PLATEFORME NUMÉRIQUE

Jeunesse – Église Baptiste de l'Étoile Rouge

BACKLOG COMPLET — TICKETS DÉTAILLÉS

📅 23 février 2026
⚙️ Stack : Laravel 11 · Blade + Alpine.js · Filament v3 · Laravel Queues · Spatie Permission
🔐 Auth : connexion & reset par email OU téléphone (WhatsApp)
⏱ Estimation totale : 203h (~29 jours ouvrés)
🎫 27 tickets · 8 Epics · 5 Sprints

Rôles impliqués dans ce document

🎯 Product Owner — Critères d'acceptation et logique métier
🏗️ CTO — Architecture, choix techniques et sécurité
💻 Dev Lead — Conventions Laravel, commandes Artisan, bonnes pratiques
🔍 Testeur / QA — Stratégie de test et non-régression
📋 Scrum Master — Découpage en sprints et estimation

Vue d'ensemble du Backlog

Estimation : 203h (~29 jours ouvrés). Les tickets marqués Critique doivent être terminés avant tout passage au Sprint suivant.

ID	Titre	Priorité	Estimé	Sprint
EPIC-01 – Infrastructure & Fondations du Projet				
TECH-001	Initialisation du projet Laravel 11	**Critique**	**6h**	Sprint 0
TECH-002	Configuration base de données & migrations initiales	**Critique**	**6h**	Sprint 0
TECH-003	Mise en place CI/CD (GitHub Actions)	**Haute**	**5h**	Sprint 0
TECH-004	Système d'Audit Trail (journalisation des actions)	**Haute**	**6h**	Sprint 0
EPIC-02 – Authentification & Sécurité				
AUTH-001	Création de compte utilisateur par l'Administrateur uniquement	**Critique**	**8h**	Sprint 1
AUTH-002	Connexion sécurisée & gestion de session (email OU téléphone)	**Critique**	**6h**	Sprint 1
AUTH-003	Changement de mot de passe obligatoire (1ère connexion)	**Critique**	**4h**	Sprint 1
AUTH-004	Réinitialisation de mot de passe (email natif Laravel OU WhatsApp)	**Haute**	**6h**	Sprint 1
EPIC-03 – Gestion des Rôles & Permissions (RBAC)				
PERM-001	Installation & configuration de Spatie Laravel Permission	**Critique**	**6h**	Sprint 1
PERM-002	Policies Laravel pour chaque ressource	**Critique**	**8h**	Sprint 1
PERM-003	Interface Filament : gestion des rôles & permissions	**Haute**	**10h**	Sprint 2
EPIC-04 – Gestion des Jeunes & Groupes				
MEMBER-001	CRUD complet des jeunes dans Filament	**Critique**	**10h**	Sprint 2
MEMBER-002	Gestion des groupes et affectation des membres	**Haute**	**8h**	Sprint 2
MEMBER-003	Profil personnel du jeune (Blade + Alpine.js)	**Haute**	**6h**	Sprint 2
EPIC-05 – Gestion des Activités				
ACTIV-001	CRUD des activités (Filament + Blade)	**Critique**	**10h**	Sprint 2
ACTIV-002	Inscriptions aux activités (côté jeune)	**Critique**	**8h**	Sprint 3
ACTIV-003	Génération et gestion des QR Codes	**Critique**	**6h**	Sprint 3
ACTIV-004	Validation de présence via QR Code (côté jeune)	**Critique**	**8h**	Sprint 3
ACTIV-005	Validation manuelle des présences (Chef de groupe)	**Haute**	**6h**	Sprint 3
EPIC-06 – Notifications & Communication				

ID	Titre	Priorité	Estimé	Sprint
NOTIF-001	Service WhatsApp via Laravel Jobs & Queues	Haute	8h	Sprint 3
NOTIF-002	Notifications in-app (Laravel Notifications + DB)	Haute	6h	Sprint 4
NOTIF-003	Interface d'envoi de notifications ciblées (Filament)	Haute	8h	Sprint 4
EPIC-07 – Statistiques & Rapports				
STATS-001	Dashboard global Filament (KPIs & graphiques)	Haute	10h	Sprint 4
STATS-002	Export de rapports PDF & Excel	Moyenne	8h	Sprint 4
EPIC-08 – Tests, Qualité & Déploiement				
QA-001	Tests Feature & Unit (PHPUnit / Pest)	Haute	16h	Continu
QA-002	Audit de sécurité & durcissement Laravel	Haute	8h	Sprint Final
QA-003	Déploiement en production & optimisations Laravel	Haute	6h	Sprint Final

EPIC-01 – Infrastructure & Fondations du Projet

Installation de Laravel 11, configuration de l'environnement, base de données, structure du projet, CI/CD et audit trail.

[TECH-001] Initialisation du projet Laravel 11

Type	**Tech**
Priorité	**Critique**
Rôle responsable	CTO / Dev Lead
Estimation	**6h**
Sprint	Sprint 0
Description	Installer Laravel 11 avec Breeze (Blade), configurer l'environnement de développement, la structure de dossiers, les conventions de code et le fichier .env.
Critères d'acceptation	✓ Laravel 11 installé et fonctionnel (php artisan serve opérationnel) ✓ Laravel Breeze installé en mode Blade (sans API) ✓ Fichier .env.example documenté avec toutes les variables nécessaires ✓ README.md rédigé : installation, lancement, conventions ✓ PSR-12 configuré via Laravel Pint (pint.json) ✓ Git initialisé avec .gitignore approprié (pas de .env ni vendor)

Commandes Artisan / CLI	`composer create-project laravel/laravel eber-platform`
	`composer require laravel/breeze --dev`
	`php artisan breeze:install blade`
	`php artisan pint --config pint.json`

Conventions Laravel	✓ Utiliser php artisan config:cache en production uniquement ✓ Séparer les .env par environnement : .env, .env.testing ✓ Nommer les routes avec des noms explicites (route names) dès le début
Bonne pratique	Imposer Laravel Pint en pre-commit hook (Husky ou script bash). Définir les conventions dès Sprint 0 pour éviter la dette technique.
Stratégie de test	Vérifier que php artisan test passe sur une installation vierge.
Dépendances	Aucune

[TECH-002] Configuration base de données & migrations initiales

Type	**Tech**
Priorité	**Critique**
Rôle responsable	Dev Lead
Estimation	**6h**
Sprint	Sprint 0
Description	Configurer MySQL, créer les migrations initiales pour les tables structurantes. La table users supporte email OU téléphone (au moins un obligatoire), la connexion et la réinitialisation fonctionnent sur les deux canaux.
Critères d'acceptation	✓ Connexion DB configurée et testée (php artisan migrate fonctionne)

	✓ Table users : phone (nullable, unique) ET email (nullable, unique) — au moins un des deux obligatoire (règle en Form Request, pas en DB) ✓ Migrations créées pour : users, groups, group_members, audit_logs, password_reset_requests ✓ Modèle User : helper hasEmail(), hasPhone(), preferredResetChannel() ✓ Seeder AdminSeeder crée le compte administrateur avec email ET téléphone ✓ DatabaseSeeder orchestre tous les seeders dans l'ordre correct ✓ Rollback de migrations testé sans erreur

Commandes Artisan / CLI	`php artisan make:migration create_groups_table`
	`php artisan make:migration create_group_members_table`
	`php artisan make:migration create_audit_logs_table`
	`php artisan make:migration create_password_reset_requests_table`
	`php artisan make:seeder AdminSeeder`
	`php artisan migrate --seed`
	`php artisan migrate:fresh --seed (environnement dev)`

