VIDMA

milestoneBased

SMART CONTRACT AUDIT

Project: milestoneBased
Date: August 17th, 2022

TABLE OF CONTENTS

SUMMARY

Vidma is pleased to present this audit report outlining our assessment of code, smart contracts, and other important audit insights and suggestions for management, developers, and users.

milestoneBased is building an innovative startup financing solution that leverages smart contracts and a Decentralized Autonomous Organization (DAO) to make funds management more efficient, flexible, standardized, and secure for both investors and startup teams.

The audited scope included a contract that allows swapping MILE tokens to sMILE tokens by the market price. The swapping process allows conversation some amount of ERC20 tokens to MILE before staking by PancakeSwap.Anyone is able to create sMILE by depositing $MILE. The initial conversion rate is 1-1 but may change if the $MILE rewards are added to the MILE/sMILE pool.

There are defined entities for interactions on the platform, that were also included in the scope of the audit. User, company, and project entities for representation of the user, company, or project capital and actions. Only entity contracts may get sMILE tokens from the contract.

Audited smart contracts are written according to security best practices.

During the audit process, the Vidma team found several issues, including those with critical severity. A detailed summary and the current state are displayed in the table below.

Severity of the issue	Total found	Resolved	Unresolved
Critical	1 issue	1 issue	0 issues
High	1 issue	1 issue	0 issues
Medium	4 issues	1 issue	3 issues
Low	7 issues	7 issues	0 issues
Informational	7 issues	6 issues	1 issue
Total	**20 issues**	**16 issues**	**4 issues**

After evaluating the findings in this report and the final state after fixes, the Vidma auditors can state that the to make the contracts fully operational and secure a couple of fixes should be done by milestoneBased team. Under the given circumstances, we set the following risk level:

Our auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. This approach helps us adequately and sequentially evaluate the quality of the code. Code style, optimization of the contracts, the number of issues, and risk level of the issues are all taken into consideration. The Vidma team has developed a transparent scoring system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Please note that the points are deducted out of 100 for each and every issue on the list of findings (according to the current status of the issue). Issues marked as "not valid" are not subject to point deduction.

Based on the **overall result of the audit**, the Vidma audit team grants the following score:

In addition to manual check and static analysis, the auditing team has conducted a number of integrated autotests to ensure the given codebase has an adequate performance and security level.

The test results and the coverage can be found in the accompanying section of this audit report.

Please be aware that this audit does not certify the definitive reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the Vidma audit team. If the code is still under development, we highly recommend running one more audit once the code is finalized.

SCOPE OF WORK

milestoneBased

milestoneBased is the first company to leverage a blockchain DAO and escrow smart contract capabilities, in an automated governance and milestone achievement management platform.

Within the scope of this audit, two independent auditors thoroughly investigated the given codebase and analyzed the overall security and performance of the smart contracts.

The audit was conducted from July 26, 2022 to August 17, 2022. The outcome is disclosed in this document.

The scope of work for the given audit consists of the following contracts:

- BaseEntity;
- UserEntity;
- CompanyEntity;
- ProjectEntity;
- EntityFactory;
- SingleSignEntityStrategy;
- SwapMILE;

The source code was taken from the following **source**:
https://bitbucket.org/applicature/milestonebased.contracts

Initial commit submitted for the audit:
e544674e3152e69e4ab78b8e59078162bbab8d77

Last commit reviewed by the auditing team:
0918d9f4445420c1d36aa5579b79d44db944e6cb

WORKFLOW OF THE AUDITING PROCESS

Vidma audit team uses the most sophisticated and contemporary methods and well-developed techniques to ensure contracts are free of vulnerabilities and security risks. The overall workflow consists of the following phases:

Phase 1: The research phase

Research

After the Audit kick-off, our security team conducts research on the contract's logic and expected behavior of the audited contracts.

Documentation reading

Vidma auditors do a deep dive into your tech documentation with the aim of discovering all the behavior patterns of your codebase and analyzing the potential audit and testing scenarios.

The outcome

At this point, the Vidma auditors are ready to kick off the process. We set the auditing strategies and methods and are prepared to conduct the first audit part.

Phase 2: Manual part of the audit

Manual check

During the manual phase of the audit, the Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract. The initial commit as stated in the agreement is taken into consideration.

Static analysis check

Static analysis tools are used to find any other vulnerabilities in smart contracts that were missed after a manual check.

The outcome

An interim report with the list of issues.

Phase 3: Testing part of the audit

Integration tests

Within the testing part, Vidma auditors run integration tests using the Truffle or Hardhat testing framework. The test coverage and the test results are inserted in the accompanying section of this audit report.

The outcome

Second interim report with the list of new issues found during the testing part of the audit process.

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For simplicity in reviewing the findings in this report, Vidma auditors classify the findings in accordance with the severity level of the issues. (from most critical to least critical).

All issues are marked as “Resolved” or “Unresolved”, depending on if they have been fixed by milestoneBased or not. The issues with “Not Valid” status are left on the list of findings but are not eligible for the score points deduction.

