Decentralized Finance Application

COMP5521 Group04

Zhu Dongsheng	24038342g
Li Tianye	24063117g
Yu Shaoyu	24118601g
Feng Yujie	24054389g
Xia Yishan	24045097g

Contents

01 Introduction

background, motivation,
DApp vs centralized app

02 System Design

describe the process of a
system framework diagram

03 Prototype and Technology Stack

frontend(react), backend

04 Demonstration and Conclusion

showcase the features, functionalities

01

Introduction

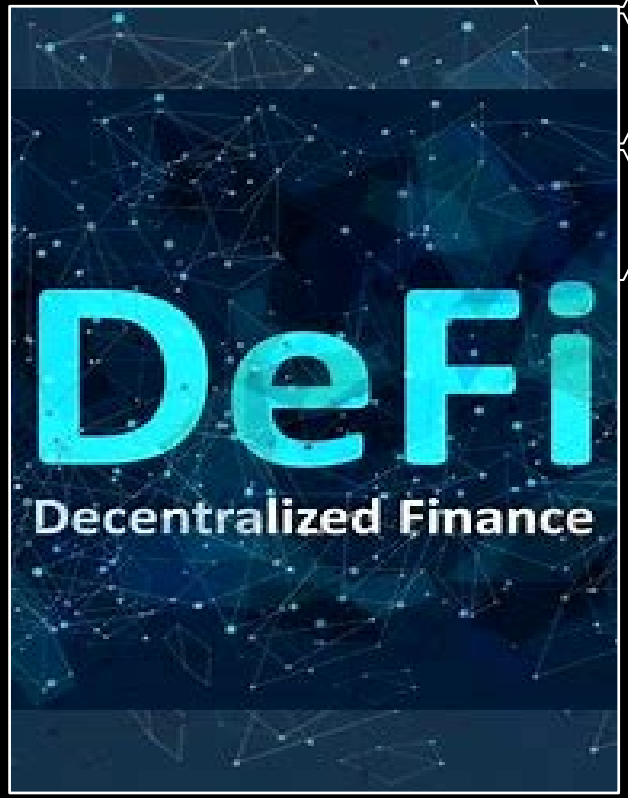
Introduce the background and motivation for developing the DApp. introduce the DApp and explain how it can address certain limitations of traditional financial applications.

What is DeFi?

Decentralised Finance (DeFi) is an emerging technology based on blockchain. It is a collective term for financial products and services that do not rely on any intermediary or centralised institution.

Goal of DeFi

Harnessing smart contracts, it aims at providing an open, transparent and trustless financial ecosystem for everyone.



Background and Motivation

At present, there are many problems in centralized financial systems (such as banks and payment platforms):

- **High costs and low efficiency** : Cross- border transfer fees are high, transaction confirmation times are long, and they rely on intermediaries, with complex processes.
- **Lack of transparency** : Users cannot track the flow of funds in real time, and the system operation lacks openness.
- **Centralized risks** : A single point of failure may lead to service disruptions or fund freezes.
- **Financial exclusion** : There are still billions of people worldwide who cannot access basic financial services, especially in economically underdeveloped countries or regions.

The goal of this project is to provide us an opportunity to apply what we have learnt in class to develop DeFi applications using distributed ledger technology.

