Documentation de préparation d'entretien pour stage - Java, Spring Boot et Angular

Table des matières

Préparation personnelle

Introduction

Préparez une présentation d'environ 2-3 minutes qui couvre:

- Votre nom et votre formation
- Votre parcours et ce qui vous a attiré vers le développement
- Vos expériences (projets académiques, projets personnels, autres stages)
- Vos compétences techniques (même si débutantes)
- Votre motivation pour ce stage spécifique

Exemple d'introduction

```
Je m'appelle [Votre Nom], je suis actuellement étudiant(e) en [formation] à
[école/université].

Depuis toujours, j'ai été passionné(e) par l'informatique et la création de
solutions. C'est en découvrant le développement web que j'ai réalisé que je voulais
en faire mon métier, car j'apprécie particulièrement la combinaison de logique et de
créativité que cela demande.

Durant ma formation, j'ai travaillé sur plusieurs projets qui m'ont permis de me
familiariser avec Java et les bases de données. J'ai notamment développé [mentionnez
un projet simple] qui m'a permis d'appliquer mes connaissances théoriques.

Je maîtrise les bases de Java et HTML/CSS, et je suis en cours d'apprentissage de
Spring Boot et Angular. Je suis particulièrement à l'aise avec [mentionnez une
compétence où vous êtes plus confiant].

Je suis très motivé(e) par ce stage chez [nom de l'entreprise] car il représente une
opportunité d'apprendre dans un environnement professionnel et de contribuer à des
projets concrets tout en développant mes compétences techniques.
```

Pour vos expériences

Même en tant que débutant, vous pouvez mentionner:

- Projets académiques
- Travaux pratiques réalisés en cours
- Projets personnels, même simples
- Expériences de groupe/projets d'équipe
- Participation à des hackathons ou challenges de code

Questions et réponses sur Java

Q1: Qu'est-ce que Java et quelles sont ses principales caractéristiques?

Réponse:
Java est un langage de programmation orienté objet créé par Sun Microsystems (maintenant Oracle). Ses principales caractéristiques sont:

- **Orienté objet**: Tout en Java est un objet, ce qui permet une meilleure organisation du code
- **Indépendant de la plateforme**: Les programmes Java peuvent fonctionner sur n'importe quel appareil grâce à la JVM (Java Virtual Machine)
- **Simple à apprendre**: Syntaxe proche du C/C++ mais avec moins de complexités
- **Sécurisé**: Pas d'accès direct à la mémoire et vérification du bytecode
- **Robuste**: Gestion des erreurs et vérification forte des types
- **Multithreading**: Possibilité d'exécuter plusieurs tâches simultanément

Q2: Expliquez la différence entre JDK, JRE et JVM?

Réponse:

- **JDK (Java Development Kit)**: Ensemble d'outils pour développer des applications Java (compilateur, débogueur, etc.) + JRE
- **JRE (Java Runtime Environment)**: Environnement pour exécuter des applications Java (contient la JVM et des bibliothèques standard)
- **JVM (Java Virtual Machine)**: Machine virtuelle qui exécute le bytecode Java, rendant Java indépendant de la plateforme

Q3: Qu'est-ce qu'une classe et un objet en Java?

Réponse:

- **Classe**: C'est un modèle ou un plan qui définit les attributs et les comportements communs à un ensemble d'objets. Exemple:

```java
public class Voiture {
    // Attributs
    String marque;
    String couleur;

    // Méthodes
    public void demarrer() {
        System.out.println("La voiture démarre");
    }

    public void arreter() {
        System.out.println("La voiture s'arrête");
    }
}
```

- **Objet**: C'est une instance d'une classe. Exemple:

```java
Voiture maVoiture = new Voiture();
maVoiture.marque = "Toyota";
maVoiture.couleur = "Rouge";
maVoiture.demarrer();
```

Q4: Qu'est-ce que l'encapsulation en Java?