The latest commit with the fixes reviewed by the auditors is indicated in the “Scope of Work” section of the report.

The Vidma team always provides a detailed description of the issues and recommendations on how to fix them.

Classification of found issues is graded according to 6 levels of severity described below:

Critical

The issue affects the contract in such a way that funds may be lost or allocated incorrectly, or the issue could result in a significant loss.

Example: Underflow/overflow, precisions, locked funds.

High

The issue significantly affects the ability of the contract to compile or operate. These are potential security or operational issues.

Example: Compilation errors, pausing/unpausing of some functionality, a random value, recursion, the logic that can use all gas from block (too many iterations in the loop), no limitations for locking period, cooldown, arithmetic errors which can cause underflow, etc.

Medium

The issue slightly impacts the contract's ability to operate by slightly hindering its intended behavior.

Example: Absence of emergency withdrawal of funds, using assert for parameter sanitization.

Low

The issue doesn't contain operational or security risks, but are more related to optimization of the codebase.

Example: Unused variables, inappropriate function visibility (public instead of external), useless importing of SCs, misuse or disuse of constant and immutable, absent indexing of parameters in events, absent events to track important state changes, absence of getters for important variables, usage of string as a key instead of a hash, etc.

Informational

Are classified as every point that increases onboarding time and code reading, as well as the issues which have no impact on the contract's ability to operate.

Example: Code style, NatSpec, typos, license, refactoring, naming convention (or unclear naming), layout order, functions order, lack of any type of documentation.

MANUAL REPORT

The unhandled case when UserEntity contract hasn't owner

Critical | Resolved

In contract UserEntity there is a possibility to remove the last owner from the contract. After that the contract becomes useless (no one will be able to call the functions that the contract has).

Recommendation:

Consider adding a requirement that will check whether it's the last owner of the contract.

Incorrect commission calculation

High | Resolved

In contract SwapMILE.sol in function *calculateWithdrawFee()* return incorrect commission calculation. After creating a request for withdrawal and calling function *withdraw()*, withdrawal fee is always 40% before the withdrawal period and after the cooldown period is passed - 10%.

The next line of code will always return zero:

```
(currentTime - creationTimestamp) / cooldownPeriod
```

Recommendation:

Consider changing the commission calculation formula.

Unchecked returned value of ERC20 token *transfer()* and *withdrawFromStaking()* functions

Medium | Unresolved

There are some token transfer and approve calls in contracts but there is no check on the result of the function execution.

Another case when the function execution result isn't changed appears in the contract BaseEntity when the function *withdrawFromStaking()* is called.

Recommendation:

Consider handling the returned result properly and/or use SafeERC20 library for all *transfer()* or *transferFrom()* function in the next functions:

- *BaseEntity.withdrawFromEntity()*;
- *ProjectEntity._managingTokenSupply()*;
- *CompanyEntity._managingTokenSupply()*;
- *UserEntity.stake()*;
- *UserEntity.stakeTo()*;
- *UserEntity.swapStake()*;
- *UserEntity.swapStakeTo()*;
- *SwapMILE.withdraw()*;
- *BaseEntity.withdrawFromStaking()*.

Consider implement check of the return value for the ERC20 *approve()* function or use SafeERC20 *safeIncreaseAllowance()* for the next functions:

- *BaseEntity.__BaseEntity_init*;
- *BaseEntity._stake*;
- *BaseEntity._stakeTo*;
- *BaseEntity._swapStake*;
- *BaseEntity._swapStakeTo*;
- *UserEntity.approve*;
- *SwapMILE.swapStake*;
- *SwapMILE.swapStakeTo*.

Loss of whitelist addresses that are used on the milestoneBased platform

Medium | Unresolved

In contract EntityFactory.sol in function *updateWhiteList()* owner setting new address of WhiteList contract (It is used by sMILE token for transactions restriction), after calling function *updateWhiteList()* the newer WhiteList contract hasn't addresses that was before.

Recommendation:

Consider setting up addresses from the old WhiteList contract to the new one.

There is no possibility to add admin

Medium | Unresolved

In the contract BaseEntity, one of the access roles should be an administrator but in the current state, there is no way to add a new admin as the functions *addAmin()* and *removeAdmin()* that are supposed to do this are internal and never called in the source code.

Recommendation:

Consider adding the possibility to manage to add and remove admin properly.

Unsafe transfer usage

Medium | Resolved

In contract SwapMILE.sol in function *withdrawUnusedBNB()* is used *transfer()* function for transferring unexpected BNB from the contract. Since transfers have fixed 2300 gas for executing it is unsafe to use it with new versions of solidity.

Recommendation:

Consider to use *call()* instead of *transfer()*.

Useless requirements

Low | Resolved

In contract EntityFactory.sol in functions *createCompanyEntity()*, *createProjectEntity()*, and *createUserEntity()* is used useless require statement:

```
require(
      !companiesEntities[msg.sender] && !
projectsEntities[msg.sender],
      "The caller cannot be other entities"
    );
```

CompanyEntity or ProjectEntity can't call functions listed above, so the requirement is useless.

