

Conception agile de projets informatiques :

Planning poker

Responsable de TD : Valentin Lachand-Pascal

1. Table des matières

1. Table des matières.....	2
2. Présentation de notre projet.....	2
3. Fonctionnalités et outils.....	4
4. Architecture.....	5
5. Intégration continue.....	6
6. Améliorations possibles.....	7
7. Conclusion.....	7
8. Annexe.....	7

2. Présentation de notre projet

a) Contexte et choix

Ce projet est réalisé dans le cadre de la matière conception agile de projets informatiques. Celui-ci consiste à réaliser un planning poker. Une application dont le but est d'estimer la complexité de certaines tâches, et ce à plusieurs.

Nous avons réalisé ce projet en binôme, l'un des objectifs premier était de faire du "pair programming". Nous avons donc essayé de commencer par cette méthode. Cependant, nous nous sommes très vite rendu compte que nous n'étions pas en accord sur plusieurs choix dont le langage à choisir. Nous avons eu l'impression que chaque langage que nous choissions présentait des limites et nos compétences n'étaient pas similaires.

Nous avons alors pris la décision d'essayer chacune de notre côté, chacune sur une version différente et des langages différents et de continuer sur la version qui semblerait la plus fonctionnelle. Ilyna a essayé de travailler en JavaScript en local et Miléna en python sur une version en ligne grâce à socket.io et flask.

La version abandonnée fut celle en Python de Miléna, qui, bien qu'avancée, a rencontré quelques problèmes avec la partie sur les enregistrements JSON. Cette application était basée au départ sur un chat qui fonctionnait bien. Cette version offrait la possibilité d'entrer dans une room personnelle, avec un groupe composé d'un administrateur pour diriger la partie. Le code de la room était généré aléatoirement. On pouvait également avoir la possibilité de voter et avoir un rendu des résultats. La carte café ne fonctionnait pas et il était impossible de charger des fichiers. Cela devenait trop complexe et cette version commençait à prendre beaucoup trop de temps. Voici un lien qui redirige vers le dépôt de cette version python : <https://github.com/milignp/CAPL.git>

De l'autre côté, la version Javascript d'Ilyna fonctionnait correctement et semblait plus facile à finir. Nous avons donc décidé de continuer à travailler sur cette version à deux en se partageant les tâches pour aller plus vite. De plus, nous avons utilisé la méthode du Pair programming lorsque nous en avons l'occasion, tout en essayant de s'aider et

comprendre les parties de l'une et de l'autre. Voici le lien du dépôt github de notre projet : <https://github.com/Ilyna-MACHANE/Capi.git>.

De plus, nous avons à de nombreuses reprises utilisé ChatGPT, en l'utilisant comme une aide pour comprendre et résoudre les difficultés que l'on rencontrait et en ré-écrivant des résolutions qu'il proposait à certains de nos problèmes. Nous l'avons utilisé afin de déboguer certaines parties du code et parfois même pour générer du code : comme par exemple pour le minuteur.

b) Objectifs

Notre projet a pour objectif de produire un code simple, fonctionnel et compréhensible, aussi bien par nous que par les utilisateurs. Pour atteindre cet objectif, nous avons choisi de nous concentrer sur une architecture qui gère facilement les applications web dynamiques.

Le choix de JavaScript associé à HTML/CSS nous permet de construire une interface ergonomique, accessible et compréhensible par tous, offrant une expérience fluide et agréable.

Nous avons construit ce projet par étape :

1. Pouvoir entrer toutes les informations des joueurs
2. Faire en sorte que les cartes fonctionnent, et ce en appuyant sur l'image
3. Faire fonctionner les modes de jeu et respecter les tours
4. Mettre en place les tests et la documentation
5. Ajouter un minuteur
6. Afficher les tâches avec chargement d'un fichier backlog.json
7. Sauvegarder des résultats dans un fichier qui se télécharge
8. Faire en sorte que la carte café fonctionne correctement

En somme, nous avons essayé d'allier simplicité, ergonomie et fonctionnalité, en créant une application dynamique qui répond aux exigences d'une gestion collaborative et ludique des tâches.

c) Difficultés

Nous avons commencé ce projet avec la plus grande difficulté de cette année pour nous : GitHub. Au cours de nos études, nous n'avons jamais été confrontées à son utilisation. Nous avons donc dû apprendre sur le tas, grâce à certaines vidéos YouTube et les explications données pendant certains cours.

