

NICOLAS WALTER

CONCEVOIR ET DÉVELOPPER DES APPLICATIONS LOGICIELLES

Projet : Todo App Cloud

Réf. Évaluation : Bloc 2 (Certification RNCP39583)

Formation : Mastère 2 Expert en Développement Full Stack

École : Ynov Campus Strasbourg

23/07/2026

Résumé Exécutif

Ce dossier technique détaille la refonte Cloud-Native et DevSecOps de l'application [Todo App](#) des tutoriels Docker. Réalisé dans le cadre scolaire durant les cours "Développer pour le cloud" et amélioré pour la certification RNCP (Bloc 2), ce projet démontre la maîtrise d'une architecture microservices complète.

Stack Technique & Approche :

- **Développement** : Frontend React, Backend et Auth en Node.js (Express/Knex), avec typage strict (TypeScript).
- **Infrastructure (IaC)** : Provisionnement Azure via Terraform (Cluster AKS, Registre ACR, KeyVault, MySQL Managé).
- **Sécurité (Zero Trust)** : Authentification OIDC (sans mot de passe), RBAC Azure, secrets montés via CSI driver, réseaux virtuels (VNet) et bases de données isolées d'Internet.
- **Qualité & CI/CD** : Couverture par tests unitaires (Jest, Vitest) et End-to-End (Playwright), scans de vulnérabilités (Trivy, Dependabot) et déploiement continu automatisé (GitHub Actions) via Helm.

Le résultat est une solution logicielle hautement disponible, résiliente, conforme aux exigences de sécurité de l'OWASP Top 10 et aux standards d'accessibilité.

Table des Matières

Introduction	4
1. Architecture Logicielle, Modèle Métier & Prototype	6
1.1. Bounded Contexts (Contextes Métier)	6
1.2. Relations entre les Contextes (Upstream/Downstream)	7
1.3. Structure en Trois Couches (Three-Tier Architecture)	7
1.4. Injection de Dépendances Manuelle (DI)	7
1.5. Versioning et Migrations SQL de la Base de Données	8
1.6. Interface d'Authentification (Mire de Connexion et Inscription)	8
1.7. Tableau de Bord (Gestion de la Liste de Tâches)	9
1.8. Espace de Gestion de Compte (Conformité RGPD & Sécurité)	10
2. Intégration Continue, Tests & Critères de Qualité	11
2.1. Protocole d'Intégration Continue (CI)	11
2.2. Seuils de Performance & SLA (Service Level Agreements)	12
2.3. Protocole de Test de Charge (k6)	12
2.4. Harnais de Tests Unitaires & d'Intégration	13
3. Sécurité Applicative & Cloud (DevSecOps)	14
3.1. Alignement sur les Menaces de l'OWASP Top 10	14
3.2. Hardening DevSecOps et Sécurisation de l'Infrastructure	14
4. Accessibilité Numérique (A11y)	16
4.1. Justification des Référentiels Adoptés	16
4.2. Actions Techniques Mises en Œuvre	16
5. Protocole de Déploiement Continu (CD) & Infrastructure	17
5.1. Prérequis et Initialisation (IaC)	17
5.2. Construction et Publication des Conteneurs (ACR)	17
5.3. Déploiement applicatif (Helm)	17
5.4. Déploiement Continu Automatisé (CD GitHub Actions)	18
6. Recette Applicative & Cahier de Recettes	19
6.1. Stratégie de Test & Couverture	19
6.2. Matrice des Scénarios de Test Fonctionnels	19
6.3. Scénarios de Test de Sécurité (DevSecOps)	20
6.4. Scénarios de Test de Performance & SLA	21
7. Gestion des Anomalies & Plan de Correction	22
7.1. Suivi et Qualification des Anomalies	22
7.2. Registre des Bogues et Correctifs Appliqués	22
7.3. Plan d'Amélioration Continue (Maintien en Condition)	23
8. Manuels d'Exploitation & Maintenance	24
8.1. Manuel d'Utilisation	24
8.2. Manuel de Mise à Jour et Exploitation	25
8.3. Runbooks d'Exploitation (Fiches Réflexes)	26
8.4. Historique des Versions (Changelog)	27

Introduction

Ce dossier technique présente le travail de conception, de développement, de sécurisation et de déploiement réalisé sur l'application **Todo App Cloud**. L'objectif de ce projet est de transformer une application de démonstration locale en une architecture microservices prête pour le Cloud (Cloud-Ready), robuste, hautement disponible, sécurisée selon les principes DevSecOps, accessible, et instrumentée pour le suivi de performance.

Ce projet valide les compétences du **Bloc 2 (Concevoir et développer des applications logicielles)** pour la certification professionnelle. Afin de faciliter la lecture de ce document et d'attester de la conformité des livrables techniques, la matrice de correspondance ci-dessous identifie les chapitres répondant précisément aux exigences de l'évaluation :

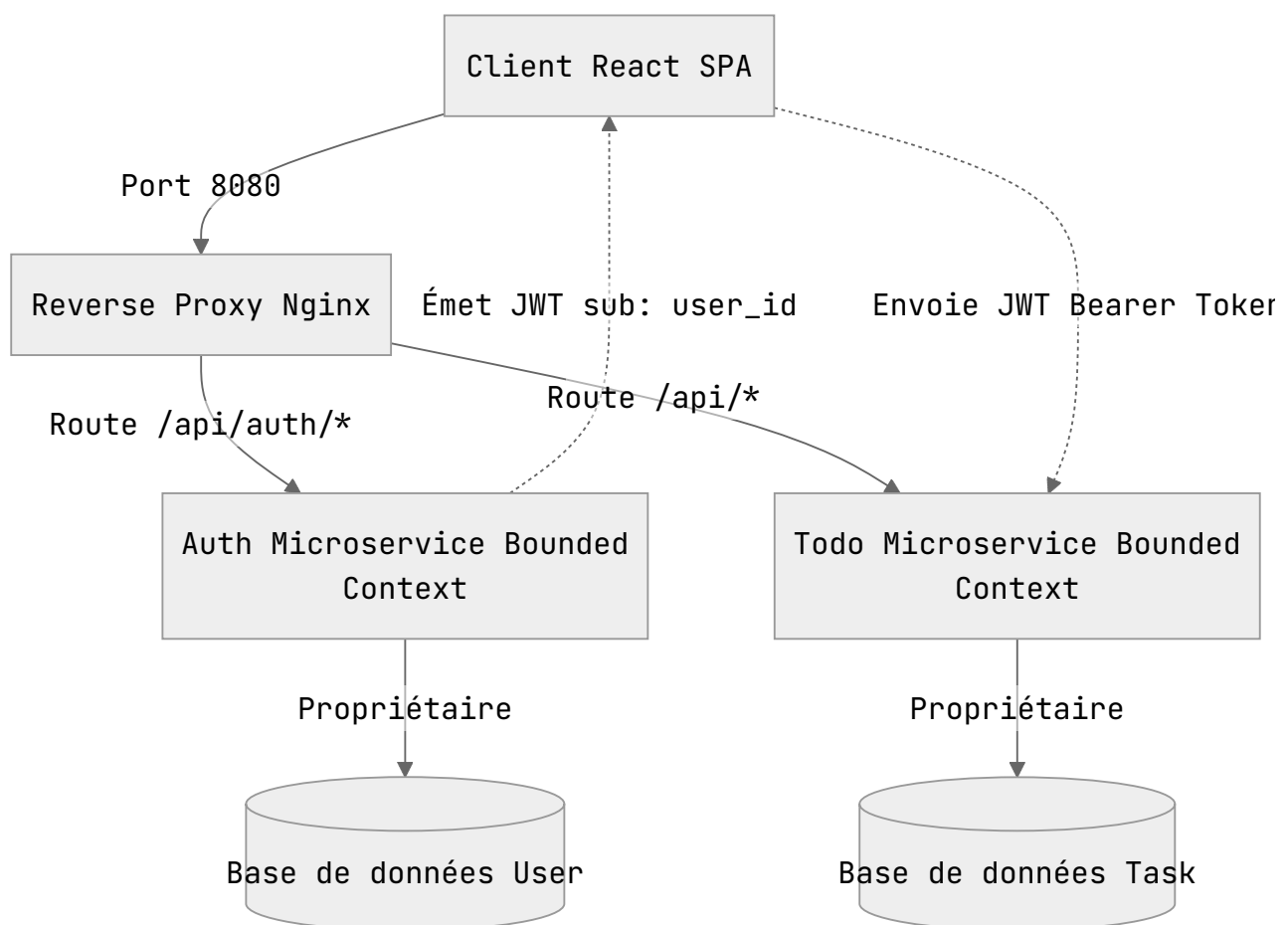
Livrable requis pour l'évaluation	Compétences RNCP	Localisation dans ce dossier technique
Le protocole de déploiement continu	C2.1.1 & C2.2.4	Chapitre 5 (Protocole de Déploiement Continu & Infrastructure)
Les critères de qualité et de performance	C2.1.1	Chapitre 2 (Intégration Continue, Tests & Critères de Qualité)
Le protocole d'intégration continue	C2.1.2	Chapitre 2 (Intégration Continue, Tests & Critères de Qualité)
Une architecture logicielle structurée	C2.2.1	Chapitre 1 (Architecture Logicielle, Modèle Métier & Prototype)
Une présentation d'un des prototypes réalisés	C2.2.1	Chapitre 1 (Architecture Logicielle, Modèle Métier & Prototype)
L'utilisation de Framework et paradigmes	C2.2.1	Chapitre 1 (Architecture Logicielle, Modèle Métier & Prototype)
Un jeu de tests unitaires (Harnais de test)	C2.2.2	Chapitre 2 (Intégration Continue, Tests & Critères de Qualité)
Une présentation des mesures de sécurité	C2.2.3	Chapitre 3 (Sécurité Applicative & Cloud)
Présentation de l'accès aux personnes handicapées	C2.2.3	Chapitre 4 (Accessibilité Numérique)
L'historique des différentes versions (Changelog)	C2.2.4	Chapitre 8 (Manuels d'Exploitation & Maintenance)
La dernière version fonctionnelle et fiable	C2.2.4	Code source fourni (Dépôt Git & Release v2.1.0)
Le cahier de recettes	C2.3.1	Chapitre 6 (Recette Applicative & Cahier de Recettes)
Le plan de correction des bogues	C2.3.2	Chapitre 7 (Gestion des Anomalies & Plan de Correction)
Le manuel de déploiement	C2.4.1	Chapitre 5 (Protocole de Déploiement Continu & Infrastructure)
Le manuel d'utilisation	C2.4.1	Chapitre 8 (Manuels d'Exploitation & Maintenance)

