completed')),
        ("EXEMPTED", _('Exempted')),
    ]
    status = models.CharField(
        max_length=9,
        choices=MODULE_REGISTRATION_REPORT_STATUS,
        default="PENDING",
        verbose_name=_("Status")
    )

    class Meta:
        """Meta definition for ModuleRegistrationReport."""

        verbose_name = _('Module Registration Report')
        verbose_name_plural = _('Modules Registration Reports')

    # @property  # FIXME: School years from date_start and date_end
    # def school_year(self):
    #     """Compute the schoolyear of the module."""

    #     start = self.date_start
    #     end = self.date_end
    #     return start.strftime("%Y")

    @property
    def payed(self):
        """Check if the module registration request has been payed.

        A payed request is either payed or completed.
        """

        return self.status == "PAYED" or self.status == "COMPLETED"

    @property
    def approved(self):
        """Check if the module registration request has been approved.

        An approved request is either approved, payed or completed.
        """

        return self.status == "APPROVED" or self.payed

    @property
    def succeeded(self):
        """Check if the student succeeded the module modules.
```

```
        The module succeess is not valid if the student didn't payed his/her
        registration to it.
        """

        return self.status == "EXEMPTED" \
               or (self.status == "COMPLETED" and self.final_score >= 50)

    def __str__(self):
        """Unicode representation of ModuleRegistrationReport."""

        return "[{}] {}'s module registration for {} ({})".format(
            self.pk,
            self.student_rr.created_by.get_full_name(),
            self.module.title,
            self.status,
        )

    def get_absolute_url(self):
        """Return absolute url for DegreeRegistrationReport."""

        return reverse('module_rr_detailview', kwargs={"pk": self.pk})


@receiver(post_save, sender=DegreeRegistrationReport)
def generate_all_modules_rrs_of_degree_rr(sender, instance, **kwargs):
    """When a DegreeRegistrationReport is created, all the related
    ModuleRegistrationReports of the related modules are generated too and
    linked to the StudentRegistrationReport.

    NOTE: This couldn't be done in the DegreeRegistrationReport.save() because
    of a circular import issue.
    """

    for module in instance.degree.modules.all():
        for module_rr in module.modules_rrs.all():
            if module_rr.student_rr == instance.student_rr:
                if module_rr.status == "DENIED" or module_rr.status == "COMPLE
TED":
                    pass



        if module.modules_rrs.filter :
            pass
        else:
            pass
        ModuleRegistrationReport.objects.create(
            student_rr=instance.student_rr,
            degree_rr=instance,
```

```
            module=module,
        )
```

serina-project/registration/models/ressource.py

```
from django.db import models
from django.utils.translation import ugettext as _


class FrontOfficeResource(models.Model):
    """Model definition for FrontOfficeResource.

    A ressource contains a creation and last update timestamp.
    The FrontOfficeResource model is inherited by each front-office model.
    """

    date_created = models.DateTimeField(auto_now_add=True,
                                        verbose_name=_('Created on'))
    date_updated = models.DateTimeField(auto_now=True,
                                        verbose_name=_('Updated on'))
    notes = models.TextField(null=True, blank=True,
                             verbose_name=_("Additional notes"))

    class Meta:
        """Meta definition for FrontOfficeResource."""

        abstract = True
        ordering = ("-date_updated", "-date_created")


def modules_average_score(student_or_degree_rr):
    """Compute the average score the student got for each of his/her finished
    module.

    If the student didn't followed a single module, the result will be -1.
    """

    sum_score = 0
    total_module = student_or_degree_rr.modules_rrs \
                                       .filter(final_score__isnull=False) \
                                       .count()

    if total_module > 0:
        for module_student_or_degree_rr in student_or_degree_rr.modules_rrs \
                                               .filter(final_score__isnull=False):
```

```
            sum_score += module_student_or_degree_rr.final_score

        return sum_score / total_module
    else:
        return -1
```

serina-project/registration/models/student_rr.py

```
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError
from django.db import models
from django.shortcuts import reverse
from django.utils.translation import ugettext as _

from . import resource


class StudentRegistrationReport(resource.FrontOfficeResource):
    """Model definition for StudentRegistrationReport.

    Report containing all the information related to a guest registered user
    whom wants to be promoted as a student in order to follow his/her wished
    degree(s) and/or module(s).
    """

    created_by = models.OneToOneField(  # TODO: Add validator: guest/student
        User,
        on_delete=models.CASCADE,
        related_name="student_rr",
        verbose_name=_("Student"),
    )
    birthday = models.DateField(verbose_name=_("Birthday date"))
    nationality = models.CharField(max_length=50,
                                   verbose_name=_("Nationality"))
    address = models.CharField(max_length=255, verbose_name=_("Address"))
    additional_address = models.CharField(
        max_length=255,
        null=True,
        blank=True,
        verbose_name=_("Additional address"),
    )
    postal_code = models.CharField(max_length=50,
                                   verbose_name=_("Postal code"))
    postal_locality = models.CharField(max_length=50,
                                       verbose_name=_("Locality"))
```

```
    # Student files

    # TODO: Add upload_to argument

    id_picture = models.ImageField(verbose_name=_("ID picture"))
    # TODO: Add validators to accept pdf files only
    # https://stackoverflow.com/questions/6460848/how-to-limit-file-types-on-
file-uploads-for-modelforms-with-filefields
    id_card = models.FileField(verbose_name=_("ID card"))
    secondary_education_certificate = models.FileField(
        verbose_name=_("Secondary Education Certificate")
    )
    annex_403 = models.FileField(
        null=True,
        blank=True,
        verbose_name=_("Annex 403")
    )
    other_school_inscription_certificate = models.FileField(
        null=True,
        blank=True,
        verbose_name=_("Other schools inscription certificate")
    )
    national_register_extract = models.FileField(
        null=True,
        blank=True,
        verbose_name=_("National Register Extract")
    )
    belgian_studies_history = models.FileField(
        null=True,
        blank=True,
        verbose_name=_("Belgian Studies History")
    )
    archievement_certificates = models.FileField(
        null=True,
        blank=True,
        verbose_name=_("Modules archievement certificates")
    )
    job_organization_certificates = models.FileField(
        null=True,
        blank=True,
        verbose_name=_("Job organizations certificates")
    )
    exemption_report = models.FileField(  # TODO: Add validator: only zip file
        null=True,
        blank=True,
        verbose_name=_("Exemption reports")
    )
```

```
    class Meta:
        """Meta definition for StudentRegistrationReport."""

        verbose_name = _('Student registration report')
        verbose_name_plural = _('Student registration reports')

    @property
    def all_modules_approved(self):
        """Check if all the modules_rrs were approved."""

        modules_approved = True

        for module_rr in self.modules_rrs.all():
            if not module_rr.approved:
                modules_approved = False
                break

        return modules_approved


    @property
    def all_modules_payed(self):
        """Check if all the approved modules_rrs were payed."""

        modules_payed = True

        for module_rr in self.modules_rrs.all():
            if module_rr.approved and not module_rr.payed:
                modules_payed = False
                break

        return modules_payed

    @property
    def total_expenses(self):
        """Compute the total registration expenses of the student."""

        total_expenses = 0
        for module_rr in self.modules_rrs.all():
            total_expenses += module_rr.module.charge_price

        return total_expenses

    @property
    def success_rate(self):
        """Compute the success rate of the student.

        The success rate is the amount of succeeded modules divided by the
        total followed modules.
```

```
        """

        succeeded_modules = 0
        total_modules = 0

        for module_rr in self.modules_rrs.all():
            total_modules += 1

            if module_rr.succeeded:
                succeeded_modules += 1

        success_rate = succeeded_modules / total_modules * 100
        return "{}%".format(success_rate)

    @property
    def spent_ECTS(self):
        """Compute the total amount of spent ECTS.

        The ECTS is the value measurment of a module.
        When a student register to a module, (s)he must pay with his/her ECTS.
        """

        spent_ECTS = 0

        for module_rr in self.modules_rrs.all():
            if module_rr.payed:
                spent_ECTS += module_rr.module.ECTS_value

        return spent_ECTS

    @property
    def won_ECTS(self):
        """Compute the total amount of won ECTS.

        The ECTS is the value measurment of a module.
        When a student succeed a module, (s)he get his/her ECTS back.
        """

        won_ECTS = 0

        for module_rr in self.modules_rrs.all():
            if module_rr.succeeded and module_rr.payed:
                won_ECTS += module_rr.module.ECTS_value

        return won_ECTS

    @property
    def average_score(self):
        """Compute the average final score of the student."""
```

```
        return resource.modules_average_score(self)

    @property
    def has_been_student(self):
        """Check if the user was registered as student for at least one
        module.
        """

        return self.modules_rrs.count() > 0

    @property
    def has_graduated(self):
        """Check if the studend did graduate for a degree at least once."""

        pass

    def __str__(self):
        """Unicode representation of StudentRegistrationReport."""

        return "[{}] {}'s student registration report".format(
            self.pk,
            self.created_by.get_full_name(),
        )

    # TODO: Must be defined
    # def clean(self):
    #     """Clean method for StudentRegistrationReport.

    #     Check if the user is a guest or a student, if the student is not
    #     underaged and if the uploaded files are on the correct format.
    #     """

    #     pass

    def get_absolute_url(self):
        """Return absolute url for StudentRegistrationReport."""

        return reverse("student_rr_detailview", kwargs={"pk": self.pk})


# def user_directory_path(instance, filename):
#     return "{}-{}.{}/{}".format(
#         instance.user.username,
#         instance.user.first_name,
#         instance.user.last_name,
#         filename,
#     )
```

```
# class MyModel(models.Model):
#     upload = models.FileField(upload_to=user_directory_path)
```

serina-project/registration/views/authentication.py

```
from datetime import date

from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from django.contrib.auth.views import (
    LoginView,
    PasswordChangeView,
    PasswordChangeDoneView,
    PasswordResetConfirmView,
    PasswordResetCompleteView,
    PasswordResetDoneView,
    PasswordResetView,
)
from django.shortcuts import redirect, render
from django.urls import reverse, reverse_lazy

from ..forms import (
    CustomAuthenticationForm,
    CustomPasswordChangeForm,
    CustomPasswordResetForm,
    RegistrationForm,
)
from ..utils import (
    groups as groups_utils,
    messages as messages_utils,
    users as users_utils,
)


def register(request):
    """Register function which creates an new User and a new linked
    UserProfile."""

    if messages_utils.user_is_authenticated(request):
        return redirect('home')
    else:
        form = RegistrationForm(request.POST or None)

        if form.is_valid():
```

```
            # Clean data

            first_name = form.cleaned_data["first_name"]
            last_name = form.cleaned_data["last_name"]
            email = form.cleaned_data["email"]

            # Generate username (unique registration number)

            if User.objects.count() == 0:
                latest_user_pk = 0
            else:
                latest_user_pk = User.objects.latest('pk').pk

            date_today = date.today()

            username = users_utils.username_generator(
                latest_user_pk+1,
                date_today,
            )

            # User creation

            user = User.objects.create(
                username=username,
                email=email,
                first_name=first_name.title(),
                last_name=last_name.title(),
            )

            user.set_password(form.cleaned_data["password"])
            groups_utils.promote_to_guest(user)
            user.save()

            # Authentication

            login(request, user)

            return redirect('home')
        else:
            return render(
                request,
                "registration/authentication/register.html",
                locals(),
            )


class CustomLoginView(LoginView):
    """Customized LoginView."""
```

```
    template_name = "registration/authentication/login.html"
    authentication_form = CustomAuthenticationForm
    redirect_authenticated_user = True


@login_required
def customLogout(request):
    """Logout redirection."""

    logout(request)
    messages_utils.user_logged_out(request)
    return redirect('home')


class CustomPasswordChangeView(PasswordChangeView):
    """Customized PasswordChangeView."""

    template_name = "registration/authentication/passwd_change.html"
    form_class = CustomPasswordChangeForm


class CustomPasswordResetView(PasswordResetView):
    """Customized PasswordResetView."""

    # TODO: Use custom template mail + change subject and from_mail
    template_name = "registration/authentication/passwd_reset.html"
    form_class = CustomPasswordResetForm
    # email_template_name = "registration/authentication/password_reset_email.
html"
    # subject_template_name = "registration/authentication/password_reset_subj
ect.txt"
    success_url = reverse_lazy("password_reset_done")
    # from_email = "Serina@SerinaProject.com"  # DEFAULT_FROM_EMAIL = "Serina@
SerinaProject.com"


class CustomPasswordResetDoneView(PasswordResetDoneView):
    """Customized PasswordResetDoneView."""

    template_name = "registration/authentication/passwd_reset_done.html"


class CustomPasswordResetConfirmView(PasswordResetConfirmView):
    """ Curtomized PasswordResetConfirmView."""

    template_name = "registration/authentication/passwd_reset_confirm.html"
    # form_class = CustomSetPasswordForm
    success_url = reverse_lazy("password_reset_complete")
```

```python
class CustomPasswordResetCompleteView(PasswordResetCompleteView):
    """Customized PasswordResetCompleteView."""

    template_name = "registration/authentication/passwd_reset_complete.html"


def post_password_change_logout(request):
    """Log the user out after his/her password has been changed or reset.

    Display a message to the user too.
    """

    messages_utils.password_changed(request)
    logout(request)

    return redirect('home')
```

serina-project/registration/views/base.py

```python
import random

from django.utils import translation
from django.shortcuts import render

from management.models import Degree, Module
from rating.models import StudentRating


def home(request):
    """Homepage render.

