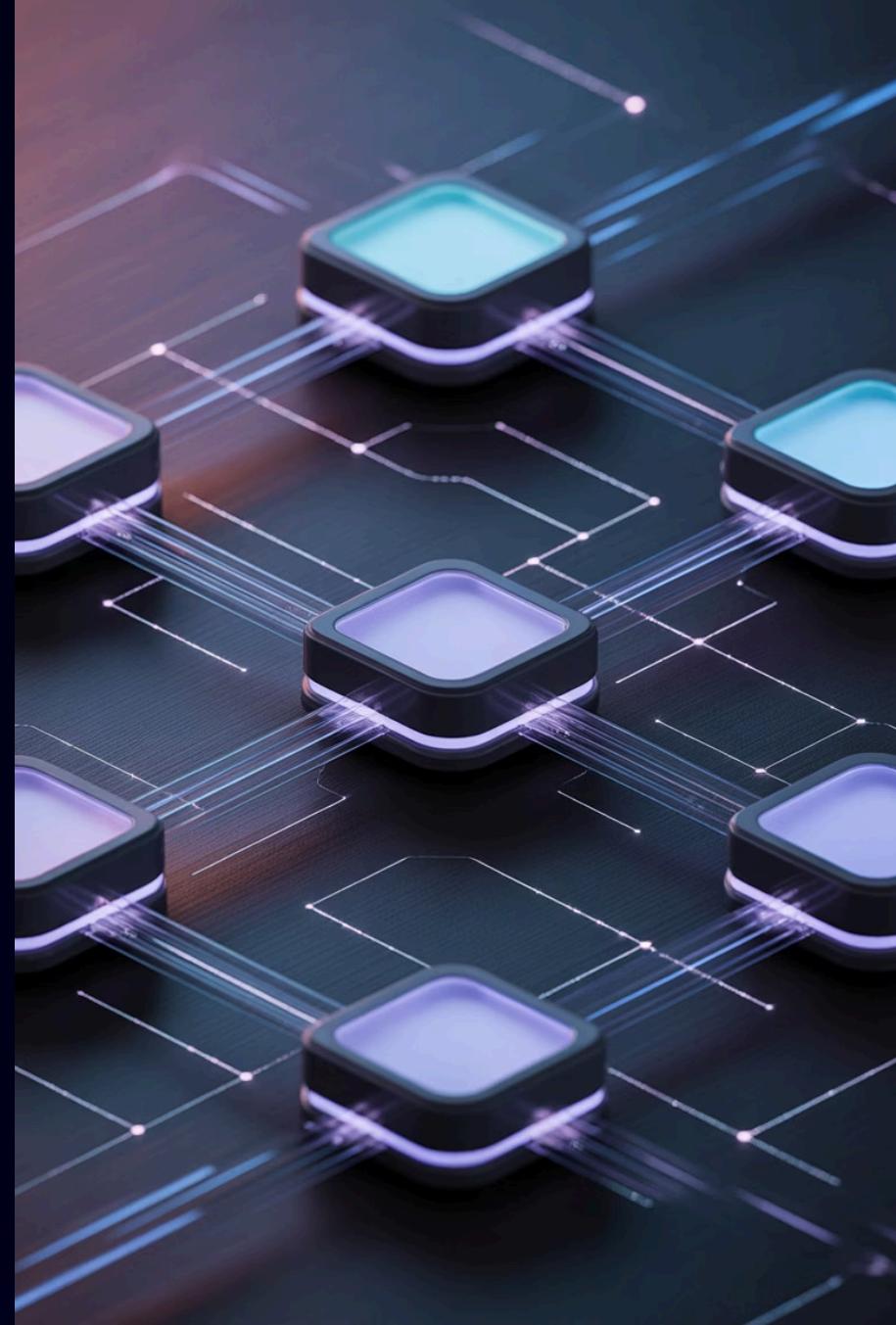


Maîtriser PlantUML

Guide Pratique pour les Diagrammes de Cas d'Utilisation

PlantUML est un outil puissant qui permet de créer des diagrammes UML à partir de code textuel. Ce guide vous accompagnera pas à pas dans la création de votre premier diagramme de cas d'utilisation, une compétence essentielle pour tout étudiant en BTS SIO. Vous découvrirez comment transformer des spécifications textuelles en représentations visuelles professionnelles.



Pourquoi PlantUML pour vos Diagrammes ?

Avantages Techniques

- Syntaxe simple et intuitive basée sur du texte
- Génération automatique de diagrammes professionnels
- Versionnement facile avec Git
- Compatible avec de nombreux éditeurs de code
- Export en plusieurs formats (PNG, SVG, PDF)

Bénéfices Pédagogiques

- Focus sur la logique plutôt que le design
- Modifications rapides et efficaces
- Apprentissage progressif de la syntaxe
- Cohérence visuelle garantie
- Collaboration facilitée entre développeurs

PlantUML s'impose comme un standard dans l'industrie pour la documentation technique. Sa maîtrise représente un atout majeur pour votre future carrière professionnelle.

Prérequis et Configuration

01

Installation de Java

PlantUML nécessite Java Runtime Environment (JRE) version 8 ou supérieure. Vérifiez votre installation avec la commande `java -version` dans votre terminal.

02

Téléchargement de PlantUML

Récupérez le fichier `plantuml.jar` depuis le site officiel plantuml.com. Ce fichier unique contient tout le nécessaire pour générer vos diagrammes.

