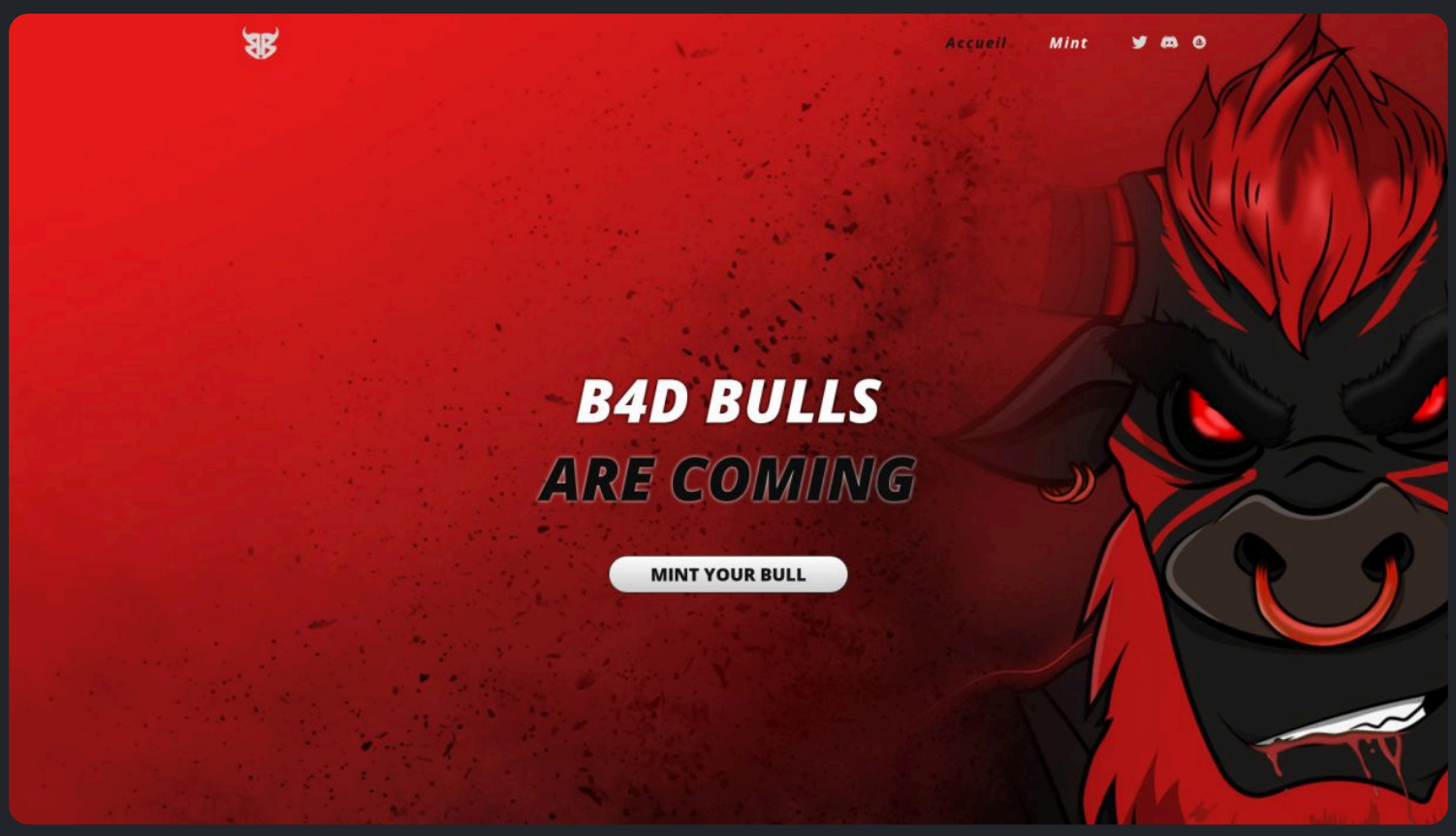


B4D Bull

Les B4D Bulls débarquent, et ils sont là pour tout casser !

Voir le site Internet

Liens directs

Smart Contract : Etherscan

DApp : https://b4dbull.cryptocademia.com

Dans la continuité du projet **B3AR Market**, l'équipe a récidivé et a décidé de sortir la contre offensive aux **B3ars**, les **B4D Bulls** !

666 NFTs uniques réalisés par **plus de 30 artistes** dont **NoNegocio** qui est derrière la totalité des NFTs non légendaires ainsi que certains légendaires.

Directement inscrit dans le lore de **Cryptocademia** (dont je participe activement au développement), cette nouvelle collection devait proposer des avantages (WL/prix) aux propriétaires de B3ARs (collection précédente) tout en ayant une réelle utilité future.