Nous avons pris beaucoup de temps à travailler sur la documentation, les tests ou encore à gérer les actions de déploiement. Si nous devions refaire ce projet, nous pensons que cela irait beaucoup plus vite car maîtrisons beaucoup mieux ces compétences.

3. Fonctionnalités et outils

Notre application prend en charge plusieurs fonctionnalités, d'abord la gestion des joueurs, leur nom et leurs votes. Ensuite, en ce qui concerne les modes de jeux, nous avons choisi le mode moyenne, et le mode unanimité nous était imposé. Nous avons choisi de ne

pas mettre en place un chat car nous partons du principe que les joueurs jouent dans la même pièce étant donné que notre application fonctionne en local.

a) Modes

Lors du mode unanimité, les joueurs choisissent chacun une carte. À chaque tour les deux joueurs aux extrémités (minimum et maximum) doivent discuter et justifier leur choix. Le tour se termine seulement lorsque tous les joueurs se mettent d'accord sur la même carte.

Pour le mode moyenne, le premier tour se passe exactement comme pour le mode unanimité, les joueurs des extrémités discutent. Au deuxième tour, chaque joueur est libre de choisir une carte, la fonction `gerermoyenne` se charge de calculer la moyenne des cartes choisies.

b) Téléchargement et enregistrement JSON

Nous avons décidé de mettre la liste des tâches à voter en backlog, et chacune porte un id. Une fois que le fichier est téléchargé, les tâches sont affichées une à une et ce grâce à l'incrémentation de ce dernier. Lorsqu'il n'y a pas plus de tâches à voter, le vote s'arrête. Il nous est ensuite possible de télécharger le document "resultat_final" qui regroupe les tâches et les votes finaux pour chaque tâche.

c) Carte café

En ce qui concerne le fonctionnement de la carte café, nous nous sommes dirigées vers une approche plus simple car nous avons passé beaucoup de temps dessus sans réussite. Elle permet aux joueurs de signaler qu'ils souhaitent faire une pause au cours de la session.

Lorsque tous les joueurs choisissent cette carte, une fenêtre de confirmation s'affiche pour demander s'ils veulent bien mettre la réunion en pause. Si la pause est validée, un fichier sous le nom de `backlog.json` est enregistré avec la progression : le nom de la tâche, les noms de joueurs... Cette partie peut être reprise plus tard grâce au bouton dans la page d'accueil "reprendre une partie" qui charge le fichier.

Cependant, si l'intégralité des joueurs n'a pas choisi la Carte Café, la pause ne sera pas déclenchée et nous n'avons pas géré ce cas. Dans l'ensemble cette carte fonctionne donc plutôt bien.

d) Minuteur

Nous avons décidé d'ajouter un minuteur, qui se présente plus sous forme de décompte. Celui-ci s'active lorsque les joueurs des extrémités doivent discuter. Il sert donc à poser une limite afin que les joueurs ne prennent pas trop de temps à débattre sur leur choix. Cependant, nous l'avons mis à trente secondes pour éviter de trop attendre lorsque nous essayons le jeu.

