ÉCRIT DE FIN D'ÉTUDES

Giggr.

Plateforme de mise en relation entre musiciens.

Démarche de conception et de réalisation.

Hugo Girona — Bachelier en techniques graphiques, orientation Web.

HEPL
Haute Ecole de la Province de Liège

Techniques infographiques
PFE - Dossier écrit

Sommaire

AVANT L'ÉCRIT

Ressources & produit

Giggr. est une plateforme web de mise en relation entre musiciens. Les livrables ci-dessous donnent accès au produit déployé, au dossier de design, au code source et au dossier qualité qui atteste objectivement de ce que cet écrit avance.

Produit déployé	giggr.be
Identifiants de test	`dvilain@giggr.be / dvilain0626` `fparmentier@giggr.be / fparmentier0626`
Dossier de design (Figma)	figma.com/design/Giggr — PFE
Code source — application	github.com/hugogirona/pfe
Code source — documentation	github.com/hugogirona/hugo-girona-doc-PFE
Documentation technique (dossier qualité)	doc.giggr.be

Les deux livrables sont déployés sur la même infrastructure **Laravel Forge**, sous le domaine `giggr.be` : l'application sur le domaine racine, le site de documentation sur le sous-domaine `doc.giggr.be`. Un choix délibéré : que le dossier qualité vive aux côtés du produit qu'il documente, plutôt que dispersé sur un hébergement tiers.

PRÉSENTATION DU PROJET ET DE SON ESPRIT

1. Note d'intention

> « Concevoir l'outil que j'aurais voulu trouver quand je cherchais des musiciens — et le concevoir avec l'exigence que j'attends d'un produit professionnel. »

Giggr est une plateforme web de mise en relation entre musiciens. Elle répond à une situation que je connais de l'intérieur : lorsqu'on veut monter ou rejoindre un groupe, l'information est dispersée — un message dans un groupe Facebook par-ci, une petite annonce par-là — sans moyen de filtrer par instrument ou par ville, et sans cadre clair pour entrer en contact. Giggr met de l'ordre dans cette pratique éclatée : il centralise des profils de musiciens et des annonces, cherchables par instrument et par localité, et fournit une messagerie intégrée pour prendre contact sans avoir à donner son numéro.

Trois usages structurent le produit, et toute l'interface est pensée autour d'eux : trouver un groupe à rejoindre, trouver des musiciens pour le sien, et publier une annonce. Ce ne sont pas des fonctionnalités empilées au hasard — ce sont les trois intentions réelles avec lesquelles un musicien arrive sur le site, et chacune dispose de son propre parcours, de la page d'accueil jusqu'à la conversation.

L'esprit dans lequel j'ai abordé le projet tient en deux mots : **accessible** et **structuré**. Accessible, parce que je voulais qu'aucun utilisateur ne soit laissé de côté — ni au clavier, ni au lecteur d'écran, ni sur un petit écran. Structuré, parce que je crois qu'un produit se juge autant à la cohérence de ses fondations qu'à ce qu'il affiche : un design system tenu, une hiérarchie de l'information claire, un code que quelqu'un d'autre pourrait reprendre sans mener l'enquête. C'est cette exigence-là que j'ai voulu démontrer, du premier jeton de couleur jusqu'au dernier test qui l'atteste.

Enfin, j'ai voulu que cette qualité ne soit pas une affirmation, mais une preuve. C'est tout le sens du site de documentation technique qui accompagne le produit : il rassemble les mesures, les audits et les tests — accessibilité, performances, couverture, conformité Opquast, tests utilisateurs — qui attestent objectivement de ce que cet écrit avance. Je n'y annonce rien que je ne puisse montrer.

LE FIL CONDUCTEUR DU TRAVAIL

2. Motivation personnelle — le fil conducteur

> **Fil conducteur :** Concevoir une expérience accessible et rigoureusement structurée — du design system jusqu'au code — pour un utilisateur que je connais de l'intérieur.

Le point de départ de ce projet n'est pas le thème musical en soi. Je suis moi-même musicien, et j'ai vécu ce besoin un peu frustrant de chercher un binôme ou un groupe avec qui jouer ; j'ai donc longtemps été l'utilisateur de ce type de service avant d'en devenir le concepteur. Je ne le présente pas comme un hobby qui aurait choisi mon sujet à ma place, mais comme un avantage de conception : pouvoir partir d'un usage réel et éprouvé plutôt que d'un public imaginé de l'extérieur. Concevoir pour un utilisateur qu'on connaît de l'intérieur, c'est se donner une boussole — à chaque arbitrage, je pouvais me demander honnêtement ce que j'aurais attendu, moi, à cet endroit précis du parcours.

Ce que je voulais réellement démontrer avec ce projet, c'est moins une fonctionnalité qu'une **exigence** : qu'une expérience peut être à la fois accessible et rigoureusement structurée, du design system jusqu'au code. C'est là que je me situe le mieux professionnellement, et c'est une conviction autant qu'une préférence — je crois qu'un produit se juge à la cohérence de ses fondations, pas seulement à ce qu'il montre. C'est donc le fil conducteur que j'annonce ici, et qui sert de grille de lecture à tout le reste de cet écrit comme au produit lui-même.

Annoncer une exigence crée une attente, et je ne veux avancer que ce que je peux montrer. La trace de cette exigence se trouve, concrètement, à trois endroits. Une accessibilité mesurée, d'abord : un niveau WCAG AA visé et contrôlé par axe-core, Lighthouse et un contrôle des contrastes — zéro violation sur les pages publiques, des cibles tactiles et des contrastes vérifiés. Un design system explicite, ensuite : des jetons de couleur et de typographie, des composants réutilisés plutôt que redessinés. Une architecture de l'information, enfin, pensée page par page, du repère de navigation jusqu'à la hiérarchie des titres. Rien de tout cela n'est affirmé au doigt mouillé : c'est audité et prouvé dans le dossier de documentation technique qui accompagne le produit.

Sur le plan professionnel, ce projet m'a fait franchir une marche que j'avais identifiée bien avant de commencer. Le back-end était mon point le moins solide ; je l'écrivais déjà noir sur blanc dans mon rapport de stage, où je regrettais de ne pas avoir pu travailler avec les

WebSockets et où je me promettais de m'y former seul pour bâtir une messagerie en temps réel dans ce projet de fin d'études. Giggr est l'aboutissement de cette promesse : j'y ai conçu une messagerie temps réel — diffusion d'événements côté serveur, serveur Reverb, réception par Laravel Echo — que je ne savais pas faire il y a quelques mois. J'y ai passé un temps considérable, sur le temps réel comme sur l'optimisation des performances, et j'en ressors avec une maîtrise de Livewire et des WebSockets que je n'avais pas. Ce que je ne savais pas faire est devenu ce que je sais faire, et je peux le démontrer.

C'est, au fond, ma réponse à la question posée : ce projet m'a fait évoluer comme professionnel du web en me forçant à tenir, seul et de bout en bout, une exigence de qualité que je n'avais jusque-là surtout fait qu'observer — du premier jeton de couleur jusqu'à la dernière requête optimisée.

CAHIER DES CHARGES — POINT DE VUE DU COMMANDITAIRE

3. Définition des objectifs du projet

> « Avant de parler d'un site, il faut parler d'un problème bien réel : des musiciens qui veulent jouer ensemble et qui n'arrivent pas à se trouver. »

Le besoin, indépendamment du site

Le besoin que Giggr vient servir existe sans le numérique, et c'est par lui qu'il faut commencer. Des musiciens amateurs — un guitariste qui cherche un groupe, une chanteuse qui veut monter un projet, un batteur qui débarque dans une ville — veulent une chose simple : jouer avec d'autres. Mais pour jouer ensemble, il faut d'abord se trouver, et c'est précisément là que ça coince. Aujourd'hui, cette mise en relation se fait de façon dispersée et informelle : des messages postés dans des groupes Facebook locaux, quelques petites annonces éparpillées sur des sites généralistes. L'information n'est ni centralisée, ni filtrable par instrument ou par ville, ni structurée — elle est noyée dans un fil, et le contact suppose souvent de livrer ses coordonnées personnelles à un inconnu.

Le commanditaire — ici, la communauté des musiciens amateurs — a donc besoin d'un endroit unique où cette rencontre devient possible, rapide et cadrée. Ce n'est pas un besoin de « site sur la musique » ; c'est un besoin de **mise en relation locale et qualifiée**.

