

Introduction à la modélisation conceptuelle de données avec UML

Table des matières

I. Contexte	3
II. Les différentes phases d'un projet informatique	3
III. L'Unified Modeling Language	5
IV. Exercice : Appliquez la notion	6
V. Les spécifications fonctionnelles	6
VI. Exercice : Appliquez la notion	10
VII. Les spécifications techniques	11
VIII. Exercice : Appliquez la notion	13
IX. Essentiel	13
X. Auto-évaluation	13
A. Exercice final	13
B. Exercice : Défi	15
Solutions des exercices	16

I. Contexte

Durée : 1 h

Environnement de travail : Local

Pré-requis : Aucun

Contexte

Développer un projet informatique est quelque chose de souvent très complexe. En effet, pour mener à bien cette tâche, une équipe de développeurs va devoir créer un nombre important de fonctions et de classes pour gérer les besoins d'un client.

Or, c'est quelque chose qui est beaucoup plus facile à dire qu'à faire : dans la pratique, il faut à la fois s'assurer que tous les membres de l'équipe ont compris le besoin initial et que chacun respecte une structure de code commune, afin de pouvoir réutiliser les fonctionnalités développées par les autres. C'est pourquoi des phases d'analyse et de conception sont réalisées en amont de tout projet informatique.

Dans ce cours, nous allons donc voir tout ce qu'il se passe entre le moment où un client arrive avec une idée et celui où l'équipe commence à écrire du code.

II. Les différentes phases d'un projet informatique

Objectifs

- Comprendre les différentes phases en amont d'un projet informatique
- Distinguer les différents documents et les différents acteurs de ces phases

Mise en situation

Développer un projet, c'est transformer les idées d'un client en outil informatique : le rôle d'un développeur est de traduire un besoin en lignes de code. Or, un projet est la confrontation entre deux mondes totalement différents : le client est rarement initié à l'informatique et les développeurs ne connaissent pas le métier du client. Avant de pouvoir commencer un projet, il faut donc s'assurer que les volontés du client soient bien réalisables et que tous les détails du projet sont décrits de manière compréhensible pour les développeurs.

Le cahier des charges

La toute première étape nécessaire au lancement d'un projet informatique est la présentation du **cahier des charges**. Il contient l'ensemble des demandes du client, qu'il a rédigées lui-même. Il peut parfois être aidé par un membre de l'équipe : le Product Owner, dont le rôle est d'être l'interlocuteur principal du client. Pour cela, il doit comprendre son métier et ses besoins afin de pouvoir aider à leur retranscription à l'équipe.

Les spécifications fonctionnelles

Une fois le cahier des charges réalisé, le projet entre en **phase d'analyse**. Pendant cette phase, le Product Owner va analyser le cahier des charges pour en déterminer les spécifications fonctionnelles, c'est-à-dire la liste exhaustive de toutes les fonctionnalités de l'application.

Les spécifications fonctionnelles ont un double rôle : elles permettent d'établir un premier découpage de l'application par fonctionnalité (ce qui permettra une meilleure répartition des tâches pendant la phase de développement), mais sont aussi la preuve de la bonne compréhension du cahier des charges et du contexte par l'équipe de développement. Le client doit valider ces spécifications fonctionnelles, ou les corriger si besoin, avant de passer à l'étape suivante.

Remarque

Chaque entreprise possède sa propre organisation et peut confier certaines responsabilités à d'autres rôles. Par exemple, il arrive que certaines équipes n'aient pas de Product Owner : l'élaboration des spécifications fonctionnelles est, dans ce cas, à la charge du chef de projet.

Les spécifications techniques

Une fois l'analyse terminée, il faut réaliser une **phase de conception**. Cette étape est importante, car il existe autant de manières de coder un même projet que de développeurs. Sans ligne directrice, il y a de fortes chances pour que le découpage en classes ou en fonctions soit différent d'un membre de l'équipe à l'autre. Il est donc nécessaire de réfléchir à l'avance à l'architecture du programme afin que toute l'équipe puisse avancer dans le même sens. Pour cela, l'architecte logiciel doit écrire les spécifications techniques du projet, c'est-à-dire l'architecture générale du code.

Remarque

Il arrive souvent que les spécifications techniques soulèvent des interrogations remettant en question certains points de la phase d'analyse. Il est donc nécessaire de faire une nouvelle phase d'analyse avec le client pour répondre à ces questions et ajuster la conception en conséquence. C'est pourquoi on parle de « cycle de vie » : les phases d'analyse et de conception vont alterner jusqu'à arriver aux spécifications les plus complètes possible.

Une fois la conception finalisée, les spécifications techniques et fonctionnelles sont données aux développeurs, qui vont se charger de réaliser le projet.

Exemple

Pour illustrer ces différentes étapes, prenons un exemple éloigné de l'informatique : la construction d'une maison.

Un client explique à un architecte qu'il souhaite une maison plain-pied avec 3 chambres : c'est le cahier des charges.

En réponse, l'architecte va créer un modèle 3D correspondant à ce cahier des charges : ce sont les spécifications fonctionnelles. Le client pourra ainsi faire des retours sur la disposition des pièces, par exemple, tout en écoutant les conseils de l'architecte lorsque certaines de ces attentes ne sont pas réalisables.

Une fois le modèle 3D stable et validé par le client, l'architecte va en faire un plan technique contenant les mesures, les différents matériaux à utiliser, le sens d'ouverture des portes et tous les angles de vue nécessaires aux ouvriers pour réaliser le projet : ce sont les spécifications techniques.

Il peut arriver, lors de la réalisation des plans, que de nouvelles questions émergent : on peut se rendre compte, par exemple, que certains matériaux sont en rupture de stock, ou que la topologie du terrain nécessite de faire des ajustements au niveau de la taille des cloisons. Il faudra donc refaire une phase d'analyse avec le client pour savoir de quel côté épaissir le mur, puis mettre à jour les plans en conséquence.

Syntaxe **À retenir**

- Avant de commencer un projet informatique, un client arrive avec un cahier des charges décrivant sa vision du projet. Un Product Owner en extrait les fonctionnalités pour faire des spécifications fonctionnelles, qui serviront à l'architecte logiciel pour en déduire les spécifications techniques.

III. L'Unified Modeling Language

Objectifs

- Découvrir l'UML
- Comprendre l'utilité d'un langage de modélisation graphique

Mise en situation

Les spécifications fonctionnelles et techniques sont composées de diagrammes permettant d'expliquer simplement de nombreux aspects du projet. Pour que ces diagrammes soient compris universellement, il a fallu établir une liste de symboles avec leur signification : c'est le but de l'*Unified Modeling Language*, ou UML.

