Cours Génie Logiciel (partie 2)

EILCO-ING2

ABAKARIM Fadwa

Le modèle en spirale

Le modèle en spirale est une approche de développement logiciel qui combine des éléments de la méthodologie en cascade et de l’approche itérative.

L’approche itérative est le développement progressif du produit avec des améliorations régulières, à chaque étape (itération).

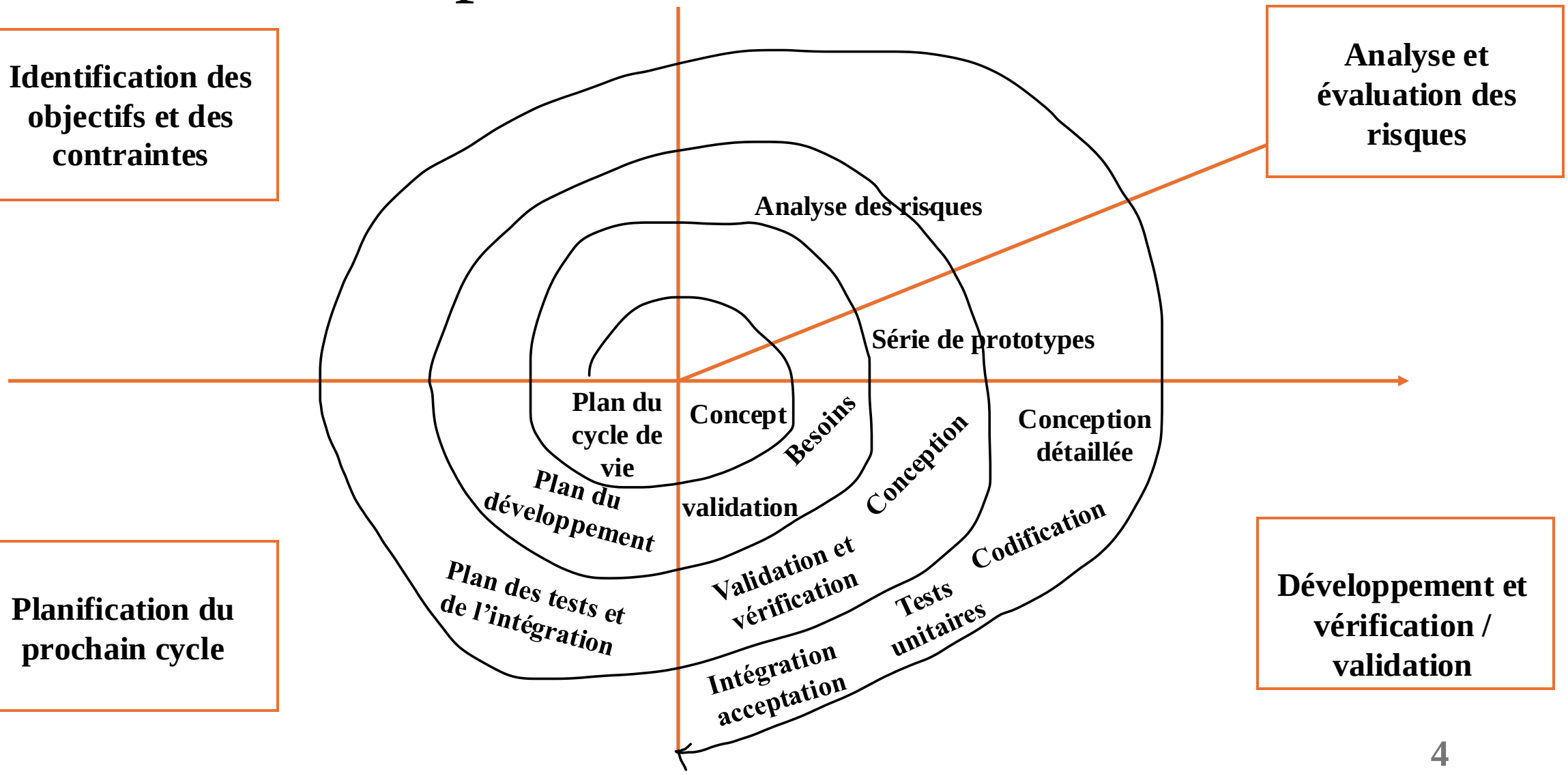
Le modèle en spirale est structuré en quatre phases principales répétées en cycles (ou spirales) jusqu'à l'achèvement du projet.