4. Architecture

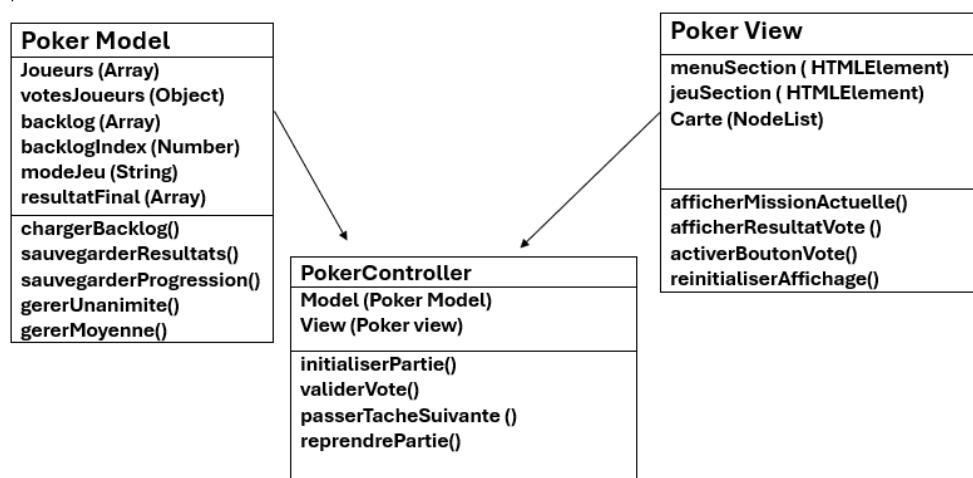
a) Fonctions et MVC

Notre projet est construit sur l'architecture du modèle MVC (Modèle-Vue-Contrôleur), c'est-à-dire que les fonctionnalités de cette application sont divisées en trois parties distinctes, bien que de notre côté nous n'ayons pas vraiment instauré de classes spécifiques pour chacune de ces catégories.

La partie Model représente les données du jeu. Nous avons les données telles que les joueurs, les votes, les tâches qui sont la plupart du temps structurées comme des listes ou tableaux (joueurs et backlog) et des objets (votesJoueurs).

La partie vue représente tout ce que les utilisateurs pourront voir. Cela est surtout défini par le fichier HTML et CSS et toutes ses interactions. Il s'agit donc du menu pour configurer le jeu. Les utilisateurs peuvent eux même configurer le nombre de joueurs, le chargement du backlog etc...Il y a également la zone de votes, quand les utilisateurs peuvent voter et cliquer sur les éléments du HTML.

Enfin, la partie contrôleur agit comme intermédiaire entre le modèle et la vue. Elle gère les événements des utilisateurs (vue) tout en actualisant les données qui sont contenues dans le modèle.



5. Intégration continue

a) Documentation

Lors de la mise en place de la documentation pour notre projet, nous avons rencontré plusieurs obstacles. Initialement, nous avons décidé d'intégrer un GitHub Action pour générer la documentation, sans comprendre qu'il était nécessaire de configurer un outil

comme JSDoc. Mais nous n'avions pas compris toutes les consignes et des documents manquants nous empêchaient d'avoir le résultat attendu. Nous avons ensuite essayé d'utiliser Doxygen, en raison d'une confusion entre les consignes du projet et les informations trouvées en ligne. Malheureusement, cela n'a pas donné les résultats escomptés car Doxygen ne prend pas les commentaires Js en compte.

Tous ces essais problématiques ont engendré de nombreux commits inutiles, ce qui a provoqué des conflits entre nos branches et rendu impossible les fusions (merge). Nous avons donc dû repartir avec JSDoc pour poser des bases plus solides, mais certains problèmes persistent encore, notamment liés à la configuration et à l'intégration de la documentation dans le flux du projet. Nous avons trouvé le problème après plusieurs heures passées dessus, le souci était simple mais a manqué à notre attention. Nous avons mis les mauvaises balises de documentation. Nous avons donc changé cela et tout fonctionnait parfaitement. Voici le lien vers la documentation : <https://ilyna-machane.github.io/Capi/>.

b) Tests unitaires

En ce qui concerne les tests unitaires, nous avons procédé de la même façon que pour les documentation mais cette fois-ci en utilisant Jest et en configurant le fichier node. Nous l'avons également automatisé grâce aux github actions. Nos tests principaux concernent les parties calcul de la moyenne, la détection de l'unanimité et aussi l'association entre un joueur et son vote. Cette partie ne nous a pas trop causé de problème, nous l'avons effectuée en classe et en demandant de l'aide au professeur.

6. Améliorations possibles

Pour commencer, l'amélioration que nous n'avons pas réussi à mettre en place est le fait de déployer notre application en ligne, nous pensons que plus de temps nous aurait permis de le faire. Miléna était sur la bonne voie pour le faire mais nous avons eu peur que cela prenne trop de temps. Deuxièmement, nous pensons que l'interface aussi pourrait être améliorée avec par exemple des menus.

7. Conclusion

Dans l'ensemble, ce projet répond à nos attentes. Grâce à celui-ci nous avons pu apprendre à travailler en groupe avec une bonne utilisation de GitHub et s'initier au pair programming. C'était la première fois que nous mettions en place des tests unitaires et une documentation. Certains points sont tout de même à améliorer, nous en sommes conscientes. Cependant nous sommes satisfaites de ce projet et de cette matière qui nous prépare bien au monde professionnel et à la gestion de projet.

