LOG4420 – Conception de sites web dynam. et transact.

Travail pratique 3

Chargés de laboratoire:

Konstantinos Lambrou-Latreille

Automne 2019

Département de génie informatique et génie logiciel

1 Objectifs

Le but de ce travail pratique est de vous familiariser avec l'environnement du côté serveur en utilisant le cadriciel express. De plus, vous aurez à vous familiariser avec les services web et avec l'utilisation d'une base de données avec MongoDB.

Plus particulièrement, vous aurez à migrer votre projet que vous avez réalisé au travail pratique 1 et 2 vers le cadriciel express tout en utilisant Pug pour réaliser le rendu des différentes vues du site web. De plus, vous aurez à définir plusieurs API pour manipuler les données utilisées par le site web (nouvelles, membres, projets, publications) et vous aurez à utiliser MongoDB pour gérer ces mêmes données.

2 Introduction

Lors des deux premiers travaux pratiques, vous aviez à mettre en place un site web qui était uniquement géré du côté client. En effet, le traitement des données se faisait grâce à des scripts JavaScript qui étaient exécutés sur le navigateur web du client. Aucune sauvegarde des données avait lieu et les informations des différentes collections sont encodées directement dans des variables du Pug.

Afin de s'assurer de l'intégrité des données manipulées, il est nécessaire de gérer les données du côté serveur. En effet, cela permet de masquer le traitement des données tout en cachant certaines données sensibles qui ne doivent pas être divulguées aux différents clients. Le présent travail pratique vous permettra donc de vous initier aux différents aspects et avantages de la programmation du côté serveur.

3 Travail à réaliser

Vous aurez à réaliser l'application du côté serveur pour le site web de groupe de recherche. Avant de débuter, assurez-vous d'avoir récupéré l'archive associée à ce travail pratique sur Moodle. Cette archive contient le squelette d'une application utilisant le cadriciel express.

3.1 Migration du projet vers express

Une fois l'archive associée à ce travail pratique téléchargée et décompressée, la première étape à réaliser est d'installer toutes les dépendances nécessaires au fonctionnement du cadriciel. Pour ce faire, tapez la commande suivante dans un terminal à la racine du projet :

```
npm install
```

Une fois toutes les dépendances installées, il vous suffit de taper la commande suivante dans le terminal pour démarrer l'application :

```
npm start
```

Si tout se passe comme prévu, l'application *express* devrait être fonctionnelle à l'adresse http://localhost:3000.

3.2 Création de services web

La première étape de ce travail est de mettre en place plusieurs services web. L'objectif de ces services web est de publier les données du groupe de recherche et ces données seront utilisées par la partie du site web qui envoie du HTML/Pug. Il est à noter que cette partie est la plus importante puisque vous réutiliserez intégralement les services web que vous réaliserez lors du travail pratique 4.

Les données ont été définies dans le format de fichier YAML qui se trouve dans le dossier « `data/` ». Un ensemble de modules vous sont fournis pour communiquer avec ces données qui se trouvent dans le dossier « `services/yaml/` ». Dans cette partie du TP, vous allez utiliser ces services, donc vous devez regarder leur implémentation pour connaître leur fonctionnement, leur valeur de retour, ainsi que les erreurs générées.

Assurez-vous de définir les routes décrites par les API dans le dossier « `routes/rest` » de l'application.

Les sous-sections qui suivent décrivent les services web qui devront être mis en place. L'ensemble des services web retournent du JSON et les données manipulées sont gérées par des services. L'ensemble des données qui seront manipulées par les services web sont gérées par des services. Ces services ont besoin d'un objet Javascript qui contient des traductions d'un ensemble de bout de texte qui est disponible à partir de « `req.app.locals.t` ». P. ex., « `t["ERRORS"]["MEMBERS_ERRORS"]` » retourne un texte traduit en fonction de la langue en cours.

? Comment tester vos API ?

Un moyen simple de tester vos services web est d'utiliser Postman. De plus, des tests d'intégration vous sont fournis afin que vous puissiez tester vos API (voir la section §Exécution de tests d'intégration et d'acceptation).