Définition **UML**

UML est un langage de modélisation graphique orienté objet. Contrairement aux langages de programmation, qui définissent des mots-clés qui seront utilisés dans le code d'un programme, l'UML définit des pictogrammes et des diagrammes qui permettent de visualiser tous les aspects d'un projet informatique.

Fondamental **Histoire de l'UML**

Au début des années 1980, le paradigme Orienté Objet se popularise au travers de langages tels que le C++ ou l'Eiffel. Or, si le pseudo-code permet de décrire des algorithmes sans tenir compte du langage, ils ne permettent pas de décrire la complexité d'un objet. En effet, plus que du code, un objet représente une structure de données.

Devant cette problématique, de nombreuses méthodes de modélisation des objets émergent. Toutes ces méthodes reposent sur les mêmes concepts : la représentation graphique des classes, avec leurs propriétés, leurs méthodes et leurs relations. Mais certaines vont même jusqu'à permettre de décrire les interactions entre les différents acteurs d'un système (utilisateurs, base de données, application...) et de délimiter les fonctionnalités d'un projet.

Avec la multiplicité des méthodes (plus d'une cinquantaine dans les années 1990), il a été décidé de créer un standard universel, l'UML, en fusionnant trois langages de modélisation : Booch, OMT (*Object Modeling Technique*) et OOSE (*Object Oriented Software Engineering*). Les prémices de l'UML voient le jour en 1995 et il sera normalisé deux ans plus tard par l'Object Management Group, un consortium dont le rôle est de définir les standards du modèle objet.

Malgré son âge, ce modèle est régulièrement mis à jour, ce qui en fait une référence encore aujourd'hui.

Méthode **Les deux types de diagrammes**

L'UML définit 14 diagrammes séparés en deux types : les diagrammes de structure et les diagrammes de comportement.

Les diagrammes de comportement permettent de définir les fonctionnalités et les interactions dans une application. Ces diagrammes permettent de répondre à la question : « Qu'est-ce qu'on va faire ? ». Ils sont donc surtout utilisés dans les spécifications fonctionnelles.

Les diagrammes de structure permettent de définir les différents composants de l'application : les classes, les packages, les objets... Ces diagrammes permettent de répondre à la question : « Comment est-ce qu'on va faire ? ». Ils sont donc surtout utilisés dans les spécifications techniques.

Syntaxe À retenir

- L'UML est un langage de modélisation graphique orienté objet. Il est la fusion de trois autres langages (Booch, OMT et OOSE) et est aujourd'hui le standard reconnu par l'OMG.
- L'UML définit 14 diagrammes séparés en deux types : les diagrammes de comportement, utilisés dans les spécifications fonctionnelles, et les diagrammes de structure, utilisés dans les spécifications techniques.

Complément

L'UML¹

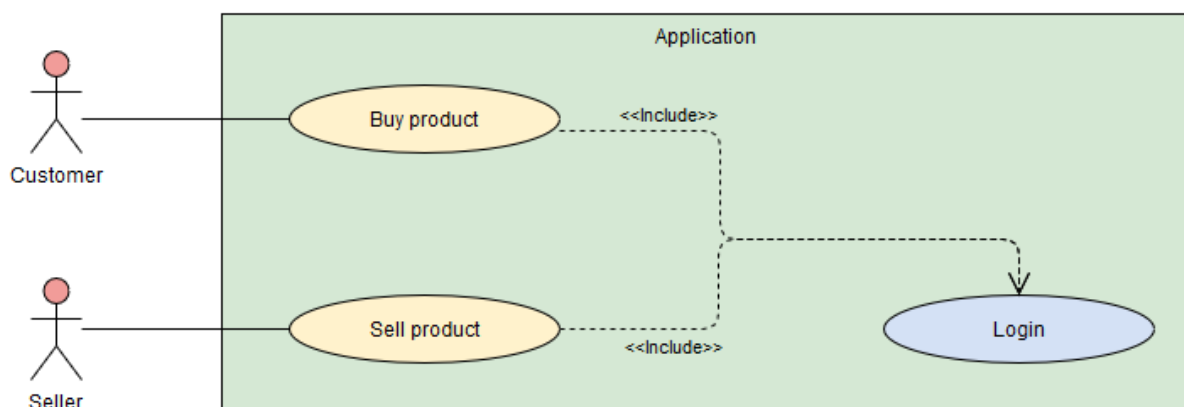
Le site de l'OMG²

IV. Exercice : Appliquez la notion

Question

[solution n°1 p.17]

Les diagrammes définis par le standard UML sont compréhensibles sans forcément connaître toutes les subtilités du langage. Prenons l'exemple d'un des diagrammes de comportement les plus utilisés en UML : le diagramme de cas d'utilisation. Il permet de définir différents types d'utilisateurs et les fonctionnalités auxquelles ceux-ci ont accès. Voici un exemple simpliste de diagramme de cas d'utilisation pour un site de vente en ligne :



Quels sont les différents types d'utilisateurs présents sur l'application, et quelles actions peuvent-ils réaliser ?

V. Les spécifications fonctionnelles

Objectifs

- Comprendre l'utilité des spécifications fonctionnelles
- Découvrir les diagrammes qui composent ces spécifications

¹ https://en.wikipedia.org/wiki/Unified_Modeling_Language

² <https://www.omg.org/>

Mise en situation

Les spécifications fonctionnelles représentent une retranscription des fonctionnalités issues du cahier des charges. Elles doivent contenir l'intégralité des fonctionnalités de l'application, avec tous les cas possibles et les liens entre elles.

Les spécifications fonctionnelles jouent un double rôle : elles permettent de s'assurer que le Product Owner a compris les attentes du client, et fournissent à l'équipe de développement le contexte du projet. Les diagrammes définis par le langage UML ont donc été conçus de manière à pouvoir assurer ces deux rôles : ils sont visuellement assez simples pour être compris par le client, tout en donnant assez d'informations aux développeurs sur les dépendances entre les fonctionnalités et les liens entre différents acteurs.

Dans cette partie, nous allons survoler trois exemples de diagrammes parmi les plus utilisés en UML : le diagramme de cas d'utilisation, le diagramme de séquence et le diagramme d'activité.

Méthode Le diagramme de cas d'utilisation

Le diagramme de cas d'utilisation permet de représenter toutes les interactions possibles dans l'application. Pour cela, il met en valeur deux éléments :

- Les **acteurs**, qui désignent quelqu'un, ou quelque chose, qui va interagir avec notre système. La plupart du temps, les acteurs seront les différents utilisateurs, mais dans certains cas, ils peuvent symboliser un programme externe (comme une tâche automatisée). Ils sont représentés par des bonhommes bâtons et un nom.
- Les **cas d'utilisation**, c'est-à-dire une fonctionnalité générale de l'application. Ils sont représentés par des ellipses contenant le nom de la fonctionnalité.

