Evaluation finale - énoncé

Introduction

Les exercices

Pour cette évaluation vous allez être livrés à vous-même devant une série d'exercices qui consisteront à améliorer une application existante.

Vous aurez à réaliser différents exercices. La plupart ont été conçus pour pouvoir être réalisés indépendamment les uns des autres. Il est demandé de créer une branche git par exercice, ceci afin que vous puissiez facilement revenir à un code fonctionnel si vous cassez tout. Vous n'êtes pas obligés de faire les exercices dans l'ordre ! Sachez cependant qu'ils ont été conçus avec des problématiques concrètes, pensez donc à soigner l'user experience ☺

Les exercices ont 1 temps indicatif de réalisation + 1 niveau de difficulté allant de 1 (simple) à 3 (complexe). Cela vous aidera à savoir quels exercices faire en priorité et lesquels faire plus tard.

Chaque exercice possède également un type :

- **A : Algorithme – résolution d'un problème d'algo comme en dojo**
- **C : Code – développement pur (débogage, ajout de fonctionnalité, refacto etc.)**
- **T : Terminal – utilisation du terminal bash, ce sont des exercices requis (le 1er et dernier en fait)**
- **Q : Question – des questions de cours / des rédactions à réaliser**

Il vous sera demandé de rédiger des articles dans le blog pour répondre aux exercices de type Q. A la fin de cet énoncé vous trouverez donc des instructions pour extraire votre base de données Mongo et l'ajouter à votre dépôt github. Ceci permettra à votre formateur de récupérer vos données Mongo depuis votre dépôt et donc vos réponses. Cet export est très important, sans lui le formateur ne pourra pas valider vos réponses.

Tous les exercices avec une étoile devant leur numéro sont recommandés. Faites ceux sans étoile selon le temps qu'il vous reste. Pour le deuxième exo d'algo (recommandé d'en faire au moins 2), faites le complexe uniquement si vous vous sentez aventureux.

Le projet

En guise de projet, nous allons partir d'une application de blog existante tournant sur la stack MEAN : https://github.com/joysoft33/2017-bdx-checkpoint-5-js

Ce projet utilise les technologies suivantes :

- **Angular**
- **Angular UI Router**
- **Express**
- **JSONWebToken**
- **Mongoose**
- **MaterializeCSS**
- **Passport**
- **Babel**
- **SCSS**

- **Nodemon**
- **Webpack**
- **Livereload**

La structure de l'application est classique :

- **Dossier app : contient le code du backend**
  - **Routes : les routes de l'API express**
  - **Models : les models Mongoose**
  - **Controllers : code exécuté lors d'un appel à une route Express et pouvant récupérer des données via les models**
- **Dossier config : configuration de l'app (passport / environnement)**
- **Dossier public : contient le code angular & le scss**
  - **Js/components : composants angular**
    - **Blog : les composants du blog**
    - **Common : les composants « communs » à toute l'app (navbar)**
    - **Login : les composants en charge des formulaires d'authentification / d'inscription**
  - **Js/config : configurations de l'app angular, notamment de angular ui router**
  - **Js/services : services angular, utilisés pour faire les appels à l'API express**
  - **app.js : fichier d'entrée de l'app angular**
  - **css/style.scss : le style de base de l'app**
  - **css/other.scss : Le fichier où vous mettrez vos styles CSS**
- **package.json : fichier de config des paquets de nodejs**
- **wild-blog-app.js : fichier d'entrée de notre app node**
- **server.js : fichier de configuration de notre serveur web / express**
- **webpack.config.js : fichier de configuration de webpack**

Vous aurez le droit à toutes les ressources accessibles en ligne. Un bon développeur est d'abord un développeur sachant trouver les réponses rapidement sur les sujets qui lui font défaut.

Notez bien

Pensez à faire de petits commits durant vos travaux et à rédiger des bons messages. Cela aidera le formateur à mieux vous évaluer. L'usage des branches dans votre projet et l'écriture / réalisations de bons commits font partie de l'évaluation.