Le manuel de mise à jour	C2.4.1	Chapitre 8 (Manuels d’Exploitation & Maintenance)
--------------------------	--------	---

1. Architecture Logicielle, Modèle Métier & Prototype

Ce chapitre documente la structure globale du projet (Bounded Contexts, couches logicielles, injection de dépendances manuelle) et présente l'ergonomie générale du prototype réalisé, répondant aux exigences de conception et de maintenabilité de la compétence **C2.2.1**.

1.1. Bounded Contexts (Contextes Métier)



1.1.1. Contexte d'Authentification (Auth Service)

- **Responsabilité** : Gestion du cycle de vie des comptes utilisateurs (création, connexion, modifications du profil) et conformité RGPD (portabilité des données utilisateur et suppression définitive du compte).
- **Modèle de données** : Gère la table `users` dans sa base de données dédiée (`auth`).
- **Langage omniprésent** : *User, Email, Password Hash, Register, Login, Export, Delete Account*.

1.1.2. Contexte des Tâches (Todo Service / Backend)

- **Responsabilité** : Gestion de la liste de tâches des utilisateurs (création, mise à jour de l'état complété, filtrage, suppression globale).
- **Modèle de données** : Gère la table `todo_items` dans sa base de données dédiée (`todos`).
- **Langage omniprésent** : *Todo Item, Task Name, Completed State, Owner (user_id)*.

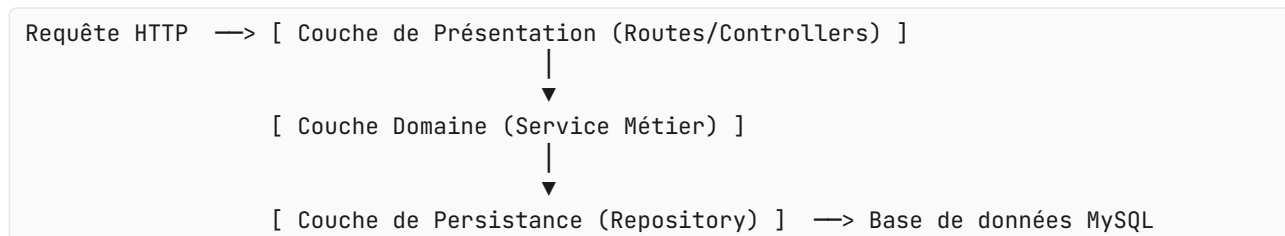
1.2. Relations entre les Contextes (Upstream/Downstream)

1.2.1. Relation : Client-Side Orchestration (Partage Asymétrique)

- **Amont (Upstream - Auth) → Aval (Downstream - Todo).**
- **Mécanisme de liaison :** L'intégration se fait via des **JSON Web Tokens (JWT)**. Le service d'authentification (amont) génère le jeton signé contenant l'identifiant de l'utilisateur (`sub: user_id`). Le service de tâches (aval) vérifie le jeton à l'aide d'un secret partagé (`JWT_SECRET`) pour autoriser et associer les tâches créées à cet utilisateur.
- **Aucun appel direct réseau :** Le backend Todo ne contacte jamais le service Auth par réseau lors des requêtes HTTP normales. Il décode le JWT de façon autonome.
- **Purge RGPD :** Lors de la suppression de compte, c'est le client React qui orchestre de façon synchrone les appels de suppression successive (d'abord `DELETE /api/items` pour vider les todos, puis `DELETE /api/auth/me` pour supprimer l'utilisateur), évitant la présence d'éléments orphelins dans la base de tâches.

1.3. Structure en Trois Couches (Three-Tier Architecture)

Chaque service métier est séparé de manière hermétique en trois niveaux de responsabilité :



1.3.1. La Couche Présentation (Routes & Controllers)

- **Rôle :** Réceptionner les flux HTTP, valider les paramètres entrants (`body`, `query`, `headers`), déléguer l'exécution à la couche domaine, et formater la réponse (`JSON`, codes de statut HTTP : 200, 201, 400, 401, 403, 404, 500).
- **Structure :** Dans les fichiers sous `src/routes/`, les routeurs Express sont modélisés comme des factories (fonctions prenant le service en argument et retournant le routeur Express). Cela permet d'isoler Express des autres couches.

1.3.2. La Couche Domaine (Service)

- **Rôle :** Contenir la logique métier centrale de l'application (validation de règles d'affaires, calculs, génération d'identifiants uniques `UUID`, orchestration des flux métier).
- **Indépendance :** La couche domaine n'a aucune connaissance d'Express (elle ne manipule ni objets `req` ou `res`, ni codes HTTP) et aucune connaissance directe de la base de données physique (elle manipule une interface abstraite de persistance).

1.3.3. La Couche de Persistance (Repository)

- **Rôle :** Interagir avec les bases de données SQL via le query-builder **Knex.js** (écriture des requêtes `SELECT`, `INSERT`, `UPDATE`, `DELETE`).
- **Abstraction par Interface :** La persistance implémente une interface TypeScript stricte (par exemple, `TodoRepository` dans `backend/src/types.ts`). Le service ne dépend que de cette interface.

1.4. Injection de Dépendances Manuelle (DI)

L'injection de dépendances consiste à passer à un objet ses collaborateurs (dépendances) lors de sa construction plutôt qu'il ne les instancie lui-même.

1.4.1. Justification technique

- **Testabilité** : Permet de tester la logique métier du service en lui passant un dépôt factice en mémoire (InMemoryRepository) sans dépendre d'une base de données active (évite les mocks globaux de modules Jest qui rendent le code fragile).
- **Évolutivité** : Permet de changer d'implémentation de stockage (ex: passer d'un fichier plat à MySQL) sans modifier une seule ligne de code de la logique métier.