03

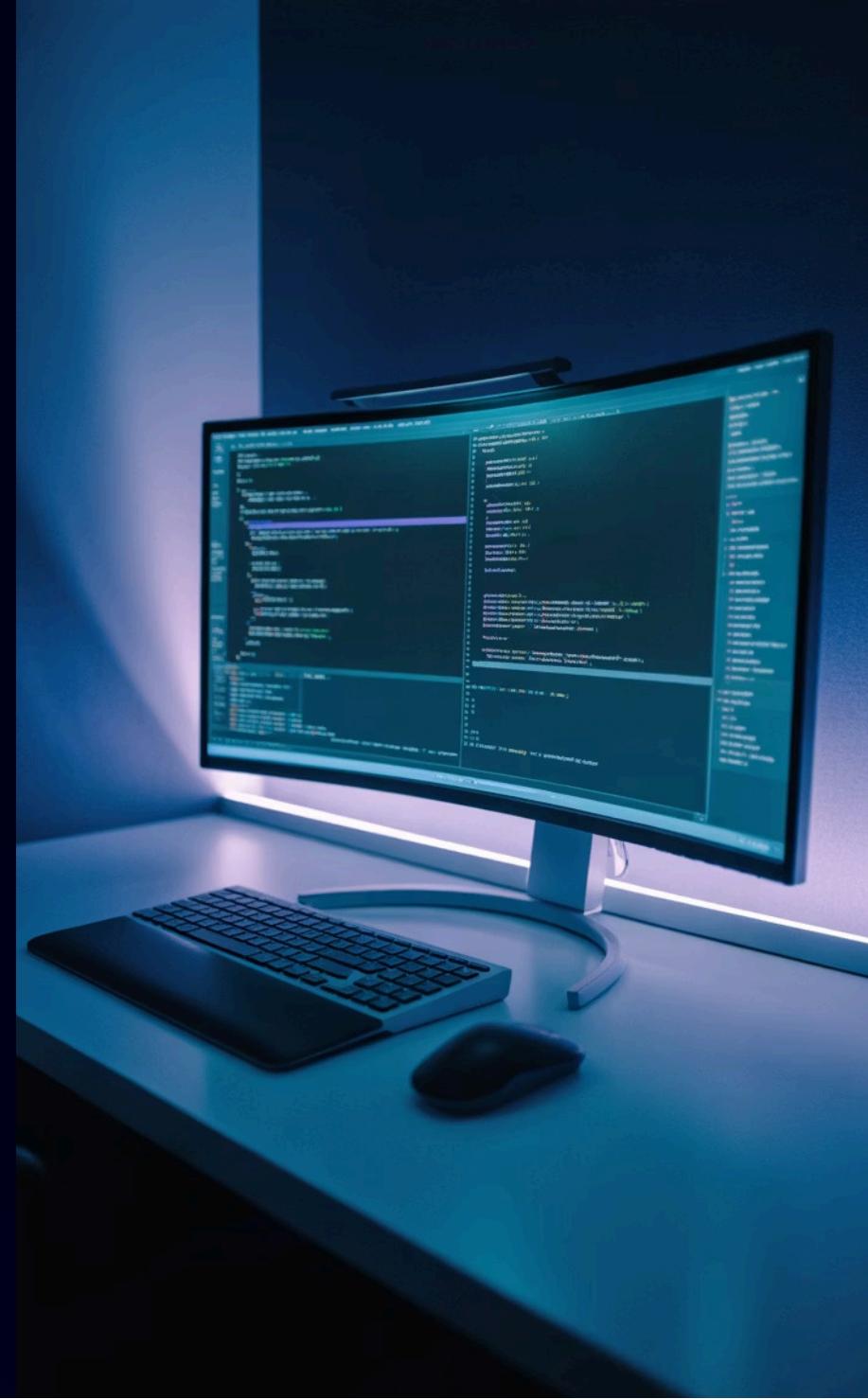
Choix de l'Éditeur

Sélectionnez votre environnement : Visual Studio Code avec l'extension PlantUML, IntelliJ IDEA, Eclipse, ou l'éditeur en ligne disponible sur le site officiel.

04

Test de Configuration

Créez un fichier `test.puml` et générez votre premier diagramme simple pour valider que tout fonctionne correctement.



Étape 1 : Structure de Base du Diagramme

Tout diagramme PlantUML commence par la balise `@startuml` et se termine par `@enduml`. C'est la structure minimale obligatoire qui encadre votre code. Créons notre premier fichier avec ces éléments essentiels.

```
@startuml  
/  
Diagramme de cas d'utilisation d'une application  
Ce diagramme montre :  
- Les acteurs : à compléter  
- Les cas : à compléter  
à compléter  
'/  
@enduml
```

PlantUML supporte deux types de commentaires pour documenter votre code. Les commentaires sur une seule ligne commencent par une apostrophe simple ('), comme par exemple : ' Diagramme UML de cas d'utilisation concernant une application de gestion scolaire. Pour des blocs de texte plus longs, vous pouvez utiliser des commentaires multilignes qui débutent par /' et se terminent par '/, comme illustré dans l'exemple ci-dessus. Ces commentaires sont essentiels pour rendre votre code lisible et sont entièrement ignorés lors de la génération de votre diagramme. À ce stade, le diagramme généré sera vide, mais la structure est prête à recevoir nos éléments.

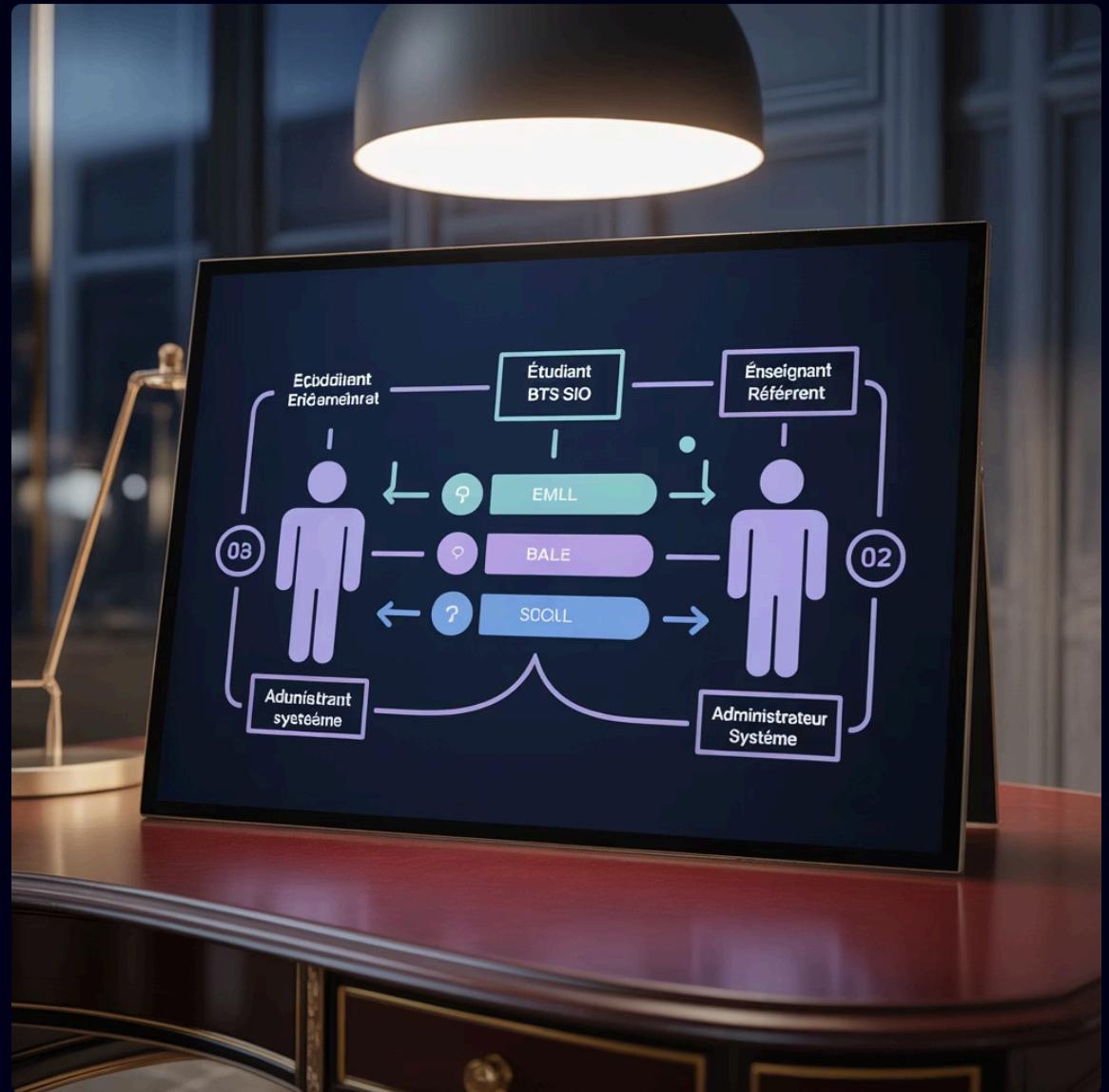
- **Astuce :** Sauvegardez toujours vos fichiers PlantUML avec l'extension `.puml` ou `.plantuml` pour une meilleure reconnaissance par les éditeurs et outils de versionnement.

