🎓 FICHES DE RÉVISION · NIVEAU 6

Concepteur développeur d'applications

17 fiches de révision pour préparer la soutenance du titre professionnel de niveau 6. Un thème par fiche : les points à maîtriser, les définitions clés, les questions du jury et l'essentiel à retenir.

Chaque point est pensé pour être dicible à l'oral. Le contenu est volontairement générique : à relier à ton propre projet le jour de l'examen.

Référentiel officiel : fiche RNCP37873 — francecompetences.fr

---

DÉVELOPPEMENT

- ⚙️ Environnement de développement
- 🎨 Interfaces utilisateur et accessibilité
- 🧩 Composants métier et patterns de conception
- ✨ Qualité et sécurité du code
- 🌳 Gestion de versions avec Git

CONCEPTION

- 🏛️ Architecture logicielle et couches (MVC)
- 📐 Modèles de données et ORM
- 🗄️ Sécurité et optimisation des bases de données
- 🛡️ Sécurité applicative (OWASP)

DÉPLOIEMENT

- 🐋 Conteneurisation avec Docker
- ✅ Tests automatisés
- ⚡ CI/CD et démarche DevOps
- 🚢 Documentation et déploiement

TRANSVERSAL

- ⚖️ RGPD et protection des données
- 🌱 Éco-conception
- 🚴 Gestion de projet et méthode agile
- 🎤 Recul et posture à l'oral

Développement

DÉVELOPPEMENT

Environnement de développement

EN BREF

Un environnement bien configuré est **reproductible, versionné et sécurisé**. On installe et configure ses outils en fonction du projet, jamais l'inverse.

Points à maîtriser

- **IDE et extensions :** VS Code ou IntelliJ selon la stack, avec linter, formateur et debugger. L'éditeur n'est pas neutre : il porte le linting et le formatage de l'équipe.
- **Versions de runtime figées :** fnm ou nvm pour Node, SDKMAN pour Java. On fige la version par projet (`.nvmrc` , `.sdkmanrc`) pour que toute l'équipe ait le même runtime.
- **Dépendances et fichier de lock :** npm, Maven, pip. Le fichier de lock (`package-lock.json` , `pom.xml`) garantit des installations identiques d'une machine à l'autre.
- **Variables d'environnement :** Secrets et configuration hors du code, dans un `.env` jamais commité (listé dans `.gitignore`). La config vit dans l'environnement, pas dans le code.
- **Reproductibilité :** Un README de prise en main, et parfois un conteneur ou un devcontainer, pour qu'une nouvelle personne soit opérationnelle en quelques minutes.

Définitions clés

IDE	Environnement de développement intégré : édition, exécution, debug et refactoring au même endroit.
Linter	Outil d'analyse statique qui repère erreurs et écarts de style (ESLint, Checkstyle).
.env	Fichier listant les variables d'environnement (secrets, URLs), chargées au runtime et exclues du dépôt Git.
Fichier de lock	Fige les versions exactes des dépendances installées pour des builds reproductibles.

Questions du jury

« Pourquoi versionner la version de Node ou de Java ? »

Pour éviter le « ça marche chez moi » : même runtime pour tout le monde, donc mêmes comportements et bugs reproductibles.

« Où mettez-vous vos secrets ? »

Jamais dans le code ni dans Git. Dans des variables d'environnement ou un gestionnaire de secrets, injectées au déploiement.

✓ À RETENIR

Un bon environnement de travail est reproductible et sécurisé : on clone le dépôt et on démarre en quelques minutes, sans aucun secret en clair.

DÉVELOPPEMENT

🎨 Interfaces utilisateur et accessibilité

EN BREF

On développe les interfaces à partir des maquettes en respectant le **RGAA**, pour que l'application soit utilisable par toutes les personnes, y compris en situation de handicap.

Points à maîtriser

- **HTML sémantique d'abord :** Les bonnes balises (`button` , `nav` , `main` , titres `h1` à `h6` , `label`) portent l'accessibilité avant tout attribut ARIA.
- **RGAA et WCAG :** Le RGAA est la déclinaison française des WCAG. Quatre principes : Perceptible, Utilisable, Compréhensible, Robuste.
- **Contraste et couleur :** Contraste texte AA minimum (4.5:1), et l'information n'est jamais portée par la seule couleur : on ajoute un libellé, une icône ou une forme.
- **Navigation clavier :** Tout élément cliquable est atteignable et actionnable au clavier, avec un focus visible et un ordre de tabulation logique.
- **Formulaires :** Chaque champ a un `label` associé ; les erreurs sont explicites et reliées au champ (`aria-describedby`), pas seulement affichées en rouge.