Ce que ce besoin implique pour l'interface

Décrit ainsi, le besoin dicte directement les fonctionnalités. Centraliser suppose un lieu unique — une page d'exploration qui rassemble à la fois les profils de musiciens et les annonces. Qualifier suppose des **filtres** au cœur de l'expérience, par instrument et par ville, puisque ce sont les deux critères avec lesquels un musicien pense réellement. Cadrer la rencontre suppose des annonces typées (« je cherche », « je propose ») qui disent d'emblée l'intention, et une messagerie intégrée pour prendre contact sans échanger de numéro. Chaque besoin se traduit ainsi en un élément d'interface précis : aucune fonctionnalité « parce que c'est joli », chacune répond à un point du problème réel.

Priorisation des objectifs

Toutes les fonctionnalités ne se valent pas, et il fallait choisir ce que l'interface met le plus en avant. L'**objectif de conversion** du produit est unique et clair : qu'une mise en relation aboutisse, c'est-à-dire qu'un premier message parte. Tout le reste est au service de cette action.

C'est ce qui justifie la hiérarchie graphique du produit. La recherche en est l'action centrale : sur l'exploration, les filtres instrument et ville sont immédiatement accessibles, pour qu'on puisse qualifier sa demande sans avoir à réfléchir. Le bouton « Contacter » en est le point d'arrivée — traité comme l'action principale partout où il apparaît, sur un profil comme sur une annonce, puisque c'est lui qui déclenche la conversion. Viennent juste après les gestes qui alimentent la base et rendent les recherches des autres fructueuses : publier une annonce, compléter son profil. Tout le reste — paramètres, pages institutionnelles — demeure volontairement discret.

En clair, le produit est conçu pour amener, le plus directement possible, deux musiciens à s'écrire.

PROFILS D'UTILISATION ET TÂCHES

4. Définition du public cible

> « Mon public ne se définit pas par un âge, mais par une intention : trois manières d'arriver sur Giggr, donc trois parcours à servir. »

Après les objectifs du commanditaire, viennent ceux des utilisateurs. La tentation serait de décrire une tranche d'âge — ce serait inutile : dire « des musiciens de 18 à 35 ans » n'aide en rien à concevoir l'interface. Ce qui compte, c'est ce qui les fait venir et la tâche précise qu'ils veulent accomplir. J'ai donc raisonné par **profils d'utilisation**, c'est-à-dire par intentions, et synthétisé la recherche utilisateur en trois personas. Leur version détaillée — contexte, frustrations, besoins et parcours pas à pas — figure dans le site de documentation ; j'en retiens ici l'essentiel pour la conception.

Trois intentions, trois personas

Léa, guitariste, veut « trouver un groupe ». Elle joue depuis quelques années et cherche à rejoindre un projet près de chez elle ; sa frustration tient à des annonces éparpillées, sans filtre par instrument ni par ville. Ce qu'il lui faut : repérer vite les groupes en quête d'un guitariste dans sa région, et les contacter sans livrer son numéro.

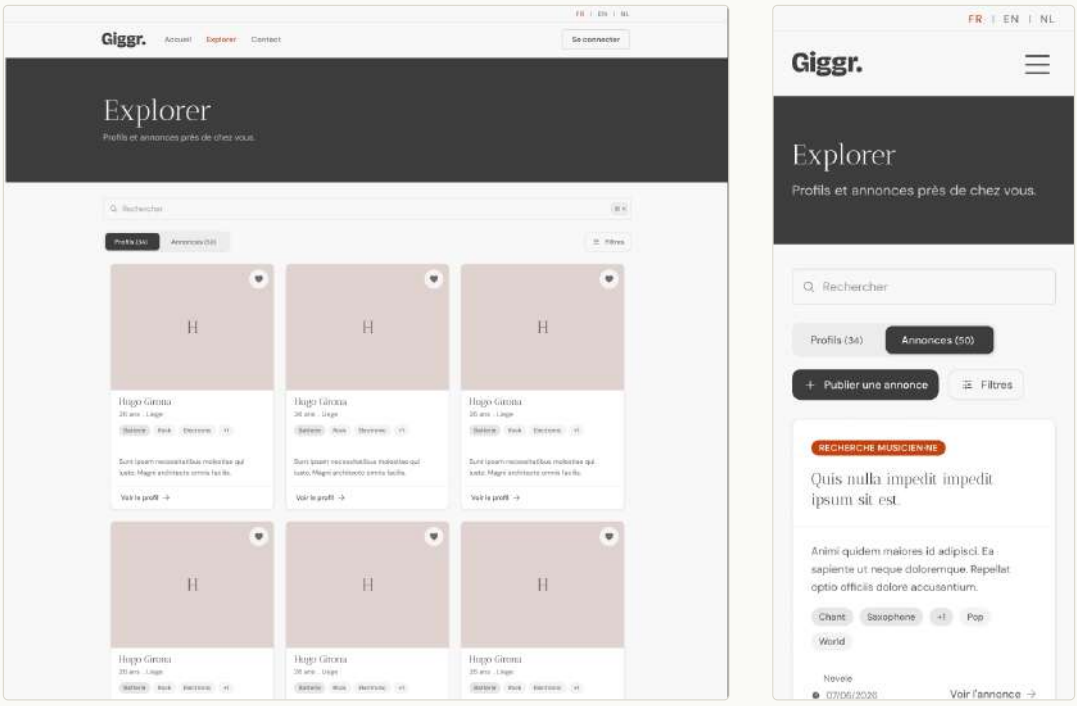
Marc, batteur, veut « publier une annonce ». Il monte un projet et cherche des musiciens, mais partout où il poste, son annonce est vite noyée et ne lui vaut que des contacts hors sujet. Ce qu'il lui faut : une annonce claire — instruments recherchés, ville — qui ne lui amène que des contacts pertinents.

Sophie, chanteuse, veut « trouver des musiciens ». Elle part de zéro pour composer un groupe et ne trouve nulle part centralisé où chercher par instrument. Ce qu'il lui faut : parcourir des profils filtrés par instrument et par ville, puis entamer la conversation sans effort.

Ces trois personas ne sont pas décoratifs : chacun correspond à un **parcours de référence** que l'on retrouve tel quel dans l'interface, et qui a servi de scénario aux tests utilisateurs.

Ce que ces besoins impliquent sur l'interface

Pris ensemble, ces profils dessinent les priorités de conception, presque page par page. Parce que Léa et Sophie pensent d'abord « instrument et ville », les filtres doivent être visibles et combinables d'emblée sur l'exploration, jamais relégués derrière un réglage avancé. Parce que cette même exploration mêle deux objets distincts — des musiciens et des annonces —, il faut une bascule claire entre les deux pour ne pas désorienter. Parce que Marc attend des contacts qualifiés, le formulaire d'annonce doit imposer le type (« je cherche » ou « je propose »), l'instrument et la ville, ce qui cadre la demande à la source. Et parce que l'aboutissement commun des trois reste la prise de contact, le bouton « Contacter » doit être l'élément le plus mis en valeur de chaque profil et de chaque annonce.



L'autre vue de l'exploration : les annonces, du desktop au mobile. Une bascule claire fait passer des profils de musiciens aux annonces, sans quitter la page ni perdre ses filtres.

C'est là que les deux points de vue se rejoignent : l'interface qui satisfait l'utilisateur (trouver et contacter vite) est exactement celle qui satisfait le commanditaire (provoquer la mise en relation). Concevoir pour ces personas, c'est servir les deux d'un même geste.

ANALYSE DES SOLUTIONS COMPARABLES

5. Évaluation de l'existant

> « Aucune solution ne réunit la recherche locale par instrument, le contact gratuit et une interface moderne et accessible — c'est exactement la place que Giggr vient prendre. »

J'ai analysé trois plateformes comparables, plus la solution informelle de référence (les groupes Facebook), sous les angles que demande le référentiel : interface, architecture, utilisabilité, accessibilité, référencement. Pour rester sur des faits et non des impressions, je les ai auditées avec les mêmes outils que mon propre produit — capture de la page d'accueil, relevé des signaux de structure et de référencement, et audit d'accessibilité automatisé avec axe-core. L'analyse détaillée, chiffrée et illustrée de captures annotées figure dans le site de documentation ; j'en résume ici les enseignements.

