IUT Nantes
Pôle Sciences et technologie

Nantes Université

BUT2 Initiaux
SAÉ4.Real.01 : Qualité de developpement

Partie : R4.02 | MASI

LEPORT Clovis
MARTEL Floran
GROMARD Alexys
JOUAULT Lancelot
CHEVREUX Arthur

Année Scolaire 2023 - 2024
Réf : eq_01_01

Introduction

Dans le cadre de la SAE du 4ème semestre, nous avons réalisé un site web React.js permettant d'interagir avec des API REST. Nous avons également dû déployer nos propres API REST pour gérer les utilisateurs ainsi que les différentes informations qu'ils peuvent ajouter concernant un transport en commun (Affluence).

Nous avons donc choisi de développer le site web NaoLibre pour que ces utilisateurs puissent consulter le temps d'attente des bus et tram de leurs arrêts favoris et avoir des informations sur l'encombrement moyen d'une ligne. mais comme l'api de Naolibe n'indique pas l'encombrement des lignes. Nous avons alors créé une API pour gérer le compte des utilisateurs et permettre aux utilisateurs de noter l'encombrement de lignes.
Enfin nous avons aussi créé un autre API qui est un filtre de l'api de naolibe. Elle a pour but de remettre en forme les données et de faciliter l'utilisation de ces données.

Organisation :

Durant le projet nous avons appliqué la méthode agile pour avoir une organisation flexible durent le projet. de plus nous avons utilisé certain concept de scrum comme.
les concepte de scrum que nous avons appliqué sont:

- **Les Sprint**: pendant le projet et surtout à la fin quand les heures consacrées à la SAE était plus dense nous avons planifié des sprites d'une semaine. Grâce à cela nous avons pu planifier / répartir les tâches durent le projet et ne pas se faire surprendre par la date de fin.

- **Le Scrum master** : nous avons choisi d'avoir un scrum master puisque durant notre dernière SAE personne n'avait une vision globale du projet et chacun était focalisé sur ça tache sans penser à l'étape d'implémentation. ce qui a fait que nous avons été victime de l'effet tunnel. Floran MARTEL c'est proposé pour prendre le rôle de Scrum master et il avait donc pour mission de s'informer de l'avancement des différentes parties.

Tests :

Dans le but d'avoir un programme maintenable et pour garantir la qualité de l'application, nous avons utilisé les différentes méthodes de test vus en R4.02 :

- **Tests fonctionnels**
- **Tests structurels**
- **Tests de mutation**

Nous allons désormais voir pour 3 méthodes de notre modèle *Avis* comment nous avons procédé à l'utilisation des ces 3 approches.

1. Test fonctionnels:

Dans notre application nous avons une classe nommé "ModelAvis" qui à comme attribut "nomlignes" qui est le nom de la lignes qui à les avis et l'attribut "avis" qui est une list d'objet qui contient l'id du user qui a fait l'avis, la date à laquel à été fait l'avis, le jours ou l'avis il a été fait (exemple: mardi) et la note de l'avis qui indique le niveau d'encombrement.

Dans cette classe nous avons une méthode nommée moyenne, qui permet de retourner la moyenne des notes de l'attribut avis. Pour cela, elle vérifie si la liste n'est pas vide au quel cas elle renvoie null. Ensuite, elle fait la somme des avis de l'attribut. Enfin, elle renvoie la somme divisée par la longueur de la liste.

```
moyenne(){
    if (this.avis.length === 0){
        return null
    }
    let moy = 0
    this.avis.forEach(av => {
        moy += av.note
    });
    return moy / this.avis.length
}
```

Moyenne

1. Analyse Partitionnelle

Nous allons utiliser cette méthode pour montrer un exemple d'implémentation de test fonctionne dans notre projet.

Voici le tableau de l'analyse fonctionnel:

moyenne()				
	[]	X		
this.avis	[liste int entre 1 et 3]		X	
	[liste avec un int en dehors de 1 à 3]			X
	null	X		
return	Float entre 1 et 3		X	
	Err			X

On définit ici toutes les différentes valeurs que peut prendre la fonction en paramètre et les attributs de la classe. Puis, on détermine le retour de la méthode suite à son exécution. Ici, trois fin de fonctions sont possibles : null, un float. On peut donc ensuite, en déduire les tests à faire pour couvrir tous les retours possibles.

