BLOCKSEC

Security Audit Report for Fiat24

Date: September 29, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Mantle
Target	Fiat24

Version History

Version	Date	Description
1.0	September 29, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-10-02 08:45:46+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository [1] of Fiat24 of Mantle.

Fiat24 is a pluggable fiat infrastructure for Web3 projects. It streamlines the fiat money transfer and payments for both traditional and web3 native users by providing experiences that they're most comfortable with. With support for most of the major payment channels such as SEPA, SIC, SWIFT and Debit Card processings, Fiat24 results in a standard payment provider to web3 users and applications.

Note this audit only focuses on the smart contracts in the following directories/files:

- Fiat24CardAuthorizationMarqeta.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
Fiat24	Version 0	bfb808cfd1a8d6123efb7174355ebd93caca364f
	Version 1	165d56839b7bd28b0440f83a257ec99c1f507271
	Version 2	1806ec8b43bf5d7e35390e7c1488324b7ee1df94

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any war-

[1] https://github.com/mantle-xyz/fiat24contracts

ranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

- Access control
- Permission management
- Whitelist and blacklist mechanisms
- Initialization consistency
- Improper use of the proxy system
- Reentrancy
- Denial of Service (DoS)
- Untrusted external call and control flow
- Exception handling
- Data handling and flow
- Events operation
- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization

∗ Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

		Likelihood	
		High	*Low*
Impact	*High*	High	Medium
	Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **three** potential security issues. Besides, we have **five** recommendations and **four** notes.

- Medium Risk: 1
- Low Risk: 2
- Recommendation: 5
- Note: 4

ID	Severity	Description	Category	Status
1	Medium	Incorrect access control logic in the function `configureTokenPair()`	Security Issue	Fixed
2	Low	Insufficient input amount due to rounding down	Security Issue	Confirmed
3	Low	Insufficient payment amount due to rounding down	Security Issue	Confirmed
4	-	Add non-zero address checks	Recommendation	Fixed
5	-	Revise the error message in the function `unpause()`	Recommendation	Fixed
6	-	Add validation to avoid duplicate addresses assignment	Recommendation	Fixed
7	-	Implement event emissions for all state-changing functions	Recommendation	Fixed
8	-	Early revert for insufficient token allowance	Recommendation	Fixed
9	-	Potential centralization risks	Note	-
10	-	OpenZeppelin `Initializable` upgrade migration risks	Note	-
11	-	Hardcoded values on initialization	Note	-
12	-	Ensure timely updates of `exchangeRate` in `TokenPairConfig`	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Incorrect access control logic in the function `configureTokenPair()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `Fiat24CardAuthorizationMarqeta` implements role-based access control in the function `configureTokenPair()`, which uses an incorrect logical operator for its permission check. Specifically, the condition requires the caller to possess both roles due to the `OR` operator being used with negative checks, which contradicts the intended functionality of allowing users with either role to configure token pairs. This issue prevents authorized users who possess only one of the required roles from successfully executing the function.

```
function configureTokenPair(
    address tokenIn_,
    address tokenOut_,
    uint256 exchangeRate_,
    bool isActive_
) external {
    if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) || !(hasRole(CRYPTO_CONFIG_UPDATER_ROLE, _msgSender()))) revert Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
    require(tokenIn_ != address(0), "Invalid input token address");
    require(tokenOut_ != address(0), "Invalid output token address");
    require(tokenIn_ != tokenOut_, "Input and output tokens cannot be the same");

    if (isActive_) {
        require(exchangeRate_ > 0, "Exchange rate must be > 0");
    }

    tokenPairConfigs[tokenIn_][tokenOut_] = TokenPairConfig({
        exchangeRate: exchangeRate_,
        isActive: isActive_
    });

    emit TokenPairConfigured(tokenIn_, tokenOut_, exchangeRate_, isActive_);
}
```

Listing 2.1: src/Fiat24CardAuthorizationMarqeta.sol

Impact This access control flaw prevents legitimate operators from configuring token pairs, disrupting essential system maintenance operations.

