**SYSTEM IMPLEMENTATION**

**CHAPTER 6**

**SYSTEM IMPLEMENTATION**

System Implementation uses the structure created during architectural design and the results of system analysis to construct system elements that meet the stakeholder requirements and system requirements developed in the early life cycle phases. These system elements are then integrated to form intermediate aggregates and finally the complete system-of-interest (SoI). Implementation is the process that actually yields the lowest-level system elements in the system hierarchy (system breakdown structure). System elements are made, bought, or reused. Production involves the hardware fabrication processes of forming, removing, joining, and finishing, the software realization processes of coding and testing, or the operational procedures development processes for operators' roles.

**6.1 Modular Description**

Modular design, or "modularity in design", is a design approach that subdivides a system into smaller parts called modules, that can be independently created and then used in different systems. A modular system can be characterized by functional partitioning into discrete scalable, reusable modules; rigorous use of well-defined modular interfaces; and making use of industry standards for interfaces.

Besides reduction in cost (due to less customization, and shorter learning time), and flexibility in design, modularity offers other benefits such as augmentation (adding new solution by merely plugging in a new module), and exclusion.

Our project consists of five main modules that can be further subdivided and programmed individually.

**6.1.1 Data Collection Module**

The data collection module for Crowdfunding using Blockchain involves gathering a dataset comprising project information, funding details, and user contributions. Each project entry includes images, descriptions, funding goals, and timelines. The dataset ensures transparency and accountability in crowdfunding campaigns.

**6.1.2 Blockchain Integration Module**

The core of our system lies in integrating blockchain technology, specifically Ethereum, for secure and transparent crowdfunding operations. Ethereum was chosen for its robust transaction processing capabilities, relatively low transaction fees, active development ecosystem, and support for smart contracts.

**6.1.3 Smart Contract Module**

Solidity, the programming language for Ethereum smart contracts, is utilized to develop and manage the smart contracts governing various aspects of crowdfunding. These smart contracts handle fund collection, distribution, withdrawal, rejection, and funder reimbursements, ensuring automated and secure fund management.

**6.1.4 Crypto Wallet Integration Module**

For secure and convenient transactions, we integrate popular crypto wallets such as MetaMask and WalletConnect. These wallets enable users to interact with the blockchain, manage their funds, and participate in crowdfunding campaigns directly from their preferred wallets.

**6.1.5 Transparent Transaction Module**

Our Transaction Tracking Module ensures transparent and secure fund management within the crowdfunding platform. It records all financial activities in real-time and provides transparent reporting on fund utilization. Integrated with a voting system, it enables contributors to participate in fund release decisions, enhancing transparency and accountability.

**6.2 Program Code**

**Code for Campaign Creation**

import { useState, useEffect } from 'react';

import { useRouter } from 'next/router';

import { useContract } from '@/components/ContractProvider';

import { useWeb3Modal } from '@web3modal/wagmi/react';

import { parseEther } from 'viem';

import { Bounce, toast } from 'react-toastify';

import { Box, Typography, Button } from '@mui/material';

import { useAccount } from 'wagmi';

const CreateCampaignForm = () => {

const [formData, setFormData] = useState({

name: '',

details: '',

thumbnailUrl: '',

targetAmount: 0,

stage1Amount: 0,

stage2Amount: 0,

documentsLinks: [''],

campaignExpiryDate: '',

});

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

const { crowdFundindContract, executeContractWrite } = useContract();

const { isConnected } = useAccount();

const { open } = useWeb3Modal();

const router = useRouter();

useEffect(() => {

if (!isConnected) router.push('/');

}, [isConnected]);

const handleSubmit = async (e) => {

e.preventDefault();

// Validation logic

// ...

try {

if (!isConnected) return open();

setLoading(true);

const [result, hash] = await executeContractWrite({

address: crowdFundindContract.address,

abi: crowdFundindContract.abi,

functionName: 'createCampaign',

args: [formData.name, formData.details, formData.thumbnailUrl, [parseEther(`${formData.targetAmount}`), parseEther(`${formData.stage1Amount}`), parseEther(`${formData.stage2Amount}`)], epoch\_time, formData.documentsLinks],