Le modèle en spirale

Les phases :

- **Identification des objectifs et des contraintes** : définir les objectifs du projet, identifier les alternatives et les contraintes pour chaque cycle.
- **Analyse et évaluation des risques** : identifier et analyser les risques potentiels, et proposer des solutions pour les minimiser.
- **Développement et validation** : développement initial des fonctionnalités et réalisation des tests et validations.
- **Planification du prochain cycle** : évaluation du progrès réalisé, ajustements si nécessaires et planification de la prochaine spirale.

Le modèle en spirale



Le modèle en spirale

Nombre de cycles :

- Le nombre de cycles dépend de la durée du projet, de l'évolution des exigences, des risques identifiés et des retours des utilisateurs.

- Il n'y a pas de limite fixe, l'objectif est de continuer les cycles tant que les risques sont présents et que des ajustements sont nécessaires avant la livraison finale du produit.

Le modèle en spirale

Avantages :

- **Gestion des risques** : Permet d'identifier et de gérer les risques dès le début, ce qui aide à éviter des erreurs coûteuses en les testant tôt.

- **Flexibilité** : Le modèle est adaptable, permettant de changer le projet en fonction des nouveaux besoins ou des retours au fur et à mesure du développement.

- **Prototypes pour tester les idées** : Avant de commencer à tout développer, on peut créer un prototype (une version simple du produit) pour tester certaines fonctionnalités. Cela permet de valider si l’idée fonctionne réellement, avant d’aller plus loin.

Le modèle en spirale

Inconvénients

- **Coût élevé** : En raison de son approche itérative et de la gestion des risques, le modèle peut entraîner des coûts supplémentaires par rapport à d'autres modèles plus linéaires.

- **Dépendance à l’expertise en gestion des risques** : Le succès du modèle dépend fortement de la capacité à identifier et gérer efficacement les risques. Si cette gestion est mal réalisée, cela peut affecter le projet.

- **Difficile à appliquer à de petits projets** : Pour les projets simples, l’approche spirale peut sembler trop lourde et coûteuse par rapport à des méthodes plus simples comme le modèle en V ou en cascade.

Le modèle évolutif

Le modèle évolutif est une méthode de développement logiciel où le produit est créé petit à petit, version après version.

Chaque version ajoute de nouvelles fonctionnalités ou améliore celles qui existent déjà, en fonction des retours des utilisateurs.

Le modèle évolutif

Les phases :

- **Définir les exigences de base** : Identifier les fonctionnalités minimales nécessaires pour démarrer.
- **Concevoir l'architecture** : Créer une base solide pour construire les futures fonctionnalités.
- **Développer un incrément** : Ajouter une fonctionnalité ou une amélioration au produit.
- **Intégrer et tester** : Vérifier que l’incrément fonctionne bien avec les éléments déjà développés.
- **Collecter des retours** : Recevoir des avis pour affiner le produit.
- **Planifier l'incrément suivant** : Préparer le prochain cycle d'améliorations.
- **Répéter les cycles** jusqu'à l'obtention du produit final.

Le modèle évolutif

Exemple d’application de gestion de tâches :

- **Définir les exigences de base** : L'équipe identifie les fonctionnalités minimales pour une première version, comme la possibilité d'ajouter, de modifier et de supprimer des tâches.
- **Concevoir l'architecture** : Ils créent une structure de base de l'application, avec une interface simple, une base de données pour stocker les tâches, et une logique pour les opérations de base (ajouter, modifier, supprimer).
- **Développer le premier incrément** : La première version de l'application permet uniquement aux utilisateurs de créer et de gérer des tâches simples. L'équipe teste cette version de base avec quelques utilisateurs.
- **Intégrer et tester** : Ils vérifient que l'application fonctionne sans bugs et que la gestion des tâches se fait correctement.

Le modèle évolutif

Exemple d’application de gestion de tâches (suite) :

- **Collecter des retours** : Les utilisateurs indiquent qu'ils aimeraient pouvoir organiser les tâches par date et par priorité.
- **Planifier l'incrément suivant** : L'équipe décide de développer des options pour trier les tâches par date et ajouter une priorité.
- **Répéter les cycles** :
  - **Incrément 2** : Ajoute la possibilité de trier les tâches par date et d'ajouter une priorité.
  - **Incrément 3** : Ajoute des notifications pour rappeler les tâches à faire.
  - **Incrément 4** : Introduit le partage de tâches avec d'autres utilisateurs.
- **Produit final** : Après plusieurs cycles, l'application est devenue complète, avec une gestion avancée des tâches, des priorités, des rappels et le partage.