Les routes à définir pour cette API sont les suivantes :

1. **GET** `/api/members`

   Obtient tous les membres à partir du service de membres.

   Réponses

`200 (OK)`	Retourne une liste contenant les membres. Le format demandé pour un membre est illustré à l'annexe B. S'il n'y a aucun membre dans la base de données, une liste vide est retournée.
`500 (Server Internal Error)`	Indique qu'une erreur est servenue lors de la récuparation des données. Utilisez l'erreur « `t['ERRORS']['MEMBERS_ERROR']` » si définie, sinon l'objet « `err.message` »

2. **GET** `/api/publications`

   Obtient la liste de publications à partir du service de publications.

   Paramètres d'URL

`limit`	Le nombre de publications à retourner à l'utilisateur. Par défaut, lorsque ce paramètre n'est pas spécifié, la valeur sera 10.
`page`	Le numéro de page de la pagination des publications. Par défaut, lorsque ce paramètre n'est pas spécifié, la valeur est 1.
`sort_by`	Le critère de tri qui doit être utilisé pour trier la liste de publications qui doit être retournée. Par défaut, lorsque ce paramètre n'est pas spécifié, le critère de tri à utiliser est « `date` ». Les valeurs possibles pour le critère sont les suivantes : — `date` : selon la date de publication ; — `title` : selon le titre de la publication ;
`order_by`	L'ordre à utiliser pour la publication. Par défaut, lorsque ce paramètre n'est pas spécifié, l'ordre à utiliser est « `desc` ». Les valeurs possibles pour le critère sont les suivantes : — `desc` : en ordre décroissant ; — `asc` : en ordre croissant ;

Réponses

`200 (OK)`	Retourne une liste de publications. Le format demandé pour un projet est illustré à l'annexe B. S'il n'y a aucune publication dans la base de données, une liste vide est retournée.
`500 (Server Internal Error)`	Indique qu'une erreur est servenue lors de la récuparation des données. Utilisez l'erreur « `t['ERRORS']['PUBS_ERROR']` » si définie, sinon l'objet « `err.message` »

3. **`POST`** `/api/publications/`

   Crée une nouvelle publication à partir du service de publications.

Paramètres *Body* (format `JSON`)

`title`	Le titre de la publication (chaîne de caractères avec au moins 5 caractères).
`month`	Le mois (entre 0 et 11).
`year`	L'année (nombre entier positif).
`authors`	Tableau d'auteurs de la publication où chaque auteur est une chaîne de caractères non vide.
`venue`	Endroit où est publié la publication (chaîne de caractères avec au moins 5 caractères.).

Réponses

`201 (Created)`	Indique que le produit a été créé dans la base de données.
`400 (Bad Request)`	Indique qu'un ou plusieurs paramètres spécifiés sont invalides (p. ex. corps vide (« `t['ERRORS']['PUB_CREATE_ERROR']` »), aucun auteur définie (« `t['ERRORS']['AUTHOR_EMPTY_FORM']` »), l'année n'est pas un nombre entier positif (« `t['ERRORS']['YEAR_NOT_INT_FORM']` »), le mois n'est pas une valeur entre 0 et 11 (« `t['ERRORS']['MONTH_ERROR_FORM']` »), le titre ne contient pas au moins 5 caractères (« `t['ERRORS']['PUB_AT_LEAST_5_CHAR_FORM']` ») et la conférence ne contient pas au moins 5 caractères (« `t['ERRORS']['VENUE_AT_LEAST_5_CHAR_FORM']` »).
`500 (Server Internal Error)`	Indique qu'une erreur est servenue lors de la récuparation des données. Utilisez l'erreur « `t['ERRORS']['PUB_CREATE_ERROR']` » si définie, sinon l'objet « `err.message` »

4. **`DELETE`** `/api/publications/`**`:id`**

   Supprime une publication associé à l'identifiant spécifié (**`:id`**) à partir du service de publications.

