**Day 4 - Dynamic Frontend Components - TokenRent**

**1. Project Overview**

TokenRent is a **Web3-powered decentralized rental marketplace** that enables users to rent high-value equipment securely using **blockchain-based smart contracts** and **NFT ownership verification**. The frontend is built using **Next.js**, interacting with **Ethereum smart contracts** and **Sanity CMS** for content management.

The core features of the frontend include:

* **Metamask Authentication** for user login.
* **NFT Minting and Equipment Listing** on the blockchain.
* **Dynamic Product Display** from Sanity CMS.
* **Web3-powered Checkout** where users interact with smart contracts to rent equipment

**2. Key Frontend Components**

**2.1 Product Listing Page**

* Dynamically fetches available rental items from **Sanity CMS**.
* Displays **NFT metadata**, including name, price, image, and availability.
* Users can **filter** by category, price range, and availability.

**Code (Fetching Products from Sanity CMS)**:

const EquipmentCollection = () => {

  const router = useRouter();

  const [equipment, setEquipment] = useState<SanityEquipment[]>([]);

  const [loading, setLoading] = useState(true);

  const [error, setError] = useState<string | null>(null);

  useEffect(() => {

    const fetchEquipment = async () => {

      try {

        // Modified query to get first 4 items without specific ordering

        const query = `\*[\_type == "equipment"][0...5] {

          \_id,

          name,

          slug,

          dailyRate,

          images,

          status,

          nftAddress,

          tokenId

        }`;

        const data = await client.fetch<SanityEquipment[]>(query);

        // Remove duplicates based on NFT address and token ID

        const uniqueEquipment = data.reduce((acc: SanityEquipment[], current: SanityEquipment) => {

          const isDuplicate = acc.find(item =>

            item.nftAddress === current.nftAddress &&

            item.tokenId === current.tokenId

          );

          if (!isDuplicate) {

            acc.push(current);

          }

          return acc;

        }, []);

        if (!data || data.length === 0) {

          throw new Error('No equipment found');

        }

        setEquipment(uniqueEquipment);

      } catch (err) {

        console.error('Fetch error:', err);

        setError(err instanceof Error ? err.message : 'Failed to load equipment');

      } finally {

        setLoading(false);

      }

    };

    fetchEquipment();

  }, []);

  const getImageUrl = (image: SanityImage) => {

    return builder.image(image).width(800).height(800).url();

  };