4 phases de mint, un système poussé de **whitelisting** avec des prix très différents, sans oublier un **NFT évolutif** dont le visuel change avec le temps… Le challenge était au rendez-vous !

> Très professionnel du début à la fin, Rp est toujours très efficace et autonome. Il a compris très vite ce que l'on souhaitait avoir et a été très pertinent dans ces suggestions. Les difficultés ne lui font pas peur et c'est exactement ce dont on avait besoin. Hâte de pouvoir retravailler avec lui dans un futur projet !
>
> **CryptNaAb (fondateur)**

Génération des 666 NFTs

Comme lors de la précédente collection, je dus procéder à la **génération de l'intégralité des NFTs non légendaires**. Ces derniers devaient respecter **des critères de combinaisons encore plus nombreux & poussés qu'auparavant**, et furent ensuite passés au peigne fin afin de déceler la moindre petite erreur d'association !

Par la suite, **l'intégration des différents NFTs légendaires** ainsi que le **NFT évolutif** fut réalisée, les **JSON** créés et le tout envoyé sur le protocole **IPFS**.

JSON post génération d'un NFT de la collection B4D Bull


```json
{
  "name": "B4D BULL #0",
  "description": "Each of these NFTs will be a unique witness to the market's recovery [...]",
  "image": "ipfs://bafybeiearjzjglpgq77piebstu3slunek7h3kljs7gc3ri67ke2xz5oh44/0.png",
  "imageHash": "e4d81a0548d8bd3f0f5d75868b14693c1d9435ae8c033bb3b97537c8fefec7c7",
  "edition": 0,
  "date": 1680091590735,
  "attributes": [
    {
      "trait_type": "Background",
      "value": "Big Foot Feet"
    },
    {
      "trait_type": "Body",
      "value": "Red Brown"
    },
    {
      "trait_type": "Label",
      "value": "Whale"
    },
    {
      "trait_type": "Clothes",
      "value": "The Pioneer"
    },
    {
      "trait_type": "Necklace",
      "value": "None"
    },
    [...]
  ]
}
```

Réalisation du Smart Contract

A contrario de la collection précédente, cette fois-ci j'étais chargé de la réalisation du **Smart Contract**.

Celui-ci fut le plus complexe que j'ai eu à faire jusqu'à présent, le nombre important de phases et de whitelists (via Merkle Tree) m'ont poussé à créer des **groupes de mint** au sein du SC, avec des **fonctions permettant d'agir sur ces derniers**, de sorte à changer le prix ou toute autre variable à tout moment.

Chaque phase était associée à différents groupes éligibles, proposant des quantités max et des prix différents.
Qui dit quantité max, dit possibilité de mint en lot, raison pour laquelle je suis passé sur de l'**ERC721A**.

Enfin, un **NFT légendaire** devait offrir une **évolution dynamique liée au temps passé à le détenir**. Ce dernier devait par ailleurs être totalement inconnu jusqu'au reveal. Un système d'évolution a été **directement implémenté au sein du SC**.

Un travail de réflexion et de réorganisation du code fut nécessaire, de sorte à avoir un SC compact et lisible, modulable et couvrant l'ensemble des demandes du cahier des charges... Le tout en un temps record (timing très serré).

Pour finir, le nécessaire pour les **royalties** a été mis en place, avec **l'ERC-2981** & l'habituel **Operator Filter d'OpenSea**.
Un **splitter de paiement** a été intégré afin de pouvoir dispatcher les fonds liés aux royalties sur les différents wallets de l'équipe.