1.4.2. Câblage au démarrage (Bootstrapping)

Le point d'entrée de l'application (index.ts) orchestre la création des instances dans l'ordre de leurs dépendances. Voici comment s'effectue le câblage dans backend/src/index.ts :

```
// 1. Initialisation de la connexion Knex (Base de données)
const dbConfig = getKnexConfig();
const dbConnection = knex(dbConfig);

// 2. Instanciation du Repository (MySQL) avec la connexion SQL
const repository = new KnexRepository(dbConfig);

// 3. Instanciation du Service Métier avec injection du Repository
const service = new TodoService(repository);

// 4. Lancement de l'application Express avec injection du Service dans les contrôleurs
const app = createApp(service);
app.listen(port, () => { ... });
```

1.5. Versioning et Migrations SQL de la Base de Données

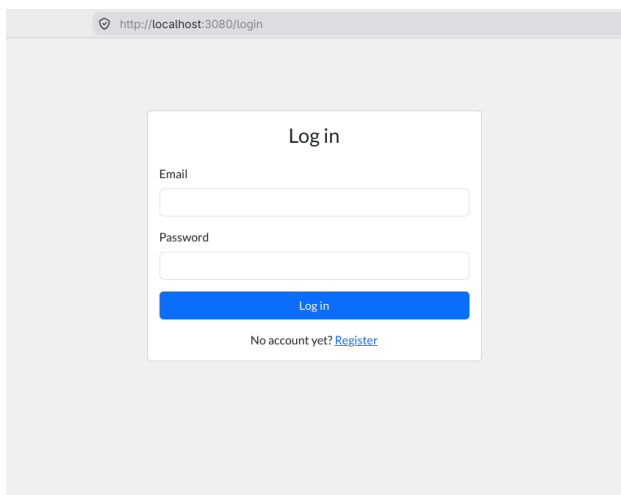
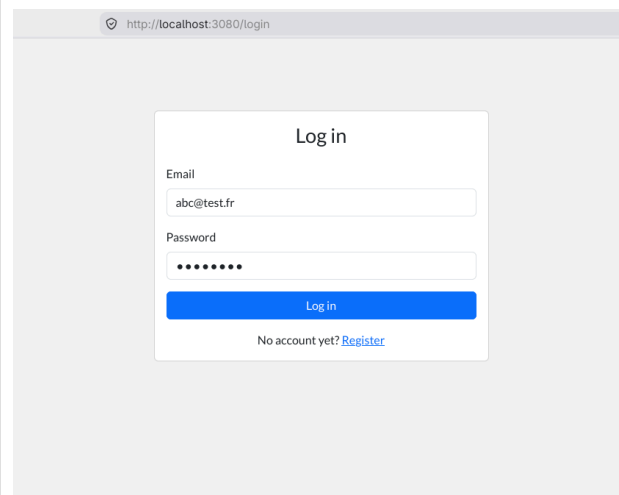
Pour répondre aux exigences de maintenabilité et de reproductibilité des environnements, la structure de la base de données est versionnée à l'aide des **migrations Knex.js** sous src/migrations/.

- **Automatisation** : Lors du démarrage d'un conteneur en staging ou en production, le microservice exécute automatiquement les migrations en attente (await db.migrate.latest()). Aucun script SQL manuel n'est requis sur le serveur.
- **Historique** : Les modifications de schéma (ajout de la colonne user_id pour la sécurité multi-utilisateur) sont ainsi tracées au même titre que le code applicatif dans Git.

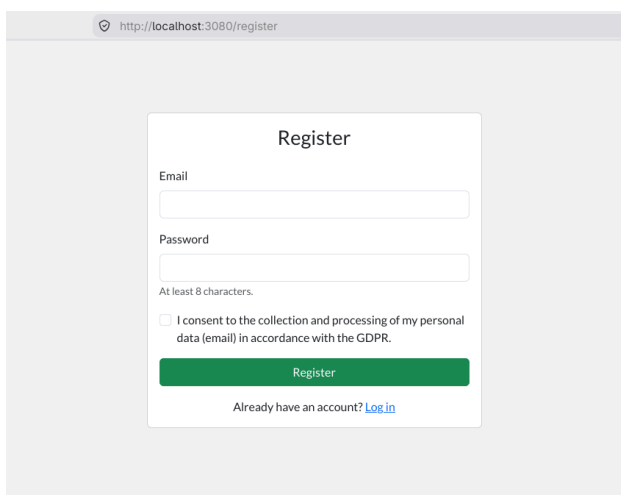
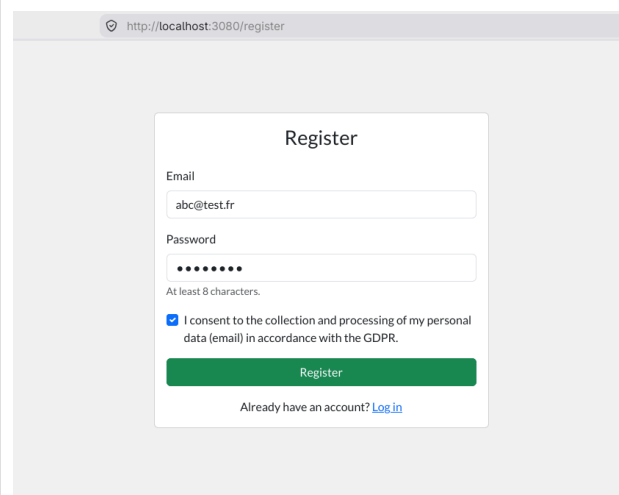
1.6. Interface d'Authentification (Mire de Connexion et Inscription)

L'accès à l'application est protégé. L'interface propose deux parcours simples, sémantiques et conformes aux bonnes pratiques d'accessibilité (champs de saisie liés à leurs libellés HTML).

- **Connexion** : L'utilisateur fournit son adresse email et son mot de passe.
 - *État initial* : Le formulaire est vide et l'action de soumission est désactivée tant que les critères minimaux ne sont pas remplis.
 - *Saisie complétée* : Le bouton "Se connecter" s'active et passe au bleu primaire, guidant visuellement l'utilisateur.

Formulaire de connexion vide	Formulaire de connexion complété
	

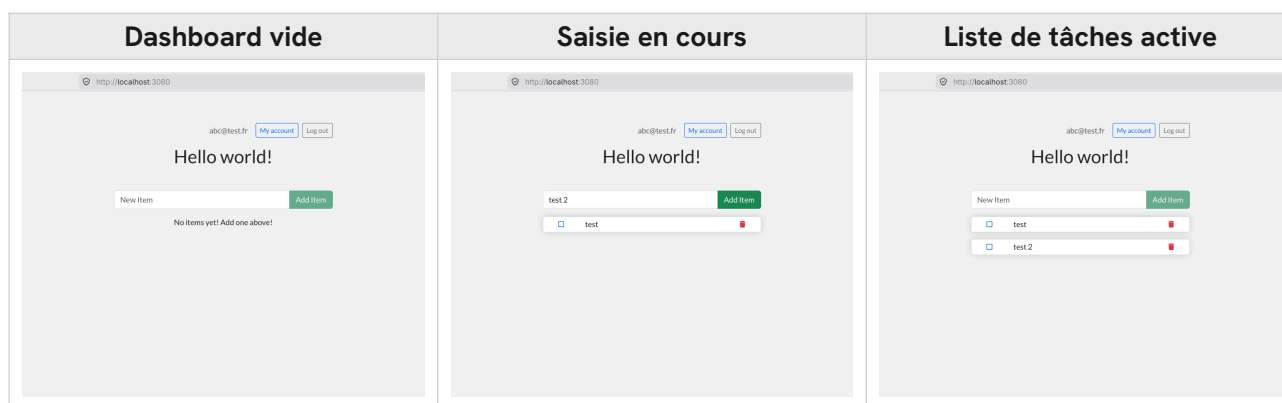
- **Inscription** : Pour s'enregistrer, un email et un mot de passe fort sont requis.
 - **Conformité RGPD** : L'inscription intègre une case à cocher explicite de consentement pour le traitement des données personnelles, bloquant la validation si elle n'est pas cochée.

Inscription vide	Inscription complétée
	

1.7. Tableau de Bord (Gestion de la Liste de Tâches)

Une fois connecté, l'utilisateur est redirigé vers son tableau de bord personnel. La navigation est conçue pour limiter la charge cognitive et permettre une interaction rapide, au clavier comme à la souris.

