Rapport de projet - Planning poker

-

Rémi Galant et Thomas Grosjean

1. Introduction, Contexte et Organisation

L'objectif de ce projet est de réaliser et implémenter une application fonctionnelle de Planning Poker, à savoir une application de vote permettant à l'utilisateur d'estimer la difficulté d'une tâche proposée par un "scrum master", c'est-à-dire un maître de salle qui doit pouvoir créer la session et proposer une tâche à estimer . L'estimation de la tâche se fait en sélectionnant une carte, allant de 0 à 100, 0 représentant une tâche qui semble triviale à l'utilisateur qui la catégorise comme telle là où 100 représente une tâche insurmontable. Ce projet a été réalisé à deux avec une équipe composée de Rémi Galant et Thomas Grosjean. Ce projet présente de nombreux défis sur le plan organisationnel, c'est un projet lourd qui demande du temps de conception, de développement ainsi que d'apprentissage de certaines technologies comme Tkinter que nous avons utilisées pour la première fois au cours de la réalisation de notre application. C'est pourquoi il a fallu diviser les tâches pour avancer efficacement et mettre à profit aux mieux nos capacités individuelles, l'utilisation de Github fût aussi une technologie majeure dans l'organisation du projet car celui-ci nous a permit d'avancer chacun de notre côté tout en mettant à disposition notre travail à notre binôme en "pushant" ce qui était fait sur Github. Nous avons décidé d'organiser notre travail sous forme de grosses sessions hebdomadaires, nous savions tous les deux que nous préférions travailler par sessions longues en dépit de courtes sessions plus régulières. Cette méthode nous permettait d'avancer chacun de notre côté par longue session avant de mettre à disposition notre travail sur le git pour que l'autre puisse reprendre à partir de cette nouvelle base. Une communication efficace nous a également permis de nous tenir informer de nos avancées personnelles et également d'insister sur des difficultés que l'un pouvait avoir, afin que l'autre puisse revenir sur le point qui pouvait poser problème afin d'éviter de rester bloqué sur certains aspects de développement. Le fait que l'on se connaisse bien avant même d'avoir débuté le projet a beaucoup facilité l'aspect communication.

2. Choix Techniques et Architecture

Langage

Le premier choix à faire fut celui du langage. Celui-ci nous a causé de nombreuses hésitations, n'étant pas habitués à la création d'interface graphique, nous comptions

initialement partir sur des langages Web, qui semblaient les plus adaptés à la création d'interface graphique. Le Web avait aussi cet aspect confortable au niveau du déploiement et de la facilité de faire interagir différents utilisateurs entre eux en temps réel. C'est pourquoi nous voulions partir sur un projet réalisé sous NodeJS, pour créer une application web dynamique en exécutant du JavaScript côté serveur, ainsi que React pour le côté client et la création d'interface graphique. Cependant, ces deux technologies nous posent un problème simple mais majeur : aucun de nous deux n'avait auparavant utilisé ni React ni NodeJS, nous faisant abandonner cette possibilité.
Nous avons donc décidé de nous concentrer sur un langage que nous maîtrisons tous les deux bien plus, à savoir Python. Quelques recherches nous ont suffit pour découvrir une bibliothèque de python qui allait nous être très utile pour toute la partie interface graphique qui n'est autre que Tkinter, qui nous semblait plus simple d'apprentissage que des technologies et bibliothèques javascript comme Node et React. Nous avons donc décidé d'oublier la partie déploiement web du projet pour une utilisation plus locale de l'application, avec une interface graphique simple et intuitive sous Tkinter, une technologie donc adaptée et plus simple d'utilisation si on laisse de côté la partie web.
Pour la gestion des données, nous utilisons à la fois SQLite pour stocker les informations structurées, par exemple les utilisateurs ou encore les sessions actives de manière sécurisée et accessible depuis Python. Ainsi que JSON, pour importer le backlog des sessions, c'est-à-dire les user stories sur lesquelles les joueurs vont voter. Le JSON permet ainsi de préparer facilement différentes sessions et de conserver un historique des tâches à traiter, tout en restant directement manipulable depuis notre code Python.

