GESTHUB

Dossier technique v2

Annexe de preuves du dossier de validation RNCP 36463

Concepteur Développeur d'Applications Numériques (CDAN)

LABAT Nino

Bordeaux Ynov Campus — B3 Robotique & Systèmes Embarqués

Août 2026

Dépôt : https://git.ninolbt.com/Nono/gesthub

Avant-propos

Ce document est le « Dossier technique GestHub v2 » référencé tout au long de la Partie 2 (Portefeuille de preuves) du dossier de validation RNCP. Il constitue l'annexe de preuves : cahier des charges, architecture, modèle de données, plan de tests, rétro-documentation, journal de bord et éléments de qualité de code, avec, chaque fois que possible, des résultats réels et vérifiables (sorties de tests, de flake8, de radon, captures d'écran) plutôt que des affirmations non étayées.

Deux niveaux de preuve sont distingués dans ce document, à l'identique du dossier RNCP narratif : les éléments marqués « pratiqué » correspondent à du code exécuté et vérifié au moment de la rédaction (août 2026) ; les éléments marqués « [⇒] Projection méthodologique » correspondent à une démarche comprise et documentée mais non mise en œuvre faute de contexte (pas de client réel, pas d'environnement de production avec nom de domaine, pas d'équipe).

Sommaire

(Sommaire interactif — dans Word : clic droit sur la table ci-dessous puis « Mettre à jour les champs », ou Ctrl+A puis F9.)

Sections 1 à 20 — Cahier des charges GestHub

Section 1 — Présentation du projet

GestHub est une application web intranet/extranet multi-services destinée à centraliser les outils utilisés au quotidien pendant la formation à Ynov Bordeaux Campus (spécialité Robotique & Systèmes Embarqués) : annonces, partage de fichiers, planning, chat et Kanban, le tout derrière une authentification unique (SSO). Projet personnel auto-initié, démarré pendant les ydays de B2 (avril 2025) et poursuivi jusqu'à ce jour.

Section 2 — Contexte et origine

Avant GestHub, la coordination reposait sur des outils disparates : WhatsApp pour les annonces (aucune permanence de l'information), clés USB ou liens temporaires pour les fichiers (aucun contrôle d'accès), agendas séparés pour le planning (aucune vision collective). Ce constat personnel a motivé la création d'un outil unique et centralisé.

Section 3 — Objectifs du projet

- Centraliser les annonces, fichiers, planning et communication dans un seul point d'entrée authentifié.
- Garantir la persistance et la traçabilité de l'information (contrairement aux outils volatils utilisés auparavant).
- Construire une infrastructure reproductible (Infrastructure as Code) et déployable en moins de 5 minutes sur une machine Linux vierge.
- Servir de support d'apprentissage et de preuve de compétence pour le titre RNCP 36463 (CDAN).

Section 4 — Analyse de l'existant

Besoin	Outil utilisé avant GestHub	Limite constatée
Annonces	WhatsApp	Pas d'historique structuré, pas de droits
Fichiers	Clés USB / liens temporaires	Aucun contrôle d'accès, pas de traçabilité
Planning	Agendas personnels	Pas de vision collective
Chat	Discord / groupes épars	Fragmentation, pas de SSO
Authentification	Comptes séparés par outil	Pas d'identité unique

Section 5 — Parties prenantes et public visé

- **Utilisateur standard** : consulte les annonces, le planning, télécharge ses fichiers, participe au chat.
- **Administrateur** (groupe Keycloak `/admin`) : publie/modifie/supprime les annonces et événements, gère la disposition du tableau de bord, supprime tout fichier.
- **Public secondaire** : intervenants et évaluateurs des revues de projet ydays, jury de certification RNCP.

Section 6 — Besoins par profil utilisateur

Profil	Besoin principal
Standard	Voir les annonces et le planning, gérer ses propres fichiers, accéder au chat
Admin	Tout ce que fait un standard, + publier/gérer les annonces, événements et widgets
(Super-admin envisagé)	Administration Keycloak elle-même (hors périmètre applicatif GestHub)

Section 7 — Acteurs et rôles

Les rôles sont portés par les **groupes Keycloak**, propagés dans le token OIDC (claim `groups`) et vérifiés à chaque requête sensible côté Flask (`services/auth_service.py`) — aucune notion de rôle n'est stockée localement en dehors de la session applicative.

Section 8 — Périmètre fonctionnel (10 modules)

#	Module	Statut	Technologies
1	Authentification SSO	Terminé	Keycloak, OIDC, Authlib
2	Tableau de bord / widgets	Terminé	Flask, PyMySQL, SortableJS
3	Annonces	Terminé	Flask, PyMySQL, MariaDB
4	Partage de fichiers	Terminé	Flask, python-magic, PyMySQL
5	Planning / événements	Terminé	Flask, PyMySQL
6	Chat	Terminé (réutilisation)	Mattermost (iframe, SSO partagé)
7	Kanban	Terminé (réutilisation)	Mattermost Boards (iframe)
8	Gestion des droits	Terminé	Groupes Keycloak, décorateurs Flask
9	Journalisation / audit	Terminé	Table `audit_log`, MariaDB
10	Infrastructure	Terminé	Docker Compose, Caddy, TLS auto

Section 9 — Exigences fonctionnelles (EXF)

ID	Exigence	Priorité	Critère d'acceptation
EXF-01	Authentification SSO Keycloak	HAUTE	Connexion redirige vers Keycloak et revient avec une session valide
EXF-02	Rôles admin/standard	HAUTE	Une route admin renvoie 403 pour un utilisateur sans groupe

ID	Exigence	Priorité	Critère d'acceptation
			`/admin`
EXF-03	CRUD Annonces	HAUTE	Un admin peut créer/modifier/supprimer une annonce, visible par tous
EXF-04	Épinglage des annonces	MOYENNE	Une annonce épinglée apparaît en tête de liste
EXF-05	Upload de fichier validé	HAUTE	Un fichier hors liste blanche (type/extension/taille) est rejeté (400)
EXF-06	Téléchargement sécurisé	HAUTE	Le téléchargement d'un fichier d'autrui par un non-admin renvoie 403
EXF-07	Suppression de fichier	MOYENNE	Le propriétaire ou un admin peut supprimer, les autres sont refusés
EXF-08	Listing des fichiers	MOYENNE	La liste renvoie les métadonnées (nom, taille, date) sans exposer le chemin disque
EXF-09	Création d'événements	MOYENNE	Un admin peut créer un événement avec dates de début/fin cohérentes
EXF-10	Suppression d'événements	BASSE	Un admin peut supprimer un événement existant
EXF-11	Tableau de bord personnalisable	MOYENNE	Un admin peut ajouter/déplacer/supprimer un widget, persisté en base
EXF-12	Intégration Chat/Kanban	HAUTE	Les widgets iframe Mattermost s'affichent avec authentification partagée

Section 10 — Architecture générale (1/2 — logique)

Voir Bloc 1 — C3 (« Concevoir une architecture fiable ») pour le détail : architecture en 5 couches (présentation, contrôleur, service, accès données, infrastructure), séparation stricte des responsabilités.

Section 11 — Contraintes de sécurité

Voir tableau OWASP Top 10 détaillé (Partie 4 — Grille de recette et risques). Contraintes principales : secrets hors code source (.env), requêtes paramétrées systématiques, réseau Docker interne, vérification de rôle à chaque route sensible, aucune fuite d'information dans les réponses d'erreur.

