Code Explanation For Online Facility Booking Sysytem

This is a follow-up document of `"5. Source Code in Solution Details"` section in my final thesis for the Global Master's in Blockchain Technologies. The files containing the major part of my work was done for the final thesis listed below. All the comments are added inline of each file.

Smart contract - FacilityBooking.sol

file path `/packages/hardhat/contracts/FacilityBooking.sol`

```
//SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;
pragma experimental ABIEncoderV2; // required to return
struct(BookingData) data on getBookingData function

import "hardhat/console.sol"; // to output log in hardhat. Delete this
when deploying to production
import "@openzeppelin/contracts/access/Ownable.sol"; // Use Openzeppelin's
Ownable contract

/**
 * @title FacilityBooking
 * @dev Online facility booking smart contract
 */
contract FacilityBooking is Ownable {

    /*
      Parameters
    */
    string building_name = "Riverside Residence";
    string[] facilities = ["tennis"]; // For now, just handle a facility
    uint[] fees = [0.001 ether]; // Fees per slot. Must be as same as the
frontend fee setting

    /*
      Structs
    */
    // Representation of each booking data
    struct BookingData {
        uint bookingStart;
        uint bookingEnd;
        address resident;
        uint bookingFee;
    }

    // Representation of each facility
    struct FacilityInfo {
        BookingData[] bookingData; // FacilityInfo contains an array of
booking data
```

```
        bool isValid;
        uint fee;
    }

    /*
      Mappings
     */
    mapping(string => FacilityInfo) facilityData; // contains all booking
data by facility name
    mapping(address => uint) refunds; // keeps resident's fees for
canceled bookings

    /*
      Events
     */
    // Emit when new booking request is added
    event NewBooking(string facility, address resident, uint start, uint
end);
    // Emit when an existing booking request is cancled
    event CancelBooking(string facility, address resident, uint start,
uint end, bool byAdmin);
    // Emit when owen withdraw the contract balance
    event Withdrawal(address admin, uint amount);

    /*
      Errors
     */
    /// Wrong Fee. Needed `required` but sent `fee`
    /// @param facility facility name
    /// @param resident resident's wallet address
    /// @param start    start time of booking slot
    /// @param end      end time of booking slot
    /// @param fee      sent fee amount
    /// @param required required fee amount
    error WrongFee(string facility, address resident, uint start, uint
end, uint fee, uint required);

    /// The time slot from `start` to `end` is taken
    /// @param facility facility name
    /// @param resident resident's wallet address
    /// @param start    start time of booking slot
    /// @param end      end time of booking slot
    error AlreadyTaken(string facility, address resident, uint start, uint
end);

    /*
      Modifier
     */
    // Check if facility exist in this contract
    modifier isExist(string memory _facility) {
        require(facilityData[_facility].isValid == true, "The facility
does not exist");
        _;
    }
    constructor() {
        // initialize the facility information using the parameters. For
now, only tennis is set.
        for (uint i = 0; i < facilities.length; i++) {
            facilityData[facilities[i]].isValid = true;
            facilityData[facilities[i]].fee = fees[i];
        }
    }

    /**
     * @dev register a booking request
     * @param _start    time of booking slot
     * @param _end      end time of booking slot
     * @param _facility facility name
     */
    function register(uint _start, uint _end, string calldata _facility)
        isExist(_facility)
        external
        payable
        returns (bool)
        {
            if (owner() != msg.sender) { // owner does not need to pay fee
because she is supposed to receive fee in other way(i.e., cash payment)
                uint thirtyMinsInMsec = 30 * 60 * 1000; // 30 mins in
millisecond. the current time slot length is 30 min.
                uint numOfSlots = (_end - _start) / thirtyMinsInMsec; //
calculate the number of slots from booking start to end time passed in
millisecond
                uint required = facilityData[_facility].fee * numOfSlots;
                if (msg.value != required)
                    revert WrongFee(_facility, msg.sender, _start, _end,
msg.value, required);
            }

            uint current = block.timestamp;
            if (current > _start || isBooked(_start, _end, _facility) ==
true) {
                revert AlreadyTaken(_facility, msg.sender, _start, _end);
            } else {
                facilityData[_facility].bookingData.push(
                    BookingData({
                        bookingStart: _start,
                        bookingEnd:   _end,
                        resident: msg.sender,
                        bookingFee: msg.value
                    })
                );
                emit NewBooking(_facility , msg.sender, _start, _end);
                return true;
            }
        }

    /**
```

```
     * @dev check if new booking slot is already take
     * @param _start    time of booking slot
     * @param _end      end time of booking slot
     * @param _facility facility name
     */
    function isBooked(uint _start, uint _end, string calldata _facility)
        private
        view
        returns(bool)
        {
            for (uint i = 0; i <
facilityData[_facility].bookingData.length; i++) {
                if (_start <
facilityData[_facility].bookingData[i].bookingEnd &&
                    _end >
facilityData[_facility].bookingData[i].bookingStart) {
                        return true;
                    }
            }
            return false;
        }

    /**
     * @dev return all booking data of the facility
     * @param _facility facility name
     */
    function getBookingData(string calldata _facility) isExist(_facility)
external view returns (BookingData[] memory) {
        return facilityData[_facility].bookingData;
    }

    /**
     * @dev cancel an existing booking
     * @param _start    time of booking slot
     * @param _end      end time of booking slot
     * @param _facility facility name
     */
    function cancelBooking(uint _start, uint _end, string calldata
_facility) isExist(_facility) external returns (bool) {
        address owner = owner();
        bool isOwner = owner == msg.sender;
        for (uint i = 0; i < facilityData[_facility].bookingData.length;
i++) {
            if (!isOwner) { // owner is allowed to cancel any booking
                // Check if the sender and booker are the same
                if (facilityData[_facility].bookingData[i].resident !=
msg.sender) {
                    continue;
                }
            }

            // If the sender and booker are the same, find the
corresponding booking slot
            if (_start ==
facilityData[_facility].bookingData[i].bookingStart &&
                _end == facilityData[_facility].bookingData[i].bookingEnd)
{
                    address resident =
facilityData[_facility].bookingData[i].resident;
                    uint bookingFee =
facilityData[_facility].bookingData[i].bookingFee;
                    delete facilityData[_facility].bookingData[i];
                    // Follow the best practice "Favor pull over push for
external calls"
                    // Thus, do not refund in this function.