Étape 2 : Ajout des Acteurs avec Alias

Les acteurs représentent les utilisateurs ou systèmes externes qui interagissent avec votre application. Dans PlantUML, on les déclare avec le mot-clé `actor`. Ajoutons nos premiers acteurs au diagramme, en utilisant des alias pour simplifier leurs références.

```
@startuml  
' Diagramme UML de cas d'utilisation concernant une  
application de gestion scolaire  
actor "Étudiant BTS SIO" as etudiant  
actor "Enseignant Référent" as enseignant  
actor "Administrateur Système" as admin  
@enduml
```

Chaque acteur apparaît comme une figure en bâton (stick figure) dans le diagramme. Nous avons ici utilisé des noms plus descriptifs avec des espaces. L'utilisation des alias (comme `etudiant` pour "Étudiant BTS SIO") est particulièrement utile dans ce cas, car elle permet de simplifier les références à ces acteurs dans le reste du diagramme, surtout quand leurs noms complets contiennent des espaces ou des caractères spéciaux. Cela rend le code plus propre et plus facile à manipuler.



Les trois acteurs sont maintenant visibles dans votre diagramme, avec leurs noms complets et positionnés automatiquement par PlantUML.

Les acteurs sont généralement placés à gauche du diagramme par défaut, mais nous verrons comment contrôler leur position dans les étapes suivantes. L'utilisation des alias ne modifie pas l'apparence du diagramme, mais simplifie grandement la lisibilité du code PlantUML.

Étape 3 : Définition des Cas d'Utilisation

Les cas d'utilisation décrivent les fonctionnalités du système. Ils sont représentés par des ellipses et déclarés entre parenthèses. Enrichissons notre diagramme avec des cas d'utilisation concrets pour un système de gestion scolaire.

```
@startuml  
actor Étudiant  
actor Enseignant  
actor Administrateur  
  
usecase "Consulter les notes" as UC1  
usecase "Déposer un devoir" as UC2  
usecase "Saisir les notes" as UC3  
usecase "Gérer les utilisateurs" as UC4  
  
@enduml
```

L'alias (introduit par `as`) permet de référencer facilement chaque cas d'utilisation dans la suite du code. C'est une bonne pratique qui simplifie la maintenance, surtout pour les diagrammes complexes avec de nombreux éléments.



Étape 4 : Crédit des Relations

Les relations connectent les acteurs aux cas d'utilisation qu'ils peuvent réaliser. On utilise des flèches simples pour créer ces associations. C'est ici que le diagramme prend tout son sens fonctionnel.

```
@startuml
actor Étudiant
actor Enseignant
actor Administrateur

usecase "Consulter les notes" as UC1
usecase "Déposer un devoir" as UC2
usecase "Saisir les notes" as UC3
usecase "Gérer les utilisateurs" as UC4

Étudiant --> UC1
Étudiant --> UC2
Enseignant --> UC1
Enseignant --> UC3
Administrateur --> UC4
@enduml
```

Chaque ligne avec `-->` crée une association entre un acteur et un cas d'utilisation. Le diagramme montre maintenant clairement qui peut faire quoi dans le système. Les flèches peuvent également être inversées avec `<--` selon votre préférence de lecture.

