420-W1R-SW - Projet 2 - Grille de correction

Fonctionnalités (85 points)

Critère	Description	Pondération
Structure du Dashboard	Le Dashboard fonctionne comme prévu (superposition interactive sur une page d'accueil).	**12 pts**
Personnalisation des Modules	Chaque module peut être déplacé, redimensionné, masqué, affiché et sauvegarde ses paramètres.	**12 pts**
Implémentation de modules minimum Individuel : 7 modules (2x6+5x2) Équipe : 10 modules (2x3+8x2)	Au moins 10 modules fonctionnels et pertinents sont développés et fonctionnels (ex. météo, suivi des tâches, statistiques, calendrier, etc.).	**22 pts**
Sauvegarde Locale	Les préférences utilisateur sont bien sauvegardées via LocalStorage, SessionStorage ou IndexedDB.	**10 pts**
Système de Gamification	XP, niveaux et pénalités bien intégrés et fonctionnels.	**8 pts**
Gestion des interactions	Possibilité d'ajouter, supprimer, modifier et déplacer des modules de façon fluide.	**8 pts**
Expérience Utilisateur (Animations, Transitions)	Interface moderne, fluide et agréable à utiliser.	**8 pts**
Gestion des raccourcis clavier	Une combinaison de touches permet d'afficher/masquer l'overlay des modules, mais toutes les autres touches fonctionnent normalement pour que la navigation soit transparente.	**5 pts**
Bonus (**Max 110 / 100**) Les points peuvent servir aussi à compenser d'éventuelles erreurs.		
Bonus : Chiffrement des données	Stockage sécurisé des préférences utilisateur via cryptographie (ex. chiffrement des données sensibles en LocalStorage).	+5 pts
Bonus : Importation/Exportation des données	Permet à l'utilisateur de sauvegarder et restaurer ses préférences sur un autre appareil.	+5 pts
Bonus : Thèmes avancés	Création de thèmes templates personnalisés (ex. mode Star Wars, Pokémon, etc.).	+5 pts
Bonus : Gestion avancée des To-Do Lists	Génération automatique de tâches dans une nouvelle liste selon un modèle défini par l'utilisateur. (Ex : Daily, brosser les dents, faire le lit, ...)	+5 pts
Bonus : Implémentation de 2 modes de recherches configurable	Permettre de lancer la recherche dans un nouvel onglet OU dans le même.	+1 pts
Bonus : Interface multilingue	Possibilité de changer la langue du Dashboard.	**+5 pts**

Expérience Utilisateur et Design (15 points)

Critère	Description	Pondération
Interface intuitive et bien organisée	L’agencement est cohérent, clair et facile à utiliser.	**7 pts**
Personnalisation et ergonomie	Le Dashboard est agréable à utiliser avec une bonne gestion de la personnalisation (thèmes, affichage, couleurs, disposition).	**5 pts**
Transitions et animations fluides	Présence d’effets agréables sans surcharger l’interface.	**3 pts**

Pénalités

Barème de pénalité pour fautes de français (Max 10%) :

- **1 à 5 fautes** → -2 pts
- **6 à 10 fautes** → -5 pts
- **11 fautes et plus** → -10 pts

Il n'y a pas de limite de pénalités pour les erreurs de standards et bonnes pratiques! Faites attention !!! Liste de cas qui seront vérifiés :

Erreur	Description
Structuration du code	
Indentation incorrecte	Code mal formaté, niveaux d'indentation incohérents. Mauvaise utilisation de raccourcis (Manque {} sur les blocs d'une seule ligne)
Code spaghetti	Code désorganisé, difficile à lire et comprendre.
Mauvaise séparation HTML / CSS / JS / Classes / Interfaces / ...	JavaScript écrit directement dans le HTML au lieu d'un fichier séparé.
Code dupliqué	Répétition excessive de blocs de code au lieu de réutiliser des fonctions.
Trop de commentaires inutiles	Code surchargé de commentaires inutiles qui ne clarifient rien.
Absence de commentaires utiles	Manque d’explications sur des parties complexes ou algorithmiques.
Utilisation de variables globales inutiles	Mauvaise encapsulation, pollution de l’espace global.
Nommage des variables, fonctions et classes	
Nom non descriptif	Utilisation de noms comme `toto`, `var1`, `test`, `temp`.
Noms mélangés (français/anglais)	`compteur` et `scoreTotal` dans le même code.
Non-respect des conventions de casse	Mélange `camelCase`, `snake_case`, `PascalCase` sans cohérence.
Abréviations non standard	Utilisation de `btnQst` au lieu de `questionButton`.
Écriture du code JavaScript	
Utilisation excessive de `var` au lieu de `let` ou `const`	`var` est obsolète et entraîne des erreurs de portée.
Pas de validation des entrées utilisateur	Absence de `if` ou `try/catch` pour éviter les erreurs de saisie.

Erreur	Description
Utilisation de `innerHTML` sans précaution	Risque de faille XSS (exploitation de scripts).
Trop de code dans les actions comme `onclick` directement dans HTML	Ex: `<button onclick="ajouter()">Ajouter</button>`.
Pas d'écouteurs d'événements (`addEventListener`)	Mauvaise séparation des interactions et du HTML.
Utilisation excessive de boucles `for` au lieu de `map`, `filter`	Code plus verbeux et moins lisible.
Logique et structures de données	
Utilisation d'un tableau au lieu d'un objet	Quand un objet serait plus logique (ex: une question au lieu d'un simple array).
Utilisation d'un `if/else` imbriqué au lieu d'un `switch/case`	Code moins lisible et maintenable.
Boucle infinie ou mal optimisée	Exécute trop d'itérations inutiles.
Code inefficace	Plusieurs appels DOM inutiles, code trop complexe pour une tâche simple.
Gestion des erreurs et robustesse	
Absence de gestion des erreurs (`try/catch`)	Surtout si des erreurs peuvent interrompre l'exécution.
Pas de vérification avant accès DOM	`document.getElementById("inexistant")` sans test préalable.
Absence de contrôle des valeurs `undefined` ou `null`	Peut générer des erreurs inattendues.
Absence de `default` dans un `switch/case`	Oublier de gérer un cas imprévu.
Utilisation de Git	
Un seul commit pour tout le projet	Manque de progression et d'historique clair.
Commits avec messages vagues (`update`, `fix`...)	Peu clair pour un suivi du projet.
Fichiers inutiles dans le repo (ex: `.DS_Store`, fichiers temporaires)	Mauvaise gestion du dépôt.
Structure des fichiers chaotique	HTML, CSS et JS mélangés n'importe comment.