Les plateformes analysées

BandMix est le concurrent le plus proche : des petites annonces de musiciens cherchables par instrument et par code postal. La recherche par instrument et localisation y est l'action centrale, offerte dès l'accueil — c'est ce qu'il faut en retenir. En revanche, l'interface est dense et datée, le contact réservé aux abonnés payants, et l'accessibilité faible : 81 nœuds en échec à l'audit, des cibles tactiles trop petites, des images sans alternative.

Vampr est un réseau social de musiciens fondé sur un appariement par profil. Son visuel moderne et sa dimension communautaire séduisent ; mais c'est avant tout une application mobile, dont le web n'est qu'une vitrine de téléchargement, l'appariement reste opaque faute de filtres explicites, et la hiérarchie des titres est incorrecte.

JoinMyBand est un site d'annonces gratuit, à l'ancienne. Sa gratuité, sa simplicité et sa sobriété technique le rendent, paradoxalement, irréprochable à l'audit d'accessibilité (zéro violation) ; mais son esthétique est très datée et son expérience, minimaliste, ne propose aucune fiche structurée.

Restent les **groupes Facebook**, la pratique réelle aujourd'hui : gratuits et déjà peuplés, mais dépourvus du moindre filtre par instrument ou par ville. L'information y est dispersée, noyée dans le fil — précisément le manque que Giggr comble.

Les trois plateformes comparables, auditées avec les mêmes outils que Giggr. De gauche à droite : BandMix (recherche centrale, mais interface dense et contact payant), Vampr (visuel moderne, mais application mobile au matching opaque), JoinMyBand (gratuit et accessible, mais daté). Analyse chiffrée et captures annotées dans le site de documentation.

Ce que j'en retiens pour Giggr

L'analyse fait apparaître une place vacante. Les solutions existantes sont soit payantes au contact (BandMix), soit datées (JoinMyBand), soit enfermées dans une application au matching opaque (Vampr), soit dispersées sans structure (Facebook). Aucune ne réunit en même temps les trois choses qui comptent : la recherche locale par instrument, le contact gratuit, et une interface moderne **et** accessible.

C'est exactement le positionnement de Giggr — et le fait que ce soit aussi le terrain où je me sens le plus fort (l'accessibilité, la structure) n'est pas un hasard : la faiblesse récurrente des concurrents sur ce point est devenue un argument de différenciation que j'ai décidé d'assumer pleinement.

DÉMARCHE, IDENTITÉ ET CHARTE GRAPHIQUE

6. Conception de l'interface

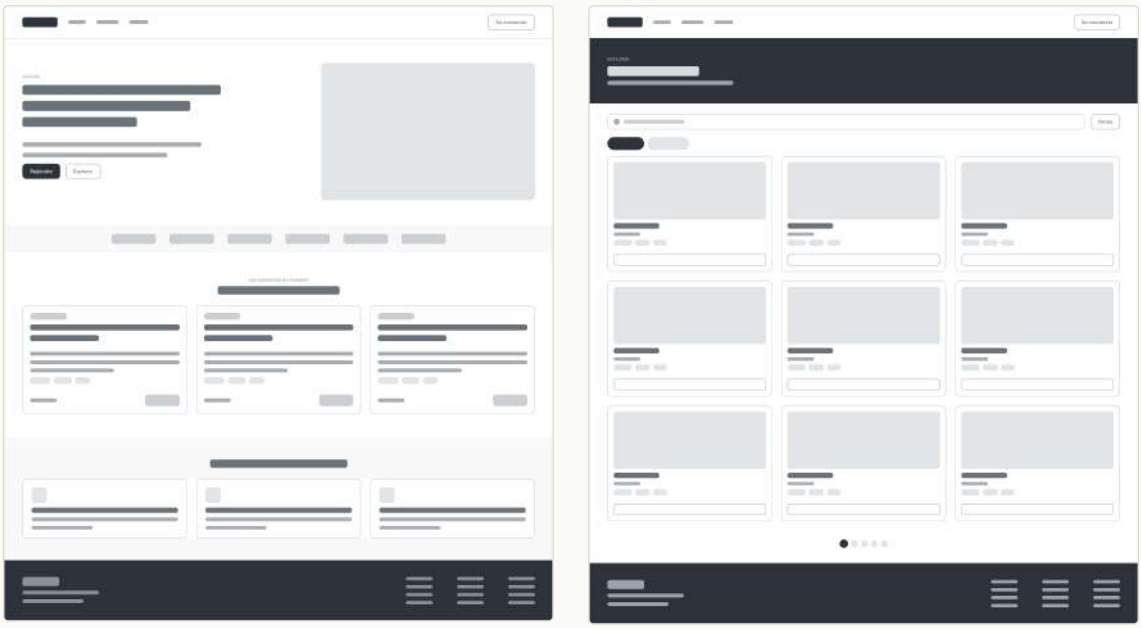
> « La conception a été la colonne vertébrale du projet : partir des intentions des personas, itérer écran par écran, puis tenir une charte unique jusqu'au dernier composant. »

La conception a occupé une place centrale, en amont du développement. J'ai voulu en faire une vraie démarche, avec des étapes et des itérations, et non une simple mise en forme du résultat.

Démarche et maquettage

Je suis parti des trois intentions dégagées de la recherche utilisateur — trouver un groupe, trouver des musiciens, publier une annonce — et j'ai descendu progressivement le niveau de détail, dans un fichier **Figma** unique organisé en pages : *Moodboard, UX, Desktop, Mobile, Composants*. Tout commence par une recherche visuelle, un moodboard qui fixe l'atmosphère avant toute interface — références d'applications musicales, photographies chaudes de musiciens, ambiances éditoriales. Viennent ensuite les wireframes basse-fidélité : des écrans en niveaux de gris, sans couleur ni contenu réel, qui valident la structure, l'ordre de lecture et l'emplacement des actions avant tout choix esthétique — accueil, exploration, fiche profil, conversation, détail d'annonce —, et qui prolongent un premier zoning crayonné sur papier. Enfin, les maquettes haute-fidélité appliquent la direction graphique et les composants, déclinées en desktop et en mobile.

Le fil est volontairement linéaire — de l'intention à l'écran, du gris à la couleur, du structurel au détaillé — pour que chaque décision visuelle repose sur une décision de structure déjà validée. Les outils sont à l'avenant : le papier pour penser vite, Figma pour structurer, maquetter et tenir la charte.



Wireframes basse-fidélité — l'accueil et l'exploration en niveaux de gris : valider la structure, l'ordre de lecture et l'emplacement des actions avant tout choix esthétique.

Des itérations, pas seulement un résultat

La conception a réellement évolué ; deux exemples l'ont changée en profondeur. L'exploration, d'abord : dans une première version, les filtres étaient repliés dans un panneau « recherche avancée ». En confrontant cet écran aux personas — qui pensent d'emblée « instrument et ville » —, j'ai mesuré le contresens : reléguer le filtre, c'est masquer l'action principale. Je l'ai donc remonté au premier plan, dans un tiroir immédiatement accessible. La page d'accueil, ensuite : son premier jet empilait trop de messages concurrents ; je l'ai resserrée autour d'une proposition de valeur unique et d'un seul appel à l'action dominant, pour que l'œil ne se disperse pas.

Navigation

Le modèle retenu est plat et stable : un en-tête persistant, et l'**exploration comme plaque tournante** d'où l'on bascule entre profils et annonces, filtre, et rejoint une fiche puis une conversation. J'ai écarté une arborescence profonde : les parcours de mes personas sont courts et tendus vers une seule action — contacter —, et un menu plat réduit la charge de navigation.

Moodboard et identité de marque

L'identité cherche un équilibre entre *chaleur* — l'humain, la musique — et *rigueur* — la lisibilité, la structure. Mon moodboard puisait dans une esthétique éditoriale, faite de beaucoup de blanc, de filets fins et d'une mise en page assumée, réchauffée par une

couleur d'accent et quelques aplats pastel. Les formes restent sobres et généreuses en espace : c'est ce qui donne au produit son sérieux sans le rendre froid.