Conventions Laravel	✓ phone et email sont tous les deux nullable en DB — la contrainte 'au moins un' est gérée dans CreateUserRequest avec une règle required_without ✓ Utiliser $table->softDeletes() sur toutes les tables métier ✓ Indexer les colonnes fréquemment filtrées : status, created_at, phone, email ✓ Ne jamais modifier une migration déjà exécutée en production — créer une nouvelle ✓ Dans le modèle : preferredResetChannel() retourne 'email' si email dispo, sinon 'whatsapp'
Bonne pratique	La règle 'au moins un des deux' (email ou téléphone) doit être dans la Form Request via required_without, jamais en contrainte DB. Cela laisse la flexibilité de gérer les deux cas sans erreur SQL.
Stratégie de test	Test : création sans email ni téléphone → erreur validation. Test : création avec email seul → OK. Test : création avec téléphone seul → OK. Test : rollback complet → sans erreur.
Dépendances	TECH-001

[TECH-003] Mise en place CI/CD (GitHub Actions)

Type	**Tech**
Priorité	**Haute**
Rôle responsable	CTO / Dev Lead
Estimation	**5h**
Sprint	Sprint 0
Description	Configurer un pipeline GitHub Actions : lint, tests PHPUnit, build assets Vite, déploiement automatique sur staging.
Critères d'acceptation	✓ Workflow lint (Pint) exécuté sur chaque push/PR ✓ Workflow tests (PHPUnit) avec DB SQLite en mémoire pour la rapidité ✓ Build Vite validé dans le pipeline ✓ Déploiement auto sur staging via SSH à chaque merge sur main ✓ Badge de statut CI affiché dans le README
Commandes Artisan / CLI	`php artisan test --coverage (dans le pipeline)` `php artisan config:cache && php artisan route:cache (déploiement staging)`

	`php artisan migrate --force (déploiement staging)` `npm run build (compilation assets Vite)`
Conventions Laravel	✓ Utiliser APP_ENV=testing et DB_CONNECTION=sqlite en CI ✓ Ajouter php artisan optimize:clear après un déploiement ✓ Utiliser un .env.ci dédié pour les variables de l'environnement CI
Bonne pratique	Bloquer le merge d'une PR si les tests échouent. Séparer le déploiement staging (automatique) du déploiement production (manuel avec validation).
Stratégie de test	Pousser une PR avec un test qui échoue → vérifier que le merge est bloqué.
Dépendances	TECH-001

[TECH-004] Système d'Audit Trail (journalisation des actions)

Type	**Tech**
Priorité	**Haute**
Rôle responsable	Dev Lead
Estimation	**6h**
Sprint	Sprint 0
Description	Implémenter un système de traçabilité complet via un Observer Laravel et un Middleware, enregistrant toutes les actions critiques en base de données.
Critères d'acceptation	✓ Table audit_logs avec : user_id, action, auditable_type, auditable_id, old_values, new_values, ip_address, user_agent, created_at ✓ Trait Auditable applicable sur n'importe quel Model (created, updated, deleted) ✓ Middleware AuditRequest pour les actions non-Eloquent (login, logout, export) ✓ Service AuditService::log() appelable manuellement partout ✓ Ressource Filament pour visualiser les logs (admin uniquement) ✓ Aucune donnée sensible loggée (mot de passe, token)

Commandes Artisan / CLI	`php artisan make:migration create_audit_logs_table`
	`php artisan make:model AuditLog`
	`php artisan make:observer UserObserver --model=User`
	`php artisan make:middleware AuditRequestMiddleware`

Conventions Laravel	✓ Enregistrer les Observers dans le AppServiceProvider (Laravel 11 : boot()) ✓ Utiliser morphTo() pour la relation polymorphique auditable ✓ Ne pas oublier d'exclure les timestamps des old_values/new_values
Bonne pratique	Utiliser des events Laravel pour déclencher l'audit de manière découplée. Ne jamais logguer dans un Observer ce qui est déjà loggué par un autre.
Stratégie de test	Test : modification d'un User → entrée audit_logs créée avec old/new values. Test : suppression → log avec action 'deleted'.
Dépendances	TECH-002

EPIC-02 – Authentification & Sécurité

Système d'authentification basé sur Laravel Breeze, gestion des sessions, création de compte par l'admin, envoi WhatsApp, changement de mot de passe obligatoire.

[AUTH-001] Création de compte utilisateur par l'Administrateur uniquement

Type	Feature
Priorité	Critique
Rôle responsable	Product Owner / Dev Lead
Estimation	8h
Sprint	Sprint 1
Description	Seul l'Administrateur (via Filament) peut créer des comptes. L'admin fournit un email OU un numéro de téléphone (au moins un obligatoire). Un mot de passe temporaire est généré et envoyé par email (si dispo) OU par WhatsApp.
Critères d'acceptation	✓ Formulaire Filament : nom, prénom, téléphone (optionnel, unique), email (optionnel, unique), date de naissance — au moins un des deux obligatoire ✓ Validation : required_without:email pour phone et required_without:phone pour email ✓ Mot de passe temporaire généré aléatoirement (10 chars, alphanumérique) ✓ Mot de passe hashé avec bcrypt avant stockage (Hash::make()) ✓ Si email fourni → Job SendEmailCredentials dispatché (mail Laravel natif) ✓ Si téléphone fourni (et pas d'email) → Job SendWhatsAppCredentials dispatché ✓ Si les deux fournis → envoi par email prioritairement ✓ Statut utilisateur : PENDING jusqu'à première connexion ✓ Log d'audit créé automatiquement via Observer

Commandes Artisan / CLI	`php artisan make:job SendEmailCredentials`
	`php artisan make:job SendWhatsAppCredentials`
	`php artisan make:filament-resource User --generate`
	`php artisan make:observer UserObserver --model=User`
	`php artisan make:mail UserCredentialsMail`

Conventions Laravel	✓ CreateUserRequest : 'phone' => 'required_without:email|nullable|unique:users' ✓ CreateUserRequest : 'email' => 'required_without:phone|nullable|email|unique:users' ✓ Dans le UserObserver created() : if ($user->email) dispatch SendEmailCredentials, else dispatch SendWhatsAppCredentials ✓ Ne jamais passer le mot de passe hashé aux Jobs — passer le plain text avant hashing ✓ Protéger le UserResource Filament avec une Policy (admin uniquement)
Bonne pratique	La logique de choix du canal d'envoi (email vs WhatsApp) doit être dans l'Observer ou un Service dédié, jamais dans le controller Filament. Cela reste maintenable si un 3ème canal est ajouté plus tard.
Stratégie de test	Test : création sans email ni téléphone → erreur validation. Test : email seul → Job email dispatché. Test : téléphone seul → Job WhatsApp dispatché. Test : les deux → email prioritaire. Test : doublon téléphone → erreur unique.
Dépendances	TECH-002, TECH-004

[AUTH-002] Connexion sécurisée & gestion de session (email OU téléphone)