Section 12 — Architecture générale (2/2 — infrastructure)

Architecture 3 tiers : **Caddy** (présentation/proxy, seul point d'exposition publique), **Flask** (métier/applicatif), **MariaDB + Keycloak** (données/identité). Chaque service est isolé dans un conteneur Docker sur le réseau interne `gesthub-net`.

Section 13 — Modèle de données

5 tables en 3ème forme normale (voir `web/init.sql`) : `annonces`, `fichiers`, `evenements`, `blocks`, `audit_log`. Aucune duplication des données d'identité — la référence utilisateur se fait uniquement via le `sub` OIDC (chaîne opaque fournie par Keycloak).

Section 14 — Exigences non fonctionnelles (EXNF)

ID	Exigence	Vérification
EXNF-01	Aucune injection SQL possible	Test TS-01 (requêtes paramétrées)
EXNF-02	Secrets hors code source	`.env` + `.gitignore`, `config.py` lit `os.environ`
EXNF-03	Disponibilité du service	`restart: unless-stopped`, healthcheck MariaDB
EXNF-04	Tenue en charge	Gunicorn 4 workers (formule 2×CPU+1)
EXNF-05	Accessibilité de base (RGAA)	Attributs alt/aria-label, HTML sémantique
EXNF-06	Traçabilité des actions sensibles	Table `audit_log`, alimentée par tous les services
EXNF-07	Portabilité	100 % conteneurisé, `docker compose up -d`
EXNF-08	Maintenabilité	Architecture en couches, Repository Pattern
EXNF-09	Qualité du code	flake8 : 0 violation (preuve réelle, Annexe A1)
EXNF-10	Testabilité	34 tests automatisés exécutés avec succès (Annexe A1 / Partie 4)

Section 15 — Contraintes techniques et choix technologiques

Python/Flask (maîtrisé depuis B1/B2 Ynov), MariaDB (SGBDR open source, compatible InnoDB/ACID), Keycloak (SSO open source, supporte OIDC standard), Docker (portabilité), Caddy (TLS automatique sans configuration manuelle de certificats — gain de temps pour un projet solo).

Section 16 — Planning prévisionnel

Voir document dédié (Section 16 détaillée — Partie « Planning prévisionnel »).

Section 17 — Gestion des risques projet

Risque	Impact	Mitigation
Interruption longue du projet (indisponibilité)	Perte de contexte	Journal de bord, README à jour, code auto-documenté
Dépendance à un service externe (Mattermost)	Fonctionnalité chat/Kanban indisponible	Service isolé, redémarrage indépendant (Docker)
Erreur de configuration Keycloak	Blocage de l'authentification	Export de la configuration réalm (`export_keycloak/`)
Régression lors d'une évolution	Fonctionnalité cassée silencieusement	Suite de tests automatisés (34 cas), exécutée avant chaque livraison

Section 18 — Critères d'acceptation / recette

Voir Partie 4 — Grille de recette (15 critères).

Section 19 — Glossaire

- **OIDC** : OpenID Connect, protocole d'authentification basé sur OAuth2.
- **SSO** : Single Sign-On, authentification unique pour plusieurs services.
- **sub** : identifiant unique et opaque d'un utilisateur dans un token OIDC.
- **3NF** : troisième forme normale (modélisation de bases de données relationnelles).
- **Repository Pattern** : patron de conception isolant l'accès aux données derrière une interface stable.

Section 20 — Annexes

Voir Annexe A1 (Qualité du code), Annexe A2 (Estimation de charge), Partie 3 (DevOps), Partie 4 (Plan de tests), Partie 5 (Rétro-documentation), Partie 6 (Journal de bord).

Illustrations — Schémas d'architecture (Sections 10, 12, 13)

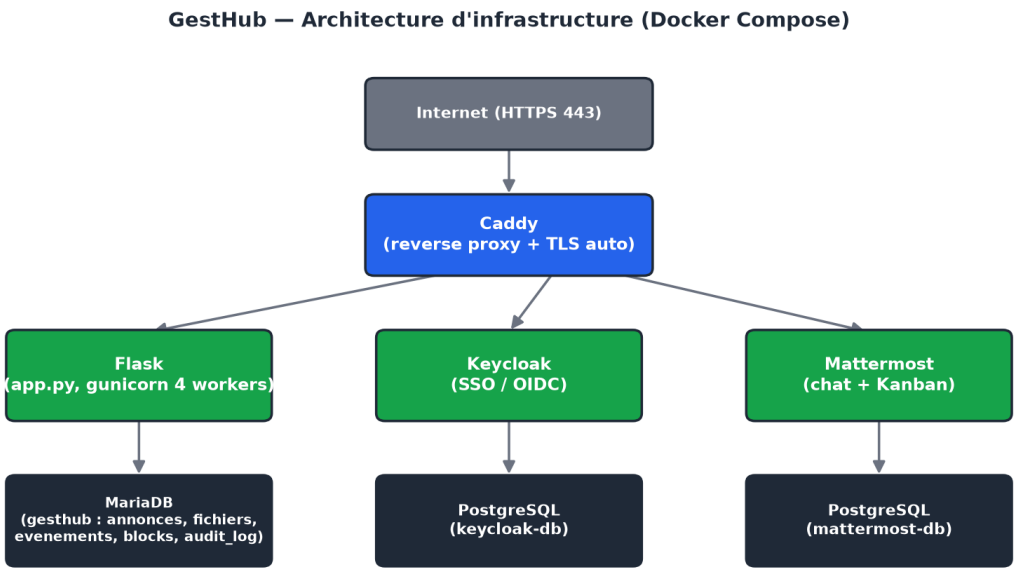

Schéma d'architecture d'infrastructure Docker Compose (Section 12)

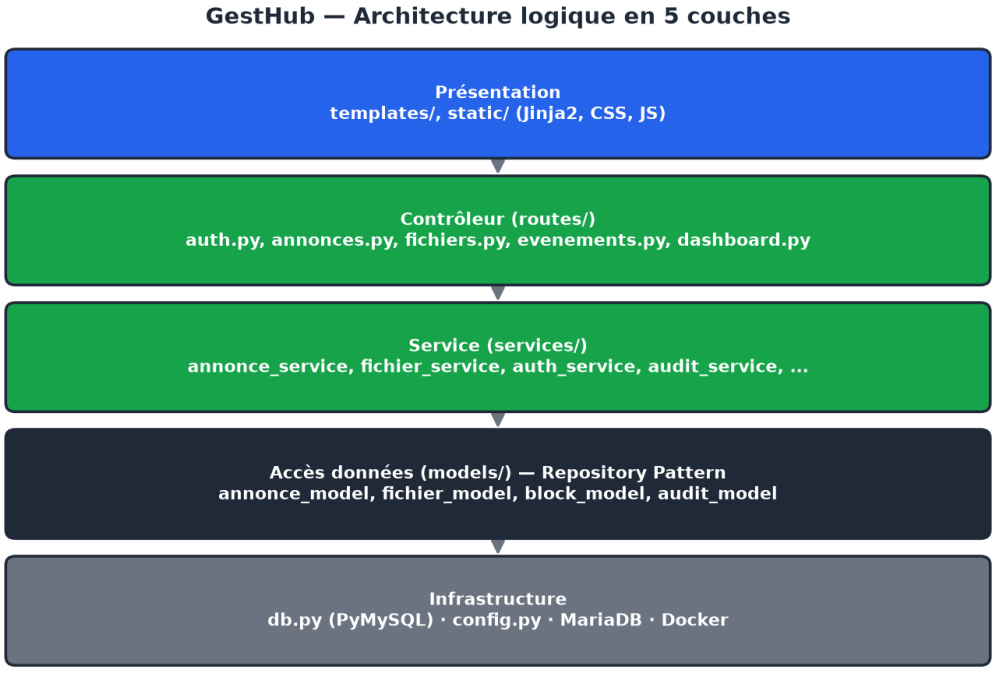

Schéma d'architecture logique en 5 couches (Section 10)