    Add 3 random Degree and Module instances to the context."""

    degrees = Degree.objects.order_by("?")[:3]
    modules = Module.objects.order_by("?")[:3]
    ratings = StudentRating.objects.order_by("?")[:4]

    return render(
        request,
        "registration/general/home.html",
        locals(),
    )


def terms_and_conditions(request):
```

```
    """Render the 'Terms and conditions' page."""

    return render(request, "registration/general/terms_and_conditions.html")


def privacy_policy(request):
    """Render the 'Pricacy Policy' page."""

    return render(request, "registration/general/privacy_policy.html")


def cookies_policy(request):
    """Render the 'Cookies policy' page."""

    return render(request, "registration/general/cookies_policy.html")


def home_old(request):  # TODO: Debug view
    """Old omepage render."""

    return render(request, "registration/general/home_old.html", {})
```

serina-project/registration/views/payment.py

```
from decimal import Decimal

from django.contrib import messages
from django.conf import settings
from django.shortcuts import get_object_or_404, redirect, render
from django.views.decorators.csrf import csrf_exempt
from django.urls import reverse

from paypal.standard.forms import PayPalPaymentsForm

from ..forms import PaymentForm


from django.utils.translation import ugettext as _

from ..models import ModuleRegistrationReport
from ..utils import messages as messages_utils


def module_payment(request, pk):
    """CheckoutView of the module registration request in order to proceed to
a
    PayPal payment.
```

```
    Send the payment data to PayPal IPN in order for the user to procced to th
e
    payment.
    """

    module_rr = get_object_or_404(ModuleRegistrationReport, pk=pk)
    request.session['module_rr'] = module_rr.pk

    if module_rr.status != "APPROVED":
        messages_utils.module_not_payable(request)
        redirect(module_rr.get_absolute_url())
    else:
        host = request.get_host()

        paypal_dict = {
            'business': settings.PAYPAL_RECEIVER_EMAIL,
            'amount': module_rr.module.price,
            'item_name': _("Registration for {} to {} (From {} to {})".format(
                module_rr.student_rr.created_by.get_full_name(),
                module_rr.module.title,
                module_rr.date_start,
                module_rr.date_end,
            )),
            'currency_code': 'EUR',
            'notify_url': 'http://{}{}'.format(host, reverse('paypal-ipn')),
            'return_url': 'http://{}{}'.format(host, reverse('payment_done')),
            'cancel_return': 'http://{}{}'.format(host,
                                                  reverse('payment_cancelled')),
        }

        form = PayPalPaymentsForm(initial=paypal_dict)
        return render(
            request,
            'registration/payment/process_payment.html',
            {'module_rr': module_rr, 'form': form},
        )


def get_module_rr_and_clean_session_pk(request):
    """Get the module registration instance based on the session variable
    stored and clean it."""

    module_rr_pk = request.session.get('module_rr')
    module_rr = ModuleRegistrationReport.objects.get(pk=module_rr_pk)

    del request.session['module_rr']

    return module_rr
```

```
@csrf_exempt
def payment_done(request):
    """Payement done view that flags the module registration request as 'PAYED
'
    and redirect to the DetailView."""

    module_rr = get_module_rr_and_clean_session_pk(request)

    if module_rr.final_score:
        module_rr.status = "COMPLETED"
    else:
        module_rr.status = "PAYED"

    module_rr.save()

    messages_utils.module_payment_succeeded(request)

    return redirect(module_rr.get_absolute_url())


@csrf_exempt
def payment_canceled(request):
    """Redirect the user to the module registration DetailView with a message
    indicating that the payment has failed."""

    module_rr = get_module_rr_and_clean_session_pk(request)

    messages_utils.module_payment_failed(request)

    return redirect(module_rr.get_absolute_url())
```

serina-project/registration/views/profile.py

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib.auth.models import User
from django.db.models import Q
from django.views.generic import DetailView, FormView
from django.urls import reverse

from ..forms import StudentProfileUpdateForm, UserProfileUpdateForm
from ..models import(
    DegreeRegistrationReport,
    ModuleRegistrationReport,
```

```
    StudentRegistrationReport,
)
from ..utils.groups import is_student, main_group_i18n
from management.models import Course


class UserProfileDetailView(LoginRequiredMixin, DetailView):
    """User's profile view with all his/her related data.

    If the user is a student, (s)he should have a related StudentRegistration-
    Report. Fetch the StudentRegistrationReport data with the related
    DegreeRegistrationReports and ModuleRegistrationReports.
    """

    model = User
    context_object_name = "user"
    template_name = "registration/userprofile/userprofile_detailview.html"

    def get_context_data(self, **kwargs):
        """Add more data related to user to the context. The user's profile
        page is used as overview the user's activity too.

        Add the user's main group name to the context.
        If the user is a registered student, his/her StudentRegistrationReport
        and its related DegreeRegistrationReports, related
        ModuleRegistrationReports and the Courses the student is subscribed on
        are added to the context too.
        """

        context = super(UserProfileDetailView, self).get_context_data(**kwargs
)
        context["main_group"] = main_group_i18n(self.request.user)

        if is_student(self.request.user):
            context["student_rr"] = StudentRegistrationReport.objects.get(
                created_by=self.request.user
            )
            context["modules_rrs"] = ModuleRegistrationReport.objects.filter(
                student_rr=context["student_rr"]
            )
            context["degrees_rrs"] = DegreeRegistrationReport.objects.filter(
                student_rr=context["student_rr"]
            )
            context["courses"] = []  # TODO: Find a way to express this in que
ryset
            for module_rr in context["modules_rrs"]:
                context["courses"].append(module_rr.course)

        return context
```

```
class UserProfileUpdateView(LoginRequiredMixin, FormView):
    """User's profile UpdateView.

    If the user is a student, (s)he can also change the changeable fields of
    his/het StudentRegistrationReport: 'nationality', 'address',
    'additional_address', 'postal_code' and 'postal_locality'.
    """

    template_name = "registration/userprofile/userprofile_updateview.html"

    def get_form_class(self):
        """Return the StudentProfileUpdateForm if the user is a registered
        student. Otherwise, return the standard UserProfileUpdateForm."""

        if is_student(self.request.user):
            form_class = StudentProfileUpdateForm
        else:
            form_class = UserProfileUpdateForm

        return form_class

    def get_initial(self):
        """Returns the initial data to use for forms on this view."""

        initial = super().get_initial()

        user = User.objects.get(pk=self.request.user.pk)

        initial['first_name'] = user.first_name
        initial['last_name'] = user.last_name
        initial['email'] = user.email

        if is_student(self.request.user):
            student_rr = StudentRegistrationReport.objects.get(created_by=user
)

            initial['nationality'] = student_rr.nationality
            initial['address'] = student_rr.address
            initial['additional_address'] = student_rr.additional_address
            initial['postal_code'] = student_rr.postal_code
            initial['postal_locality'] = student_rr.postal_locality

        return initial

    def form_valid(self, form):
        """If the form is valid, save the updated user profile data. Update th
e
```

```
        StudentRegistrationReport as well if the user is a registered student.
        """

        user = User.objects.get(pk=self.request.user.pk)

        user.first_name = form.cleaned_data["first_name"]
        user.last_name = form.cleaned_data["last_name"]
        user.email = form.cleaned_data["email"]

        user.save()

        if is_student(self.request.user):

            student_rr = StudentRegistrationReport.objects.get(created_by=user
)

            student_rr.nationality = form.cleaned_data["nationality"]
            student_rr.address = form.cleaned_data["address"]
            student_rr.additional_address = form.cleaned_data[
                "additional_address"
            ]
            student_rr.postal_code = form.cleaned_data["postal_code"]
            student_rr.postal_locality = form.cleaned_data["postal_locality"]

            student_rr.save()

        return super().form_valid(form)

    def get_success_url(self):
        """Redirect to the user's profile if the submitted form was valid and
        the profile updated."""

        return reverse("userprofile_detailview",
                       kwargs={"pk": self.request.user.pk})
```

serina-projects/registration/views/registration_actions.py

```
from django.contrib import messages
from django.db.models import Q
from django.shortcuts import get_object_or_404, redirect
from django.utils.translation import ugettext as _

from ..forms import SubmitFinalScoreForm
from ..models import ModuleRegistrationReport
from ..utils import decorators as decorators_utils, messages as messages_utils
```

```
from management.models import Course


@decorators_utils.managers_or_administrators_only
def module_validation(request, pk):
    """Validate a ModuleRegistrationReport submitted based on its 'pk'.
    Register the student to the less populated course available for the chosen
    module. Also save the amount of attempts done by the student for the
    chosen module."""

    module_rr = get_object_or_404(ModuleRegistrationReport, pk=pk)
    if module_rr.approved:
        messages_utils.module_rr_already_approved(request)
    else:
        courses = Course.objects.filter(module=module_rr.module) \
                                .order_by("nb_registrants")

        if courses.count() == 0:
            messages_utils.module_rr_has_no_course(request, module_rr.module)
        else:
            selected_course = courses[0]
            selected_course.nb_registrants += 1
            selected_course.save()

            module_rr.nb_attempt = ModuleRegistrationReport.objects.filter(
                Q(student_rr=module_rr.student_rr)
                & Q(status="COMPLETED")
            ).count()
            module_rr.course = selected_course
            module_rr.status = "APPROVED"
            module_rr.save()

            # TODO: Send mail to student
            messages_utils.module_rr_approved(request)

    return redirect(module_rr.get_absolute_url())


def module_score_submit(request, pk):
    """Check if the module registration request has a valid status ('APPROVED'
,
    'PAYED' or 'EXEMPTED') in order to submit a final score to it."""

    module_rr = get_object_or_404(ModuleRegistrationReport, pk=pk)

    if module_rr.status == "APPROVED" \
       or module_rr.status == "PAYED" \
       or module_rr.status == "EXEMPTED":
```

```
        form = SubmitFinalScoreForm(request.POST or None)

        if form.is_valid():
            score = form.cleaned_data['score']
            module_rr.final_score = score

            if module_rr.status == "PAYED":
                module_rr.status = "COMPLETED"

            module_rr.save()

        if module_rr.status == "APPROVED":
            messages_utils.module_rr_not_payed(request)
        else:
            messages_utils.module_rr_final_score_submitted(request)

    elif module_rr.status == "COMPLETED":
        messages_utils.module_rr_already_completed(request)
    else:
        messages_utils.module_rr_not_approved(request)

    return redirect(module_rr.get_absolute_url())
```

