

**NEXUS LAB**  
CREATIVE AND GENERATIVE  
CODING ART

# DOSSIER PROJET

NexusLab : Plateforme collaborative  
de creative coding et d'art génératif

Candidat : TABURET Patrick

Titre Professionnel :

Concepteur Développeur d'Applications

Session : Du /11/2023 au 31/10/2024



**ACGD FORMATION**

|   |           |
|---|-----------|
| <b>1. INTRODUCTION</b>                                      | <b>4</b>  |
| 1.1 Contexte et présentation du projet                      | 4         |
| 1.2 Rappel du cahier des charges                            | 4         |
| <b>2. GESTION DE PROJET</b>                                 | <b>8</b>  |
| 2.1 Méthodologie adoptée                                    | 8         |
| 2.2 Outils de gestion utilisés                              | 8         |
| 2.3 Gestion des priorités et suivi des tâches               | 9         |
| <b>3. DESIGN ET ERGONOMIE</b>                               | <b>10</b> |
| 3.1 Processus de conception                                 | 10        |
| 3.2 Design (UX/UI)  | 13        |
| 3.3 Accès aux fonctionnalités                               | 15        |
| <b>4. ASPECT FONCTIONNEL ET TECHNIQUE</b>                   | <b>16</b> |
| 4.1 Choix techniques et technologiques                      | 16        |
| 4.2 Architecture globale du système                         | 18        |
| 4.3 Bilan des fonctions implémentées et des moyens utilisés | 19        |
| 4.4 Difficultés rencontrées et solutions apportées          | 21        |
| 4.5 Optimisation et ajout de fonctionnalités à venir        | 22        |
| <b>5. CREATIVE CODING</b>                                   | <b>23</b> |
| 5.1 Fonctionnement général de p5.js                         | 23        |
| 5.2 Intégration de p5.js au projet                          | 24        |
| <b>6. CONFIGURATION DE L'ENVIRONNEMENT DE TRAVAIL</b>       | <b>26</b> |
| 6.1 Backend – Symfony                                       | 26        |
| 6.2 Integration de React                                    | 27        |
| 6.3 Frontend Mobile   | 27        |
| 6.4 Gestion des dépendances                                 | 28        |
| 6.5 Gestion des versions                                    | 28        |

|   |           |
|---|-----------|
| <b>7. RÉALISATION DU PROJET</b>                             | <b>29</b> |
| 7.1 Développement global du projet                          | 29        |
| Structure finale du projet                                  | 29        |
| Gestion des données   | 30        |
| Gestion sécurisée des utilisateurs                          | 31        |
| Accessibilité   | 34        |
| 7.2 Développement de l'application web                      | 35        |
| Intégration des composants dynamiques React                 | 35        |
| Gestion des fichiers médias                                 | 37        |
| Gestion des données volumineuses                            | 39        |
| Implémentation d'une scène générative                       | 41        |
| Optimisation du code en vue d'intégrer de nombreuses scènes | 43        |
| 7.3 Développement de l'application mobile                   | 45        |
| Structure de la navigation                                  | 45        |
| Sécurité globale de l'application : client et serveur       | 47        |
| Intégration des scènes p5.js dans React Native              | 51        |
| Utilisation de bibliothèques et composants personnalisés    | 53        |
| <b>8. CONCLUSION ET PERSPECTIVES</b>                        | <b>55</b> |
| 8.1 Bilan du travail réalisé                                | 55        |
| 8.2 Réflexion personnelle                                   | 56        |
| <b>9. ANNEXES</b>   | <b>57</b> |
| 9.1 Bibliographie et références techniques                  | 57        |
| 9.2 Documents supplémentaires                               | 58        |
| 9.3 Code source   | 63        |

# LISTE DES COMPÉTENCES DU RÉFÉRENTIEL

RÉFÉRENTIEL CDA

## Développer une application sécurisée

- Installer et configurer son environnement de travail en fonction du projet.
- Développer des interfaces utilisateur
- Développer des composants métier
- Contribuer à la gestion d'un projet informatique

## Concevoir et développer une application sécurisée organisée en couches

- Analyser les besoins et maquetter une application
- Définir l'architecture logicielle d'une application
- Concevoir et mettre en place une base de données relationnelle
- Développer des composants d'accès aux données SQL et NoSQL

## Préparer le déploiement d'une application sécurisée

- Préparer et exécuter les plans de tests d'une application
- Préparer et documenter le déploiement d'une application
- Contribuer à la mise en production dans une démarche DevOps

# 1. INTRODUCTION

## 1.1 Contexte et présentation du projet

Ce dossier s'inscrit dans le cadre de la validation du Titre Professionnel de Concepteur Développeur d'Applications.

Le projet NexusLab est né d'une volonté d'allier deux passions : l'art et la programmation informatique. Les technologies numériques, et en particulier le code, permettent de développer une multitude de processus créatifs et favorisent l'expérimentation artistique. Cette alliance née du croisement de ces deux domaines est appelée le creative coding, cela regroupe une variété de pratiques artistiques ayant toutes en commun l'utilisation du code au cœur de leur processus créatif. Parmi elles on retrouve notamment l'art génératif et le data art, qui seront intégrés au projet.

L'objectif principal du projet est de développer une plateforme collaborative en ligne dédiée à ces pratiques, où artistes, développeurs, et amateurs de nouvelles technologies pourront créer, partager et collaborer sur des œuvres interactives. Ce projet inclut le développement d'une application web et d'une application mobile. Il a pour vocation de démocratiser la création numérique auprès du grand public, et de fédérer une communauté autour du creative coding.

## 1.2 Rappel du cahier des charges

Une des premières étapes de la conception a été l'élaboration d'un cahier des charges. Son rôle a été de fixer des objectifs clairs dès le début du projet et d'analyser les besoins, tout en prenant en compte les différentes contraintes et limites techniques. Ce document est joint en annexe de ce dossier, mais voici une synthèse rapide.

### Définitions

- **Creative coding :** Discipline artistique impliquant l'utilisation du code dans son processus de création, que ce soit pour des visuels, du son, des performances interactives, ou d'autres formes d'expression artistique. Cela englobe une large gamme de pratiques artistiques utilisant la programmation, dont l'art Génératif, le Live Coding et le Data Art.
- **Art Génératif :** Forme d'expression artistique qui utilise des algorithmes et des systèmes informatiques pour créer des visuels uniques de manière autonome ou semi-autonome.
- **Data Art / Visualisation :** Utilisation d'ensembles de données comme matériau artistique, transformant des informations (statistiques, données environnementales, réseaux sociaux, etc.) en

œuvres visuelles ou interactives. Cela permet de rendre des données abstraites plus accessibles et esthétiques tout en mettant en relief certains patterns, tendances ou réalités à travers ces données.

- **Live Coding** : Consiste à écrire et modifier du code en temps réel. Cette technique est généralement utilisée pour créer des médias audiovisuels.
- **p5.js** : Bibliothèque JavaScript open-source conçue pour simplifier la création graphique et interactive dans les navigateurs. P5.js est issu du projet *Processing*, un langage de programmation basé sur Java orienté vers la création graphique et interactive. L'objectif de p5.js est de transposer l'esprit de Processing au web en utilisant JavaScript.

## Vision & Objectifs

**Objectifs à court terme** : Mettre en place une plateforme fonctionnelle où les utilisateurs peuvent s'authentifier, créer, partager et collaborer sur des œuvres numériques interactives. Participer à la création d'une communauté autour de ces pratiques artistiques, encourager l'échange entre créateurs et utilisateurs.

**Objectifs à long terme** : Devenir une référence dans le domaine du creative coding, en rendant ces pratiques accessibles à un public plus large, notamment aux non-initiés, grâce à des outils simples d'utilisation et une communauté active et inclusive.

Développer avec les membres les plus impliqués des ressources pédagogiques, tutoriels, et des événements collaboratifs.

Intégrer une grande variété de technologies pour les scènes génératives, permettant ainsi à plus d'artistes développeurs de partager leurs créations.

**Ethique** : Philosophie open source, NexusLab se focalise sur la création, le partage et l'apprentissage plutôt que sur la monétisation des œuvres.

Absence de fonctionnalités de vente directe ou de marketplace pour préserver l'intégrité artistique et promotion d'une éthique de partage et de collaboration, en contraste avec les plateformes orientées profit.

Accessibilité et inclusivité de la plateforme.

Respect de la confidentialité et protection des données utilisateurs.

## Public cible

- **Artistes et développeurs** : Utilisateurs expérimentés en art numérique et en programmation, souhaitant créer ou partager des projets d'œuvres interactives.
- **Débutants et amateurs** : Public curieux de découvrir et d'apprendre les bases du creative coding, souvent non-initié mais intéressé par le creative coding ou par l'art en général.
- **Communauté créative** : Toute personne intéressée par l'exploration de nouveaux médias et la collaboration artistique à travers le code et les nouvelles technologies.

## Fonctionnalités principales

- **Scènes et œuvres** : Partage et intégration de scènes d'arts génératifs et de visualisation de data dotés d'interfaces manipulables par les utilisateurs.  
Création et partage d'œuvres numériques générées à partir de ces scènes.  
Accès aux scènes et aux créations pour tous les utilisateurs (galerie), possibilité de retravailler ces créations à partir des derniers paramètres utilisés au moment de la sauvegarde de l'œuvre (système de remix).  
Système de collaboration entre utilisateurs : espace de co-création en temps réel (collective drawing / live coding)
- **Interactivité** : Système de likes, de feedback, de suivi de membres et de notifications.
- **Gestion des utilisateurs** : Système d'authentification sécurisée, gestion des profils utilisateurs, galerie personnelle, et permissions variées selon les rôles (Artist, Admin, User).
- **Administration** : Espace administrateur pour la gestion des utilisateurs, de la galerie, des demandes de rôles et des requêtes pour ajouter une scène.

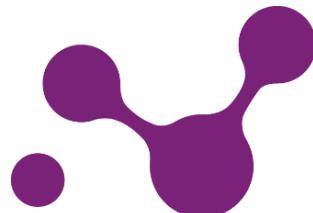
## Exigences techniques

La plateforme s'appuiera sur une architecture robuste, conçue pour assurer la maintenabilité, la sécurité et la scalabilité de l'application.

- **Backend** : Utilisation du framework Symfony (PHP) avec une architecture MVC. Une API sera mise en place pour permettre la communication avec les clients (web et mobile).
- **Base de données** : Une base de données relationnelle MySQL stockera les informations utilisateurs, les œuvres numériques et leurs métadonnées. Elle devra être optimisée pour gérer un grand nombre d'œuvres et garantir des performances stables, même en cas de forte charge.
- **Frontend Web** : Combinaison de Twig pour le rendu serveur et React pour des composants dynamiques, offrant une interface responsive et performante.
- **Application Mobile** : Développement en React Native pour une solution cross-platform, permettant de partager du code avec la version web.
- **Creative coding** : Intégration de sketches p5.js dans Symfony et React Native.  
Intégration d'un éditeur de code en ligne pour le live coding.  
Possibilité d'interagir simultanément sur la même interface que d'autres utilisateurs (WebSockets).
- **Sécurité** : Authentification sécurisée via Symfony (web) et JWT (mobile), cryptage des données sensibles, protection contre les attaques XSS et CSRF.
- **Performances et scalabilité** : Garantir une expérience utilisateur fluide, même lors de la manipulation de scènes génératives complexes et gourmandes en ressources. Optimisation du code backend et frontend pour assurer des performances optimales.  
L'infrastructure cloud choisie doit pouvoir s'adapter à une augmentation du nombre d'utilisateurs et des œuvres hébergées sans dégradation des performances.
- **Responsive design** : Interface web adaptée à toutes les plateformes (mobile, tablette, desktop).

## Contraintes

- **Temps** : Le projet se déroule en deux phases, avec un prototype fonctionnel à livrer avant le 31 octobre 2024, suivi de tests et du déploiement.
- **Budget** : L'approche open source limite les coûts, avec des dépenses principalement liées à l'hébergement cloud. L'objectif est de maintenir les coûts au minimum tout en assurant la qualité et la pérennité du projet.
- **Techniques** : Compatibilité avec les principaux navigateurs, systèmes d'exploitation et plateformes mobiles. Intégration de technologies spécifiques (p5.js, live coding, WebSockets) tout en optimisant les performances.
- **Utilisateurs** : Interface accessible conforme aux normes d'accessibilité (WCAG) et intuitive pour des utilisateurs novices. Respect du RGPD, gestion des droits d'auteur, et promotion d'un environnement inclusif et éthique.
- **Maintenance à long terme** : Infrastructure conçue de manière modulaire pour faciliter les mises à jour et les correctifs futurs, sans interrompre l'expérience utilisateur.



## 2. GESTION DE PROJET

La gestion du projet NexusLab a reposé sur une méthodologie **Agile**, avec une mise en œuvre de la méthode **Kanban**. Cette approche a été choisie pour sa flexibilité, et sa capacité à s'adapter rapidement à l'évolution des besoins tout au long du développement.

Ce projet a été mené individuellement, mais cette méthode est également adaptée pour de futures collaborations, car elle facilite la gestion d'équipes en favorisant la transparence et la communication.

### 2.1 Méthodologie adoptée

L'approche Kanban offre plusieurs avantages pour un projet créatif et technique comme NexusLab :

**Flexibilité et adaptabilité** : Contrairement à des méthodes plus rigides comme Scrum, Kanban permet une grande souplesse dans la gestion des priorités et des nouvelles tâches. Il se montre particulièrement efficace pour identifier les ajustements nécessaires en cours de projet, que ce soit en réponse aux retours d'expérience ou aux tests effectués.

**Visualisation du flux de travail** : Le tableau Kanban permet de visualiser toutes les tâches en cours et à venir, favorisant une gestion claire et structurée du projet. Cette visualisation structure le développement et permettrait, dans un cadre d'équipe, d'améliorer la transparence en rendant chaque tâche visible par tous les membres. Cela favorise une compréhension partagée du processus de développement et une meilleure organisation collective.

**Livraison continue** : Avec cette méthode, les cycles de livraison sont courts et continus. Cette itération rapide permet de délivrer des fonctionnalités à intervalles réguliers, de tester et de récolter des retours en temps réel. Cela permet d'améliorer rapidement l'application à chaque phase de développement.

### 2.2 Outils de gestion utilisés

Plusieurs outils ont été intégrés dans le cadre de la gestion de projet afin d'assurer un suivi rigoureux des tâches et un développement plus fluide :



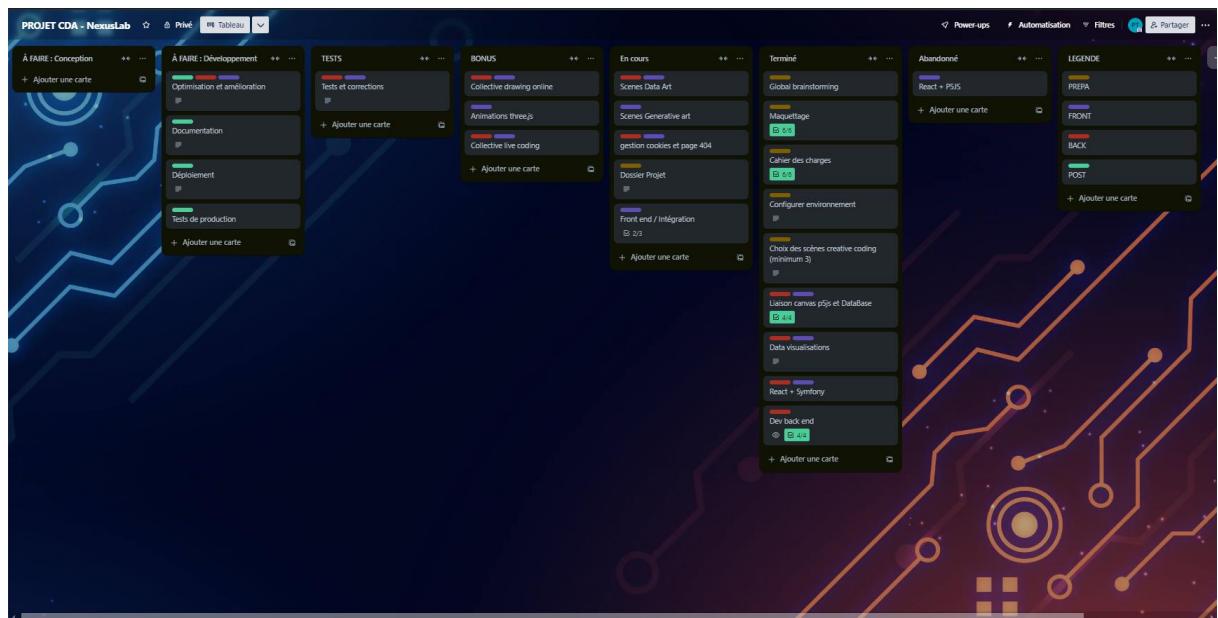
**Trello** : Utilisé pour organiser le flux de travail via un tableau Kanban. Les tâches étaient divisées en plusieurs colonnes (À faire, En cours, Terminé, etc.), avec une description détaillée des fonctionnalités à développer, les critères de validation, ainsi que les dépendances entre les différentes tâches. Le suivi des tâches a été facilité par les étiquettes, permettant de catégoriser chaque tâche dans un ou plusieurs domaines (frontend, backend, conception, etc.).



**GitHub** : Utilisé à la fois pour le contrôle de version et pour la gestion du code source, GitHub a permis de suivre l'évolution du projet sur différentes branches et de gérer les fusions sur la branche principale. Son utilisation facilitera également le travail en équipe, les pull requests permettront une validation rigoureuse du code avant chaque merge, garantissant ainsi une qualité constante tout au long du projet.

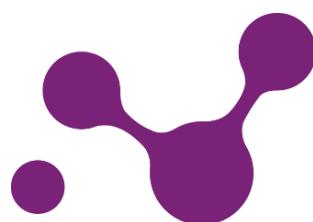
## 2.3 Gestion des priorités et suivi des tâches

Les principales fonctionnalités ont été développées en suivant les objectifs initiaux du cahier des charges, en mettant en priorité les éléments essentiels comme la création et le partage d'œuvres, ainsi que la gestion des utilisateurs. Le tableau Kanban a permis une gestion efficace de ces priorités.



Trello - Tableau Kanban : suivi de projet

Des points réguliers ont été effectués pour évaluer l'avancement des tâches en cours. Cela a permis de redistribuer les priorités en fonction des progrès réalisés et de résoudre rapidement les obstacles rencontrés.



# 3. DESIGN ET ERGONOMIE

Le design et l'ergonomie jouent un rôle central dans l'expérience utilisateur du projet NexusLab. La conception visuelle et l'interface utilisateur ont été élaborées dans le but d'offrir une navigation fluide, de minimiser la complexité visuelle, et de créer une expérience immersive qui reflète l'univers du creative coding.



Toutes les planches graphiques décrites ci-dessous ont été réalisées avec **Figma**, un outil de conception d'interface utilisateur (UI) et d'expérience utilisateur (UX) basé sur le cloud, permettant aux designers de créer et de collaborer en temps réel.

Figma facilite la création de prototypes et de designs graphiques pour les sites web et les applications mobiles, il est accessible via un navigateur web ou une application de bureau.

## 3.1 Processus de conception

Le processus de conception s'est articulé autour d'une démarche itérative, divisée en deux phases principales : La **création de l'identité visuelle** et le **prototypage**.

### Création de l'identité visuelle

**Moodboarding** : Cette première étape a permis d'explorer des inspirations visuelles évoquant l'univers de l'art génératif et des nouvelles technologies. Le moodboard a servi de référence pour définir l'esthétique globale de la plateforme, en s'inspirant de la science-fiction (notamment du courant *cyberpunk*), du *generative design*, et de formes aux structures particulières présentes à la fois dans la nature et dans certaines œuvres génératives. L'objectif était de refléter à la fois l'innovation technologique et la créativité artistique.

**Définition de la charte graphique** : La charte graphique de NexusLab a été conçue à partir du moodboard dans le but de refléter l'essence technologique et artistique de la plateforme, tout en garantissant une interface utilisateur agréable.

- **Palette de couleurs** : Principalement composée de teintes sombres et de tons néon, cette palette rappelle l'univers numérique et futuriste. Les arrière-plans sombres mettent en valeur les œuvres d'art génératives lumineuses, créant ainsi un contraste visuel fort et une immersion dans l'art digital.

Des touches de couleurs vives (comme le cyan) sont utilisées pour la typographie et certains éléments interactifs, apportant à la fois contraste et lisibilité.

- **Typographie** : Une typographie moderne, géométrique et minimalist a été sélectionnée pour renforcer l'aspect technologique du projet. Des polices sobres, facilement lisible et adaptées aux différents supports (web et mobile), garantissent une lisibilité optimale.

- **Formes et motifs** : Le design est inspiré par les motifs organiques et géométriques que l'on retrouve dans l'art génératif, ces patterns structurés en réseaux évoquent les processus algorithmiques et l'interconnexion des systèmes.