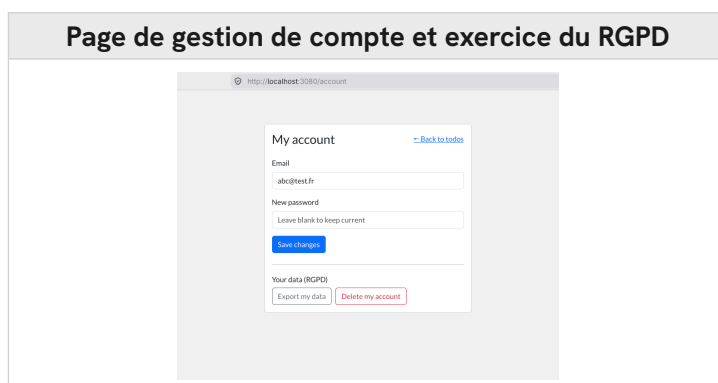
- **État vide (Empty State)** : Conformément aux recommandations OPQUAST, si l'utilisateur n'a aucune tâche, un message informatif et imagé l'invite chaleureusement à saisir sa première note.
- **Saisie dynamique** : L'interface permet d'écrire le titre d'une tâche. Une validation en temps réel bloque la saisie de chaînes vides.
- **Liste interactive** : Chaque tâche apparaît sous forme de ligne avec deux contrôles d'accessibilité (des boutons d'icône équipés d'attributs `aria-label` dynamiques pour la lecture vocale). La tâche peut être cochée (elle apparaît alors barrée) ou supprimée (clic sur l'icône de corbeille).



1.8. Espace de Gestion de Compte (Conformité RGPD & Sécurité)

Une page dédiée « Mon Compte » permet à l'utilisateur de gérer ses informations et d'exercer ses droits réglementaires :

- **Évolutivité des informations** : Modification sécurisée de l'adresse email et du mot de passe en cours de session.
- **Portabilité des données (RGPD - Art. 20)** : Un bouton permet d'exporter instantanément toutes les données détenues (profil et ensemble des tâches associé) sous format structuré JSON.
- **Droit à l'oubli (RGPD - Art. 17)** : Un bouton de suppression définitive permet d'effacer le profil de la base d'authentification et de purger toutes ses tâches en base de données, sans laisser d'éléments orphelins.



2. Intégration Continue, Tests & Critères de Qualité

Ce chapitre détaille le pipeline d'intégration continue unifié, les seuils de performance SLA (validés par k6) ainsi que la conception du harnais de test unitaire (Jest/Vitest), répondant aux exigences des compétences **C2.1.2** et **C2.2.2**.

2.1. Protocole d'Intégration Continue (CI)

Le projet est conçu pour être totalement agnostique vis-à-vis de l'environnement de développement : le choix de l'éditeur de code (IDE) est libre, et toutes les commandes s'exécutent en ligne de commande (CLI). L'unique contrainte stricte est la présence de **Docker Engine** et **Docker Compose**. Cela permet d'isoler l'environnement local et d'exécuter les tests avec la même architecture logicielle (microservices et bases de données), sans avoir à reproduire localement la complexité opérationnelle de la production (qui s'appuie sur Kubernetes, Azure Key Vault et des bases managées).

Le protocole d'intégration continue est au cœur de notre démarche d'assurance qualité et de prévention des régressions. Configuré via GitHub Actions dans `.github/workflows/ci.yml`, il s'exécute automatiquement à chaque soumission de Pull Request ou push sur la branche principale main.

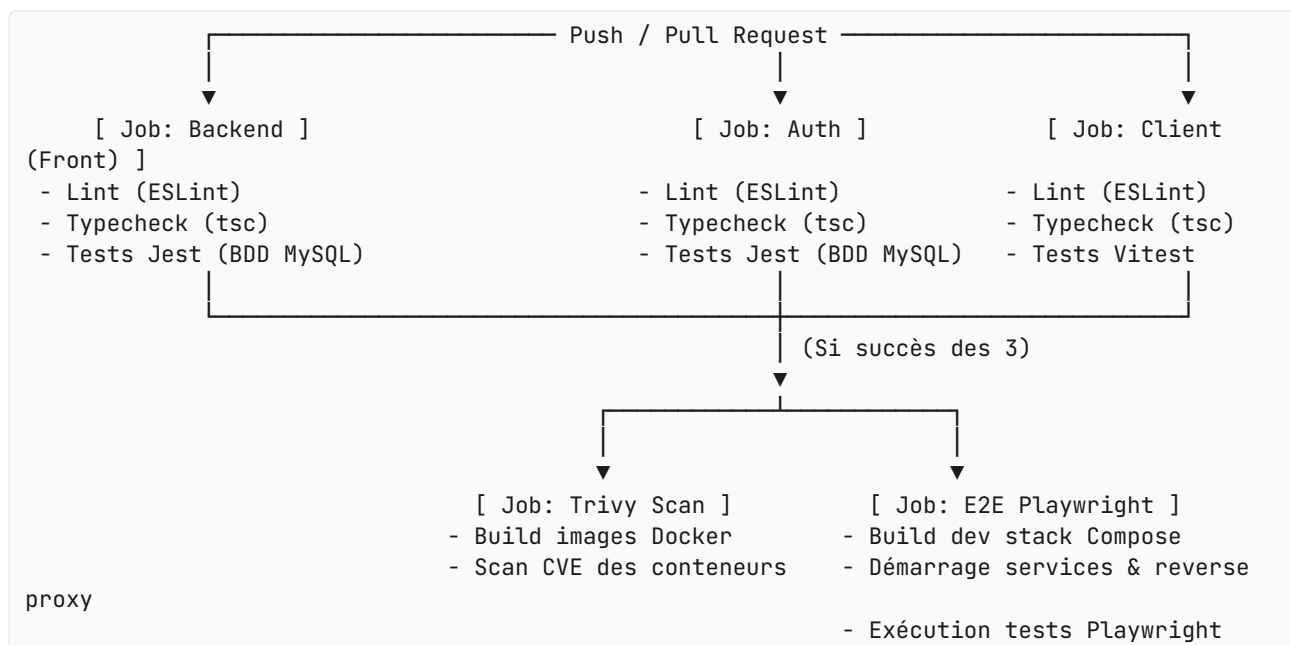
2.1.1. Cycle de validation et "Merge Gates"

Pour garantir la stabilité du tronc commun, aucun développeur ne peut fusionner directement du code non validé. Le dépôt applique une politique stricte de **Merge Gates** (barrières de fusion) exigeant :

1. Que la pipeline d'intégration continue s'exécute avec succès (statut vert).
2. Que la couverture de test ne régresse pas.
3. Qu'aucune vulnérabilité critique de dépendance ou d'image conteneur ne soit introduite.

2.1.2. Séquence détaillée d'intégration

Le workflow CI orchestre les validations de manière parallèle et progressive afin d'accélérer les retours (Fast Feedback) :



2.1.2.1. Étape 1 : Validation Statique (Linting et Typechecking)

Pour chaque composant (backend, auth, client), le pipeline installe les dépendances Node.js et exécute :

- **ESLint** : Pour détecter les erreurs de syntaxe, l’usage de variables obsolètes ou non définies, et faire respecter les standards de codage.
- **TypeScript (tsc --noEmit)** : Pour valider la cohérence et l’intégrité du typage statique à travers tout le projet.

2.1.2.2. Étape 2 : Validation Dynamique (Harnais de Tests Unitaires et d’Intégration)

Les tests unitaires sont exécutés automatiquement :

- Les microservices backend et auth s’appuient sur des conteneurs MySQL de test démarrés à la volée en tant que services GitHub Actions pour valider les requêtes Knex réelles.
- Le client React exécute ses tests de composants via Vitest.

2.1.2.3. Étape 3 : Construction et Scan de Sécurité des Conteneurs (Trivy)

Une fois le code validé, les Dockerfiles multi-stage de production sont compilés. L’action **Trivy Vulnerability Scanner** analyse immédiatement les images compilées pour identifier :

- Les vulnérabilités du système d’exploitation de base (Debian/Alpine).
- Les packages ou dépendances logicielles obsolètes contenant des failles de sécurité connues (CVE). Si une faille de niveau HIGH ou CRITICAL sans patch existant est identifiée, le build est interrompu.