serina-project/registration/views/registration_report.py

```
from django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin
from django.shortcuts import redirect
from django.utils.translation import ugettext as _
from django.views.generic import CreateView, DetailView, ListView

from ..forms import (
    DegreeRegistrationReportCreateFrom,
    ModuleRegistrationReportCreateFrom,
    StudentRegistrationReportCreateFrom,
    SubmitFinalScoreForm,
)
from ..models import (
    DegreeRegistrationReport,
    ModuleRegistrationReport,
    StudentRegistrationReport,
)
from ..utils import (
    groups as groups_utils,
    messages as messages_utils,
    mixins as mixins_utils,
```

```
)


# StudentRegistrationReport

class StudentRegistrationReportListView(
    LoginRequiredMixin,
    mixins_utils.ManagerAdministratorOnlyMixin,
    ListView,
):  # TODO: Debug view
    """ListView for StudentRegistrationReports."""

    model = StudentRegistrationReport
    context_object_name = "student_rrs"
    template_name = "registration/registration_report/student_rr_listview." \
                    "html"


class StudentRegistrationReportDetailView(
    LoginRequiredMixin,
    mixins_utils.SelfStudentManagerAdministratorOnlyMixin,
    DetailView,
):  # TODO: Debug view
    """DetailView for StudentRegistrationReport."""

    model = StudentRegistrationReport
    context_object_name = "student_rr"
    template_name = "registration/registration_report/student_rr_detailview."
\
                    "html"


class StudentRegistrationReportCreateView(
    LoginRequiredMixin,
    UserPassesTestMixin,
    CreateView,
    mixins_utils.AutofillCreatedByRequestUser,
):  # TODO: Debug view
    """CreateView for StudentRegistrationReport.

    Only registered 'Guest'-group members can submit a
    StudentRegistrationReport. Once done, the 'Guest'-user is automatically
    promoted to the 'Student'-group.
    """

    model = StudentRegistrationReport
    form_class = StudentRegistrationReportCreateFrom
    template_name = "registration/registration_report/student_rr_createview."
\
```

```
                    "html"

    def test_func(self):
        """Check if the user is a registered guest.

        Registered guest users are the only members who are allowed to submit
a
        StudentRegistrationReport. The student cannot because they already did
        once. The other groups are not allowed to be student with the same
        account.
        """

        return self.request.user.groups.filter(name="Guest")

    def handle_no_permission(self):
        """Send an error message and redirect the home page."""

        if self.request.user.groups.filter(name="Student"):
            messages_utils.student_rr_already_created(self.request,
                                                      self.request.user)
        else:
            messages_utils.permission_denied(self.request)

        return redirect('home')

    def form_valid(self, form):
        """Promote the user to the 'Student'-group and notificate him/her with
        a message."""

        groups_utils.promote_to_student(self.request.user)
        messages_utils.student_rr_created(self.request)
        return super().form_valid(form)


# ModuleRegistrationReport

class ModuleRegistrationReportListView(
    LoginRequiredMixin,
    mixins_utils.ManagerAdministratorOnlyMixin,
    ListView,
):  # TODO: Debug view
    """ListView for ModuleRegistrationReport."""

    model = ModuleRegistrationReport
    context_object_name = "modules_rrs"
    template_name = "registration/registration_report/module_rr_listview.html"

# TODO: Add ListView for student's modules_rrs only
```

```python
class ModuleRegistrationReportDetailView(
    LoginRequiredMixin,
    mixins_utils.SelfStudentManagerAdministratorOnlyMixin,
    DetailView,
):  # TODO: Debug view
    """DetailView for ModuleRegistrationReport."""

    model = ModuleRegistrationReport
    context_object_name = "module_rr"
    template_name = "registration/registration_report/module_rr_detailview." \
                    "html"

    def get_context_data(self, **kwargs):
        """Add score submission form to context for back-office user only."""

        context = super().get_context_data(**kwargs)

        if groups_utils.is_back_office_user(self.request.user):
            context["form"] = SubmitFinalScoreForm

        return context


class ModuleRegistrationReportCreateView(
    LoginRequiredMixin,
    # mixins_utils.StudentOnlyMixin,  # TODO: Disabled for debug purposes
    CreateView,
    mixins_utils.AutofillCreatedByRequestUser,
):  # TODO: Debug view
    """CreateView for ModuleRegistrationReport."""

    model = ModuleRegistrationReport
    form_class = ModuleRegistrationReportCreateFrom
    template_name = "registration/registration_report/module_rr_createview." \
                    "html"

    def get_initial(self):
        """Returns the initial data to use for forms on this view."""

        initial = super().get_initial()
        initial['student_rr'] = self.request.user.student_rr
        return initial


# DegreeRegistrationReport

class DegreeRegistrationReportListView(
    LoginRequiredMixin,
    mixins_utils.ManagerAdministratorOnlyMixin,
```

```
    ListView,
):  # TODO: Debug view
    """ListView for DegreeRegistrationReportListView."""

    model = DegreeRegistrationReport
    context_object_name = "degrees_rrs"
    template_name = "registration/registration_report/degree_rr_listview.html"


class DegreeRegistrationReportDetailView(
    LoginRequiredMixin,
    mixins_utils.SelfStudentManagerAdministratorOnlyMixin,
    DetailView,
):  # TODO: Debug view
    """DetailView for DegreeRegistrationReportListView."""

    model = DegreeRegistrationReport
    context_object_name = "degree_rr"
    template_name = "registration/registration_report/degree_rr_detailview" \
                    ".html"


class DegreeRegistrationReportCreateView(
    LoginRequiredMixin,
    # mixins_utils.StudentOnlyMixin,  # TODO: Disabled for debug purposes
    CreateView,
    mixins_utils.AutofillCreatedByRequestUser,
):  # TODO: Debug view
    """CreateView for DegreeRegistrationReport."""

    model = DegreeRegistrationReport
    form_class = DegreeRegistrationReportCreateFrom
    template_name = "registration/registration_report/degree_rr_createview." \
                    "html"

    def get_initial(self):
        """Returns the initial data to use for forms on this view."""

        initial = super().get_initial()
        initial['student_rr'] = self.request.user.student_rr
        return initial
```

serina-project/registration/forms/authentication.py

```
from django import forms
```

```
from django.conf import settings
from django.contrib.auth.forms import (
    AuthenticationForm,
    PasswordChangeForm,
    PasswordResetForm,
    UserCreationForm
)
from django.contrib.auth.models import User
from django.utils.translation import gettext as _


class RegistrationForm(forms.Form):
    """Customized UserCreationForm."""

    first_name = forms.CharField(label=_("First name"), required=True)
    last_name = forms.CharField(label=_("Last name"), required=True)
    email = forms.EmailField(
        label=_("Email"),
        required=True,
        error_messages={
            'invalid': _("Mail address invalid. Check the spelling or try "
                         "another one.")
        }
    )
    password = forms.CharField(
        label=_("Password"),
        required=True,
        widget=forms.PasswordInput
    )
    confirm_password = forms.CharField(
        label=_("Password confirmation"),
        required=True,
        widget=forms.PasswordInput
    )

    def clean(self):
        """Check if the email is not already used by another user and check if
        the password and password confirmation match together.

        Raise an error with a displayed error message if one of these
        conditions failed.
        """

        cleaned_data = super(RegistrationForm, self).clean()

        email = cleaned_data.get("email")
        if User.objects.filter(email=email).exists():
            raise forms.ValidationError(_(
                "The entered mail address is already in use. Please use "
```

```
                "another one or contact our support team ({})."
                .format(settings.CONTACT_MAILS["support"])
                # TODO: Add clickable mailto link
            ))

        password = cleaned_data.get("password")
        confirm_password = cleaned_data.get("confirm_password")
        if password != confirm_password:
            raise forms.ValidationError(_(
                "Your password and your password confirmation does not match.
"
                "Please try again."
            ))


class CustomAuthenticationForm(AuthenticationForm):
    """Custom AuthenticationForm supporting i18n."""

    username = forms.CharField(
        label=_('Username'),
        widget=forms.TextInput(attrs={'autofocus': True})
    )
    password = forms.CharField(
        label=_('Password'),
        widget=forms.PasswordInput()
    )


class CustomPasswordChangeForm(PasswordChangeForm):
    """Custom Authentication Form supporting i18n"""

    old_password = forms.CharField(
        label=_('Old password'),
        widget=forms.PasswordInput(attrs={'autofocus': True})
    )

    new_password1 = forms.CharField(
        label=_('New password'),
        widget=forms.PasswordInput()
    )

    new_password2 = forms.CharField(
        label=_('New password confirmation'),
        widget=forms.PasswordInput()
    )


class CustomPasswordResetForm(PasswordResetForm):
    """Custom PasswordResetForm supporting i18n"""
```

```
    email = forms.EmailField(
        label=_('Email address')
    )
```

serina-project/registration/forms/payment.py

```
from django import forms

from ..models import ModuleRegistrationReport

class PaymentForm(forms.Form):
    """PaymentForm definition."""

    module_rr = forms.ModelChoiceField(queryset=ModuleRegistrationReport.objec
ts.all(), empty_label=None)
```

serina-project/registration/forms/profile.py

```
from django import forms


class UserProfileUpdateForm(forms.Form):
    """Form to update the user's profile."""

    first_name = forms.CharField(max_length=50)
    last_name = forms.CharField(max_length=50)
    email = forms.EmailField(max_length=50)


class StudentProfileUpdateForm(UserProfileUpdateForm):
    """Form to update the student's profile and StudentRegistrationReport's
    changeable data."""

    nationality = forms.CharField(max_length=50)
    address = forms.CharField(max_length=255)
    additional_address = forms.CharField(max_length=255)
    postal_code = forms.CharField(max_length=50)
    postal_locality = forms.CharField(max_length=50)
```

serina-project/registration/forms/registration_actions.py

```python
from django import forms
from django.utils.translation import gettext as _


class SubmitFinalScoreForm(forms.Form):
    """Form to submit a final score to a ModuleRegistrationReport instance."""

    score = forms.DecimalField(
        min_value=0,
        max_value=100,
        decimal_places=2,
        label=_("Student's module final score"),
    )
```

serina-project/registration/forms/registration_report.py

```python
from django import forms

from ..models import (
    DegreeRegistrationReport,
    ModuleRegistrationReport,
    StudentRegistrationReport,
)
from ..utils.mixins import (
    HideCreatedByFieldFormMixin,
    VerboseDegreeModuleChoiceField,
)
from management.models import Degree, Module

# TODO: Comment correctly

class StudentRegistrationReportCreateFrom(HideCreatedByFieldFormMixin):
    """ModelForm for Module."""

    class Meta(HideCreatedByFieldFormMixin.Meta):
        """Meta definition for ModuleLevelForm."""

        model = StudentRegistrationReport
        fields = "__all__"
```

```
class ModuleRegistrationReportCreateFrom(forms.ModelForm):
    """ModelForm for Module."""

    module = VerboseDegreeModuleChoiceField(queryset=Module.objects.all(),
                                            empty_label=None)

    class Meta:
        """Meta definition for ModuleLevelForm."""

        model = ModuleRegistrationReport
        fields = ("student_rr", "module", "notes")
        widgets = {
            'student_rr': forms.HiddenInput(),
        }


class DegreeRegistrationReportCreateFrom(forms.ModelForm):
    """ModelForm for Module."""

    degree = VerboseDegreeModuleChoiceField(queryset=Degree.objects.all(),
                                            empty_label=None)

    class Meta:
        """Meta definition for ModuleLevelForm."""

        model = DegreeRegistrationReport
        fields = ("student_rr", "degree", "notes")
        widgets = {
            'student_rr': forms.HiddenInput(),
        }
```

serina-project/registration/utils/decorators.py

```
from django.contrib.auth.decorators import user_passes_test
from django.db.models import Q
from django.shortcuts import redirect

from . import groups as groups_utils
from . import messages as messages_utils


def managers_or_administrators_only(function):
    """Restrict a function acces to managers or administrators only."""

    def wrapper(request, *args, **kwargs):
```

```
        """managers_or_administrators_only main wrapper."""

        if groups_utils.is_manager_or_administrator(request.user):
            return function(request, *args, **kwargs)
        else:
            messages_utils.permission_denied(request)
            return redirect("home")

    return wrapper
```

serina-project/registration/utils/groups.py

```
from django.contrib.auth.models import Group
from django.db.models import Q
from django.utils.translation import ugettext as _


# Read utilities

def is_student(user):
    """Check if the user is a registered student."""

    return user.groups.filter(name="Student").exists() \
        and user.student_rr is not None


def is_back_office_user(user):
    """Check if the user is allowed to access the back office.

    The user must be member of one of the granted groups which are
    'Teacher', 'Manager', 'Administrator'.
    """

    return user.groups.filter(Q(name="Teacher")
                              | Q(name="Manager")
                              | Q(name="Administrator")).exists()


def is_manager_or_administrator(user):
    """Check if the user is member of the 'Manager'-group or the
    'Administrator'-group."""

    return user.groups.filter(Q(name="Manager")
                              | Q(name="Administrator")).exists()
```

```
def main_group_i18n(user):
    """Main group of the user in the case one has multiple groups.

    The group name is given with in its i18n version.
    """

    if user.groups.filter(name="Administrator").exists():
        main_group = _("Administrator")
    elif user.groups.filter(name="Manager").exists():
        main_group = _("Manager")
    elif user.groups.filter(name="Teacher").exists():
        main_group = _("Teacher")
    elif user.groups.filter(name="Student").exists():
        main_group = _("Student")
    else:
        main_group = _("Guest")

    return main_group


# Update utilities: Group alteration

def remove_from_all_groups(user):
    """Remove a user from all its groups.
    Remove the user access as staff member and superuser."""

    groups = Group.objects.all()

    for group in groups:
        group.user_set.remove(user)

    user.is_staff = False
    user.is_superuser = False


def add_to_group(user, group_name):
    """Add a user to a given group. Creates it if the group does not exist yet
.
    """

    group, isCreated = Group.objects.get_or_create(name=group_name)
    group.user_set.add(user)

    return isCreated


def change_group(user, group_name):
    """Remove a user from all its group and add him/her to the given one."""
```

```
    remove_from_all_groups(user)
    add_to_group(user, group_name)


# Update utilities: Group promotion

def promote_to_guest(user):
    """Promote a registered user to the 'Guest'-group."""

    change_group(user, 'Guest')


def promote_to_student(user):
    """Promote a registered user to the 'Student'-group."""

    change_group(user, 'Student')


def promote_to_teacher(user):
    """Promote a registered user to the 'Teacher'-group."""

    change_group(user, 'Teacher')
    user.is_staff = True


def promote_to_manager(user):
    """Promote a registered user to the 'Manager'-group and make is staff
    member."""

    change_group(user, 'Manager')
    user.is_staff = True


def promote_to_administrator(user):
    """Promote a registered user to the 'Administrator'-group and make it
    superuser."""

    change_group(user, 'Administrator')
    user.is_staff = True
    user.is_superuser = True
```

