Rapport de Projet: Clone de r/Place avec Canvas Personnalisés

Introduction

Ce projet vise à créer une plateforme collaborative d'art pixelisé inspirée par r/Place de Reddit. Notre application permet aux utilisateurs de créer et de rejoindre des toiles personnalisées, de parcourir des toiles publiques avec des mises à jour en temps réel, et de dessiner des pixels en collaboration avec d'autres utilisateurs.

Architecture Technique

Stack Technologique Actuelle

Frontend:

- React pour l'interface utilisateur
- Tailwind CSS pour le stylage
- React-Konva pour le rendu du canvas

Backend:

- Firebase comme solution serverless complète
- Firestore pour le stockage des métadonnées
- Realtime Database pour les mises à jour en temps réel
- Authentication pour la gestion des utilisateurs

Transition de l'Architecture Précédente

Notre projet a connu une évolution significative au niveau de son architecture. Initialement, nous avions opté pour une approche plus traditionnelle:

Architecture Initiale:

- Backend en Go (Golang)
- Communication via WebSockets
- Frontend en React
- Déploiement nécessitant une infrastructure serveur

Raisons de la Transition:

1. **Simplicité de Déploiement**
   Notre architecture serverless Firebase élimine le besoin de gérer des serveurs, permettant un déploiement rapide et sans friction.

2. **Intégration Native du Temps Réel**
   La Realtime Database de Firebase offre des capacités de synchronisation en temps réel sans avoir à implémenter et maintenir notre propre infrastructure WebSocket.

3. **Développement Accéléré**
   L'élimination de la complexité backend nous permet de déployer très facilement l'application.

Fonctionnalités Principales

1. Gestion des Canvas (Firestore)

Les utilisateurs peuvent créer de nouvelles toiles avec des métadonnées stockées dans Firestore:

```
{
  "canvasId": "example123",
  "name": "Mon Tableau d'Art",
  "size": [100, 100],
  "public": true,
  "createdBy": "user123",
  "createdAt": 1710950400
}
```

Cette structure permet une recherche efficace et une gestion flexible des toiles publiques et privées.

2. Dessin de Pixels (Realtime Database)

Chaque toile stocke ses données de pixels dans la Realtime Database:

```
{
  "canvases": {
    "example123": {
      "pixels": {
        "0,0": "#FFFFFF",
```

```
      "0,1": "#FF0000"
    },
    "updatedAt": 1710950400
  }
 }
}
```


Cette architecture permet:

- Des mises à jour instantanées pour tous les utilisateurs
- Une synchronisation automatique entre les clients
- Une gestion efficace des données avec une transmission minimale

3. Découverte et Aperçus des Canvas

Notre approche ingénieuse génère des aperçus dynamiques sans stocker d'images:

- Les aperçus sont générés à partir des pixels les plus récemment modifiés
- Réduit considérablement les besoins en stockage
- Offre des aperçus toujours à jour reflétant l'activité récente

4. Authentification (Firebase Auth)

L'intégration de Firebase Authentication nous offre:

- Connexion sécurisée via Google
- Gestion simplifiée des utilisateurs
- Suivi des contributions individuelles
- Sécurité robuste sans développement supplémentaire

Infrastructure de Déploiement

1. **Render Free Tier**
   - Déploiement automatisé à partir du dépôt GitHub
   - Intégration continue avec déploiement à chaque push sur la branche principale
   - Environnement d'exécution optimisé pour les applications React
2. **Gestion des Variables d'Environnement**
   - Les secrets et clés d'API Firebase sont stockés en tant que variables d'environnement dans Render
   - Fichier .env.example dans le dépôt pour documenter les variables nécessaires:

```
VITE_FIREBASE_API_KEY=your-api-key
VITE_FIREBASE_AUTH_DOMAIN=your-auth-domain
VITE_FIREBASE_DATABASE_URL=your-database-url
VITE_FIREBASE_PROJECT_ID=your-project-id
VITE_FIREBASE_STORAGE_BUCKET=your-storage-bucket
VITE_FIREBASE_MESSAGING_SENDER_ID=your-messaging-sender-id
VITE_FIREBASE_APP_ID=your-app-id
```

3. **Configuration Nginx**
   - Serveur web Nginx pour servir l'application React compilée
   - Configuration pour la redirection des routes SPA:

```
server {
  listen 80;

  location / {
    root /app/build;
    index index.html;
    try_files $uri $uri/ /index.html;
  }

  # Compression gzip pour améliorer les performances
  gzip on;
  gzip_types text/plain text/css application/json application/javascript text/xml application/xml
application/xml+rss text/javascript;
}
```

Perspectives d’Évolution

Améliorations Futures

1. **Fonctionnalités Sociales**
    - Système de collaboration en équipes
    - Partage facile sur les réseaux sociaux
    - Classements et statistiques
2. **Outils de Dessin Avancés**
    - Outils de sélection et de remplissage
    - Palette de couleurs personnalisable
    - Fonctionnalités d’annulation/rétablissement
3. **Expérience Mobile Optimisée**
    - Application PWA pour installation sur appareils
    - Interactions tactiles améliorées
    - Mode hors ligne avec synchronisation différée

Conclusion

La transition de notre architecture traditionnelle Golang/WebSockets vers une solution serverless basée sur Firebase représente une évolution stratégique majeure pour notre projet. Cette décision nous a permis de simplifier considérablement notre infrastructure, d’accélérer notre cycle de développement, et d’offrir une expérience utilisateur plus fiable et réactive.

Les avantages en termes de déploiement, de scalabilité et de maintenance justifient pleinement ce choix architectural, malgré quelques défis inhérents aux solutions serverless. Notre équipe peut désormais se concentrer davantage sur l’innovation et l’amélioration des fonctionnalités, plutôt que sur la gestion d’infrastructure.

Cette approche moderne nous positionne idéalement pour une croissance future, avec la flexibilité nécessaire pour évoluer en fonction des besoins des utilisateurs et des tendances technologiques.