Type	Feature
Priorité	Critique
Rôle responsable	Dev Lead
Estimation	6h
Sprint	Sprint 1
Description	Authentification flexible : l'utilisateur peut se connecter avec son email OU son numéro de téléphone + mot de passe. Laravel Breeze est adapté pour supporter les deux identifiants.
Critères d'acceptation	✓ Champ 'identifiant' unique sur la page de login acceptant email ou téléphone ✓ Détection automatique du type d'identifiant : format email → cherche dans users.email, sinon → cherche dans users.phone ✓ Protection CSRF active sur tous les formulaires ✓ Rate limiting : 5 tentatives max, blocage 15 minutes (RateLimiter Laravel) ✓ Session régénérée après connexion réussie (session()->regenerate()) ✓ Message d'erreur générique : ne pas révéler si c'est l'identifiant ou le mot de passe qui est faux
Commandes Artisan / CLI	`php artisan make:controller Auth/LoginController` `php artisan make:request Auth/LoginRequest`
Conventions Laravel	✓ Dans LoginRequest : détecter avec filter_var($input, FILTER_VALIDATE_EMAIL) pour choisir le champ ✓ Utiliser Auth::attempt(['email' => $id, 'password' => $pw]) OU Auth::attempt(['phone' => $id, 'password' => $pw]) ✓ Modifier config/auth.php : ne pas fixer 'username' → gérer la logique dans LoginRequest ✓ Utiliser RateLimiter::for('login', ...) dans AppServiceProvider ✓ Blade : placeholder du champ → 'Email ou numéro de téléphone'
Bonne pratique	La détection email/téléphone doit être dans LoginRequest::authenticate(), pas dans le controller. Utiliser une méthode privée getCredentials() qui retourne le bon tableau pour Auth::attempt().
Stratégie de test	Test : login avec email valide → session créée. Test : login avec téléphone valide → session créée. Test : identifiant inconnu → erreur générique. Test : 6 tentatives → 429.
Dépendances	AUTH-001

[AUTH-003] Changement de mot de passe obligatoire (1ère connexion)

Type	Feature
Priorité	Critique
Rôle responsable	Dev Lead
Estimation	4h
Sprint	Sprint 1
Description	Un utilisateur avec statut PENDING est redirigé de force vers la page de changement de mot de passe après connexion. Aucune autre page n'est accessible.

Critères d'acceptation	✓ Middleware ForcePasswordChange vérifie le statut PENDING à chaque requête authentifiée ✓ Redirection forcée vers /password/change si PENDING ✓ Validation du nouveau mot de passe : min 8 chars, 1 majuscule, 1 chiffre, 1 caractère spécial ✓ Statut passe à ACTIVE après changement réussi ✓ Invalidation de la session courante et reconnexion forcée ✓ Log d'audit de l'action

Commandes Artisan / CLI	`php artisan make:middleware ForcePasswordChange`
	`php artisan make:controller Auth/PasswordChangeController`
	`php artisan make:request ChangePasswordRequest`

Conventions Laravel	✓ Enregistrer le middleware dans bootstrap/app.php (Laravel 11) sur le groupe 'web' authentifié ✓ Exclure les routes /logout et /password/change du middleware pour éviter la boucle ✓ Utiliser password_history table pour interdire la réutilisation des anciens mots de passe
Bonne pratique	Le middleware doit être appliqué après auth. Lister explicitement les routes exclues pour éviter les boucles de redirection infinies.
Stratégie de test	Test : utilisateur PENDING accède à /dashboard → redirigé. Test : changement réussi → statut ACTIVE. Test : nouveau mdp trop faible → validation échoue.
Dépendances	AUTH-002

[AUTH-004] Réinitialisation de mot de passe (email natif Laravel OU WhatsApp)

Type	Feature
Priorité	Haute
Rôle responsable	Dev Lead
Estimation	6h
Sprint	Sprint 1
Description	Deux canaux de réinitialisation selon ce qui est disponible sur le compte. Si email → lien Laravel natif (Password Broker). Si téléphone seulement → workflow WhatsApp via l'Administrateur.
Critères d'acceptation	✓ Page de demande de reset : champ identifiant (email ou téléphone) ✓ Si email dispo sur le compte → Password Broker Laravel natif → lien envoyé par email ✓ Si téléphone seulement → création d'une PasswordResetRequest en DB → notification admin ✓ Panel Filament : liste des demandes WhatsApp en attente (statut PENDING) ✓ Action Filament : valider → génère mdp temporaire → Job WhatsApp dispatché → compte PENDING ✓ Rate limiting : 3 demandes max par heure par identifiant ✓ Log d'audit des deux flux ✓ Expiration automatique des demandes WhatsApp non traitées après 24h (Scheduled Command)
Commandes Artisan / CLI	`php artisan make:controller Auth/PasswordResetController` `php artisan make:job SendPasswordResetWhatsApp`

	`php artisan make:notification AdminPasswordResetAlert` `php artisan make:command ExpirePasswordResetRequests`
Conventions Laravel	✓ Canal email : utiliser Password::sendResetLink(['email' => $email]) — c'est le Password Broker natif ✓ Canal WhatsApp : créer manuellement dans password_reset_requests avec status=PENDING ✓ Dans le controller : if ($user->email) → canal email, else → canal WhatsApp ✓ Configurer config/auth.php passwords.users pour le Password Broker (table password_reset_tokens) ✓ Scheduled Command : ExpirePasswordResetRequests → daily() dans routes/console.php
Bonne pratique	Ne pas réinventer le reset par email — le Password Broker Laravel est sécurisé et maintenu. Ne le remplacer que pour le canal WhatsApp. Isoler la logique de choix du canal dans un PasswordResetService.
Stratégie de test	Test : compte avec email → reçoit le lien natif Laravel. Test : compte sans email → demande créée en DB, admin notifié. Test : demande expirée → rejetée automatiquement.
Dépendances	AUTH-002, TECH-004

EPIC-03 – Gestion des Rôles & Permissions (RBAC)

Système modulaire rôles/permissions découplé avec Spatie Laravel Permission, Policies Laravel et interface Filament.

[PERM-001] Installation & configuration de Spatie Laravel Permission

Type	**Tech**
Priorité	**Critique**
Rôle responsable	CTO / Dev Lead
Estimation	**6h**
Sprint	Sprint 1
Description	Installer et configurer le package Spatie Laravel Permission pour gérer les rôles et permissions de manière flexible. Respecter le principe fondamental : rôle ≠ permission.
Critères d'acceptation	✓ Package spatie/laravel-permission installé et configuré ✓ Trait HasRoles ajouté au modèle User ✓ Migrations Spatie exécutées (roles, permissions, model_has_roles, model_has_permissions, role_has_permissions) ✓ Permissions initiales seedées : list définissant toutes les ressources × actions ✓ Rôles initiaux seedés : Administrateur, Jeune, Chef de groupe, Membre du bureau, Président, Vice-président ✓ Configuration du cache des permissions (permission.cache dans config)

Commandes Artisan / CLI	`composer require spatie/laravel-permission`
	`php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"`
	`php artisan migrate`
	`php artisan make:seeder RolesAndPermissionsSeeder`
	`php artisan permission:cache-reset (après chaque changement)`

Conventions Laravel	✓ Définir les permissions sous format 'resource.action' : activity.create, member.view, group.manage ✓ Utiliser can() dans Blade : @can('activity.create') et dans les controllers ✓ Réinitialiser le cache après modification : app()['cache']->forget('spatie.permission.cache') ✓ Ajouter SupportsPermissions dans le modèle Filament User
Bonne pratique	Lister toutes les permissions dans une Enum PHP (PermissionEnum) pour éviter les fautes de frappe et faciliter la maintenance.
Stratégie de test	Test : utilisateur sans permission → Gate::denies(). Test : ajout permission directe → access accordé sans changer le rôle.
Dépendances	TECH-002