                    // https://consensys.github.io/smart-contract-best-
practices/recommendations/#favor-pull-over-push-for-external-calls
                    if (resident != address(0)) {
                        refunds[resident] += bookingFee;
                    }
                    emit CancelBooking(_facility , resident, _start, _end,
isOwner);
                    return true;
                }
            }
            return false;
        }

        /**
         * @dev transfer the resident's booking fee for cancel slots
         */
        function refund() external {
            // Checks-Effects-Interactions Pattern to prevent Reentrancy
Attacks
            // https://docs.soliditylang.org/en/develop/security-
considerations.html?highlight=check%20effects#use-the-checks-effects-
interactions-pattern
            uint amount = refunds[msg.sender];
            require(amount > 0, "You have no refund");
            require(amount < address(this).balance, "Please try it later. Not
enough balance available to return your fund");
            refunds[msg.sender] = 0;

            (bool success, ) = msg.sender.call{value: amount}("");
            require(success, "Failed to send the amount");
        }

        /**
         * @dev return the amount of a resident's booking fee for canceled
slots
         * @param _resident resident wallet address
         */
        function getRefundAmount(address _resident) external view returns
(uint){
            return refunds[_resident];
        }

        /**
```

```
     * @dev return facility name
     */
    function getFacilityNames() external view returns (string memory){
        return facilities[0]; // temporally return the first facility name
    }

    /**
     * @dev return building name
     */
    function getBuildingName() external view returns (string memory){
        return building_name;
    }

    /**
     * @dev return fee of the facility
     * @param _facility facility name
     */
    function getFees(string calldata _facility) external view returns
(uint){
        return facilityData[_facility].fee;
    }

    /**
     * @dev return owner
     */
    function getOwner() external view returns (address){
        return owner();
    }

    /**
     * @dev return contract balance
     */
    function getBalance() external onlyOwner view returns (uint){
        return address(this).balance;
    }

    /**
     * @dev to receive Ether. msg.data must be empty
     */
    receive() external payable {}

    /**
     * @dev withdraw all balance of this contract
     */
    function withdraw() onlyOwner external {
        require(address(this).balance > 0, "Not enough balance available
to return your fund");
        address owner = owner();
        uint amount = address(this).balance;

        (bool success, ) = owner.call{value: amount}("");
        require(success, "Failed to send the amount");
        emit Withdrawal(owner, amount);
    }
```

```
}
```

Deploy tool - 00_deploy_your_contract.js

file path `/packages/hardhat/deploy/00_deploy_your_contract.js`

```
const { ethers } = require("hardhat");

module.exports = async ({ getNamedAccounts, deployments }) => {
  const { deploy } = deployments;
  const { deployer } = await getNamedAccounts();

  // Deploy our FacilityBooking.sol
  await deploy("FacilityBooking", {
    // Learn more about args here: https://www.npmjs.com/package/hardhat-
deploy#deploymentsdeploy
    from: deployer,
    log: true,
  });

  // Uncomment below to transfer ownership
  // const yourContract = await ethers.getContract("FacilityBooking",
deployer);
  // await
yourContract.transferOwnership("0xbcC898616822C3e44154eCc64aD794790B73A3a7
");

};
module.exports.tags = ["FacilityBooking"];
```

React main file containing routing, and web3 - App.jsx

file path `/packages/react-app/src/App.jsx`
NOTE: This project is aming to become one of the sample project of Scaffold-ETH framework. Thus, boiler plates and console logs are kept unless they are not conflicting with the implementation of this project.

```
import WalletConnectProvider from "@walletconnect/web3-provider";
//import Torus from "@toruslabs/torus-embed"
import WalletLink from "walletlink";
import { Alert, Button, Col, Menu, Row } from "antd";
import "antd/dist/antd.css";
import React, { useCallback, useEffect, useState } from "react";
import { BrowserRouter, Link, Route, Switch } from "react-router-dom";
import Web3Modal from "web3modal";
import "./App.css";
import { Account, Contract, Faucet, GasGauge, Header, Ramp, ThemeSwitch }
from "./components";
import { INFURA_ID, NETWORK, NETWORKS } from "./constants";
import { Transactor } from "./helpers";
import {
  useBalance,
  useContractLoader,
  useContractReader,
  useGasPrice,
  useOnBlock,
  useUserProviderAndSigner,
} from "eth-hooks";
import { useEventListener } from "eth-hooks/events/useEventListener";
import { useExchangeEthPrice } from "eth-hooks/dapps/dex";
// import Hints from "./Hints";
import { BookingUI, AdminUI, Hints, Subgraph } from "./views";

// contracts
import deployedContracts from "./contracts/hardhat_contracts.json";
import externalContracts from "./contracts/external_contracts";

import { useContractConfig } from "./hooks";
import Portis from "@portis/web3";
import Fortmatic from "fortmatic";
import Authereum from "authereum";

const { ethers } = require("ethers");
/*
    Welcome to 🏗 scaffold-eth !

    Code:
    https://github.com/scaffold-eth/scaffold-eth

    Support:
    https://t.me/joinchat/KByvmRe5wkR-8F_zz6AjpA
    or DM @austingriffith on twitter or telegram

    You should get your own Infura.io ID and put it in `constants.js`
    (this is your connection to the main Ethereum network for ENS etc.)


    🌏 EXTERNAL CONTRACTS:
    You can also bring in contract artifacts in `constants.js`
    (and then use the `useExternalContractLoader()` hook!)
*/

/// 📡 What chain are your contracts deployed to?
const targetNetwork = NETWORKS.localhost; // <------- select your target
frontend network (localhost, rinkeby, xdai, mainnet)

// 😬 Sorry for all the console logging
const DEBUG = true;
const NETWORKCHECK = true;

// 🛰 providers
if (DEBUG) console.log("📡 Connecting to Mainnet Ethereum");
// const mainnetProvider = getDefaultProvider("mainnet", { infura:
INFURA_ID, etherscan: ETHERSCAN_KEY, quorum: 1 });
// const mainnetProvider = new InfuraProvider("mainnet",INFURA_ID);
//
// attempt to connect to our own scaffold eth rpc and if that fails fall
back to infura...
// Using StaticJsonRpcProvider as the chainId won't change see
https://github.com/ethers-io/ethers.js/issues/901
const scaffoldEthProvider = navigator.onLine
  ? new
ethers.providers.StaticJsonRpcProvider("https://rpc.scaffoldeth.io:48544")
  : null;
const poktMainnetProvider = navigator.onLine
  ? new ethers.providers.StaticJsonRpcProvider(
      "https://eth-
mainnet.gateway.pokt.network/v1/lb/611156b4a585a20035148406",
    )
  : null;
const mainnetInfura = navigator.onLine
  ? new
ethers.providers.StaticJsonRpcProvider("https://mainnet.infura.io/v3/" +
INFURA_ID)
  : null;
// ( ⚠ Getting "failed to meet quorum" errors? Check your INFURA_ID
// 🏠 Your local provider is usually pointed at your local blockchain
const localProviderUrl = targetNetwork.rpcUrl;
// as you deploy to other networks you can set
REACT_APP_PROVIDER=https://dai.poa.network in packages/react-app/.env
const localProviderUrlFromEnv = process.env.REACT_APP_PROVIDER ?
process.env.REACT_APP_PROVIDER : localProviderUrl;
if (DEBUG) console.log("🏠 Connecting to provider:",
localProviderUrlFromEnv);
const localProvider = new
ethers.providers.StaticJsonRpcProvider(localProviderUrlFromEnv);

// 🔭 block explorer URL
const blockExplorer = targetNetwork.blockExplorer;

// Coinbase walletLink init
const walletLink = new WalletLink({
  appName: "coinbase",
});

// WalletLink provider
const walletLinkProvider =
walletLink.makeWeb3Provider(`https://mainnet.infura.io/v3/${INFURA_ID}`,
1);

// Portis ID: 6255fb2b-58c8-433b-a2c9-62098c05ddc9
/*
```

```
  Web3 modal helps us "connect" external wallets:
*/
const web3Modal = new Web3Modal({
  network: "mainnet", // Optional. If using WalletConnect on xDai, change
network to "xdai" and add RPC info below for xDai chain.
  cacheProvider: true, // optional
  theme: "light", // optional. Change to "dark" for a dark theme.