Définitions clés

RGAA	Référentiel général d'amélioration de l'accessibilité : cadre français basé sur les WCAG.
WCAG	Web Content Accessibility Guidelines, le standard international du W3C.
ARIA	Attributs complétant la sémantique HTML pour les technologies d'assistance, à utiliser en dernier recours.
Lecteur d'écran	Logiciel restituant l'interface en audio ou en braille (NVDA, VoiceOver).

💬 Questions du jury

« Comment garantissez-vous l'accessibilité ? »

Sémantique HTML d'abord, contraste AA, navigation clavier, puis test avec un lecteur d'écran et un audit (axe, Lighthouse). Ce n'est pas qu'une checklist finale.

« Le rouge suffit pour signaler une erreur de saisie ? »

Non : l'information ne doit jamais reposer sur la seule couleur. On ajoute un texte et une icône explicites.

« Première règle de l'ARIA ? »

Ne pas utiliser ARIA si une balise HTML native existe. Une bonne sémantique vaut mieux qu'un attribut ajouté.

✓ À RETENIR

L'accessibilité se conçoit dès la maquette : sémantique, contraste, clavier. Ce n'est pas une couche qu'on ajoute à la fin.

POUR ALLER PLUS LOIN

↗ RGAA : le référentiel officiel · accessibilite.numerique.gouv.fr

DÉVELOPPEMENT

Composants métier et patterns de conception

EN BREF

Les composants métier portent les **règles de gestion** de l'application. Les design patterns sont des solutions éprouvées à des problèmes de conception récurrents.

Points à maîtriser

- **Logique métier isolée :** Les règles de gestion vivent dans une couche service ou domaine, séparées de l'interface et de l'accès aux données.
- **Patterns de création :** Singleton (une seule instance), Factory (déléguer la création d'objets). Utiles, mais à ne pas sur-employer.
- **Patterns de structure :** DTO (objet de transfert entre couches), Repository (abstraction de l'accès aux données derrière une interface).
- **SOLID :** Cinq principes de conception objet, dont Single Responsibility (une classe, une responsabilité) et Dependency Inversion (dépendre d'abstractions).
- **Injection de dépendances :** On fournit à une classe ses dépendances depuis l'extérieur plutôt qu'elle ne les crée : couplage plus faible et code testable.

Définitions clés

Design pattern	Solution réutilisable et documentée à un problème de conception courant (catalogue GoF).
DTO	Data Transfer Object : objet qui transporte des données entre couches, sans logique.
Repository	Pattern isolant l'accès aux données derrière une interface stable.
SOLID	Acronyme de cinq principes de conception objet (SRP, OCP, LSP, ISP, DIP).

Questions du jury

« Citez un design pattern que vous avez utilisé. »

Donne un exemple concret de ton projet (Repository pour l'accès BDD, Factory pour instancier un service) et explique le problème qu'il résout.

« Singleton : avantage et danger ? »

Une seule instance partagée, mais couplage fort et difficulté à tester. À utiliser avec parcimonie.

« Faut-il connaître les 23 patterns par cœur ? »

Non : mieux vaut deux ou trois patterns bien compris et reliés à ton code que de réciter un catalogue.

✓ À RETENIR

Un pattern répond à un problème précis. On ne le place pas pour « faire joli », on le choisit parce qu'il résout un vrai besoin.

DÉVELOPPEMENT

Qualité et sécurité du code

EN BREF

Un code de qualité est **lisible, testé, cohérent et sûr**. La qualité se mesure et s'outille, elle ne se décrète pas.

Points à maîtriser

- **Conventions et lint :** Un style homogène (linter et formateur) réduit la charge mentale et les erreurs. Le code se lit bien plus souvent qu'il ne s'écrit.
- **Revue de code :** La relecture par une autre personne de l'équipe attrape des bugs et diffuse la connaissance. On commente le code, jamais la personne.
- **Dette technique :** Les raccourcis pris, volontaires ou non, coûtent plus tard. On la nomme et on la rembourse progressivement.
- **Sécurité du code :** Valider toutes les entrées, ne jamais concaténer une entrée utilisateur dans une requête (requêtes préparées), gérer les erreurs sans fuiter d'info sensible.
- **Dépendances à jour :** Maintenir les bibliothèques et scanner les vulnérabilités (npm audit, Dependabot). Une faille dans une dépendance est ta faille.