Suggestion Revise the logic accordingly.

2.1.2 Insufficient input amount due to rounding down

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The contract `Fiat24CardAuthorizationMarqeta` calculates required input amounts in the function `_calculateRequiredInput()`, which performs integer division that always rounds down. The vulnerability exists because the calculation does not round up the result, which can lead to a slight shortfall in the actual input amount needed to achieve the desired output.

```
function _calculateRequiredInput(
    address tokenIn,
    address tokenOut,
    uint256 outputAmount,
    uint256 exchangeRate
) internal view returns (uint256 requiredInput) {
    uint8 tokenInDecimals = ERC20Upgradeable(tokenIn).decimals();
    uint8 tokenOutDecimals = ERC20Upgradeable(tokenOut).decimals();

    if (tokenInDecimals != tokenOutDecimals) {

        uint256 normalizedOutput;
        uint256 divisor;

        if (tokenOutDecimals < tokenInDecimals) {
            // Output token has fewer decimals, scale up for calculation
            uint256 decimalDiff = tokenInDecimals - tokenOutDecimals;
            normalizedOutput = outputAmount * (10 ** decimalDiff);
            divisor = 1e18;
        } else {
            // Output token has more decimals, adjust divisor
            uint256 decimalDiff = tokenOutDecimals - tokenInDecimals;
            normalizedOutput = outputAmount;
            divisor = 1e18 * (10 ** decimalDiff);
        }

        return (normalizedOutput * exchangeRate) / divisor;
    } else {

        return (outputAmount * exchangeRate) / 1e18;
    }
}
```

Listing 2.2: src/Fiat24CardAuthorizationMarqeta.sol

Impact This can lead to a slight shortfall in the actual input amount needed to achieve the desired output.

Suggestion Round up the result.

2.1.3 Insufficient payment amount due to rounding down

Severity Low

Status Confirmed

Introduced by Version 1

Description The contract Fiat24CardAuthorizationMarqeta processes payment calculations in the function authorize(), which performs mathematical operations that truncate fractional results. The vulnerability exists because the paidAmount variable is calculated using integer division that always rounds down, which can result in a slightly insufficient payment amount being requested from the user. This issue may cause the transferred amount to be marginally less than the precise value that is required for the transaction. The function increment() has the same issue.

```
function authorize(
    string memory authorizationToken_,
    string memory cardId_,
    uint256 tokenId_,
    address cardCurrency_,
    string memory transactionCurrency_,
    address settlementCurrency_,
    uint256 transactionAmount_,
    uint256 settlementAmount_
) public {
    if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
    if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
    if (!validXXX24Tokens[settlementCurrency_]) revert Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(settlementCurrency_);
    address sender = IFiat24Account(fiat24AccountAddress).ownerOf(tokenId_);
    address booked = IFiat24Account(fiat24AccountAddress).ownerOf(CARD_BOOKED);
    address paidCurrency = cardCurrency_;
    uint256 paidAmount;

    address txnToken = XXX24Tokens[transactionCurrency_];

    if (validXXX24Tokens[txnToken]) {

        if (
           IERC20Upgradeable(txnToken).balanceOf(sender) >= transactionAmount_
           && IERC20Upgradeable(txnToken).allowance(sender, address(this)) >= transactionAmount_
        ) {

           paidCurrency = txnToken;
           paidAmount = transactionAmount_;
        } else {

           paidAmount = transactionAmount_ * getRate(txnToken, cardCurrency_)
               * getSpread(txnToken, cardCurrency_, false) / 100000000;

           if (cardCurrency_ == usd24Address) {
               uint256 userUsd24Balance = IERC20Upgradeable(usd24Address).balanceOf(sender);
               uint256 userUsd24Allowance = IERC20Upgradeable(usd24Address).allowance(sender, address(this));
               uint256 availableUsd24 = userUsd24Balance < userUsd24Allowance ? userUsd24Balance : userUsd24Allowance;

               if (availableUsd24 < paidAmount) {
                   require(_trySwapAlternativeTokens(usd24Address, sender, paidAmount - availableUsd24),
                      "Failed to swap alternative tokens for USD24");
               }
           }
        }
    } else {
        if (settlementCurrency_ != eur24Address) revert Fiat24CardAuthorizationMarqeta__DefaultSettlementCurrencyIsNotEUR(settlementCurrency_);
        paidAmount = settlementAmount_ * (100 + interchange) * getRate(eur24Address, cardCurrency_) * getSpread(eur24Address, cardCurrency_, false) / 10000000000;

        if (cardCurrency_ == usd24Address) {
```

```
            uint256 userUsd24Balance = IERC20Upgradeable(usd24Address).balanceOf(sender);
            uint256 userUsd24Allowance = IERC20Upgradeable(usd24Address).allowance(sender,
                address(this));
            uint256 availableUsd24 = userUsd24Balance < userUsd24Allowance ? userUsd24Balance :
                 userUsd24Allowance;

            if (availableUsd24 < paidAmount) {
                require(_trySwapAlternativeTokens(usd24Address, sender, paidAmount -
                    availableUsd24),
                    "Failed to swap alternative tokens for USD24");
            }
        }
    }

    IERC20Upgradeable(paidCurrency).safeTransferFrom(sender, booked, paidAmount == 0 ? 1 :
        paidAmount);
    emit Authorized(authorizationToken_, tokenId_, sender, cardId_, paidCurrency, paidAmount);
}
```