Étape 5 : Ajout d'un Rectangle Système

Le rectangle système délimite visuellement la frontière de votre application. Il permet de distinguer clairement ce qui est à l'intérieur du système (les cas d'utilisation) et ce qui est à l'extérieur (les acteurs). C'est un élément crucial pour la clarté du diagramme.

```
@startuml
left to right direction

actor Étudiant
actor Enseignant
actor Administrateur

rectangle "Système de Gestion Scolaire" {
    usecase "Consulter les notes" as UC1
    usecase "Déposer un devoir" as UC2
    usecase "Saisir les notes" as UC3
    usecase "Gérer les utilisateurs" as UC4
}

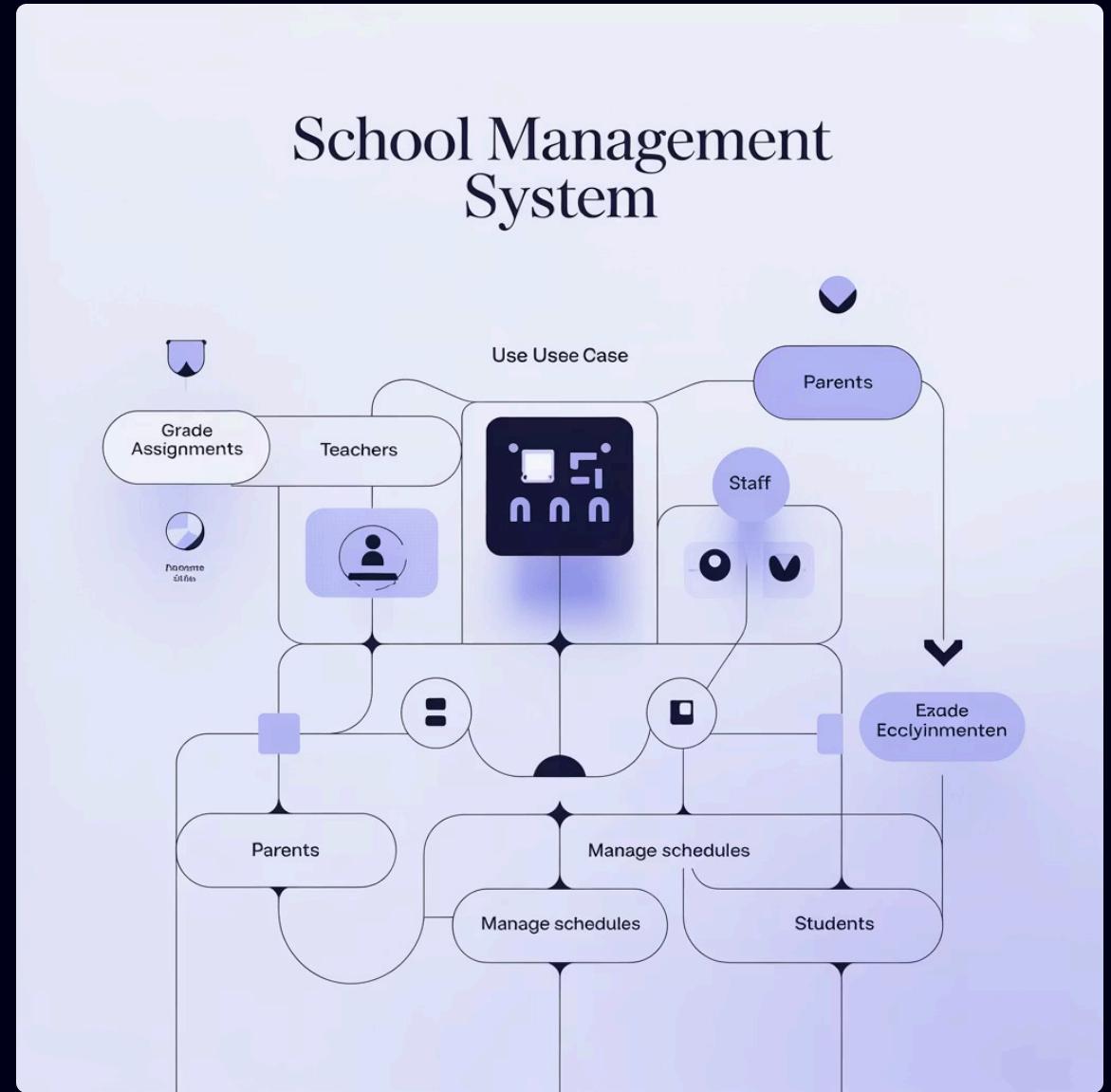
Étudiant --> UC1
Étudiant --> UC2
Enseignant --> UC1
Enseignant --> UC3
Administrateur --> UC4
@enduml
```

La directive `left to right direction` améliore la lisibilité en orientant le diagramme horizontalement. Le rectangle englobant les cas d'utilisation matérialise les limites du système étudié.

Étape 6 : Organisation avec les Packages

Les packages sont un excellent moyen de regrouper logiquement les acteurs ou les cas d'utilisation, ce qui améliore considérablement la lisibilité et la structure des diagrammes complexes. Ils permettent de segmenter votre diagramme en zones fonctionnelles ou organisationnelles claires.

```
@startuml  
left to right direction  
  
package "Acteurs Internes" {  
    actor "Enseignant Référent" as enseignant  
    actor "Administrateur Système" as admin  
}  
  
package "Acteurs Externes" {  
    actor "Étudiant BTS SIO" as etudiant  
}  
  
rectangle "Système de Gestion Scolaire" {  
    package "Gestion Pédagogique" {  
        usecase "Consulter les notes" as UC1  
        usecase "Déposer un devoir" as UC2  
        usecase "Saisir les notes" as UC3  
    }  
  
    package "Administration" {  
        usecase "Gérer les utilisateurs" as UC4  
    }  
}  
  
etudiant --> UC1  
etudiant --> UC2  
enseignant --> UC1  
enseignant --> UC3  
admin --> UC4  
@enduml
```



Le diagramme montre une organisation claire des acteurs et des cas d'utilisation grâce à l'utilisation des packages.

L'utilisation des packages dans PlantUML, comme illustré ci-dessus, offre une meilleure organisation visuelle, facilite la maintenance du code et apporte une clarté indispensable pour la compréhension des diagrammes complexes. Chaque package agit comme un conteneur logique, rendant les grandes architectures beaucoup plus maniables.

Relations Avancées : Include et Extend

En PlantUML, au-delà des associations simples, il est possible de modéliser des relations plus complexes entre les cas d'utilisation, notamment les relations `<<include>>` et `<<extend>>`. Ces relations permettent de décrire la dépendance fonctionnelle entre différents cas d'utilisation, enrichissant ainsi la précision de vos diagrammes.

`<<include>>` : Nécessite obligatoirement

La relation `<<include>>` est utilisée lorsque l'exécution d'un cas d'utilisation de base nécessite obligatoirement l'exécution d'un autre cas d'utilisation. Le cas d'utilisation inclus est une partie intégrante du cas d'utilisation de base et se déroule à un moment spécifié par le cas de base.

Exemple : "S'authentifier" `<<include>>` "Consulter les notes"

(obligatoire)

`<<extend>>` : Nécessite éventuellement

La relation `<<extend>>` indique qu'un cas d'utilisation peut optionnellement étendre le comportement d'un autre cas d'utilisation. L'extension se produit sous certaines conditions et à des points d'extension définis dans le cas d'utilisation de base. Le cas d'utilisation étendu reste complet et fonctionnel même sans l'extension.

Exemple : "Consulter les notes" `<<extend>>` "Imprimer les notes"

(optionnel)

Voici le code PlantUML illustrant ces relations :

```
@startuml
left to right direction

actor "Étudiant BTS SIO" as etudiant

usecase "Consulter les notes" as UC_ConsulterNotes
usecase "S'authentifier" as UC_Authentifier
usecase "Imprimer les notes" as UC_ImprimerNotes

UC_ConsulterNotes .left.> UC_Authentifier : <<include>>
UC_ConsulterNotes .right.> UC_ImprimerNotes : <<extend>>

etudiant --> UC_ConsulterNotes
@enduml
```

Dans l'exemple ci-dessus, l'authentification est une étape obligatoire avant de pouvoir consulter les notes (`<<include>>`), tandis que l'impression des notes est une action optionnelle que l'étudiant peut choisir d'effectuer après avoir consulté ses notes (`<<extend>>`). Comprendre cette différence conceptuelle est crucial pour modéliser avec précision les interactions et les dépendances dans votre système.