Architecture Logicielle

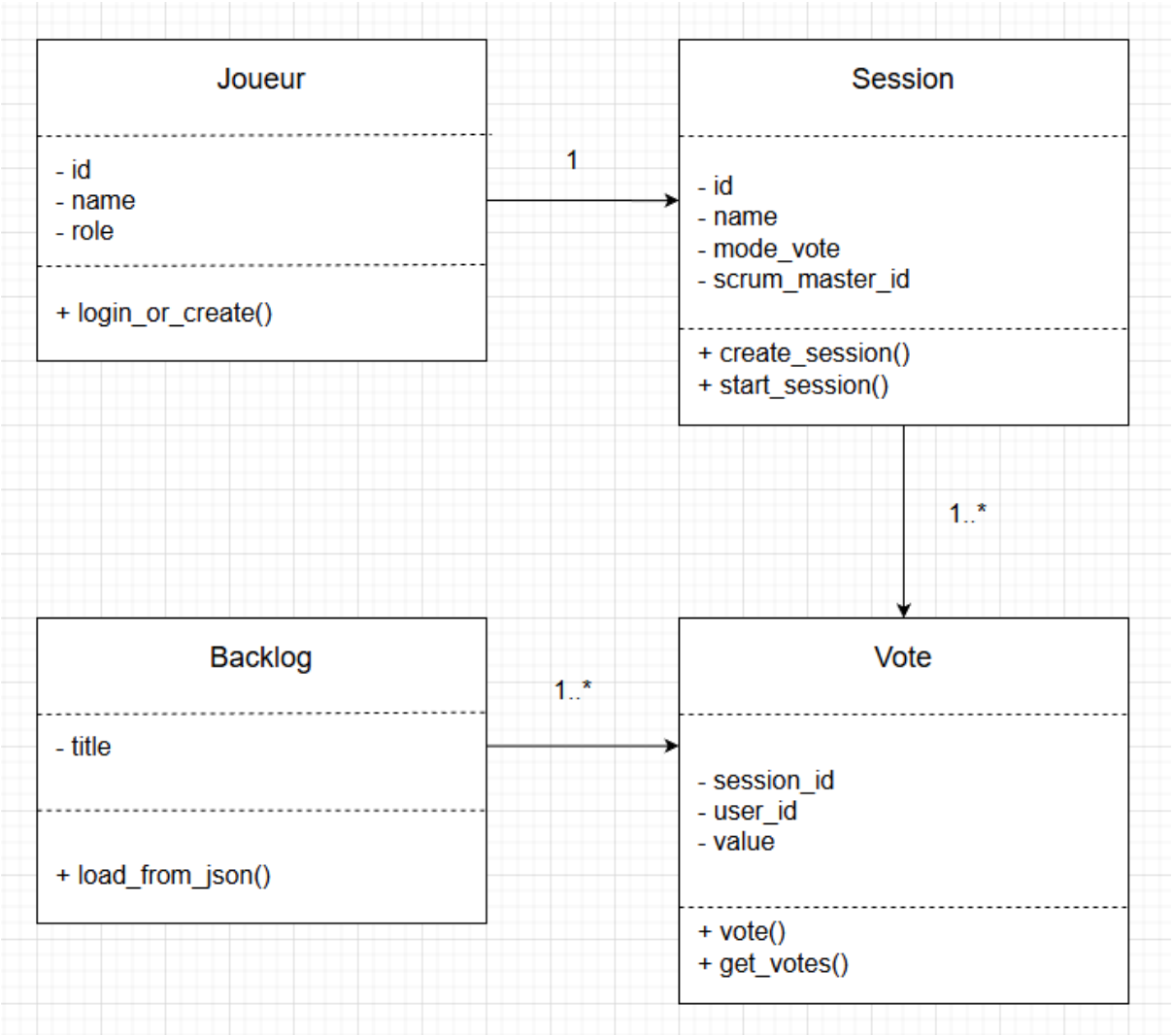
Notre projet suit une architecture MVC simplifiée, adaptée à la création d'une interface graphique avec Tkinter et à la gestion des données avec à la fois une base de donnée SQLite et le fichier JSON, l'architecture est la suivante :

- **Model** : Il regroupe toutes les fonctions d'intéractions avec SQLite et JSON, comme par exemple la création d'utilisateurs, la gestion des sessions ou encore le chargement des user stories.
- **View** : C'est la partie interface graphique réalisée avec Tkinter. Cette interface présente différents écrans de connexions, un menu principal avec les différentes sessions publiques, la création de session si vous êtes enregistré en tant que scrum master, un écran de votes avec les différentes cartes de poker ou encore un écran d'affichage des résultats.
- **Controller** : Dans notre implémentation, le Contrôleur est intégré à la Vue, c'est pourquoi on parle de MVC simplifiée. Les fonctions Tkinter appellent directement le Modèle pour exécuter les actions demandées par l'utilisateur, comme rejoindre une session ou voter. Dans une version plus évoluée, cette

logique pourrait être centralisée dans un fichier séparé pour améliorer la maintenabilité.