*Moodboard*

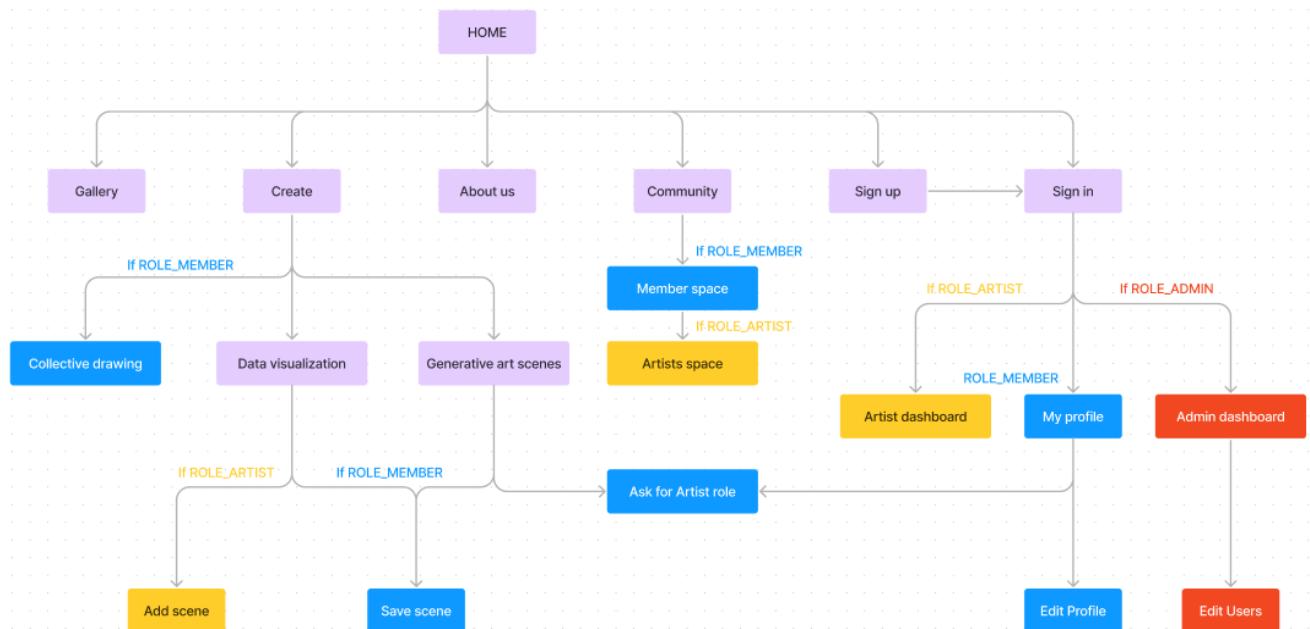


*Charte graphique*

**Création du logo :** Un logo spécifique a été conçu pour NexusLab. Il représente la lettre "N", symbole de "Nexus", qui fait référence à la connexion, à l'interaction de multiples éléments. Inspiré des motifs organiques et des structures de réseau que l'on retrouve dans les systèmes autonomes et les simulations génératives, ce logo symbolise l'interconnexion et la dynamique collaborative, deux éléments centraux dans l'expérience NexusLab.

## Prototypage

**Arborescence fonctionnelle :** L'architecture des pages a été conçue pour offrir une navigation fluide et intuitive. Voici une représentation schématique des connexions entre pages, pensée pour assurer un accès rapide et logique aux principales fonctionnalités.



**Wireframing :** Des wireframes ont été créés pour établir la structure fonctionnelle des pages et les parcours utilisateurs. Cette étape a permis de visualiser l'organisation des fonctionnalités, en s'assurant que l'interface soit simple et intuitive. Chaque section a été pensée pour optimiser l'expérience utilisateur, facilitant la navigation entre les fonctions de création, la consultation de la galerie, et la gestion du profil. Cette étape a permis de réfléchir à l'ergonomie avant d'aborder les aspects esthétiques détaillés.

Voir annexes aux pages 57 et 58.

**Maquettage haute-fidélité :** Après validation des wireframes, des maquettes haute-fidélité ont été développées pour concrétiser le design final. Ces maquettes intègrent la charte graphique, les palettes de couleurs, les typographies, et les éléments interactifs tels que les boutons et les menus déroulants. Ce prototype final met en avant une interface moderne, immersive et interactive, où les œuvres génératives sont mises au premier plan, tout en assurant une expérience utilisateur optimale.

Voir annexes aux pages 59 et 60.

## 3.2 Design (UX/UI)

### Interface Utilisateur

L'interface utilisateur de NexusLab adopte un design à la fois minimaliste et esthétique, favorisant une immersion totale dans le processus créatif. Voici les grands principes qui ont guidé la conception de l'interface :

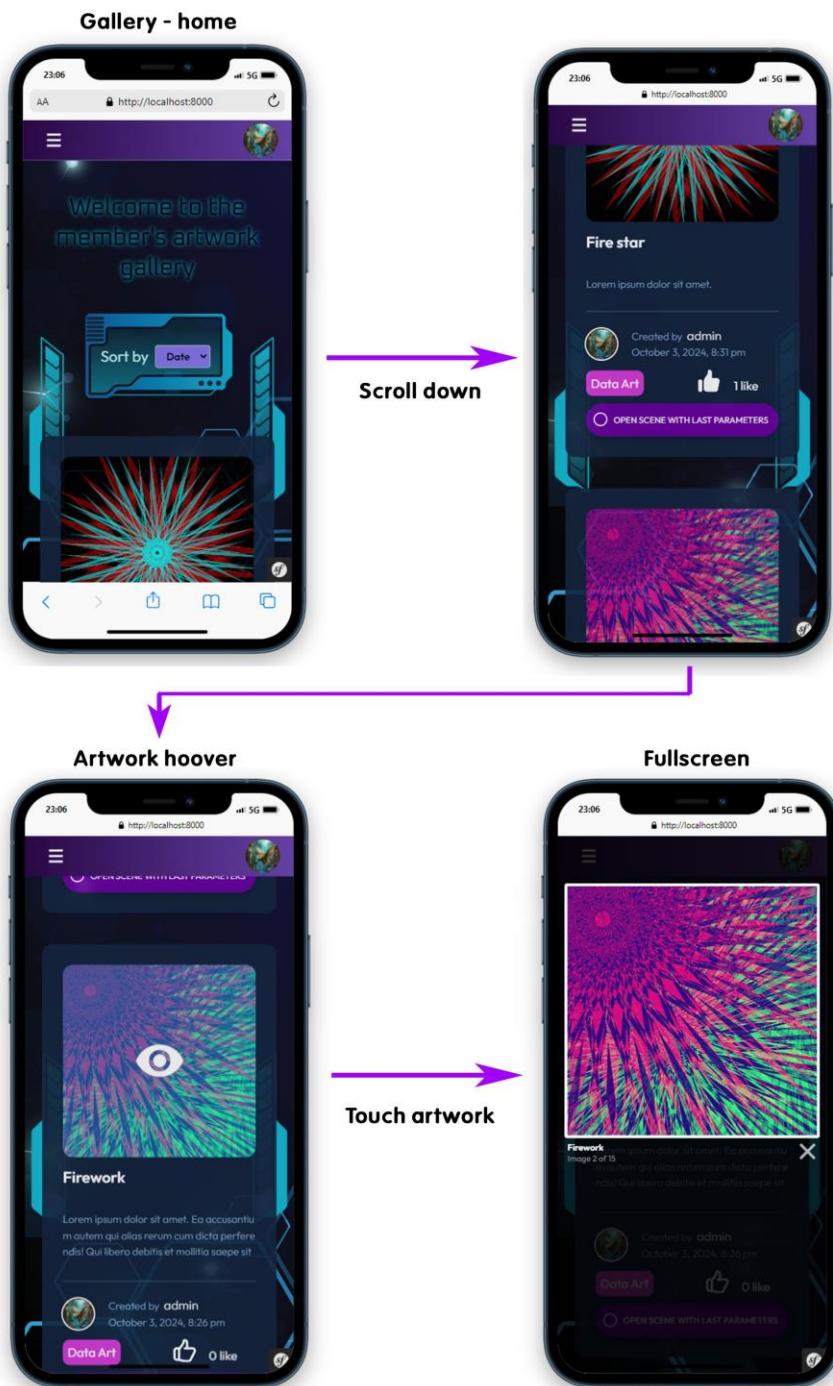
- **Efficacité et simplicité** : Le design vise à éliminer toute complexité superflue, en mettant en avant les œuvres d'art génératives et les fonctionnalités essentielles. Chaque page est pensée pour être épurée, et les utilisateurs peuvent rapidement accéder aux sections principales.
- **Style graphique** : L'inclusion d'éléments de design rappelant la science-fiction et le rétro-gaming a aussi permis d'apporter une dimension ludique à l'expérience utilisateur, favorisant la gamification. Cette approche rend l'interaction avec la plateforme plus engageante, et contribue à créer une atmosphère nostalgie, familiale et amusante, ce qui facilite l'adoption de la plateforme par un public non-expert en code. En rendant l'application visuellement accessible et conviviale, NexusLab encourage la curiosité et invite à l'expérimentation artistique à travers une approche divertissante.
- **Responsive Design** : Un soin particulier a été apporté au responsive design pour garantir une expérience utilisateur cohérente et optimale sur tous les types d'appareils (ordinateurs, tablettes, smartphones). L'interface s'adapte dynamiquement à la taille de l'écran, assurant une navigation ergonomique et efficace, quel que soit le support utilisé. L'agencement des contenus et des fonctionnalités est repensé pour chaque taille d'appareil, offrant ainsi une expérience sur mesure qui facilite la compréhension de l'interface.

### Expérience Utilisateur

L'expérience utilisateur de NexusLab a été pensée pour rendre la navigation intuitive, tout en maximisant l'engagement des utilisateurs grâce à des éléments de gamification. Une attention particulière est accordée aux retours des tests utilisateurs pour constamment optimiser la navigation et l'accès aux différentes fonctionnalités.

- **Gamification** : Afin d'encourager l'exploration et l'apprentissage progressif, des mécaniques de gamification seront progressivement intégrées dans la plateforme. L'idée est d'introduire des objectifs spécifiques à atteindre, comme la création d'un certain nombre d'œuvres ou la participation à des projets collaboratifs. Les utilisateurs peuvent débloquer des niveaux de progression au fur et à mesure qu'ils explorent les différentes fonctionnalités de l'application. Des badges ou récompenses symboliques sont également attribués pour marquer des étapes clés, valorisant ainsi l'implication active. Ces éléments ludiques, combinés à l'esthétique inspirée du rétro-gaming, créent une atmosphère stimulante et immersive. Cette approche contribue également à attirer un public plus large, en rendant l'interaction moins intimidante et plus divertissante.

- Retour des tests utilisateurs :** Des tests utilisateurs réguliers ont été effectués tout au long du développement pour s'assurer que l'expérience utilisateur soit satisfaisante et en adéquation avec les attentes du public cible. Ces tests ont permis de recueillir des retours précieux sur la navigation, l'accessibilité des fonctionnalités et la compréhension générale de l'interface. À partir de ces retours, des ajustements ont été apportés pour améliorer la clarté de l'arborescence, optimiser la réactivité des actions et simplifier encore davantage certains parcours utilisateur. Ces itérations successives ont permis d'affiner l'expérience globale, garantissant que chaque utilisateur, quel que soit son niveau de familiarité avec les outils numériques, puisse naviguer et créer aisément sur la plateforme.



Parcours UX – exploration de la galerie

### 3.3 Accès aux fonctionnalités

L'interface a été conçue pour faciliter l'accès rapide aux principales fonctionnalités de la plateforme. L'accent a été mis sur la simplicité d'utilisation, de manière à ce que les utilisateurs puissent se concentrer sur le processus créatif sans se perdre dans une navigation complexe. Les principales fonctionnalités sont ainsi accessibles de manière claire et ergonomique.

**Créer et partager des œuvres :** La création d'œuvres numériques se fait de manière simple et intuitive, avec des outils accessibles dès l'écran d'accueil. Des boutons bien visibles, permettant de choisir une scène et de commencer une nouvelle œuvre en quelques clics. Une interface épurée permet d'accéder facilement aux outils de création.

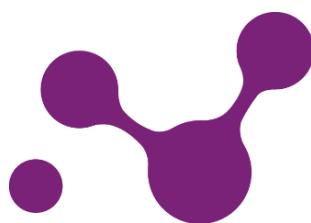
Une fois leur travail terminé, les utilisateurs peuvent publier leurs œuvres en un seul clic, et les partager avec la communauté pour obtenir des retours sous forme de mentions "like". A terme des fonctions de commentaires et la possibilité de "follow" des créateurs seront également implémentés.

**Explorer la galerie et collaborer :** La galerie d'œuvres est au cœur de l'expérience communautaire de NexusLab. Un accès direct à la galerie est offert via la navigation principale, permettant aux utilisateurs de découvrir et d'explorer les créations des autres membres. Chaque œuvre peut être visualisée en détail, "liké", ou même remixée grâce à la fonctionnalité de collaboration. En effet, les utilisateurs peuvent, en un clic, repartir de l'œuvre d'un autre membre pour la retravailler, la modifier ou l'enrichir, favorisant ainsi la collaboration et l'expérimentation artistique collective.

**Navigation rapide et ergonomique :** Afin de fluidifier l'expérience utilisateur, des raccourcis stratégiquement placés ont été intégrés dans l'interface. Sur la version web, une barre de navigation (navbar) située en haut de la page permet un accès rapide aux sections principales de la plateforme : création d'œuvres, galerie, espace communautaire et profil utilisateur.

Sur la version mobile, une barre de navigation inférieure (bottombar) assure une accessibilité similaire, s'adaptant à la taille des écrans pour garantir une expérience cohérente quel que soit le support utilisé. Ce système de navigation réduit le nombre d'étapes nécessaires pour accéder aux principales fonctionnalités, optimisant ainsi le flux de travail des utilisateurs.

**Navigation dynamique selon les rôles :** Les fonctionnalités accessibles varient dynamiquement en fonction des autorisations associées à chaque profil utilisateur. Par exemple, les utilisateurs créateurs et les administrateurs disposent de tableaux de bord dédiés (dashboard Artist ou Admin), leur offrant un accès direct aux outils nécessaires à la gestion de leurs créations ou des tâches d'administration. Cette gestion contextuelle des droits permet d'optimiser l'expérience utilisateur en affichant uniquement les fonctionnalités pertinentes selon leur rôle, garantissant ainsi une interface simplifiée et adaptée à chacun.



# 4. ASPECT FONCTIONNEL ET TECHNIQUE

L'analyse fonctionnelle détaillée du projet NexusLab a été abordée dans le cahier des charges initial. Dans cette section, nous allons examiner les choix technologiques qui ont été retenus et les solutions concrètes mises en œuvre pour répondre aux objectifs fixés. Nous examinerons ensuite les écarts entre les attentes définies et les solutions réellement implémentées, ainsi que les défis rencontrés au cours du développement.

## 4.1 Choix techniques et technologiques

Voici les principales technologies, langages et outils sélectionnés pour le développement de la plateforme NexusLab. Les choix techniques ont été guidés par plusieurs critères, notamment la flexibilité, la facilité d'intégration et la compatibilité avec les fonctionnalités requises.

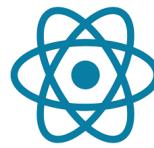
**Langages de programmation et frameworks:**



- **Symfony (PHP)** : Sélectionné pour le développement du backend en raison de sa robustesse, de ses performances et de sa flexibilité. Ce framework de PHP repose sur l'architecture MVC (Modèle-Vue-Contrôleur) garantissant une séparation claire des responsabilités et facilitant ainsi la maintenabilité du code. Sa structure modulaire permet de développer des applications complexes de manière évolutive, tout en simplifiant l'ajout de nouvelles fonctionnalités. Étant un projet open source soutenu par une large communauté, Symfony dispose également d'un vaste écosystème de *bundles* (extensions prêtées à l'emploi), ce qui accélère le développement et la mise en place de fonctionnalités spécifiques. Cela réduit les coûts de développement tout en maintenant des standards élevés de sécurité et de fiabilité.



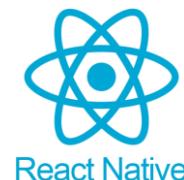
- **Twig (Symfony)** : Ce moteur de templates est utilisé avec Symfony pour le rendu des vues côté serveur. Il permet de générer du HTML de manière efficace et sécurisée, tout en offrant une syntaxe simple et lisible, et il automatise l'échappement des variables pour prévenir les failles XSS. Associé à Symfony, Twig facilite la création d'interfaces web dynamiques et maintenables.



- **React (JavaScript)** : Bibliothèque de JavaScript sélectionnée pour sa capacité à créer des interfaces dynamiques et performantes. En combinant Twig pour le rendu serveur et React pour les composants interactifs, nous assurons une fluidité dans l'affichage tout en améliorant l'expérience utilisateur. Ces composants pourront ensuite être réutilisés dans la version mobile développée en React Native.



- **p5.js (JavaScript)** : Bibliothèque spécialisée choisie pour la création d'œuvres d'art génératives. Elle s'intègre facilement avec l'environnement web et permet une exécution en temps réel de code graphique interactif directement dans le navigateur.



- **React Native (JavaScript)** : Framework choisi pour le développement de l'application mobile, il permet de déployer rapidement une application sur iOS et Android, avec une expérience utilisateur cohérente et fluide sur les deux plateformes. Grâce à son architecture réactive, React Native favorise la création de composants dynamiques et réutilisables, optimisant ainsi le temps de développement et la maintenabilité de l'application. Il permet le partage de code avec les composants React de la version web.

## Outils de développement :



- **IDE (Integrated Development Environment)** : **Visual Studio Code**

Cet éditeur de code est reconnu pour sa flexibilité, ses nombreuses extensions, et sa capacité à gérer de nombreuses technologies. Il offre une excellente prise en charge de Symfony, React et React Native, les principales technologies du projet. Sa compatibilité avec GitHub via des extensions facilite également le contrôle des versions directement dans l'éditeur.



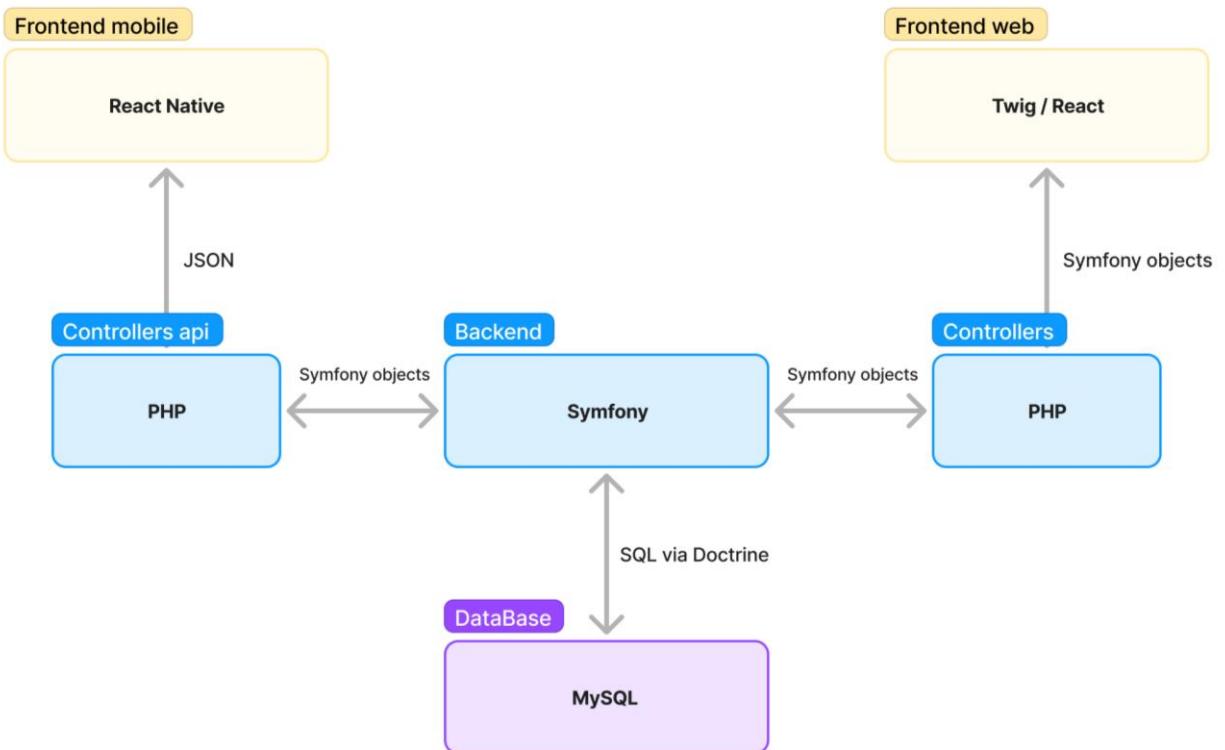
- **GitHub** : Utilisé pour la gestion du versionning et des branches de développement.



- **phpMyAdmin** : Utilisé comme interface web pour la gestion de la base de données MySQL, cet outil offre une interface claire pour exécuter des requêtes SQL, gérer les tables, et effectuer des opérations de maintenance sur la base de données.

## 4.2 Architecture globale du système

Le projet NexusLab repose sur une architecture modulaire, organisée selon le modèle MVC (Modèle-Vue-Contrôleur), qui permet de dissocier les différentes couches de l'application, améliorant ainsi la clarté du code et facilitant sa maintenabilité.



## Architecture MVC

**Modèle (Backend Symfony) :** Le modèle de données est géré par Symfony, qui communique avec une base de données relationnelle (MySQL) via son ORM Doctrine (Object-Relational Mapping). Cette architecture permet une manipulation simplifiée et efficace des données tout en assurant une sécurité et une cohérence dans leur traitement.

**Vue (Frontend Twig + React) :** Le rendu initial des pages côté serveur est assuré par Twig, celui-ci reçoit les données des contrôleurs. Ces données sont également transmises aux composants React avant qu'ils ne soient rendus côté client.

**Contrôleur (Controllers Symfony)** : Le contrôleur sert d'intermédiaire entre le modèle et la vue, il gère la logique métier ainsi que la communication avec le frontend. Dans le projet, les controllers sont répartis en deux sections distinctes :

- Les controllers classiques communiquent avec le frontend web basé sur Twig, et assurent le rendu des templates HTML.
- Les controllers API agissent eux comme une interface permettant l'échange de données entre l'application mobile et le backend via le format JSON. Ils gèrent la conversion des données JSON en objets PHP et vice versa, ce qui permet de traiter et d'intégrer ces données au sein de l'application.