Réponses

`200 (OK)`	Indique que le publication a été supprimé.
`404 (Not Found)`	Indique que l'identifiant n'existe pas. Utilisez l'erreur « `t['ERRORS']['PUB_NOT_FOUND_ERROR']` » si définie, sinon l'objet « `err.message` » Erreur dans le test, PUB_DELETE_ERROR pour le 404 dans les tests.
`500 (Server Internal Error)`	Indique qu'une erreur est servenue lors de la récuparation des données. Utilisez l'erreur « `t['ERRORS']['PUB_DELETE_ERROR']` » si définie, sinon l'objet « `err.message` »

5. **GET** `/api/projects`

Obtient la liste de projets à partir du service de projets.

Réponses

`200 (OK)`	Retourne une liste de projets. Le format demandé pour les projets est illustré à l'annexe B. S'il n'y a aucun projet dans la base de données, une liste vide est retournée.
`500 (Server Internal Error)`	Indique qu'une erreur est servenue lors de la récupération des données.

6. **GET** `/api/projects/:id`

Obtient le produit associé à l'identifiant spécifié (**`:id`**) en utilisant le service de projets et de publications.

Réponses

`200 (OK)`	Retourne un projet spécifique. Le format demandé pour un projet est illustré à l'annexe B.
`404 (Not Found)`	Indique que l'identifiant n'existe pas. Utilisez l'erreur « `t['ERRORS']['PROJECT_NOT_FOUND_ERROR']` » si définie, sinon l'objet « `err.message` »
`500 (Server Internal Error)`	Indique qu'une erreur est servenue lors de la récuparation des données. Utilisez l'erreur « `t['ERRORS']['PROJECT_ERROR']` » si définie, sinon l'objet « `err.message` »

7. **GET** `/api/feed`

Obtient le flux de nouvelles à l'aide du service de flux de nouvelles.

Réponses

`200 (OK)`	Retourne une liste de flux de nouvelles. Le format demandé pour les flux de nouvelles est illustré à l'annexe B. S'il n'y a aucun flux de nouvelles dans la base de données, une liste vide est retournée.
`500 (Server Internal Error)`	Indique qu'une erreur est servenue lors de la récuparation des données. Utilisez l'erreur « `t['ERRORS']['FEEDS_ERROR']` » si définie, sinon l'objet « `err.message` »

3.3 Définition des routes pour obtenir le rendu à partir des *templates* Pug

Cette deuxième étape du travail pratique consiste à créer les routes qui permettront d'afficher les différentes du site web. Chaque route devra faire les appels vers les bons services web pour la manipulation des données. Par exemple, dans la page d'accueil, il faut appeller le service web qui récupère les flux de nouvelles.

Les routes seront définies dans le dossier « `routes` » de l'application. Vous devrez séparer le code qui récupère les données des services web dans des *middleware* différents que vous inclurez dans les routes. Pour communiquer avec les services web, plusieurs façons sont possibles, donc choisissez-en une.

3.3.1 Page d'accueil « `/` »

La route de la page d'accueil récupère le flux de nouvelles à partir des services web et elle effectue le rendu de la page *index.pug*. S'il y a une erreur, alors on lance l'erreur avec « `throw err` ».

La variable à envoyer au *template* est « `feeds` » qui contient la liste de flux de nouvelles.

L'internationalisation est déjà mise en place dans le fichier « `app.js` » avec la libraire « `i18n-express` ». Son fonctionnement est dès que vous ajoutez le paramètre de requête « `clang` » à l'URL, « `i18n-express` » s'occupe de changer de charger le bon fichier de « `locales` » pour effectuer la traduction. Également, nous utilisons la libraire « `moment.js` » pour afficher dans n'importe quelle langue les dates. Cependant, « `moment` » n'est pas informé lorsqu'il y a changement de langue, donc vous devez définir un *middlware* supplémentaire que vous appellerez « `changeLang` » et qui s'occupera de modifier la langue de la libraire « `moment` » à partir du paramètres de la requête « `clang` ». Trouvez comment faire.

3.3.2 Page de membres « `/team` »

La route de la page d'équipe récupère les membres à partir des services web et elle effectue le rendu de la page *team.pug*. S'il y a une erreur, alors on lance l'erreur avec « `throw err` ».