Grâce à ce tableau on peut identifier les cas de test suivant

1er cas de test :
input : ([{ notes : 3}, {notes : 3}]
oracle : (1,5)

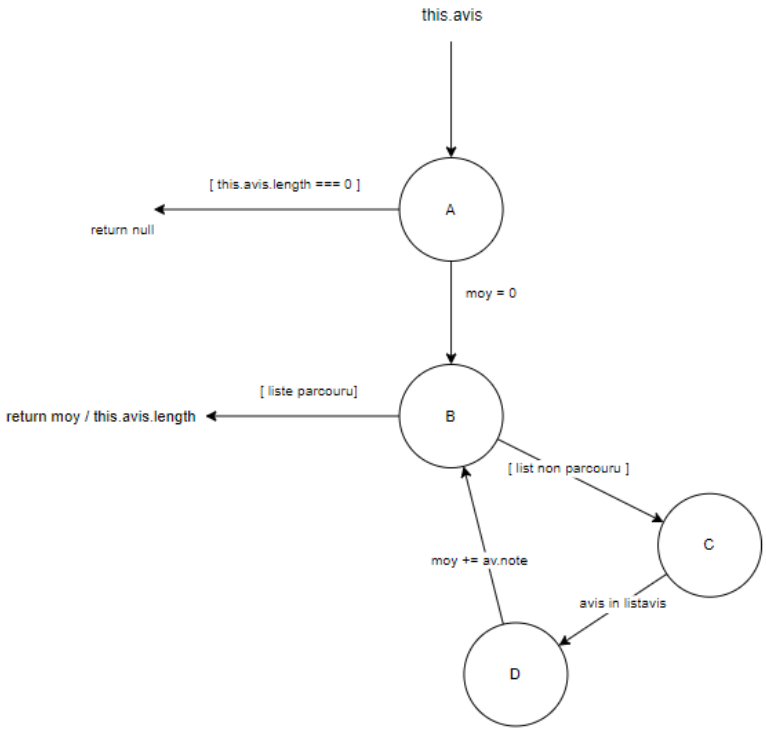
```
describe('ModelAvis', () => {
    describe('moyenne', () => {
        it('devrait retourner la moyenne des avis', () => {

            avis = [{
                    "iduser": "661590640d2e07b6883b4880",
                    "note": 3,
                    "date": "2024-04-09T19:07:05.973Z",
                    "dayweek": "Tuesday"
                    },
                    {
                    "iduser": "6615b5e6f09874cf462f14d1",
                    "note": 1,
                    "date": "2024-04-09T21:48:51.123Z",
                    "dayweek": "Tuesday"
                    }
                ]
            const avismod = new ModelAvis({nomLigne:"11", avis: avis });
            assert.strictEqual(avismod.moyenne(), 1.5);
        });
```

2ème cas de test :
input : ([])
oracl : (null)

```
it('devrait retourner null si aucun avis n\'est disponible', () => {
    const avis = new ModelAvis({nomLigne:"11", avis: [] });
    assert.strictEqual(avis.moyenne(), null);
});
```

2. Test structurel :

Voici le schéma structurel de la fonction moyenne :



Ce schéma nous permet de définir les chemins que parcourt la fonction en fonction des paramètres. On peut donc ensuite définir les paramètres à utiliser pour couvrir les chemins suivants : tous les nœuds, tous les arcs, tous les n-chemins. Ce qui nous permet de nous assurer d'avoir tester toutes les lignes du code.

Expression :
Tous les arcs : ABCD
Tous les noeuds : ABCD
Tous les 0 et 1 chemins : ABCD, AB , A

Oracle :
ct(dt([]), null) explore le chemin A
ct(dt([1,3]),1,5) explore le chemin ABCD
le chemin AB n'est pas explorable

On constate donc que l'on refait les même tests que pour les tests fonctionnelles

moyenneOftheHoures(date) :

Cette fonction permet de retourner la moyenne des avis à chaque heure de la journée à "date". Si il n'y a pas d'avis à une heure alors la moyenne renvois null.

```
moyenneOftheHoures(date){
        if (!(date instanceof Date)){
            throw Error("date is not a Date")
        }
        let moyHoure = new Array(24) //24 houre
        for (let pas = 0; pas < 24; pas++){
            moyHoure[pas] = {sum : 0, nbvalue : 0}
        }


        this.avis.forEach(av => {
            const avieDate = new Date(av.date)
            if (avieDate.getDate() === date.getDate()){
                moyHoure[avieDate.getHours()].sum += av.note
                moyHoure[avieDate.getHours()].nbvalue++
            }
        });

        //divisiton pour obtenire la moyenne
        const moyresult = moyHoure.map(moy =>{
            if (moy.value === 0){
                moy.sum = null
            }else{
                moy.sem /= moy.nb value
            }
            return moy.sum
        })
        return moyresult

    }
```

