Tout d'abord, je tiens à vous remercier pour votre temps et pour cette opportunité que vous m’offrez.

Je me présente, je m'appelle Oussama Zerrad, je suis diplômé de l'Ecole Nationale des Sciences Appliquées de Kénitra avec une spécialisation en ingénierie logicielle.

Au cours de ma formation, j'ai développé de solides compétences en développement full stack, aussi bien en backend qu'en frontend.

J'ai effectué mon stage de fin d'études à la DSI de l'ONCF, où j'ai rejoint une équipe dédiée au conception et développement d'une plateforme visant à intégrer différents systèmes d'information au sein d'un système ERP. Mon rôle initial consistait à développer le backend avec Java Spring Boot et à rédiger la documentation des différentes API. À mi-parcours, j'ai également pris en charge le développement du frontend avec Angular. Cette expérience m'a permis d'acquérir de nombreuses compétences, aussi bien techniques que personnelles, tout en m'adaptant aux exigences du projet.

C'était une expérience très enrichissante où j'ai appris beaucoup de nouvelles compétences. Cela m'a permis d'évoluer à plusieurs niveaux. Malgré quelques difficultés, j'ai réussi à les surmonter

Parmi les difficultés rencontrées, il y avait :

1. L'apprentissage de nouvelles technologies dans un délai très court, ce qui demandait beaucoup de concentration et d'adaptation.
2. La gestion du temps, un défi majeur pour équilibrer les différentes tâches et respecter les délais fixés.
3. Le travail en équipe, où il a fallu développer une bonne communication et apprendre à collaborer efficacement avec les autres membres du groupe.

Par ailleurs, j'ai effectué un stage de formation dans une entreprise de fourniture 3M Precision situé à rabat

En effet, l'ANAPEC, en partenariat avec des entreprises canadiennes, a lancé une formation de 450 heures en **Java JEE** et **Oracle**. J'ai eu la chance d'être accepté après un entretien technique avec Oracle pour accéder à cette formation. Celle-ci s'est déroulée en présentiel, et en parallèle,  
  
, où j'ai contribué à la réalisation d'une application de gestion de stock visant à optimiser les processus internes. J'ai utilisé plusieurs technologies, notamment SQL pour la base de données, JEE pour le backend, et React pour le frontend.

Récemment, j'ai finalisé un projet freelance pour une école privée. Nous avons développé une application web dédiée à la gestion et au suivi des démarches des étudiants, enseignants et administrateurs. Dans ce projet. Mon rôle comprenait la conception et le développement de microservices. Cette expérience m'a permis non seulement d'apprendre de nouvelles technologies, mais aussi de renforcer mes compétences en travail à distance et en gestion de situations complexes.

**PFE :**

**Problematique :**

L’état actuel de l’échange d’informations au sein du département informatique se caractérise par plusieurs inefficacités :

1. **Extraction manuelle des données :** Lorsqu’un service a besoin de données d’un autre (par exemple, la finance a besoin d’informations sur les passagers auprès du service voyage), il doit recourir à des méthodes manuelles telles que l’envoi d’e-mails ou la création de rapports personnalisés. Cette approche est chronophage, sujette aux erreurs et entrave la prise de décision en temps réel.
2. **Formats de données incohérents :** Les données sont souvent stockées dans des formats différents selon les systèmes départementaux, ce qui nécessite des efforts supplémentaires pour les traduire et les harmoniser en vue de leur utilisation dans d’autres applications.
3. **Visibilité limitée des données :** Les données d´départementales restent souvent invisibles pour les autres services, ce qui entrave une vision holistique des opérations et nuit aux efforts coordonnés.
4. **Latence des données :** Les retards dans la mise à jour des informations entre les systèmes conduisent à des données obsolètes, ce qui peut avoir un impact sur des taches critiques telles que la planification ou l’allocation des ressources. Ces inefficacités entrainent plusieurs conséquences négatives :
5. **Productivité réduite :** Le temps consacré à l’extraction et à la manipulation manuelles des données détourne des activités principales, ce qui nuit à l’efficacité des services.
6. **Augmentation des erreurs :** Les processus manuels sont plus sujets aux erreurs, ce qui peut avoir un impact sur les calculs financiers, la précision des rapports et les d´décisions opérationnelles.
7. **Frustration et retards :** La nature chronophage de l’acquisition de données entraine de la frustration et des retards dans l’accomplissement des tâches.
8. **Collaboration limitée** : Les difficultés d’accès et de partage des données créent des obstacles à la collaboration entre les services, ce qui entrave une approche unifiée de la résolution des problèmes.