Ces éléments peuvent être liés entre eux pour définir quel acteur a accès à quelle fonctionnalité, et quels cas d'utilisation sont dépendants les uns des autres.

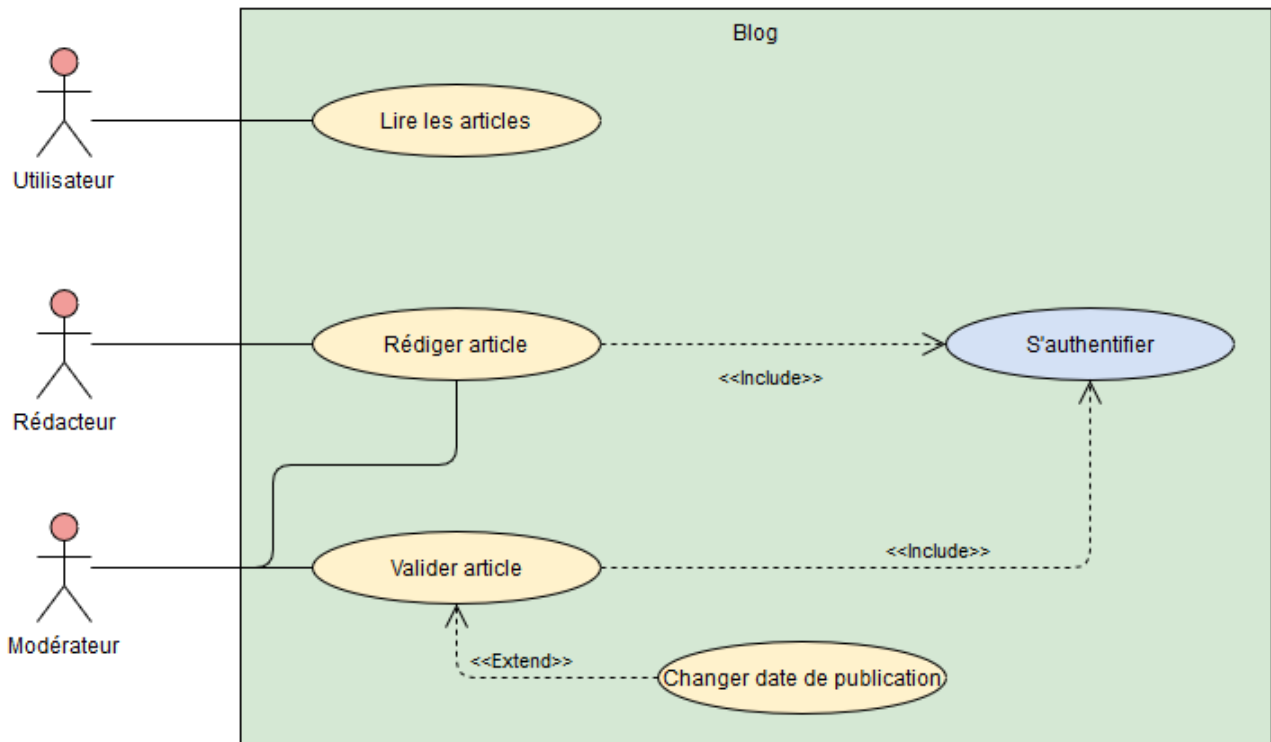
Exemple Cahier des charges pour un blog communautaire

Pour gérer un blog, il doit y avoir trois types d'utilisateurs : les rédacteurs, qui rédigent des articles ; les modérateurs, qui valident les articles, mais peuvent également en écrire ; et les utilisateurs normaux, qui peuvent simplement lire les articles.

Au moment de la validation d'un article, un modérateur peut choisir une date de publication s'il souhaite qu'il n'apparaisse pas immédiatement sur le blog.

Les rédacteurs et les modérateurs doivent s'authentifier pour pouvoir accéder à leurs fonctionnalités.

Voici le diagramme de cas d'utilisation représentant ce cahier des charges :



Le diagramme permet ainsi d'exprimer de manière visuelle toutes les fonctionnalités du cahier des charges.

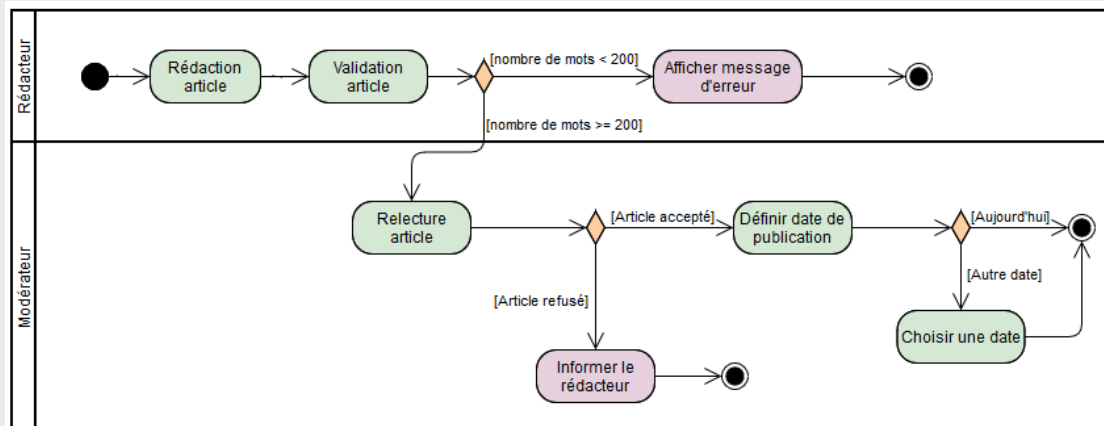
Méthode Diagramme d'activité

Le diagramme d'activité permet de retracer toutes les étapes nécessaires à la complétion d'une fonctionnalité. Il permet de définir le *workflow* de l'application et met en lumière tous les cas possibles auxquels les utilisateurs pourraient être confrontés.

Un diagramme d'activité possède un nœud initial, qui est le point d'entrée du process, et un ou plusieurs nœuds de fin d'activité. Les deux sont reliés par une succession d'actions et de flèches indiquant le sens du *workflow*. Il est possible de créer des conditions dans un *workflow* grâce aux décisions, symbolisées par un losange.