Elle fait :

1. Vérification de la validité de la date : La fonction commence par vérifier si l'argument `date` est bien une instance de l'objet `Date`. Si ce n'est pas le cas, elle lève une erreur.
2. Initialisation d'un tableau pour stocker les moyennes par heure : Un tableau `malheur` est initialisé avec une taille de 24, correspondant aux 24 heures d'une journée.
3. Parcours des avis : La fonction parcourt chaque avis (stockés dans `this.avis`). Pour chaque avis, elle vérifie si la date de l'avis correspond à la date passée en argument. Si c'est le cas, elle ajoute la note de l'avis à la somme correspondante dans `moyHoure` à l'heure de l'avis, et incrémente le nombre de valeurs pour cette heure.
4. Calcul des moyennes : La fonction parcourt ensuite le tableau `moyHoure` et calcule la moyenne de chaque heure en divisant la somme des notes par le nombre d'avis reçus pour cette heure. Si aucun avis n'a été reçu pour une heure donnée, la moyenne pour cette heure est définie comme null.
5. Retour des résultats : Les moyennes calculées sont stockées dans un nouveau tableau `moyresult`, qui est retourné à la fin de la fonction.

1. Analyse partitionnelle :

moyenneOfthe Houres(date)				
	2 (autre chose qu'un string)	X		
date	date		X	X
this.avis	[]	|		X
	une liste d'objet avis	|	X	
	throw Error()	X		
return	un array avec la lists des moyenne a chaque heures		X	X

Ici on à deux variables utilisées dans la fonction date et this.avis. Les retours sont soit un array avec la liste des avis qui peut être remplis de *null* si la date n'apparaît pas dans la liste. La fonction peut aussi lever une exception si jamais la date en paramètre n'est pas un objet *Date*.

cas de test

input : (2,[]) oracle : erreur
variable avis =

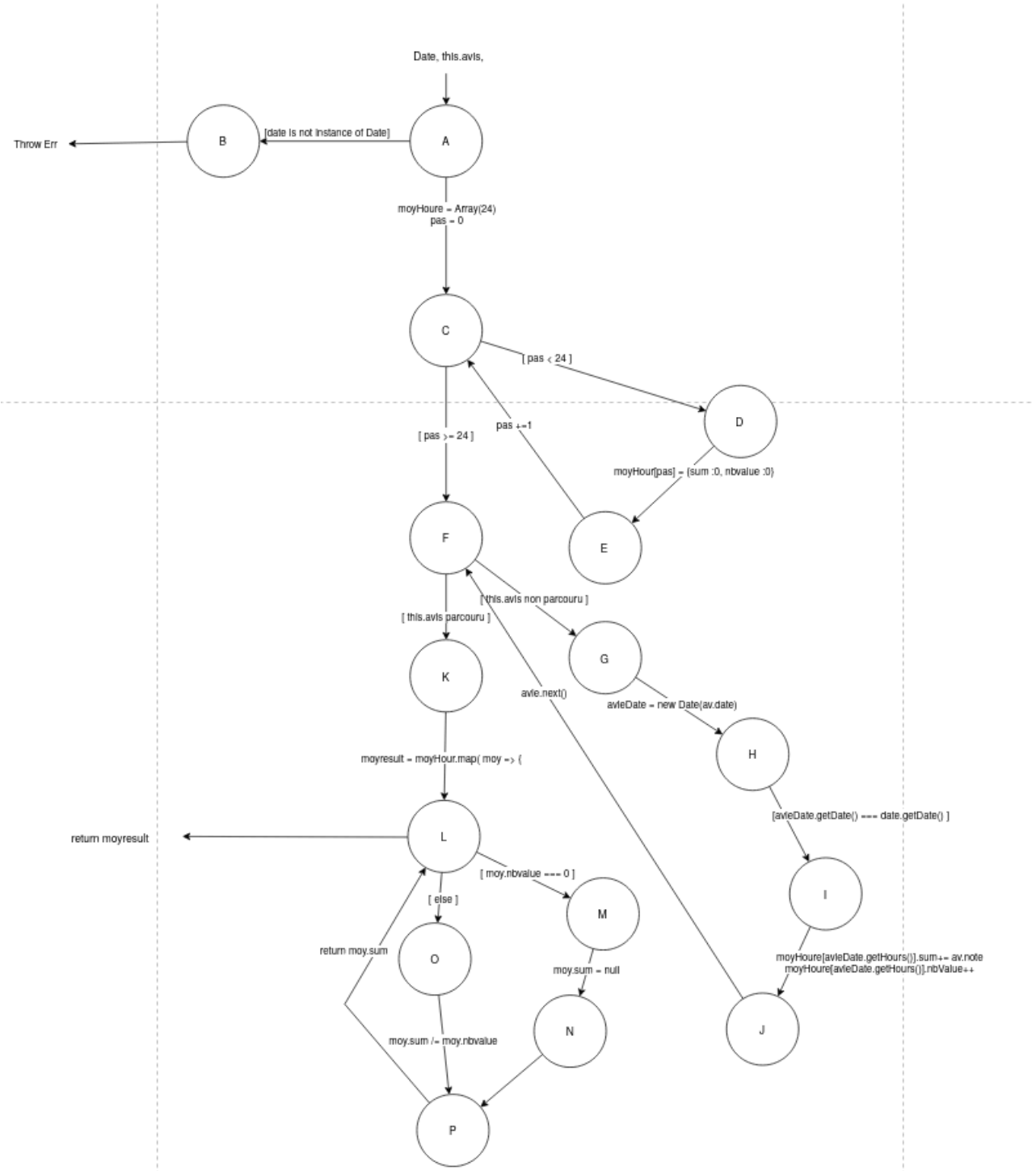
```
{
      "iduser":  "661590640d2e07b6883b4880",
      "note": 2,
      "date": "2024-04-09T19:07:05.973Z",
      "dayweek": "Tuesday"
    },
```