2.1.2.4. Étape 4 : Tests de Bout en Bout Applicatifs (Playwright E2E)

Pour valider le comportement fonctionnel réel (parcours utilisateur) :

1. Une instance éphémère complète de l’application est démarrée via Docker Compose (incluant le Reverse Proxy Traefik, les deux bases de données MySQL de test, et les microservices backend, auth et front).
2. Un script d’attente s’assure que tous les endpoints HTTP répondent correctement.
3. Les tests **Playwright E2E** sont lancés en simulant des interactions réelles sur un navigateur Chromium sans tête (Headless) : inscription, connexion, création de tâche, complétion, suppression, et suppression définitive de compte RGPD.
4. Les résultats et vidéos de test sont archivés en tant qu’artefacts GitHub.

2.2. Seuils de Performance & SLA (Service Level Agreements)

L’application déployée sur Kubernetes doit respecter les seuils de performance mesurables suivants :

Métrique (Metric)	Seuil (SLA)	Descriptif et Justification
Temps de réponse p(95)	< 300 ms	95% des appels HTTP de l’API doivent être traités sous les 300ms.
Taux d’échec des requêtes	< 1 %	Moins de 1% des appels doivent renvoyer des erreurs HTTP 5xx.
Démarrage des services	< 5 s	Temps requis pour démarrer un nouveau pod lors de l’autoscaling.

2.3. Protocole de Test de Charge (k6)

Un test de charge automatisé a été développé avec **k6** (voir [k6-load-test.js](#)).

- **Scénario d’usage (nominal)** : 1. Accueil API (GET /greeting), 2. Inscription d’un nouvel utilisateur (POST /auth/register), 3. Ajout d’une tâche avec jeton JWT (POST /items), 4. Lecture de la liste (GET /items), 5. Suppression de la tâche et suppression du compte (droit à l’oubli).
- **Profil de charge (60s)** : Montée progressive (Ramp-up) de 0 à 15 utilisateurs virtuels simultanés (VUs) pour tester le comportement de la mise à l’échelle.
- **Assertions (Thresholds k6)** :

```
thresholds: {
  http_req_duration: ['p(95)<300'], // Durée de requête
  http_req_failed: ['rate<0.01'], // Taux d'échec
}
```

▪ Exécution des tests de charge :

```
# Exécution locale
k6 run performance/k6-load-test.js

# Exécution Docker (alternative)
docker run --rm --network host -i grafana/k6 run - <performance/k6-load-test.js
```

2.4. Harnais de Tests Unitaires & d'Intégration

L'assurance qualité repose sur un harnais de tests unitaires et d'intégration automatisés (Jest pour le backend/auth, Vitest pour le client, Playwright pour le E2E).

2.4.1. Isolation de l'environnement de test (Jest)

- **Base de données éphémère** : Pour les tests unitaires et d'intégration, nous utilisons des instances **MySQL éphémères (via Docker)**.
- **Hooks de nettoyage** : Le hook `beforeEach` initialise le schéma SQL par migration à blanc. Le hook `afterEach` effectue la fermeture propre (`teardown`) pour éviter toute fuite de ressources.

2.4.2. Illustration d'un scénario de test d'isolation (Jest)

Le scénario ci-dessous utilise le modèle standardisé **AAA (Arrange, Act, Assert)** pour valider l'isolation logique des données entre deux utilisateurs distincts :

```
const { TodoService } = require("../src/service/TodoService");
const { KnexRepository } = require("../src/repository/KnexRepository");
const { getKnexConfig } = require("../src/repository/knexConfig");

const USER = "user-1";
const OTHER_USER = "user-2";
let service;

beforeEach(async () => {
  service = new TodoService(new KnexRepository(getKnexConfig()));
  await service.init();
});

test("a user cannot see another user items", async () => {
  // 1. ARRANGE : L'utilisateur B crée une tâche privée
  await service.addItem(OTHER_USER, "Tâche Secrète");

  // 2. ACT : L'utilisateur A tente de lister ses tâches
  const items = await service.getAllItems(USER);

  // 3. ASSERT : On valide que la liste retournée pour A est vide
  expect(items).toEqual([]);
});
```

Le rapport de couverture (`npm run test:coverage`) atteint **plus de 90 %** sur les fichiers de services et repositories, assurant une forte fiabilité.

3. Sécurité Applicative & Cloud (DevSecOps)

Ce chapitre présente l'évaluation et les contre-mesures appliquées face à l'OWASP Top 10 ainsi que le durcissement DevSecOps de l'infrastructure cloud AKS/MySQL (Zero Trust), répondant aux exigences de sécurisation de la compétence **C2.2.3**.

3.1. Alignement sur les Menaces de l'OWASP Top 10

L'architecture logicielle apporte des réponses techniques systématiques aux failles de sécurité majeures :

- **Contrôle d'accès défaillant (A01: Broken Access Control) :**
 - L'accès aux ressources est cloisonné par identifiant utilisateur (`userId`).
 - Le middleware `requireAuth` extrait l'`userId` du jeton JWT et le propage. Le dépôt Knex applique systématiquement la clause `WHERE user_id = userId` sur toutes les opérations d'écriture et de lecture.
- **Défaillances cryptographiques (A02: Cryptographic Failures) :**
 - Les mots de passe sont hachés de manière asymétrique via **bcrypt** (facteur de coût de 10).
 - Les sessions s'appuient sur des cookies de session JWT sécurisés : configurés en `HttpOnly` (invisibles au JavaScript contre les attaques XSS), `Secure` (transmission HTTPS stricte) et `SameSite=Strict` (protection contre le CSRF).
- **Injections (A03: Injection) :**
 - Toutes les interactions avec les bases de données SQL sont gérées par le query-builder **Knex.js** qui utilise des requêtes paramétrées par défaut, éliminant tout risque d'injection SQL.
- **Mauvaises configurations de sécurité (A05: Security Misconfiguration) :**
 - Les conteneurs de production s'exécutent avec des comptes restreints non-privilegiés (`node` et `nginx` sur port 8080) avec l'option `runAsNonRoot: true`.
 - Le démarrage de l'authentification crash instantanément (`throw Error`) si la clé de signature du jeton est absente ou non sécurisée en production.
- **Faiblesses d'identification et d'authentification (A07: Identification and Authentication Failures) :**
 - Le middleware `express-rate-limit` restreint les tentatives d'authentification brute-force à 5 essais par tranche de 15 minutes par adresse IP.

3.2. Hardening DevSecOps et Sécurisation de l'Infrastructure

L'infrastructure provisionnée (Terraform) et déployée (Helm) met en œuvre des mécanismes d'isolation avancés :

- **Gestion des Secrets (Key Vault & CSI Driver) :**
 - Aucun secret ou mot de passe n'est écrit en clair dans le code ou sur Git.
 - Les secrets de production sont stockés dans **Azure Key Vault** (protection anti-suppression active).
 - L'identité managée du cluster AKS utilise le pilote **Secrets Store CSI Driver** pour monter de façon éphémère les secrets sous forme de fichiers temporaires (`/mnt/secrets/*`) au démarrage des pods.
- **Fédération d'Identité OIDC en CI/CD :**
 - Le pipeline GitHub Actions se connecte à Azure via le protocole **OpenID Connect (OIDC)** sans utiliser de secret ou mot de passe statique stocké sur GitHub.
- **Isolation Réseau Privée (VNet Integration) :**
 - Le serveur Azure MySQL Flexible est isolé dans un sous-réseau privé délégué (VNet) sans exposition publique.
 - La communication entre AKS et MySQL s'effectue exclusivement en interne via une **Zone DNS Privée** (`private.mysql.database.azure.com`).

- **Cloisonnement Kubernetes (Network Policies) :**
 - Des règles réseau **NetworkPolicies** bloquent par défaut tout trafic entrant dans les pods applicatifs. Seul le Reverse Proxy Nginx peut recevoir du trafic internet public et rediriger de façon sécurisée vers les microservices internes.

4. Accessibilité Numérique (A11y)

Ce chapitre documente les actions techniques d'accessibilité (RGAA v4.1.2 et bonnes pratiques OPQUAST) pour l'inclusion des utilisateurs en situation de handicap, répondant aux exigences d'accessibilité de la compétence **C2.2.3**.

4.1. Justification des Référentiels Adoptés

- **RGAA v4.1.2 (Référentiel Général d'Amélioration de l'Accessibilité)** : Choisi comme référentiel principal. Il transpose les normes internationales **WCAG 2.1** au niveau de conformité réglementaire **AA** (double A) obligatoire en France.
- **Référentiel OPQUAST (Bonnes Pratiques de Qualité Web)** : Mobilisé pour harmoniser la qualité globale (formulaires, navigation, gestion du consentement RGPD et prévention des erreurs d'utilisabilité).

