BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Kedudukan selama melakukan proses kerja magang dalam perusahaan Pixel8labs Pte. Ltd. pada divisi *Tinkers* adalah sebagai *Solidity Engineer* bersama dengan MagicVault *team*. MagicVault *Team* memiliki 4 anggota dengan pembagian satu orang *project manager*, satu orang *frontend engineer*, satu orang *backend engineer*, dan satu orang *solidity engineer*.

MagicVault pada Pixel8labs Pte. Ltd. dikembangkan dengan metode *scrum* dengan menerapkan prinsip *agile* dengan menggunakan *sprint*. *Sprint* merupakan aktivitas yang terdiri dari beberapa tugas (*backlog*) yang harus diselesaikan dalam batasan waktu yang telah ditentukan [9]. Tugas baru akan dibagikan kepada setiap anggota tim saat *sprint planning* yang dilakukan oleh *Chief Technology Officer* setiap satu minggu sekali dengan tujuan untuk merencanakan apa yang akan dikerjakan (*backlog*) oleh setiap anggota tim selama satu minggu kedepan dalam proyek ini. Dalam proses kerja magang, terdapat juga *daily stand-up* yang dilakukan setiap hari Selasa dan Kamis pada pukul 18.00 WIB dengan MagicVault *Team* untuk mengetahui perkembangan terhadap tugas yang dikerjakan serta masalah apa yang dihadapi oleh para *developer*.

3.2 Tugas yang Dilakukan

Dalam pelaksanaan kerja magang pada Pixel8labs Pte. Ltd., tugas diberikan secara daring melalui aplikasi Jira Software. Tugas yang diberikan berupa bangun *smart contract* MagicVault. Selama pelaksanaan kerja magang, tanggung jawab yang diberikan adalah sebagai berikut.

1. Memahami konsep dan cara kerja MagicVault
2. Mempelajari dan membangun *smart contract* MagicVault dengan bahasa *solidity*
3. Membangun fungsi *staking* dan *unstaking* token USDC
4. Membangun fungsi *claim* token MAGIC

5. Membangun fungsi *withdraw* token MAGIC yang tersimpan pada *smart contract*
6. Membangun fungsi *convert* token STG menjadi token *USDC*
7. Melakukan *contract deployment* pada jaringan *testnet*
8. Melakukan *E2E Testing* terhadap *smart contract* yang telah dibangun.

3.3 Uraian Pelaksanaan Magang

Selama praktik kerja magang, tugas yang dilakukan adalah membangun *smart contract* MagicVault. Tugas-tugas yang dikerjakan setiap minggunya selama praktik kerja magang mengikuti *backlog* yang telah diberikan dan dapat dilihat lebih rinci pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	Pengenalan konsep proyek BAYC dan mempelajari pengembangan *smart contract* menggunakan *foundry library*.
2	Membuat fitur *whitelist free mint*, membuat fitur *get price*, mengintegrasikan ERC721A
3	Membuat fitur *mapping* tokenId NFT, membuat fitur validasi parameter fungsi *mint* pada *smart contract* BAYC,
4	Membuat fitur *onchain metadata* pada *smart contract* BAYC
5	Pengenalan dan pemahaman konsep MagicVault
6	Mempelajari dan memahami konsep MagicVault, Mempelajari konsep aplikasi DeFi *Stargate* dan cara menggunakan fungsi pada *smart contract Stargate*
7	Membuat fitur *staking* token USDC pada *smart contract* MagicVault dan membuat *unit test* terhadap fitur *staking* token USDC
8	Membuat fitur *claim* token MAGIC dan *unstaking* token USDC pada *smart contract* MagicVault
9	Memperbaiki fitur *unstaking* token USDC dan memperbaiki verifikasi *signature* terhadap fungsi *claim* token MAGIC pada *smart contract* MagicVault

Tabel 3.1. (lanjutan)

Minggu Ke -	Pekerjaan yang dilakukan
10	Membuat fitur *settle* pada *smart contract* MagicVault dan membuat *abusable unit test* terhadap *smart contract* MagicVault
11	Mengintegrasikan fungsi *claim MAGIC smart contract* pada *frontend* MagicVault
12	Migrasi *smart contract* MagicVault ke jaringan *Arbitrum Goerli* dan melakukan E2E Testing terhadap *smart contract* MagicVault
13	Pengenalan konsep proyek SOF, membuat fitur *public mint* pada *smart contract* SOF, dan membuat unit test terhadap fitur *public mint*.
14	Memperbaiki paremeter serta penambahan logika pada fitur *public mint* dan memperbaiki unit test terhadap fitur *public mint*.
15	Membuat fitur *whitelist mint* pada *smart contract* SOF, dan membuat unit test terhadap fitur *whitelist mint*.
16	Melakukan *deployment* terhadap *smart contract* SOF ke jaringan *Goerli* dan melakukan E2E testing terhadap *smart contract* SOF.

3.3.1 Spesifikasi

Proses pembuatan *frontend* portal keuangan dan pembayaran pada PT. Aset Digital Berkat menggunakan ASUS ROG Strix G531GT dengan spesifikasi sebagai berikut.