Définitions clés

Dette technique	Coût futur induit par des choix de code sous-optimaux pris pour aller plus vite.
Revue de code	Relecture d'une modification par une autre personne de l'équipe avant sa fusion.
Refactoring	Améliorer la structure interne du code sans changer son comportement observable.
Analyse statique	Inspection du code sans l'exécuter, pour mesurer la qualité et détecter des vulnérabilités (SonarQube).

💬 Questions du jury

« Comment garantissez-vous la qualité de votre code ? »

Lint et formateur, tests automatisés, revue de code, analyse statique. C'est un ensemble de réflexes, pas une seule mesure.

« Peut-on concaténer une variable dans une requête SQL ? »

Jamais : c'est une porte ouverte à l'injection. On utilise des requêtes paramétrées.

✓ À RETENIR

La qualité et la sécurité s'outillent : lint, tests, revue, requêtes préparées, dépendances à jour. Des réflexes quotidiens, pas une étape finale.

DÉVELOPPEMENT

Gestion de versions avec Git

EN BREF

Git est un gestionnaire de versions **distribué** qui trace l'historique du code et permet de collaborer sans s'écraser. Chaque personne a une copie complète du dépôt.

Points à maîtriser

- **Commit :** Une photo cohérente du projet à un instant T, avec un message clair. Un commit correspond à un changement logique.
- **Branches :** On isole une fonctionnalité ou un correctif sur une branche, puis on la fusionne. La branche principale reste stable.
- **Pull request :** Demande de fusion d'une branche : c'est le point de revue de code avant intégration.
- **Merge et rebase :** Merge conserve l'historique réel et crée un commit de fusion ; rebase rejoue les commits pour un historique linéaire mais réécrit l'histoire.
- **Remote :** `origin` est le dépôt distant. `push` envoie, `fetch` récupère sans fusionner, `pull` récupère et fusionne.

Définitions clés

Dépôt	L'ensemble du projet et son historique de versions.
Branche	Ligne de développement parallèle et isolée.
Conflit	Divergence entre deux versions d'une même ligne, à résoudre manuellement.
.gitignore	Fichier listant ce qui ne doit pas être versionné (node_modules, .env, artefacts de build).

💬 Questions du jury

« Différence entre merge et rebase ? »

Merge garde l'historique réel avec un commit de fusion ; rebase produit un historique linéaire mais réécrit l'histoire, à éviter sur une branche partagée.

« Que ne commit-on jamais ? »

Les secrets et les fichiers générés : .env, clés, node_modules, artefacts de build.

« git pull ou git fetch ? »

fetch récupère seulement les changements ; pull fait fetch puis merge.

✓ À RETENIR

Git = historique tracé, travail en branches et revue par pull request. On ne commit jamais de secret et la branche principale reste toujours déployable.

POUR ALLER PLUS LOIN

↗ GitQuest, apprendre Git en jouant · gitquest.app

Conception

CONCEPTION

Architecture logicielle et couches (MVC)

EN BREF

Une architecture **en couches** sépare les responsabilités pour rendre l'application maintenable, testable et évolutive. MVC en est le modèle le plus connu côté web.

Points à maîtriser

- **Architecture multicouche :** Présentation (interface), métier (services), accès aux données (repository). Chaque couche ne dialogue qu'avec la couche voisine.
- **MVC :** Modèle (données et règles), Vue (affichage), Contrôleur (reçoit la requête, orchestre, renvoie la réponse).
- **Séparation des responsabilités :** Chaque composant a un rôle unique. On peut changer de base de données sans toucher à l'interface.
- **Couplage faible, cohésion forte :** Les couches sont indépendantes (couplage faible) et chacune est cohérente autour d'un rôle (cohésion forte).
- **Sécurité côté serveur :** L'ANSSI recommande une architecture en couches. La validation et les contrôles d'accès se font côté serveur, jamais uniquement côté client.

Définitions clés

MVC	Modèle-Vue-Contrôleur : patron séparant données, présentation et contrôle.
Couche métier	Contient les règles de gestion, indépendante de l'interface et de la persistance.
Couplage	Degré de dépendance entre deux composants ; on vise un couplage faible.
Architecture multicouche	Organisation en couches empilées où chacune a une responsabilité propre.