Héritage et Généralisation d'Acteurs

La notion d'héritage, représentée par la flèche --|>, est essentielle pour modéliser des hiérarchies claires entre les acteurs dans un diagramme de cas d'utilisation. Elle permet de créer une structure où un acteur spécialisé hérite automatiquement de toutes les capacités (c'est-à-dire les cas d'utilisation auxquels il est associé) de l'acteur général dont il dérive. Cela simplifie le diagramme en évitant de dupliquer les associations et en reflétant la relation "est un type de".

```
@startuml  
left to right direction
```

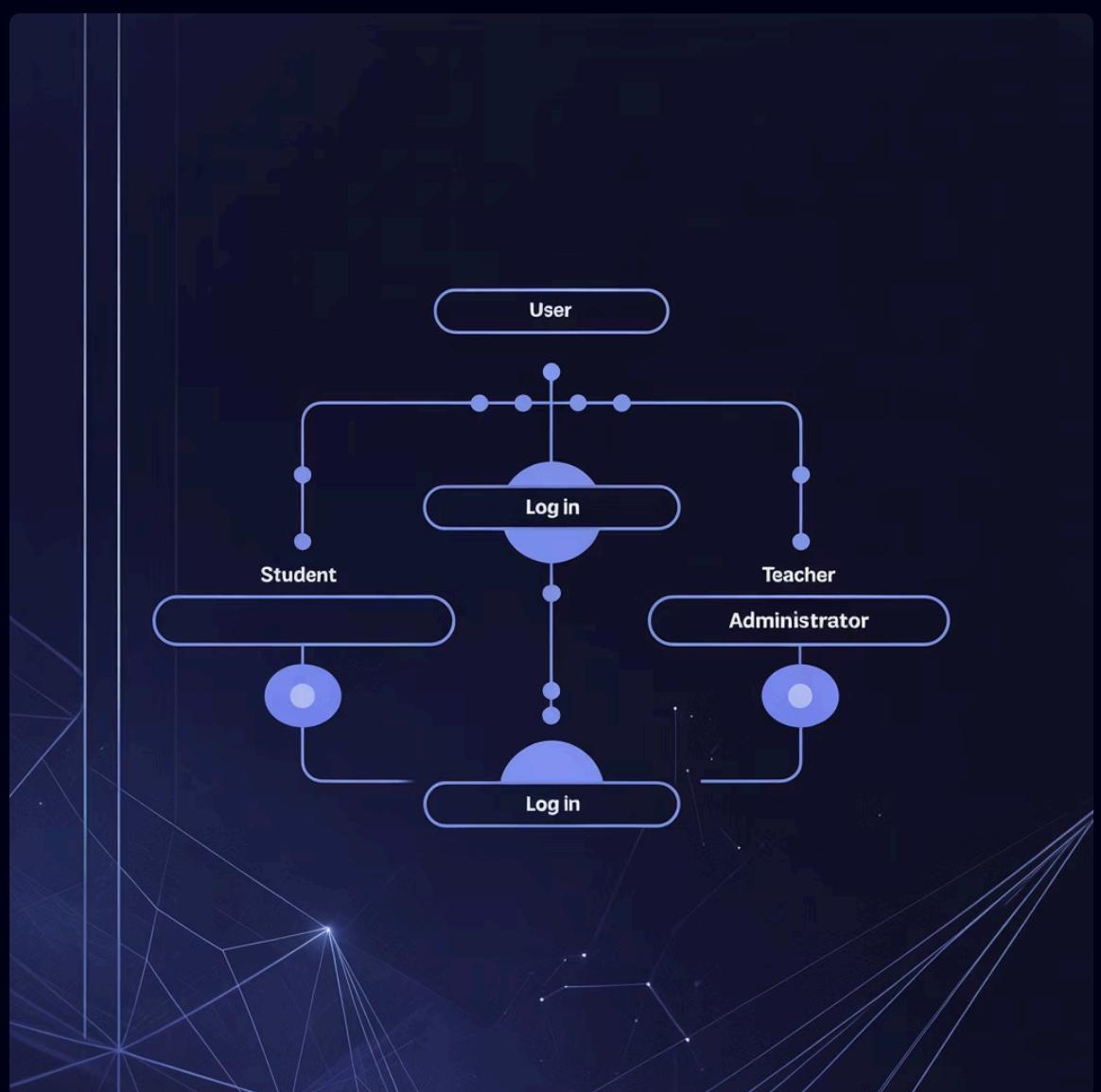
```
actor "Utilisateur" as user
```

```
actor "Étudiant" as etudiant  
actor "Enseignant" as teacher  
actor "Administrateur" as admin
```

```
user <|-- etudiant  
user <|-- teacher  
user <|-- admin
```

```
etudiant --> (Consulter les notes)  
teacher --> (Saisir les notes)  
admin --> (Gérer les utilisateurs)
```

```
user --> (Se connecter)  
@enduml
```



Dans cet exemple, l'acteur général "Utilisateur" peut se connecter. Les acteurs "Étudiant", "Enseignant" et "Administrateur" héritent de cette capacité et peuvent donc aussi se connecter, en plus de leurs cas d'utilisation spécifiques.

Cette approche montre que chaque acteur spécialisé (Étudiant, Enseignant, Administrateur) peut non seulement effectuer ses propres cas d'utilisation spécifiques (comme "Consulter les notes" pour l'Étudiant ou "Saisir les notes" pour l'Enseignant), mais aussi toutes les actions de l'acteur général "Utilisateur" (ici, "Se connecter").

De manière similaire, les cas d'utilisation peuvent également être généralisés. Cela permet de regrouper des cas d'utilisation similaires sous un concept plus large, augmentant la lisibilité et la modularité du diagramme.