4.2. Actions Techniques Mises en Œuvre

L'application a été audité et corrigée pour répondre aux critères essentiels des référentiels :

- **Structure Sémantique (Landmarks)** :
 - L'ensemble de l'application est enveloppé dans une balise `<main>` (via l'attribut `as="main"` sur le conteneur principal dans [App.tsx](#)) définissant le repère principal pour les lecteurs d'écran.
 - L'en-tête de page et sa navigation sont délimités par une balise `<header role="banner">` et un élément `<nav aria-label="Account navigation">`.
- **Hierarchie des Titres (Headings)** :
 - Chaque page indépendante possède **un unique titre principal h1** de premier niveau (Greeting sur l'accueil, titre de carte sur les pages Login et Register, et page de compte).
 - Les sous-titres (comme la section RGPD sur la page compte) sont passés en titre h2 (avec classe visuelle de bootstrap pour le style) pour éviter les sauts hiérarchiques (erreur d'accessibilité fréquente).
- **Contrôles de Formulaires & Boutons d'Icônes** :
 - Les boutons d'action sans texte visible (case à cocher interactive de complétion et icône corbeille dans [ItemDisplay.tsx](#)) sont équipés d'attributs `aria-label` descriptifs et dynamiques (ex: "Mark item as incomplete", "Remove Item"). Les icônes SVG disposent de l'attribut `aria-hidden="true"`.
 - Tous les champs de saisie (Login, Inscription, Compte) lient sémantiquement les balises `<label>` et `<input>` via la propriété `controlId` de React-Bootstrap, garantissant l'annonce vocale correcte lors de la focalisation.
- **Navigation au Clavier** :
 - Aucun contour de focus par défaut n'a été masqué via CSS (pas de `outline: none`). L'utilisateur naviguant à l'aide de la touche Tab bénéficie d'une visibilité claire sur l'élément actif.

5. Protocole de Déploiement Continu (CD) & Infrastructure

Ce chapitre détaille les procédures de déploiement multi-environnement (IaC Terraform, conteneurs, Helm) et le protocole de CD automatisé par fédération OIDC, répondant aux exigences des compétences **C2.1.1** et **C2.2.4**.

5.1. Prérequis et Initialisation (IaC)

- **Outils requis** : Azure CLI (az connecté via `az login`), Terraform, Helm (v3) et Docker.
- **Provisionnement de l'infrastructure (Azure)** :

```
cd azure
terraform init
terraform apply -auto-approve
```

Notez les valeurs de sorties (outputs) générées : `resource_group_name`, `kubernetes_cluster_name`, `container_registry_name`, `key_vault_name` et `mysql_server_name`.

5.2. Construction et Publication des Conteneurs (ACR)

Connectez-vous au registre privé Azure Container Registry (ACR) pour compiler et pousser les images de production :

```
# Connexion au registre
az acr login --name <NOM_ACR>

# Compilation et push matriciel
docker build -t <NOM_ACR>.azurecr.io/backend:latest ./backend
docker build -t <NOM_ACR>.azurecr.io/auth:latest ./auth
docker build -t <NOM_ACR>.azurecr.io/client:latest ./client

docker push <NOM_ACR>.azurecr.io/backend:latest
docker push <NOM_ACR>.azurecr.io/auth:latest
docker push <NOM_ACR>.azurecr.io/client:latest
```

5.3. Déploiement applicatif (Helm)

5.3.1. Déploiement Local (Test)

```
helm upgrade --install todo-app ./k8s/todo-app -f ./k8s/todo-app/values-test.yaml
```

5.3.2. Déploiement en Production (Azure AKS)

Connectez-vous au cluster AKS, générez le certificat TLS pour le proxy, et déployez à l'aide de Helm :

```
# Connexion au cluster AKS
az aks get-credentials --resource-group <NOM_RG> --name <NOM_AKS>

# Génération et chargement du certificat TLS
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout tls.key -out tls.crt -subj "/CN=todo.localhost"
kubectl create secret tls todo-tls-cert --key tls.key --cert tls.crt
```

```
# Déploiement Helm de Production
helm upgrade --install todo-app ./k8s/todo-app \
  -f ./k8s/todo-app/values-prod.yaml \
  --set mysql.host=<DNS_MYSQL> \
  --set mysql.authHost=<DNS_MYSQL> \
  --set keyvault.name=<NOM_KEY_VAULT> \
  --set keyvault.tenantId=<TENANT_ID> \
  --set keyvault.clientId=<KEYVAULT_CLIENT_ID> \
  --set client.image.repository=<NOM_ACR>.azurecr.io/client \
  --set backend.image.repository=<NOM_ACR>.azurecr.io/backend \
  --set auth.image.repository=<NOM_ACR>.azurecr.io/auth
```

Vérifiez le statut avec `kubectl get pods` et obtenez l'IP publique du proxy via `kubectl get svc reverse-proxy`.

5.4. Déploiement Continu Automatisé (CD GitHub Actions)

Le pipeline de CD automatisé est configuré dans [.github/workflows/cd.yml](#).

5.4.1. Sécurisation de l'intégration par OIDC (OpenID Connect)

Le pipeline n'utilise aucun secret statique d'API (mot de passe). Il s'appuie sur une fédération d'identité temporaire d'Azure Entra ID :

1. **Service Principal** : Créé avec les droits de contributeur sur le groupe de ressources.
2. **Federated Credentials** : Une relation de confiance est configurée sur le portail Azure pour l'organisation GitHub, liée à la branche `main` (staging) et aux tags `v*` (production).
3. **Secrets GitHub** : Seuls les identifiants publics sont configurés dans les variables GitHub (`AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, `AZURE_SUBSCRIPTION_ID`, `ACR_NAME`, `RESOURCE_GROUP`, `CLUSTER_NAME`).

5.4.2. Déclenchement du pipeline (GitFlow)

- **Déploiement en Staging (Automatique)** : Tout push ou merge réussi sur `main` déploie l'application dans le namespace `staging` (BDD locale éphémère, pas de Key Vault).
- **Déploiement en Production (Automatique)** : La création d'un tag de version correspondant au format `v*` (ex: `v2.1.0`) compile, scanne via Trivy, et déploie dans le namespace `production` (BDD MySQL managée isolée, Key Vault CSI Secrets active).

6. Recette Applicative & Cahier de Recettes

Ce chapitre regroupe la matrice des scénarios de test fonctionnels, de sécurité et de performance exécutés pour valider la livraison du logiciel, répondant aux exigences de recette de la compétence C2.3.1.

6.1. Stratégie de Test & Couverture

L'application s'appuie sur une pyramide des tests automatisés et manuels :

- 1. **Tests Unitaires & Intégration API (Backend/Auth)** : Écrit en Jest, s'exécutant sur une base MySQL éphémère Docker. (29 tests pour le backend, 23 tests pour l'authentification).
- 2. **Tests Frontend (Client)** : Écrit en Vitest, simulant le rendu des composants React et le comportement de l'UI. (23 tests).
- 3. **Tests de Charge & Performance** : Script k6 simulant des parcours utilisateurs sous charge réseau.
- 4. **Tests de Sécurité** : Scans de vulnérabilités Trivy (images Docker) et Dependabot (code).

