CdCT

Cahier des Charges Technique

Simulation de Stabilité des Structures Civiles par Apprentissage Profond

Projet académique – École Supérieure d'Ingénierie

Informations du document

Version : 1.0
Date : 10 novembre 2025
Statut : Validé
Confidentialité : Confidentiel

Notice de confidentialité

DOCUMENT CONFIDENTIEL

Ce document contient des informations confidentielles et propriétaires.
Il est strictement interdit de le diffuser sans autorisation écrite.

Maître d'ouvrage : École Supérieure d'Ingénierie

Historique des versions

rowgray**Version**	**Date**	**Auteur**	**Commentaires**
1.0	10/11/2025	Équipe Projet	Version initiale validée

Table des matières

1 Objet du document

Ce Cahier des Charges Technique (CdCT) a pour objectif de définir les choix technologiques, les architectures logicielles et matérielles, ainsi que les méthodes d'assurance qualité à mettre en œuvre pour le développement du projet *Simulation de Stabilité des Structures Civiles par Apprentissage Profond*.

Il complète le Cahier des Charges Fonctionnel (CdCF) et précise comment les exigences fonctionnelles seront réalisées techniquement.

2 Architecture technique globale

2.1 Schéma général

```
Application Mobile Flutter
HTTPS (REST JSON)
API Gateway - Spring Boot
Internal REST / gRPC
Service IA Python (PyTorch)
JDBC
Base de Données PostgreSQL

Stockage Fichiers / Backups / Monitoring
```

2.2 Description des composants

Composant	Rôle	Technologie	Communication
Application Mobile	Interface utilisateur	Flutter / Dart	HTTPS REST
API Backend	Logique métier et sécurité	Spring Boot 3 (Java 17)	REST JSON
Moteur IA	Inférence IA et prétraitement	Python 3.10 / PyTorch	REST interne
Base de Données	Persistance	PostgreSQL 16	JPA / Hibernate
Stockage	Backups et journaux	MinIO / AWS S3	HTTPS
CI/CD	Intégration & déploiement	GitHub Actions / Jenkins	SSH / API
Visualisation	3D interactive	Flutter + three_dart	Local

3 Architecture logicielle

3.1 Backend (Spring Boot)

— Langage : Java 17
— Framework : Spring Boot 3.x
— Sécurité : Spring Security + JWT
— Base de données : PostgreSQL via Hibernate
— API REST : Swagger / OpenAPI 3.0
— Tests : JUnit 5, Mockito, Testcontainers

3.1.1 Structure du code

```
com.simstruct
    config/          → Sécurité, CORS, JWT, Swagger
    controllers/     → Endpoints REST
    services/        → Logique métier
    repositories/    → Accès BD
    models/          → Entités / DTOs
    ai/              → Service d'inférence IA
    logs/            → Audit et traçabilité
```

3.2 Moteur IA (Python)

— Langage : Python 3.10
— Frameworks IA : PyTorch / TensorFlow
— API exposée : FastAPI (REST JSON)
— Gestion des modèles : MLflow / DVC
— Monitoring IA : Prometheus exporter

3.2.1 Pipeline IA

1. Prétraitement des entrées (normalisation, unité SI)
2. Inférence via modèle .pt
3. Post-traitement (stress, déformation, verdict)
4. Renvoi résultat JSON vers Backend
5. Sauvegarde logs IA + métriques

3.3 Application Mobile (Flutter)

— Langage : Dart 3.x
— Framework : Flutter 3.24
— Architecture : Provider / Riverpod
— 3D Rendering : three_dart, flutter_3d_obj
— API client : Dio (HTTPS, JWT)
— Stockage local : Hive / SecureStorage
— Tests : flutter_test, mockito

3.3.1 Structure Flutter

```
lib/
    screens/
        login.dart
        dashboard.dart
        parameters_form.dart
        results_3d.dart
        profile.dart
    models/
    services/
    providers/
    widgets/
```

4 Infrastructure et déploiement

Environnement	Objectif	Hébergement
DEV	Développement local	Docker Compose
STAGING	Tests intégrés	AWS / GCP
PROD	Production	Kubernetes (EKS / GKE)

4.1 Conteneurisation

— Backend → openjdk :17-jdk
— IA → python :3.10-slim
— PostgreSQL → postgres :16
— Docker Compose pour intégration locale
— Kubernetes pour déploiement (auto-scaling, secrets, config maps)

4.2 CI/CD

— CI : GitHub Actions (build, test, sonar, docker push)
— CD : Jenkins / ArgoCD (staging → prod)
— Déploiement Blue-Green + rollback auto