Questions du jury

« Pourquoi séparer en couches ? »

Maintenabilité, testabilité, évolutivité : on remplace une couche sans casser les autres.

« Où validez-vous les données ? »

Côté serveur systématiquement. Le contrôle côté client améliore l'ergonomie mais ne protège rien.

« En MVC, qui fait quoi ? »

Le contrôleur orchestre, le modèle porte la logique et les données, la vue affiche. Le contrôleur ne contient pas la logique métier.

✓ À RETENIR

Séparer les couches, c'est pouvoir modifier une partie sans casser le reste. La logique métier et la sécurité vivent côté serveur.

CONCEPTION

Modèles de données et ORM

EN BREF

On modélise les données (**MCD, MLD**) avant de créer la base relationnelle. L'ORM fait le pont entre les objets du code et les tables de la base.

Points à maîtriser

- **Modélisation :** Merise (MCD puis MLD) ou diagramme de classes UML. On identifie entités, attributs et relations avant d'écrire du SQL.
- **Cardinalités :** Elles décrivent les relations (1-1, 1-n, n-n). Une relation n-n se traduit par une table d'association.
- **Normalisation :** Éliminer les redondances (1re, 2e, 3e forme normale) pour garantir la cohérence et éviter les anomalies.
- **Clés :** Clé primaire (identifie une ligne) et clé étrangère (référence une autre table, garantit l'intégrité référentielle).
- **Problème N+1 :** Un ORM mal utilisé déclenche une requête par élément d'une liste. On charge en une seule requête (jointure, eager loading).

Définitions clés

MCD	Modèle Conceptuel de Données (Merise) : vue métier indépendante de la technique.
ORM	Object-Relational Mapping : couche traduisant objets et tables (JPA/Hibernate, Prisma, Eloquent).
Normalisation	Processus d'organisation des tables pour réduire la redondance.
Intégrité référentielle	Garantie qu'une clé étrangère pointe toujours vers une ligne existante.

💬 Questions du jury

« À quoi sert un ORM, et quel est son risque ? »
Productivité et code portable, mais requêtes cachées et problème N+1 si on ne surveille pas le SQL généré.

« Comment traduit-on une relation n-n ? »
Par une table d'association (table de jonction) portant les deux clés étrangères.

« SQL ou NoSQL ? »
SQL pour des données structurées et relationnelles avec des garanties fortes ; NoSQL pour de la flexibilité, du volume ou des documents.

✓ À RETENIR

On modélise avant de coder la base. L'ORM accélère mais ne dispense pas de comprendre le SQL qu'il génère : attention au N+1.

CONCEPTION

Sécurité et optimisation des bases de données

EN BREF

Une base doit être **rapide, fiable et sécurisée**. Index, transactions et gestion des droits sont les leviers principaux.

Points à maîtriser

- **Index :** Accélèrent la lecture en évitant de parcourir toute la table, mais ralentissent l'écriture et occupent de l'espace. On indexe les colonnes filtrées ou jointes.
- **EXPLAIN :** Analyse le plan d'exécution d'une requête pour repérer un parcours complet de table à optimiser.
- **Transactions et ACID :** Une transaction est atomique (tout ou rien), cohérente, isolée, durable. Indispensable aux opérations critiques comme un virement.
- **Requêtes préparées :** Elles séparent le code SQL des données : cela bloque les injections et améliore les performances.
- **Moindre privilège :** Le compte applicatif n'a que les droits nécessaires. On ne fait pas tourner l'application avec un compte administrateur.

Définitions clés

Index	Structure annexe accélérant la recherche sur une ou plusieurs colonnes.
Transaction	Ensemble d'opérations exécutées comme un tout indivisible.
ACID	Atomicité, Cohérence, Isolation, Durabilité.
Requête préparée	Requête paramétrée où les valeurs sont transmises séparément du code SQL.

Questions du jury

« Un index n'a-t-il que des avantages ? »

Non : il accélère les lectures mais ralentit les écritures et occupe de l'espace. On indexe ce qui est réellement filtré.

« Comment évitez-vous les injections SQL ? »

Requêtes préparées et paramétrées, jamais de concaténation d'entrée utilisateur.

« À quoi sert une transaction ? »

Garantir qu'une suite d'opérations réussit ou échoue en bloc, comme le débit et le crédit d'un virement.

✓ À RETENIR

Index pour lire vite (pas partout), transactions pour la fiabilité, requêtes préparées et moindre privilège pour la sécurité.