Faites également attention à la qualité de votre code, la consistance de la convention de nommage, le nom de vos variables, l'usage de la langue de Shakespeare, vos commentaires pour documenter votre code. Ce n'est pas explicitement dit dans les exercices mais vous serez aussi évalués sur tout ça.

Enfin, prenez garde au code que vous pourriez récupérer en ligne. Ce n'est pas tout de trouver la solution en ligne, il faut aussi savoir l'intégrer dans le projet existant, au bon endroit et savoir l'expliquer. Notez que vous avez tout à fait le droit d'aller voir le code que vous avez déjà réalisé jusqu'à présent dans vos projets & tutoriels et de le réutiliser.

Vous êtes prêts ?

C'est parti ☺

Table des matières

Exercice 1* : Faire fonctionner le projet – 20 minutes – difficulté 1 – type T

Objectif :

Bon, évidemment ce premier exercice est requis pour faire tous les autres. Et si vous vous retrouvez coincés dessus, n'hésitez pas à appeler votre formateur. Ne restez pas plus de 15/20 minutes sur cet exercice.

Le but est de voir si vous vous débrouillez sur la reprise d'un projet existant et avec GIT/GITHUB.

Instructions :

1. **Faites un fork du projet : https://github.com/WildCodeSchool/wild-blog**
2. **Envoyez l'adresse de votre fork en message privé à votre formateur sur le wildchat avec vos prénoms/noms, c'est sur cette adresse que le formateur pourra récupérer votre travail de votre évaluation.**
3. **Faites un clone de votre fork en local**
4. **Effectuez la/les commande(s) pour installer le projet une fois cloné**
5. **Lancez le projet et comprenez l'erreur affichée**
6. **Si vous avez compris l'erreur, sollicitez votre formateur en lui posant la bonne question par tchat.**
7. **Après réception de la réponse du formateur, vous devriez ne plus avoir trop de mal pour bien lancer le projet.**
8. **Une fois le projet lancé, créez un utilisateur sur le site, gardez précieusement ses identifiants**
9. **Une fois authentifié avec l'utilisateur, créez un article dans le blog :**
    a. **Titre : Exercice 1**
    b. **Contenu :**
        i. **Recopiez d'abord les identifiants de votre utilisateur, ceci afin que le formateur puisse s'authentifier avec ce dernier lorsqu'il aura récupéré vos données**
        ii. **Ensuite, expliquez dans cet article :**
            1. **Ce qu'est un fork Github ?**
            2. **A quoi sert-il ?**
            3. **Qu'est-ce qu'une pull request ? Quel est l'intérêt ?**
            4. **Qu'est-ce qu'une branche ?**
            5. **Quelle est la différence entre une branche et un fork ?**

Exercice 2* : Quelques questions – 30 minutes – difficulté 1 – type Q

Objectif :

Avant de poursuivre, il est important que vous vous rafraîchissiez la mémoire et que vous prouviez à votre formateur que vous avez bien suivi toute la formation.

Instructions :

1. **Créez un article du blog (titre : La MEAN stack). Pour chaque technologie du projet (celles listées ci-après), rédigez un court paragraphe dans le contenu de l'article expliquant :**
    a. **A quoi elle sert ?**
    b. **Où va-t-on l'utiliser (côté front, côté back, front & back, terminal…) ?**
    c. **Si elle est facultative ou non pour ce projet : une technologie qu'on pourrait facilement enlever du projet peut être considérée comme "facultative". Si vous devez tout casser pour enlever la techno c'est qu'elle est sûrement requise.**
2. **Une fois fait, rajoutez un paragraphe expliquant ce qu'est la MEAN stack**

Technologies du projet :

- **Angular**
- **Angular UI Router**
- **Express**
- **JSONWebToken**
- **Mongoose**
- **MaterializeCSS**
- **Passport**
- **Babel**
- **SCSS**
- **Nodemon**
- **Webpack**
- **Livereload**

Exercice 3* : Réaliser une petite horloge – 30 minutes – difficulté 1.5 – type C

Objectif :

A l'aide de la bibliothèque « momentjs », réaliser une horloge qui se met à jour chaque seconde.

Souvenez-vous de créer une branche & des petits commits avec un bon message explicatif. Je vous le rappelle ici, mais c'est la dernière fois.

Instructions :

1. **Cherchez la bibliothèque « momentjs » en ligne et installez/ajoutez la dans votre projet**
2. **Importez-la dans le composant de la navbar pour pouvoir l'utiliser dans le controller**
3. **Dans un premier temps, affichez juste la date du jour et l'heure actuelle dans la navbar en la mettant à droite du logo :**

Note : aucun format d'affichage de date n'est imposé, un affichage anglo-saxon suffira.

4. **Une fois fait, mettez à jour cette date toutes les secondes. Vous n'avez, normalement, juste qu'à toucher au controller angular de la navbar.**
5. **Si vous n'avez pas la date qui se met bien à jour via l'usage du JS natif, pensez à regarder s'il n'existerait pas quelque chose en angular**
6. **Astuce : une bonne recherche sur google pourrait donner la solution (à 2/3 petites choses près) :)**

Exercice 4* : Algo 1 – 20 minutes – difficulté 1 – type A

Objectif :

Un petit exercice d'algo pour tester vos connaissances en JS natif ☺

Celui-ci est relativement simple. Un autre exercice, plus complexe, sera également à rendre.

Instructions :

Cet exercice est issu de www.codewars.com un site proposant justement de s'entraîner sur divers langages (il y en a beaucoup) à travers des exercices d'algo (comme ce que nous faisons en dojo). L'intérêt principal étant qu'une fois l'exercice résolu, on a accès à la réponse d'autres personnes ayant utilisé d'autres méthodes / fonctions pour parvenir à la solution.

1. **Avant de commencer, créez une nouvelle route angular ui dans « config/routes ». Nommez l'état « algo1 » et mettez l'url : « /algo1 ».**
2. **Ensuite, libre à vous de créer un composant angular ou de tout mettre dans le controller directement au niveau de la route. Evidemment, créer un composant serait plus propre. Mais il s'agit ici uniquement de résoudre un exercice d'algo, donc on ne va pas être trop embêtant. Cependant si vous n'utilisez pas de composant, vous devrez utiliser $scope pour afficher le résultat dans la page.**
3. **Et maintenant l'énoncé de l'algo (source : www.codewars.com/kata/friend-or-foe/train/javascript) :**
    a. **Recevant un tableau de prénoms, vous devez en extraire vos amis. Vos amis sont tous les prénoms contenant exactement 4 lettres. Affichez la liste d'amis ainsi récupérés dans le template de la route /algo1 à partir du tableau :** `["Ryan", "Kieran", "Mark"]` **Attention, un simple « return » de la solution ne sera pas accepté ! (On n'acceptera pas : return** `["Ryan", "Mark"])`
    b. **Nous vous fournissons ci-dessous les tests unitaires qui sont censés fonctionner une fois l'algo résolu :**

```
Test.assertSimilar(friend(["Ryan", "Kieran", "Mark"]), ["Ryan", "Mark"]);

Test.assertSimilar(friend(["Ryan", "Jimmy", "123", "4", "Cool Man"]),
["Ryan"]);

Test.assertSimilar(friend(["Jimm", "Cari", "aret", "truehdnviegkwgvke",
"sixtyiscooooool"]), ["Jimm", "Cari", "aret"]);

Test.assertSimilar(friend(["Love", "Your", "Face", "1"]), ["Love", "Your",
"Face"]);
```

Note : vous pouvez vous créer un compte sur codewars grâce à votre compte github si vous le souhaitez et accéder à l'exercice ci-dessus via le lien source. Cela vous permettrait d'utiliser les tests unitaires en ligne sur leur site. Mais ce n'est pas obligatoire pour cet exo d'algo.

Exercice 5* : Ajouter un champ « date » à l'article – 30 minutes – difficulté 1 – type C

Objectif :

Nous aimerions ajouter une date de publication à l'article sous la forme «JJ-MM-AAAA », par exemple : 03-05-2017.

Instructions :

- **Lors de l'ajout/édition d'un article, un troisième champ « date de publication » doit être visible et on doit pouvoir y inscrire une date**
- **L'usage d'un Datepicker est un gros plus mais pas indispensable. Si vous n'utilisez pas de datepicker, mettez au moins un message d'aide pour que l'utilisateur renseigne la date sous le bon format.**
- **Le contenu de ce champ doit apparaître sur la page de visualisation d'un article, en dessous de son titre : « publié le JJ-MM-AAAA »**
- **Pensez à la mise à jour du schéma Mongoose pour l'enregistrement, réfléchissez à un bon nom de champs, en prenant exemple sur les « createdAt » / « updatedAt »**

Exercice 6* : Ajouter un champ « publié » à l'article – 30 minutes – difficulté 2 – type C

Objectif :

Comme l'exercice 5, nous aimerions ajouter un nouveau champ à l'article, pour indiquer s'il doit être publié ou non. S'il n'est pas publié, il ne doit ni s'afficher dans la page de la liste des articles ni être accessible via son id sur l'url http://localhost:8080/#!/posts/5908c1b19fb91b8da99f4630

Instructions :

- **Sur la page d'édition d'un article, ajouter un champ « on/off » de materialize : http://materializecss.com/forms.html (chercher « switch »)**
- **Par défaut considérez les articles où « published » n'est pas défini comme non publiés**
- **Faites le filtrage des articles « publiés » côté express dans la route de l'API. Trouvez la meilleure méthode pour que ça filtre bien sans forcément casser/modifier les méthodes de « Controller.js ».**
- **Lorsque l'on essaie d'accéder à un article non publié, ça doit nous rediriger vers l'index des articles avec un message : « Article not published yet » - utiliser Materialize.toast pour afficher ce message. Idem le renvoi ou non de l'article sur cette url : http://localhost:8080/#!/posts/post_id doit aussi être géré côté express**

Exercice 7* : Culture SCRUM – 30 minutes – difficulté 1 – type Q

Objectif :

Tester vos connaissances sur la méthode agile SCRUM

Instructions :

Rédigez un article présentant les informations suivantes :

- **Que sont les méthodes agiles ?**
- **Qu'est-ce que SCRUM ?**
- **Définir le Product BackLog, de quoi est-il constitué ?**
- **Définir le Product Owner**
- **Définir le SCRUM Master**
- **Définir la « Development team »**
- **Définir le Sprint, de quoi est-il constitué ?**
- **Définir le daily scrum**
- **Qu'est-ce que la planification d'un sprint ?**
- **Qu'est-ce que la révision d'un sprint ?**
- **Qu'est-ce que la rétrospective d'un sprint ?**
    - ***Attention : en cours nous appelions « rétrospective » ce qui est en réalité la « révision du sprint » au sens de la méthodologie SCRUM. Nous n'avons pas réellement fait de rétrospective sur un sprint, essayez de trouver l'info en ligne sur la différence entre ces 2 notions (vous avez le droit au cours)***
- **Pourquoi est-il important de définir la notion de « Done » pour les tâches ?**
- **La notion de « Done » peut-elle évoluer au cours d'un projet SCRUM ?**
- **SCRUM est-elle une méthodologie itérative ? Justifier.**

Ecrivez l'article comme une FAQ :

Question …

Réponse …

Attention : ne copiez/collez pas des phrases de wikipedia. Ne copiez/collez pas non plus des phrases en anglais. Expliquez tout ça avec vos mots et comment vous les comprenez. Il faut que ça soit pour votre formateur l'occasion de voir si vous avez bien tout compris ou non.

Exercice 8* : SCRUM – du concret – 45 minutes – difficulté 2 – type Q

Objectif :

Réaliser un product backlog et un premier sprint pour une nouvelle fonctionnalité sur le blog.

Instructions :

- **Créer (et oui encore) un nouvel article.**
- **Voilà le contexte :**
    - **Le client (pour qui vous avez réalisé ce blog) souhaite ajouter une partie administrateur à l'application pour pouvoir gérer les utilisateurs et les articles postés.**
    - **Il aimerait notamment pouvoir lui-même ajouter/définir de nouveaux administrateurs. Seul un administrateur doit pouvoir dire si un autre utilisateur peut être administrateur ou non.**
    - **Il aimerait aussi pouvoir modifier les articles, les supprimer, changer la date de publication, etc.**
    - **Il souhaite enfin que seul un administrateur puisse publier/dépublier un article**
    - **Seul un utilisateur dont le paramètre « isAdmin » vaut true (voir le model Mongoose de User) peut accéder à la partie administrateur.**
- **Rédiger dans l'article le product backlog de ces fonctionnalités. Tâchez de ne rien oublier. Prenez en compte – dans votre product backlog – que vous avez déjà réalisé les exercices précédentes, y compris l'ajout du champ « publié » dans le formulaire d'un article.**
    - **Définissez, pour chaque item du product backlog, les conditions de validation et les étapes de test**
    - **Priorisez enfin les items du product backlog**
- **Rédiger ensuite un premier sprint « réaliste ». Estimez vos temps de réalisation. N'oubliez pas que les items d'un sprint sont différents des items d'un product backlog.**

Exercice 9 : Partie admin – X heures – difficulté 1 à 3 – type C

Objectif :

Réaliser la partie admin pour laquelle vous avez fait le product backlog en exercice 8 (si vous faites les exos dans l'ordre).

Cet exercice a une difficulté de 1 à 3 car certaines choses sont relativement simples (créer une route admin, une page, etc.) et d'autres plus complexes (bloquer les accès convenablement, éditer les utilisateurs, etc.)

Instructions :

- **Cet exercice peut être relativement long, attaquez le plutôt dans les derniers exos.**
- **Prenez en compte toutes les instructions de l'exercice 8 pour réaliser cette partie admin**
- **Les instructions sont volontairement succinctes pour voir comment vous vous en sortez sur une vraie demande cliente d'ajout de fonctionnalités dans une application existante**
- **Exploitez au maximum HTML 5, Materialize, etc.**
- **Ne pensez pas que vous n'y arriverez pas, quel que soit votre niveau, vous POUVEZ faire des éléments de cet exercice. Dans les grandes lignes :**
    - **Commencez par faire les pages webs (les vues) angular, dans un premier temps vous pouvez simplement faire un template dans un state d'angular ui router, c'est le plus simple. Seule chose imposée : le bandeau du menu administrateur en haut ne doit pas être bleu mais noir, pour distinguer partie user de partie admin**
    - **Poursuivez par les composants et les controllers angulars, par les state pour changer de page**
    - **Ajoutez la logique des routes express,**
    - **Verrouillez-les routes de l'API express**
    - **En cas d'accès à la partie admin avec un utilisateur n'ayant pas les droits, redirigez le vers la page d'accueil /posts.**
- **Faites votre max sur cet exercice, peu importe que vous le finissiez ou non**

Exercice 10 : Algo – 2 – 30 minutes – difficulté 2 – type A

Objectif :

Un plus gros exercice d'algo.

Celui-ci est de niveau un peu plus élevé que le premier.

Instructions : (quasi identiques à algo 1)

Cet exercice est issu de www.codewars.com un site proposant justement de s'entraîner sur divers langages (il y en a beaucoup) à travers des exercices d'algo (comme ce que nous faisons en dojo). L'intérêt principal étant qu'une fois l'exercice résolu, on a accès à la réponse d'autres personnes ayant utilisé d'autres méthodes / fonctions pour parvenir à la solution.

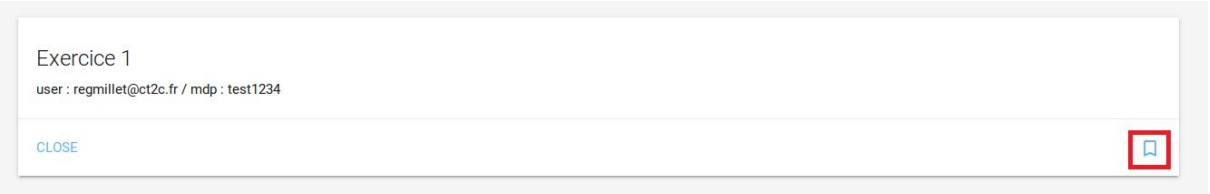
1. **Avant de commencer, créez une nouvelle route angular ui dans « config/routes ». Nommez l'état « algo2 » et mettez l'url : « /algo2 ».**
2. **Ensuite, libre à vous de créer un composant angular ou de tout mettre dans le controller directement au niveau de la route. Evidemment, créer un composant serait plus propre. Mais il s'agit ici uniquement de résoudre un exercice d'algo, donc on ne va pas être trop embêtant. Cependant si vous n'utilisez pas de composant, vous devrez utiliser $scope pour afficher le résultat dans la page.**
3. **L'algo est déjà présent en ligne avec l'énoncé et les tests unitaires : http://www.codewars.com/kata/folding-your-way-to-the-moon/javascript**
4. **Affichez dans la page /algo2, le résultat de : foldTo(14928418679754190000)**

Note : Pour cet exo d'algo, vous devez vous créer un compte sur codewars grâce à votre compte github si vous le souhaitez et accéder à l'exercice ci-dessus via le lien source.

Exercice 11* : Debugger – 1h – difficulté 2 – type C

Objectif :

Faire fonctionner ce bouton :



Instructions :

- **Des bugs ont été introduits dans le code pour faire fonctionner ce bouton…**
- **A vous de trouver où et de les résoudre.**
- **Il n'y en a pas beaucoup (des bugs) et ils sont très localisés dans le code (dans un seul fichier).**
- **Si vous vous retrouvez à réécrire toute une méthode c'est que vous n'êtes pas sur la bonne piste**
- **Utilisez des console.log, lisez les erreurs (si vous en avez)**
- **Comprenez ce qui se passe**
- **S'il n'y avait pas de bug :**
    - **Un clic sur le bouton afficherait un message « Added » en vert**
    - **Le bouton serait alors rempli en bleu**
    - **Si vous rafraîchissez, le bouton reste rempli en bleu**
    - **Un clic sur un bouton rempli en bleu affiche un message « Removed » en orange**
    - **Le bouton se vide alors comme dans l'image ci-dessus**
    - **Si vous rafraîchissez, le bouton reste vide**
- **Comme il s'agit d'un prérequis pour l'exercice suivant, si vous restez coincés trop longtemps (>30 min), n'hésitez pas à poser des questions au formateur**

Exercice 12 : Terminer une fonctionnalité – 1h – difficulté 2 – type C

Objectif :

Faire fonctionner le bouton « bookmarked » du menu qui est censé afficher la liste de tous les articles mis en favoris :

Actuellement lorsqu'on clique dessus, ça ne fait absolument rien.

Evidemment il faut avoir fini l'exercice 11 pour pouvoir faire celui-ci.

Instructions :

Votre client n'a définitivement pas de chance. Le développeur qui avait travaillé sur la fonctionnalité des favoris a non seulement livré une fonctionnalité bugguée mais en plus non finie.

Comme il est très content que vous ayez débuggé cette fonctionnalité (cf exercice 11), il souhaiterait maintenant que vous fassiez fonctionner ce bouton « bookmarked ».

Le principe du bouton est simple, si on clique dessus on ne doit plus voir que ses articles favoris.

Gérez-le avec une nouvelle route angular et une API express pour ne renvoyer que les articles en favoris. C'est le plus simple ☺

Exercice 13 : un peu de refactoring – 20 minutes – difficulté 2 – type C

Objectif :

Rendre du code plus propre

Instructions :

- **Le fichier HTML « blogItem.html » du composant angular « blogItem » n'est pas très beau… Il y a des « ng-if » partout et plein de conditions chaînées du genre : « (!$ctrl.editable || ($ctrl.editable && !$ctrl.editMode)) » par très parlant en l'état**
- **Voyez si vous ne pouvez pas mutualiser du code**
- **N'hésitez pas à créer des méthodes dans votre controller avec des noms plus parlant que ces conditions chaînées**
- **Par exemple une méthode nommée « isEditing() » serait la bienvenue ☺**
- **Evidemment il faut que le code fonctionne toujours après refactoring**

D'une manière général il est préférable d'avoir dans un seul endroit les conditions testées. Même si ça vous fait bizarre d'avoir une méthode toute petite qui ne fait que 1 ou 2 lignes. Dites-vous que vous vous remercierez de l'avoir fait le jour où il faudra ajouter une condition impliquant qu'un utilisateur est en mode édition (genre : user.isAdmin par exemple ☺).

Exercice 14 : Algo – 3 - 1h30 – difficulté 3 – type A

Objectif :

Un plus gros exercice d'algo.

Celui-ci est le plus complexe des exos d'algo. Son énoncé étant assez fourni, je vous invite à la résoudre directement en ligne sur codewars :
http://www.codewars.com/kata/magic-the-gathering-number-1-creatures/train/javascript

Instructions : (quasi identiques à algo 1)

Cet exercice est issu de www.codewars.com un site proposant justement de s'entraîner sur divers langages (il y en a beaucoup) à travers des exercices d'algo (comme ce que nous faisons en dojo). L'intérêt principal étant qu'une fois l'exercice résolu, on a accès à la réponse d'autres personnes ayant utilisé d'autres méthodes / fonctions pour parvenir à la solution.

5. **Avant de commencer, créez une nouvelle route angular ui dans « config/routes ». Nommez l'état « algo3 » et mettez l'url : « /algo3 ».**
6. **Ensuite, libre à vous de créer un composant angular ou de tout mettre dans le controller directement au niveau de la route. Evidemment, créer un composant serait plus propre. Mais il s'agit ici uniquement de résoudre un exercice d'algo, donc on ne va pas être trop embêtant. Cependant si vous n'utilisez pas de composant, vous devrez utiliser $scope pour afficher le résultat dans la page.**
7. **L'algo est déjà présent en ligne avec l'énoncé et les tests unitaires : http://www.codewars.com/kata/magic-the-gathering-number-1-creatures/javascript**
8. **Affichez dans /algo3 le résultat de : battle([[1, 3], [3, 4]], [[2, 8], [5, 2]])**
9. **Une solution par pseudo code rédigée dans un article sera également acceptée pour celui-ci ☺**

Note : Pour cet exo d'algo, vous devez vous créer un compte sur codewars grâce à votre compte github si vous le souhaitez et accéder à l'exercice ci-dessus via le lien source.

Exercice 15* : Meilleur README – 45 minutes – difficulté 1 – type Q

Objectif :

Rédiger le README

Instructions :

- **Vous avez eu un cours sur la documentation et on vous a normalement parlé un peu des readme et ce qu'ils devaient contenir**
- **Eh bien il est temps de prouver que vous avez écouté en cours. Rédigez le readme du projet en y mettant toutes les informations que vous pensez/savez être utiles dans un readme.**
- **Attention, un trop plein d'informations peut aussi être négatif pour votre projet. Soyez concis mais exhaustifs**

Exercice 16* : Fin d'évaluation – 1h – difficulté 1 – type T

Objectif :

Rendez votre base de données dans votre projet

Instructions :

Suivez ce tuto : https://docs.mongodb.com/manual/reference/program/mongoexport/ pour exporter votre base de données « blog ».

En sautant l'étape « insert » évidemment.

Ajoutez l'export dans votre git dans une branche nommée « with-db-dump »

Oubliez pas de pusher toutes vos branches en ligne !

Demandez à votre formateur de valider que vous avez bien fait ce qu'il faut.

Si c'est ok, vous avez fini l'évaluation ☺

Bravo !