  if (loading) {

    return (

      <div className="bg-white py-16 px-4 sm:px-6 lg:px-8">

        <div className="max-w-7xl mx-auto text-center">

          <div className="animate-pulse space-y-8">

            <div className="h-12 bg-gray-200 rounded w-3/4 mx-auto"></div>

            <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-8">

              {[...Array(4)].map((\_, i) => (

                <div key={i} className="aspect-square bg-gray-200 rounded-2xl" />

              ))}

            </div>

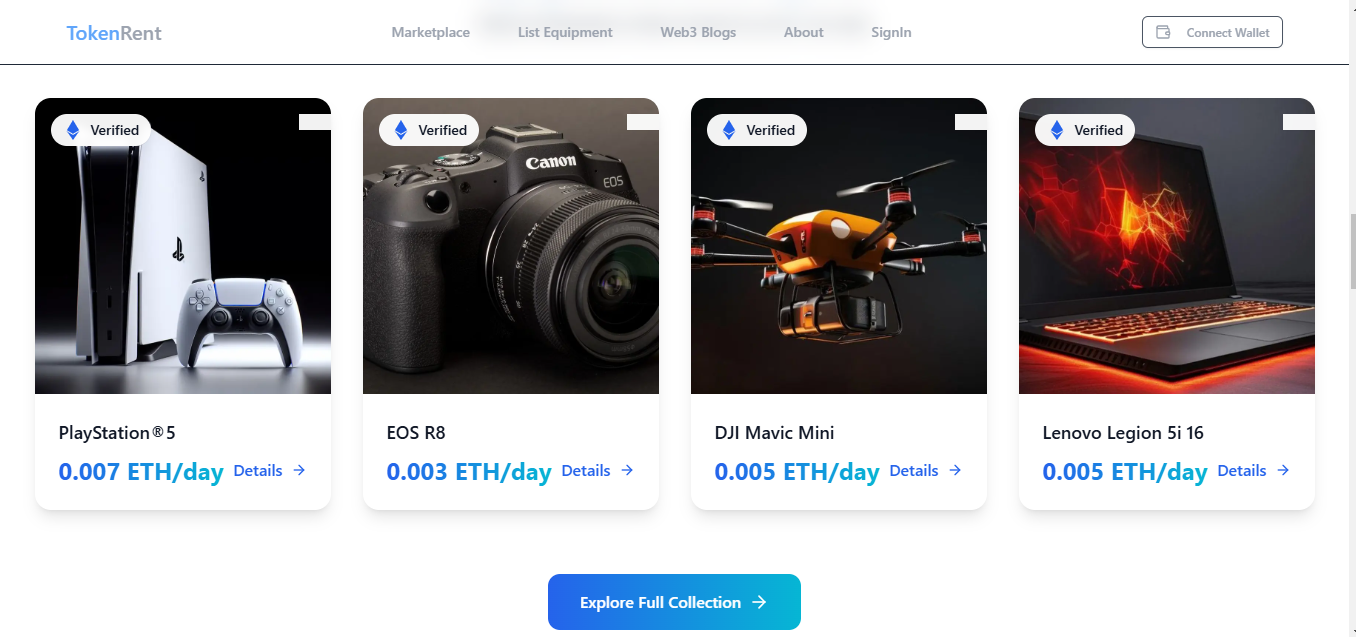
          </div>

        </div>

      </div>

    );

  }



**2.2 Product Detail Page**

* Displays complete **NFT details**, including **rental price, deposit, and rental duration**.
* Users can initiate a **rental agreement** by interacting with the **RentalAgreement smart contract**.

**Code (Fetching Product Details and Interacting with Smart Contract)**:

const mintAndListEquipment = async () => {

  if (!signer || !imageFile) {

    setError("Please select an image file");

    return;

  }

  setLoading(true);

  setError(null);

  setSuccess(null);

  const tempId = Date.now();

  try {

    // First upload image to Sanity

    const imageAsset = await uploadImageToSanity(imageFile);

    if (!imageAsset) {

      throw new Error('Failed to upload image');

    }

    // Create optimistic equipment object

    const optimisticItem: Equipment = {

      tokenId: tempId,

      name: newEquipment.name,

      imageUri: previewUrl || '',

      rentalPrice: newEquipment.dailyPrice,

      deposit: newEquipment.deposit,

      isListed: true,

      isRented: false,

      isPending: true

    };

    // Add to optimistic display

    setOptimisticEquipment(prev => [...prev, optimisticItem]);

    setShowAddModal(false);

    // Create NFT contract instance

    const nftContract = new ethers.Contract(

      NFT\_CONTRACT\_ADDRESS,

      EquipmentNFTABI.abi,

      signer

    );

    const rentalContract = new ethers.Contract(

      RENTAL\_CONTRACT\_ADDRESS,

      RentalAgreementABI.abi,

      signer

    );

    // Get the Sanity image URL

    const imageUrl = urlFor(imageAsset).url();

    // Mint NFT

    const mintTx = await nftContract.mintEquipment(

      newEquipment.name,

      newEquipment.description,

      imageUrl

    );

    setPendingTransactions(prev => new Set(prev).add(mintTx.hash));

    const mintReceipt = await mintTx.wait();

    // const transferEventInterface = nftContract.interface.getEvent('Transfer');

    // const transferEvent = mintReceipt.logs.find(

    //   (log: ethers.Log | ethers.LogDescription) => log.fragment?.name === 'Transfer'

    // );

    // const transferEvent = mintReceipt.logs.find(log => {

    //   try {

    //     const parsed = nftContract.interface.parseLog({

    //       topics: log.topics,

    //       data: log.data

    //     });

    //     return parsed.name === 'Transfer';

    //   } catch {

    //     return false;

    //   }

    // });

    // const transferLog = mintReceipt.logs.find(log => {

    //   try {

    //     const parsed = nftContract.interface.parseLog(log);

    //     return parsed?.name === 'Transfer';

    //   } catch {

    //     return false;