In contract EntityFactory.sol in functions *addUser()* and *removeUser()* is used useless require statement:

```
require(entityOwner != address(0), "User address cannot be zero");
```

User is calling function *addOwner()* contract UserEntity.sol contract that have require statement for zero address then UserEntity is calling *addUser()* that have the same require statement.

In contract EntityFactory.sol in function *removeUser()* is used useless require statement:

```
require(
   ownersOfUserEntity[entityOwner] == msg.sender,
     "Passed address isn't owner of user entity that is caller"
   );
```

UserEntity has the same require statement.

Recommendation:

Consider removing useless requirements.

Absence of event

Low | Resolved

In contract BaseEntity.sol in function *BaseEntity_init()* when initial owner is added there is no AddedOwner event.

Recommendation:

Consider adding AddedOwner event to *BaseEntity_init()* function.

Useless function

Low | Resolved

Contract SwapMILE.sol hasn't *fallback()* or *receive()* for receiving native coins of the blockchain. So in the contract SwapMILE.sol there is useless function *withdrawUnusedBNB()* (there is no possibility to send native coins to contract SwapMILE)

Recommendation:

Consider removing useless function *withdrawUnusedBNB()*.

Functions visibility optimization

Low | Resolved

There are some functions that never called in the contarct itself with visibility *public* so they can be marked as external.

Recommendation:

Consider changing visibility from public to external to safe gas usage on calling in the next functions:

- *ProjectEntity.swapOwner*;
- *CompanyEntity.swapOwner*;
- *BaseEntity.getOwners*.

Lack of zero-check

Low | Resolved

In SingleSignEntityStrategy's constructor and *setTrustedSigner()* function there is no check if the *trustedSigner* address isn't zero address.

Recommendation:

Consider adding a check for zero address in the constructor and function.

Local variables shadowing

Low | Resolved

There is a situation in the functions when the local variable *owner* shadows the existing *owner* variable from the Ownable/OwnableUpgradeable contract. It was detected in the next function: *EntityFactory.getUserEntityOfOwner()*, *EntityFactory.onlyValidSignature()*, *SingleSignEntityStrategy.getNonce()*, *SingleSignEntityStrategy._useNonce()*.

Recommendation:

Consider renaming the local variables in the mentioned above functions to avoid shadowing another component.

Inappropriate function visibility

Low | Resolved

In contract SingleSignEntityStrategy function *getDigest()* is public so everyone can call it and the function triggers change of the nonce for the entity owner address used for the signature.

Recommendation:

Consider change visibility to private to avoid unexpected flow of changing nonce for the entity owner address by third party addresses.

The same signature arguments for *createCompanyEntity()* and *createProjectEntity()* functions

Informational | Unresolved

The signature from BE for creating a company or project entity is the same. The user can use it to create a company entity when BE passed a signature for creating a project entity or vice versa.

Recommendation:

Consider changing a signature message if it can lead to unexpected case.

Useless (if-and-else) statement

Informational | Resolved

In contract EntityFactory.sol in function *updateUpgradeableBeacon()* there is useless (if-and-else) statement:

```
else if (entityType == EntityType.ProjectEntity) {
      projectEntityBeacon = UpgradeableBeacon(newBeacon);
}
Consider changing from `else if` to `else`:
else {
      projectEntityBeacon = UpgradeableBeacon(newBeacon);
}
```

Recommendation:

Consider changing from *else if* to *else* statement.

Useless import

Informational | Resolved

In contract SwapMILE there is *ReentrancyGuardUpgradeable* import which only is initializing (*ReentrancyGuard_init()*) but never is used. There is also a useless commented import IPancakeRouter02.

Recommendation:

Consider removing useless import.

Code optimization

Informational | Resolved

In the SwapMILE contract there is the next code (10**18) that can be changed to (1 ether) for code optimization.

Recommendation:

Consider changing next code (10**18) for code optimization.

Useless events

Informational | Resolved

The events that are never used: IBaseEntity.sol (*FundingRoadmap*, *CancelRequestWithdraw*).

Recommendation:

Consider removing useless events.

Functions order at contracts

Informational | Resolved

The functions in contracts BaseEntity, SingleSignEntityStrategy and SwapMile are not grouped according to their visibility and order.

Functions should be grouped according to their visibility and ordered in the following way:

- *constructor*;
- *receive function (if exists)*;
- *fallback function (if exists)*;
- *external*;
- *public*;
- *internal*;
- *private*.

Recommendation:

Consider changing functions order according to solidity documentation and style guide.

Lack of NatSpec annotations

Informational | Resolved

Smart contracts SingleSignEntityStrategy and SwapMILE is not fully covered by NatSpec annotations.