Le modèle évolutif

Avantages :

- **Flexible** : Permet de modifier les exigences en cours de projet.
- **Retour continu** : Les utilisateurs peuvent tester et donner leur avis après chaque incrément.
- **Réduit les risques** : Les erreurs sont corrigées plus tôt grâce aux tests réguliers.

Le modèle évolutif

Inconvénients

- **Difficile à estimer** : Le coût et la durée sont difficiles à prévoir.
- **Risque de dérive** : Le projet peut s'étendre avec des ajouts imprévus.
- **Gestion complexe** : Exige une organisation stricte pour bien planifier les itérations.

Question

Expliquez la différence entre le modèle en spirale et le modèle évolutif.

Problématique

En suivant ces cycles de vie, on passe souvent beaucoup de temps à planifier et à organiser les étapes avant de commencer réellement à développer le logiciel.

Cela signifie qu'on se concentre beaucoup sur "comment" on va construire le système, parfois même plus que sur le développement lui-même.

Pour cette raison, les **méthodes agiles** ont été créées :

- Se concentrent sur le développement : elles mettent l'accent sur la programmation, un aspect apprécié des ingénieurs.

- Reposent sur une approche itérative : elles avancent par étapes courtes et répétitives, permettant des améliorations constantes.

- Visent à livrer rapidement un logiciel exécutable : les clients peuvent tester le produit, donner leur avis et demander des ajustements.

Méthodes agiles

Les méthodes agiles comprennent plusieurs modèles ou frameworks populaires. Voici les principaux :

- Extreme Programming (XP) / Programmation Extrême.
- Feature-Driven Development (FDD) / Développement Dirigé par les Fonctionnalités.

Méthodes agiles

L'Extreme Programming (XP), Programmation Extrême en français est une méthode agile de développement logiciel qui privilégie la collaboration étroite avec le client, la livraison rapide de fonctionnalités et une amélioration continue du code.

Elle repose sur des itérations courtes et une grande flexibilité, avec un focus sur la qualité du code grâce à des pratiques comme les tests automatisés et la programmation en binôme.

Méthodes agiles

Feature Driven Development (FDD), Développement Dirigé par les Fonctionnalités est une méthode de gestion de projet qui se concentre sur la gestion des risques et le développement par fonctionnalités.

Les projets sont organisés en itérations courtes autour de fonctionnalités testables, avec une forte implication du client tout au long du processus.

L’objectif est de développer des fonctionnalités en réponse aux besoins spécifiques de l'utilisateur.

Méthodes agiles

Critère principal	XP	FDD
Objectif principal	Fournir un logiciel de haute qualité avec une adaptation rapide aux besoins du client.	Développer un logiciel fonctionnel rapidement et de manière prévisible en se concentrant sur les fonctionnalités.
Approche de développement	Itérations courtes avec un focus sur la collaboration, la flexibilité et l'amélioration continue du code.	Développement basé sur des itérations centrées sur des fonctionnalités spécifiques à livrer.
Méthode de programmation	Pratiques comme la programmation en binôme, les tests automatisés, et la refactorisation continue.	Aucune méthode spécifique de programmation, mais un focus sur la conception de fonctionnalités avant leur développement.
Livraison	Livraison fréquente de petites fonctionnalités après chaque itération.	Livraison de fonctionnalités complètes après chaque itération.

Question

Quelle méthode est la plus adaptée aux grands projets : FDD ou XP ? Et quelle méthode conviendrait mieux aux petits projets ? Justifiez votre réponse.

Méthode	Adaptée à	Pourquoi
FDD		
XP		

La conception

La conception en génie logiciel est la phase du développement d'un logiciel où l'on planifie et structure le système à construire.

Elle se situe entre l’analyse des besoins (ou exigences) et la phase de développement (ou codage) proprement dite.