DApp vs Centralized app

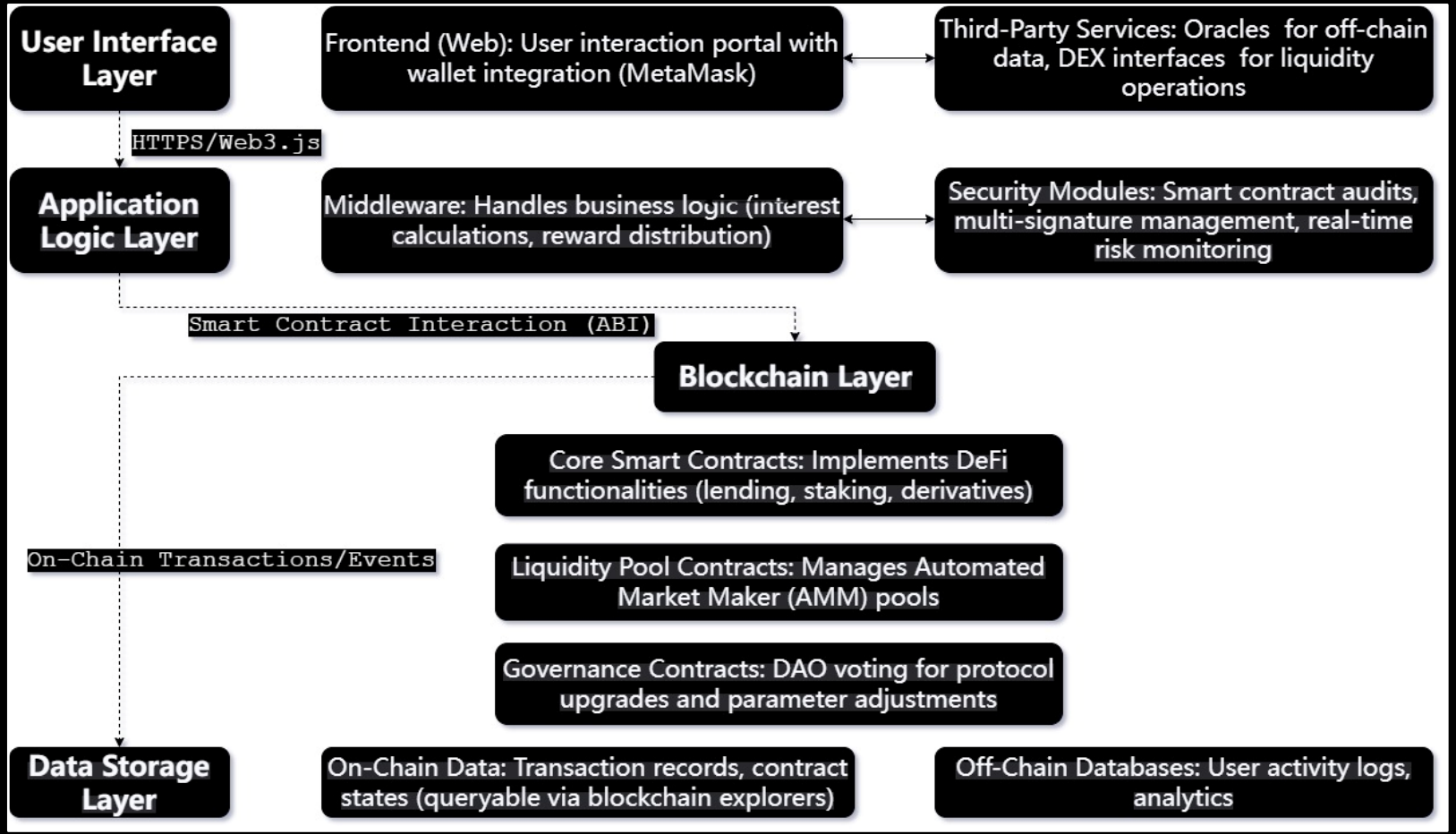
	DApp	Centralized app
Architecture	Based on blockchain, no central server; operates via distributed nodes	Relies on centralized servers; data and services managed centrally
Data Security	Data encrypted and stored on-chain, immutable and publicly verifiable	Data stored on private servers; risks include tampering, leaks, or single-point failures
Control	Executed by smart contracts; users fully own their assets and data	Controlled by corporations; users must trust third parties with data and permissions
Censorsip Resistance	Decentralized network resists shutdowns or censorship; globally accessible	Vulnerable to regulatory censorship; services can be restricted or terminated
Transparency	Open-source code; all transactions on-chain and auditable	Closed-source code; opaque backend operations and logic

02

System Design

Introduce the development process of the system using a system architecture diagram.

System architecture diagram



Where is the point?