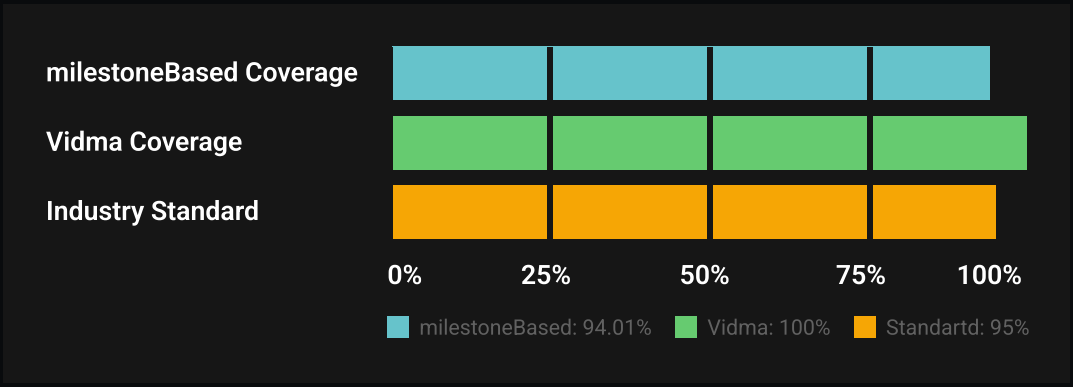
Recommendation:

Consider to cover by NatSpec all contract`s methods.

TEST RESULTS

To verify the security of contracts and the performance, a number of integration tests were carried out using the Hardhat testing framework.

In this section, we provide both tests written by milestoneBased and tests written by Vidma auditors.



It is important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the milestoneBased repository. We write totally separate tests with code coverage of a minimum of 95% to meet the industry standards.

Tests written by milestoneBased

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	94.01	80.58	91.00	94.34
BaseEntity.sol	93.65	60.00	89.47	94.20
CompanyEntity.sol	42.31	30.00	60.00	44.00
EntityFactory.sol	98.53	79.41	92.86	98.51
ProjectEntity.sol	76.92	70.00	80.00	76.00
SingleSignEntityStrategy.sol	100.00	80.00	100.00	100.00
SwapMILE.sol	100.00	96.00	100.00	100.00
UserEntity.sol	100.00	58.33	100.00	100.00
All Files	**94.01**	**80.58**	**91.00**	**94.34**