input:(new Date(“2024-04-09T19:07:05.973Z”),[avis ,avis])
oracle : [null,...2,null,null, null, null

input : (new Date(“2024-04-09T19:07:05.973Z”),[]) oracle : [null, …, null]

2. Test structurel :

Voici le schéma structurel de la fonction moyenneOftheHoures() :



chemin :
Tous les arcs : AB, ACDEFGHIJFKLMNOPL
Tous les noeuds : AB, ACDEFGHIJFKLMNOPL
Tous les 0 et 1-chemins : AB, ACDEFGHIJFKLMNOPL, ACDEKLMNOPL , ACDEFGHIJFKL, etc

Oracle :

ct(dt([“nimportequoi”,”encore nimportequoi”)]), Err) explore le chemin AB

ct(dt(Date(15/04/2024), {Avis(“15/04/2024 15:36”, 1), }), Array(null,null… 1, null, null…)) explore le chemin ACDEFGHIJFKLMNOPL

ct(dt(Date(15/04/2024), {Avis(“15/04/2024 15:36”, 1),“15/04/2024 6:36”, 2) }), Array(null,..,2,..,null…,1,null,null…)) explore le chemin ACDEKLMNOPL

ct(dt(Date(15/04/2024), {Avis(“15/04/2022 15:36”, 1),“18/06/2024 18:26”, 2) }), Array(null,..,null)) explore le chemin ACDEFGHIJFKL

3. constructor(Obj)

Le constructeur permet d'initialiser la ligne de l'avis et les avis de cette ligne.

```
constructor(obj) {
       if ('nom Line' in obj && 'avis' in obj) {
           if (typeof obj.nomLigne !== "string" || !Array.isArray(obj.avis)){
               throw new Error("Object attributes are not array")
           }
           this.nomLigne = obj.nomLigne;
           this.avis = obj.avis;
       }
       else {
           // error
           throw new Error('Invalid object type');
       }
   }
```

Le constructeur fait :