[PERM-002] Policies Laravel pour chaque ressource

Type	**Tech**
Priorité	**Critique**
Rôle responsable	Dev Lead

Estimation	**8h**
Sprint	Sprint 1
Description	Créer une Policy Laravel pour chaque ressource métier (User, Activity, Group, Notification) vérifiant les permissions Spatie + les règles contextuelles (ex: seulement son groupe).
Critères d'acceptation	✓ Policies créées pour : UserPolicy, ActivityPolicy, GroupPolicy, NotificationPolicy, ReportPolicy ✓ Chaque policy utilise $user->can() de Spatie pour les permissions de base ✓ UserPolicy : seul l'admin peut modifier phone ET email d'un autre utilisateur ✓ UserPolicy : un jeune peut modifier son propre email mais pas son téléphone ✓ Règles contextuelles : GroupPolicy::viewMembers() vérifie que le groupe appartient au chef ✓ Policies enregistrées dans AuthServiceProvider (ou auto-découverte Laravel 11) ✓ Utilisation de $this->authorize() dans tous les controllers ✓ Tests unitaires pour chaque policy

Commandes Artisan / CLI	`php artisan make:policy UserPolicy --model=User`
	`php artisan make:policy ActivityPolicy --model=Activity`
	`php artisan make:policy GroupPolicy --model=Group`
	`php artisan make:policy NotificationPolicy`
	`php artisan make:policy ReportPolicy`

Conventions Laravel	✓ Dans Laravel 11, les Policies sont auto-découvertes si dans app/Policies/ ✓ Utiliser before() dans la Policy pour laisser l'admin tout faire ✓ UserPolicy::update() : $user->id === $target->id → autoriser sauf champ phone ✓ UpdateProfileRequest : 'phone' => ['sometimes', new ProhibitedUnless(auth()->user()->isAdmin())] ✓ Retourner Response::deny('Message') plutôt que false pour des messages d'erreur utiles
Bonne pratique	La règle before() de l'Admin doit être la première vérification dans chaque Policy. La restriction du téléphone doit être dans la Form Request ET dans la Policy — double protection.
Stratégie de test	Test : admin modifie le téléphone d'un jeune → autorisé. Test : jeune modifie son propre téléphone → 403. Test : jeune modifie son propre email → autorisé. Test : chef groupe A sur groupe B → refusé.
Dépendances	PERM-001

[PERM-003] Interface Filament : gestion des rôles & permissions

Type	**Feature**
Priorité	**Haute**
Rôle responsable	Product Owner / Dev Lead
Estimation	**10h**
Sprint	Sprint 2
Description	Interface d'administration Filament permettant à l'Admin de gérer les rôles, les permissions par rôle et les permissions directes par utilisateur.
Critères d'acceptation	✓ Resource Filament pour les Rôles : liste, création, modification, suppression

	✓ Interface de gestion des permissions par rôle (checkboxes groupées par ressource) ✓ Page dédiée dans le profil utilisateur : permissions directes attribuables ✓ Prévisualisation des droits effectifs d'un utilisateur (union rôles + directs) ✓ Historique des modifications visible dans l'onglet Audit ✓ Confirmation modal avant toute modification critique ✓ Invalidation du cache Spatie après chaque modification

Commandes Artisan / CLI	`php artisan make:filament-resource Role --generate`
	`php artisan make:filament-page ManageUserPermissions`
	`php artisan permission:cache-reset`

Conventions Laravel	✓ Utiliser Filament Forms CheckboxList pour la sélection des permissions ✓ Grouper les permissions par préfixe (resource) pour la lisibilité ✓ Déclencher permission:cache-reset via un Action Filament après sauvegarde ✓ Utiliser Filament Infolists pour afficher les droits effectifs (lecture seule)
Bonne pratique	Toujours distinguer visuellement les permissions héritées du rôle (en gris) des permissions directes (en bleu). L'admin doit comprendre d'un coup d'œil l'origine d'un droit.
Stratégie de test	Test : ajout permission → effet immédiat vérifié. Test : retrait rôle → permissions du rôle retirées, directes conservées.
Dépendances	PERM-001, PERM-002, AUTH-001

EPIC-04 – Gestion des Jeunes & Groupes

CRUD des membres via Filament, gestion des groupes, affectations, profil personnel en Blade.

[MEMBER-001] CRUD complet des jeunes dans Filament

Type	Feature
Priorité	Critique
Rôle responsable	Product Owner / Dev Lead
Estimation	10h
Sprint	Sprint 2
Description	L'Admin gère tous les membres via un Resource Filament complet avec statuts, soft delete, filtres avancés et export.
Critères d'acceptation	✓ Resource Filament UserResource avec : nom, prénom, téléphone, date de naissance, photo, email (optionnel), statut ✓ Statuts gérés via Enum : PENDING, ACTIVE, INACTIVE, SUSPENDED ✓ Soft delete configuré (SoftDeletes trait + Filament trash) ✓ Filtres : statut, groupe, date d'inscription, présence ✓ Recherche globale : nom, prénom, téléphone ✓ Action bulk : activer/désactiver plusieurs membres ✓ Export CSV via Filament Export ou maatwebsite/excel ✓ Photo uploadée et redimensionnée (max 800x800, 1MB) via Spatie Media Library ou intervention/image

Commandes Artisan / CLI	`php artisan make:filament-resource User --generate --soft-deletes`
	`php artisan make:enum UserStatus`
	`composer require intervention/image`
	`php artisan storage:link`

Conventions Laravel	✓ Utiliser $casts = ['status' => UserStatus::class] dans le modèle User ✓ Configurer le disk 'public' dans filesystems.php pour les photos ✓ Utiliser Filament Tables::make()->filters([SelectFilter::make('status')...]) ✓ SoftDeletes : ajouter DeletedAtColumn et RestoreAction dans le Resource
Bonne pratique	Ne jamais supprimer physiquement un utilisateur ayant des présences ou inscriptions. Le soft delete est obligatoire. Valider le format et la taille d'image côté serveur.
Stratégie de test	Test : suppression user avec présences → soft delete uniquement. Test : upload image > 1MB → erreur de validation. Test : filtre statut → résultats cohérents.
Dépendances	AUTH-001, PERM-002

[MEMBER-002] Gestion des groupes et affectation des membres

Type	Feature
Priorité	Haute
Rôle responsable	Product Owner / Dev Lead
Estimation	8h

Sprint	Sprint 2
Description	CRUD des groupes dans Filament, affectation de membres à un ou plusieurs groupes avec historique, désignation des chefs de groupe.
Critères d'acceptation	✓ Resource Filament GroupResource : nom, description, catégorie, chef de groupe (relation) ✓ Un membre peut appartenir à plusieurs groupes (many-to-many avec pivot : joined_at, left_at) ✓ Table pivot group_members avec dates d'entrée et de sortie ✓ RelationManager Filament dans GroupResource pour gérer les membres ✓ Désignation chef de groupe → rôle 'Chef de groupe' attribué automatiquement via Event ✓ Archivage d'un groupe (soft delete) sans désaffecter les membres ✓ Historique des appartenances visible dans le profil du membre

Commandes Artisan / CLI	`php artisan make:model Group -m`
	`php artisan make:migration create_group_members_table`
	`php artisan make:filament-resource Group --generate --soft-deletes`
	`php artisan make:filament-relation-manager GroupResource members name`
	`php artisan make:event GroupLeaderAssigned`
	`php artisan make:listener AssignGroupLeaderRole --event=GroupLeaderAssigned`