1. Prosesor: Intel Core i7-9750H
2. Memori: 8GB RAM
3. Penyimpanan: SSD 256 GB

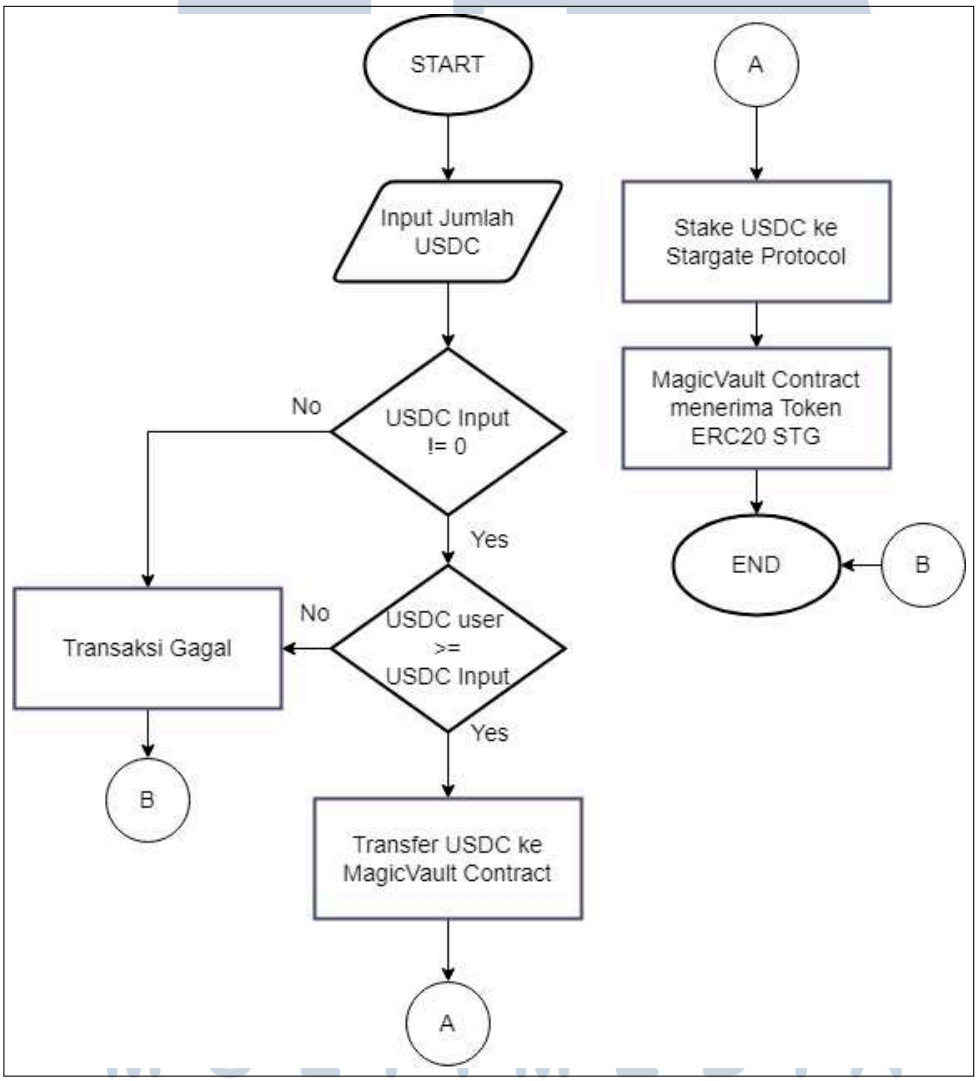
Adapun perangkat lunak yang digunakan untuk mengembangkan portal adalah sebagai berikut.

1. Sistem Operasi: Windows 11
2. Visual Studio Code, sebagai *text editor* untuk penulisan kode program

3.3.2 Perancangan

Berikut merupakan perancangan dalam bentuk *flowchart* yang menggambarkan alur dari fitur-fitur yang akan dibangun pada *smart contract* MagicVault.

A. USDC Staking

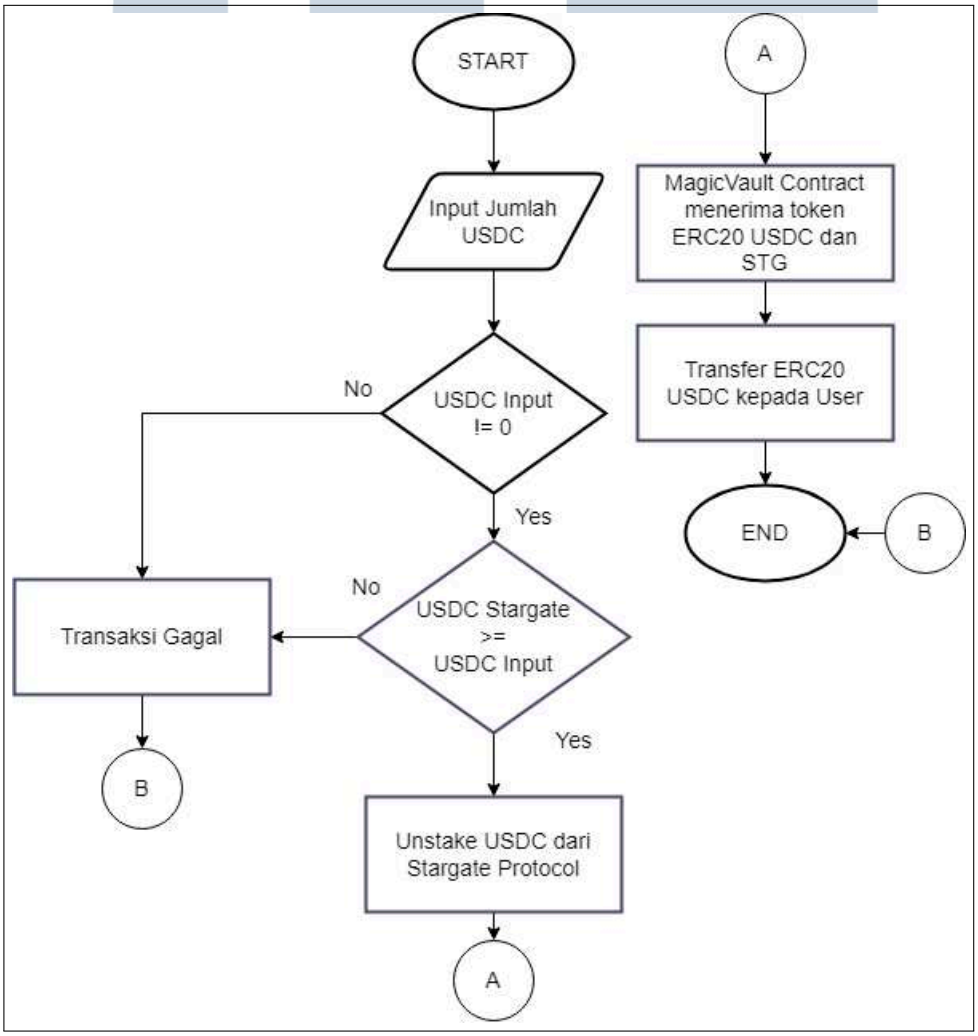


Gambar 3.1. *Flowchart Staking* Token USDC

Pada gambar 3.1, menampilkan *flowchart* terhadap fitur *staking* pada *smart contract* yang dimulai dari penginputan jumlah token USDC yang ingin dilakukan *staking*. Kemudian akan dilakukan pengecekkan terhadap jumlah token USDC yang diinput oleh *user*. Jika USDC yang diinput sama dengan 0 maka transaksi akan

gagal, jika tidak maka akan dilanjutkan dengan pengecekkan jumlah token USDC yang dimiliki *user*. Jika token USDC yang dimiliki *user* kurang dari jumlah yang diinput *user*, maka transaksi akan gagal. Jika tidak maka akan dilanjutkan dengan transfer token USDC ke *smart contract*. Selanjutnya, token USDC akan ditransfer ke *stargate finance protocol* untuk melakukan *staking*. Setelah itu, MagicVault mendapatkan imbalan berupa sejumlah token ERC20 yaitu STG.

