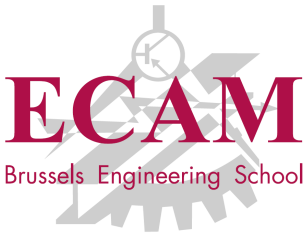
Haute Ecole LEONARD de VINCI



Institut Supérieur Industriel

Design, développement et intégration à une plateforme de gestion d'école d'un module de gestion des stages

Travail de fin d'études présenté par

Damien VANHOVE

En vue de l'obtention du diplôme de

Master en Sciences de l'Ingénieur Industriel orientation Informatique

Année académique 2016-2017

CAHIER DES CHARGES RELATIF au TRAVAIL DE FIN D'ETUDES de

Damien Vanhove inscrit(e) en 5EI

Année académique :
2016-2017

Titre provisoire :
Design, implémentation et intégration à une plateforme de gestion d'école d'un module de gestion des stages.

Objectifs à atteindre :

Découvrir, comprendre, exploiter une plateforme existante et son processus de développement. Prise en main du stack *MEAN*, utilisation de *git* et de *gitHub*, tests avec le système *Travis*.

Design et proposition du *user design experience* pour chaque acteur concerné. Création de diagrammes d'activités.

Implémentation de la solution après concertation avec le chef de projet et tests unitaires de cette dernière.

Test d'utilisabilité auprès des différents acteurs.

Principales étapes :

Prise en main du stack *MEAN*, de *git*, de *gitHub* ainsi que la plateforme concernée. Cela se fera par la création d'un module ajoutant une fonctionnalité supplémentaire au site.

Interview des différents acteurs et intervenants. Recherche des besoins utilisateur, compréhension du domaine et du processus à informatiser. Design du *user experience*. Création de mock-ups.

Conception des modèles de données, identification, utilisation et modification éventuelle des modèles existant.

Cycles de développement en méthodologie Agile (développement par étapes et tests consécutifs).

Tests auprès des utilisateurs finaux.

Fait en trois exemplaires à Bruxelles, le 21/11/2016

L'Etudiant	Le Tuteur	Le Promoteur
Nom-prénom : Vanhove Damien	Nom-prénom : LURKIN Quentin	Nom-prénom : COMBEFIS Sébastien
	Département/Unité ..... GEI	Société ..... ECAM .....
Signature	Signature	Signature

Je tiens à remercier tous ceux qui, de près ou de loin, ont pu contribuer à la réussite de ce travail de fin d’études.

En premier lieu, je souhaite remercier les personnes interrogées durant les enquêtes de ce projet. Je les remercie pour le temps qu’ils ont pu me consacrer, ainsi que pour leurs explications, indispensables au développement de mon travail.

Je souhaite également remercier le Dr Ir. Sébastien Combéfis, enseignant à l’ECAM et administrateur de la plateforme OpenSM, de m’avoir donné l’occasion d’effectuer ce travail de fin d’études. J’espère que ce module de gestion des stages lui permettra de perfectionner d’avantage sa plateforme, et que la finalité de ce projet ravira tant les élèves que les membres du corps enseignant le l’ECAM.

Je tenais enfin à remercier mes parents pour leur patiente et leur soutien.

Table des matières

1 Introduction

Chaque année à l'ECAM, les élèves en troisième année de bachelier en sciences industrielles suivent obligatoirement un stage d'immersion dans le milieu professionnel. Les démarches administratives assurant le suivi des stages pour chaque élève font partie d'un mécanisme qui a depuis longtemps été adopté par membres de l'ECAM. Si les mécanismes administratifs actuels ont su faire leur preuve d'année en année, ils sont parfois jugés peu modernes par ceux qui les utilisent. Dans le cadre de ce travail de fin d'études, il m'a été demandé de développer une solution informatisée intégrée qui permettra de moderniser le suivi des stages de troisièmes bachelier et de rendre les démarches administratives les accompagnant plus souples et confortables.

Le sujet de ce TFE s'inscrit dans une lignée de démarches progressistes visant à moderniser la gestion de l'ECAM. Parmi ces démarches, nous pouvons citer l'installation d'écrans (valves électroniques), ou même la création d'une plateforme de gestion des cours et des examens. Pour une école d'ingénieurs offrant notamment un master en informatique à la pointe, la modernité est un élément clé pour assurer un enseignement de qualité. Cette modernité ne s'applique cependant pas uniquement aux matières enseignées aux élèves, mais passe également par l'informatisation progressive des démarches administratives.

2 Analyse Fonctionnelle, Cahier des charges

Cette section présente les enquêtes qui ont été menées auprès des utilisateurs futurs de l'application. Ce qui ressort de ces enquêtes est ensuite utilisé dans le but d'établir une analyse fonctionnelle de l'application, et ainsi dresser un cahier des charges fidèle aux besoins du projet.

2.1 Enquête et Analyses

Les enquêtes auprès des utilisateurs constituent la première étape vers l'élaboration d'un cahier des charges. Leur objectif est de faire un état des lieux de la situation actuelle, et de répondre à trois questions clés qui sont indispensables au projet :

(Q1) **Quels sont les acteurs qui participent au stage ?**

Il est important d'identifier toutes les parties prenantes qui interviennent lors du déroulement d'un stage. Ensuite, il est nécessaire d'identifier quel est le rôle tenu par chaque intervenant. La réponse à cette question permettra d'identifier les utilisateurs de l'application.

(Q2) **Comment se déroule un stage ?**

Il est nécessaire de comprendre le mécanisme général qui permet de mener à bien le suivi d'un stage de son début jusqu'à sa fin. Quelles sont les étapes à franchir ? Les conditions à remplir? La réponse à ces question permettra de comprendre la chronologie du déroulement d'un stage.

(Q3) **Quels sont les souhaits et besoins de chaque acteur ?**

Si la première question a permis d'identifier qui étaient les acteurs, celle-ci doit permettre d'identifier leurs besoins. À quelles informations doivent-ils avoir accès ? Lesquels doivent-ils pouvoir modifier ? La réponse à ces questions permettra de structurer les données et de définir les actions les modifiant.

Afin de répondre à ces questions, j'ai mené mon enquête auprès de trois membres du corps enseignant de l'-unité génie électrique et informatique :

- François Defrance, enseignant et chef de l'unité. Il est notamment en charge de gérer les ressources humaines de cette dernière.
- Sébastien Combéfis, enseignant et superviseur de stage.
- André Lorge, enseignant et coordinateur des stages.

Afin de mieux comprendre les aspects purement administratifs de la gestion d'un stage, j'ai également interrogé Madame Bernadette Vandevenne, secrétaire administrative de l'ECAM.

2.1.1 Enquête auprès de Monsieur Francois Defrance

La première personne interrogée était Monsieur Defrance, qui est le chef de l'unité génie électrique et informatique . Il m'a expliqué que la première chose que devait faire un est de rédiger une proposition de stage, et que l'encodage de cette dernière était déjà en partie informatisée. Pour être approuvée, cette proposition doit être faite en accord avec l'entreprise dans laquelle l'étudiant souhaiterait effectuer son stage. L'étudiant doit également faire approuver sa proposition de stage par le coordinateur de stages ainsi que par un enseignant quelconque.

Un autre rôle du chef d'unité est de donner son accord pour le stage de l'étudiant une fois que les formalités administratives ont été remplies. Cette validation équivaut à dire « *je suis au courant que j'ai un étudiant en stage* ». Si tout est en ordre, Madame Vandevenne sort alors la convention de stage.

Ensuite, le chef d'unité attribue les stages aux différents enseignants de son unité qui deviendront alors superviseurs de stage. Dans la proposition de stage, l'élève peut indiquer sa préférences pour le superviseur de stage qu'il souhaiterait se voir attribuer. Le chef d'unité peut ainsi s'aider des préférences des élèves pour attribuer les superviseurs. Dans le cas où un étudiant n'a pas encore de préférence ou de superviseur attribué, Monsieur Defrance envoie une liste des élèves aux enseignants. Si un enseignant le souhaite, il peut exprimer son souhait de superviser un étudiant en particulier. S'en suit alors le déroulement du stage.

Au début du stage, l'élève rédige la note d'activités. Cette note reprend les objectifs généraux et spécifiques du stage, et doit être signée par l'entreprise, l'étudiant, puis communiquée au superviseur.

A alors lieu une première visite du superviseur en entreprise pour s'assurer que le stage se déroule correctement. Une évaluation formative est ensuite prévue entre l'élève et le maître de stage au milieu du stage. Le maître de stage peut encoder ce qu'on appelle l'évaluation intermédiaire, et peut également faire des commentaires à l'élève.

S'en suit alors l'évaluation finale, qui est remplie par le maître de stage seul ou avec le superviseur. Ce dernier doit ensuite réencoder les points de l'élève sur la plateforme de l'école.

À la fin du stage, l'élève doit envoyer sur papier à entête de l'entreprise son certificat de stage. Après que celui-ci ait été enregistré et validé par Madame Vandevenne, l'étudiant doit rendre son rapport de stage, dont la remise est également enregistrée par Madame Vandevenne.

Finalement, le superviseur de stage doit encoder les points finaux obtenus par l'étudiant. Un e-mail est ensuite envoyé au maître de stage pour être certain que les deux parties sont bien d'accords la note finale obtenue par l'étudient.

2.1.2 Enquêtes auprès de Messieurs Sébastien Combéfis et André Lorge

Les résultats des enquêtes auprès de Messieurs Sébastien Combéfis et André Lorge ont été regroupés car forts similaires. Ce qui suit décrit ce qui est ressorti de mon enquête auprès de ces deux enseignants.

L'élève commence tout d'abord par soumettre sa proposition de stage. Deux acteurs ont alors accès à cette proposition : premièrement, un enseignant quelconque que l'étudiant aura consulté, et qui donnera son accord ou non pour le sujet du stage. Deuxièmement, le coordinateur d'année qui devra vérifier cette proposition et y apposer sa signature. L'enseignant consulté peut également accepter de suivre le stage en temps que superviseur si l'élève en a fait le souhait.

Un validateur, souvent le responsable d'unité, doit ensuite attribuer un enseignant superviseur à chaque stage de l'unité. Si un enseignant a déjà été proposé, et que le validateur ainsi que l'enseignant sont d'accords avec la proposition faite par l'étudiant, il suffit de valider l'attribution du superviseur. En revanche s'il n'y a pas eu de proposition, il est nécessaire de choisir un enseignant pour superviser le stage.

Pour l'attribution des superviseurs, Madame Lefebvre doit avoir une vue d'ensemble de tous les élèves et professeurs. Si un élève n'a pas de superviseur, elle doit pouvoir lui en fournir un. Elle fait en quelque sorte office de « *filet de sécurité* », et évite qu'un étudiant ne se retrouve sans superviseur de stage.

Le validateur de la section donne ensuite son accord pour la mise en stage des élèves. Cela équivaut administrativement à dire "*je suis au courant que j'ai un élève en stage*". Une fois que tout est valide, Madame Vandevenne sort la convention de stage. Cette dernière doit être signée par l'entreprise, l'étudiant, et l'Ecam. Une fois rendue et validée, Madame Lefebvre la signe et la renvoie à l'entreprise avec une lettre d'accompagnement. La convention de stage fixe les dates de début et de fin du stage, et définit donc aussi la date butoir des délivrables.

Une fois le stage débuté, l'élève rédige la note d'activité en accord avec le maître de stage, et doit également obtenir l'accord du superviseur pour qu'elle soit validée.

Durant le stage, l'élève doit organiser la première visite du superviseur dans l'entreprise. Il est nécessaire de garder une trace de ce passage. S'en suit alors une évaluation intermédiaire faite avec le maître de stage et qui est purement formative.

Durant tout le stage, l'étudiant doit également tenir un journal de bord qu'il remplit de manière régulière. Le maître de stage ainsi que le superviseur doivent pouvoir consulter ce journal de bord s'ils le souhaitent.

À la fin du stage, l'étudiant organise sa défense orale. Il est important, comme pour la première visite en entreprise, de garder une trace de cette présentation. L'évaluation continue est faite uniquement par le maître de stage, alors que l'évaluation de la présentation est faite par le maître de stage et superviseur en accord commun.

Une fois le stage terminé, l'étudiant doit alors rendre son rapport de stage. Si le stage se termine à la fin de la date butoir, l'étudiant dispose d'une semaine supplémentaire pour le rendre.

Le certificat de stage est à rendre en même temps que le rapport de stage. Madame Vandevenne reçoit les deux documents, les valide, et marque ensuite qu'ils ont bien été rendus.

En dernier lieu, le superviseur encode les points de l'évaluation continue, ainsi que la défense orale. Il évalue ensuite le rapport de stage et encode les points obtenus par l'élève.

Lors de mon enquête auprès de Monsieur Combéfis et Monsieur Lorge, tous deux ont exprimé la volonté de voir un log mis en place pour la proposition de stage. Ils souhaitent ainsi pouvoir garder un historique des propositions faites par l'élève. Il était également souhaité que l'étudiant puisse tenir à jour son journal de bord de façon informatisée, mais qu'il soit limité à une entrée par jour.

2.1.3 Enquête auprès de Madame Bernadette Vandevenne

La dernière personne que j'ai interrogée était Madame Vandevenne. Nous n'allons pas revenir ici sur les différentes étapes de déroulement d'un stage, car ce qui est ressorti de mon entretien avec elle n'a pas apporté d'informations supplémentaires par rapport à ce qui a déjà été présenté. En revanche, cet entretien a été l'occasion de voir les stages sous un autre angle, et ainsi de mettre en avant des besoins supplémentaires d'un point de vue de la gestion administrative.

Madame Vandevenne gère les dossiers de tous les élèves confondus. Elle a su me décrire certaines des fonctionnalités dont elle souhaiterait disposer dans l'application, et qui, d'après elle, faciliteraient sa gestion des stages :

- Une vue récapitulative, sous forme d'un tableau ou d'une liste, afin de voir tous les élèves en un coup d'oeil.
- Pouvoir déterminer rapidement où en est chaque élève, si celui-ci est en ordre administrativement ou non.
- Une fiche récapitulative avec toutes les informations sur le stage d'un élève.
- Une série de filtres pour trier les stages, et pouvoir ainsi directement accéder aux propositions, conventions, remises de rapport…

Madame Vandevenne se charge également de valider les conventions de stage, et de noter qu'elles ont bien été délivrées et reçues. Elle souhaite pouvoir indiquer facilement, pour chaque élève, si ces différents documents ont été rendus et validés. Si possible, un mécanisme similaire serait d'une grande utilité pour les autres délivrables des stages, à savoir le rapport et certificat de stage.

2.1.4 Conclusion des enquêtes

Cette enquête nous ont permis de récolter une quantité importante d'informations qui vont nous être utiles pour le développement du projet.

Tout d'abord, nous avons pu répondre à **Q1** et avons identifié les acteurs impliqués dans le déroulement d'un stage :

- l'Élève
- Le Maître de stage
- Le Superviseur de stage
- L'Administrateur de la plateforme.
- L'Enseignant (quelconque)
- Madame Vandevenne
- Madame Lefebvre
- Le Coordinateur d'année
- Le validateur de section

En plus d'avoir identifié ces acteurs, nous avons également récolté toutes les informations pour répondre à **Q2** et **Q3**. En particulier, nous avons mis le doigt sur certains de besoins des acteurs, et nous avons également vu que plusieurs démarches administratives étaient nécessaires lors du déroulement d'un stage.

Cette enquête fait office d'une première approche un peu grossière, mais nécessaire au projet. Nous pouvons dès à présent analyser le sujet de façon beaucoup plus formelle, et ainsi dresser un cahier des charges en bon et du forme.

2.2 Cahier des charges

L’analyse des résultats des enquêtes que nous avons menées a permis d’obtenir une vision d’ensemble du problème qui se présentait. Le cahier des charges qui suit formalise la solution que nous proposons.

2.2.1 Introduction au problème posé

Les démarches administratives liées au déroulement et suivi des stages à l’ECAM se sont depuis longtemps faites au format papier. Cette méthode a fait ses preuves, et un effort pour informatiser une partie de la démarche à déjà été fait. Ainsi, par exemple, les propositions de stage sont encodées via un formulaire web. Néanmoins, la solution actuelle suscite certaines critiques. Beaucoup trouvent que les démarches sont lourdes, lentes, et parfois opaques. Les élèves ne s’y retrouvent d’ailleurs souvent pas. La quantité de papiers à gérer est également assez importante, ce qui rend les recherches et démarches plus compliquées et moins efficaces pour l’administration.

2.2.2 Objectifs

Ce projet vise à informatiser les démarches administratives liées au déroulement et suivi des stages de troisième année. Il se veut également un outil de communication efficace entre le trio élève, superviseur et maître de stage. Avec ce projet, nous espérons améliorer plusieurs points :

- Centraliser les données.
- Faciliter les démarches administratives tant pour les élèves que pour le corps enseignant et l’administration.
- Gagner en traçabilité.
- Offrir une meilleure transparence, visibilité et compréhension des démarches.
- Permettre aux utilisateurs de gagner du temps.
- Faciliter l’emploi ultérieur des données pour des analyses et statistiques.

2.2.3 Périmètre

Dès le départ, cette application se veut générale, et vise avant tout à gérer des stages, peu importe leur contexte. Elle doit être modulaire et facilement adaptable aux besoins de l’école dans laquelle il sera déployé. Dans le cadre de l’ECAM, le projet est plus restreint et se concentre sur les stages de 3ème année. S’il se montre à la hauteur, l’objectif serait de faire évoluer le projet vers les stages et travaux de fin d’études des étudiants de 5ème année, pour ensuite s’étendre à l’ensemble de la Haute École .

2.2.4 Description fonctionnelle

Au terme de son développement, le programme proposé devra correspondre à un certain niveau fonctionnel. Par conséquent, il est attendu que le programme offre les fonctionnalités principales listées ci-dessous.

(F1) Encodage et validation de la proposition de stage

1.1. Encoder les informations générales de l'entreprise
1.2. Encoder la proposition de stage
1.3. Modifier la proposition de stage
1.4. Valider ou non la proposition de stage
1.5. Ajouter des commentaires

(F2) Attribution d'un superviseur

2.1. Choix d'une préférence pour le superviseur
2.2. Accepter ou non de superviser un stage
2.3. Valider le choix d'un superviseur
2.4. Attribuer un superviseur

(F3) Gérer ses stages

3.1. Créer et gérer sa liste de stages
3.2. Voir l'avancement du stage, voir les étapes suivantes à suivre

(F4) Encoder la note d'activités

4.1. Encoder la note d'activités, ajouter des objectifs généraux et spécifiques
4.2. Valider ou non la note d'activités
4.3. Modifier la note d'activités
4.4. Ajouter des commentaires sur la note d'activités

(F5) Planification de la première visite du superviseur en entreprise

5.1. Encoder le lieu et la date
5.2. Prendre des notes
5.3. Valider le bon déroulement de la visite

(F6) Planification de l'évaluation intermédiaire

6.1. Encoder le lieu et la date
6.2. Prendre des notes
6.3. Valider le bon déroulement de l'évaluation

(F7) Planification de la défense orale

7.1. Encoder le lieu et la date
7.2. Prendre des notes
7.3. Valider le bon déroulement de la présentation

(F8) Journal de bord

8.1. Remplir le journal de bord de façon journalière
8.2. Consulter le journal de bord

(F9) Gérer les contacts avec les entreprises

9.1. Encoder l'entreprise, son adresse
9.2. Encoder et gérer les contacts au sein de l'enterprise

(F10) Gérer l’ensemble des stages

10.1. Lister tous les stages
10.2. Consulter chaque stage
10.3. Filtrer les stages, n’en montrer que certains, faire des recherches
10.4. Montrer différentes informations sur les stages en fonction du critère choisi
10.5. Valider la convention de stage
10.6. Gérer les dates de début et de fin d’un ou plusieurs stages
10.7. Valider que les différents délivrables ont bien été rendus
10.8. Ajouter un stage
10.9. Supprimer un stage
10.10. Modifier les informations d’un stage

(F11) Gérer uniquement ses propres stages

11.1. Pouvoir différencier l’ensemble des stages et le ou les stages qui concernent l’utilisateur
11.2. Restreindre l’accès aux stages uniquement aux ayants-droit

(F12) Gestion informatisée des documents publics et administratifs

12.1. Générer des conventions automatiquement, pouvoir les imprimer
12.2. Offrir une zone de dépôt pour les rapports
12.3. Pouvoir consulter une bibliothèque des rapports
12.4. Offrir un répertoire des documents administratifs à destination de l’élève

(F13) Notifications automatiques par e-mail

13.1. Rappels en cas de manque d’action d’un utilisateur
13.2. Rappels pour les réunions, défenses, deadlines des documents
13.3. Un broadcast de warnings destinés à des batch d’étudiants filtrés

3 Analyse technique

Ce projet de gestion des stages n'est pas un projet stand-alone, mais plutôt une extension pour une plateforme existante. OpenSM est un projet open-source développé par le Dr Ir. Sébastien Combéfis sur base de la stack MEAN. *Open School Management* se veut une plateforme polyvalente intégrant tous les composants entrant dans la gestion d'un environnement éducatif scolaire.

OpenSM est divisé en plusieurs parties, en plusieurs outils de gestion d'éléments spécifiques de l'école. Parmi ces outils, nous retrouvons la gestion des auditoires, des cours (activités), des copies d'examens, des sessions d'examens, ainsi que la gestion de l'ensemble des utilisateurs de la plateforme, des membres de la haute école. Ces parties ou blocs de programme sont indépendants et appelés modules. Dans le cadre de ce TFE, c'est donc un module de gestion des stages que nous étudions.

3.1 Aperçu de La stack MEAN

MEAN est une stack software open-source écrite entièrement en JavaScript. Il permet de construire des sites et applications web dynamiques et est composé de quatre éléments : MongoDB, Express, NodeJS dans le backend, et AngularJS dans le frontend.

Il existe principalement deux stacks MEAN, Mean.js et Mean.io, qui découlement d'un même projet inicial projet : Mean.io. Ce projet open-source était et continue d'être sponsorisé par la compagnie Linnovate. Après un litige avec le directeur du projet, il a été de le forker, ce qui donnât la version Mean.JS. C'est cette version que nous avons employé pour ce projet.

3.1.1 NodeJS

NodeJS est le serveur backend de notre application. Ce serveur est entièrement écrit en JavaScript, rapide pour les applications fortement interactives, et idéal pour les single page applications. NodeJS bénéficie d'une grande communauté de développeurs très actifs. Comme il repose sur JavaScript, NodeJS est asynchrone par nature, ce qui veut dire qu'il se base sur un modèle d'entrées-sorties non bloquant. Ainsi, il peut gérer plusieurs tâches sans bloquer le thread principal. En revanche, NodeJS est mono-thread. Dans certains cas, des tâches lourdes telles que des gros calculs peuvent bloquer le serveur pour tous les utilisateurs durant quelques secondes.

Pourquoi ne pas l'utiliser ?

De très bas niveau, il n'est pas adapté aux débutants en JavaScript. Son architecture mono-thread est également handicapante dans le cas de calculs intensifs. NodeJS est asynchrone, et repose donc très lourdement sur des fonctions de callback. Des queues de callbacks imbriquées peuvent très rapidement détériorer la qualité du code, surtout si le développeur manque d'expérience. L'API change également de façon relativement fréquente, ce qui force les développeurs à faire des modifications au codebase pour le rendre compatible avec l'API NodeJS.

Pourquoi l'utiliser ?

NPM, le gestionnaire de packages de NodeJS est devenu énorme et continue à grandir de jour en jour. Il utilise le moteur JavaScript V8. Écrit en C, ce moteur très performant est plus rapide que Ruby, Python, ou Perl. NodeJS est également non bloquant de par sa nature asynchrone. Il offre néanmoins la possibilité de programmer des fonctionnalités synchrones au moyen de fonctions de callback. Enfin, NodeJS est léger et dynamique, rendant plus facile l'évolution et l'extension des application qu'avec ses concurrents (LAMP, Rails, Django…)

3.1.2 ExpressJS

NodeJS en stand-alone peut vite devenir compliqué. Étant donné que toute fonctionnalité doit être codée manuellement, sans racourcis, la complexité et quantité de code devient vite très difficile à gérer dès que l'application commence à grandir. Express permet de créer des applications web plus facilement, en offrant une interface plus simple. Ainsi, il est plus aisé de créer des end-points API pour gérer des requêtes, d'éventuels cookies, des sessions utilisateur. C'est en quelque sorte une *« boite à outils »* pour créer des applications avec NodeJS. Si l'on devait écrire les mêmes fonctionnalités avec un NodeJS « *cru* », on devrait écrire beaucoup plus de code.

L'illustration 1 montre la facilité avec laquelle on peut définir des routes REST pour une API. Ce code définit la fonction à exécuter lorsqu'une requête GET est adressée à la route */internship/:id*. Elle imprime l'identifiant passé en paramètre de l'appel de route dans la console.

```
app.get('/internship/:id', function(req, res) {
  console.log(req.params('id'));
});
```

Illustration 1: Définition d'une route REST à l'aide du framework ExpressJS

Express propose aussi un système de middleware qui permet d'enchaîner l'exécution de fonctions synchrones. Un exemple d'utilisation de middleware est l'authentification, qui est un mécanisme qui doit avoir lieu avant toute autre fonction. Il existe également des outils pour parser le corps de requêtes POST, GET, UPDATE, pour créer des headers HTTP… Express regorge donc d'outils en tout genre pour écrire des applications dynamiques en peu de temps.

Pourquoi ne pas l'utiliser ?

Pour certains développeurs, Express peut paraître un peu minimal. Il ne fournit pas une infrastructure assez riche pour étendre son application vers un projet de très grande ampleur. La quantité de code peut très vite grandir et devenir difficile à maintenir. Certaines dépendances à des packages de middleware peut rendre l'application vulnérable aux incompatibilités.

Pourquoi utiliser ExpressJS ?

Facilité d'utilisation, et rapidité de développement grâce à une API simple sont des arguments en faveur de l'utilisation d'ExpressJS. Développé depuis 2009, ExpressJS est stable et bénéficie d'une très large communauté. C'est également un des projets NodeJS les plus utilisés. C'est un bon choix pour des projets de taille moyenne, avec une équipe de développement réduite.

3.1.3 AngularJS

AngularJS est un framework open-source pour applications web développé dans le but de créer des single page applications. Son rôle est de gérer le frontend de notre plateforme, de gérer le routage côté client et de rendre les pages de l'application dynamiques de façon générale. AngularJS utilise HTML comme langage structurel, comme template de base, puis étend la syntaxe HTML pour ajouter de la fonctionnalité aux composants de l'application. Grâce au data-binding d'AngularJS et aux injections de dépendances, il permet de réduire drastiquement la quantité de code nécessaire pour ajouter des comportements dynamiques aux pages de notre plateforme.

Pourquoi ne pas l'utiliser ?

Certains concepts tels que les injections de dépendances peuvent être compliqués pour des développeurs non avertis. L'utilisation de scopes est également un concept qui peut vite devenir compliqué. Des confusions entre scopes locaux et globaux sont un exemple d'erreur typique que l'on peut faire avec ce framework. Les répercussions de ce type d'erreur sont souvent importantes et parfois très difficiles à débuger. L'interconnexion des éléments constitutifs d'AngularJS le rend également compliqué à apprendre pour ceux qui manquent d'expérience avec ce type de framework.

Pourquoi utiliser AngularJS?

Angular est puissant, et permet de créer des applications web riches et dynamiques. Il utilise également le modèle MVC, ce qui permet d'écrire des applications propres et bien structurées. Le two-way data-binding rend la manipulation de données aisée, des changements dans la vue étant répercutés dans le modèle et vice-versa. AngularJS permet de manipuler facilement le DOM, et est compatible avec tous les navigateurs récents. Ce framework est particulièrement adapté au développement et prototypage rapide. L'ensemble du traitement se faisant côté client, il permet également de décharger considérablement la charge imposée côté serveur.

3.1.4 MongoDB

MongoDB est une base de données NoSQL orientée documents. Les données sont structurées au format JSON sous la forme de paires clé-valeur et non plus organisées sous forme de tables. Les bases de données orientées document sont particulièrement bien adaptées aux données formant un agrégat naturel. Dans notre cas, un stage est un ensemble concret. On le lit, pull, modifie et push en temps que bloc unique, ce qui le rend adapté au modèle de type document.

Pourquoi ne pas l'utiliser ?

Si MongoDB offre une certaine souplesse au niveau de la structure des données, il en est autrement pour les requêtes. Un exemple de ce manque de souplesse est l'absence d'opérations de jointure. Pour effectuer une opération équivalente à un JOIN, il est nécessaire de faire plusieurs requêtes et puis de coder un regroupement et filtrage des données à la main. Il n'y a pas non plus de support pour les transactions, ni d'opérations atomiques autrement que pour un document unique.

Pourquoi utiliser MongoDB ?

Il n'y a pas de schéma en base de données NoSQL, ce qui rend la modélisation en amont de la base de données très flexible. Ceci nous permet, par exemple, de rajouter ou de supprimer des champs à un stage facilement, ce qui est beaucoup plus compliqué dans des bases de données relationnelles. Les requêtes MongoDB sont également souvent plus rapides, car les données sont toutes regroupées au même endroit.

3.1.5 Pourquoi MEAN?

L'avantage principal d'employer la stack MEAN est que tout fonctionne avec le même langage, ce qui simplifie grandement le développement. La communication entre les différents composants de la stack est également beaucoup plus facile, tout échange d'information se faisant alors au format JSON. La stack MEAN peut également s'exécuter sur n'importe quel système d'exploitation, et peut donc être déployée à peu près n'importe où. De plus, une application basée sur la stack MEAN est très facile à étendre et à faire grandir, sa structure permettant d'ajouter des blocs fonctionnels sous la forme de modules indépendants.

3.1.6 Pourquoi ne pas employer la stack MEAN ?

En guise de comparaison, la stack MEAN est beaucoup moins stable qu'une stack LAMP classique. La documentation ainsi que le support communautaire sont également moins disponibles car MEAN ne bénéficie pas de la même ancienneté. La prise en mains requière aussi plus de temps et de connaissances avant de pouvoir exploiter pleinement les capacités de la stack.

Il est important de ne pas oublier que MEAN est un projet open-source qui évolue régulièrement. Pour cette raison, et en cas de modifications majeures, un développeur risque de devoir sévèrement revoir le code-base de son projet pour assurer la compatibilité avec la version en cours. Ce risque est particulièrement présent lorsque des modules tiers dont dépendent une application évoluent.

3.2 Une structure modulaire

Nous avons vu que la plateforme OpenSM était découpée en plusieurs modules. Chaque module gère un aspect bien spécifique de la plateforme, et le module internships ancre sa fonctionnalité autour de la gestion des stages. Cette notion de modules dans la stack MEAN nous vient d'AngularJS. Dans AngularJS, il n'existe pas de fichier principal tel qu'une *Main* regroupant l'ensemble des fonctionnalités de l'application. Au lieu de cela, les fonctionnalités sont découpées en modules. Un module, c'est une collection de services, de directives, de contrôleurs, filtres et informations de configuration.

Là où MEAN fait les choses un peu différemment, c'est dans son choix de structure de ces modules. AngularJS se base sur une structure horizontale, présentée à l'illustration 2. Au contraire, la stack MEAN a décidé d'adopter une structure verticale, présentée à l'illustration 3, plus intuitive et logique. Alors que les modules AngularJS représentent plus des unités fonctionnelles (ex : contrôleurs, directives, services), les modules MEAN représentent plutôt des unités logiques par rapport à l'application développée.

Application			
Controllers	Directives	Filters	Services
Controller1	Directive1	Filter1	Service1
Controller2	Directive2	Filter2	Service2
Controller3	Directive3	Filter3	Service3

Illustration 2: Structure unitaire d'AngularJS

Application		
Core Module	Users Module	Articles Module
CoreController1	UsersController1	ArticlesController1
CoreController2	UsersController1	ArticlesController2
CoreService1	UsersService1	ArticlesService1
CoreFilter1	UsersFilter1	ArticlesFilter1
CoreDirective1	UsersDirective1	ArticlesDirective2

Illustration 3: Structure modulaire de MEAN

Dans cette optique, une fonctionnalité de l'application correspond donc à un module. Au moment de la création d'un module, des liens appelés *injections* sont faits avec modules. Grâce à cela, ils peuvent ainsi partager leurs fonctionnalités. Par exemple, dans le module de gestion des examens, l'injection d'un des services du module *users* nous permet d'obtenir la liste des élèves et des enseignants.

3.3 Architecture de la stack MEAN

Cette section décrit une série d'éléments architecturaux de la stack MEAN.

3.3.1 Structure des dossiers

Afin de quelque peu guider la suite des explications concernant l'architecture, il est intéressant de commencer par la structure des dossiers particulière à la stack MEAN.

La racine du projet tel que présentée à l'illustration 4 est divisé en quatre dossiers principaux : c*onfig*, *modules*, *public* et *scripts.*

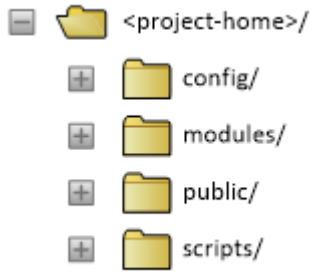


Illustration 4: Structure des dossiers racine de la stack MEAN

Le dossier *config* contient l'ensemble des paramètres de configuration de l'application. Le dossier *Public* contient quant à lui l'ensemble des fichiers statiques du frontend de notre application. Cela comprend, par exemple, des images ou des librairies tierces externes dont dépend l'application. Concernant le dossier *scripts, il* contient des scripts utilitaires parfois utilisés lors du développement.

Le dossier qui nous intéresse le plus est le dossier m*odules*. C'est ce dossier qui va contenir l'ensemble des modules composant l'application. Ces modules sont visibles dans la structure de dossiers présentée à L'illustration 5.

Illustration 5: Structure de dossiers du projet OpenSM

Chaque module est responsable d'un élément fonctionnel de l'application et est divisé en deux sous dossiers. Le dossier *client* contient la logique applicative côté client, et le dossier *server* contient la logique applicative côté serveur. Le dossier t*ests* contient lui l'ensemble des dossiers et fichiers responsables des tests d'unité du module.

La figure 6 présente la structure des dossiers client du module *internship*. Cette structure est ensuite divisée en 4 sous-dossiers. Le dossier *config* contient l'ensemble des informations de routage ainsi que la configuration des menus du front-end. Il est suivi du dossier *controllers* qui contient l'intelligence et la logique applicative gérant les interactions entre l'utilisateur et l'interface graphique. Le dossier *services* contient l'ensemble des services qui gèrent les appels au backend côté serveur, et le dossier *views* contient finalement l'ensemble des templates HTML. Le fichier *internships.client.modules.js* permet de d'enregistrer et de déclarer le module *internships* auprès des autres modules de l'application.

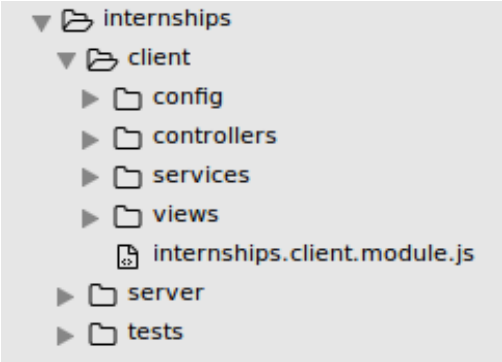


Illustration 6: structure des dossiers client du module internships

La figure 7 présente la structure des dossiers côté serveur. Cette structure est elle aussi disisée en 4 sous dossiers. Le dossier *controllers* contient la logique applicative du serveur. C'est dans ce dossier que nous retrouvons l'ensemble des fonctions qui vont permettre de charger, sauvegarder ou de modifier les données associées à ce module. Le dossier *models* contient les modèles de données. C'est donc dans ce dossier que nous retrouvons le modèle de données correspondant à un stage. Le dossier p*olicies* permet de définir les règles d'accès pour chaque route de l'API REST. Ces routes sont définies dans le dossier *routes* qui établit alors le lien entre les routes de l'API et les fonctions contenues dans le contrôleur.

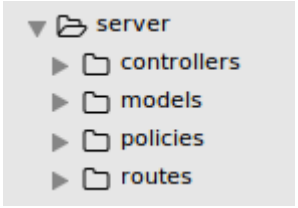


Illustration 7: structure des dossiers serveur

3.3.2 Architecture MVC avec AngularJS

L'architecture MEAN peut se résumer à une architecture client-serveur en 3tier. Côté client, nous retrouvons un premier tier, AngularJS. AngularJS gère l'ensemble des interactions entre la vue et le modèle côté client. Il repose sur une architecture de type Model - View - Controller (MVC), et plus précisément une architecture MVVM, c'est à dire Model - View - ViewModel.

De manière générale, le principe du MVC, présenté à l'illustration 8 est assez simple et direct. Il s'agit d'un pattern architectural qui permet d'implémenter des interfaces utilisateur. Il divise une application en trois parties interconnectées. L'utilisateur voit et interagit avec la vue. La vue utilise et manipule le contrôleur, qui à son tour manipule le modèle. La vue lit alors le modèle.

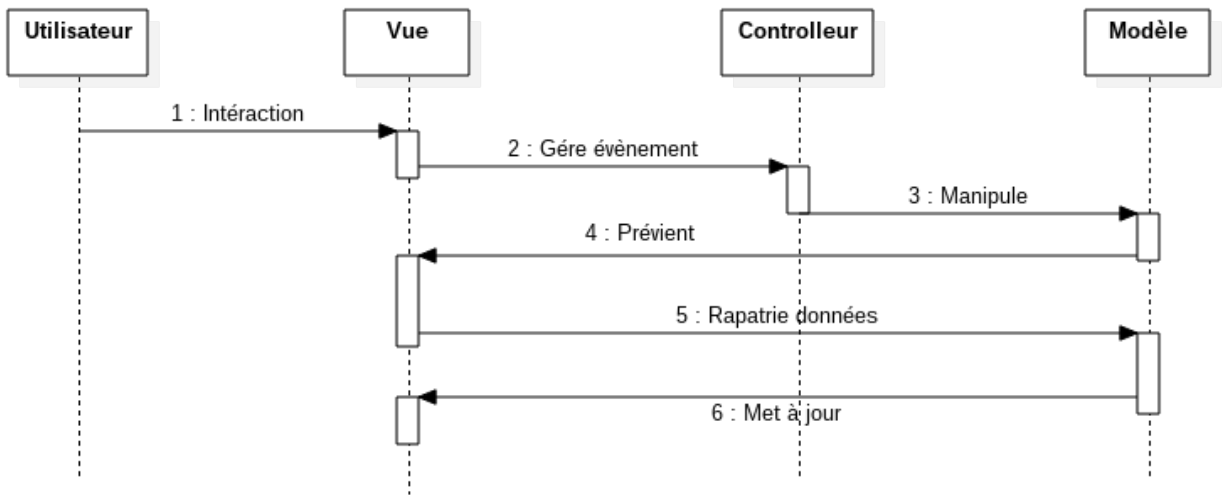


Illustration 8: Modèle MVC

Cette méthode à toutefois un inconvénient. Si on modifie le modèle de données, il est probablement nécessaire de modifier une grande partie de ce qui repose dessus. Aussi, la division entre vue et contrôleur n'est pas toujours très claire. On sépare l'interface mais pas le modèle. Le contrôleur peut également vite devenir très grand. Que se passe-t-il si je veux afficher des points en rouge par exemple ? Je dois alors également stocker dans le modèle la couleur des points, alors que cette information n'est d'utilité que pour la vue. On est ici en présence d'un couplage fort Vue-Modèle, ce qui n'est pas pratique.

AngularJS utilise une variante du MVC, le MVVM. Dans cette architecture, le ViewModel offre un modèle de données et de comportement à la vue qui peut alors se binder à lui. Les interactions entre le ViewModel et la View peuvent être définies dans l'HTML de la vue, et se font aussi dans AngularJS via le *Controller*. Ce mécanisme est repris à l'illustration 9.

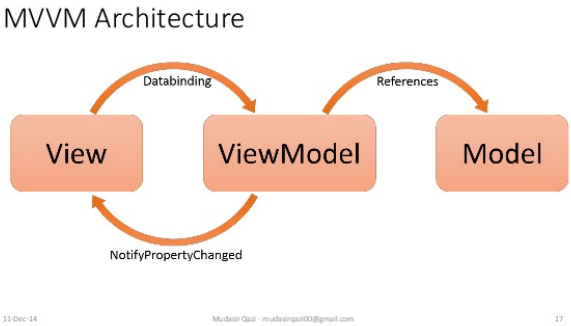


Illustration 9: mécanisme générale du modèle MVVM

Le modèle enveloppe l'ensemble des données. Pour notre application, l'accès à ces données se fera à travers un ensemble de services Web, via une API REST que nous verrons plus loin. Le modèle est séparé du ViewModel de tel façon à ce que les données de la vue (le ViewModel) se retrouvent isolées des vraies données et contiennent une copie de celles-ci. Dans cette architecture, la vue n'a aucune conscience de l'existence du modèle, et le ViewModel ainsi que le Model ignorent l'existence de la vue.