Avec cette architecture, notre projet possède une séparation claire entre l'interface et les données et permet un compromis entre organisation et simplicité, selon moi adapté aux besoins du projet.

Modélisation



Le diagramme ci-dessus présente les entités principales de notre application Planning Poker et leurs relations. Nous avons sélectionné uniquement les parties clés pour expliquer le modèle de données, sans inclure tous les détails techniques.

1. Classes principales :

- Joueur : représente un participant de la session.

Attributs : id, name, role (user ou scrum master)

Méthode principale : login_or_create() pour connecter ou créer un utilisateur.

- Session : correspond à une session de vote.

Attributs : id, name, mode_vote (Strict, Moyenne, Médiane), scrum_master_id

Méthodes : create_session(), start_session() pour initialiser la session et lancer le vote.

- Backlog / User Story : contient les tâches ou user stories sur lesquelles les joueurs votent.

Attribut : title

Méthode : load_from_json() pour importer le backlog depuis un fichier JSON.

- Vote : représente un vote d'un joueur pour une user story spécifique.

Attributs : session_id, user_id, value

Méthodes : vote() pour enregistrer un vote, get_votes() pour récupérer tous les votes d'une session.

2. Relations

- Joueur → Session : chaque session a exactement un Scrum Master, d'où la cardinalité
- Session → Vote : chaque session contient au moins un vote, et plusieurs votes sont possibles (un par joueur et par user story).
- Backlog → Vote : chaque user story peut être votée par plusieurs joueurs.

3. Utilisation du JSON

Le JSON est utilisé pour importer le backlog des sessions, c'est-à-dire la liste des user stories sur lesquelles les joueurs vont voter. Cette approche permet de préparer facilement différentes sessions et de conserver un historique clair des tâches à traiter.

Un deuxième fichier JSON est également généré pour enregistrer les résultats d'une session.

Gestion des données

Backlog.json :


```
{
  "issues": [
    { "title": "Création de la base de données" },
    { "title": "Design de l'interface graphique" },
    { "title": "Système de vote tour par tour" },
    { "title": "Exportation des résultats finaux" }
  ]
}
```


Le backlog contient la liste des user story à estimer lors de la session de Planning Poker. Il est structuré sous forme de dictionnaire avec une clé “issues” qui contient la liste des tâches.

Celui-ci est chargé grâce à la fonction load_json() dans notre code :

```
def load_json():
    global backlog
    path = filedialog.askopenfilename(filetypes=[("JSON files", "*.json")])
    if path:
        with open(path, 'r', encoding='utf-8') as f:
            backlog = json.load(f).get("issues", [])
        messagebox.showinfo("Succès", f"{len(backlog)} tâches importées.")

tk.Button(self.main_frame, text="Charger le Backlog JSON", command=load_json, bg="#D5DBDB").pack(pady=10)
```

L’utilisateur peut sélectionner le fichier JSON grâce à une fenêtre de dialogue puis est lu avec json.load() qui récupère les différentes tâches via la clé “issues”.

Cette méthode permet une flexibilité évidente car le backlog peut facilement être remplacé par un autre fichier JSON sans modifier le code ce qui permet au scrum master d’importer ses propres user storys qu’il aura préalablement défini.

resultats_session.json :

Après chaque tour de vote, un second fichier JSON est généré en enregistrant le résultat de la session de Planning Poker ayant eu lieu. Cette sauvegarde permet de conserver l’historique des votes par tâches afin de pouvoir les relire ultérieurement.

```python
def save_results_to_json(self, task_title, score, votes_detail):
    """Sauvegarde les résultats définitifs dans un fichier JSON externe."""
    filename = "resultats_session.json"
    new_entry = {
        "tache": task_title,
        "resultat_final": score,
        "details_votes": {u: v for u, v in votes_detail}
    }

    data = []
    if os.path.exists(filename):
        with open(filename, 'r', encoding='utf-8') as f:
            try: data = json.load(f)
            except: data = []

    data.append(new_entry)
    with open(filename, 'w', encoding='utf-8') as f:
        json.dump(data, f, indent=4, ensure_ascii=False)
```

La fonction prend en paramètres le titre de la tâche, le résultat final calculé selon le mode choisi (Strict/ Moyenne/ Médiane) et le détail des notes individuelles, la fonction va ensuite créer un dictionnaire associant chaque joueur à son vote, si le fichier existe déjà, il est chargé avec json.load() et les nouvelles données sont réécrites dans le fichier avec json.dump(). Si il n'existe pas, alors il est créé automatiquement.