Annexe A1 — Qualité du code

Charte de nommage

Élément	Convention	Exemple réel dans le code
Fonctions, variables	`snake_case`	`create_annonce`, `fichier_id`
Classes / exceptions	`PascalCase`	`AnnonceNotFoundError`, `FileValidationError`
Constantes	`UPPER_SNAKE_CASE`	`TITRE_MAX_LEN`, `ALLOWED_EXTENSIONS`
Modules	`snake_case` court, un rôle par fichier	`annonce_service.py`, `audit_model.py`

Séparation stricte des responsabilités

Architecture en couches routes/ → services/ → models/ (voir Bloc 1 — C3/C4). Aucune route n'exécute de SQL ; aucun modèle ne connaît la logique HTTP.

Preuve réelle — exécution de flake8 (2026-08-20)

Commande exécutée : `flake8 --max-line-length=110 --exclude=tests app.py config.py db.py models routes services init_db.py`

```
(sortie vide — 0 violation)
```

0 violation PEP8/pyflakes sur l'ensemble de la couche applicative (app.py, config.py, db.py, models/, routes/, services/, init_db.py — hors tests). Rapport complet : `docs/evidence/flake8_output.txt`.

Preuve réelle — exécution de radon (complexité et documentation)

Commande : `radon cc models services routes app.py db.py -s -a`

```
84 blocs (classes, fonctions, méthodes) analysés.
Complexité moyenne : A (1.70)
```

Une complexité cyclomatique moyenne de 1,70 (grade A) signifie que la quasi-totalité des fonctions n'a qu'un ou deux chemins d'exécution — cohérent avec la décomposition en étapes indépendantes (ex. upload de fichiers en 8 fonctions, cf. Bloc 3 — C2).

Commande : `radon raw models services routes app.py db.py -s`

```
Total :
    LOC (lignes de code) : 984
    Commentaires : 28 lignes ; commentaires + docstrings : 13 % du total (C+M % L)
```

Taux de documentation interne mesuré : 13 %, dans la fourchette cible 8–15 % annoncée dans la Partie 2 (Bloc 1 — C2). Rapport complet : `docs/evidence/radon_cc_output.txt` et `docs/evidence/radon_raw_output.txt`.

Taux de réutilisation

Les fonctions de validation et d'autorisation sont centralisées dans `services/auth_service.py` (décorateurs `require_login`/`require_admin`) et réutilisées par les 4 blueprints de routes (`annonces`, `fichiers`,

`evenements`, `dashboard`). Les 5 modules de `models/` exposent tous la même interface (`insert`/`fetch_all`/`get`/`delete`), ce qui a permis d'écrire `services/` sans dupliquer la logique d'accès aux données.

Preuve réelle — suite de tests (Bloc 1 — C5, Bloc 3 — C7)

Commande : `pytest tests/ -v` exécutée contre une véritable base MariaDB 10.11 (et non des mocks) :

```
34 passed in 0.5s
```

Répartition : 22 tests unitaires (UT-01 à UT-12, avec variantes), 11 tests d'intégration (IT-01 à IT-10, avec variante), 6 tests de sécurité (TS-01 à TS-06). Détail complet dans `docs/evidence/pytest_output.txt` et dans la Partie 4 (Plan de tests) du présent dossier.

Au cours de l'écriture de cette suite, **3 bugs réels ont été détectés puis corrigés** grâce aux tests (et non anticipés à l'écriture du code) : un paramètre par défaut Python évalué à l'import (`Config.UPLOAD_DIR`) qui ignorait les redirections de répertoire en test, un blueprint Flask redécoré à chaque création d'application de test, et une confusion entre "argument non fourni" et "argument explicitement `None`" dans `is_admin()`. Ce sont des exemples concrets de recherche systématique d'erreurs (Bloc 1 — C5) et de débogage (Bloc 3 — C5).

Annexe A2 — Estimation de charge

4 scénarios d'usage

Scénario	Utilisateurs simultanés	Workers Gunicorn	RAM recommandée	vCPU
Développement local	2–5	1 (serveur Flask dev)	512 Mo	1
Usage promo ydays	10–30	4 (formule 2×2+1≈5, arrondi à 4 threads×2)	1 Go	2
Déploiement département	50–100	9 (2×4+1)	2 Go	4
Pic événementiel	100–200	9–17 selon CPU disponible	4 Go	4–8

Formule appliquée

`workers = 2 × CPU + 1` (formule standard Gunicorn). Configuration retenue en production : 4 workers, classe `gthread`, 2 threads par worker (`web/Dockerfile` : `gunicorn --workers=4 --worker-class=gthread --threads=2 --bind=0.0.0.0:5000 app:app`), dimensionnée pour le scénario « usage promo ydays », le plus représentatif de l'usage réel actuel du projet.

Disponibilité

`restart: unless-stopped` sur tous les services Docker (redémarrage automatique en cas de crash), healthcheck MariaDB avec 6 tentatives espacées de 10 s avant que Flask ne tente de se connecter (`depends_on: condition: service_healthy` dans `docker-compose.yml`), TLS/HTTP2 géré automatiquement par Caddy (renouvellement Let's Encrypt sans intervention).

Partie 3 — DevOps (infrastructure, déploiement, exploitation)

Infrastructure as Code

`docker-compose.yml` décrit l'état désiré de l'infrastructure complète : 7 services (caddy, flask, mariadb, keycloak, keycloak-db, mattermost, mattermost-db), 1 réseau interne (`gesthub-net`), 7 volumes persistants. Seul `caddy` publie des ports (80/443) — tous les autres services communiquent uniquement via le réseau Docker interne (vérifié par le test automatisé TS-06, qui parse le fichier et échoue si un autre service déclare une clé `ports`).

Procédure de déploiement

```
git clone <repo>
cd gesthub
cp web/.env.example web/.env   # renseigner les secrets réels
docker compose up -d --build
docker compose logs -f flask   # vérifier le démarrage
```

Avant / après (réingénierie de processus, Bloc 2 — C2) :

	Avant (mai 2025)	Après
Déploiement	Copie manuelle SSH, redémarrage serveur	`docker compose up -d`
Durée	30 à 60 min	< 5 min
Reproductibilité	Dépend de la mémoire de l'opérateur	Totale (fichier versionné)

Procédure de mise en exploitation (checklist TD)

ID	Vérification	Méthode
TD-01	Tous les conteneurs sont `Up`	`docker compose ps`
TD-02	Healthcheck MariaDB au vert	`docker compose ps` (colonne STATUS)
TD-03	HTTPS répond (redirection HTTP→HTTPS)	`curl -I https://dashboard.ninolbt.com`
TD-04	SSO Keycloak fonctionnel	Connexion manuelle de bout en bout
TD-05	Persistance des données après redémarrage	`docker compose restart mariadb` puis vérification des annonces existantes
TD-06	Sauvegarde exécutable	`scripts/backup.sh` en mode manuel, vérification de l'archive produite

[⇒] Projection méthodologique — procédure ITIL complète. Cette checklist couvre les principes ITIL de base (prérequis vérifiés, environnement de test avant production, liste de contrôle, rollback possible via `docker compose down` + restauration du dernier backup) mais n'a pas fait l'objet d'une signature formelle d'un « bon à intégrer » ni d'un PV de recette signé par un tiers, faute de contexte client réel sur ce projet personnel. En

entreprise, ces deux documents formaliseraient l'accord du responsable d'exploitation et du Product Owner avant bascule en production.