});

// Display success message and redirect

// ...

setLoading(false);

} catch (err) {

setLoading(false);

console.log(err.message.details);

}

};

return (

<>

{!loading ? (

<Box mx="auto">

{/\* Form UI \*/}

</Box>

) : (

<FullScreenLoading />

)}

</>

);

};

export default CreateCampaignForm;

**Code for Explore Campaign**

import React, { useEffect, useState } from 'react';

import { Card, CardContent, Typography, CardMedia, LinearProgress, Box, FormControl, InputLabel, Select, MenuItem, TextField } from '@mui/material';

import { useContract } from '@/components/ContractProvider';

import { useWeb3Modal } from '@web3modal/wagmi/react';

import { useAccount } from 'wagmi';

import { useRouter } from 'next/navigation';

import Link from 'next/link';

import Web3 from 'web3';

const CampaignCard = () => {

const [filter, setFilter] = useState('all');

const [searchQuery, setSearchQuery] = useState('');

const [campaigns, setCampaigns] = useState([]);

const { crowdFundindContract, executeContractRead } = useContract();

const { isConnected } = useAccount();

const { open } = useWeb3Modal();

const router = useRouter();

useEffect(() => {

async function fetchData() {

try {

if (!isConnected) {

await open();

router.push(isConnected ? '/explore' : '/');

} else {

await handleGetData();

}

} catch (e) {

console.error(e);

}

}

fetchData();

}, []);

const handleGetData = async () => {

try {

const result = await executeContractRead({

address: crowdFundindContract.address,

abi: crowdFundindContract.abi,

functionName: 'getData',

args: [],

});

const filteredCampaigns = result.filter((campaign) => {

return campaign.whichStage || campaign.adminApproved;

});

const adminRejectedCampaigns = filteredCampaigns.filter((campaign) => {

return campaign.whichStage !== 'Admin Rejected';

});

setCampaigns(adminRejectedCampaigns.reverse());

} catch (e) {

console.error(e);

}

};

const handleChange = (event) => {

setFilter(event.target.value);

};

const handleSearchChange = (event) => {

setSearchQuery(event.target.value);

};

const formatDateString = (dateString) => {

// Date formatting logic

};

const CheckStatus = (campaign) => {

// Check campaign status logic

};

const filteredCampaigns = campaigns.filter((campaign) => {

// Filter logic

});

return (

<div>

{/\* Filter and search section \*/}

<div>

<TextField

id="search"

label="Search Campaigns"

variant="outlined"

value={searchQuery}

onChange={handleSearchChange}

style={{ margin: '8px', width: '100%' }}

/>

</div>

<div style={{ display: 'flex', gap: '20px' }}>

<div style={{ width: "30%" }}>

{isConnected ? (

<>

<FormControl style={{ margin: '8px', minWidth: '120px', width: '100%' }}>

<InputLabel

id="filter-label"

style={{ marginBottom: '4px', backgroundColor: '#fff', paddingLeft: '4px', paddingRight: '4px' }}

>

Filter

</InputLabel>

<Select

labelId="filter-label"

id="filter-select"

value={filter}

onChange={handleChange}

style={{ width: '100%' }}

>

{/\* Menu items \*/}

</Select>

</FormControl>

</>

) : (

<Typography variant="body1">Please connect your wallet</Typography>

)}

</div>

{/\* Campaign cards section \*/}

{/\* Display filtered campaigns \*/}

</div>

</div>

);

};

export default CampaignCard;

**Code for ContractAddress**

import { Abi, Address, getAddress } from 'viem'

import { crowdFundingAbi } from '../../abis/CrowdFunding'

import { localhost } from 'viem/chains'

export type ContractABIPair = {

ADDRESS: Address

ABI: Abi

}

// TODO: Add in contract deployments and their ABIs for each network supported

type ContractDeployments = {

CROWD\_FUNDING: ContractABIPair

}

const LOCALHOST: ContractDeployments = {

// SimpleNFT: https://sepolia.etherscan.io/address/0x1cfD246a218b35e359584979dDBeAD1f567d9C88