## Base de données



Une base de données relationnelle MySQL a été choisie pour sa fiabilité et sa capacité à gérer un volume important de données, notamment les œuvres d'art des utilisateurs et leurs informations personnelles. MySQL s'intègre parfaitement avec Symfony, facilitant ainsi la gestion des relations entre les différentes entités du système. Doctrine simplifiera également l'exécution des requêtes en évitant d'avoir à écrire directement en SQL, tout en optimisant les performances lors des requêtes complexes.

## Choix d'un backend commun

L'utilisation d'un backend commun, basé sur Symfony, pour les deux versions (web et mobile) présente plusieurs avantages. Cela permet d'économiser la logique métier en centralisant les traitements des données et les contrôles de sécurité, évitant ainsi une duplication du code. De plus, cette approche garantit une expérience utilisateur cohérente, car les mêmes fonctionnalités et données sont accessibles quel que soit le support utilisé, que ce soit sur l'application web ou mobile. Ainsi, en changeant de support, l'utilisateur retrouve le même contenu et les mêmes interactions, renforçant l'uniformité et la familiarité de l'interface.

## 4.3 Bilan des fonctions implémentées et des moyens utilisés

Voici une analyse comparative entre les attentes initiales spécifiées dans le cahier des charges et les choix techniques finalement réalisés. Ce tableau synthétise les fonctionnalités implémentées, les technologies ou méthodes utilisées pour les concrétiser, ainsi que les raisons pour lesquelles certaines fonctionnalités n'ont pas été mises en place. Il permet ainsi d'évaluer les écarts entre les objectifs du projet et les solutions réellement adoptées.

| Fonction                                   | Application Web |   | Application Mobile |  |
|--|-----------------|---|--------------------|--|
|  | Implémentation  | Moyens utilisés (ou raison de non-implémentation) | Implémentation     | Moyens utilisés (ou raison de non-implémentation)              |
| Authentification sécurisée                 | Oui             | Symfony SecurityBundle + MakerBundle              | Oui                | JSON Web Token (JWT) + Refresh token                           |
| Création d'œuvres : Générative Art         | Oui             | Symfony, Twig, p5.js, MySQL                       | Oui                | React Native + webview, p5.js, API Symfony, MySQL              |
| Création d'œuvres : Data Vizualisation Art | Oui             | Symfony, Twig, p5.js                              | Oui                | React Native + webview, p5.js, API Symfony                     |
| Sauvegarde et partage des œuvres           | Oui             | Symfony, Twig, p5.js, MySQL                       | Oui                | React Native + webview, p5.js, API Symfony, MySQL              |
| Galerie communautaire                      | Oui             | Symfony, Twig, MySQL                              | Oui                | React Native, MySQL  |
| Remix d'œuvres existantes                  | Oui             | Symfony, Twig, MySQL                              | En cours           | En cours d'implémentation                                      |
| Système de like                            | Oui             | React, MySQL                                      | Oui                | React Native, MySQL  |
| Système de commentaires                    | Non             | Contrainte de temps                               | Non                | Contrainte de temps  |
| Système de follower                        | Non             | Contrainte de temps                               | Non                | Contrainte de temps  |
| Profil utilisateur                         | Oui             | React, MySQL                                      | Oui                | React Native, MySQL  |
| Editer profil                              | Oui             | Symfony, Twig, MySQL                              | Oui                | React Native, MySQL  |
| Galerie utilisateur                        | Oui             | Symfony, Twig, MySQL                              | Oui                | React Native, MySQL  |
| Editer Œuvres                              | Oui             | Symfony, Twig, MySQL                              | Oui                | React Native, MySQL  |
| Demandeur rôle "Artist"                    | Oui             | Symfony, Twig, MySQL                              | Non                | Redirection vers la version web                                |
| Ajouter une scène                          | Oui             | Symfony, Twig, MySQL                              | Non                | Disponible seulement sur la version web (envoi de code source) |
| Artist Dashboard                           | Oui             | Symfony, Twig, MySQL                              | Non                | Disponible seulement sur la version web                        |
| Admin Dashboard                            | Oui             | Symfony, Twig, MySQL                              | Non                | Disponible seulement sur la version web                        |
| Live Coding                                | Non             | Contrainte de temps                               | Non                | Contrainte de temps  |
| Collective drawing                         | Non             | Contrainte de temps                               | Non                | Contrainte de temps  |
| Espace communautaire (forum)               | Non             | Non prévu dans le premier prototype               | Non                | Non prévu dans le premier prototype                            |
| Gamification (quêtes, badges, niveaux)     | Non             | Non prévu dans le premier prototype               | Non                | Non prévu dans le premier prototype                            |
| Système de notifications                   | Non             | Contrainte de temps                               | En cours           | En cours d'implémentation                                      |
| Accessibilité (normes WCAG)                | En cours        | En cours d'optimisation                           | En cours           | En cours d'optimisation  |

## Cas de l'ajout de nouvelles scènes

Pour ajouter de nouvelles scènes, les utilisateurs ayant obtenu le rôle "Artist" peuvent soumettre une proposition via un formulaire destiné à l'équipe NexusLab. Ce formulaire doit inclure le code source de la scène, rédigé selon les directives fournies dans la documentation de l'application. Après validation du code par l'équipe, la scène pourra être intégrée à la section dédiée de la plateforme. Cette fonctionnalité nécessitant l'upload de code source, il a été naturellement décidé de la rendre disponible seulement sur la version desktop.

## Analyse des écarts entre les attentes initiales et les choix techniques réels

Certaines fonctionnalités prévues initialement dans le cahier des charges n'ont pas pu être implémentées, principalement pour des raisons de contraintes temporelles et techniques.

Par exemple, bien que l'application ait été conçue pour implémenter un système de commentaires et de followers dès le premier prototype, ces fonctionnalités n'ont pas pu être intégrées en raison de priorités de développement et de manque de temps.

Certaines fonctionnalités comme le live coding et le collective drawing, bien qu'anticipées, ont été reportées à des versions ultérieures en raison de leur complexité technique (notamment la synchronisation en temps réel de multiples utilisateurs via des WebSockets) et des délais serrés.

Cependant, les fonctionnalités principales, telles que la création et le partage d'œuvres, l'authentification sécurisée, et la gestion des profils utilisateurs, ont été implémentées avec succès. Ce prototype constituera une base pour recueillir les retours des premiers utilisateurs, afin d'orienter les améliorations à venir. De nouvelles fonctionnalités seront progressivement ajoutées à travers les mises à jour, permettant d'enrichir l'expérience utilisateur et d'optimiser la plateforme.

## 4.4 Difficultés rencontrées et solutions apportées

### Contraintes de temps

Le principal défi rencontré a été le manque de temps pour développer l'ensemble des fonctionnalités initialement prévues. Il a donc été nécessaire de prioriser les fonctionnalités essentielles afin de pouvoir livrer un premier prototype viable.

Certaines fonctionnalités sont encore en cours d'implémentation, tandis que d'autres ont été reportées à des phases de développement ultérieures.

### Complexité du cross-plateforme web / mobile

L'adaptation de l'application web à la version mobile a posé des défis importants. Pour l'intégration des scènes génératives notamment, la bibliothèque p5.js repose sur le DOM pour son fonctionnement, or React Native ne dispose pas d'un DOM. Pour contourner cette limitation, il a fallu utiliser une *WebView*, un composant natif qui permet d'exécuter du code HTML/JavaScript au sein de l'application mobile, mais complexifie l'intégration directe de ces fonctionnalités.

La gestion de l'authentification et la sécurisation des pages et des requêtes ont également présenté un réel challenge technique. Contrairement à la version web qui utilise Symfony SecurityBundle pour la gestion des sessions et des accès sécurisés, la version mobile a nécessité la mise en place d'un système basé sur JSON Web Tokens (JWT) associé à un refresh tokens. Cette solution permet d'assurer une authentification sécurisée tout en répondant aux spécificités des échanges avec une application mobile, mais elle a impliqué des ajustements considérables dans la logique d'authentification et la gestion des accès, particulièrement pour le système de refresh token.

### Optimisation des performances

Afin d'assurer une expérience utilisateur fluide malgré la charge importante que représente l'art génératif en temps réel, des ajustements ont été apportés au code p5.js pour limiter la consommation de ressources des scènes. La complexité des scènes est donc contrainte par la technologie employée et par les capacités de la machine de l'utilisateur.

Une solution envisagée serait de créer les scènes en utilisant des langages plus performants en termes de gestion des ressources, comme le GLSL, et de tester leur exécution dans des environnements variés. Cela permettrait de garantir des performances optimales pour tous les utilisateurs, indépendamment de leur configuration matérielle.

## 4.5 Optimisation et ajout de fonctionnalités à venir

Les futures optimisations concerteront principalement l'amélioration des performances et l'ajout de nouvelles fonctionnalités :

### Fonctionnalités à venir

- **Espace communautaire** : Une plateforme d'échanges entre utilisateurs de type forum est en cours de planification, permettant de favoriser l'interaction autour de la création d'œuvre et de scènes génératives.
- **Live coding et collective drawing** : Ces fonctionnalités complexes sont prévues pour les futures versions, avec l'intégration de **WebSockets** pour la collaboration en temps réel.
- **Gamification** : Des éléments de gamification tels que des quêtes, des badges et des niveaux d'utilisateur sont en cours de réflexion et seront ajoutés pour stimuler l'engagement des utilisateurs.
- **Système de notifications** : Actuellement en cours d'implémentation, il permettra d'informer les utilisateurs en temps réel sur l'activité de la plateforme.
- **Accessibilité** : L'optimisation continue de la conformité aux normes WCAG pour assurer que la plateforme soit pleinement accessible à tous les utilisateurs.

### Optimisations techniques

- Des tests de charge plus poussés seront effectués pour garantir la capacité du système à gérer un grand nombre d'utilisateurs simultanés, en particulier pour des fonctionnalités de co-creation comme le live coding et le collective drawing.
- Les temps de réponse des requêtes seront également optimisés, notamment via la mise en cache des données les plus demandées.
- Une attention continue sera portée à l'évolution des technologies de creative coding, en cherchant à optimiser les scènes existantes et à adopter les technologies les plus performantes et les plus largement utilisées au sein de la communauté des créateurs.

Ces évolutions garantiront que NexusLab continue de s'améliorer en termes de performances et d'expérience utilisateur tout en restant fidèle à ses objectifs créatifs et communautaires.

# 5. CREATIVE CODING

Le creative coding est au cœur du projet NexusLab, permettant aux utilisateurs de créer des œuvres d'art numériques en temps réel à partir de scènes algorithmiques paramétrables.

La bibliothèque principale employée pour la réalisation de ces scènes est p5.js, voici une explication de son fonctionnement et une analyse de son rôle dans le projet.

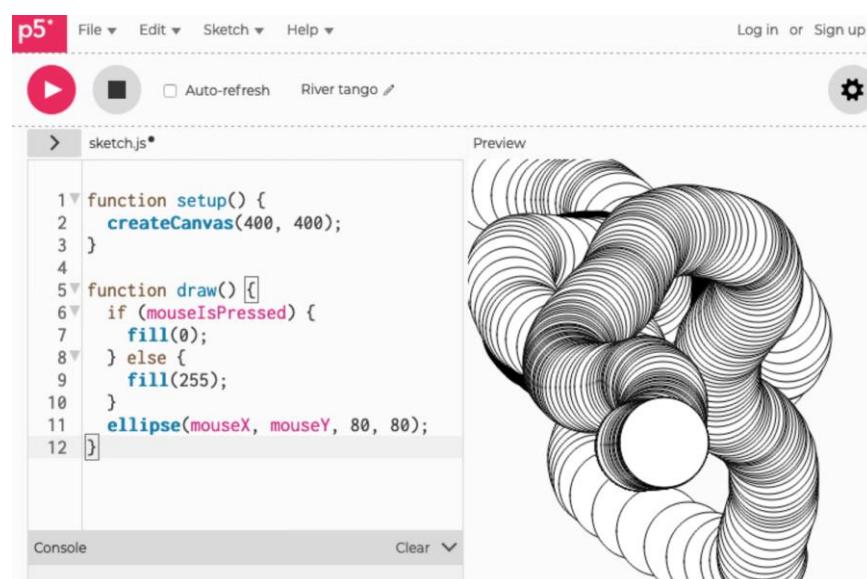
## 5.1 Fonctionnement général de p5.js

### Structure d'un sketch

p5.js repose sur une structure simple mais puissante, divisée en deux fonctions principales : *setup()* et *draw()*. Ces fonctions définissent la base de tout sketch en p5.js, permettant aux créateurs de spécifier l'initialisation de leurs éléments graphiques et de définir ce qui est dessiné à l'écran.

- **setup()** : Cette fonction est exécutée une seule fois, au démarrage du sketch. C'est ici que les paramètres globaux sont définis, comme la taille du canevas, les couleurs ou les variables de base.
- **draw()** : Après *setup()*, la fonction *draw()* est exécutée en boucle, rafraîchissant en permanence l'affichage de l'œuvre. Elle permet de créer des animations, de réagir aux interactions utilisateur, ou de générer des visualisations en temps réel.

Ces deux fonctions permettent de gérer le cycle de vie du sketch. Par exemple, un artiste peut définir des formes géométriques, des animations ou des images dans *setup()*, puis gérer leur mouvement ou leur évolution dans *draw()*.



Cet exemple fait apparaître des ellipses d'un diamètre de 80px, la position de chaque ellipse dépend de celle du curseur de la souris. Lors ce que l'utilisateur clique sur le bouton gauche de sa souris, la couleur de l'ellipse passe du blanc au noir jusqu'à ce que le bouton soit relâché.

## Aspect fonctionnel

Cette bibliothèque offre également un large éventail de fonctions pour gérer :

- L'animation (2D et 3D).
- Les interactions utilisateur avec `mouseClicked()`, `keyPressed()` par exemple.
- La manipulation de formes et de couleurs.
- Des fonctions mathématiques avancées comme `noise()` et `random()`, qui permettent de créer des effets visuels dynamiques et des comportements aléatoires.
- La gestion des médias (son, vidéo) avec des fonctions comme `loadSound()` ou `createVideo()`.
- L'utilisation de shaders avec WebGL, pour des effets 3D et graphiques encore plus complexes.
- La manipulation des données avec des fonctions pour charger les données externes comme `loadJSON()`, `loadTable()`, ou encore `loadXML()`.

### Manipulation de données

p5.js permet donc également de manipuler des données externes pour créer des visualisations interactives ou dynamiques. Il est possible de charger des fichiers de data sous différents formats, des flux de données ou d'interagir avec des API. Ces données peuvent ensuite être interprétées pour modifier des formes graphiques ou des animations en temps réel.

## Les possibilités

Les utilisateurs peuvent créer un large éventail de visuels allant de simples dessins géométriques à des visualisations complexes de données. Les interactions avec la souris, le clavier, ou avec des éléments du DOM, peuvent être facilement intégrées pour rendre les créations plus immersives et interactives. Cette flexibilité offre une infinité de possibilités créatives, depuis l'art génératif jusqu'à des œuvres interactives et expérimentales.

## 5.2 Intégration de p5.js au projet

### Utilisation de p5.js dans la plateforme

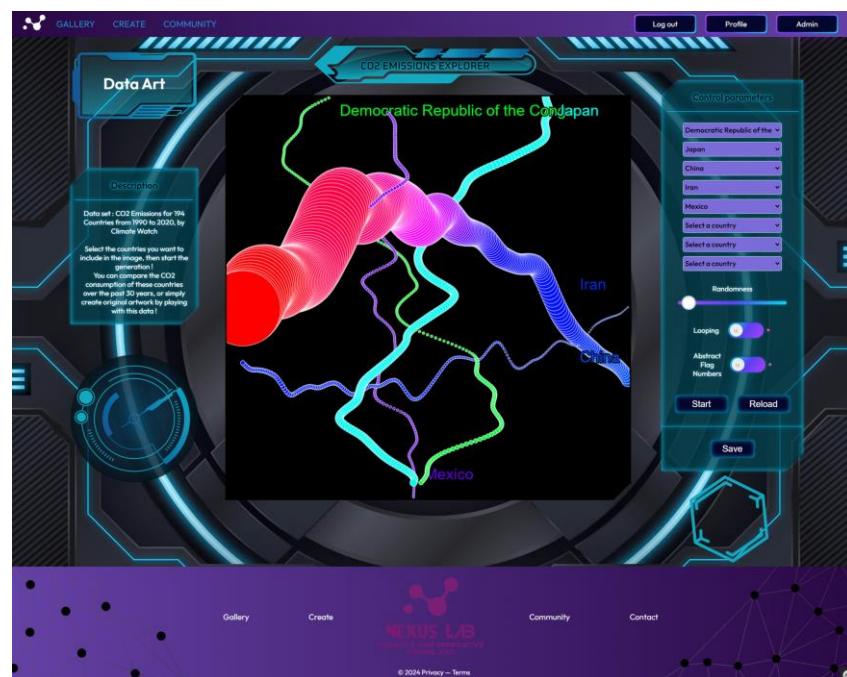
p5.js est utilisé pour permettre aux utilisateurs de créer des œuvres génératives directement dans leur navigateur. L'intégration des sketches p5.js avec la structure de l'application permet une interaction fluide avec le backend, notamment pour la sauvegarde, le partage et le remix d'œuvres d'art. Les utilisateurs peuvent éditer et partager leurs créations dans un environnement collaboratif, où p5.js devient un outil créatif central.

### Exemples d'applications concrètes

- **Création d'art génératif :** Ici l'utilisateur peut créer des œuvres basées sur un algorithme génératif de « random line walkers ». Chaque clic sur le canvas va lancer une animation qui ensuite évoluera de manière autonome. La forme, le mouvement et la couleur de ces animations varient en fonction des paramètres que l'utilisateur aura choisis.



- **Data Art Visualisations :** Intégration de datasets pour créer des animations basées sur des visualisations de données réelles. Ici l'évolution des émissions de CO2 par pays au cours du temps.



## Et les autres langages ?

Bien que p5.js soit un choix naturel pour une application web, d'autres technologies pourraient être envisagées à l'avenir pour améliorer la performance et étendre les possibilités créatives. Des langages comme GLSL (pour la création de shaders) ou des frameworks graphiques plus optimisés pour des calculs complexes pourraient être utilisés pour implémenter des scènes plus performantes. L'architecture de NexusLab reste ouverte à l'intégration d'autres outils et langages pour soutenir l'évolution des besoins créatifs des utilisateurs.

# 6. CONFIGURATION DE L'ENVIRONNEMENT DE TRAVAIL

La mise en place de l'environnement de travail pour ce projet a nécessité l'utilisation et la configuration de nombreux outils et dépendances, afin de garantir une intégration cohérente et efficace des différents composants du système.

## 6.1 Backend Symfony

Le backend du projet NexusLab repose sur le framework PHP Symfony. La première étape a donc consisté à installer *WampServer*, un serveur local permettant de gérer PHP, MySQL, et Apache. Cet environnement permet d'exécuter le code PHP et de gérer la base de données MySQL via phpMyAdmin.

**Composer** a ensuite été installé via le terminal dans Visual Studio Code, ce gestionnaire de dépendances PHP permet d'installer facilement des bibliothèques externes (bundle, packages, frameworks), tout en garantissant la compatibilité des versions. Il a été utilisé pour initialiser le projet Symfony.

Parmi les principales dépendances intégrées au projet, on retrouve :

- **Doctrine ORM** : Utilisé pour la gestion des entités et la manipulation de la base de données.
- **SecurityBundle** : Permettant la gestion des autorisations, des authentifications et des droits d'accès aux différentes routes de l'application web.
- **JWT Authentication (Lexit JWT bundle)** : Utilisé spécifiquement pour gérer l'authentification des utilisateurs sur la version mobile via un système de jetons sécurisés.

Ainsi que d'autres bundles installés automatiquement lors de l'initialisation du projet Symfony, et nécessaires à son bon fonctionnement (twig-bundle, framework-bundle, form, etc).

La gestion des données est donc assurée par Doctrine ORM, facilitant l'interaction avec la base de données en utilisant des objets PHP plutôt que des requêtes SQL directes. La configuration des accès à la base de données, ainsi que d'autres paramètres essentiels pour chaque environnement (développement, test, production), est définie dans le fichier .env de Symfony.

```
> vendor           26 # DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
.ENV             27 DATABASE_URL="mysql://root:@127.0.0.1:3306/projet-cda?serverVersion=8&charset=utf8mb4"
$ .ENV.test       28 # DATABASE_URL="postgresql://app:1ChangeMe@127.0.0.1:5432/app?serverVersion=16&charset=utf8"
                     29 ##### doctrine/doctrine-bundle #####
```

*Configuration Symfony - connexion à la base de données MySQL*

## 6.2 Intégration de React

L'intégration de composants React dans les vues Twig de l'application web a nécessité l'installation de plusieurs bundles :

- **ux-react-bundle** : Un bundle Symfony issu de la collection Symfony UX. Il permet une intégration naturelle de composants React directement dans les templates Twig. Grâce à ce bundle, il est possible de rendre et d'injecter dynamiquement des composants React au sein des pages Symfony en utilisant une syntaxe simple dans les fichiers Twig. ux-react-bundle prend également en charge la gestion des propriétés des composants (props), facilitant ainsi la communication entre le backend Symfony et le frontend React.
- **Webpack Encore** : Joue un rôle déterminant en facilitant la gestion des assets (fichiers JavaScript, CSS, images) et l'intégration de modules JavaScript. Il simplifie le processus de regroupement et de minification des fichiers, améliorant ainsi la performance et le chargement des pages. Grâce à Webpack Encore, les modules React sont automatiquement empaquetés pour être utilisés dans l'application, tout en s'assurant que les dépendances sont correctement importées et intégrées.
- **Babel** : Ce compilateur JavaScript se charge de traduire les fichiers JSX (React) en JavaScript standard compatible avec la plupart des navigateurs web. Babel permet donc de rendre les composants React fonctionnels dans n'importe quel environnement web, quel que soit le niveau de support JavaScript des navigateurs.