Test Results

```
Contract: BaseEntity
  initialization
    ✓ should initialize variables correctly (126ms)
    ✓ should add deployer address to linked list correctly
  addOwner
    ✓ should add new address to linked list correctly (129ms)
```

```
removeOwner
  ✓ should remove address from linked list correctly (118ms)
swapOwner
  ✓ should replace owner address correctly (276ms)
getOwners
  ✓ should return array of owners correctly (451ms)
stake
  ✓ should stake tokens correctly (926ms)
  ✓ should catch event
stakeTo
  ✓ should stake tokens correctly (489ms)
  ✓ should catch event
swapStake
  ✓ should stake tokens correctly (536ms)
swapStakeTo
  ✓ should stake tokens to address correctly (479ms)
requestWithdraw
  ✓ should create withdraw request to SwapMILE contract
    correctly (231ms)
  ✓ should catch event
cancelWithdraw
  ✓ should cancel withdraw request correctly (101ms)
  ✓ should withdraw MILE token from SwapMILE contract to the
    Entity contract correctly (633ms)
withdrawFromEntity
  ✓ should revert if the amount exceeds the balance of the
    contract (353ms)
  ✓ should withdraw tokens from the Entity contract to the
    recipient address correctly (136ms)
approve
  ✓ should approve tokens on entity address correctly (167ms)
  ✓  should catch event

Contract: EntityFactory
  Constructor
  ✓ should initialize variables correctly
  UpdateUpgradeableBeacon
  ✓ should revert [ZERO ADDRESS] (41ms)
  ✓ should update entity upgradeable beacon address
    correctly (311ms)
  ✓ should catch event
  Create User Entity
```

```
  Create User Entity
    ✓ should create entity contract correctly (210ms)
    ✓ should revert if the caller has already created
      UserEntity (95ms)
    ✓ should catch event
  Create Company Entity
    ✓ should create entity contract correctly (189ms)
    ✓ should catch event
  Create Project Entity
    ✓ should create entity contract correctly (184ms)
    ✓ should catch event
  updateWhiteList
    ✓ should revert [ZERO_ADDRESS]
    ✓ should change WhiteList contract address correctly (104ms)
    ✓ should catch event
  updatedStrategy
    ✓ should revert [ZERO_ADDRESS]
    ✓ should change WhiteList contract address correctly (101ms)
    ✓ should catch event

Contract: Entity Admin role test
  Check for correct deploy
    ✓ EntityFactory must return correct owner
  Creating entities
    ✓ Must create userEntity correctly (163ms)
    ✓ Must fail when signature is not valid (66ms)
    ✓ Must fail if you try create second userEntity (73ms)
    ✓ Must create companyEntity correctly by userEntity (196ms)
    ✓ Must create projectEntity correctly by userEntity (194ms)
  Entities` admin and owner roles
    ✓ Must add new owner to all children entities of userEntity
      after adding new owner to userEntity (69ms)
    ✓ Must revert if administrator try to swap owners (60ms)
    ✓ Must revert if owner of company entity try to swap owners
      with not user entity (63ms)
    ✓ Must revert if owner of project entity try to swap owners
      with not user entity (75ms)
  Definition of Entity type
    ✓ Entity factory must return correct type of entity
  Adding, Swapping, Removing Owner
    ✓ Entities must return correct list of owners (46ms)
    ✓ Entities must swap owner correctly (593ms)
```

```
  ✓ UserEntity must remove Owner correctly (68ms)
Staking
  ✓ Must fail to stake from user Entity if caller isn't owner
  ✓ Must fail to stake from user Entity if caller hasn't approved
    tokens to the userEntity (233ms)
  ✓ Must stake from user Entity correctly using user's
    tokens (404ms)
  ✓ Must stake from user Entity correctly using entity's
    tokens (1150ms)
  ✓ Must fail to stake to not entity contract (267ms)
  ✓ Must stake to another entity from user Entity correctly
    using user's tokens (233ms)
  ✓ Must fail to stake from project/company Entity if caller
    isn't admin or owner (165ms)
  ✓ Must fail to stake from project/company Entity if caller
    hasn't approved tokens to the userEntity (115ms)
  ✓ Must stake from project/company Entity correctly using
    user's tokens (198ms)
  ✓ Must stake from project/company Entity correctly using
    the entity's tokens (201ms)
  ✓ Must stake from project/company Entity correctly using the
    user entity's tokens (234ms)
  ✓ Must fail to stake to not entity contract (174ms)
  ✓ Must stake to entity contract from project/company Entity
    correctly using user's tokens (208ms)
  ✓ Must stake from project/company Entity correctly using the
    entity's tokens (183ms)
  ✓ Must stake from project/company Entity correctly using the
    parent user entity's tokens (231ms)
Withdraw from Entity
  ✓ Must withdraw from Entity correctly (115ms)
Withdraw from SwapMILE and requestWithdraw keeping
  ✓ Check requests to withdraw
  ✓ getRequestsByEntity: must return correct amount of request
    with correct offset (801ms)
  ✓ getAvailableSMILEToWithdraw: must return correct amount of
    requested sMLIE tokens which entity can swap to MILE (474ms)