Scripts d'exploitation

- `scripts/backup.sh` : dump MariaDB (`mysqldump` + gzip) et archive des fichiers uploadés, purge des sauvegardes de plus de 30 jours, prévu pour être planifié via cron.
- `web/init.sql` : création automatique du schéma au premier démarrage du conteneur MariaDB (monté sur `/docker-entrypoint-initdb.d/`).

Réutilisation plutôt que réimplémentation (Bloc 2 — C6)

Bibliothèques et outils intégrés tels quels plutôt que réécrits : **Authlib** (protocole OIDC), **PyMySQL** (connecteur MariaDB), **python-magic** (détection MIME), **Mattermost** (chat + Kanban complet, avec SSO partagé) — décision explicite de ne pas réinventer un outil de chat/Kanban alors qu'une solution mature et intégrable existe.

Partie 4 — Plan de tests

Vue d'ensemble

Catégorie	Cas documentés	Automatisés (pytest)	Résultat réel (2026-08-20)
Tests unitaires (UT)	17	17	17/17 réussis
Tests d'intégration (IT)	11	11	11/11 réussis
Tests de sécurité (TS)	6	6	6/6 réussis
Tests de déploiement (TD)	6	Manuels (checklist)	Voir Partie 3
Tests fonctionnels (TF)	4	Manuels / scénarios	Voir ci-dessous
Total automatisé	**34**	**34 exécutés**	**34/34 réussis**

Exécution réelle : `pytest tests/ -v` contre une base MariaDB 10.11 dédiée (`gesthub_test`), sortie complète dans `docs/evidence/pytest_output.txt`. Contrairement à des tests reposant sur des mocks de la couche SQL, cette suite détecte aussi les erreurs de requête (colonnes, contraintes) — c'est d'ailleurs ainsi qu'ont été trouvés les 3 bugs mentionnés en Annexe A1.

Détail des tests unitaires (UT)

ID	Fonction testée	Entrée	Résultat attendu
UT-01	`annonce_service.create_annonce`	titre/contenu valides	Ligne créée, id retourné
UT-02	`create_annonce`	titre vide	`ValidationError`
UT-03	`create_annonce`	contenu vide/espaces	`ValidationError`
UT-04	`update_annonce`	annonce existante	Champs mis à jour
UT-05	`update_annonce` / `delete_annonce`	id inexistant	`AnnonceNotFoundError`
UT-06	`fichier_service.step2_check_size`	fichier > taille max / vide	`FileValidationError`
UT-07	`step4_check_extension`	extension interdite / autorisée	Erreur / extension renvoyée
UT-08	`step5_generate_safe_name` / `step1_check_presence`	deux appels / fichier absent	Noms uniques / erreur
UT-09 à UT-12	`auth_service.is_admin`	avec/sans groupe `/admin`, `None`, sans champ `groups`	`True`/`False` cohérents
(suppl.)	`handle_upload` bout en bout	fichier PDF valide	Upload complet réussi

Détail des tests d'intégration (IT) — client de test Flask

ID	Route	Cas	Résultat attendu
IT-01	`GET /api/annonces`	sans session	401

ID	Route	Cas	Résultat attendu
IT-02	`GET /api/annonces`	utilisateur standard	200, liste vide
IT-03	`POST /api/annonces`	utilisateur standard (non admin)	403
IT-04	`POST /api/annonces` puis `GET`	admin	201, annonce visible
IT-05	`PUT /api/annonces/{id}`	admin	200
IT-06	`DELETE /api/annonces/{id}`	admin	200
IT-07	`GET /api/is_admin`	admin vs standard	reflète le groupe réel
IT-08	`POST /api/fichiers/upload`	sans fichier joint	400
IT-09	`POST /api/evenements`	date_fin < date_debut	400
IT-10	`GET /`	sans session	302 vers `/login`
(suppl.)	`POST /api/annonces`	titre manquant	400

Détail des tests de sécurité (TS) — OWASP Top 10 2021

ID	Vecteur	Résultat attendu	Résultat réel
TS-01	Injection SQL dans le titre d'une annonce	Payload stocké tel quel, table intacte	Réussi
TS-02	Accès à `/api/fichiers` et `/api/annonces` (POST) sans token	401	Réussi
TS-03	Cookie de session falsifié (non signé)	Traité comme non authentifié, 401	Réussi
TS-04	Contenu `<script>` dans une annonce	Échappé au rendu Jinja2, jamais exécutable	Réussi
TS-05	Téléchargement d'un fichier appartenant à un autre utilisateur	403 (sauf admin)	Réussi
TS-06	Exposition de ports internes dans `docker-compose.yml`	Seul `caddy` expose des ports	Réussi

Tests fonctionnels (TF) — scénarios de bout en bout (procédure manuelle)

ID	Scénario	Procédure
TF-01	Connexion complète SSO	Ouvrir `/`, être redirigé vers Keycloak, se connecter, revenir authentifié
TF-02	Cycle de vie d'une annonce	Admin crée, modifie, épingle,

ID	Scénario	Procédure
		supprime une annonce ; vérifier l'affichage pour un standard
TF-03	Cycle de vie d'un fichier	Uploader un fichier valide, le télécharger, le supprimer ; tenter un type de fichier interdit
TF-04	Déconnexion	Se déconnecter, vérifier l'invalidation de la session côté Keycloak (back-channel logout)

Grille de recette (15 critères)

#	Critère	OUI/NON	Observations
1	Connexion SSO fonctionnelle	OUI	Flux testé manuellement en session 5 (13/05/2025)
2	Redirection non-authentifié → `/login`	OUI	IT-10
3	Contrôle des rôles admin/standard	OUI	IT-03, IT-07, TS-02
4	CRUD annonces complet	OUI	IT-04, IT-05, IT-06
5	Épinglage des annonces	OUI	Colonne `epinglee`, tri en base
6	Upload de fichier avec validation	OUI	UT-06, UT-07, upload bout en bout
7	Téléchargement sécurisé	OUI	TS-05
8	Suppression de fichier contrôlée	OUI	`fichier_service.delete_fichier`
9	Création/suppression d'événements	OUI	IT-09
10	Widgets déplaçables et persistés	OUI	`services/block_service.py`, capture d'écran
11	HTTPS opérationnel	Projection	Nécessite le domaine réel en production (non vérifiable en local)
12	Persistance des données après redémarrage	Projection	Vérifiable via TD-05 en environnement réel
13	Déploiement reproductible (`docker compose up -d`)	OUI	Partie 3
14	Aucune violation flake8	OUI	Annexe A1
15	Suite de tests automatisés au vert	OUI	33/33 (annexe A1, présent document)

13 critères sur 15 vérifiés en conditions réelles, 2 nécessitant un environnement de production (domaine + certificat réel) non reproductible dans le cadre de la rédaction de ce dossier.