Exemple

Voici le diagramme d'activité représentant la vie d'un article, de son écriture à sa validation :

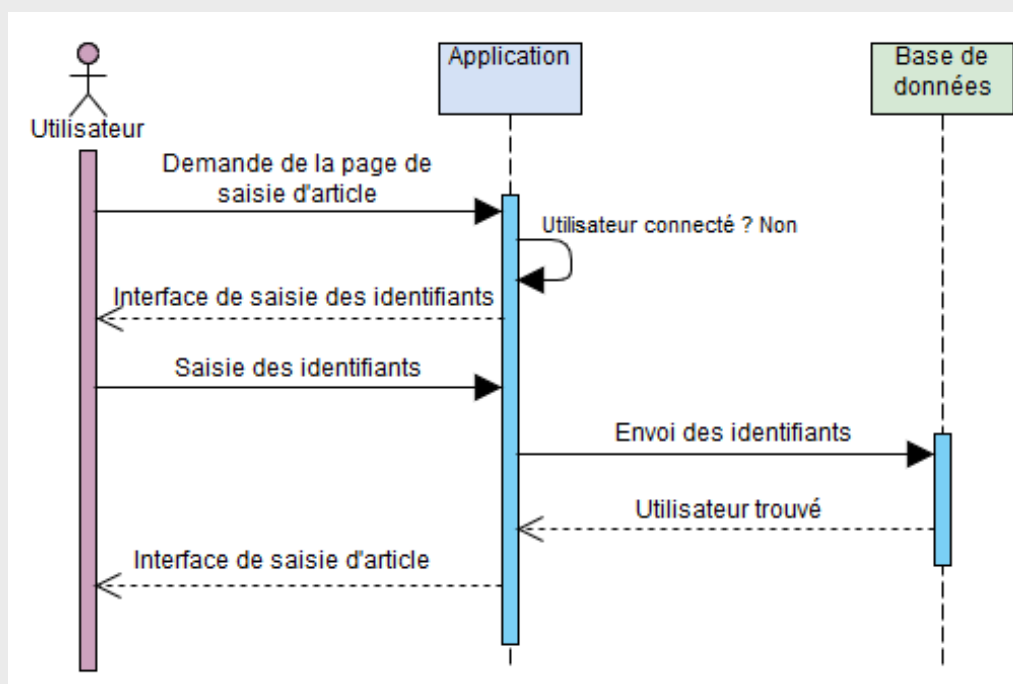
**Méthode** Diagramme de séquence

Le diagramme de séquence met en valeur les communications entre les différents éléments constitutifs de l'application. Il permet d'établir une chronologie dans les envois de messages entre les acteurs et les objets, c'est-à-dire les différents systèmes qui vont échanger des données pendant le processus.

Un diagramme de séquence décrit les échanges pour un chemin possible (ou une partie de chemin) du diagramme d'activité. Pour une phase d'authentification d'un utilisateur, par exemple, il va falloir un diagramme pour une authentification réussie et un autre lors d'un échec.

Exemple

Voici le diagramme de séquence lorsqu'un utilisateur non connecté demande l'affichage de la page permettant de rédiger un article sur le blog. L'utilisateur est un acteur et deux objets doivent communiquer ensemble : l'application et la base de données.



Remarque

Bien que ces diagrammes soient des diagrammes de comportement, utilisés pour faire les spécifications fonctionnelles, ils peuvent également être complétés par des éléments techniques. Dans les exemples ci-dessus, on peut le voir dans le diagramme de séquence avec la présence de la base de données.

Syntaxe À retenir

- Les spécifications fonctionnelles permettent de détailler toutes les fonctionnalités d'une application. Pour cela, l'UML dispose de différents diagrammes, dont le but est d'être compréhensibles à la fois par le client qui va les valider, et par l'équipe de développement.
- Les trois diagrammes les plus utilisés sont les diagrammes de cas d'utilisation, d'activité et de séquence.

Complément

Diagramme de cas d'utilisation¹

Diagramme de séquence²

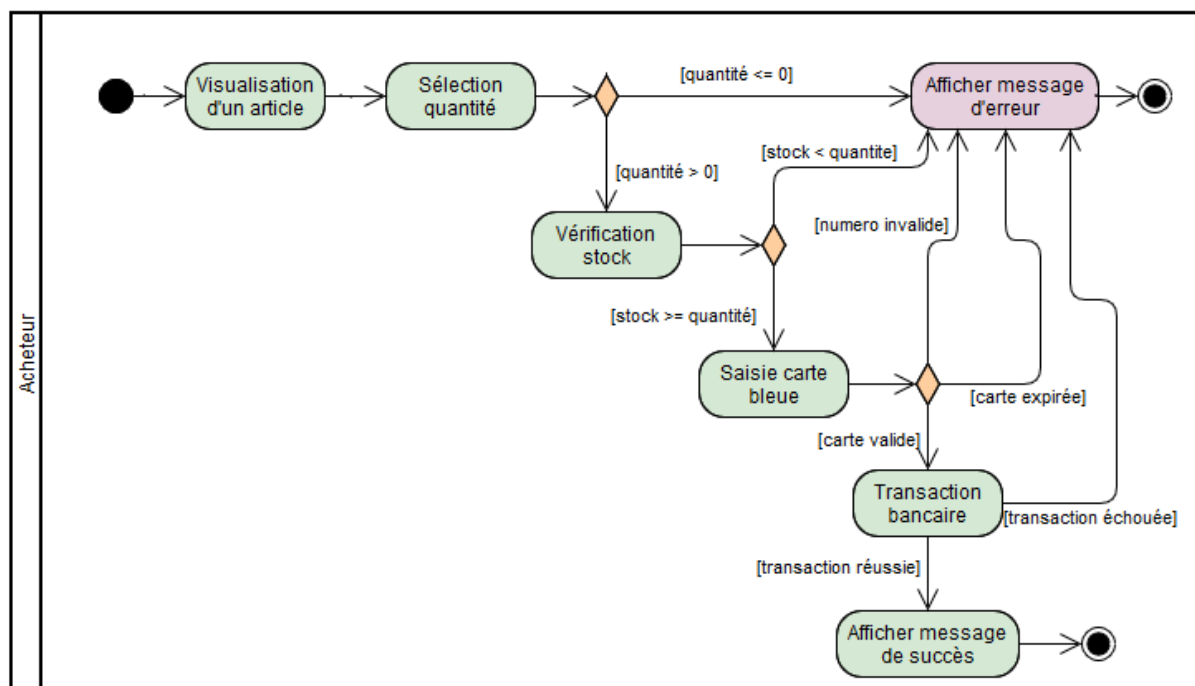
Diagramme d'activité³

VI. Exercice : Appliquez la notion

Question 1

[solution n°2 p.17]

Voici un diagramme trouvé dans les spécifications fonctionnelles d'une application de vente en ligne :



Quel type de diagramme est-ce ?