Exemple de contenu généré :


```json
[
    {
        "tache": "Creation de la base de donnees",
        "resultat_final": "Débat requis",
        "details_votes": {
            "tigre": "20",
            "tigron": "1",
            "ligre": "8"
        }
    },
    {
        "tache": "Design de l'interface graphique",
        "resultat_final": "Débat requis",
        "details_votes": {
            "tigre": "3",
            "tigron": "8",
            "ligre": "13"
        }
    },
    {
        "tache": "Systeme de vote tour par tour",
        "resultat_final": "Débat requis",
        "details_votes": {
            "tigre": "3",
            "tigron": "0",
            "ligre": "Cafe"
        }
    },
    {
        "tache": "Exportation des resultats finaux",
        "resultat_final": 0,
        "details_votes": {
            "tigre": "0",
            "tigron": "0",
            "ligre": "0"
        }
    }
]
```

3. Mise en place de l'Intégration Continue

Workflow

Pour automatiser les tests unitaires ainsi que la génération de documentation, une pipeline d'intégration continue (CI) a été mise en place à l'aide de GitHub Actions. Cette pipeline est déclenchée automatiquement à chaque push sur la branche main. L'objectif est de s'assurer que chaque modification du code est testée et documentée.

Le code source est récupéré depuis le dépôt GitHub afin d'être utilisé dans le pipeline :

```
15          steps:
16          - name: Checkout repository
17            uses: actions/checkout@v2
18            with:
19              fetch-depth: 0
```

On installe ensuite les outils nécessaire à la génération de la documentation à savoir Doxygen et Graphviz (Dot)

```
29          - name: Install Doxygen
30            run: sudo apt-get install doxygen -y
31
32          - name : Install Dot
33            run : sudo apt-get install graphviz -y
```

La documentation HTML est générée automatiquement à partir du code source en Python grâce à Doxygen puis déployer automatiquement sur GitHub Pages afin d'être consultable par tous :

```
35          - name: Generate Documentation
36            run: doxygen conf/Doxyfile
37
38          - name: Deploy Documentation
39            uses: peaceiris/actions-gh-pages@v3
40            with:
41              github_token: ${{ secrets.GITHUB_TOKEN }}
42              publish_dir: ./documentation
```

Tests Unitaires

Afin de garantir la qualité du code ainsi que la validité de la logique métier, nous avons mis en place un jeu de tests unitaires avec la bibliothèque Python unittest. Les tests sont automatiquement exécutés à chaque intégration, ce qui permet de détecter toute régression ou erreur dans le code.

Les tests couvrent plusieurs aspects essentiels du projet, par exemple la fonction login_or_create() est testée pour s'assurer qu'un utilisateur peut bien être créé s'il n'existe pas encore, et si un utilisateur existant est correctement récupéré :

```python
def test_login_or_create(self):
    """Teste la création et la récupération d'un utilisateur"""
    uid, role = login_or_create(self.cur, self.conn, "Rémi", "scrum_master")
    self.assertIsNotNone(uid)
    self.assertEqual(role, "scrum_master")
```

ou encore la logique de vote est également vérifiée pour s'assurer que ceux-ci sont correctement enregistrés, mis à jour et récupérés :

```python
def test_voting_logic(self):
    """Vérifie l'enregistrement et la mise à jour d'un vote"""
    # 1. Créer les données nécessaires
    uid, _ = login_or_create(self.cur, self.conn, "Thomas", "user")
    sid = create_session(self.cur, self.conn, "Test Session", "Strict", uid)

    # 2. Effectuer un premier vote
    vote(self.cur, self.conn, sid, uid, "5")

    # 3. Vérifier que le vote existe
    votes = get_votes(self.cur, sid)
    self.assertEqual(len(votes), 1, "Le vote devrait être enregistré")
    self.assertEqual(votes[0][0], "Thomas")
    self.assertEqual(votes[0][1], "5")

    # 4. Modifier le vote (Mise à jour)
    vote(self.cur, self.conn, sid, uid, "13")
    votes_updated = get_votes(self.cur, sid)
```

Tous les tests sont exécutés dans une base de données en mémoire, ce qui évite de modifier la base réelle et garantit l'isolation de chaque test. Cela permet au pipeline de vérifier automatiquement la logique métier du projet à chaque modification du code.