8. Annexe

a) Ancienne version de Milena

Create or Join a Room

Name:

Room Code (for joining):

Meeting Title:

Select Game Mode:

Unanimity ▼

Unanimity

Average

Majority

Room: PLPZ

Task: Configurer un serveur

Task Title:

Vote:

1

2

3

5

8

13

21

Start Vote

Stop Vote

Chat Room: PZDD

Meeting Title: Sprint1

Activate the Button

Milena: has entered the room
System: The admin of this room is Milena.

b) Modes

Mode unanimité :

Votez :

Tache : Tache 1 : Configurer le serveur



Valider

Tous les joueurs sont d'accord : 20.

Continuer

Nouvelle Partie

Mode moyenne :

Votez :

Tache : Tache 2 : Intégrer l'authentification



Valider

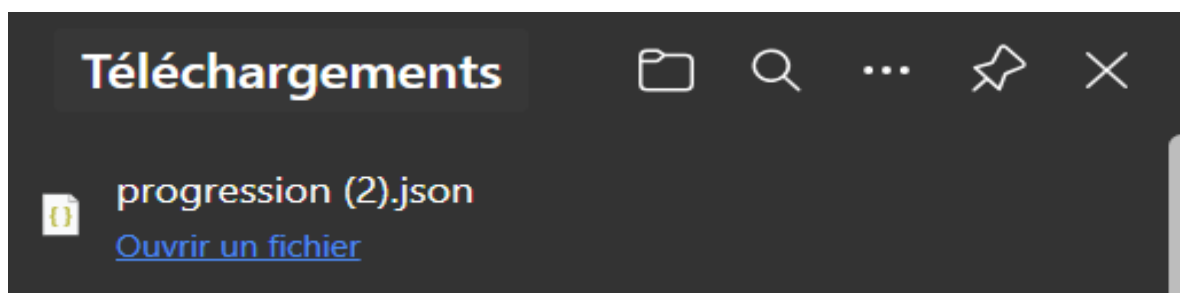
La moyenne des votes est : 70.00.

Continuer

Nouvelle Partie

c) Enregistrement json et carte café

```
{
  "joueurs": [
    "Milena",
    "Ilyna"
  ],
  "votesJoueurs": {
    "Milena": "café",
    "Ilyna": "café"
  },
  "backlog": [
    {
      "id": 1,
      "feature": "Tache 1 : Configurer le serveur"
    },
    {
      "id": 2,
      "feature": "Tache 2 : Intégrer l'authentification"
    },
    {
      "id": 3,
      "feature": "Tache 3 : Déploiement sur le cloud"
    }
  ],
  "backlogIndex": 1,
  "modeJeu": "moyenne",
  "tacheActuelle": "Tache 2 : Intégrer l'authentification",
  "notesJouees": [
    {
      "tache": "Tache 1 : Configurer le serveur",
      "note": "30.00"
    },
    {
      "tache": "Tache 2 : Intégrer l'authentification",
      "note": "café"
    }
  ]
}
```



d) Documentation

This screenshot shows the top status bar of a GitHub repository. It features a search bar with the text 'Go to file', a tab labeled 't', a plus sign, and a dropdown menu with a green background and the text '<> Code'. Below this, a dark notification box states 'All checks have passed' with a close button. It lists '2 successful checks':

- ✓ **Generate JSDoc Documentation / build (push)** Successful in 48s [Details](#)
- ✓ **Tests unitaire Javascript / test (push)** Successful in 10s [Details](#)

At the bottom of the notification, it says 'test fails v1' and '10 minutes ago'.

This screenshot shows the 'Actions' tab in a GitHub repository. On the left, there's a sidebar with 'All workflows' and a 'New workflow' button. Below it, a list of workflows is shown: 'Generate JSDoc Documentation', 'pages-build-deployment', and 'Tests unitaire Javascript'. Under 'Management', there are links for 'Caches', 'Deployments', 'Attestations', 'Runners', 'Usage metrics', and 'Performance metrics'. The main area displays a list of workflow runs for 'Generate JSDoc Documentation':

- ✓ **Merge branch 'main' of https://github.com/Ilyna-MA...** (last week, 19s)
- ✓ **Create node.js.yml** (2 weeks ago, 16s)
- ✓ **docu** (2 weeks ago, 21s)
- ✓ **Merge pull request #2 from Ilyna-MACHANE/master** (2 weeks ago, 22s)
- ✗ **Create main.yml** (2 weeks ago, 17s)