6.2. Matrice des Scénarios de Test Fonctionnels

ID	Titre du Scénario	Conditions préalables	Description des actions	Résultat attendu	Statut
TC-01	Inscription utilisateur	Base de données active	Appeler POST /api/auth/register avec un email et un mot de passe valides.	Retourne 201 Created avec le JWT de session.	PASS
TC-02	Inscription doublon	Compte test@example.com existant	Retenter une inscription avec le même email test@example.com.	Retourne 409 Conflict (sécurité d'unicité).	PASS
TC-03	Connexion réussie	Compte existant en base	Appeler POST /api/auth/login avec identifiants corrects.	Retourne 200 OK et injecte le cookie sécurisé auth_token (HttpOnly).	PASS
TC-04	Connexion erronée	-	Tenter une connexion avec un mot de passe incorrect.	Retourne 401 Unauthorized.	PASS
TC-05	Création de tâche	Utilisateur authentifié	Cliquer sur "Ajouter une tâche" avec le nom Acheter du pain.	Envoi POST /api/items. La tâche apparaît instantanément dans l'UI.	PASS
TC-06	Validation de saisie	Utilisateur authentifié	Tenter de soumettre une tâche avec un nom	Le bouton est désactivé ou l'API retourne 400 Bad Request.	PASS

ID	Titre du Scénario	Conditions préalables	Description des actions	Résultat attendu	Statut
			vide ou de plus de 255 caractères.		
TC-07	Modification (Toggle)	Tâche existante	Cliquer sur la case à cocher d'une tâche.	La tâche est marquée comme complétée (completed: true) en base et barrée visuellement.	PASS
TC-08	Suppression de tâche	Tâche existante	Cliquer sur l'icône de corbeille d'une tâche.	La tâche disparaît de l'affichage. La ligne est supprimée de la base.	PASS
TC-09	Isolation des données (IDOR)	Deux utilisateurs A et B actifs	L'utilisateur A tente de lire ou modifier les tâches de l'utilisateur B via des requêtes directes.	Le serveur renvoie une erreur 403 Forbidden ou 404 Not Found.	PASS
TC-10	Suppression de compte (RGPD)	Utilisateur avec 5 tâches en base	Aller dans "Mon compte" et cliquer sur "Supprimer mon compte".	L'utilisateur est supprimé d'Auth, et toutes ses tâches associées sont purgées (droit à l'oubli).	PASS

6.3. Scénarios de Test de Sécurité (DevSecOps)

ID	Cible de Sécurité	Outil / Procédure	Risque analysé	Résultat constaté	Statut
SEC-01	Exfiltration de session	Console développeur (JS)	Exfiltration du JWT via une attaque XSS.	document.cookie ne retourne pas le token (protégé par HttpOnly).	PASS
SEC-02	Injection de dépendances	Scan Trivy en pipeline CD	Paquets ou images contenant des vulnérabilités HIGH/CRITICAL.	Build bloqué en cas de faille exploitable non corrigée.	PASS
SEC-03	Brute force sur mot de passe	Requêtes répétées sur /login	Épuisement de CPU ou découverte de mot de passe.	Le middleware de rate-limiting bloque l'IP après 5 tentatives échouées.	PASS

6.4. Scénarios de Test de Performance & SLA

ID	Métrique testée	Outil	Seuil (SLA)	Résultat constaté	Statut
PERF-01	Temps de réponse API (Lecture/Écriture)	k6 (15 utilisateurs virtuels)	p(95) < 300 ms	225,16 ms (local Docker)	PASS
PERF-02	Taux de réussite des requêtes	k6 (Scénario nominal complet)	Taux d'erreur < 1 %	0.00 % d'erreurs	PASS

7. Gestion des Anomalies & Plan de Correction

Ce chapitre trace le registre des anomalies qualifiées durant la recette technique ainsi que le plan de correction appliqué, répondant aux exigences de suivi de la compétence **C2.3.2**.

7.1. Suivi et Qualification des Anomalies

Nous utilisons une classification par sévérité pour traiter les bogues :

- **Bloquant** : L'application ou une partie majeure (ex: les tests, le build Docker, l'authentification) crash.
- **Majeur** : Une fonctionnalité métier essentielle échoue ou comporte une faille de sécurité importante.
- **Mineur** : Problème esthétique ou d'utilisabilité sans impact sur la stabilité.

7.2. Registre des Bogues et Correctifs Appliqués

7.2.1. B-01 : Crash des tests en environnement Docker (SQLite3 / GLIBC)

- **Symptôme** : Lors de l'exécution des tests du backend en conteneur, Jest échouait immédiatement avec l'erreur `/lib/x86_64-linux-gnu/libm.so.6: version 'GLIBC_2.38' not found`.
- **Sévérité** : Bloquant.
- **Analyse** : Le package `sqlite3` avait été mis à jour en version 6.0.1. Les binaires pré-compilés de cette version exigent la version 2.38 de GLIBC. Or, l'image officielle de base `node:22` (Debian Bookworm) embarque GLIBC 2.36.
- **Correction** : Migration complète vers MySQL pour les tests (iso-production) et suppression définitive du package `sqlite3` de l'application (cf. ADR 016).
- **Résultat** : La dépendance problématique `sqlite3` est éliminée. Les tests s'exécutent avec succès contre la base MySQL éphémère.

7.2.2. B-02 : Crash serveur (TypeError 500) sur mise à jour de tâche inexistante

- **Symptôme** : Si un utilisateur tentait d'appeler l'API de modification sur un identifiant de tâche inexistant ou appartenant à un autre utilisateur, le serveur backend crashait avec un code HTTP 500.
- **Sévérité** : Majeur (Sécurité & Robustesse).
- **Analyse** : La fonction `updateItem` modifiait la ligne dans Knex, puis appelait `getItem(userId, id)` pour retourner l'état mis à jour. N'ayant rien trouvé, Knex renvoyait `undefined`. La fonction `mapRow(row)` tentait alors de lire `row.id` sur une valeur non définie, levant une exception non gérée.
- **Correction** :
 1. Modification de `mapRow(row)` dans `KnexRepository.ts` pour retourner `null` si `row` est indéfini.
 2. Ajout d'une vérification d'existence dans le contrôleur `updateItem.ts` renvoyant proprement un code 404 `Not Found` au lieu d'une erreur 500.

7.2.3. B-03 : Risque de Fail-Open sur la configuration JWT

- **Symptôme** : En cas de problème de lecture du fichier secret JWT en production, le microservice d'authentification basculait silencieusement sur la clé de secours par défaut `'dev-insecure-secret'`.
- **Sévérité** : Majeur (Sécurité).
- **Analyse** : Ce fallback automatique permettait à un attaquant de forger de faux jetons en production si le fichier secret n'était pas monté correctement dans Kubernetes.
- **Correction** : Modification de `AuthService.ts` pour lever explicitement une exception critique (`throw Error`) si la configuration de production est invalide, empêchant le conteneur de démarrer et alertant l'équipe d'exploitation.

7.3. Plan d'Amélioration Continue (Maintenance en Condition)

Pour éviter la régression de la qualité du code :

1. **Intégration Continue (CI)** : Exécution automatisée de `npm run test` et `npm run lint` à chaque Pull Request pour intercepter les bogues de syntaxe ou de logique avant la fusion sur la branche principale.
2. **Dependabot** : Analyse automatique hebdomadaire de la sécurité du code pour détecter les CVEs dans les dépendances tierces et générer les PRs de mise à jour.

8. Manuels d'Exploitation & Maintenance

Ce chapitre réunit les manuels d'utilisation, de mise à jour et d'exploitation requis pour le suivi des équipes et l'historique des versions, répondant aux exigences de documentation de la compétence **C2.4.1**.

Pour assurer la traçabilité et le suivi par les équipes opérationnelles, nous fournissons ici le manuel d'utilisation (incluant les aspects RGPD/droit à l'oubli), le manuel de mise à jour/rollback de l'application, et les fiches reflexes d'exploitation.

8.1. Manuel d'Utilisation

8.1.1. Accès à l'Application et Authentification

L'application requiert un compte utilisateur pour isoler de manière sécurisée vos tâches.

8.1.1.1. Créer un compte (Inscription)

1. Sur la page d'accueil, cliquez sur **S'inscrire** (Register).
2. Saisissez une adresse email valide et un mot de passe robuste.
3. Cliquez sur le bouton de soumission. Votre compte est créé et vous êtes automatiquement connecté.

8.1.1.2. Se connecter (Connexion)

1. Sur la page de connexion, saisissez vos identifiants (email et mot de passe).
2. Cliquez sur **Se connecter**.
3. *Note de sécurité* : L'application utilise des cookies sécurisés (HttpOnly). Vous restez connecté de manière sécurisée même si vous fermez l'onglet de votre navigateur.

8.1.2. Gestion des Tâches (Todos)

Une fois connecté, vous arrivez sur votre tableau de bord personnel.

+-----+ Mon Compte [Bouton] Déconnexion +-----+	
[Saisir une nouvelle tâche...] [Ajouter]	
[] Acheter du pain [Corbeille]	
[x] Déployer sur AKS (Terminé) [Corbeille]	
+-----+	

8.1.2.1. Ajouter une tâche

1. Saisissez le libellé de votre tâche dans le champ de texte (ex: Acheter du pain).
2. Appuyez sur la touche **Entrée** de votre clavier ou cliquez sur le bouton **Ajouter**.
3. La tâche s'ajoute instantanément en haut de votre liste.