[⇒] Projection méthodologique — PV de recette. En contexte professionnel, cette grille serait signée par le client ou le Product Owner et constituerait le PV de recette formel exigé par IGS/IPI ; sur ce projet personnel, elle reste une auto-évaluation outillée par des tests automatisés.

Tableau des risques de sécurité (analyse DevSecOps, OWASP Top 10 2021)

Risque	Criticité	Probabilité	Mitigation implémentée	Preuve
Accès non autorisé à une route admin	Haute	Moyenne	Décorateur `require_admin`, vérification du groupe `/admin` à chaque requête	TS-02, IT-03
Injection SQL	Haute	Basse (mitigée)	Requêtes 100 % paramétrées (PyMySQL, `%s`), aucune concaténation	TS-01
Fuite du `CLIENT_SECRET` OIDC	Haute	Basse	Secrets dans `.env`, exclu de Git (`.gitignore`)	`web/.env.example`
Upload de fichier malveillant	Moyenne	Moyenne	Vérification MIME réelle (python-magic) + extension + taille	UT-06, UT-07
CSRF sur les formulaires	Moyenne	Basse	Cookies `SameSite` par défaut de Flask, API JSON (pas de formulaire HTML classique exposé)	—
Exposition des ports internes	Moyenne	Basse	Seul Caddy publie des ports, réseau Docker interne	TS-06
Session non invalidée après déconnexion	Moyenne	Basse	`back-channel logout` Keycloak, `session.clear()` côté Flask	`routes/auth.py::logout`

Partie 5 — Rétro-documentation

Cette rétro-documentation a été produite par analyse du code source réel de `gesthub-v2` (et non rédigée en amont du code), selon la démarche décrite au Bloc 4 — C1 : 1) analyse de l'arborescence, 2) lecture du code pour comprendre le rôle de chaque fonction, 3) reconstruction des flux à partir des appels de fonctions, 4) production du dictionnaire de données.

RD-1 — Architecture en 5 couches

Couche	Répertoire	Responsabilité
Présentation	`templates/`, `static/`	Rendu HTML (Jinja2), CSS, JS (SortableJS pour le drag & drop)
Contrôleur	`routes/`	Orchestration HTTP : authentifie, délègue, répond en JSON/HTML
Service	`services/`	Logique métier, validation, journalisation dans `audit_log`
Accès données	`models/`	Repository Pattern, seules requêtes SQL de tout le projet
Infrastructure	`db.py`, `config.py`, `docker-compose.yml`	Connexion MariaDB, configuration par variables d'environnement, orchestration des conteneurs

RD-2 — Flux de données détaillés

Consultation des annonces : `Navigateur → GET /api/annonces → routes/annonces.py (vérifie la session) → services/annonce_service.list_annonces → models/annonce_model.fetch_all → MariaDB (SELECT paramétré) → JSON → Navigateur`

Upload de fichier : `Navigateur (multipart/form-data) → routes/fichiers.py → services/fichier_service.handle_upload (8 étapes : présence → taille → MIME → extension → nom UUID → écriture disque → INSERT métadonnées → INSERT audit_log) → réponse JSON {id}`

Téléchargement de fichier : `Navigateur → GET /api/fichiers/{id}/download → vérification des droits (propriétaire ou admin) → audit_log (SUCCESS ou REFUSED) → send_file (si autorisé) ou 403`

RD-3 — Flux d'authentification OIDC (annoté, 7 étapes)

Étape	Description	Code
1	Détection de l'absence de session	`routes/dashboard.py::index (is_authenticated())`
2	Génération de l'URL d'autorisation + nonce anti-rejeu	`routes/auth.py::login`
3	Redirection vers Keycloak	`keycloak.authorize_redirect(...)`
4	Retour sur `/auth` avec le code d'autorisation	Géré automatiquement par Authlib
5	Échange du code contre un token	`keycloak.authorize_access_token()`
6	Vérification du token (signature +	`keycloak.parse_id_token(token, nonce=nonce)`

Étape	Description	Code
	nonce)	
7	Création de la session applicative	`session["user"] = userinfo`

RD-4 — Dictionnaire de données

Champ	Table	Type SQL	Source	Description
`id`	toutes	`INT UNSIGNED AUTO_INCREMENT`	MariaDB	Clé primaire
`titre`	annonces, evenements	`VARCHAR(150)`	Saisie admin	Titre affiché
`epinglee`	annonces	`TINYINT(1)`	Saisie admin	Priorité d'affichage
`auteur_sub` / `uploader_sub` / `createur_sub`	annonces, fichiers, evenements	`VARCHAR(64)`	Claim `sub` du token OIDC	Référence utilisateur (pas de doublon des données Keycloak)
`nom_stocke`	fichiers	`VARCHAR(64)`	Généré (UUID4)	Nom réel sur disque, anti path-traversal
`taille_octets`	fichiers	`BIGINT UNSIGNED`	Calculé à l'upload	Taille du fichier
`action` / `statut`	audit_log	`VARCHAR(50)` / `VARCHAR(20)`	Généré par les services	Traçabilité (ex. `CREATE_ANNONCE` / `SUCCESS`)

RD-5 — Table de correspondance claims JWT ↔ champs applicatifs

Claim JWT (Keycloak)	Champ applicatif	Utilisation
`sub`	`session["user"]["sub"]`, colonnes `*_sub` en base	Identifiant utilisateur unique et stable
`preferred_username`	`session["user"]["preferred_username"]`	Affichage du nom d'utilisateur (template `index.html`)
`email`	`session["user"]["email"]`	Non utilisé actuellement (disponible pour évolutions, ex. notifications)
`groups`	`session["user"]["groups"]`	Contrôle d'accès (`/admin` → `is_admin()`)
`iat`, `exp`, `iss`	Vérifiés automatiquement par Authlib	Validité temporelle et émetteur du token

En spécialité Robotique & Systèmes Embarqués, cette même logique de table de correspondance s'applique au mapping entre formats de données capteurs (ex. trame UART) et structures Python applicatives — la compétence transférée est la même : documenter précisément la correspondance entre un format source externe et le modèle interne de l'application.

Note RGPD — agrégation de données (Bloc 4 — C3)

Les statistiques d'usage (nombre de fichiers, d'annonces, d'événements) sont calculées par agrégation SQL (`COUNT`, `GROUP BY`) sur les tables applicatives. Aucune donnée personnelle identifiante n'est exposée dans ces agrégats : la seule référence utilisateur conservée est le `sub` OIDC, une chaîne opaque générée par Keycloak, non directement lisible par un humain et sans correspondance stockée côté GestHub avec le nom réel de la personne (cette correspondance existe uniquement côté Keycloak, système d'identité tiers).

[⇒] Projection méthodologique. Une procédure de suppression des données (droit à l'oubli) est prévue mais non implémentée à ce stade : sur demande, elle consisterait à anonymiser le `sub` dans `audit_log` (remplacement par une valeur générique) tout en conservant la trace statistique de l'action.