Lors du développement, la commande "**npm run watch**" est utilisée pour activer un processus de veille sur les fichiers. À chaque modification des composants React, Webpack Encore recompile automatiquement le code, s'assurant que les changements sont immédiatement reflétés dans l'application sans nécessiter une recompilation manuelle.

Ainsi, l'utilisation combinée de ces trois bundles permet une intégration optimale de React dans une architecture Symfony.

## 6.3 Frontend Mobile

La version mobile est entièrement développée en **React Native**. L'environnement mobile a été configuré en utilisant **Node.js** pour gérer le développement JavaScript, il intègre **npm** (Node Package Manager) pour l'installation des dépendances. Les principaux outils et bibliothèques incluent :

- **Expo**, un ensemble d'outils et d'API qui facilite l'initialisation, le développement et le déploiement des applications React Native tout en simplifiant la configuration de l'environnement natif.
- **Metro bundler**, outil essentiel dans le développement React Native, permettant de démarrer un serveur local mais aussi de regrouper et de compiler les fichiers pour qu'ils soient exécutés correctement dans un émulateur ou sur un appareil mobile réel.
- **Expo go**, une application mobile fonctionnant avec Metro Bundler. Elle permet de tester et de prévisualiser l'application React Native directement sur un appareil mobile en scannant un QR code généré par Expo au lancement du serveur.

- **Axios**, pour gérer les appels API. Cette bibliothèque JavaScript permet de gérer efficacement les requêtes HTTP, facilitant la communication entre le frontend React Native et les controllers API du backend Symfony.

## 6.4 Gestion des dépendances

Dans la configuration du projet, des fichiers spécifiques au format JSON gèrent les dépendances pour le backend et le frontend. Ils assurant une gestion cohérente des dépendances et facilitent l'installation sur différents environnements.

- **composer.json** : Utilisé avec Symfony, il liste les bibliothèques PHP nécessaires au projet. Composer s'assure que les versions sont compatibles et permet d'ajouter ou mettre à jour facilement ces dépendances.
- **package.json** : Pour le frontend (React et React Native), il répertorie les bibliothèques JavaScript et automatise des tâches comme la compilation et le démarrage du serveur de développement avec npm.

## 6.5 Gestion des versions

La gestion des versions est un aspect essentiel du développement de projet, permettant de sécuriser le code, d'assurer le suivi des modifications et de faciliter la collaboration. Pour ce projet, le système de contrôle de version distribué Git a été utilisé en association avec GitHub pour héberger le code.

### Utilisation de Git

Dès le début du projet, un dépôt Git a été initialisé localement. Les principales actions effectuées incluent :

- **Initialisation du dépôt** : La commande `git init` a été utilisée pour créer un dépôt Git dans le répertoire du projet, permettant ainsi de suivre les modifications apportées au code.
- **Commits réguliers** : Des commits ont été réalisés à chaque étape significative du développement, enregistrant les changements apportés au code avec des messages descriptifs. Cela a permis de garder une trace des évolutions et de revenir facilement à des versions antérieures si nécessaire.

### Utilisation de GitHub

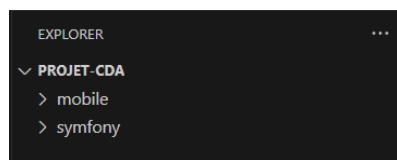
Pour faciliter la collaboration et la sauvegarde des données, un dépôt distant a été créé sur GitHub. Cela a permis de :

- **Héberger le code** : Le dépôt distant a servi de sauvegarde pour le code source, garantissant que les modifications sont sécurisées et accessibles depuis n'importe quel endroit.
- **Collaboration** : Possibilité de partager le code avec d'autres développeurs potentiels et de leur permettre d'apporter des contributions via des pull requests.
- **Suivi des versions** : Grâce à GitHub, chaque commit et chaque pull request sont automatiquement enregistrés, fournissant un historique clair des modifications et facilitant la gestion des versions au fur et à mesure que le projet évolue.

# 7. RÉALISATION DU PROJET

## 7.1 Développement global du projet

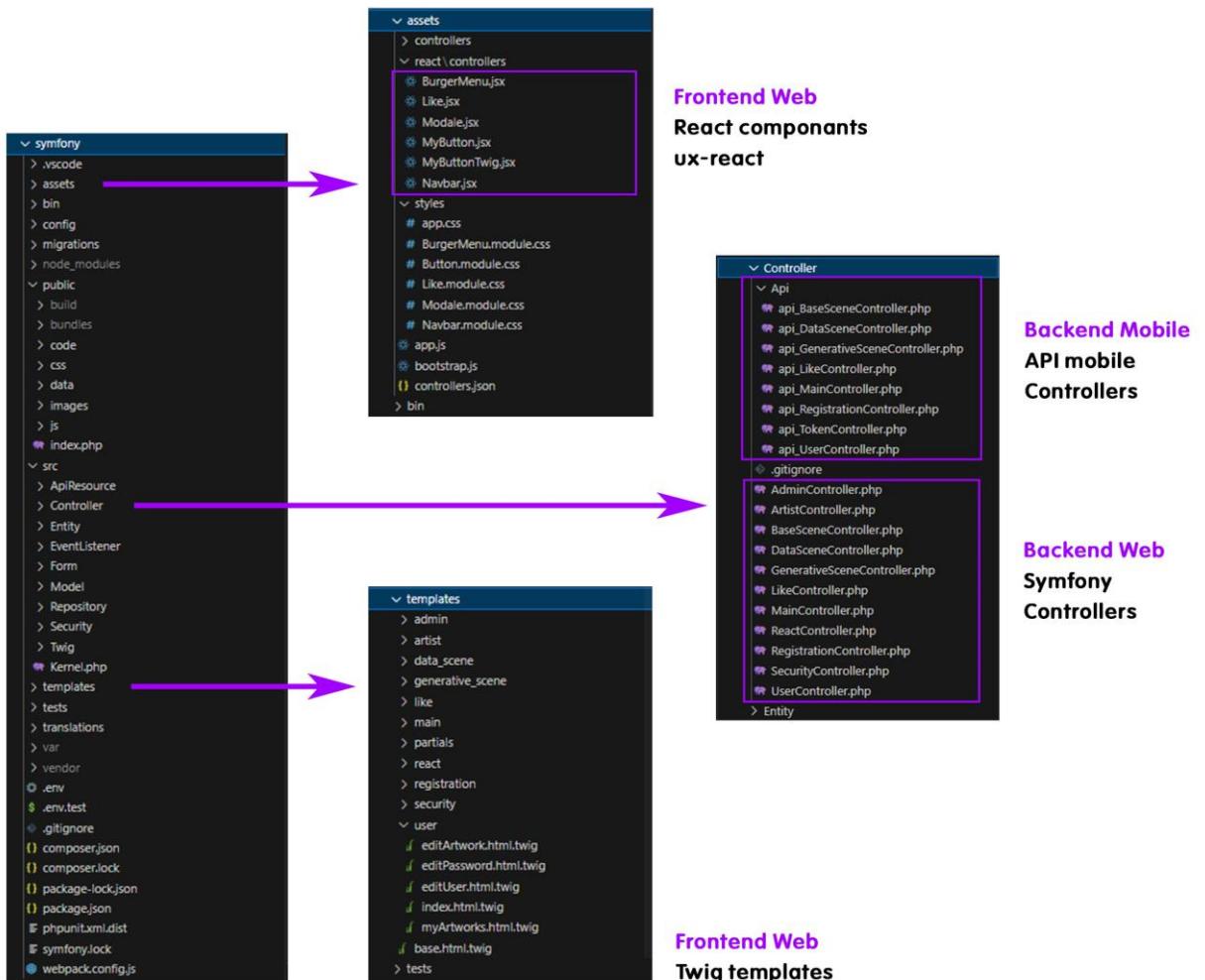
### Structure finale du projet



L'architecture du projet a été structurée de façon modulaire afin de séparer l'application web basée sur Symfony, de l'application mobile basée sur React Native.

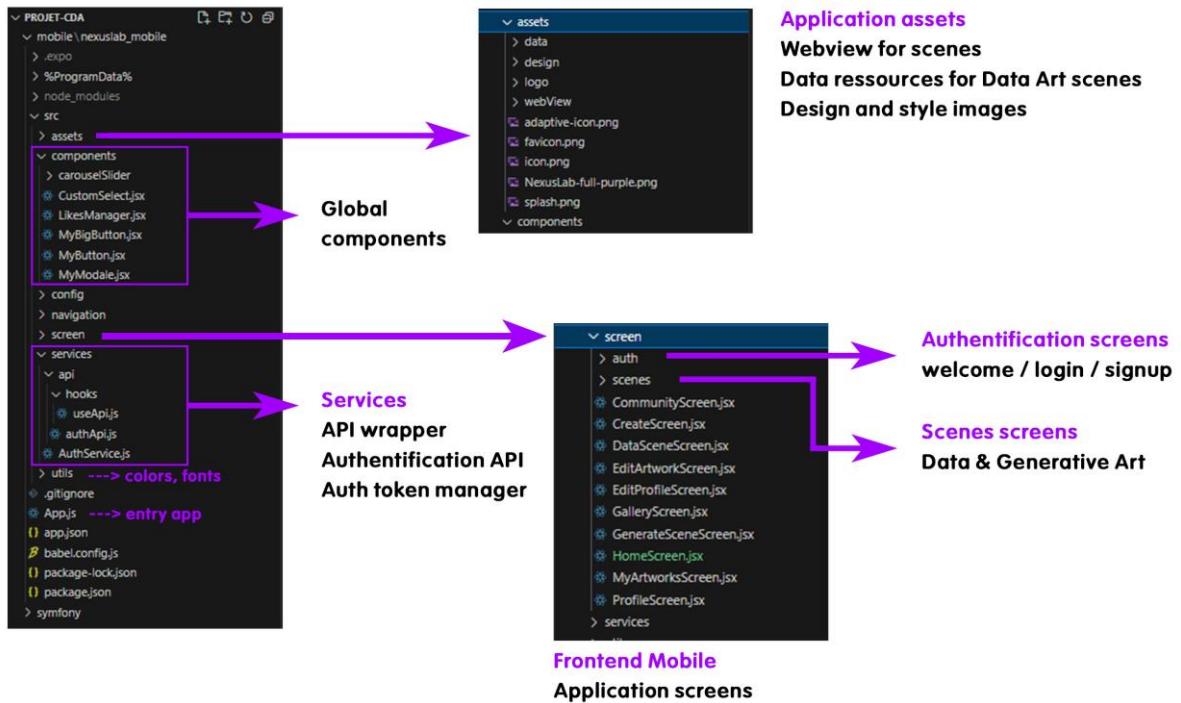
#### Arborescence de l'application web

L'application web est organisée selon la structure classique d'un projet Symfony. Avec des dossiers dédiés à la gestion du backend (Controller, Repository, Entity), un dossier réservé au frontend (templates Twig), mais aussi un espace dédié à React pour les composants interactifs.



## Arborescence de l'application mobile

L'application mobile suit la structure standard d'un projet React Native.



## Gestion des données

Le modèle de données a été structuré autour de plusieurs entités principales (entity), définissant les relations entre utilisateurs, œuvres, interactions (likes) et demandes spécifiques (demande de rôle Artist et soumission de nouvelles scènes). Doctrine ORM facilite la gestion de ces entités et leurs relations, en permettant à chaque entité d'être transformée en table dans la base de données lors des migrations.

Les tables et leurs propriétés sont donc créées ou modifiées en fonction de l'évolution du modèle de données. Ce système permet de suivre un historique des migrations, garantissant une gestion efficace et une traçabilité des changements dans la structure de la base de données.

```
// ----- Generative art scenes -----
#[ORM\OneToMany(targetEntity: Scene1::class, mappedBy: "user")]
private $scene1;

#[ORM\OneToMany(targetEntity: Scene2::class, mappedBy: "user")]
private $scene2;

// ----- Data art scenes -----
#[ORM\OneToMany(targetEntity: SceneD1::class, mappedBy: "user")]
private $sceneD1;

#[ORM\OneToMany(targetEntity: SceneD2::class, mappedBy: "user")]
private $sceneD2;

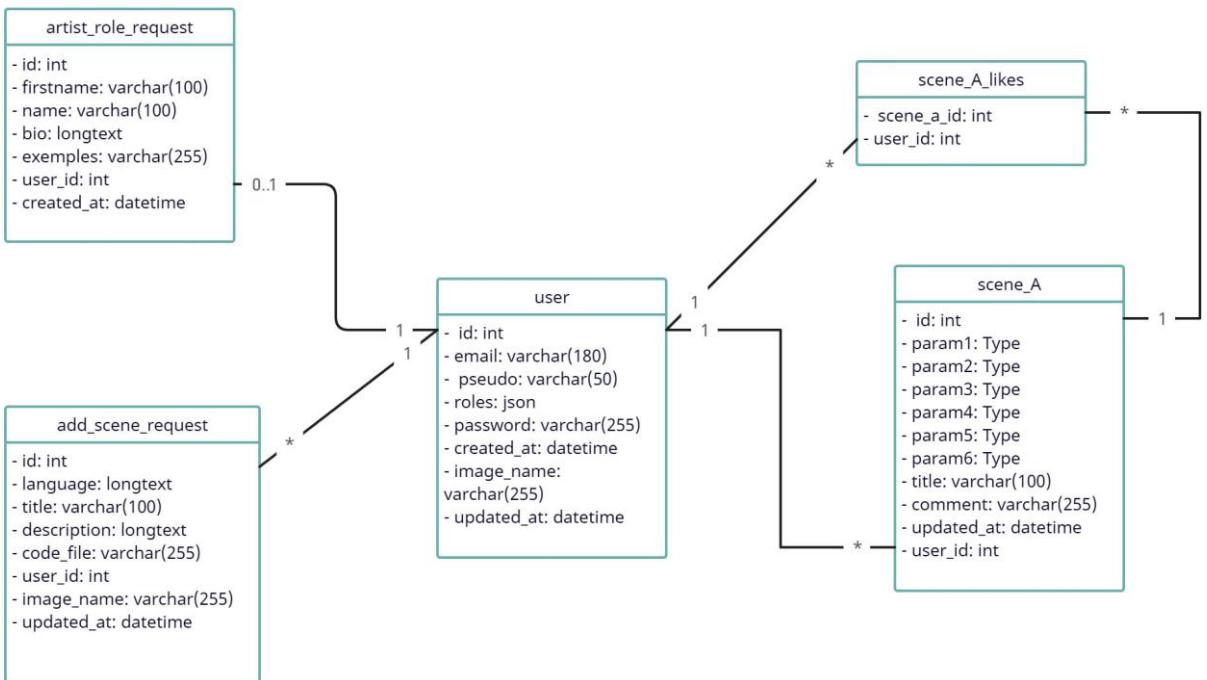
// ----- Requests -----
#[ORM\OneToOne(targetEntity: ArtistRole::class, mappedBy: "user")]
private $role_request;

#[ORM\OneToMany(targetEntity: AddScene::class, mappedBy: "user")]
private $add_scene;
```

Doctrine gère la création et la gestion des tables via des attributs dans les entités (qui ont remplacé les annotations depuis Symfony 5.2). Par exemple, les relations entre l'utilisateur et les œuvres sont représentées par des annotations `OneToMany` dans l'entity `User`.

## Schéma MCD (Modèle Conceptuel de Données)

Le schéma MCD ci-dessous illustre les relations entre les principales entités :



- Chaque utilisateur peut créer et "liker" plusieurs œuvres.
- Un utilisateur peut soumettre une seule demande pour obtenir le rôle *Artist*.
- Un utilisateur *Artist* peut soumettre plusieurs scènes pour validation et ajout à la plateforme.

**Concepteur complet : voir l'annexe à la page 61.**

## Gestion sécurisée des utilisateurs

La gestion sécurisée des utilisateurs dans le projet repose sur le système de sécurité de Symfony, utilisant des pare-feu (firewalls) pour protéger les différentes sections de l'application web et mobile. Chaque firewall gère une partie spécifique de la sécurité, adaptée à l'usage de l'application, que ce soit pour l'authentification classique sur la version web ou via des token JWT pour la version mobile.

Lors de l'installation du SecurityBundle, un fichier *security.yaml* est généré. Il contient la configuration pour la sécurité de l'application, incluant : Le hashage de mot de passe, les fournisseurs d'utilisateurs (providers), les firewalls, le contrôle des accès.

### Firewall principal (main) - Version Web

Le firewall principal *main* est dédié à la gestion des utilisateurs de l'application web. Il utilise l'authentification par formulaire classique avec gestion de session et protection CSRF (Cross-Site Request Forgery) pour sécuriser l'accès aux routes protégées.

## Firewall API - Version Mobile

```
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    # Refresh Token firewall, use JWT for mobiles app
    refresh_token:
        pattern: ^/api/refresh_token
        stateless: true
        security: false
    # API Firewall, use JWT for mobiles app
    api:
        pattern: ^/api
        stateless: true
        provider: app_user_provider
        jwt: ~
        json_login:
            check_path: api_login_check
            username_path: email
            password_path: password
            success_handler: lexi_jwt_authentication.handler.authentication_success
            failure_handler: lexi_jwt_authentication.handler.authentication_failure
```

*security.yaml*

- **Form Login** : Les utilisateurs se connectent via un formulaire d'authentification. Le LoginAuthenticator personnalisé de Symfony est utilisé pour valider les identifiants.
- **Hashage des mots de passe** : Lors de l'inscription ou de la modification du mot de passe, Symfony utilise son système de hachage sécurisé pour protéger les mots de passe des utilisateurs avant de les enregistrer dans la base de données. En cas de compromission de la base de données, les mots de passe ne sont pas stockés en clair, ils sont cryptés par un algorithme robuste contre les attaques par force brute.

```
password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface: 'auto'
```

- **Gestion de session** : Le firewall n'est pas stateless, ce qui signifie que les sessions des utilisateurs sont conservées afin de maintenir leur état entre les requêtes.
- **Protection CSRF** : Le firewall inclut un token CSRF pour sécuriser les formulaires d'authentification contre les attaques par falsification de requêtes.
- **Système de "Remember Me"** : Permet de maintenir la session de l'utilisateur active pendant une semaine, en stockant un cookie sécurisé.

Le firewall api est utilisé pour gérer l'authentification des utilisateurs sur la version mobile de l'application via des tokens JWT (JSON Web Tokens). Cette approche stateless ne conserve pas de session côté serveur, ce qui la rend plus adaptée aux applications mobiles.

- **Authentification JWT** : Chaque requête envoyée à l'API doit inclure un token JWT pour authentifier l'utilisateur. Ces tokens sont générés lors de la connexion et validés pour chaque requête.
- **Login via JSON** : Les utilisateurs mobiles se connectent en envoyant leurs identifiants (email et mot de passe) au point d'entrée /api\_login\_check. En cas de succès, un token JWT est généré par le lexik\_jwt\_authentication.handler et renvoyé au client mobile.
- **Firewall Refresh Token** : Le firewall refresh\_token est configuré pour permettre la régénération de tokens JWT expirés. Lorsqu'un token expire, un refresh token est utilisé pour obtenir un nouveau token JWT sans avoir à redemander les identifiants de l'utilisateur.

### Gestion des rôles et contrôle d'accès

Symfony utilise un mécanisme de role hierarchy pour attribuer différents niveaux d'autorisation aux utilisateurs :

**ROLE\_USER** : Le rôle de base, attribué à tout utilisateur authentifié.

**ROLE\_ARTIST** : Les utilisateurs ayant ce rôle peuvent accéder à des fonctionnalités supplémentaires comme la soumission de nouvelles scènes.

**ROLE\_ADMIN** : Les administrateurs peuvent gérer les utilisateurs et le contenu de l'application via un tableau de bord dédié.

| ACCESS \ ROLES   | VISITOR<br>(no role) | USER | ARTIST | ADMIN |
|--|----------------------|------|--------|-------|
| Visit Gallery / Manipulate scenes<br>Read community space                                    | ✗                    | ✗    | ✗      | ✗     |
| Save scenes results<br>Like & comment on gallery<br>Write in community space<br>User section |                      | ✗    | ✗      | ✗     |
| Artist Dashboard (manage my scenes)<br>Artist community section                              |                      |      | ✗      | ✗     |
| Admin dashboard (manage users and gallery)<br>Moderate community                             |                      |      |        | ✗     |

Rôles et accès utilisateurs

```

access_control:
    - { path: ^/api/refresh_token, roles: PUBLIC_ACCESS }
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profile, roles: ROLE_USER }
    - { path: ^/artist, roles: ROLE_ARTIST }

role_hierarchy:
    ROLE_ARTIST: ROLE_USER
    ROLE_ADMIN: ROLE_ARTIST
  
```

Le fichier security.yaml définit également des règles d'accès pour les différentes routes de l'application, ainsi que les relations entre les rôles.

## Tableau de bord administrateur (admin dashboard)

Le tableau de bord administrateur est un espace sécurisé où les administrateurs peuvent gérer les utilisateurs (ajout ou suppression de rôles, modifier ou supprimer un utilisateur), mais aussi superviser le contenu de l'application :

- Modifier ou supprimer les œuvres de la galerie.
- Gérer les demandes de rôles.
- Gérer les demandes de nouvelles scènes.

Ce tableau de bord est accessible uniquement sur la version web de l'application, et son accès est protégé par le ROLE\_ADMIN.