La variable à envoyer au *template* est « `members` » qui contient la liste des membres.

3.3.3 Page de projets « `/projects` »

La route de la page de projets récupère les projets à partir des services web et elle effectue le rendu de la page *projets.pug*. S'il y a une erreur, alors on lance l'erreur avec « `throw err` ».

La variable à envoyer au *template* est « `projects` » qui contient la liste de projets.

3.3.4 Page d'un projet « `/projects/:id` »

La route de la page de projets récupère un projet spécifique à partir des services web et elle effectue le rendu de la page *projet.pug*. S'il y a une erreur, alors on lance l'erreur avec « `throw err` ».

Les variables à envoyer au *template* sont « `project` » (projet spécifique) et « `publications` » (publications du projet).

3.3.5 Page de publications « `/publications` »

La route de la page de publications récupère les publications (GET) à l'aide des services web et elle effectue le rendu de la page *publications.pug*. S'il y a une erreur, alors on lance l'erreur avec « `throw err` ». Les variables à envoyer au *template* sont « `publications` » (liste de publications), « `pubFormErrors` » (tableau d'erreurs récupérer du service web), « `pagingOptions` » (tableau Javascript qui représente les options de pagination avec les attributs : limit, pageNumber, sortBy, orderBy), « `numberOfPages` » (nombre de pages possibles selon la pagination) et « `monthNames` » (Liste de mois en fonction de la langue ; utilisez *moment*).

Elle permet également d'ajouter une nouvelle publication (POST) à l'aide des services web. Après l'ajout d'une nouvelle publication, elle effectue le rendu de la page *publications.pug* par un (GET) Attention à la duplication de code dans cette section (pensez aux middlware) !

3.4 Initialisation de la base de données MongoDB

Cette troisième étape du travail pratique consiste à compléter le script « `setup.js` » qui migre les données des fichiers YAML vers la base de données MongoDB. Chaque ficheir YAML sera représenté par une collection Mongo.

Avant d'aller plus loin, suivez les instructions se trouvant à l'annexe A de ce document afin de mettre en place une base de données MongoDB sur mLab. Ensuite, vous devez modifier le fichier « config.json » pour modifier l'URL de la base de données et le nom de la base de données. Utilisez la libraire « `mongodb` » (déjà installée) de NodeJS pour communiquer avec MongoDB.

Voici les étapes :

1. Connection à la base de données MongoDB à partir du fichier de configuration « `config.js` ». Vous devez changer l'URL vers la bonne.
2. Suppression de toutes les collections existantes.
3. Migration du fichier « `data/news.yml` » vers la collection « `news` »
4. Migration du fichier « `data/seminars.yml` » vers la collection « `seminars` »
5. Migration du fichier « `data/team.yml` » vers la collection « `members` »
6. Migration du fichier « `data/publications.yml` » vers la collection « `publications` »
7. Migration du fichier « `data/projects.yml` » vers la collection « `projects` ». Cependant, les identifiants dans l'attribut « `publications` » devront modifié pour représenter un identifiant MongoDB qui a été généré lorsqu'on a migré les publications vers MongoDB.
8. L'attribut « `key` » devra être retiré de toutes les publications dans MongoDB.

Dans cette étape du travail pratique, vous devez utiliser les fonctions « `async/await` » de ES8 qui utilisent le concept de promesses (« `Promise` »). Ces fonctions permettent notament d'écrire du code asynchrone en style synchrone et éviter un code en `sapin de Noël`. Lisez `cette article` pour vous familiariser avec ces fonctions.

3.5 Intégration de MongoDB dans le site web

Cette quatrième et dernière étape du travail pratique consiste à intégrer MongoDB.

Tout d'abord, vous devez effectuer l'initialisation de la base de données dans « `app.js` ». Plus spécifiquement, le site web ne doit pas s'amorcer tant et aussi longtemps que la connection à la base de données n'est pas établie. Vous devrez mettre la variable de la base de données dans l'objet requête (« `req.app.db` »).