B. USDC Unstaking

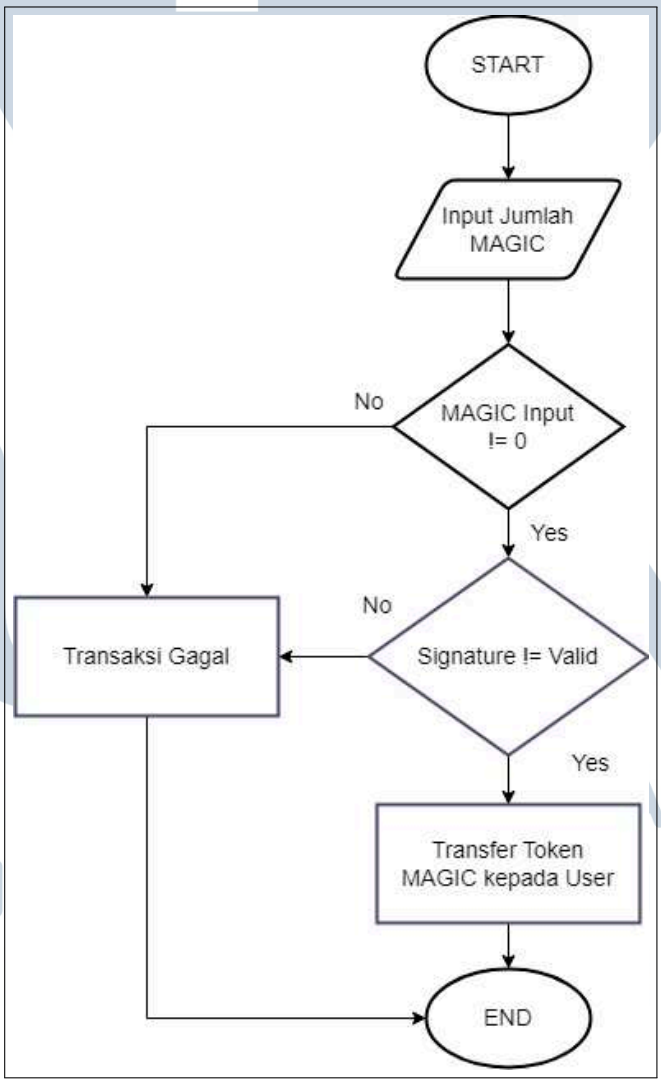


Gambar 3.2. *Flowchart Unstaking* Token USDC

Pada gambar 3.2, menampilkan *flowchart* terhadap fitur *unstaking* pada *smart contract* yang dimulai dari penginputan jumlah token USDC yang ingin dilakukan *unstaking*. Kemudian akan dilakukan pengecekkan terhadap jumlah

USDC yang diinput oleh *user*. Jika USDC yang diinput sama dengan 0 maka transaksi akan gagal, jika tidak maka akan dilanjutkan dengan pengecekkan jumlah USDC yang dapat dilakukan *unstaking*. Jika jumlah USDC yang diinput *user* melebihi jumlah USDC yang tersimpan pada *Stargate Finance Protocol*, maka transaksi akan gagal. Jika tidak maka akan dilanjutkan dengan *unstaking* token USDC dari *Stargate Finance Protocol*. Setelah itu, MagicVault mendapatkan imbalan melalui *unstaking* dari *stargate finance protocol* berupa sejumlah token ERC20 yaitu STG. Selanjutnya, token USDC akan ditransfer kepada *user*.