**Solution**

Le projet **ISIP** vise à relever ces défis en développant une plateforme complète d’intégration des systèmes d’information.

**Objectifs du projet**

* **Accès centralisé aux données** :  
  Créer un référentiel central où toutes les données départementales résident dans un format cohérent et standardisé. Cela garantit à chacun l’accès aux informations les plus récentes.
* **Échange de données standardisé** :  
  Développer des **API** (interfaces de programmation d’applications) pour le système de chaque service. Ces API serviront d’interfaces normalisées, facilitant l’échange de données transparent entre les applications et éliminant le besoin d’intégrations personnalisées.
* **Disponibilité des données en temps réel** :  
  Mettre en œuvre des mécanismes pour garantir que les mises à jour des données se reflètent sur la plateforme en temps réel. Cela permet aux utilisateurs d’accéder aux informations les plus récentes pour une prise de décision éclairée.
* **Collaboration améliorée** :  
  Favoriser une culture de collaboration en fournissant une plateforme permettant aux services de partager et d’utiliser facilement les données, facilitant ainsi les efforts conjoints.

**Solution proposée**

Pour répondre aux défis existants, le projet **ISIP** propose les solutions suivantes :

* **Développement d’API** :  
  Des API seront développées pour chaque service au sein du département informatique, permettant un accès standardisé aux données et aux fonctionnalités.
* **Plateforme d’Intégration** :  
  Une plateforme d’intégration centralisée sera créée, où toutes les API seront hébergées et gérées. Cette plateforme servira de **hub d’échange de données**, garantissant la cohérence et la fiabilité.
* **Modèle d’Abonnement** :  
  Les utilisateurs des différents services pourront s’abonner aux API pertinentes en fonction de leurs besoins. Ce modèle d’abonnement permettra un accès transparent aux données, éliminant ainsi le besoin d’intervention manuelle.
* **Mesures de Sécurité** :  
  Des mesures de sécurité robustes seront mises en œuvre pour protéger les données sensibles et assurer la conformité aux réglementations sur la protection des données.

**Présentation génerale du projet :**

Le projet consiste en une plateforme destinée à intégrer plusieurs systèmes d’information. Lorsqu’on me demande, en entretien, de présenter mon projet de fin d’études, j’explique d’abord la problématique à laquelle nous avons cherché à répondre.

Actuellement, l’extraction des données se fait de manière manuelle. Par exemple, lorsqu’un service a besoin des données d’un autre (par exemple, le service RH qui souhaite obtenir une liste des marchés du service financier), il doit envoyer un e-mail, passer un appel ou se déplacer physiquement. Ce processus est long et nuit à la productivité. De plus, les données sont souvent disponibles sous différents formats (papier, Excel, e-mail, etc.), ce qui complique leur exploitation. La collaboration entre les services est ainsi limitée.

Pour remédier à ces problèmes, l’idée initiale fut de développer une API pour chaque service afin de faciliter l’échange de données. Cependant, cette approche soulevait d’autres défis, notamment en termes de sécurité et de gestion des accès, puisque fournir un accès total à chaque API n’est pas envisageable.

Nous avons donc opté pour une plateforme API centralisée qui intègre l’ensemble des API et des systèmes d’information. Le principe est le suivant : la plateforme gère les utilisateurs, les API et les autorisations. Un utilisateur crée un compte pour accéder à un catalogue des API disponibles et peut alors demander l’autorisation d’accéder à une API en précisant la période souhaitée et en justifiant sa demande.

Une notification est alors envoyée à l’administrateur, qui, ayant pour mission de gérer les utilisateurs et les demandes, peut accepter ou refuser la demande. Concernant l’accès aux API, nous avons adopté le mécanisme classique de la clé API. Toutefois, fournir une clé différente pour chaque API (par exemple, 20 clés pour 20 API) n’est pas pratique. J’ai donc proposé d’attribuer une seule clé API à l’utilisateur. Cette clé est un JWT contenant l’ID de l’utilisateur et une date d’expiration, et elle est utilisée de manière similaire à une clé API classique. La plateforme, qui joue également le rôle d’API Gateway, se charge de la vérification : elle extrait l’ID et la date d’expiration, puis, à partir de l’ID, consulte les autorisations correspondantes pour chaque API. Par ailleurs, nous avons sécurisé le chemin /api/ dans la configuration Spring, ce qui signifie que seul un utilisateur autorisé peut y accéder. Le filtre de sécurité extrait ensuite le nom de l’API (exemple : /api/api\_name) et vérifie si l’utilisateur possède l’autorisation nécessaire.

Voilà l’idée générale de la plateforme.

