Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2021 – 2022
PRJG5

PRJG5 – Gestion de projet

Formation initiale

Consignes

1 Un projet Web en équipe

Durant ce cours vous allez devoir réaliser une application Web en équipe. La difficulté principale de ce travail est la coordination des tâches. Vous réaliserez cette application en utilisant le framework `Laravel` introduit dans l'unité WEBG4. Vous pouvez trouver les supports pertinents (présentations et vidéos) WEBG4 sur la page poESI du cours PRJG5.

Afin de se familiariser avec ce framework nous vous proposons de l'utiliser tel qu'étudié, c'est à dire via l'utilisation de `Blade` comme moteur de template.

Durant cet exercice nous allons limiter l'équipe à un binôme. Associez-vous par binôme avant de continuer la suite de l'exercice. Chaque binôme travaille sur **un projet commun**.

Choix de l'architecture

Afin de faciliter la coordination des tâches, une pratique répandue est de diviser l'application web en deux parties indépendantes :

- le `Backend` : partie invisible pour l'utilisateur développée via un framework comme `Laravel` ;
- le `Frontend` : partie visible par l'utilisateur développée via un framework javascript comme `Vue.js` ou `React`.

La partie `backend` contient la base de données et les règles métiers de l'application. Cette partie de l'application met à disposition des services web afin de rendre accessible des données sous format JSON. La partie `frontend` appelle les services web du `backend` et affiche les résultats à l'écran. Vous serez libre d'explorer cette possibilité plus loin dans le cours, une fois que l'organisation des tâches sera maîtrisée.

2 L'application web de prise de présences

Dans les méthodologies agiles, les besoins de l'utilisateur sont décrits via des histoires. En lisant une histoire, le développeur doit être capable de comprendre quel utilisateur a quel besoin et dans quel but.

Commençons la description de l'application par une première histoire.

Version du document
16 septembre 2021

2.1 Première story

Consultation des étudiants

En tant qu'**enseignant** je souhaite consulter via un browser la liste des étudiants. Le but est de prendre les présences lorsque mon cours commence.

Une histoire est également suivie d'une description des tests que l'utilisateur va effectuer pour valider le développement. Pour cette première histoire, l'utilisateur réalisera le test suivant :

- ▷ Lorsque je consulte la liste je vois tous les étudiants triés par matricule.

2.2 Passer d'une histoire à une liste de tâches

Pour réaliser cette première histoire, il va falloir effectuer une série de développements. On peut penser par exemple à la création d'une table dans la base de données, d'une façade DB pour lire une table, d'un modèle, d'un contrôleur, d'une route, d'une vue,...

Faites la liste des tâches dont vous avez besoin pour réaliser cette histoire. Durant cette décomposition en tâches, si des points ne sont pas assez explicites dans la description de l'histoire, discutez avec l'utilisateur [a] afin d'identifier le besoin.

a. l'enseignant dans ce cas

3 Dépôt git et serveur gitlab

Lorsque l'on travaille à plusieurs sur un dépôt, il faut se créer un espace de travail afin de ne pas gêner les autres développeurs. Nous utiliserons git et le serveur gitlab de l'école.

Créez un dépôt sur votre compte `git.esi` intitulé `esi_attendance` ajoutez-y votre professeur comme membre.

3.1 Création du projet laravel

Créez votre projet `Laravel` :

```
composer create-project laravel/laravel attendance-app
```

Déposez le ensuite sur le dépôt `esi_attendance`.

3.2 Gestion des tâches

Pour développer votre application il vous faut réaliser différentes tâches liées aux histoires. Votre dépôt va vous permettre de tenir à jour la liste des tâches sur lesquelles vous travaillez grâce au mécanisme d'issue.

Choisissez la première tâche à développer et attribuez lui un nom. Dans la suite de ce document nous intitulerons cette tâche *crud-étudiant*.

L'acronyme *CRUD* désigne les quatre opérations de base pour la persistance des données : *create*, *read*, *update*, *delete*

Créez une issue

Consultez votre dépôt via le site `https://git.esi-bru.be/` et accédez au menu **Issues**. Ajoutez ensuite grâce au bouton **New Issue** votre tâche :

- **Title** : g12345-crud-étudiant
- **Description** : Crud d'un étudiant.
- **Assignee** : g12345
- **Due Date** : Demain
- **Milestone** : laisser à vide
- **Labels** : laisser à vide

Créez l'issue via le bouton **Submit Issue**.

g12345 représente évidement votre matricule.

Créez le tableau des tâches

Dans les méthodes agiles, une représentation fréquente des tâches a effectuer passe par un tableau divisé en plusieurs colonnes [1], chaque colonne représentant l'état d'une tâche :

- **Open** : tâche non planifiée ;
- **To Do** : tâche a effectuer dans les prochains jours ;
- **Doing** : tâche en cours ;
- **Closed** : tâche terminées.

Nous reviendrons en détail sur l'organisation de ce tableau, pour l'instant ajoutez les colonnes manquantes en ajoutant le label associé (To Do et Doing).