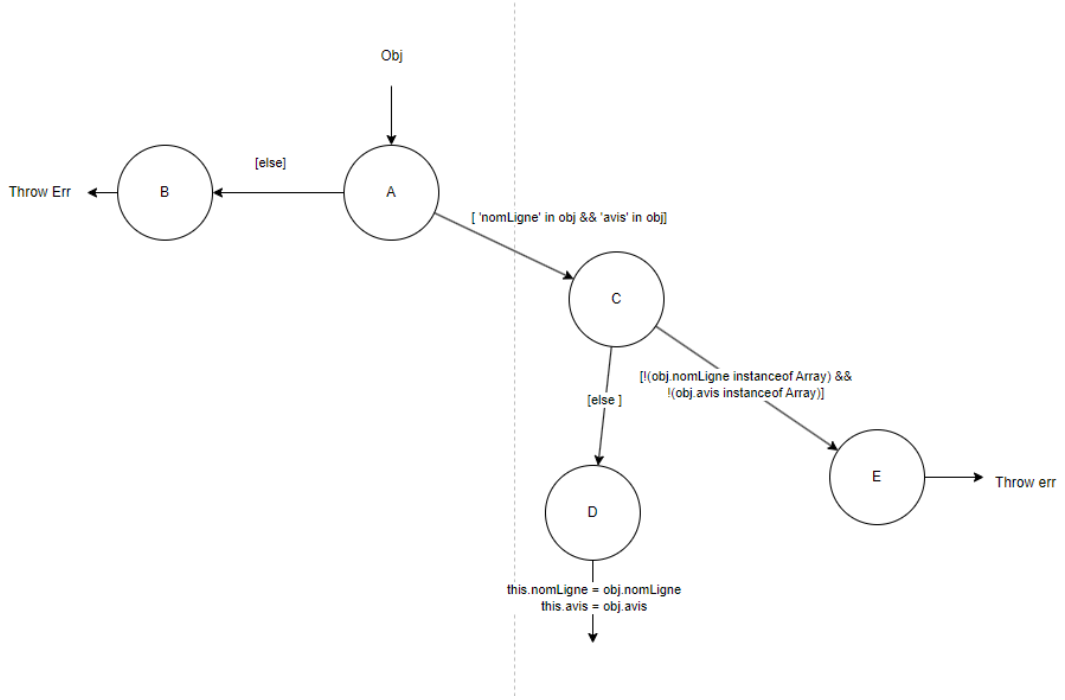
1. **Paramètres** : Le constructeur prend un objet `obj` en argument. Cet objet est supposé avoir deux propriétés : `nomLigne` (une chaîne de caractères) et `avis` (un tableau).
2. **Vérification des propriétés de l'objet** : Le constructeur vérifie d'abord si les propriétés nécessaires (`nomLigne` et `avis`) sont présentes dans l'objet passé en paramètre. Si ces propriétés sont absentes, il lance une erreur indiquant que le type d'objet passé en argument est invalide.
3. **Vérification des types de données** : Si les propriétés existent, le constructeur vérifie ensuite si `nomLigne` est une chaîne de caractères et si `avis` est un tableau. S'il détecte un type de données incorrect, il lance une erreur spécifiant que les attributs de l'objet ne sont pas conformes aux types attendus.
4. **Attribution des valeurs aux propriétés** : Si toutes les vérifications sont passées avec succès, le constructeur assigne les valeurs des propriétés `nomLigne` et `avis` de l'objet passé en paramètre aux propriétés correspondantes de l'instance nouvellement créée.
5. **Gestion des erreurs** : Si à un moment donné une condition n'est pas remplie, une erreur est lancée pour indiquer le problème.

1. Analyse partitionnelle :

Constructor()				
Obj	un objet avec un attribut nomLigne correct et un attribut avis correct	X		
	un objet avec un attribut nomLigne incorrect ou un attribut avis incorrect		X	
	un objet sans l'attribut nomLigne ou avis			X
return	Error('Object attributes are not array')		X	
	Error('Invalid object type')			X
	Les attributs sont affectés	X		

On voit que le constructeur prend en paramètre un obj dont il utilise les attributs pour initialiser les attributs nomLigne et avis. Ce constructeur soulève soit 2 exceptions différentes soit initialise bien les attributs.

2. Test structurel :



Les chemins :

Tous les arcs : ABC, ABD
Tous les noeuds : ABC, ABD
Tous les n-chemins : : ABC, ABD

Les oracles :

CT(dt(azeaz1441éé), Err)

CT(dt(Avis avec mauvais attributs), Err)