CROWD\_FUNDING: {

ADDRESS: getAddress('0x1E528898Dd0cFf9242ba2fb3A27c445EeD0EA6dF', localhost.id),

ABI: crowdFundingAbi,

},

}

const CONTRACTS = {

LOCALHOST,

}

export default CONTRACTS

**Code for Smart Contract**

pragma solidity ^0.8.1;

contract FundingCampaign {

address public admin;

enum CampaignStage {

NotStarted,

Stage1,

Stage2,

Completed

}

struct Campaign {

string name;

string details;

uint256[] amounts;

uint256 totalFunds;

CampaignStage status;

bool stage1Completed;

bool stage2Completed;

address creator;

bool adminApproved;

address[] contributors;

mapping(address => uint256) contributions;

mapping(address => uint256) stageContributions;

mapping(address => bool) votedForStage;

mapping(CampaignStage => uint256) stageVotesFor;

mapping(CampaignStage => uint256) stageVotesAgainst;

}

struct User {

uint256[] campaignIndices;

}

Campaign[] public campaigns;

uint256 public platformBalance = 0;

uint256 constant PLATFORM\_FEE\_PERCENTAGE = 1;

event Contribution(

address indexed contributor,

uint256 amount,

uint256 campaignIndex

);

event Withdrawal(uint256 amount, address recipient, uint256 campaignIndex);

event StageCompletion(uint256 stage, uint256 campaignIndex);

constructor() {

admin = msg.sender;

}

modifier onlyAdmin() {

require(msg.sender == admin, "Only admin can call this function");

\_;

}

function createCampaign(

string memory \_name,

string memory \_details,

uint256[] memory \_amounts,

uint256 \_expiryTimestamp,

string[] memory \_documentsLinks

) external {

require(msg.sender != address(0), "Creator address cannot be zero");

require(bytes(\_name).length > 0, "Campaign name cannot be empty");

require(\_amounts.length == 3, "Invalid amounts array length");

require(

\_amounts[0] > 0 && \_amounts[1] > 0 && \_amounts[2] > 0,

"Amounts must be greater than 0"

);

require(

\_amounts[1] + \_amounts[2] == \_amounts[0],

"Stage 2 amount + Stage 1 amount must be equal to target amount"

);

Campaign storage newCampaign = campaigns.push();

newCampaign.name = \_name;

newCampaign.details = \_details;

newCampaign.amounts = \_amounts;

newCampaign.creator = msg.sender;

newCampaign.adminApproved = false;

newCampaign.status = CampaignStage.NotStarted;

newCampaign.campaignExpiry = \_expiryTimestamp; // Assigning campaign expiry from user input

newCampaign.documentsLinks = \_documentsLinks; // Assigning document links from user input

}

function contribute(uint256 \_campaignIndex) external payable {

require(\_campaignIndex < campaigns.length, "Invalid campaign index");

require(msg.value > 0, "Contribution amount must be greater than 0");

Campaign storage campaign = campaigns[\_campaignIndex];

require(

campaign.status != CampaignStage.Completed,

"Contribution not allowed in completed campaigns"

);

// Check if the campaign is expired

require(

block.timestamp < campaign.campaignExpiry,

"Contribution not allowed, campaign has expired"

);

// Check if the campaign is in Stage 1 or Stage 2

if (

keccak256(bytes(campaign.whichStage)) == keccak256("stage1") ||

keccak256(bytes(campaign.whichStage)) == keccak256("stage2")

) {

uint256 stageIndex = getIndex(campaign.whichStage); // 2

uint256 stageAmount;

if (keccak256(bytes(campaign.whichStage)) == keccak256("stage1")) {

stageAmount = campaign.amounts[stageIndex];

} else {

stageAmount = campaign.amounts[0]; // 5000000000000000000

}

require(

campaign.totalFunds + msg.value <= stageAmount,

"Contribution exceeds stage amount"

);

campaign.totalFunds = campaign.totalFunds + msg.value;

// Update the contribution mapping based on the current stage

if (

keccak256(abi.encodePacked(campaign.whichStage)) ==

keccak256("stage1")