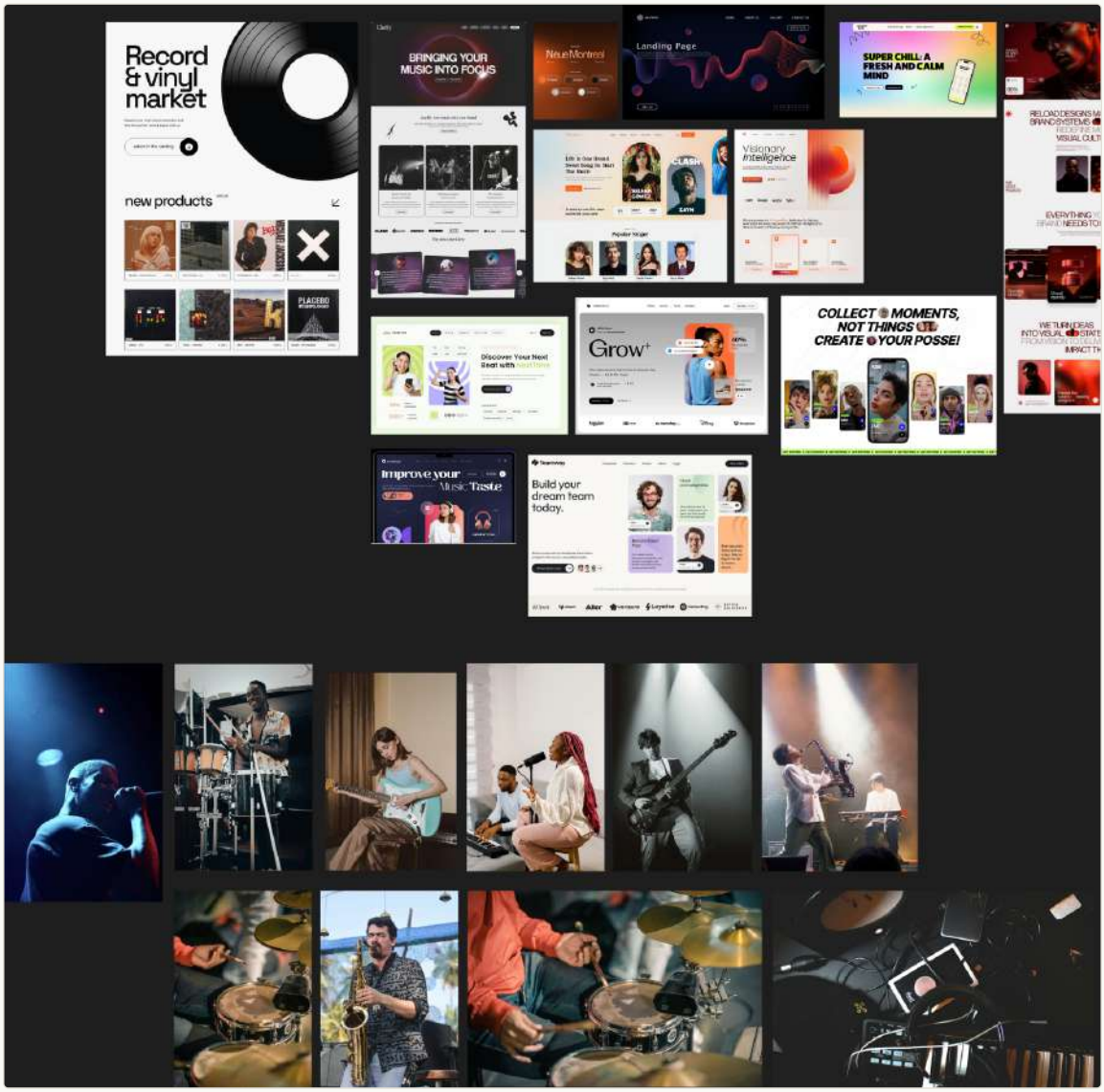


Planche moodboard — références d'applications musicales et ambiances éditoriales (en haut), photographies chaudes de musiciens (en bas). Page « Moodboard » du fichier Figma.

Couleurs

La palette est pensée par **rôle**, jamais par teinte : chaque couleur reçoit une affectation fonctionnelle, ce qui garantit la cohérence et le maintien des contrastes. Un accent unique — `#c2410c`, une terracotta chaude — est réservé aux actions et aux éléments interactifs : c'est la couleur de la conversion. Une hiérarchie de gris nommés par usage (`heading`, `body`, `subtle`, `caption`, `placeholder`) donne à chaque niveau de texte son ton, dosé pour rester au-dessus des seuils de contraste. Des déclinaisons `on-dark` habillent les fonds sombres, tandis que quelques pastels — saumon, bleu, taupe — distinguent des sections

sans jamais porter d'information à eux seuls. Une couleur ne se choisit donc pas : elle se réfère à un rôle.

Typographie

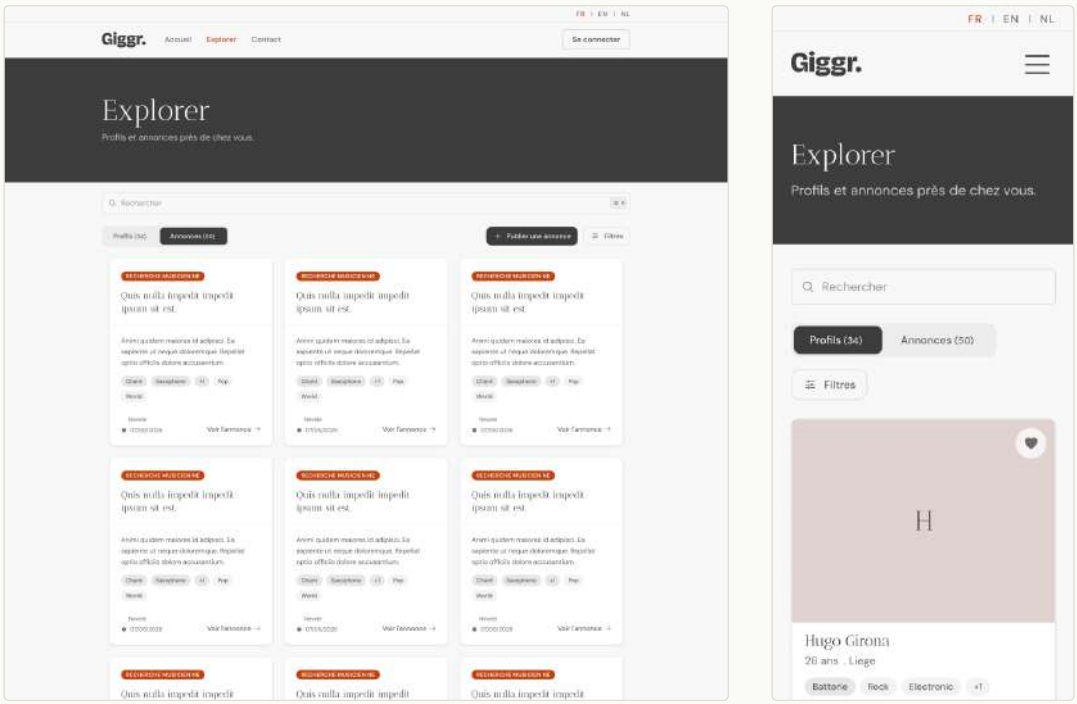
Deux familles au contraste assumé : **Antic Didone**, un sérif élégant réservé aux titres, qui donne le caractère ; et **DM Sans**, un sans-serif neutre et très lisible pour tout le texte courant. Cette opposition sérif / sans-serif installe une hiérarchie immédiate entre ce qui annonce et ce qui se lit. L'échelle de tailles reste limitée et cohérente, et la hiérarchie passe par la taille et la graisse plutôt que par la multiplication des styles.

Contenus non textuels

Les icônes sont regroupées dans une **sprite map SVG** générée automatiquement au build — un workflow découvert en stage et réutilisé ici : un seul fichier, chaque icône piochée par son identifiant, pour un gain de performance et de maintenabilité. Les images, avatars et médias de profil, sont fournies par les utilisateurs ; elles sont redimensionnées côté serveur en plusieurs variantes et servies de façon adaptative (`srcset`), afin de ne jamais peser plus que nécessaire.

Adaptabilité

L'interface est pensée **mobile d'abord** — l'usage premier d'un musicien étant le téléphone — puis enrichie vers le grand écran. Les grilles passent d'une à plusieurs colonnes, les filtres d'un tiroir à un panneau permanent, la navigation d'un menu plein écran à une barre. Les interactions tactiles sont prises en compte d'emblée : des cibles suffisamment grandes, et aucune action essentielle réservée au survol.