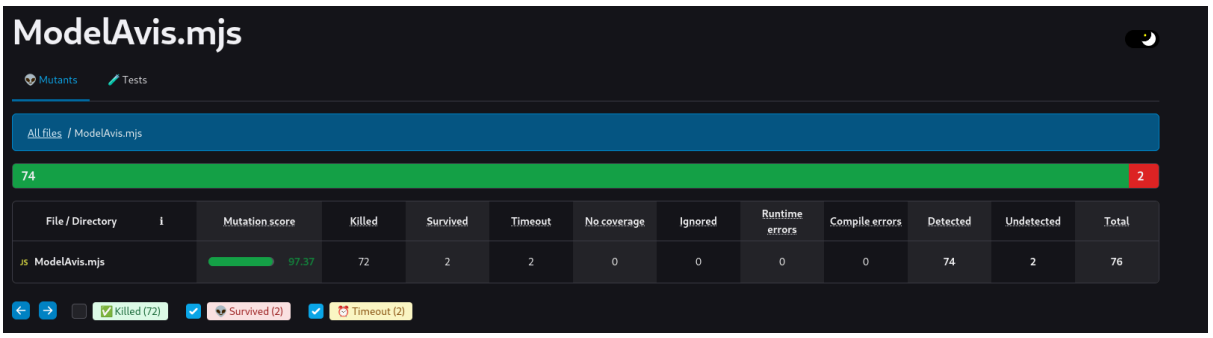
CT(dt(Avis)) pas erreur et l'attributs initialisés

On sait désormais les cas de test à réaliser.

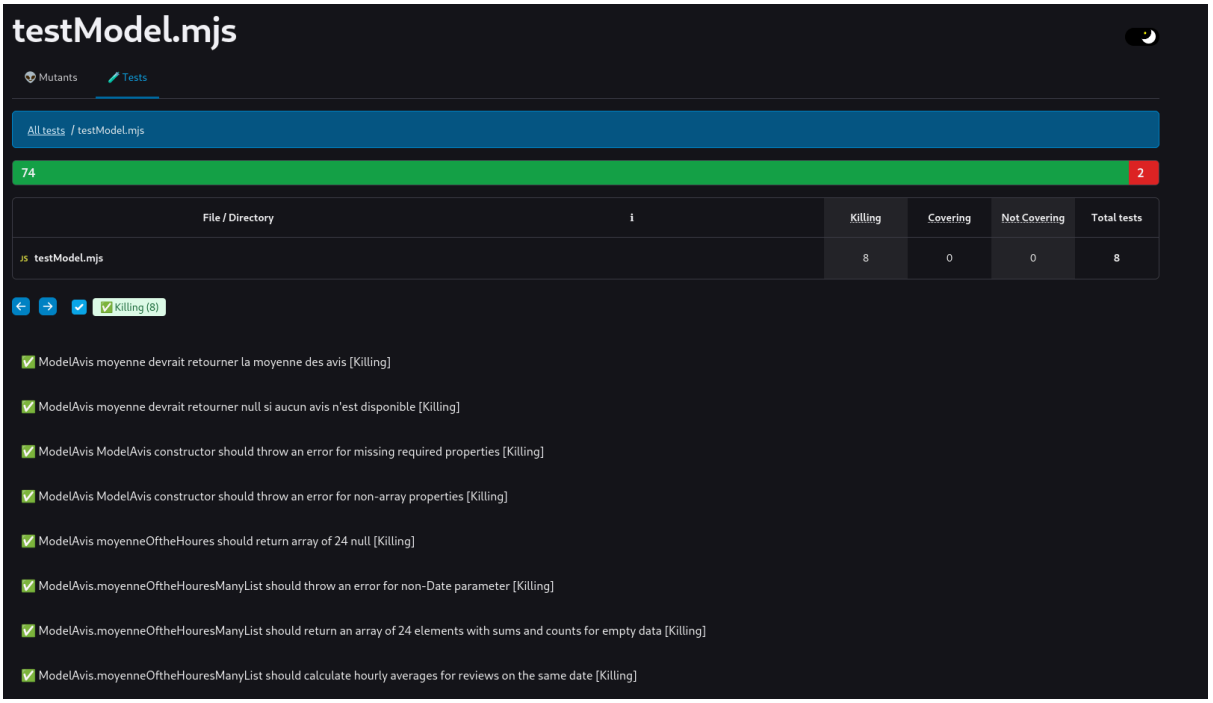
3. Analyse de mutation:

Pour effectuer des mutations sur notre code nous avons utilisé la librairie mocha qui permet de générer des mutations afin d'encore mieux tester l'application. Nous avons pu ainsi corriger les bugs potentiels de notre code. Cela nous à par exemple permis d'ajouter des levers d'exceptions à des endroits qui pouvaient poser problème.