Ensuite, dans le fichier « `routes/rest/index.js` », remplacez l'initialisation de la variable « `factory` » par l'initialisation qui utilise le service de mongo.

Finalement, vous devez compléter chaque fonction dans chacun des fichiers de service qui communique avec MongoDB. Ces modules se trouvent dans le fichier « `services/mongo` ».

3.6 Exécution de tests d'intégration et d'acceptation

Afin que vous puissiez valider vos API, des tests d'intégration vous sont fournis. Ainsi, ces tests seront utilisés par le correcteur pour évaluer vos services web. Il est donc important de valider que l'ensemble des tests fonctionnent **avant** que vous remettiez votre travail. Pour exécuter les tests d'intégration, veuillez entrer la commande suivante dans un terminal à la racine du projet :

```
npm test
```

Tout comme au travail pratique 2, des tests d'acceptation automatisés vous ont également été fournis afin que vous puissiez vérifier votre travail. En effet, ces tests permettront de valider que tous les changements que vous avez apportés sont valides. De plus, ces tests serviront à l'évaluation de votre travail. En ce sens, il est important que vous vous assuriez que tous les tests fonctionnent **avant** de remettre votre travail pratique. Pour exécuter les tests d'acceptation, vous devez entrer la commande suivante dans un terminal :

```
npm run e2e
```

Avertissement

Il est important de n'apporter **aucune** modification aux fichiers dans le dossier `tests` (sauf « `nightwatch.json` »). En effet, ces fichiers contiennent tous les tests automatisés qui sont exécutés sur le site web et se doivent de ne pas être modifiés.

Conseils pour la réalisation du travail pratique

1. Référez-vous aux tutoriels sur express et sur MongoDB sur le site de référence du cours.
2. Utilisez Postman pour vérifier rapidement les routes de vos API.
3. Assurez-vous que votre base de données contient les bonnes valeurs. Pour ce faire, vérifiez l'état de votre base de données sur le site web de mLab.
4. N'attendez pas à la dernière minute pour commencer le laboratoire ! Le débogage de services web peut s'avérer long et ardu.

4 Remise

Voici les consignes à suivre pour la remise de ce travail pratique :

1. Vous devez placer le code de votre projet dans un dossier compressé au format ZIP nommé « `TP3_matricule1_matricule2.zip` ». Assurez-vous d'exclure les dossiers « `node_modules` » avant de remettre votre travail.
2. Vous devrez également créer un fichier nommé « `temps.txt` » à l'intérieur du dossier de votre projet. Vous indiquerez le temps passé au total pour ce travail.
3. Le travail pratique doit être remis avant **23h55**, le **13 novembre 2019** sur Moodle.

Aucun retard ne sera accepté pour la remise de ce travail. En cas de retard, le travail se verra attribuer la note de zéro. Également, si les consignes 1 et 2 concernant la remise ne sont pas respectées, une pénalité de -5% est applicable.

Le navigateur web **Firefox** sera utilisé pour tester votre site web.

À vérifier

Avant de remettre votre travail, assurez-vous que tous les paquets de npm que vous avez installés dans votre projet sont listés dans le fichier « `package.json` ». Si vous avez des dépendances manquantes, assurez-vous de les installer avec l'argument « `--save` ».

5 Évaluation

Globalement, vous serez évalué sur le respect des exigences des API à réaliser. Plus précisément, le barème de correction est le suivant :

Exigences	Points
Création de services web	3
Bon fonctionnement du site web avec express	3
Migration des données de YAML vers MongoDB	3
Completion des services qui communiquent avec MongoDB	3
Résultats des tests d'acceptation automatisés	8
Total	20

L'évaluation se fera en grande partie grâce aux tests d'intégration et d'acceptation qui vous sont fournis.

Ce travail pratique a une pondération de **6 %** sur la note du cours.

6 Questions

Si vous avez des interrogations concernant ce travail pratique, vous pouvez poser vos questions sur le forum de laboratoire sur Moodle. N'hésitez pas à poser vos questions sur ce canal afin qu'elles puissent également profiter aux autres étudiants.