L'illsutration 10 présente le fonctionnement du MVVM. Le ViewModel est un modèle dédié à une vue, et contenant les données provenant du Model dont a besoin la vue. La vue se lie alors à certaines propriétés d'un ViewModel qui expose les données sous forme d'objets modèles dépendant de l'état de la vue. Si des valeurs ou des propriétés changent dans le ViewModel, les changements sont automatiquement propagées vers la vue par data-binding. Lorsqu'un utilisateur clique sur un bouton de la vue, la fonction correspondante dans le viewModel exécute l'action associée. La vue n'effectue jamais de modifications elle même sur le modèle de données, seul le ViewModel peut le faire.

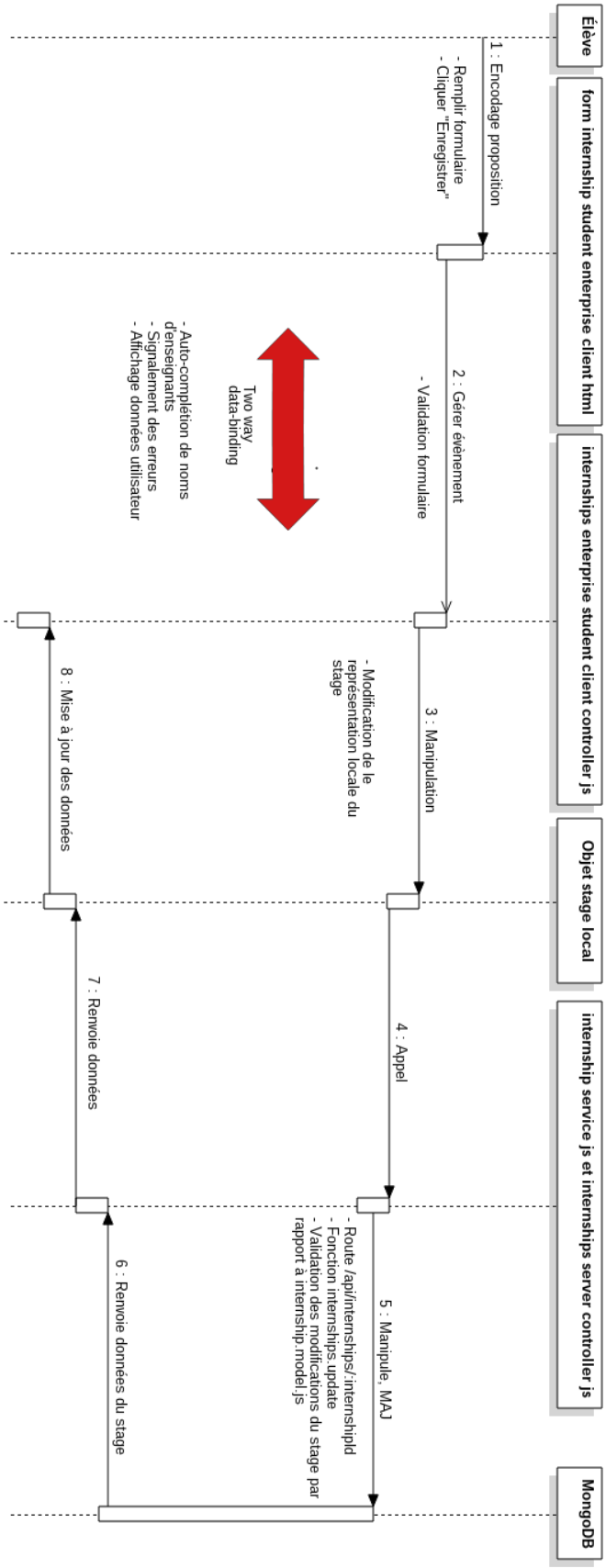


Illustration 10: Exemple du fonctionnement de l'architecture MVVM

3.3.3 Routage applicatif client avec AngularJS

AngularJS gère également le routage au sein de l'application côté client. En fonction du rôle de chaque utilisateur, différents menus sont présentés et différentes routes mises à sa disposition. Ainsi, un élève aura par exemple un menu « *mes stages* » et pourra naviguer au sein de son stage, alors que le coordinateur aura deux menus, un pour tous les stages, et un pour les stages dont il est superviseur. Les routes de navigation au sein de l'application seront alors différentes pour ces deux rôles.

Chaque route est représentée par un *état*. Le routage se fait alors d'un état à un autre. À chaque état correspond un URL ainsi qu'un contrôleur. On associe également un *template* à l'état, c'est-à-dire une vue HTML qui sera bindée au contrôleur.

L'illustration 11 présente un exemple d'état pour l'admin lui permettant de modifier un stage. On y retrouve l'URL, le template correspondant, ainsi que le contrôleur. Le resolve permet d'appeler la fonction *getInternship* qui va chercher les informations du bon stage avant d'ouvrir la page.

```
  .state('admin.manage.internships.edit', {
    url: '/:internshipId/edit',
    templateUrl: 'modules/internships/client/views/form-internship.client.view.html',
    controller: 'InternshipsController',
    controllerAs: 'vm',
    resolve: {
      internshipResolve: getInternship
    },
    data: {
      pageTitle: 'Edit internship'
    }
  });
}
```

Illustration 11: routage administrateur pour l'état manage.internships.edit

3.3.4 Routage Serveur et API

L'Accès à l'ensemble des ressources côté serveur est défini au moyen d'une API. Celle-ci reprend l'ensemble des routes disponibles et permet de faire le lien jusqu'aux fonctionnalités du backend. Le routage définit les routes au moyen d'une série d'URL. Une route peut être associée à plusieurs fonctions, chacune étant dépendante de la méthode avec laquelle la route a été appelée.

Prenons l'exemple de la route présentée à la l'illustration 12, dont l'URL est */api/internships/:internshipId.* Si cette route est appelée via une requête de type GET, c'est la fonction *internship.read* qui sera exécutée. Si elle est appelée avec DELETE, ce sera la fonction *internship.remove* qui sera appellée dans le contrôleur. De façon similaire, PUT provoquera l'exécution de *internships.update*. Cet exemple simple illustre comment le routage permet de faire le lien entre les URL exposées et les fonctionnalités du backend.

```
app.route('/api/internships/:internshipId').all(internshipsPolicy.isAllowed)
  .get(internships.read)
  .delete(internships.remove)
  .put(internships.update);
```

Illustration 12: exemple de routage serveur

Le tableau de la page suivante reprend les routes qui sont accessibles en fonction du rôle des utilisateurs. L'ensemble des fonctionnalités du backend qu'expose notre API sont reprise ci-dessous.

- Opérations groupées sur plusieurs stages
  - Lister tous les stages
  - Mettre à jour les Superviseurs
  - Mettre à jour les statuts des conventions de stage
  - Valider les stages avant leur début
  - Modifier les dates de début et de fin des stages.
- Opérations sur un stage unique
  - Créer un stage vierge, supprimer un stage
  - Lire un stage
  - Mettre à jour un stage dans sa généralité (plusieurs points en même temps)
  - Modifier la date des délivrables du stage
  - Modifier les informations de l'Entreprise du stage
  - Modifier la proposition de stage
  - Ajouter une entrée dans le journal de bord de l'élève
  - Modifier la première visite (date, lieu, commentaires..)
  - Modifier la note d'activités du stage
  - Modifier l'évaluation intermédiaire du stage
  - Modifier la présentation orale du stage
  - Modifier le superviseur attribué au stage.

Rôle	Accès API
Student	/api/internships /api/internships/deadlines /api/internships/:internshipId /api/internships/:internshipId/editEnterprise /api/internships/:internshipId/editProposition /api/internships/:internshipId/editJournal /api/internships/:internshipId/editFirstVisit /api/internships/:internshipId/editActivitiesNote /api/internships/:internshipId/editOralPresentation /api/internships/:internshipId/editIntermediateEvaluation /api/internships/:internshipId/editSupervisor
Master	/api/internships /api/internships/:internshipId /api/internships/:internshipId/editProposition /api/internships/:internshipId/editFirstVisit /api/internships/:internshipId/editOralPresentation /api/internships/:internshipId/editIntermediateEvaluation /api/internships/:internshipId/editActivitiesNote
Manager	/api/internships /api/internships/supervisors /api/internships/convention /api/internships/startEndDates /api/internships/:internshipId /api/internships/validation /api/internships/:internshipId/editSupervisor /api/internships/:internshipId/editEnterprise
Teacher	/api/internships /api/internships/:internshipId /api/internships/:internshipId/editProposition /api/internships/:internshipId/editFirstVisit /api/internships/:internshipId/editSupervisor /api/internships/:internshipId/editOralPresentation /api/internships/:internshipId/editActivitiesNote
Coordinator	/api/internships /api/internships/supervisors /api/internships/:internshipId /api/internships/:internshipId/editProposition
Validator	/api/internships /api/teachers /api/internships/supervisors /api/internships/validation /api/internships/:internshipId /api/internships/:internshipId/editProposition

3.3.5 Contrôle d’accès à l’API

Si les routes serveur permettent d’accéder à des fonctions modifiant un ou plusieurs stages, elles ne peuvent être utilisées que si l’utilisateur y a droit. C’est le rôle du *internshipsPolicy.isAllowed* présent à l’illustration 12 de la page 25. Une des responsabilités du serveur est de contrôler l’accès a ses fonctionnalités. Cela se fait au moyen d’une *Policy.* Elle permet de préciser les droits d’accès au routage, de déterminer quel utilisateur peut accéder à quelle ressource, et ce au moyen de quelle requête.

Chaque utilisateur possède un ou plusieurs rôles définissant ses droits. On distingue plusieurs rôles au sein de l’application :

- **Administrateur** : a tout les droits. Il peut modifier n’importe quels éléments d’un stage.
- **Student** : peut uniquement voir et modifier son ou ses propres stages. En fonction de l’avancée du stage, l’élève peut encoder ou modifier les éléments suivants :
  - Entreprise
  - Proposition de stage
  - Choix du superviseur
  - Note d’activités
  - Journal de bord
  - Première visite
  - Evaluation intermédiaire
  - Défense orale
- **Master** : peut voir et modifier uniquement les stages de ses élèves. Il ne peut apporter des modifications qu’à certaines étapes du stage.
  - Proposition de stage
  - Note d’activité
  - Première visite
  - Évaluation intermédiaire
  - Défense orale
- **Manager** : peut consulter et modifier n’importe quel stage. Il ne peut cependant modifier que les éléments suivants
  - Informations de l’entreprise
  - Superviseur de stage
  - Statut de la convention de stage
  - Dates de début et de fin de stage
  - Deadlines
  - Validation des modalités administratives avant le début du stage
  - Validation du rendu des délivrables (convention, certificat, rapport…)

- **Teacher** : peut accéder au stage d’un étudiant s’il lui a demandé son accord pour un sujet, ou s’il est superviseur de stage. Ce dernier peut modifier :
  - Proposition de stage
  - Acceptation / refus d’une demande de supervisorat.
  - Première visite
  - Note d’activités
  - Défence orale.
- **Coordinator** : A accès à la liste de tous les stages de la section. Ce dernier peut modifier :
  - Superviseur
  - Proposition de stage.
- **Validator** : A accès à la liste de tous les stages. Il peut modifier
  - Superviseurs de stage
  - Validation pré-stage.

3.4 Mongoose ODM et le Schéma de données

Mongoose est un ODM pour MongoDB, c'est à dire l'équivalent d'un ORM en base de données relationnelles, mais ici adapté pour du NoSQL. Ce framework offre la validation des données ainsi qu'une API permetant de formuler des requêtes facilement afin de dialoguer avec la base de données MongoDB.

Mongoose est également nécessaire car l'échange de données entre MongoDB et NodeJS ne se fait pas en JSON comme pour le reste de la stack MEAN. Ce dernier se fait au format BSON, c'est à dire du JSON Binaire. Mongoose traduit le BSON en documents JSON, permetant aux développeurs de continuer à utiliser JavaScript comme langage à travers l'application.

Comme nous avons pu le mentionner, MongoDB n'a pas de schéma, et il est donc nécessaire de passer par un tiers pour valider nos données. C'est dans Mongoose que nous définissons le modèle de données qui sera persisté en base de données. Cet ODM se charge alors de valider les données en concordance avec ce schéma avant d'effectuer des transactions vers la base de données. Le détail de la structure des documents représentant un stage est repris dans les annexes.

3.5 Déroulement chronologique d’un stage

En tenant compte des informations collectées durant l’enquête, le déroulement chronologique d’un stage a été modélisé à la figure 14. Ce modèle correspond à la solution que nous avons jugé la plus pertinente et qui permet au mieux d’informatiser la mécanique de gestion des stages.

1. **L’élève crée son stage**

L’élève commence par accéder à la liste de ses stages. L’élève crée alors un stage qui vient s’ajouter à cette liste. Le stage est pour l’instant vierge et dépourvu d’information. L’élève peut cliquer dessus et accède au tableau de bord de son stage.

2. **Encodage de l’entreprise**

L’élève doit premièrement encoder l’entreprise dans laquelle il compte faire son stage. Il rempli un formulaire contenant les informations sur l’entreprise, son nom, domaine, adresse, mais aussi le représentant de l’entreprise ainsi que sa position.

3. **Encodage et validation de la proposition de stage**

Ce mécanisme est repris de manière graphique à l’illustration 13. Après avoir encodé l’entreprise, l’élève peut encoder sa proposition de stage. Un formulaire lui permet d’encoder le thème, domaine, lieu et description du stage. Il doit également encoder le professeur consulté pour l’accord du sujet, ainsi que le maître de stage. L’élève ne peut alors plus modifier sa proposition de stage avant d’avoir reçu le feedback du maître de stage, de l’enseignant consulté et du coordinateur d’année.

Durant cette période d’attente, l’élève peut encoder l’enseignant qu’il souhaiterait avoir comme superviseur.

Lors de sa connexion, le maître de stage peut voir qu’un étudiant s’est ajouté à sa liste. En cliquant sur l’étudiant concerné, il peut accéder aux fiches des stages de l’étudiant pour lesquels il est maître de stage. Dans cette fiche, il accède et lit la proposition de stage. Il a alors deux choix. Il peut accepter la proposition de stage ou alors la refuser. En cas de refus, il est tenu de laisser un commentaire.

Lorsqu’un enseignant se connecte, s’il a été consulté pour une proposition de stage, une liste avec des demandes d’approbations est générée et lui est présentée. Il peut également lire puis accepter ou refuser la proposition de stage de l’élève. Pareillement, en cas de refus, il est tenu d’indiquer la raison de ce refus.

D’une manière similaire, le coordinateur a accès à une liste des propositions de stage pour lesquels il doit donner son accord. Il peut accepter ou refuser, et doit indiquer la raison en commentaire en cas de refus.

L’élève ne peut accéder à sa proposition de stage QUE si les 3 ont répondu à la proposition. Si l’un deux à refusé la proposition, l’élève peut réencoder sa proposition en tenant compte des commentaires émis. L’ancienne proposition est archivée, et le processus de validation recommence. Si l’élève obtient 3 accords pour sa proposition de stage, il peut passer à l’étape suivante.

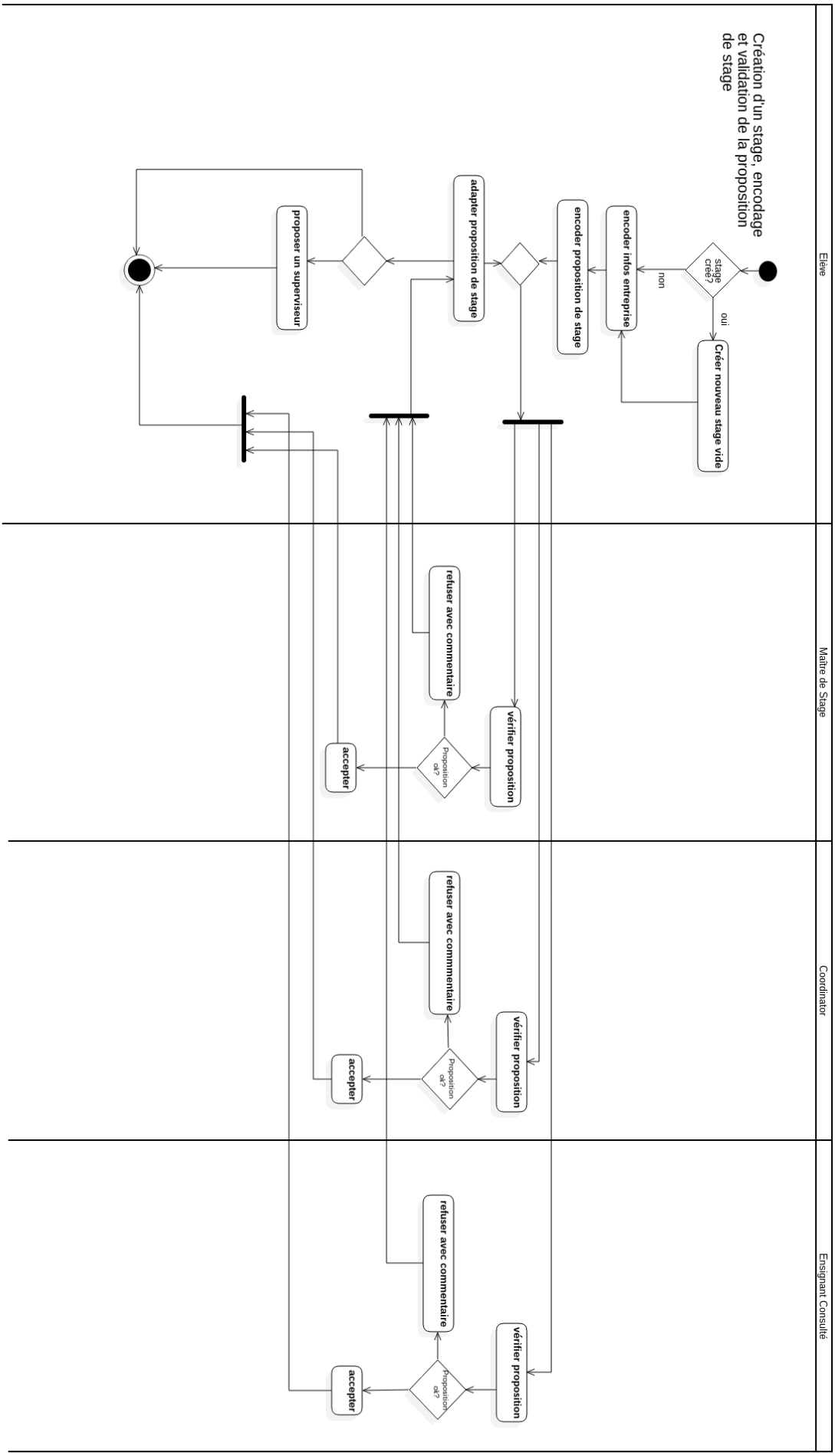


Illustration 13: Diagrame d'activités du mécanisme de création d'un stage

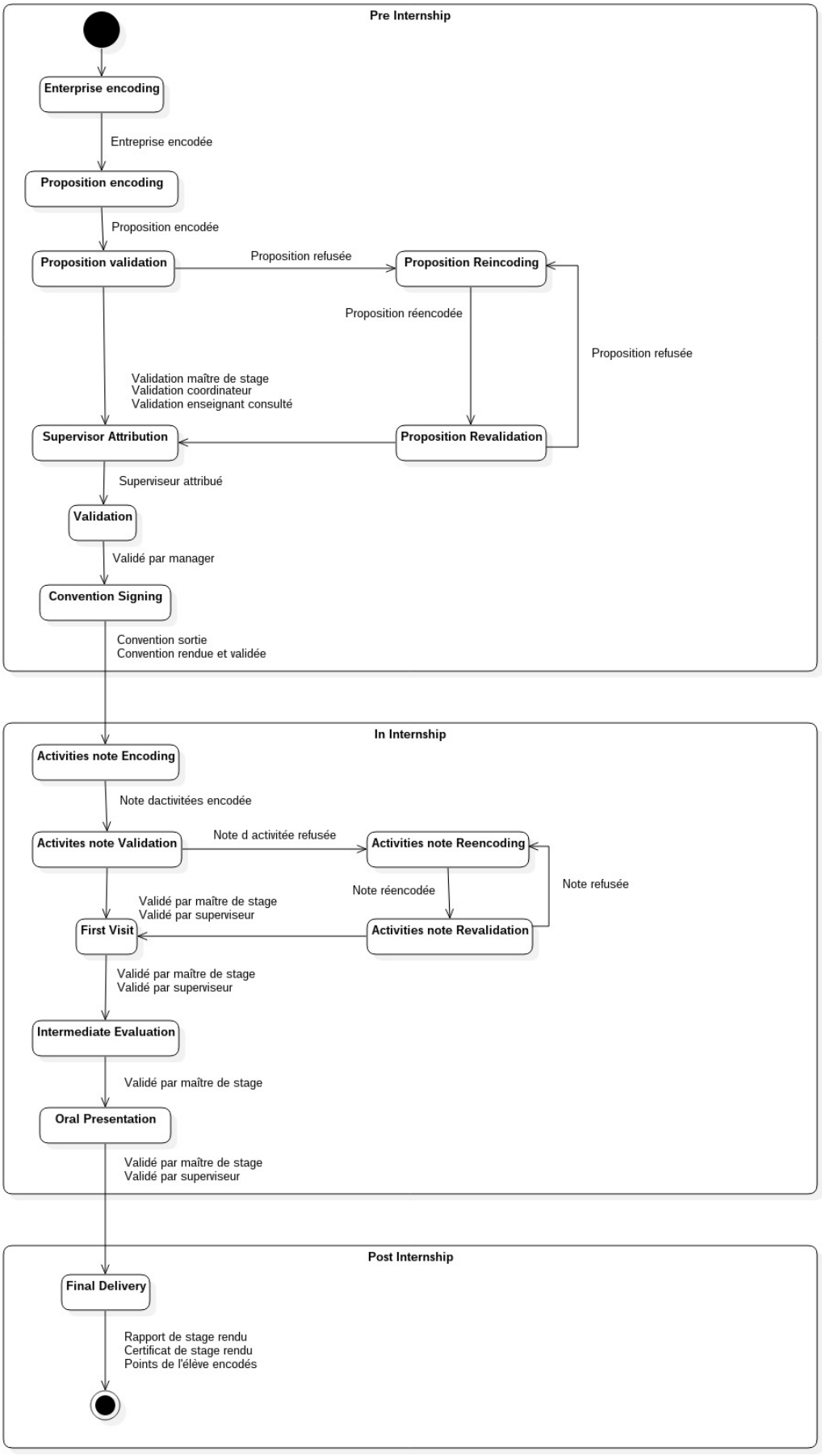


Illustration 14: Diagramme d'activités du déroulement d'un stage

4. Attribution d’un superviseur

L’illustration 15 reprend de manière graphique le processus d’attribution d’un superviseur. Si l’élève a fait une demande pour avoir un superviseur en particulier, l’enseignant concerné verra le stage de l’élève apparaître dans une liste avec les demandes de supervisorat. Il peut alors prendre la décision d’accepter de devenir superviseur, ou bien de refuser. En cas de refus, il peut indiquer la raison de son choix. En cas de refus ou d’acceptation, l’élève est tenu dans l’ombre jusqu’à la validation finale du stage.

Intervient alors le validateur. Il a une liste de tous les stages, et peut appliquer un filtre afin de modifier un ou plusieurs stages afin de leur attribuer/changer le superviseur. Pour chaque stage de la liste, il peut voir si l’élève a proposé ou non un superviseur, et si l’enseignant proposé a accepté ou refusé cette proposition. Dans les deux cas, il peut assigner le superviseur de plusieurs stages en même temps pour ensuite sauvegarder l’ensemble des stages d’un coup.

Le validateur a l’occasion de revenir sur sa décision et peut changer le superviseur d’un ou plusieurs stages, mais uniquement si le choix de ceux-ci n’a pas été verrouillé par le manager. Une fois la validation des superviseurs faite par le manager, il n’est plus possible de changer de superviseur.

5. Validation par le validator

Avant de lâcher un élève en stage, le validator désigné par la section doit donner son accord et dire « *je suis au courant que j’ai un élève en stage* ». Lorsqu’un stage est arrivé à ce stade, un filtre *prêt pour validation* supplémentaire apparaît. Une fois que le validator a donné son accord, on peut procéder avec le dernier contrôle du manager, suivi de la convention de stage.

6. Contrôle et attribution finale des superviseurs par le manager

Le manager dispose d’un tableau de bord avec tous les stages. Pour faciliter la visibilité et prise d’action, il dispose également de filtres qui permettent de voir directement les stages en fonction des différentes étapes, mais aussi d’agir dessus. Choisir, par exemple, le filtre *superviseurs* permet d’agir sur les superviseurs de tous les stages. Le manager peut donc voir si tous les stages ont un superviseur, et quel superviseur a été attribué à quel stage.

Le manager joue le rôle de « *filet de sécurité* ». S’il s’aperçoit qu’un stage n’a toujours pas de superviseur, il peut en attribuer un de force. De même, il peut changer le superviseur d’un stage comme il veut, c’est lui qui a le dernier mot.

Une fois qu’il est d’accord avec l’attribution des superviseurs, il peut valider cette attribution pour l’ensemble des stages d’un coup. À partir de ce moment, il n’est plus possible de changer de superviseur, ni par le manager, ni par le validator. Seul l’administrateur détient les droits nécessaires pour encore faire une modification.

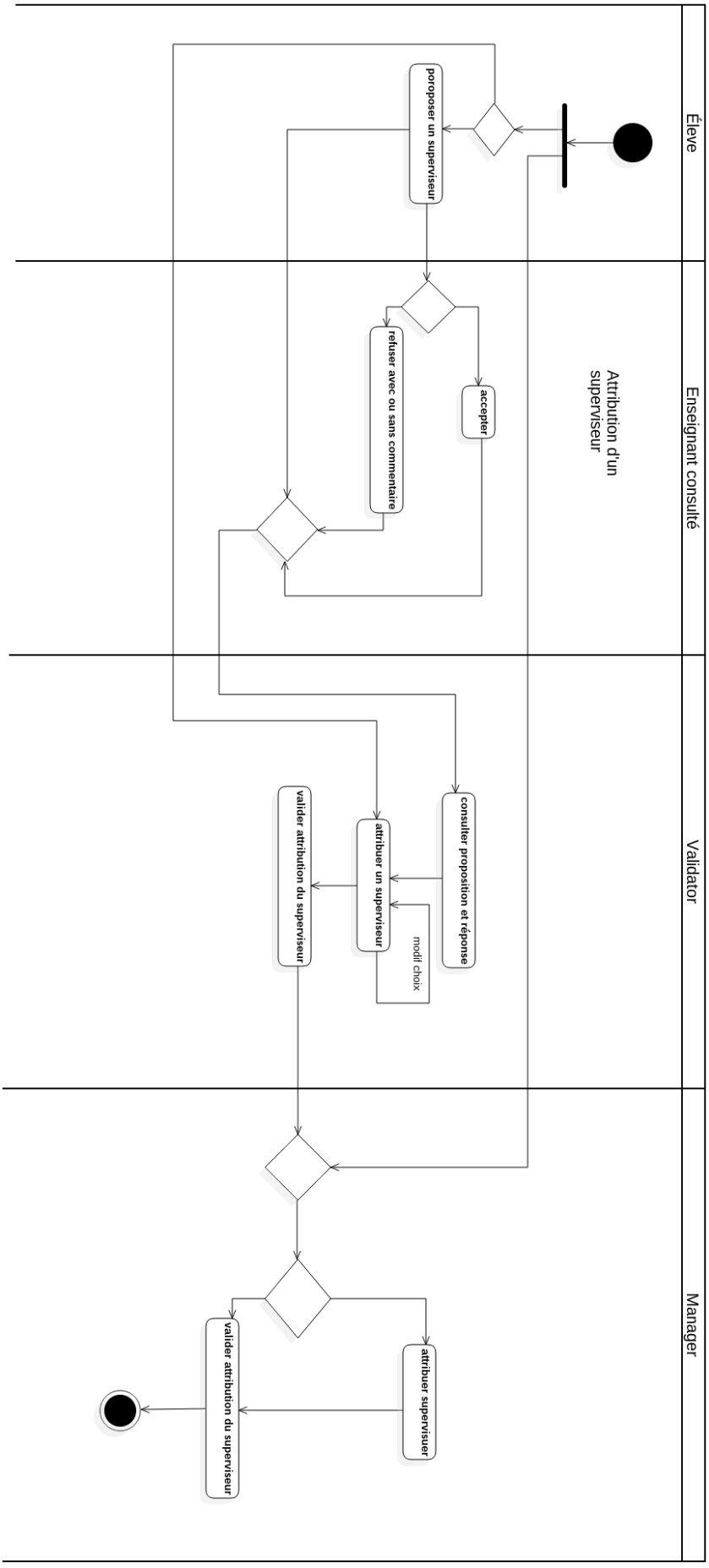


Illustration 15: Diagramme d'activités pour l'attribution d'un superviseur

7. La convention de stage

Si tout est en ordre, Bernadette peut sortir la convention de stage. Dans le tableau de bord du manager, sélectionner *convention* permet d'agir sur le statut de la convention de stage. Si elle a été donnée, le manager peut l'indiquer. Une fois que la convention a été rendue, une opération similaire permet de le valider. Comme la convention fixe la date de début et fin du stage, c'est également sur ce panel que le manager peut encoder les dates de début et de fin pour tous les stages. L'élève est tenu au courant de l'avancée de ces étapes sur son tableau de bord personnel.

8. Note d'activités

D'une façon similaire à ce qui a été fait pour la proposition de stage, l'élève peut ensuite encoder sa note d'activités. Il y rajoute des objectifs généraux avec des sous-objectifs spécifiques. Une fois que la note d'activités a été encodée, elle doit alors être validée par le maître de stage ainsi que par le superviseur de stage.

Le maître de stage et superviseur de stage peuvent tous deux choisir d'accepter ou de refuser la note d'activités du stagiaire. En cas de non validation, ils peuvent indiquer en commentaire les motifs du refus. Tant que les deux acteurs n'ont pas répondu, l'élève n'a plus accès à sa note d'activité. Si l'un des deux a refusé la note, la version précédente est archivée, et l'élève peut apporter des modifications à sa note d'activités en tenant compte des commentaires faits. Après une nouvelle soumission, le mécanisme se répète alors jusqu'a ce que les trois parties soient d'accord.

9. Première visite

L'illustration 16 reprend sous la forme d'un diagramme d'activités le déroulement de la première visite en entreprise. Une fois que la note d'activités a été encodée, l'élève doit organiser la première visite du superviseur dans l'entreprise ou se déroule le stage. Pour cela, l'élève accède au formulaire correspondant, et encode ainsi le lieu et la date du rendez-vous.

Pour cette étape, nous avons choisi de ne pas informatiser la recherche d'une date et lieu. Un des objectifs du stage et de pouvoir organiser un rendez-vous en prenant contact avec les parties prenantes. Dès lors, un outil destiné à se mettre d'accord sur une date retirerait une composante du stage. L'élève doit donc trouver une date et un lieu qui convient au maître de stage et superviseur, et peut ensuite les encoder.

Pour la première visite en entreprise, le maître de stage dispose de deux options supplémentaires sur son tableau de bord. Tout d'abord, il peut ajouter des notes prises pendant ou après la présentation. La deuxième action est de valider la première visite. Tant que la visite n'a pas été validée par le maître de stage, il peut toujours éditer et rajouter des notes à sa guise.

Pour le superviseur de stage, le mécanisme est le même. Il peut, tout comme le maître de stage, ajouter des notes prises pendant ou après la visite en entreprise, et puis valider le tout. Une fois que la visite a été validée, le superviseur ne sait plus rajouter ou éditer de notes, elles sont alors figées.

Ce n'est que lorsque le maître de stage et superviseur ont validé la visite en entreprise que l'élève peut encoder la suite du stage, à savoir l'évaluation intermédiaire.

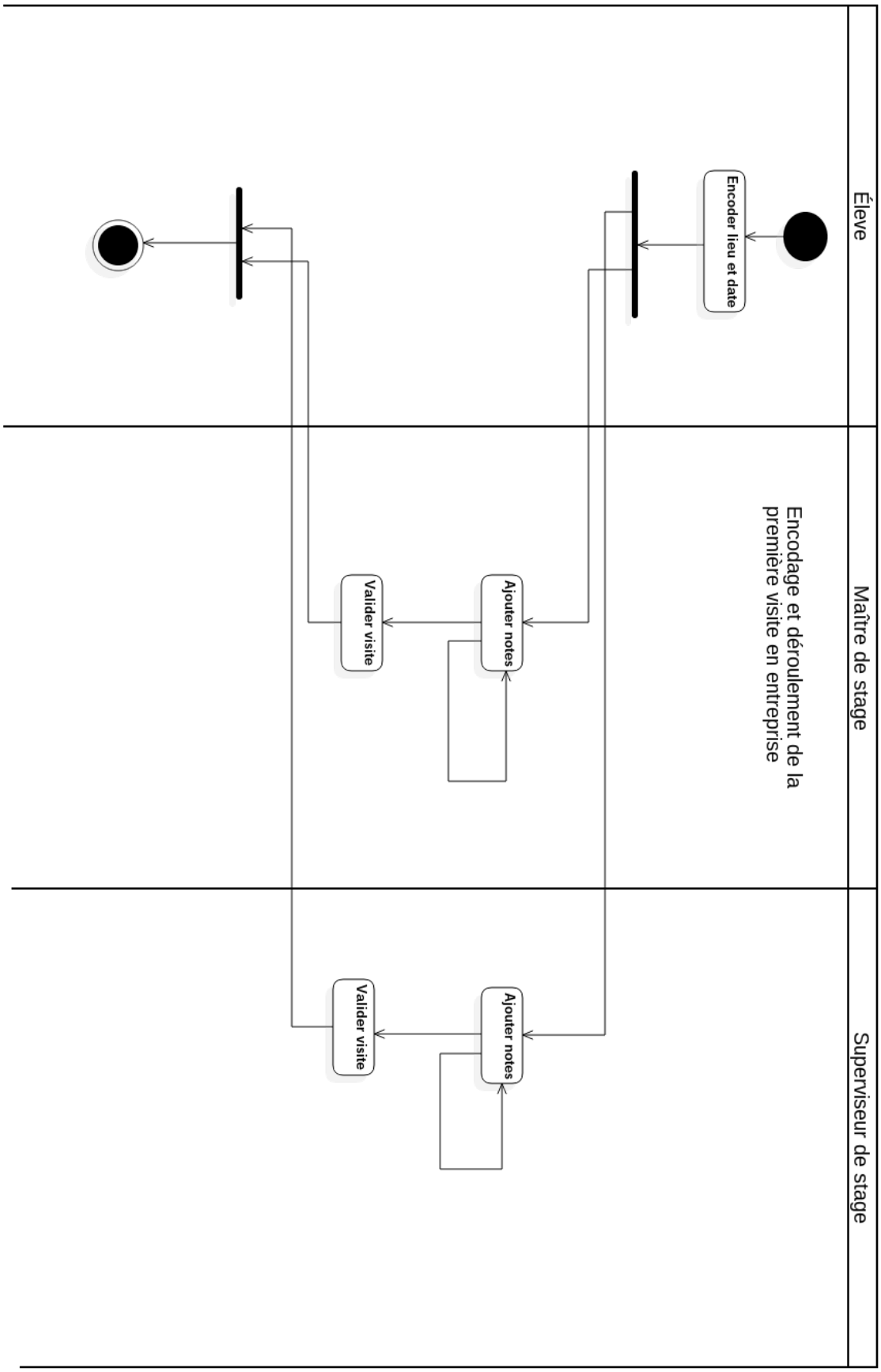


Illustration 16: Déroulement de la première visite en entreprise du superviseur

10. Evaluation intermédiaire

Le procédé pour l'évaluation intermédiaire est assez semblable que pour la première visite, mais n'implique que l'élève et le maître de stage. L'élève doit tout d'abord convenir d'une date et d'un lieu pour effectuer l'évaluation intermédiaire. Une fois que ces informations sont encodées, le maître de stage peut accéder à l'onglet dédié à l'évaluation intermédiaire.

Le maître de stage peut ajouter des notes prises durant ou après l'évaluation intermédiaire, et peut alors les éditer jusqu'à ce qu'il valide tout. Dans la suite de développement du projet OpenSM, le maître de stage aura accès à une grille d'évaluation en ligne qu'il pourra compléter.

11. Défence Orale

Pour la défence orale, le procédé est tout à fait similaire à ce qui se fait pour la première visite en entreprise. L'élève doit trouver une date et lieu qui arrange maître de stage et superviseur, et les deux peuvent alors ajouter des notes avant de valider la présentation.

Comme expliqué plus haut, OpenSM fonctionne sur base de modules. La mise à disposition de grilles d'évaluation n'a pas encore été ajoutée car elle fera justement l'objet d'un module supplémentaire développé ultérieurement. Un projet futur aura pour but de rajouter le module « Grilles d'évaluation » à la plateforme. Les enseignants et membres de l'administration pourront alors créer et modifier des grilles d'évaluation pour leurs cours, examens, travaux, et aussi les stages des élèves. Une fois ce module créé, il suffira d'injecter le module dans les stages pour bénéficier de ses services.

12. Rendu des délivrables

Une fois que la défense orale a été présentée, l'élève doit rendre son rapport de stage ainsi que son certificat de stage. Sur la plateforme, il ne doit pas effectuer d'action, mais peut voir sur son tableau de bord la confirmation que ces documents ont bel et bien été rendus et validés.

Il est du rôle du manager de valider que le certificat de stage et rapport de stage ont bien été rendus. Dans son tableau de bord, l'onglet *délivrables* permet d'afficher une liste des tous les stages en regroupant ces deux informations. Il lui suffit alors de cliquer sur un bouton pour valider que le certificat du stage concerné a bien été reçu, de même pour le rapport de stage.

13. Journal de bord

Durant toute la durée du stage, l'élève doit tenir à jour un journal de bord. Pour cela, un onglet *journal de bord* est mis à sa disposition. Chaque jour, l'élève peut créer une nouvelle entrée dans son journal de bord, et la date d'ajout est générée automatiquement. L'élève ne peut cependant créer qu'une seule entrée par jour. S'il essaye d'en créer deux le même jour, on charge alors la plus récente pour pouvoir la modifier.

3.5.1 Machine à états

Pour s'y retrouver au sein de toutes les étapes et conditions d'avancement d'un stage, la progression et passage d'état en état d'un stage s'est faite au moyen d'une logique très proche de celle d'une machine à états. Elle entre en jeux dès qu'il y a une modification du stage devant être persistée en base de données, lorsque un enseignant valide la proposition de stage, par exemple. A chacune de ces actions, le stage passe alors par notre machine à états.

À la suite d'une modification, et juste avant de persister le stage dans la base de données, on vérifie l'état actuel du stage. En fonction de ce dernier, des conditions peuvent sont vérifiées, faisant avancer l'état du stage vers un autre état ultérieur. Le stage est ensuite- sauvegardé avec son état en base de données.

Pourquoi l'état actuel du stage fait-il partie intégrante du document en base de données ? Deux mécanismes étaient envisageables. Le premier est celui que nous venons de décrire, ou l'état est mémorisé, et on se base dessus pour avancer de manière incrémentale. Une autre méthode aurait été de ne pas retenir l'état du stage et de le déterminer à la volée. Cela rend la manipulation du stage plus facile. On se base sur le document en entier, et à chaque chargement, on regarde l'ensemble des conditions remplie et non remplies, et on en déduit un état général. Cette deuxième méthode a pour avantage d'être plus souple, on n'est pas obligé de suivre un ordre chronologie fixé. Dans le cas de la première solution, une fois un état franchi, il n'est pas possible de revenir en arrière. Si on souhaite le faire, l'administrateur doit modifier les données et réencoder l'état correspondant. Avec un calcul fait sur le volet, l'état actuel ne dépenderait pas de l'état précédent.

La première solution a été retenue car elle est beaucoup plus facile à mettre en place, et surtout beaucoup plus facile a maintenir. Lors du développement, on s'est aperçu qu'un énorme tableau de conditions et de conditions imbriquées serait très difficile a gérer et lourd à calculer. Chaque rajout d'une condition supplémentaire nécessiterait d'ailleurs de devoir adapter plusieurs autres, ce qui augmenterait très rapidement la complexité du code.

Avec une solution basée sur un état déjà mémorisé, il n'est plus nécessaire d'effectuer un calcul total, d'appliquer toutes nos conditions pour trouver l'état final. Pour chaque état, il y a un set réduit de conditions à remplir pour passer au suivant.

4 L’interface utilisateur

Les actions et besoins sont différents d’un rôle à l’autre au sein de l’application. Pour cette raison, plusieurs interfaces et dashboard utilisateur ont été créés. Toutefois, ils partagent tous le même layout racine. Chaque utilisateur a le même écran de login, ainsi que les mêmes header. En fonction du rôle de chacun, des menus et options supplémentaires sont alors rajoutés. Les captures d’écran des interfaces utilisateur sont reprises dans les annexes page 59.

4.1 Interface de l’élève

Lorsque l’élève se connecte à la plateforme, il est d’abord dirigé vers le dashboard généraliste. Un sous menu de gestion de ses stages est disponible dans le menu déroulant principale. La première page qui lui est présentée est alors une liste avec l’ensemble de ses stages s’ils ont déjà été créés. S’il n’y a pas encore de stage, il peut cliquer sur « *ajouter un stage* » pour peupler la liste.