  providerOptions: {
    walletconnect: {
      package: WalletConnectProvider, // required
      options: {
        bridge: "https://polygon.bridge.walletconnect.org",
        infuraId: INFURA_ID,
        rpc: {
          1: `https://mainnet.infura.io/v3/${INFURA_ID}`, // mainnet //
For more WalletConnect providers: https://docs.walletconnect.org/quick-
start/dapps/web3-provider#required
          42: `https://kovan.infura.io/v3/${INFURA_ID}`,
          100: "https://dai.poa.network", // xDai
        },
      },
    },
    portis: {
      display: {
        logo: "https://user-
images.githubusercontent.com/9419140/128913641-d025bc0c-e059-42de-a57b-
422f196867ce.png",
        name: "Portis",
        description: "Connect to Portis App",
      },
      package: Portis,
      options: {
        id: "6255fb2b-58c8-433b-a2c9-62098c05ddc9",
      },
    },
    fortmatic: {
      package: Fortmatic, // required
      options: {
        key: "pk_live_5A7C91B2FC585A17", // required
      },
    },
    // torus: {
    //   package: Torus,
    //   options: {
    //     networkParams: {
    //       host: "https://localhost:8545", // optional
    //       chainId: 1337, // optional
    //       networkId: 1337 // optional
    //     },
    //     config: {
    //       buildEnv: "development" // optional
    //     },
    //   },
    // },
    "custom-walletlink": {
      display: {
        logo: "https://play-
lh.googleusercontent.com/PjoJoG27miSglVBXoXrxBSLveV6e3EeBPpNY55aiUUBM9Q1RC
ETKCOqdOkX2ZydqVf0",
        name: "Coinbase",
        description: "Connect to Coinbase Wallet (not Coinbase App)",
      },
      package: walletLinkProvider,
      connector: async (provider, _options) => {
        await provider.enable();
        return provider;
      },
    },
    authereum: {
      package: Authereum, // required
    },
  },
});


export const OwnerContext = React.createContext();

function App(props) {
  const mainnetProvider =
    poktMainnetProvider && poktMainnetProvider._isProvider
      ? poktMainnetProvider
      : scaffoldEthProvider && scaffoldEthProvider._network
      ? scaffoldEthProvider
      : mainnetInfura;

  const [injectedProvider, setInjectedProvider] = useState();
  const [address, setAddress] = useState();

  const logoutOfWeb3Modal = async () => {
    await web3Modal.clearCachedProvider();
    if (injectedProvider && injectedProvider.provider && typeof
injectedProvider.provider.disconnect == "function") {
      await injectedProvider.provider.disconnect();
    }
    setTimeout(() => {
      window.location.reload();
    }, 1);
  };

  /* 💵 This hook will get the price of ETH from 🦄 Uniswap: */
  const price = useExchangeEthPrice(targetNetwork, mainnetProvider);

  /* 🔥 This hook will get the price of Gas from ⛽ EtherGasStation */
  const gasPrice = useGasPrice(targetNetwork, "fast");
  // Use your injected provider from 🦊 Metamask or if you don't have it
then instantly generate a 🔥 burner wallet.
  const userProviderAndSigner = useUserProviderAndSigner(injectedProvider,
localProvider);
```

```
  const userSigner = userProviderAndSigner.signer;

  useEffect(() => {
    async function getAddress() {
      if (userSigner) {
        const newAddress = await userSigner.getAddress();
        setAddress(newAddress);
      }
    }
    getAddress();
  }, [userSigner]);

  // You can warn the user if you would like them to be on a specific
network
  const localChainId = localProvider && localProvider._network &&
localProvider._network.chainId;
  const selectedChainId =
    userSigner && userSigner.provider && userSigner.provider._network &&
userSigner.provider._network.chainId;

  // For more hooks, check out 🔗 eth-hooks at:
https://www.npmjs.com/package/eth-hooks

  // The transactor wraps transactions and provides notificiations
  const tx = Transactor(userSigner, gasPrice);

  // Faucet Tx can be used to send funds from the faucet
  const faucetTx = Transactor(localProvider, gasPrice);

  // 🏗 scaffold-eth is full of handy hooks like this one to get your
balance:
  const yourLocalBalance = useBalance(localProvider, address);

  // Just plug in different 🛰 providers to get your balance on different
chains:
  const yourMainnetBalance = useBalance(mainnetProvider, address);

  // const contractConfig = useContractConfig();

  const contractConfig = { deployedContracts: deployedContracts || {},
externalContracts: externalContracts || {} };

  // Load in your local 📝 contract and read a value from it:
  const readContracts = useContractLoader(localProvider, contractConfig);

  // If you want to make 🔐 write transactions to your contracts, use the
userSigner:
  const writeContracts = useContractLoader(userSigner, contractConfig,
localChainId);

  // EXTERNAL CONTRACT EXAMPLE:
  //
  // If you want to bring in the mainnet DAI contract it would look like:
  const mainnetContracts = useContractLoader(mainnetProvider,
contractConfig);

  // If you want to call a function on a new block
  useOnBlock(mainnetProvider, () => {
    console.log(`⛓ A new mainnet block is here:
${mainnetProvider._lastBlockNumber}`);
  });

  // Then read your DAI balance like:
  const myMainnetDAIBalance = useContractReader(mainnetContracts, "DAI",
"balanceOf", [
    "0x34aA3F359A9D614239015126635CE7732c18fDF3",
  ]);

  /*
    Event listener for FacilityBooking.sol
  */
  // 📟 Listen for broadcast events
  const setNewBookingEvents = useEventListener(readContracts,
"FacilityBooking", "NewBooking", localProvider, 1);

  const setCancelBookingEvents = useEventListener(readContracts,
"FacilityBooking", "CancelBooking", localProvider, 1);

  const setOwnershipTransferredEvents = useEventListener(readContracts,
"FacilityBooking", "OwnershipTransferred", localProvider, 1);

  const [owner, setOwner] = useState();
  const value = {
    owner,
    setOwner,
  }
  useEffect(async () => {
    async function getOwnerAddress() {
      if (readContracts.FacilityBooking) {
        const newAddress = await readContracts.FacilityBooking.getOwner();
        setOwner(newAddress);
      }
    }
    getOwnerAddress();
  }, [setOwnershipTransferredEvents]);

  const [buildingName, setBuildingName] = useState();
  const [facilityName, setFacilityName] = useState();
  const [fees, setFees] = useState();
  const updateFees = async (_facility) => {
    if (readContracts.FacilityBooking) {
      const result = await
readContracts.FacilityBooking.getFees(_facility);
      const etherBalance = ethers.utils.formatEther(result);
      parseFloat(etherBalance).toFixed(2);
      const floatBalance = parseFloat(etherBalance);
      return floatBalance;
    }
```

```
      return 0;
    }

    /*
      Update contract parameters from FacilityBooking.sol
    */
    useEffect(async () => {
      async function getContractParams() {
        if (readContracts.FacilityBooking) {
          const building = await
  readContracts.FacilityBooking.getBuildingName();
          const facility = await
  readContracts.FacilityBooking.getFacilityNames();
          const fees = await updateFees("tennis");
          setBuildingName(building);
          setFacilityName(facility);
          setFees(fees);
        }
      }
      getContractParams();
    }, [readContracts]);

    /*
    const addressFromENS = useResolveName(mainnetProvider,
  "austingriffith.eth");
    console.log("🏷 Resolved austingriffith.eth as:",addressFromENS)
    */

    //
    // 🧫 DEBUG 👨🏻‍🔬
    //
    // https://reffect.co.jp/react/react-useeffect-understanding#useEffect
    useEffect(() => {
      if (
        DEBUG &&
        mainnetProvider &&
        address &&
        selectedChainId &&
        yourLocalBalance &&
        yourMainnetBalance &&
        readContracts &&
        writeContracts &&
        mainnetContracts
      ) {
        console.log("_____________________________________ 🏗 scaffold-eth
  _____________________________________");
        console.log("🌎 mainnetProvider", mainnetProvider);
        console.log("🏠 localChainId", localChainId);
        console.log("👩‍💼 selected address:", address);
        console.log("🕵🏻‍♂️ selectedChainId:", selectedChainId);
        console.log("💵 yourLocalBalance", yourLocalBalance ?