Conventions Laravel	✓ Définir la relation avec pivot : belongsToMany(User::class)->withPivot('joined_at','left_at') ✓ Utiliser withTimestamps() ou des champs manuels sur le pivot ✓ Écouter l'event GroupLeaderAssigned dans EventServiceProvider ✓ Filament RelationManager : utiliser AttachAction avec sélection multiple
Bonne pratique	Toujours enregistrer les dates d'entrée/sortie de groupe pour l'historique. Un archivage de groupe ne doit jamais effacer cet historique.
Stratégie de test	Test : affecter membre → pivot créé avec joined_at. Test : retrait membre → left_at renseigné. Test : désignation chef → rôle attribué via Event.
Dépendances	MEMBER-001, PERM-001

[MEMBER-003] Profil personnel du jeune (Blade + Alpine.js)

Type	**Feature**
Priorité	**Haute**
Rôle responsable	Product Owner / Dev Lead
Estimation	**6h**
Sprint	Sprint 2
Description	Page de profil accessible à chaque jeune connecté : consultation des informations, modification partielle, historique de participation et statistiques personnelles.
Critères d'acceptation	✓ Page /profile avec : infos personnelles, groupes, taux de présence, historique activités ✓ Modification autorisée par le jeune : photo, email (s'il n'en a pas encore ou pour le modifier) ✓ Téléphone modifiable uniquement par l'admin (bloqué par Policy + Form Request)

	✓ Si le jeune n'a pas d'email, il peut en ajouter un depuis son profil ✓ Validation formulaire en temps réel avec Alpine.js ✓ Taux de présence calculé et affiché (présences / activités auxquelles inscrit) ✓ Liste des 10 dernières activités avec statut de présence
Commandes Artisan / CLI	`php artisan make:controller ProfileController` `php artisan make:request UpdateProfileRequest`
Conventions Laravel	✓ Utiliser $this->authorize('update', $user) dans le controller ✓ UpdateProfileRequest : 'phone' => ['sometimes', new ProhibitedUnless(auth()->user()->isAdmin())] ✓ UpdateProfileRequest : 'email' => 'sometimes|nullable|email|unique:users,email,'.auth()->id() ✓ Blade : @auth et @can pour afficher/masquer les sections selon rôle ✓ Alpine.js x-data pour la prévisualisation de photo avant upload
Bonne pratique	Permettre au jeune d'ajouter un email depuis son profil est important : cela lui débloque le canal de reset natif Laravel pour la prochaine fois. La restriction du téléphone doit être côté serveur, pas seulement en Blade.
Stratégie de test	Test : jeune modifie son téléphone → 403. Test : jeune ajoute un email → OK. Test : upload photo → redimensionnement. Test : taux de présence → calcul correct.
Dépendances	AUTH-002, MEMBER-001

EPIC-05 – Gestion des Activités

CRUD des activités, inscriptions, génération de QR Codes, validation de présence automatique et manuelle.

[ACTIV-001] CRUD des activités (Filament + Blade)

Type	Feature
Priorité	Critique
Rôle responsable	Product Owner / Dev Lead
Estimation	10h
Sprint	Sprint 2
Description	Création et gestion des activités via Filament (admin/bureau). Affichage public des activités via Blade pour les jeunes.
Critères d'acceptation	✓ Resource Filament ActivityResource : titre, description, type (Enum), date, heure début/fin, lieu, responsable, visibilité ✓ Statuts via Enum : DRAFT, PUBLISHED, CANCELLED, ARCHIVED ✓ Visibilité : ALL, GROUP, ROLE ✓ Annulation obligatoire avec motif (champ texte requis) ✓ Archivage automatique des activités passées de plus de 90 jours (Scheduled Command) ✓ Page Blade /activities listant les activités publiées du jeune connecté ✓ Event ActivityCreated dispatché → notification aux membres concernés

Commandes Artisan / CLI	`php artisan make:model Activity -m`
	`php artisan make:enum ActivityStatus`
	`php artisan make:enum ActivityType`
	`php artisan make:filament-resource Activity --generate --soft-deletes`
	`php artisan make:event ActivityCreated`
	`php artisan make:command ArchiveOldActivities`
	`php artisan schedule:list (vérifier le cron)`

Conventions Laravel	✓ Enregistrer ArchiveOldActivities dans routes/console.php (Laravel 11) : Schedule::command(...)->daily() ✓ Utiliser Filament Repeater ou Select pour la visibilité ciblée ✓ Blade : @forelse($activities as $activity) avec message vide élégant ✓ Utiliser ActivityResource::getEloquentQuery() pour filtrer selon les permissions de l'utilisateur
Bonne pratique	Un Event ActivityCreated découple la création de l'activité de l'envoi de notifications. Ne jamais appeler le service de notification directement dans le controller.
Stratégie de test	Test : création → Event dispatché. Test : annulation sans motif → erreur de validation. Test : cron archivage → activités > 90j archivées.
Dépendances	MEMBER-001, PERM-002

[ACTIV-002] Inscriptions aux activités (côté jeune)

Type	Feature
Priorité	Critique
Rôle responsable	Product Owner / Dev Lead

Estimation	**8h**
Sprint	Sprint 3
Description	Les jeunes s'inscrivent aux activités avec un statut (présent/absent justifié/incertain). Modification possible jusqu'à 2h avant l'activité.
Critères d'acceptation	✓ Table registrations : user_id, activity_id, status (Enum : PRESENT, ABSENT_JUSTIFIED, UNCERTAIN), justification, registered_at ✓ Page Blade de détail d'une activité avec formulaire d'inscription ✓ Statuts via Enum RegistrationStatus ✓ Modification bloquée 2h avant le début de l'activité ✓ Places limitées gérées (champ capacity sur Activity) avec liste d'attente ✓ Notification de confirmation envoyée via Job après inscription ✓ Rappel automatique 24h avant l'activité (Scheduled Job)

Commandes Artisan / CLI	`php artisan make:model Registration -m`
	`php artisan make:enum RegistrationStatus`
	`php artisan make:controller RegistrationController`
	`php artisan make:request StoreRegistrationRequest`
	`php artisan make:job SendRegistrationConfirmation`
	`php artisan make:job SendActivityReminder`

Conventions Laravel	✓ Utiliser DB::transaction() pour gérer les places limitées (éviter race condition) ✓ Vérifier la date limite dans la Form Request : rule('activity_id', [new RegistrationDeadline]) ✓ Custom Rule : RegistrationDeadline vérifie activity.start_time - 2h ✓ Rappel 24h : dans routes/console.php → Schedule::job(SendActivityReminder::class)->daily()
Bonne pratique	La vérification de la date limite doit être dans une Custom Rule réutilisable, pas dans le controller. Utiliser une transaction DB pour les places limitées.
Stratégie de test	Test : inscription après fermeture → 422. Test : double inscription même activité → erreur. Test : place limite atteinte → liste d'attente.
Dépendances	ACTIV-001, AUTH-002

[ACTIV-003] Génération et gestion des QR Codes

Type	**Feature**
Priorité	**Critique**
Rôle responsable	Dev Lead
Estimation	**6h**
Sprint	Sprint 3
Description	Générer un QR Code sécurisé et unique par activité, avec token signé et expiration. Téléchargeable en PDF depuis Filament.
Critères d'acceptation	✓ Chaque activité publiée génère un QR Code unique à la demande ✓ Token signé via URL::temporarySignedRoute() ou HMAC-SHA256 custom ✓ Paramètres : activity_id, expires_at (configurable, défaut : fin de l'activité + 30min) ✓ QR Code affiché dans Filament avec timer de validité restante (Alpine.js) ✓ Téléchargement PDF du QR Code via Action Filament

