🌍 Projet : Marketplace Blockchain de Billets Mondial 2030

🎫 Objectif général

Créer une **Marketplace décentralisée** où les utilisateurs peuvent :

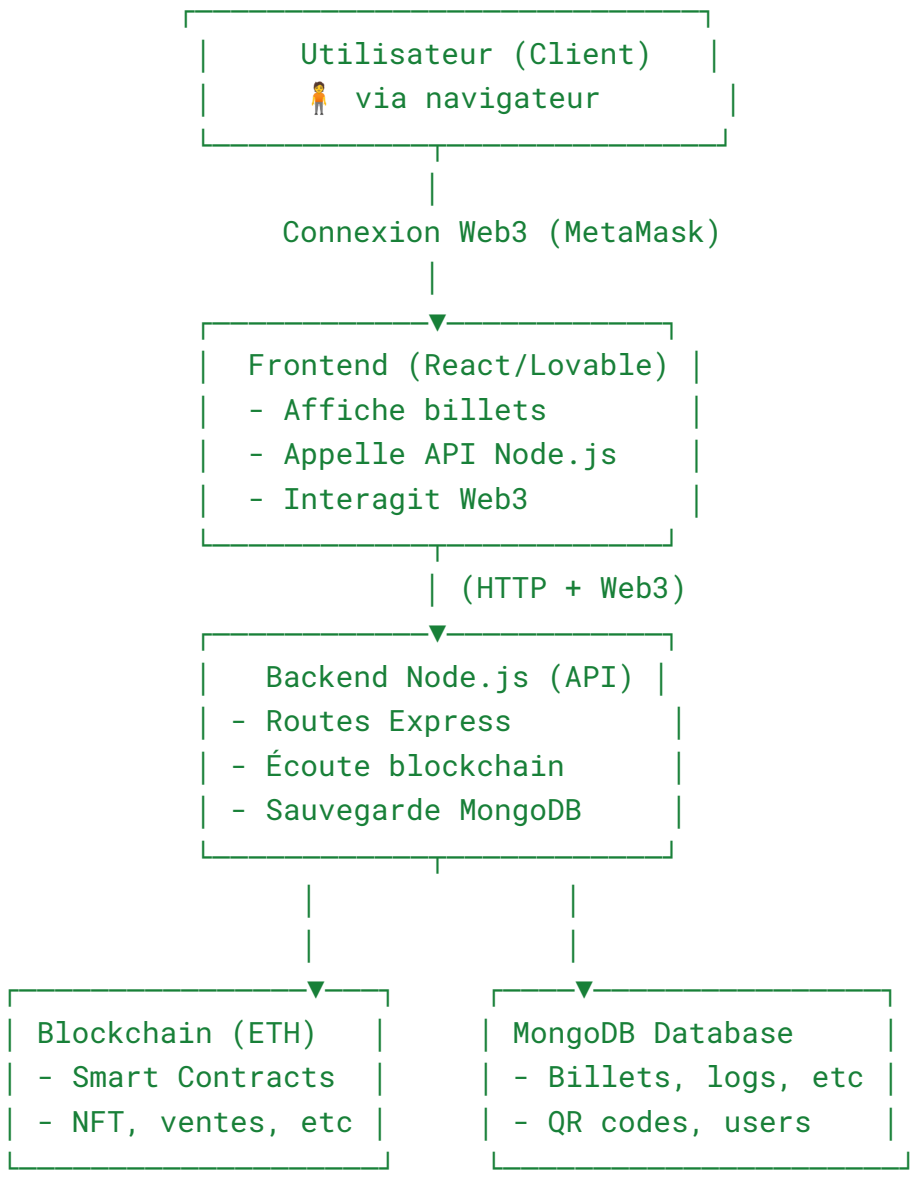
- Acheter des **billets numériques (NFT)** pour le Mondial 2030
- Revendre leurs billets de manière **sécurisée et transparente**
- Effectuer les paiements en **USDT stablecoin** (pour éviter la volatilité de l'ETH)
- Vérifier l'**authenticité** du billet grâce à un **QR code unique** infalsifiable

---

⚙️ Technologie principale

Composant	Technologie	Rôle
Smart Contracts	Solidity / Foundry	Gèrent la création des billets, les ventes, reventes, royalties et historique.
Blockchain	Ethereum (Testnet Sepolia ou Polygon Amoy)	Stocke les transactions et la propriété des billets.
Backend (API)	Node.js + Express.js	Pont entre le Frontend et la Blockchain, écoute les événements et stocke les infos dans MongoDB.
Base de données	MongoDB	Stocke les données "off-chain" : historique, QR codes, images, utilisateurs, logs.
Frontend (UI)	React (Lovable AI)	Interface visuelle : affichage des billets, connexion Metamask, achat/revente.
Token de paiement	USDT (ERC-20 stablecoin)	Permet de payer en devise stable sans volatilité.
Stockage décentralisé	IPFS	Pour stocker les images des billets et leurs métadonnées.