The screenshot shows the Admin Dashboard interface. At the top, there are navigation links for 'GALLERY', 'CREATE', and 'COMMUNITY', along with user account links for 'Log out', 'Profile', and 'Admin'. Below the header, the title 'Admin Dashboard' is centered. Underneath, there are two main sections: 'Artist role requests' and 'New scene proposals', each with a table and an 'Open' button.

| User    | Date                         | Action                |
|---------|------------------------------|-----------------------|
| dada    | July 21, 2024, 5:32 pm       | <button>Open</button> |
| patrick | November 30, -0001, 12:00 am | <button>Open</button> |

| User      | Date                    | Action                |
|-----------|-------------------------|-----------------------|
| patrick   | July 26, 2024, 4:11 pm  | <button>Open</button> |
| admin     | June 11, 2024, 1:38 pm  | <button>Open</button> |
| patrick   | June 11, 2024, 12:46 pm | <button>Open</button> |
| jeanmaxxx | May 8, 2024, 6:20 pm    | <button>Open</button> |

## Accessibilité

L'accessibilité est un enjeu majeur dans le développement d'applications modernes, afin de garantir que tous les utilisateurs, y compris ceux en situation de handicap, puissent interagir avec la plateforme de manière optimale. Pour le projet NexusLab, des efforts particuliers ont été faits pour améliorer l'accessibilité, aussi bien pour la version web que pour l'application mobile.

La charte graphique a été conçue en tenant compte de l'importance des contrastes des couleurs, afin de garantir une lisibilité maximale, même pour les utilisateurs malvoyants ou souffrant de daltonisme. En respectant les bonnes pratiques en matière de contraste de texte et d'éléments visuels, l'application assure que l'information est claire et accessible à tous.

### Application web

Dans la version web de NexusLab, plusieurs pratiques ont été mises en œuvre afin de respecter les standards d'accessibilité et d'offrir une navigation fluide à tous les utilisateurs, notamment :

- **Utilisation correcte des balises HTML** : Un balisage sémantique approprié a été appliqué dans l'ensemble du site. Par exemple, les balises telles que <header>, <body>, <main>, et <footer> ont été utilisées pour structurer correctement les pages et améliorer l'accessibilité pour les lecteurs d'écran.
- **Navigation au clavier** : Le site a été conçu pour permettre une navigation complète au clavier. Chaque élément interactif est accessible via la touche Tab, garantissant que les utilisateurs qui ne peuvent pas utiliser une souris puissent tout de même accéder à l'ensemble des fonctionnalités du site, y compris l'espace de création.
- **Texte alternatif pour les images** : Tous les éléments visuels importants du site, notamment les images et les illustrations, disposent d'un texte alternatif descriptif via l'attribut alt. Cela permet aux lecteurs d'écran de décrire ces éléments pour les personnes malvoyantes ou non-voyantes.

### Application mobile

Pour l'application mobile développée avec React Native, des mesures spécifiques à ce langage ont été prises afin d'améliorer l'accessibilité :

**Utilisation d'attributs d'accessibilité sur les éléments front-end** : Chaque composant interactif de l'application mobile dispose d'attributs accessibility, garantissant que ces éléments soient lisibles et compréhensibles par les technologies d'assistance comme les lecteurs d'écran.

```
<TouchableOpacity
  onPress={() => onLabelPress(idPrefix)}
  accessibilityRole="button"
  accessibilityLabel="View art type"
  accessibilityHint="Tap to view more scenes of this type of art"
>
```

*GalleryScreen.jsx - Accessibility attributes*

## 7.2 Développement de l'application web

Cette section détaille les aspects essentiels du développement de l'application web, et explique les solutions mises en place pour surmonter les principaux défis techniques.

### Intégration des composants dynamiques React

Une fois les bundles nécessaires configurés (ux-react, Webpack Encore et Babel), l'intégration de React dans Symfony a permis de créer des interfaces utilisateur dynamiques et réactives, tout en profitant de la structure robuste offerte par Symfony pour le backend.

### Mise en pratique

Chaque composant React est encapsulé dans une vue Twig grâce à des balises spécifiques, les propriétés nécessaires lui sont passés en arguments pour qu'il puisse être rendu correctement dans le navigateur côté client.

```

<div {{ react_component('Like', {"props": {
    "user" : app.user,
    "userId": scene.user.id,
    "sceneId": scene.id,
    "likes": scene.likes.length,
    "entity": entityName,
    "isLikedByUser": app.user in scene.likes ? true : false
}} )}>
</div>

```

*gallery.html.twig*

*Ici le composant Like, permettant aux utilisateurs d'aimer une œuvre dans la galerie.*

Ensuite le composant React communique directement avec le backend via des appels AJAX pour l'asynchronisme. Dans ce cas, fetch est utilisé pour envoyer une requête HTTP POST au serveur chaque fois qu'un utilisateur clique sur le bouton "Like" ou "Unlike".

```

const handleLike = async () => {
  if (props.user){
    setIsLiked(!isLiked);

    const response = await fetch(`/like/artwork/${props.sceneId}/${props.entity}`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ user: props.userId }),
    });

    const data = await response.json();
    // Update the like count
    if (data.message === 'Like successfully added.'){
      setLikes(likes + 1);
    } else if (data.message === 'Like successfully deleted.'){
      setLikes(likes - 1);
    }
  };
}

```

*Like.jsx*

Cette requête interagit avec un controller Symfony qui met à jour la base de données et renvoie une réponse JSON pour indiquer le nombre total de likes mis à jour, ainsi que le message correspondant à l'ajout ou au retrait du like.

**LikeController :** voir l'annexe à la page 62.

### Problèmes résolus

L'utilisation de ux-react a permis de surmonter plusieurs défis liés à l'intégration de JavaScript dynamique dans un environnement Symfony :

- Gestion des états complexes : Des composants comme le bouton de "Like" nécessitent une gestion d'état dynamique en fonction des actions de l'utilisateur (ajouter ou retirer un like), ce qui est facilité par l'utilisation de React.
- Modifications en temps réel : Cette implémentation permet d'envoyer des informations au serveur et de mettre à jour instantanément des données telles que les "likes", grâce à l'utilisation d'appels AJAX.
- Réutilisabilité : React permet de créer des composants modulaires qui peuvent être réutilisés dans d'autres parties de l'application, réduisant ainsi la duplication de code et facilitant la maintenance. Le composant Like sera également utilisé dans la version mobile de l'application, après quelques ajustements mineurs de compatibilité avec React Native.

# Gestion des fichiers médias

L'upload des fichiers médias (images et code source) a été géré grâce à l'utilisation du *VichUploaderBundle*, un bundle Symfony spécialisé dans la gestion des fichiers.

Ce bundle facilite l'upload des fichiers et leur association avec des entités. Il permet également de configurer le stockage des fichiers sur le serveur et de manipuler ces fichiers directement via les entités associées.

Pour chaque fichier uploadé par un utilisateur, VichUploaderBundle assure que le fichier est stocké de manière optimisée (avec la gestion des doublons, des formats, et du chemin de stockage). Seulement le nom du fichier est stocké dans la base de données, au niveau de la table liée à l'entité correspondante. Les fichiers sont ensuite facilement récupérés et affichés dans les vues, le bundle faisant le lien entre le nom du fichier et son chemin de sauvegarde.

```
! vich_uploader.yaml x
symfony > config > packages > ! vich_uploader.yaml > {} vich_uploader > db_driver
1   vich_uploader:
2     db_driver: orm
3
4     metadata:
5       type: attribute
6
7     mappings:
8       picture_profile:
9         uri_prefix: /images/avatar
10        upload_destination: '%kernel.project_dir%/public/images/avatar'
11        namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
12        delete_on_update: false
13        delete_on_remove: false
14
15       scene1Images:
16         uri_prefix: /images/scene1Img
17         upload_destination: '%kernel.project_dir%/public/images/scene1Img'
18         namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
19         delete_on_update: true
20         delete_on_remove: true
21
22       scene2Images:
23         uri_prefix: /images/scene2Img
24         upload_destination: '%kernel.project_dir%/public/images/scene2Img'
25         namer: Vich\UploaderBundle\Naming\SmartUniqueNamer
26         delete_on_update: true
27         delete_on_remove: true
28
```

Dans cet extrait du fichier de configuration *vich\_uploader.yaml*, on retrouve les *mappings* définissant les emplacements de stockage, le renommage unique des fichiers et les règles de suppression lors des mises à jour ou suppressions d'entités.

```

```

Affichage de l'image d'une œuvre issue de la *scene1* dans une vue Twig : "scene" correspond à l'entity, et "imageFile" à son champ utilisé pour gérer le fichier.

```
#[UploadableField(mapping: "scene1Images", fileNameProperty: "imageName")]
private ?File $imageFile = null;
```

Dans l'entity *Scene1*, ce champ configure un attribut pour le téléchargement de fichiers, spécifiant le mapping à utiliser, la propriété pour stocker le nom du fichier, et initialisant une variable pour le fichier temporaire lors de l'upload.

```

#[Vich\UploadableField(mapping: "picture_profile", fileNameProperty: "imageName")]
#[Assert\File(
    maxSize: "2048M",
    maxSizeMessage: "The file is too large {{ size }} MB. Maximum file size is {{ limit }} MB.",
    mimeTypes: ["image/jpeg", "image/png", "image/gif"],
    mimeTypesMessage: "Please upload a valid image file (JPG, PNG, GIF)"
)]
private ?File $imageFile = null;

```

*Les contraintes de taille, de format et les messages d'erreur, peuvent être paramétrés directement dans l'entity ou dans le formtype Symfony gérant l'upload.*

*Ici dans l'entity User pour l'image de profil.*

L'utilisation de ce bundle a été essentielle pour simplifier le processus d'upload et de gestion des médias en général, résolvant ainsi les problèmes de stockage et d'organisation de ces fichiers. Mais malgré son efficacité, certaines situations spécifiques ont nécessité une approche adaptée.

### Cas particulier de l'enregistrement d'une œuvre

Pour la sauvegarde d'une image créée par un utilisateur lors de la manipulation d'une scène, il n'est plus question d'upload mais de capture d'image. Une conversion en base64 est alors nécessaire pour envoyer l'image depuis le sketch p5.js au controller Symfony.

```

// Capture l'image du canvas dans un format base64
const myCanvas = document.getElementById("myCanvas");
const imageBase64 = myCanvas.toDataURL();

// Créez une nouvelle image à partir de l'URL base64
const image = new Image();
image.src = imageBase64;

// Lorsque l'image est chargée, envoyez-la au serveur
image.onload = function() {
    const formData = new FormData();
    formData.append('color', color);
    formData.append('weight', weight);
    formData.append('numLine', numLine);
    formData.append('saturation', saturation);
    formData.append('opacity', opacity);
    formData.append('velocity', velocity);
    formData.append('noiseOctave', noiseOctave);
    formData.append('noiseFalloff', noiseFalloff);
    formData.append('userId', userId);
    formData.append('file', image.src);

    fetch('/generative/sendDataG1', {
        method: 'POST',
        body: formData
    })
    .then(response => {

```

*sketchSceneG1.js (p5.js) - sendData()*

Un clic sur le bouton « save » déclenche la capture de l'image et son envoi au backend via une requête fetch. Les autres données de l'œuvre sont également incluses dans la requête.

```



GenerativeSceneController.php - sendDataToSceneG1()


```

Le controller convertit ensuite l'image base64 en fichier PNG temporaire, puis crée un objet `UploadedFile` à partir de celui-ci pour le transmettre au champ `imageFile` de l'entity correspondante. `VichUploaderBundle` prend ensuite le relais pour le stockage et l'affichage de l'image.

## Gestion des données volumineuses

Dans une page qui gère un grand volume de données, comme la galerie, un affichage efficace de ces données est essentiel pour offrir une expérience fluide. Sans optimisation, charger un trop grand nombre d'œuvres simultanément peut entraîner des ralentissements, une surcharge du serveur et une mauvaise expérience utilisateur.

Pour résoudre ce problème, l'implémentation d'une pagination a joué un rôle important. Un système de tri dans le controller `Symfony` a également été intégré, afin de faciliter l'exploration de la galerie pouvant contenir un grand nombre d'images.

### Pagination avec `KnpPaginatorBundle`

Avec l'intégration de `KnpPaginatorBundle`, il est possible de diviser les œuvres en plusieurs pages, en limitant le nombre d'éléments affichés par page.

L'exemple ci-dessus montre comment le composant de pagination est utilisé pour découper la liste complète des scènes en groupes de 15 éléments par page. Le bundle gère automatiquement les détails techniques de la pagination, tels que les liens de navigation et la logique pour récupérer uniquement les œuvres nécessaires à chaque page.

```

$scenes = $paginator->paginate(
    $allScenes,
    $request->query->getInt('page', 1), /*page number*/
    15 /*limit per page*/
);

return $this->render('main/gallery.html.twig', [
    'scenes' => $scenes,
    'form' => $form->createView(),
]);

```

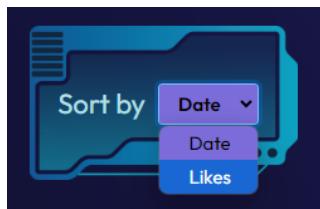
## Tri des œuvres dans le controller

En complément de la pagination, un système de tri dynamique a été implémenté pour offrir aux utilisateurs une meilleure manière d'explorer les œuvres. Deux options principales de tri sont actuellement disponibles :

- **Tri par likes** : Les œuvres les plus "liké" sont affichées en premier, ce qui permet aux utilisateurs de découvrir les œuvres les plus populaires.
- **Tri par date** : Les œuvres les plus récentes sont placées en haut de la liste, permettant aux utilisateurs de voir les dernières créations ajoutées sur la plateforme.

Un système de tri par type de scène ou par auteur est prévu dans les futures mises à jour du projet.

Lorsque une des options de tri est choisie par l'utilisateur au niveau du select de la galerie, le changement est détecté par un script JavaScript. Le formulaire est alors envoyé au controller et l'option choisie est récupérée et stockée en session pour être appliquée à la prochaine visite de la galerie.



```
document.addEventListener('DOMContentLoaded', function() {
    let sortSelect = document.querySelector('.formOptions');
    sortSelect.addEventListener('change', function() {
        let sortForm = document.querySelector('#selectForm');
        sortForm.submit();
    });
});
```

Le controller gère également les cas où le formulaire n'est pas soumis, en appliquant par défaut l'option de tri stockée en session, ou le tri par date si aucune préférence n'a été définie. Cela permet de garantir une continuité dans l'affichage, même si l'utilisateur navigue à travers plusieurs pages ou revient plus tard.

```
$form = $this->createForm(SortArtworkType::class);
$form->handleRequest($request);

// Sort artworks according to the choice of the form (date or likes)
if ($form->isSubmitted() && $form->isValid()) {

    $sort = $form->get('sortSelect')->getData();
    $session->set('sort_option', $sort);

    if ($sort == 'likes') {
        usort($allScenes, function($a, $b) {
            return ($b->getLikes()->count() <= $a->getLikes()->count());
        });
    } elseif ($sort == 'date') {
        usort($allScenes, function($a, $b) {
            return ($b->getUpdatedAt() <= $a->getUpdatedAt());
        });
    }
} else {
    // Sort artworks based on the previous sort option stored in the session, otherwise sorted by date by default
    $sortOption = $session->get('sort_option', 'date');
    if ($sortOption == 'likes') {
        usort($allScenes, function ($a, $b) {
            return ($b->getLikes()->count() <= $a->getLikes()->count());
        });
    } else {
        usort($allScenes, function ($a, $b) {
            return ($b->getUpdatedAt() <= $a->getUpdatedAt());
        });
    }
}
```

MainController.php - gallery()

Cette approche permet d'optimiser les performances et de garder une expérience utilisateur fluide, même avec une augmentation significative du nombre d'œuvres générées.

## Implémentation d'une scène générative

L'implémentation des scènes dans l'application web a été une étape majeure du projet. En plus d'intégrer la bibliothèque p5.js à Symfony, il a fallu développer des interfaces utilisateur permettant de manipuler l'algorithme pour interagir avec les scènes. Mais également permettre la sauvegarde des créations et de leurs paramètres, pour pouvoir les réutiliser ultérieurement.

### Intégration de p5.js

L'intégration de p5.js a nécessité l'ajout du script de la bibliothèque directement dans le template base.twig.html via le bloc javascripts. Les autres templates hériteront ensuite de ce fichier de base et des fonctionnalités de p5.js en utilisant la ligne `{% extends 'base.html.twig' %}`

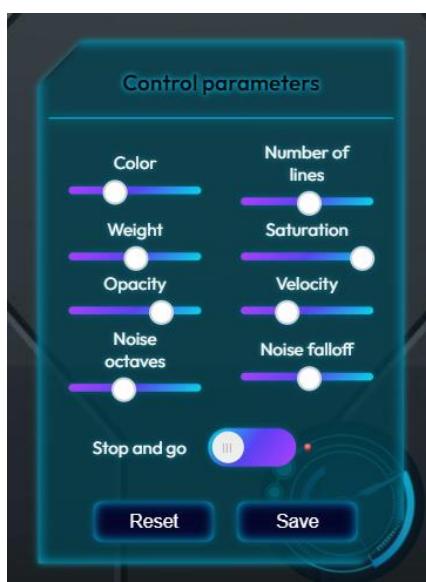
```
<script src="https://cdn.jsdelivr.net/npm/p5@1.9.2/lib/p5.js"></script>
```

Ensuite, dans le fichier Twig de la scène, le sketch p5.js correspondant est chargé à la fin du bloc body pour s'assurer que tous les éléments de la page soient disponibles avant l'exécution du script.

```
<script defer src="{{ asset('js/sketchSceneG1.js') }}"></script>
```

De cette façon le sketch peu s'exécuter correctement, il ne reste qu'à disposer le canvas p5.js dans la vue via des éléments HTML et CSS.

### Injection des données dynamiques dans les sketches p5.js



Control parameters

Pour permettre la création, les scènes générées par p5.js doivent être interactives et modulables en fonction des actions de l'utilisateur.

Pour cela, une interface utilisateur nommée le *control parameters* a été développée, réunissant les éléments HTML nécessaire à la manipulation de chaque scène (button, range-slider, select, checkbox).

Scène "Random Line Walkers"

Il a alors fallu faire le lien entre les valeurs issues de ces éléments et les variables correspondantes définies dans le sketch, pour que ces valeurs soient capturées et injectées en temps réel. Cela a été rendu possible grâce à l'utilisation de la méthode `select()` dans p5.js, qui permet de récupérer les éléments de la page et d'accéder ensuite à leurs valeurs dynamiquement.

```
<input type="range" id="colorSlider" min="0" max="360" value="5" step="10">
```



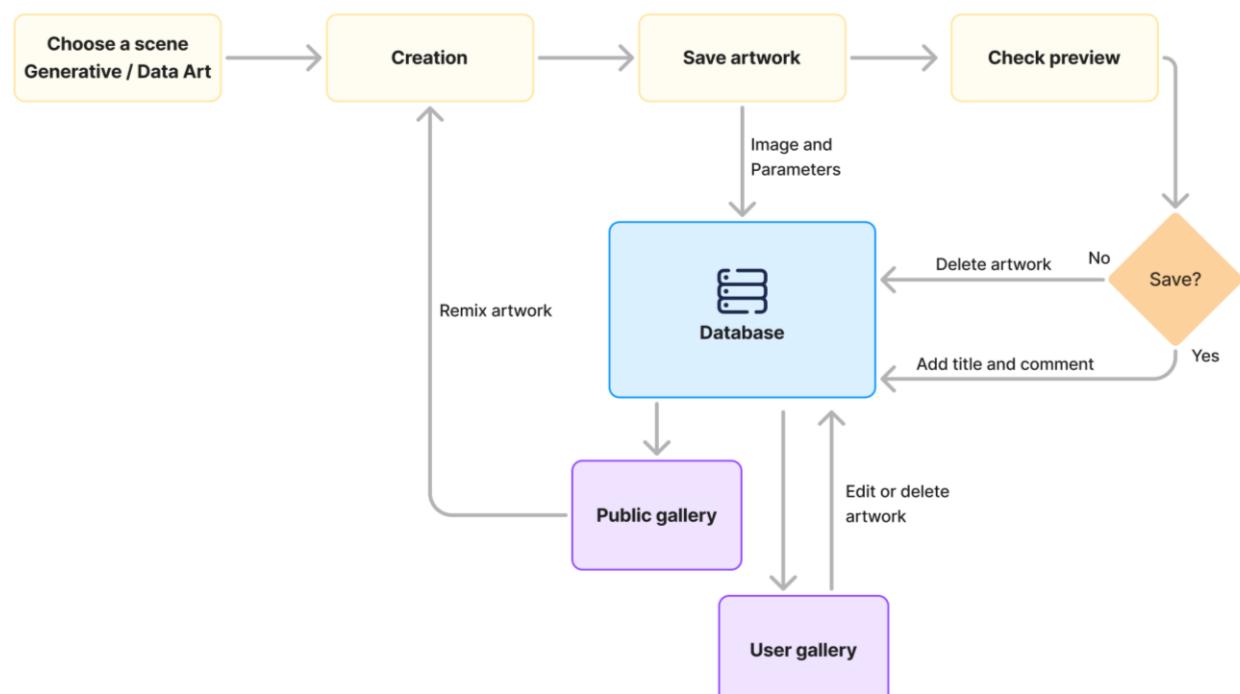
*Exemple simple d'interaction entre un élément HTML et un sketch p5.js*

## Sauvegarde et remix des créations

Une fois satisfait de son travail, l'utilisateur peut sauvegarder sa création. L'image générée est alors envoyée au serveur et les valeurs de tous les éléments du « control parameters » sont stockées en base de données.