Déplacez ensuite la tâche g12345-crud-étudiant de la colonne open vers la colonne Doing. C'est cette tâche que vous allez réaliser en premier.

Workflow git recommandé

Nous présentons ici le workflow que nous recommandons pour réaliser une tâche.

Créez la branche de travail

Lorsque vous démarrez une nouvelle tâche créez une branche dédiée.

Créons pour l'exemple une branche pour la tâche crud-étudiant. Pour ce faire dans le dossier de l'application exécutez la commande :

```
git branch g12345-crud-étudiant
```

1. `https://fr.wikipedia.org/wiki/Tableau_kanban`

g12345 représente évidement votre matricule.

Pour voir la liste des branches

Vous pouvez facilement consulter les branches de travail en cours et vérifier que votre branche est créée en local via la commande

```
git branch -v
```

Placez votre dépôt local dans cette branche

Pour l'instant votre machine travail dans la branche `master`. Pour demander à votre machine de travailler dans votre nouvelle branche g12345-crud-étudiant vous devez exécuter la commande

```
git checkout g12345-crud-étudiant
```

Réalisez votre développement

Développez dans votre projet `Laravel` la tâche que vous avez choisie.

Synchronisez votre branche avec le serveur

Lorsqu'une étape de votre développement est terminé, n'oubliez pas de sauvegarder votre travail sur votre dépôt :

```
git add .
git commit -m "Fin de la création du modèle Student, closes \#1"
git push -u origin g12345-crud-étudiant
```

Remarquez que dans le message du commit nous mentionnons l'issue #1 précédée du mot-clef `closes`. Cela à pour effet de fermer *automatiquement* l'issue concernée. Plusieurs mots-clefs peuvent être utilisés tels que `closes, Closes, Closing, implements` et d'autres encore [2].

Tout commit, à quelques exceptions près, est lié à une issue, celle-ci doit être mentionnée dnas le message. Si un commit concerne une issue mais ne la clôture pas, utilisez alors le mot clef `Related to #1`.

3.3 Fusionnez votre branche avec la branche master

Une fois vos développements terminés dans votre branche g12345-crud-étudiant, vous pouvez fusionner (merger) votre travail avec la branche master. Pour cela commencez par vous placer dans la branche master via la commande

```
git checkout master
```

Récupérez la dernière version de la branche master présente sur le serveur

2. https://docs.gitlab.com/ee/user/project/issues/managing_issues.html#default-closing-pattern

```
git pull
```

Ensuite vous pouvez demander à git de fusionner votre travail à la branche master

```
git merge g12345-crud-étudiant
```

Deux cas s'offrent à vous.

Cas 1 : Aucun conflit

Aucun conflit n'est détecté et la console présente un message de ce type :

```
Updating 0885fda..1cda754
Fast-forward
....
x file changed, x insertion(+)
```

Vous pouvez mettre à jour le **serveur**

```
git push -u origin master
```

Cas 2 : Conflit détecté

Un conflit est détecté et la console présente un message de ce type

```
Auto-merging ******
CONFLICT (content): Merge conflict in******
Automatic merge failed; fix conflicts and then commit the result.
```

Si des conflits apparaissent lancez la commande suivante pour en avoir le détail

```
git status
```

Plusieurs cas peuvent se produire :

- ▷ Plusieurs personnes ont modifié un fichier en même temps
- ▷ Une personne a supprimé un fichier que vous avez modifié
- ▷ Plusieurs personnes ont ajouté un fichier avec un même nom

Le cas le plus courant est l'édition simultanée d'un même fichier. Dans ce cas pour résoudre le conflit, il faut ouvrir le fichier et parcourir ce fichier pour y trouver la présence des marqueurs `«««« HEAD` et `«««« g12345-crud-étudiant`

```
<<<<<<< HEAD
Ligne présente dans le master au départ
======
Mise à jour de g12345 faite le 2020/09/15 à 9h02
>>>>>>> g12345-crud-étudiant
```

Si on souhaite garder les deux lignes dans le fichier, il faut alors modifier cette partie du fichier manuellement pour la transformer en ceci par exemple :

```
Ligne présente dans le master au départ
Mise à jour de g12345 faite le 2019/11/25 à 9h02
```

Si on considère que seule votre modification du fichier est juste, il faut modifier le fichier comme ceci

```
Mise à jour de g12345 faite le 2019/11/25 à 9h02
```

Une fois le fichier mis à jour, vous devez déposer votre nouvelle modification sur le **serveur** :

```
git add mon_fichier
git commit -m "Merge de la branche g12345-crud-étudiant : résolutions des
    ↪ conflits"
git push -u origin master
```

Nettoyer le serveur

Lorsque votre branche g12345-crud-étudiant à été fusionnée avec succès avec la branche master il n'y a pas de raison de garder votre branche g12345-crud-étudiant. Vous pouvez la supprimer :

```
git branch -d g12345-crud-étudiant
git push origin --delete g12345-crud-étudiant
```

3.3.1 Fin de la tâche

Une fois que votre dernier commit est réalisé, vérifier que la tâche g12345-crud-étudiant est dans la colonne **Closed** du tableau des tâches.