🧩 Schéma global du projet



🏗️ Phases du Projet (Étapes à suivre)

🔹 Phase 1 : Smart Contracts (Cœur décentralisé)

Objectif :

Développer et tester les deux contrats Solidity.

Contrats à créer :

1. **MondialTicketNFT.sol**
    - Génère les billets sous forme de NFT (ERC721).
    - Chaque billet contient :
      → équipe, catégorie, prix initial, QR code unique, etc.
    - Empêche la duplication du QR code.
2. **TicketMarketplace.sol**
    - Permet aux utilisateurs de vendre / acheter des billets NFT.
    - Paiement en USDT stablecoin.
    - Royalties de 5 % pour l'organisateur.
    - Prix de revente limité à +20 % du prix initial.
    - Historique des ventes enregistré.

Étapes :

1. Compiler les contrats avec Foundry → `forge build`
2. Écrire les tests unitaires (`forge test`) :
    - Vérifier les règles de revente.
    - Vérifier les paiements en USDT.
    - Vérifier les transferts et l'historique.
3. Déployer sur le **testnet Polygon Amoy ou Sepolia**.

🔹 Phase 2 : Backend (Node.js + MongoDB)

Objectif :

Créer un **pont entre la blockchain et le frontend**.

Étapes :

1. **Initialisation :**
    - `npm init`
    - Installer : `express`, `mongoose`, `ethers`, `dotenv`, `cors`
2. **Connexion MongoDB :**

Créer un modèle `Ticket.js` :

```
const TicketSchema = new mongoose.Schema({
  tokenId: Number,
  owner: String,
  price: String,
  qrCode: String,
  isListed: Boolean,
});
```

    -
    - `mongoose.model("Ticket", TicketSchema);`

3. **Listener Blockchain :**
    - Utiliser `ethers.js` pour écouter les événements du contrat :
        - `TicketListed`
        - `TicketSold`
        - `ListingCanceled`
    - Quand un événement est reçu :
        - Mettre à jour ou insérer les infos dans MongoDB.
4. **API REST :**
    - `GET /api/listings` → Liste des billets disponibles
    - `GET /api/listing/:id` → Détails d'un billet
    - `POST /api/buy` → Achat via la blockchain

- `POST /api/list` → Mise en vente d'un billet

🔹 Phase 3 : Frontend (React / Lovable)

Objectif :

Créer une interface claire et intuitive.

Pages principales :

1. 🏠 **Accueil :**
   - Liste des billets en vente (chargés depuis `/api/listings`).
   - Chaque billet → image, prix, catégorie, bouton "Acheter".
2. 💼 **Mes billets :**
   - Affiche les billets que l'utilisateur possède.
   - Bouton "Lister pour revente".
3. 💳 **Achat / Vente :**
   - Achat via MetaMask → appelle `buyBillet()` sur le contrat.
   - Revente → appelle `approve()` puis `listBilletForSale()`.
4. 🕵️ **Historique :**
   - Affiche les transactions passées du NFT (via `getSalesHistory()`).

🔹 Phase 4 : Test et Déploiement

Étapes :

1. Tester sur le **testnet Sepolia/Polygon Amoy**
2. Vérifier :
   - Achat / vente fonctionne

    - Royalties bien versées
    - QR code non dupliqué
3. Déployer le **frontend sur Vercel**
4. Connecter ton **API Node.js** à un hébergeur (Render / Railway / VPS)

🔒 Sécurité & Bonnes pratiques

Mesure	Description
Royalties fixes (5%)	Empêchent la revente abusive
Prix max 120%	Évite la spéculation excessive
Paiement stable (USDT)	Évite la volatilité
QR code unique	Garantit l'unicité du billet
Contrat vérifié	Transparence et confiance
Backend indexé	Rapidité de chargement du front-end

📊 Exemple de workflow utilisateur

1. L'organisateur crée un billet NFT (via MondialTicketNFT).
2. L'utilisateur A achète ce billet → paie en USDT.
3. Le billet devient sa propriété (on-chain).
4. Plus tard, il décide de le revendre sur la Marketplace.
5. L'utilisateur B l'achète → A reçoit 95 %, organisateur reçoit 5 %.
6. L'historique de la transaction est visible sur la blockchain.
7. Le QR code du billet reste valide et infalsifiable.

🧠 En résumé

Ton projet combine :

- **Décentralisation** → Smart contracts (Solidity)
- **Stabilité** → Paiement en USDT
- **Performance** → Backend Node.js + MongoDB
- **Accessibilité** → Frontend React / Lovable
- **Sécurité** → QR code unique, historique traçable, royalties automatiques