Réponse:
L'encapsulation est le principe qui consiste à regrouper les données (attributs) et les méthodes qui les

manipulent dans une classe, tout en restreignant l'accès direct aux attributs. On utilise des modificateurs d'accès comme `private`, `public` et `protected`.

Exemple simple:

```java
public class Personne {
    // Attributs privés
    private String nom;
    private int age;

    // Accesseurs (getters)
    public String getNom() {
        return nom;
    }

    public int getAge() {
        return age;
    }

    // Mutateurs (setters)
    public void setNom(String nom) {
        this.nom = nom;
    }

    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
    }
}
```

Q5: Qu'est-ce que l'héritage en Java?

Réponse:
L'héritage est un mécanisme qui permet à une classe d'hériter des propriétés et des comportements d'une autre classe. La classe qui hérite est appelée "sous-classe" ou "classe fille", et la classe dont elle hérite est appelée "super-classe" ou "classe mère".

Exemple simple:

java

```java
// Classe mère
public class Animal {
    void manger() {
        System.out.println("L'animal mange");
    }
}

// Classe fille qui hérite d'Animal
public class Chien extends Animal {
    void aboyer() {
        System.out.println("Le chien aboie");
    }
}

// Utilisation
Chien monChien = new Chien();
monChien.manger(); // Méthode héritée
monChien.aboyer(); // Méthode propre à Chien
```

Q6: Quelle est la différence entre `ArrayList` et `LinkedList` en Java?

Réponse:

- **ArrayList**: Implémente une liste basée sur un tableau dynamique. Accès rapide aux éléments par index, mais plus lent pour les insertions/suppressions au milieu de la liste.
- **LinkedList**: Implémente une liste doublement chaînée. Insertions/suppressions rapides, mais accès plus lent aux éléments par index.

Exemple simple:

java

```java
// ArrayList - bon pour l'accès aléatoire
ArrayList<String> listeArray = new ArrayList<>();
listeArray.add("Pomme");
listeArray.add("Banane");
String fruit = listeArray.get(1); // Accès rapide

// LinkedList - bon pour les insertions/suppressions
LinkedList<String> listeChainee = new LinkedList<>();
listeChainee.add("Pomme");
listeChainee.addFirst("Banane"); // Ajout facile au début
listeChainee.remove("Pomme");    // Suppression facile
```

Q7: Comment gérer les exceptions en Java?

Réponse:
En Java, on gère les exceptions avec les blocs `try`, `catch`, `finally`:

```java
try {
    // Code susceptible de générer une exception
    int resultat = 10 / 0; // Division par zéro
} catch (ArithmeticException e) {
    // Gestion de l'exception spécifique
    System.out.println("Erreur de calcul: " + e.getMessage());
} catch (Exception e) {
    // Gestion des autres exceptions
    System.out.println("Une erreur est survenue: " + e.getMessage());
} finally {
    // Code exécuté qu'il y ait une exception ou non
    System.out.println("Cette partie s'exécute toujours");
}
```

Questions et réponses sur Spring Boot

Q1: Qu'est-ce que Spring Boot et quels sont ses avantages?

Réponse:
Spring Boot est un framework basé sur Spring qui simplifie le développement d'applications Java. Ses principaux avantages sont:

- **Configuration automatique**: Spring Boot configure automatiquement l'application selon les dépendances présentes
- **Serveur embarqué**: Pas besoin d'installer un serveur externe, il est intégré (comme Tomcat)
- **Démarrage rapide**: Permet de créer des applications rapidement sans configuration complexe
- **Applications autonomes**: Création d'applications Java qui peuvent être exécutées directement (JAR)
- **Écosystème riche**: Intégration facile avec d'autres technologies (bases de données, sécurité, etc.)

Q2: Qu'est-ce que l'Inversion de Contrôle (IoC) et l'Injection de Dépendances dans Spring?

