Test technique – Aproma Conseils – JavaScript

Veillez à mettre vos réponses dans un dépôt public sur GitHub.

Format de nom de dépôt GitHub attendu : NOM_PRENOM_APROMA

NOM Prénom	UPRAJAY Dylan
Date d'examen (JJ,MM,AAAA)	16/05/2023
Lien du dépôt GitHub	https://github.com/kikerslay/-UPRAJAY_DYLAN_APROMA

Durée : 1h

Contexte :

Vous êtes un développeur dans l'entreprise Aproma Conseils, une société de conseil spécialisée dans le domaine du développement web. Vous travaillez sur un projet stratégique visant à développer une plateforme de traitement des transactions de matières premières à l'échelle mondiale. Cette plateforme permettra aux entreprises du secteur de gérer leurs opérations commerciales de manière efficace et transparente.

Votre rôle principal est de gérer au mieux la base de données du système ainsi que le code back-end. Vous travaillez en étroite collaboration avec une équipe de développeurs front-end, des analystes métier et des experts du domaine des matières premières.

En ce qui concerne le code back-end, vous devez développer les fonctionnalités clés du système, telles que la gestion des utilisateurs, l'authentification, l'autorisation, le traitement des transactions, la génération de rapports, etc. Vous devez garantir la fiabilité, la sécurité et l'évolutivité du code back-end, en utilisant les bonnes pratiques de développement.

Vous êtes également responsable de l'intégration avec des fournisseurs de données externes, tels que les marchés des matières premières et les organismes de réglementation, pour obtenir des informations en temps réel sur les prix, les réglementations et les tendances du marché.

Le projet exige des compétences solides en développement back-end et une expertise dans la gestion de bases de données. Vous travaillerez dans un environnement dynamique et collaboratif, où la communication et la résolution de problèmes seront essentielles pour atteindre les objectifs du projet.

Votre mission est donc de contribuer activement au succès de ce projet en fournissant des solutions techniques robustes et innovantes pour la gestion des transactions de matières premières à l'échelle mondiale.

Durée : 1h

1) Question sur la fonction aggregateByPeriod

Expliquez en détail ce que fait la fonction aggregateByPeriod. Quels sont les paramètres qu'elle prend en entrée et quel est le résultat renvoyé par la fonction ? Quelle est l'utilité des paramètres optionnels tradeType et flow ?

```
function aggregateByPeriod(transactions, valueKey, quantityKey, filters = {}
    const { tradeType, flow } = filters;
    const aggregatedData = {};

    transactions.forEach(transaction => {
        if (tradeType && transaction.TRADE_TYPE !== tradeType) return;
        if (flow && transaction.FLOW !== flow) return;

        const period = transaction.PERIOD;
        if (!aggregatedData[period]) {
            aggregatedData[period] = {
                period: period,
                [quantityKey]: 0,
                declarants: new Set()
            };
        }
        aggregatedData[period][quantityKey] += transaction[valueKey] || 0;
    });

    return Object.values(aggregatedData);
}
```

Les paramètres : transactions un tableau, valueKey une clé type entier, quantitykey une clé type entier, filters type object

elle retourne un un tableau contenant les valeurs de l'object aggregatedData

les paramètres optionnels tradeType et Flow permettent de filtrer les transactions

La fonction permet d'agréger en fonction d'une periode donnée , d'abord elle initialise deux constantes ; de type filters tradeType, flow qui seront utilisées pour filtrer les transactions dans la boucle forEach

Durée : 1h

un object vide aggregatedData qui sera rempli au fur et à mesure de la boucle forEach

Pour chaque élément du tableau transactions on vérifie si la valeur TRADE_TYPE est différente de tradeType alors on passe à l'itération suivante, on fait de même pour FLOW

ensuite on créer une valeur period qui prend en valeur transaction.PERIOD on fait une condition if si aggregatedData[period] n'existe pas alors on crée l'object pour period : period, quantitykey : 0 et declarants: un nouveau set d'élément

et on incrémente aggregatedData[period][quantityKey] par transaction[valeurKey] ou 0 s'il n'exsite pas

2) Question sur l'appel à la fonction aggregateByPeriod

Pourquoi ce code n'est-il pas optimisé ? Écrivez un code optimisé qui réalise les mêmes appels, mais de manière plus efficiente.

```
let chart_qty_i_i = aggregateByPeriod(arr, "QUANTITY_IN_KG", "QTY_IN_KG", { tradeType: "I", flow: "1" });
let chart_qty_i_e = aggregateByPeriod(arr, "QUANTITY_IN_KG", "QTY_IN_KG", { tradeType: "I", flow: "2" });
let chart_qty_e_i = aggregateByPeriod(arr, "QUANTITY_IN_KG", "QTY_IN_KG", { tradeType: "E", flow: "1" });
let chart_qty_e_e = aggregateByPeriod(arr, "QUANTITY_IN_KG", "QTY_IN_KG", { tradeType: "E", flow: "2" });

let chart_value_i_i = aggregateByPeriod(arr, "VALUE_IN_EUROS", "VALUE", { tradeType: "I", flow: "1" });
let chart_value_i_e = aggregateByPeriod(arr, "VALUE_IN_EUROS", "VALUE", { tradeType: "I", flow: "2" });
let chart_value_e_i = aggregateByPeriod(arr, "VALUE_IN_EUROS", "VALUE", { tradeType: "E", flow: "1" });
let chart_value_e_e = aggregateByPeriod(arr, "VALUE_IN_EUROS", "VALUE", { tradeType: "E", flow: "2" });
```

Le code n'est pas optimisé car fait les mêmes appels à la fonction aggregateByPeriod plusieurs fois avec les mêmes paramètres

.