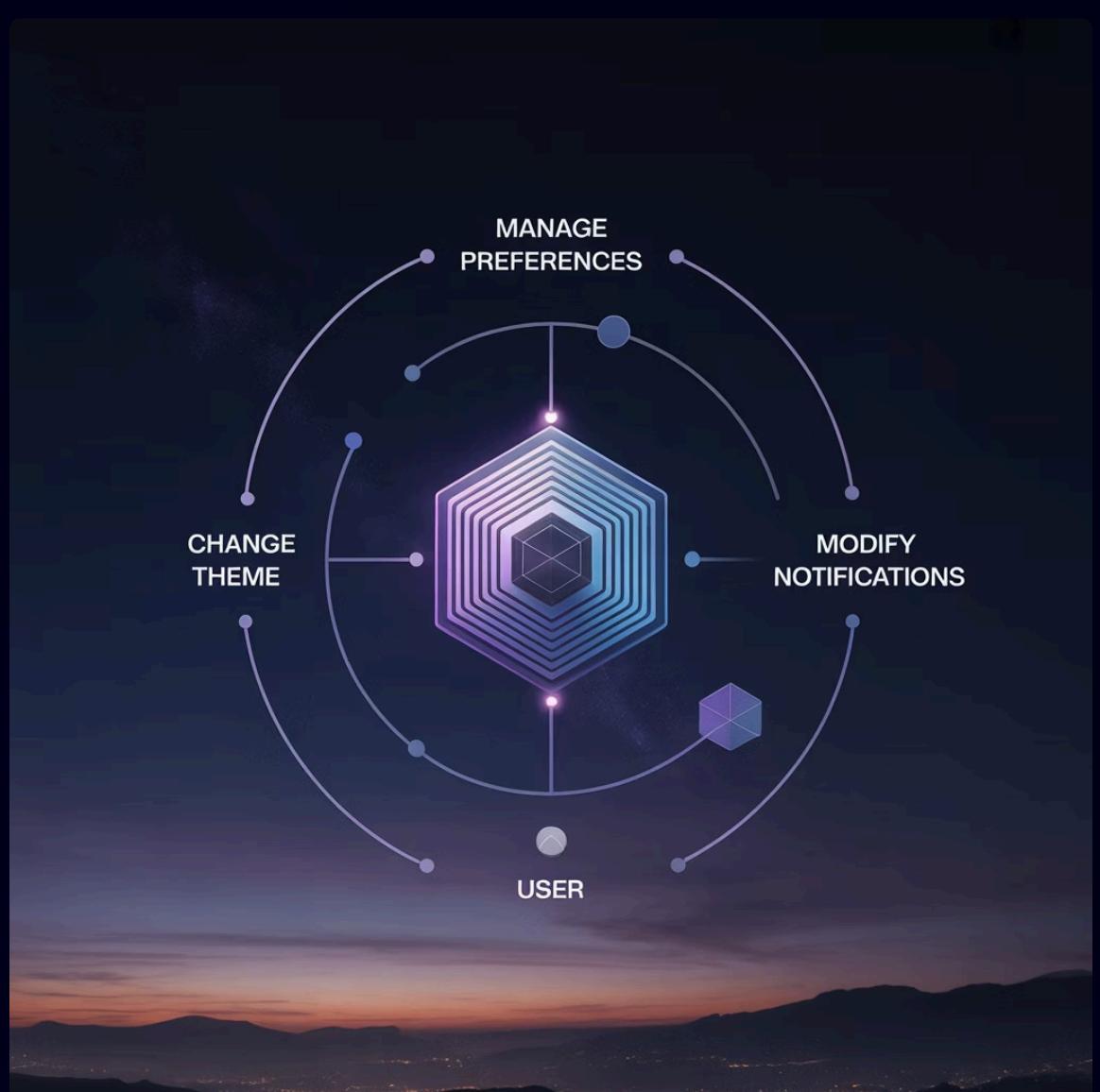
```
@startuml  
left to right direction
```

```
actor "Utilisateur" as user
```

```
usecase "Gérer les préférences" as UC_General  
usecase "Changer le thème" as UC_Theme  
usecase "Modifier les notifications" as UC_Notifications
```

```
UC_General <|-- UC_Theme  
UC_General <|-- UC_Notifications
```

```
user --> UC_General  
@enduml
```



Ici, "Gérer les préférences" est un cas d'utilisation général. "Changer le thème" et "Modifier les notifications" sont des cas d'utilisation spécialisés qui en héritent, signifiant qu'ils sont des façons spécifiques de gérer les préférences.

Interaction avec d'Autres Systèmes

Dans la modélisation des cas d'utilisation, il est fréquent qu'un système doive interagir avec d'autres systèmes externes pour effectuer des contrôles, des validations ou des échanges de données. Ces systèmes externes sont des entités autonomes avec lesquelles notre système principal communique. Pour les représenter dans un diagramme de cas d'utilisation, on peut les modéliser simplement comme des acteurs normaux, en utilisant des noms descriptifs qui indiquent clairement leur nature de système, sans nécessairement recourir à des stéréotypes.

Voici quelques exemples concrets de systèmes externes avec lesquels notre système pourrait interagir :

- Un système de paiement externe pour traiter les transactions financières liées aux frais de scolarité ou autres services.
- Un Service Email pour envoyer des communications automatisées aux utilisateurs.
- Un système d'authentification tel qu'un Serveur LDAP (Lightweight Directory Access Protocol) ou Active Directory pour vérifier l'identité des utilisateurs.
- Une Base Établissements externe pour récupérer des informations standardisées.

Le code PlantUML ci-dessous illustre comment ces interactions peuvent être modélisées en utilisant des noms explicites pour les acteurs représentant des systèmes externes :

```
@startuml
left to right direction

actor "Système Principal" as main_system

actor "Système de Paiement" as payment_system
actor "Serveur LDAP" as ldap_server
actor "Service Email" as email_api
actor "Base Établissements" as external_db

main_system -- (Effectuer un paiement)
(Effectuer un paiement) -- payment_system

main_system -- (Authentifier l'utilisateur)
(Authentifier l'utilisateur) -- ldap_server

main_system -- (Envoyer une notification)
(Envoyer une notification) -- email_api

main_system -- (Consulter les établissements)
(Consulter les établissements) -- external_db

@enduml
```



Ce diagramme met en évidence les points d'intégration entre le Système Principal et les services externes. L'utilisation de noms clairs et descriptifs comme "Système de Paiement" ou "Serveur LDAP" permet de distinguer naturellement les acteurs représentant des systèmes informatiques des acteurs humains, sans avoir besoin de stéréotypes spécifiques. Cette approche simplifie la lecture du diagramme tout en restant parfaitement claire sur la nature de chaque acteur.

