Khalil ELBOKAILY

Examen pour : Candidature au poste de développeur full stack chez Ellipse.

PARTIE THÉORIQUE:

1-Une interface de programmation d'application (API pour application programmation interface) correspond à un ensemble de règles grâce auxquelles un logiciel transmet des données à un autre logiciel.

Exemple d'utilisation d'une API dans un projet :

Dans le cas d'Ellipse Bikes, un exemple d'utilisation d'une API dans un projet pourrait être l'intégration d'une API météo dans leur application de cyclisme. Cette API météo pourrait fournir des mises à jour météorologiques en temps réel aux cyclistes, leur permettant de planifier leurs trajets de manière plus efficace et de rester informés des conditions météorologiques changeantes.

En intégrant l'API météo dans leur application, Ellipse Bikes pourrait améliorer l'expérience utilisateur et fournir des informations précieuses aux cyclistes, améliorant ainsi la sécurité et la convivialité. Il s'agit là d'un exemple de la manière dont les API peuvent être utilisées pour améliorer les fonctionnalités des applications et des services dans divers secteurs, y compris celui du cyclisme.

2-Un webhook est une fonction de rappel basée sur le protocole HTTP. Il permet à deux interfaces de programmation d'application (API) d'établir une communication légère et orientée événements.

Le fonctionnement d'un webhook est le suivant :

- Configuration : Tout d'abord, l'application qui reçoit les données (le serveur webhook) doit être configurée pour recevoir des requêtes HTTP POST à une URL spécifique, généralement fournie par le fournisseur de webhook.
- Déclenchement de l'événement : L'application qui envoie les données (l'émetteur de webhook) doit être configurée pour déclencher un webhook lorsqu'un événement spécifique se produit. Cela peut être n'importe quel événement pertinent pour le scénario d'utilisation, tel qu'une nouvelle commande passée, une inscription utilisateur, ou tout autre événement défini par l'application.
- Envoi des données : Lorsque l'événement se produit, l'application émettrice de webhook envoie automatiquement une requête HTTP POST au serveur webhook, en incluant les données pertinentes associées à l'événement.
- Réception des données : Le serveur webhook reçoit la requête POST et traite les données qu'il contient. Il peut effectuer des actions spécifiques en fonction des données reçues,

telles que la mise à jour d'une base de données, l'envoi de notifications aux utilisateurs, ou toute autre action définie par l'application.

Un exemple d'utilisation courante des webhooks est dans les applications de commerce électronique. Lorsqu'un client passe une nouvelle commande, l'application de commerce électronique peut déclencher un webhook pour envoyer les détails de la commande à un système de gestion des commandes ou à un service de notification par SMS. Cela permet au système de gestion des commandes de traiter automatiquement la commande et d'informer le client de son statut en temps réel, sans nécessiter d'intervention manuelle de la part des commerçants.

3-Les bases de données relationnelles et non relationnelles se distinguent principalement par leur modèle de données et leur structure. Les bases de données relationnelles utilisent un modèle tabulaire avec des tables, des lignes et des colonnes, ce qui impose une structure rigide avec un schéma préétabli. En revanche, les bases de données non relationnelles adoptent des modèles de données plus flexibles, tels que les documents, les colonnes, les paires clé-valeur ou les graphes, ce qui permet de stocker des données dans des formats variés sans imposer de schéma fixe. Cette différence fondamentale influence la manière dont les données sont organisées, stockées et manipulées, ainsi que leur évolutivité et leur adaptabilité aux différents types d'applications.

4-Oui, j'ai déjà utilisé Python dans le cadre de mon stage où j'ai travaillé avec Django, un framework de développement web en Python. J'ai développé des applications web en utilisant Django pour la création de backends robustes et sécurisés. En dehors de mon expérience en entreprise, j'ai également suivi des cours de Python où j'ai acquis des connaissances sur la syntaxe du langage, les structures de données, la manipulation des fichiers, la programmation orientée objet, ainsi que sur l'utilisation de bibliothèques populaires telles que NumPy, Pandas et Matplotlib pour l'analyse de données et la visualisation. Python m'a également permis de travailler sur des projets personnels, ce qui m'a permis de renforcer mes compétences dans ce langage polyvalent et puissant.