La même exploration, du grand écran au mobile : la grille de profils passe de trois colonnes à une, les filtres d'un panneau permanent à un tiroir, la navigation à un menu compact. Maquettes haute-fidélité (Figma).

Contraintes

Le projet n'avait pas de charte client préexistante : j'ai donc défini l'identité de A à Z. En revanche, je me suis imposé deux contraintes fortes, traitées dès la conception comme des règles de design et non comme des vérifications finales : un niveau d'accessibilité AA — contrastes, focus, structure — et un budget de performance — poids des images, sobriété du JavaScript.

Charte graphique synthétique

Au terme, les guidelines tenus tout au long du projet :

- **Typographie** : Antic Didone (titres) · DM Sans (texte) · échelle restreinte, hiérarchie par taille et graisse.
- **Couleurs** : jetons fonctionnels (accent unique pour l'action ; gris nommés par rôle ; `on-dark` ; pastels de fond) — affectation, jamais décoration arbitraire.
- **Composants** : réutilisés plutôt que redessinés (boutons, cartes, champs, chips, tableaux), pour une cohérence d'ensemble — les principaux formalisés en composants Figma avec leurs états (défaut, survol, sélectionné) sur la page *Composants*.
- **Accessibilité dans la charte** : contrastes vérifiés, focus visible, repères de structure — intégrée aux règles de design, pas ajoutée après coup.

CHARTE — JETONS DE COULEUR

Marque & action

accent
#c2410c

accent-on-dark
#f67649

danger
#b91c1c

success
#15803d

Échelle de texte (fond clair)

heading
#2d2d2d

body
#3d3d3d

subtle
#5c5c5c

caption
#6b6b6b

placeholder
#949494

Fonds de section & surfaces

pastel-salmon
#f9e5de

pastel-blue
#e0ebf4

pastel-taupe
#e0d2ce

bg
#f8f8f8

dark
#3d3d3d

CHARTE — TYPOGRAPHIE

Antic Didone — Titres — un sérif élégant, qui donne le caractère.

DM Sans — Texte courant — un sans-serif neutre et très lisible, graisses de 100 à 900.

Trouver un groupe, trouver des musiciens, publier une annonce : les trois intentions qui structurent Giggr, composées ici en DM Sans — du light au **bold**, et son *italique*.

SECTIONS, CONTENUS ET CONTENU GÉNÉRÉ

7. Architecture de l'information

> « Une grande partie du contenu de Giggr ne m'appartient pas : mon travail a été de bâtir une structure assez solide pour accueillir ce que les utilisateurs y déposent. »

Les grandes sections

L'architecture suit les intentions des utilisateurs plutôt que la logique de la base de données ; elle s'organise autour de quelques ensembles. L'accueil est une vitrine : il présente la proposition de valeur et oriente vers l'action — explorer, publier. L'exploration forme le cœur du produit, scindée en deux vues — les profils de musiciens et les annonces — qui partagent les mêmes filtres d'instrument et de ville. De là, on atteint les fiches : la fiche profil, qui présente un musicien, ses médias et son point de contact, et la fiche annonce, qui décrit une recherche précise. Puis vient la messagerie, ces conversations privées où aboutit la mise en relation. Restent enfin l'espace personnel — profil, paramètres de compte, publication d'annonce — et les pages institutionnelles, contact et politique de confidentialité.

Des contenus pensés pour le web

Les contenus ont été conçus pour être **scannés**, pas lus de bout en bout : titres explicites, hiérarchie sans saut de niveau, informations clés d'un profil ou d'une annonce (instrument, ville, intention) immédiatement repérables. Chaque page porte un titre unique de premier niveau et des repères de structure (en-tête, navigation, contenu, pied) — autant pour la clarté visuelle que pour les lecteurs d'écran et le référencement. La microdata (Schema.org) enrichit les pages où c'est pertinent.

Ce qui est sous mon contrôle, et ce qui m'échappe

Une distinction a structuré toute la conception : **la quasi-totalité du contenu de Giggr est générée par les utilisateurs**. Les profils, les annonces, les messages, les avatars — tout cela, je ne l'écris pas, je le reçois.

Ce qui est sous mon contrôle, c'est le *contenant* : la structure des pages, la hiérarchie de l'information, les gabarits, les libellés de l'interface, les états — vide, en chargement, erreur. Ce qui m'échappe, c'est le *contenu* : un profil peut être très complet ou presque vide, une annonce bien rédigée ou laconique, un nom long ou court, une image au mauvais ratio.

Cette réalité a une conséquence directe sur la conception : il faut bâtir une structure qui reste lisible et digne quel que soit ce qu'on y verse. Concrètement, cela signifie des états vides soignés — que voit-on quand une recherche ne donne rien, quand un profil n'a pas encore de média ? —, des images normalisées côté serveur pour absorber les ratios imprévus, et des libellés qui ne supposent jamais que le contenu sera « idéal ». Concevoir l'architecture de l'information, ici, c'est surtout concevoir pour l'imperfection du réel.

Une fiche profil : le gabarit (avatar, nom, intention, instruments, genres, point de contact) reste structuré et digne, quel que soit le contenu déposé par l'utilisateur.

TESTS UTILISATEURS ET AJUSTEMENTS

8. Utilisabilité

> « Une faiblesse repérée en test n'est pas une mauvaise nouvelle : c'est exactement ce que le test est censé produire. »

Démarche

Pour ne pas juger l'utilisabilité depuis mon propre point de vue — biaisé, puisque je connais le produit par cœur —, je l'ai mise à l'épreuve auprès de vrais utilisateurs. Le protocole couvre **cinq tâches clés** — rechercher, créer un compte, publier une annonce, filtrer des profils, prendre contact —, abordées à travers **trois scénarios** (un par persona) avec **trois participants** représentatifs du public cible. Les sessions se sont tenues le même jour, sur trois appareils volontairement différents — un grand écran de bureau, un iPhone et un iPad —, afin de couvrir plusieurs tailles d'écran. La méthode : observation directe et **verbalisation à voix haute**, chaque session enregistrée en screencast, ce qui permet de relier chaque difficulté à un écran précis.

Chaque participant déroule le **parcours de référence** correspondant à son profil — « trouver un groupe » (P1, guitariste), « publier une annonce » (P2, chanteuse), « trouver des musiciens » (P3, violoniste) —, ceux-là mêmes établis lors de la recherche utilisateur. Le déroulé complet, les screencasts et l'analyse commentée figurent dans le site de documentation ; j'en restitue ici la synthèse et les enseignements de conception.

Enseignements et ajustements

J'organise les observations selon le même cadre en trois temps que le dossier qualité. En un coup d'œil :

PARTICIPANT	APPAREIL	PARCOURS	VERDICT
P1 — guitariste	Bureau, écran 34″	Trouver un groupe	Réussi ; hésitation sur les filtres
P2 — chanteuse	iPhone 13 mini	Publier une annonce	Réussi ; aucun blocage
P3 — violoniste	iPad 11″	Trouver des musiciens	Réussi ; filtres réinitialisés au retour

Ce qui fonctionne. L'essentiel tient : les trois participants ont mené leur parcours jusqu'au bout, jusqu'à convenir d'un rendez-vous. L'onboarding complet — création de compte, vérification d'e-mail, profil, publication — s'enchaîne sans guidage (P1). Surtout, deux points que la conception redoutait ont tenu : la **bascule profils / annonces** a été repérée sans difficulté (P1), et le type d'annonce « Je recherche » compris d'emblée (P2). Sur iPhone comme sur iPad, le parcours reste fluide malgré le petit écran (P2, P3) — la conception mobile d'abord paie.

Ce qui pose problème. Le point de friction est récurrent et bien localisé : la zone des **filtres**. Non pas leur découvrabilité — les trois les ont trouvés —, mais leur ergonomie. Pour P1, trop de cases à cocher s'affichent d'emblée, ce qui provoque une hésitation. Pour P3, l'état des filtres n'est **pas conservé** au retour depuis une fiche profil : elle doit tout refiltrer et réappliquer à chaque aller-retour. Un point secondaire, relevé par P3 : il faut ouvrir un profil pour découvrir s'il porte une annonce pertinente, alors que cette information aiderait à décider plus tôt dans le parcours. À la marge, un échec de Face ID a introduit une friction côté appareil, hors produit.

