**Day 3 - API Integration and Data Migration - TokenRent**

For Day 3, the focus was on integrating dynamic data sources into TokenRent and migrating data to create a fully dynamic and responsive marketplace. While the standard approach involved using the provided API keys to access external data, I took a different route by connecting directly to a blockchain node. This approach allowed me to fetch real-time data from the blockchain, ensuring that information such as NFT ownership, rental statuses, and transaction records is always up-to-date and directly verifiable on-chain.

Instead of relying on traditional API endpoints secured with API keys, the system queries a blockchain node to retrieve critical data for our decentralized marketplace. By doing so, TokenRent benefits from enhanced transparency and trust, as all the data comes straight from the blockchain without intermediaries. This method was integrated into our Next.js frontend, where data from the blockchain is merged seamlessly with content managed in Sanity CMS. This hybrid approach ensures that the product listings, rental agreements, and NFT metadata remain current and verifiable.

During the integration process, various challenges such as ensuring low latency and handling data consistency were addressed. I employed robust error handling mechanisms and fallback strategies to manage scenarios where blockchain data might temporarily be unavailable. Additionally, the smart contracts were designed to work in harmony with the data fetched from the blockchain node, enabling accurate and efficient updates to the marketplace’s state.

As part of the submission here is some image of my code and real time logs from blockchain.

  const loadEquipment = async () => {

    if (!provider || !signer || !walletAddress) return;

    setLoading(true);

    setError(null);

    try {

      const nftContract = new ethers.Contract(

        NFT\_CONTRACT\_ADDRESS,

        EquipmentNFTABI.abi,

        signer

      );

      const rentalContract = new ethers.Contract(

        RENTAL\_CONTRACT\_ADDRESS,

        RentalAgreementABI.abi,

        signer

      );

      // Get total minted tokens

      const totalSupply = Number(await nftContract.getTotalSupply());

      const tokens: bigint[] = [];

      // Fetch all tokens owned by the user

      for (let tokenId = 0; tokenId < totalSupply; tokenId++) {

        try {

          const owner = await nftContract.ownerOf(tokenId);

          if (owner.toLowerCase() === walletAddress.toLowerCase()) {

            tokens.push(BigInt(tokenId));

          }

        } catch (error) {

          console.error(`Token ${tokenId} not found or error:`, error);

        }

      }

        interface NFTData {

          name: string;

          imageUri: string;

          // Add other fields from your contract as needed

        }

      // Process equipment data

      const equipmentData = await Promise.all(

        tokens.map(async (tokenIdBigInt) => {

          const tokenId = Number(tokenIdBigInt);

          try {

            const nftData = await nftContract.getEquipment(tokenId) as NFTData;

            const [rentalPrice, deposit, isListed, isRented] =

              await rentalContract.getEquipmentDetails(tokenId)

                .catch(() => [0, 0, false, false]);

            const rentalDetails = isRented ?

              await rentalContract.getRentalDetails(tokenId).catch(() => null) :

              null;

          //   return {

          //     tokenId,

          //     name: nftData.name,

          //     imageUri: nftData.imageUri,

          //     rentalPrice: ethers.formatEther(rentalPrice),

          //     deposit: ethers.formatEther(deposit),

          //     isListed: isListed === true,

          //     isRented: isRented === true,

          //     rentalDetails: rentalDetails ? {

          //       renter: rentalDetails.renter,

          //       startTime: Number(rentalDetails.startTime),

          //       duration: Number(rentalDetails.duration),

          //       depositAmount: ethers.formatEther(rentalDetails.depositAmount),

          //       isActive: rentalDetails.isActive,

          //       timeRemaining: Number(rentalDetails.timeRemaining)

          //     } : undefined

          //   };

          // } catch (error) {

          //   console.error(`Error processing token ${tokenId}:`, error);

          //   return null;

          return {

            tokenId,

            name: nftData.name, // Now properly typed as string

            imageUri: nftData.imageUri, // Now properly typed as string

            rentalPrice: ethers.formatEther(rentalPrice),

            deposit: ethers.formatEther(deposit),

            isListed: isListed === true,

            isRented: isRented === true,

            rentalDetails: rentalDetails ? {

              renter: rentalDetails.renter,

              startTime: Number(rentalDetails.startTime),

              duration: Number(rentalDetails.duration),

              depositAmount: ethers.formatEther(rentalDetails.depositAmount),

              isActive: rentalDetails.isActive,

              timeRemaining: Number(rentalDetails.timeRemaining)