  ✓ getRequestedSMILE: must return correct amount of requested
    sMLIE tokens to withdraw (604ms)
  ✓ cancelWithdraw: must cancel withdraw request correctly (171ms)
  ✓ cancelWithdraw: must fail to cancel already canceled withdraw
    request (169ms)
```

```
    ✓ withdraw: must execute withdraw request  correctly (260ms)
    ✓ withdraw: must fail to withdraw already executed withdraw
      request (1307ms)
    ✓ withdraw: must fail if withdraw request is expired (223ms)

Contract: SingleSignEntityStrategy
  functions:
    setTrustedSigner
      ✓ should revert if caller is not owner
      ✓ should revert if trusted signer with such an address is
        already setted
      ✓ should set trusted signer correctly (38ms)
      ✓ should catch event
    isValid
      ✓ should return true with a valid signature (51ms)
      ✓ should return false with an invalid signature (115ms)
      ✓ should revert if argumentsU256 not one (46ms)
      ✓ should revert if argumentsB32 not two (52ms)

Contract: SwapMile
  functions:
    fee calculation check
      ✓ should get fee after immidiatly withdraw (102ms)
      ✓ should get fee after 1/5 cooldown period (89ms)
      ✓ should get fee after 1/2 cooldown period (95ms)
      ✓ should get fee after 3/5 cooldown period (100ms)
      ✓ should get fee after cooldown period (114ms)
    initialize
      ✓ should revert when admin is zero address (220ms)
      ✓ should revert when MILE is zero address (129ms)
      ✓ should revert when sMILE is zero address (139ms)
      ✓ should revert when busdToken is zero address (153ms)
      ✓ should revert when wbnbToken is zero address (136ms)
      ✓ should revert when router is zero address (129ms)
      ✓ should revert when entityFactory is zero address (221ms)
      ✓ should revert when cooldownPeriod is zero (126ms)
      ✓ should revert when withdrawPeriod is zero (144ms)
      ✓ should revert when cooldownPeriod is less than
        withdrawPeriod (143ms)
    stake
      ✓ should revert: Zero amount (59ms)
      ✓ should revert: Recipient isn't entity (54ms)
```

```
    ✓ should stake (104ms)
    ✓ should catch event
  stakeTo
    ✓ should revert: Zero amount (236ms)
    ✓ should revert: Recipient isn't entity (162ms)
    ✓ should stake (98ms)
    ✓ should catch event
  swapStake
    ✓ should revert: Zero amount (99ms)
    ✓ should revert: Recipient isn't entity (98ms)
    ✓ should revert: Zero token address (81ms)
    ✓ should stake (133ms)
    ✓ should catch event
  swapStakeTo
    ✓ should revert: Zero amount (67ms)
    ✓ should revert: Recipient isn't entity (55ms)
    ✓ should revert: Zero token address (73ms)
    ✓ should stake (150ms)
    ✓ should catch event
  requestWithdraw
    ✓ should revert: Zero amount (51ms)
    ✓ should revert: Not enough MILE tokens to transfer (45ms)
    ✓ should revert: Not enough sMILE to withdraw (111ms)
    ✓ should create withdraw request (74ms)
    ✓ should catch event
  withdraw
    ✓ should revert: Request does not exis (107ms)
    ✓ should revert: Not enough sMILE tokens on user
      balance (149ms)
    ✓ withdraw immediate (224ms)
    ✓ withdraw in 1/2 of standard cooldown duration (191ms)
    ✓ withdraw after cooldown period (202ms)
    ✓ withdraw after withdraw period (82ms)
    ✓ should catch event
  cancel withdraw
    ✓ should revert: Request does not exis (49ms)
    ✓ should cancel (48ms)
    ✓ should catch event
   add rewards
    ✓ should revert: Zero amount (50ms)
    ✓ should add reward (116ms)
    ✓ should catch event
```

```
withdraw unused MILE
  ✓ should revert: Caller not admin (163ms)
  ✓ should revert: Zero amount (38ms)
  ✓ should revert: Zero address
  ✓ should revert: Not enough MILE to withdraw (72ms)
  ✓ should withdraw (79ms)
  ✓ should catch event
withdraw unused sMILE
  ✓ should revert: Caller not admin (174ms)
  ✓ should revert: Zero amount (45ms)
  ✓ should revert: Zero address (47ms)
  ✓ should revert: Not enough sMILE to withdraw (66ms)
  ✓ should withdraw (123ms)
  ✓ should catch event
get available sMILE to withdraw
  ✓ should calculate correctly
getRequestIdsByEntity & getRequestsByEntity
  ✓ should fail if entity doesn't have any withdrawal requests
  ✓ should return correct array of withdrawal request
  ✓ should return correct array of withdrawal request
  ✓ should return correct array of withdrawal request (159ms)
  ✓ should return correct array of withdrawal request with
    offset (95ms)
set cooldown period
  ✓ should revert: Caller not admin (153ms)
  ✓ should revert: Zero value
  ✓ should set
  ✓ should catch event
set withdraw period
  ✓ should revert: Caller not admin (147ms)
  ✓ should revert: Zero value (44ms)
  ✓ should set
  ✓ should catch event
set entityFactory contract
  ✓ should revert: Caller not admin
  ✓ should revert: Zero value (48ms)
  ✓ should set
  ✓ should catch event
get MILE price
  ✓ should calculate correctly
get sMILE price
  ✓ should calculate correctly
```

```
167 passing (37s)
```