serina-project/registration/utils/messages.py

```
from django.conf import settings
from django.contrib import messages
from django.utils.translation import ugettext as _
```

```
# Authentication

def user_logged_out(request):
    """Inform the user that (s)he has successfully been logged out."""

    messages.success(request, _("You have been logged out successfully."))


def password_changed(request):
    """Inform the user his/her password has correctly been changed."""

    messages.success(
        request,
        _("Your password has been changed. You must log yourself in again with
"
          " the new password.")
    )


def user_is_authenticated(request):  # TODO: Must be decorator
    """Check if a user is authenticated and send message if this is true."""

    is_authenticated = False
    if request.user.is_authenticated:
        is_authenticated = True
        messages.warning(
            request,
            _("You are already signed in. "
              "Please sign out to use a different account.")
        )
    return is_authenticated


# Access control

def permission_denied(request):
    """Warns the user (s)he tried to perform an action without having the
    right permissions."""

    messages.error(
        request,
        _("You are not allowed to perform this action. Please contact the "
          "support team ({}) for more information."
          .format(settings.CONTACT_MAILS["support"]))
    )
```

```
# StudentRegistrationReport

def student_rr_created(request):
    """Inform the user that his/her StudentRegistrationReport has correctly
    been saved and that the user was promoted to the 'Student'-group."""

    messages.success(
        request,
        _("Your registration report has successfully been submitted. You now "
          "have the 'Student' status and can subscrib to any degree or any "
          "module wanted.")
    )


def student_rr_already_created(request, student):
    """Warns the user that a StudentRegistrationReport object as already been
    linked to the selected student."""

    messages.error(
        request,
        _("The student registration file has already been created for your "
          "account: {} ({}). You can still change some information from your "
          "student's profile.".format(
              student.get_full_name(),
              student.username,
          ))
    )


# ModuleResgitrationReport Validation

def module_rr_has_no_course(request, module):
    """Warns the user that the ModuleRegistrationReport object cannot be
    validated because there aren't any course related to the module.

    When a module doesn't have any course related to it, the students cannot b
e
    assign to the course and so the module itself. An error message is prompt.
    """

    messages.error(
        request,
        _("There is no course available for the requested module: {} ({}). In
"
          "order to accept any new registration request, a new course must be
"
          "created for this module.".format(module.title, module.reference))
    )
```

```
def module_rr_approved(request):
    """Inform the user that the ModuleRegistrationReport object has
    successfully been approved."""

    messages.success(
        request,
        _("The module's registration has been approved. A notification mail "
          "has been sent to the student.")
    )


def module_rr_already_approved(request):
    """Warns the user that the ModuleRegistrationReport object (s)he wants to
    approve has already been approved."""

    messages.warning(
        request,
        _("This module registration request has already been approved.")
    )


# ModuleResgitrationReport Final Score Submission

def module_rr_final_score_submitted(request):
    """Inform the user that the final score has been successfully applied to
    the module registration request."""

    messages.success(
        request,
        _("The module score has been added and has now been completed.")
    )


def module_rr_already_completed(request):
    """Warns the user that the module on which (s)he tries to submit a score
    was already completed."""

    messages.error(
        request,
        _("This module registration request is already completed. The final "
          "score cannot be changed anymore without an adminitrator ({})"
          .format(settings.CONTACT_MAILS["administrators"]))
    )


def module_rr_not_approved(request):
    """Warns the user that the module on which (s)he tries to submit a score
    has not been approved yet."""
```

```
    messages.error(
        request,
        _("This module registration request has not been approved or yet and "
          "can therefore not been rated.")
    )


def module_rr_not_payed(request):
    """Warns the user that the module on which (s)he tries to submit a score
    has not been payed yet."""

    messages.warning(
        request,
        _("This module registration request has not been payed by the sudent "
          "yet. The final score was saved anyway but the module cannot be "
          "coonsidered as completed as long as the payment wasn't done.")
    )


# ModuleRegistrationReport payment

def module_payment_succeeded(request):
    """Inform the user the payment has been successfully completed."""

    messages.success(
        request,
        _("The module has been successfully payed.")
    )


def module_payment_failed(request):
    """Warn the user the payment has failed."""

    messages.error(
        request,
        _("The module payment has unexpectedly been aborted.")
    )


def module_not_payable(request):
    """Warns the user the module request cannot be payed because it has not an
    'APPROVED' status."""

    messages.error(
        request,
        _("This module registration request cannot be payed. It has either "
          "not been approved, is already payed, completed or exempted.")
    )
```

serina-project/registration/utils/mixins.py

```
from django import forms
from django.conf import settings
from django.contrib import messages
from django.contrib.auth.mixins import UserPassesTestMixin
from django.http import Http404
from django.views.generic import FormView
from django.shortcuts import redirect
from django.utils.translation import gettext as _

from .. import models
from . import groups as groups_utils
from . import messages as messages_utils
from rating import models as rating_models


# Access restriction mixins


class AccessRestrictionMixin(UserPassesTestMixin):
    """Display an error message to the user which access has been denied and
    redirect him/her to the homepage."""

    def handle_no_permission(self):
        """Send an error message and redirect the home page."""

        messages_utils.permission_denied(self.request)
        return redirect('home')


class StudentOnlyMixin(AccessRestrictionMixin):
    """Restrict view access to Students users."""

    def test_func(self):
        """Check if the user is a registered student user."""

        return groups_utils.is_student(self.request.user)


class BackOfficeUsersOnlyMixin(AccessRestrictionMixin):
    """Restrict view access to Back-Office users."""

    def test_func(self):
        """Check if the user is a Back-Office user."""
```

```
        return groups_utils.is_back_office_user(self.request.user)


class ManagerAdministratorOnlyMixin(AccessRestrictionMixin):
    """Restrict view access to 'Manager'-group members and
    'Administrator'-group members."""

    def test_func(self):
        """Check if the user is a Manager or an Administrator."""

        return groups_utils.is_manager_or_administrator(self.request.user)


class StudentManagerAdministratorOnlyMixin(ManagerAdministratorOnlyMixin):
    """Restrict view access to 'Student'-group members, the 'Manager'-group
    members and 'Administrator'-group members."""

    def test_func(self):
        """Check if the user is a Student, Manager or an Administrator."""

        return super(StudentManagerAdministratorOnlyMixin, self).test_func() \
            or groups_utils.is_student(self.request.user)


class SelfStudentManagerAdministratorOnlyMixin(
    StudentManagerAdministratorOnlyMixin,
):
    """Restrict view access to 'Manager'-group members and
    'Administrator'-group members and the student who created the object.

    This mixins works for students, modules and degrees regitrations reports.
    """

    def test_func(self):
        """Check if the user is a Student, Manager or an Administrator. If the
        user is a student, check if (s)he is the one who created the object.

        The object being variable, the queryset is adapted according to the
        object-type received.
        """

        super_test_valid = super(SelfStudentManagerAdministratorOnlyMixin,
                                 self).test_func()
        self_test_valid = False

        # Check if user is a student

        if groups_utils.is_student(self.request.user):
```

```
            # StudentRegistrationReport

            if type(self.get_object()) is models.student_rr \
                                                .StudentRegistrationReport:
                self_test_valid = \
                    self.request.user.student_rr.pk == self.get_object().pk

            # ModuleRegistrationReport

            elif type(self.get_object()) is models.module_rr \
                                                .ModuleRegistrationReport:
                self_test_valid = self.request.user.student_rr.modules_rrs \
                                    .filter(pk=self.get_object().pk) \
                                    .exists()

            # DegreeRegistrationReport

            elif type(self.get_object()) is models.degree_rr \
                                                .DegreeRegistrationReport:
                self_test_valid = self.request.user.student_rr.degrees_rrs \
                                    .filter(pk=self.get_object().pk) \
                                    .exists()

            # StudentRating

            elif type(self.get_object()) is rating_models.StudentRating:
                self_test_valid = self.request.user.ratings.filter(
                    pk=self.get_object().pk).exists()

        # Otherwise user is manager or administrator

        else:
            self_test_valid = True

        return super_test_valid and self_test_valid


# Autofill 'created_by' formfield mixins (form and view)

class HideCreatedByFieldFormMixin(forms.ModelForm):
    """Mixin for ModelForms thats hide the 'created_by' field.

    The 'created_by'-field is still created and can be populated by the view
    without an input from the user.
    """

    class Meta:
        """Meta definition for HideCreatedByFieldFormMixin."""
```

```
        widgets = {
            'created_by': forms.HiddenInput(),
        }


class AutofillCreatedByRequestUser(FormView):
    """Autofill the 'created_by'-formfield by the request.user."""

    def get_initial(self):
        """Returns the initial data to use for forms on this view."""

        initial = super().get_initial()
        initial['created_by'] = self.request.user
        return initial


# FormFields mixins

class VerboseDegreeModuleChoiceField(forms.ModelChoiceField):
    """Display the reference and the title of each degree or module in a
    verbose format as a ChoiceField."""

    def label_from_instance(self, degree_or_module):
        """Return the verbose value."""

        return "{} ({})".format(degree_or_module.title,
                                degree_or_module.reference)
```

serina-project/registration/utils/registration.py

```
from django.conf import settings
from django.contrib.auth.models import User
from django.db.models import Q

from ..models import ModuleRegistrationReport
from . import groups as groups_utils


def module_already_validated_by_user(user, module):
    """Check if a module has already been validated by a user."""

    return groups_utils.is_student(user) \
        and ModuleRegistrationReport.objects.filter(
        Q(student_rr__created_by=user),
        Q(module=module),
```

```
        (Q(status="EXEMPTED") | Q(status="COMPLETED")),
        Q(final_score__gte=settings.SUCCESS_SCORE_THRESHOLD)
    ).exists()


def all_prerequisites_validated_by_user(user, module):
    """Check if all the prerequisites modules has already been validated by th
e
    user."""

    all_prerequisites_validated = True

    for prerequisite in module.prerequisites.all():
        if not module_already_validated_by_user(user, prerequisite):
            all_prerequisites_validated = False
            break

    return groups_utils.is_student(user) and all_prerequisites_validated
```

serina-project/registration/utils/users.py

```
from django.contrib.auth.models import User


def username_already_exist(user):
    """Check if a username is already taken."""

    return User.objects.exclude(pk=user.pk).filter(username=user.username) \
        .exists()


def username_generator(pk, date):
    """Generate a username registration number.

    The registration number has the (YYMMDDxxx) format with YY current year,
    MM the current month, DD the current day and xxx the pk given as argument
    filled with leading zeros.
    """

    return date.strftime("%y%m%d") + str(pk).zfill(3)
```

serina-project/registration/templatetags/registration_extra.py

```python
from django import template
from django.contrib.auth.models import Group

from ..utils import groups as groups_utils


register = template.Library()


@register.filter
def is_student(user):
    """Template tags that checks if a user is a student or not."""

    return groups_utils.is_student(user)


@register.filter
def is_manager_or_administrator(user):
    """Template tags that checks if a user is is manager or administrator."""

    return groups_utils.is_manager_or_administrator(user)


# @register.filter(takes_context=True)
# def user_is_student(context):
#     """Template tags that checks if a user is a student or not."""

#     request = context.get("request")
#     return groups_utils.is_student(request.user)
```

serina-project/registration/urls.py

```python
from django.conf.urls import url

from . import views


urlpatterns = [


    # General pages

    url(
        r"^$",
        views.home,
```

```
        name="home",
    ),
    url(  # TODO: Debug root
        r"^home$",
        views.home_old,
        name="home_old",
    ),
    url(
        r"^who_are_we/$",
        views.home,
        name="who_are_we"
    ),
    url(
        r"^contact/$",
        views.home,
        name="contact"
    ),
    url(
        r"^terms_and_conditions/$",
        views.terms_and_conditions,
        name="terms_and_conditions"
    ),
    url(
        r"^privacy_policy/$",
        views.privacy_policy,
        name="privacy_policy"
    ),
    url(
        r"^cookies_policy/$",
        views.cookies_policy,
        name="cookies_policy"
    ),


    # Authentication

    url(
        r"^login/$",
        views.CustomLoginView.as_view(),
        name="login",
    ),
    url(
        r"^logout/$",
        views.customLogout,
        name="logout",
    ),
    url(
        r"^register/$",
        views.register,
```

```
        name="register",
    ),

    # Password change

    url(
        r"^password/change/$",
        views.CustomPasswordChangeView.as_view(),
        name="password_change",
    ),
    url(
        r"^password/change/done/$",
        views.post_password_change_logout,
        name="password_change_done",
    ),

    # Password reset

    url(
        r"^password/reset/$",
        views.CustomPasswordResetView.as_view(),
        name="password_reset",
    ),
    url(
        r"^password/reset/done/$",
        views.CustomPasswordResetDoneView.as_view(),
        name="password_reset_done",
    ),
    url(
        r"^password/reset/confirm/(?P<uidb64>[0-9A-Za-z_\-]+)/" \
        r"(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1,20})/$",
        views.CustomPasswordResetConfirmView.as_view(),
        name="password_reset_confirm",
    ),
    url(
        r"^password/reset/complete/$",
        views.post_password_change_logout,
        name="password_reset_complete",
    ),


    # Profile

    url(
        r"^profile/r/(?P<pk>[0-9]+)/$",
        views.UserProfileDetailView.as_view(),
        name="userprofile_detailview",
    ),
    url(
```

```
        r"^profile/u/(?P<pk>[0-9]+)/$",
        views.UserProfileUpdateView.as_view(),
        name="userprofile_updateview",
    ),


    # RegistrationReports

    # StudentRegistrationReport

    url(
        r"^report/student/l/",
        views.StudentRegistrationReportListView.as_view(),
        name="student_rr_listview",
    ),
    url(
        r"^report/student/r/(?P<pk>[0-9]+)/$",
        views.StudentRegistrationReportDetailView.as_view(),
        name="student_rr_detailview",
    ),
    url(
        r"^report/student/c/$",
        views.StudentRegistrationReportCreateView.as_view(),
        name="student_rr_createview",
    ),

    # ModuleRegistrationReport

    url(
        r"^report/module/l/$",
        views.ModuleRegistrationReportListView.as_view(),
        name="module_rr_listview",
    ),
    url(
        r"^report/module/r/(?P<pk>[0-9]+)/$",
        views.ModuleRegistrationReportDetailView.as_view(),
        name="module_rr_detailview",
    ),
    url(
        r"^report/module/c/$",
        views.ModuleRegistrationReportCreateView.as_view(),
        name="module_rr_createview",
    ),

    # DegreeRegistrationReport

    url(
        r"^report/degree/l/$",
        views.DegreeRegistrationReportListView.as_view(),
```

```
        name="degree_rr_listview"
    ),
    url(
        r"^report/degree/r/(?P<pk>[0-9]+)/$",
        views.DegreeRegistrationReportDetailView.as_view(),
        name="degree_rr_detailview"
    ),
    url(
        r"^report/degree/c/$",
        views.DegreeRegistrationReportCreateView.as_view(),
        name="degree_rr_createview"
    ),


    # Back-Office functions

    url(
        r'^back_office/module_validation/(?P<pk>[0-9]+)/$',
        views.module_validation,
        name='backoffice_module_validation'
    ),

    url(
        r'^back_office/module_score_submit/(?P<pk>[0-9]+)/$',
        views.module_score_submit,
        name='backoffice_module_score_submit'
    ),


    # Payment

    url(
        r"^payment/checkout/(?P<pk>\d+)/$",
        views.module_payment,
        name="module_payment"
    ),
    url(r'payment-done/', views.payment_done, name='payment_done'),
    url(
        r'payment-cancelled/',
        views.payment_canceled,
        name='payment_cancelled'
    ),
]
```