Voici le score final des mutations: nous avons réussi à tuer 72 mutations sur 74.



Voici des exemples de mutants détectés :



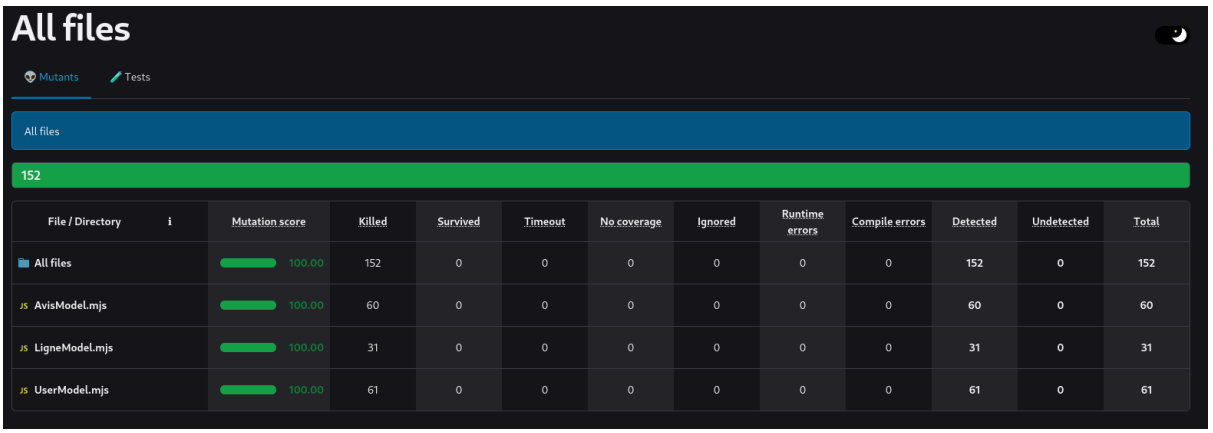
Les étapes de l'utilisation de l'analyse de mutation :

- On a rajouté les tests manquants et modifié une partie du code. (Utilisation d'un if inutile)
- On a atteint 97,37% avec comme erreur non résolu :
    - Une définition d'un Array(24) qui est utilisé pour réduire le coût, et 2 boucles qui sont testées en décrémentant qui donnent un timeout.

Cependant comme on vient de voir 2 mutants persistent en effet, si jamais la liste déclarée n'a pas une taille de 24 le code ne le detecte pas mais cela est normal car ça ne change rien à l'aboutissement de la méthode. On retrouve aussi des changements dans les constantes comme dans une boucle for où les pas sont dégressifs au lieu de progressif. Ce ne sont donc que des mutants qui soient n'impactent pas l'algorithme ou bien, n'arrivent pas en pratique.

```
55  -       let moyHoure = new Array(24) //24 houre
    +       let moyHoure = new Array() //24 houre
56  -       for (let pas = 0; pas < 24; pas++){
    +       for (let pas = 0; pas < 24; pas--){
```

Nous avons ensuite appliqué l'analyse de mutation pour tous les modèles :



Conclusion

Dans notre projet d'application web JavaScript, nous avons adopté une approche de développement itératif pour garantir la qualité et la non-régression du code. Cette méthode nous a permis de rester focalisés sur nos objectifs, d'obtenir des retours utilisateurs plus rapides, et de réduire les risques liés aux changements. Nous avons appliqué cette approche en découpant le projet en fonctionnalités, en les priorisant, puis en les développant par itérations successives.

Pour assurer la qualité de notre application, nous avons réalisé différents types de tests, tels que les tests unitaires, d'intégration, de bout en bout, de performance et de sécurité. Nous avons utilisé des approches telles que le test structurel, l'analyse fonctionnelle et les tests de mutation pour couvrir toutes les possibilités.

En illustrant notre démarche avec trois méthodes de notre modèle, nous avons détaillé leur analyse partitionnelle, leur schéma structurel et les résultats des tests de mutation. Ces exemples ont démontré l'efficacité de nos pratiques de test dans l'identification et la correction des bugs potentiels, renforçant ainsi la robustesse de notre application.

Malgré quelques mutants persistants, ceux-ci n'affectent pas significativement le fonctionnement de notre algorithme ou ne se produisent pas dans des situations réelles. En conclusion, l'intégration de pratiques de développement itératif et de tests

structuraux a été cruciale pour garantir la qualité, la fiabilité et la maintenabilité de notre application web JavaScript