5-Non, je n'ai pas encore utilisé Wordpress ou Odoo, mais je suis intéressé à étudier ces plateformes pour développer mes compétences. Je comprends que Wordpress est un système de gestion de contenu (CMS) largement utilisé pour la création de sites web, tandis qu'Odoo est un système de gestion d'entreprise (ERP) open source qui offre une gamme complète d'applications pour divers processus commerciaux.

J'ai l'intention d'étudier la course automobile pour développer mes compétences et comprendre comment ces outils pourraient être utilisés dans le domaine. Si je décide d'utiliser Wordpress ou Odoo à l'avenir, je prévois de m'entraîner et de me familiariser avec leurs fonctionnalités avant de les mettre en pratique dans des projets réels. Je crois que cela me permettra de devenir plus compétent et efficace dans le domaine de la gestion de contenu web ou de la gestion d'entreprise.

PARTIE PRATIQUE:

1-

```python
def get_bike_data():
    # Clé API JCDecaux
    api_key = "e0a1bf2c844edb9084efc764c089dd748676cc14"
    url = f"https://api.jcdecaux.com/vls/v3/stations?apiKey={api_key}"

    response = requests.get(url)

    if response.status_code == 200:
        bike_data = response.json()
        return bike_data
    else:
        print("Erreur lors de la récupération des données:", response.status_code)
        return None

def analyze_bike_data(bike_data):
    if bike_data:
        total_bikes = len(bike_data)
        mechanical_bikes = sum(1 for bike in bike_data if bike['type'] == 'mechanical')
        electric_bikes = sum(1 for bike in bike_data if bike['type'] == 'electric')
        mechanical_percentage = (mechanical_bikes / total_bikes) * 100
        electric_percentage = (electric_bikes / total_bikes) * 100

        print("Statistiques sur les vélos :")
        print("Nombre total de vélos :", total_bikes)
        print("Nombre de vélos mécaniques :", mechanical_bikes)
        print("Nombre de vélos électriques :", electric_bikes)
        print("Pourcentage de vélos mécaniques :", mechanical_percentage)
        print("Pourcentage de vélos électriques :", electric_percentage)
    else:
        print("Aucune donnée à analyser.")

bike_data = get_bike_data()
analyze_bike_data(bike_data)
```

- La fonction **get_bike_data** envoie une requête GET à l'URL de l'API pour récupérer les données sur les vélos.
- La fonction **analyze_bike_data** analyse les données récupérées et calcule des statistiques telles que le nombre total de vélos, le nombre de vélos mécaniques et électriques, ainsi que les pourcentages correspondants.
- Enfin, les données sont récupérées à partir de l'API et analysées à l'aide de ces fonctions, et les résultats sont affichés dans la console.