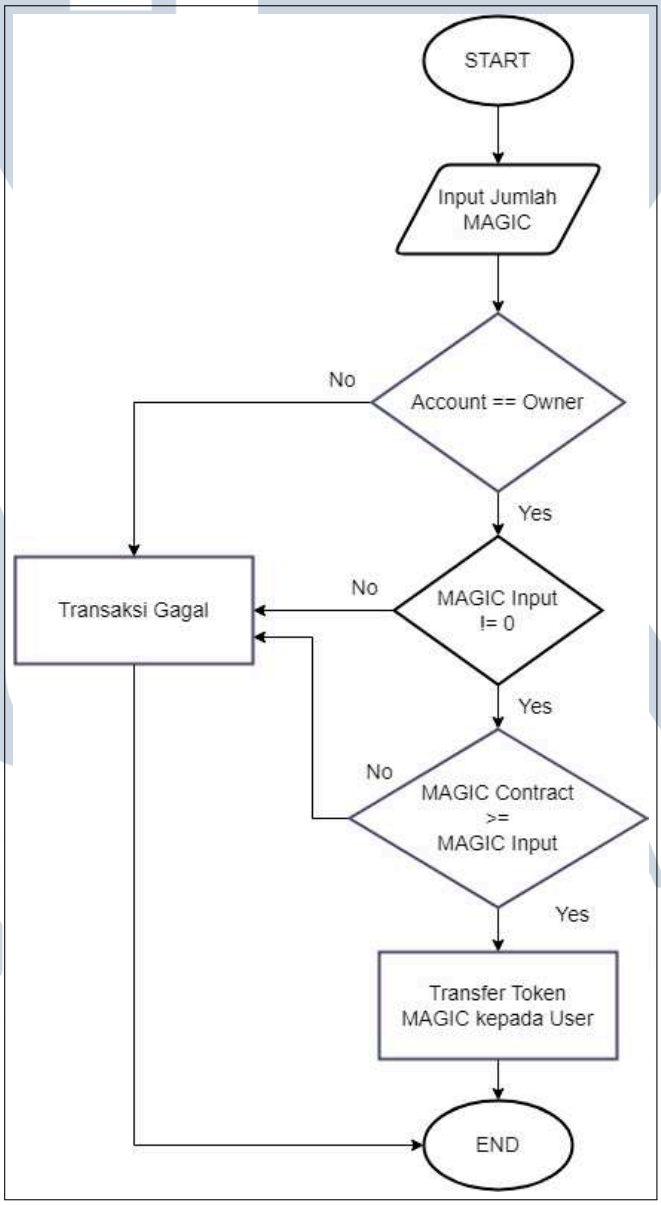
C. MAGIC Claim



Gambar 3.3. *Flowchart Claim* Token MAGIC

Pada gambar 3.3, menampilkan *flowchart* terhadap fitur *claim* token MAGIC pada *smart contract* yang dimulai dari penginputan jumlah token MAGIC dan *signature* dalam tipe data *bytes*. Kemudian akan dilanjutkan dengan pengecekkan token MAGIC yang diinput yaitu jika token MAGIC yang diinput sama dengan 0, maka transaksi akan gagal. Jika tidak akan dilanjutkan pengecekan *signature*. Jika *signature* tidak valid, maka transaksi akan gagal. Namun jika valid, maka akan dilanjutkan dengan transfer token MAGIC dari *smart contract* kepada *user*.

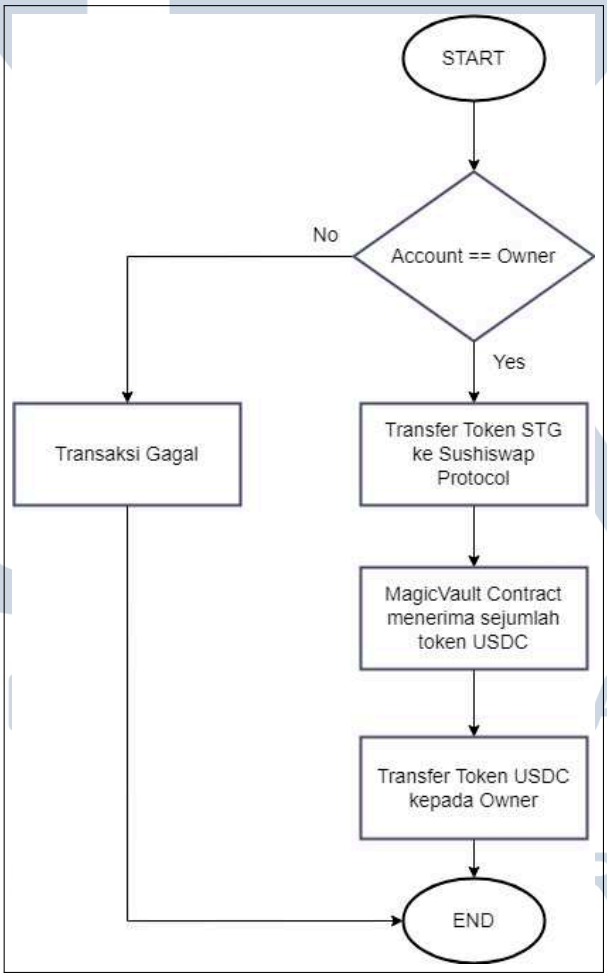
D. MAGIC Withdrawal



Gambar 3.4. *Flowchart Withdraw* Token MAGIC

Pada gambar 3.4, menampilkan *flowchart* terhadap fitur penarikan token MAGIC dari *smart contract* yang dimulai dari penginputan jumlah token MAGIC yang ingin ditarik dari *smart contract*. Setelah itu, akan dilakukan pengecekkan *owner* yaitu jika *user* yang memanggil fungsi tersebut bukan *owner*, maka transaksi akan gagal. Jika iya, maka akan dilanjutkan dengan pengecekkan jumlah token MAGIC yang diinput yaitu jika token MAGIC yang diinput sama dengan 0, maka transaksi akan gagal. Jika tidak, maka akan dilanjutkan dengan pengecekkan jumlah token MAGIC yang dapat ditarik. Jika jumlah token MAGIC yang diinput melebihi token MAGIC yang tersimpan pada *smart contract* maka transaksi akan gagal. Jika tidak, maka akan dilanjutkan dengan transfer token MAGIC kepada *user* tersebut.

E. STG Settle



Gambar 3.5. *Flowchart* Penukaran ERC20 STG ke USDC

Pada gambar 3.5, menampilkan *flowchart* terhadap fitur penukaran token STG yang terdapat pada *smart contract* menjadi token USDC. Fitur ini dimulai dari pengecekkan *owner* yaitu jika *user* yang memanggil fungsi tersebut bukan *owner* maka transaksi akan gagal. Jika iya, maka akan dilanjutkan transfer seluruh token STG yang ada di *smart contract* ke *sushiswap protocol* untuk melakukan penukaran token STG menjadi token USDC. Setelah itu, *smart contract* MagicVault akan menerima sejumlah token USDC dari hasil penukaran tersebut. Selanjutnya, token USDC akan ditransfer kepada *user* atau *owner*.

3.3.3 Implementasi

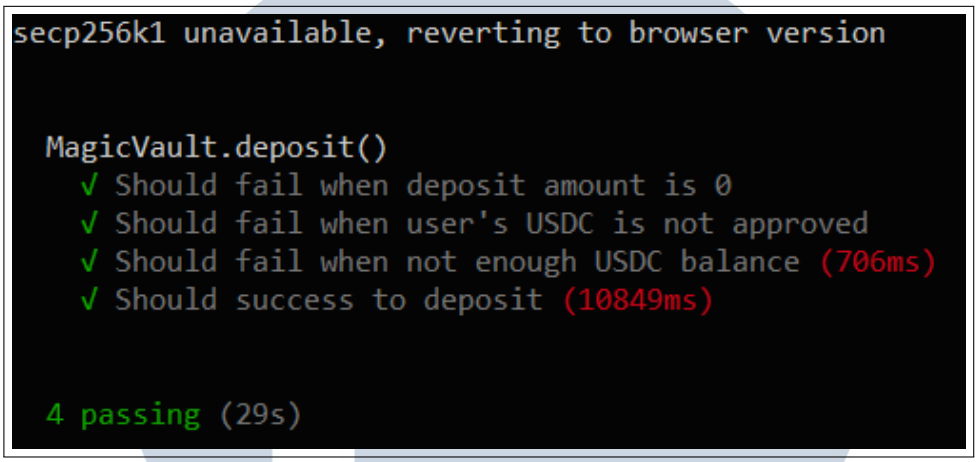
Berikut merupakan hasil implementasi fitur-fitur yang terdapat pada *smart contract* MagicVault yang mencakup gambar kode, gambar hasil testing, dan gambar hasil dari transaksi pada setiap fitur tersebut.