CONCEPTION

Sécurité applicative (OWASP)

EN BREF

On intègre la sécurité à chaque étape en suivant les recommandations de l'**ANSSI** et l'**OWASP Top 10**. La sécurité est une préoccupation constante, pas une option.

Points à maîtriser

- **Injection :** Du code injecté via une entrée non filtrée (SQL notamment). Parade : requêtes préparées et validation des entrées.
- **XSS :** Du script injecté et exécuté dans le navigateur d'une autre personne. Parade : échapper et encoder les sorties, en-tête CSP.
- **CSRF :** On force le navigateur d'une victime authentifiée à envoyer une requête. Parade : jeton anti-CSRF et cookies SameSite.
- **Authentification et autorisation :** Authentifier vérifie l'identité, autoriser vérifie les droits. Les mots de passe sont hachés (bcrypt, argon2), jamais stockés en clair.
- **Défense en profondeur :** Plusieurs couches de protection, moindre privilège, HTTPS partout. On ne fait jamais confiance à une entrée.

Définitions clés

OWASP Top 10	Liste de référence des dix risques de sécurité web les plus critiques.
XSS	Cross-Site Scripting : injection de script exécuté côté client.
CSRF	Falsification de requête inter-site exploitant une session active.
Hachage	Transformation irréversible d'un mot de passe (avec sel), à distinguer du chiffrement réversible.

💬 Questions du jury

« Différence entre authentification et autorisation ? »
Authentification : qui es-tu. Autorisation : à quoi as-tu droit. Deux étapes distinctes.

« Comment stockez-vous les mots de passe ? »
Hachés et salés avec un algorithme lent dédié (bcrypt, argon2), jamais chiffrés ni en clair.

« XSS ou injection SQL ? »
Le XSS s'exécute dans le navigateur de la victime ; l'injection SQL vise la base de données.

✓ À RETENIR

On ne fait jamais confiance à une entrée. Requêtes préparées contre l'injection, encodage des sorties contre le XSS, mots de passe hachés, HTTPS partout.

POUR ALLER PLUS LOIN

↗ OWASP Top 10 · owasp.org (en anglais)
↗ Guides et recommandations de l'ANSSI · cyber.gouv.fr

Déploiement

DÉPLOIEMENT

Conteneurisation avec Docker

EN BREF

Docker empaquète une application et ses dépendances dans un **conteneur reproductible**, identique du poste de dev à la production.

Points à maîtriser

- **Image et conteneur :** L'image est un modèle immuable (couches figées) ; le conteneur est une instance en exécution de cette image.
- **Docker et machine virtuelle :** Docker partage le noyau de l'hôte (léger, démarrage rapide) ; une VM embarque un système d'exploitation complet.
- **Dockerfile :** Décrit étape par étape la construction d'une image : base, dépendances, code, commande de lancement.
- **docker-compose :** Orchestre plusieurs services (application, base de données, reverse proxy) dans un seul fichier.
- **Volumes :** Persistent les données hors du cycle de vie du conteneur : une base ne doit pas vivre dans le conteneur lui-même.
- **Sécurité :** Ne pas tourner en root, secrets hors de l'image, images officielles à jour et légères (multi-stage build).

Définitions clés

Dockerfile	Fichier décrivant, étape par étape, comment construire une image.
Volume	Espace de stockage persistant monté dans un conteneur.
Registry	Dépôt d'images (Docker Hub) d'où l'on pull ou push des images.
docker-compose	Outil de définition et de lancement d'applications multi-conteneurs.

Questions du jury

« À quoi sert Docker ? »
Ne réponds pas « ça marche sur ma machine ». Dis : reproductibilité dev/prod, isolation, onboarding rapide.

« Différence entre image et conteneur ? »
L'image est le modèle figé, le conteneur son exécution. Relance fréquente ensuite : Docker face à une VM.

« Où stockez-vous les données d'une base en conteneur ? »
Dans un volume, jamais dans le conteneur lui-même.

✓ À RETENIR

Un conteneur = même environnement partout, de mon poste à la prod. L'image est le modèle, le conteneur son instance en cours d'exécution.

DÉPLOIEMENT

✅ Tests automatisés

EN BREF

On rédige et exécute un **plan de tests** pour garantir que l'application fonctionne et qu'une modification ne casse rien. Les tests sécurisent les évolutions.