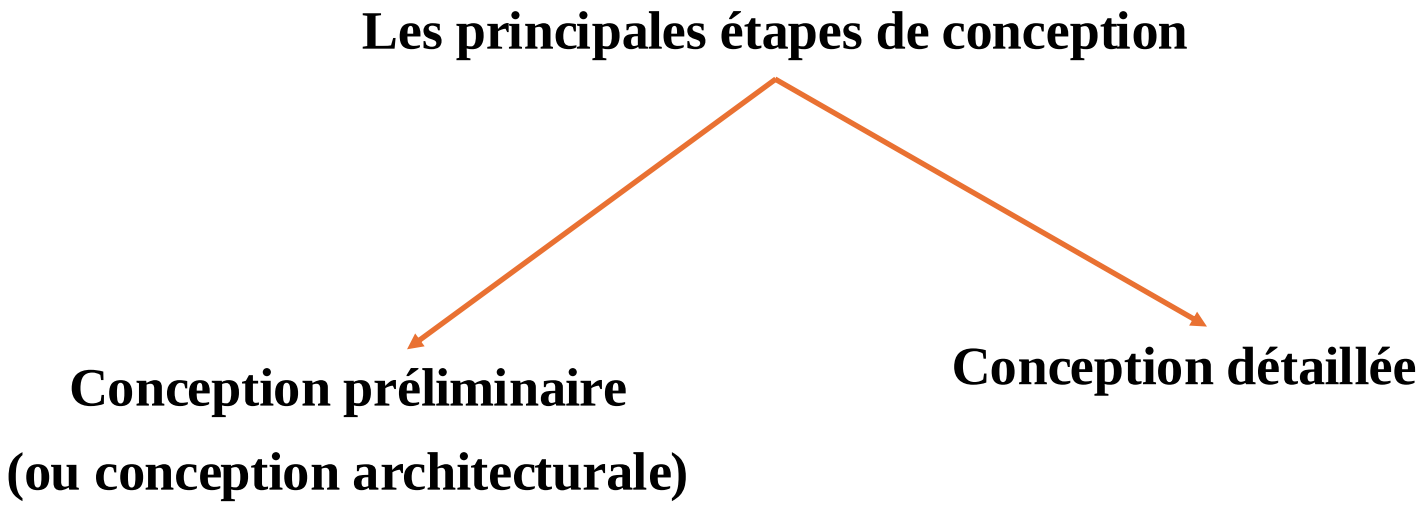
La conception

Objectif :

- L’objectif principal de la conception est de créer une architecture et une structure détaillée du logiciel qui permettent de répondre aux exigences définies lors de l’analyse.

La conception décrit comment le système va fonctionner, en termes d’architecture, de modules, d’interfaces et de détails techniques.

La conception



La conception

Conception préliminaire (ou conception architecturale) :

- Elle consiste à définir la structure générale du système, les composants principaux et leurs interactions.
- C’est à ce moment que l’on choisit les technologies et les outils à utiliser (langages de programmation, base de données, etc.).
- Objectif : Définir les grandes lignes de l'architecture du système, y compris le choix du type d'architecture à adopter.

La conception

L'architecture du système désigne la façon dont un logiciel ou une application est structuré, c'est-à-dire comment les différentes parties du système interagissent et s'organisent.

Cela inclut la division du système en différentes couches ou modules qui assurent chacun des tâches spécifiques.

La conception

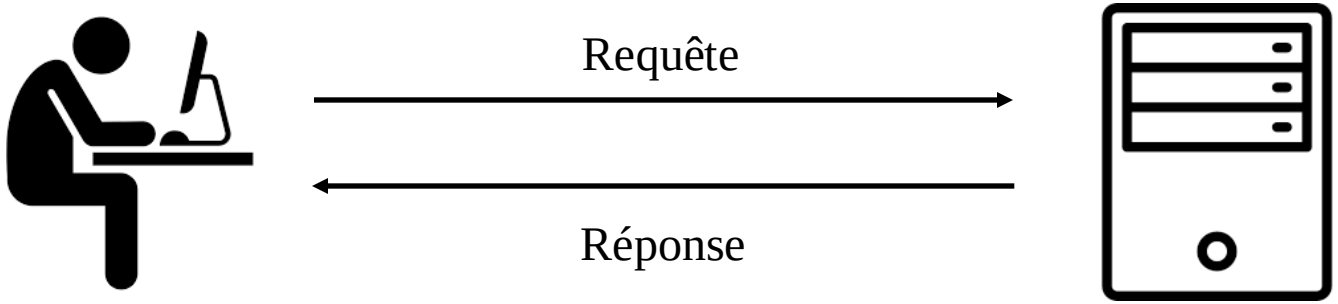
Il existe plusieurs types d'architectures de systèmes, et les plus courantes sont :

- Architecture 2-tiers (client-serveur).
- Architecture 3-tiers.