Annexes

A Créer une base de données avec mLab

Voici les étapes à suivre afin de mettre en place une base de données avec mLab :

1. Rendez-vous sur le site web de mLab pour vous créer un compte. Assurez-vous de choisir un mot de passe que vous pourrez partager avec votre coéquipier et le chargé de laboratoire.

2. Une fois votre compte créé, vous recevrez un courriel de mLab afin de confirmer votre adresse courriel. Assurez-vous donc de cliquer sur le lien du courriel que vous avez reçu afin de confirmer votre adresse courriel.

3. Une fois votre adresse courriel confirmée auprès de mLab, il vous sera possible de créer une base de données MongoDB. Pour ce faire, vous devez cliquer sur le bouton « *Create new* » se trouvant dans la section « *MongoDB Deployments* ».

4. Lorsque vous aurez cliqué sur le bouton « *Create new* », vous serez dirigé vers un formulaire qui permettra de créer la base de données. Assurez-vous de sélectionner « Amazon Web Services » pour le « *Cloud Provider* » et « ***Sandbox*** » pour le « *Plan Type* ». Par la suite, cliquez sur le bouton « *Continue* ».

5. La page suivante du formulaire vous demandera de sélectionner une région pour la base de données. Assurez-vous de sélectionner la région la plus près du Canada parmi les choix afin de réduire la latence puis cliquez sur le bouton « *Continue* ».

6. La page suivante vous invitera à saisir un nom pour la base de données. Saisissez un nom (p. ex. *online-shop*) puis cliquez sur le bouton « *Continue* ».

7. Une fois arrivé sur la dernière page du formulaire, cliquez sur le bouton « *Submit Order* » afin de créer la base de données. Vous serez alors redirigé vers la liste des « *MongoDB Deployments* ». Si tout s'est passé comme prévu, le nom de votre base de données devrait être affiché dans cette liste.

8. Cliquez sur le nom de votre base de données pour accéder aux paramètres de celle-ci. Par la suite, cliquez sur l'onglet « *Users* » puis sur le bouton « *Add database user* » pour créer un utilisateur pouvant se connecter à la base de données. Remplissez le formulaire puis appuyez sur le bouton « *Create* ». Assurez-vous de saisir un mot de passe qui pourra être partagé entre plusieurs personnes.

B Format des objets de l'API

Cette annexe illustre le format attendu pour les objets retournés par les API que vous avez à implémenter.

Member

```
{
  "_id": 14,
  "lastname": "Nom de famille",
  "firstname": "Prénom",
  "titles": [{"title": "phdStudent", "current": false}]
}
```

Project

```
{
  "project": {
    "_id": 1,
    "type": "master",
    "student": "Nom complet",
    "supervisor": "Nom du directeur",
    "cosupervisors": "Noms des codirecteurs",
    "current": false,
    "publications": [10, 22]
  },
  "publications": [
    {
      "_id": 10,
      "year": 2019,
      "month": "janvier",
      "title": "Titre de la publication",
      "authors": [ "Author1", "Author2" ],
      "venue": "Nom de la conférence"
    },
    {
      "_id": 22,
      "year": 2019,
      "month": "janvier",
      "title": "Titre de la publication",
      "authors": [ "Author1", "Author2" ],
      "venue": "Nom de la conférence"
    }
  ]
}
```

Publication

```
{
  "_id": 1,
  "year": 2019,
```

```
  "month": "janvier",
  "title": "Titre de la publication",
  "authors": [ "Author1", "Author2" ],
  "venue": "Nom de la conférence"
}
```

Feed

Si c'est un séminaire :

```
{
  "_id": 1,
  "type": "seminar",
  "title": "Titre",
  "presentator": "Nom du présentateur",
  "createdAt": "2020-11-12T10:00:00.000Z",
  "date": "2020-11-12T10:00:00.000Z",
  "location": "Lieu",
  "description": "Description du séminaire"
}
```

Si c'est une nouvelle :

```
{
  "_id": 1,
  "type": "news",
  "text": "Titre de la nouvelle",
  "createdAt": "2020-11-12T10:00:00.000Z"
}
```