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
BaseEntity.sol	100.00	100.00	100.00	100.00
CompanyEntity.sol	100.00	100.00	100.00	100.00
EntityFactory.sol	100.00	100.00	100.00	100.00
ProjectEntity.sol	100.00	100.00	100.00	100.00
SingleSignEntityStrategy.sol	100.00	100.00	100.00	100.00
SwapMILE.sol	100.00	100.00	100.00	100.00
UserEntity.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**100.00**	**100.00**	**100.00**

Test Results

```
Contract: CompanyEntity
  CompanyEntity Test Cases
    CompanyEntity Deployment Test Cases
      ✓ shouldn't allow initializing contract twice (261ms)
      ✓ shouldn't allow initializing contract with zero
        addresses (180ms)
```

```
    ✓ should deploy contract correctly (200ms)
  CompanyEntity Owner Test Cases
    ✓ shouldn't allow calling `swapOwner` by not the owner
    ✓ shouldn't allow swapping if the new owner isn't UserEntity
      contract (40ms)
    ✓ should return whether passed user is admin
    ✓ should allow to swap an owner with another address (224ms)
  CompanyEntity Staking Test Cases
    ✓ should calling `stake` from user's wallet
      successfully (103ms)
    ✓ should calling `stake` from parent userEntity
      successfully (99ms)
    ✓ should calling `stake` from current entity
      successfully (108ms)
    ✓ should calling `stakeTo` from user's wallet
      successfully (128ms)
    ✓ should calling `stakeTo` from parent userEntity
      successfully (128ms)
    ✓ should calling `stakeTo` from the current entity
      successfully (90ms)
    ✓ should calling `swapStake` from user's wallet
      successfully (165ms)
    ✓ should calling `swapStake` from parent userEntity
      successfully (260ms)
    ✓ should calling `swapStake` from the current entity
      successfully (199ms)
    ✓ should calling `swapStakeTo` from user's wallet
      successfully (193ms)
    ✓ should calling `swapStakeTo` from parent userEntity
      successfully (165ms)
    ✓ should calling `swapStakeTo` from the current entity
      successfully (113ms)

Contract: EntityFactory
  EntityFactory Test Cases
    EntityFactory Deployment Test Cases
      ✓ shouldn't allow initializing contract twice
      ✓ shouldn't allow initializing contract with zero
        addresses (315ms)
      ✓ should deploy EntityFactory contract correctly (75ms)
    EntityFactory Creating Entities Test Cases
      ✓ should calling `addUser` correctly (64ms)
```

```
    ✓ should calling `removeUser` correctly (91ms)
    ✓ shouldn't allow adding a user if the user already
      has UserEntity (291ms)
    ✓ should adding and removing users correctly (143ms)
    ✓ shouldn't allow creating user entity if a signature is
      invalid (55ms)
    ✓ shouldn't allow creating user entity if a caller hasn't
      factory contract role (187ms)
    ✓ shouldn't allow creating user entity twice from the same
      caller (53ms)
    ✓ shouldn't allow creating company entity if the caller isn't
      user entity
    ✓ shouldn't allow creating company entity if a signature is
      invalid (102ms)
    ✓ shouldn't allow creating company entity if a caller hasn't
      factory contract role (135ms)
    ✓ should reverting if caller is company entity but isn't
      user entity (98ms)
    ✓ shouldn't allow creating project entity if the caller isn't
      user entity
    ✓ shouldn't allow creating project entity if a signature is
      invalid (125ms)
    ✓ shouldn't allow creating project entity if a caller hasn't
      factory contract role (131ms)
    ✓ should reverting if caller is project entity but isn't
      user entity (105ms)
    ✓ should creating a user entity successfully
    ✓ should creating a company entities successfully (242ms)
    ✓ should creating a project entities successfully (145ms)
  EntityFactory Update Test Cases
    ✓ shouldn't allow update upgradeable beacon by not the
      owner (122ms)
    ✓ shouldn't allow update upgradeable beacon with zero address
    ✓ shouldn't allow update EntityStrategy contract by not the
      owner (52ms)
    ✓ shouldn't allow update EntityStrategy contract with zero
      address
    ✓ shouldn't allow update WhiteList contract by not the
      owner (39ms)
    ✓ shouldn't allow update WhiteList contract with zero address
    ✓ should calling `updateWhiteList` successfully (49ms)
    ✓ should calling `updatedStrategy` successfully
```

```
      ✓ should calling `updateUpgradeableBeacon` successfully (383ms)

Contract: ProjectEntity
  ProjectEntity Test Cases
    ProjectEntity Deployment Test Cases
      ✓ shouldn't allow initializing contract twice (144ms)
      ✓ shouldn't allow initializing contract with zero
        addresses (98ms)
      ✓ should deploy contract correctly (94ms)
    ProjectEntity Owner Test Cases
      ✓ shouldn't allow calling `swapOwner` by not the owner
      ✓ shouldn't allow swapping if the new owner isn't
        UserEntity contract
      ✓ should return whether passed user is admin
      ✓ should allow to swap an owner with another address (145ms)
    ProjectEntity Staking Test Cases
      ✓ should calling `stake` from user's wallet
        successfully (90ms)
      ✓ should calling `stake` from parent userEntity
        successfully (127ms)
      ✓ should calling `stake` from current entity
        successfully (121ms)
      ✓ should calling `stakeTo` from user's wallet
        successfully (198ms)
      ✓ should calling `stakeTo` from parent userEntity
        successfully (190ms)
      ✓ should calling `stakeTo` from the current entity
        successfully (155ms)
      ✓ should calling `swapStake` from user's wallet
        successfully (242ms)
      ✓ should calling `swapStake` from parent userEntity
        successfully (153ms)
      ✓ should calling `swapStake` from the current entity
        successfully (96ms)
      ✓ should calling `swapStakeTo` from user's wallet
        successfully (214ms)