L'utilisateur est ensuite redirigé vers une page de prévisualisation, où il peut ajouter un titre et un commentaire avant de partager son œuvre dans la galerie.

La fonctionnalité de "remix" offre la possibilité à n'importe quel utilisateur de choisir une œuvre dans la galerie, et de l'ouvrir dans sa scène correspondante avec les paramètres qui lui sont associés. Cela permet de retravailler des œuvres existantes, et d'en créer des variations dans un processus itératif et collaboratif.



*Diagramme du cycle de vie d'une œuvre*

Au moment où le template Twig d'une scène charge, des valeurs conditionnelles sont attribués aux éléments HTML du « control parameters ». Si l'objet transmis par le controller est null, les valeurs par défaut sont attribuées, dans le cas contraire ce sont les données de l'objet qui seront appliquées. Le sketch p5.js s'initialise ensuite en fonction de ces paramètres, que l'utilisateur soit en train de créer une nouvelle scène ou de modifier une œuvre existante.

Dans l'exemple ci-dessous la valeur des *range-sliders* dépend de l'état de l'objet *scene*.

```
{# Parameters sliders #}
{% set sliders = [
    {'label': 'Color', 'id': 'colorSlider', 'min': 1, 'max': 360, 'value': scene is null ? 5 : scene.color, 'step': 10},
    {'label': 'Number of lines', 'id': 'lineSlider', 'min': 1, 'max': 200, 'value': scene is null ? 100 : scene.numLine, 'step': 1},
    {'label': 'Weight', 'id': 'weightSlider', 'min': 0.2, 'max': 10, 'value': scene is null ? 5 : scene.weight, 'step': 0.1},
    {'label': 'Saturation', 'id': 'saturationSlider', 'min': 0, 'max': 100, 'value': scene is null ? 90 : scene.saturation, 'step': 5},
    {'label': 'Opacity', 'id': 'opacitySlider', 'min': 0.05, 'max': 1, 'value': scene is null ? 0.7 : scene.opacity, 'step': 0.05},
    {'label': 'Velocity', 'id': 'velocitySlider', 'min': 0, 'max': 15, 'value': scene is null ? 5 : scene.velocity, 'step': 0.1},
    {'label': 'Noise octaves', 'id': 'noiseOctaveSlider', 'min': 0, 'max': 10, 'value': scene is null ? 4 : scene.noiseOctave, 'step': 1},
    {'label': 'Noise falloff', 'id': 'noiseFalloffSlider', 'min': 0, 'max': 1, 'value': scene is null ? 0.5 : scene.noiseFalloff, 'step': 0.05}
] %}

{% for slider in sliders %}
<div class="sliderBox">
    <p>{{ slider.label }}</p>
    <div class="range-wrap">
        <input type="range" class="range" id="{{ slider.id }}" min="{{ slider.min }}" max="{{ slider.max }}" value="{{ slider.value }}" step="{{ slider.step }}>
        <output class="bubble">{{ slider.value }}</output>
    </div>
</div>
{% endfor %}
```

*sceneG1.html.twig – control parameters HTML*

## Optimisation du code en vue d'intégrer de nombreuses scènes :

Après l'intégration des scènes sur la plateforme, il a été nécessaire d'optimiser la structure du code en vue de garantir la performance et la maintenabilité à long terme de l'application. D'autant plus avec les potentielles contributions de futurs utilisateurs, pouvant amener à l'intégration de nombreuses scènes. Voici les principales stratégies adoptées pour optimiser le code :

### Utilisation de Modèles de Données

Une classe *SceneData* a été créée pour structurer efficacement les données des scènes. Ce modèle encapsule toutes les propriétés essentielles d'une scène (comme le type d'entity, le type de formulaire, le nom de la route, ou le repository associé.), permettant une gestion centralisée des informations. Cette approche favorise la clarté et facilite les modifications futures. Chaque scène peut ainsi être ajoutée sans nécessiter de changements majeurs dans le code existant.

### Mise en Place d'une Factory

L'utilisation du **design pattern Factory** à travers la classe *SceneDataFactory*, permet de créer des instances de *SceneData*. Cette factory gère la création des objets *SceneData* en fonction des paramètres fournis. En séparant la logique de création des objets de leur utilisation, le code des controllers a pu être allégé considérablement. Cette séparation permet également d'ajouter facilement de nouvelles scènes à l'avenir, simplement en mettant à jour la configuration de la factory.

```
<?php

namespace App\Model;

class SceneData
{
    private string $entityClass;
    private string $formType;
    private string $routeName;
    private string $sceneType;
    private object $repository;

    public function __construct(
        string $entityClass,
        string $formType,
        string $routeName,
        string $sceneType = 'default',
        object $repository = null)
    {
        $this->entityClass = $entityClass;
        $this->formType = $formType;
        $this->routeName = $routeName;
        $this->sceneType = $sceneType;
        $this->repository = $repository;
    }

    public function getEntityClass(): string
    {
        return $this->entityClass;
    }

    public function getFormType(): string
    {
        return $this->formType;
    }

    public function getRouteName(): string
    {
        return $this->routeName;
    }

    public function getSceneType(): string
    {
        return $this->sceneType;
    }

    public function getRepository(): object
    {
        return $this->repository;
    }

    public function setRepository(object $repository): void
    {
        $this->repository = $repository;
    }
}
```

*SceneData.php – Model*

```
namespace App\Factory;

use App\Model\SceneData;

class SceneDataFactory
{
    private array $scenesMap;

    public function __construct(array $scenesMap)
    {
        // Tableau de correspondance entre les entity et leurs propriétés
        // (injection via le service)
        $this->scenesMap = $scenesMap;
    }

    public function createSceneData(string $entityClass): ?SceneData
    {
        if (!isset($this->scenesMap[$entityClass])) {
            return null;
        }

        // Récupérer les détails de la scène et créer l'objet SceneData
        $repoData = $this->scenesMap[$entityClass];

        return new SceneData(
            $repoData['entityClass'],
            $repoData['formType'],
            $repoData['routeName'],
            $repoData['sceneType'],
            $repoData['repository']
        );
    }
}
```

*SceneDataFactory.php - Factory*

## Centralisation des Données

Un tableau de correspondance (*scenesMap*) est utilisé dans *SceneDataFactory* pour associer chaque type de scène à ses propriétés respectives. Il est défini dans *services.yaml*, un fichier de configuration qui définit comment les services de l'application Symfony sont instanciés et configurés. Il permet la centralisation de la configuration des différents composants, et l'accès à des paramètres globaux qui peuvent être utilisés dans toute l'application, ici le tableau *sceneMap*.

```
App\Factory\SceneDataFactory:
    arguments:
        $scenesMap:
            SceneD1:
                entityClass: App\Entity\SceneD1
                formType: App\Form\SaveArtworkD1Type
                routeName: sceneD1
                sceneType: data_scene
                repository: '@App\Repository\SceneD1Repository'
            SceneD2:
                entityClass: App\Entity\SceneD2
                formType: App\Form\SaveArtworkD2Type
                routeName: sceneD2
                sceneType: data_scene
                repository: '@App\Repository\SceneD2Repository'
            Scene1:
```

*services.yaml - SceneDataFactory est configuré comme un service, et on lui passe le tableau sceneMap comme argument.*

*services.yaml* permet également d'injecter facilement des dépendances dans un service, ce qui favorise une architecture propre et maintenable.

En définissant ce mapping de manière dynamique, la duplication de code a été clairement réduite et la clarté des relations entre les différentes entités améliorée.

Un controller commun et parent à toutes les scènes intègre dans son constructeur la dépendance *SceneDataFactory*, ce qui lui permet d'accéder à toutes les scènes de l'application. Ce controller utilise une route dynamique pour gérer et afficher les scènes, qu'elles aient ou non des paramètres initiaux {id?}. Grâce à cette architecture, le controller peut centraliser la logique de rendu des différentes scènes en utilisant la factory pour récupérer l'objet de la scène voulue et ses propriétés.

```
abstract class BaseSceneController extends AbstractController
{
    protected EntityManagerInterface $entityManager;
    protected Security $security;
    private SceneDataFactory $sceneDataFactory;

    public function __construct(SceneDataFactory $sceneDataFactory, Security $security, EntityManagerInterface $entityManager)
    {
        $this->sceneDataFactory = $sceneDataFactory;
        $this->entityManager = $entityManager;
        $this->security = $security;
    }

#[Route("scene/{entity}/{id?}", name: "getScene")]
    public function getScene(string $entity, $id = null): Response
    {
        $sceneDataObj = $this->sceneDataFactory->createSceneData($entity);
        if (!$sceneDataObj) {
            throw $this->createNotFoundException('Scene type not found.');
        }
        $scene = $id !== null ? $sceneDataObj->getRepository()->find($id) : null;

        return $this->render("${sceneDataObj->getSceneType()}/{$sceneDataObj->getRouteName()}.html.twig", [
            'scene' => $scene,
        ]);
    }
}
```

*BaseSceneController.php – getScene()*

## 7.3 Développement de l'application mobile

Le développement de l'application mobile a été entrepris après celui de la version web, il a donc fallu transposer les fonctionnalités de l'application web au mobile.

Cette section présentera les principales étapes du développement en React Native, les méthodes mises en œuvre pour assurer l'intégration des fonctionnalités, et les mesures de sécurité adoptées pour protéger cette version de l'application.

### Structure de la navigation

La navigation dans l'application mobile a été conçue pour offrir une expérience utilisateur intuitive et fluide adaptée spécialement au support mobile. Pour cela, l'utilisation du package **React Navigation** a permis de gérer la navigation entre les différentes vues.

La structure de navigation repose sur plusieurs niveaux, elle se compose de plusieurs navigateurs, notamment :

# Développement application mobile

- **Stack Navigator (@react-navigation/native-stack)** : Utilisé pour gérer la navigation entre les écrans empilés, permettant aux utilisateurs de revenir à l'écran précédent avec des gestes ou des boutons de retour.  
Son option header permet d'afficher une icône en haut de toutes les vues du TabNavigator pour accéder au profil utilisateur.
- **Tab Navigator (@react-navigation/bottom-tabs)** : Permet de naviguer entre les sections principales de l'application via des onglets situés en bas de l'écran, offrant ainsi un accès rapide aux fonctionnalités clés telles que la galerie, la création de scènes.

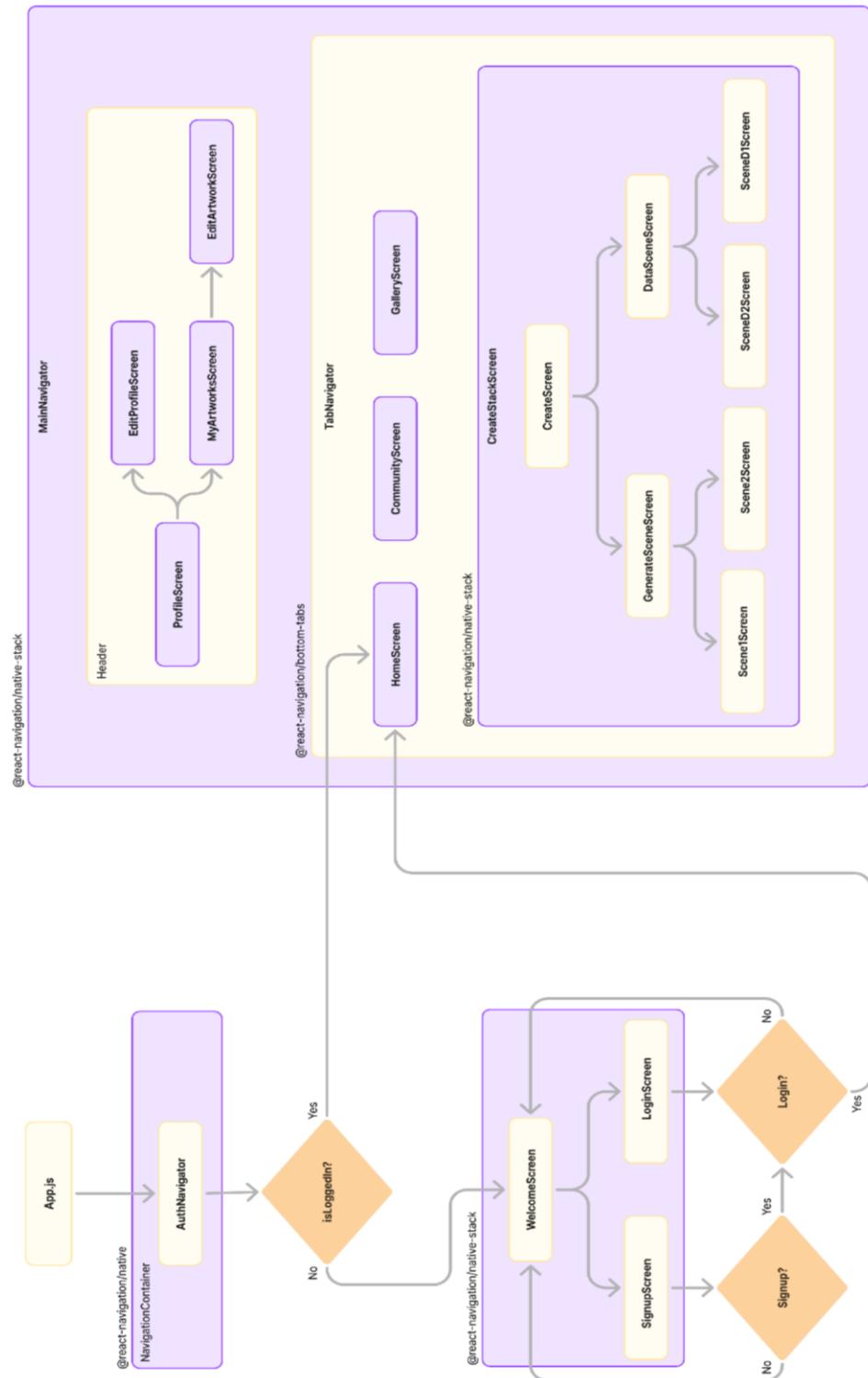


Diagramme de navigation – NexusLab Mobile

La navigation a été pensée pour garantir un accès rapide et intuitif aux différentes sections de l'application, tout en conservant une cohérence avec l'application web.

La séparation des espaces de navigation entre l'authentification et le reste de l'application permet d'inclure une première couche de sécurité. Cette fonction attribuée à l'*AuthNavigator* empêchera un utilisateur non authentifié d'entrer sur l'application.

*isLoggedIn* est un state global issu d'un *AuthProvider* (*useContext*), son état dépend de la vérification du token de connexion (JWT) attribué à l'utilisateur authentifié.

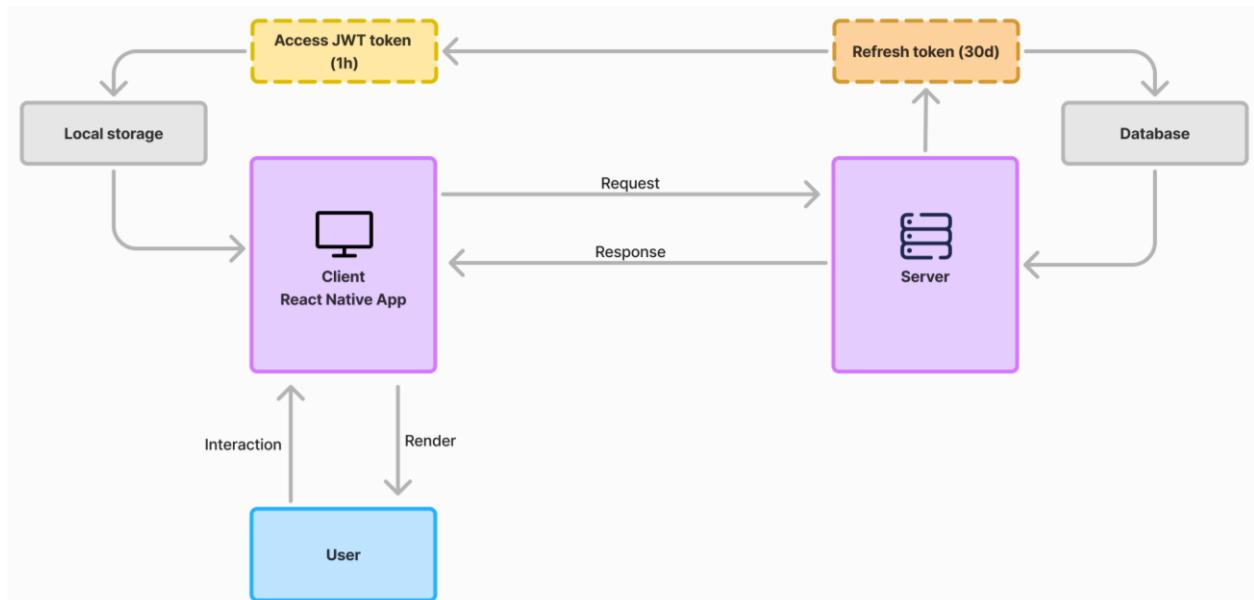
## Sécurité globale de l'application : client et serveur

### Authentification

La sécurité a été une préoccupation majeure lors du développement de l'application mobile. Pour garantir la protection des sessions utilisateurs, la mise en place d'une authentification basée sur des JWT access tokens s'est présentée comme une solution logique. En effet, le bundle *LexikJWTAuthentication* de Symfony facilite l'implémentation de ce système d'authentification au niveau du backend.

Lors de la connexion, un JWT token est généré et renvoyé au client, puis stocké localement sur l'appareil. Ce token a une durée de validité limité (1 heure), et c'est sa vérification lors des envois de requêtes ou lors des accès à certaines pages qui permet d'assurer que c'est bien l'utilisateur authentifié qui interagit avec l'application.

Pour gérer l'expiration des tokens, un système de refresh token a été mis en place. Il permet de renouveler le token d'accès sans nécessiter une nouvelle connexion, offrant ainsi une expérience utilisateur fluide et sans déconnexions.

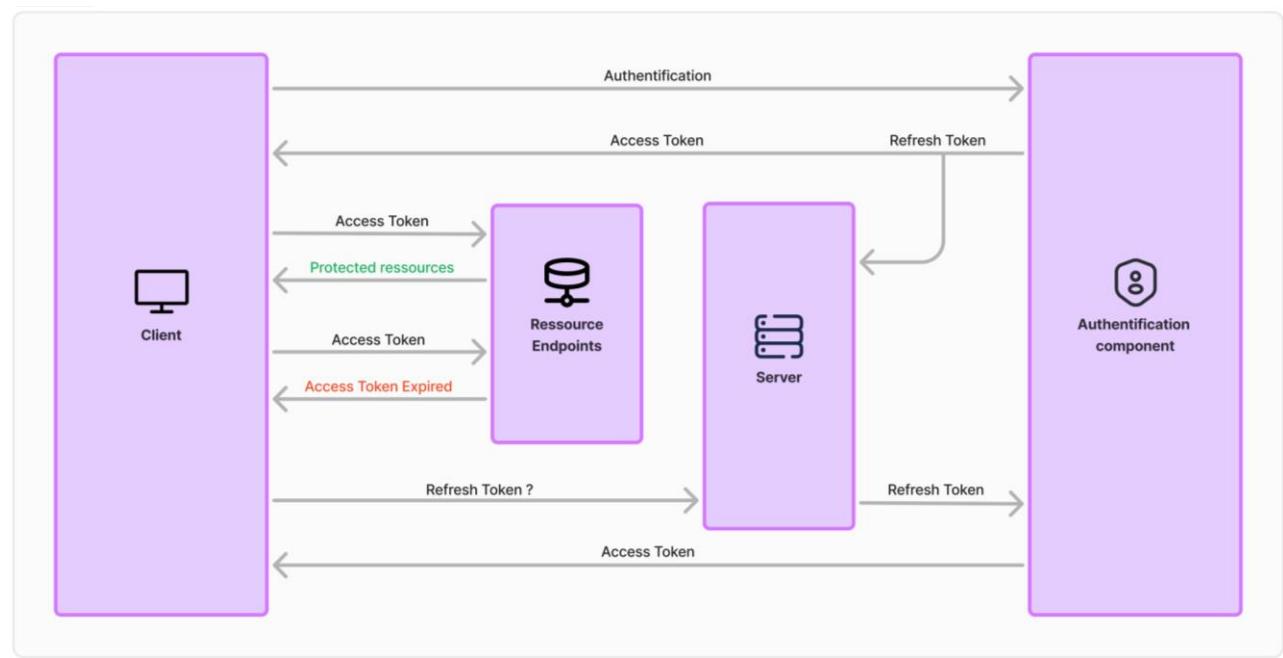


Fonctionnement global du système d'authentification : JWT + Refresh token

## Système de refresh token

Pour pallier au problème de validité des JWT tokens, il a été nécessaire d'implémenter un système permettant de renouveler ces token en toute sécurité, et ce sans dégrader l'expérience utilisateur. Ce système permet de générer un refresh token en plus du token d'accès au moment de la première connexion. Ce nouveau token, d'une durée de validité bien plus longue que le token d'accès (30 jours), est uniquement stocké côté serveur pour un maximum de protection et de sécurité. C'est lui qui va permettre d'identifier l'utilisateur au moment du renouvellement du JWT token.

Une logique assez complexe a dû être implémentée pour mettre en place ce système, tant dans la partie React Native que dans la partie backend Symfony. Des cas particuliers ont dû être pris en compte au fil des tests et du développement, permettant au code d'être ajusté en conséquence.