4 Les tests

La méthodologie XP insiste sur un point : un code **non testé** est un code **inutile**. L'idée est de valider les développements effectués avant que d'autres personnes de l'équipe n'utilisent ce code comme base pour leurs propres développements. Deux types de tests sont nécessaires afin de valider les développement.

Les tests unitaires

Au vu de l'histoire développée, on peut lister quelques cas de tests pertinents. Par exemple il faut implémenter des tests unitaires vérifiant le bon fonctionnement du modèle :

- ajout d'un étudiant classique : `1 - SpongeBob - SquarePants`
- ajout avec exception d'un étudiant avec un numéro négatif : `-1 - SpongeBob - SquarePants`
- ajout avec exception d'un étudiant avec un numéro déjà existant : `1 - SpongeBob - SquarePants`

Il faut également tester le bon fonctionnement des requêtes HTTP :

- **créer** un étudiant et recevoir un code retour **201** ;
- **consulter** tous les étudiants et recevoir un code retour **200** ;
- **consulter** un étudiant via sa clé et recevoir un code retour **200** ;
- **mettre à un jour** un étudiant via sa clé et recevoir un code retour **201** ;
- **supprimer** un étudiant via sa clé et recevoir un code retour **204** ;

Implémentez ces différents cas de tests en vous aidant de la documentation disponibles :

- https://laravel.com/docs/8.x/testing
- https://laravel.com/docs/8.x/http-tests
- https://laravel.com/docs/8.x/database-testing

Les tests d'acceptances

Les tests d'acceptances sont décrits dans l'histoire, dans notre cas il n'y a qu'un seul test à réaliser :

- Lorsque je consulte la liste je vois tous les étudiants triés par matricule.

Une fois l'histoire implémentée, il faut réaliser via le browser la consultation de la liste des étudiants et vérifier manuellement la présence des étudiants à l'écran.

Laravel possède une solution pour automatiser ce type de tests via l'utilisation de Dusk [3]. Consultez la documentation afin d'implémenter le test demandé : https://laravel.com/docs/8.x/dusk.

5 Déployer l'application pour une démo

Il est très utile de déployer l'application sur un serveur publique (ou accessible en interne). Cela permet de montrer l'application dans son état actuel et de discuter avec le client.

Plusieurs services existent pour un tel déploiement. On peut penser à *alwaysdata*, *Pivotal Web Services* ou à *Heroku*.

Nous vous proposons de créer un compte étudiant sur https://www.heroku.com/ ou https://www.alwaysdata.com/en/.

Ensuite consultez le tutoriel sur le déploiement d'une application essayez de déployer votre application.

6 La suite des histoires

Maintenant que vous avez expérimenté l'organisation du développement d'histoire, essayez de développer les histoires décrites ci-dessous.

Mettez en pratique ce que vous avez appris lors de la première histoire :

- discuter avec l'utilisateur pour comprendre le besoin ;
- décomposer l'histoire en tâches ;
- utiliser git pour répertorier les différentes tâches ;
- utiliser git pour voir l'état d'avancement des tâches ;
- utiliser les branches pour isoler votre travail.

3. d'autres solutions indépendantes de Laravel existe comme https://www.cypress.io/

Suite à votre expérience acquise sur la première histoire, n'hésitez pas à revoir la décomposition en tâches afin de trouver une découpe qui vous convienne. peut-être que certaines de vos tâches peuvent fusionner ou au contraire certaines tâches doivent être divisées.

Essayez également avant de commencer le développement d'estimer le temps de travail que vous associez à chaque tâche.

6.1 Deuxième story

Consultation des étudiants

En tant qu'**enseignant** je souhaite ajouter ou supprimer un étudiant de la liste afin que celle-ci soit à jour.

- ▷ J'ajoute un nouvel étudiant et vois dans la liste ce nouvel étudiant.
- ▷ Je supprime cet étudiant, il n'apparait plus dans la liste.

6.2 Troisième story

Consultation des leçons

En tant qu'**enseignant** je souhaite consulter via un browser la liste des leçons prévues. Le but est de connaître l'horaire des cours donnés.

Une leçon est définie par un cours, une date et une heure.

Par exemple `PRGL - 24/09/2021 - 13:30`.

Pour valider votre développement l'utilisateur va procéder au test suivant :

- ▷ Lorsque je consulte la liste je vois toutes les leçons triées par date et heure.

On ne demande donc pas de gèrer les leçons.

6.3 Quatrième story

Ajout des présences

En tant qu'**enseignant** je souhaite prendre les présences lors d'une leçon. Le but est de connaître le nombre d'étudiants par leçon.

Pour valider votre développement l'utilisateur va procéder aux tests suivants :

- ▷ Lorsque j'ajoute une présence, je peux la consulter dans une liste des présences liée à la leçon.