  ethers.utils.formatEther(yourLocalBalance) : "...");
        console.log("💵 yourMainnetBalance", yourMainnetBalance ?
  ethers.utils.formatEther(yourMainnetBalance) : "...");
```

```
      console.log("📝 readContracts", readContracts);
      console.log("🌍 DAI contract on mainnet:", mainnetContracts);
      console.log("💵 yourMainnetDAIBalance", myMainnetDAIBalance);
      console.log("🔐 writeContracts", writeContracts);
    }
  }, [
    mainnetProvider,
    address,
    selectedChainId,
    yourLocalBalance,
    yourMainnetBalance,
    readContracts,
    writeContracts,
    mainnetContracts,
  ]);

  let networkDisplay = "";
  if (NETWORKCHECK && localChainId && selectedChainId && localChainId !==
selectedChainId) {
    const networkSelected = NETWORK(selectedChainId);
    const networkLocal = NETWORK(localChainId);
    if (selectedChainId === 1337 && localChainId === 31337) {
      networkDisplay = (
        <div style={{ zIndex: 2, position: "absolute", right: 0, top: 60,
padding: 16 }}>
          <Alert
            message="⚠ Wrong Network ID"
            description={
              <div>
                You have <b>chain id 1337</b> for localhost and you need
to change it to <b>31337</b> to work with
                HardHat.
                <div>(MetaMask -&gt; Settings -&gt; Networks -&gt; Chain
ID -&gt; 31337)</div>
              </div>
            }
            type="error"
            closable={false}
          />
        </div>
      );
    } else {
      networkDisplay = (
        <div style={{ zIndex: 2, position: "absolute", right: 0, top: 60,
padding: 16 }}>
          <Alert
            message="⚠ Wrong Network"
            description={
              <div>
                You have <b>{networkSelected && networkSelected.name}</b>
selected and you need to be on{" "}
                <Button
                  onClick={async () => {
                    const ethereum = window.ethereum;
                    const data = [
                      {
                        chainId: "0x" +
targetNetwork.chainId.toString(16),
                        chainName: targetNetwork.name,
                        nativeCurrency: targetNetwork.nativeCurrency,
                        rpcUrls: [targetNetwork.rpcUrl],
                        blockExplorerUrls: [targetNetwork.blockExplorer],
                      },
                    ];
                    console.log("data", data);

                    let switchTx;
                    // https://docs.metamask.io/guide/rpc-api.html#other-
rpc-methods
                    try {
                      switchTx = await ethereum.request({
                        method: "wallet_switchEthereumChain",
                        params: [{ chainId: data[0].chainId }],
                      });
                    } catch (switchError) {
                      // not checking specific error code, because maybe
we're not using MetaMask
                      try {
                        switchTx = await ethereum.request({
                          method: "wallet_addEthereumChain",
                          params: data,
                        });
                      } catch (addError) {
                        // handle "add" error
                      }
                    }

                    if (switchTx) {
                      console.log(switchTx);
                    }
                  }}
                >
                  <b>{networkLocal && networkLocal.name}</b>
                </Button>
              </div>
            }
            type="error"
            closable={false}
          />
        </div>
      );
    }
  } else {
    networkDisplay = (
      <div style={{ zIndex: -1, position: "absolute", right: 154, top: 28,
padding: 16, color: targetNetwork.color }}>
        {targetNetwork.name}
      </div>
    );
  }

  const loadWeb3Modal = useCallback(async () => {
    const provider = await web3Modal.connect();
    setInjectedProvider(new ethers.providers.Web3Provider(provider));

    provider.on("chainChanged", chainId => {
      console.log(`chain changed to ${chainId}! updating providers`);
      setInjectedProvider(new ethers.providers.Web3Provider(provider));
    });

    provider.on("accountsChanged", () => {
      console.log(`account changed!`);
      setInjectedProvider(new ethers.providers.Web3Provider(provider));
    });

    // Subscribe to session disconnection
    provider.on("disconnect", (code, reason) => {
      console.log(code, reason);
      logoutOfWeb3Modal();
    });
  }, [setInjectedProvider]);

  useEffect(() => {
    if (web3Modal.cachedProvider) {
      loadWeb3Modal();
    }
  }, [loadWeb3Modal]);

  const [route, setRoute] = useState();
  useEffect(() => {
    setRoute(window.location.pathname);
  }, [setRoute]);

  let faucetHint = "";
  const faucetAvailable = localProvider && localProvider.connection &&
targetNetwork.name.indexOf("local") !== -1;

  const [faucetClicked, setFaucetClicked] = useState(false);
  if (
    !faucetClicked &&
    localProvider &&
    localProvider._network &&
    localProvider._network.chainId === 31337 &&
    yourLocalBalance &&
    ethers.utils.formatEther(yourLocalBalance) <= 0
  ) {
    faucetHint = (
      <div style={{ padding: 16 }}>
        <Button
          type="primary"
          onClick={() => {
            faucetTx({
              to: address,
              value: ethers.utils.parseEther("0.01"),
            });
            setFaucetClicked(true);
          }}
        >
          💰 Grab funds from the faucet ⛽
        </Button>
      </div>
    );
  }

  /*
    Main Structure
  */
  return (
    <div className="App">
      <Header />
      {networkDisplay}
      <BrowserRouter>
        <Menu style={{ textAlign: "center" }} selectedKeys={[route]}
mode="horizontal">
          {/* Resident page */}
          <Menu.Item key="/">
            <Link
              onClick={() => {
                setRoute("/");
              }}
              to="/"
            >
              For Residents
            </Link>
          </Menu.Item>
          {/* Admin page */}
          <Menu.Item key="/admin">
            <Link
              onClick={() => {
                setRoute("/admin");
              }}
              to="/admin"
            >
              For Admin
            </Link>
          </Menu.Item>
        </Menu>

        <Switch>
          {/* parameters for resident page*/}
          <Route exact path="/">
            <BookingUI
              buildingName={buildingName}
              facilityName={facilityName}
              address={address}
              userSigner={userSigner}
              mainnetProvider={mainnetProvider}
              localProvider={localProvider}
              yourLocalBalance={yourLocalBalance}
              tx={tx}
              writeContracts={writeContracts}
              readContracts={readContracts}
              setNewBookingEvents={setNewBookingEvents}
              setCancelBookingEvents={setCancelBookingEvents}
              fees={fees}
            />
          </Route>
          {/* parameters for admin page*/}
          <Route exact path="/admin">
            <OwnerContext.Provider value={value}>
              <AdminUI
                buildingName={buildingName}
                facilityName={facilityName}
                address={address}
                userSigner={userSigner}
                mainnetProvider={mainnetProvider}
                localProvider={localProvider}
                price={price}
                yourLocalBalance={yourLocalBalance}
                tx={tx}