1 https://fr.wikipedia.org/wiki/Diagramme_de_cas_d%27utilisation

2 https://fr.wikipedia.org/wiki/Diagramme_de_s%C3%A9quence

3 https://fr.wikipedia.org/wiki/Diagramme_d%27activit%C3%A9

Question 2

[solution n°3 p.17]

D'après ce diagramme, que se passe-t-il si le client essaye d'acheter exactement la quantité demandée en stock ?

Question 3

[solution n°4 p.17]

Dans quels cas est-ce qu'un message d'erreur est retourné à l'utilisateur ?

VII. Les spécifications techniques**Objectifs**

- Comprendre l'utilité des spécifications techniques
- Découvrir le diagramme principal qui compose ces spécifications

Mise en situation

Les spécifications techniques permettent à l'équipe d'avancer le développement dans la même direction en s'assurant que tout le monde va utiliser la même structure. Elles sont issues des spécifications fonctionnelles et permettent d'établir un découpage du code en packages, en composants, en classes ou en objets.

Bien qu'il existe plusieurs diagrammes de structure permettant différents niveaux de précision, le diagramme le plus utilisé – et également le plus iconique de l'UML – est le diagramme de classes. C'est celui que nous allons voir dans cette partie.

Méthode **Le diagramme de classes**

Le diagramme de classes permet de représenter les différentes structures de données de l'application et les relations qu'il y a entre elles. Ce diagramme est le point central de l'UML : il représente toutes les classes de notre application et leur hiérarchie.

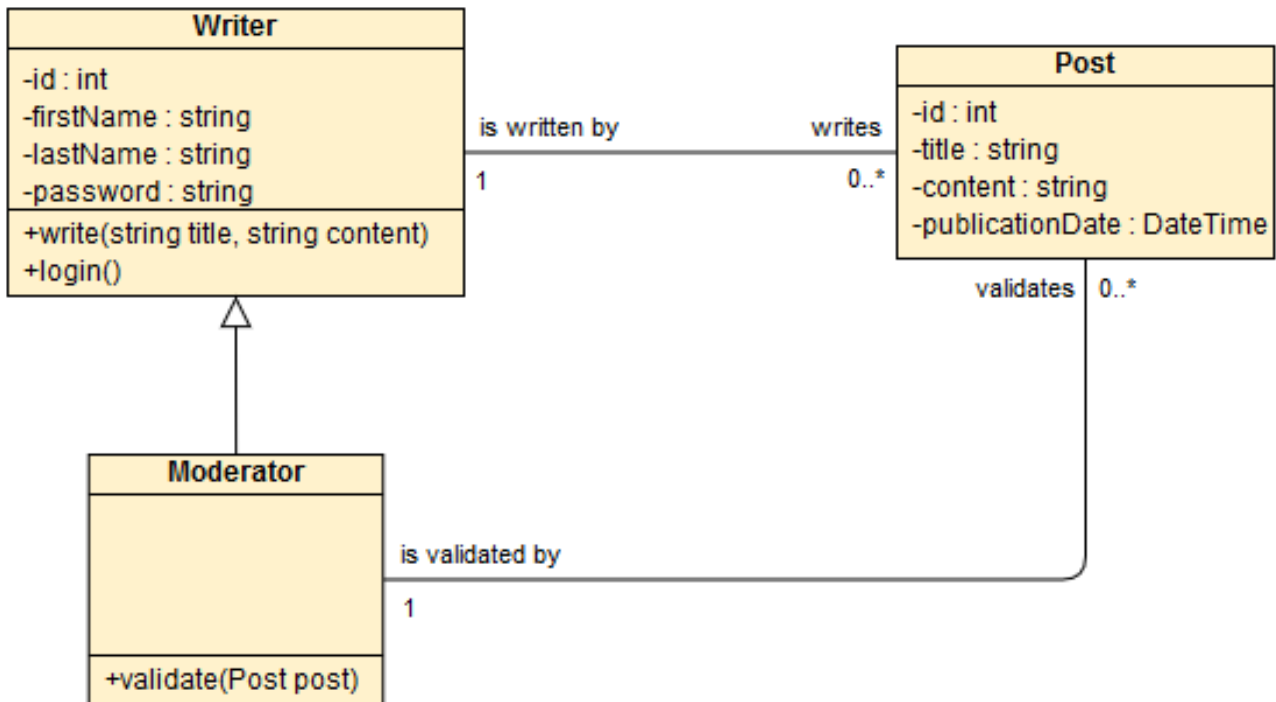
Le diagramme de classes sert non seulement à proposer un découpage uniforme du code, mais va également permettre de définir le modèle relationnel du projet, c'est-à-dire les différentes tables de notre base de données, leurs colonnes et leurs contraintes.

Un diagramme de classes est composé de classes, dont les propriétés et les méthodes sont stockées dans des rectangles, et des associations décrivant les relations entre les classes. Chaque côté d'une association possède des cardinalités, c'est-à-dire le nombre minimal et maximal d'instances de l'objet impliquées dans l'association. Par exemple, en décrivant une classe `Employé` et une classe `Entreprise`, on peut dire qu'un employé appartient à une seule entreprise, mais qu'une entreprise possède plusieurs employés (au moins 1). La cardinalité de la classe `Employé` vers `Entreprise` est donc **1**, et la cardinalité de la classe `Entreprise` vers `Employé` est notée **1..***, qui signifie « plusieurs, avec un minimum de 1 ».

Il est également possible de symboliser un héritage de classes à l'aide d'une flèche.

Exemple

Voici le diagramme de classes de notre application de blog :



Les associations entre un rédacteur et un article sont : « Un rédacteur écrit plusieurs articles (mais ce nombre peut être nul) et un article est écrit par un rédacteur ».

Remarque

Si tous les diagrammes des spécifications fonctionnelles étaient rédigés en français, il est à noter que le diagramme des classes respecte les bonnes pratiques du code (formatage des noms de classes, de propriétés et de méthodes, langue anglaise utilisée). En effet, le but des spécifications techniques est d'être au plus proche du code final : les noms utilisés dans ce diagramme seront ceux utilisés par l'équipe de développement.

Syntaxe À retenir

- Les spécifications techniques permettent de définir le découpage du code de l'application et de définir la structure des données qu'elle utilise. Le diagramme le plus important des spécifications techniques est le diagramme de classes.