Listing 2.3: src/Fiat24CardAuthorizationMarqeta.sol

Impact This calculation error can lead to underpayment in transactions, potentially causing financial discrepancies in the accounting system.

Suggestion Round up the result of `paidAmount`.

2.2 Recommendation

2.2.1 Add non-zero address checks

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `setAlternativeInputTokens()`, the address variables (e.g., `outputToken` and `inputTokens`) are not checked to ensure they are not zero. It is recommended to add such checks to prevent potential misoperations.

```
function setAlternativeInputTokens(address outputToken, address[] calldata inputTokens)
     external {
    if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender()))) revert
        Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
    alternativeInputTokens[outputToken] = inputTokens;
}
```

Listing 2.4: src/Fiat24CardAuthorizationMarqeta.sol

Suggestion Add non-zero address checks accordingly.

2.2.2 Revise the error message in the function `unpause()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The functions pause() and unpause() revert with identical error messages if the caller lacks the respective PAUSE_ROLE or UNPAUSE_ROLE. To enhance code clarity and prevent confusion, it is recommended to revise the error message in the function unpause() to clearly differentiate it from the function pause()'s error message.

```
function pause() external {
    if (!(hasRole(PAUSE_ROLE, _msgSender()))) revert Fiat24CardAuthorizationMarqeta__NotPauser(_msgSender());
    _pause();
}
```

Listing 2.5: src/Fiat24CardAuthorizationMarqeta.sol

```
function unpause() external {
    if (!(hasRole(UNPAUSE_ROLE, _msgSender()))) revert Fiat24CardAuthorizationMarqeta__NotPauser(_msgSender());
    _unpause();
}
```

Listing 2.6: src/Fiat24CardAuthorizationMarqeta.sol

Suggestion Revise the error message in the function unpause().

2.2.3 Add validation to avoid duplicate addresses assignment

Status Fixed in Version 2

Introduced by Version 1

Description In the contract Fiat24CardAuthorizationMarqeta, the function setTreasuryAddress() should check whether the input parameter treasury_ is the same as the current value of the state variable treasuryAddress.

```
function setTreasuryAddress(address treasury_) external {
    if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender()))) revert Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
    require(treasury_ != address(0), "Invalid treasury address");

    address oldTreasury = treasuryAddress;
    treasuryAddress = treasury_;

    emit TreasuryAddressUpdated(oldTreasury, treasury_);
}
```

Listing 2.7: src/Fiat24CardAuthorizationMarqeta.sol

Suggestion Add a check for duplicate addresses.