Génération de Documentation

Nous utilisons Doxygen pour générer la documentation du projet, la configuration de celui-ci est définie dans le fichier Doxyfile du Git (dans le dossier conf/).

Les fichiers Python sont inclus grâce à la commande INPUT = main et les diagrammes de classes et de relations sont activés grâce aux commandes HAVE_DOT = YES, CALL_GRAPH = YES.

La documentation est ensuite générée au format HTML afin d'être accessible à tous. Grâce à notre Pipeline d'Intégration Continue, cette documentation est automatiquement mise à jour à chaque push sur la branche principale (main) ce qui facilite la lecture et la maintenance du projet.

4. Manuel Utilisateur et Fonctionnalités

Tout d'abord pour exécuter le projet localement l'utilisateur doit avoir python 3.10 ou supérieur d'installer. On utilise les bibliothèques standards : tkinter, sqlite3 et json.

Installation et Lancement

Clonage du dépôt :

- **git clone https://github.com/ThomasGrosjean42/Planning_Poker.git**
- **cd main/**
- **python main/gui.py**

Modes de jeu et règles

L'application contient trois modes de calcul pour l'estimation des tâches des user stories (définis dans le fichier backlog). On choisi ce mode lors de la création de la session :

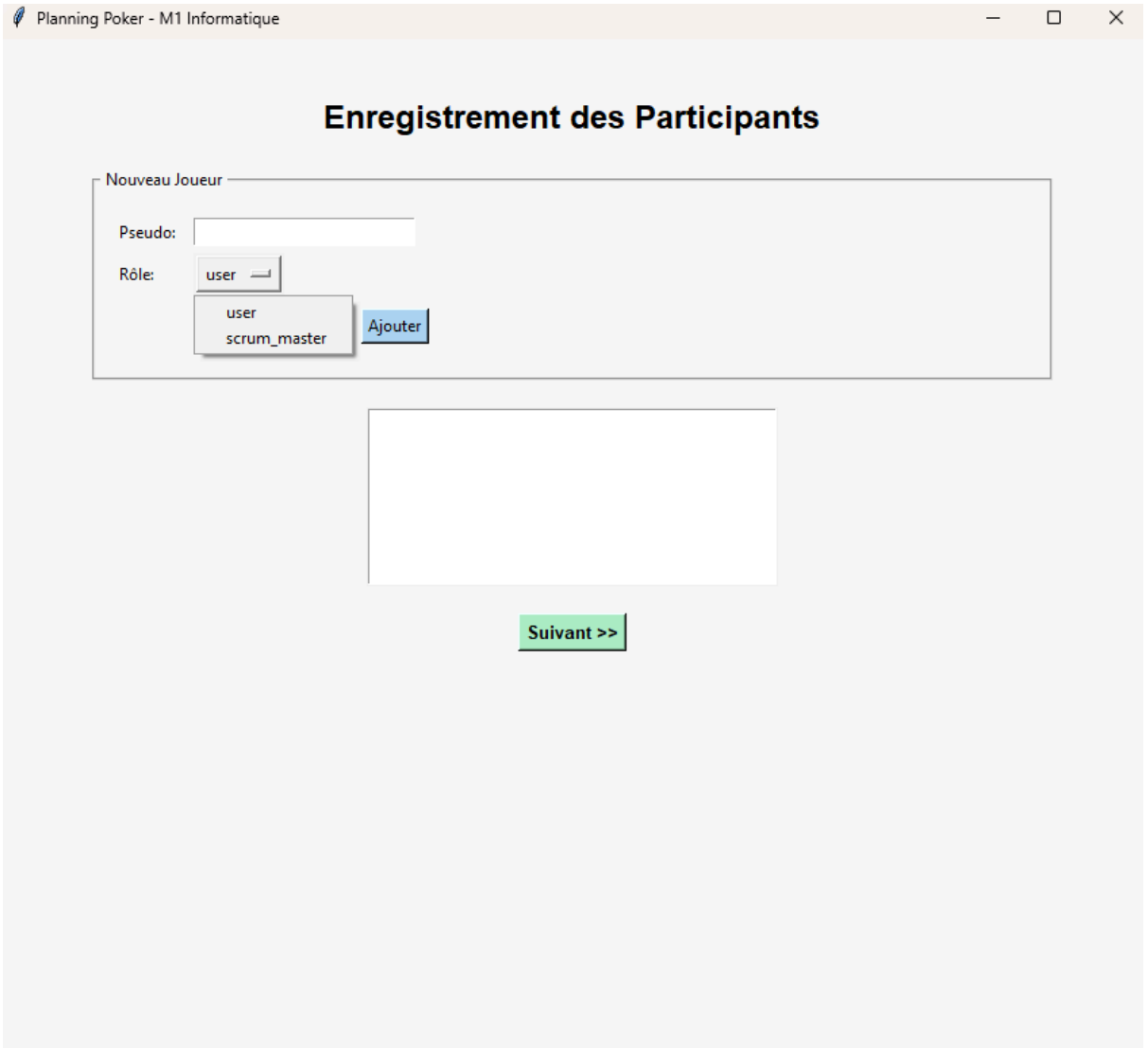
- Mode strict (unanimité) : Tous les participants doivent avoir la même valeur, sinon on affiche "Débat requis".
- Mode Moyenne : Calcul de la moyenne de tous les votes des participants de la session en cours.
- Mode Médiane : Affiche la valeur centrale de la valeur des votes de la session