Extrait du Smart Contract B4D BULL

```
constructor(string memory defaultUri, address[] memory teamMembers, uint[] memory teamShares) ERC721A("B4D BULL", "B4D")
    PaymentSplitter(teamMembers, teamShares)
    {
        uri = defaultUri;
        _setDefaultRoyalty(_msgSender(), 750); // Royalties par défaut fixées à 7.5%

        dynamicTokenId = 0;

        mintGroups[0] = mintGroup({ name: "PhaseOneYoungBull", price: 0.019 ether, maxPerWallet: 1, merkleTreeRoot: bytes32(0x62cb0af8199dd1c757bda80ed6f2041886f480fcda31c10bc1a5214f7c7dd3dc), exists: true });
        mintGroups[1] = mintGroup({ name: "PhaseOneBull", price: 0.016 ether, maxPerWallet: 1, merkleTreeRoot: bytes32(0xe49e59ec59f0814db223376378b686b7c609cc7a572e0150c3842993b7c354df), exists: true });
        mintGroups[2] = mintGroup({ name: "PhaseOneOldBull", price: 0.014 ether, maxPerWallet: 2, merkleTreeRoot: bytes32(0xffdfcd83207f952764f8aa33dad74aaeafb9176bf3110c18937719b50ad74a06), exists: true });
        mintGroups[3] = mintGroup({ name: "Public", price: 0.025 ether, maxPerWallet: 3, merkleTreeRoot: bytes32(0x0), exists: true });
        mintGroups[4] = mintGroup({ name: "PhaseThreeBull", price: 0.022 ether, maxPerWallet: 3, merkleTreeRoot: bytes32(0xb55c3e3c47aff881310504da35633c02dedf285e27fec663b70a666079617900), exists: true });
        mintGroups[5] = mintGroup({ name: "PhaseTwoOldBull", price: 0.019 ether, maxPerWallet: 3, merkleTreeRoot: bytes32(0xd2d8bb4234e4063e481b537cd7ea31c0ab009d1894e8fc1464ea7829313d18de), exists: true });
        mintGroups[6] = mintGroup({ name: "FreeMint", price: 0 ether, maxPerWallet: 1, merkleTreeRoot: bytes32(0x465a7b33c5171b235a401f260b9ee53cf66d087076f645ff6e8ffe8d5840ef56), exists: true });

    }

    // Mint
    function mint(bytes32[] calldata proof, uint256 group, uint256 amount) public payable nonReentrant {
        require(currentStep >= Step.PhaseOne && currentStep <= Step.PhaseFour, "Please wait for the mint phase");
        require(totalSupply() + amount <= supply, "Max supply reached");

        // Forcing des groupes disponibles suivant la phase
        if (currentStep == Step.PhaseOne) {
            require(group == 0 || group == 1 || group == 2 || group == 6, "Invalid mint group for this phase");
        } else if (currentStep == Step.PhaseTwo) {
            require(group >= 5 && group <= 6, "Invalid mint group for this phase");
        } else if (currentStep == Step.PhaseThree) {
            require(group >= 4 && group <= 6, "Invalid mint group for this phase");
        } else if (currentStep == Step.PhaseFour) {
            require(group >= 3 && group <= 6, "Invalid mint group for this phase");
        }

        require(isOnList(msg.sender, proof, group), "Not whitelisted for this mint group"); // WL check
        require(msg.value >= mintGroups[group].price * amount, "Not enough ETH");
        require(mintTrackers[msg.sender].groupTracker[group] + amount <= mintGroups[group].maxPerWallet, "Amount exceed the limit or max mint reached for this mint group"); // Tracker check
        mintTrackers[msg.sender].groupTracker[group] += amount; // Tracker update
        _safeMint(msg.sender, amount);
        emit newMint(msg.sender, totalSupply() - amount, amount);
    }

    // Retourne l'URI, soit dans sa version de base en mode "unrevealed", soit dans sa version définitive avec un code spécifique pour le token dynamique
    function tokenURI(uint256 tokenId) override public view returns (string memory) {
        require(_exists(tokenId), "Inexistent token");

        if (!isRevealed) { return uri; }

        if (tokenId == dynamicTokenId) {
            uint256 lastTransfer = _ownershipOf(tokenId).startTimestamp; // Specific ERC721A
            uint256 diff = block.timestamp - lastTransfer;
            string memory evPhase;

            if (diff > 120 days) {
                evPhase = "5";
            } else if (diff > 60 days) {
                evPhase = "4";
            } else if (diff > 30 days) {
                evPhase = "3";
            } else if (diff > 15 days) {
                evPhase = "2";
            } else { evPhase = "1"; }

            return string(abi.encodePacked(uri, tokenId.toString(), "_", evPhase, ".json"));
        } else {
            return string(abi.encodePacked(uri, tokenId.toString(), ".json"));
        }
    }
```