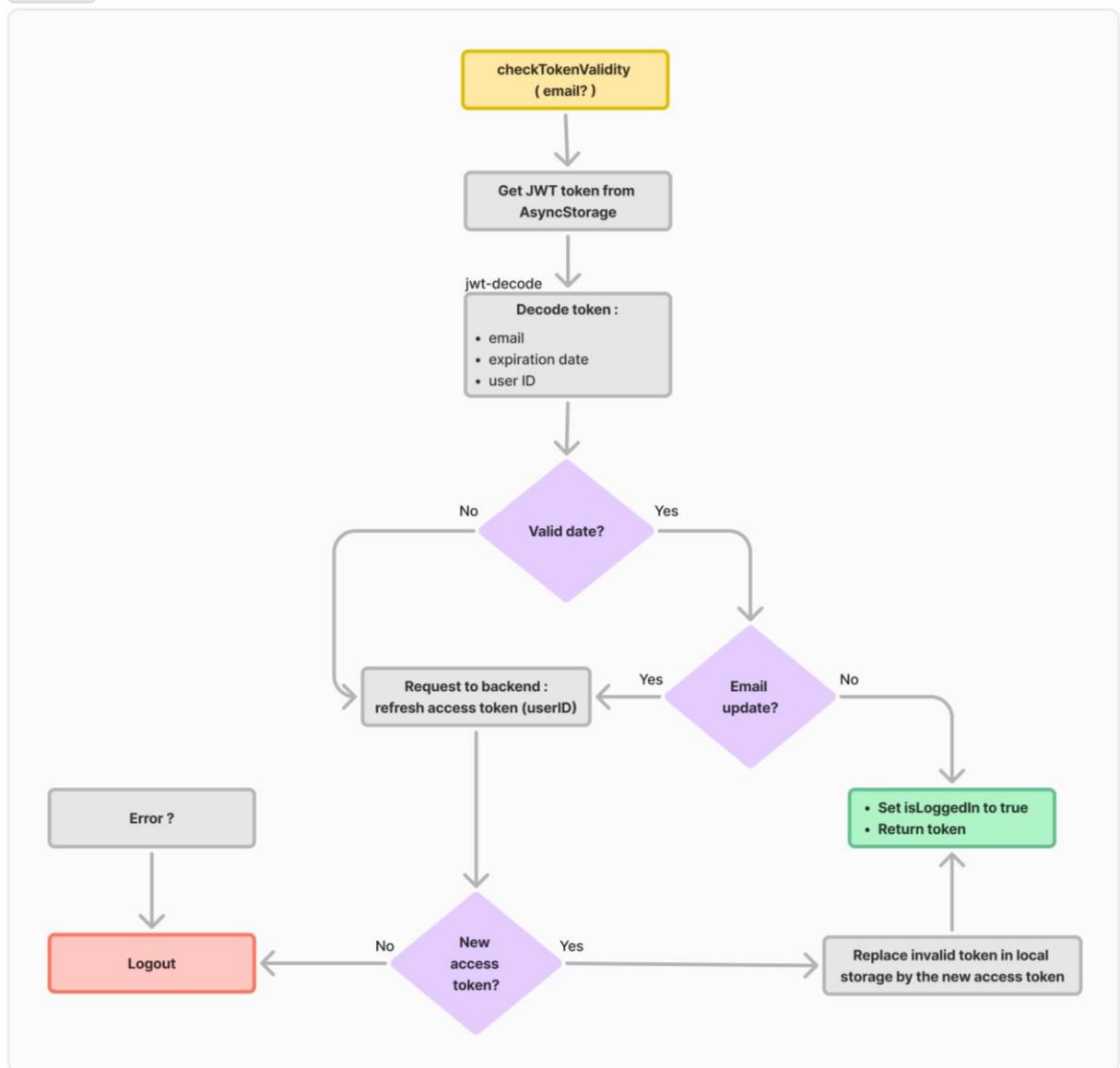
Parcours d'authentification – système de refresh token

## Côté client

Un service dédié (*AuthService*) a été implémenté côté React Native pour vérifier la validité du token d'accès et le mettre à jour si nécessaire.

Si le token n'est pas valide une requête est envoyée au serveur pour demander le renouvellement du token. C'est dans le controller dédié (*api\_TokenController*) qu'est traitée cette requête et qu'est vérifiée l'identité de l'utilisateur.

Le refresh token stocké en base de données est lié à un utilisateur par son identifiant de connexion (email). Il a donc fallu gérer le cas où l'utilisateur update son profil et change son email, car le lien avec le refresh token serait rompu.



*AuthService.js - checkTokenValidity()*

Cette fonction est nécessaire à la sécurité globale de l'application, on la retrouve :

- Dans le fichier d'entrée `App.js`, pour gérer la vérification du token d'accès lors de l'ouverture de l'application.
- Lors de l'édition du profil utilisateur si l'adresse email a été modifiée, pour gérer ce cas particulier en mettant à jour les tokens.
- Dans le système d'api sécurisé, pour le cas où une requête échoue à cause d'un token d'accès non valide.

## Côté serveur

C'est au niveau du backend que s'opèrent la génération et le renouvellement des tokens. Voici les deux principaux composants répondant à ces besoins :

### ● RefreshTokenListener :

Il a fallu en premier lieu développer un `EventListener` pour générer le refresh token, en plus de l'access token, lorsqu'une authentification réussie.

Lorsqu'un utilisateur s'authentifie, l'écouteur vérifie en base de données s'il existe déjà un refresh token valide pour cet utilisateur. Si c'est le cas, il utilise ce token ; sinon, il en génère un nouveau et le sauvegarde dans la base de données.

Ce code met également à jour les données de l'événement avec le refresh token et sa date d'expiration. Des méthodes privées gèrent la génération, la sauvegarde des tokens et la récupération de leur date d'expiration.

- **api\_TokenController :**

Ce controller reçoit les requêtes de lAuthService vu précédemment, il gère la génération d'un nouveau JWT access token à partir du refresh token :

```
class api_TokenController extends AbstractController
{
    private $jwtManager;
    private $manager;

    public function __construct(JWTTokenManagerInterface $jwtManager, EntityManagerInterface $manager)
    {
        $this->jwtManager = $jwtManager;
        $this->manager = $manager;
    }

    #[Route('/api/refresh_token', name: 'api_refresh_token', methods: ['POST'])]
    public function api_refreshToken(Request $request): JsonResponse
    {
        $data = json_decode($request->getContent(), true);
        $userId = $data['userId'] ?? null;
        if (! $userId) {
            return new JsonResponse(['error' => 'Username is missing'], 400);
        }

        $user = $this->manager->getRepository(User::class)->findOneBy([
            'id' => $userId,
        ]);
        if (! $user instanceof User){
            throw new AccessDeniedException('Invalid User');
        }
        $existingRefreshToken = $this->manager->getRepository(RefreshToken::class)->findOneBy([
            'username' => $user->getEmail(),
        ]);
        if (! $existingRefreshToken) {
            return new JsonResponse(['error' => 'Refresh token not found'], 403);
        }
        if ($existingRefreshToken && $existingRefreshToken->isValid()) {
            $accessToken = $this->jwtManager->create($user);
            return new JsonResponse(['token' => $accessToken]);
        } else {
            throw new AccessDeniedException('Invalid Token');
        }
    }
}
```

Extraction de l'identifiant utilisateur du contenu de la requête.

Recherche de l'utilisateur correspondant dans la base de données.

Recherche du refresh token associé à l'utilisateur.

Si un refresh token valide est trouvé, un nouveau access token est généré et renvoyé au client. Sinon, une exception est levée.

Ainsi ce système de refresh token, couplé à l'authentification JWT, permet de renforcer la sécurité des sessions utilisateurs tout en assurant une continuité d'usage, offrant ainsi une expérience optimisée sans interruptions intempestives.

### Appel api sécurisé

Dans l'objectif de protéger la communication avec le serveur, un hook personnalisé *useApi* a été créé pour gérer les requête API dans l'application. Il utilise Axios pour effectuer des appels API et inclut des interceptors pour gérer les en-têtes d'autorisation et les erreurs de réponse.

- **Intercepteur de requête :** Avant d'envoyer une requête, il ajoute un jeton d'authentification (*Bearer token*) aux en-têtes. Dans *useApi* c'est le JWT access token qui est transmis en en-tête des

requêtes, côté serveur le *Lexik JWT Bundle* de Symfony se charge de vérifier la validité de ce token avant d'accepter les requêtes.

- **Intercepteur de réponse** : En cas d'erreur 401 (non autorisé), il tente de rafraîchir l'access token en appelant la fonction *checkTokenValidity*. Si le rafraîchissement réussit, il renvoie la requête originale avec le nouveau token. Si le rafraîchissement échoue, il déconnecter l'utilisateur par mesure de sécurité.

## Intégration des scènes p5.js dans React Native

Le principal obstacle rencontré lors de la transposition des scènes sur React Native est la compatibilité de l'environnement, en effet p5js est conçu pour fonctionner dans un environnement de navigateur web et dépend fortement du DOM. Mais React Native n'utilise pas le DOM traditionnel, ce qui cause des problèmes de compatibilité.

### Utilisation de Webview

Le composants *WebView* est une solution cross-platforme permettant d'afficher des contenus HTML et JavaScript à l'intérieur d'une application mobile.

```
{htmlContent && (
  <WebView
    ref={webViewRef}
    originWhitelist={[ '*' ]}
    source={{ uri: htmlContent }}
    javaScriptEnabled={true}
    domStorageEnabled={true}
    style={styles.webview}
    onMessage={handleWebViewMessage}
    onLoadStart={() => setInitialLoading(true)}
    onLoadEnd={() => setInitialLoading(false)}
  />
)}
```

*Scene1Screen.jsx - WebView*

Pour alimenter cette *WebView* il faut lui fournir un fichier HTML unique. Il a donc fallu condenser le code nécessaire au fonctionnement de chaque scène dans un seul fichier, incluant le CSS, le code JavaScript, et la structure HTML.

Ce fichier doit être chargé avant l'affichage de la *WebView*, une gestion cohérente de l'asynchronisme est donc très importante. Un *useEffect* récupère donc ce fichier au chargement du screen, et attend qu'il soit complètement chargé avant de poursuivre l'exécution du code.

```
useEffect(() => {
  async function loadHtmlFile() {
    const htmlAsset = Asset.fromModule(require(' ../../assets/webView/SceneG1.html'));
    await htmlAsset.downloadAsync();
    setHtmlContent(htmlAsset.uri);
    setInitialLoading(false);
  }
  loadHtmlFile();
}, []);
```

*Scene1Screen.jsx - Chargement du fichier source de la WebView*

La communication entre la *WebView* et le code du screen React Native qui l'implémente est possible par le biais de la fonction *postMessage*. Elle permet, dans le fichier contenant le sketch p5.js, d'envoyer un message au moment de la sauvegarde de l'œuvre générée. Ici les paramètres de l'œuvre et son image :

```
function sendDataToReactNative() {
  const myCanvas = document.getElementById("myCanvas");
  const imageBase64 = myCanvas.toDataURL('image/png');

  const data = {
    color: colorSlider.value(),
    numLine: lineSlider.value(),
    weight: weightSlider.value(),
    saturation: saturationSlider.value(),
    opacity: opacitySlider.value(),
    velocity: velocitySlider.value(),
    noiseOctave: noiseOctaveSlider.value(),
    noiseFalloff: noiseFalloffSlider.value(),
    file: imageBase64
  };
  window.ReactNativeWebView.postMessage(JSON.stringify(data));
}
```

*SceneG1.html – Fichier source de la WebView*

Ce message est intercepté grâce à l'option *onMessage* de la *WebView*. Il est ensuite transmis à une fonction de rappel (callback) qui va traiter les données reçues dans l'événement. Cette fonction extrait les informations de *event.nativeEvent.data*, les envoie au backend, et déclenche le processus de sauvegarde de l'œuvre (ajout du titre et du commentaire).

```
const handleWebViewMessage = (event) => {
  const data = JSON.parse(event.nativeEvent.data);
  sendDataToBackend(data);
};

const sendDataToBackend = async (data) => {
  try {
    setSendingDataLoading(true);
    const token = await AsyncStorage.getItem('token');
    const decodedToken = jwtDecode(token);
    const userId = decodedToken.id;
    const fullData = {
      ...data,
      userId: userId
    };
    const response = await api.post('/generative/sendDataG1', fullData, {
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (response.status !== 200) {
      throw new Error("Erreur lors de l'envoi des données à l'API");
    }

    const result = await response.data;
    setCurrentSceneId(result.sceneId);
    setModalVisible(true);
    setSendingDataLoading(false);
  } catch (error) {
    console.error("Erreur lors de l'envoi des données:", error);
    setSendingDataLoading(false);
  }
};
```

Interception du message et conversion du format JSON en objet JavaScript.

L'ID de l'utilisateur connecté est récupérée en décodant l'access token, et est ajoutée aux données de la WebView.

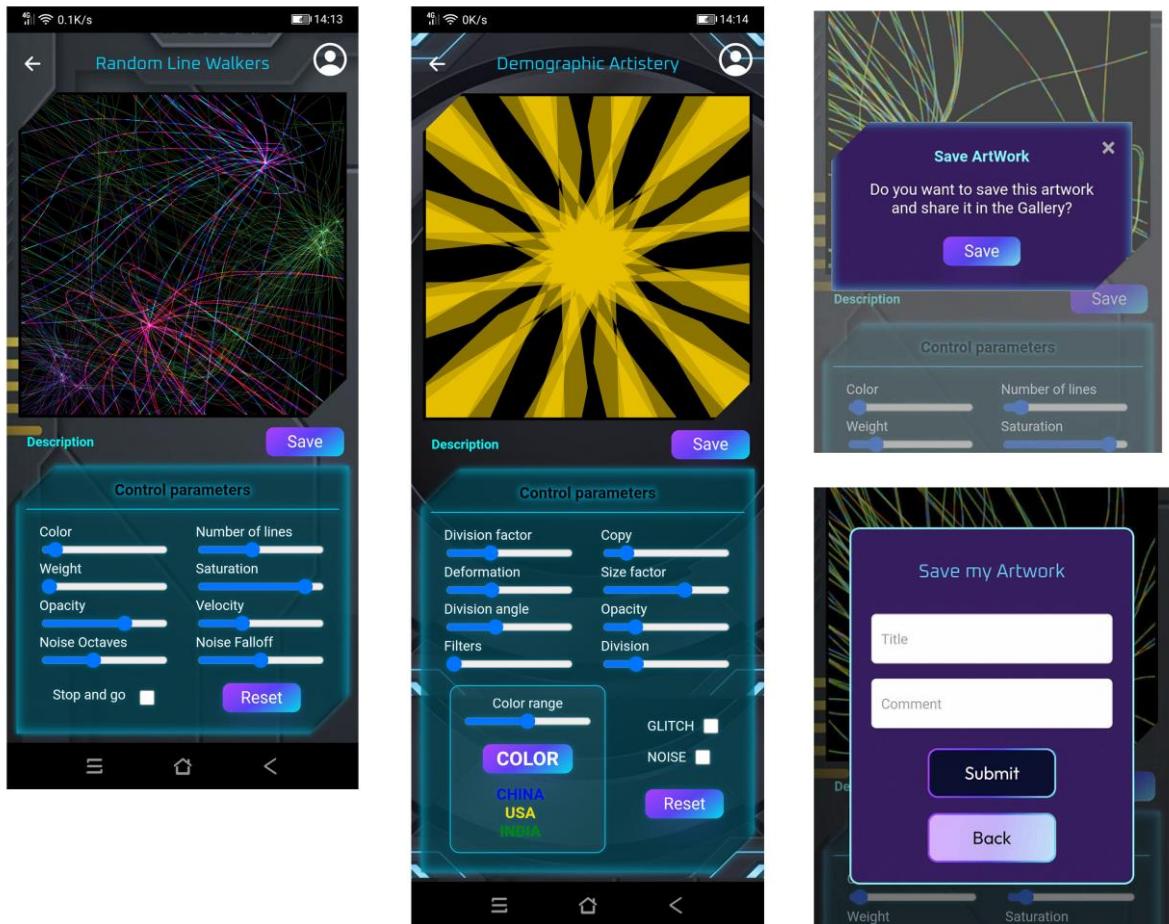
Une requête est envoyée à l'API Symfony, un controller se chargera du traitement et du stockage des données.

L'ID de l'œuvre est récupéré dans la réponse et stocké dans un state. L'ouverture de la modale d'enregistrement est déclenchée, l'ID servira pour l'update du title et du comment, ainsi qu'à la suppression en cas d'annulation.

La gestion du state *SendingDataLoading* permet l'affichage d'un écran de chargement le temps de l'exécution de la fonction.

*Scene1Screen.jsx – Récupération et envoi des données de la WebView au backend*

Les données sont finalement traitées par `api_GenerativeSceneController` qui est sensiblement similaire au `GenerativeSceneController` de l'application web. Ce dernier récupère les données à partir d'une requête HTTP sous forme de paires clé-valeur (key-value), tandis que l'API controller récupère les données contenues dans la requête en format JSON.



Screenshot app mobile – Scènes Generative & Data Art, modales de sauvegarde

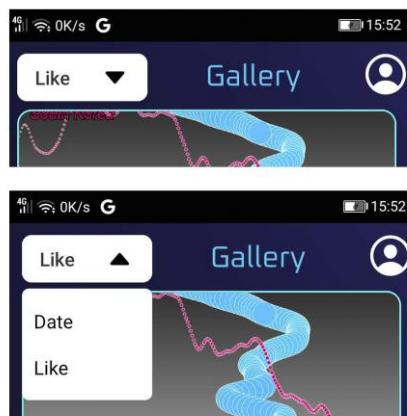
Dans cet exemple tous les éléments du screen sont générés par la WebView sauf l'image de fond et les éléments du header (titre, icônes retour et profil).

La première modale de sauvegarde est également générée par la WebView, mais la modale finale est affichée via React Native.

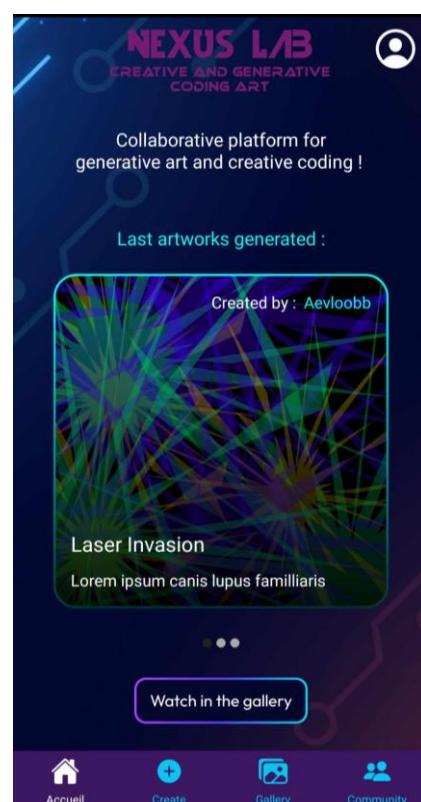
## Utilisation de bibliothèques et composants personnalisés

Au cours du développement de l'application mobile, plusieurs composants natifs de React Native ont été utilisés pour tirer parti de ses fonctionnalités intégrées. Cependant, il a également été nécessaire de recourir à des bibliothèques externes pour répondre à des besoins spécifiques en matière d'interface utilisateur. Par exemple, la bibliothèque `react-native-paper` a été employée pour gérer des composants comme les `checkbox`, offrant une solution fiable et déjà stylisée pour certains éléments courants.

Toutefois, lorsque des composants nécessitent un contrôle plus poussé sur leur apparence et leur comportement, la création de composants personnalisés s'avère indispensable.



Un exemple notable est le select (liste déroulante) pour classer les œuvres dans la galerie. Plusieurs dépendances externes ont été testées sans en trouver une de réellement satisfaisante. La création d'un composant personnalisé a apporté une plus grande liberté de stylisation, ce qui a permis de mieux l'intégrer dans le header du screen.



Dans cet exemple, le carrousel de la page d'accueil est un composant entièrement personnalisé. Il permet de faire défiler les trois dernières œuvres qui ont été générées sur la plateforme. À chaque visite de l'utilisateur sur la page d'accueil, une requête est envoyée au serveur pour récupérer les trois dernières générations (toutes scènes confondues). Le composant est ensuite réactualisé.

Le développement de l'application mobile a représenté une étape importante pour étendre l'accessibilité et l'interaction des utilisateurs avec la plateforme NexusLab. Malgré les défis techniques, l'intégration des principales fonctionnalités s'est révélée être un succès, offrant une expérience utilisateur fluide, cohérente et sécurisée. L'application mobile reprend l'esthétique et l'ergonomie de la version web, permettant aux utilisateurs de conserver leurs habitudes et d'accéder au même contenu, quel que soit le support utilisé. Cette extension mobile renforce ainsi la présence de NexusLab et offre une continuité d'utilisation entre web et mobile.

# 8. CONCLUSION ET PERSPECTIVES

## 8.1 Bilan du travail réalisé

### Résultat obtenu

Le projet NexusLab a permis d'atteindre ses principaux objectifs, tant sur le plan fonctionnel que technique. Ce premier prototype d'application, dans ses versions web et mobile, a rempli la majorité des objectifs fixés par le cahier des charges initial. Les fonctionnalités majeures ont été implémentées avec succès, le résultat répond globalement aux attentes initiales du cahier des charges.

### Fonctionnalités à venir et perspective d'évolution

Bien que ce projet soit un prototype fonctionnel, il reste encore des fonctionnalités à développer pour que NexusLab atteigne pleinement son ambition de plateforme créative communautaire. En plus de l'optimisation du code et de l'ajout des fonctionnalités restantes, il sera important de planifier les tests (unitaires, d'intégration et de performance), ainsi que la mise en production et le déploiement. Des améliorations au niveau de la sécurité, de la stabilité, des performances, et de l'intégration avec différents systèmes, seront également indispensables pour garantir une application évolutive et fiable.