Complément

Diagramme de classes¹

¹ https://fr.wikipedia.org/wiki/Diagramme_de_classes

VIII. Exercice : Appliquez la notion

Question

[solution n°5 p.17]

Avec ou sans l'aide de vos cours, réalisez un diagramme de séquence pour la situation suivante.

Un bibliothécaire voudrait une application pour que les utilisateurs ayant un compte puissent réserver des livres depuis l'application directement. La réservation n'est possible que si le livre est disponible.

Scénario d'utilisation :

- Un utilisateur non-connecté veut effectuer une recherche de livre sur l'application. L'application renvoie alors l'interface de saisie des identifiants de connexion.
- Une fois connecté, l'utilisateur peut effectuer une recherche de livre qui sera directement soumise à la Base De Données.
- La base de données renvoie si le livre est disponible ou non (dans notre cas, le livre sera disponible).
- L'utilisateur peut effectuer la réservation directement sur l'application.
- L'application renvoie une confirmation de réservation.

IX. Essentiel

X. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°6 p.17]

Exercice

Quel est le document initial permettant de lancer un projet informatique ?

- ☐ Les spécifications fonctionnelles
- ☐ Les spécifications techniques
- ☐ Le cahier des charges

Exercice

Quel membre de l'équipe connaît le mieux la partie métier d'un projet, et donc est le plus proche du client ?

- ☐ Le développeur
- ☐ Le Product Owner
- ☐ L'architecte logiciel

Exercice

À quoi servent les spécifications fonctionnelles ?

- ☐ À confirmer que l'équipe a bien compris le besoin du client
- ☐ À lister l'intégralité des fonctionnalités attendues par le client
- ☐ À s'assurer que les développeurs vont suivre la même logique de code
- ☐ À indiquer le contexte général du projet
- ☐ À définir le modèle relationnel

Exercice

Qu'est-ce que l'UML ?

- ☐ Un diagramme indispensable aux spécifications techniques
- ☐ Un langage de programmation
- ☐ Un ensemble de symboles permettant de faire des diagrammes

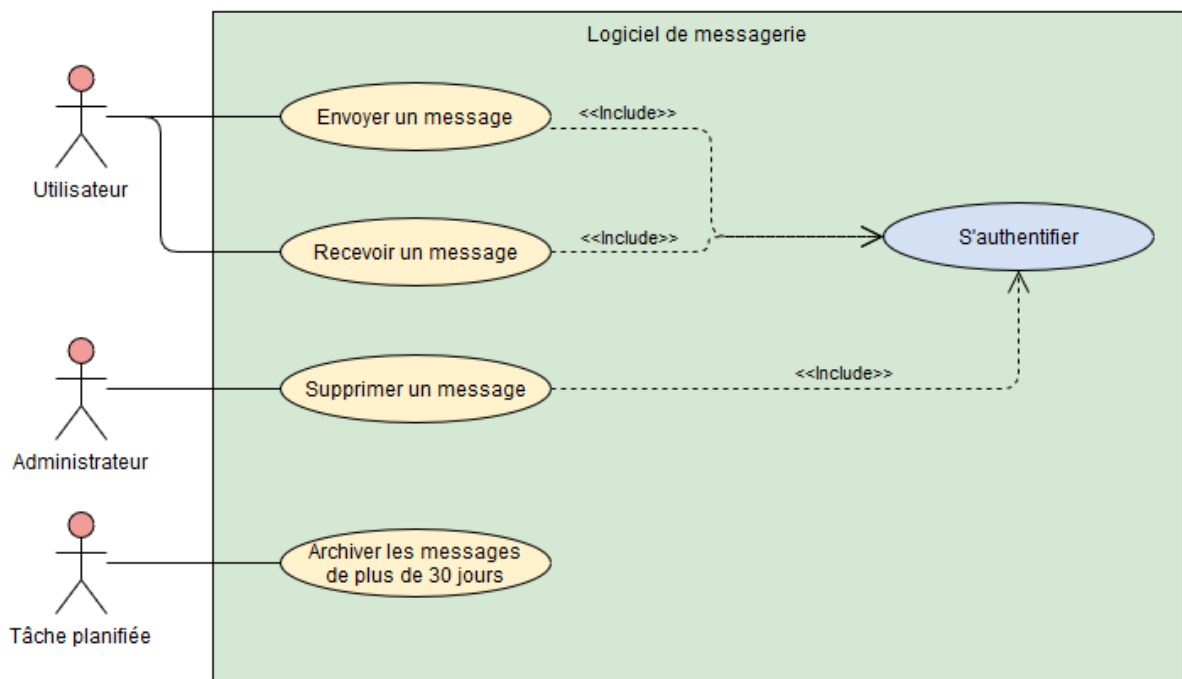
Exercice

Quel diagramme permet de définir le *workflow* d'une fonctionnalité et d'en déterminer les étapes nécessaires à sa réalisation ?

- ☐ Le diagramme de cas d'utilisation
- ☐ Le diagramme d'activité
- ☐ Le diagramme de séquences
- ☐ Le diagramme de classes

Exercice

Quel est le nom de ce diagramme ?



- ☐ Le diagramme de cas d'utilisation
- ☐ Le diagramme d'activité
- ☐ Le diagramme de séquences
- ☐ Le diagramme de classes

Exercice

Dans le diagramme précédent, l'administrateur peut recevoir des messages.

- ☐ Vrai
- ☐ Faux

Exercice

Dans le diagramme précédent, qui doit s'authentifier pour avoir accès à ses fonctionnalités ?

- ☐ L'utilisateur
- ☐ L'administrateur
- ☐ La tâche planifiée

Exercice

Quel diagramme est le diagramme principal des spécifications techniques ?

- ☐ Le diagramme de cas d'utilisation
- ☐ Le diagramme d'activité
- ☐ Le diagramme de séquences
- ☐ Le diagramme de classes

Exercice

Le diagramme de classes permet de définir...

- ☐ La structure du code
- ☐ Le framework à utiliser
- ☐ Le modèle relationnel
- ☐ La procédure de déploiement
- ☐ Les bibliothèques à importer

B. Exercice : Défi

Pour ce défi, vous allez devoir vous aider des exemples du cours pour réaliser les spécifications fonctionnelles et techniques d'un projet de gestion de rencontres sportives, à partir du cahier des charges fourni.

Question 1

[solution n°7 p.20]

Une association de volley-ball souhaiterait pouvoir gérer les rencontres qu'elle organise entre les différentes équipes de la région. Une équipe est composée de joueurs, chacun possédant un identifiant, un nom, un prénom et un nom d'équipe. Un des joueurs est désigné comme étant capitaine d'équipe. Pour pouvoir participer à un match, le capitaine doit inscrire son équipe. Une fois cette étape réalisée, chaque joueur, capitaine inclus, doit confirmer sa présence au match.