5.3.2. Module de gestion des ressources

serina-project/management/urls.py

```python
from django.conf.urls import url

from . import views


urlpatterns = [


    # Classroom

    url(
        r"^classroom/l/$",
        views.ClassroomListView.as_view(),
        name="classroom_listview",
    ),
    url(
        r"^classroom/r/(?P<pk>[0-9]+)/$",
        views.ClassroomDetailView.as_view(),
        name="classroom_detailview",
    ),
    url(
        r"^classroom/c/$",
        views.ClassroomCreateView.as_view(),
        name="classroom_createview",
    ),
    url(
        r"^classroom/u/(?P<pk>[0-9]+)/$",
        views.ClassroomUpdateView.as_view(),
        name="classroom_updateview",
    ),
    url(
        r"^classroom/d/(?P<pk>[0-9]+)/$",
        views.ClassroomDeleteView.as_view(),
        name="classroom_deleteview",
    ),


    # Course

    url(
        r"^course/l/$",
        views.CourseListView.as_view(),
        name="course_listview",
```

```
    ),
    url(
        r"^course/r/(?P<pk>[0-9]+)/$",
        views.CourseDetailView.as_view(),
        name="course_detailview",
    ),
    url(
        r"^course/c/$",
        views.CourseCreateView.as_view(),
        name="course_createview",
    ),
    url(
        r"^course/u/(?P<pk>[0-9]+)/$",
        views.CourseUpdateView.as_view(),
        name="course_updateview",
    ),
    url(
        r"^course/d/(?P<pk>[0-9]+)/$",
        views.CourseDeleteView.as_view(),
        name="course_deleteview",
    ),


    # Degree

    url(
        r"^degree/l/$",
        views.DegreeListView.as_view(),
        name="degree_listview",
    ),
    url(
        r"^degree/r/(?P<pk>[0-9]+)/$",
        views.DegreeDetailView.as_view(),
        name="degree_detailview",
    ),
    url(
        r"^degree/c/$",
        views.DegreeCreateView.as_view(),
        name="degree_createview",
    ),
    url(
        r"^degree/u/(?P<pk>[0-9]+)/$",
        views.DegreeUpdateView.as_view(),
        name="degree_updateview",
    ),
    url(
        r"^degree/d/(?P<pk>[0-9]+)/$",
        views.DegreeDeleteView.as_view(),
        name="degree_deleteview",
```

```
    ),

    # DegreeCategory

    url(
        r"^degree/category/l/$",
        views.DegreeCategoryListView.as_view(),
        name="degreecategory_listview",
    ),
    url(
        r"^degree/category/r/(?P<pk>[0-9]+)/$",
        views.DegreeCategoryDetailView.as_view(),
        name="degreecategory_detailview",
    ),
    url(
        r"^degree/category/c/$",
        views.DegreeCategoryCreateView.as_view(),
        name="degreecategory_createview",
    ),
    url(
        r"^degree/category/u/(?P<pk>[0-9]+)/$",
        views.DegreeCategoryUpdateView.as_view(),
        name="degreecategory_updateview",
    ),
    url(
        r"^degree/category/d/(?P<pk>[0-9]+)/$",
        views.DegreeCategoryDeleteView.as_view(),
        name="degreecategory_deleteview",
    ),


    # Module

    url(
        r"^module/l/$",
        views.ModuleListView.as_view(),
        name="module_listview",
    ),
    url(
        r"^module/r/(?P<pk>[0-9]+)/$",
        views.ModuleDetailView.as_view(),
        name="module_detailview",
    ),
    url(
        r"^module/c/$",
        views.ModuleCreateView.as_view(),
        name="module_createview",
    ),
    url(
```

```
        r"^module/u/(?P<pk>[0-9]+)/$",
        views.ModuleUpdateView.as_view(),
        name="module_updateview",
    ),
    url(
        r"^module/d/(?P<pk>[0-9]+)/$",
        views.ModuleDeleteView.as_view(),
        name="module_deleteview",
    ),


    # ModuleCategory

    url(
        r"^module/level/l/$",
        views.ModuleLevelListView.as_view(),
        name="modulelevel_listview",
    ),
    url(
        r"^module/level/r/(?P<pk>[0-9]+)/$",
        views.ModuleLevelDetailView.as_view(),
        name="modulelevel_detailview",
    ),
    url(
        r"^module/level/c/$",
        views.ModuleLevelCreateView.as_view(),
        name="modulelevel_createview",
    ),
    url(
        r"^module/level/u/(?P<pk>[0-9]+)/$",
        views.ModuleLevelUpdateView.as_view(),
        name="modulelevel_updateview",
    ),
    url(
        r"^module/level/d/(?P<pk>[0-9]+)/$",
        views.ModuleLevelDeleteView.as_view(),
        name="modulelevel_deleteview",
    ),
]
```

serina-project/management/models/course.py

```
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError
```

```python
from django.db import models
from django.shortcuts import reverse
from django.utils.translation import ugettext as _

from .module import Module
from .resource import BackOfficeResource
from .room import Classroom


class Course(BackOfficeResource):
    """Model definition for Course."""

    reference = models.CharField(max_length=11, unique=True, blank=True,
                                 verbose_name=_('Reference'))
    module = models.ForeignKey(
        Module,
        on_delete=models.CASCADE,
        related_name="courses",
        verbose_name=_("Modules")
    )
    teacher = models.ForeignKey(
        User,
        null=True,
        on_delete=models.SET_NULL,
        related_name="teaches",
        verbose_name=_('Teached by')
    )
    room = models.ForeignKey(
        Classroom,
        null=True,
        on_delete=models.SET_NULL,
        related_name="courses",
        verbose_name=_("Classroom")
    )
    date_start = models.DateField(null=True, blank=True,
                                  verbose_name=_("Start date"))
    date_end = models.DateField(null=True, blank=True,
                                verbose_name=_("End date"))
    nb_registrants = models.PositiveIntegerField(
        default=0,
        verbose_name=_("Amount of registrants")
    )
    picture = models.ImageField(
        upload_to='management/courses/',
        default='management/undraw_Books_l33t.png',
        null=True,
        blank=True,
        max_length=225,
        verbose_name=_("Picture"),
```

```
    )

    class Meta:
        """Meta definition for Course."""

        verbose_name = _('Course')
        verbose_name_plural = _('Courses')
        ordering = ('date_start', 'reference')

    @property
    def recommended_seats_available(self):
        """Compute the amount of seats left of this course until the
        classroom's recommended capacity is reached."""

        return self.room.recommended_capacity - self.nb_registrants

    @property
    def max_seats_available(self):
        """Compute the amount of seats left of this course until the
        classroom's maximal capacity is reached."""

        return self.room.max_capacity - self.nb_registrants

    @property
    def over_attendance(self):
        """Check if the recommended_capacity has been reached by the expected
        attendance.

        Return None if no room was assigned to the course yet.
        """

        if self.room:
            return self.nb_registrants > self.room.recommended_capacity
        else:
            return None

    def __str__(self):
        """Unicode representation of Course.

        Indictate the teacher's full name if any and the classroom's name if
        any.
        """

        str_result = _("({}) {} course".format(self.reference, self.module.tit
le))

        if self.teacher or self.room:
            str_result += _(" given")
```

```
            if self.teacher:
                str_result += _(" by {}".format(self.teacher.get_full_name()))

            if self.room:
                str_result += _(" at {}".format(self.room.name))

        return str_result


    def clean(self):
        """Clean method for Course.

        Check if the creation date is not set after the last update date, if
        the creator of the instance is a user from a promoted group
        ('Professor', 'Manager' or 'Administrator'), if the teacher is
        eligible to teach this module, if the start date is set before the end
        date and if the amount of registrants is not higher than the maximum
        capacity of the assigned classroom.

        Check if the creator is a promoted-group's user.
        """

        # Creator must be a promoted user
        super().clean()

        # Teacher must be eligilble for module
        # NOTE: Didn't understand why I must use username and not user
        if self.teacher and not self.module.eligible_teachers.filter(
            username=self.teacher.username
        ).exists():
            raise ValidationError(
                _("{} is not eligible to teach this course."
                  .format(self.teacher.get_full_name()))
            )

        # date_start cannot be set after date_end
        if self.date_start and self.date_end \
           and self.date_start >= self.date_end:
            raise ValidationError(
                _("Start date ({}) must be set before end date ({}).".format(
                    self.date_start,
                    self.date_end
                ))
            )

        # nb_registrants must not exceed room.max_capacity
        if self.room and self.nb_registrants > self.room.max_capacity:
            raise ValidationError(
                _("Amount of registrants ({}) cannot be higher than the "
```

```
                "maximum capacity of the assigned classroom ({}).".format(
                    self.nb_registrants,
                    self.room.max_capacity
                ))
            )

    def save(self, *args, **kwargs):
        """Save method for Course.

        Add a reference based on the course's pk and it's module name.
        """

        module_reference = self.module.reference
        self.reference = module_reference + "-"

        if not self.pk:
            super().save(*args, **kwargs)

        self.reference += str(self.pk).zfill(3)
        super().save(*args, **kwargs)

    def get_absolute_url(self):
        """Return absolute url for Course."""

        return reverse('course_detailview', kwargs={'pk': self.pk})
```

serina-project/management/models/degree.py

```
from django.contrib.auth.models import User
from django.db import models
from django.shortcuts import reverse
from django.utils.translation import ugettext as _

from .module import Module
from .resource import BackOfficeResource


class DegreeCategory(BackOfficeResource):
    """Model definition for DegreeCategory."""

    name = models.CharField(max_length=50, verbose_name=_("Name"))

    class Meta:
        """Meta definition for DegreeCategory."""
```

```
        verbose_name = _('Degree category')
        verbose_name_plural = _('Degree categories')
        ordering = ("name",)

    def __str__(self):
        """Unicode representation of DegreeCategory."""

        return "[{}] {}".format(self.pk, self.name)

    def clean(self):
        """Clean method for DegreeCategory.

        Check if the creator is a promoted-group's user.
        """

        super().clean()

    def get_absolute_url(self):
        """Return absolute url for DegreeCategory."""

        return reverse('degreecategory_detailview', kwargs={'pk': self.pk})


class Degree(BackOfficeResource):
    """Model definition for Degree.

    A degree is a back-office general representation of a collection of
    modules leading to a diploma when a student graduate. All the modules from
    the degree must been passed in order to pass the degree itself.
    """

    title = models.CharField(max_length=255, verbose_name="Title")
    reference = models.CharField(max_length=7, unique=True, blank=True,
                                 verbose_name=_('Reference'))
    category = models.ForeignKey(
        DegreeCategory,
        null=True,
        on_delete=models.SET_NULL,
        related_name="degrees",
        verbose_name=_("Category")
    )
    modules = models.ManyToManyField(
        Module,
        related_name="degrees",
        verbose_name=_("Modules")
    )
    description = models.TextField(null=True, blank=True,
                                   verbose_name=_("Description"))
```

```
    picture = models.ImageField(
        upload_to='management/degrees/',
        default='management/undraw_Books_l33t.png',
        null=True,
        blank=True,
        max_length=225,
        verbose_name=_("Picture"),
    )

    class Meta:
        """Meta definition for Degree."""

        verbose_name = _('Degree')
        verbose_name_plural = _('Degrees')
        ordering = ('title', 'reference')

    @property
    def nb_modules(self):
        """Get the total of modules being part of the degree."""

        return self.modules.count()

    @property
    def total_ECTS_value(self):
        """Compute the total ECTS value of the degree."""

        total_ects = 0
        for module in self.modules.all():
            total_ects += module.ECTS_value

        return total_ects

    @property
    def total_costs(self):
        """Compute the total costs of the degree."""

        total_costs = 0
        for module in self.modules.all():
            total_costs += module.cost

        return total_costs

    @property
    def total_price(self):
        """Compute the total charges price of the degree."""

        total_costs = 0
        for module in self.modules.all():
            total_costs += module.price
```

```
        return total_costs

    @property
    def total_benefits(self):
        """Compute the benefits margin made by one instance of the module."""

        return self.total_price - self.total_costs

    def __str__(self):
        """Unicode representation of Degree."""

        return "[{}] ({}) {}".format(self.pk, self.reference, self.title)

    def clean(self):
        """Clean method for Degree.

        Check if the creator is a promoted-group's user.
        """

        super().clean()

    def save(self, *args, **kwargs):
        """Save method for Degree.

        Add a reference based on the degree's title and pk.
        """

        self.reference = self.title[0:4].upper()

        if not self.pk:
            super().save(*args, **kwargs)

        self.reference += str(self.pk).zfill(3)
        super().save(*args, **kwargs)

    def get_absolute_url(self):
        """Return absolute url for Degree."""

        return reverse('degree_detailview', kwargs={'pk': self.pk})
```