Conclusions. Trois ajustements priorisés en découlent. D'abord, **conserver l'état des filtres** lors des retours arrière (P3) — le plus pénalisant, car répété à chaque consultation de profil. Ensuite, **regrouper les filtres dans un accordéon** qui ne déploie les cases qu'à l'ouverture de la section (P1), pour alléger la charge visuelle. Enfin, **exposer plus tôt** — dès la fiche, voire la carte — les annonces du musicien consulté (P3). Le constat le plus utile est ailleurs : ce que je prenais pour le risque principal — comprendre la bascule, trouver les filtres — n'en était pas un ; le vrai sujet est l'ergonomie fine du filtrage. C'est précisément ce que le test était censé produire — non pas l'absence de faiblesse, mais des faiblesses repérées, analysées et traduites en décisions de conception, à vérifier ensuite selon la même boucle *observer → corriger → re-mesurer* que je tiens pour l'accessibilité (section 9).

Itérations déjà menées

Au-delà des tests formels, l'utilisabilité s'est améliorée en continu, au fil des audits techniques, selon la même boucle *observer → corriger → re-mesurer* que je détaille pour l'accessibilité (section 9). Les tests utilisateurs prolongent cette démarche, mais en déplacent l'objet : ils ne mesurent plus la **conformité** d'un écran, mais sa **compréhension** par quelqu'un qui le découvre. C'est la seule chose que ni un audit automatisé, ni moi qui connais le produit par cœur, ne pouvons constater à leur place.

NIVEAU VISÉ, OUTILS ET RÉSULTATS

9. Accessibilité

> « L'accessibilité n'est pas une case à cocher en fin de projet : c'est une règle de design que j'ai tenue du début, et que j'ai surtout cherché à prouver. »

L'accessibilité est l'une des deux moitiés de mon fil conducteur, et le terrain où les concurrents sont les plus faibles. Je ne voulais donc pas l'affirmer, mais la **mesurer**.

Niveau visé, et pourquoi

J'ai visé la conformité **WCAG 2.2 niveau AA** sur les pages publiques. Ce niveau couvre les critères A et AA — contrastes suffisants, navigation au clavier, libellés de formulaires, alternatives textuelles, ordre de lecture cohérent — et c'est le niveau de référence attendu d'un produit sérieux. Je n'ai pas visé le AAA, et c'est un choix assumé : certains critères AAA (contrastes renforcés, contraintes typographiques) auraient aplati la direction graphique sans bénéfice réel pour mon public. Viser AA, c'est tenir un haut niveau d'exigence sans sacrifier l'identité — un arbitrage, pas un renoncement.

Concernant **AnySurfer** (le label belge), j'ai fait le choix de ne pas y recourir : c'est un service payant que mon cadre ne permettait pas de mobiliser. Plutôt que d'annoncer une validation que je ne pouvais pas produire, j'ai préféré m'appuyer sur des outils reproductibles et m'en tenir à ce que je peux montrer.

Outils et résultats

J'ai croisé plusieurs outils, chacun pour ce qu'il fait de mieux. **axe-core** — le moteur d'audit de référence, celui qu'utilise Lighthouse — a été exécuté via Playwright sur les six pages publiques, avec les jeux de règles WCAG 2.0, 2.1 et 2.2, niveaux A et AA : zéro violation, sur 135 règles vérifiées au total. **Lighthouse** le confirme, avec un score d'accessibilité de 100/100 sur desktop comme sur mobile. À cela s'ajoutent un contrôle des contrastes de la palette, couple par couple, au regard des seuils AA — ce qui rejoint la logique de jetons de couleur fonctionnels du design system —, et **Total Validator** pour la validation du balisage HTML, dont l'accessibilité dépend elle aussi.

Les rares alertes restantes sont documentées comme faux positifs — par exemple un contraste qu'axe ne peut pas calculer à cause d'un pseudo-élément d'animation, mais réellement conforme. Je ne les balaie pas : je les justifie une à une.

Une accessibilité qui se corrige et se re-mesure

L'accessibilité a aussi vécu une vraie boucle de remédiation. L'audit avait d'abord trouvé un seul écart — un compteur décoratif au contraste insuffisant sur l'exploration. Je l'ai corrigé (en retirant l'opacité qui le délavait, le faisant passer de 2,5:1 à environ 5,7:1), puis re-mesuré : le contrôle suivant ne relève plus aucune violation. Ce petit épisode résume ma méthode : mesurer, corriger à la source, re-mesurer — et garder la trace des deux.

L'ensemble de ces relevés, captures et tableaux figure dans le site de documentation.

OUTILS, FRAMEWORKS ET ARBITRAGES

10. Choix techniques d'intégration

> « À chaque choix d'outil, la même question : qu'est-ce que je gagne, et qu'est-ce que j'accepte de perdre ? »

L'intégration a été menée avec une exigence cohérente avec le fil conducteur : un rendu **accessible et structuré**, donc des outils qui servent le HTML plutôt que de le remplacer.

Les choix, et leurs raisons

Pour le style, j'ai retenu **Tailwind CSS** (v4), pour sa logique utilitaire et surtout pour ses jetons de design : la palette et la typographie sont déclarées une seule fois, via `@theme`, puis réutilisées partout. Le design system cesse alors d'être un document posé à côté du code — il *est* le code.

```
/* resources/css/colors.css — les jetons, déclarés une seule fois… */
@theme {
    --color-accent: #c2410c;        /* l'action, la conversion */
    --color-heading: #2d2d2d;       /* titres */
    --color-body: #3d3d3d;          /* texte courant */
    --color-subtle: #5c5c5c;
    --color-pastel-salmon: #f9e5de; /* fonds de section */
    --color-pastel-blue: #e0ebf4;
    --color-pastel-taupe: #e0d2ce;
}
```

…et réutilisés partout par des classes utilitaires (`bg-accent`, `text-heading`). J'y gagne une cohérence garantie et une refactorisation aisée ; j'y perds un balisage plus verbeux en classes, qu'il faut discipliner avec des composants.

Le choix le plus structurant est **Livewire 4**, pour la réactivité. Plutôt qu'un front découplé — de type Vue et Inertia, que je voulais explorer après mon stage —, Livewire rend l'interface dynamique tout en gardant le rendu côté serveur : le HTML existe d'abord, JavaScript l'enrichit. C'est exactement ce qu'exigeaient mon accessibilité et ma dégradation gracieuse,

puisque le contenu reste présent sans JavaScript. Je gagne une réactivité sans avoir à écrire d'API cliente, et un contenu accessible par défaut ; je perds une dépendance forte au cycle serveur et un modèle mental à acquérir.

Autour de ce socle viennent quelques couches plus légères. **Alpine.js**, embarqué par Livewire, gère l'état d'interface côté client — panneaux ouverts ou fermés, menu, modes édition — juste ce qu'il faut, sans framework supplémentaire. **Laravel Reverb** et **Echo** assurent le temps réel de la messagerie, le défi que je voulais relever (voir l'architecture technique). **GSAP** anime quelques apparitions au scroll, toujours conçues pour se désactiver proprement lorsqu'un mouvement réduit est demandé, le contenu restant lisible sans elles. Enfin **Vite** orchestre le build, où une sprite map SVG d'icônes est générée automatiquement — un workflow appris en stage que je tenais à réutiliser, pour ses gains de performance et de maintenabilité.

Ce que j'ai gagné, ce que j'ai perdu

Le fil rouge de ces choix est l'**amélioration progressive** : un socle HTML solide rendu côté serveur, enrichi par des couches JavaScript dont l'absence dégrade l'expérience sans la casser. J'y gagne une accessibilité et une robustesse réelles, et un code dont la responsabilité de chaque couche est claire. Ce que j'y perds, je l'assume : j'ai renoncé à explorer une stack 100 % cliente (Vue/Inertia) qui me tentait, parce qu'elle servait moins bien mon fil conducteur. C'est précisément le genre d'arbitrage — séduction technique contre cohérence du projet — que ce travail m'a appris à trancher.