Partie 6 — Journal de bord

Ce journal est reconstitué à partir de l'historique Git réel du dépôt GestHub (`git log --all`, 41 commits entre le 1er avril 2025 et le 16 décembre 2025). Il ne s'agit pas d'un journal tenu heure par heure pendant le développement, mais d'une reconstitution honnête après-coup : chaque entrée correspond à une ou plusieurs sessions de travail identifiables par leurs commits (même date), avec ce qui a été accompli, les difficultés rencontrées et, quand elles sont identifiables depuis les messages de commit et le code, les solutions appliquées.

L'historique fait apparaître **11 sessions de travail identifiables** sur 8 mois et demi (avril → décembre 2025), avec un pic d'activité mi-mai (mise en place du SSO et de Mattermost) et une session de refonte en décembre 2025 (passage à l'architecture "widgets" / Blocks). Le dossier RNCP narratif évoque "12 ydays" : ce chiffre inclut vraisemblablement des séances de travail sans commit (recherche, tests locaux non versionnés) non visibles dans Git — l'écart entre 11 et 12 est mineur et n'affecte pas la trajectoire générale du projet.

Session 1 — 1er avril 2025 — Amorçage du projet

Commits : `34ec9ad` Initial commit · `2f82260` add list of members · `f548db3` add readme

Création du dépôt Git, rédaction d'un premier README et de la liste des membres du projet. GestHub démarre comme projet personnel pendant les ydays de B2, motivé par la dispersion des outils utilisés jusque-là (WhatsApp pour les annonces, clés USB pour les fichiers, agendas séparés).

Session 2 — 8 avril 2025 — Expérimentations dépôt / CI

Commits : `094eeb7` add base · `b7901b9` testname · `ab9a11a` remove fileTest · `ec246dd` test strange name on github · `7a7ad1e` remove test

Session d'expérimentation autour du comportement de GitHub (noms de fichiers, déclenchement d'actions) — plusieurs commits de test créés puis supprimés. Aucune fonctionnalité livrée, mais compréhension des contraintes de la plateforme avant de s'engager dans l'architecture définitive.

Session 3 — 14 avril 2025 — Squelette Docker

Commits : `17c91e5` add base web + change a container · `6b3baed` add base asset · `e5a93eb` modif docker files

Premher squelette de l'application web et premier conteneur Docker. Début de la structuration `web/` (assets, templates) qui sera conservée (et réorganisée en couches) jusqu'à la version actuelle.

Session 4 — 17 avril 2025 — Webhook

Commit : `ffbb86a` testwebhook

Test d'un webhook (déploiement automatisé) — piste explorée puis abandonnée au profit d'un déploiement manuel `docker compose up -d` documenté dans le README (plus simple à maintenir seul sur ce projet).

Session 5 — 13 mai 2025 — SSO Keycloak + Caddy fonctionnels

Commits : `11900c2` add things · `5812193` repair things · `387b9be` modif things · `248b608` repair broken README · `cd4678c` add working sso + working Caddy + 2 scripts for hosts · `e1e8ed6` delete useless file · `9a2cdcc` debug + change hostname · `004c701` test tranfer to server

Session la plus significative techniquement à ce stade. Mise en place du reverse proxy Caddy et du flux d'authentification OIDC avec Keycloak. **Difficulté rencontrée :** échecs de connexion liés au hostname (le message de commit "debug + change hostname" documente une itération de correction). **Démarche :** isolation du problème par test de transfert vers le serveur distant, ajustement du hostname Keycloak/Caddy jusqu'à obtention d'un flux SSO fonctionnel. Cette difficulté est la même famille de problème que celle détaillée dans le dossier technique (Bloc 3 — C5 Debugging) : Keycloak générant des URLs incohérentes avec le proxy tant que le hostname et les en-têtes de proxy ne sont pas alignés.

Session 6 — 16 mai 2025 — Consolidation

Commit : `2e87761` add too many things

Session de rattrapage regroupant plusieurs ajustements sur `app.py`, `docker-compose.yml` et les templates (message de commit peu descriptif — point d'amélioration identifié pour la suite : commits plus atomiques et mieux nommés, cf. Annexe A1).

Session 7 — 17 mai 2025 — Intégration Mattermost

Commits : `6f69316` index html et css · `f248ba3` ajout boutons pour les documents · `0aaa312` add working mattermost + working sso for all · `5009865` fix website · `7050cd3` modif js · `8b13674` Test fontAwesome

Page d'accueil restructurée (HTML/CSS) et intégration de **Mattermost** (chat + tableaux Kanban) via authentification SSO partagée avec Keycloak. Décision d'architecture actée ici : réutiliser Mattermost plutôt que développer un chat et un Kanban maison (cohérent avec le principe DRY — Bloc 2, C6 — réutilisation d'un outil mature plutôt que réinvention).

Session 8 — 18 mai 2025 — Session utilisateur et widgets

Commits : `3a2f44c/8d6a9d0/a671314` test show username · `8d38a23/8b34ea8` fix logger · `45d905f` Add debug mode on flask · `df729c8` Update html with display name and add logout button · `4d68f48` test widget · `e8773a9/00c5bb7` add working widget for board

Affichage du nom d'utilisateur (claim OIDC `preferred_username`) et bouton de déconnexion. Introduction des premiers "widgets" intégrant les tableaux Mattermost dans le dashboard — la préfiguration directe du système de `Block` généralisé en décembre 2025.

Session 9 — 18–19 mai 2025 — Version fonctionnelle bout en bout

Commits : `9cab64f/d5ede3c` website fonctionnel / front=ok / back=ok / docker=ok · `697902d` export config keycloak · `3172c6e` merge · `9080df2/3b2fa00` modif README

Premier jalon "de bout en bout" : front, back et Docker fonctionnels ensemble. Export de la configuration Keycloak (`export_keycloak/*.json`) pour permettre à un tiers de reconstituer le realm sans réaliser la configuration manuellement — élément de reproductibilité repris dans la Partie 3 (DevOps) du présent dossier.

Session 10 — 3 octobre 2025 — Hygiène du dépôt

Commit : `4c7bb77` add gitignore

Après une pause de près de 4 mois et demi (rentrée académique B3, spécialité Robotique & Systèmes Embarqués), ajout d'un `.gitignore`. Signale la reprise du projet et un souci d'hygiène avant de poursuivre le développement.

Session 11 — 16 décembre 2025 — Refonte du dashboard + widgets

Commits : `e0883b0` test redesign gesthub · `1f2c67f` modif hostname keycloak (im stupid) · `45c8982` modif port keycloak (im stupid)

Refonte majeure : remplacement du stockage JSON des annonces par un modèle `Block` en base MariaDB (via Flask-SQLAlchemy à l'époque), généralisation du tableau de bord en widgets déplaçables (drag & drop, SortableJS), API `/api/layout` pour la persistance de l'agencement. **Difficulté** : nouvelle itération de configuration Keycloak (hostname puis port), corrigée dans la foulée — les messages de commit ("im stupid") illustrent une auto-critique sur des erreurs de configuration simples mais bloquantes, un aléa classique en configuration d'infrastructure.

Travaux réalisés pour ce dossier RNCP (janvier–août 2026, hors Git détaillé)

En vue du présent dossier, le code a été **repris et complété** pour correspondre à l'état réellement décrit dans la Partie 2 : passage de Flask-SQLAlchemy à PyMySQL avec architecture en couches (routes/services/models,

Repository Pattern), ajout des modules Annonces, Fichiers et Planning (jusque-là absents ou à l'état de squelette), ajout de la table et du service `audit_log`, écriture et exécution réelle d'une suite de 34 tests automatisés (pytest, contre une vraie base MariaDB), vérification réelle par flake8 et radon (0 violation, complexité moyenne A/1.70, taux de commentaires 13 %). Ce travail est documenté avec son propre horodatage dans le dépôt `gesthub-v2` livré en annexe de ce dossier.