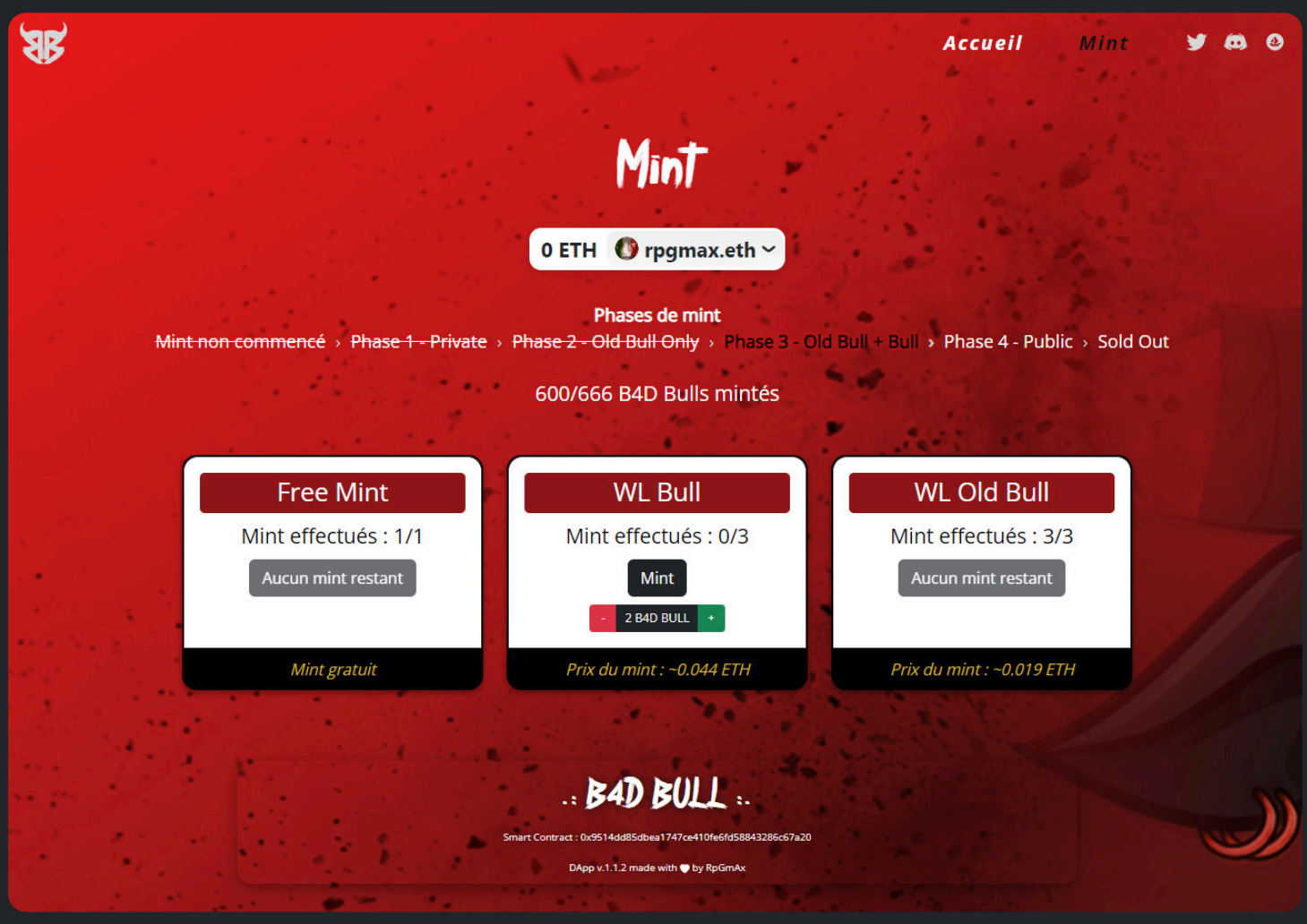
Réalisation de la DApp

A l'image du **Smart Contract**, la DApp fut davantage complexe, la « faute » aux nombreuses phases et aux nombreux groupes de mint.

Le challenge était donc de rendre tout ceci le plus digeste possible pour l'utilisateur, qu'il s'agisse des phases mais aussi et surtout des différentes options de mint allouées à chacun. **Des sessions de tests** ont été réalisées avec un groupe d'utilisateurs, incluant une écoute toute particulière sur leur « feeling » vis à vis du mint. Quelques ajustements ont été réalisés, notamment en ce qui concerne le multi-mint (mint de X exemplaires en une seule transaction).

La DApp fut développée de manière à réagir instantanément aux différentes modifications opérées sur le smart contract et de proposer une **expérience parfaitement fluide**.

Enfin, et comme toujours, **le responsive a été travaillé de façon à offrir la même expérience pour les utilisateurs mobiles,** toujours plus nombreux.



Loterie

Les deux collections étant intrinsèquement liées, cela devait se refléter sur la loterie « **Cryptocademia** » offrant la possibilité aux propriétaires des NFTs concernés de participer à une loterie mensuelle.

Celle-ci a été réalisée sous forme d'affrontement « dynamique » entre un B3AR et un BULL, avec des images animées successives donnant lieu à une victoire pour l'un ou l'autre. Chaque NFT dispose d'un « poids » qui lui est propre, jouant directement sur les probabilités de victoire.

Ce combat permet de déterminer un vainqueur pour la Cryptocademia Lottery 🎫

DÉBUTER UN NOUVEAU COMBAT ⚔️