                writeContracts={writeContracts}
                readContracts={readContracts}
                setNewBookingEvents={setNewBookingEvents}
                setCancelBookingEvents={setCancelBookingEvents}
                setOwnershipTransferredEvents=
{setOwnershipTransferredEvents}
                fees={fees}
              />
            </OwnerContext.Provider>
          </Route>
        </Switch>
      </BrowserRouter>

      <ThemeSwitch />

      {/* 👨‍💼 Your account is in the top right with a wallet at connect
options */}
      <div style={{ position: "fixed", textAlign: "right", right: 0, top:
0, padding: 10 }}>
        <Account
          address={address}
          localProvider={localProvider}
          userSigner={userSigner}
          mainnetProvider={mainnetProvider}
          price={price}
          web3Modal={web3Modal}
          loadWeb3Modal={loadWeb3Modal}
          logoutOfWeb3Modal={logoutOfWeb3Modal}
          blockExplorer={blockExplorer}
        />
        {faucetHint}
      </div>

      {/* 🗺 Extra UI like gas price, eth price, faucet, and support: */}
      <div style={{ position: "fixed", textAlign: "left", left: 0, bottom:
20, padding: 10 }}>
        <Row align="middle" gutter={[4, 4]}>
          <Col span={8}>
            <Ramp price={price} address={address} networks={NETWORKS} />
          </Col>

          <Col span={8} style={{ textAlign: "center", opacity: 0.8 }}>
            <GasGauge gasPrice={gasPrice} />
          </Col>
          <Col span={8} style={{ textAlign: "center", opacity: 1 }}>
            <Button
              onClick={() => {
                window.open("https://t.me/joinchat/KByvmRe5wkR-
8F_zz6AjpA");
              }}
              size="large"
              shape="round"
            >
              <span style={{ marginRight: 8 }} role="img" aria-
label="support">
                💬
              </span>
              Support
            </Button>
          </Col>
        </Row>

        <Row align="middle" gutter={[4, 4]}>
          <Col span={24}>
            {
              /*  if the local provider has a signer, let's show the
faucet:  */
              faucetAvailable ? (
                <Faucet localProvider={localProvider} price={price}
ensProvider={mainnetProvider} />
              ) : (
                ""
              )
            }
          </Col>
        </Row>
      </div>
    </div>
  );
}

export default App;
```

Resident page - BookingUI.jsx

file path `/packages/react-app/src/views/BookingUI.jsx`

```
import { utils } from "ethers";
import { Button, Divider, List, Tabs } from "antd";
import React, { useState, useEffect } from "react";
import moment from "moment";
import DatePicker from "react-datepicker";

import "react-datepicker/dist/react-datepicker.css";

export default function BookingUI({
  buildingName,
  facilityName,
  setNewBookingEvents,
  setCancelBookingEvents,
  address,
  tx,
  readContracts,
  writeContracts,
  fees,
}) {
  /*
    States
  */
  const { TabPane } = Tabs;
  const defaultTime = new Date().setHours(9, 0);
  const [startDate, setStartDate] = useState(defaultTime);
  const [bookingStartDisplay, setBookingStartDisplay] =
useState(defaultTime);
  const [bookingEndDisplay, setBookingEndDisplay] = useState(defaultTime);
  const [newBookingStart, setNewBookingStart] = useState(new
Date(defaultTime).getTime());
  const [newBookingEnd, setNewBookingEnd] = useState(new
Date(defaultTime).getTime());
  const [endTimeExceedStartError, setEndTimeExceedStartError] =
useState(false);
  const [oldDateSelectedError, setOldDateSelectedError] = useState(false);
  const [refundAmount, setRefundAmount] = useState(0.0);

  const dateToTime = new Map(); // Used to identify booked slots on the
calendar
  const addressToBooking = new Map(); // Used to identify upcoing bookings

  /*
    Functions
  */
  // Set all booking slots in dateToTime
  // and allUpcomingBooking from NewBookingEvents
  const refreshUpcomingBooking = () => setNewBookingEvents.map(item => {
    const bookerAddress = item.args[1];
    const date = moment(parseInt(item.args[2]._hex)).format("YYYYMMDD");
    const start = parseInt(item.args[2]._hex);
    const end = parseInt(item.args[3]._hex);
    if (!dateToTime.has(date)) {
      dateToTime.set(date, [[start, end]]);
    } else {
      let existingDates = dateToTime.get(date);
      existingDates = existingDates.filter(time => !(time[0] < end &&
time[1] > start));
      dateToTime.set(date, [...existingDates, [start, end]]);
    }

    if (bookerAddress != address) {
      return;
    }

    if (!addressToBooking.has(bookerAddress)) {
      addressToBooking.set(bookerAddress, [[start, end]]);
    } else {
      let existingDates = addressToBooking.get(bookerAddress);
      existingDates = existingDates.filter(time => !(time[0] < end &&
time[1] > start));
      addressToBooking.set(bookerAddress, [...existingDates, [start,
end]]);
    }
  });
  refreshUpcomingBooking();

  // Remove canceled booking from dateToTime and allUpcomingBooking
variables
  const refrechCanceledBooking = () => setCancelBookingEvents.map(item =>
{
    const bookerAddress = item.args[1];
    const date = moment(parseInt(item.args[2]._hex)).format("YYYYMMDD");
    const start = parseInt(item.args[2]._hex);
    const end = parseInt(item.args[3]._hex);

    if (dateToTime.has(date)) {
      let existingDates = dateToTime.get(date);
      existingDates = existingDates.filter(time => !(time[0] < end &&
time[1] > start));
      dateToTime.set(date, [...existingDates]);
    }

    if (bookerAddress != address) {
      return;
    }

    if (addressToBooking.has(bookerAddress)) {
      let existingDates = addressToBooking.get(bookerAddress);
      existingDates = existingDates.filter(time => !(time[0] < end &&
time[1] > start))
```

```
        addressToBooking.set(bookerAddress, [...existingDates]);
      }
    });
    refrechCanceledBooking();

    const getRefundAmount = async () => {
      if (readContracts.FacilityBooking) {
        let result = await
  readContracts.FacilityBooking.getRefundAmount(address);
        const etherBalance = utils.formatEther(result);
        parseFloat(etherBalance).toFixed(2);
        const floatBalance = parseFloat(etherBalance);

        setRefundAmount(floatBalance);
      }
      return 0;
    }

    // Filter unavailable days
    const filterPassedDate = time => {
      const day = moment(time).day();
      return day !== 0 && day !== 6;
    };

    // Filter unavailable times
    const currentTime = new Date();
    const filterPassedTime = time => {
      if (!filterPassedDate(time)) {
        return false;
      }
      if (time < currentTime) {
        return false;
      }
      const start = new Date(time).setHours(9, 0, 0);
      const end = new Date(time).setHours(22, 0, 0);
      if (time < start || end < time) {
        return false;
      }

      const date = moment(time).format("YYYYMMDD");
      const selectedTimestamp = time.getTime();

      if (dateToTime.has(date)) {
        const result = dateToTime.get(date).filter(item => {
          return item[0] <=  selectedTimestamp  && selectedTimestamp <
  item[1];
        });
        return result.length > 0 ? false : true;
      }
      return true;
    };

    /*
      Update State
    */
    // Refresh the refundable amount of the current user
    useEffect(async () => {
      getRefundAmount();
    }, [setNewBookingEvents, setCancelBookingEvents]);

    /*
      Error messages
    */
    const EndTimeExceedStartError = () => (
      <p style={{ color: "red" }}>The start time cannot exceed the end time!