La conception

Architecture 2-tiers (**client-serveur**) : Cette architecture sépare le système en deux parties principales.

- **Le client** : C'est l'interface avec laquelle l'utilisateur interagit (par exemple, une application ou un navigateur web).
- **Le serveur** : Il contient les données et la logique métier, c'est-à-dire le traitement des informations.



La conception

Exemple d’Architecture 2-tiers : Application de Gestion des Étudiants

Une petite application de gestion des étudiants pour une école qui permet de voir les informations des étudiants et enregistrer leurs notes.

- **Client** : Une application ou un site web utilisé par le personnel de l’école. Par exemple, le secrétariat accède à une interface qui leur permet d’ajouter des étudiants, de voir leurs notes, etc.
- **Serveur** : Le serveur prend en charge toute la logique et la base de données. Il reçoit les demandes du client (par exemple, ajouter une note pour un étudiant), traite la demande (vérifie si l’étudiant existe et enregistre la note) et renvoie le résultat.

La conception

Exemple d’Architecture 2-tiers : Application de Gestion des Étudiants (suite)

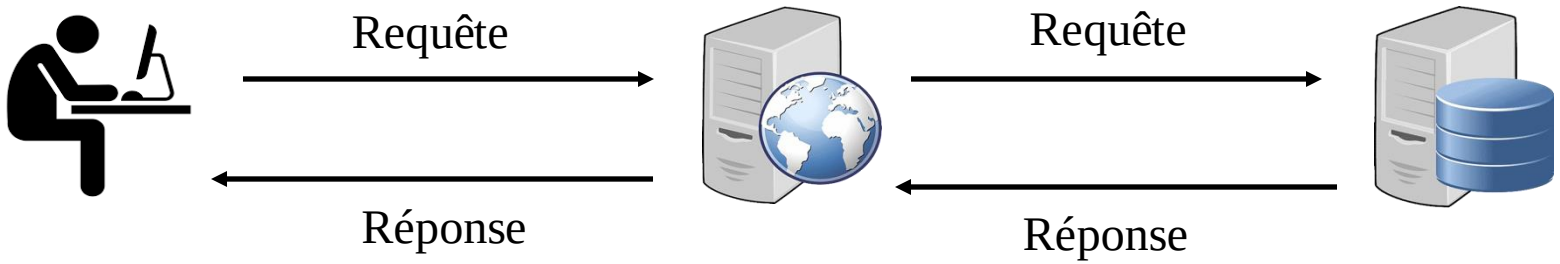
Dans ce cas :

- Le **client** est l’application ou le site web.
- Le **serveur** gère à la fois la logique métier (validation des notes et des informations) et l'accès à la base de données.

La conception

L'architecture 3-tiers divise le système en **3 couches** :

- **Couche de présentation** : C'est l'interface utilisateur, celle avec laquelle l'utilisateur interagit (par exemple, un site web ou une application mobile).
- **Couche métier** : Elle gère la logique de traitement des données et les règles d'affaires (ex : calcul des impôts, gestion des comptes utilisateurs).
- **Couche de données** : Elle est responsable de la gestion de la base de données, du stockage et de la récupération des informations.



La conception

Exemple d’Architecture 3-tiers : Application de Commande de Restaurant

Une application de commande dans un restaurant, qui permet aux clients de commander en ligne et au personnel de gérer les commandes.

- **Client** : Un site web ou une application mobile utilisée par les clients pour naviguer dans le menu, passer des commandes, et par le personnel pour voir les commandes.
- **Serveur de logique métier** : Ce serveur reçoit les commandes, vérifie si les articles sont en stock, applique des réductions ou frais de livraison, et envoie la commande au système de la cuisine.
- **Serveur de base de données** : Ce serveur est utilisé uniquement pour stocker les données des commandes, les informations sur les clients et les détails du menu.

La conception

Exemple d’Architecture 3-tiers : Application de Commande de Restaurant

Dans ce cas :

- Le **client** est le site web ou l'application.
- Le **serveur de logique métier** gère toutes les règles et traitements (exemple : vérifier la disponibilité d’un plat).
- Le **serveur de base de données** gère les informations de manière indépendante pour le stockage.

La conception

Conception détaillée :

Elle consiste à affiner la conception préliminaire en définissant précisément les détails des modules, des interfaces, des structures de données et des algorithmes à utiliser.

On utilise des diagrammes (comme les diagrammes de classe) et des spécifications pour définir les fonctions et les comportements des composants individuels du système.

Suite …