Cliquer sur un stage l’amène ensuite au dashboard de ce stage en particulier. Le stage est divisé en plusieurs onglets représentant les différents milestones du stage. L’onglet sélectionné par défaut est l’onglet Overview, qui regroupe les informations généralistes à propos du stage. Il y voit un graphique reprendant son parcours. Neuf bulles représentent les 9 grandes étapes du déroulement du stage. Elles sont grisées à l’exception de la bulle active qui est alors colorée. L’élève bénéficie ainsi d’une représentation visuelle de où il en est dans son parcours.

En plus d’une vue graphique, l’overview offre l’essentiel des informations, un récapitulatif de l’état du stage à l’étudiant :

- Dans quel état général se situe mon stage ? Pré-internship, Internship , Ou post-internship.
- Dans quelle sous-étape du stage je me situe précisément ? Chaque grande étape du stage est divisée en sous-étapes plus précises et locales. Par exemple, dans l’étape *proposition* du stage, l’élève pourra savoir s’il est dans la sous-étape *encodage*, *validation*, ou bien même *réencodage* en cas de refus.
- Des informations supplémentaires sur l’avancement de l’étape. Par exemple, si l’encodage de l’entreprise n’a pas encore été fait, le champ info contiendra le texte suivant : *« les informations de l'enterprise ne sont pas complètes, merci de les encoder. »*

Parmi les autres informations disponibles dans l’overview, on retrouve les participants au stage, ainsi que des valves génériques. Ces valves permettront d’afficher des rappels, notifications importantes, deadlines, ou tout autre information jugée utile et pertinente.

Les huit onglets restants sont les suivants :

- Onglet Entreprise

Pour encoder les informations de l'entreprise, l'élève doit se rendre sur l'onglet *entreprise*. Si les informations n'ont pas encore été encodées, une petite notification ainsi qu'un bouton d'accès au formulaire s'offrent à l'utilisateur. Une fois dans le formulaire, l'élève peut encoder toutes les informations concernant l'entreprise.

- Onglet Proposition

L'onglet proposition permet d'accéder au formulaire d'encodage de la proposition de stage. L'élève n'aura cependant accès au formulaire d'une étape QUE si l'étape précédente à été complétée. Ici, l'élève pourra accéder au formulaire de la proposition parce qu'il a complété l'encodage de l'entreprise.

Le formulaire de la proposition est assez simple. L'élève doit encoder l'ensemble des informations génériques du stage, puis encoder son maître de stage et enseignant consulté pour le sujet. Pour les deux derniers, l'auto-complétion des deux champs s'assure que l'élève ne sait encoder que des utilisateurs valides, à savoir des enseignants ou maîtres de stage faisant partie de la base de données. En cas d'erreur ou de champ non complété, le formulaire n'est pas envoyé, et le champ problématique est alors surligné de rouge.

Lorsque la proposition de stage est encodée, les données encodées sont affichées dans l'onglet *Proposition*. L'élève peut alors suivre l'évolution de la validation de sa proposition. Tout d'abord, il peut voir si le maître de stage, l'enseignant consulté et le coordinateur ont donné leur réponse. Il peut ensuite voir si elle est positive ou négative, ainsi que les commentaires associés aux réponses négatives. Une fois que les trois acteurs ont répondu, et en cas de réponse négative, un bouton est alors visible et permet à l'élève d'accéder à nouveau au formulaire de sa proposition.

- Onglet Superviseur

L'onglet superviseur permet à l'élève de faire connaître ses préférences dans le choix d'attribution de son superviseur. L'auto-complétion du champ permet de trouver plus rapidement l'enseignant, et assure aussi que l'élève ne choisi qu'un enseignant valide. Il était souhaité que l'élève ne connaisse pas la réponse de l'enseignant à cette requête. Les informations ne sont modifiées qu'une fois le superviseur attribué, qu'il soit celui souhaité par l'étudiant ou non.

- Onglet Notes d'activités

Tout comme pour la proposition de stage, l'onglet Note d'Activités permet à l'étudiant d'accéder au formulaire de la note d'activités. Dans le formulaire, l'étudiant peut étendre le nombre de champs pour les objectifs généraux, et ensuite rajouter autant de champs pour les objectifs spécifiques qu'il le souhaite. Le mécanisme de validation est tout à fait similaire à ce qui se fait pour la proposition de stage et a lieu entre le maître de stage et le superviseur.

- Onglets First meeting, intermediate evaluation, oral presentation

Ces trois onglets sont très similaires, car le procédé pour ces trois étapes sont presque identiques. Pour chaque onglet, l'élève peut sélectionner une date dans un calendrier interactif, et encode aussi le lieu de déroulement du meeting. Une fois les informations encodées, un récapitulatif de ces informations est affiché dans chaque onglet.

- Onglet Journal de bord

Comme son nom l'indique, cet onglet permet à l'élève d'accéder à son journal de bord. On lui présente d'abord l'historique, la liste de toutes ses entrées passées. Il peut alors cliquer sur *create Entry* pour rajouter une entrée au journal. L'élève ne pas pas créer deux entrées le même jour. S'il essaye, il sera redirigé vers l'entrée la plus récente qu'il pourra ainsi éditer.

On constatera qu'à chaque fois que l'étudiant encode une étape supplémentaire de son stage, les informations affichées dans l'overview sont mises à jour.

4.2 Interface du coordinateur

Le coordinateur retrouve dans son menu l'option *internships* pour gérer les stages en temps que coordinateur. Il dispose tout d'abord d'une liste avec tous les stages. Elle est pour l'instant générique, mais sera adaptée par la suite pour ne reprendre que les stages gérés par le coordinateur. Cette liste présente en un coup d'oeil les informations principales sur les stages, à savoir l'élève, l'entreprise, le maître de stage, le superviseur, et l'enseignant consulté pour le sujet.

Parallèlement sur la même page, une autre liste est mise à disposition du coordinateur. Celle-ci reprend uniquement les stages pour lesquels le coordinateur doit donner son accord pour la proposition de stage. Si le coordinateur a donné son accord pour tous les stages, la liste est alors vide et disparaît. De même, un stage ne fera pas partie de la liste tant qu'il n'a pas atteint l'étape de validation de la proposition de stage.

Le coordinateur peut cliquer sur un stage de la liste, ce qui lui permet d'accéder aux informations de celle-ci. Les informations sont limitées aux onglets *Overview, Participants, Entreprise* et *Proposition*. Dans l'onglet proposition, le coordinateur peut lire la proposition de stage, et ensuite décider de l'accepter ou de la refuser. S'il décide de cliquer sur refuser, il est tenu de remplir le champ juste en dessous expliquant les raisons de son choix. Il peut également voir la décision et commentaires éventuels de l'enseignant consulté ainsi que du maître de stage.

4.3 Interface du maître de stage

Contrairement aux autres rôles, le maître de stage dispose tout d'abord d'une liste des étudiants pour lesquels il est maître de stage. Il accède à cette liste via le menu déroulant et en cliquant sur *manage → students*. Lorsqu'il clique sur un étudiant, il est redirigé vers une deuxième liste. Celle-ci liste reprend le ou les stages de l'étudiant dont il est maître. Lorsqu'il clique sur un stage, il est alors redirigé vers la page de ce stage. Sur la page du stage, il y retrouve plusieurs onglets pour gérer ce stage.

- Onglet Overview

Cet onglet reprend les informations de base sur le stage. On y retrouve le nom des participants, ainsi que les dates de début et de fin pour la période de stage.

- Onglet Proposition

Cet onglet permet au maître de stage de consulter la proposition de stage encodée par l'élève. Si elle n'a pas encore été complétée, la page indique qu'il n'y a pas de proposition, et le restant de la page est alors caché.

Si l'élève a rempli sa proposition, le maître de stage peut la lire, et peut indiquer s'il l'accepte ou non au moyen d'un bouton *accepter* ou *refuser*. Une fois qu'il a fait son choix, les boutons sont alors cachés.

- Onglet Notes d'activités

Cet onglet fonctionne de la même façon que pour la proposition. Le maître de stage a accès à la note d'activités si elle a été encodée, et deux boutons permettent de l'accepter ou de la refuser. S'il n'y a pas de notes d'activités, la page reste cachée, et de même si le maître de stage a fait un choix alors les boutons sont eux aussi cachés.

- Onglet First Meeting

Cet onglet, qui permet de gérer la première visite en entreprise, cache initialement son contenu tant que l'élève n'a pas encore encodé de lieu et de date pour la visite.

Lorsque ces informations sont encodées, le maître de stage dispose de deux boutons. Le premier le redirige vers une page qui lui permet de rajouter ou de retirer des notes personnelles, de les éditer puis de les sauvegarder. Le deuxième bouton permet de valider la visite en entreprise. S'il clique dessus, on lui demande confirmation, car une fois l'évaluation validée les deux boutons disparaissent, et le maître de stage ne sait plus modifier ses notes.

- Les onglets évaluation intermédiaire et défense Orale

Ces deux onglets permettent de gérer respectivement l'évaluation intermédiaire et la défense orale. Ils fonctionnent tous deux exactement de la même manière que l'onglet *First Meeting*. Un bouton permet de rajouter et d'éditer des notes tandis que l'autre permet de valider le bon déroulement de l'évaluation.

4.4 Interface de l’enseignant / superviseur

L’enseignant peut accéder à sa liste personnelle de stages en cliquant sur *my internships* dans le menu déroulant. Sur cette page, on lui présente alors trois listes.

La première liste est peuplée avec les stages pour lesquels on a demandé l’accord de l’enseignant pour le sujet dans la proposition de stage. L’enseignant peut cliquer sur le stage, et le procédé est alors le même que pour le coordinateur. Si la liste de ces stages ayant besoin d’un accord de l’enseignant est vide, ou que l’enseignant a déjà donné une réponse à chaque stage, elle disparaît.

La deuxième liste est celle des demandes de supervisorats. Si un élève souhaite que l’enseignant devienne superviseur, alors son stage apparaît dans cette liste. En cliquant dessus, l’enseignant peut alors accéder à l’onglet *supervisor*. Là, il peut indiquer s’il souhaite ou non devenir superviseur. Si la réponse est négative, il doit indiquer les motifs de son choix. Quel que soit sa décision, l’élève ne verra rien avant l’attribution finale du superviseur. Pareillement, si cette liste est vide, elle disparaît.

La troisième liste est celle des stages pour lesquels l’enseignant est effectivement superviseur de stage. Pour ces stages, le superviseur à accès à plusieurs Onglets : *Overview, Proposition, Enterprise, Activities Note, First Meeting, Oral Presentation* et *Journal de bord.* Les trois premiers onglets ainsi que l’onglet *journal de bord* sont des onglets d’information, ou il n’y a donc pas d’action à prendre. Les onglets concernant la note d’activités, première visite en entreprise et défense orale sont fort similaires.

Tout d’abord, le superviseur peut donner son accord ou non pour la note d’activités. En cas de refus, il peut remplir un champ indiquant les raisons de son choix. Cette procédure est tout à fait identique pour le maître de stage.

Pour la visite en entreprise et la défense orale, le superviseur à l’occasion de laisser des notes et de valider ces deux étapes. On retrouve ici ce qui a été décrit plus haut, et cette démarche est à nouveau tout à fait identique pour le maître de stage.

Dernièrement, l’onglet Journal de bord permet au superviseur de stage de consulter le journal tenu à jour par son élève.

4.5 Inteface administrateur

L'interface de l'administrateur est beaucoup plus simpliste que celles des autres acteurs. Son but est avant tout fonctionnel, et ne devrait à priori pas servir si le programme fonctionne correctement.

L'administrateur accède à la gestion des stages via le menu déroulant, en cliquant sur *Manage → Internships*. Il dispose alors d'une liste avec tous les stages des la plateforme. Cette liste affiche pour chaque stage le matricule de l'élève, son nom, son prénom ainsi que l'entreprise dans laquelle se déroule le stage. S'il le souhaite, l'administrateur peut cliquer sur *Internship Create*, ce qui crée un stage vide, ou alors cliquer directement sur un stage. Dans les deux cas, il est redirigé vers la page de gestion du stage.

Sur cette page, l'administrateur a accès à toutes les informations du stage, qui lui sont présentées sous la forme d'une grande fiche. Il peut ensuite effectuer deux actions : cliquer sur l'icône poubelle pour supprimer le stage, ou cliquer sur l'icône crayon pour le modifier. S'il souhaite modifier le stage, il est alors redirigé vers un grand formulaire. Ce dernier permet de modifier n'importe quel élément du stage, ou même changer le statut général du stage sans passer par la machine à états. Pour sauvegarder les modifications apportées au stage, il lui suffit de cliquer sur le bouton *Update* en haut de page.

4.6 Interface du Validateur

Dans l'interface du validateur, on retrouve une liste avec tous les stages, ainsi qu'une série de trois filtres :

- Le premier filtre est le filtre *view all*. Comme son nom l'indique, il permet de voir la liste de l'ensemble des stages, en mettant en avant les informations principales de ceux-ci.
- Le deuxième est le filtre *superviseurs*. Un des rôles du validateur est de valider l'attribution des superviseurs au sein de sa section. Cette liste contient donc l'ensemble des stages pour lesquels un superviseur n'a pas encore été attribué. Dans la liste, le validateur peut voir pour chaque stage si l'étudiant a fait une demande de superviseur, et quelle a été la réponse de l'enseignant en question. Pour chaque stage, il peut indiquer quel enseignant sera attribué en tant que superviseur. Une fois qu'il a fini de répartir les enseignants, au moment de sauvegarder, tous les stages dont la checkbox est tiquée seront enregistrés.
- Le troisième filtre est celui des stages prêts à être validés. Pour donner son accord au départ d'élèves en stage, le validateur peut simplement cocher les stages pour lesquels il donne son accord, puis valider.

Pour chaque filtre, si aucun stage n'est présent dans la liste, elle est alors retirée de l'interface, et le filtre n'apparaît même plus, permettant de simplifier l'utilisation de la plateforme.

4.7 Interface du manager

Le manager dispose d'une liste avec tous les stages de la haute école, ainsi que d'une série de filtres pour gérer leur affichage. Aux choix, le manager peut appliquer les filtres *overview, enterprise, proposition, supervisor, convention, delivrables* pour structurer l'information de la liste des stages :

- Le filtre *overview* permet d'afficher uniquement les informations essentielles. On y retrouve l'étudiant, son matricule, nom, prénom, ainsi que l'état dans lequel se situe son stage.
- Le filtre *enterprise* permet de voir si pour chaque stage l'entreprise a bel et bien été encodée. Il permet aussi de voir en un coup d'oeil quel maître de stage est associé à quel stage.
- Le filtre proposition offre un aperçu sur l'avancée de la validation de la proposition de stage. Le manager peut voir pour chaque stage si le maître de stage, l'enseignant consulté et le coordinateur ont répondu à la proposition de stage, et quel a été leur choix par rapport à celle-ci.
- Le filtre *supervisor* permet au manager de voir en un coup d'oeil quels enseignants ont étés attribués en tant que superviseur, et si un stage n'a pas encore de superviseur. On peut également voir si le coordinateur d'année a validé l'attribution du superviseur pour un stage ou non. Le manager a un rôle de « *filet de sécurité »* et a ainsi le dernier mot en ce qui concerne l'attribution des superviseurs. Pour chaque stage, il peut choisir de changer l'enseignant attribué, ou alors en attribuer un si le stage n'a pas encore de superviseur. Une fois qu'il est content avec ses modifications, le manager valide l'ensemble des stages en un seul coup. Cette action n'est pas réversible, et fixe définitivement l'attribution des superviseurs.
- Le filtre *convention* permet de valider la convention de stage. Le manager peut, pour chaque stage, marquer qu'elle a été délivrée, et si elle a été rendue et validée. Comme la convention de stage fixe le début et la fin de la période de stage, c'est également ici que le manager fixe les dates de début et de fin de chaque stage. Cela se fait au moyen d'un calendrier interactif, et une fois les dates entrées, le superviseur peut sauvegarder l'ensemble de ses modifications.
- Le filtre *délivrables* est assez similaire au filtre *convention*. Il permet tout d'abord au manager de valider que le certificat de stage a bien été rendu et est valide. Pareillement, il permet ensuite de valider que le rapport écrit a également été rendu.

5 Tests de l’application

Comment pouvons nous vérifier que notre programme fonctionne correctement ? Nous avons vu à travers la machine à états que certaines actions ne peuvent être effectuées qu’à certains moments par certains acteurs. Par exemple, une fois que l’élève a encodé la note d’activités, il ne peut plus y accéder tant que le superviseur et le maître de stage n’ont pas fait connaître leur décision. Comment pouvons nous vérifier que cette condition est belle et bien respectée ?

5.1 Automatisation des tests

Une méthode consisterait à répliquer le scénario nous-même. On se connecte avec les trois acteurs concernés, et on remplit un nouveau stage. Cette méthode fonctionne, certes, mais présente des désavantages évidents :

- Elle est lente et demande beaucoup trop de ressources. Il est tout d’abord nécessaire de répliquer le parcours d’un stage depuis son début, ce qui est lent. Ensuite, il est nécessaire de se connecter au moyen de plusieurs comptes différents, ce qui là encore est lent, et franchement assez pénible.
- Elle n’est pas viable. Si l’on doit répéter ces opérations à chaque fois que l’on souhaite faire un test après une légère modification, cela entraînerait une grande perte de temps qui ne serait tout simplement pas envisageable sur le long terme.
- Elle ne garantit que le bon fonctionnement des vues du frontend. Nous savons qu’AngularJS permet, entre autres, de masquer des éléments graphiques en fonction du contexte. On peut ainsi cacher du contenu, des boutons, et donc priver l’utilisateur de certaines fonctionnalités à des moments souhaités, sur l’interface. Mais un utilisateur pourrait-il accéder à une ressource privée via URL par exemple ? Ou bien alors pourrait-il modifier son stage en forgeant une requête HTTP et en l’envoyant directement sur l’API ? Qu’en serait-il alors des règles établies ? Un élève pourrait-il modifier son stage à un moment ou il n’est pas censé le faire s’il contourne l’interface utilisateur ?

On se rend vite compte que cette méthode de test ne permet pas de vérifier cela. Il n’est pas suffisant de se connecter en temps qu’utilisateur et de vérifier que tout fonctionne, il est alors nécessaire d’utiliser un outil de test adapté.

Les tests réalisés reposent sur un t*est runner* appelé *Karma*. Un tel dispositif crée un serveur factice, puis exécute des tests dans un ou plusieurs navigateurs en utilisant les données dérivées de ce serveur. Cependant, *Karma* n’est « que » un t*est runner*, et a besoin d’un ou plusieurs frameworks auxquels il peut se connecter pour faire tourner les tests. Les frameworks sont sur un niveau d’abstraction inférieur au *test runner*, qui est le niveau d’abstraction le plus haut. C’est dans les frameworks que se situe le code décrivant les tests proprement dits. Les deux frameworks employés au sein de la stack MEAN sont *Jasmine* et *Mocha.* Ils décrivent les tests à exécuter, respectivement côté client et serveur.

Nous avons vu que la stack MEAN est structurée en modules. Heureusement pour nous, l'ajout des tests ne fait pas exception à la règle et respecte cette même structure. À chaque module s'ajoute alors un dossier *tests*, subdivisé à son tour en tests côté serveur puis côté client, comme le témoigne l'illustration 17.

Illustration 17: Structure des dossiers test

L'automatisation des tests permet finalement de tendre vers une méthode de développement avec intégration continue. La version *prod* du projet, c'est à dire celle qui est mise en production, est d'ailleurs couplée à la plateforme *Travis*. Celle-ci assure un service de tests et d'intégration continue pour un projet en cours de développement. Dès qu'un développeur *push* une nouvelle version du projet, *Travis* se charge de le tester automatiquement.

5.2 Que tester?

Tester un programme, c'est une partie du développement d'un projet qui demande une attention particulière. Cela demande beaucoup de temps, d'étude et d'investissement. On dit d'ailleurs parfois des tests qu'ils constituent presque un « mini projet » à part entière. Dans des projets de grande ampleur, des équipes entières peuvent être attribuées à cette tâche.

Dans le projet OpenSM, il n'a évidemment pas été possible de couvrir le programme dans son entièreté. Néanmoins, l'ensemble des tests réalisés s'inscrivent dans une démarche générale visant à répondre à certains objectifs principaux :

1. S'assurer que l'implémentation de notre programme est correcte. Est-ce qu'il se comporte correctement ? Est-ce qu'il fait ce qu'on attend de lui ? Ceci prend toute son importance dans le back-end, où il est impératif de vérifier :
    - La sécurité des données et de l'API. Les accès aux ressources sont-ils bien autorisées ou bloquées en fonction du rôle de l'utilisateur ?
    - L'intégrité des données. Les données respectent-elles le modèle de données imposé ? Des données partagées sont-elles modifiables uniquement par les ayants droit ?
    - Le respect des procédures. Certaines modifications de données ne peuvent être faites qu'à des moments précis. Est-ce que le déroulement chronologique du stage est bien respecté ?
2. Vérifier que des modifications apportées à un module n'ont pas altéré le bon fonctionnement du programme. Si un développeur apporte des changements à un module, il suffit de relancer les tests de ce module pour vérifier qu'il se comporte toujours de la même façon.

5.3 Tests Serveur

Cette section présente les tests du backend, permettant de s'assurer que l'API serveur correspond bien à nos exigeances.

5.3.1 Mocha

Les tests côté serveur sont décrits au moyen du framework *Mocha*. Il s'agit d'un outil puissant qui fonctionne avec NodeJS et qui, comme tout dans la stack MEAN, est également écrit en JavaScript. Si ce framework permet de tester à peu près tout ce qu'on voudrait côté serveur, trois catégories de tests peuvent être distinguées.

- Les tests sur les accès ou *de routage* : on vérifie que pour chaque route, seuls les utilisateurs qui ont les bons rôles peuvent y accéder.
- Les tests de *fonctionnalité* : une route est censée offrir une fonctionnalité précise. Elle doit être invariante et ainsi toujours offrir la même fonctionnalité sans effets de bord.
- Les tests sur les modèles de données : les données sauvegardées en base de données doivent avoir une structure et des typages précis.

5.3.2 Mécanisme et syntaxe des tests

La syntaxe d'un test *Mocha* est simple à comprendre. Il était de la volonté des développeurs de ce framework de rapprocher au plus possible les tests de la langue anglaise. Pour cette raison, la description des tests est intuitive et assez verbeuse.

Un test commence par la description générale de ce que l'on souhaite tester. Ainsi, chaque batterie de tests est contenue dans une instruction générale *describe()*. Cette fonction est ensuite ponctuée d'une ou de plusieurs instructions *it()*. Elles contiennent les tests à proprement parler, décrivant ce que l'on souhaite tester, puis imposant les conditions devant être remplies.

Dans l'exemple de l'illustration 18, nous voulons tester le modèle de données pour un stage. On commence par décrire le test, en lui donnant un nom : « *internship validation test* ». Ensuite, le *it()* décrit en anglais le comportement attendu. Ici, on voudrait qu'il ne soit pas possible de sauvegarder un stage si un des types de données n'est pas respecté. On remplit donc un des champs du stage avec un string au lieu d'une valeur booléenne. Ensuite, on sauvegarde le stage, et on précise les conditions pour que le test soit réussi. On utilise les directives *should* et *exist* pour indiquer qu'on devrait normalement recevoir un message d'erreur.

```
describe('Internship validation test', function () {
  it('should not be able to save an internship with wrong data type', function (done) {
    internship.supervisorApproval = "this field should be boolean and not text";
    internship.save(function (err, savedInternship) {
      should.exist(err);
      return done();
    });
  });
});
```

Illustration 18: exemple de description d'un test Mocha

Lorsqu'on fait tourner les tests, un récapitulatif comme celui de la figure 19 indique si tous les tests sont passés, et lesquels ont éventuellement échoué.

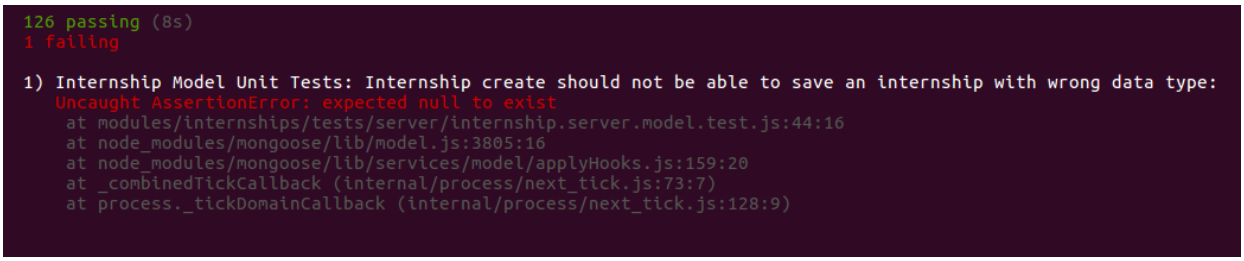


Illustration 19: récapitulatif des résultats d'un test Mocha

Cet exemple n'a pas été choisi au hasard. On voit ici que dans notre test, on s'attendait à recevoir un message d'erreur étant donné que le type de données n'était pas respecté. Au lieu de ça, le test ne passe pas, car il n'y a pas eu de message d'erreur.

Pour corriger ce problème, la première chose à faire était de vérifier que le modèle de données soit correct. Or, après vérification, il s'est avéré que c'était le cas, et que le type de données pour *supervisorApproval* devait bel et bien être booléen ; le problème venait donc d'ailleurs.

```
supervisorStatus: {type: String},
supervisorApproval: {type: Boolean},
coordinatorApproval: {type: Boolean},
```

Illustration 20: extrait du modèle de données d'un stage

Après enquête, il s'est avéré que le problème venait non pas d'OpenSM mais de l'ORM *Mongoose* qui gère la base de données. Il existe un problème de non respect de la validation du type de données dans ce programme et qui est connu de la communauté. C'est un problème face auquel nous ne pouvons malheureusement pas faire grand chose, à part attendre la nouvelle release de *mongoose* qui corrigera ce problème.

Néanmoins, cet exemple est bien choisi, car il permet d'illustrer comment de simples tests peuvent mettre en évidence des problèmes qu'un développeur aurait peut-être raté. En les détectant au plus tôt, on évite ainsi que ces problèmes surgissent dans le futur, évitant ainsi la panne alors que l'application « *fonctionnait très bien ce matin* ».

5.4 Tests Client

La section suivante reprend la description du framework *Jasmine*, destiné aux tests côté client.

5.4.1 Jasmine

Si les tests côté serveur sont décrits au moyen du framework Mocha, ceux côté client sont décrits au moyen du framework *Jasmine*. Ce framework est dit *behaviour driven*, cela signifie que l'ensemble des tests visent à vérifier le comportement de l'application. C'est un framework qui se veut également facile à lire, tout comme *mocha*. Pour cette raison, les tests sont décrits de façon assez similaire à ce qu'on a pu voir précédemment.

5.4.2 Mécanisme et syntaxe des tests

L'exemple repris à l'illustration 21 reprend un test effectué sur le contrôleur du maître de stage. Tout d'abord, le *it()* décrit le test que nous sommes en train d'exécuter. Ensuite, nous utilisons l'outil *httpBackend*, qui simule un backend HTTP et répond aux requêtes de l'application. Cela permet de vérifier que notre application appelle les bonnes adresses avec le bon protocole.

Après avoir appelé une des fonctions du contrôleur, on décrit le comportement que l'on attend. Au moyen de la directive *expect*, on décrit que l'on s'attend à ce que:

- La valeur *oralPresentation.masterValidation* soit passée à *True*
- La fenêtre ait demandé confirmation à l'utilisateur avec un message précis
- À ce qu'une alerte ait également été présentée à l'utilisateur avec un message précis
- À ce que la page ait été rechargée après l'opération.

```
it('should validate validateOralPresentation and call correct adress', inject(function (ArticlesService) {
  $httpBackend.expectGET('/lang/en_GB.json').respond(200);
  $httpBackend.expectPUT('/api/internships/5984721af93c3272f4d2063f/editOralPresentation').respond(200);
  $httpBackend.expectGET('/modules/core/client/views/home.client.view.html').respond(200);

  $scope.vm.validateOralPresentation();
  $httpBackend.flush();

  expect($scope.vm.internship.oralPresentation.masterValidation).toEqual(true);
  expect(window.confirm).toHaveBeenCalledWith('Warning! If you validate, you will no longer be able to add notes. Do you wish to continue?');
  expect(window.alert).toHaveBeenCalledWith('Oral Presentation Updated!');
  expect($state.reload).toHaveBeenCalled();
}));
```

Illustration 21: Description d'un test Jasmine

Pour détecter les actions sur la page, *Jasmine* utilise ce qu'on appelle des *spies*, c'est-à-dire des observateurs programmables. Ces observateurs attendent un événement précis et collectent des informations sur cet événement. Ce sont ces observateurs qui alimentent ensuite les *expect* que nous utilisons dans les tests. L'illustration 22 contient un exemple de l'utilisation de ces spies.

```
  // Spy on window confirm
  spyOn(window, 'confirm').and.callFake(function () {
      return true;
    });

  // Spy on state reload
  spyOn($state, 'reload').and.callFake(function(state, params) {
    return true;
    });

  // spyOn($state, 'reload');
  spyOn($state, 'go');
  spyOn(Notification, 'error');
  spyOn(Notification, 'success');
  spyOn($scope.vm, 'masterActivitiesNoteCommentAdd');

  spyOn(window, 'alert').and.callFake(function () {
      return true;
    });
}));
```

Illustration 22: utilisation de spies dans Jasmine

Il est également possible d'utiliser les *spies* pour agir sur un événement. Dans l'exemple proposé, si un spy observe une fenêtre de confirmation, il renvoie *true* pour simuler une confirmation de la part de l'utilisateur. Il est ainsi possible de répliquer tout comportement d'un utilisateur sur la page, et de tester la réponse de l'application en conséquence.

5.5 Liste des tests

Les listes suivantes contiennent l'ensemble des tests qui ont été réalisés avec *Mocha* et *Jasmine* du côté serveur et client.

5.5.1 Tests Serveur

Comme mentionné précédemment, il n'était pas possible dans le cadre de ce travail de tester l'application dans son intégralité. La liste suivante reprend l'ensemble des tests qui ont été effectués côté serveur.

Tests côté serveur	Type
Un utilisateur ne peut pas accéder à l'API s'il n'est pas connecté	Routage
Un admin doit pouvoir créer un stage vide le serveur doit lui renvoyer un stage vide.	Fonctionnalité
Un élève doit pouvoir créer un stage vide, le serveur doit lui renvoyer un stage partiellement completé avec l'identifiant de l'élève.	Fonctionnalité
Un élève ne peut appeler la route */api/auth/internships* qu'avec GET ou POST	Routage
Un élève ne peut proposer un deuxième superviseur s'il en a déjà proposé un.	Fonctionnalité
Un élève ne peut modifier sa proposition de stage si elle a été acceptée.	Fonctionnalité
Un élève ne peut modifier sa proposition que si le statut du stage est *encoding* ou *proposition reincoding*	Fonctionnalité
Si le validateur a validé le superviseur, seul le manager peut encore le changer.	Fonctionnalité
Si le manager a validé le superviseur, seul l'admin peut encore le changer.	Fonctionnalité
Un maître de stage ne peut modifier un stage qui ne lui appartient pas.	Fonctionnalité
Un enseignant ne peut modifier un stage qui ne lui appartient pas	Fonctionnalité
Un maître de stage ne peut pas ajouter de commentaires à la première visite en entreprise si le lieu et la date n'ont pas été fixés.	Fonctionnalité
Un superviseur de stage ne peut pas ajouter de commentaires à la première visite en entreprise si le lieu et la date n'ont pas été fixés.	Fonctionnalité
Il doit être possible de créer un stage vide	Données
Il ne doit pas être possible de sauvegarder un stage avec le mauvais type de données.	Données.

5.5.2 Tests client

Du côté client, les premiers tests effectués étaient ceux du routage AngularJS. Ce dernier permet de réaliser des single page applications, et pour ce faire met en place tout un système de routage entre les pages de l'application. Il était nécessaire de vérifier que ce routage était correct, et que les bons éléments étaient chargés au bons endroits. Pour le routage, nous avons donc vérifié les éléments suivants :

- Les bonne routes sont elles appelées ?
- Les routes correspondent-elles aux bonnes URL ?
- Les bons contrôleurs sont-ils chargés pour chaque route?
- Est-ce que le bon template est chargé à chaque route ?
- Pour les routes qui ont besoin d'une liste des stages, est-elle bien chargée ?
- Pour les routes qui ont besoin d'un stage précis, est-il bien chargé ?

Ces tests ont été menés pour les routes du coordinateur, du maître de stage, ainsi que pour les routes de l'élève. Les routes des autres acteurs doivent encore être testées.

Nous avons ensuite testé les contrôleurs afin de valider leur comportement. Pour le contrôleur du maître de stage, nous avons testé :

- Les bonnes adresses de l'API sont-elles appelées au moment du chargement ?
- Lors de la validation de la visite en entreprise, ainsi que lors de la validation de la défence orale :
  - les bonnes routes sont-elle appellées ?
  - les bons messages d'alerte et de confirmation sont-ils affichés à l'utilisateur?
  - la page est-elle ensuite rechargée ?
- Si le formulaire à remplir lors de la validation de la note d'activités n'est pas rempli :
  - est-ce que le contrôleur évite d'appeler les routes de l'API ?
  - est-ce qu'il return correctement, sans entreprendre d'actions supplémentaires ?

Pour le contrôleur de l'enseignant, nous avons voulu tester le bon fonctionnement du tri des stages. Lors du chargement du contrôleur, il est censé prendre la liste des stages reçus, et en fonction des caractéristiques de chaque stage, les ranger dans trois listes. Au début du test, nous créons donc une série de cinq « faux » stages, dont un deux est présenté à l'illustration 23.

```
// internship should be in supervisor demands list"
internship2 = {
  _id : "IdNumber2",
  state : "In Internship",
  generalStatus : "Proposition Validation",
  student : "Student2",
  supervisor: {
    proposedSupervisor: {
      username: "SebCom"
    },
  propositionResponse : null
  }
},
```

Illustration 23: utilisation d'un faux stage pour tester le backend

Sur le stage de l'illustration 24, on peut voir que le champ « proposedSupervisor » contient le nom de notre enseignant factice. Le champ « proposition Response » en revanche est null, ce qui indique que l'enseignant n'a pas encore donné de réponse. Ce stage devrait donc être rangé dans la liste « *demandes de supervisorats* » de l'enseignant.

Le test principal pour ce contrôleur est donc de vérifier que le contenu de chaque liste correspond bien au résultat attendu après le tri.

```
// verify correct sorting of internships
expect($scope.vm.subjetApprovalDemands.map( function (internship) { return internship._id})).toEqual([internship1._id]);
expect($scope.vm.supervisorDemands.map( function (internship) { return internship._id})).toEqual([internship2._id, internship3._id]);
expect($scope.vm.myInternships.map( function (internship) { return internship._id})).toEqual([internship4._id]);
```

Illustration 24: résultats attendus après le tri des stages

Ces mêmes vérifications sont effectuées dans le contrôleur du coordinateur avec des listes différentes ainsi que des stages avec des caractéristiques différents. À ce stade-ci, les contrôleurs du maître de stage, enseignant, coordinateur et un contrôleur de l'élève ont été testés. Les contrôleurs des autres rôles doivent encore être testés et validés.

5.6 Test Driven Developpment

Les tests ont apporté un élément nouveau et inattendu en terme de réflexion durant ce TFE. Cela peut paraître trivial, mais afin de tester l'application, il était nécessaire de réfléchir aux tests qu'il faudrait mener, et à la façon dont ils devaient être formulés puis mis en application. Cet exercice m'a forcé a prendre du recul par rapport au cahier des charges qui a été établi et, ce faisant, a changé l'angle avec lequel j'abordais le développement de l'application.

Durant les tests, j'ai vu ce projet non plus du point de vue d'un développeur, mais du point de vue d'un testeur. Il s'agit là d'un mode de pensée assez différent. On ne pense plus à *comment* on va implémenter une solution, mais plutôt à ce qu'on *veut* implémenter. On part du cahier des charges et on établit une *spec* ou liste des fonctionnalités attendues. Cette liste se traduit alors en tests qui vont ensuite modeler l'implémentation de l'application.

Il s'agit d'un approche de type TDD ou *Test Driven Developpment*, c'est-à-dire une méthode de développement où les tests sont à la base de toute l'implémentation du programme. On définit d'abord les tests, et puis on développe la solution qui permet de valider ces tests. L'approche ici considérée est tout à fait contraire à ce qui a été fait lors de ce TFE, ou le développement a d'abord été fait, puis les tests.

À y repenser, si ce projet était à refaire il serait intéressant d'envisager un style de développement proche du TDD. Cela demanderait cependant deux choses qui n'ont pas été possibles durant ce travail de fin d'études. La première serait sans doute et avant tout une bonne connaissance préalable de la stack MEAN. La deuxième serait de constituer une équipe divisée en deux groupes de testeurs et de développeurs, et de ne plus avoir une personne cumulant tous les rôles.

6 Conclusion

Aujourd'hui, le projet est désormais fonctionnel et en phase de tests et d'intégration. Nous pouvons à présent dresser le bilan de son état d'avancement.

Chaque acteur est en mesure de remplir son rôle pleinement au sein d'un stage. Un stage peut ainsi se dérouler de façon complète et non bloquante, c'est-à-dire que de la création du stage jusqu'à son évaluation finale, tous les blocs nécessaires à son déroulement sont fonctionnels.

Dans l'éventualité d'un problème, scénario non envisagé ou bug pouvant entraver la progression d'un stage, l'admin peut toujours intervenir et corriger le problème, ce qui assure la progression du stage. Ainsi, un stage ne sait pas se retrouver bloqué et cesser de progresser.

Si le coeur de l'application est bel et bien fonctionnel, certains éléments non essentiels n'ont pas encore étés rajoutés :

- La génération automatique des conventions de stage (F12)
- La possibilité pour l'élève d'uploader son rapport de stage, et la création d'une bibliothèque avec l'ensemble des rapports de stage (F12)
- Un dépôt regroupant l'ensemble des documents administratifs mis à disposition de l'élève (F12)
- Un système de notifications par alertes et e-mails (F13)

D'autres fonctionnalités n'ont pas encore été créées car elles feront l'objet de projets séparés. Parmi ceux-ci, il y a par exemple l'encodage des points du stage. Cette fonctionnalité sera rajoutée une fois que le module gérant les grilles d'évaluation aura été créé. Une fois ce module créé, il sera alors facile d'importer les grilles d'évaluation dans le module stage. Il suffira de compléter la grille, puis de l'attacher soit directement, soit par référencement au document du stage concerné. Dans la même optique, le regroupement et encodage des deadlines fera aussi l'objet d'un module supplémentaire.

Quelle sera la suite pour le développement de ce module de gestion des stages ? L'application est arrivée à son étape d'intégration. Il faudra tout d'abord la tester avec plusieurs élèves, enseignants et membres de l'administration. Ces tests sur des stages réels fourniront un feedback précieux sur les modifications et améliorations à apporter au programme. Ils seront effectués durant l'année académique 2017-2018 sur les élèves de bac-3, les premiers encodages de propositions débutant fin octobre. Ce sera aussi l'occasion de résoudre d'éventuels bugs de la plateforme. Une fois ces corrections et adaptations faites, suivra la création et l'ajout de modules complémentaires pour enrichir la solution.

Personnellement, j’ai pris beaucoup de plaisir lors de ce tfe, principalement parce qu’il m’a permis d’en apprendre beaucoup non seulement sur le développement web, mais aussi sur la structure d’un projet assez complexe. Modéliser le procédé de déroulement d’un stage était un des grands challenge pour moi. Si comprendre l’organisation administrative était en soit parfois compliqué, la modéliser de façon informatique l’était encore plus. Je suis assez content d’avoir réussi au final à proposer un modèle assez simple et compréhensible.

C’était aussi la première fois que je travaillais avec une stack complète. L’utilisation et l’apprentissage de la stack MEAN m’a permis de mieux appréhender la structure modulaire des sites modernes d’aujourd’hui. C’était également l’occasion de me familiariser avec ses composants : Mongo, Express, AngularJS et NodeJS. J’ai beaucoup aimé découvrir l’utilisation d’une base de données NoSQL (MongoDB), surtout après avoir vu ce type de bases de données de manière théorique lors de cours à l’ECAM. C’était aussi l’occasion de me familiariser avec AngularJS, que j’ai ensuite réutilisé dans de nombreux projets à l’ECAM.

Si ce projet était à refaire, je pense qu’il vaudrait la peine de d’abord constituer une petite équipe de développement. Cette équipe ne serait pas grande, deux ou trois personnes tout au plus, mais cela permettrait déjà d’avancer de façon plus rapide, et d’aborder certains problèmes avec des points de vue différents. Si développer seul est en soit confortable et offre une liberté d’action quasi totale, il y a des moments ou l’on se retrouve parfois seul et désemparé face à certaines adversités. Travailler en groupe force également à avoir une certaine rigueur dans l’organisation du travail, et permet également de maintenir une meilleure vue d’ensemble sur le projet.

Il me hâte de voir les résultats de déploiement de ce module de gestion des stages. Je souhaite de tout coeur qu’il soit bénéfique à l’ECAM ainsi qu’à l’ensemble des utilisateurs qui seront amenés à l’employer.

