RNCP 5 - Règles d'évaluation technique des dossiers

Version 1.0

Les compétences de validation technique du titre sont un **minimum** à atteindre. Vous devez **faire mieux** que ce qui est noté ici et aller plus loin que ce que vous avez vu en cours.

Principe général de notation

- Compétences front requises :
    - mettre en place une interface utilisateur responsive et dynamique
    - produire du code propre et valide
    - utiliser les bonnes balises en fonction des besoins
    - privilégier le flex/grid
- Compétences back requises :
    - utilisations de structures de contrôles
    - mettre en place un backoffice complet et sécurisé
    - mettre en place un CRUD
    - respecter le MVC
    - utilisation de la POO

Validation des dossiers

- Validation minimum pour l'intégration (HTML et CSS) :
    - OK pour les deux
    - LÉGER & BIEN
- Validation minimum pour le développement (JS, PHP et SQL) :
    - OK pour les trois
    - un BIEN et deux LÉGER
- Une matière en inacceptable et le dossier est ajourné

Barèmes général

- **INACCEPTABLE** *(au moins un des points suivants)* :
    - failles de sécurité (xss, sql)
    - erreurs (500)
    - pas responsive
    - triche
    - exo vu en cours
    - mauvaise dénomination des fichiers
- **LÉGER** *(au moins un des points suivants)* :
    - minimum (flemme)
    - suspicion de copier coller intégral (exo vu en cours, trouvable sur le net)
    - mauvaise dénomination (des variables/urls)
- **OK** :
    - pas de maladresse on sent que l'élève maîtrise le langage.
- **BIEN** :
    - on respecte **toutes** les bonnes pratiques
    - commentaires / SEO / Optimisation / bon nommage / découpage des fichiers
- **TB**:
    - l'élève va plus loin que le cours

Notation du code HTML

- **INACCEPTABLE** *(au moins un des points suivants)* :
    - duplication d'id
    - css inline
    - balise style dans le html (en-tête ou corps)
    - uniquement bootstrap
    - js inline
    - charset invalide ou non pertinent
- **LÉGER** *(tous les points suivants)* :
    - bonne indentation
    - bonne structure
    - balises fermantes fermées
    - responsive
    - alt sur les <img>
    - bonne utilisation des balises sémantiques
        - pas d'imbrication incompréhensible (ex : article dans section dans article)
        - pas d'imbrication douteuse (article dans header)
    - accessibilité ok
    - types supportés dans les input
    - formulaire propre et utilisation de label
    - pas trop de
- **OK** *(tous les points suivants)* :
    - Utilisation de font-family (même prise sur Google Fonts)
    - commentaires
    - title sur les <a>
    - utilisation correcte des balises HTML5
    - aucun
- **BIEN** *(tous les points suivants)* :
    - "valide" W3C
    - code parfaitement propre
- **TB** *(deux des points suivants)* :
    - SEO-friendly
    - création SVG
    - backend admin étoffé

Notation du code CSS

- **INACCEPTABLE** *(au moins un des points suivants)* :
    - bootstrap/équivalent seul
    - erreurs classes ou sélecteurs
    - plusieurs feuilles css
- **LÉGER** *(tous les points suivants)* :
    - code personnel
    - responsive et media query
    - utilisation des id uniquement pour les ancres
    - base bootstrap/équivalent acceptée (mais non obligatoire)
- **OK** *(tous les points suivants)* :
    - commentaires (ex : /* HEADER */ pour les balises du header)
    - pas de !important (sauf exceptionnellement dans un framework)
    - code structuré
    - code ordonné par rapport au html (header en haut, footer en bas, etc…)
    - dénomination intelligente des classes
    - définition globale (ex : h1 toujours rouge)
        - *(et au moins un des points suivants)* :
            - animation
            - transform
            - gradient
- **BIEN** *(tous les points suivants)* :
    - pas de css externe (sauf font/normalize/reset/plugins jquery)
    - mobile first
    - rgba
        - *(et au moins un des points suivants)* :
            - sélecteurs avancés
            - pseudo classe
            - utilisation d'une nomenclature reconnue (ex : BEM, OOCSS, etc…)
        - *(et au moins un des points suivants)* :
            - flex
            - grid
- **TB** *(au moins un des points suivants)* :
    - responsive tailles intermédiaires
    - préprocesseur (sass/less/stylus)
    - utilisation de variables css
    - utilisation d'unité relative (ex : em, rem) plutôt qu'absolue (ex : px)