serina-project/management/models/module.py

```
from django.contrib.auth.models import User
```

```python
from django.core.exceptions import ValidationError
from django.db import models
from django.shortcuts import reverse
from django.utils.translation import ugettext as _

from .resource import BackOfficeResource


class ModuleLevel(BackOfficeResource):
    """Model definition for ModuleLevel."""

    rank = models.PositiveIntegerField(unique=True, verbose_name=_("Rank"))
    name = models.CharField(max_length=50, verbose_name=_("Name"))

    class Meta:
        """Meta definition for ModuleLevel."""

        verbose_name = _('Module level')
        verbose_name_plural = _('Module levels')
        ordering = ("rank",)

    def __str__(self):
        """Unicode representation of ModuleLevel."""

        return "[{}] (Rank: {}) {}".format(self.pk, self.rank, self.name)

    def clean(self):
        """Clean method for ModuleLevel.

        Check if the creator is a promoted-group's user.
        """

        super().clean()

    def get_absolute_url(self):
        """Return absolute url for ModuleLevel."""

        return reverse('modulelevel_detailview', kwargs={'pk': self.pk})


class Module(BackOfficeResource):
    """Model definition for Module.

    A module is a back-office general representation of a given course.
    A degree is composed of multiple modules.. Only a group of specific
    teachers can teach the module. Some modules cannot be done if the
    prerequisites modules are not finished yet.
    """
```

```
    title = models.CharField(max_length=255, verbose_name=_('Module'))
    reference = models.CharField(max_length=7, blank=True, unique=True,
                                 verbose_name=_('Reference'))
    description = models.TextField(null=True, blank=True,
                                   verbose_name=_("Description"))
    level = models.ForeignKey(
        ModuleLevel,
        null=True,
        on_delete=models.SET_NULL,
        related_name="modules",
        verbose_name=_("Difficultiy level")
    )
    prerequisites = models.ManyToManyField(
        "self",
        blank=True,
        symmetrical=False,
        related_name="postrequisites",
        verbose_name=_("Prerequisites")
    )
    eligible_teachers = models.ManyToManyField(
        User,
        blank=True,
        related_name="teachable_modules",
        verbose_name=_("Eligible teachers")
    )
    ECTS_value = models.PositiveIntegerField(null=True, blank=True,
                                             verbose_name=_("ECTS value"))
    cost = models.DecimalField(
        null=True,
        max_digits=5,
        decimal_places=2,
        verbose_name=_('Cost'),
    )
    price = models.DecimalField(
        null=True,
        max_digits=5,
        decimal_places=2,
        verbose_name=_('Charge price'),
    )
    picture = models.ImageField(
        upload_to='management/modules/',
        default='management/undraw_Books_l33t.png',
        null=True,
        blank=True,
        max_length=225,
        verbose_name=_("Picture"),
    )

    class Meta:
```

```
        """Meta definition for Module."""

        verbose_name = _('Module')
        verbose_name_plural = _('Modules')
        ordering = ('title', 'reference')

    @property
    def module_benefits(self):
        """Compute the benefits margin made by one instance of the module."""

        return self.price - self.cost

    @property
    def courses_benefits(self):
        """Compute the benefits margin made by all the module's courses."""

        return self.courses.count() * self.module_benefits

    def __str__(self):
        """Unicode representation of Module."""

        return "[{}] ({}) {}".format(self.pk, self.reference, self.title)

    def clean(self):  # TODO: Fix validators
        """Clean method for Module.

        Check if the creator is a promoted-group's user, if the
        eligible_teachers are from the 'Teacher'-group, if the module has
        not itself has prerequisite and if a postrequisite has been added as
        prerequisite as well.
        """

        # Creator must be a promoted user
        super().clean()

        # TODO: Hide eligible_teachers and prerequisites fields on creation in
 admin
        # LINK: https://books.agiliq.com/projects/django-admin-
cookbook/en/latest/uneditable_existing.html
        if self.pk:
            # eligible_teachers must be "Teacher"-group members
            # TODO: Filter choices in M2M box in admin panel
            for user in self.eligible_teachers.all():
                if not user.groups.filter(name="Teacher").exists():
                    raise ValidationError(
                        _("{} cannot be added as eligible teacher. The user is
 not"
                          " a 'Teacher'-group member.".format(user.username))
                    )
```

```
            # Module can not be its own prerequisite
            # TODO: Does not work on creation, 'pk' empty before save !
            # TODO: Filter choices in M2M box in admin panel
            if self.prerequisites.filter(pk=self.pk).exists():
                raise ValidationError(
                    _("This module can not be its own prerequisite.")
                )

            # prerequisite module cannot be postrequisite too
            # TODO: Does not work on creation, 'pk' empty before save !
            # TODO: Not OK
            # TODO: Filter choices in M2M box in admin panel
            for module in self.prerequisites.all():
                if self.postrequisites.filter(pk=module.pk).exists():
                    raise ValidationError(
                        _("[{0}] {1} cannot be a prerequisite for this module.
"
                          " [{0}] {1} already has this module as prerequisite.
"
                          .format(module.reference, module.title))
                    )

    def save(self, *args, **kwargs):
        """Save method for Module.

        Prevent adding eligible_teachers and prerequisites on creation.
        Otherwise, the validations in the clean() won't work (They need
        self.pk which isn't defined yet on creation. These fields can be
        populated on update. Also add a reference based on the module's title
        and pk after generating the pk if it wasn't defined yet.
        """

        self.reference = self.title[0:4].upper()

        if not self.pk:
            super().save(*args, **kwargs)
            self.eligible_teachers.clear()
            self.prerequisites.clear()

        self.reference += str(self.pk).zfill(3)
        super().save(*args, **kwargs)

    def get_absolute_url(self):
        """Return absolute url for Module."""

        return reverse('module_detailview', kwargs={'pk': self.pk})
```

serina-project/management/models/resource.py

```
from django.conf import settings
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError
from django.db import models
from django.utils.translation import ugettext as _

from registration.utils.groups import is_back_office_user


class BackOfficeResource(models.Model):
    """Model definition for BackOfficeResource.

    A ressource contains a creation and last update timestamp.
    The BackOfficeResource model is inherited by each back-office model.
    """

    created_by = models.ForeignKey(
        User,
        null=True,
        on_delete=models.SET_NULL,
        related_name="created_%(class)s",
        verbose_name=_('Created by')
    )
    date_created = models.DateTimeField(auto_now_add=True,
                                        verbose_name=_('Created on'))
    date_updated = models.DateTimeField(auto_now=True,
                                        verbose_name=_('Updated on'))

    class Meta:
        """Meta definition for BackOfficeResource."""

        abstract = True

    def clean(self):
        """Check if a the created_by user is part of a promoted group and
        raise an error otherwise.

        The promoted groups are 'Professor', 'Manager' and 'Administrator'.
        """

        if not is_back_office_user(self.created_by):
            raise ValidationError(
                _("{} is not allowed to perform tasks. This action must be "
                  "performed by a back-
office user. Please contact the support"
                  " team ({}) for more information."
                  .format(
```

```
                    self.created_by.username,
                    settings.CONTACT_MAILS["support"]),
                )
            )
```

serina-project/management/models/room.py

```python
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError
from django.db import models
from django.shortcuts import reverse
from django.utils.translation import ugettext as _

from .resource import BackOfficeResource


class Classroom(BackOfficeResource):
    """Model definition for Classroom.

    A Classroom is a room which can be assigned to a course and has a
    specific capacity.
    """

    name = models.CharField(max_length=50, verbose_name=_("Name"))
    reference = models.CharField(max_length=7, blank=True, unique=True,
                                 verbose_name=_('Reference'))
    description = models.TextField(
        null=True,
        blank=True,
        verbose_name=_("Description")
    )
    recommended_capacity = models.PositiveIntegerField(
        verbose_name=_("Recommended capacity")
    )
    max_capacity = models.PositiveIntegerField(
        verbose_name=_("Maximum capacity")
    )
    picture = models.ImageField(
        upload_to='management/rooms/',
        default='management/undraw_Books_l33t.png',
        null=True,
        blank=True,
        max_length=225,
        verbose_name=_("Picture"),
    )

    class Meta:
        """Meta definition for Classroom."""
```

```
        verbose_name = _('Classroom')
        verbose_name_plural = _('Classrooms')
        ordering = ('name','reference')

    def __str__(self):
        """Unicode representation of Classroom."""

        return "({}) {} (Capacity: {}/{})".format(
            self.reference,
            self.name,
            self.recommended_capacity,
            self.max_capacity
        )

    def clean(self):
        """Clean method for Classroom.

        Check if the creator is a promoted-group's user.
        """

        # Creator must be a promoted user
        super().clean()

        # recommended_capacity cannot be higher than max_capacity
        if self.recommended_capacity > self.max_capacity:
            raise ValidationError(
                _("The recommended capacity ({}) cannot be higher than the "
                  "maximum capacity ({}).".format(self.recommended_capacity,
                                                  self.max_capacity))
            )

    def save(self, *args, **kwargs):
        """Save method for Classroom.

        Append the Classroom's pk with leading zeros to the label in order to
        make it unique.
        """

        self.reference = self.name[0:4].upper()

        if not self.pk:
            super().save(*args, **kwargs)

        self.reference += str(self.pk).zfill(3)
        super().save(*args, **kwargs)

    def get_absolute_url(self):
        """Return absolute url for Classroom."""
```

```
        return reverse('classroom_detailview', kwargs={'pk': self.pk})
```

serina-project/management/views/course.py

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.shortcuts import render
from django.urls import reverse_lazy
from django.views.generic import DeleteView, DetailView, ListView

from ..forms import CourseCreateForm, CourseUpdateForm
from ..models import Course
from .resource import (
    BackOfficeResourceCreateViewMixin,
    BackOfficeResourceUpdateViewMixin,
)
from registration.utils.mixins import ManagerAdministratorOnlyMixin


class CourseListView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                     ListView):  # TODO: Debug view
    """ListView for Course."""

    model = Course
    context_object_name = "courses"
    template_name = "management/course/course_listview.html"
    paginate_by = 10


class CourseDetailView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       DetailView):  # TODO: Debug view
    """DetailView for Course."""

    model = Course
    context_object_name = "course"
    template_name = "management/course/course_detailview.html"


class CourseCreateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       BackOfficeResourceCreateViewMixin):  # TODO: Debug view
    """CreateView for Course."""

    model = Course
    form_class = CourseCreateForm
    template_name = "management/course/course_createview.html"


class CourseUpdateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
```

```
                        BackOfficeResourceUpdateViewMixin):  # TODO: Debug view
    """UpdateView for Course."""

    model = Course
    form_class = CourseUpdateForm
    context_object_name = "course"
    template_name = "management/course/course_updateview.html"


class CourseDeleteView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       DeleteView):  # TODO: Debug view
    """DeleteView for Course."""