) {

if (campaign.stageContributions[msg.sender] == 0) {

User storage user = users[msg.sender];

user.campaignIndices.push(\_campaignIndex);

campaign.contributors.push(msg.sender);

}

campaign.stageContributions[msg.sender] += msg.value;

} else if (

keccak256(abi.encodePacked(campaign.whichStage)) ==

keccak256("stage2")

) {

if (campaign.contributions[msg.sender] == 0) {

User storage user = users[msg.sender];

user.campaignIndices.push(\_campaignIndex);

bool senderExists = false;

for (uint256 i = 0; i < campaign.contributors.length; i++) {

if (campaign.contributors[i] == msg.sender) {

senderExists = true;

break;

}

}

if (!senderExists) {

campaign.contributors.push(msg.sender);

}

}

campaign.contributions[msg.sender] += msg.value;

}

} else {

revert("Contribution not allowed in this stage");

}

emit Contribution(msg.sender, msg.value, \_campaignIndex);

}

function startStage(uint256 \_campaignIndex) external onlyAdmin {

require(\_campaignIndex < campaigns.length, "Invalid campaign index");

Campaign storage campaign = campaigns[\_campaignIndex];

require(

block.timestamp < campaign.campaignExpiry,

"Stage Change not allowed, campaign has expired"

);

if (!campaign.stage1Completed || !campaign.stage2Completed) {

if (

keccak256(abi.encodePacked(campaign.whichStage)) ==

keccak256(abi.encodePacked(""))

) {

campaign.status = CampaignStage.Stage1;

campaign.whichStage = "stage1";

} else if (

keccak256(abi.encodePacked(campaign.whichStage)) ==

keccak256(abi.encodePacked("stage1")) &&

campaign.stage1Completed

) {

campaign.status = CampaignStage.Stage2;

campaign.whichStage = "stage2";

for (uint256 i = 0; i < campaign.contributors.length; i++) {

address contributor = campaign.contributors[i];

campaign.votedForStage[contributor] = false;

}

}

} else {

revert("Campaign already completed");

}

}

// Other functions like voteForStage, getStageContributionsLength, endStageVoting, refundContributors, approveCampaign, getData, getBasicCampaignData, and getUserCampaignIndices are also part of the contract.

}  
  
 // Function to vote for a stage in a campaign

function voteForStage(uint256 \_campaignIndex, bool \_vote) external {

require(\_campaignIndex < campaigns.length, "Invalid campaign index");

Campaign storage campaign = campaigns[\_campaignIndex];

require(

campaign.status != CampaignStage.NotStarted &&

campaign.status != CampaignStage.Completed,

"Voting not allowed in this stage"

);

// Check if the caller is eligible to vote based on the campaign stage

if (campaign.whichStage == "stage1") {

require(

campaign.stageContributions[msg.sender] > 0,

"Only contributors to Stage 1 can vote"

);

} else if (campaign.whichStage == "stage2") {

require(

campaign.contributions[msg.sender] > 0,

"Only contributors to Stage 2 can vote"

);

}

// Check if the caller has not already voted

require(

!campaign.votedForStage[msg.sender],

"You have already voted for this stage"

);

// Update voting status and counts based on the vote

campaign.votedForStage[msg.sender] = true;

if (\_vote) {

campaign.stageVotesFor[campaign.status]++;

} else {

campaign.stageVotesAgainst[campaign.status]++;

}

}

// Function to get the length of stage contributions in a campaign

function getStageContributionsLength(uint256 \_campaignIndex) external view returns (uint256) {

require(\_campaignIndex < campaigns.length, "Invalid campaign index");

Campaign storage campaign = campaigns[\_campaignIndex];

uint256 count = 0;

string memory currentStage = campaign.whichStage;

// Count contributions based on the current stage

if (keccak256(bytes(currentStage)) == keccak256(bytes("stage1"))) {

for (uint256 i = 0; i < campaign.contributors.length; i++) {

address contributor = campaign.contributors[i];

if (campaign.stageContributions[contributor] > 0) {

count++;