2.2.4 Implement event emissions for all state-changing functions

Status Fixed in Version 2

Introduced by Version 1

Description The contract `Fiat24CardAuthorizationMarqeta` contains several critical setter functions that modify state variables without emitting corresponding events. It is recommended to implement proper event emissions for all state-changing functions to improve contract observability and code clarity.

```
function setFiat24CryptoRelayAddress(address fiat24CryptoRelayAddress_) external {
    if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender()))) revert
        Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
    require(fiat24CryptoRelayAddress_ != address(0), "Invalid relay address");
    fiat24CryptoRelayAddress = fiat24CryptoRelayAddress_;
}
```

Listing 2.8: src/Fiat24CardAuthorizationMarqeta.sol

```
function setInterchange(uint256 interchange_) public {
    if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender()))) revert
        Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
    if (interchange_ > 100) revert Fiat24CardAuthorizationMarqeta__InterchangeOutOfRange(
        interchange_);
    interchange = interchange_;
}
```

Listing 2.9: src/Fiat24CardAuthorizationMarqeta.sol

```
function setExchangeSpread(uint256 newExchangeSpread) public {
    if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender()))) revert
        Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());

    require(newExchangeSpread > 9000 && newExchangeSpread <= 11000, "Spread must be between
        9000 and 11000");
    exchangeSpread = newExchangeSpread;
}
```

Listing 2.10: src/Fiat24CardAuthorizationMarqeta.sol

```
function setMarketClosed(bool newMarketClosed) public {
    if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender()))) revert
        Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
    marketClosed = newMarketClosed;
}
```

Listing 2.11: src/Fiat24CardAuthorizationMarqeta.sol

```
function setAlternativeInputTokens(address outputToken, address[] calldata inputTokens)
    external {
    if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender()))) revert
        Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
    alternativeInputTokens[outputToken] = inputTokens;
}
```

Listing 2.12: src/Fiat24CardAuthorizationMarqeta.sol

Suggestion Implement event emissions for all state-changing functions.

2.2.5 Early revert for insufficient token allowance

Status Fixed in Version 2

Introduced by Version 1

Description In the function _tryDirectTokenSwap(), the protocol attempts to obtain the output token (e.g., USD24) from the sender by swapping the other provided tokens. However, the payment may fail in the function authorize() if the user has not granted sufficient allowance for the output token. To optimize gas usage and improve code readability, it is recommended to implement an early revert if the user's allowance is insufficient.

```
function increment(
    string memory authorizationToken_,
    string memory cardId_,
    uint256 tokenId_,
    address cardCurrency_,
    string memory transactionCurrency_,
    address settlementCurrency_,
    uint256 transactionAmount_,
    uint256 settlementAmount_
) public {
    if (!(hasRole(AUTHORIZER_ROLE, _msgSender()))) revert
        Fiat24CardAuthorizationMarqeta__NotAuthorizer(_msgSender());
    if (paused()) revert Fiat24CardAuthorizationMarqeta__Suspended();
    if (!validXXX24Tokens[settlementCurrency_]) revert
        Fiat24CardAuthorizationMarqeta__NotValidSettlementCurrency(settlementCurrency_);
    address sender = IFiat24Account(fiat24AccountAddress).ownerOf(tokenId_);
    address booked = IFiat24Account(fiat24AccountAddress).ownerOf(CARD_BOOKED);
    address paidCurrency = cardCurrency_;
    uint256 paidAmount;

    address txnToken = XXX24Tokens[transactionCurrency_];

    if (validXXX24Tokens[txnToken]) {
        if (
            IERC20Upgradeable(txnToken).balanceOf(sender) >= transactionAmount_
            && IERC20Upgradeable(txnToken).allowance(sender, address(this)) >=
                transactionAmount_
        ) {

            paidCurrency = txnToken;
            paidAmount = transactionAmount_;
        } else {

            paidAmount = transactionAmount_ * getRate(txnToken, cardCurrency_)
                * getSpread(txnToken, cardCurrency_, false) / 100000000;

            if (cardCurrency_ == usd24Address) {
                uint256 userUsd24Balance = IERC20Upgradeable(usd24Address).balanceOf(sender);
                uint256 userUsd24Allowance = IERC20Upgradeable(usd24Address).allowance(sender,
                    address(this));
                uint256 availableUsd24 = userUsd24Balance < userUsd24Allowance ?
                    userUsd24Balance : userUsd24Allowance;

                if (availableUsd24 < paidAmount) {
                    require(_trySwapAlternativeTokens(usd24Address, sender, paidAmount -
                        availableUsd24),
                        "Failed to swap alternative tokens for USD24");
                }
            }
        }
```