    model = Course
    context_object_name = "course"
    template_name = "management/course/course_deleteview.html"
    success_url = reverse_lazy('course_listview')
```

serina-project/management/views/degree.py

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.shortcuts import render
from django.urls import reverse, reverse_lazy
from django.views.generic import DeleteView, DetailView, ListView

from ..forms import DegreeCreateForm, DegreeCategoryForm, DegreeUpdateForm
from ..models import Degree, DegreeCategory
from .resource import (
    BackOfficeResourceCreateViewMixin,
    BackOfficeResourceUpdateViewMixin,
)
from registration.utils.mixins import ManagerAdministratorOnlyMixin


# Degree

class DegreeListView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                     ListView):  # TODO: Debug view
    """ListView for Degree."""

    model = Degree
    template_name = "management/degree/degree_listview.html"
    context_object_name = "degrees"
    paginate_by = 10


class DegreeDetailView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       DetailView):  # TODO: Debug view
```

```
    """DetailView for Degree."""

    model = Degree
    template_name = "management/degree/degree_detailview.html"
    context_object_name = "degree"


class DegreeCreateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       BackOfficeResourceCreateViewMixin):  # TODO: Debug view
    """CreateView for Degree."""

    model = Degree
    form_class = DegreeCreateForm
    template_name = "management/degree/degree_createview.html"
    success_url = reverse_lazy('degree_listview')


class DegreeUpdateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       BackOfficeResourceUpdateViewMixin):  # TODO: Debug view
    """UpdateView for Degree."""

    model = Degree
    form_class = DegreeUpdateForm
    template_name = "management/degree/degree_updateview.html"

    def get_success_url(self):
        """Redirect the user to the newly created DegreeDetailView."""

        return reverse('degree_detailview', kwargs={"pk": self.object.pk})


class DegreeDeleteView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       DeleteView):  # TODO: Debug view
    """DeleteView for Degree."""

    model = Degree
    template_name = "management/degree/degree_deleteview.html"
    context_object_name = "degree"
    success_url = reverse_lazy('degree_listview')


# DegreeCategory

class DegreeCategoryListView(LoginRequiredMixin, ManagerAdministratorOnlyMixin
,
                             ListView):  # TODO: Debug view
    """ListView for DegreeCategory."""

    model = DegreeCategory
```

```
    template_name = "management/degree/degreecategory_listview.html"
    context_object_name = "categories"
    paginate_by = 10


class DegreeCategoryDetailView(LoginRequiredMixin,
                               ManagerAdministratorOnlyMixin, DetailView):  #
TODO: Debug view
    """DetailView for DegreeCategory."""

    model = DegreeCategory
    template_name = "management/degree/degreecategory_detailview.html"
    context_object_name = "category"


class DegreeCategoryCreateView(
    LoginRequiredMixin,
    ManagerAdministratorOnlyMixin,
    BackOfficeResourceCreateViewMixin,
):  # TODO: Debug view
    """CreateView for DegreeCategory."""

    model = DegreeCategory
    form_class = DegreeCategoryForm
    template_name = "management/degree/degreecategory_createview.html"

    def get_success_url(self):
        """Redirect the user to the newly created DegreeCategoryDetailView."""

        return reverse('degreecategory_detailview',
                       kwargs={"pk": self.object.pk})


class DegreeCategoryUpdateView(
    LoginRequiredMixin,
    ManagerAdministratorOnlyMixin,
    BackOfficeResourceUpdateViewMixin,
):  # TODO: Debug view
    """UpdateView for DegreeCategory."""

    model = DegreeCategory
    form_class = DegreeCategoryForm
    context_object_name = "category"
    template_name = "management/degree/degreecategory_updateview.html"


class DegreeCategoryDeleteView(LoginRequiredMixin,
                               ManagerAdministratorOnlyMixin, DeleteView):  #
TODO: Debug view
```

```
    """DeleteView for DegreeCategory."""

    model = DegreeCategory
    template_name = "management/degree/degreecategory_deleteview.html"
    context_object_name = "category"
    success_url = reverse_lazy('degreecategory_listview')
```

serina-project/management/views/module.py

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.shortcuts import render
from django.urls import reverse_lazy
from django.views.generic import DeleteView, DetailView, ListView

from ..forms import ModuleCreateForm, ModuleUpdateForm, ModuleLevelForm
from ..models import Module, ModuleLevel
from .resource import (
    BackOfficeResourceCreateViewMixin,
    BackOfficeResourceUpdateViewMixin,
)
from registration.utils import registration
from registration.utils.mixins import ManagerAdministratorOnlyMixin



# Module views

class ModuleListView(ListView):
    """ListView for Modules."""

    model = Module
    context_object_name = "modules"
    template_name = "management/module/module_listview.html"
    paginate_by = 10


class ModuleDetailView(DetailView):
    """DetailView for Modules."""

    model = Module
    context_object_name = "module"
    template_name = "management/module/module_detailview.html"

    def get_context_data(self, **kwargs):
        """Add 'already_validated' and 'all_prerequisites_validated' to the
        context of each module."""

        context = super().get_context_data(**kwargs)
```

```
        context["already_validated"] = registration \
            .module_already_validated_by_user(self.request.user, self.object)

        context["all_prerequisites_validated"] = registration \
            .all_prerequisites_validated_by_user(self.request.user,
                                                 self.object)

        return context


class ModuleCreateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       BackOfficeResourceCreateViewMixin):
    """CreateView for Modules."""

    model = Module
    form_class = ModuleCreateForm
    template_name = "management/module/module_createview.html"


class ModuleUpdateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       BackOfficeResourceUpdateViewMixin):
    """UpdateView for Modules."""

    model = Module
    form_class = ModuleUpdateForm
    context_object_name = "module"
    template_name = "management/module/module_updateview.html"


class ModuleDeleteView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                       DeleteView):
    """DeleteView for Modules."""

    model = Module
    context_object_name = "module"
    template_name = "management/module/module_deleteview.html"
    success_url = reverse_lazy('module_listview')


# ModuleLevel Views

class ModuleLevelListView(ListView):
    """ListView for ModuleLevels."""

    model = ModuleLevel
    context_object_name = "levels"
    template_name = "management/module/modulelevel_listview.html"
```

```
class ModuleLevelDetailView(DetailView):
    """DetailView for ModuleLevels."""

    model = ModuleLevel
    context_object_name = "level"
    template_name = "management/module/modulelevel_detailview.html"


class ModuleLevelCreateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                            BackOfficeResourceCreateViewMixin):
    """CreateView for ModuleLevels."""

    model = ModuleLevel
    form_class = ModuleLevelForm
    template_name = "management/module/modulelevel_createview.html"


class ModuleLevelUpdateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                            BackOfficeResourceUpdateViewMixin):
    """UpdateView for ModuleLevels."""

    model = ModuleLevel
    form_class = ModuleLevelForm
    context_object_name = "level"
    template_name = "management/module/modulelevel_updateview.html"


class ModuleLevelDeleteView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                            DeleteView):
    """DeleteView for ModuleLevels."""

    model = ModuleLevel
    template_name = "management/module/modulelevel_deleteview.html"
    context_object_name = "level"
    success_url = reverse_lazy('modulelevel_listview')
```

serina-project/management/views/resource.py

```
# TODO: Move this file to a mixin file

from django.views.generic import (
    CreateView,
    UpdateView
)


# TODO: Find a way to merge those two mixins.
#       Find the common mixin used by both.
```

```python
class BackOfficeResourceCreateViewMixin(CreateView):
    """BackOfficeResourceEditViewMixin that populate the 'created_by' field by
    the current user."""

    def get_initial(self):
        """Returns the initial data to use for forms on this view."""

        initial = super().get_initial()
        initial['created_by'] = self.request.user
        return initial


class BackOfficeResourceUpdateViewMixin(UpdateView):
    """BackOfficeResourceEditViewMixin that populate the 'created_by' field by
    the current user."""

    def get_initial(self):
        """Returns the initial data to use for forms on this view."""

        initial = super().get_initial()
        initial['created_by'] = self.request.user
        return initial
```

serina-project/management/views/room.py

```python
from django.contrib.auth.mixins import LoginRequiredMixin
from django.shortcuts import render
from django.urls import reverse_lazy
from django.views.generic import DeleteView, DetailView, ListView

from ..forms import ClassroomForm
from ..models import Classroom
from .resource import (
    BackOfficeResourceCreateViewMixin,
    BackOfficeResourceUpdateViewMixin,
)
from registration.utils.mixins import ManagerAdministratorOnlyMixin


class ClassroomListView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                        ListView):  # TODO: Debug view
    """ListView for Classroom."""

    model = Classroom
    template_name = "management/room/classroom_listview.html"
    context_object_name = "classrooms"
    paginate_by = 10
```

```
class ClassroomDetailView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                          DetailView):  # TODO: Debug view
    """DetailView for Classroom."""

    model = Classroom
    template_name = "management/room/classroom_detailview.html"
    context_object_name = "classroom"


class ClassroomCreateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                          BackOfficeResourceCreateViewMixin):  # TODO: Debug v
iew
    """CreateView for Classroom."""

    model = Classroom
    form_class = ClassroomForm
    template_name = "management/room/classroom_createview.html"


class ClassroomUpdateView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                          BackOfficeResourceUpdateViewMixin):  # TODO: Debug v
iew
    """UpdateView for Classroom."""

    model = Classroom
    form_class = ClassroomForm
    context_object_name = "classroom"
    template_name = "management/room/classroom_updateview.html"


class ClassroomDeleteView(LoginRequiredMixin, ManagerAdministratorOnlyMixin,
                          DeleteView):  # TODO: Debug view
    """DeleteView for Classroom."""

    model = Classroom
    context_object_name = "classroom"
    template_name = "management/room/classroom_deleteview.html"
    success_url = reverse_lazy('classroom_listview')
```

serina-project/management/forms/course.py

```
from django import forms
from django.contrib.auth.models import User

from ..models import Classroom, Course, Module
from .resource import (
    BackOfficeResourceFormMixin,
    ClassroomChoiceField,
```

```
    ModuleChoiceField,
    TeacherChoiceField,
)


# Course forms

class CourseCreateForm(BackOfficeResourceFormMixin):
    """ModelForm for Course."""

    module = ModuleChoiceField(queryset=Module.objects.all(), empty_label=None
)
    teacher = TeacherChoiceField(queryset=None, required=False)
    room = ClassroomChoiceField(queryset=Classroom.objects.all(),
                                required=False)

    def __init__(self, *args, **kwargs):
        """Init of the 'prerequisites' and the 'eligible_teacher'-fields
        queryset."""

        super().__init__(*args, **kwargs)
        self.fields['teacher'].queryset = \
            User.objects.filter(groups__name="Teacher")

    class Meta(BackOfficeResourceFormMixin.Meta):
        """Meta definition for ModuleLevelForm."""

        model = Course


class CourseUpdateForm(CourseCreateForm):
    """UpdateForm for Course."""

    def __init__(self, *args, **kwargs):
        """Init of the 'prerequisites' and the 'eligible_teacher'-fields
        queryset."""

        super().__init__(*args, **kwargs)
        self.fields['teacher'].queryset = \
            User.objects.filter(groups__name="Teacher") \
                .filter(teachable_modules=self.instance.module)
```

serina-project/management/forms/degree.py

```
from django import forms

from ..models import Degree, DegreeCategory, Module
from .resource import (
```

```
    BackOfficeResourceFormMixin,
    CategoryLevelChoiceField,
    ModuleMultipleChoiceField,
    TeacherMultipleChoiceField,
)

# Degree forms

class DegreeCreateForm(BackOfficeResourceFormMixin):
    """ModelForm for Degree creation."""

    category = CategoryLevelChoiceField(queryset=DegreeCategory.objects.all(),
                                        empty_label=None)

    class Meta(BackOfficeResourceFormMixin.Meta):
        """Meta definition for DegreeCreateForm."""

        model = Degree
        exclude = ("reference", "modules")


class DegreeUpdateForm(BackOfficeResourceFormMixin):
    """ModelForm for Degree update.

    Prevent the user to add the instance to its own prerequisites. Also preven
t
    adding a postrequisite module to the prerequisites."""

    category = CategoryLevelChoiceField(queryset=DegreeCategory.objects.all(),
                                        empty_label=None)
    modules = ModuleMultipleChoiceField(queryset=Module.objects.all(),
                                        required=False)

    class Meta(BackOfficeResourceFormMixin.Meta):
        """Meta definition for ModuleLevelForm."""

        model = Degree
        exclude = ("reference",)


# DegreeCategory forms

class DegreeCategoryForm(BackOfficeResourceFormMixin):
    """ModelForm for DegreeCategory."""

    class Meta(BackOfficeResourceFormMixin.Meta):
        """Meta definition for ModuleLevelForm."""

        model = DegreeCategory
```

serina-project/management/forms/module.py

```
from django import forms
from django.contrib.auth.models import User

from ..models import Module, ModuleLevel
from .resource import (
    BackOfficeResourceFormMixin,
    CategoryLevelChoiceField,
    ModuleMultipleChoiceField,
    TeacherMultipleChoiceField,
)


# Module forms

class ModuleCreateForm(BackOfficeResourceFormMixin):
    """ModelForm for Module."""

    level = CategoryLevelChoiceField(queryset=ModuleLevel.objects.all(),
                                     empty_label=None)

    class Meta(BackOfficeResourceFormMixin.Meta):
        """Meta definition for ModuleLevelForm."""

        model = Module
        exclude = ("reference", "prerequisites", "eligible_teachers")


class ModuleUpdateForm(BackOfficeResourceFormMixin):
    """ModelForm for Module.

    Prevent the user to add the instance to its own prerequisites. Also preven
t
    adding a postrequisite module to the prerequisites."""

    level = CategoryLevelChoiceField(queryset=ModuleLevel.objects.all(),
                                     empty_label=None)
    prerequisites = ModuleMultipleChoiceField(queryset=None, required=False)
    eligible_teachers = TeacherMultipleChoiceField(queryset=None,
                                                   required=False)

    def __init__(self, *args, **kwargs):
        """Init of the 'prerequisites' and the 'eligible_teacher'-fields
        queryset."""

        super().__init__(*args, **kwargs)
```

```
        self.fields['prerequisites'].queryset = \
            Module.objects.exclude(pk=self.instance.pk) \
                          .exclude(prerequisites=self.instance)

        self.fields['eligible_teachers'].queryset = \
            User.objects.filter(groups__name="Teacher")

    class Meta(BackOfficeResourceFormMixin.Meta):
        """Meta definition for ModuleLevelForm."""

        model = Module
        exclude = ("reference",)


# ModuleLevel forms

class ModuleLevelForm(BackOfficeResourceFormMixin):
    """ModelForm for ModuleLevel."""

    class Meta(BackOfficeResourceFormMixin.Meta):
        """Meta definition for ModuleLevelForm."""

        model = ModuleLevel
```

serina-project/management/forms/resource.py

```
# TODO: Move this file to a mixin file

from django import forms
from django.utils.translation import ugettext as _


class BackOfficeResourceFormMixin(forms.ModelForm):
    """Mixin for ModelForms thats hide the 'created_by' field and exclude the
    'reference' field if it exist."""

    class Meta:
        """Meta definition for BackOfficeResourceFormMixin."""

        exclude = ("reference",)
        widgets = {
            'created_by': forms.HiddenInput(),
        }


# ModelChoiceFields and ModelMutipleChoiceFields customization

class CategoryLevelChoiceField(forms.ModelChoiceField):
```

```
    """Display a formatted name for each DegreeCategory and ModuleLevel in the
    ModelChoiceField."""

    def label_from_instance(self, category_or_level):
        return "{}".format(category_or_level.name)


# TODO: Merge these two classes by finding common inherrited mixin

class ModuleChoiceField(forms.ModelChoiceField):
    """Display the reference and the title of each module in the
    ChoiceField."""

    def label_from_instance(self, module):
        return "{} ({})".format(module.title, module.reference)


class ModuleMultipleChoiceField(forms.ModelMultipleChoiceField):
    """Display the reference and the title of each module in the
    MultipleChoiceField."""

    def label_from_instance(self, module):
        return "{} ({})".format(module.title, module.reference)


# TODO: Merge these two classes by finding common inherrited mixin

class TeacherChoiceField(forms.ModelChoiceField):
    """Display the full name of each teacher in the ChoiceField."""

    def label_from_instance(self, teacher):
        return "{} ({})".format(teacher.get_full_name(), teacher.username)


class TeacherMultipleChoiceField(forms.ModelMultipleChoiceField):
    """Display the full name of each teacher in the MultipleChoiceField."""

    def label_from_instance(self, teacher):
        return "{} ({})".format(teacher.get_full_name(), teacher.username)


class ClassroomChoiceField(forms.ModelChoiceField):
    """Display the full name of each teacher in the ChoiceField."""

    def label_from_instance(self, room):
        return _("{} (Capacity: {}/{})").format(
            room.name,
            room.recommended_capacity,
            room.max_capacity
```

```
        )
```

serina-project/management/forms/room.py

```python
from django import forms

from ..models import Classroom
from .resource import BackOfficeResourceFormMixin


# Classroom forms

class ClassroomForm(BackOfficeResourceFormMixin):
    """ModelForm for Classroom."""

    class Meta(BackOfficeResourceFormMixin.Meta):
        """Meta definition for ModuleLevelForm."""

        model = Classroom
```

5.3.3. Module de notation

serina-project/rating/urls.py

```
from django.conf.urls import url

from . import views


urlpatterns = [
    url(
        r"l/$",
        views.StudentRatingListView.as_view(),
        name="rating_listview"
    ),
    url(
        r"r/(?P<pk>[0-9]+)/$",
        views.StudentRatingDetailView.as_view(),
        name="rating_detailview"
    ),
    url(
        r"c/$",
        views.StudentRatingCreateView.as_view(),
        name="rating_createview"
    ),
    url(
        r"u/(?P<pk>[0-9]+)/$",
        views.StudentRatingUpdateView.as_view(),
        name="rating_updateview"
    ),
    url(
        r"d/(?P<pk>[0-9]+)/$",
        views.StudentRatingDeleteView.as_view(),
        name="rating_deleteview"
    ),
]
```

serina-project/rating/models.py

```
from django.contrib.auth.models import User
from django.core.validators import MaxValueValidator, MinValueValidator
from django.db import models
from django.urls import reverse
from django.utils.translation import ugettext as _

from management.models import Module
```

```
class StudentRating(models.Model):
    """Model definition for StudentRating.

    A StudentRating is a rate left by a student that succeeded a module.
    This is in order to give a feed-back to the teachers and improve  how the
    module is been teached and evaluated.
    """

    created_by = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name="ratings",
        verbose_name=_("Student"),
    )
    date_created = models.DateField(auto_now_add=True,
                                    verbose_name=_("Creation date"))
    date_updated = models.DateField(auto_now=True,
                                    verbose_name=_("Creation date"))
    module = models.ForeignKey(
        Module,
        on_delete=models.CASCADE,
        related_name="ratings",
        verbose_name=_("Module"),
    )
    rate = models.PositiveIntegerField(
        validators=[MinValueValidator(1), MaxValueValidator(5)],
        verbose_name=_("Rate"),
    )
    comment = models.TextField(verbose_name=_("Comment"))

    class Meta:
        """Meta definition for StudentRating."""

        verbose_name = 'Rating'
        verbose_name_plural = 'Ratings'
        ordering = ("-date_updated",)

    def __str__(self):
        """Unicode representation of StudentRating."""

        return "[{}] {}'s rating on {}".format(
            self.pk,
            self.created_by.get_full_name(),
            self.module.title,
        )

    # def save(self):
    #     """Save method for StudentRating."""
```

```
        # TODO: Prevent student from leaving multiple rates on the same module

    def get_absolute_url(self):
        """Return absolute url for StudentRating."""

        return reverse('rating_detailview', kwargs={'pk': self.pk})
```

serina-project/rating/views.py

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.shortcuts import render
from django.views import generic
from django.urls import reverse_lazy

from .forms import StudentRatingForm
from .models import StudentRating
from registration.utils import mixins as mixins_utils


class StudentRatingListView(LoginRequiredMixin, generic.ListView,):  # TODO: D
ebug view
    """ListView for StudentRating"""

    model = StudentRating
    context_object_name = "ratings"
    template_name = "rating/rating_listview.html"
    paginate_by = 10


class StudentRatingDetailView(LoginRequiredMixin, generic.DetailView,):  # TOD
O: Debug view
    """DetailView for StudentRating"""

    model = StudentRating
    context_object_name = "rating"
    template_name = "rating/rating_detailview.html"


class StudentRatingCreateView(
    LoginRequiredMixin,
    mixins_utils.StudentOnlyMixin,
    generic.CreateView,
    mixins_utils.AutofillCreatedByRequestUser,
):  # TODO: Debug view
    """CreateView for StudentRating"""

    model = StudentRating
    form_class = StudentRatingForm
```

```
    template_name = "rating/rating_createview.html"


class StudentRatingUpdateView(
    LoginRequiredMixin,
    mixins_utils.SelfStudentManagerAdministratorOnlyMixin,
    generic.UpdateView,
    mixins_utils.AutofillCreatedByRequestUser,
):  # TODO: Debug view
    """UpdateView for StudentRating"""

    model = StudentRating
    form_class = StudentRatingForm
    context_object_name = "rating"
    template_name = "rating/rating_updateview.html"


class StudentRatingDeleteView(
    LoginRequiredMixin,
    mixins_utils.SelfStudentManagerAdministratorOnlyMixin,
    generic.DeleteView,
    mixins_utils.AutofillCreatedByRequestUser,
):  # TODO: Debug view
    """DeleteView for StudentRating"""

    model = StudentRating
    form_class = StudentRatingForm
    template_name = "rating/rating_deleteview.html"
    success_url = reverse_lazy('module_listview')
```

serina-project/rating/forms.py

```
from .models import StudentRating
from management.models import Module
from registration.utils import mixins as mixins_utils


class StudentRatingForm(mixins_utils.HideCreatedByFieldFormMixin):
    """ModelForm for StudentRating."""

    module = mixins_utils.VerboseDegreeModuleChoiceField(
        queryset=Module.objects.all(),
        empty_label=None,
    )

    class Meta(mixins_utils.HideCreatedByFieldFormMixin.Meta):
        """Meta definition for StudentRegistrationForm."""
```

```
        model = StudentRating
        fields = ("created_by", "module", "rate", "comment",)
```

6. Conclusion

Le défi était à ma hauteur mais le manque de temps ne m'a malheureusement pas permis d'en profiter pleinement pour montrer mes compétences. Dans un contexte plus favorable, l'expérience aurait pu être plus agréable tout en étant plus aboutie.

Le projet met un point d'honneur à ce que l'application respecte au mieux les exigences les plus communes dans le monde professionnel. L'application qui en découle reste générique afin de pouvoir être utilisée par tous et reste ouverte à l'amélioration et donc à son extension selon les demandes particulières d'un client.

L'application de principes les plus élémentaires dans le monde du développement a fortement contribué à la conception et la mise en place de modules facilement réutilisables tout en assurant une maintenance minime. L'automatisation du processus de développement par des pipelines et sa grande portabilité sont un atout de choix étant donné que ces processus rendent le développement, le déploiement et la maintenance de l'application très facile, ce qui représente un coût nettement plus faible pour le client final.

La conséquence directe de cette mise en place est une installation extrêmement simplifiée et complètement automatisée, s'adaptant à pratiquement tous les environnements et pouvant donc être reproduites sur pratiquement n'importe quelle machine.

Même si la contrainte de temps a eu raison de la qualité du frontend, notamment à cause de l'absence du framework React.js, le backend et l'analyse restent acceptables malgré mes exigences élevées. La conteneurisation et la démarche DevOps restent cependant les atouts majeurs du projet.
Ce n'est certainement pas ma meilleure œuvre, mais j'en éprouve aucune honte, en particulier vis-à-vis du temps et des conditions difficiles n'ayant pas aidé à son aboutissement complet.

7. Bibliographie

BEKY, Ariane. Mise à jour : le 29-08-2019. « Python 2 : le clap de fin à haut risque » sur *Silicon.fr*. Site Web sur INTERNET. <https://www.silicon.fr/python-2-clap-fin-risque-259807.html>. Dernière consultation : le 24-08-2020.

BRIDAY, Guillaume. Mise à jour : le 24-02-2018. « Installer et utiliser les GitLab Runners » sur *GuillaumeBriday.fr*. Site Web sur INTERNET. <https://guillaumebriday.fr/installer-et-utiliser-les-gitlab-runners>. Dernière consultation : le 06-08-2020.

Bruno. Mise à jour : le 11-06-2019. « Nginx est maintenant le serveur web le plus utilisé par les sites les plus fréquentés au monde » sur *Developpez.com*. Site Web sur INTERNET. <https://web.developpez.com/actu/265652/Nginx-est-maintenant-le-serveur-web-le-plus-utilise-par-les-sites-les-plus-frequentes-au-monde-devant-Apache-et-Microsoft-IIS-selon-W3Tech/>. Dernière consultation : le 24-08-2020.

C., Florian. Mise à jour: le 10-06-2019. « Qu'est-ce qu'un framework ? » sur Wild Code School. Site Web sur INTERNET. <https://www.wildcodeschool.com/fr-FR/blog/quest-ce-quun-framework>. Dernière consultation : le 19-01-2020.

CALAMIER, Romain. Mise à jour : le 09-08-2018. « GraphQL: Et pour quoi faire ? » sur *OCTO talks*. Site Web sur INTERNET. <https://blog.octo.com/graphql-et-pourquoi-faire/>. Dernière consultation : le 16-08-2020.

Christophe. Mise à jour : le 08-03-2018. « ORM » sur *Base de données*. Site Web sur INTERNET. <https://www.base-de-donnees.com/orm/>. Dernière consultation : le 13-08-2020.

D-BOOKER. 2012-2020. *PostgreSQL : Robuste, performant, stable et open-source*. Site Web sur INTERNET. <https://www.d-booker.fr/content/72-postgresql>. Dernière consultation : le 24-08-2020.

GIT. 2020. *1.1 Démarrage rapide - À propos de la gestion de version*. Site Web sur INTERNET. <https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-%C3%80-propos-de-la-gestion-de-version>. Dernière consultation : le 06-08-2020.

Malick. Mise à jour : le 17-10-2018. « Quels sont vos frameworks Web Python préférés en 2018 ? Pourquoi ? » sur *Developpez.com*. Site Web sur INTERNET. <https://www.developpez.com/actu/229336/Quels-sont-vos-frameworks-Web-Python-preferes-en-2018-Pourquoi-Partagez-vos-avis/>. Dernière consultation : le 24-08-2020.

MARCILLAUD, Matthieu. Mise à jour : le 29-12-2008. « Qu'est-ce qu'un pipeline ? » sur *Programmer.Spip.net*. Site Web sur INTERNET. <https://programmer.spip.net/Qu-est-ce-qu-un-pipeline>. Dernière consultation : le 06-08-2020.

MICROSOFT AZURE. 2020. *Qu'est-ce que le DevOps ?*. Site Web sur INTERNET. <https://azure.microsoft.com/fr-fr/overview/what-is-devops/>. Dernière consultation : le 06-08-2020.

PYTHON. 2020. *PEP 8 -- Style Guide for Python Code*. Site Web sur INTERNET. <https://www.python.org/dev/peps/pep-0008/>. Dernière consultation : le 24-08-2020.

REGNAULT, Camille. Mise à jour : le 24-02-2017. « GitLab, c'est quoi ? » sur *AXOPEN*. Site Web sur INTERNET. <https://blog.axopen.com/2017/02/gitlab-cest-quoi/>. Dernière consultation : le 14-08-2020.

ROUSE, Margaret. Mise à jour : le 24-02-2016. « Conteneur (container) » sur *LeMagIT*. Site Web sur INTERNET. <https://www.lemagit.fr/definition/Conteneurs>. Dernière consultation : le 04-01-2020.

SAUNIER, Sébastien. Mise à jour : le 06-09-2018. « Pourquoi apprendre Ruby on Rails ? » sur *le wagon*. Site Web sur INTERNET. <https://www.lewagon.com/fr/blog/apprendre-ruby-on-rails>. Dernière consultation : le 24-08-2020.

TAIEB, John. 2020. « Pourquoi apprendre Python ? » sur *apprendre-a-coder.com*. Site Web sur INTERNET. <https://apprendre-a-coder.com/pourquoi-apprendre-python/>. Dernière consultation : le 24-08-2020.