Il est important de noter que l'association possède déjà un outil pour gérer les inscriptions et les matches. Son besoin porte uniquement sur la gestion des droits des joueurs.

Réalisez le diagramme de cas d'utilisation correspondant à ce cahier des charges.

Indice :

Il y a deux acteurs : les joueurs et les capitaines.

Il y a deux fonctionnalités : « inscrire l'équipe » et « confirmer sa présence au match », mais tous les acteurs n'y ont pas accès.

Question 2

[solution n°8 p.21]

Réalisez le diagramme des classes correspondant aux spécifications fonctionnelles.

Indice :

La gestion des droits pour l'inscription et la confirmation des joueurs sont deux méthodes qui doivent apparaître dans le diagramme.

Solutions des exercices

p. 6 Solution n°1

Il y a deux types d'utilisateurs sur l'application : les vendeurs (*Seller*) et les acheteurs (*Customer*). Les vendeurs peuvent vendre des produits et les acheteurs peuvent les acheter. On remarque que, pour pouvoir réaliser leurs actions respectives, les deux doivent se connecter (*login*).

Bien entendu, un vrai diagramme de cas d'utilisation sera certainement plus complexe et inclura l'inscription des utilisateurs, la gestion des stocks (avec leur mise à jour selon les actions de chacun), le traitement des erreurs et toutes les autres fonctionnalités de notre application.

p. 10 Solution n°2

C'est un diagramme d'activité. On le reconnaît notamment à son nœud initial et ses nœuds de fin d'activité.

p. 11 Solution n°3

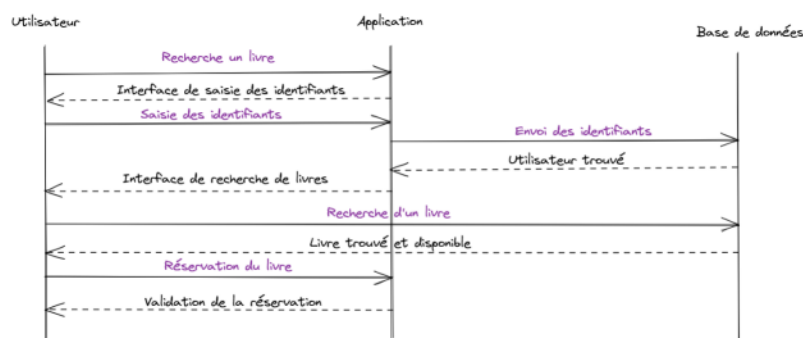
Si le client essaye d'acheter exactement la quantité demandée en stock, alors on passe dans la condition `[stock >= quantité]`. La prochaine étape est donc de demander à l'utilisateur son numéro de carte bleue.

p. 11 Solution n°4

Un message d'erreur apparaît si :

- L'utilisateur choisit une quantité de produits invalide (0 ou un nombre négatif)
- Si le produit n'est pas en stock
- Si le numéro de sa carte bleue n'est pas valide
- Si sa carte a expiré
- Si la transaction bancaire a échoué

p. 13 Solution n°5




Si vous n'avez pas les mêmes termes que cette solution mais que l'idée y est, ne considérez pas que vous avez faux, bien au contraire. Si cette partie est encore un peu floue, n'hésitez pas à relire le cours et à prendre le temps d'analyser les schémas.

Exercice p. 13 Solution n°6

Exercice

Quel est le document initial permettant de lancer un projet informatique ?


- ☐ Les spécifications fonctionnelles
- ☐ Les spécifications techniques
- ☒ Le cahier des charges

 Le cahier des charges est le document permettant d'établir les spécifications fonctionnelles, puis techniques, d'un projet, avant de se lancer dans sa réalisation. C'est donc le document initial, indispensable au lancement d'un projet.

Exercice

Quel membre de l'équipe connaît le mieux la partie métier d'un projet, et donc est le plus proche du client ?


- ☐ Le développeur
- ☒ Le Product Owner
- ☐ L'architecte logiciel

 Le Product Owner est là pour comprendre les besoins du client et les remonter à l'équipe : c'est lui qui rédige les spécifications fonctionnelles, et il peut même aider à l'écriture du cahier des charges. Pour cela, une connaissance du métier du client est indispensable.

Exercice

À quoi servent les spécifications fonctionnelles ?


- ☒ À confirmer que l'équipe a bien compris le besoin du client
- ☒ À lister l'intégralité des fonctionnalités attendues par le client
- ☐ À s'assurer que les développeurs vont suivre la même logique de code
- ☒ À indiquer le contexte général du projet
- ☐ À définir le modèle relationnel

 Les spécifications fonctionnelles permettent de retranscrire l'intégralité des besoins du client et doivent être validées par ce dernier avant d'en déduire les spécifications techniques, ce qui permet de s'assurer que l'application qui va être développée est bien conforme à ses attentes. Elles servent également à introduire le contexte du projet aux membres de l'équipe.

Exercice

Qu'est-ce que l'UML ?

- ☐ Un diagramme indispensable aux spécifications techniques
- ☐ Un langage de programmation
- ☒ Un ensemble de symboles permettant de faire des diagrammes

 L'UML est un langage de modélisation graphique : il fournit un ensemble de pictogrammes ayant un sens particulier, qui permettent de réaliser des diagrammes.