  </p>
    )

    const OldDateSelectedError = () => (
      <p style={{color: "red"}}>You chose a wrong pair of the start and the
  end slot. Booking time must be future!</p>
    );

    /*
      Structure
    */
    return (
      <div style={{ paddingBottom: 20 }}>
        <div style={{ border: "1px solid #cccccc", padding: 16, width: 600,
  margin: "auto", marginTop: 64 }}>
          <h2>{buildingName}</h2>
          <div style={{ fontSize: 20 }}>Facility: {facilityName}</div>
          <div style={{ fontSize: 15 }}>booking fee per slot: {fees}
  ETH</div>
          <Divider />
          <h2>Select date and time</h2>
          <div>
            <DatePicker
              inline
              selected={startDate}
              minDate={defaultTime}
              filterDate={filterPassedDate}
              onChange={date => {
                setOldDateSelectedError(false);
                setStartDate(date);
                const start = new Date(date).setHours(9, 0, 0);
                const end = new Date(date).setHours(9, 30, 0);
                setBookingStartDisplay(start);
                setBookingEndDisplay(end);
                setNewBookingStart(new Date(start).getTime());
                setNewBookingEnd(new Date(end).getTime());
              }}
              dateFormat="yyyy-MM-dd hh:mm"
              shouldCloseOnSelect={false}
            >
              <div style={{ color: "grey" }}>**Gray color are not avaiable**
  </div>
            </DatePicker>
          </div>
          <div style={{ display: "flex", alignItems: "center",
  justifyContent: "center", marginTop: 20 }}>
            <div>
              <div>
                Start
              </div>
              <div>
                <DatePicker
                  selected={bookingStartDisplay}
                  showTimeSelect
                  showTimeSelectOnly
                  filterTime={filterPassedTime}
                  timeIntervals={30}
                  onChange={date => {
                    setOldDateSelectedError(false);
                    setBookingStartDisplay(date);
                    setNewBookingStart(new Date(date).getTime());
                  }}
                  dateFormat="hh:mm aa"
                />
              </div>
            </div>
            <div style={{marginLeft: 5}}>
              <div>
                End
              </div>
              <div>
                <DatePicker
                  selected={bookingEndDisplay}
                  showTimeSelect
                  showTimeSelectOnly
                  filterTime={filterPassedTime}
                  timeIntervals={30}
                  onChange={date => {
                    setOldDateSelectedError(false);
                    setBookingEndDisplay(date);
                    setNewBookingEnd(new Date(date).getTime());
                  }}
                  dateFormat="hh:mm aa"
                />
              </div>
            </div>
          </div>
          <div>
            { endTimeExceedStartError ? <EndTimeExceedStartError/> : null }
            { oldDateSelectedError ? <OldDateSelectedError/> : null }
          </div>
          <div style={{ margin: 8 }}>
            <Button
              style={{ marginTop: 8 }}
              onClick={async () => {
                if (bookingStartDisplay < currentTime || bookingEndDisplay <
  currentTime) {
```

```
                  setOldDateSelectedError(true);
                  return;
                }
                if (bookingStartDisplay >= bookingEndDisplay) {
                  setEndTimeExceedStartError(true);
                  return;
                }
                setOldDateSelectedError(false);
                setEndTimeExceedStartError(false);

                // Here's booking fee calculatioon logic
                // calculate the number of slots from start to end time
which are passed in millisecond
                const thirtyMinsInMsec =  30 * 60 * 1000;
                const numOfSlots = (newBookingEnd - newBookingStart) /
thirtyMinsInMsec; // end, start time passed in millisecond
                const feePerSlot = 1000000000000000; // in wei(~=0.001
ether)
                const fee = numOfSlots * feePerSlot;
                const result = tx(
                  writeContracts.FacilityBooking.register(newBookingStart,
newBookingEnd, "tennis", {
                    value: fee,
                  }),
                  update => {
                  console.log("📡 Transaction Update:", update);
                  if (update && (update.status === "confirmed" ||
update.status === 1)) {
                      console.log(" 🍾 Transaction " + update.hash + "
finished!");
                      console.log(
                        " ⛽️ " +
                          update.gasUsed +
                          "/" +
                        (update.gasLimit || update.gas) +
                          " @ " +
                          parseFloat(update.gasPrice) / 1000000000 +
                        " gwei",
                      );
                    }
                  },
                );
                console.log("awaiting metamask/web3 confirm result...",
await result);
              }}
            >
              Book
            </Button>
          </div>
        </div>

        <div style={{ width: 600, margin: "auto", marginTop: 32,
paddingBottom: 32 }}>
          <Tabs defaultActiveKey="1" type="card" size="small">
```

```
              <TabPane tab="Upcoming" key="1">
                <h2>Your Upcoming Booking</h2>
                <List
                  bordered
                  dataSource={addressToBooking.get(address)}
                  renderItem={date_time => {
                    if (date_time[0] < new Date().getTime()) {
                      return;
                    }
                    const start = moment(date_time[0]).format("YYYY/MM/DD
ddd hh:mm a");
                    const end = moment(date_time[1]).format("hh:mm a");
                    return (
                      <List.Item>
                        {start} - {end}
                        <Button
                          style={{ marginLeft: 10}}
                          onClick={async () => {
                            const result =
tx(writeContracts.FacilityBooking.cancelBooking(date_time[0],
date_time[1], "tennis"),
                              update => {
                              console.log("📡 Transaction Update:", update);
                              if (update && (update.status === "confirmed"
|| update.status === 1)) {
                                  console.log(" 🍾 Transaction " +
update.hash + " finished!");
                                  console.log(
                                    " ⛽ " +
                                      update.gasUsed +
                                      "/" +
                                    (update.gasLimit || update.gas) +
                                      " @ " +
                                      parseFloat(update.gasPrice) /
1000000000 +
                                    " gwei",
                                  );
                                }
                              },
                            );
                            console.log("CANCEL awaiting metamask/web3
confirm result...", await result);
                          }}
                        >Cancel</Button>
                      </List.Item>
                    );
                  }}
                />
              </TabPane>
              <TabPane tab="Refund" key="2">
                <h2>Refund Your Canceled Booking Fee</h2>
                <div>Total amount: {refundAmount} ETH</div>
                <div style={{ margin: 8 }}>