A. USDC Staking

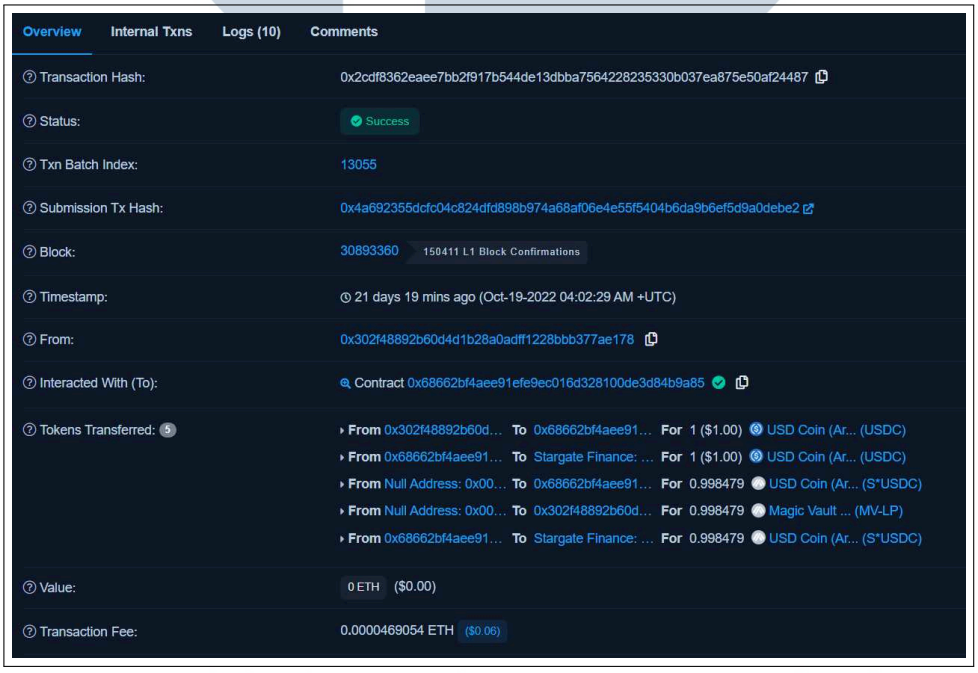
```
function deposit(uint256 _amountUSDC) public nonReentrant {
    if (_amountUSDC == 0) revert ZeroDepositAmount();

    // Transfer USDC from user to contract
    usdc.transferFrom(msg.sender, address(this), _amountUSDC);

    // Snapshot Stargate LP token balance before add liquidity
    uint256 prevBalance = sgLPToken.balanceOf(address(this));

    // Deposit USDC to Stargate Pool via Stargate Router
    stargateRouter.addLiquidity(stargatePoolID, _amountUSDC, address(this));

    // Snapshot Stargate LP token balance after add liquidity
    uint256 afterBalance = sgLPToken.balanceOf(address(this));

    // Get exact amount of sgLpToken received after adding liquidity
    uint256 sgLPTokenReceived = afterBalance.sub(prevBalance);

    // Mint MV-LP token based on deposited USDC amount
    _mint(msg.sender, sgLPTokenReceived);

    // Deposit LP token to Stargate LP Farming contract
    stargateLPFarming.deposit(farmingPoolID, sgLPTokenReceived);

    emit LPTokenMinted(msg.sender, sgLPTokenReceived);
}
```

Gambar 3.6. Baris Kode USDC *Staking*

Pada gambar 3.6 menunjukkan baris kode *smart contract* yang mengimplementasikan fitur USDC *staking* yang akan dijalankan oleh pengguna aplikasi MagicVault untuk melakukan *staking* token USDC.



Gambar 3.7. Hasil *Testing* pada USDC *Staking*



Gambar 3.8. Hasil Penggunaan USDC *Staking*

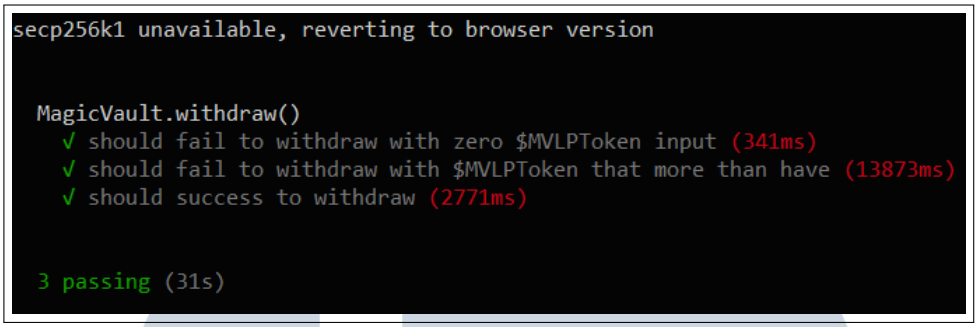
Pada gambar 3.7 menampilkan hasil *testing* terhadap fitur USDC *staking* yang menunjukkan bahwa, fitur tersebut berhasil dijalankan sesuai dengan beberapa *case* tertentu dan hasil penggunaan dari fitur tersebut dapat dilihat pada gambar 3.8 dibagian *tokens Transferred* yang menunjukkan transaksi token USDC ketika fitur tersebut dijalankan.