**Les bases de JAVA :**

**Pourqui Java ?**

1. Portabilité (write once, run everywhere!)

* Le code java fonctionne sur toutes les platformes
* Pas besoin de reécrire le code pour differents systemes
* Compatible avec windows, mac, linux, android…

1. Robustesse :

* Gestion automatique de mémoire grace au Garbage collector

1. Riche écosystème :

* Nombreuses bibliothèques standards (JAVA API)
* Frameworks populaires (Spring,Hibernate)

1. Applications variées :

Applications d’entreprises, desktop, android, web…

1. JDK, JRE, JVM :
2. JVM (Java Virtuel Machine)

- Machine virtuelle qui exécute le bytecode java

- Assure la portabilité du code java.

- Gère la mémoire et la sécurité des applications

b- JRE (Java Runtime Envirement)

* L’environnement d’exécution java
* Néssecaire pour exécuter les applications java
* Contient : la JVM et les bibliothèques standards java

1. JDK (Java Developement Kit)

* C’est l’environnement complet de développement Java
* Inclus tous les outils nécessaires pour développer des applications java
* Contient : compilateur (Javac), débogueur, documentation et le JRE

1. OOP :

Définition :

* Un paradigme de programmation basé sur le concept de l’OBJET(une entité de la vrai vie, comme une arbre, person,voiture…)
* Permet de structurer le code d’une manière modulaire et réutilisable.

Concepts fondamentaux :

1. Classe : blueprint

* C’est un modèle ou un plan
* Définit les propriétés et comportement des objets

1. Object :

* Instance d’une classe
* Representation concrete d’une classe

1. Attributs : (l’état)

* Variables de la classe
* Representent l’etat de l’objet

1. Methods : (Conportement)

* Fonctions de la classe
* Définissent le comportement de l’objet

1. Les Quatres pilliers de la POO :
2. L’Héritage :

* Permet a une classe d’hériter des propriétés d’une autre
* Favorise la réutilisation du code

Example: class Chat extends Animal

1. L’Encapsulation :

* Protection des données en les rendant privées
* Accès controlé via des getters et setters.

1. Le Polymorphisme :

* Capacité d’un Objet de prendre plusieurs formes
* Inclut la Surcharge(Overloading) et la redéfinition(Overriding) des méthodes.

1. L’Abstraction :

* Cacher les détails complexes, montrer uniquement l’essentiel
* Utiliser des classes et des méthodes abstraites.

Example :

Abstract Class Animal{….

}

1. Override et Overload :
2. Override (Redéfinition)

* Concerne l’héritage (relation parent - fille)
* Même signature de méthode dans la classe mère
* Utiliser l’annotation @Override
* même nom, paramètres, type de retour.

Example :

public class Animal {

public void sound(){

Systeme.out.println(“make sound”);

}

}

public class Chat extends Animal{

@Override

public void sound(){  
Systeme.out.println(“MEOW”);  
}

}

1. Overload (Surcharge):

* Ne concern pas l’heritage
* Dans la meme classe
* Même nom de méthode mais paramètres différents
* Doit avoir une liste des paramètres différents

Example :

public class Calculatrice{

public int add(int a, int b){

return a+b;

}

public double add(double a,double b){

return a+b;

}

public int add(int a,int b, int c){

return a+b+c;

}

}

1. Abstract class vs interface :

|  |  |
| --- | --- |
| Class | interface |
| Methods | Can have both abstract and concrete methods | Can only have abstract until java 8 they added default and static methods |
| Constructor | Can have a constructor | Cant have a constructor |
| Instance variables | Can have instance variables (fields) | Can only have constants (implicitly `public static final`) |
| Acces modifiers | Can have any acces modifiers (public,private,protected) | Methods are public by default |
| inheritance | A class can extend only one abstract class | A class can implement multiple interfaces |

**Les nouveautés de Java 8 :**

* Lambda expression: enables functional programming by writing functions inline   
   function((…) -> {………..})
* Stream API : Process collections in a functional style.
* Functional interfaces : interfaces with a single abstract method (Predicate,Function…)
* Optional : Avoid NullPointerException
* Defaults methods : add default implementations in interfaces.
* Date and Time API: Improved handling of times and dates.
* Method Reference : simplified syntax for calling methods (Class::Method)

1. Functional interfaces :

Functional interfaces have exactly one abstract method and supports lambda expression.