8.1.2.2. Marquer une tâche comme terminée (ou en cours)

1. Cochez la case (checkbox) située à gauche de la tâche.
2. Le texte de la tâche se barre pour indiquer sa complétion.
3. *Note* : Décocher la case rétablit le statut de la tâche en cours.

8.1.2.3. Supprimer une tâche

1. Cliquez sur le bouton **Corbeille** rouge situé à droite de la tâche.
2. La tâche est définitivement supprimée de votre liste.

8.1.3. Gestion du Compte & Conformité RGPD

Conformément à la réglementation sur la protection des données (RGPD), vous disposez d'un contrôle total sur vos informations personnelles.

8.1.3.1. Accéder aux paramètres

1. Cliquez sur le bouton **Mon Compte** (Account) situé en haut de l'interface.

8.1.3.2. Droit à l'oubli (Suppression définitive)

1. Sur la page de gestion du compte, cliquez sur le bouton rouge **Supprimer mon compte**.
2. **Attention** : Cette action est irréversible. Elle va supprimer immédiatement votre profil utilisateur ainsi que l'intégralité des tâches associées stockées en base de données.

8.1.4. Accessibilité et Raccourcis

L'application a été conçue pour être utilisable par tous :

- **Navigation au clavier** : Utilisez la touche **Tab** pour naviguer de manière ordonnée entre les champs de saisie, les cases à cocher et les boutons. Utilisez la touche **Espace** pour cocher/décocher une tâche et la touche **Entrée** pour valider les formulaires.
- **Lecteurs d'écran** : Tous les éléments interactifs possèdent des labels explicites (ex: les boutons de suppression ou de complétion d'icônes sont annoncés vocalement grâce aux attributs `aria-label`).

8.2. Manuel de Mise à Jour et Exploitation

8.2.1. Mise à jour du Code et des Dépendances (CI/CD)

- **Mise à jour locale des dépendances** :

```
npm update && npm run test
```

- **Mise à jour automatique par Dependabot** : Les analyses de sécurité hebdomadaires génèrent des Pull Requests. Si les vérifications de la CI sont au vert (statut de build vert), la PR peut être fusionnée directement.
- **Déclenchement du build d'images** : Créez et poussez un tag Git pour compiler et publier automatiquement la version :

```
git tag v2.1.0
git push origin v2.1.0
```

8.2.2. Déploiement et Migration (Kubernetes/Helm)

Le déploiement continu applique la nouvelle version de manière progressive sur le cluster AKS.

- **Mise à jour progressive (Rolling Update)** :

```
helm upgrade --install todo-app ./k8s/todo-app \
  --namespace production \
  -f ./k8s/todo-app/values-prod.yaml \
  --set client.image.tag="v2.1.0" \
  --set backend.image.tag="v2.1.0" \
  --set auth.image.tag="v2.1.0"
```

- **Migrations SQL Automatiques** : Les microservices exécutent leurs migrations Knex au démarrage (`await db.migrate.latest()`). Les modifications structurelles de la base sont appliquées automatiquement sans script SQL manuel.

8.2.3. Vérification & Procédure de Retour Arrière (Rollback)

- **Vérifier le statut du déploiement** :

```
kubectl rollout status deployment/backend -n production
```

- **Annuler la mise à jour (Rollback)** : Si une anomalie bloquante est détectée, restaurez instantanément la version précédente stable :

1. Listez l'historique des déploiements Helm :

```
helm history todo-app --namespace production
```

2. Restaurez la version précédente (ex: révision 12) :

```
helm rollback todo-app 12 --namespace production
```

8.3. Runbooks d'Exploitation (Fiches Réflexes)

8.3.1. Fiche 1 : Service indisponible (Erreur 502/504 - Pod Crash)

8.3.1.1. Diagnostic

Vérifiez l'état des pods Kubernetes et affichez les logs du service défaillant :

```
kubectl get pods
kubectl logs -f deployment/backend
## ou pour le service d'authentification
kubectl logs -f deployment/auth
```

8.3.1.2. Résolution (Redémarrage progressif)

Forcez un redémarrage progressif (Rolling Restart) pour recréer des instances saines :

```
kubectl rollout restart deployment/backend
kubectl rollout restart deployment/auth
kubectl rollout status deployment/backend
```

8.3.2. Fiche 2 : Restauration de la Base de Données (Azure MySQL PITR)

La base de données Azure MySQL Flexible Server effectue des sauvegardes automatiques avec une rétention de 7 jours.

8.3.2.1. Procédure de restauration (Point-in-Time Restore)

1. Identifiez l'horodatage cible (ex. 2026-06-24T18:00:00Z).
2. Lancer la restauration via Azure CLI (crée un nouveau serveur MySQL) :

```
az mysql flexible-server restore \
  --resource-group <NOM_RG> \
  --name <NOM_SERVEUR_RESTOURE> \
  --source-server <NOM_SERVEUR_ORIGINAL> \
  --restore-time "2026-06-24T18:00:00Z"
```

3. Déployez à nouveau avec Helm en modifiant l'hôte pour pointer vers la nouvelle base :

```
helm upgrade todo-app ./k8s/todo-app \
  --reuse-values \
  --set mysql.host="<NOM_SERVEUR_RESTOURE>.mysql.database.azure.com" \
  --set mysql.authHost="<NOM_SERVEUR_RESTOURE>.mysql.database.azure.com"
```

8.3.3. Fiche 3 : Montée en charge soudaine (Manual Scaling)

Si l'autoscaling dynamique (HPA) ne réagit pas assez vite, ajustez manuellement le nombre de réplicas :

```
## Augmenter à 3 instances
kubectl scale deployment/backend --replicas=3
kubectl scale deployment/auth --replicas=3
```

Pour pérenniser la configuration, modifiez les clés `replicaCount` dans le fichier `values-prod.yaml` pour le prochain déploiement.

8.4. Historique des Versions (Changelog)

Le suivi du cycle de vie du projet s'appuie sur une gestion sémantique des versions (Git Tagging). Le tableau ci-dessous retrace l'historique des versions majeures et mineures, montrant l'évolution continue du logiciel depuis un prototype simple local jusqu'à une architecture Cloud durcie et sécurisée.

Version	Date	Type de modification	Description des évolutions majeures
v1.0.0	12/04/2026	Logiciel (Prototype initial)	- Création du prototype initial (React SPA & Express Backend). - Stockage des données local (SQLite en fichier). - Pas d'authentification (toutes les tâches sont publiques).
v1.1.0	04/05/2026	Qualité & Structure	- Migration de l'ensemble du projet vers TypeScript (typage strict). - Refonte du backend selon le modèle en 3 couches avec injection de dépendances. - Développement du premier harnais de test unitaire Jest (couverture initiale).
v1.2.0	28/05/2026	Logiciel & RGPD	- Découpage en microservices (Client / Backend / Auth). - Création d'une base MySQL dédiée pour chaque service de données. - Implémentation du hachage de mots de passe (bcrypt) et de sessions JWT (HttpOnly). - Ajout des fonctions RGPD : export JSON (portabilité) et suppression (droit à l'oubli).
v2.0.0	22/06/2026	CI & Packaging Helm	- Création des Dockerfiles multi-stage (dev, test, prod). - Création du workflow d'intégration continue (CI) automatisé sur GitHub Actions. - Packaging Kubernetes à l'aide de Charts Helm unifiés.
v2.1.0	20/07/2026	Infrastructure & DevSecOps	- Version actuelle validée et fonctionnelle. - Provisionnement de l'infrastructure managée sur Azure (AKS, MySQL, Key Vault) via Terraform. - Durcissement Zero Trust (VNet MySQL privé, NetworkPolicies Kubernetes, Key Vault CSI Secrets Provider, conteneurs Non-Root). - Déploiement continu (CD) automatisé via OIDC.

Merci de votre lecture.

Candidat : Nicolas Walter
Projet : Todo App Cloud
Certification : Bloc 2 - RNCP39583

[Consulter le code source sur GitHub](#)

Ce document a été généré automatiquement à partir de la documentation interne en Markdown du projet (Approche *Documentation as Code*) et compilé avec le moteur de composition [Typst](#).