STACK SERVEUR ET MESSAGERIE TEMPS RÉEL

11. Architecture technique

> « C'est ici qu'est mon défi : la messagerie temps réel, le morceau de back-end que je ne savais pas faire et que ce projet m'a appris. »

La stack côté serveur, et pourquoi

Le socle est **Laravel 13**. Je l'ai retenu pour des raisons à la fois personnelles et techniques : c'est le framework que j'ai pratiqué en agence durant mon stage, donc celui sur lequel je pouvais viser un vrai niveau de qualité plutôt que de découvrir en chemin ; et c'est un écosystème mûr, dont les briques (authentification, broadcasting, files d'attente, tests) couvrent précisément les besoins de Giggr.

Autour de ce socle, j'ai réuni des briques éprouvées. **Livewire 4** porte l'interface réactive rendue côté serveur (cf. choix d'intégration). **Laravel Fortify** prend en charge l'authentification — inscription, connexion, vérification d'e-mail —, une base solide plutôt qu'un système maison fragile sur un sujet aussi sensible. Pour le temps réel, **Laravel Reverb**, le serveur WebSocket officiel, est couplé à **Laravel Echo** côté client. Les tests, enfin, reposent sur **Pest**.

Le modèle de données

L'application s'organise autour d'une douzaine de modèles aux responsabilités claires : `User` et `Profile` (l'identité), `Announcement` (les annonces), `Conversation` et `Message` (la messagerie), `City`, `Instrument`, `Genre` (les données de référence qui alimentent les filtres), `Follow` (les abonnements) et `Media` (les images de profil) ; s'y ajoutent les blocages entre utilisateurs (`UserBlock`) et les notifications (cloche temps réel, e-mails). Les relations entre ces modèles sont chargées par *eager loading* pour éviter les requêtes en cascade (le problème dit « N+1 ») : une grille d'exploration entière tient ainsi en une quinzaine de requêtes plutôt qu'en une soixantaine — un point mesuré et documenté dans le dossier qualité. Le schéma entité-association complet de ce modèle est reporté en **annexe A**, en fin de document et au format paysage pour en préserver la lisibilité.

Le module clé : la messagerie en temps réel

C'est le morceau dont je suis le plus fier, parce que c'est exactement celui que je m'étais promis d'apprendre. Le chemin d'un message :

1. À l'envoi, une action serveur (`SendMessage`) enregistre le message en base.
2. Elle diffuse alors un événement (`MessageSent`, marqué `ShouldBroadcast`) sur un canal privé propre à la conversation (`ConversationChannel`).
3. Ce canal étant privé, son autorisation ne laisse s'y abonner que les deux participants : un tiers ne peut pas écouter.
4. Côté client, `Echo`, connecté au serveur Reverb par WebSocket, reçoit l'événement et met à jour le fil sans rechargement.
5. À la lecture, une seconde action (`MarkConversationAsRead`) diffuse un événement `MessagesRead` qui met à jour l'état « lu » et le compteur de messages non lus, là encore en direct.

Trois extraits suffisent à le prouver : l'action diffuse l'événement, l'événement déclare les canaux qu'il vise, et l'autorisation du canal n'admet que les participants de la conversation.

```php
// app/Actions/SendMessage.php — après l'enregistrement en base, on diffuse
broadcast(new MessageSent($message))->toOthers();

// app/Events/MessageSent.php — l'événement diffusé en temps réel
class MessageSent implements ShouldBroadcastNow
{
    public function broadcastOn(): array
    {
        // $recipientId : l'autre participant de la conversation
        return [
            new PrivateChannel('conversation.'.$this->message->conversation_id),
            new PrivateChannel('App.Models.User.'.$recipientId),
        ];
    }
}

// app/Broadcasting/ConversationChannel.php — le canal est privé :
// l'abonnement n'est autorisé que pour un membre de la conversation
public function join(User $user, int $conversationId): bool
{
    return $user->conversations()->whereKey($conversationId)->exists();
}
```

Le diagramme de séquence complet de ce flux figure en **annexe B**, lui aussi au format paysage.

J'insiste sur ce point parce qu'il porte mon récit de progression : dans mon rapport de stage, je ne connaissais des WebSockets que « les grandes lignes » et je me promettais de m'y former seul. Cette architecture est la preuve que je l'ai fait.

Suggérer des annonces pertinentes

Sur chaque page d'annonce, un bloc « Ces annonces pourraient vous intéresser » propose trois autres annonces choisies par **affinité**, jamais au hasard. Le critère de pertinence réutilise la taxonomie qui structure déjà tout le produit : une annonce est jugée proche dès lors qu'elle **partage au moins un instrument ou un genre** avec celle qu'on consulte. La requête écarte l'annonce courante, ne retient que les annonces actives, et en garde trois.

```
// pages/announcement/show.blade.php — les annonces apparentées
$instrumentIds = $this->announcement->instruments->pluck('id');
$genreIds = $this->announcement->genres->pluck('id');

$this->related = Announcement::with(['city', 'instruments', 'genres'])
    ->active()                          // annonces publiées
    ->where('id', '!=', $this->announcement->id)
    ->where(function ($q) use ($instrumentIds, $genreIds) {
        // au moins un instrument OU un genre en commun
        $q->whereHas('instruments', fn ($q2) => $q2->whereIn('id', $instrumentIds))
          ->orWhereHas('genres', fn ($q2) => $q2->whereIn('id', $genreIds));
    })
    ->limit(3)
    ->get();
```

Deux partis pris me tiennent ici. D'abord, la pertinence reste **lisible** : contrairement au *matching* opaque que je reprochais à certains concurrents (section 5), elle s'appuie sur des critères que l'utilisateur comprend et pourrait vérifier lui-même — les instruments et les genres de l'annonce. Ensuite, les relations sont **chargées d'avance** (`with([...])`) pour que l'affichage des trois cartes ne déclenche aucune requête en cascade, dans la même logique de performance que le reste du produit.

Modules développés et qualité

Au-delà de la messagerie, j'ai développé les composants Livewire de l'interface, des actions métier (envoi de message, upload de média), des jobs asynchrones (traitement des images uploadées) et des événements de diffusion. L'ensemble est couvert par 660 tests Pest — dont 15 tests navigateur dans un vrai Chromium — et une couverture de 89,4 % des lignes sur la logique applicative, preuves là encore consignées dans le site de documentation.

Le code est accessible publiquement via le dépôt Git du projet : github.com/hugogirona/pfe. Le site de documentation a son propre dépôt (github.com/hugogirona/hugo-girona-doc-PFE), et j'ai choisi de le déployer lui aussi sur **Laravel Forge**, sur un sous-domaine de l'application (`doc.giggr.be`) : tout vit sur la même infrastructure que le produit, plutôt que sur un hébergement tiers.

SYNTHÈSE CRITIQUE DU PROCESSUS

12. Retour d'expérience

> « La vraie leçon de ce projet n'est pas une compétence de plus, mais une question de dosage : où placer le risque dans le temps. »

Ce retour n'est pas un bilan satisfait, mais une synthèse critique de ma façon de travailler.

Ce qui a bien fonctionné

Trois décisions ont payé. La première, fondatrice : avoir conçu en amont, par étapes — zoning, wireframes, maquettes — et par itérations. Chaque fois que j'ai investi du temps dans la conception plutôt que de foncer au code, je l'ai récupéré ensuite : une structure pensée se code vite, une structure improvisée se recode. La deuxième : traiter le design system comme du code, jetons de couleur et de typographie déclarés une seule fois, composants réutilisés ; la cohérence n'a jamais été un effort, elle était structurelle. La troisième : vouloir prouver plutôt qu'affirmer — m'imposer de tout mesurer, accessibilité, performances, tests, m'a discipliné bien au-delà de ce que « faire au mieux » aurait produit.

Ce que je referais autrement

Mon principal regret est une question de **dosage du risque dans le temps**. J'ai concentré l'essentiel de mon énergie technique sur la partie où je me sentais le moins solide — la messagerie en temps réel et les WebSockets — et j'ai eu raison de la relever, mais je l'ai sous-estimée en durée. Elle a mordu sur la fin du projet et repoussé certaines étapes, notamment les tests utilisateurs, plus tard que je ne l'aurais voulu. Avec le recul, j'attaquerais le morceau technique le plus risqué **en premier**, dans un temps borné, pour le « dérisquer » tôt — et je testerais avec de vrais utilisateurs plus tôt dans le processus, pour que leurs retours puissent encore nourrir la conception plutôt que de la valider après coup.