Example :

Java.util.function;

* Predicate<T> ========== filter(T t) in streams

Method : boolean test(T t)

Used for boolean testing

Use case :

Predicate<String> isLong = str -> str.length() > 10;

System.out.println(isLong.test(“Hello World!”); //true

* Function<T,R> ========== map(T t,R r) in streams

Method : R apply(T t)

Transforms input into different output

Use case :

Function<String,Integer> length = str -> str.length();

System.out.println(length.apply(“hello”); //5

* Consumer<T> ========== forEach() in streams

Method : void accept(T t)

Takes input performes actions return nothing

Use case :

Consumer<String> printer = message -> System.out.println(message)

Printer.accpet(“hello World!”);// Hello World!

* Supplier<T> ============ generate() in streams

Method : T get()

Takes nothing returns a result

Use case :

Supplier<Double> random = () -> Math.random();

System.out.println(random.get()); // random number

java.util.Lang;

* Runnable ????????

Method : void run()

1. Streams :

* A stream represent a sequence of elements for processing.

List<Integer> numbers = Arrays.asList(1,2,3,4,5,6,7,8,9);

Set<Integer> evens = numbers.stream()

.filter(n -> n%2==0) .collect(Collectors.toSet());

* Intermediate Operations: (returns a stream)

.filter(n -> n%2==0)

* Terminal Operations : (consume the Stream)

.collect(Collectors.toSet());

Collectors : collect and add to Set.

Example of Collectors :

* toList(),toSet() convert to a list or set.
* joining() concatenate strings.
* groupingBy() group elements by key.
* partitioningBy() partition elements into two groups.

collect : convert the stream into collection.

1. Optional :

* Used to avoid NullPointerException
* Methods :
* of(value) Create an Optional with non-null value.
* empty() Create an empty Optional.
* ifPresent() Perform an action if a value is present.
* isPresent() checks if value is present.

1. Method Reference :

* A shorthand for lambda expression

Types :

* Static methods : Class::methodName
* Instance methods : instance::MethodName
* Consturctors : ClassName::new

**Spring Boot**

**Dans un projet, il y a les exigences fonctionnelles et les exigences techniques. Dans les exigences techniques, il faut prendre en considération la maintenance. Pour créer une application facile à maintenir, il faut qu'elle soit fermée à la modification et ouverte à l'extension.**

**Pour créer un projet qui respecte tous ces critères, il faut utiliser le principe de l'inversion de contrôle. Ce principe permet au développeur de se concentrer sur le code métier, tandis que le framework s'occupe du code technique, notamment la sécurité, les transactions et l'accès aux données. Parmi les frameworks connus, on peut citer Spring.**

**L'inversion de contrôle est rendue possible grâce au paradigme de la programmation orientée Aspect (qui permet de séparer le code en couches : sécurité, métier, DAO, etc.).**

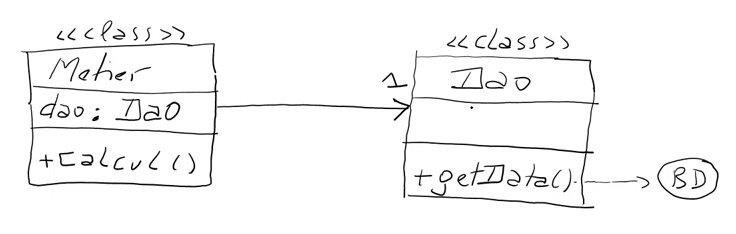
1. L’injection des dependences: (DI)

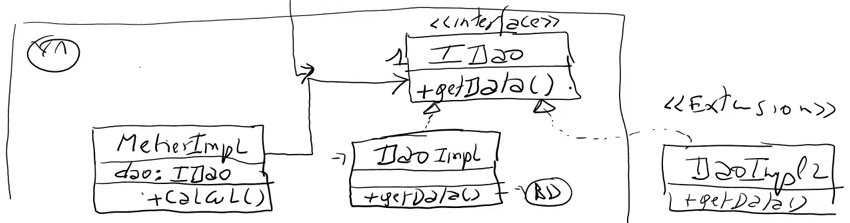
Est un design pattern utilisé pour implémenter le principe de l’inversion de contrôle (IoC) elle consiste à déléguer la gestion des dépendances d’une classe a une entité externe comme framework par exemple.

Pourquoi ?

* Découplage : la DI permet de réduire le couplage entre les classes, la classe deviennent plus modulaire et plus facile a maintenir.
* Flexibilité et évolutivité : les dépendances peuvent être remplacées ou configurées sans modifier le code de la classe.

1. **Une application fermée a la modification est ouverte à l’extension :**
2. **Couplage Fort :**

****

1. **Couplage Faible :**

* depend de l'interface instead of class (couplage faible)
* dao : IDao
* la class MetierImpl peut fonctionner avec n'importe quelles classes qui implémente l'interface IDao