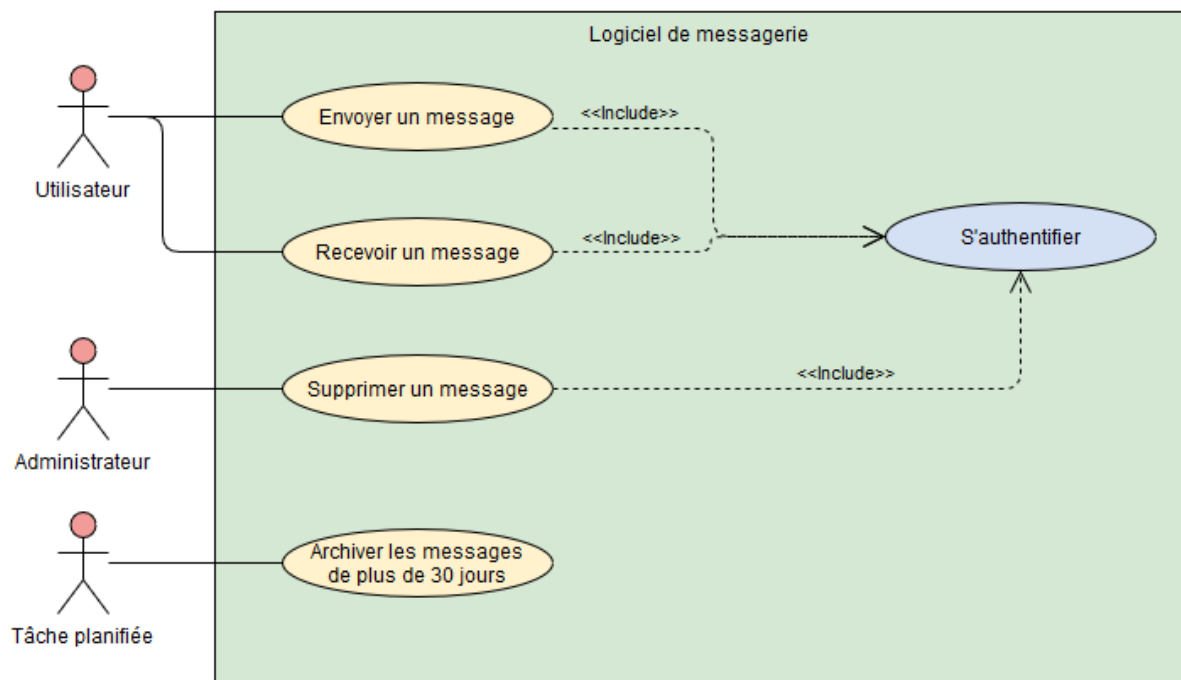
Exercice

Quel diagramme permet de définir le *workflow* d'une fonctionnalité et d'en déterminer les étapes nécessaires à sa réalisation ?

- ☐ Le diagramme de cas d'utilisation
- ☒ Le diagramme d'activité
- ☐ Le diagramme de séquences
- ☐ Le diagramme de classes
- ☐ Le diagramme d'activité permet de découper une fonctionnalité en étapes successives.

Exercice

Quel est le nom de ce diagramme ?



- ☒ Le diagramme de cas d'utilisation
- ☐ Le diagramme d'activité
- ☐ Le diagramme de séquences
- ☐ Le diagramme de classes
- ☐ Ce diagramme est un diagramme de cas d'utilisation.


Exercice

Dans le diagramme précédent, l'administrateur peut recevoir des messages.

- ☐ Vrai
- ☒ Faux
- ☐ Il n'y a aucun lien entre l'acteur **Administrateur** et la fonctionnalité « Recevoir un message ». Il ne faut donc pas que l'administrateur puisse y avoir accès.


Exercice

Dans le diagramme précédent, qui doit s'authentifier pour avoir accès à ses fonctionnalités ?

- ☒ L'utilisateur
- ☒ L'administrateur
- ☐ La tâche planifiée
-  Seule la tâche planifiée ne doit pas s'authentifier pour avoir accès à ses fonctionnalités


Exercice

Quel diagramme est le diagramme principal des spécifications techniques ?

- ☐ Le diagramme de cas d'utilisation
- ☐ Le diagramme d'activité
- ☐ Le diagramme de séquences
- ☒ Le diagramme de classes
-  Le diagramme de classes est l'un des diagrammes les plus importants de l'UML et représente une information vitale pour les spécifications techniques.

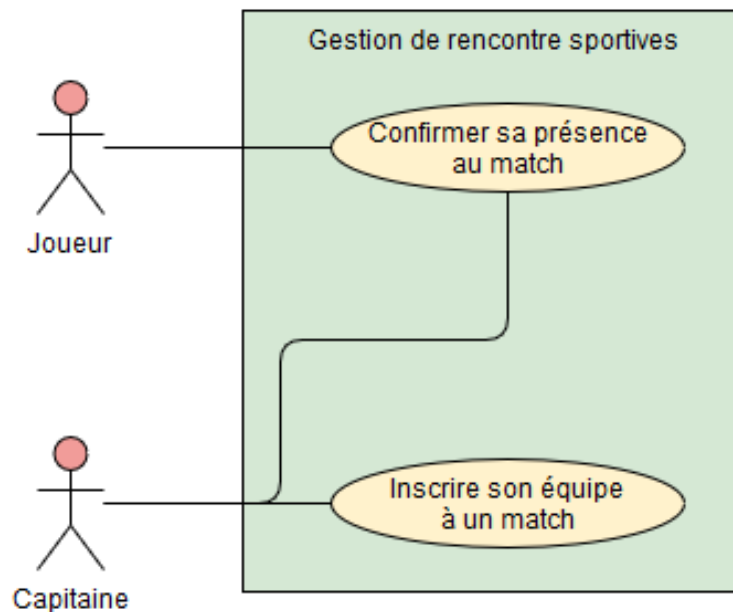
Exercice

Le diagramme de classes permet de définir...

- ☒ La structure du code
- ☐ Le framework à utiliser
- ☒ Le modèle relationnel
- ☐ La procédure de déploiement
- ☐ Les bibliothèques à importer
-  Le diagramme de classes permet de définir la structure des classes du projet, donc le découpage du code. Il est possible d'en déduire le modèle relationnel à utiliser.

p. 15 **Solution n°7**

Il y a deux acteurs dans l'application : les joueurs et les capitaines. Ces deux acteurs ont accès à la fonctionnalité « confirmer sa présence au match », mais seuls les capitaines peuvent inscrire leur équipe. Cela donne le diagramme de cas d'utilisation suivant :



À noter qu'il n'a jamais été fait mention d'un système d'authentification des utilisateurs. On peut imaginer que notre application sera un module relié à leur système de gestion des matches déjà existant, et qu'il s'occupera de cette partie.

p. 16 Solution n°8

Il existe énormément de conceptions différentes possibles pour ce cahier des charges. Voici une solution simpliste, se contentant d'une classe `User`, qui va gérer à la fois les joueurs et les capitaines grâce à un booléen `isCaptain`. Deux méthodes `canRegisterTeam` et `canConfirmPresence` retournent des booléens qui permettront de gérer les droits d'accès aux fonctionnalités.

User
-id : int -firstName : string -lastName : string -isCaptain : bool -teamName : string
+canRegisterTeam() : bool +canConfirmPresence() : bool

Attention : si vous n'avez pas la même réponse, cela ne signifie pas forcément qu'elle est fausse. Il existe d'autres possibilités : par exemple, avoir une classe mère pour les joueurs et une classe fille pour les capitaines. On peut même imaginer une classe à part pour les équipes.

Chaque développeur va avoir des idées de structures différentes pour réaliser ce projet. C'est tout l'intérêt des spécifications techniques : imposer une structure finale avant de commencer le développement, pour s'assurer que tout le monde avance dans la même direction.