	✓ Révocation possible : champ qr_token_version incrémenté → anciens tokens invalides ✓ Régénération possible par l'Admin (nouveau token, ancien révoqué)

Commandes Artisan / CLI	`composer require simplesoftwareio/simple-qrcode`
	`composer require barryvdh/laravel-dompdf`
	`php artisan make:action GenerateActivityQrCode`
	`php artisan make:filament-action DownloadQrCodePdf`

Conventions Laravel	✓ URL::temporarySignedRoute('attendance.scan', now()->addHours(3), ['activity' => $id]) ✓ Vérifier avec Request::hasValidSignature() dans le controller de scan ✓ Pour la révocation : ajouter qr_version en paramètre du token et vérifier en DB ✓ Alpine.js x-data + setInterval pour le compte à rebours de validité
Bonne pratique	Utiliser les Signed URLs Laravel plutôt qu'un système de token custom : c'est intégré, testé et maintenu par le framework.
Stratégie de test	Test : URL signée expirée → 403. Test : URL modifiée manuellement → 403. Test : révocation → ancienne URL → 403.
Dépendances	ACTIV-001

[ACTIV-004] Validation de présence via QR Code (côté jeune)

Type	**Feature**
Priorité	**Critique**
Rôle responsable	Dev Lead
Estimation	**8h**
Sprint	Sprint 3
Description	Le jeune scanne le QR Code avec son smartphone. La présence est validée automatiquement via une page Blade optimisée mobile avec scanner JS.
Critères d'acceptation	✓ Page mobile /scan accessible uniquement aux utilisateurs connectés ✓ Scanner QR intégré en Blade avec bibliothèque JS (html5-qrcode) ✓ Appel AJAX vers l'endpoint de validation (route signée) ✓ Statut de présence mis à jour : PRESENT ou LATE (si après heure de début + 15min) ✓ Feedback visuel immédiat : vert (validé), rouge (erreur), orange (déjà scanné) ✓ Endpoint idempotent : double scan → même réponse, pas de doublon ✓ Log du scan : user_id, activity_id, scanned_at, ip_address

Commandes Artisan / CLI	`php artisan make:controller AttendanceController`
	`php artisan make:model Attendance -m`
	`php artisan make:enum AttendanceStatus`

Conventions Laravel	✓ Route : Route::get('/scan/{activity}', ...)->middleware(['auth', 'signed']) ✓ Utiliser updateOrCreate() pour l'idempotence : Attendance::updateOrCreate(['user_id'=>..,'activity_id'=>..], [...]) ✓ Retourner une JsonResponse pour la réponse AJAX du scanner ✓ Blade @push('scripts') pour insérer le JS html5-qrcode uniquement sur cette page

Bonne pratique	L'idempotence via updateOrCreate() garantit qu'un double scan ne crée pas deux entrées. L'endpoint doit vérifier la signature ET l'authenticité de l'utilisateur.
Stratégie de test	Test : scan valide → Attendance créée. Test : double scan → une seule entrée, réponse 200. Test : scan expiré → 403. Test : scan autre utilisateur → 403.
Dépendances	ACTIV-003, AUTH-002

[ACTIV-005] Validation manuelle des présences (Chef de groupe)

Type	Feature
Priorité	Haute
Rôle responsable	Product Owner / Dev Lead
Estimation	6h
Sprint	Sprint 3
Description	Interface Blade temps réel pour le chef de groupe pour valider/modifier les présences de son groupe lors d'une activité.
Critères d'acceptation	✓ Page /activities/{activity}/attendance affichant les membres du groupe du chef ✓ Actions par membre : PRESENT / LATE / ABSENT / EXCUSED + note ✓ Mise à jour en temps réel via Alpine.js + fetch (polling 30s ou Livewire) ✓ Statut visuel : QR scanné (vert automatique) vs validé manuellement (bleu) ✓ Modification possible jusqu'à la clôture de l'activité (1h après fin) ✓ Accès refusé si l'activité ne concerne pas son groupe (Policy) ✓ Log d'audit : chaque validation manuelle tracée
Commandes Artisan / CLI	`php artisan make:controller AttendanceManagementController` `php artisan make:request UpdateAttendanceRequest`
Conventions Laravel	✓ Utiliser $this->authorize('manageAttendance', $activity) dans le controller ✓ Policy : vérifier que $user->groups->contains($activity->group_id) ou que l'activité est ouverte à tous ✓ Alpine.js : x-data avec fetch() pour les mises à jour sans rechargement de page ✓ Différencier scan_source : 'qr_code' vs 'manual' dans la table attendances
Bonne pratique	La différenciation QR/Manuel dans la DB permet de produire des statistiques de qualité (combien de présences auto vs manuelles). Ne pas la négliger.
Stratégie de test	Test : chef groupe A → voit seulement son groupe. Test : validation manuelle → log audit créé. Test : activité clôturée → modification refusée.
Dépendances	ACTIV-004, PERM-002, MEMBER-002

EPIC-06 – Notifications & Communication

Jobs Laravel pour WhatsApp, notifications DB in-app, interface d'envoi ciblé avec Filament.

[NOTIF-001] Service WhatsApp via Laravel Jobs & Queues

Type	**Tech**
Priorité	**Haute**
Rôle responsable	Dev Lead
Estimation	**8h**
Sprint	Sprint 3
Description	Intégrer l'API WhatsApp Business (Twilio ou Meta Cloud API) via des Jobs Laravel dispatchés en queue pour tous les envois de messages.
Critères d'acceptation	✓ Service WhatsAppService injecté par interface (contrat) pour faciliter le changement de provider ✓ Jobs : SendWhatsAppCredentials, SendWhatsAppNotification, SendActivityReminder ✓ Queue configurée avec driver 'database' (ou Redis si disponible) ✓ Retry : 3 tentatives avec backoff exponentiel (backoff() method) ✓ Table failed_jobs pour les messages échoués ✓ Log de tous les envois en DB (table whatsapp_logs) ✓ Rate limiting respectant les quotas API du provider

Commandes Artisan / CLI	`php artisan queue:table && php artisan migrate`
	`php artisan make:job SendWhatsAppNotification`
	`php artisan make:interface WhatsAppServiceInterface`
	`php artisan queue:work --tries=3 (production : Supervisor)`
	`php artisan queue:failed (superviser les échecs)`
	`php artisan queue:retry all (relancer les échoués)`

Conventions Laravel	✓ Implémenter ShouldQueue + Queueable + SerializesModels dans chaque Job ✓ Configurer Supervisor sur le serveur pour queue:work en production ✓ Utiliser $this->release(60) dans le Job pour différer en cas d'erreur API temporaire ✓ Configurer QUEUE_RETRY_AFTER dans .env selon les timeouts API
Bonne pratique	Toujours utiliser une interface pour le WhatsAppService. En test, une implémentation FakeWhatsAppService permettra de ne pas appeler l'API réelle.
Stratégie de test	Test : Job dispatché → message dans la queue. Test : échec API simulé → retry déclenché. Test : 4ème tentative → failed_jobs.
Dépendances	TECH-001

[NOTIF-002] Notifications in-app (Laravel Notifications + DB)

