Sprint final

C'est l'heure du sprint final de votre projet ! Il ne vous reste que les récits d'utilisateur suivants à programmer dans votre application :

En tant que : employé
Je veux : représentation graphique interactive de la serre
afin de : visualiser en temps réel l'état des systèmes de régulation climatique (fenêtres, chauffages, ventilateurs, arroseurs, lumière)

En tant que : employé
Je veux : utiliser des contrôles graphiques intuitifs
afin de : activer ou désactiver les systèmes climatiques de la serre.

En tant que : employé
Je veux : visualiser les conseils d'ajustement de température offerts par le système
afin de : bien ajuster les paramètres climatiques de la serre

En tant que : employé
Je veux : déclencher la lecture de données météo
afin de : de tester les recommandations du programme de façon réaliste.

Travail à effectuer

Avec la même équipe que celle du travail 2, vous devrez procéder au développement des fonctionnalités listées plus haut.

À partir du code de votre TP2, vous devez implémenter ces fonctionnalités dans une application **WPF** en **C#**.

Votre remise devra être faite sous forme d'un **lien vers votre repository Github** contenant le code **ainsi qu'un exécutable (.exe)** de l'application.

Attention, toute modification au repository après la date limite entraînera une pénalité de retard.

Votre code du TP2 est votre point de départ.

TODO

1. **Nouveau Repository**

   Copiez le code de votre TP2 dans un nouveau repository nommé : **14E-TP3**

   Votre application devra utiliser une base de données locale **MongoDB** avec la chaîne de connexion suivante : **mongodb://localhost:27017/TP3DB**

   N'oubliez pas d'inclure un dossier contenant vos collections MongoDB dans votre Repo.

2. **Intégration continue (15%) :**

   À l'aide des Github actions, vous devez mettre en place un mécanisme d'intégration continue dans votre projet. Celui-ci doit-inclure :

   - Une validation des tests unitaires à **chaque création de pull request.**

     Si ce workflow échoue, une "github issue" doit-être créée automatiquement.

   - Un **badge github** doit indiquer l'état des tests dans le readme du repository.

3. Ajout de fonctionnalités (40%) :

À partir de la mise en situation du dernier TP ainsi que des récits utilisateurs fournis, vous devez programmer les fonctionnalités suivantes :

a) Représentation graphique interactive de la serre

- On doit y voir les systèmes climatiques suivants :
  - Fenêtres
  - Chauffage
  - Ventilateur
  - Arrosage
  - Lumière
- On doit voir si le système est activé ou non

b) Activation des systèmes climatiques

- Il doit-être facile et instinctif d'activer les systèmes climatiques directement via l'interface graphique de la serre.

c) Consulter les conseils d'ajustement climatiques offerts par le système

- Automate doit présenter des suggestions d'ajustement climatique afin d'aider les employés à optimiser les conditions de la serre. Par exemple :
  - Activer le chauffage
  - Fermer les fenêtres
  - Démarrer les ventilateurs
- Vous devez faire un minimum de recherche afin de trouver les conditions optimales d'un plan de tomate.
- Pour ne pas trop complexifié le travail, vous devez vous concentrer sur les paramètres suivants :
  - Température en Celsius
  - Humidité en %
  - Luminosité en Lux (pas le champion gossant à League of legends)

4. **Lecture des données météorologiques :**

Votre programme doit inclure un bouton qui déclenche la lecture des données météorologiques fournies dans le fichier ***tempData.csv***

Le bouton de lecture des données météorologiques a pour objectif d'intégrer des données fictives dans votre programme pour simuler une serre fonctionnant en temps réel.

- **Déclenchement** :
  - Lorsqu'un utilisateur clique sur le bouton, votre programme doit commencer à lire les données climatiques ligne par ligne (chaque ligne représente une mesure horaire simulée).
  - La lecture doit se faire progressivement : une ligne toutes les 10 secondes.
- **Mise à jour des données :**
  - À chaque lecture, le programme doit interpréter les données de température, humidité et lumière contenues dans la ligne.
  - Les informations lues doivent être affichées dans l'interface.
- **Recommandations climatiques :**
  - En fonction des données interprétées, votre système doit afficher des suggestions d'ajustements climatiques pour maintenir les conditions idéales dans la serre.