                  <Button
                    style={{ marginTop: 8 }}
                    onClick={async () => {
                      const result =
tx(writeContracts.FacilityBooking.refund(), update => {
                        console.log("📡 Transaction Update:", update);
                        if (update && (update.status === "confirmed" ||
update.status === 1)) {
                          console.log(" 🍾 Transaction " + update.hash + "
finished!");
                          console.log(
                            " ⛽️ " +
                              update.gasUsed +
                              "/" +
                              (update.gasLimit || update.gas) +
                              " @ " +
                              parseFloat(update.gasPrice) / 1000000000 +
                              " gwei",
                          );
                        }
                      });
                      console.log("Transfer ownership, awaiting
metamask/web3 confirm result...", await result);
                    }}
                    >
                    Refund
                  </Button>
                </div>
              </TabPane>
              <TabPane tab="History" key="3">
                <h2>History</h2>
                <List
                  bordered
                  dataSource={addressToBooking.get(address)}
                  renderItem={date_time => {
                    const start = moment(date_time[0]).format("YYYY/MM/DD
ddd hh:mm a");
                    const end = moment(date_time[1]).format("hh:mm a");
                    return (
                      <List.Item>
                        {start} - {end}
                      </List.Item>
                    );
                  }}
                />
              </TabPane>
            </Tabs>
          </div>
        </div>
    );
}
```

Admin page - AdminUI.jsx

file path `/packages/react-app/src/views/AdminUI.jsx`

```
import { Button, Divider, Input, List, Tabs } from "antd";
import React, { useState, useEffect, useContext } from "react";
import { Address, Balance } from "../components";
import moment from "moment";
import DatePicker from "react-datepicker";
import { OwnerContext } from "../App";

import "react-datepicker/dist/react-datepicker.css";

// This view provide admin features
export default function AdminUI({
  buildingName,
  facilityName,
  setNewBookingEvents,
  setCancelBookingEvents,
  address,
  localProvider,
  price,
  tx,
  readContracts,
  writeContracts,
  fees,
}) {
  /*
    States
  */
  const { TabPane } = Tabs;
  const defaultTime = new Date().setHours(9, 0);
  const [startDate, setStartDate] = useState(defaultTime);
  const [bookingStartDisplay, setBookingStartDisplay] =
useState(defaultTime);
  const [bookingEndDisplay, setBookingEndDisplay] = useState(defaultTime);
  const [newBookingStart, setNewBookingStart] = useState(new
Date(defaultTime).getTime());
  const [newBookingEnd, setNewBookingEnd] = useState(new
Date(defaultTime).getTime());
  const [endTimeExceedStartError, setEndTimeExceedStartError] =
useState(false);
  const [oldDateSelectedError, setOldDateSelectedError] = useState(false);
  const { owner, setOwner } = useContext(OwnerContext);
  const [newOwner, setNewOwner] = useState();
  const [latestBalance, setLatestBalance] = useState("0");
  const [history, setHistory] = useState([]);

  const dateToTime = new Map(); // Used to identify booked slots on the
calendar
  let allUpcomingBooking = []; // Used to identify upcoing bookings
  const currentTime = new Date();

  /*
    Functions
  */
  // Set all booking slots in dateToTime
  // and allUpcomingBooking from NewBookingEvents
  const refreshUpcomingBooking = () => setNewBookingEvents.map((item,
index) => {
    const address = item.args[1];
    const date = moment(parseInt(item.args[2]._hex)).format("YYYYMMDD");
    const start = parseInt(item.args[2]._hex);
    const end = parseInt(item.args[3]._hex);
    if (!dateToTime.has(date)) {
      dateToTime.set(date, [[start, end]]);
    } else {
      let existingDates = dateToTime.get(date);
      existingDates = existingDates.filter(time => !(time[0] < end &&
time[1] > start));
      dateToTime.set(date, [...existingDates, [start, end]]);
    }

    if (start < currentTime) {
      return;
    }

    let filtered = allUpcomingBooking.filter(time => !(time[1] == end &&
time[0] == start));
    allUpcomingBooking = [...filtered, [start, end, address]];
  });
  refreshUpcomingBooking();

  // Remove canceled booking from dateToTime and allUpcomingBooking
variables
  const refrechCanceledBooking = () => setCancelBookingEvents.map(item =>
{
    const date = moment(parseInt(item.args[2]._hex)).format("YYYYMMDD");
    const start = parseInt(item.args[2]._hex);
    const end = parseInt(item.args[3]._hex);

    if (dateToTime.has(date)) {
      let existingDates = dateToTime.get(date);
      existingDates = existingDates.filter(time => !(time[0] < end &&
time[1] > start));
      dateToTime.set(date, [...existingDates]);
    }

    if (start < currentTime) {
      return;
    }

    allUpcomingBooking = allUpcomingBooking.filter(time => !(time[0] < end
&& time[1] > start));
  });
  refrechCanceledBooking();

  const refreshBalance = async () => {
    if (owner && owner == address && writeContracts.FacilityBooking) {
      const response = await writeContracts.FacilityBooking.getBalance();
      setLatestBalance(BigInt(response._hex));
    }
  }

  // Fetch all booking information excluding canceled ones
  const getBookingData = async () => {
    if (readContracts.FacilityBooking) {
      let result = await
readContracts.FacilityBooking.getBookingData("tennis");
      return result;
    }
    return [];
  };

  // Filter unavailable days
  const filterPassedDate = (time) => {
    const day = moment(time).day();
    return day !== 0 && day !== 6;
  };

  // Filter unavailable times
  const current = new Date();
  const filterPassedTime = time => {
    if (!filterPassedDate(time)) {
      return false;
    }
    if (time < current) {
      return false;
    }
    const start = new Date(time).setHours(9, 0, 0);
    const end = new Date(time).setHours(22, 0, 0);
    if (time < start || end < time) {
      return false;
    }
    const date = moment(time).format("YYYYMMDD");
    const selectedTimestamp = time.getTime();

    if (dateToTime.has(date)) {
      const result = dateToTime.get(date).filter(item => {
        return item[0] <=  selectedTimestamp  && selectedTimestamp <
item[1];
      });
      return result.length > 0 ? false : true;
    }
    return true;
  };

  /*
    Update State
  */
  // Refresh the account balance at new booking or booking cancelation
  useEffect(async () => {
    refreshBalance();
  }, [setNewBookingEvents, setCancelBookingEvents]);

  /*
    Error messages
  */
  const EndTimeExceedStartError = () => (
    <p style={{ color: "red" }}>The start time cannot exceed the end time!
</p>
  )

  const OldDateSelectedError = () => (
    <p style={{color: "red"}}>You chose a wrong pair of the start and the
end slot. Booking time must be future!</p>
  )

  /*
    Structure
  */
  // If the current user is admin, then show below.
  const contractDisplay = (
    <div style={{ paddingBottom: 20 }}>
      <div style={{ border: "1px solid #cccccc", padding: 16, width: 600,
margin: "auto", marginTop: 64 }}>
        <h2>{buildingName}</h2>
        <div style={{ fontSize: 20 }}>Facility: {facilityName}</div>
        <div style={{ fontSize: 15 }}>booking fee per slot: {fees}
ETH</div>
        <Divider />
        <div>
          <div>
            Contract Address:
            <Address
              address={readContracts && readContracts.FacilityBooking ?