Réponse:

- **Inversion de Contrôle (IoC)**: Principe où le contrôle de la création et de la gestion des objets est transféré du programme au conteneur Spring
- **Injection de Dépendances**: Mécanisme où le conteneur Spring "injecte" automatiquement les dépendances requises par un objet

Exemple simple:

java

```java
// Sans Spring
class ServiceUtilisateur {
    private RepositoryUtilisateur repository = new RepositoryUtilisateurImpl();
    // ...
}

// Avec Spring (injection de dépendances)
@Service
class ServiceUtilisateur {
    private final RepositoryUtilisateur repository;

    // Injection via constructeur
    @Autowired
    public ServiceUtilisateur(RepositoryUtilisateur repository) {
        this.repository = repository;
    }
    // ...
}
```

Q3: Expliquez les annotations Spring Boot les plus courantes

Réponse:

- **@SpringBootApplication**: Combine @Configuration, @EnableAutoConfiguration et @ComponentScan
- **@Controller/@RestController**: Indique qu'une classe est un contrôleur web
- **@Service**: Marque une classe comme service de la couche métier
- **@Repository**: Indique une classe qui interagit avec la base de données
- **@Autowired**: Injecte une dépendance
- **@RequestMapping**: Mappe les requêtes HTTP à des méthodes de contrôleur

Exemple simple d'un contrôleur REST:

java

```java
@RestController
public class HelloController {

    @GetMapping("/hello")
    public String hello() {
        return "Bonjour, monde!";
    }

    @GetMapping("/hello/{nom}")
    public String helloPersonnalise(@PathVariable String nom) {
        return "Bonjour, " + nom + "!";
    }
}
```

Q4: Comment Spring Boot gère-t-il les données?

Réponse:
Spring Boot propose Spring Data JPA qui simplifie l'accès aux données. Vous définissez une interface qui étend JpaRepository, et Spring crée automatiquement l'implémentation.

Exemple simple:

java

```java
// Entité
@Entity
public class Utilisateur {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nom;
    private String email;

    // Getters et setters
}

// Repository
public interface UtilisateurRepository extends JpaRepository<Utilisateur, Long> {
    // Spring génère automatiquement les méthodes CRUD

    // Méthodes personnalisées
    List<Utilisateur> findByNom(String nom);
    Utilisateur findByEmail(String email);
}
```

Questions et réponses sur Angular

Q1: Qu'est-ce qu'Angular et quels sont ses avantages?

Réponse:
Angular est un framework front-end développé par Google pour créer des applications web dynamiques. Ses avantages:

- **Structure organisée**: Architecture MVC (Modèle-Vue-Contrôleur)
- **Components**: Division de l'interface en blocs réutilisables
- **TypeScript**: Typage statique qui améliore la qualité du code
- **Tests faciles**: Architecture conçue pour la testabilité
- **Réutilisabilité**: Composants et services réutilisables
- **Outils**: Angular CLI pour faciliter le développement
- **Communauté**: Large communauté et documentation

Q2: Qu'est-ce qu'un composant dans Angular?

Réponse:
Un composant est un bloc de construction fondamental d'une application Angular. Il encapsule la logique, la vue (HTML) et le style (CSS) d'une partie de l'interface utilisateur.

Exemple simple:

typescript

```typescript
// compteur.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-compteur',
  template: `
    <div>
      <h2>Compteur: {{ compte }}</h2>
      <button (click)="incrementer()">+</button>
      <button (click)="decrementer()">-</button>
    </div>
  `,
  styles: ['h2 { color: blue; }']
})
export class CompteurComponent {
  compte = 0;

  incrementer() {
    this.compte++;
  }

  decrementer() {
    this.compte--;
  }
}
```

Q3: Expliquez le data binding dans Angular

Réponse:
Le data binding dans Angular permet la communication entre le code TypeScript et le template HTML:

- **Interpolation** (`{{ }}`): Affiche une valeur du composant dans le template

  html

  ```html
  <h1>Bonjour {{ nom }}</h1>
  ```

- **Property binding** (`[ ]`): Lie une propriété d'un élément HTML à une valeur du composant

  html

  ```html
  <img [src]="imageUrl">
  ```

- **Event binding** (`( )`): Réagit aux événements utilisateur

  html

  ```html
  <button (click)="onSave()">Enregistrer</button>
  ```

- **Two-way binding** (`[( )]`): Combine property et event binding

```html
<input [(ngModel)]="nom">
```

Q4: Qu'est-ce qu'un service dans Angular?