Ajout de Légendes et de Notes Explicatives

Pour enrichir la documentation de vos diagrammes de cas d'utilisation et faciliter leur compréhension, PlantUML offre la possibilité d'ajouter des légendes et des notes explicatives. Ces annotations sont cruciales pour fournir un contexte, des détails supplémentaires ou des précisions sur des éléments spécifiques du diagramme ou sur le diagramme dans son ensemble.

→ Notes attachées à des éléments spécifiques

Ces notes sont directement liées à un acteur ou à un cas d'utilisation, utilisant des syntaxes comme note right of ActeurX : "Explication", note left of CasUtilisation : "Détail important", ou encore note top of et note bottom of pour des positions relatives.

→ Notes flottantes

Des notes indépendantes peuvent être ajoutées n'importe où sur le diagramme pour des explications générales ou des informations contextuelles, en les définissant avec note as N1 et en y ajoutant du texte.

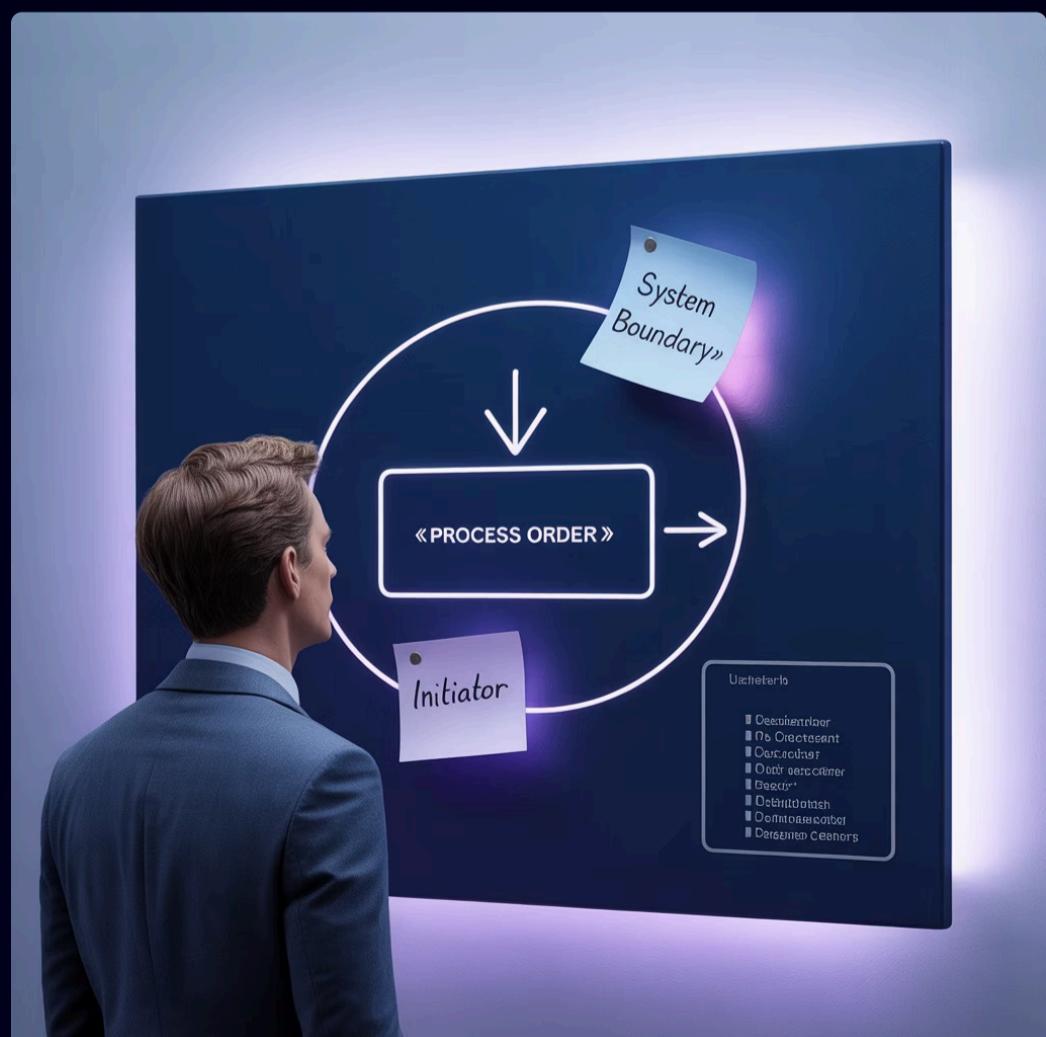
→ Légende

Une légende fournit une section dédiée à la fin du diagramme pour des informations récapitulatives, des définitions de termes ou des conventions utilisées, placée avec legend right ou legend left.

Voici un exemple complet de code PlantUML montrant ces différents types d'annotations :

```
@startuml  
left to right direction  
  
actor "Utilisateur Authentifié" as User  
usecase "Effectuer une commande" as UC_Order  
usecase "Consulter l'historique" as UC_History  
  
User --> UC_Order  
User --> UC_History  
  
note right of User : Cet acteur doit être connecté \npour accéder aux fonctionnalités.  
  
note left of UC_Order : Inclut la validation du panier \net le traitement du paiement.  
  
note "Note flottante sur les préconditions \npour toutes les actions." as N1  
(N1) .. User  
  
legend right  
= Légende du Diagramme  
* User : Représente tout utilisateur ayant un compte.  
* UC_Order : Cas d'utilisation principal pour les achats.  
* UC_History : Permet de revoir les transactions passées.  
end legend  
@enduml
```

Cet exemple illustre comment intégrer des notes et une légende dans un diagramme de cas d'utilisation. Les notes attachées aux éléments spécifiques comme l'acteur "Utilisateur Authentifié" et le cas d'utilisation "Effectuer une commande" fournissent des détails contextuels directement là où ils sont pertinents. La note flottante (N1) sert à communiquer des informations générales applicables à plusieurs éléments, tandis que la légende en bas à droite offre un récapitulatif des définitions et des rôles. Ces annotations sont essentielles pour la documentation et la compréhension du diagramme, surtout lorsqu'il est partagé avec différentes parties prenantes ou pour une référence future.



Personnalisation Visuelle avec skinParam

PlantUML offre une puissante fonctionnalité de personnalisation visuelle grâce à la commande `skinParam`. Celle-ci permet de modifier l'apparence de divers éléments du diagramme, tels que les couleurs, les polices, les styles de traits et même la forme des acteurs, afin d'améliorer l'esthétique et la lisibilité en fonction des besoins ou de la charte graphique.