            } : undefined

          } as Equipment; // Explicit cast here

        } catch (error) {

          console.error("Error processing token", tokenId, ":", error);

          return null;

          }

        })

      );

      // Filter out null entries and sort by newest first

      // setEquipments(equipmentData

      //   .filter((e): e is Equipment => e !== null)

      //   .sort((a, b) => b.tokenId - a.tokenId)

      // );

      setEquipments(equipmentData

        .filter((e): e is Equipment => e !== null)

        .sort((a, b) => b.tokenId - a.tokenId) as Equipment[]

      );

    } catch (error) {

      console.error("Error loading equipment:", error);

      setError("Failed to load equipment data. Check console for details.");

    } finally {

      setLoading(false);

    }

  };

  // correct code but without sanity

  // const mintAndListEquipment = async () => {

  //   if (!signer) return;

  //   setLoading(true);

  //   setError(null);

  //   setSuccess(null);

  //   // Create optimistic equipment object

  //   const tempId = Date.now();

  //   const optimisticItem: Equipment = {

  //     tokenId: tempId,

  //     name: newEquipment.name,

  //     imageUri: newEquipment.imageUri,

  //     rentalPrice: newEquipment.dailyPrice,

  //     deposit: newEquipment.deposit,

  //     isListed: true,

  //     isRented: false,

  //     isPending: true

  //   };

  //   // Add to optimistic display immediately

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

  //   setShowAddModal(false);

  //   try {

  //     const nftContract = new ethers.Contract(

  //       NFT\_CONTRACT\_ADDRESS,

  //       EquipmentNFTABI.abi,

  //       signer

  //     );

  //     const rentalContract = new ethers.Contract(

  //       RENTAL\_CONTRACT\_ADDRESS,

  //       RentalAgreementABI.abi,

  //       signer

  //     );

  //     // Mint NFT

  //     const mintTx = await nftContract.mintEquipment(

  //       newEquipment.name,

  //       newEquipment.description,

  //       newEquipment.imageUri

  //     );

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

  //     const mintReceipt = await mintTx.wait();

  //     const transferEvent = mintReceipt.logs.find(

  //       (log: any) => log.fragment && log.fragment.name === 'Transfer'

  //     );

  //     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();

  //     // Remove optimistic item and update real data

  //     setOptimisticEquipment(prev =>

  //       prev.filter(item => item.tokenId !== tempId)

  //     );

  //     // Wait for a few blocks for data consistency

  //     await new Promise(resolve => setTimeout(resolve, 5000));

  //     await loadEquipment();

  //     setSuccess("Equipment listed successfully!");

  //   } catch (error: any) {

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

  //     // Remove failed optimistic update

  //     setOptimisticEquipment(prev =>

  //       prev.filter(item => item.tokenId !== tempId)

  //     );

  //     setError(error.reason || "Failed to list equipment.");

  //   } finally {

  //     setLoading(false);

  //   }

  // };

// code with sanity

// const mintAndListEquipment = async () => {

//   if (!signer) return;

//   setLoading(true);

//   setError(null);

//   setSuccess(null);

//   // Create optimistic equipment object

//   const tempId = Date.now();

//   const optimisticItem: Equipment = {

//     tokenId: tempId,

//     name: newEquipment.name,

//     imageUri: newEquipment.imageUri,

//     rentalPrice: newEquipment.dailyPrice,

//     deposit: newEquipment.deposit,

//     isListed: true,

//     isRented: false,

//     isPending: true

//   };

//   // Add to optimistic display immediately

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

//   setShowAddModal(false);

//   try {

//     const nftContract = new ethers.Contract(

//       NFT\_CONTRACT\_ADDRESS,

//       EquipmentNFTABI.abi,

//       signer

//     );

//     const rentalContract = new ethers.Contract(

//       RENTAL\_CONTRACT\_ADDRESS,

//       RentalAgreementABI.abi,

//       signer

//     );

//     // Mint NFT

//     const mintTx = await nftContract.mintEquipment(

//       newEquipment.name,

//       newEquipment.description,

//       newEquipment.imageUri

//     );

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

//     const mintReceipt = await mintTx.wait();

//     const transferEvent = mintReceipt.logs.find(

//       (log: any) => log.fragment && log.fragment.name === 'Transfer'

//     );

//     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();