Multi - Layered Design

User Interface
Application Logic
Blockchain Core
Data Storage

Scalability & Security

Built on Ethereum/BSC for high throughput. Audited contracts and multi-sig protocols ensure robustness.

Key Innovations

Trustless Execution
Censorship Resistance
Transparency

03

Prototype and Technology Stack

Implementation of the Prototype

Smart Contracts

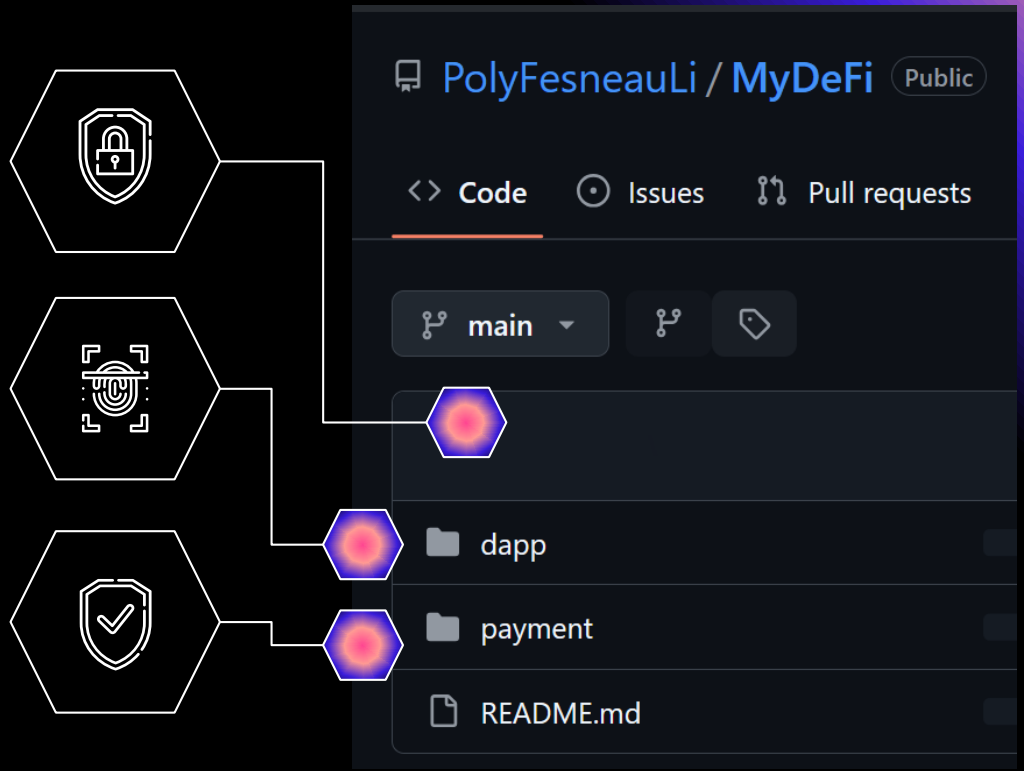
Located in the dapp/contracts directory, it is responsible for defining the core logic of DeFi

Frontend

Located in the dapp/src directory, build the user interface using React.js

Backend

Located in the payment directory, processing payment logic (buying KWT tokens via PayPal)



Implementation of Smart Contracts

Smart contracts are at the heart of the project, defining the token (KWT) issuance, wallet management, and transaction logic.

Key Documents:

KWTToken.sol: Defines the ERC-20 standard implementation of the KWT token.

WalletManager.sol: manages the creation of user wallets and the deposit and withdrawal of ETH.

Exchange.sol: Implements the exchange logic of KWT and ETH.

Focused Code Snippet Analysis

```
// KWTToken.sol
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract KWTToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("KWT Token", "KWT") {
        _mint(msg.sender, initialSupply * 10**decimals());
    }
}
```

· Use OpenZeppelin's ERC-20 contract library to ensure the security of your tokens.

· The initialSupply parameter is used to initialize the total amount of tokens.