Dans notre application, une fois que tous les joueurs ont voté, le résultat est calculé et affiché selon le mode choisi. Le scrum_master peut alors décider de passer à la user story suivante ou non. Cela dépend si un consensus a été atteint ou pas. Dans le cas où ça n'a pas été atteint ("Débat requis"), le système actuel invite les joueurs

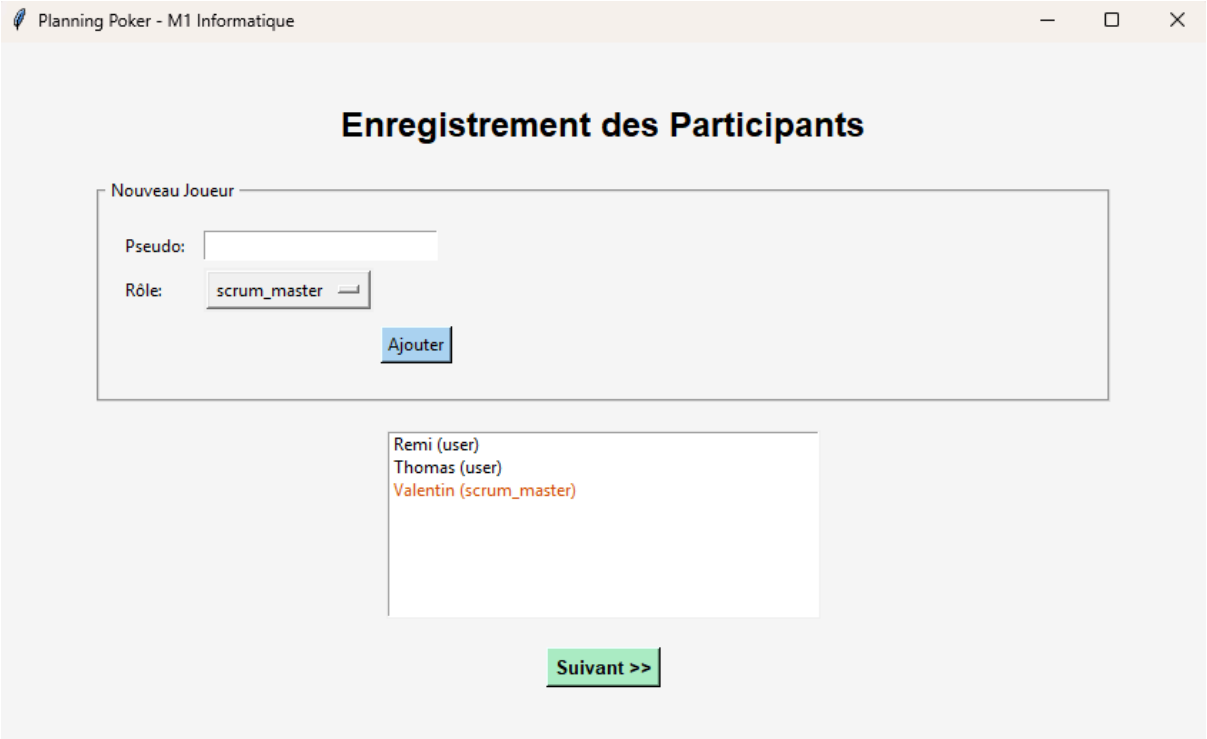
à discuter de la tâche. Pour faire un nouveau vote sur cette tâche il faut relancer une session. C'est un point qu'on pourrait améliorer en ajoutant un bouton "Reset votes" pour éviter cette manipulation.