Couleurs d'arrière-plan

- `skinParam backgroundColor #F0F0F0` (diagramme global)
- `skinParam actor.backgroundColor #lightblue` (acteurs)
- `skinParam usecase.backgroundColor #lightgreen` (cas d'utilisation)

Styles des traits et flèches

- `skinParam arrow.color #red`
- `skinParam arrow.thickness 2`

Forme des acteurs

- `skinParam actorStyle awesome` (icônes stylisées)
- `skinParam actorStyle hollow` (contour seulement)

Polices et couleurs de texte

- `skinParam defaultFontName Arial`
- `skinParam defaultFontColor #333333`

Voici un exemple complet montrant l'application de plusieurs de ces personnalisations pour un diagramme de cas d'utilisation simple :

```
@startuml
left to right direction

' Personnalisation des couleurs
skinparam backgroundColor #F0F0F0
skinparam actor.backgroundColor #lightblue
skinparam usecase.backgroundColor #lightgreen

' Personnalisation des traits et flèches
skinparam arrow.color #red
skinparam arrow.thickness 2

' Personnalisation de la forme des acteurs
skinparam actorStyle awesome

' Personnalisation des polices
skinparam defaultFontName Arial
skinparam defaultFontColor #333333

actor "Client" as Client
actor "Administrateur" as Admin

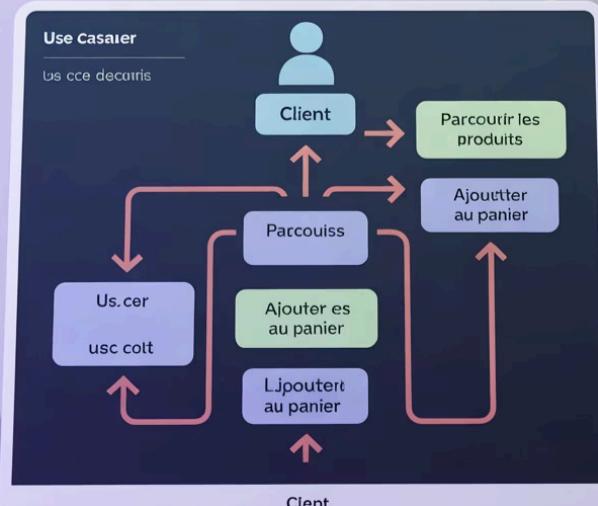
usecase "Parcourir les produits" as Browse
usecase "Ajouter au panier" as AddToCart
usecase "Payer la commande" as Pay
usecase "Gérer les produits" as ManageProducts
usecase "Consulter les ventes" as ViewSales

Client --> Browse
Client --> AddToCart
Client --> Pay

Admin --> ManageProducts
Admin --> ViewSales

@enduml
```

L'exemple ci-dessus démontre comment `skinParam` peut être utilisé pour transformer l'apparence par défaut d'un diagramme de cas d'utilisation. Le fond gris clair (#F0F0F0), les acteurs bleu clair avec un style "awesome" et les cas d'utilisation vert clair créent un contraste agréable. Les flèches rouges épaisses (arrow.color #red et arrow.thickness 2) attirent l'attention sur les relations, tandis que la police Arial et la couleur de texte gris foncé améliorent la lisibilité générale. Ces personnalisations sont essentielles pour adapter les diagrammes à un contexte visuel spécifique, les rendre plus engageants et faciliter leur interprétation par l'audience ciblée.



Bonnes Pratiques et Améliorations pour des Diagrammes PlantUML Professionnels



Organisation du Code

Groupez les déclarations par type (acteurs, cas d'utilisation, relations) et utilisez des blocs `package` pour structurer les diagrammes complexes. Ajoutez des commentaires clairs pour expliquer les sections ou les choix de conception.



Conventions de Nommage

Utilisez des alias courts et significatifs (ex: UC1, UC2). Privilégiez des noms de cas d'utilisation commençant par un verbe d'action à l'infinitif pour une meilleure clarté.



Relations Avancées (Include/Extend)

Maîtrisez les stéréotypes `include` et `extend` pour représenter des comportements optionnels ou obligatoires, améliorant la précision de vos diagrammes de cas d'utilisation.



Héritage et Généralisation

Appliquez l'héritage pour les acteurs (`<|--`) et la généralisation pour les cas d'utilisation afin de modéliser des rôles ou des fonctionnalités partagées, réduisant la redondance et augmentant la clarté.



Interaction avec d'Autres Systèmes

Représentez clairement les dépendances et les interactions avec des systèmes externes (bases de données, autres applications) en utilisant des acteurs spécifiques ou des notes explicatives.



Ajout de Légendes et Notes

Utilisez le bloc `legend` pour définir des conventions spécifiques au diagramme et les commandes `note right of` / `note left of` pour ajouter des précisions contextuelles directement sur les éléments.



Personnalisation avec `skinParam`

Exploitez `skinParam` pour aligner l'apparence de vos diagrammes avec la charte graphique de votre projet ou entreprise, en contrôlant les couleurs, polices, styles de traits et formes d'éléments.

Ces pratiques vous permettront de créer des diagrammes professionnels, maintenables et facilement compréhensibles par vos collègues et futurs recruteurs, garantissant une communication visuelle efficace des architectures logicielles.

Exercices et Prochaines Étapes

1

Exercice 1 : Organisation Avancée

Créez un diagramme de cas d'utilisation pour une application de gestion de projet. Utilisez des packages pour regrouper les cas d'utilisation par module, des alias significatifs pour les acteurs et les cas d'utilisation, et des commentaires multilignes pour expliquer les sections complexes du code.

2

Exercice 2 : Relations Complexes et Héritage

Enrichissez votre diagramme précédent en ajoutant des relations <<include>> et <<extend>> pour modéliser des comportements obligatoires et optionnels. Intégrez également l'héritage d'acteurs pour représenter des rôles spécialisés (ex: "Administrateur" héritant de "Utilisateur").

3

Exercice 3 : Projet Complet et Personnalisation

Développez un diagramme de cas d'utilisation complet pour votre projet de BTS SIO, en intégrant toutes les bonnes pratiques : représentation des interactions avec des systèmes externes, ajout de notes explicatives et de légendes, et personnalisation de l'apparence du diagramme via `skinParam` pour correspondre à une charte graphique définie.

[Documentation PlantUML](#)

[Éditeur en Ligne](#)

Continuez à pratiquer régulièrement ! La maîtrise de PlantUML s'acquiert par l'expérience. N'hésitez pas à consulter la documentation officielle pour découvrir toutes les possibilités offertes par cet outil remarquable.