B. USDC Unstaking

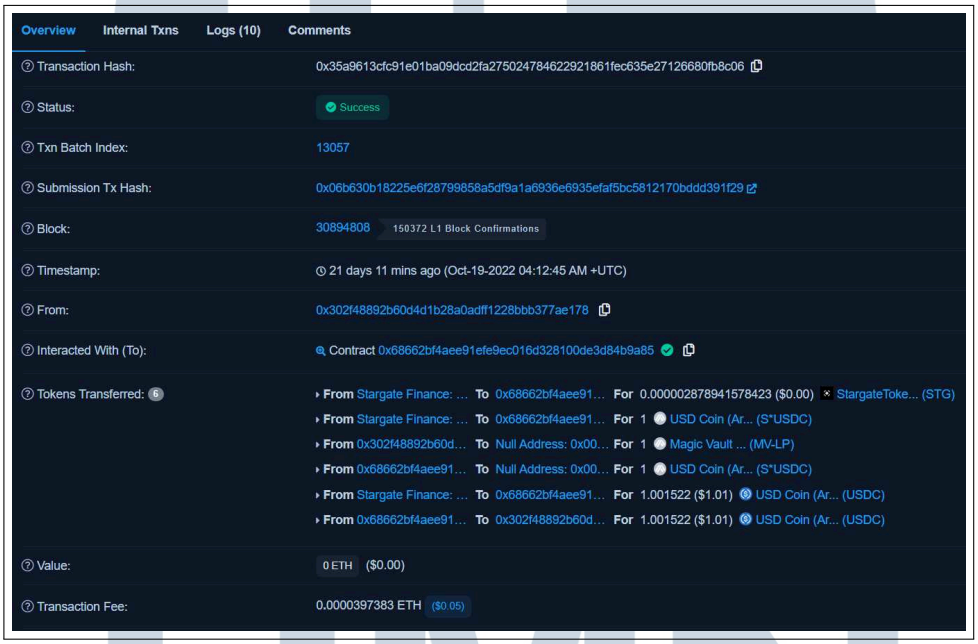
```
function withdraw(uint256 _amountMVLpToken) public nonReentrant {
    if(_amountMVLpToken == 0) revert ZeroWithdrawAmount();

    if(_amountMVLpToken > balanceOf(msg.sender)) revert NotEnoughBalance();

    // Snapshot Stargate LP token balance before unstake
    uint256 prevBalance = sgLPToken.balanceOf(address(this));

    // Withdraw LP Token (Stargate) from Stargate LP Farming contract
    stargateLPFarming.withdraw(farmingPoolID, _amountMVLpToken);

    // Snapshot Stargate LP token balance after unstake
    uint256 afterBalance = sgLPToken.balanceOf(address(this));

    // Burn User's MV-LP token
    _burn(msg.sender, _amountMVLpToken);

    // Snapshot USDC balance before remove liquidity
    uint256 prevUSDCBalance = usdc.balanceOf(address(this));

    // Get exact amount of USDC received after removing liquidity
    uint256 sgLPTokenReceived = afterBalance.sub(prevBalance);

    // Withdraw USDC from Stargate Pool via Stargate Router
    stargateRouter.instantRedeemLocal(
        stargatePoolID,
        sgLPTokenReceived,
        address(this)
    );

    // Snapshot USDC balance after remove liquidity
    uint256 afterUSDCBalance = usdc.balanceOf(address(this));

    // Get exact amount of USDC transferred to user
    uint256 amountUSDCTransferred = afterUSDCBalance.sub(prevUSDCBalance);

    // Transfer $USDC to user
    emit USDCwithdraw(msg.sender, _amountMVLpToken, amountUSDCTransferred);

    usdc.transfer(msg.sender, amountUSDCTransferred);
}
```

Gambar 3.9. Baris Kode USDC *Unstaking*

Pada gambar 3.9 menunjukkan baris kode *smart contract* yang mengimplementasikan fitur USDC *unstaking* yang akan dijalankan oleh pengguna aplikasi MagicVault untuk melakukan *unstaking* token USDC. Sebelum melakukan *unstaking*, pengecekkan akan dijalankan untuk memeriksa jumlah token USDC pengguna yang tersimpan.



Gambar 3.10. Hasil *Testing* pada USDC *Unstaking*



Gambar 3.11. Hasil Penggunaan USDC *Unstaking*

Pada gambar 3.10 menampilkan hasil *testing* terhadap fitur USDC *unstaking* yang menunjukkan bahwa, fitur tersebut berhasil dijalankan sesuai dengan beberapa *case* tertentu dan hasil penggunaan dari fitur tersebut dapat dilihat pada gambar 3.11 dibagian *token Transferred* yang menunjukkan penarikan token USDC serta pengiriman token USDC kembali kepada *user*.

C. MAGIC Claim

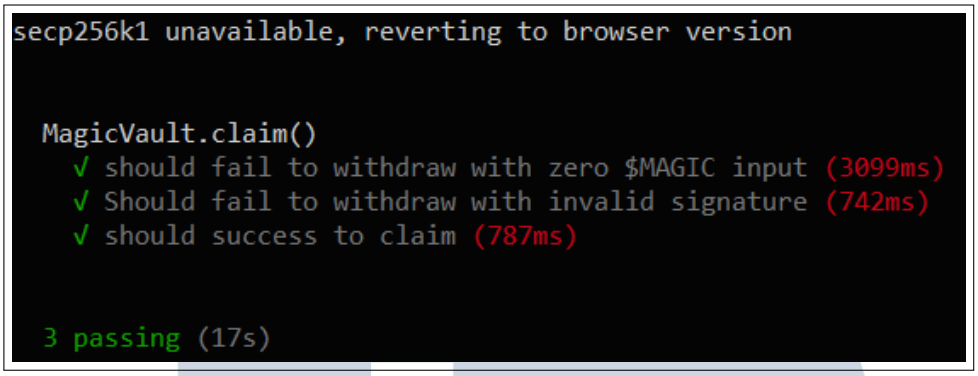
```
function claim(uint256 _amountMAGIC, bytes memory signature) public nonReentrant {
        if(_amountMAGIC == 0) revert ZeroWithdrawAmount();

        // Verify signature
        if(Signature.verify(_amountMAGIC, msg.sender, signature) != signer) revert InvalidSignature();

        // Transfer $MAGIC from contract to user
        magic.transfer(msg.sender, _amountMAGIC);

        emit MAGICCollect(msg.sender, _amountMAGIC);
    }
```