Interface

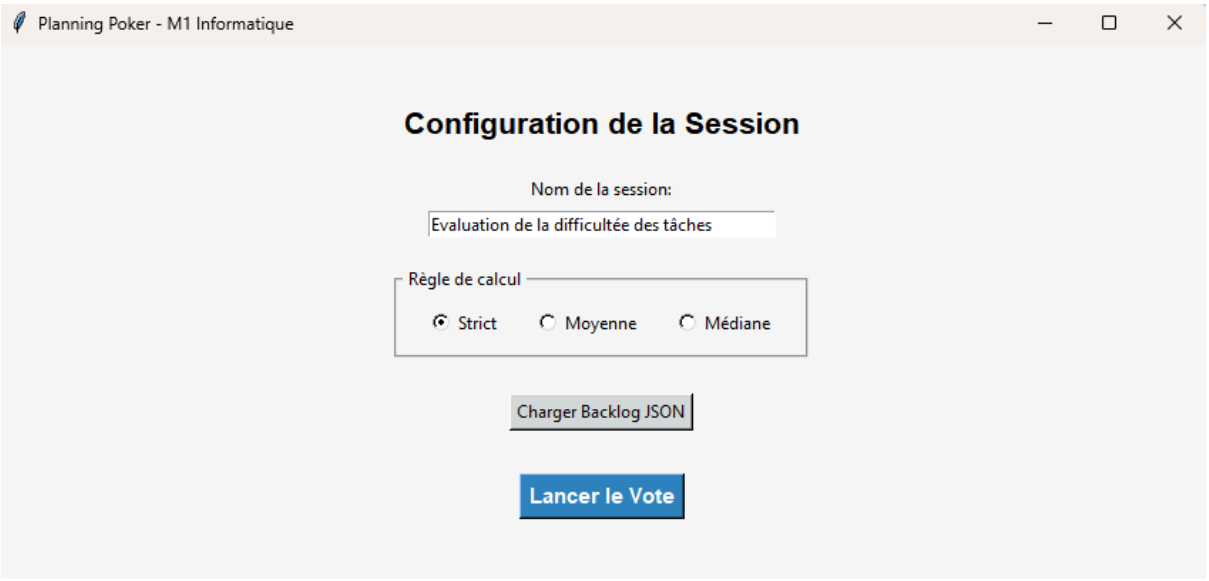
Voici comment se déroule une session de Planning Poker sur notre application :
L'utilisateur arrive premièrement sur un écran d'accueil "Enregistrement des Participants" ou il peut ajouter le nombre d'utilisateurs souhaité en inscrivant un "Pseudo", et en lui associant un "Rôle" (user/scrum_master) avant d'appuyer sur le bouton "Ajouter".