Voici une liste non exhaustive des fonctionnalités restantes à implémenter :

| Fonction  | À implémenter |        |
|---|---------------|--------|
|   | Web           | Mobile |
| Espace de discussion communautaire type forum (page community)  | X             | X      |
| Système de follow pour suivre un Artist et être informé de ses news   | X             | X      |
| Système de commentaires sur les œuvres  | X             | X      |
| Système de notification pour être informé des interactions en lien avec l'user ou des news des membres suivis | X             | X      |
| Système de remix d'œuvre  |               | X      |
| Tri de la galerie par Artist ou par scène   | X             | X      |
| Collectif Drawing   | X             | X      |
| Live Coding   | X             |        |
| Artist Dashboard et options liés au rôle  |               | X      |
| Système de gamification (quêtes, niveaux, concours)   | X             | X      |
| Messagerie privée intégrée pour faciliter l'échange entre les membres   | X             | X      |

## Veille technologique et optimisations

Tout au long du développement, une veille technologique a été menée pour rester informé des meilleures pratiques et des innovations pertinentes pour le projet. Cette démarche a permis de surmonter les défis techniques, mais également de suivre l'évolution de nouvelles méthodes afin d'optimiser les performances et de renforcer la sécurité des applications.

La veille s'est révélée essentielle pour adapter les solutions choisies en fonction de leur compatibilité et pour anticiper les futures évolutions de la plateforme.

L'ajout de nouvelles fonctionnalités a été accompagné d'une analyse de l'existant et des ressources disponibles sur internet, afin de garantir une implémentation optimale. Ces recherches ont permis de valider les choix technologiques effectués, et ont facilité leur intégration dans le projet.

Cette démarche continue permettra d'ajuster la plateforme en fonction des retours et des besoins des utilisateurs, tout en s'ouvrant aux technologies émergentes susceptibles d'améliorer le projet. En effet, l'application devra demeurer flexible pour intégrer facilement de nouvelles fonctionnalités, en particulier celles liées à l'intégration de scènes génératives écrites dans des langages spécifiques.

Cette approche permettra à NexusLab de rester à la pointe de la technologie et d'offrir une plateforme en constante évolution, capable de répondre aux besoins changeants des utilisateurs.

## 8.2 Réflexion personnelle

Le développement de NexusLab a été pour moi une expérience enrichissante, tant sur le plan technique que personnel. Passionné par l'art et la programmation, ce projet m'a offert l'opportunité de pouvoir mettre en valeur ces deux univers, révélant les possibilités incroyables issues de leur fusion.

J'ai ainsi pu approfondir mes connaissances sur les technologies abordées lors de ma formation telles que React, Symfony, et React Native, mais aussi me perfectionner dans la pratique du creative coding à travers la création des scènes en p5.js.

Ce projet m'a permis d'acquérir des compétences précieuses en matière d'architecture de projets complexes, intégrant à la fois des applications web et mobile. J'ai également renforcé ma compréhension des API, des bonnes pratiques de sécurité, et de la structuration de projets web et mobile en général.

Enfin, j'ai pris conscience de l'ampleur que peut représenter la conception et le développement d'une plateforme web complète. La rigueur dans la méthodologie, la gestion du cycle de vie du projet, et la capacité à anticiper les besoins futurs sont des éléments essentiels pour garantir la maintenabilité et la fiabilité à long terme de tels projets.

Cette expérience a permis de renforcer mes compétences dans ces domaines, tout en me préparant à mieux appréhender les défis plus complexes que je serais amené à rencontrer dans des projets futurs.

## 9. ANNEXES

### 9.1 Bibliographie et références techniques

Documentation :

- **React** : <https://react.dev/>
- **React Native** : <https://reactnative.dev/docs/getting-started>
- **Symfony** : <https://symfony.com/doc/current/index.html>
- **p5.js** : <https://p5js.org/>
- **JSON Web Tokens** : <https://jwt.io/>
- **React Navigation** : <https://reactnavigation.org/docs/getting-started/>

Guides et tutoriels :

- **Grafikart** : <https://grafikart.fr/>
- **The Coding Train** : <https://www.youtube.com/@TheCodingTrain>
- **Patt Vira** : <https://www.youtube.com/@pattvira>
- **Nouvelle Techno** : <https://www.youtube.com/@NouvelleTechno>

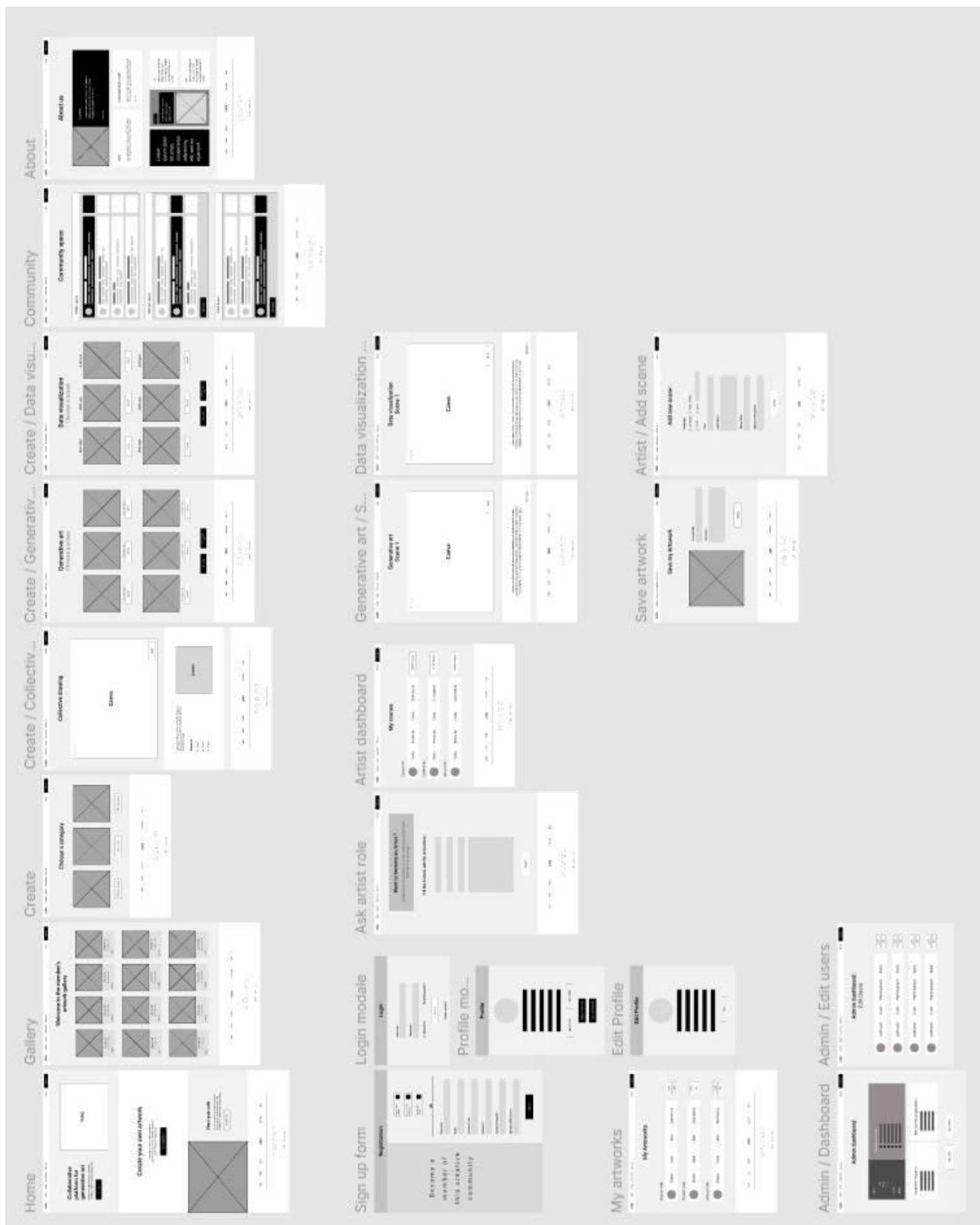
Bibliographie :

- **Software craft: TDD, Clean Code et autres pratiques essentielles :**  
de Arnaud Thiéfaine, Houssam Fakih, Dorra Bartagui, Fabien Hiegel, Cyrille Martraire

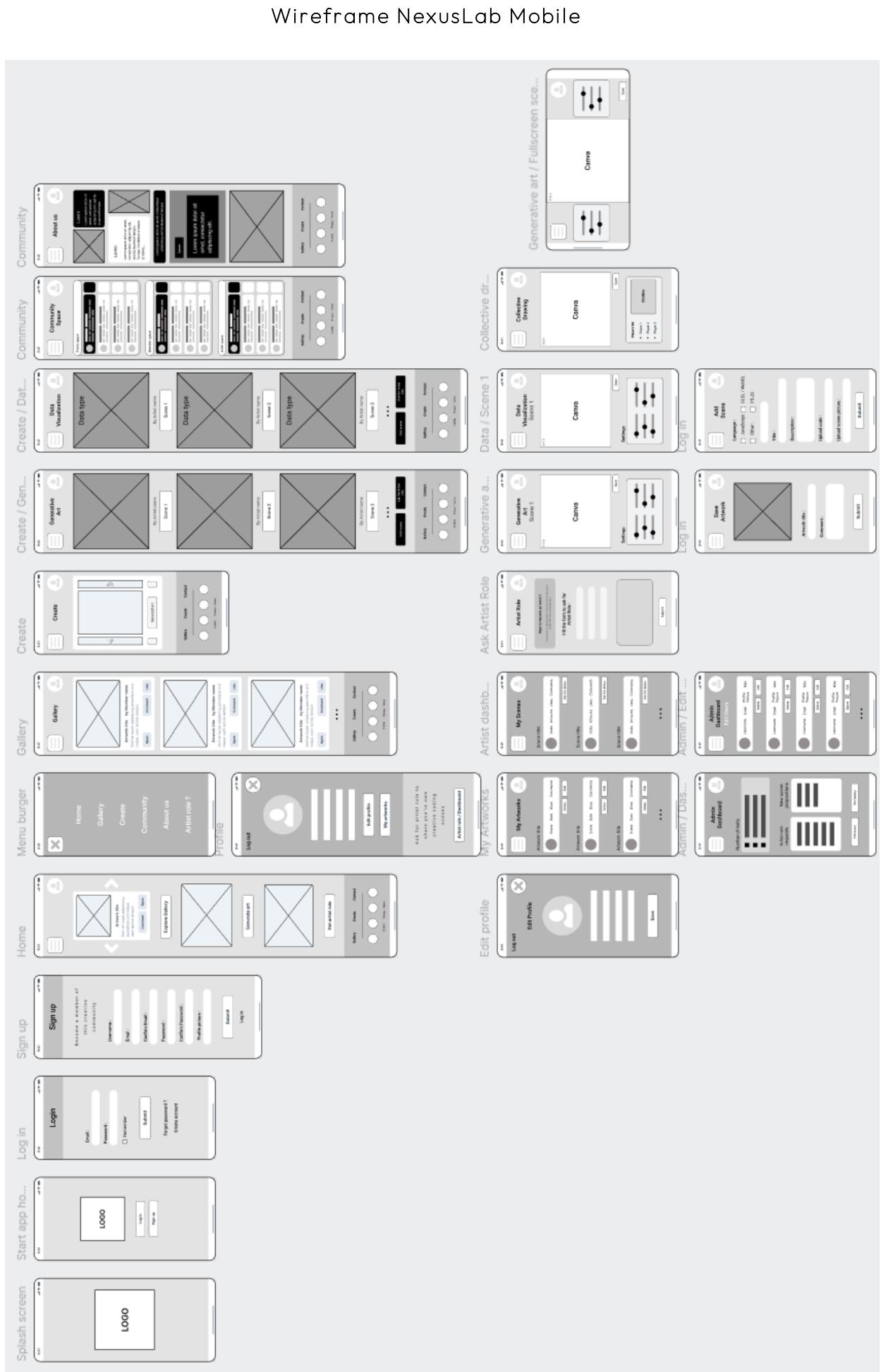
# Annexes

## 9.2 Documents supplémentaires

Wireframe NexusLab Desktop

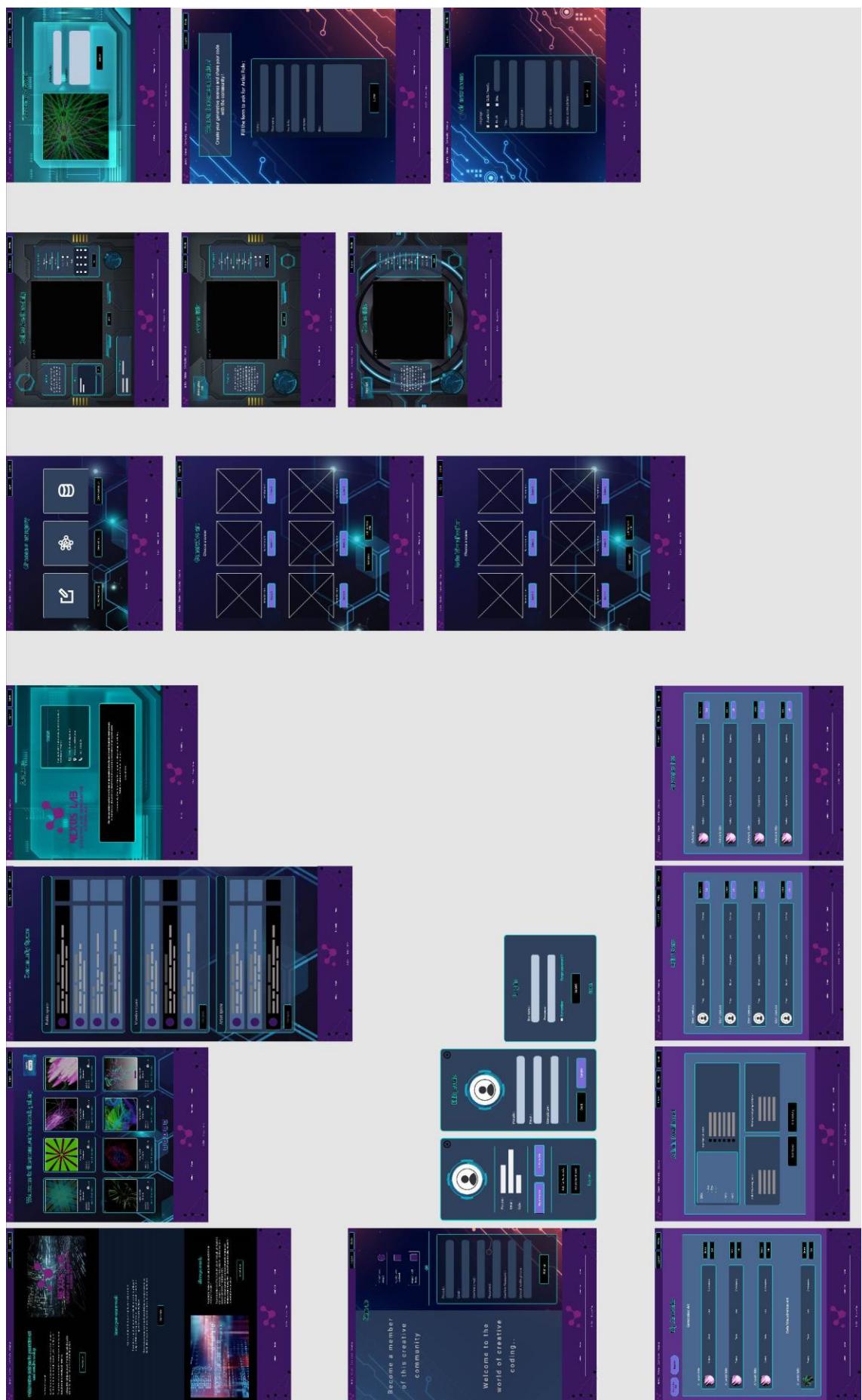


# Annexes



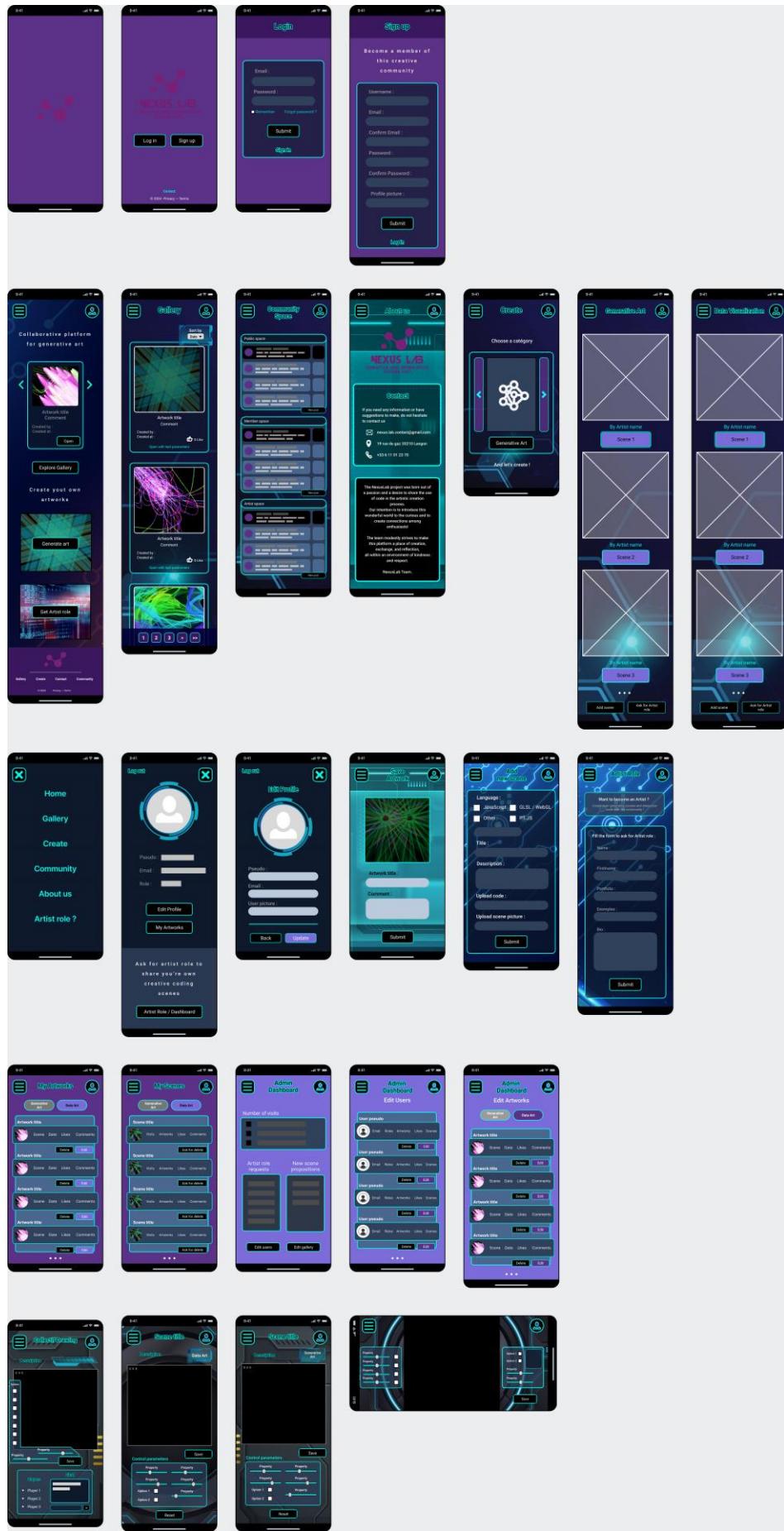
# Annexes

## Maquette NexusLab Desktop



# Annexes

## Maquette NexusLab Mobile



## Annexes

Conceleur PhpMyAdmin - base de données

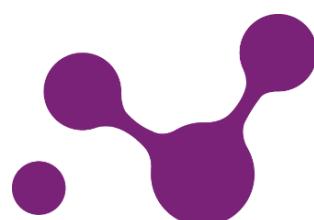


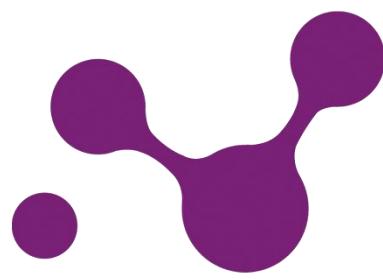
## LikeController.php

```
15  class LikeController extends AbstractController
16  {
17      #[Route("/like/artwork/{id}/{entity}", name: "artwork_like")]
18      public function like(EntityManagerInterface $entityManager, $id, string $entity): JsonResponse
19      {
20          $user = $this->getUser();
21          if (!$user) {
22              return $this->json(['error' => 'User not authenticated'], Response::HTTP_UNAUTHORIZED);
23          }
24
25          $entityClass = "App\\Entity\\" . $entity;
26          if (!class_exists($entityClass)) {
27              return $this->json(['error' => 'Invalid entity type: ' . $entity], Response::HTTP_BAD_REQUEST);
28          }
29
30          $artwork = $entityManager->getRepository($entityClass)->find($id);
31          if (!$artwork) {
32              return $this->json(['error' => 'Artwork not found'], Response::HTTP_NOT_FOUND);
33          }
34
35          if ($artwork->isLikedByUser($user)){
36              $artwork->removeLike($user);
37              $message = 'Like successfully removed.';
38          } else {
39              $artwork->addLike($user);
40              $message = 'Like successfully added.';
41          }
42
43          $entityManager->flush();
44          $likesCount = $artwork->getLikes()->count();
45
46          return $this->json(['message' => $message, 'likes' => $likesCount]);
47      }
48  }
```

## 9.3 Code Source

**GitHub :** <https://github.com/PatrickTaburet/project-cda>





# **NEXUS LAB**

**CREATIVE AND GENERATIVE  
CODING ART**