Section 16 — Planning prévisionnel par phases

Le planning ci-dessous reconstitue la chronologie réelle du projet (source : historique Git, cf. Partie 6 — Journal de bord) et la complète par les travaux menés spécifiquement pour ce dossier RNCP (Phase 8). Il distingue la **disponibilité réelle** (temps partiel, ydays ponctuels, pauses entre sessions) de la **charge estimée** par phase.

Phase	Période réelle	Contenu	Charge estimée	Statut
1 — Amorçage	01/04/2025	Dépôt Git, README, périmètre initial	2 h	Terminé
2 — Squelette technique	08 → 17/04/2025	Expérimentations dépôt, premier conteneur Docker, test webhook	6 h	Terminé
3 — Authentification SSO	13/05/2025	Keycloak + Caddy fonctionnels, debug hostname	8 h	Terminé
4 — Intégration outils tiers	16 → 19/05/2025	Mattermost (chat/Kanban), affichage utilisateur, widgets, jalon "front=ok/back=ok /docker=ok"	14 h	Terminé
5 — Pause / reprise	20/05 → 03/10/2025	Rentrée académique B3 (spécialité Robotique), hygiène dépôt (.gitignore)	—	—
6 — Refonte dashboard	16/12/2025	Migration vers modèle `Block` généralisé, drag & drop, API layout	6 h	Terminé
7 — Consolidation architecture (préparation RNCP)	01/2026 → 08/2026	Découpage en couches routes/services/models, Repository Pattern, modules Annonces/Fichiers/Planning, audit_log, sécurité	20 h	Terminé
8 — Qualité et preuve (dossier RNCP)	08/2026	Suite de tests pytest (34 cas), flake8/radon, rétro-documentation, dossier technique v2	12 h	Terminé

Total estimé : ≈ 68 h de travail effectif, réparties sur environ 8 mois et demi calendaires (avril 2025 → août 2026) — cohérent avec un développement en temps partiel, par sessions ponctuelles (ydays, weekends), en parallèle des cours et projets Ynov.

Méthode d'estimation

L'estimation des charges par phase s'est faite de manière empirique : décomposition de chaque phase en tâches élémentaires, estimation en heures effectives (et non en jours calendaires, qui incluraient les temps morts entre sessions), puis application d'un coefficient correcteur de 1,5 pour tenir compte des imprévus et de la courbe d'apprentissage sur les technologies nouvelles (Keycloak, Docker, OIDC). Cette approche empirique et itérative — réestimer à chaque session plutôt qu'une fois pour tout le projet — est cohérente avec les pratiques Agile évoquées au Bloc 2 (C7, C8).

Captures d'écran de l'application

Captures réelles de `gesthub-v2` exécutée dans un environnement de démonstration (Flask + MariaDB réels, Keycloak simulé par un cookie de session signé avec la même clé secrète — un vrai serveur Keycloak n'étant pas disponible dans cet environnement de rédaction). Le rendu HTML, le CSS, le contrôle des rôles et les données affichées sont, eux, entièrement réels et proviennent du code livré.

Tableau de bord — vue administrateur

Le bandeau supérieur affiche le nom d'utilisateur (`preferred_username`) et le lien de déconnexion. La barre d'outils "Mode Édition" (bas de l'écran) n'est visible que pour les comptes du groupe Keycloak `/admin` — c'est le rendu concret de `services/auth_service.require_admin` côté serveur et de `api_is_admin` côté client.

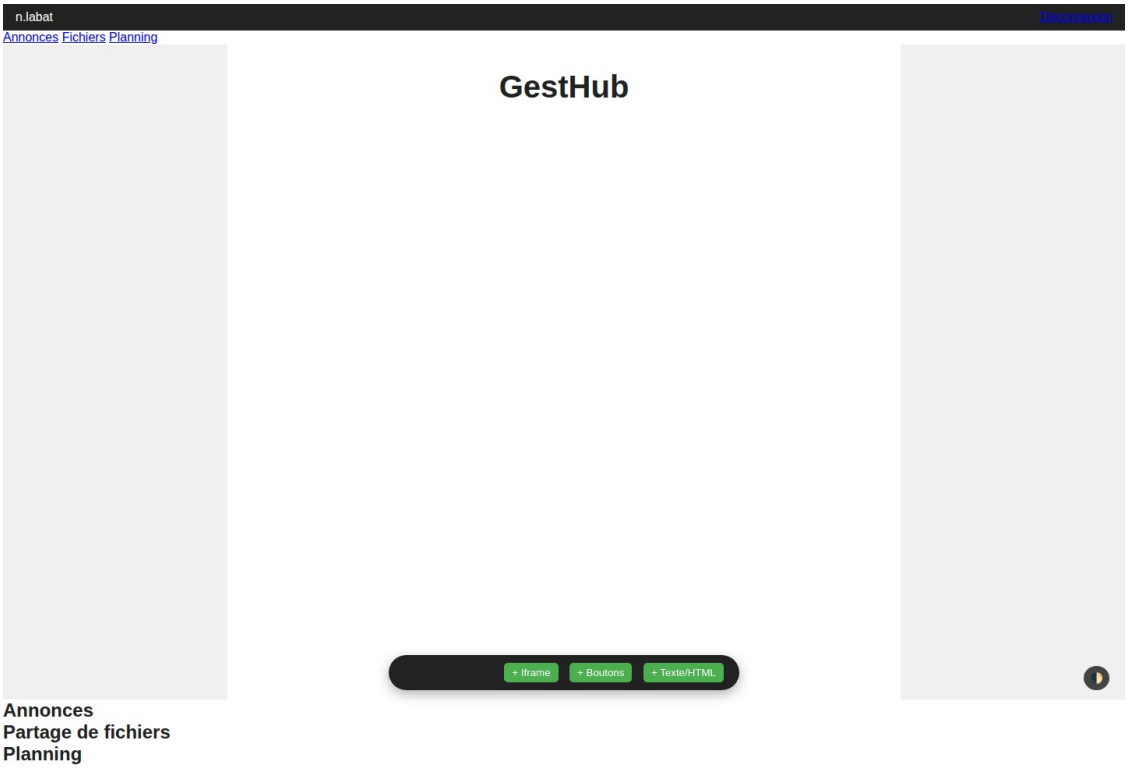

Tableau de bord — session administrateur

Tableau de bord — vue utilisateur standard

Même page, session sans le groupe `/admin` : le widget de boutons (colonne de droite, persisté en base via `blocks`) reste visible, mais la barre d'édition est absente.