}

}

} else if (keccak256(bytes(currentStage)) == keccak256(bytes("stage2"))) {

for (uint256 i = 0; i < campaign.contributors.length; i++) {

address contributor = campaign.contributors[i];

if (campaign.contributions[contributor] > 0) {

count++;

}

}

}

return count;

}

// Function to end voting for a campaign stage

function endStageVoting(uint256 \_campaignIndex) external onlyAdmin {

require(\_campaignIndex < campaigns.length, "Invalid campaign index");

Campaign storage campaign = campaigns[\_campaignIndex];

// Check if the campaign has reached the required votes for stage completion

uint256 votesForStage = campaign.stageVotesFor[campaign.status];

uint256 votesAgainstStage = campaign.stageVotesAgainst[campaign.status];

uint256 requiredVotes = getStageContributionsLength(\_campaignIndex) / 2;

if (votesForStage > votesAgainstStage && votesForStage >= requiredVotes) {

// Admin approves the current stage

// Implement stage completion logic here

} else {

// Refund all contributors to the current stage

refundContributors(\_campaignIndex);

}

}

**Code for Contract Provider**

import { createContext, useCallback, useContext, useEffect, useState } from 'react'

import { Abi, Address, getContract, GetContractReturnType, parseEther } from 'viem'

import { useAccount, usePublicClient, useWalletClient } from 'wagmi'

// Types

type TxHash = Address | undefined

type ContractReadArgs = { address: Address; abi: Abi; functionName: string; args?: unknown[] }

type ContractWriteArgs = { address: Address; abi: Abi; functionName: string; args: unknown[]; value?: number }

type ContractContextValues = {

executeContractRead: (args: ContractReadArgs) => Promise<unknown>

executeContractWrite: (args: ContractWriteArgs) => Promise<[unknown, TxHash]>

txSuccess: boolean

txError: string | null

resetTxNotifications: () => void

crowdFundindContract: GetContractReturnType

}

type ContractProviderProps = {

children: React.ReactNode

}

// Create context with initial values

const ContractContext = createContext<ContractContextValues>({

executeContractRead: () => Promise.resolve(undefined),

executeContractWrite: () => Promise.resolve([undefined, undefined]),

txSuccess: false,

txError: null,

resetTxNotifications: () => { },

crowdFundindContract: {} as GetContractReturnType,

})

// Context provider component

export const ContractProvider: React.FC<ContractProviderProps> = ({ children }: ContractProviderProps) => {

const [txSuccess, setTxSuccess] = useState<boolean>(false)

const [txError, setTxError] = useState<string | null>(null)

const [crowdFundindContract, setCrowdFundindContract] = useState<GetContractReturnType>({} as GetContractReturnType)

const publicClient = usePublicClient()

const { data: walletClient } = useWalletClient()

const { address: account } = useAccount()

const resetTxNotifications = () => {

setTxSuccess(false)

setTxError(null)

}

const executeContractRead = useCallback(async ({ address, abi, functionName, args }: ContractReadArgs): Promise<unknown> => {

// Contract read logic here

}, [publicClient])

const executeContractWrite = useCallback(async ({ address, abi, functionName, args, value }: ContractWriteArgs): Promise<[unknown, TxHash]> => {

// Contract write logic here

}, [publicClient, walletClient, account])

useEffect(() => {

if (walletClient && publicClient) {

setCrowdFundindContract(

getContract({

// Contract configuration

}),

)

}

}, [walletClient, publicClient])

return (

<ContractContext.Provider

value={{

crowdFundindContract,

executeContractRead,

executeContractWrite,

txSuccess,

txError,

resetTxNotifications,

}}

>

{children}

</ContractContext.Provider>

)

}

export const useContract = () => {

const context: ContractContextValues = useContext(ContractContext)

if (context === undefined) {

throw new Error('useContract must be used within a ContractProvider component.')

}

return context

}

**6.3 System Implementation Summary**

This chapter shows the implementation of the structure created during architectural design and the results of system analysis to construct system elements that meet the stakeholder requirements and system requirements developed in the early life cycle phases. It shows the segment of programming code that is used in order to implement this project.