Type	**Feature**
Priorité	**Haute**
Rôle responsable	Dev Lead
Estimation	**6h**

Sprint	Sprint 4
Description	Système de notifications in-app via le channel 'database' de Laravel Notifications. Cloche de notification dans le header Blade avec compteur.
Critères d'acceptation	✓ Table notifications créée (php artisan notifications:table) ✓ Notifications Eloquent pour : nouvelle activité, rappel, validation présence, message admin ✓ Header Blade : cloche avec badge compteur (unread_count), mise à jour via Alpine.js ✓ Page /notifications avec liste paginée, statut lu/non lu ✓ Marquage comme lu au clic (AJAX) ✓ Suppression possible des notifications ✓ Auto-nettoyage des notifications > 90 jours (Scheduled Command)

Commandes Artisan / CLI	`php artisan notifications:table && php artisan migrate`
	`php artisan make:notification NewActivityNotification`
	`php artisan make:notification AttendanceValidatedNotification`
	`php artisan make:command CleanOldNotifications`

Conventions Laravel	✓ via() method : return ['database'] ou ['database', 'broadcast'] ✓ Alpine.js dans le header : fetch('/notifications/unread-count') toutes les 60s ✓ Utiliser $user->unreadNotifications->count() pour le badge ✓ Route : Route::patch('/notifications/{id}/read', ...) pour marquer comme lu
Bonne pratique	Séparer les channels (database vs whatsapp) dans la méthode via() pour une flexibilité maximale. L'utilisateur peut désactiver certains channels depuis son profil.
Stratégie de test	Test : nouvelle activité → notification DB créée. Test : marquage lu → updated_at renseigné. Test : count badge → décrémenté après lecture.
Dépendances	NOTIF-001, AUTH-002

[NOTIF-003] Interface d'envoi de notifications ciblées (Filament)

Type	**Feature**
Priorité	**Haute**
Rôle responsable	Product Owner / Dev Lead
Estimation	**8h**
Sprint	Sprint 4
Description	Interface Filament permettant à l'Admin et aux responsables autorisés d'envoyer des notifications manuelles ciblées avec programmation.
Critères d'acceptation	✓ Page Filament 'Envoyer une notification' avec sélection de cible (tous / groupe / rôle / individu) ✓ Champs : titre, contenu, canal (WhatsApp / in-app / les deux), date d'envoi (immédiat ou programmé) ✓ Programmation via Laravel Scheduler ou Job avec delay() ✓ Annulation possible si programmée et non envoyée ✓ Historique des notifications envoyées visible dans Filament ✓ Permissions : chef de groupe → seulement son groupe (vérifié par Policy) ✓ Log d'audit de chaque envoi

Commandes Artisan / CLI	`php artisan make:filament-page SendNotification`
	`php artisan make:job SendScheduledNotification`
	`php artisan make:model ScheduledNotification -m`

Conventions Laravel	✓ Utiliser dispatch($job)->delay(now()->addMinutes(30)) pour la programmation ✓ Filament : utiliser un Wizard multi-étapes pour l'envoi (cible → contenu → canal → confirmation) ✓ Filtrer les destinataires disponibles selon les permissions de l'expéditeur (scoped dans la query) ✓ Stocker les notifications programmées en DB pour permettre l'annulation
Bonne pratique	Un chef de groupe ne doit JAMAIS voir les membres d'autres groupes dans le sélecteur de destinataires. Ce filtrage doit être côté serveur dans la query Eloquent.
Stratégie de test	Test : chef groupe → destinataires filtrés à son groupe. Test : programmation → envoi à l'heure. Test : annulation avant envoi → Job supprimé.
Dépendances	NOTIF-001, NOTIF-002, PERM-002

EPIC-07 – Statistiques & Rapports

Dashboards Filament, statistiques de présence, exports PDF/Excel avec Laravel.

[STATS-001] Dashboard global Filament (KPIs & graphiques)

Type	Feature
Priorité	Haute
Rôle responsable	Product Owner / Dev Lead
Estimation	10h
Sprint	Sprint 4
Description	Tableau de bord Filament avec widgets : KPIs, graphiques d'évolution, top membres, alertes groupes en baisse.
Critères d'acceptation	✓ Widget StatsOverview : membres actifs, taux présence 30j, activités ce mois, nouveaux membres ✓ Widget Chart (Filament Charts) : évolution présences sur 6 mois ✓ Widget Table : dernières activités avec taux de présence ✓ Widget : top 5 membres les plus assidus ✓ Alerte : groupes avec taux de présence < 50% sur 30j ✓ Filtres : période, groupe (selon permissions) ✓ Cache des statistiques (Cache::remember() TTL 5min)

Commandes Artisan / CLI	`php artisan make:filament-widget StatsOverview --stats-overview`
	`php artisan make:filament-widget AttendanceChart --chart`
	`php artisan make:filament-widget RecentActivitiesTable --table`
	`composer require filament/widgets`

Conventions Laravel	✓ Filament StatsOverview : Stat::make('Membres actifs', User::active()->count()) ✓ Utiliser Cache::remember('stats.global', 300, fn() => ...) pour les calculs lourds ✓ Filament Charts : LineChart ou BarChart selon les données ✓ Scoper les widgets selon les permissions : protected static function canView(): bool
Bonne pratique	Calculer les statistiques lourdes dans des Artisan Commands via un cron (toutes les heures) et stocker le résultat en cache. Ne jamais calculer à la volée en Dashboard.
Stratégie de test	Test : données cache → rafraîchies après TTL. Test : widget filtré par groupe → données cohérentes. Test : taux présence calculé correctement.
Dépendances	ACTIV-004, MEMBER-002

[STATS-002] Export de rapports PDF & Excel

Type	Feature
Priorité	Moyenne
Rôle responsable	Product Owner / Dev Lead
Estimation	8h
Sprint	Sprint 4

Description	Génération et téléchargement de rapports en PDF et Excel pour les membres, présences et statistiques.
Critères d'acceptation	✓ Export liste membres : Excel via maatwebsite/excel avec colonnes filtrables ✓ Export rapport de présence par activité : PDF via barryvdh/laravel-dompdf avec mise en page ✓ Export rapport mensuel : PDF avec statistiques et graphiques basiques ✓ Exports lourds (> 500 lignes) générés en arrière-plan (Job) avec notification quand prêt ✓ Fichier stocké sur le disk 'local' avec URL signée valable 1h ✓ Permissions : exports globaux admin/bureau, exports groupe chef de groupe

Commandes Artisan / CLI	`composer require maatwebsite/excel`
	`composer require barryvdh/laravel-dompdf`
	`php artisan make:export MembersExport --model=User`
	`php artisan make:job GenerateMembersReport`
	`php artisan make:filament-action ExportMembersAction`

Conventions Laravel	✓ maatwebsite/excel : implémenter FromQuery + WithHeadings + WithMapping ✓ Pour les gros exports : implémenter ShouldQueue sur l'export class ✓ URL signée : Storage::temporaryUrl('reports/file.pdf', now()->addHour()) ✓ DomPDF view : resources/views/reports/attendance.blade.php avec @foreach
Bonne pratique	Nettoyer automatiquement les fichiers d'export après 24h (Scheduled Command). Stocker les exports en 'local' disk (pas 'public') et toujours utiliser des URLs signées.
Stratégie de test	Test : export CSV → colonnes et données correctes. Test : export PDF → permissions vérifiées. Test : export > 500 lignes → Job dispatché, notification reçue.
Dépendances	STATS-001, PERM-002

EPIC-08 – Tests, Qualité & Déploiement

Tests PHPUnit/Pest, revue de sécurité Laravel, déploiement optimisé en production.