Ce qui reste à améliorer

Tout n'est pas fini, et je préfère le dire. La performance mobile garde un décalage de mise en page résiduel que je n'ai pas entièrement élucidé. Le dossier de design gagnerait à être prolongé d'une véritable bibliothèque de composants Figma reliée aux jetons du code — j'en ai posé les bases, pas encore l'ensemble. Et les tests utilisateurs, menés en fin de

parcours, ont livré des pistes concrètes — surtout autour de l'ergonomie des filtres : conserver leur état au retour arrière, alléger les cases affichées d'emblée — qu'il me reste à implémenter. Ce ne sont pas des échecs : ce sont les marches suivantes.

Boucle sur le fil conducteur

J'avais annoncé une exigence : concevoir une expérience **accessible et rigoureusement structurée, du design system au code**, pour un utilisateur que je connais de l'intérieur. Je crois en avoir tenu la trace — des jetons de design jusqu'aux 0 violation d'accessibilité, de la hiérarchie de l'information jusqu'à la messagerie temps réel que je ne savais pas écrire il y a quelques mois. Ce projet ne m'a pas seulement appris des techniques ; il m'a appris à **tenir une exigence de bout en bout, seul**, et à la prouver. C'est, je crois, ce qui me fait passer du statut d'étudiant qui exécute à celui de professionnel qui répond de ses choix.

ANNEXE A

Modèle de données

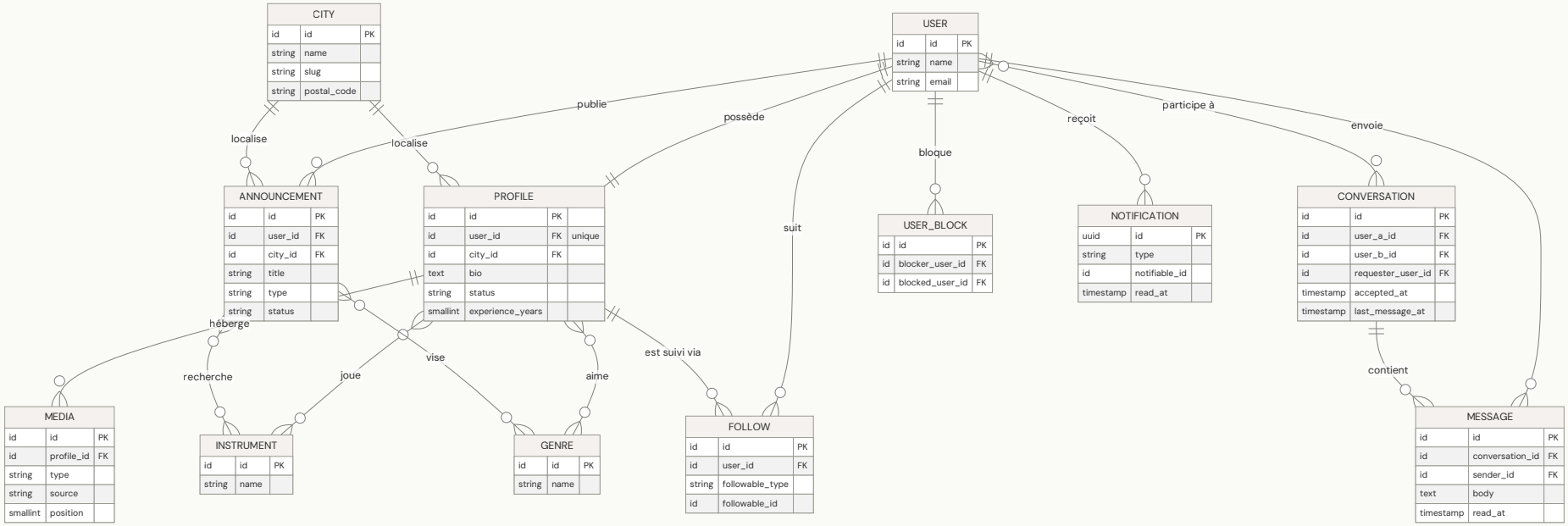


Schéma entité-association du modèle de données de Giggr — entités principales (User, Profile, City, Instrument, Genre, Announcement, Media, Conversation, Message, Follow) et leurs relations.

ANNEXE B

Messagerie en temps réel

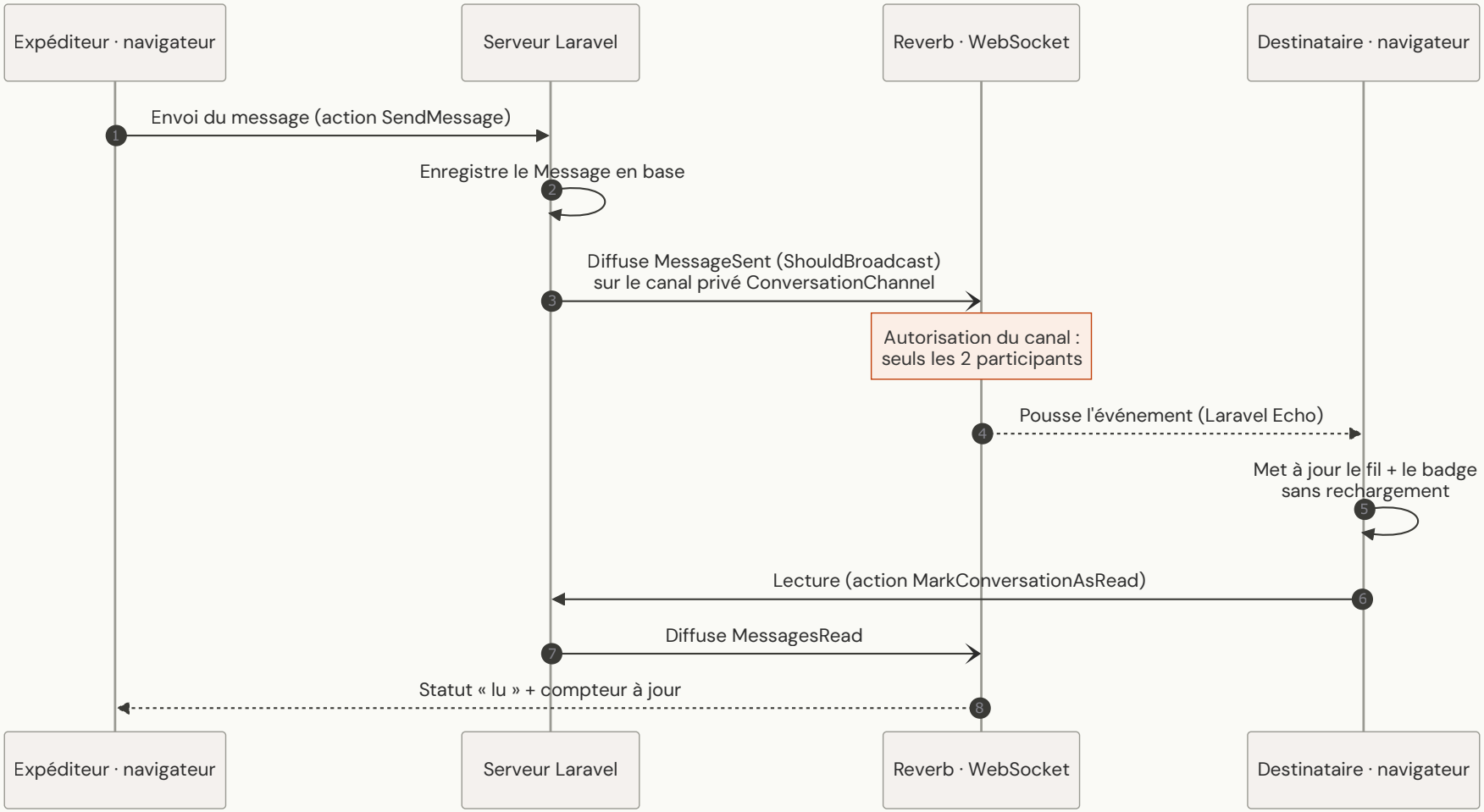


Diagramme de séquence de la messagerie en temps réel — de l'envoi du message à l'accusé de lecture : enregistrement côté Laravel, diffusion de MessageSent sur le canal privé, poussée par Reverb et réception par Laravel Echo côté client.