This screenshot shows the 'Workflow file' view for the 'Generate JSDoc Documentation' workflow. The left sidebar has 'Summary', 'Jobs' (with 'build' selected), 'Run details', 'Usage', and 'Workflow file'. The main area displays the workflow file content:

```
1 name: Generate JSDoc Documentation
2
3 on:
4   push:
5     branches:
6       - main
7
8 jobs:
9   build:
10    runs-on: ubuntu-latest
11
12    steps:
13      - name: Checkout repository
14        uses: actions/checkout@v4
15
16      - name: Setup Node
17        uses: actions/setup-node@v4
18
19      - name: Install JSDoc
20        run: npm install -g jsdoc
21
22      - name: Generate Documentation
23        run: jsdoc *.js
24
25      - name: Deploy Documentation
26
27        uses: peaceiris/actions-gh-pages@v2
28        with:
29          github_token: ${{ secrets.GITHUB_TOKEN }}
30          publish_dir: ./out
```

← ↻ 🔒

https://ilyna-machane.github.io/Capi/app.module.js.html

🔍 ⭐ ⚙️ | ☆ ⋮

Module: app.js

Fichier principal pour gérer l'application Planning Poker.

Author:

Ilyna Machane - Milena Gordien Piquet

Source:

app.js, line 1

Methods

(inner) afficherFinDePartie()

Gère la fin de la partie en désactivant les options de vote et en affichant les options de clôture. Appelée lorsque toutes les missions ont été terminées ou lorsqu'une condition de fin de partie est atteinte. Les actions effectuées sont les suivantes : Désactive le bouton "Valider Vote" pour empêcher tout nouveau vote. Affiche les boutons "Suivant" et "Nouvelle Partie" pour permettre à l'utilisateur de naviguer ou de recommencer. Si toutes les missions sont terminées : Désactive le bouton "Suivant". Affiche le bouton "Télécharger Résultats" pour permettre de sauvegarder les résultats finaux dans un fichier JSON.

Source:

app.js, line 522

See:

sauvegarderResultats

Throws:

Home

Modules

app.js

Events

btnChargerBacklog

btnDemarrer

btnNouvellePartie

btnReprendrePartie

btnSuivant

btnTelechargerResultats

btnValiderJoueurs

btnValiderVote

carte

fichierProgressionInput

e) Test unitaires

```

1 import { obtenirJoueurParVote, calculerMoyenne, verifierUnanimité } from './functions';
2
3 test('obtenirJoueurParVote - retourne le pseudo du joueur pour un vote donné', () => {
4   const votes = { Alice: 5, Bob: 8, Charlie: 3 };
5   expect(obtenirJoueurParVote(votes, 8)).toBe('Bob');
6   expect(obtenirJoueurParVote(votes, 3)).toBe('Charlie');
7   expect(obtenirJoueurParVote(votes, 10)).toBeUndefined();
8 });
9
10 test('calculerMoyenne - calcule correctement la moyenne des votes', () => {
11   const votes = [5, 8, 3];
12   expect(calculerMoyenne(votes)).toBe('5.33');
13 });
14
15 test('verifierUnanimité - vérifie si tous les votes sont identiques', () => {
16   const votesIdentiques = [5, 5, 5];
17   const votesDifférents = [5, 8, 5];
18   expect(verifierUnanimité(votesIdentiques)).toBe(true);
19 });