Réponse:
Un service est une classe qui fournit des fonctionnalités spécifiques aux composants. Les services sont utilisés pour partager de la logique, des données ou des fonctionnalités entre différents composants.

Exemple simple:

```typescript
// utilisateur.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class UtilisateurService {
  private apiUrl = 'https://api.exemple.com/utilisateurs';

  constructor(private http: HttpClient) { }

  getUtilisateurs() {
    return this.http.get(this.apiUrl);
  }

  getUtilisateur(id: number) {
    return this.http.get(`${this.apiUrl}/${id}`);
  }
}
```

Q5: Comment Angular gère-t-il le routage?

Réponse:
Angular Router permet de naviguer entre différentes vues sans recharger la page entière:

```typescript
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { AccueilComponent } from './accueil/accueil.component';
import { ProfilComponent } from './profil/profil.component';

const routes: Routes = [
  { path: '', component: AccueilComponent },
  { path: 'profil/:id', component: ProfilComponent },
  { path: '**', redirectTo: '' } // Redirection pour les routes inconnues
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Et dans le template HTML:

```html
<!-- Navigation -->
<nav>
  <a routerLink="/">Accueil</a>
  <a routerLink="/profil/123">Profil</a>
</nav>

<!-- Où afficher les composants routés -->
<router-outlet></router-outlet>
```

Questions sur les bases de données

Q1: Quelle est la différence entre les bases de données SQL et NoSQL?

Réponse:

- **Bases de données SQL** (MySQL, PostgreSQL):
  - Structure fixe (schéma prédéfini)
  - Relations entre tables
  - Langage SQL standardisé
  - Bon pour les données structurées et les relations complexes
- **Bases de données NoSQL** (MongoDB, Redis):
  - Schéma flexible
  - Stockage de données sous forme de documents, paires clé-valeur, etc.

- Bonne scalabilité horizontale
- Bon pour les données non structurées ou semi-structurées

Q2: Qu'est-ce qu'une clé primaire et une clé étrangère?

Réponse:

- **Clé primaire**: Identifiant unique pour chaque enregistrement d'une table
- **Clé étrangère**: Champ qui fait référence à la clé primaire d'une autre table, créant ainsi une relation entre les tables

Exemple simple:

```sql
-- Table avec clé primaire
CREATE TABLE Utilisateurs (
    id INT PRIMARY KEY,
    nom VARCHAR(100),
    email VARCHAR(100)
);

-- Table avec clé étrangère
CREATE TABLE Commandes (
    id INT PRIMARY KEY,
    montant DECIMAL(10,2),
    utilisateur_id INT,
    FOREIGN KEY (utilisateur_id) REFERENCES Utilisateurs(id)
);
```

Q3: Qu'est-ce qu'une requête JOIN en SQL?

Réponse:
Une requête JOIN permet de combiner des lignes de deux ou plusieurs tables en fonction d'une condition de jointure:

```sql
-- Exemple de requête JOIN
SELECT Utilisateurs.nom, Commandes.montant
FROM Utilisateurs
INNER JOIN Commandes ON Utilisateurs.id = Commandes.utilisateur_id;
```

Types de JOIN:

- **INNER JOIN**: Retourne les lignes qui ont des correspondances dans les deux tables
- **LEFT JOIN**: Retourne toutes les lignes de la table de gauche et les correspondances de la table de droite

- **RIGHT JOIN**: Retourne toutes les lignes de la table de droite et les correspondances de la table de gauche
- **FULL JOIN**: Retourne toutes les lignes quand il y a une correspondance dans l'une des tables

Questions sur les méthodologies de développement

Q1: Qu'est-ce que la méthodologie Agile?

Réponse:
La méthodologie Agile est une approche de développement logiciel qui favorise:

- La collaboration entre équipes auto-organisées
- La livraison fréquente de logiciels fonctionnels
- L'adaptation aux changements plutôt que de suivre un plan rigide
- La communication directe plutôt que la documentation excessive

Q2: Qu'est-ce que Scrum?

Réponse:
Scrum est un cadre de travail Agile avec:

- Des rôles spécifiques (Product Owner, Scrum Master, équipe de développement)
- Des événements réguliers (Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective)
- Des cycles de travail appelés "Sprints" (généralement 2-4 semaines)
- Des artefacts comme le Product Backlog (liste des fonctionnalités à développer)

Q3: Qu'est-ce que Git et pourquoi est-il important?

Réponse:
Git est un système de contrôle de version qui permet:

- De suivre les modifications du code source
- De travailler simultanément sur le même projet
- De revenir à des versions antérieures si nécessaire
- De créer des branches pour développer des fonctionnalités parallèlement

Commandes Git de base:

bash

```bash
git init          # Initialiser un dépôt
git add .         # Ajouter des fichiers à l'index
git commit -m "Message"  # Valider les modifications
git push          # Envoyer les modifications au dépôt distant
git pull          # Récupérer les modifications du dépôt distant
git branch        # Afficher les branches
git checkout -b nom-branche  # Créer et basculer sur une nouvelle branche
```

Compétences non techniques

Q1: Comment gérez-vous votre temps et vos priorités?

Réponse:
"J'utilise des listes de tâches et des outils de planification pour organiser mon travail. Je priorise les tâches en fonction de leur urgence et de leur importance. Pour mes projets académiques, j'établis un planning avec des délais intermédiaires pour éviter le stress de dernière minute."

Q2: Comment réagissez-vous face à un problème technique difficile?

Réponse:
"Je commence par analyser le problème pour bien le comprendre. Je le décompose en parties plus petites et plus faciles à gérer. Je recherche des solutions en consultant la documentation, des forums comme Stack Overflow, ou en demandant de l'aide à des personnes plus expérimentées. Je n'hésite pas à demander des conseils quand je suis bloqué, tout en montrant que j'ai d'abord essayé de résoudre le problème par moi-même."

Q3: Comment travaillez-vous en équipe?

Réponse:
"J'apprécie le travail en équipe car il permet de partager des connaissances et des perspectives différentes. Je communique régulièrement avec les membres de l'équipe, j'écoute leurs idées et je partage les miennes de manière constructive. Je respecte les délais fixés par l'équipe et je n'hésite pas à demander de l'aide ou à en offrir quand c'est nécessaire."

Q4: Comment vous tenez-vous informé des nouvelles technologies?

Réponse:
"Je suis des blogs techniques, des chaînes YouTube spécialisées et des newsletters. Je participe à des webinaires et je fais des tutoriels en ligne pour apprendre de nouvelles technologies. J'ai également rejoint des communautés de développeurs sur Discord/Slack où je peux échanger avec d'autres passionnés."

Glossaire des termes techniques

- **API (Application Programming Interface)**: Ensemble de règles permettant à des applications de communiquer entre elles

- **Framework**: Ensemble d'outils et de bibliothèques facilitant le développement
- **JVM (Java Virtual Machine)**: Machine virtuelle exécutant le bytecode Java
- **ORM (Object-Relational Mapping)**: Technique convertissant des données entre systèmes de types incompatibles
- **REST (Representational State Transfer)**: Style d'architecture pour les systèmes web
- **IDE (Integrated Development Environment)**: Environnement de développement intégré (comme Eclipse, IntelliJ)
- **MVC (Model-View-Controller)**: Modèle architectural divisant l'application en trois composants logiques
- **DevOps**: Ensemble de pratiques combinant développement logiciel et opérations informatiques
- **CI/CD (Continuous Integration/Continuous Deployment)**: Automatisation de l'intégration et du déploiement
- **Refactoring**: Restructuration du code existant sans changer son comportement externe