3) Question générale :

Comment fonctionne le design pattern MVC avec Node.js ? Quel est l'intérêt ? Que signifie l'acronyme MVC (anglais autorisé) ?

Durée : 1h

Le design pattern Model View Controller en Node.js peut être utilisé par des Framework, notamment Express.js, l'utilisation de ce pattern permet d'avoir une meilleure gestion du code en séparant plusieurs éléments en trois catégorie : Modèle Vue Contrôleur

La partie Model représente les données utilisées comme une base de données.

La View est responsable de l'affichage

Le Controller fait le lien entre le Model et la View

L'acronyme MVC : Model View Controller

4) Question requête MongoDB :

Que fait ce code ? Y a-t-il des erreurs à corriger ou des optimisations à apporter ? Que retourne cette requête ? Justifiez votre réponse.

```
async function(productNCRegex, collectionName) {
    try {

        const coll = await client.db('Europa').collection(collectionName);
            const cursor = await coll.aggregate([{
                $match: {
                    PRODUCT_NC: {
                        $regex: productNCRegex
                    },
                }
            },
            {
                $group: {
                    _id: "$PRODUCT_NC",
                    totalValueInEuros: {
                        $sum: "$VALUE_IN_EUROS"
                    },
                    totalQuantityInKg: {
                        $sum: "$QUANTITY_IN_KG"
                    }
                }
            }
        ]);

        const products = await cursor.toArray();;
        return products;
    } catch (e) {
        console.error("Error:", e);
        return [];
    }
},
```

5) Question Architecture de base :

On considère une base de données avec plusieurs centaines de millions de documents. En quoi l'architecture de cette base de données est-elle adaptée et optimisée pour avoir des temps de requêtes faibles ? Pourquoi ne pas tout regrouper en une seule collection ?

Il est préférable de les séparer pour avoir une meilleure gestion de la base de donnée, si on veut une valeur spécifique la base de donnée sera pas obligé de parcourir toutes les valeurs.

Durée : 1h

6) Question Bonus :

Pourquoi ce code cause-t-il un problème de cybersécurité lors de l'enregistrement en base de données ?

La requête ne possède aucune condition de vérification (method, session, etc)

Durée : 1h

```
admin_societe_register: (req, res) => {
    var input_forme = req.body.input_forme
    var input_capital = req.body.input_capital
    var input_siret = req.body.input_siret
    var input_ceo_name = req.body.input_ceo_name
    var input_ceo_surname = req.body.input_ceo_surname
    var input_contact_email = req.body.input_contact_email

    var input_subscription = req.body.input_subscription
    var input_users_allowed = req.body.input_users_allowed
    var input_iban = req.body.input_iban
    var input_bic = req.body.input_bic
    var input_private_note = req.body.input_private_note
    var input_company_name = req.body.input_company_name


    const company = {
        forme: input_forme,
        capital: input_capital,
        SIRET: input_siret,
        CEO_name: input_ceo_name,
        CEO_surname: input_ceo_surname,
        contact_email: input_contact_email,
        state: 'active',
        subscription: input_subscription,
        users_allowed: input_users_allowed,
        IBAN: input_iban,
        BIC: input_bic,
        note: input_private_note,
        name: input_company_name
    }
    const database = client.db("auth_domap");
    const collectionInDB = database.collection("company");

    const result = collectionInDB.insertOne(company);

    res.redirect('/admin-societe-register')
```

Annexe :

Exemple de données dans MongoDB :

Durée : 1h

```
DECLARANT: 1
DECLARANT_ISO: "FR"
PARTNER: "3"
PARTNER_ISO: "NL"
TRADE_TYPE: "I"
PRODUCT_NC: "01012100"
PRODUCT_SITC: "00150"
PRODUCT_CPA2002: "0122"
PRODUCT_CPA2008: "0143"
PRODUCT_CPA2_1: "0143"
PRODUCT_BEC: "111"
PRODUCT_BEC5: "712210"
PRODUCT_SECTION: "01"
FLOW: "1"
STAT_REGIME: "1"
SUPP_UNIT: "A"
PERIOD: "202001"
VALUE_IN_EUROS: 41548
QUANTITY_IN_KG: 1463
SUP_QUANTITY: 3
VALUE_IN_EUROS_ON_QUANTITY_IN_KG: 28.39917976760082
```


Jeux de données 2 :

Durée : 1h

```
DECLARANT: 1
DECLARANT_ISO: "FR"
PARTNER: "3"
PARTNER_ISO: "NL"
TRADE_TYPE: "I"
PRODUCT_NC: "01012100"
PRODUCT_SITC: "00150"
PRODUCT_CPA2002: "0122"
PRODUCT_CPA2008: "0143"
PRODUCT_CPA2_1: "0143"
PRODUCT_BEC: "111"
PRODUCT_BEC5: "712210"
PRODUCT_SECTION: "01"
FLOW: "2"
STAT_REGIME: "1"
SUPP_UNIT: "A"
PERIOD: "202001"
VALUE_IN_EUROS: 84399
QUANTITY_IN_KG: 500
SUP_QUANTITY: 1
VALUE_IN_EUROS_ON_QUANTITY_IN_KG: 168.798
```


Durée : 1h