Points à maîtriser

- **Pyramide des tests :** Beaucoup de tests unitaires (rapides), moins d'intégration, encore moins d'end-to-end (lents et fragiles).
- **Test unitaire :** Teste une unité isolée (fonction, méthode), sans base ni réseau, à l'aide de doublures (mocks).
- **Test d'intégration :** Vérifie que plusieurs composants fonctionnent ensemble, par exemple un service avec une vraie base.
- **Test end-to-end :** Simule un parcours utilisateur·ice complet dans un navigateur (Playwright, Cypress).
- **Structure AAA :** Arrange (préparer), Act (agir), Assert (vérifier). Un test valide un seul comportement.
- **Couverture :** Le pourcentage de code exécuté par les tests. Utile, mais pas un but en soi : 100 % ne prouve pas l'absence de bug.

Définitions clés

Test unitaire	Test d'une brique isolée, rapide et déterministe.
Mock	Objet factice simulant une dépendance pour isoler l'unité testée.
TDD	Test-Driven Development : écrire le test avant le code (rouge, vert, refactor).
Couverture de code	Proportion du code exécutée par la suite de tests.

💬 Questions du jury

« Pourquoi tester ? »

Détecter les régressions, documenter le comportement attendu, refactorer sereinement. Ce n'est pas une perte de temps.

« 100 % de couverture veut dire code sans bug ? »

Non : la couverture mesure ce qui est exécuté, pas la pertinence des assertions.

« Différence entre test unitaire et d'intégration ? »

L'unitaire isole une brique avec des mocks ; l'intégration fait collaborer plusieurs composants réels.

✓ À RETENIR

La pyramide des tests : beaucoup d'unitaires, quelques intégrations, peu d'e2e. Les tests permettent de modifier le code sans peur de tout casser.

DÉPLOIEMENT

CI/CD et démarche DevOps

EN BREF

DevOps rapproche développement et exploitation. La **CI/CD** automatise l'intégration, les tests et le déploiement pour livrer souvent et de façon fiable.

Points à maîtriser

- **Intégration continue (CI) :** À chaque push, un pipeline compile, lint et teste automatiquement. On détecte les erreurs au plus tôt.
- **Livraison et déploiement continus (CD) :** La livraison continue prépare une version déployable ; le déploiement continu la met en production automatiquement après validation.
- **Pipeline :** Suite d'étapes automatisées (build, test, analyse, déploiement) décrites dans un fichier (GitHub Actions, GitLab CI).
- **Culture DevOps :** Collaboration dev et ops, automatisation, feedback rapide, responsabilité partagée. Ce n'est pas qu'un outil.
- **Monitoring :** Logs, métriques et alertes pour surveiller la production et réagir vite en cas d'incident.

Définitions clés

CI	Continuous Integration : intégration et test automatiques à chaque changement.
CD	Continuous Delivery ou Deployment : mise à disposition ou en production automatisée.
Pipeline	Enchaînement automatisé d'étapes de build, de test et de déploiement.
Infrastructure as Code	Infrastructure décrite dans des fichiers versionnés, reproductible et auditable.

Questions du jury

« Différence entre CI et CD ? »

La CI intègre et teste en continu ; la CD livre ou déploie en continu. La CI vient d'abord.

« DevOps, est-ce un outil ? »

Non : c'est d'abord une culture (collaboration, automatisation, feedback). Les outils la servent.

« À quoi sert un pipeline ? »

Automatiser build, tests et déploiement pour livrer vite et sans erreur humaine.

✓ À RETENIR

La CI teste chaque changement automatiquement, la CD le livre. DevOps est avant tout une culture de collaboration et d'automatisation.

DÉPLOIEMENT

Documentation et déploiement

EN BREF

On prépare, documente et automatise le déploiement pour une mise en production **fiable et reproductible**. La documentation rend l'application maintenable par d'autres.

Points à maîtriser

- **Documentation technique :** README (installation, lancement), documentation d'API (Swagger/OpenAPI), commentaires utiles. Elle vit avec le code.
- **Environnements :** Dev, recette (staging), production. On valide en recette avant la prod, avec des configurations séparées.
- **Procédure de déploiement :** Documentée et reproductible, idéalement automatisée, incluant les migrations de base de données.
- **Rollback :** Pouvoir revenir à la version précédente en cas d'incident. Un déploiement sans plan de retour arrière est risqué.
- **Versionnement sémantique :** MAJEUR.MINEUR.CORRECTIF, pour communiquer clairement la portée de chaque changement.