//     // Save to Sanity

//     const equipmentData = {

//       name: newEquipment.name,

//       description: newEquipment.description,

//       dailyRate: parseFloat(newEquipment.dailyPrice),

//       securityDeposit: parseFloat(newEquipment.deposit),

//       nftAddress: NFT\_CONTRACT\_ADDRESS,

//       tokenId: tokenId.toString(),

//       images: [{ \_type: 'image', asset: { \_ref: newEquipment.imageUri } }],

//     };

//     // 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);

//     // Remove optimistic item and update real data

//     setOptimisticEquipment(prev =>

//       prev.filter(item => item.tokenId !== tempId)

//     );

//     // Wait for a few blocks for data consistency

//     await new Promise(resolve => setTimeout(resolve, 5000));

//     await loadEquipment();

//     setSuccess("Equipment listed successfully!");

//   } catch (error: any) {

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

//     // Remove failed optimistic update

//     setOptimisticEquipment(prev =>

//       prev.filter(item => item.tokenId !== tempId)

//     );

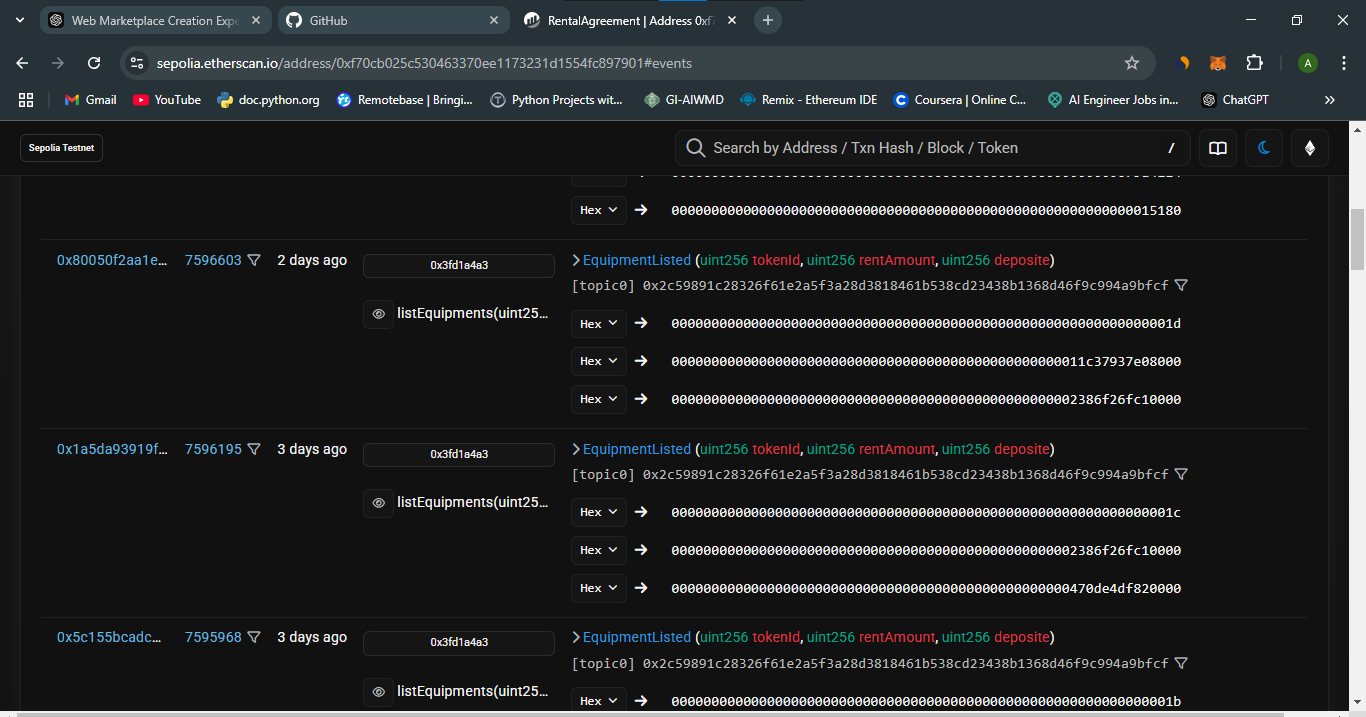
//     setError(error.reason || error.message || "Failed to list equipment.");

//   } finally {

//     setLoading(false);

//   }

// };



This approach not only strengthens the decentralization of TokenRent but also lays a robust foundation for future enhancements as we continue to build on a fully blockchain-integrated architecture.