    //   }

    // });

    const transferLog = mintReceipt.logs.find((log: ethers.Log) => {

      try {

        const parsed = nftContract.interface.parseLog(log);

        return parsed?.name === 'Transfer';

      } catch {

        return false;

      }

    });

    // const tokenId = transferLog ? nftContract.interface.parseLog(transferLog).args[2] : undefined;

  //   const tokenId = transferLog

  // ? (nftContract.interface.parseLog(transferLog)?.args as any[])?.[2]

  // : undefined;

  const parsedLog = transferLog ? nftContract.interface.parseLog(transferLog) : null;

  const tokenId = parsedLog?.args instanceof Array ? parsedLog.args[2]?.toString() : undefined;

    // const tokenId = transferEvent?.args[2];

    // Approve rental contract

    const approveTx = await nftContract.approve(RENTAL\_CONTRACT\_ADDRESS, tokenId);

    await approveTx.wait();

    // List for rental

    const dailyPriceWei = ethers.parseEther(newEquipment.dailyPrice);

    const depositWei = ethers.parseEther(newEquipment.deposit);

    const listTx = await rentalContract.listEquipments(

      tokenId,

      dailyPriceWei,

      depositWei

    );

    await listTx.wait();

    // Prepare Sanity document data

    const equipmentData = {

      name: newEquipment.name,

      description: newEquipment.description,

      dailyRate: parseFloat(newEquipment.dailyPrice),

      securityDeposit: parseFloat(newEquipment.deposit),

      nftAddress: NFT\_CONTRACT\_ADDRESS,

      tokenId: tokenId.toString(),

      images: [imageAsset],

      category: 'Other', // You can add a category selector to your form if needed

      blockchainData: {

        nftAddress: NFT\_CONTRACT\_ADDRESS,

        tokenId: tokenId.toString(),

        rentalContract: RENTAL\_CONTRACT\_ADDRESS

      }

    };

    // Check for duplicates

    const duplicate = await checkForDuplicate(NFT\_CONTRACT\_ADDRESS, tokenId.toString());

    if (duplicate) {

      throw new Error('This product already exists in Sanity.');

    }

    // Save to Sanity

    await createEquipmentListing(equipmentData);

    // Cleanup and refresh

    setOptimisticEquipment(prev => prev.filter(item => item.tokenId !== tempId));

    await loadEquipment();

    setSuccess("Equipment listed successfully!");

  } catch (error) {

    console.error("Transaction error:", error);

    setOptimisticEquipment(prev => prev.filter(item => item.tokenId !== tempId));

    setError("Failed to list equipment.");

  } finally {

    setLoading(false);

    if (previewUrl) {

      URL.revokeObjectURL(previewUrl);

    }

    setImageFile(null);

    setPreviewUrl(null);

  }

};

 const endRentalAgreement = async (tokenId: number) => {

    if (!signer) return;

    setLoading(true);

    setError(null);

    setSuccess(null);

    try {

      const rentalContract = new ethers.Contract(

        RENTAL\_CONTRACT\_ADDRESS,

        RentalAgreementABI.abi,

        signer

      );

      const tx = await rentalContract.endRentAgreement(tokenId);

      await tx.wait();

      setSuccess("Rental agreement ended successfully!");

      await loadEquipment();

    } catch (error) {

      console.error("Error ending rental:", error);

      setError("Failed to end rental agreement.");

    } finally {

      setLoading(false);

    }

  };

**2.3 Checkout Flow**

* Displays a summary of selected equipment.
* Calls **startRentalAgreement function** in the smart contract to process payments.
* Uses **Ethereum transactions** for deposit security.

**2.4 Search and Filter Components**

* Implements **search functionality** to filter equipment by name.
* Uses **category-based filtering** to refine results dynamically.

**3. Challenges Faced and Solutions Implemented**

**3.1 Issue: Handling Smart Contract Transactions on Frontend**

* **Problem**: Some users experienced failed transactions due to **incorrect gas estimation**.
* **Solution**: Integrated **Ethers.js** with proper **gas limits** and **error handling**.

**3.2 Issue: Fetching NFT Metadata Efficiently**

* **Problem**: Fetching metadata from **Sanity CMS and blockchain** caused delays.
* **Solution**: Used **Next.js API Routes** to cache blockchain data.