5 Sécurité

5.1 Authentification & Autorisation

— JWT access token (24h) + refresh (7j)
— Stockage sécurisé (SecureStorage / HTTPOnly Cookie)
— Rôles : USER / ADMIN

5.2 Sécurisation API

— TLS 1.3
— Rate Limiting (Spring filter)
— Anti-CSRF
— Headers CSP, X-XSS-Protection
— Hash mots de passe : BCrypt (cost 12)
— Secrets stockés dans AWS Secrets Manager

5.3 Sécurité du Code

— Analyse SonarQube : 0 vulnérabilité critique
— Tests OWASP ZAP / BurpSuite
— Audit semestriel

6 Données et stockage

6.1 Schéma de données

— `users` : comptes et rôles
— `simulations` : métadonnées
— `structure_parameters` : paramètres structurels (JSONB)
— `prediction_results` : résultats IA
— `audit_logs` : journalisation

6.2 Sauvegarde

— Sauvegarde quotidienne (dump + snapshot)
— Restauration testée chaque trimestre
— Réplication read-replica
— RPO 24h / RTO 4h

7 Supervision et monitoring

— Logs : ELK Stack (Elasticsearch, Logstash, Kibana)
— Métriques : Prometheus + Grafana
— Alertes : CPU $> 85\%$, Latence > 3s, Erreur $> 5\%$
— Audit : journal complet des connexions, simulations, suppressions

8 Tests techniques

Type	Outil	Objectif	Seuil
Unitaires	JUnit / Flutter Test	Vérifier logique	80%
Intégration	Postman / Testcontainers	API REST	100% OK
Charge	JMeter / K6	100 users simultanés	$<$ 3s p90
IA	PyTest	Écart FEM $<$ 10%	OK
Sécurité	OWASP ZAP	0 faille critique	OK
UAT	Beta interne	Parcours complet	100% OK

9 Maintenance technique

— Mises à jour dépendances → mensuel
— Patch sécurité → < 48h
— Revue logs erreurs → hebdo
— Tests backup/restore → trimestriel
— CI/CD → redéploiement automatique

10 Livrables techniques

— Code source : GitHub (privé)
— Images Docker : backend, IA, DB
— Modèle IA exporté (.pt)
— Documentation API (Swagger)
— Schémas UML (classes, déploiement, séquence)
— Rapport SonarQube
— Scripts SQL initiaux
— Plan de monitoring Grafana

11 Validation et recette

rowgray**Étape**	**Responsable**	**Validation**	
Architecture	Responsable technique		
Code / Intégration	Développeurs		
Sécurité / Performance	Ingénieur qualité		
IA / Prédiction	Data Scientist		
Livraison finale	Chef de Projet		

12 Management de la qualité

12.1 Objectif

Garantir la conformité du produit final avec les exigences du CdCF, en respectant les normes ISO 9001 / ISO 25010 (qualité logicielle).

12.2 Politique Qualité

— Couverture tests 80%
— 0 vulnérabilité critique
— Revue code obligatoire
— Documentation complète et validée
— Performance API < 3s p90

12.3 Organisation

Rôle	**Responsabilité**
Responsable Qualité	Définit et contrôle la politique qualité
Chef de Projet	Suivi planning et conformité livrables
Développeurs	Respect des normes et bonnes pratiques
Auditeur externe	Vérifie conformité ISO et sécurité

12.4 Contrôle Qualité

Contrôle	**Outil**	**Fréquence**
Code Review	GitHub PR	Chaque commit
Analyse statique	SonarQube	CI
Sécurité	OWASP ZAP	Mensuel
Audit interne	Rapport qualité	Trimestriel

12.5 Indicateurs (KPIs)

— Couverture test : 80%
— Temps correction bug critique : 24h
— Satisfaction utilisateur : 90%
— Disponibilité API : 99.5%

12.6 Processus d'Amélioration Continue

— Revue de sprint (Scrum Retrospective)
— Analyse des incidents / anomalies
— Plan d'action qualité mis à jour mensuellement

12.7 Validation Qualité Finale

— Tous les tests unitaires et intégration réussis
— Audit sécurité validé
— Documentation à jour
— Rapport qualité signé

13 Validation finale

Le présent Cahier des Charges Technique est approuvé par :

Chef de Projet	Responsable technique
Responsable qualité	Maître d'ouvrage

14 Conclusion

Ce Cahier des Charges Technique constitue la référence officielle pour la conception, la mise en œuvre, la qualité et la maintenance du projet *Simulation de Stabilité des Structures Civiles par Apprentissage Profond*. Il garantit la cohérence entre les exigences fonctionnelles et les choix technologiques, la sécurité, la qualité du code, et la traçabilité complète du développement.