Définitions clés

Staging (recette)	Environnement de pré-production imitant la prod pour valider avant livraison.
OpenAPI / Swagger	Standard de description d'une API REST, qui génère une documentation interactive.
Rollback	Retour à une version stable précédente après un déploiement défaillant.
Versionnement sémantique	Convention de numérotation MAJEUR.MINEUR.CORRECTIF.

Questions du jury

« Comment déployez-vous ? »

Procédure documentée et automatisée, passage par la recette, migrations gérées, plan de rollback. Pas un copier-coller manuel sur le serveur.

« À quoi sert la documentation ? »

Rendre le projet maintenable et l'onboarding rapide. Une API sans documentation est difficilement utilisable.

« Que faites-vous si le déploiement casse la prod ? »

Rollback vers la version précédente, puis analyse à froid de la cause.

✓ À RETENIR

Un déploiement se prépare, se documente et s'automatise, avec un plan de retour arrière. La documentation garde l'application maintenable.

TRANSVERSAL

RGPD et protection des données

EN BREF

Le RGPD encadre le traitement des **données personnelles**. On met en place les mesures techniques dès la conception (privacy by design).

Points à maîtriser

- **Donnée personnelle :** Toute information identifiant une personne, directement ou non : nom, email, adresse IP, identifiant.
- **Base légale :** Un traitement doit reposer sur une base : consentement, contrat, obligation légale ou intérêt légitime.
- **Grands principes :** Minimisation (ne collecter que le nécessaire), limitation de conservation, finalité déterminée, sécurité.
- **Droits des personnes :** Accès, rectification, effacement (droit à l'oubli), portabilité, opposition. L'application doit permettre de les exercer.
- **Privacy by design et by default :** La protection est intégrée dès la conception et activée par défaut, avec les options les moins intrusives.

Définitions clés

RGPD	Règlement Général sur la Protection des Données, cadre européen applicable depuis 2018.
Donnée personnelle	Information se rapportant à une personne physique identifiée ou identifiable.
Consentement	Accord libre, éclairé, spécifique et univoque au traitement.
Pseudonymisation	Traitement rendant les données non attribuables sans information supplémentaire.

Questions du jury

« Comment appliquez-vous le RGPD dans le code ? »

Minimisation des données, consentement pour les cookies non essentiels, droit à l'effacement, chiffrement, mentions légales. Du concret, pas du juridique récité.

« Le consentement est-il obligatoire pour tout ? »

Non : il existe d'autres bases légales (contrat, obligation légale, intérêt légitime).

« Que faire en cas de fuite de données ? »

Notifier la CNIL sous 72 heures, et informer les personnes concernées si le risque est élevé.

✓ À RETENIR

RGPD = privacy by design : on minimise les données, on sécurise, et on rend les droits des personnes (accès, effacement) réellement exerçables.

POUR ALLER PLUS LOIN

↗ Guide RGPD du développeur (CNIL) · cnil.fr

TRANSVERSAL

Éco-conception

EN BREF

L'éco-conception vise à réduire l'**empreinte environnementale** du numérique dès la conception, tout en améliorant souvent la performance.

Points à maîtriser

- **Sobriété :** N'implémenter que les fonctionnalités utiles. La fonctionnalité la plus verte est celle qu'on ne développe pas.
- **Poids des pages :** Optimiser les images (formats modernes, compression), limiter les scripts et les requêtes réseau. Moins de données transférées, moins d'énergie.
- **Performance et sobriété :** Cache, requêtes efficaces (éviter le N+1), chargement différé. Ce qui accélère réduit aussi la consommation.
- **Durée de vie :** Allonger la durée de vie du matériel en évitant l'obsolescence induite par des logiciels de plus en plus lourds.
- **Référentiel RGESN :** Le Référentiel général d'écoconception de services numériques guide les bonnes pratiques.

Définitions clés

Éco-conception	Démarche réduisant l'impact environnemental d'un service sur tout son cycle de vie.
RGESN	Référentiel général d'écoconception de services numériques.
Sobriété numérique	Limiter les usages et les ressources au strict nécessaire.
Lazy loading	Chargement différé des ressources, seulement quand elles deviennent nécessaires.

💬 Questions du jury

« L'éco-conception ralentit-elle le projet ? »

Au contraire : sobriété et performance vont de pair. Une page plus légère est plus rapide et moins énergivore.

« Un exemple concret d'éco-conception ? »

Compresser et servir des images au bon format, éviter les requêtes N+1, mettre en cache, supprimer le code mort.