readContracts.FacilityBooking.address : null}
              fontSize={16}
            />
          </div>
          <div>
            Balance:
            <Balance address={readContracts &&
readContracts.FacilityBooking ? readContracts.FacilityBooking.address :
null} provider={localProvider} dollarMultiplier={price} />
          </div>
        </div>
        <Divider />
        <h2>Select date and time</h2>
        <div>
          <DatePicker
            inline
            selected={startDate}
```

```
              minDate={defaultTime}
              filterDate={filterPassedDate}
              onChange={date => {
                setOldDateSelectedError(false);
                setStartDate(date);
                const start = new Date(date).setHours(9, 0, 0);
                const end = new Date(date).setHours(9, 30, 0);
                setBookingStartDisplay(start);
                setBookingEndDisplay(end);
                setNewBookingStart(new Date(start).getTime());
                setNewBookingEnd(new Date(end).getTime());
              }}
              dateFormat="yyyy-MM-dd hh:mm"
              shouldCloseOnSelect={false}
            >
              <div style={{ color: "grey" }}>**Gray color are not avaiable**
</div>
            </DatePicker>
          </div>
          <div style={{ display: "flex", alignItems: "center",
justifyContent: "center", marginTop: 20 }}>
            <div>
              <div>
                Start
              </div>
              <div>
                <DatePicker
                  selected={bookingStartDisplay}
                  showTimeSelect
                  showTimeSelectOnly
                  filterTime={filterPassedTime}
                  timeIntervals={30}
                  onChange={date => {
                    setOldDateSelectedError(false);
                    setBookingStartDisplay(date);
                    setNewBookingStart(new Date(date).getTime());
                  }}
                  dateFormat="hh:mm aa"
                />
              </div>
            </div>
            <div style={{ marginLeft: 5 }}>
              <div>
                End
              </div>
              <div>
                <DatePicker
                  selected={bookingEndDisplay}
                  showTimeSelect
                  showTimeSelectOnly
                  filterTime={filterPassedTime}
                  timeIntervals={30}
                  onChange={date => {
                    setOldDateSelectedError(false);
                    setBookingEndDisplay(date);
                    setNewBookingEnd(new Date(date).getTime());
                  }}
                  dateFormat="hh:mm aa"
                />
              </div>
            </div>
          </div>
          <div>
            { endTimeExceedStartError ? <EndTimeExceedStartError/> : null }
            { oldDateSelectedError ? <OldDateSelectedError/> : null }
          </div>
          <div style={{ margin: 8 }}>
            <Button
              style={{ marginTop: 8 }}
              onClick={async () => {
                if (bookingStartDisplay < currentTime || bookingEndDisplay <
  currentTime) {
                  setOldDateSelectedError(true);
                  return;
                }
                if (bookingStartDisplay >= bookingEndDisplay) {
                  setEndTimeExceedStartError(true);
                  return;
                }
                setOldDateSelectedError(false);
                setEndTimeExceedStartError(false);

                const result = tx(
                  writeContracts.FacilityBooking.register(newBookingStart,
  newBookingEnd, "tennis", {
                    value: 0, // Booking fee is not required because admin
  personel is supposed to collect booking fees in some way
                  }),
                  update => {
                  console.log("📡 Transaction Update:", update);
                  if (update && (update.status === "confirmed" ||
  update.status === 1)) {
                      console.log(" 🍾 Transaction " + update.hash + "
  finished!");
                      console.log(
                        " ⛽️ " +
                          update.gasUsed +
                          "/" +
                        (update.gasLimit || update.gas) +
                          " @ " +
                          parseFloat(update.gasPrice) / 1000000000 +
                        " gwei",
                      );
                    }
                  },
                );
                console.log("New Book, awaiting metamask/web3 confirm
  result...", await result);
```

```
            }}
          >
            Book
          </Button>
        </div>
      </div>

      <div style={{ width: 600, margin: "auto", marginTop: 32,
paddingBottom: 32 }}>
        <Tabs defaultActiveKey="1" type="card" size="small">
            <TabPane tab="Upcoming" key="1">
              <h2>Upcoming Booking</h2>
              <List
                bordered
                dataSource={allUpcomingBooking}
                renderItem={date_time => {
                  const start = moment(date_time[0]).format("YYYY/MM/DD
ddd hh:mm a");
                  const end = moment(date_time[1]).format("hh:mm a");
                  return (
                    <List.Item>
                      {start} - {end} {", Resident: " + date_time[2]}
                      <Button
                        style={{ marginLeft: 10 }}
                        onClick={async () => {
                          const result =
tx(writeContracts.FacilityBooking.cancelBooking(date_time[0],
date_time[1], "tennis"),
                            update => {
                            console.log("📡 Transaction Update:", update);
                            if (update && (update.status === "confirmed"
|| update.status === 1)) {
                                console.log(" 🍾 Transaction " +
update.hash + " finished!");
                                console.log(
                                  " ⛽️ " +
                                    update.gasUsed +
                                    "/" +
                                  (update.gasLimit || update.gas) +
                                    " @ " +
                                    parseFloat(update.gasPrice) /
1000000000 +

                                  " gwei",
                                );
                              }
                            },
                          );
                          console.log("Admin Cancel, awaiting
metamask/web3 confirm result...", await result);
                        }}
                      >Cancel</Button>
                    </List.Item>
                  );
                }}
              />
            </TabPane>
            <TabPane tab="ChangeAdminAddress" key="2">
              <h2>Change Admin Address</h2>
              <div style={{ margin: 8 }}>
                <Input
                  onChange={e => {
                    setNewOwner(e.target.value);
                  }}
                />
                <Button
                  style={{ marginTop: 8 }}
                  onClick={async () => {
                    const result =
tx(writeContracts.FacilityBooking.transferOwnership(newOwner), update => {
                      console.log("📡 Transaction Update:", update);
                      if (update && (update.status === "confirmed" ||
update.status === 1)) {
                        console.log(" 🍾 Transaction " + update.hash + "
finished!");
                        console.log(
                          " ⛽️ " +
                            update.gasUsed +
                            "/" +
                            (update.gasLimit || update.gas) +
                            " @ " +
                            parseFloat(update.gasPrice) / 1000000000 +
                            " gwei",
                        );
                      }
                    });
                    console.log("Transfer ownership, awaiting
metamask/web3 confirm result...", await result);
                  }}
                  >
                  Submit
                  </Button>
              </div>
            </TabPane>
            <TabPane tab="Withdraw" key="3">
              <h2>Withdraw Contract Balance</h2>
              <div style={{ margin: 8 }}>
                <Button
                  style={{ marginTop: 8 }}
                  onClick={async () => {
                    const result =
tx(writeContracts.FacilityBooking.withdraw(), update => {
                      console.log("📡 Transaction Update:", update);
                      if (update && (update.status === "confirmed" ||
update.status === 1)) {
                        console.log(" 🍾 Transaction " + update.hash + "
finished!");
                        console.log(
                          " ⛽️ " +
                            update.gasUsed +
                            "/" +
                            (update.gasLimit || update.gas) +
                            " @ " +
                            parseFloat(update.gasPrice) / 1000000000 +
                            " gwei",
                        );
                      }
                    });
                    console.log("Transfer ownership, awaiting
metamask/web3 confirm result...", await result);
                  }}
                  >
                  Withdraw
                  </Button>
              </div>
            </TabPane>
            <TabPane tab="History" key="4">
              <h2>History</h2>
              <div style={{ margin: 8 }}>
                <Button
                  style={{ marginTop: 8 }}
                  onClick={async () => {
                    const result = await getBookingData();
                    setHistory(Object.entries(result));
                  }}
                  >
                  Export
                  </Button>
              </div>
              <List
                bordered
                dataSource={history}
                renderItem={bookinData => {
                  const start = moment(parseInt(bookinData[1]
[0])).format("YYYY/MM/DD ddd hh:mm a");
                  const end = moment(parseInt(bookinData[1]
[1])).format("hh:mm a");
                  const resident = bookinData[1][2];
                  const fee = parseInt(bookinData[1][3]);

                  return (
                    <List.Item>
                      <div>Time: {start} - {end}</div>
                      <div>Who: {resident}</div>
                      <div>Paid: {fee}</div>
                    </List.Item>
                  );
                }}
              />
            </TabPane>
          </Tabs>
      </div>
    </div>
```

```
  );

  // If the current user is NOT admin, then show below.
  const noContractDisplay = (
    <div>
      <p style={{ marginTop: 20, color: "red"}}>This is admin page. You
must use the contract owner address!</p>
    </div>
  );

  return (
    <div>
      { (address == owner) ? contractDisplay : noContractDisplay }
    </div>
  );
}
```