2-

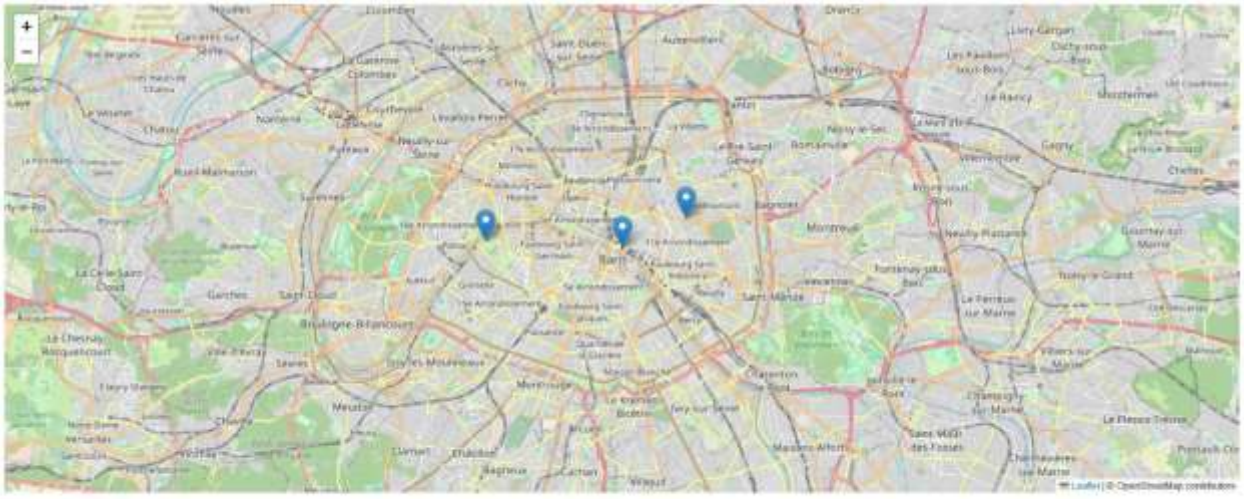
```
<html lang="en">
<head>
    <style>
    </style>
</head>
<body>
<div id="map"></div>
<script>
    // Créer une carte Leaflet centrée sur une position spécifique
    var map = L.map('map').setView([48.8566, 2.3522], 12);

    // Ajouter une couche de tuiles OpenStreetMap à la carte
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; OpenStreetMap contributors'
    }).addTo(map);

    // Données de station (exemple)
    var stations = [
        {name: 'Station 1', location: [48.8566, 2.3522], description: 'Description de la station 1'},
        {name: 'Station 2', location: [48.8584, 2.2945], description: 'Description de la station 2'},
        {name: 'Station 3', location: [48.8647, 2.3789], description: 'Description de la station 3'}
    ];

    // Parcourir les données de la station et ajouter des marqueurs à la carte
    stations.forEach(function (station) {
        L.marker(station.location).addTo(map)
            .bindPopup('<b>' + station.name + '</b><br/>' + station.description);
    });
</script>
</body>
</html>
```

Voici les résultats



Pour afficher les informations sur différentes stations sur une carte géographique, vous pouvez utiliser une bibliothèque de cartographie comme **Leaflet**. Commencez par rassembler les données des stations, telles que les noms et les emplacements géographiques. Ensuite, intégrez une carte dans votre page **HTML en utilisant Leaflet**, en initialisant une instance de carte et en définissant les paramètres de vue initiale. Parcourez ensuite les données des stations et ajoutez des marqueurs à la carte pour chaque station, personnalisant les marqueurs pour afficher les informations pertinentes lorsqu'ils sont cliqués. Personnalisez également le style de la carte et des marqueurs pour correspondre à vos besoins esthétiques. Enfin, testez votre application pour vous assurer que la carte fonctionne correctement et ajustez si nécessaire pour une meilleure expérience utilisateur.

3-

```
<script>
    // Fonction pour mettre à jour les informations sur la carte
    function updateMap() {
    const apiKey = "e0a1bf2c844edb9084efc764c089dd748676cc14";

    fetch(`https://api.jcdecaux.com/vls/v3/stations?apiKey=${apiKey}`)
        .then(response => response.json())
        .then(data => {
            data.forEach(station => {
                let existingMarker = markers.find(marker => marker.options.title === station.name);
                if (existingMarker) {
                    existingMarker.setPopupContent(`<b>${station.name}</b><br>Nombre de vélos disponibles : ${station.mainStands.availabilities.bikes}`);
                } else {
                    L.marker([station.position.latitude, station.position.longitude])
                        .addTo(map)
                        .bindPopup(`<b>${station.name}</b><br>Nombre de vélos disponibles : ${station.mainStands.availabilities.bikes}`);
                }
            (method) Promise<void>.catch<void>(onrejected?: ((reason: any) => void | PromiseLike<void>) | null | undefined): Promise<void>
        .catch(error => {
            console.error('Erreur lors de la récupération des données :', error);
        });
}
setInterval(updateMap, 60000);
</script>
```

En résumé, ce code permet de créer une carte interactive qui affiche en temps réel les informations sur les stations de vélos disponibles, en interrogeant périodiquement l'API JCDecaux et en mettant à jour les marqueurs sur la carte en conséquence.