Listing 2.13: src/Fiat24CardAuthorizationMarqeta.sol

Suggestion Revise the logic accordingly.

2.3 Note

2.3.1 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `OPERATOR_ADMIN_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, the `OPERATOR_ADMIN_ROLE` can modify critical states like `marketClosed`. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.2 OpenZeppelin `Initializable` upgrade migration risks

Introduced by `Version 1`

Description The project currently uses OpenZeppelin's Initializable contract (`v4.4.1`) to implement upgradeable contracts. It is important to note that Initializable versions `v5.0.0` and later introduce `ERC-7201` namespaced storage to mitigate storage collision risks. This change relocates initialization state variables from direct storage slots (e.g., `_initialized`) to namespaced storage structures (e.g., `$._initialized`). When upgrading to newer Initializable versions, the project must ensure proper migration of initialization state to prevent contracts from being reinitialized, which could lead to severe security vulnerabilities including state corruption and unauthorized access.

2.3.3 Hardcoded values on initialization

Introduced by `Version 1`

Description The contract `Fiat24CardAuthorizationMarqeta` initializes the variable `exchangeRates` with hardcoded values for fiat24 token conversions. The project should ensure that these exchange rates will be dynamically assigned based on real-time market conditions or implement a mechanism for periodic updates to maintain accurate valuations. Additionally, the variables `exchangeSpread`, `marketClosedSpread`, and `interchange` are also initialized with fixed values that should require similar dynamic adjustment capabilities.

```
    exchangeRates[usd24Address][usd24Address] = 10000;
    exchangeRates[usd24Address][eur24Address] = 9168;
    exchangeRates[usd24Address][chf24Address] = 8632;
    exchangeRates[usd24Address][gbp24Address] = 7674;
    exchangeRates[usd24Address][cnh24Address] = 70885;
```

Listing 2.14: src/Fiat24CardAuthorizationMarqeta.sol

```
    exchangeSpread = 10150;
    marketClosedSpread = 10005;
    interchange = 1;
```

Listing 2.15: src/Fiat24CardAuthorizationMarqeta.sol

2.3.4 Ensure timely updates of `exchangeRate` in `TokenPairConfig`

Introduced by `Version 1`

Description The `Fiat24CardAuthorizationMarqeta` contract defines a `TokenPairConfig` struct to configure the exchange rate and active status for swapping alternative tokens to `USD24`. The project must ensure that exchange rates for active alternative tokens are updated timely.

```
function configureTokenPair(
    address tokenIn_,
    address tokenOut_,
    uint256 exchangeRate_,
    bool isActive_
) external {
    if (!(hasRole(OPERATOR_ADMIN_ROLE, _msgSender())) || !(hasRole(CRYPTO_CONFIG_UPDATER_ROLE,
        _msgSender()))) revert Fiat24CardAuthorizationMarqeta__NotOperator(_msgSender());
    require(tokenIn_ != address(0), "Invalid input token address");
    require(tokenOut_ != address(0), "Invalid output token address");
    require(tokenIn_ != tokenOut_, "Input and output tokens cannot be the same");

    if (isActive_) {
        require(exchangeRate_ > 0, "Exchange rate must be > 0");
    }

    tokenPairConfigs[tokenIn_][tokenOut_] = TokenPairConfig({
        exchangeRate: exchangeRate_,
        isActive: isActive_
    });

    emit TokenPairConfigured(tokenIn_, tokenOut_, exchangeRate_, isActive_);
}
```

Listing 2.16: src/Fiat24CardAuthorizationMarqeta.sol

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS