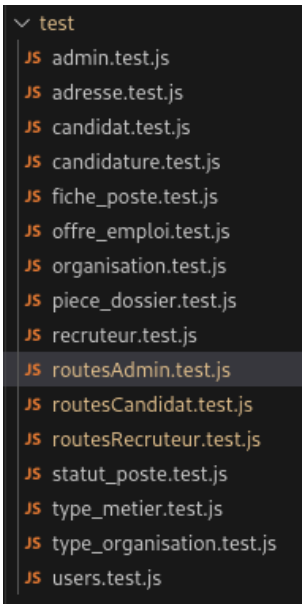
Pontoire Julien - Fouinat--Beal Quentin

Compte rendu des tests

Pour la partie test, nous avons séparé nos tests en plusieurs fichiers. Nous avons un fichier par modèle que nous testons et également un fichier par type de routes (admin, recruteur et candidat).



Pour la partie modèles, nous avons testé la majorité des fonctions que nous avons créées. Certaines n'ont pas été testées en raison du fait qu'elles ont été créées à la fin du projet et ne servent qu'à lire des données depuis notre base de données. Nous nous sommes concentrés sur les tests des fonctions de création / update / delete.

Voici un exemple de test que nous avons réalisé :

Nous avons tout d'abord défini une liste qui nous permet d'avoir accès aux données de l'utilisateur qui va nous servir de test.

```
const newCand = [
    "testCand1@test.com",
    "LeCand",
    "Jean-Test",
    "06-66-99-25-56",
    "c'estletestCand1",
];
```

Suite à cela nous avons écrit une fonction de test qui va insérer cet utilisateur dans les tables Utilisateur et Candidat. La fonction va ensuite lire la base pour récupérer la ligne qu'on vient de récupérer. On vérifie ensuite que l'id qu'on vient de récupérer correspond bien à l'id qui a été retourné dans user quand il a été créé.
L'utilisateur est ensuite supprimé des deux tables.

```
test("create candidat", async () => {
    const user = await userModel.create(newCand[0], newCand[1], newCand[2], newCand[3], newCand[4]);
    const candidat = await candidatModel.create(user);
    const result = await candidatModel.read(candidat);
    expect(result.id_candidat).toBe(user);
    await candidatModel.delete(candidat);
    await userModel.fullDelete(newCand[0]);
});
```

Pour réaliser les tests, nous avions des soucis lorsque nous lancions tous les tests d'un coup avec npm test. Nous avons donc opté pour une autre solution.

Nous avons défini les noms de nos fichiers de test comme tel :

- si c'est un fichier de test d'un modèle il s'appelle 'model*.test.js'
- si c'est un fichier de test d'une route il s'appelle 'route*.test.js'

Nous avons ensuite édité le fichier package.json pour y rajouter deux commandes permettant de lancer séparément les tests de modèles et les tests de routes.

```
"scripts": {
  "start": "node ./bin/www",
  "test": "jest",
  "test:routes": "jest --testPathPattern='routes.*\\.test\\.js'",
  "test:models": "jest --testPathPattern='model.*\\.test\\.js'"
},
```

Il faut donc utiliser les commandes :

- npm run test:routes pour tester les routes
- npm run test:models pour tester les modèles

Voici les taux de couverture pour les tests de modèles :

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	48.27	29.02	54.66	48.27	
admin.js	67.74	40	80	67.74	9,22,31-37,48,61
adresse.js	83.63	55	100	83.63	8,21,34,43,59,72,85,98,111
candidat.js	53.33	30	60	53.33	9,18-36,49,63
candidature.js	37.87	20.83	38.23	36.92	7,16-82,92,99-112,126,140
db.js	100	100	100	100	
fiche_poste.js	17.52	9.37	18.75	17.52	7,16-34,46,60,69-205
offre_emploi.js	15.88	7.5	15.51	16.03	7,16-51,63,77,87-240
organisation.js	53.73	31.81	63.63	53.73	8,22,35,44-50,61,74,87,96-128,140
piece_dossier.js	84.21	50	100	84.21	8,21,34,48,61,74
recruteur.js	45.9	25	50	45.9	9,18-23,34,44-49,60,70-76,89,102,111-130
statut_poste.js	82.6	60	100	82.6	8,20,28,42
type_metier.js	82.6	60	100	82.6	8,20,28,42
type_organisation.js	82.6	60	100	82.6	8,20,28,42
utilisateur.js	50.31	27.27	63.76	50.31	9,18-23,36,50,69,74,79-133,151,157,174,187,200,209-246,257,270,279-286,297,311,324,337

Les lignes non couvertes par les tests sont principalement les lignes qui traitent les erreurs. Pour les tester il aurait fallu essayer d'écrire des données non valides et vérifier que cela retournait une erreur. C'est donc pour cela que notre taux de couverture des branches est pour la majorité des fichiers assez faible. Cependant le taux de couverture pour les fonctions est lui élevé pour la majorité des fichiers.

Voici les taux de couverture pour les tests de routes :

myapp/routes	36.77	32.35	29.16	36.77	
admin.js	37.9	32	30.76	37.9	15,42-66,99-107,113-119,1
auth.js	51.4	43.33	57.14	51.4	17,27,39-42,70-71,73-74,8
candidat.js	50	46.66	25	50	19-22,27-30,37,72-88,96-1
index.js	100	100	100	100	
recruteur.js	29.46	24.28	23.8	29.46	19,26-29,36,58-65,72-80,8
users.js	30.33	27.08	22.72	30.33	13-16,21-24,33,50-60,67-8

Ici nous avons des taux de couverture beaucoup plus bas mais qui s'expliquent par plusieurs raisons :

Tout d'abord une bonne partie de ces routes sont des requêtes POST et M. Lounis nous a dit qu'il n'était pas nécessaire de tester ces requêtes.

Nous avons aussi quelques routes qui ne sont pas testées car nous avons récemment mutualisé les pages similaires des différents utilisateurs et nous n'avons pas eu le temps de réécrire les tests.