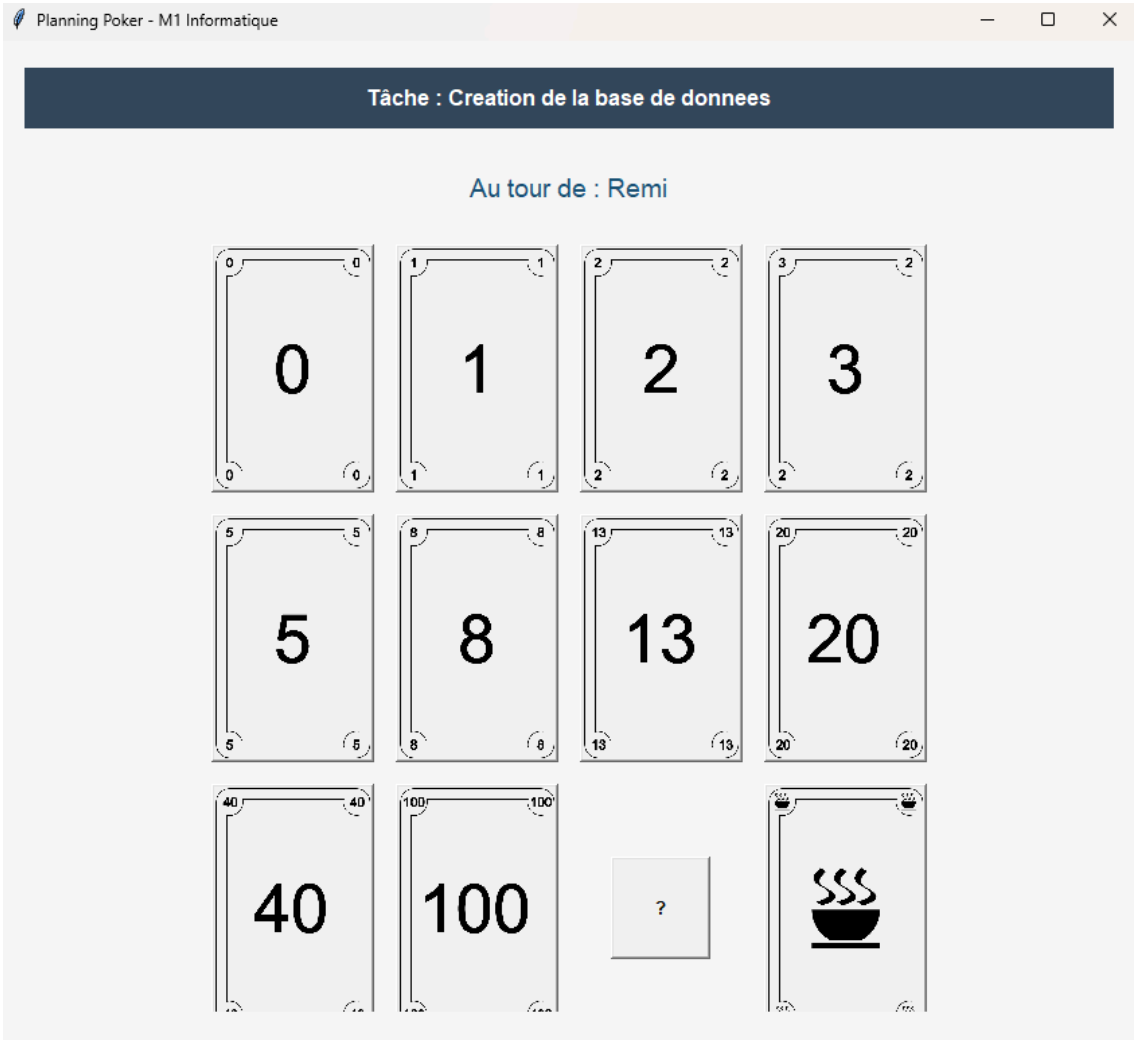
Une fois tous les utilisateurs ajoutés, il suffit de cliquer sur "Suivant >>"



Sur ce second interface ”Configuration de la Session”, le maître de session est invité à nommer la session de la façon qu’il désire, sélectionner le mode de calcul (pour cet exemple on restera en “Strict”) et charger le Backlog JSON contenant les user storys à afficher lors de la session :



Une fois le JSON chargé, il suffit de cliquer sur “Lancer le Vote” pour arriver sur le prochain interface, ou les utilisateurs préalablement enregistrés pourront à tour de rôle sélectionner une carte de poker signifiant le niveau de difficulté de la tâche affichée en haut de l’écran :



Une fois que tous les utilisateurs ont voté, un nouvel écran affiche les résultats du tour, on peut y voir chaque utilisateur avec sa note associée. En mode Strict, si les utilisateurs n'ont pas tous la même évaluation de la difficulté, le Résultat Final affiche "Débat requis"

Au contraire, si tous les utilisateurs estime la même difficulté pour la tâche donnée, Le Résultat Final affichera cette estimation (en mode Moyenne c'est la moyenne des votes qui sera affichée et en mode Médiane la médiane).

Une fois toutes les tâches évaluées, un popup "Terminé" apparaîtra au milieu de l'écran et l'utilisateur sera ramené sur l'écran d'enregistrement des participants après avoir cliqué sur "OK".