Gambar 3.12. Baris Kode MAGIC *Claim*

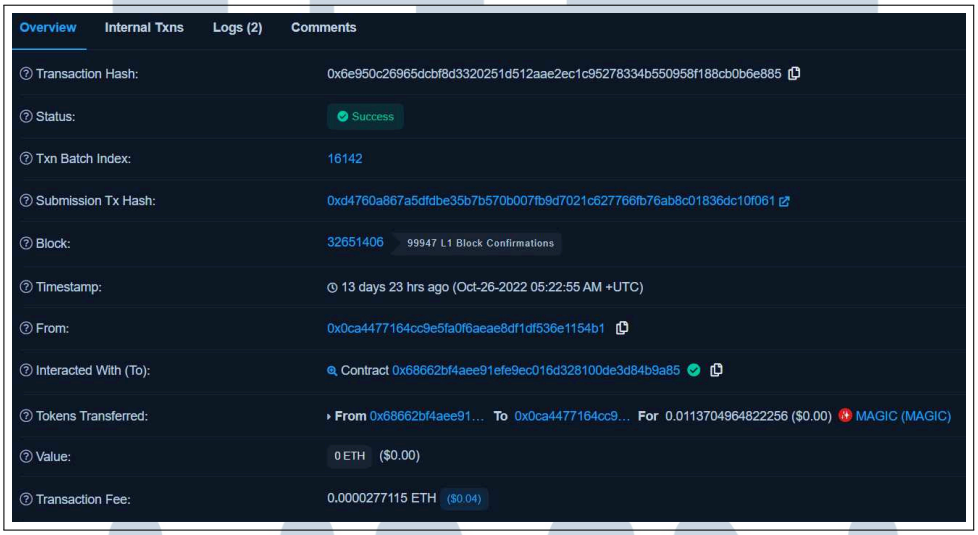
```
//SPDX-License-Identifier: Unlicense
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/Strings.sol";
import "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";

library Signature {
    function verify(uint amount, address target, bytes memory signature) internal pure returns (address) {
        bytes32 payloadHash = keccak256(abi.encode(target, amount));

        bytes32 messageHash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", payloadHash));

        uint8 v;
        bytes32 r;
        bytes32 s;
        (v,r,s) = splitSignature(signature);

        return ecrecover(messageHash, v, r, s);
    }

    function splitSignature(bytes memory sig) internal pure returns (uint8, bytes32, bytes32)
    {
        require(sig.length == 65);

        bytes32 r;
        bytes32 s;
        uint8 v;

        assembly {
            r := mload(add(sig, 32))
            s := mload(add(sig, 64))
            v := byte(0, mload(add(sig, 96)))
        }

        return (v, r, s);
    }
}
```

Gambar 3.13. Baris Kode Verifikasi MAGIC *Claim*

Pada gambar 3.12 dan 3.13 menunjukkan baris kode *smart contract* yang mengimplementasikan fitur MAGIC *claim* dan sistem verifikasi terhadap fitur tersebut. Sebelum melakukan *claim* token MAGIC, sistem verifikasi akan dijalankan terlebih dahulu untuk memeriksa apakah pengguna telah melakukan *staking*. Fitur ini akan dijalankan oleh pengguna aplikasi MagicVault untuk melakukan *claim* token MAGIC sebagai *reward*.



Gambar 3.14. Hasil *Testing* pada MAGIC *Claim*



Gambar 3.15. Hasil Penggunaan MAGIC *Claim*

Pada gambar 3.14 menampilkan hasil *testing* terhadap fitur MAGIC *claim* yang menunjukkan bahwa, fitur tersebut berhasil dijalankan sesuai dengan beberapa *case* tertentu dan hasil penggunaan dari fitur tersebut dapat dilihat pada gambar 3.15 dibagian *token Transferred* yang menunjukkan pengiriman token Magic kepada *user*.

D. MAGIC Withdrawal

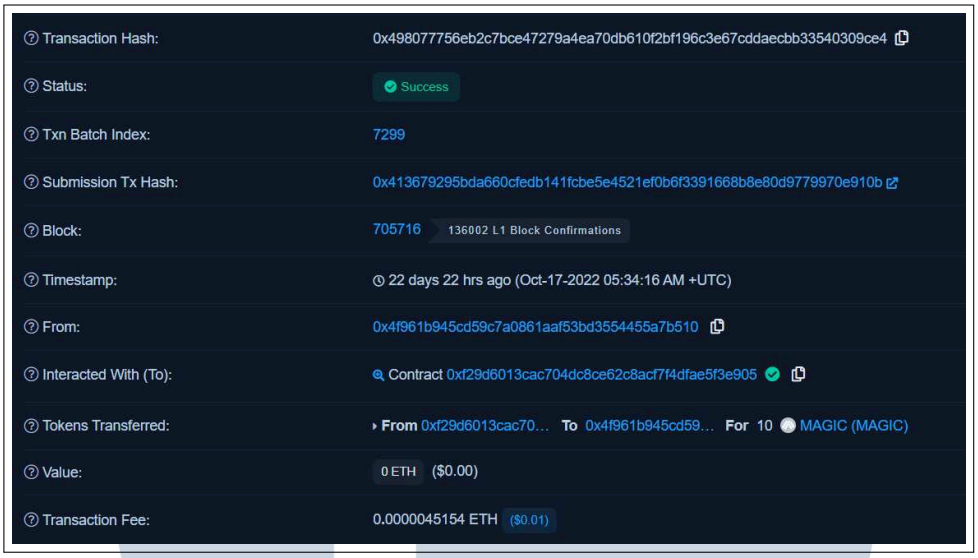
```
function withdrawMAGIC(uint256 _amountMAGIC) public nonReentrant onlyOwner {
        // Get STG Balance Contract
        uint256 amountMAGIC = magic.balanceOf(address(this));

        if(_amountMAGIC == 0) revert ZeroWithdrawMAGIC();
        if(_amountMAGIC > amountMAGIC) revert NotEnoughMAGIC();

        magic.transfer(msg.sender, _amountMAGIC);
    }
```