7 Annexes

7.1 Captures d’écran du programme

7.1.1 Interface utilisateur de l’élève

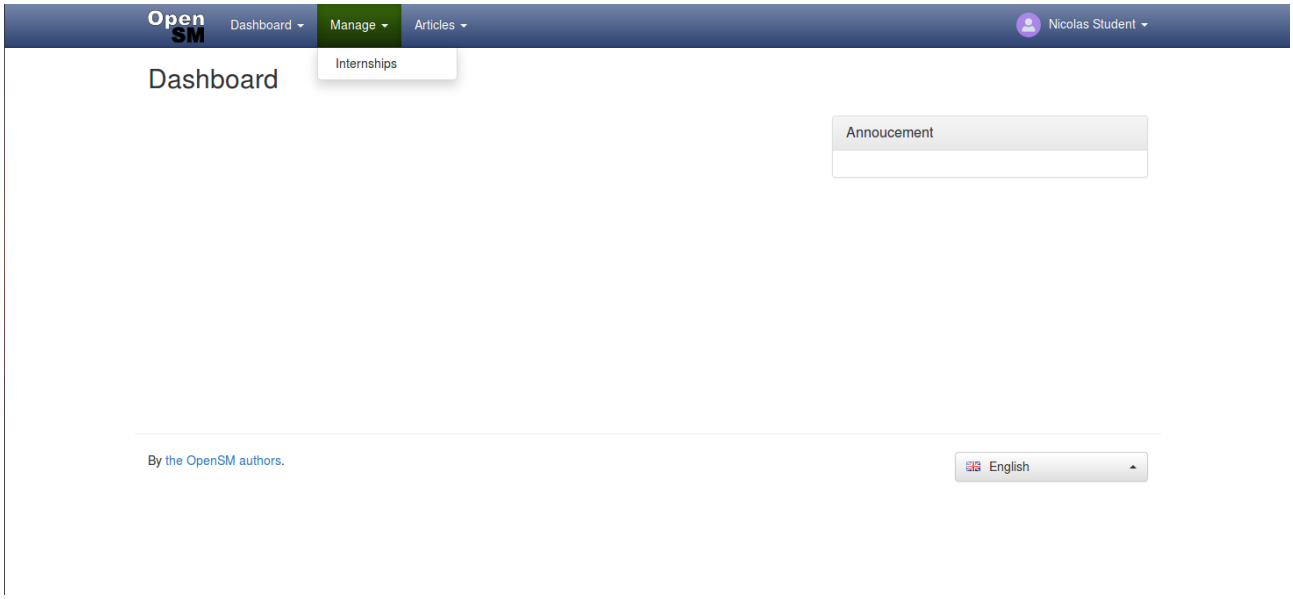


Illustration 1: le sous menu internships est ajouté au menu de gestion de l'élève

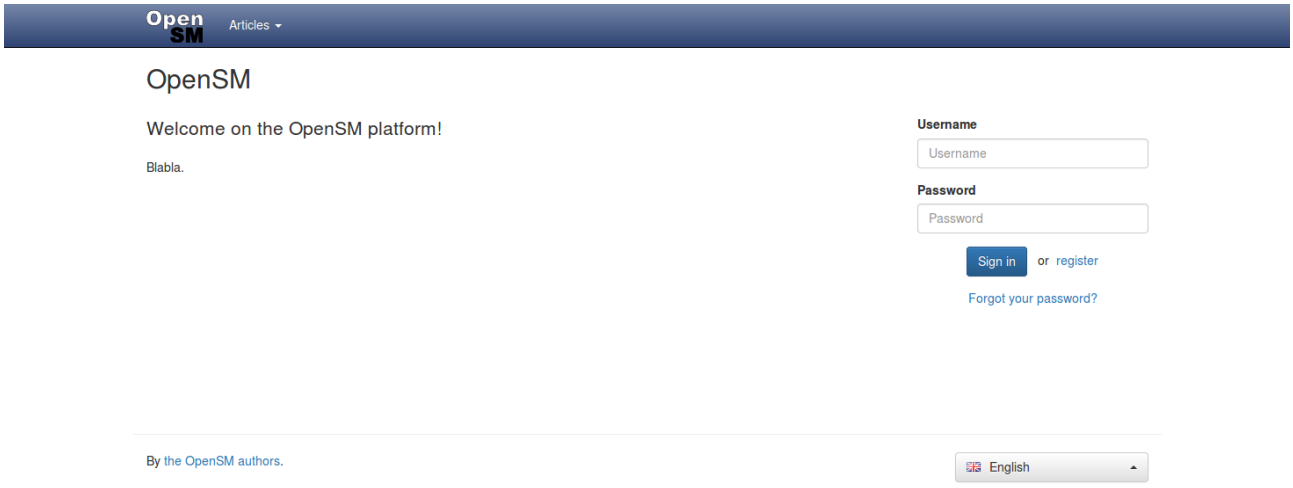


Illustration 2: page de connexion à la plateforme

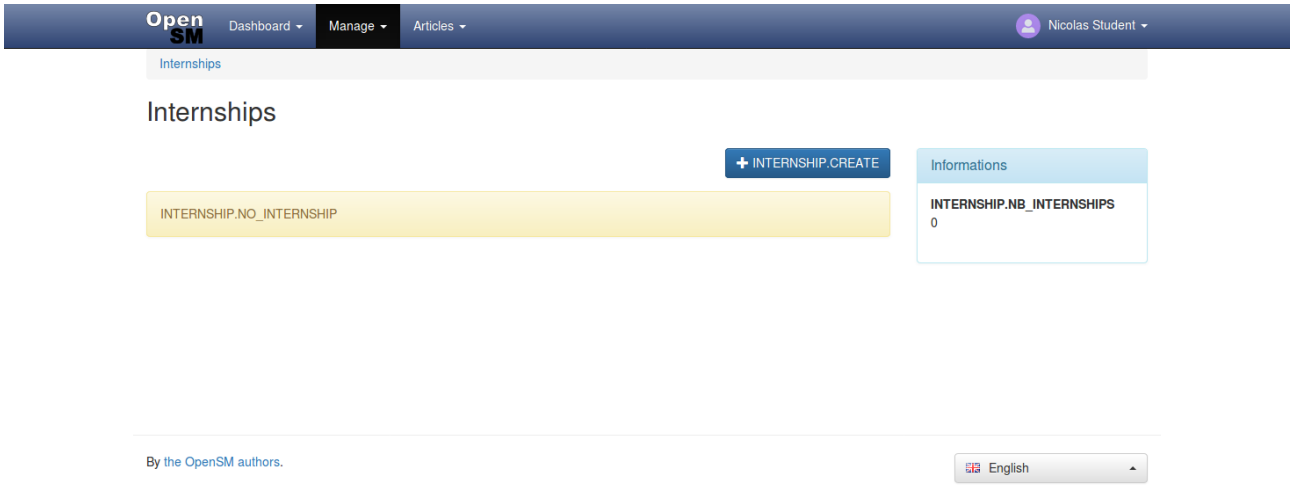


Illustration 3: Liste des stages de l'élève. Cette liste est pour le moment vide

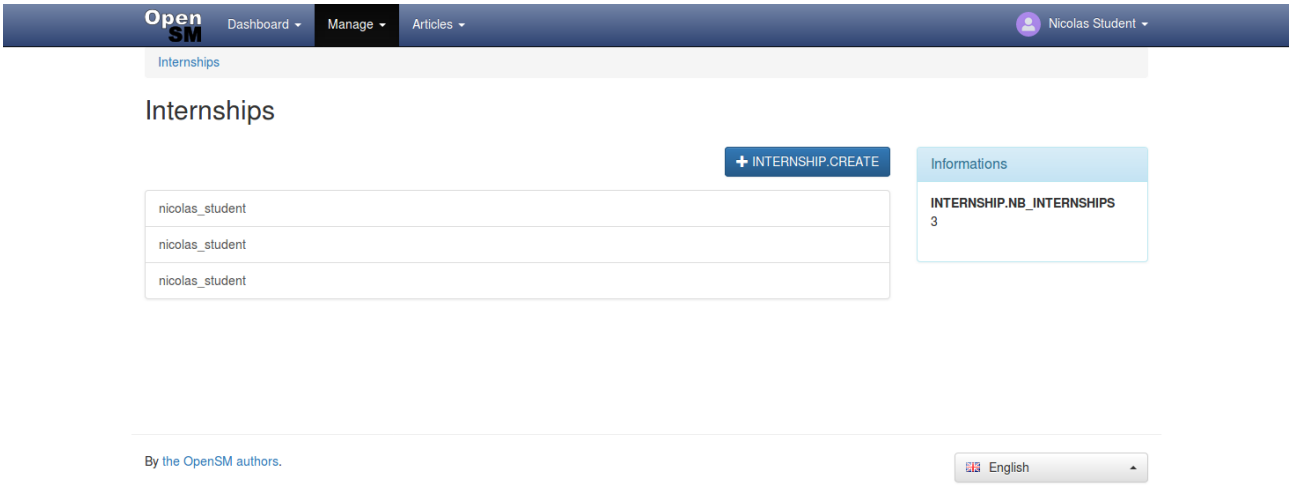


Illustration 4: S'il n'a pas de stage, l'élève peut en créer un. le stage s'ajoute à la liste de ses stages.

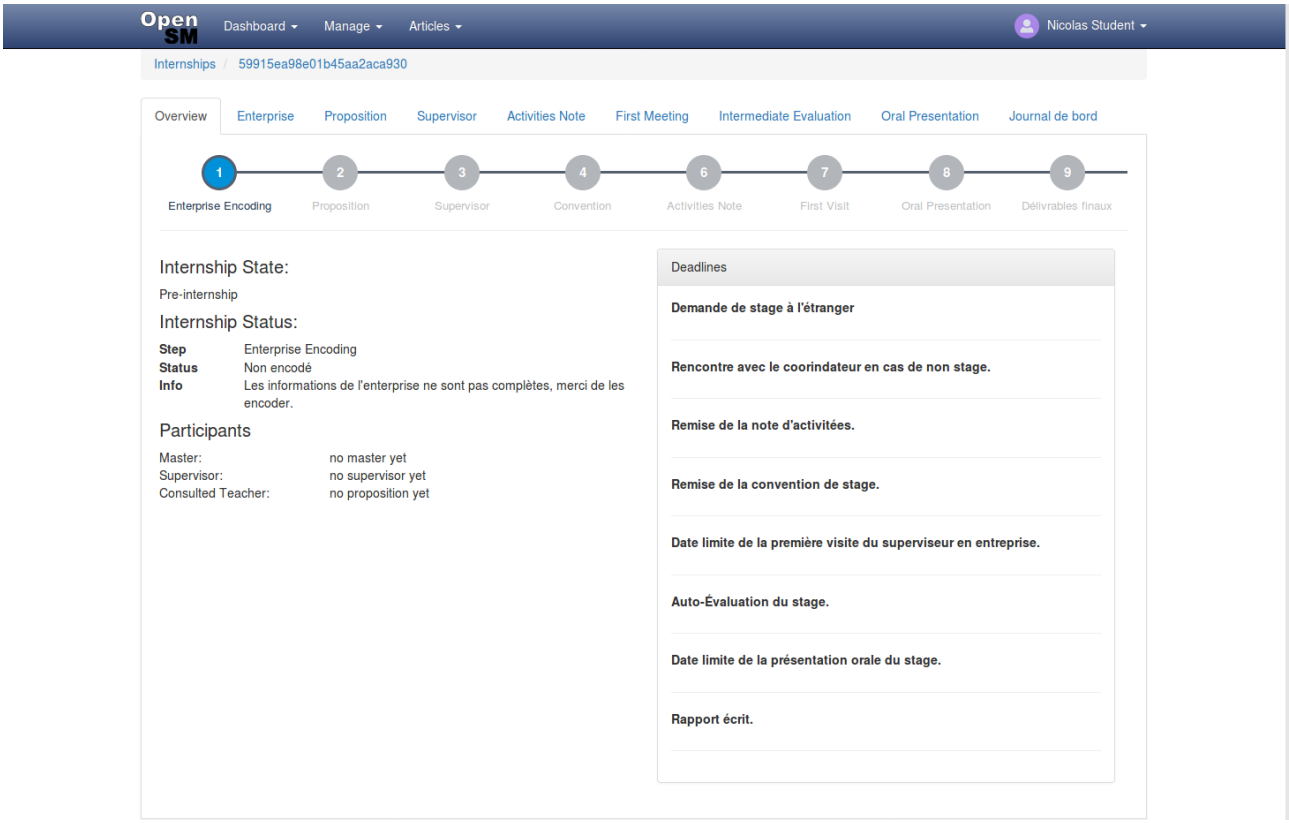


Illustration 5: tableau de bord pour le stage d'un élève

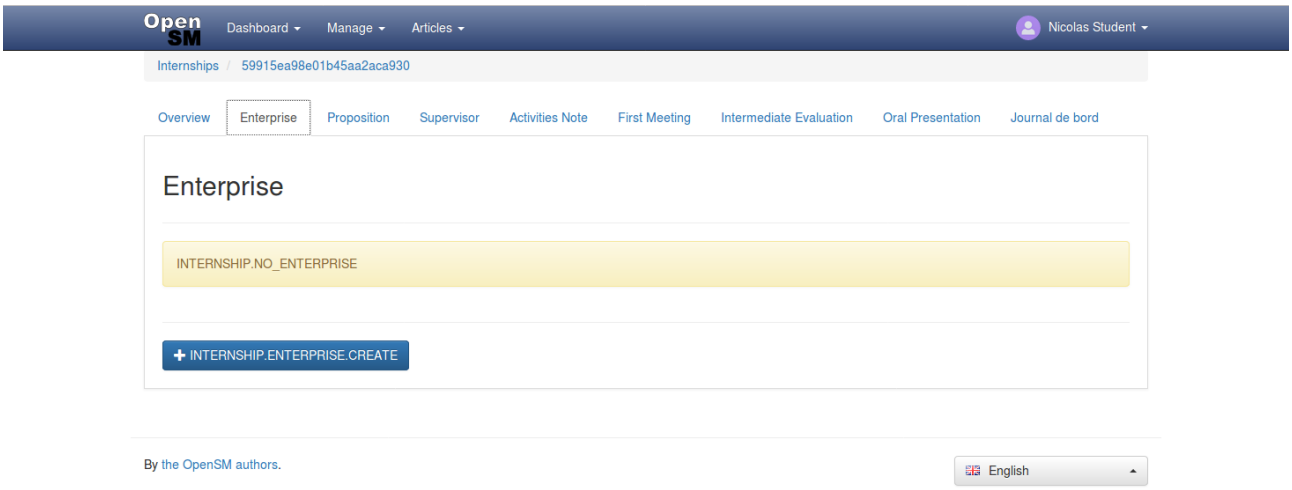


Illustration 6: L'entreprise n'a pas encore été encodée. Le bouton d'accès au formulaire est accessible à l'élève

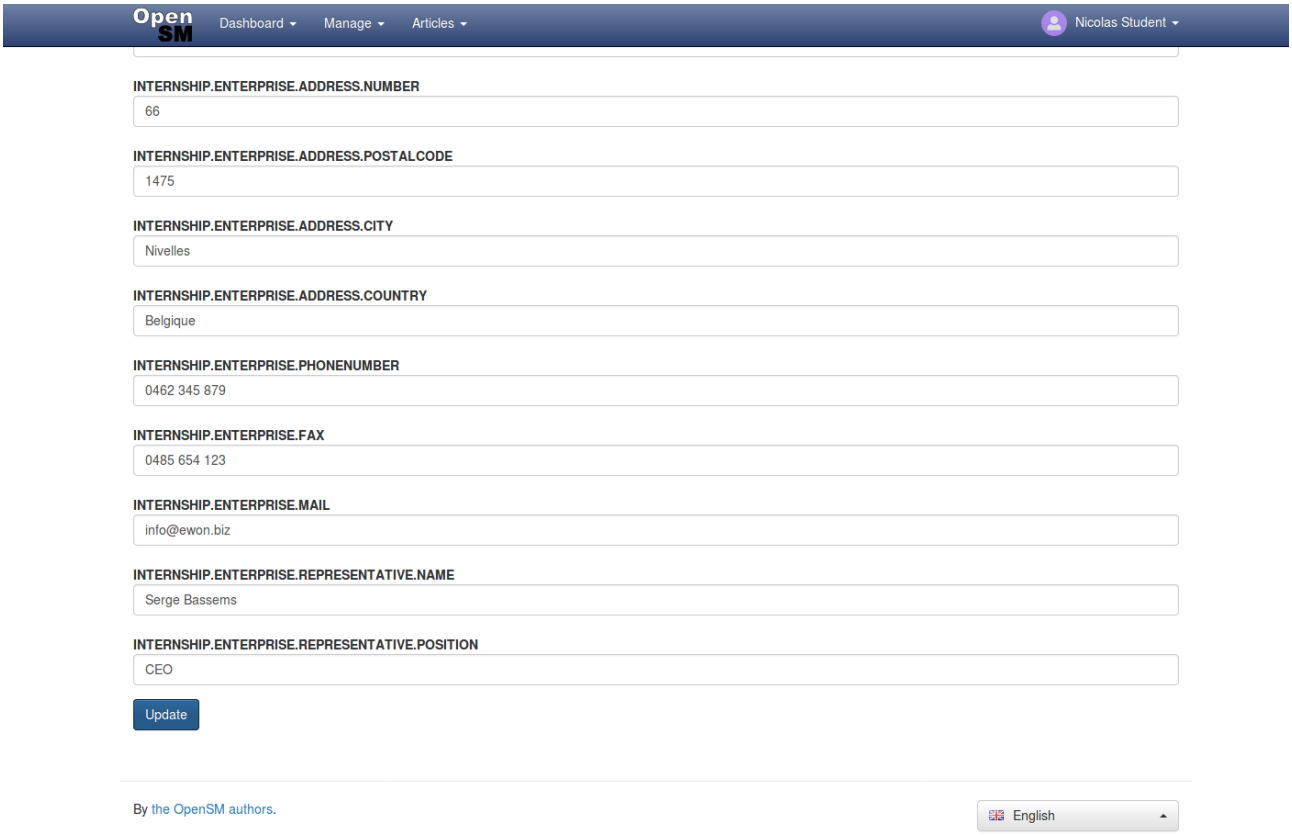


Illustration 7: formulaire d'encodage des informations de l'entrerprise

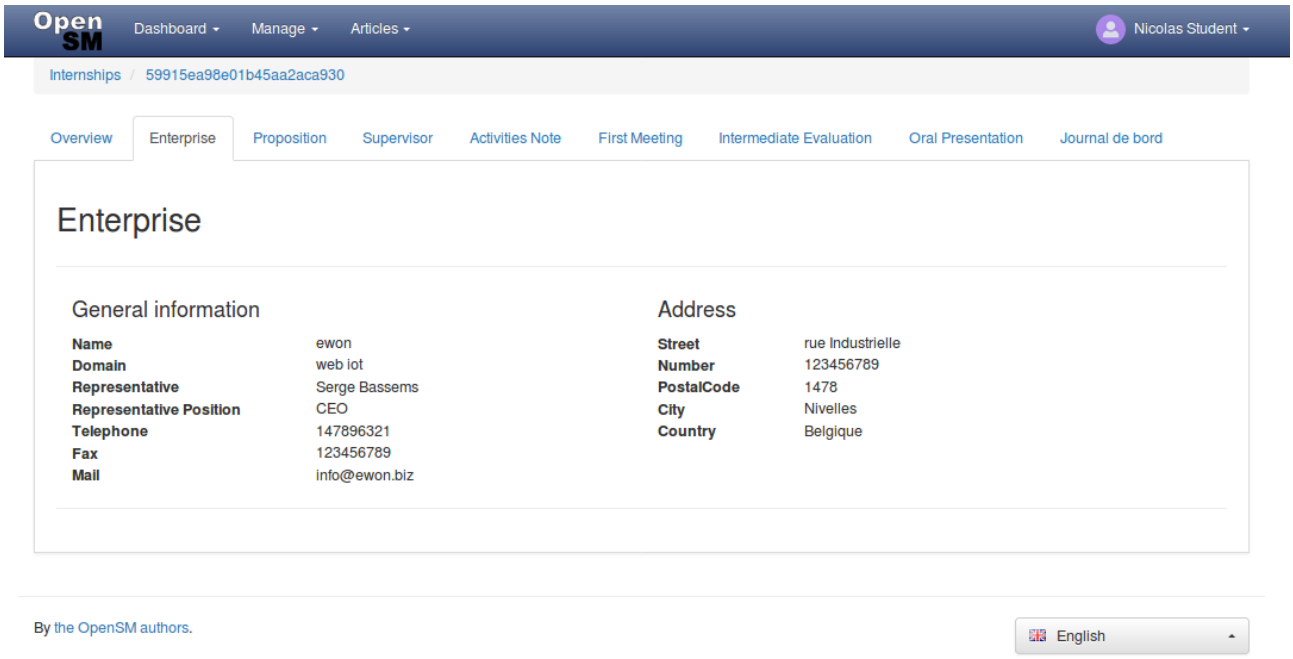


Illustration 8: Onglet entreprise après encodage des informations. le bouton d'accès au formulaire est caché

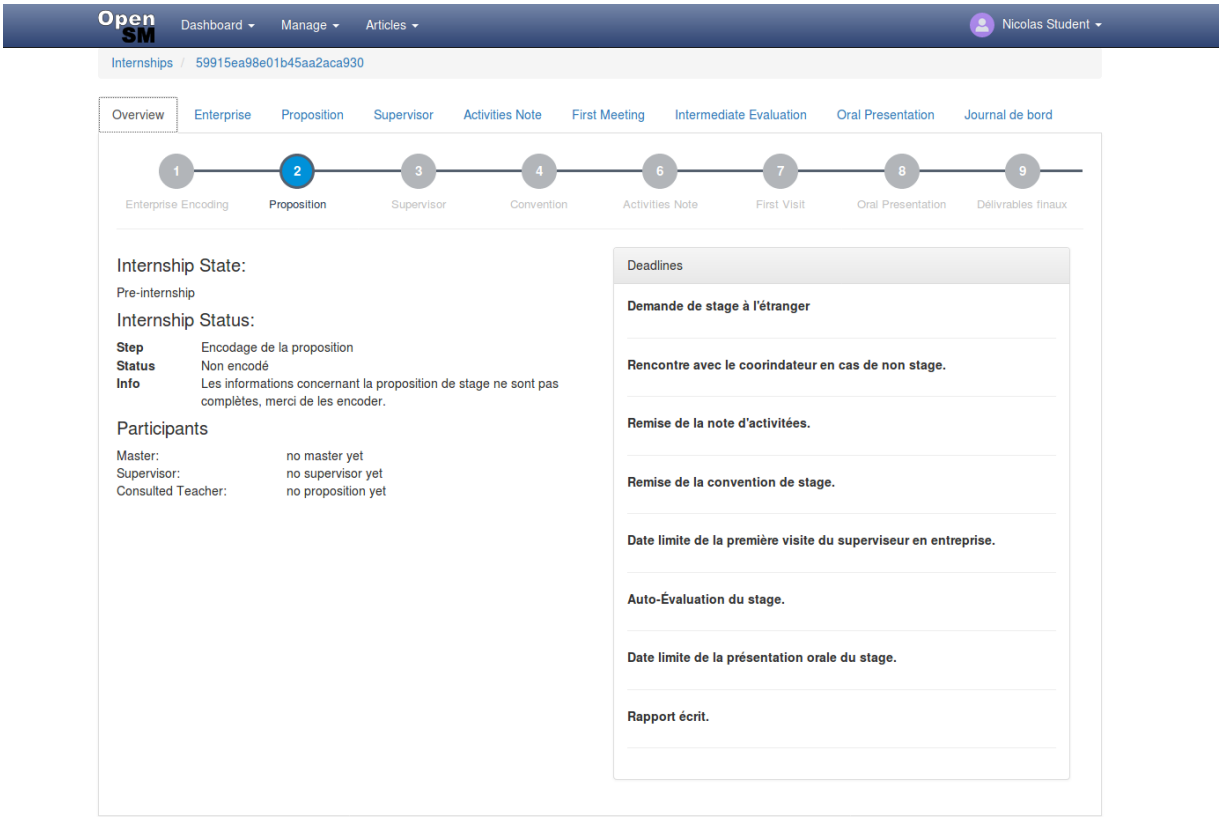


Illustration 9: progression du stage vers l'encodage de la proposition

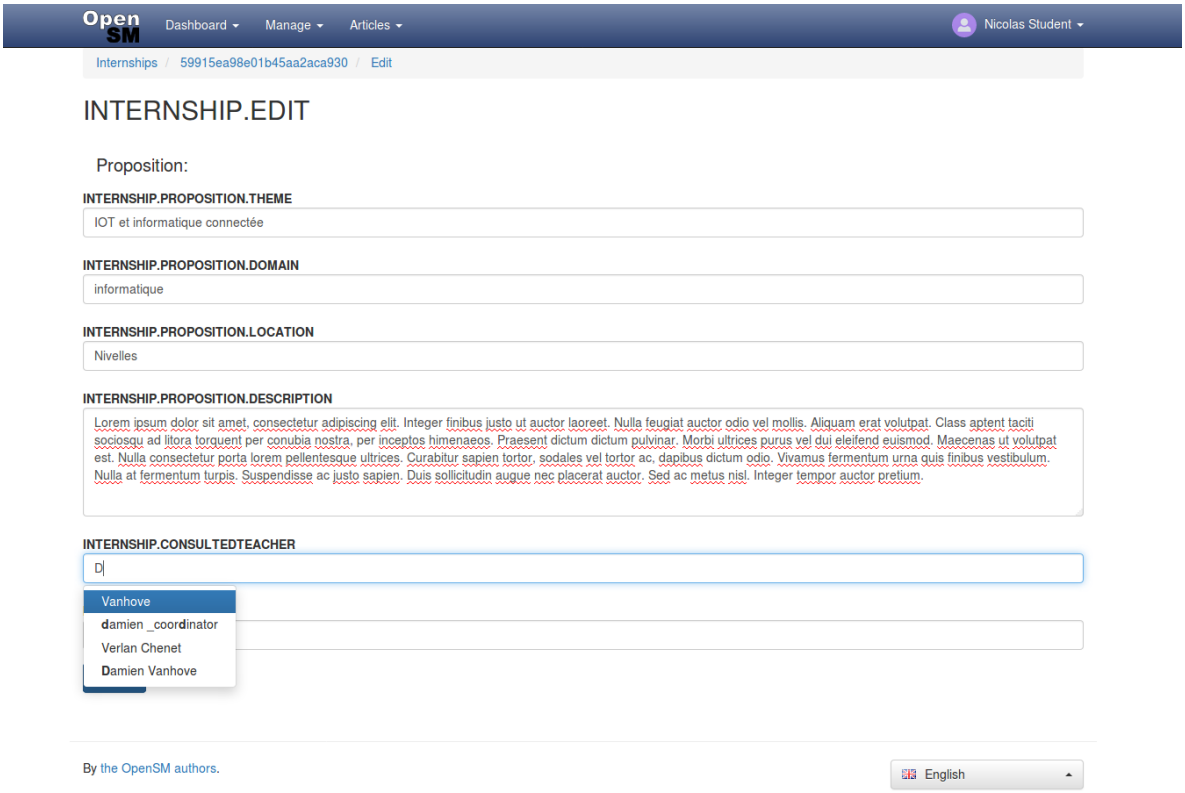


Illustration 10 : Formulaire d'encodage de la proposition de stage. Auto-complétion du champ "enseignant"

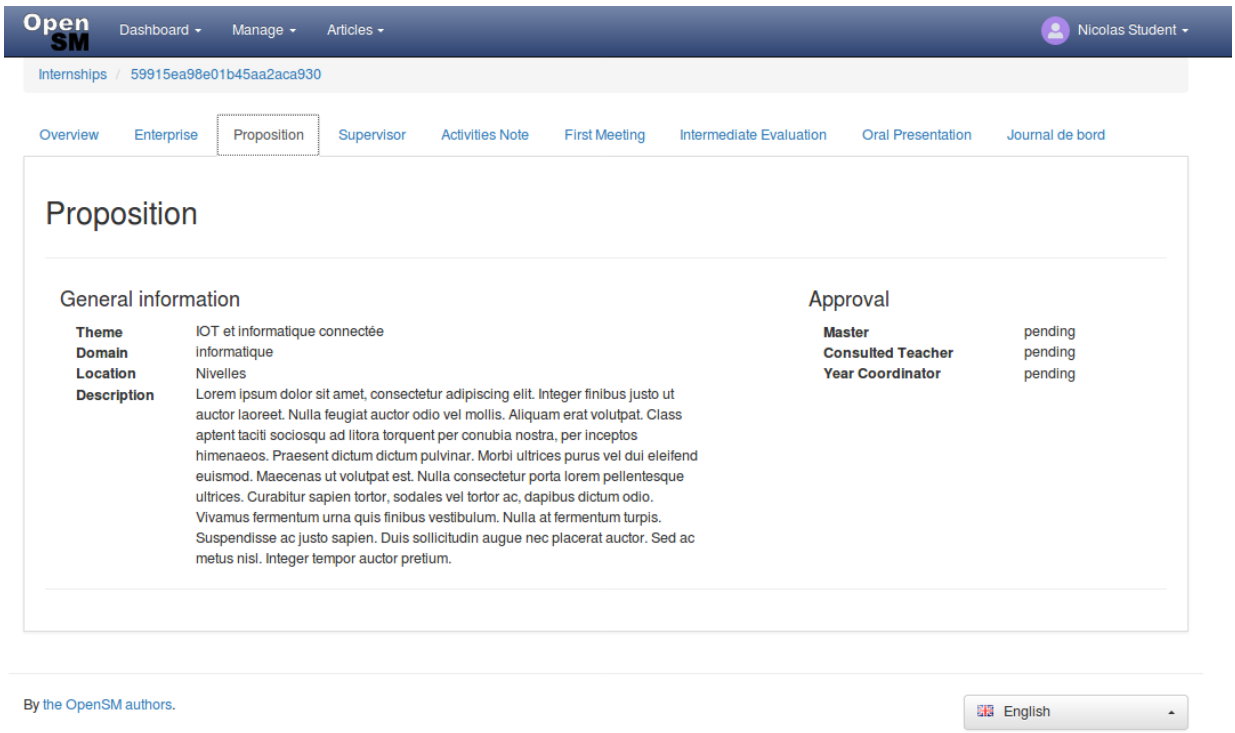


Illustration 11: en attente de validation, l'élève ne peut plus modifier sa proposition

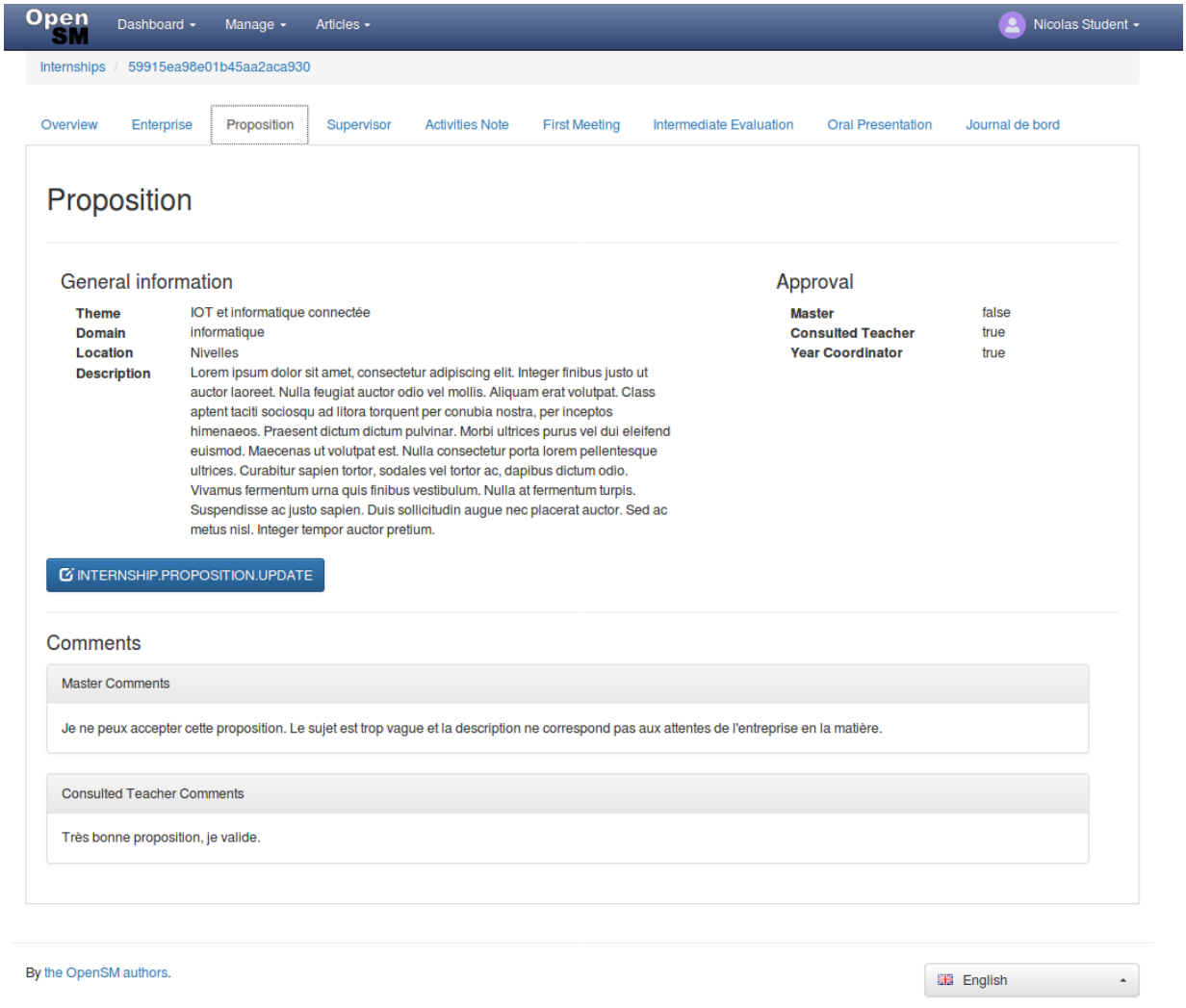


Illustration 12: proposition après validation et ajout de commentaires.

