Rapport de projet d’application

Tamsir Ndiaye & Awa Fall
Projet : “AfricaArt” – Plateforme mobile de valorisation de l’art africain
Date de remise : 27 mai 2025

---

1. Contexte et objectifs

Le secteur culturel en Afrique fait face à un double défi :

- La visibilité limitée des œuvres et artistes locaux sur les plateformes numériques.
- La gestion encore papier/Excel des catalogues et inventaires.

Objectifs

1. Créer une application mobile cross-platform (iOS/Android) intuitive pour parcourir, rechercher et valoriser les œuvres d’art africain.
2. Permettre aux galeries et artistes d’importer et de gérer leur catalogue via un modèle Excel.
3. Offrir une expérience interactive (filtres, favoris, géolocalisation des expositions).

---

2. État des lieux

- **Infrastructure existante** :
    - Catalogue géré manuellement sous Excel (modèle à compléter).
    - Site web statique non optimisé mobile.
- **Usage utilisateur** :
    - Visiteurs et collectionneurs recherchent une expérience fluide et mobile-first.
- **Contraintes** :
    - Faible bande passante dans certaines zones.
    - Nécessité de fonctionner offline (catalogue pré-chargé).

---

3. Description du besoin et faisabilité

Besoin métier	Solution proposée	Faisabilité
Consultation et recherche d'œuvres	Module catalogue avec filtre avancé	✓ Ionic/Angular
Mise à jour du catalogue via Excel	Import & parsing du fichier Excel	✓ Plugin Capacitor
Affichage offline du catalogue	Stockage local (SQLite / Filesystem)	✓ Capacitor Storage
Téléchargement de fiches descriptives (PDF)	Lien de téléchargement depuis assets	✓ File Opener

---

4. Architecture fonctionnelle

1. **Authentification** (optionnelle pour administration)
2. **Catalogue**
    - Liste paginée
    - Recherche & filtres (artiste, catégorie, période)
3. **Détail de l'œuvre**
    - Images, description, informations techniques
    - Bouton "Favori"

4. **Import catalogue**
    - Lecture du modèle Excel
    - Mappage vers entités internes
5. **Paramètres & synchronisation**
    - Mise à jour manuelle / automatique

---

5. Architecture applicative

mathematica

CopierModifier

```
[ Ionic Front-end (Angular) ]  ⟷  [ Stockage local (SQLite/FileSystem) ]
          │
          ├─ HttpClient (assets/files)
          ├─ Capacitor Plugins
          │    ├─ File Opener
          │    └─ Storage / SQLite
          └─ UI : composants ioniques (Ionic UI)
```

---

6. Modèle de données (diagramme de classes)

- **Artwork**
    - id: number
    - titre: string

    - artiste: string
    - description: string
    - catégorie: string
    - année: number
    - images: string[]
- **CatalogueImport**
    - fichierExcel: File
    - dateImport: Date
- **Utilisateur** (pour module admin)
    - id, nom, email, rôle

(Voir annexe : diagramme de classes UML)

---

7. Diagrammes de cas d’utilisation et autres diagrammes pertinents

- **Cas d’usage “Consulter Catalogue”**
    1. L’utilisateur ouvre l’app
    2. L’app charge le catalogue (local/remote)
    3. L’utilisateur applique ses filtres
    4. L’app affiche les résultats
- **Cas d’usage “Importer Excel”**
    1. L’admin sélectionne le fichier modèle
    2. L’app parse et valide les données
    3. L’app enrichit la base locale

(Voir annexe : diagrammes UML de cas d’usage & séquence)

---

8. Technologies et outils utilisés

- **Framework** : Ionic 6 + Angular 15
- **Langage** : TypeScript
- **Plugins Capacitor** :
    - `@ionic-native/file-opener`

    - `@capacitor/filesystem`
    - `@capacitor/storage`
- **Parsing Excel** : `xlsx` (SheetJS)
- **Gestion du versioning** : Git / GitHub
- **CI/CD (optionnel)** : GitHub Actions / Ionic Appflow

---

9. Architecture technique

- **Front-end** : Single Page App (Ionic) déployée dans WebView native
- **Stockage** :
    - **Offline** : Filesystem + SQLite
    - **Online (futur)** : API REST (Node.js/Express + base de données)
- **Sécurité** : Permissions de lecture/écriture (Capacitor), validation des imports

---

10. Dimensionnement

Élément	Volume estimé
Nombre d’œuvres	~1 000 par galerie
Poids moyen image	200 Ko
Stockage offline	200 Mo maximum
Performances	< 200 ms par requête UX

---

11. Planning de mise en œuvre de l’outil

Phase	Durée estimée	Dates
1. Conception & maquettage	1 semaine	1–7 juin
2. Développement front-end	3 semaines	8 juin – 28 juin
3. Intégration import Excel	1 semaine	29 juin – 5 juillet
4. Tests & recette	1 semaine	6 juillet – 12 juillet
5. Déploiement & formation	3 jours	13 – 15 juillet

---

12. Budget prévisionnel

Poste	Coût unitaire	Quantité	Total (USD)
Développement Ionic/Angular	50 $/h	120 h	6 000
Intégration Excel & tests	50 $/h	40 h	2 000
Design UI/UX	60 $/h	20 h	1 200
Formation & documentation	50 $/h	10 h	500
Total estimé			**9 700**

---

13. Équipe projet

- **Tamsir Ndiaye** : Lead Front-end & Intégration
- **Awa Fall** : UI/UX, Back-end (future API), Tests & Recette

---

14. Analyse des risques

Risque	Probabilité	Impact	Plan d’atténuation
Problème de parsing du modèle Excel	Moyen	Élevé	Valider le modèle Excel en amont
Limites de stockage offline	Faible	Moyen	Mettre en place purge images non utilisées
Bugs liés aux plugins Capacitor	Faible	Élevé	Prévoir alternatives (WebView fallback)
Retard planning (tests/recette)	Moyen	Moyen	Buffer 1 semaine, tests automatisés