Gambar 3.16. Baris Kode MAGIC *Withdrawal*

Pada gambar 3.16 menunjukkan baris kode *smart contract* yang mengimplementasikan fitur MAGIC *withdrawal* yang akan dijalankan oleh pemilik aplikasi MagicVault untuk melakukan penarikan token MAGIC yang tersimpan pada *smart contract* dan fitur ini hanya dapat dijalankan oleh pemilik aplikasi MagicVault.

```
secp256k1 unavailable, reverting to browser version


  MagicVault.withdrawMAGIC()
    √ Should success to withdraw MAGIC from MagicVault (3045ms)
    √ Should fail to withdraw 0 MAGIC from MagicVault (339ms)
    √ Should fail to withdraw Insufficient MAGIC from MagicVault (314ms)
    √ Should fail to withdraw MAGIC from MagicVault if not owner (404ms)


  4 passing (18s)
```

Gambar 3.17. Hasil *Testing* pada MAGIC *Withdrawal*



Gambar 3.18. Hasil Penggunaan MAGIC *Withdrawal*

Pada gambar 3.17 menampilkan hasil *testing* terhadap fitur MAGIC *withdrawal* yang menunjukkan bahwa, fitur tersebut berhasil dijalankan sesuai dengan beberapa *case* tertentu dan hasil penggunaan dari fitur tersebut dapat dilihat pada gambar 3.18 dibagian *token Transferred* yang menunjukkan penarikan token Magic dari *smart contract* MagicVault menuju *user* yang memanggil fitur tersebut.

E. STG Settlement

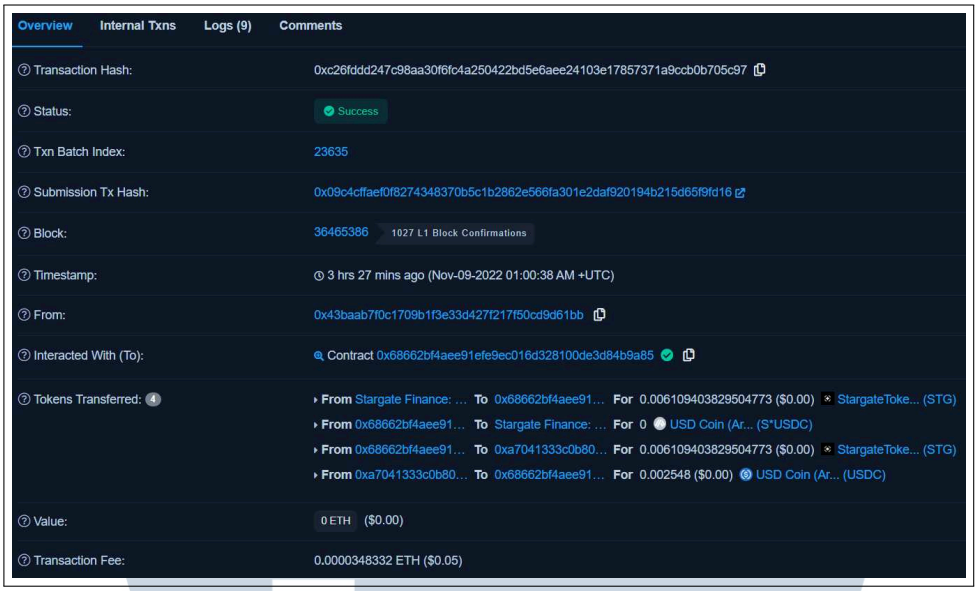
```
function settle() public nonReentrant onlyOwner{
        // Claim All $STG rewards from Stargate LP Farming contract
        stargateLPFarming.deposit(farmingPoolID, 0);

        // Get STG Balance Contract
        uint256 amountSTG = stg.balanceOf(address(this));

        // Created Path
        address[] memory path = new address[](2);
        path[0] = address(stg);
        path[1] = address(usdc);

        uint amountOutMin = sushiSwapRouter.getAmountsOut(amountSTG, path)[1];

        // Execute the Tokens Swap from $STG to $USDC
        sushiSwapRouter.swapExactTokensForTokens(
            amountSTG,
            amountOutMin,
            path,
            address(this),
            block.timestamp
        );

        // Emit Settle Event
        emit Settle(msg.sender, amountSTG, amountOutMin);
    }
```

Gambar 3.19. Baris Kode STG *Settlement*

Pada gambar 3.19 menunjukkan baris kode *token* yang mengimplementasikan fitur STG *settlement* yang akan dijalankan oleh pemilik aplikasi MagicVault untuk mengubah token STG yang tersimpan pada *token* menjadi token USDC.

```
secp256k1 unavailable, reverting to browser version


  MagicVault.settle()
    ✓ Should fail if the caller is not the owner (13417ms)
    ✓ Should fail if stargate contract is 0 (311ms)
    ✓ Should success to settle (4866ms)


  3 passing (33s)
```

Gambar 3.20. Hasil *Testing* pada STG *Settle*



Gambar 3.21. Hasil Penggunaan STG *Settle*

Pada gambar 3.20 menampilkan hasil *testing* terhadap fitur STG *settlement* yang menunjukkan bahwa, fitur tersebut berhasil dijalankan sesuai dengan beberapa *case* tertentu dan hasil penggunaan dari fitur tersebut dapat dilihat pada gambar 3.21 dibagian *token Transferred* yang menunjukkan penukaran token STG menjadi token USDC.

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala

Dalam proses membangun *smart contract* MagicVault pada Pixel8labs Pte. Ltd. ditemukan kendala sebagai berikut.

- Kurangnya pengetahuan tentang *DeFi* mengenai *Staking*, *Unstaking*, dan *APY*, sehingga membutuhkan waktu untuk membuat fitur-fitur pada *smart contract* MagicVault
- Kurangnya dokumentasi ataupun video pembelajaran mengenai penggunaan fitur *Stargate Protocol* dan *SushiSwap Protocol* sehingga kesulitan dalam mengintegrasikan kedua *protocol* tersebut ke dalam *smart contract* MagicVault

3.4.2 Solusi

Solusi yang dilakukan untuk menyelesaikan kendala yang ditemukan adalah sebagai berikut.

- Mempelajari konsep *DeFi* mengenai *Staking*, *Unstaking*, dan *APY* di saat waktu luang dan sebelum memulai pengembangan *smart contract*.
- Membaca dokumentasi resmi dengan teliti dan bertanya pada orang yang sudah berpengalaman menggunakan *Stargate Protocol* dan *SushiSwap Protocol* serta mencoba mempraktekkannya secara langsung pada saat pengembangan *smart contract*.