N'oubliez pas d'inclure une façon d'arrêter la lecture des données.

5. **Qualité du code (25%):**

Votre code doit suivre les principes du clean code vus en classe.

Vous utilisez le **GitHub flow** tout au long du projet (une branche par fonctionnalité, des **Pull Request**, revues de code, etc.)

6. **Tests unitaires (15%)**

Vous devez poursuivre la batterie de tests unitaires du TP2

Ceux-ci doivent couvrir l'ensemble du code que vous ajoutez pendant le sprint et celui du dernier sprint !

Vous serez évalué sur la **qualité** et la **pertinence** de vos tests.

Vous ne pouvez pas utiliser de BD de tests !

7. **Mocking (5%)**

Vos tests doivent utiliser au minimum deux fois le principe de **mocking** via l'extension **moq.**

Certains tests de la classe doivent utiliser un mock de votre classe d'accès au données (DAL ou autre).

8. *BONUS CI/CD (5%)*

Ajoutez à votre pipeline CI/CD :
- Créez automatiquement une nouvelle release github contenant l'exécutable de votre projet (avec ses références incluses dans le .exe) après **une pull request** ou un **push** incluant un commit avec le tag **"TP3-V*"** (l'étoile signifiant le numéro de version).

Remise

UN SEUL membre de l'équipe doit remettre un fichier .txt contenant le lien vers le repository de votre équipe sur LÉA d'ici **Mardi 17 décembre 2023 23:59** (+-4 semaines).

Grille de correction

Repository de remise	Présence d'un fichier texte avec le lien menant au repository de l'équipe.
Intégration et déploiement continu (15%)	Le repository du projet implémente un mécanisme CI via les Github Actions. Ce mécanisme a été utilisé tout au long du TP3. Les détails du pipeline décrits dans l'énoncé sont respectés.
Représentation graphique interactive de la serre (15%)	La quasi-totalité des concepts décrits dans l'énoncé du problème sont présents dans le code et celui-ci démontre une bonne compréhension du problème. Le code fonctionne et répond entièrement au récit utilisateur. Le code ne bogue pas. L'interface respecte les standards vus en classe.
Activation des systèmes climatiques (5%)	La quasi-totalité des concepts décrits dans l'énoncé du problème sont présents dans le code et celui-ci démontre une bonne compréhension du problème. Le code fonctionne et répond entièrement au récit utilisateur. Le code ne bogue pas. L'interface respecte les standards vus en classe.
Conseils d'ajustement climatiques (10%)	La quasi-totalité des concepts décrits dans l'énoncé du problème sont présents dans le code et celui-ci démontre une bonne compréhension du problème.

	Le code fonctionne et répond entièrement au récit utilisateur. Le code ne bogue pas. L'interface respecte les standards vus en classe.
lecture de données météo (10%)	La quasi-totalité des concepts décrits dans l'énoncé du problème sont présents dans le code et celui-ci démontre une bonne compréhension du problème. Le code fonctionne et répond entièrement au récit utilisateur. Le code ne bogue pas. L'interface respecte les standards vus en classe.
Qualité du code (25%)	Bonne implémentation de code respectant les principes du Clean code vus en classe. Les critères suivants sont respectés : - Nommage significatif et adéquat - Encapsulation des fonctions - Taille des fonctions - Taille des classes - Clarté du code - Efficacité du code - Qualité des tests - Alignement horizontal du code - Alignement vertical du code
Tests unitaires (15%)	Les tests couvrent l'ensemble des fonctionnalités programmées. Les tests valident un nombre suffisant de cas pour s'assurer du bon fonctionnement des fonctionnalités. Les tests sont codés de façon efficace et selon les principes du Clean code vus en classes.
mocking (5%)	Les tests utilisent un minimum de deux fois le principe de mocking pour simuler un accès à une base de données de test. Le mock est fonctionnel et bien configuré en utilisant le concept d'interfaces.
Respect des directives et de l'énoncé	Le travail remis respecte l'ensemble des points cités plus haut.