« L'éco-conception se résume-t-elle au serveur vert ? »

Non : l'essentiel se joue dans la conception et le code, pas seulement dans l'hébergement.

✓ À RETENIR

Éco-concevoir, c'est d'abord la sobriété : moins de fonctionnalités inutiles, des pages légères, un code performant. Bon pour la planète comme pour les personnes qui utilisent le service.

TRANSVERSAL

Gestion de projet et méthode agile

EN BREF

L'agilité privilégie l'**itératif et l'adaptation au changement** plutôt qu'un plan figé. On livre de la valeur régulièrement en s'ajustant au retour des utilisateur·ices.

Points à maîtriser

- **Manifeste agile :** Quatre valeurs, dont « les individus et les interactions » et « répondre au changement » plutôt que suivre un plan rigide.
- **Scrum :** Cadre itératif avec des sprints (1 à 4 semaines), trois rôles (Product Owner, Scrum Master, équipe) et des cérémonies.
- **Cérémonies Scrum :** Planification, mêlée quotidienne (daily), revue, rétrospective. La rétrospective améliore le fonctionnement de l'équipe.
- **Artefacts :** Product backlog (tout le travail priorisé), sprint backlog (le sprint en cours), incrément (le livrable).
- **Kanban :** Flux continu visualisé sur un tableau, avec limitation du travail en cours (WIP). Pas de sprints imposés.

Définitions clés

Agile	Approche itérative et incrémentale centrée sur l'adaptation et la valeur.
Sprint	Itération courte et cadencée produisant un incrément utilisable.
Product Owner	Porte la vision produit et priorise le backlog.
Scrum Master	Facilite, protège l'équipe et lève les obstacles ; ce n'est pas un·e chef·fe de projet.

💬 Questions du jury

« Différence entre Scrum et Kanban ? »

Scrum cadence par sprints avec des rôles définis ; Kanban est un flux continu avec limitation du travail en cours.

« Le Scrum Master est-il le chef ? »

Non : c'est une personne facilitatrice au service de l'équipe, qui n'attribue pas les tâches de façon hiérarchique.

« Agile ou cycle en V ? »

Le cycle en V convient à un besoin stable et documenté ; l'agile à un besoin évolutif avec du feedback fréquent.

✓ À RETENIR

L'agile livre par itérations courtes et s'adapte au changement. Scrum = sprints et rôles, Kanban = flux continu avec limite de travail en cours.

TRANSVERSAL

Recul et posture à l'oral

EN BREF

La soutenance évalue autant ta maîtrise technique que ta capacité à **expliquer, justifier tes choix et prendre du recul**. On raconte une démarche, pas seulement du code.

Points à maîtriser

- **Justifier ses choix :** Pour chaque techno ou décision, savoir dire pourquoi celle-ci plutôt qu'une autre, et ce qu'on aurait pu faire différemment.
- **Assumer les limites :** Reconnaître ce qui n'est pas parfait ou pas fini est une force. Le jury valorise la lucidité, pas le déni.
- **Vocabulaire précis :** Employer les bons termes (couche, dépendance, transaction, hachage) et savoir les définir simplement.
- **Gérer une question inconnue :** Ne pas inventer. Reformuler, dire ce qu'on sait, proposer une piste. « Je ne sais pas, mais voici comment je chercherais » est recevable.
- **Structurer sa présentation :** Contexte, besoin, conception, réalisation, tests, déploiement, bilan. Un fil clair aide le jury à suivre.

Définitions clés

Prise de recul	Capacité à analyser ses choix avec un regard critique et à envisager des alternatives.
Veille technologique	Suivi régulier des évolutions techniques et de sécurité de son domaine.
Dossier de projet	Document présentant le projet, support de l'entretien technique.

💬 Questions du jury

« Pourquoi ce choix technique ? »

Ne réponds jamais « parce que c'était imposé ». Donne une justification (écosystème, courbe d'apprentissage, besoin) même si le choix était contraint.

« Que feriez-vous différemment ? »

Prépare cette réponse : elle montre du recul. Cite une amélioration concrète et explique pourquoi.

« Une question dont tu ignores la réponse. »

Reste calme, ne bluffe pas : reformule, relie à ce que tu connais, propose comment tu chercherais.

✓ À RETENIR

Le jury évalue ta démarche et ton recul autant que ton code. Justifie tes choix, assume les limites, et n'invente jamais une réponse que tu ne connais pas.