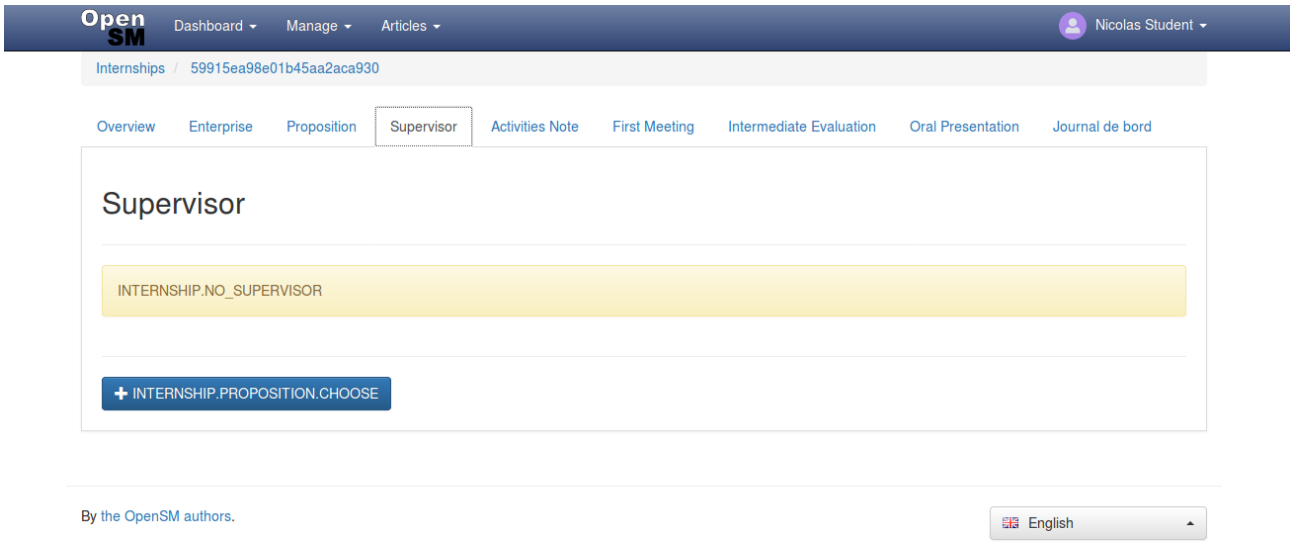


Illustration 13: l'élève peut choisir un superviseur

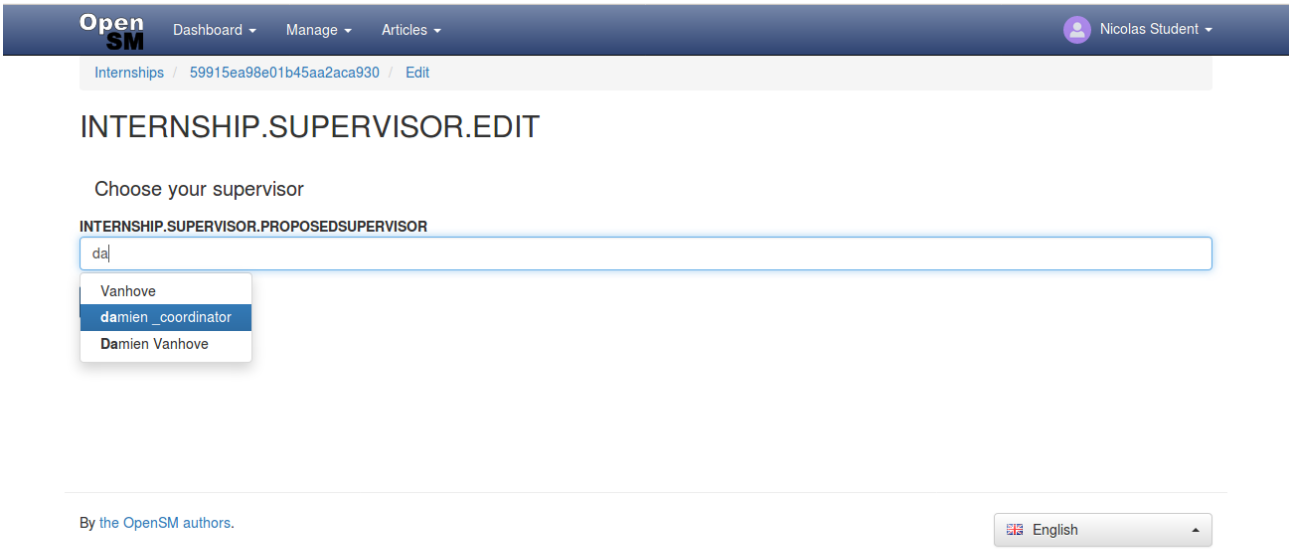


Illustration 14: formulaire d'encodage du superviseur

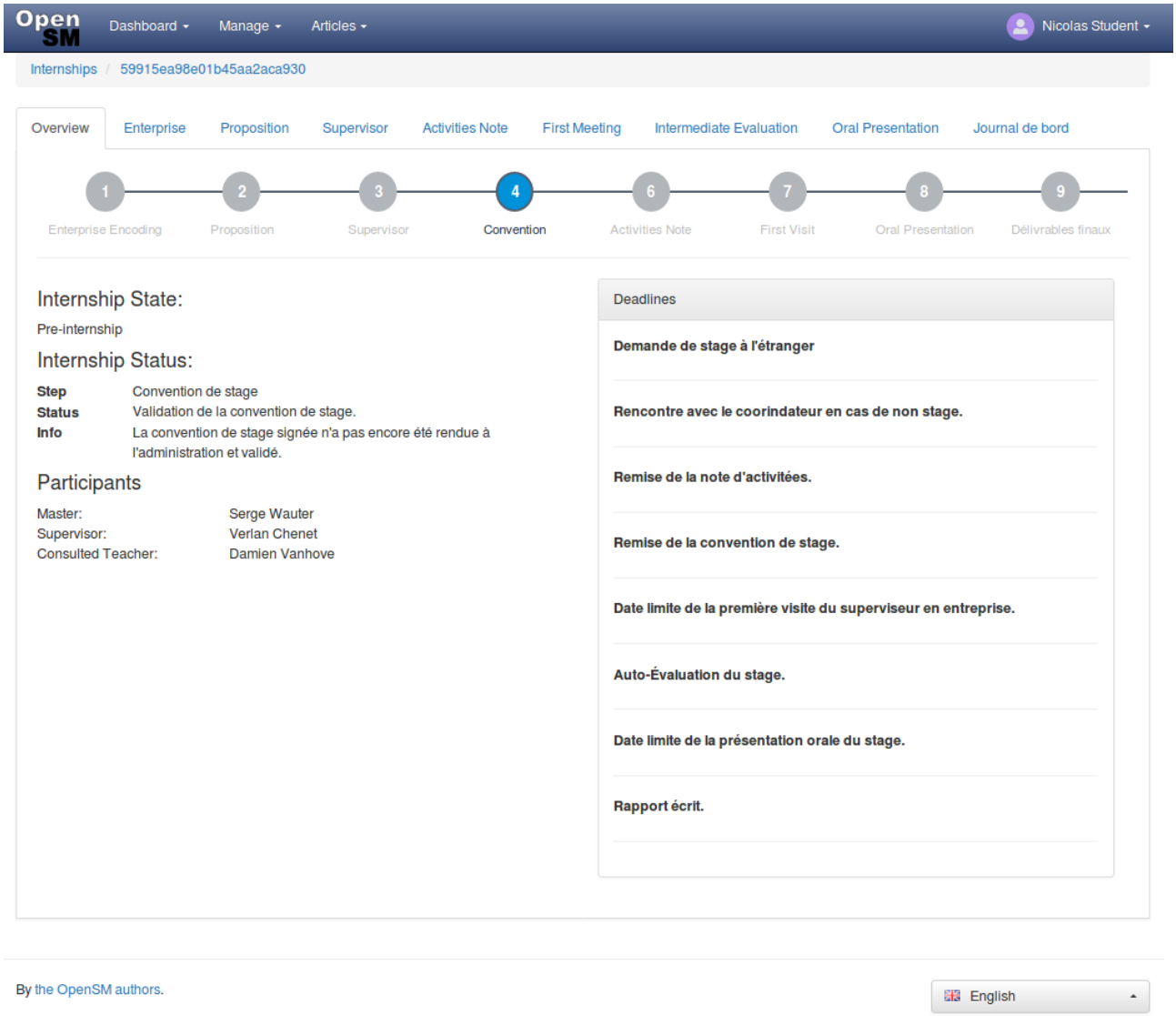


Illustration 15: Overview du stage lors de la sortie de la convention de stage

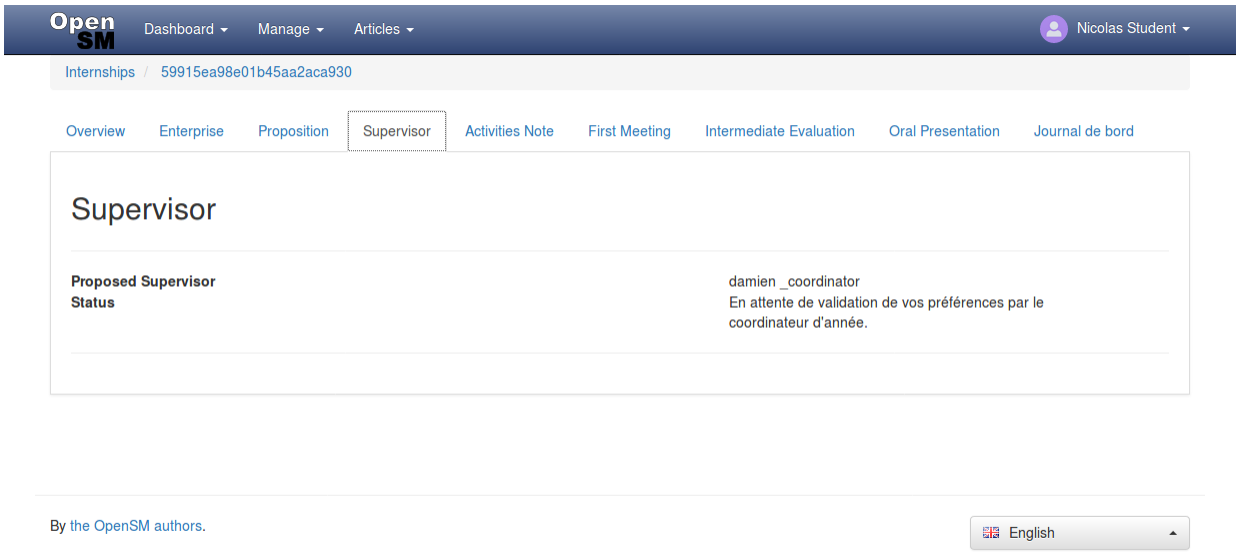


Illustration 16: Onglet supervisuer après encodage de la préférence de l'élève

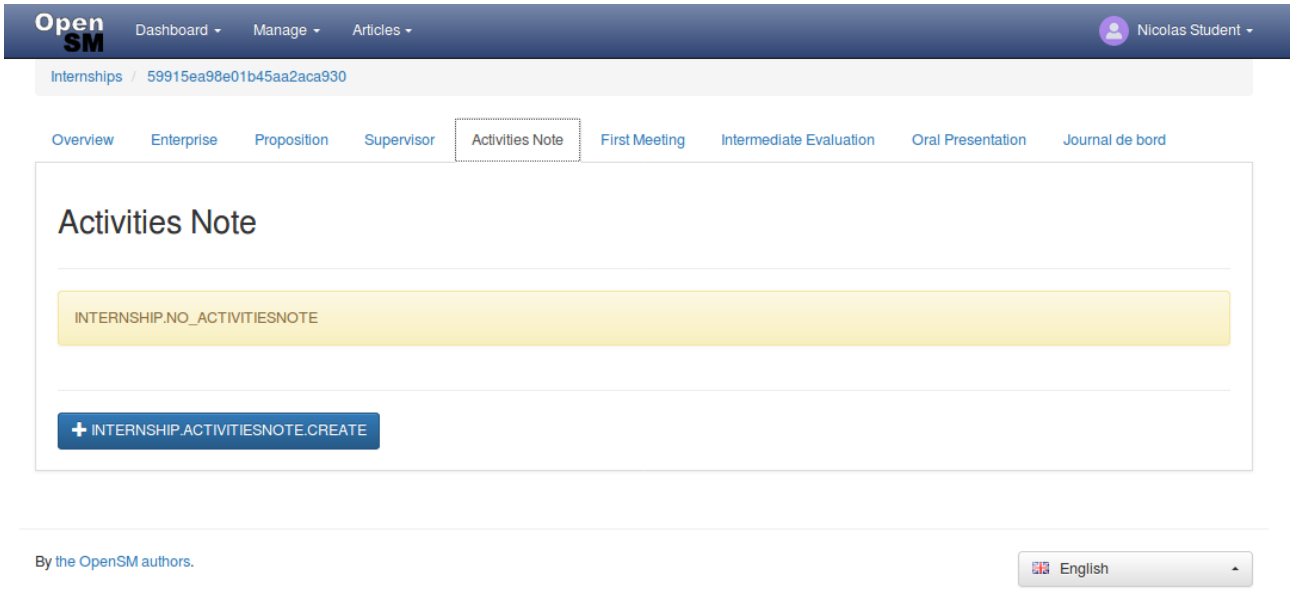


Illustration 17: Onglet Activites Note avant encodage

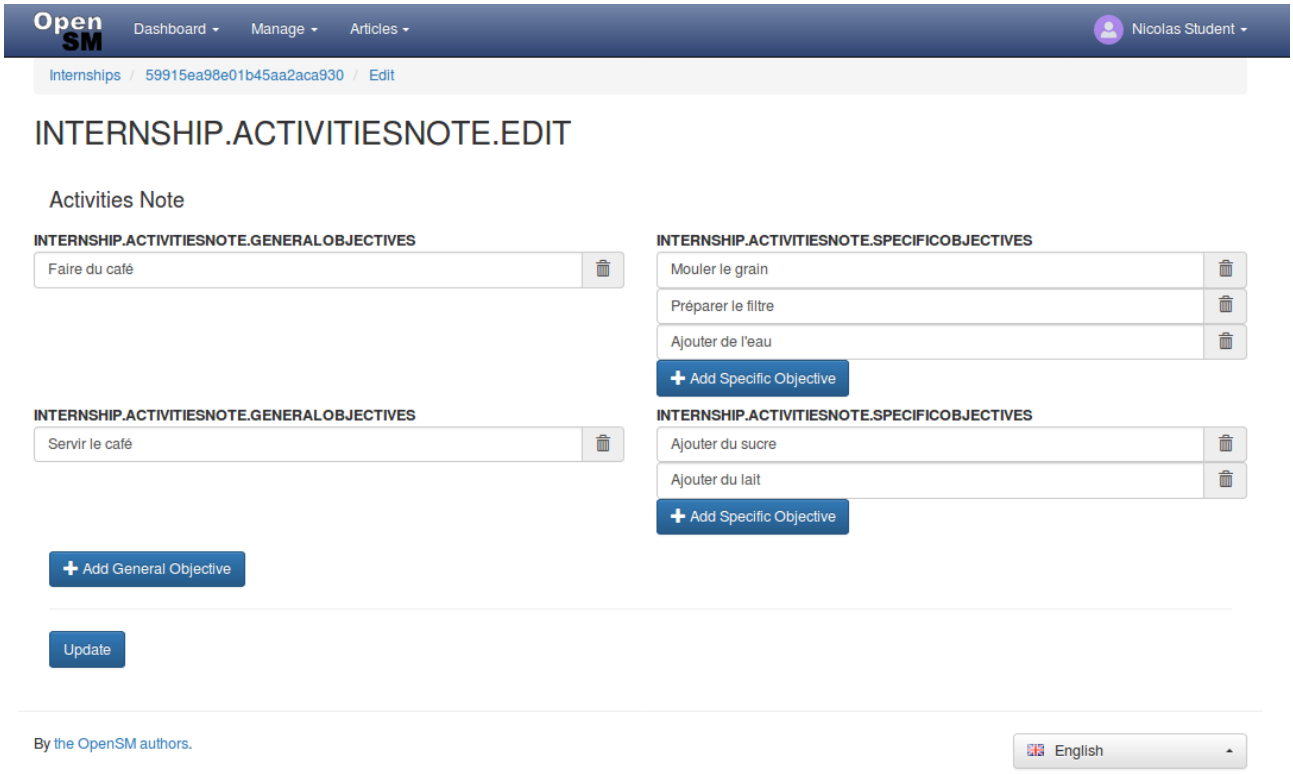


Illustration 18: formulaire d'encodage de la note d'activités

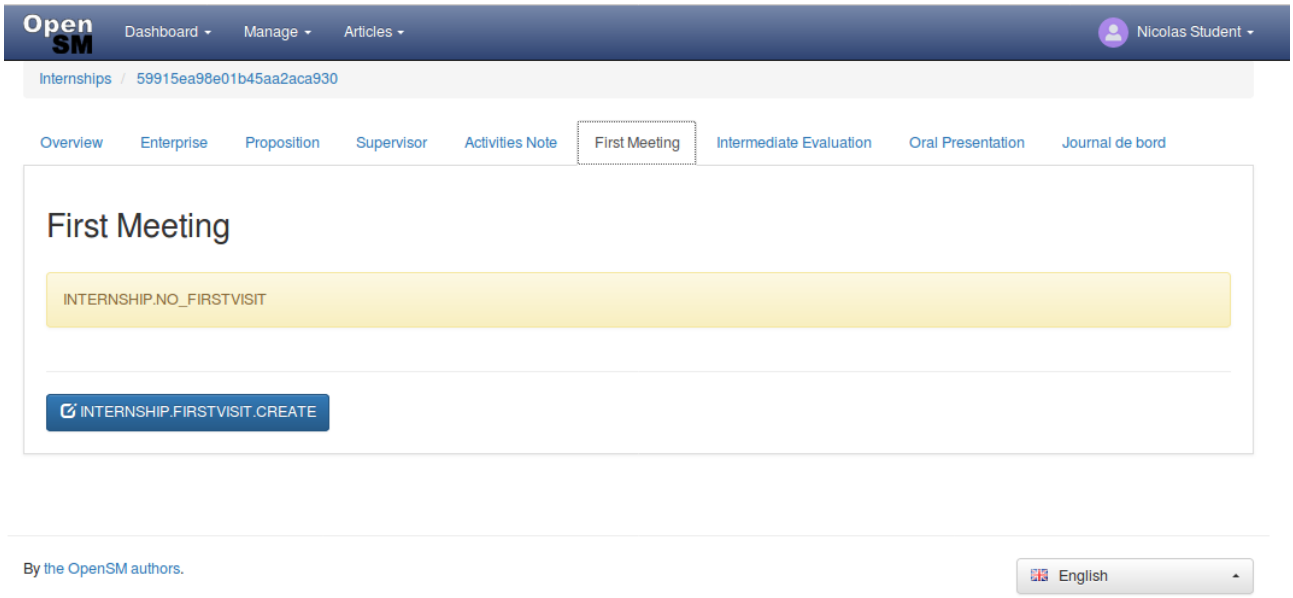


Illustration 19: L'élève n'a pas encore encodé la date et lieu de la première visite en entreprise

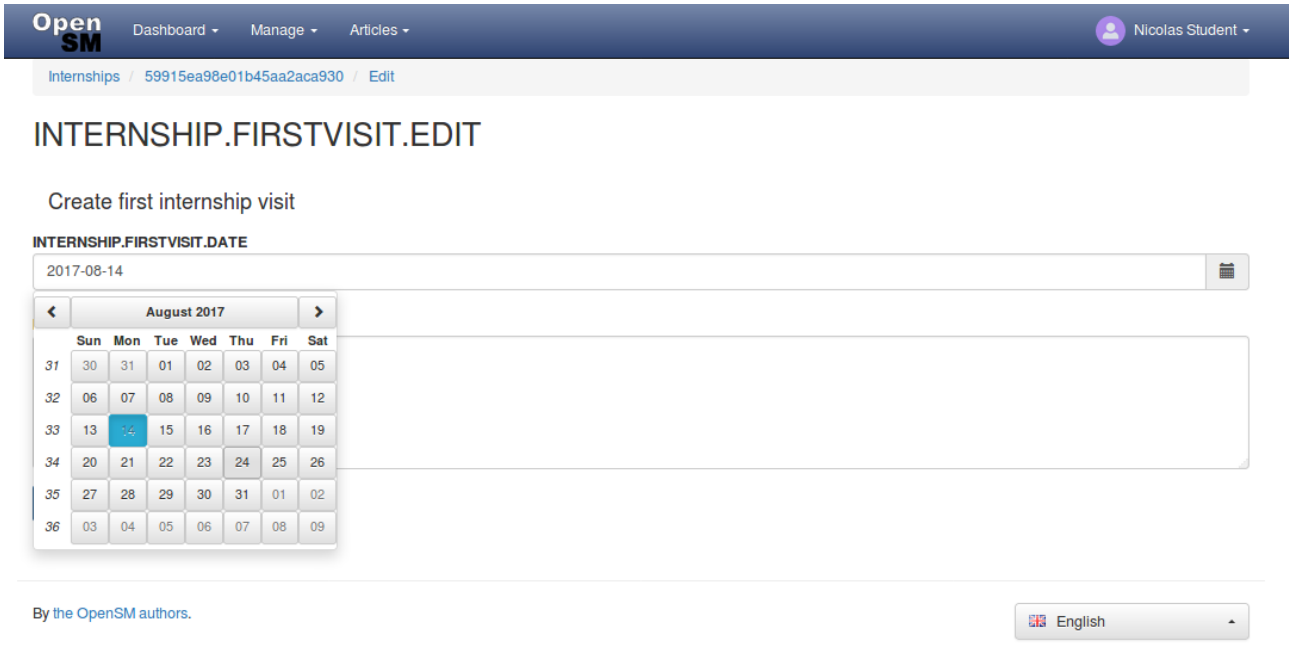


Illustration 20: encodage du lieu et de la date de la première visite

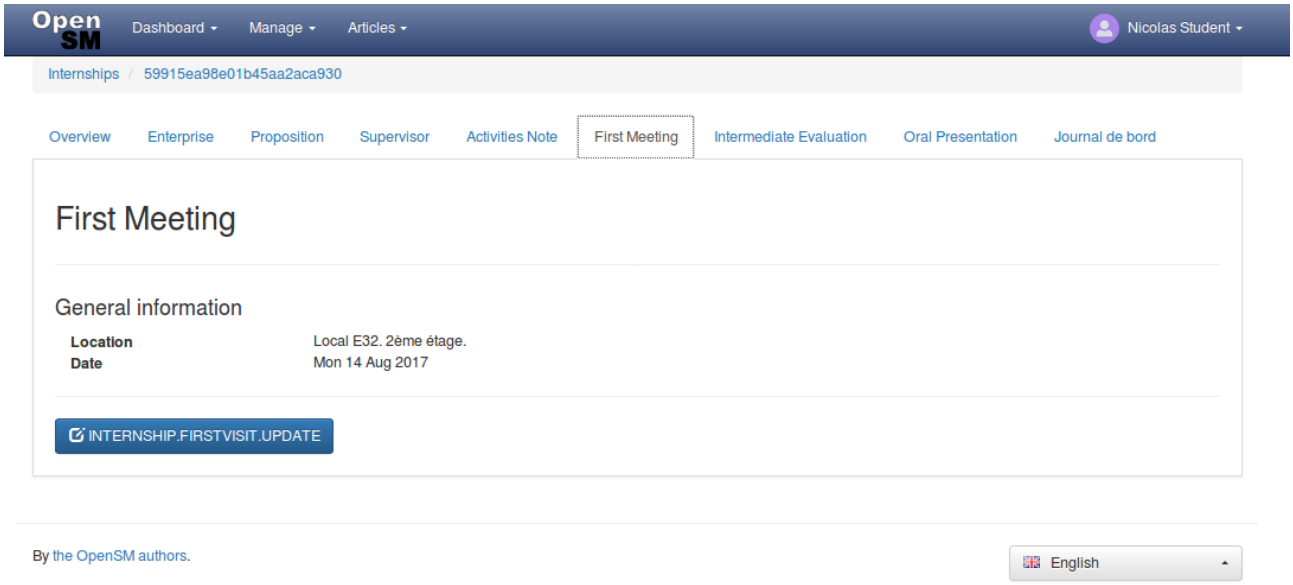


Illustration21: après encodage, l'étudiant peut encore modifier les données tant que le maître de stage ou superviseur n'ont pas encodé d'informations

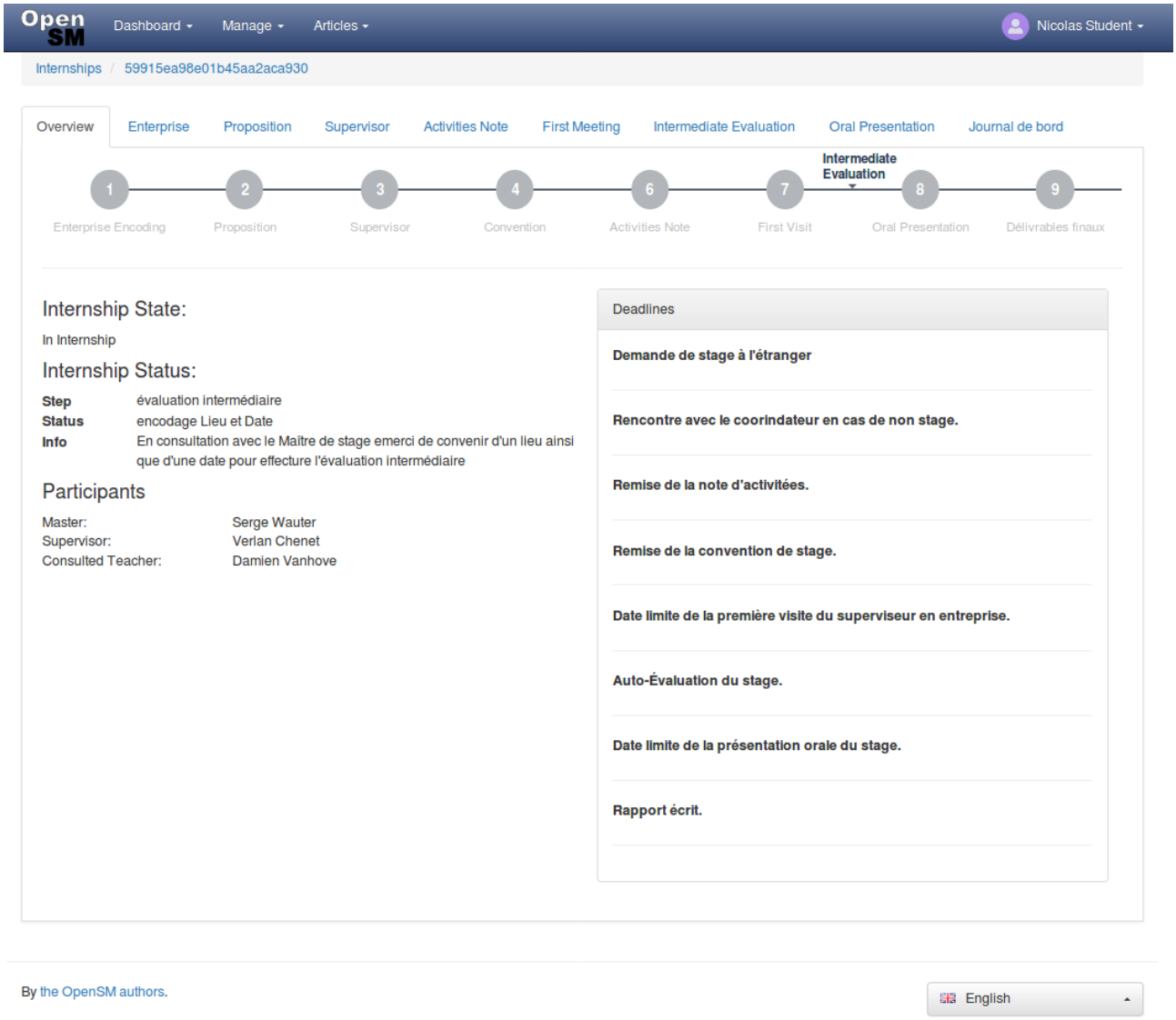


Illustration 22: avancement de l'overview du stage vers l'évaluation intermédiaire. Les onglets et formulaires pour cette évaluation ainsi que la défence orale sont identiques à ceux de la première visite en entreprise.

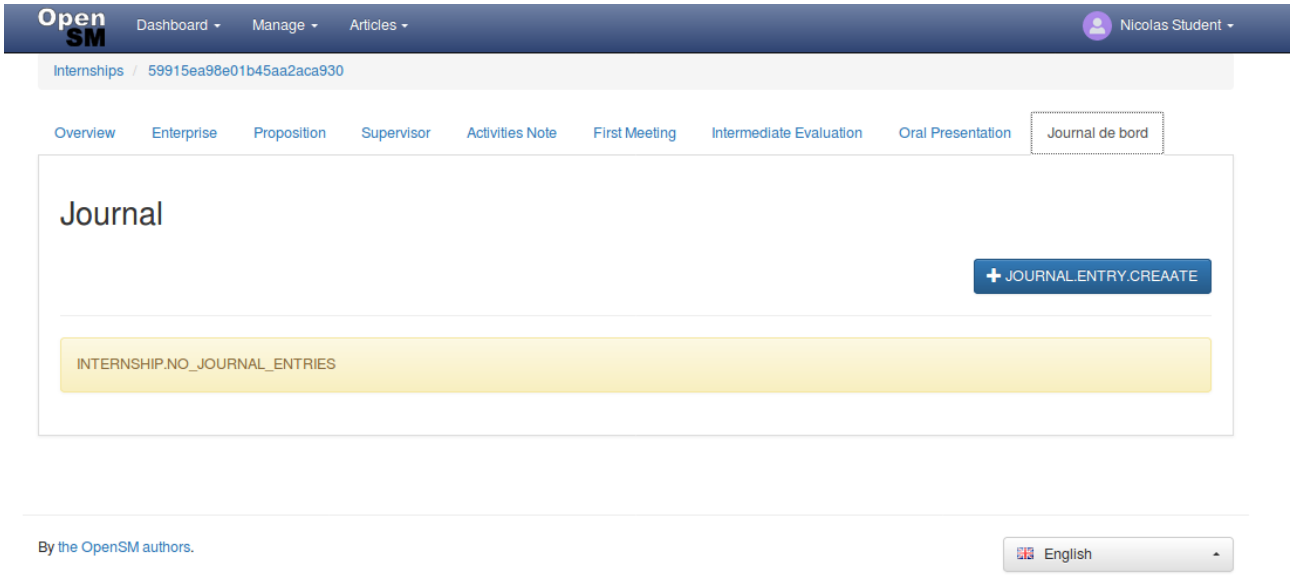


Illustration23: le journal de bord de l'élève est pour l'instant vide. Le bouton en haut à droite permet de rajouter une entrée

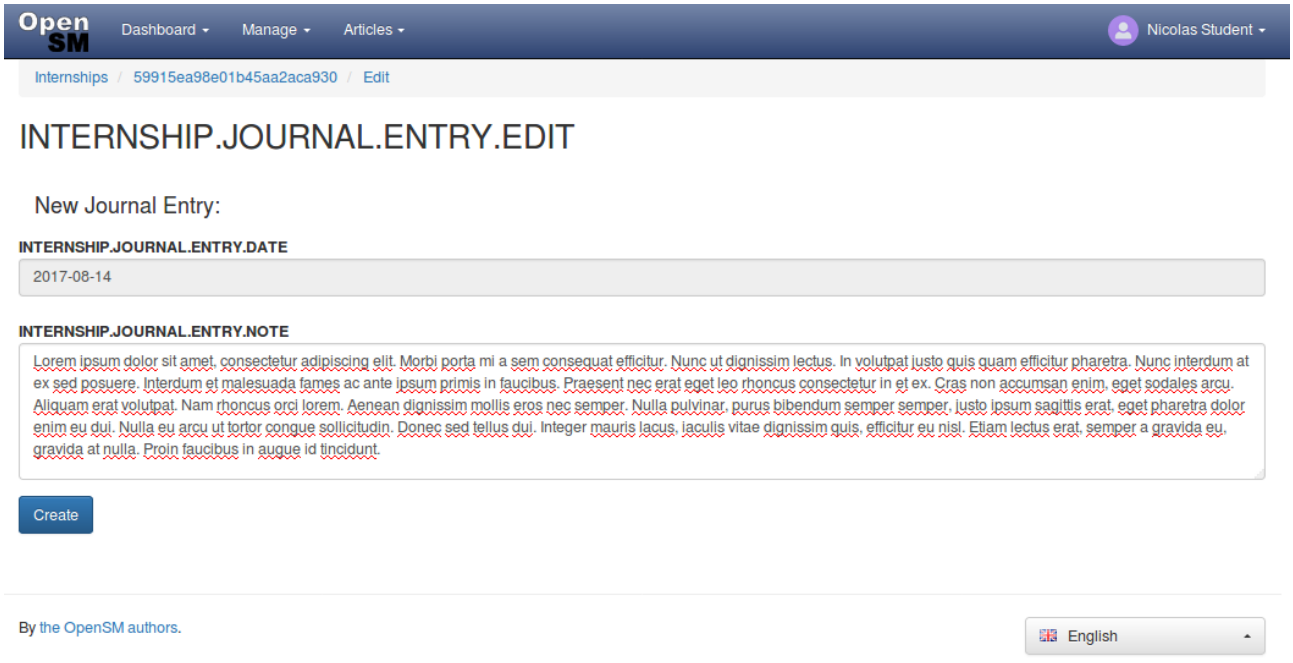


Illustration 24: la date est automatiquement ajoutée à chaque entrée du journal de bord de l'élève

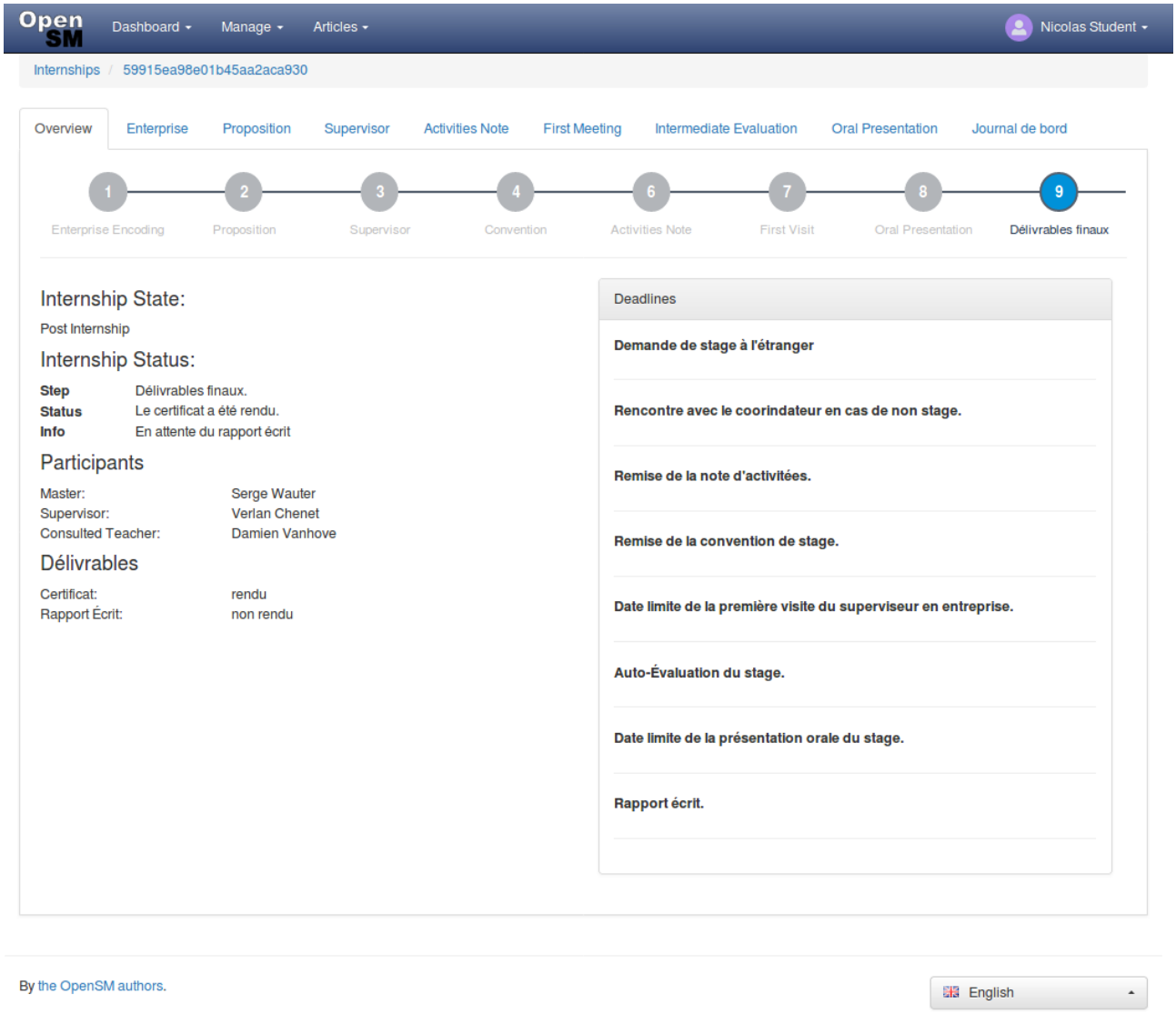


Illustration25: Overview final du stage, en attente des délivrables.

7.1.2 *Interface utilisateur du maître de stage*

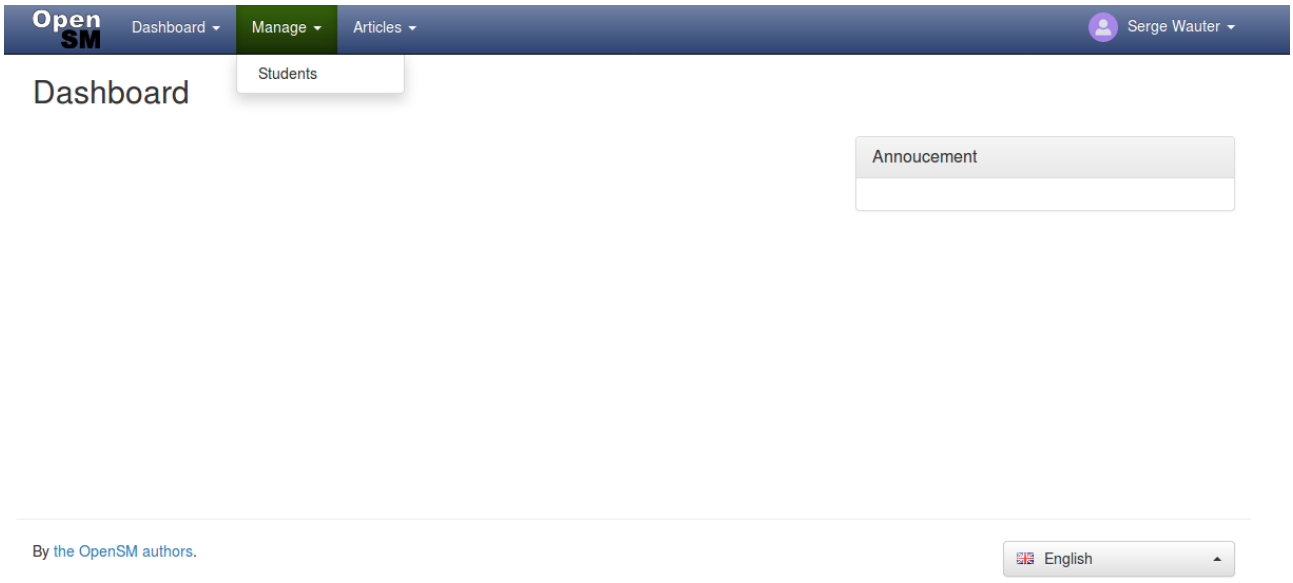


Illustration 1: le maître de stage accède à la liste des étudiants via le sous-menu "students"

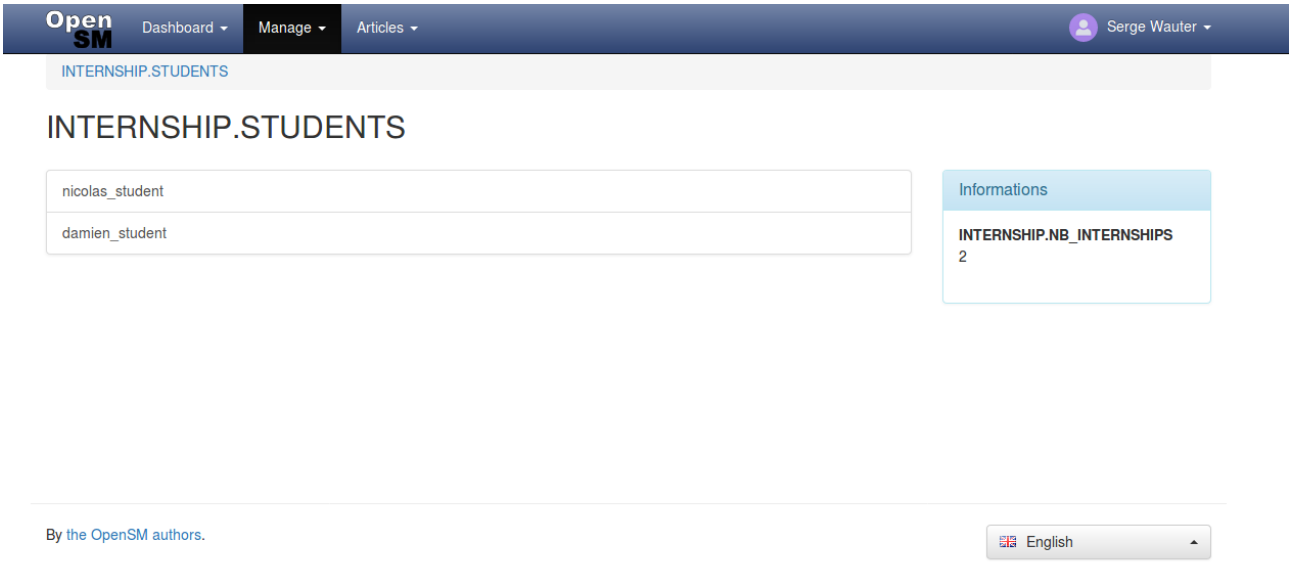


Illustration 2: liste des élèves que suit le maître de stage

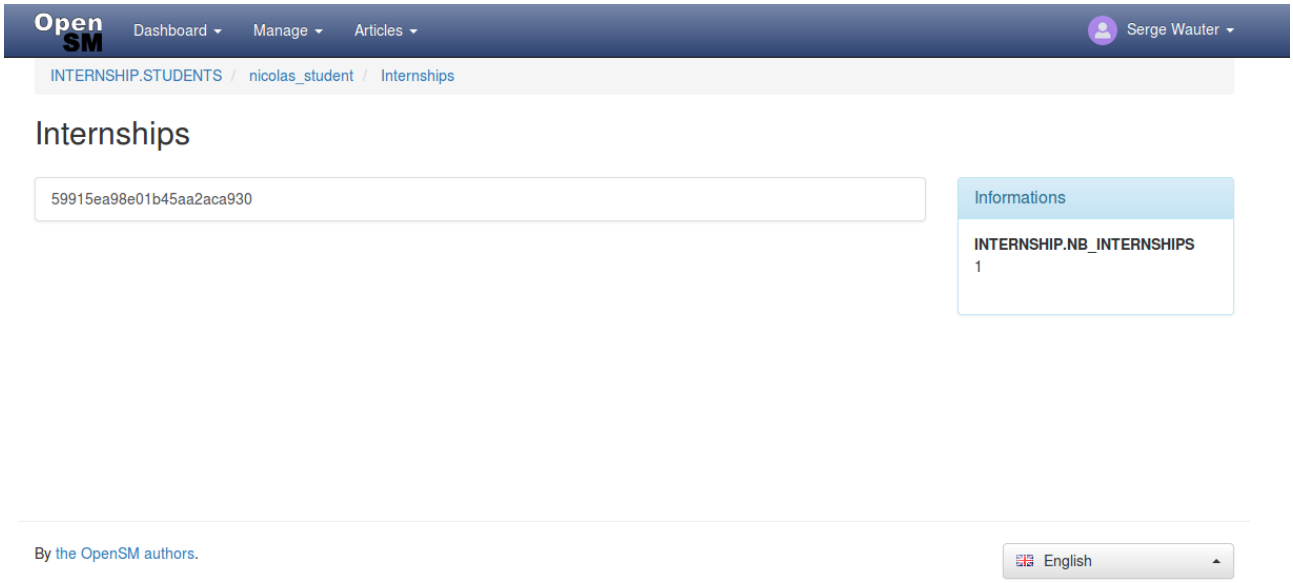


Illustration 3: en cliquant sur l'élève, le maître de stage peut accéder à ses stages

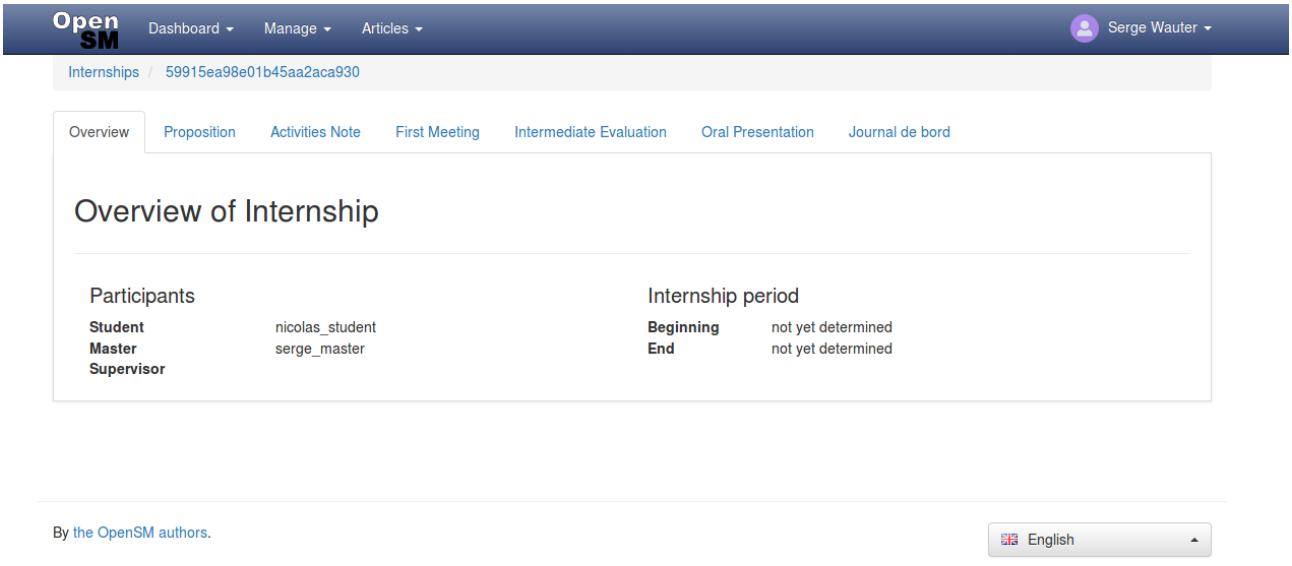


Illustration 4: overview du stage d'un élève

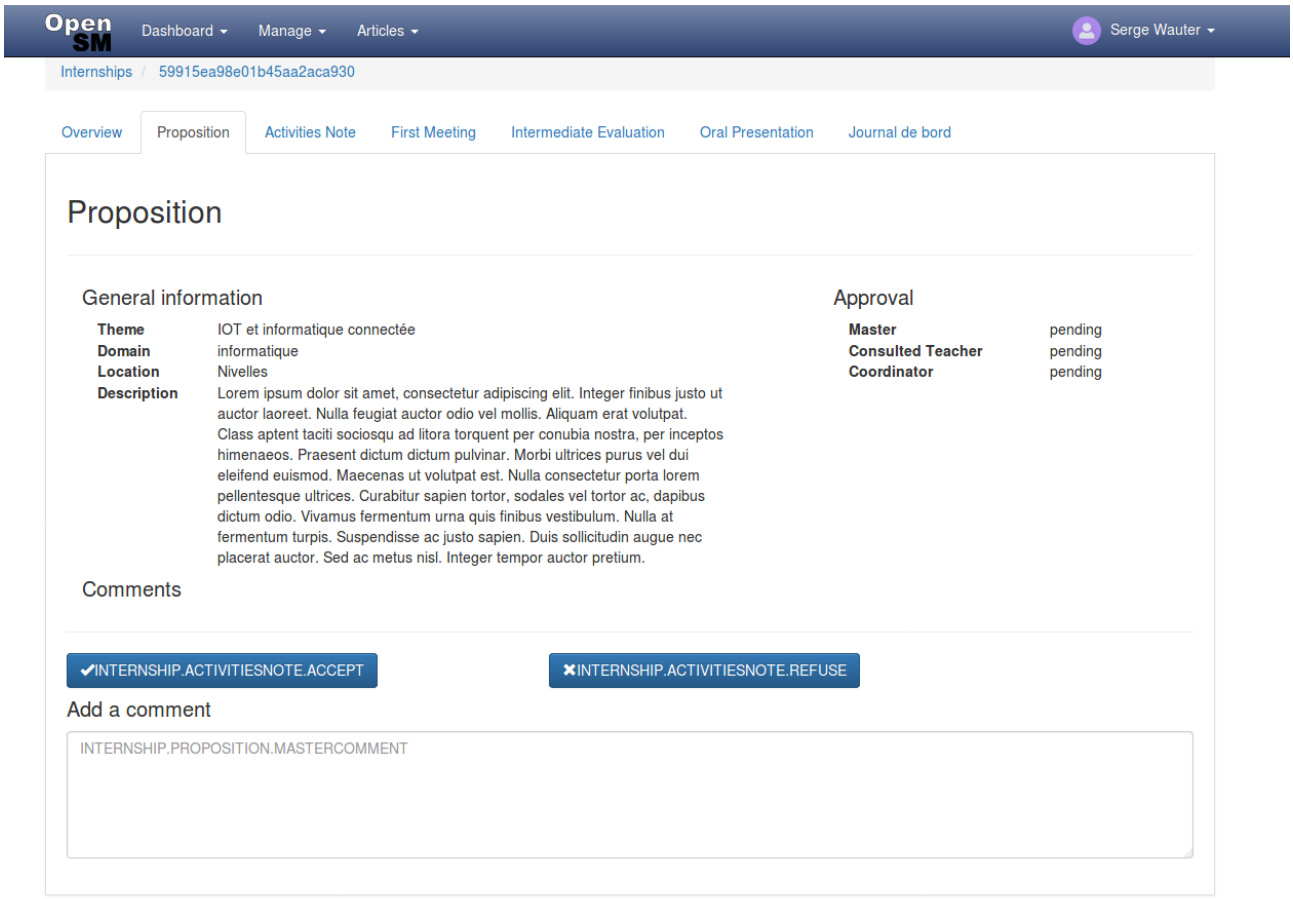


Illustration 5: l'onglet Proposition permet au maître de stage de consulter la proposition de stage

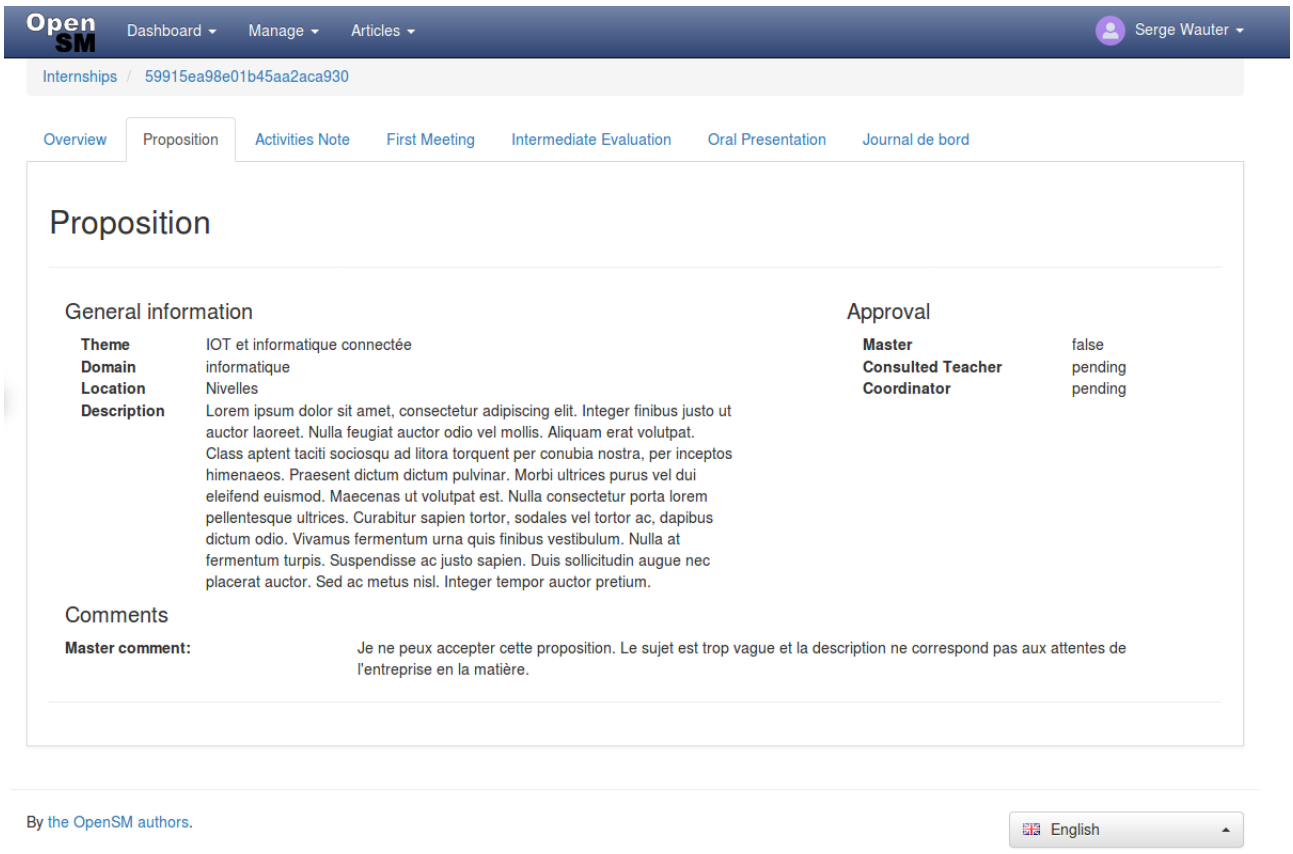


Illustration 6: Les champs d'inputs ainsi que les boutons disparaissent une fois que le aître de stage a donné ou non son accord

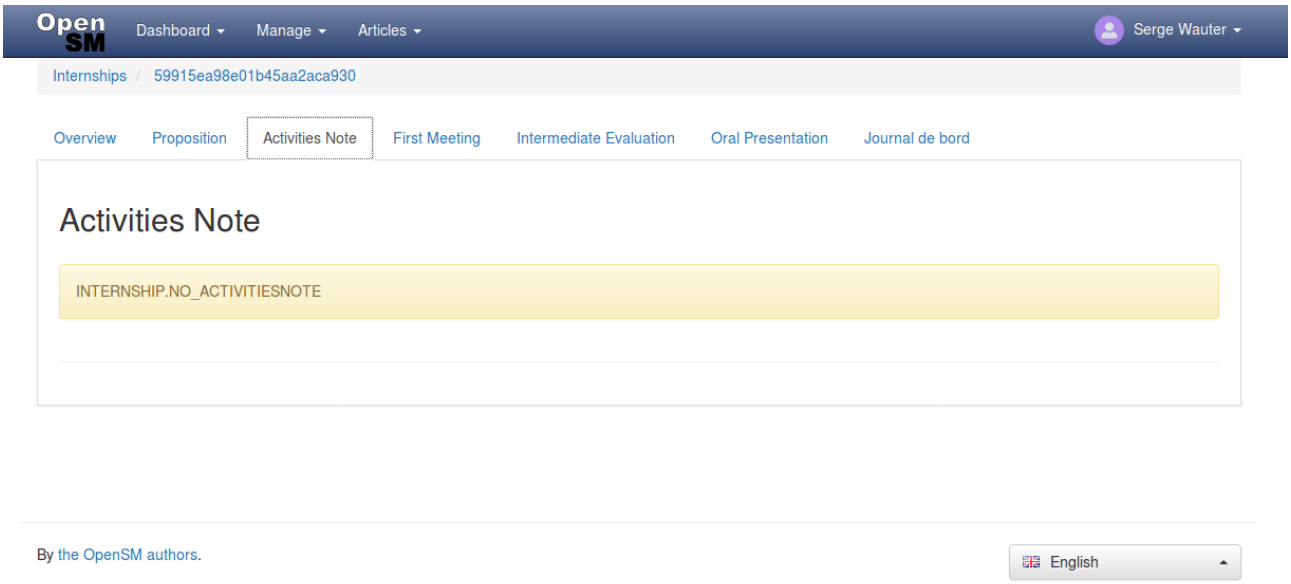


Illustration 7: Si l'élève n'a pas encore encodé sa note d'activités, il n'y a rien à montrer au maître de stage

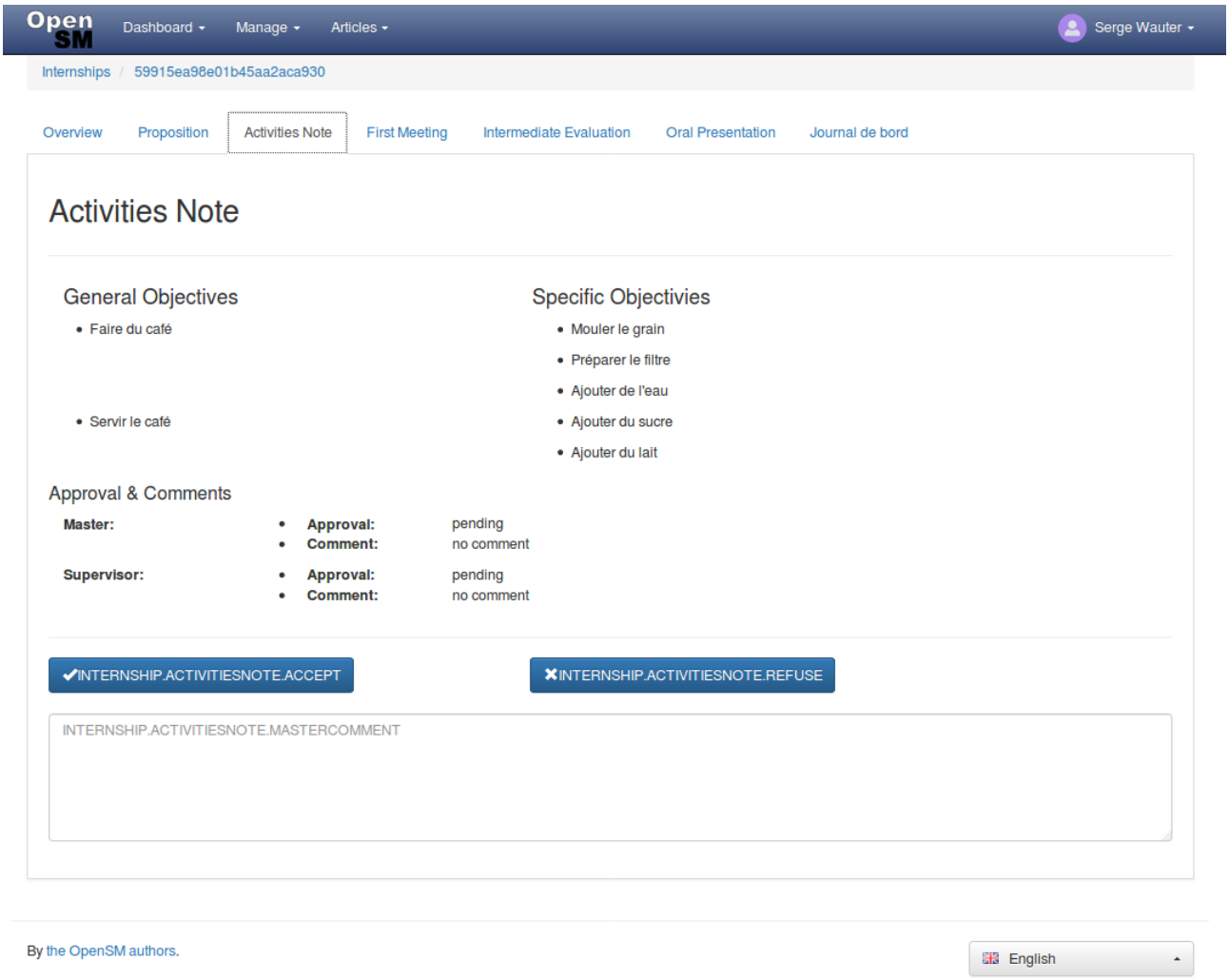


Illustration 8: la validation de la note d'activités se fait de manière similaire à la proposition de stage

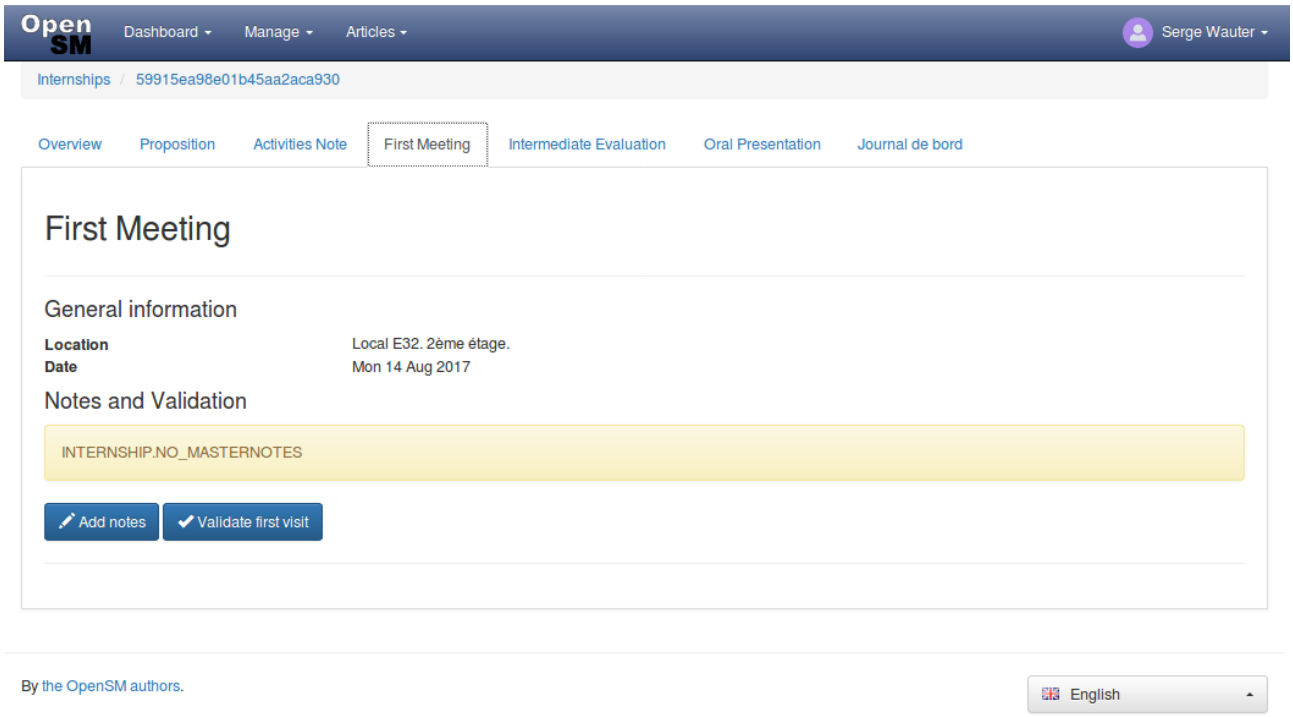


Illustration 9: Onglet first meeting avant la validation ou rajout de notes personnelles

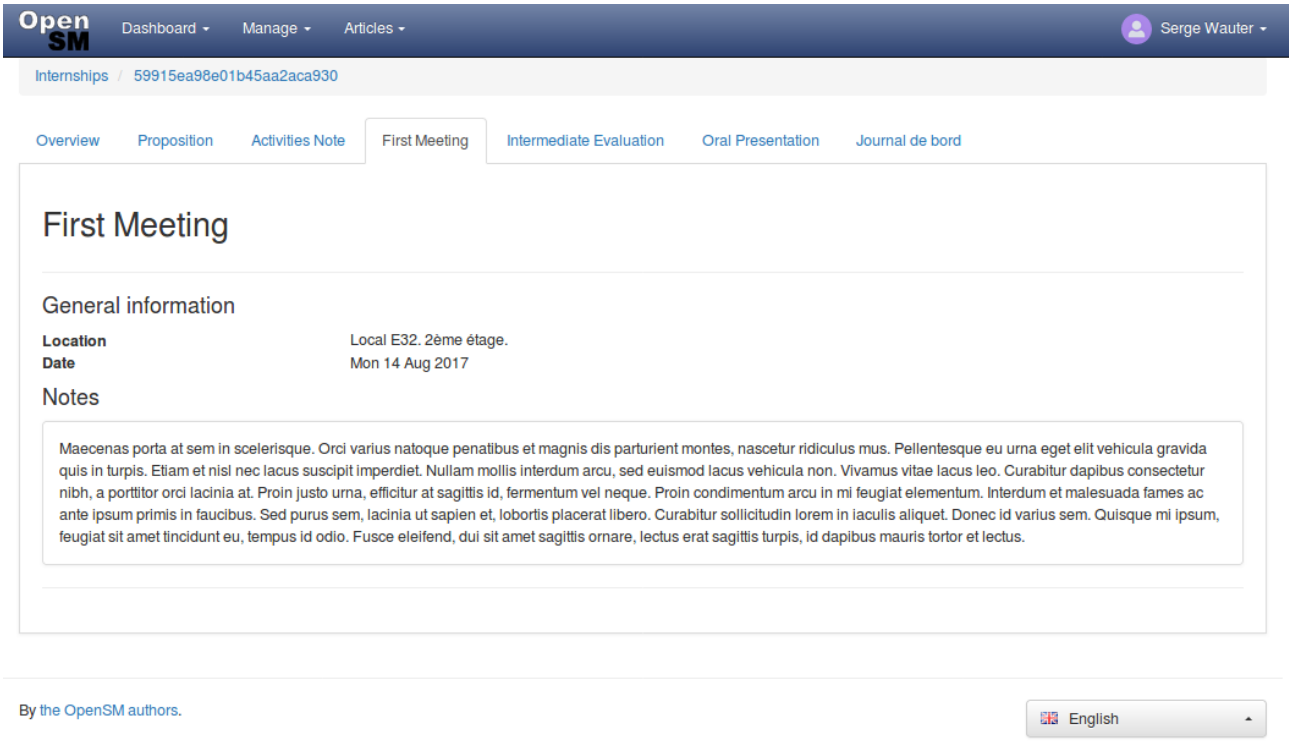


Illustration 10: Onglet first meeting après validation du maître de stage

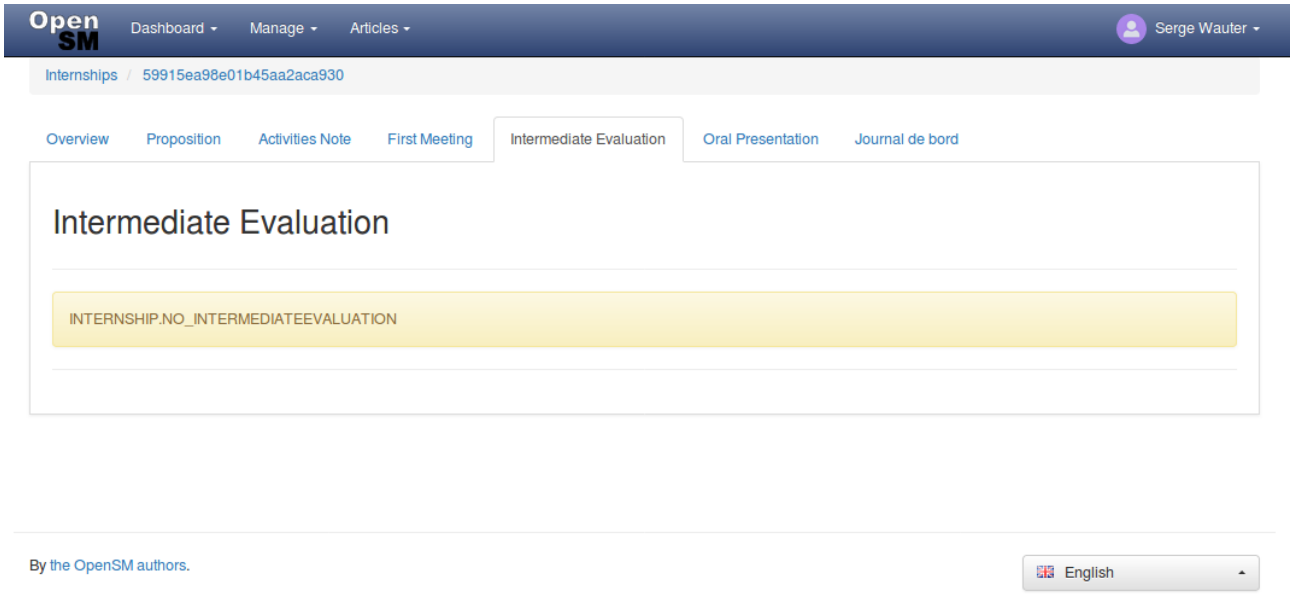


Illustration 11: Si l'élève n'a pas encodé de lieu et de date, le maître de stage ne peut pas modifier l'évaluation intermédiaire. Les formulaires et interfaces pour les onglets First Meeting, Intermediate Evaluation et Oral Presentation sont identiques, et ne sont donc pas tous repris.

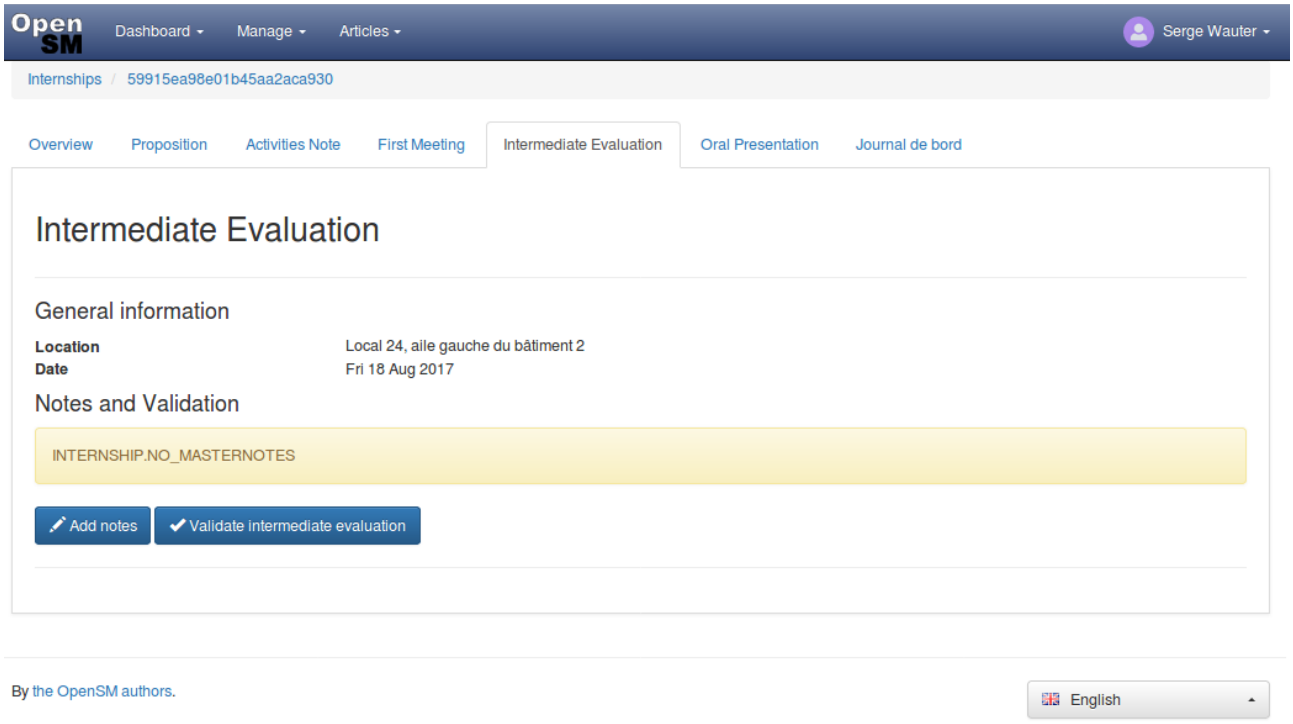


Illustration 12: Une fois la date et lieux encodés, le maître de stage peut valider ou ajouter des notes personnelles

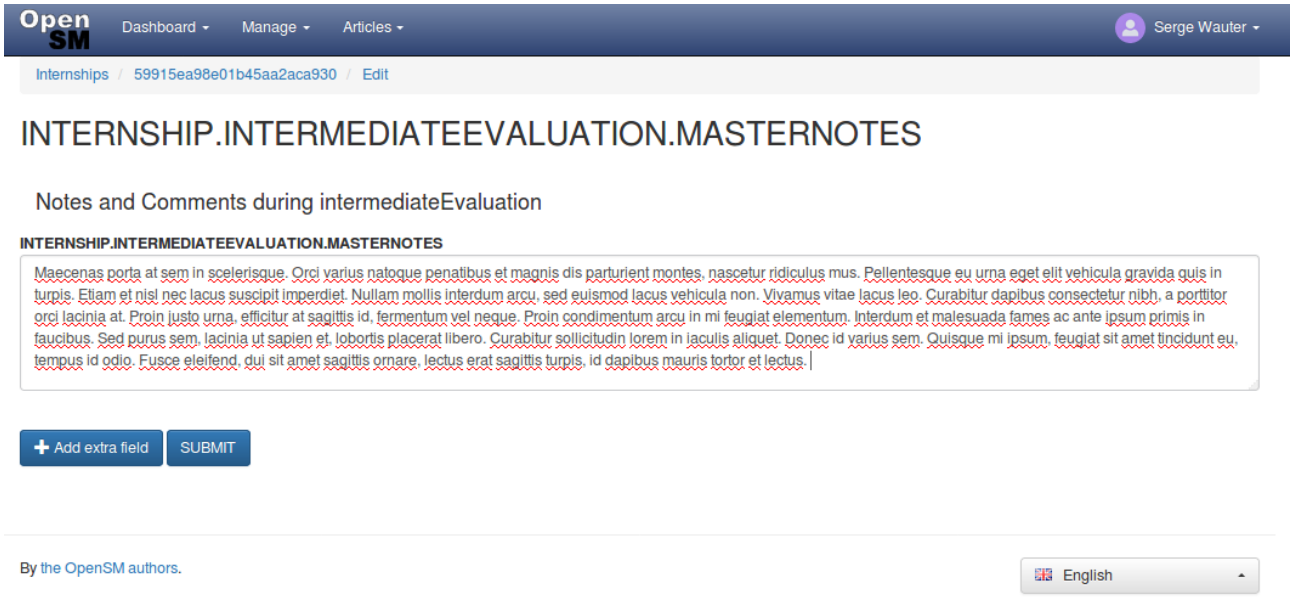


Illustration 13: Ajout et modifications des notes personnelles du maître de stage

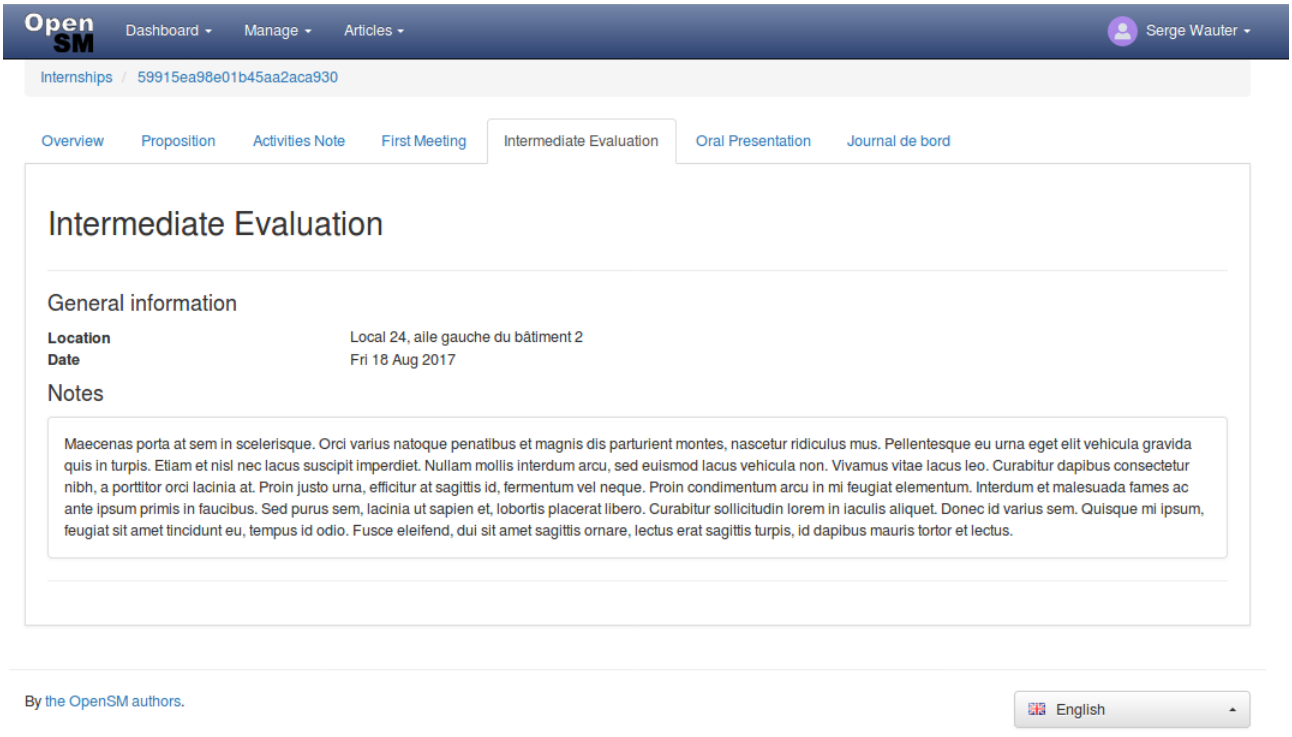


Illustration 14: après validation, le maître de stage ne peut plus modifier ses notes

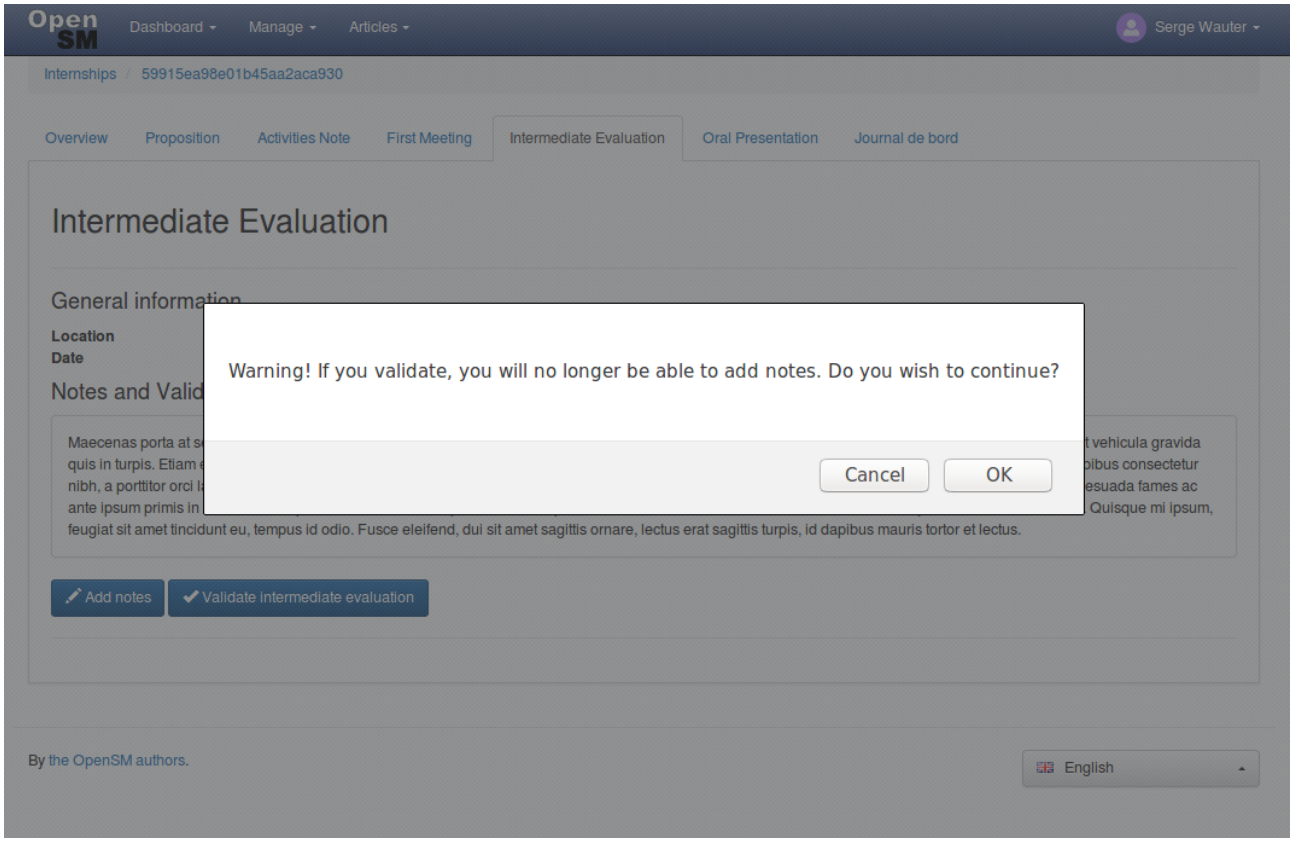


Illustration 15: demande de confirmation avant la validation de l'évaluation intermédiaire

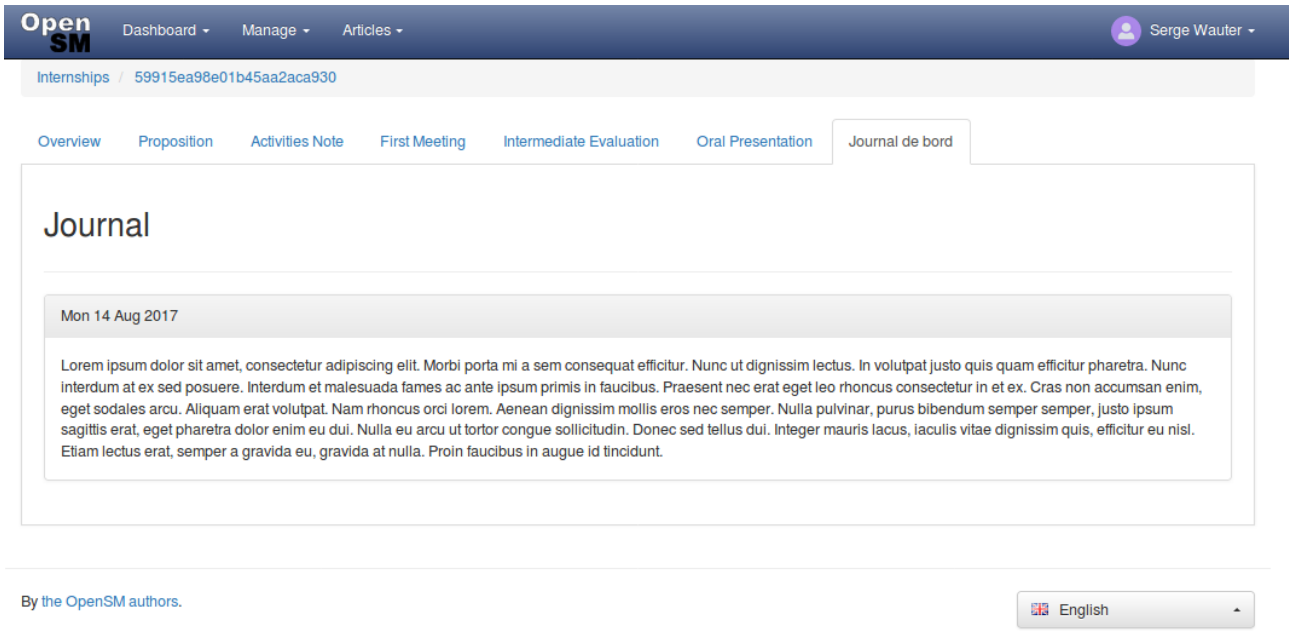


Illustration 16: L'onglet Journal permet de consulter le journal de bord de l'étudiant

7.1.3 Interface utilisateur de l'enseignant/superviseur

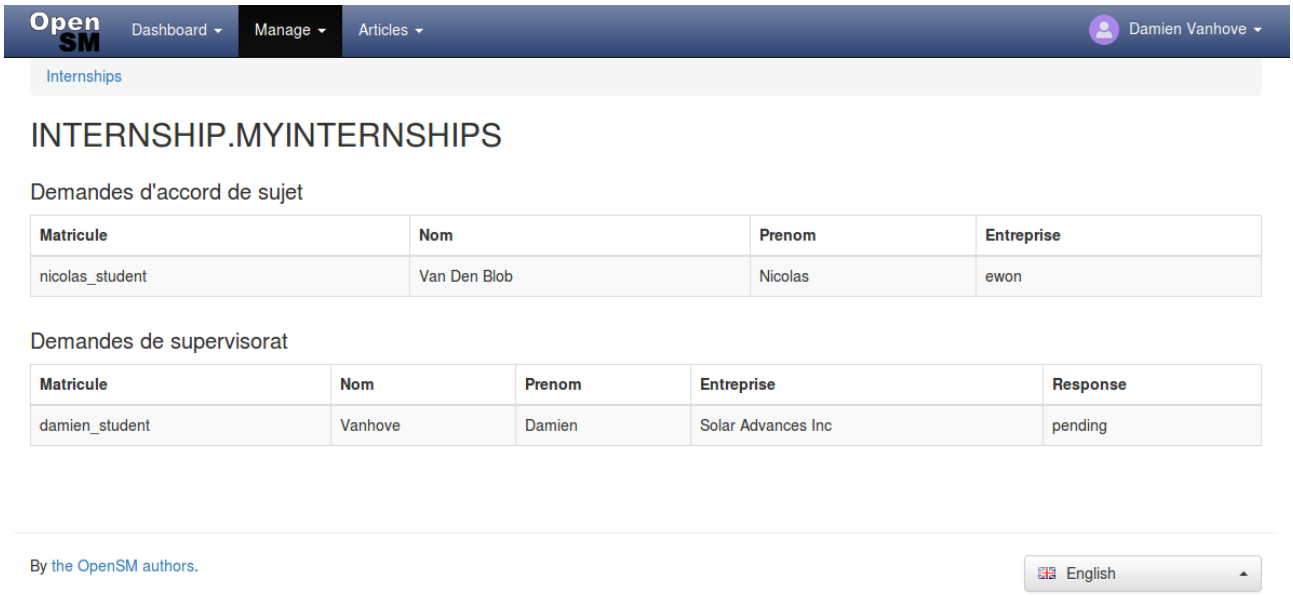


Illustration 1: Liste des stages que l'enseignant doit valider et liste des demandes de supervisorat

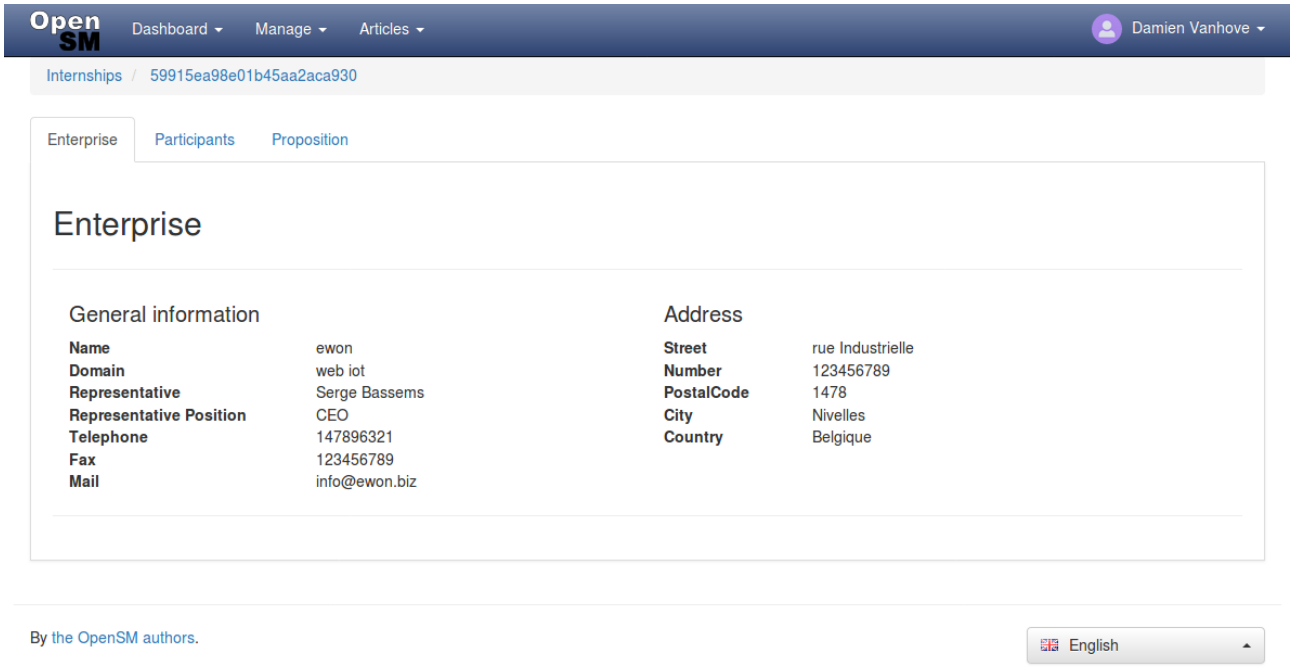


Illustration 2: si l'enseignant doit uniquement valider la proposition, il n'a accès qu'à trois onglets d'information

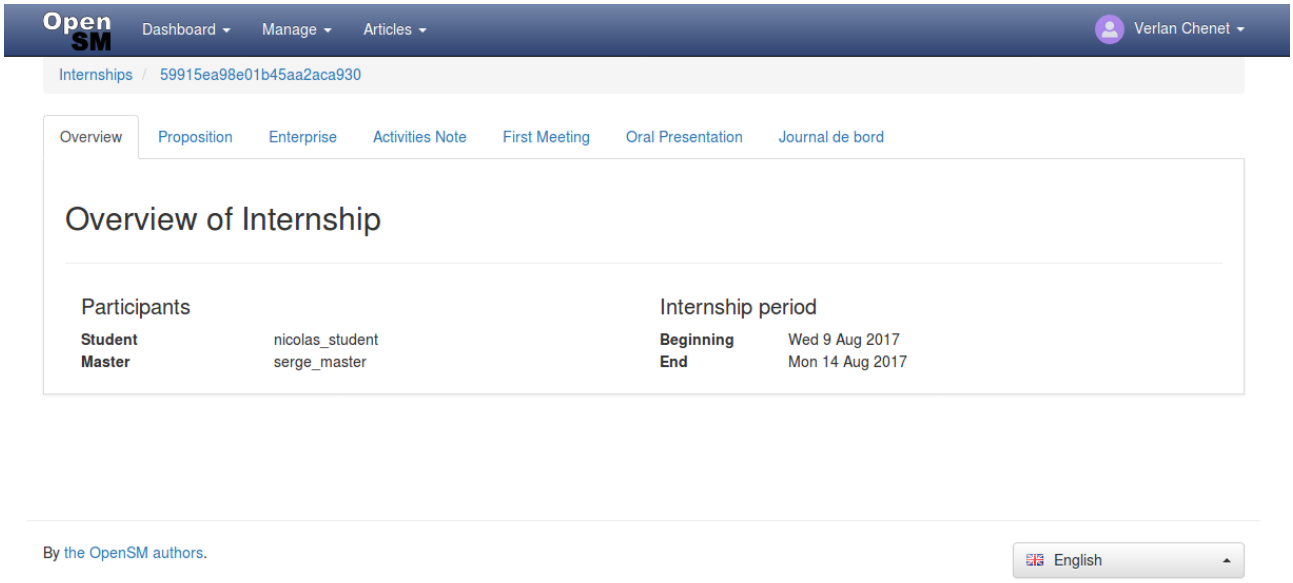


Illustration 3: si l'enseignant est superviseur de stage, il a accès à toutes les informations du stage. Les formulaires et informations reprises dans les onglets sont les mêmes que pour le maître de stage, les illustrations ne seront donc pas répétées.

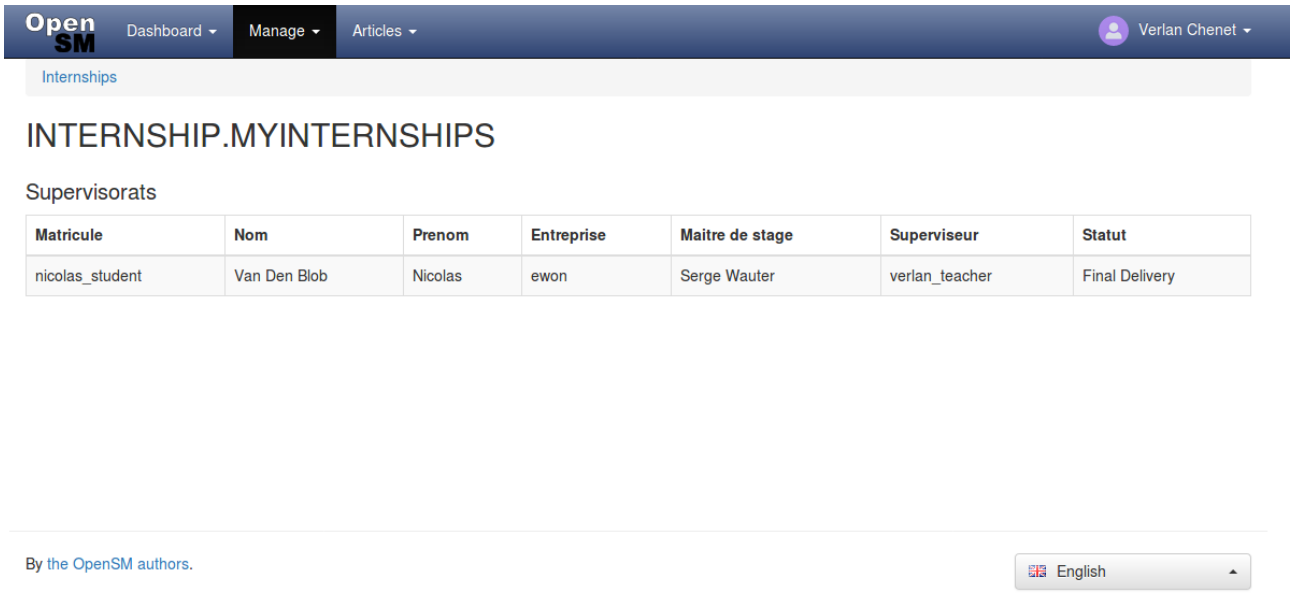


Illustration 4: les stages pour lesquels l'enseignant est superviseur sont repris dans la liste "Supervisorats". Les listes vides sont cachées.

7.1.4 Interface utilisateur de l'administrateur

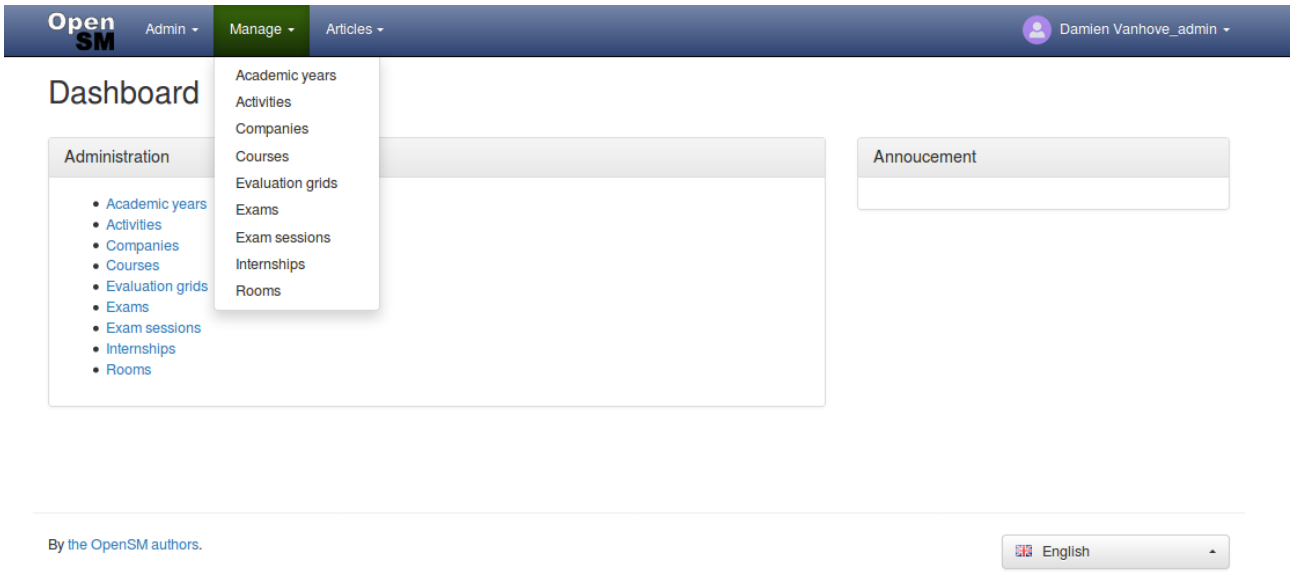


Illustration 1: le sous-menu internships permet d'accéder à la gestion des stages

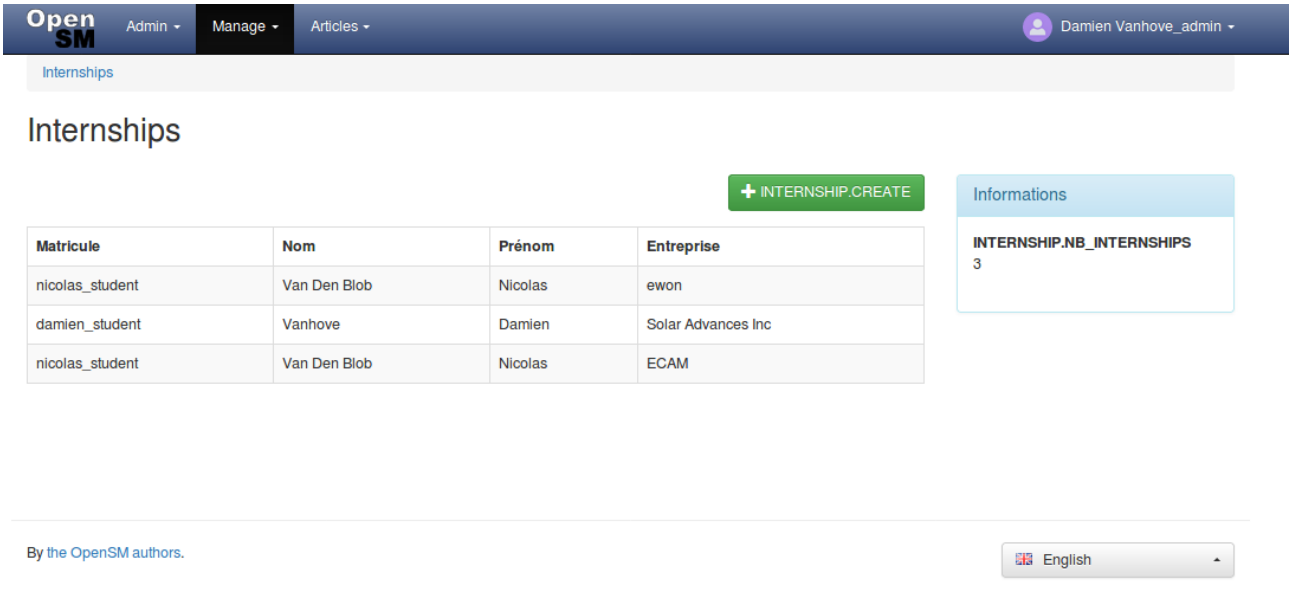


Illustration 2: liste de tous les stages de la plateforme. Le bouton create permet à l'administrateur d'en créer un nouveau.

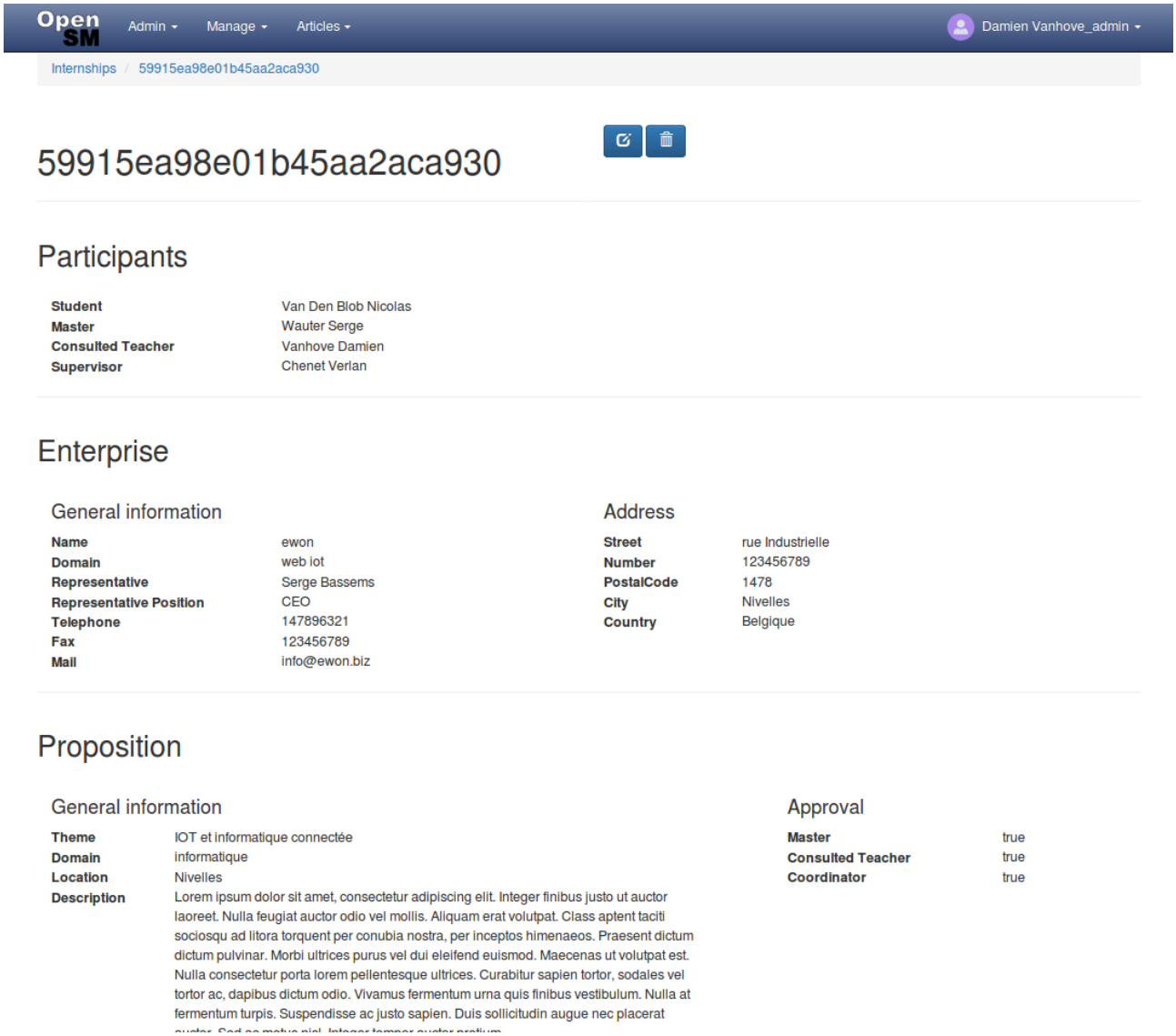


Illustration 3: Fiche récapitulative des informations d'un stage

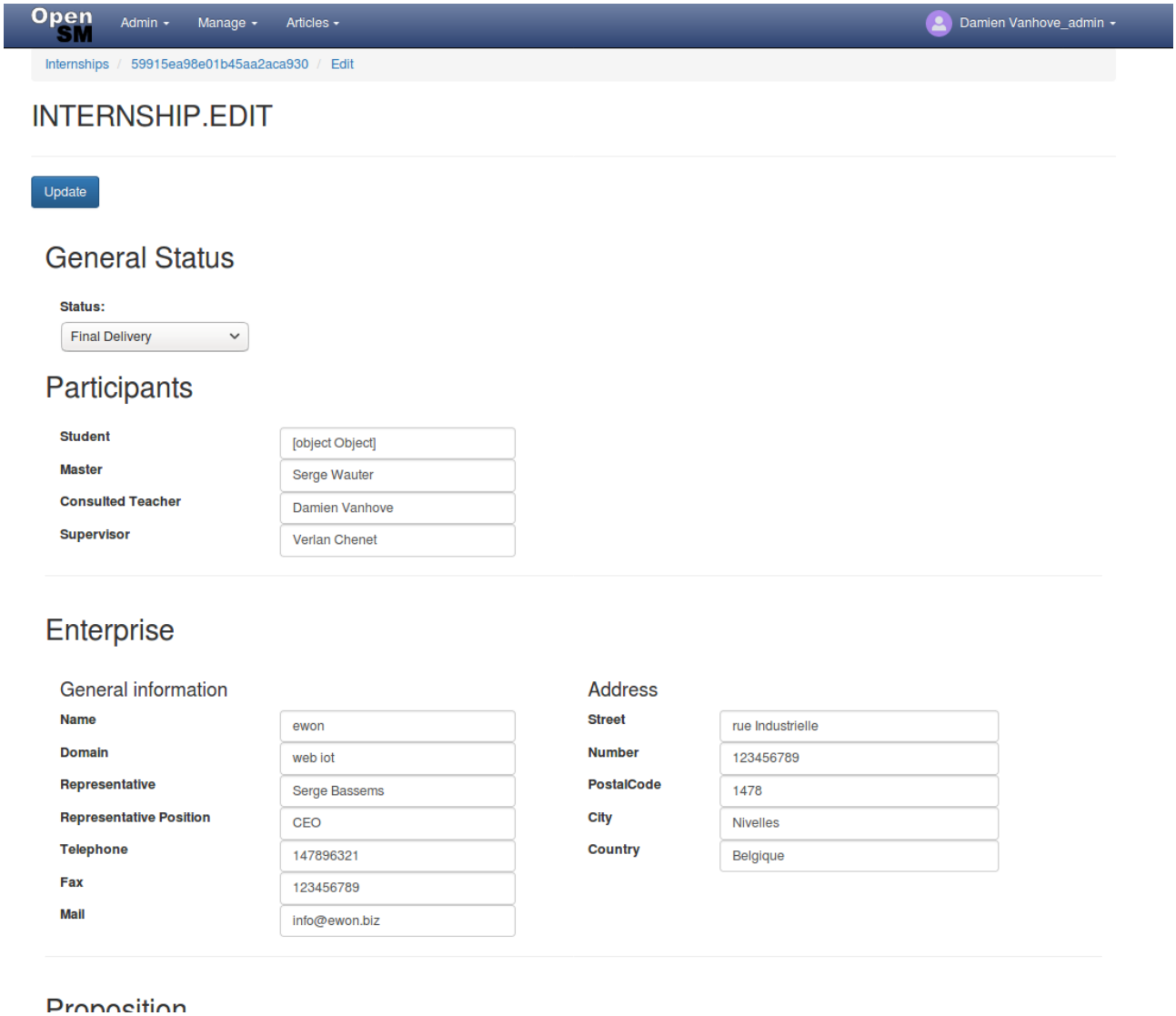


Illustration 3: le formulaire de l'administrateur permet de modifier n'importe quelle élément d'un stage

7.1.5 Internface utilisateur du validateur

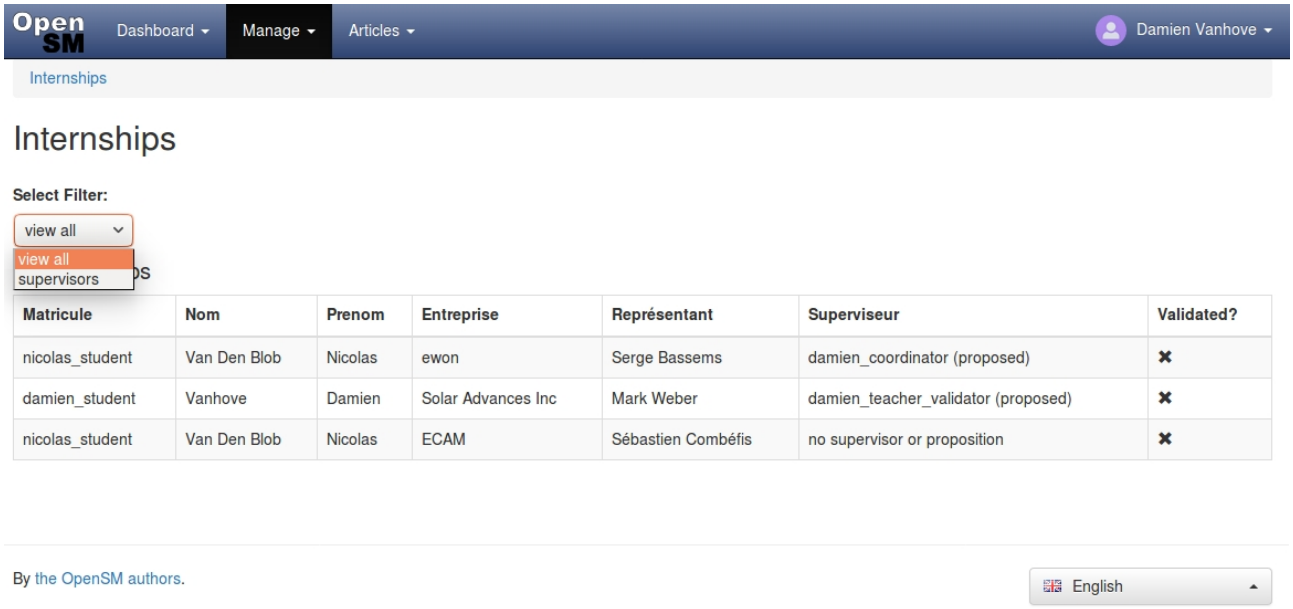


Illustration 1: le fitre par défaut est le filtre "view all" qui liste l'ensemble des stages

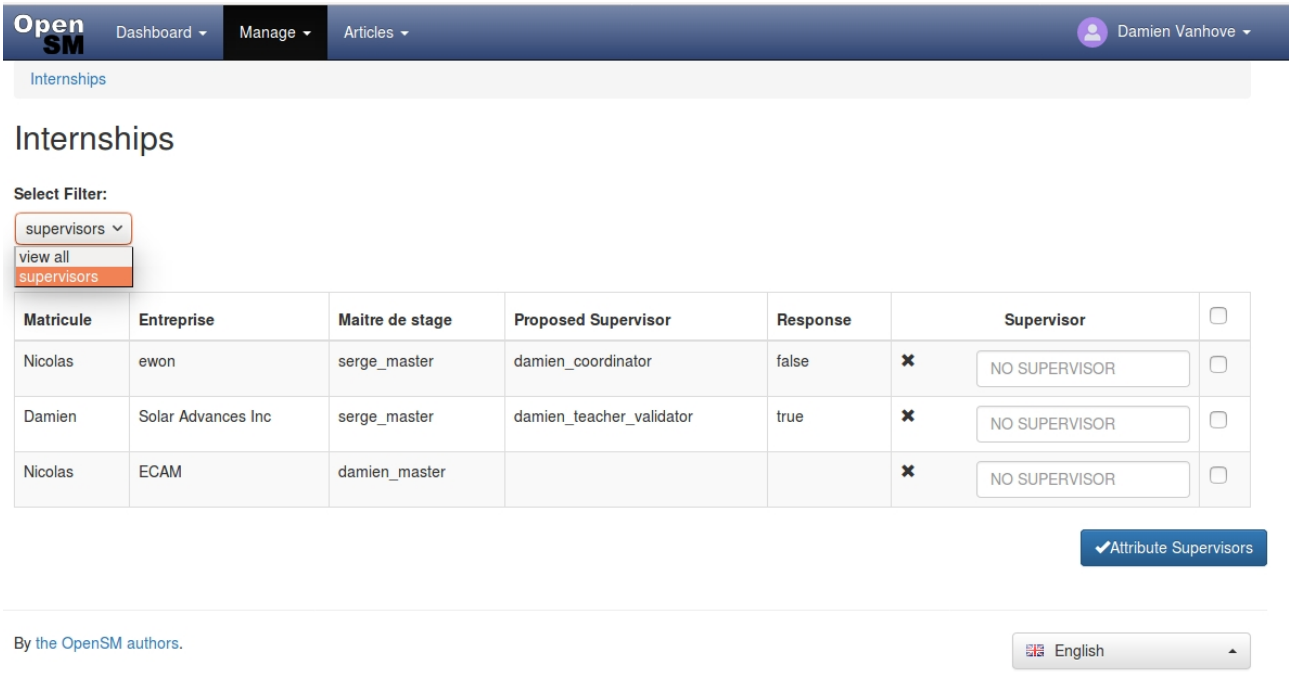


Illustration 2: le filtre "supervisors" permet d'attribuer les superviseurs de stage

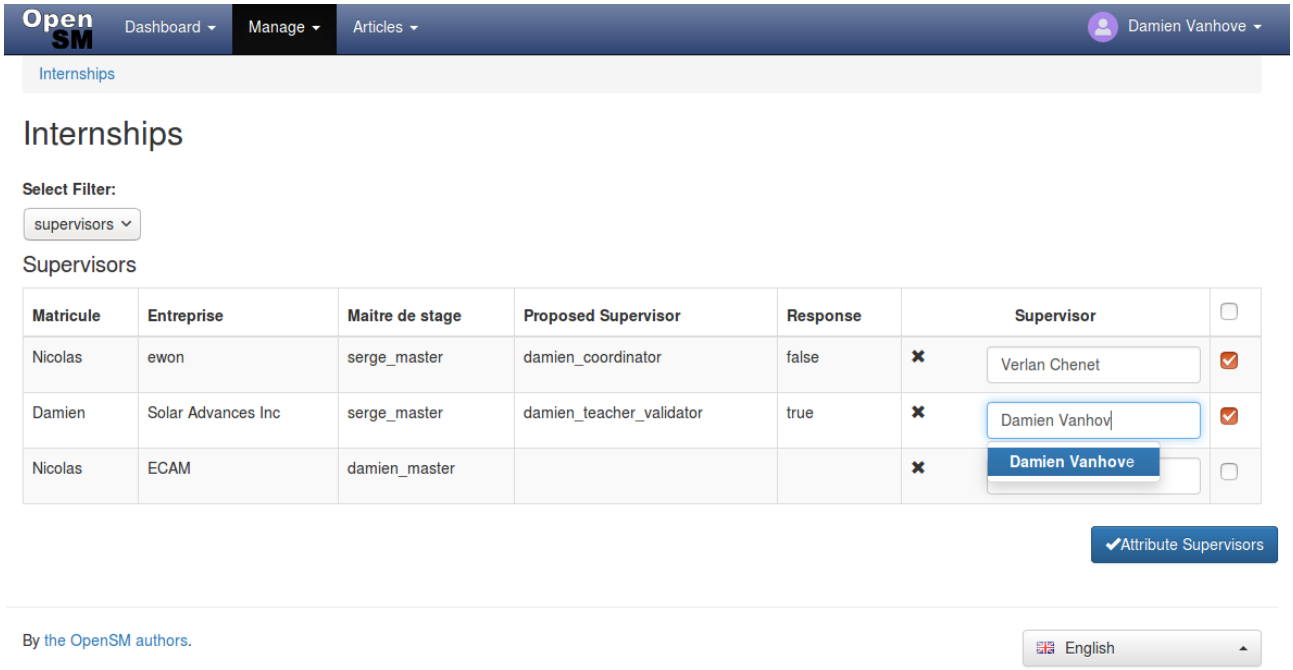


Illustration 3: autocomplétion des champs d'attribution des superviseurs

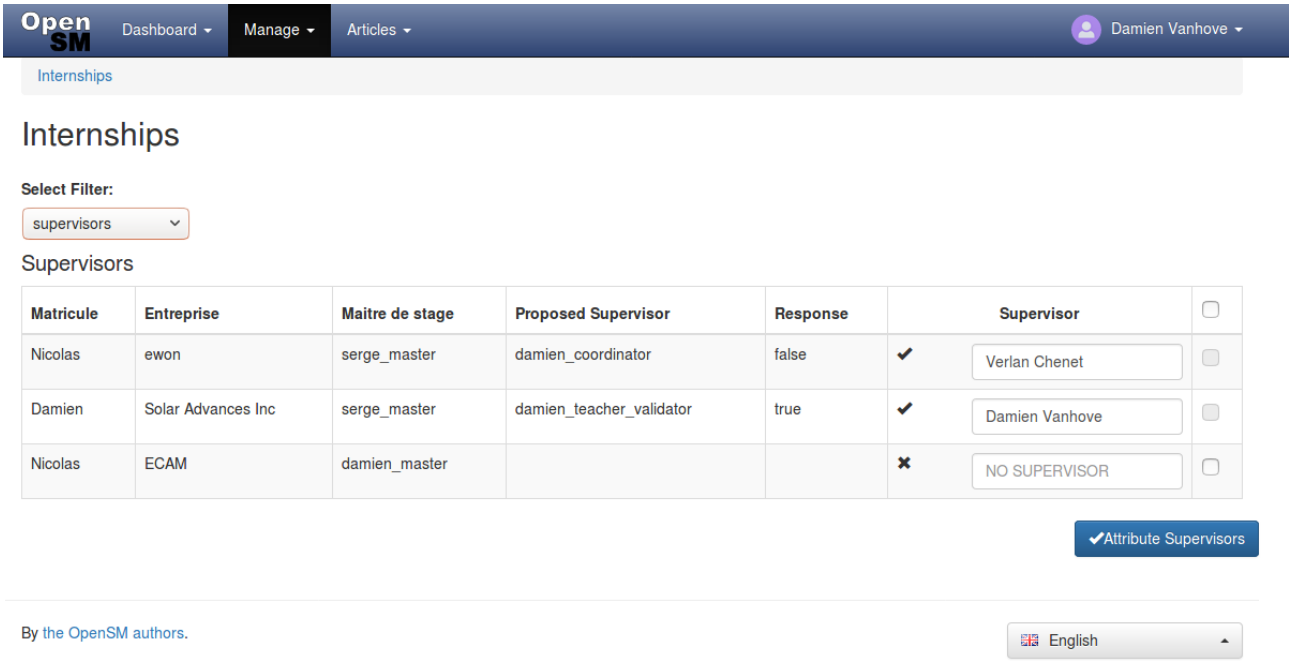


Illustration 4: liste des superviseurs après validation

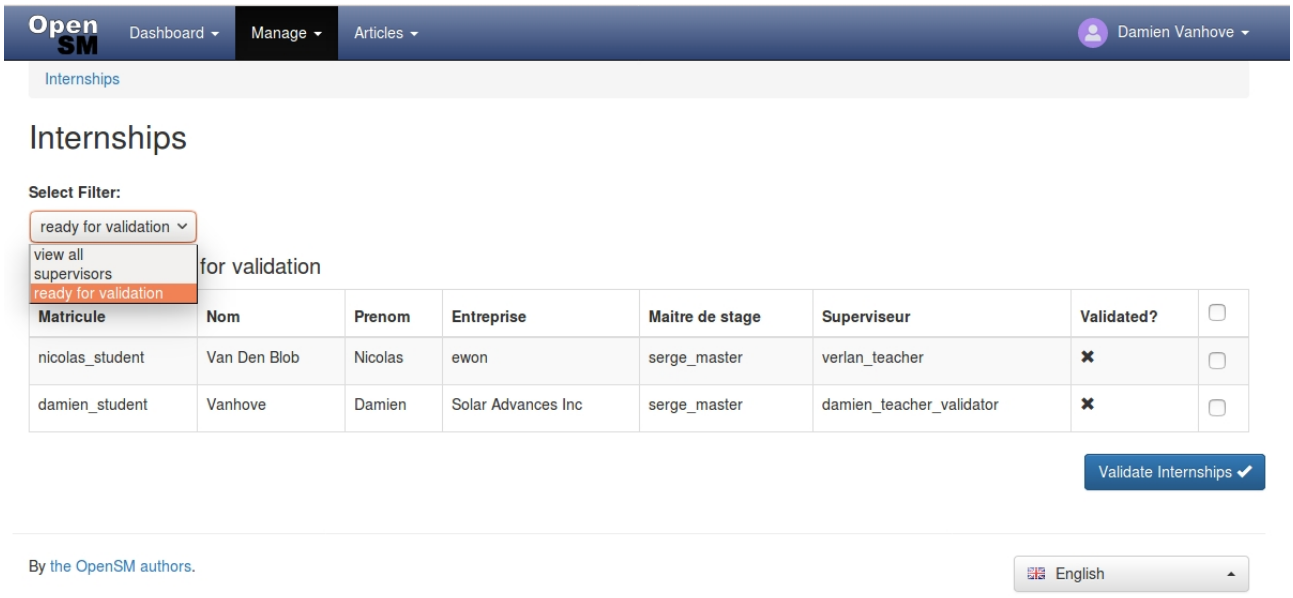


Illustration 5: si des stages sont prets à être validés par le validateur, un nouveau filtre apparaît.

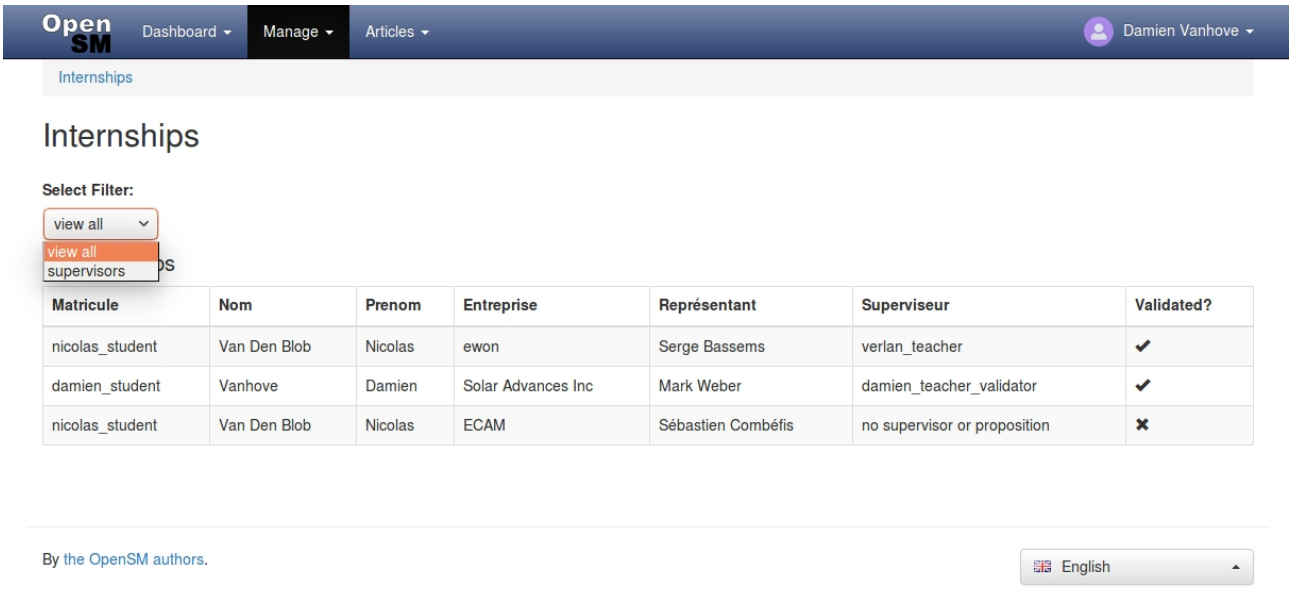


Illustration 6: Une fois les stages validés, la liste des filtres diminue

7.1.6 Interface utilisateur du manager

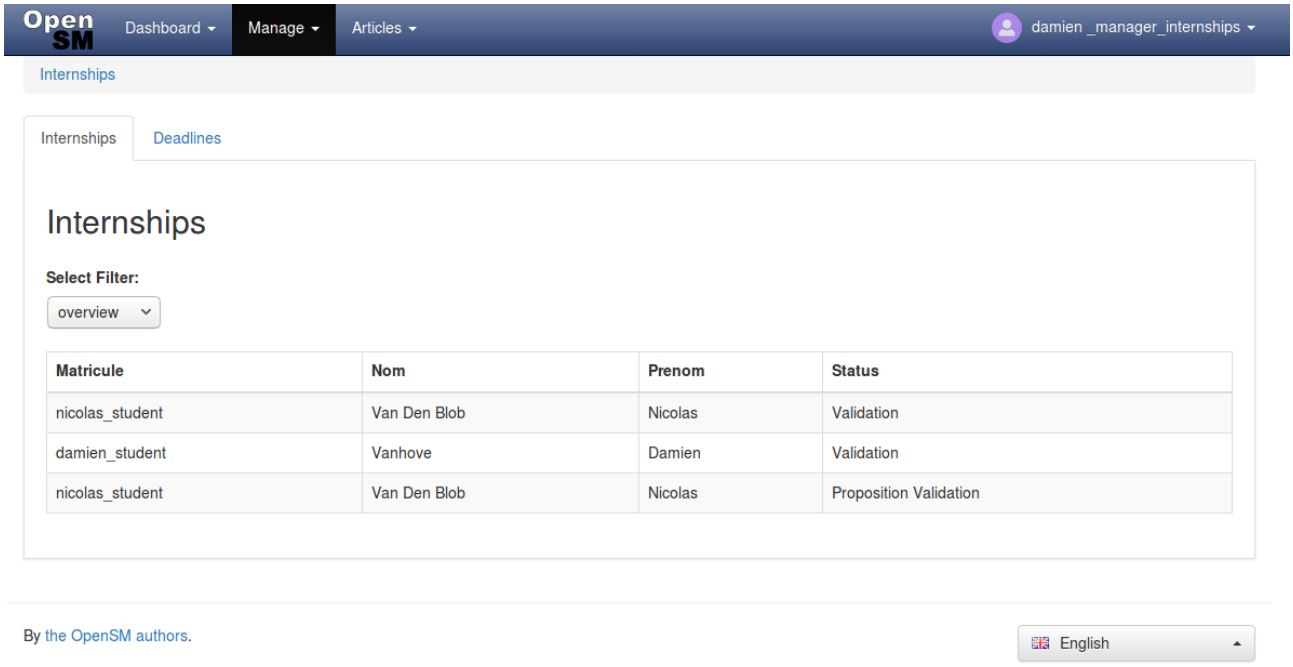


Illustration 1: overview de la liste des stages

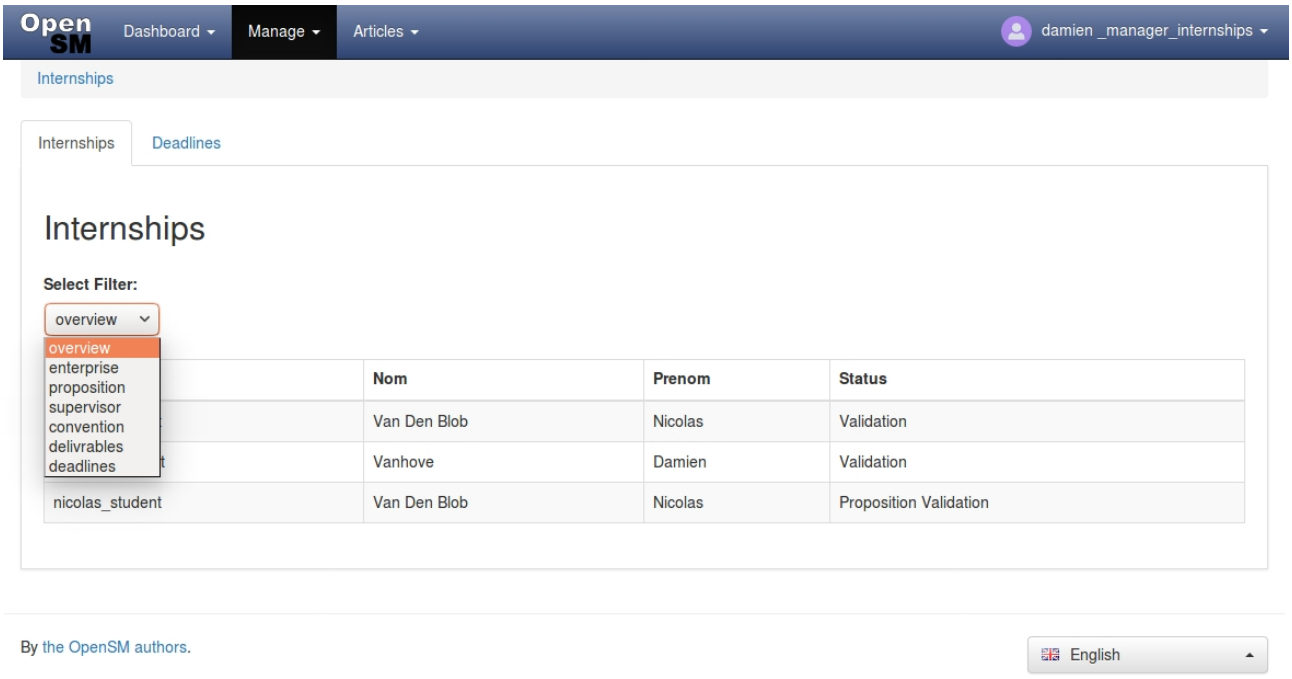


Illustration 2: liste des filtres dont dispose le manager pour gérer les stages

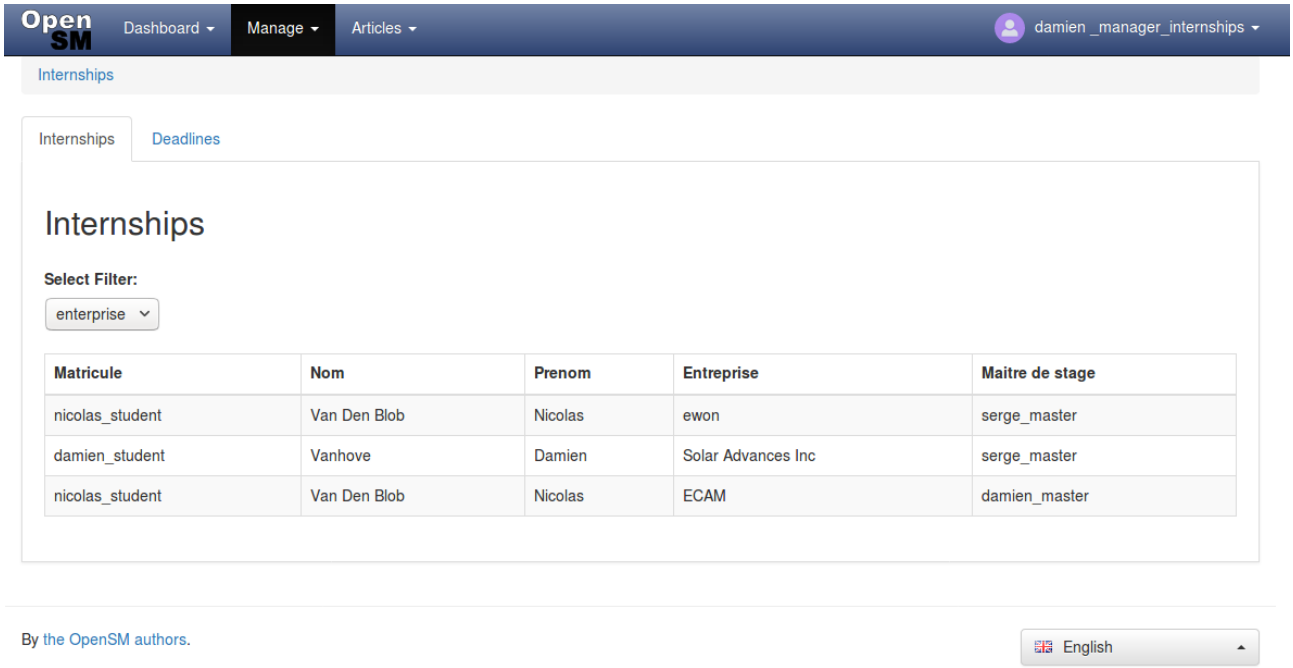


Illustration 3: application du filtre "enterprise"

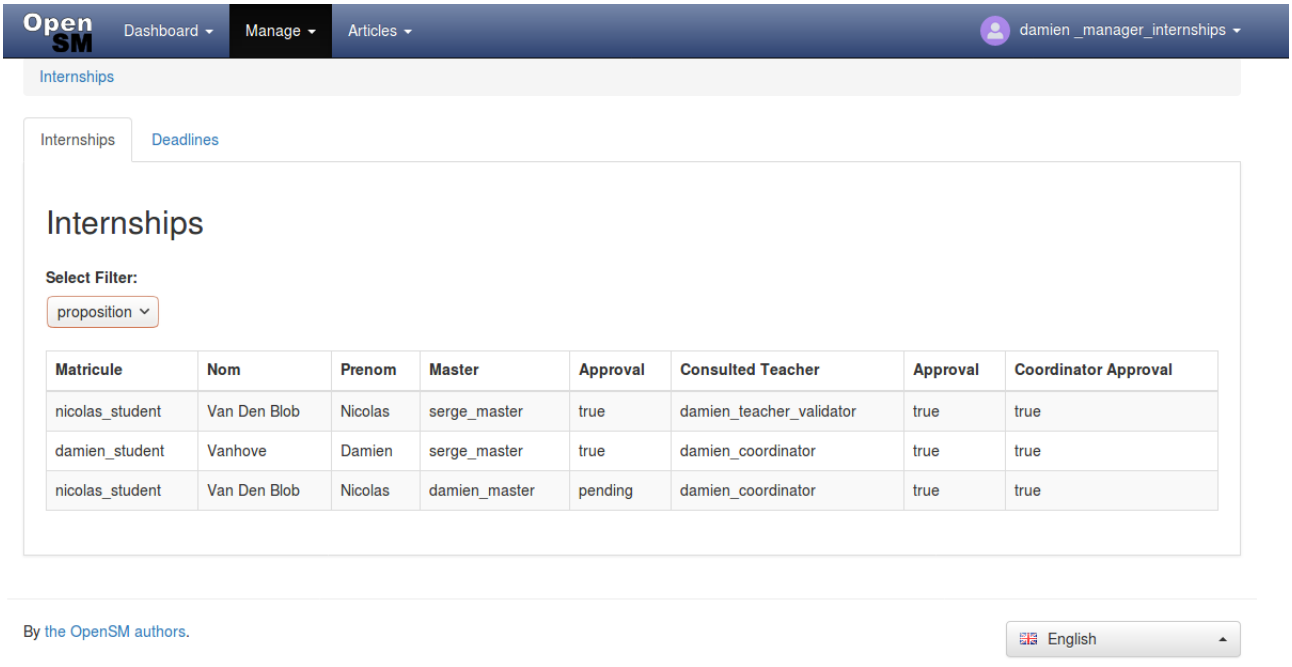


Illustration 4: le filtre "proposition" permet de voir si les propositions de stage ont toutes été validées

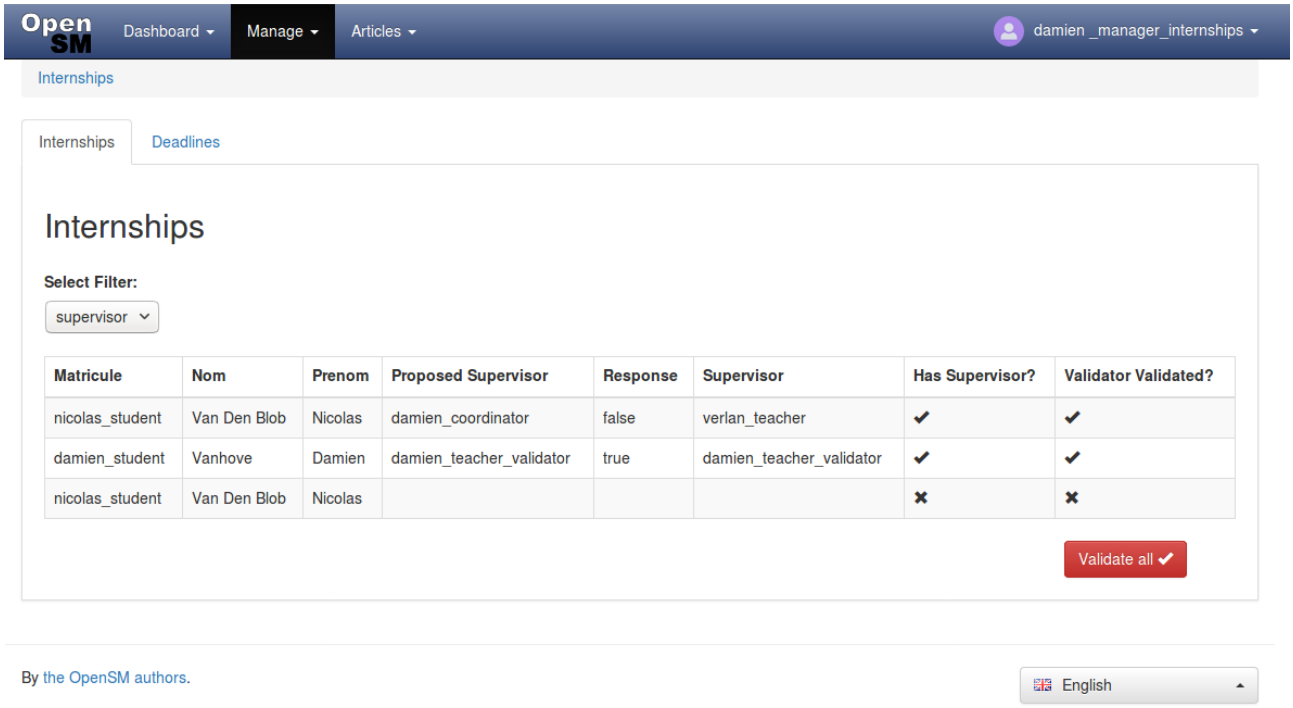


Illustration 5: le filtre "supervisor" permet de valider les superviseurs de tous les stages en une fois

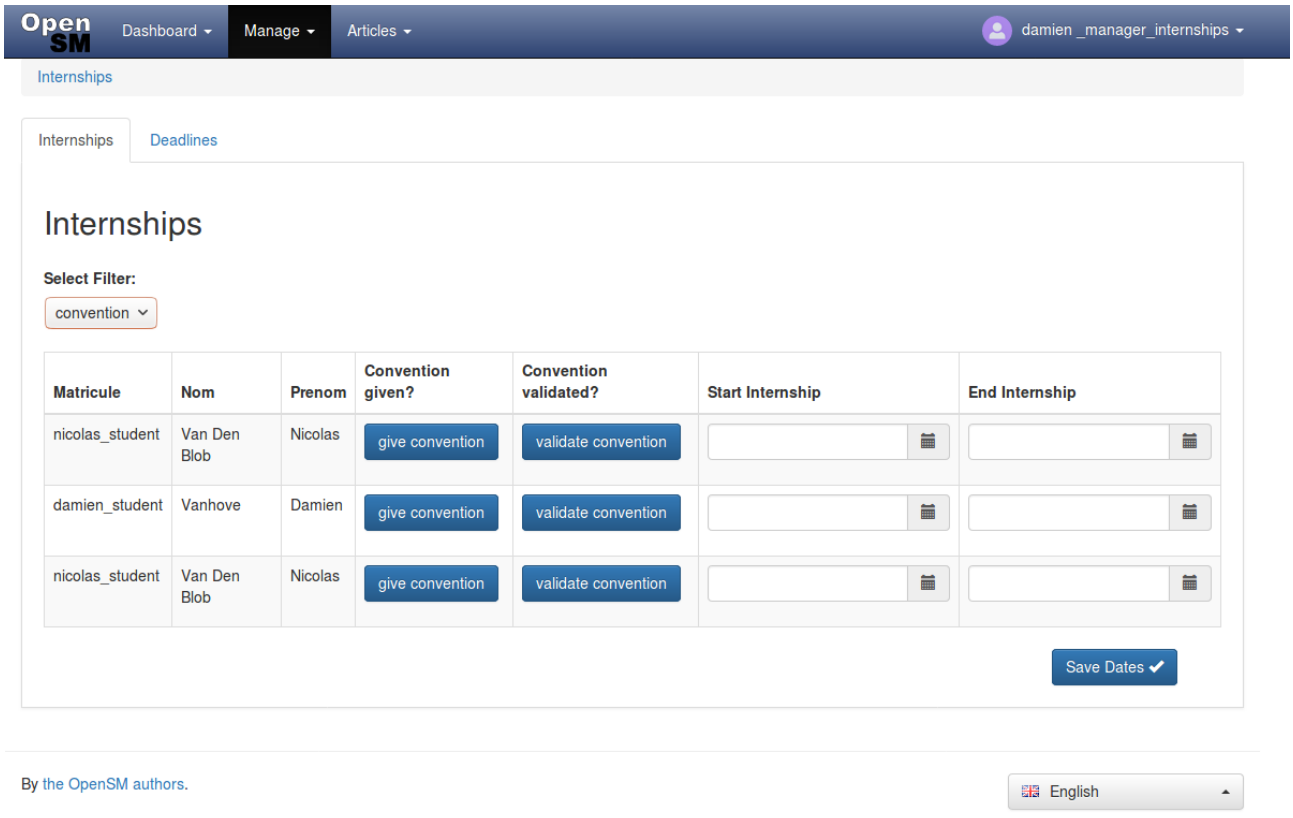


Illustration 6: application du le filtre "conventions"

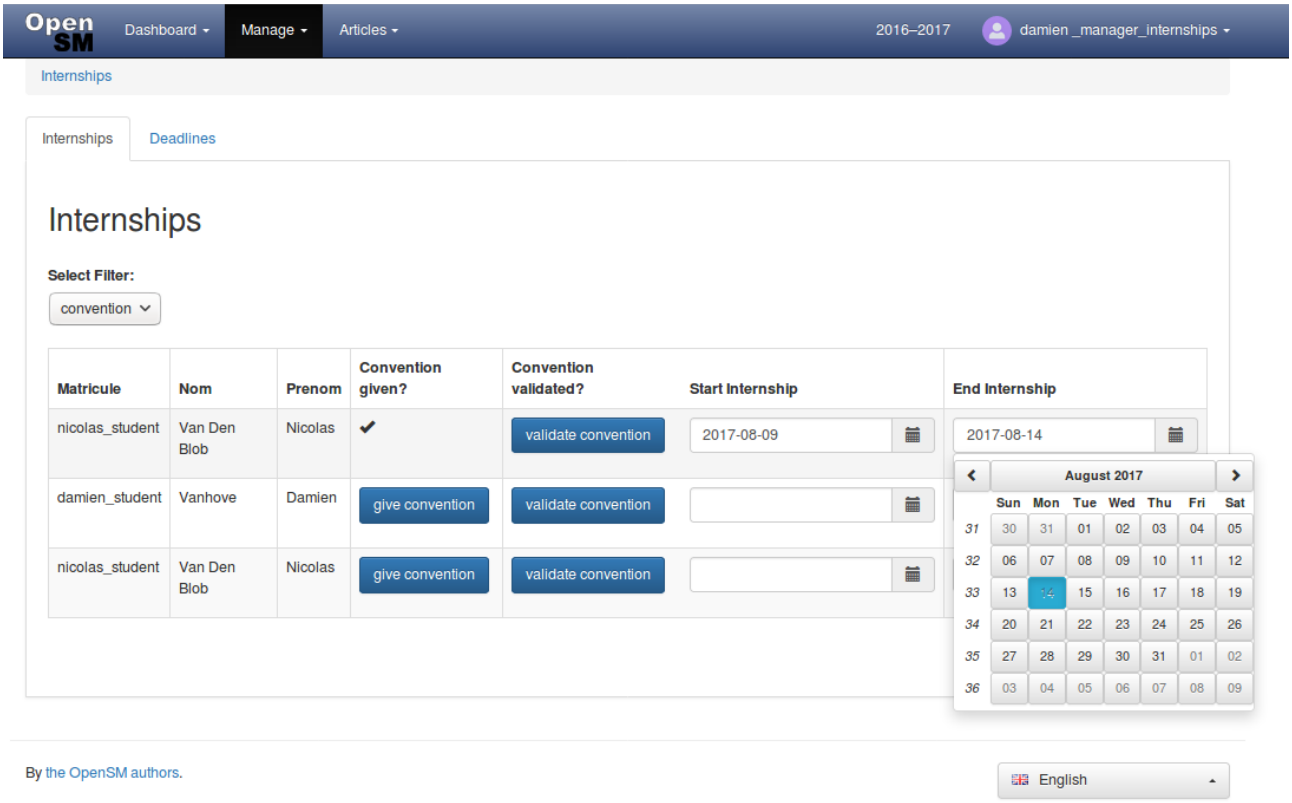


Illustration 7: validation de la sortie de la convention, encodage des dates de début et de fin d'un stage

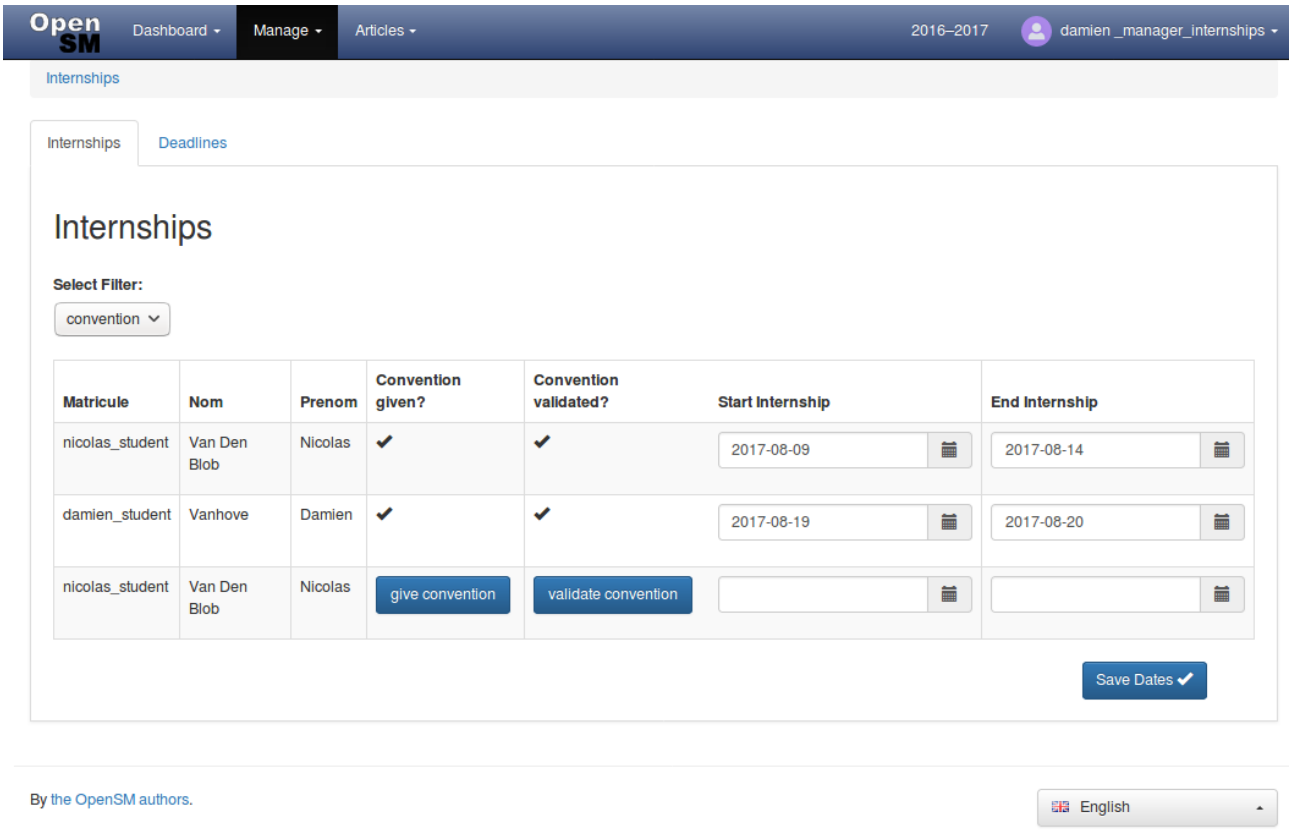


Illustration 8: validation de la convention et encodage des dates pour plusieurs stages

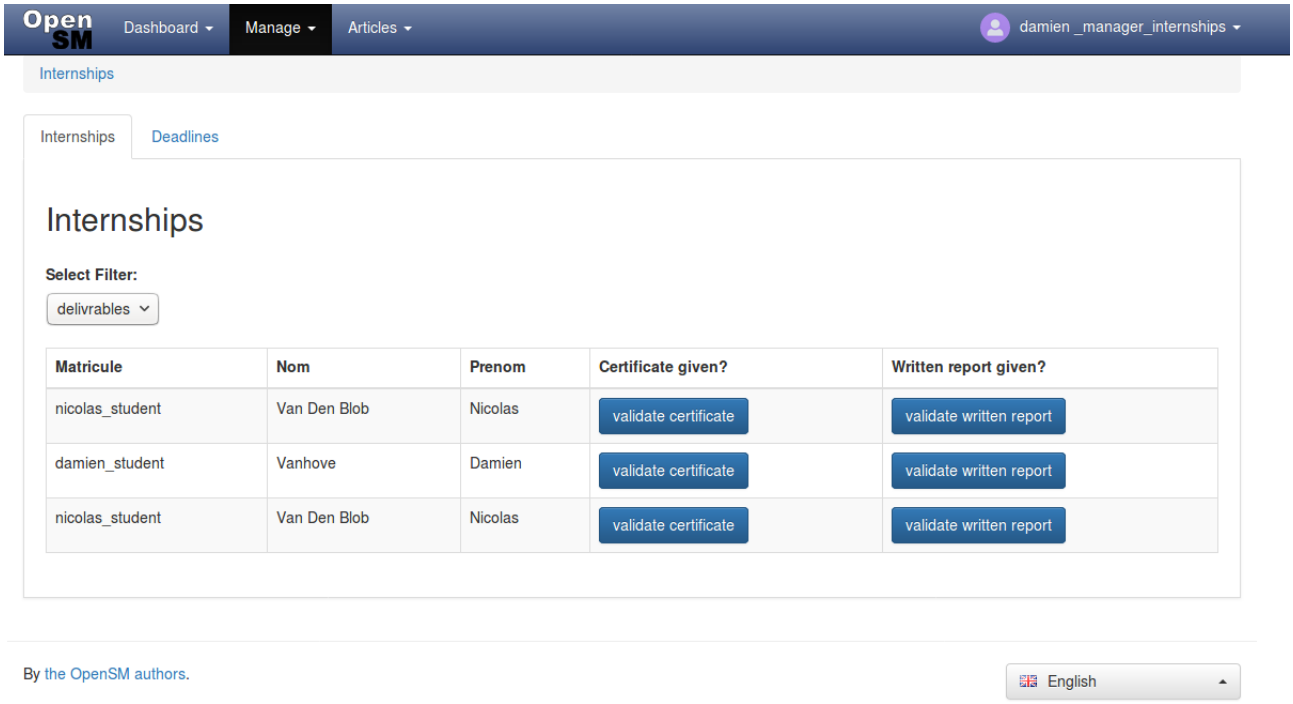


Illustration 9: gestion des délivrables au moyen du filtre "déliverables"

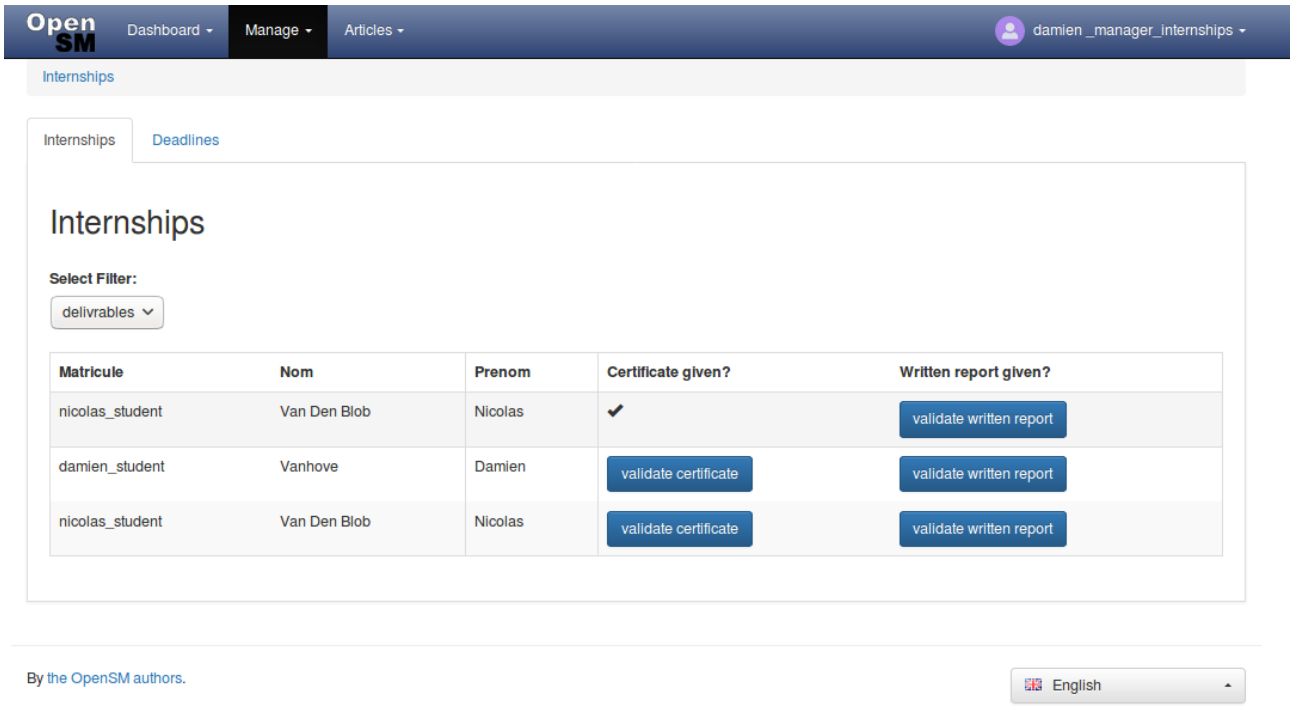


Illustration 10: en un click, le manager peut indiquer que le certificat d'un stage a bien été rendu et validé

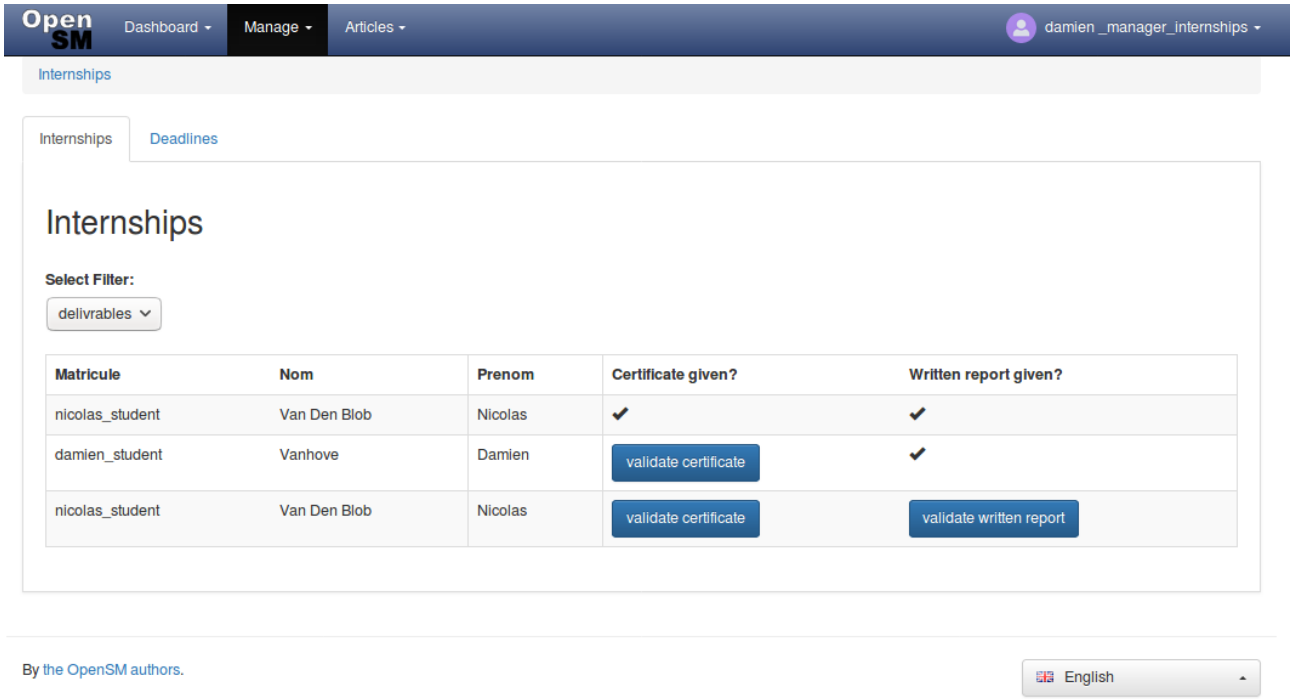


Illustration 11: En un click, le manager peut indiquer que le rapport de stage a bien été rendu

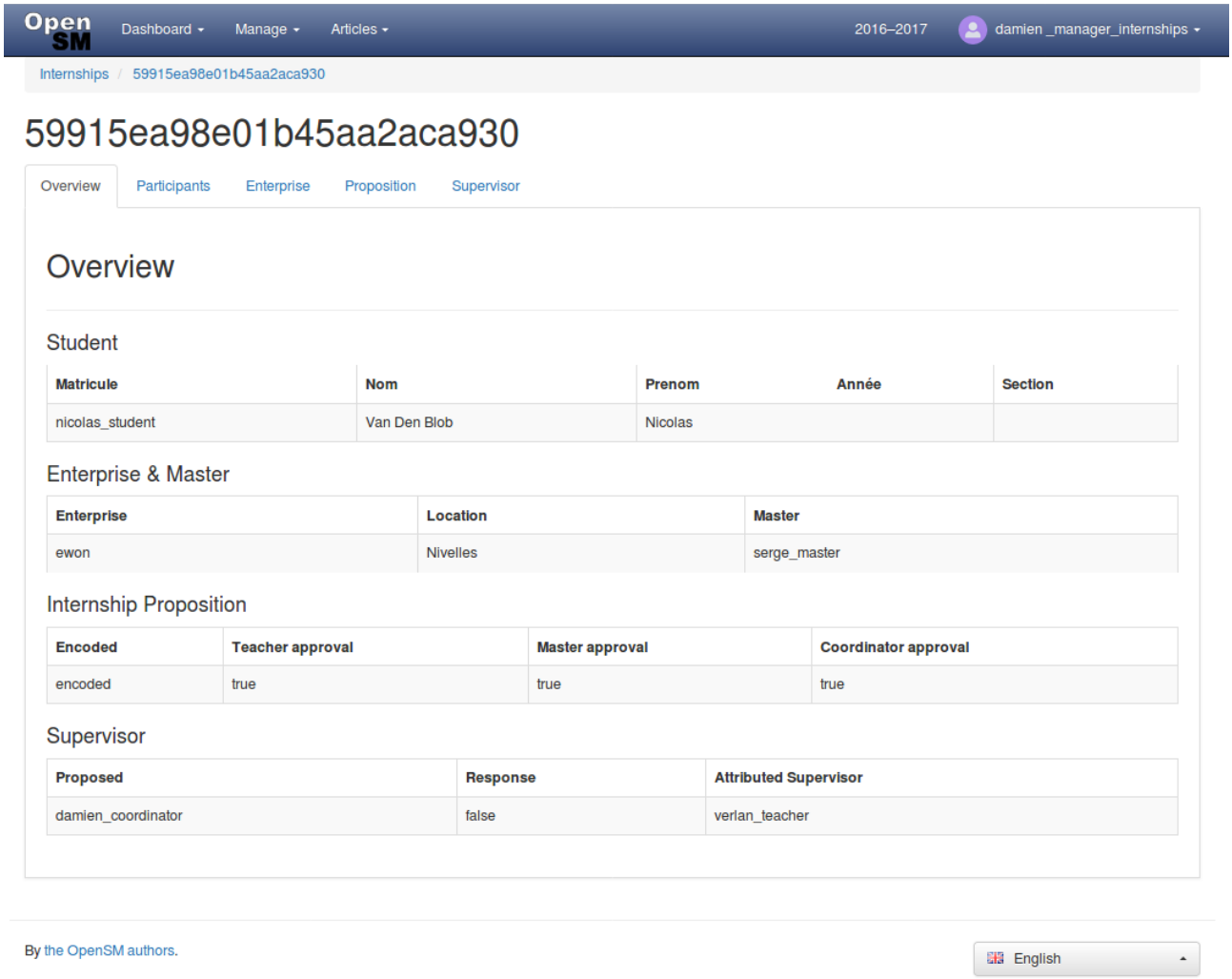


Illustration 12: le manager peut à tout moment accéder aux infos d'un stage précis

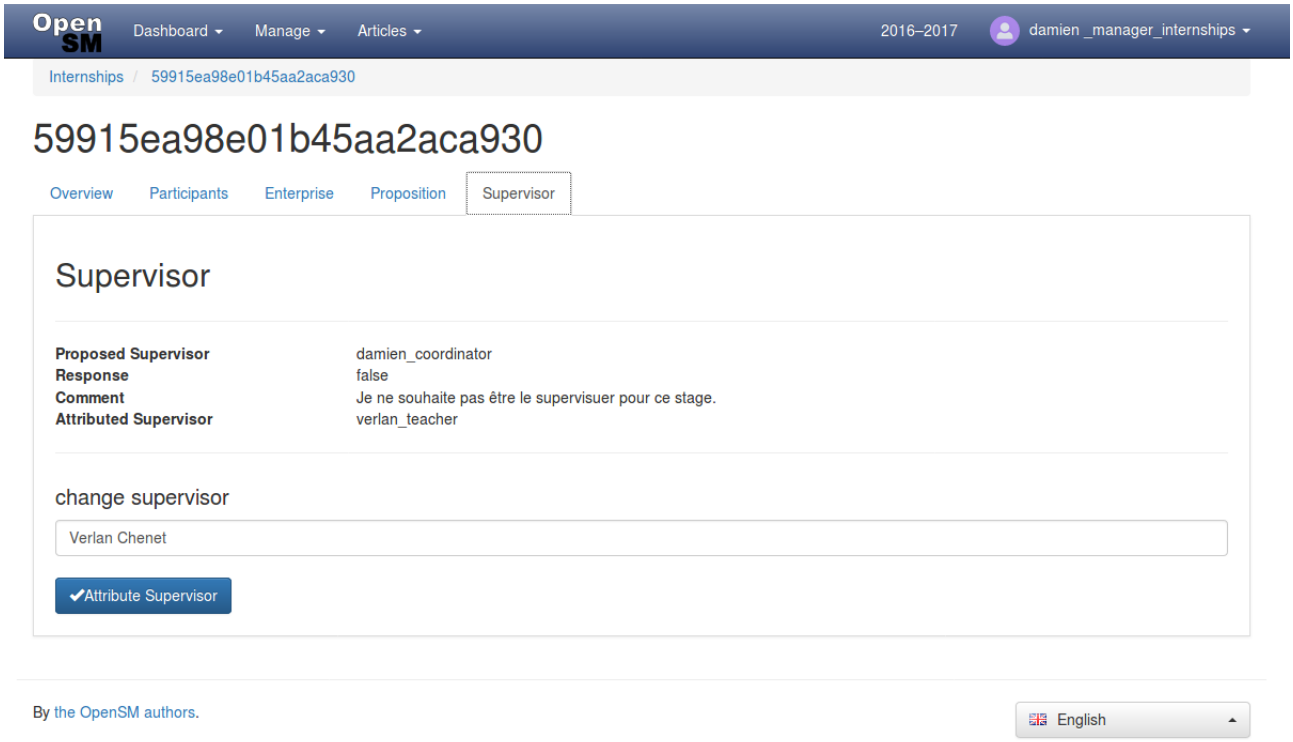


Illustration 13: le manager est le dernier a pouvoir modifier le superviseur d'un stage

7.2 Modèle de données d’un stage

- Statut général : permet de savoir dans quel étape se situe le stage
- State : pré-stage, en stage, ou post-stage (un statut général plus générique)
- L’élève : par référencement
- le maître de stage : par référencement
- le superviseur
  - L’enseignant proposé comme superviseur par l’élève
  - La réponse de l’enseignant proposé
  - Le commentaire de l’enseignant proposé
  - le superviseur attribué : par référencement
- le Validateur : par référencement
- l’enseignant consulté pour la proposition de stage
- La proposition de stage
  - thème
  - domaine
  - localisation
  - description
  - validation
    - validation de l’enseignant consulté pour le sujet
    - commentaire de l’enseignant consulté
    - validation du coordinateur pour le sujet
    - commentaire du coordinateur
    - validation du maître de stage pour le sujet
    - commentaire du maître de stage

- Validation du validator pour le début de stage
- Validation du manager pour le début de stage
- Un log des propositions de stage en cas de non validation et modification
- Informations de l’entreprise
  - nom
  - domaine
  - numéro de téléphone
  - fax
  - mail
  - adresse
  - représentant de l’entreprise

- Convention de stage
  - La convention a-t-elle été délivrée/sortie ?
  - La convention a-t-elle été rendue et validée ?
- Note d’activités
  - Objectifs généraux
    - Objectifs spécifiques
  - Validation
    - accord du superviseur de stage
    - commentaire du superviseur de stage
    - accord du maître de stage
    - commentaire du maître de stage.
- Un log des notes d’activités en cas de non validation et modification
- Première visite d’entreprise
  - date
  - lieu
  - Notes du superviseur
  - Validation de la visite par le superviseur
  - Notes du maître de stage
  - Validation de la visite par le maître de stage
- Évaluation intermédiaire
  - date
  - lieu
  - Validation de l’évaluation par le mâitre de stage
  - Commentaires du maître de stage
- Evaluation continuer
  - points en fonction de la grille d’évaluation (à déterminer)
- Rapport écrit
  - a été rendu ?
  - Points en fonction de la grille d’évaluation (à déterminer)
- Certificat de stage
  - a été rendu ?
- Présentation orale
  - date
  - lieu
  - Points en fonction de la grille d’évaluation (à déterminer)
  - Validation du superviseur de stage
  - Prises de notes du superviseur de stage
  - Validation du maître de stage
  - Prises de notes du maître de stage

- Deadlines
  - Début de stage
  - Fin de stage
  - Référence à un schéma de deadlines, car fait partie d’un module à part.

7.3 Bibliographie

- Holmes, Simon. (novembre 2015). *Getting MEAN with Mongo, Express, Angular, and Node*. Greenwich Connecticut, États-Unis: Manning Publications
- Amos Q, Haviv. (25 septembre 2014). *Mean Web Development*. Birmingham, Royaume-Uni: Packt Publishing
- Kasyap, Krishna (2016). *MongoDB nosql document database tutorial.* Mumbai, Inde: Tutorials Point (I) Pvt. Ltd.
- Kasya, Krishna (2015). *NodeJS tutorial.* Mumbai, Inde: Tutorials Point (I) Pvt. Ltd.
- Wahlin, Dan (2013). *AngularJS in 60-ish Minutes*. Phoenix, Arizona, États-Unis: Wahling Consulting
- *Issa Alain, Schiltz François (2015), Document oriented Databases, Info-H415 – Advanced Database, Université Libre de Bruxelles, Bruxelles*

- The NodeJS Foundation, *Guide and API reference for ExpressJS*, www.expressjs.com, février 2017
- The Mozilla Foundation, *Express Web Framework (Node.js/JavaScript)*, www.developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs, 6 juin 2017
- Liew Zell, *Building a simple CRUD Application with Express and MongoDB*, www.zellwk.com/blog/crud-express-mongodb/, 22 janvier 2016
- Shan, Paul, *Node.JS – reasons to use, pros and cons*, best practices!, www.voidcanvas.com/describing-node-js/, 11 octobre 2014
- Elliot, Eric, *Introduction to Node & Express*, www.medium.com/javascript-scene/introduction-to-node-express-90c431f9e6fd,15 Novembre 2016
- The NodeJS Foundation, Docs and API Reference Documentation, www.nodejs.org/en/docs/, mars 2017
- Hahn, Evan, *Understanding Express.JS*, www.evanhahn.com/understanding-express/, 5 mars 2014
- Shtylma, Roman *Express.JS Middleware Demystified*, www.safaribooksonline.com/blog/2014/03/10/express-js-middleware-demystified/, 10 mars 2014
- Sears, Alex, *Keeping API Routing Clean Using Express Routers,* www.scotch.io/tutorials/keeping-api-routing-clean-using-express-routers*, 31 mars 2016*
- Rajput, Mehul, The pros *and cons of choosing AngularJs.* www.jaxenter.com/the-pros-and-cons-of-choosing-angularjs-124850.html*, 18 mars 2016*
- W3Schools, AngularJS introduction, www.w3schools.com/angular/angular_intro.asp, novembre 2016

- Google, AngularJS Developer Guide and Introduction, www.docs.angularjs.org/guide/introduction, 2017
- Ralston, Steve, *An Introduction to AngularJS*, www.webdesignerdepot.com/2013/04/an-introduction-to-angularjs/, 10 avril 2013
- Thinkster, *A Better Way to Learn AngularJS*, www.thinkster.io/a-better-way-to-learn-angularjs, 2017
- Google, Official AngularJs documentation, www.docs.angularjs.org, 2017
- Precht, Pascal, *Services VS Factory – Once And For All*, www.blog.thoughtram.io/angular/2015/07/07/service-vs-factory-once-and-for-all.html, 8 novembre 2016
- Gough, Josh, *Understan How and Why REST APIs Work – Demonstrated with the VersionOne Data API*. www.blog.versionone.com/understand-how-and-why-rest-apis-work-demonstrated-with-the-versionone-data-api/, 7 février 2013
- Hua Siyuan, *Making API Calls in AngularJs using Angular's $http Service*, 17 janvier 2016
- Couchebase, *Comparing document-oriented and relational data*, www.developer.couchbase.com/documentation/server/3.x/developer/dev-guide-3.0/compare-docs-vs-relational.html, 2017
- MongoDB, inc, *Introduction to MongoDB*, www.docs.mongodb.com/getting-started/shell/introduction/, 2017
- MongoDB, inc, *The MongoDB 3.4 Manual*, www.docs.mongodb.com/manual/, 2017