```

```

s (2 ms)
✓ verifierUnanimité - vérifie si tous les votes sont identiques (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        3.448 s
Ran all test suites.
PS C:\Users\ilyna\OneDrive\Bureau\capi\Capi>

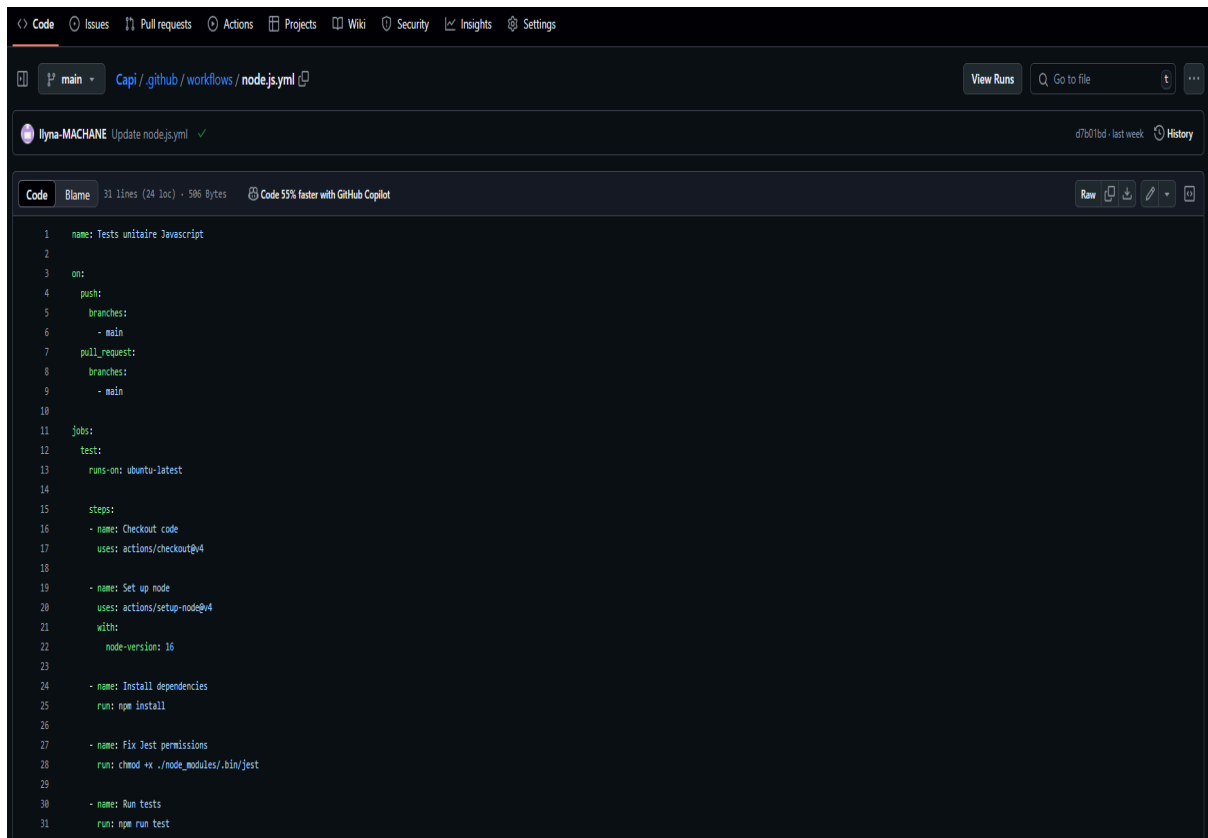
```

```

PASS ./functions.test.js
Tests pour les fonctions principales du jeu
✓ gererUnanimité - détecte une unanimité (6 ms)
✓ gererUnanimité - détecte un désaccord (2 ms)
✓ gererMoyenne - calcule correctement la moyenne (1 ms)
✓ gererMoyenne - fonctionne avec des votes identiques (1 ms)
✓ obtenirJoueurParVote - trouve le joueur correspondant à un vote (2 ms)
✓ obtenirJoueurParVote - retourne null si aucun joueur ne correspond (1 ms)

Test Suites: 1 passed, 1 total
Tests:       6 passed, 6 total
Snapshots:  0 total

