Examen FINAL — Symfony

Développement d’une application de gestion de budget

Instructions

- **Désactiver l’IA** sur vos outils de développement (Copilot, ChatGPT, assistants IDE, extensions IA, etc.).
- **Les ressources du cours ne sont pas autorisées** (supports, vidéos, PDF, dépôts internes, notes de cours).
- **La consultation de ressources sur Internet est interdite** (documentation, StackOverflow, articles, forums, etc.).

Objectif de l’examen

L’objectif est de **développer une mini-application Symfony** permettant à un utilisateur de **créer un compte, se connecter**, puis **gérer ses transactions de budget** (catégories + modes de paiement), en respectant les bonnes pratiques Symfony (routing, contrôleurs, repository, Doctrine, sécurité, validation, etc.).

Prérequis

- Symfony installé sur la machine
- Base de données opérationnelle (SQLite/MySQL/PostgreSQL)
- Compte GitHub (versionnage / rendu)

Déroulement

- **Durée : 1H30**
- **Fin :** dernier commit + push sur GitHub avant la fin du temps

Partie 1 — Questions à réponses ouvertes

Répondre de façon structurée sur un document de traitement de texte puis convertir en PDF avant le dépôt.

Question 1 — Architecture Symfony (mots-clés imposés)

Expliquer l'architecture d'une application Symfony en utilisant et reliant correctement les notions suivantes :
Request / Response, **verbes HTTP**, **codes HTTP**, **contrôleur frontal**, **routing / routes**, **controller**, **repository**, **Doctrine**, **vue (Twig) etc...**.

Attendu : décrire le cheminement complet d'une requête, depuis l'URL jusqu'au HTML retourné.

Question 2 — Cache HTTP Symfony

Expliquez **comment fonctionne le cache HTTP dans Symfony** et **ses avantages**.

Partie 2 — Projet : Application "Budget"

1) Authentification & comptes

- Mise en place d'un système **d'inscription** et **de connexion**
- Chaque utilisateur ne voit **que ses données**
- (Optionnel si tu veux) rôle **USER** (et éventuellement ADMIN si utile, mais pas obligatoire ici)

2) Modèle de données (minimum attendu)

Entités

- **User**
- **PaymentMethod** (ex : Carte, Espèces, Virement…)
- **Category** (ex : Alimentation, Transport…)
- **Transaction**

Règles de relation

- Une **Transaction** possède **1 mode de paiement** (ManyToOne vers PaymentMethod)
- Une **Transaction** possède **1 ou plusieurs catégories** (ManyToMany avec Category)
- Une **Transaction** appartient à **1 utilisateur** (ManyToOne vers User)
- Un **mode de paiement** et une **catégorie** appartiennent à **1 utilisateur** (recommandé pour cloisonner les données)

3) Fonctionnalités attendues (CRUD + pages)

Transactions

- Créer une transaction
- Lister ses transactions (tri par date décroissante)
- Voir le détail d'une transaction
- Modifier / supprimer une transaction

Catégories

- CRUD catégories (au minimum : créer + lister + supprimer)

Modes de paiement

- CRUD modes de paiement (au minimum : créer + lister + supprimer)

4) Contraintes & bonnes pratiques attendues

- Validation Symfony (NotBlank, Positive, etc.)
- Protection : impossible d'accéder/modifier les données d'un autre utilisateur (voter ou contrôle dans contrôleurs/repositories)
- Utilisation correcte :
    - **Routes** (REST simples : index/new/edit/delete/show)
    - **Repository + Doctrine**
    - **Twig** (ou JSON si API, mais à préciser dans le dépôt)
- Gestion des erreurs : pages/formulaires propres (messages flash appréciés)

Livrables attendus (rendu)

- **Code pushé sur GitHub**