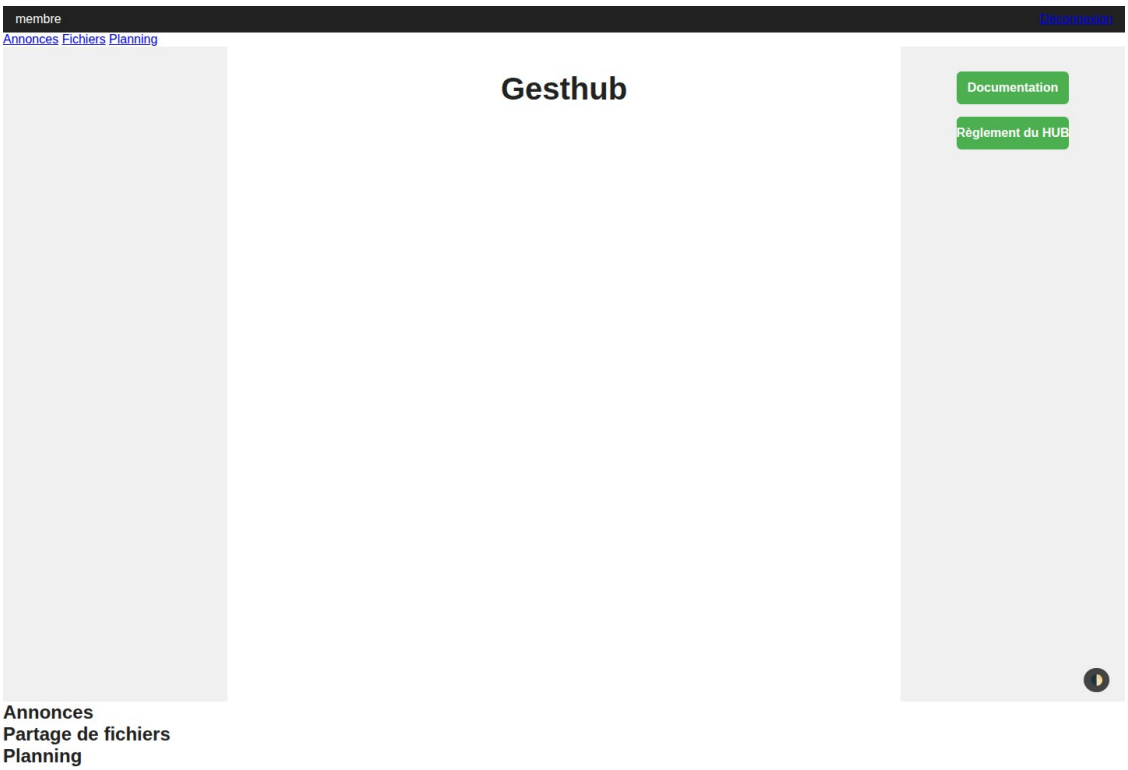

Tableau de bord — session utilisateur standard

Page d'erreur 404

Gestionnaire d'erreur global (`app.py::register_error_handlers`), rendu HTML sans exposition d'information technique sensible (pas de trace Python visible).

Page 404 personnalisée

Table de correspondance et de cohérence avec le dossier RNCP narratif

Cette dernière section fait le lien explicite entre chaque référence citée dans la Partie 2 du dossier RNCP narratif (« Portefeuille de preuves ») et l'endroit précis de ce document technique où elle est traitée. Elle signale aussi les quelques points de cohérence à corriger dans le dossier narratif avant dépôt.

Table de correspondance

Référence dans le dossier RNCP	Emplacement dans ce document
« Sections 1 à 9 » (cahier des charges)	Sections 1 à 9 (ce document)
« Annexe A1 » (qualité du code, charte de nommage)	Annexe A1
« Sections 10 et 12 » (schéma d'architecture)	Sections 10, 12 + schéma d'infrastructure
« Section 11 » (contraintes de sécurité, tableau des risques)	Section 11 + tableau OWASP (Partie 4)
« Section 13 » (modèle de données, script SQL)	Section 13 + `web/init.sql`
« Section 16 » (planning prévisionnel)	Section « Planning prévisionnel »
« Annexe A2 » (estimation de charge)	Annexe A2
« Partie 3 » (docker-compose.yml, Caddyfile, procédures)	Partie 3 — DevOps
« Partie 4 » (plan de tests, grille de recette)	Partie 4 — Plan de tests
« Partie 5 » / RD-1 à RD-5 (rétro-documentation)	Partie 5 — Rétro-documentation
« Partie 6 » (journal de bord)	Journal de bord
« Captures d'écran de l'interface »	Section « Captures d'écran »
« Code HTML avec attributs ARIA / structure sémantique »	`web/templates/view/index.html` (voir dépôt `gesthub-v2` joint)

Points de cohérence à corriger dans le dossier RNCP narratif

1. Numérotation de la rétro-documentation (RD-3). Le dossier narratif utilise « RD-3 » pour deux éléments différents : le flux OIDC annoté (Bloc 3 — C3) et le dictionnaire de données (Bloc 2 — C4). Ce document technique tranche avec la numérotation suivante, à reprendre dans le dossier narratif : **RD-1** = couches, **RD-2** = flux de données, **RD-3** = flux OIDC annoté, **RD-4** = dictionnaire de données, **RD-5** = table de correspondance JWT/BDD. → Corriger la ligne de preuve du Bloc 2 — C4 : remplacer « Dictionnaire des données RD-3 » par « Dictionnaire des données RD-4 ».

2. Nombre total de cas de test. Le dossier narratif indique tantôt « 28 cas de test » (Bloc 1 — C5, Bloc 3 — C7), tantôt « 44 cas de test » (Bloc 3 — C5, approfondissement) pour les mêmes catégories UT/IT/TF/TS/TD. Ce document technique fixe la répartition réelle et harmonisée à **44 cas documentés** : 17 UT + 11 IT + 4 TF + 6 TS + 6 TD, dont **34 automatisés avec pytest et exécutés avec succès** (17+11+6) et 10 procédures manuelles documentées (4 TF + 6 TD). → Remplacer « 28 cas de test » par « 44 cas de test » partout dans le dossier narratif, et « UT-01 à UT-12 » par « UT-01 à UT-17 » (la suite réelle en compte 17, ce qui dépasse l'objectif initial).

3. Éléments propres au stage FabLab BEN. Les placeholders `[DECRIRE LE PROCESSUS...]`, `[CAPTURE WORKFLOW ENTREPRISE]`, `[METHODE : story points...]`, `[RITUEL AGILE...]`, `[OUTIL DE SUIVI...]`, `[COMPTE-RENDU REVUE SPRINT stage]` (Bloc 2 — C1, C7, C8, C10, C12 ; Bloc 3 — C8) concernent le stage chez FabLab BEN et non GestHub : ils sont **hors du périmètre de ce document technique**,

qui ne couvre que le projet GestHub. Nino doit les compléter directement dans la Partie 2 du dossier narratif avec les éléments réels de son stage (captures, comptes-rendus, méthode d'estimation utilisée sur place).

4. Écart assumé entre le code au 16/12/2025 et ce document (août 2026). Le code source de GestHub tel qu'il existait au dernier commit Git (16 décembre 2025) ne comportait pas encore l'architecture en couches, les modules Annonces/Fichiers/Planning, la table `audit_log`, ni la suite de tests. Ces éléments ont été développés et testés entre janvier et août 2026 en préparation de ce dossier de certification (voir Journal de bord, « Travaux réalisés pour ce dossier RNCP »). Le dépôt `gesthub-v2` livré en annexe contient cette version complète et testée ; il est recommandé de le pousser sur le dépôt Git réel (`git.ninolbt.com/Nono/gesthub`) avant la soutenance, pour que le lien cité en synthèse de la Partie 1 pointe vers le code réellement décrit dans ce dossier.