```

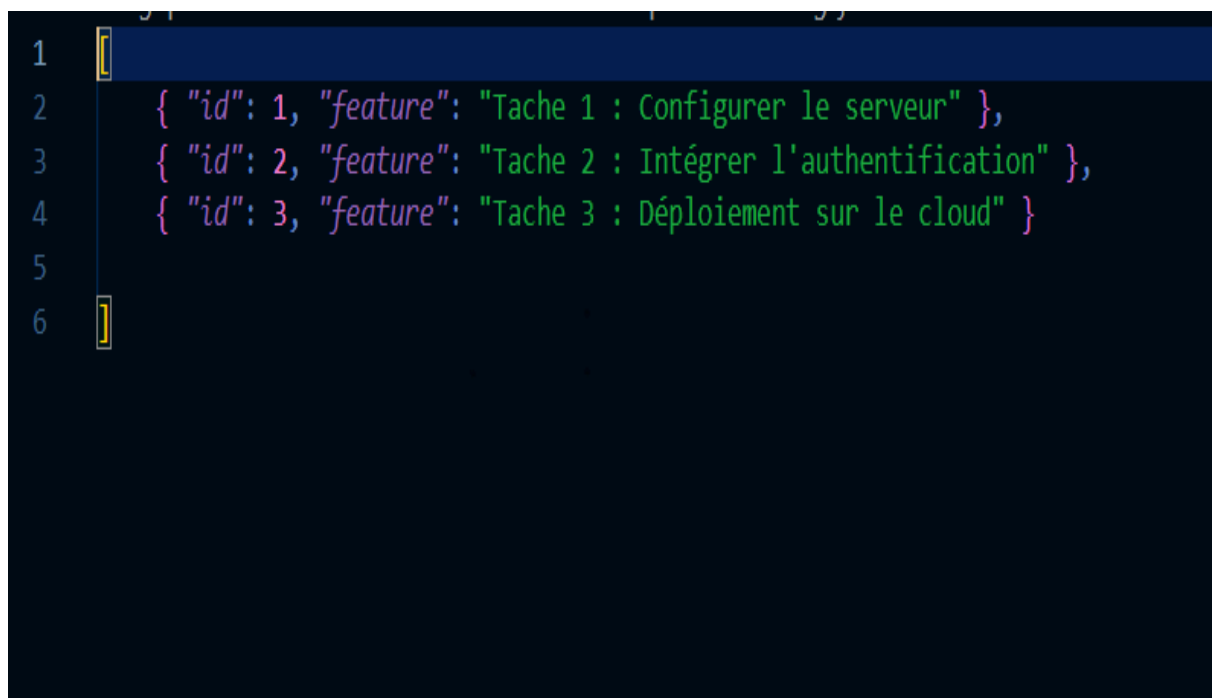


The screenshot shows a GitHub Actions workflow file named `node.js.yml` in the `main` branch. The workflow is triggered on push to the `main` branch or pull request to the `main` branch. It consists of a single job named `test` that runs on `ubuntu-latest`. The job has three steps: 1. Checkout code using `actions/checkout@v4`. 2. Set up node using `actions/setup-node@v4` with `node-version: 16`. 3. Install dependencies using `npm install`. The job also includes a `run` step for `npm run test`. The workflow is authored by `Ilyna-MACHANE` and is located at `d7b01bd`.

```
1 name: Tests unitaire Javascript
2
3 on:
4   push:
5     branches:
6       - main
7   pull_request:
8     branches:
9       - main
10
11 jobs:
12   test:
13     runs-on: ubuntu-latest
14
15     steps:
16       - name: Checkout code
17         uses: actions/checkout@v4
18
19       - name: Set up node
20         uses: actions/setup-node@v4
21         with:
22           node-version: 16
23
24       - name: Install dependencies
25         run: npm install
26
27       - name: Fix Jest permissions
28         run: chmod +x ./node_modules/.bin/jest
29
30       - name: Run tests
31         run: npm run test
```

f) Téléchargement et enregistrement JSON

Backlog.json avant réunion



The screenshot shows a JSON file named `Backlog.json` with the following content:

```
1 [
2   { "id": 1, "feature": "Tache 1 : Configurer le serveur" },
3   { "id": 2, "feature": "Tache 2 : Intégrer l'authentification" },
4   { "id": 3, "feature": "Tache 3 : Déploiement sur le cloud" }
5 ]
6
```

Compte rendu d'une réunion

```
C: > Users > gopit > Downloads > {} resultat_final (24).json > ...  
1  [  
2  {  
3    "tache": "Tache 1 : Configurer le serveur",  
4    "note": 20  
5  },  
6  {  
7    "tache": "Tache 2 : Intégrer l'authentification",  
8    "note": 8  
9  },  
10 {  
11   "tache": "Tache 3 : Déploiement sur le cloud",  
12   "note": 5  
13 }  
14 ]
```

g) Versions du projet

Planification Poker

Nb joueurs :

Mode de jeu :

Planification Poker

Votez :

0

1

2

3

5

8


13

20

40

100

a, c'est à vous de voter.



Planning Poker

Nombre de joueurs :

Ex : 3

Valider

Mode de jeu :

Unanimité

Charger un fichier JSON :

Choisir un fichier

Aucun fichier choisi

Charger Backlog

Reprendre une Partie


Nouvelle Partie

Planning Poker - Ilyna Machane / Milena Gordien Piquet

Planning Poker

Votez :

Tache : Tache 1 : Configurer le serveur

0	1	2	3	5
8	13	20	40	100
				

Valider

Temps écoulé ! Vous pouvez voter.