Focused Code Snippet Analysis

```sol
// WalletManager.sol
pragma solidity ^0.8.0;

contract WalletManager {
    mapping(address => uint256) public balances;

    function deposit() public payable {
        require(msg.value > 0, "Deposit amount must be greater than 0");
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount, "Insufficient balance");
        payable(msg.sender).transfer(amount);
        balances[msg.sender] -= amount;
    }
}
```

· The deposit and withdraw methods handle the deposit and withdrawal of ETH separately.
· Use the payable keyword to ensure that the contract can receive ETH.

Focused Code Snippet Analysis

· The exchangeETHToKWT method implements the exchange logic of ETH and KWT.
· Use the transfer method to send KWT tokens to the user.

```
// Exchange.sol
pragma solidity ^0.8.0;

interface IWalletManager {
    function balances(address) external view returns (uint256);
    function deposit() external payable;
    function withdraw(uint256) external;
}

contract Exchange {
    IWalletManager public walletManager;
    KWTToken public kwtToken;

    constructor(address _walletManager, address _kwtToken) {
        walletManager = IWalletManager(_walletManager);
        kwtToken = KWTToken(_kwtToken);
    }

    function exchangeETHToKWT(uint256 ethAmount) public {
        require(ethAmount > 0, "Amount must be greater than 0");
        walletManager.deposit{value: ethAmount}();
        kwtToken.transfer(msg.sender, ethAmount * 2); // 1 ETH = 2 KWT
    }
}
```

Implementation of Frontend

Front-end applications are based on React.js and provide a user interface.

Key Documents:

App.js: The main application portal, which is responsible for routing and status management.

components/Wallet.js: displays the user's wallet balance and transaction history.

components/Exchange.js: provides the exchange function of ETH and KWT.

Focused Code Snippet Analysis

· Use Web3.js to connect to the user's wallet (MetaMask).
· Dynamically load user account information and display wallet and exchange components.

```
// App.js
import React, { useState, useEffect } from 'react';
import Web3 from 'web3';
import Wallet from './components/Wallet';
import Exchange from './components/Exchange';

function App() {
  const [web3, setWeb3] = useState(null);
  const [account, setAccount] = useState(null);

  useEffect(() => {
    const loadWeb3 = async () => {
      if (window.ethereum) {
        const web3Instance = new Web3(window.ethereum);
        const accounts = await web3Instance.eth.requestAccounts();
        setWeb3(web3Instance);
        setAccount(accounts[0]);
      }
    };
    loadWeb3();
  }, []);

  return (
    <div className="app">
      <h1>MyDeFi</h1>
      <p>Connected account: {account}</p>
      {web3 && account && (
```

Focused Code Snippet Analysis

```
// components/Wallet.js
import React, { useState, useEffect } from 'react';
import { WalletManager } from '../contracts/WalletManager.json';

function Wallet({ web3, account }) {
  const [balance, setBalance] = useState(0);

  useEffect(() => {
    const loadBalance = async () => {
      const networkId = await web3.eth.net.getId();
      const deployedWalletManager = new web3.eth.Contract(
        WalletManager.abi,
        WalletManager.networks[networkId].address
      );
      const balance = await deployedWalletManager.methods.balances(account).call
      setBalance(web3.utils.fromWei(balance, 'ether'));
    };
    loadBalance();
  }, [web3, account]);

  const deposit = async () => {
    const amount = web3.utils.toWei('0.1', 'ether');
    await web3.eth.sendTransaction({
      from: account,
      to: WalletManager.networks[networkId].address,
      value: amount,
    });
```

· Use Web3.js to call the smart contract method to get the user's ETH balance.
· A deposit button is provided that allows users to deposit ETH into their wallets.