[QA-001] Tests Feature & Unit (PHPUnit / Pest)

Type	QA
Priorité	Haute
Rôle responsable	Testeur / Dev Lead
Estimation	16h
Sprint	Continu
Description	Écrire et maintenir les tests automatisés pour tous les modules critiques avec Pest (ou PHPUnit).
Critères d'acceptation	✓ Couverture de code ≥ 80% sur les services et controllers critiques ✓ Tests Feature Auth : login par email, login par téléphone, login identifiant inconnu ✓ Tests Feature Auth : reset par email (lien natif), reset par téléphone (workflow WhatsApp) ✓ Tests Feature Auth : création compte email seul, téléphone seul, les deux, aucun → erreur ✓ Tests Feature : Permissions, Activities, Registrations, QR Scan ✓ Tests Unit : WhatsAppService, PasswordResetService (choix du canal), PasswordGenerator ✓ Utilisation de Factories pour toutes les données de test ✓ DB de test : SQLite in-memory (phpunit.xml configuré) ✓ Rapport de couverture généré en CI (--coverage-clover)

Commandes Artisan / CLI	`composer require pestphp/pest --dev`
	`php artisan test --coverage`
	`php artisan make:factory UserFactory --model=User`
	`php artisan make:factory ActivityFactory --model=Activity`
	`php artisan test --filter=AuthTest`

Conventions Laravel	✓ Utiliser RefreshDatabase dans chaque test Feature pour une DB propre ✓ actingAs($user) pour simuler l'authentification ✓ withoutExceptionHandling() pour débugger les tests qui masquent les erreurs ✓ Pest : it('peut se connecter avec son email', fn() => ...) et it('peut se connecter avec son téléphone', fn() => ...) ✓ Utiliser fake() ou Mockery pour mocker WhatsAppService sans appeler l'API réelle
Bonne pratique	Tester les deux canaux d'authentification et de reset séparément. Créer une Factory avec states : User::factory()->withEmailOnly(), withPhoneOnly(), withBoth().
Stratégie de test	Auto-référentiel.
Dépendances	TECH-002

[QA-002] Audit de sécurité & durcissement Laravel

Type	QA
Priorité	Haute

Rôle responsable	CTO / Dev Lead
Estimation	**8h**
Sprint	Sprint Final
Description	Audit de sécurité complet selon les best practices Laravel et OWASP : headers, CSRF, injection, permissions, secrets.
Critères d'acceptation	✓ Headers de sécurité configurés via middleware (X-Frame-Options, X-Content-Type-Options, Referrer-Policy) ✓ HTTPS forcé (HSTS en production, config/trustedproxies) ✓ CORS configuré dans config/cors.php (whitelist des origines) ✓ Vérification : aucune donnée sensible dans les logs (config/logging.php) ✓ Audit npm : npm audit + composer audit sans vulnérabilité critique ✓ Vérification : APP_DEBUG=false en production ✓ Rate limiting appliqué sur tous les endpoints d'authentification ✓ Validation de toutes les entrées via Form Requests (aucun input non validé)

Commandes Artisan / CLI	`composer audit`
	`npm audit`
	`php artisan route:list (vérifier qu'aucune route sensible n'est exposée)`
	`php artisan config:show (vérifier APP_DEBUG, APP_ENV)`

Conventions Laravel	✓ Utiliser SecureHeaders middleware ou spatie/laravel-csp pour les headers CSP ✓ config('app.debug') doit être false en production via .env APP_DEBUG=false ✓ Utiliser $request->validate() ou Form Requests sur TOUTES les entrées ✓ Vérifier que les routes admin sont toutes protégées par le middleware auth + can
Bonne pratique	L'audit de sécurité doit être exécuté avant chaque déploiement en production. Créer une checklist sécurité dans le CONTRIBUTING.md du projet.
Stratégie de test	Rapport d'audit produit, relu et signé par le CTO. Checklist OWASP Top 10 complétée.
Dépendances	Tous les EPICs précédents

[QA-003] Déploiement en production & optimisations Laravel

Type	**Config**
Priorité	**Haute**
Rôle responsable	CTO / Dev Lead
Estimation	**6h**
Sprint	Sprint Final
Description	Configurer le déploiement en production avec toutes les optimisations Laravel, Supervisor pour les queues et les variables d'environnement sécurisées.
Critères d'acceptation	✓ Script de déploiement documenté (deploy.sh ou GitHub Action production) ✓ Commandes de déploiement : migrate --force, config:cache, route:cache, view:cache, optimize ✓ Supervisor configuré pour queue:work avec restart automatique ✓ Cron Laravel configuré (crontab : * * * * * php artisan schedule:run) ✓ Variables d'environnement en production via le serveur (pas de .env versionné)

	✓ Rollback documenté en cas de problème ✓ Monitoring basique : Laravel Telescope en staging uniquement

Commandes Artisan / CLI	`php artisan optimize (config + route + view cache en une commande)`
	`php artisan migrate --force (production)`
	`php artisan queue:restart (après déploiement)`
	`php artisan schedule:run (cron)`
	`composer require laravel/telescope --dev (staging uniquement)`

Conventions Laravel	✓ Ne jamais cacher les routes en production avant de vérifier qu'elles sont toutes correctes ✓ php artisan optimize:clear avant php artisan optimize lors d'un déploiement ✓ Utiliser php artisan down (maintenance mode) pendant les migrations longues ✓ Supervisor config : autostart=true, autorestart=true, numprocs=2 (selon charge)
Bonne pratique	Documenter chaque étape du déploiement. Tester le rollback en staging avant la première mise en production. Garder les anciens releases pendant 48h avant nettoyage.
Stratégie de test	Déploiement testé intégralement sur staging. Rollback simulé et validé. Queue:work vérifié via Supervisor status.
Dépendances	QA-001, QA-002

Conventions Laravel & Règles du Projet

Conventions de code

Style : PSR-12 enforced via Laravel Pint (composer pint). Pint lancé en pre-commit hook.

Branches : feature/TICKET-ID-titre · fix/TICKET-ID-titre · chore/description

Commits : Conventional Commits → feat(auth): redirect PENDING users to password change

PR obligatoire pour tout merge. Review par 1 pair minimum. CI doit passer avant merge.

Architecture Laravel

Controllers minces : uniquement la validation (Form Request), l'autorisation (Policy) et l'appel au Service.

Logique métier dans les Services ou Actions (app/Actions/). Jamais dans les Controllers ni les Models.

Events & Listeners pour le découplage : toute action ayant des effets secondaires (notification, audit) passe par un Event.

Jobs pour toute tâche asynchrone : envoi WhatsApp, génération de rapport, rappels.

Sécurité

Toute entrée utilisateur est validée via une Form Request dédiée. Jamais de $request->all() sans validation.

$this->authorize() dans chaque méthode de controller. Les Policies centralisent toutes les règles d'autorisation.

Aucun secret en code source. Variables sensibles uniquement dans .env (non versionné).

APP_DEBUG=false en production. php artisan optimize après chaque déploiement.

Définition of Done (DoD) — Un ticket est TERMINÉ si :

✅ Code implémenté, reviewé et mergé sur main

✅ Form Requests validant toutes les entrées

✅ Policy ou Gate appliqué sur chaque action sensible

✅ Tests Feature/Unit écrits et passants (couverture ≥ 80% sur le module)

✅ Log d'audit déclenché pour toute mutation critique

✅ Déployé sur staging et validé fonctionnellement

✅ Documentation mise à jour si impact sur l'API ou l'architecture