Notation du code JavaScript

- **INACCEPTABLE** *(au moins un des points suivants)* :
    - initiation de plugins uniquement
    - js inline
- **LÉGER** *(tous les points suivants)* :
    - peu de code personnel
    - au moins un écouteur d'événement (hors DOMContentLoaded)
    - fichier js séparé
- **OK** *(tous les points suivants)* :
    - JS bien découpé (dans plusieurs fichiers si besoin)
    - code commenté
    - respect des bonnes pratiques (sélecteurs DOM intelligents, let, const, performances friendly...)
    - ne pas bloquer de fonctionnalités si le JS est désactivé (ex : formulaire toujours utilisable)
- **BIEN** *(au moins deux des points suivants)* :
    - utilisation de ES6
    - AJAX
    - local storage
    - JSON
    - manipulation du DOM sans jquery
    - API HTML5
    - POO
- **TB** *(au moins un des points suivants)* :
    - fonctionnalités utiles (ex : formvalidator)
    - fonctionnalités originales (ex : animation poussée, mini jeu)
    - un gestionnaire de package genre NPM

Notation du code PHP

- **INACCEPTABLE** *(au moins un des points suivants)* :
    - failles
    - erreur 500
    - plus de 2 langages par page (ex: sql, php et html dans la même page)
- **LÉGER** *(tous les points suivants)* :
    - structures de contrôles (conditions et boucles)
    - utilisation de empty ou isset
    - utilisation de $_POST et $_GET
    - pas de traitement dans les vues
    - utilisation d'un framework avec peu de code personnel accepté (mais non obligatoire)
- **OK** *(au moins un des points suivants)* :
    - Templating (avec un moteur de templates ou en php natif, syntaxe alternative)
    - POO
- **BIEN** *(tous les points suivants)* :
    - architecture MVC parfaite
    - utilisation du typage (types nullables, return type)
    - type hinting (class, self, array, callable, bool, float, int, string)
    - une bonne gestion des erreurs
- **TB** *(au moins un des points suivants)* :
    - POO avancée (ex : héritage, namespace, classe abstraite, propriétés typées, etc)
    - création d'un micro-framework sur-mesure pour le projet
    - composer

Notation du code SQL

- **INACCEPTABLE** *(au moins un des points suivants)* :
    - injections sql
    - sql dans le html
    - dénomination des champs invalide (espaces, caractères spéciaux, etc…)
    - mot de passe en clair ou hash non sécurisé (md5, sha1)
    - interclassement invalide ou non pertinent
- **LÉGER** *(tous les points suivants)* :
    - Dump
    - Présence d'un CRUD complet fait à la main
        - SELECT (simple)
        - INSERT
        - UPDATE
        - DELETE
    - utilisation de PDO
    - requêtes préparées
- **OK** *(tous les points suivants)* :
    - stockage mot de passe (hash)
    - utilisation de jointure
    - diagramme UML
- **BIEN** *(tous les points suivants)* :
    - Quelque chose de plus que login / messagerie (au moins 4 tables)
        - *(et au moins un des points suivants)* :
            - gestion du SQL dans un fichier indépendant genre database.php
            - ORM + un CRUD à la main obligatoire
- **TB** *(au moins un des points suivants)* :
    - colonnes calculés/groupes
    - utilisation des vues
    - statistiques (fonctions d'agrégation)
    - pas d'utilisation du *
    - requêtes imbriqués