Focused Code Snippet Analysis

· Input boxes and buttons are provided that allow users to exchange ETH for KWT.
· Call the exchangeETHToKWT method of the smart contract to complete the exchange.

```
// components/Exchange.js
import React, { useState, useEffect } from 'react';
import { Exchange } from '../contracts/Exchange.json';

function Exchange({ web3, account }) {
  const [ethAmount, setEthAmount] = useState('0.1');

  const exchange = async () => {
    const networkId = await web3.eth.net.getId();
    const deployedExchange = new web3.eth.Contract(
      Exchange.abi,
      Exchange.networks[networkId].address
    );
    await deployedExchange.methods.exchangeETHToKWT(ethAmount).send({
      from: account,
      value: web3.utils.toWei(ethAmount, 'ether'),
    });
  };

  return (
    <div className="exchange">
      <h2>Exchange</h2>
      <input
        type="number"
        value={ethAmount}
        onChange={(e) => setEthAmount(e.target.value)}
        placeholder="Enter ETH amount"
      />
```

Implementation of Backend

The backend service is responsible for handling the payment logic, specifically the purchase of KWT tokens through PayPal.

Key Documents:

server.js: The main service entrance, which uses Express.js to build APIs.

routes/payment.js: Handles PayPal payment callbacks.

Focused Code Snippet Analysis

```
// server.js
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const paypal = require('paypal-rest-sdk');

paypal.configure({
  mode: 'sandbox', // Use 'live' for production
  client_id: process.env.PAYPAL_CLIENT_ID,
  client_secret: process.env.PAYPAL_CLIENT_SECRET,
});

const app = express();
app.use(bodyParser.json());
app.use(cors());

app.post('/api/create-payment', (req, res) => {
  const create_payment_json = {
    intent: 'sale',
    payer: {
      payment_method: 'paypal',
    },
    redirect_urls: {
      return_url: 'http://localhost:3000/success',
      cancel_url: 'http://localhost:3000/cancel',
    },
    transactions: [
```

```
    transactions: [
      {
        amount: {
          currency: 'USD',
          total: req.body.amount,
        },
        description: 'Purchase KWT tokens',
      },
    ],
  };

  paypal.payment.create(create_payment_json, (error, payment) => {
    if (error) {
      throw error;
    }
    res.json(payment);
  });
});

app.listen(5000, () => {
  console.log('Server running on port 5000');
});
```

· Configure the PayPal SDK to process payment requests.
· Provide the /api/create-payment API to generate a payment link.

Focused Code Snippet Analysis

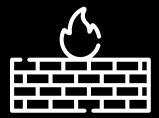
```
// routes/payment.js
const express = require('express');
const router = express.Router();
const paypal = require('paypal-rest-sdk');

router.post('/execute', (req, res) => {
  const payerId = req.body.payerID;
  const paymentId = req.body.paymentID;

  const execute_payment_json = {
    payer_id: payerId,
  };

  paypal.payment.execute(paymentId, execute_payment_json, (error, payment) => {
    if (error) {
      throw error;
    }
    // Trigger KWT token minting logic here
    res.json(payment);
  });
});
```

Process the callback after a successful PayPal payment to trigger the minting logic of KWT tokens.

04

Demonstration and Conclusion

Conclusion



Innovation and Technology :

Hybrid DeFi-Fiat Integration
Ethereum Smart Contract Backend
React.js Frontend
Truffle & Ganache Development
Enhanced User Accessibility

Functionality:

User-Friendly Wallet Management
Decentralized Smart Contract Backend
Multi-Currency Support (ETH & Fiat)
Seamless Transaction Processing
PayPal Sandbox Integration

Name	ID	Points
Li Tianye	24063117g	20%
Yu Shaoyu	24118601g	20%
Feng Yujie	24054389g	20%
Xia Yishan	24045097g	20%
Zhu Dongsheng	24038342g	20%

Appendix

github link:

https://github.com/PolyFesneauLi/MyDeFi.git

docker link:

https://drive.google.com/file/d/1gabfyH4NaZZ6CXFCjxCJ5sBG9mrWnNqs/view?usp=sharing

readme.md:

https://drive.google.com/drive/folders/1IfnY8UxxoW6YS9QxhMpfAEJ86jVv36F4?usp=sharing

Thanks

Open for questions

Group04 for COMP5521