      ✓ should calling `swapStakeTo` from parent userEntity
        successfully (162ms)
      ✓ should calling `swapStakeTo` from the current entity
        successfully (140ms)

Contract: SingleSignEntityStrategy
```

```
  SingleSignEntityStrategy Test Cases
    SingleSignEntityStrategy Deployment Test Cases
      ✓ shouldn't allow deploying contract with zero address (155ms)
      ✓ should deploy SingleSignEntityStrategy contract correctly
    SingleSignEntityStrategy Signature Test Cases
      ✓ shouldn't allow setting trusted signer with zero address
      ✓ shouldn't allow setting trusted signer with the same
        address
      ✓ shouldn't allow if the deadline of passed signature has
        expired
      ✓ shouldn't allow if the length of the array of parameters
        of signature is wrong (51ms)
      ✓ should setting trusted signer successfully
      ✓ should return false if signer is wrong (179ms)
      ✓ should validating signature successfully

Contract: SwapMILE
  SwapMILE Test Cases
    SwapMILE Deployment Test Cases
      ✓ shouldn't allow initializing contract twice
      ✓ shouldn't allow initializing contract with zero
        addresses (306ms)
      ✓ should deploy contract correctly (147ms)
    SwapMILE Staking Test Cases
      ✓ shouldn't allow calling `setCoolDownPeriod`,
        `setWithdrawPeriod` and `setEntityFactoryContract`
        by not the admin (319ms)
      ✓ shouldn't allow setting an invalid cooldown period (39ms)
      ✓ shouldn't allow setting an invalid withdrawal period
      ✓ shouldn't allow update EntityFactory contract with zero
        address (40ms)
      ✓ should set cooldown period successfully
      ✓ should set withdrawal period successfully
      ✓ should calling `updateWhiteList` successfully (172ms)
      ✓ shouldn't allow calling `stake`, `stakeTo`, `swapStake`,
        `swapStakeTo` by not the entity (116ms)
      ✓ shouldn't allow calling `swapStake`, `swapStakeTo` with
        zero ERC20 token address (223ms)
      ✓ shouldn't allow swapping zero amount of MILE tokens (184ms)
      ✓ should calling `stake` from user's wallet
        successfully (138ms)
```

```
    ✓ should calling `stake` from user's entity
      successfully (88ms)
    ✓ should calling `stakeTo` from user's wallet
      successfully (288ms)
    ✓ should calling `stakeTo` from user's entity
      successfully (178ms)
    ✓ should calling `swapStake` from user's wallet
      successfully (176ms)
    ✓ should calling `swapStake` from user's entity
      successfully (140ms)
    ✓ should calling `swapStakeTo` from user's wallet
      successfully (225ms)
    ✓ should calling `swapStakeTo` from user's entity
      successfully (335ms)
  SwapMILE Withdrawal Test Cases
    ✓ shouldn't allow requesting withdrawal with zero amount
      of SMile tokens (68ms)
    ✓ shouldn't allow requesting withdrawal if not enough MILE
      tokens to transfer
    ✓ shouldn't allow requesting withdrawal if not enough sMILE
      to transfer (55ms)
    ✓ shouldn't allow canceling withdrawal if the request does
      not exist
    ✓ shouldn't allow canceling withdrawal if the passed withdraw
      request isn't active (66ms)
    ✓ should calling `requestWithdraw` successfully
    ✓ should calling `cancelWithdraw` successfully (73ms)
    ✓ shouldn't allow withdrawing from staking if the passed
      withdraw request does not exist
    ✓ shouldn't allow withdrawing from staking if the passed
      withdraw request isn't active (107ms)
    ✓ shouldn't allow adding MILE tokens to pool with zero amount
    ✓ shouldn't allow withdrawing from staking if not enough
      sMILE tokens on entity balance (119ms)
    ✓ shouldn't allow withdrawing from staking if not enough
      MILE tokens on SwapMILE balance (840ms)
    ✓ should calling `addRewards` successfully (59ms)
    ✓ should calling `withdrawFromStaking` immediate (113ms)
    ✓ should calling `withdrawFromStaking` after half of the
      cooldown duration passed (147ms)
    ✓ should calling `withdrawFromStaking` after cooldown
      period (134ms)
```

```
      ✓ should calling `withdrawFromStaking` after withdrawal
        period (96ms)
      ✓ should calculate the price of tokens successfully when
        SMile pool is zero (131ms)
      ✓ should calculate the price of tokens successfully when
        MILE pool > SMile pool (384ms)
      ✓ should allow withdrawing tokens from entity successfully
        successfully (366ms)

Contract: UserEntity
  UserEntity Test Cases
    UserEntity Deployment Test Cases
      ✓ shouldn't allow initializing contract twice
      ✓ shouldn't allow initializing contract with zero
        addresses (247ms)
      ✓ should deploy contract correctly (107ms)
    UserEntity Owner Test Cases
      ✓ shouldn't allow calling `addOwner`,`removeOwner` and
        `swapOwner` by not the owner (92ms)
      ✓ shouldn't allow adding a new owner with an invalid
        address (148ms)
      ✓ shouldn't allow removing an owner from the Safe with
        an invalid address (96ms)
      ✓ shouldn't allow to remove the last owner from the Safe
      ✓ shouldn't allow to swap an owner with an invalid
        address (225ms)
      ✓ shouldn't allow approving erc20 tokens if spender
        isn't entity (42ms)
      ✓ should allow to swap an owner from the Safe with another
        address (78ms)
      ✓ should allow to remove an owner from the Safe (162ms)
      ✓ should allow to add a new owner to the Safe (68ms)
      ✓ should creating a company entity successfully (164ms)
      ✓ should creating a project entity successfully (95ms)
      ✓ should allow adding and removing admin (192ms)

  133 passing (1m)
```

We are delighted to have a chance to work with the milestoneBased team and contribute to your company's success by reviewing and certifying the security of your smart contracts.

The statements made in this document should be interpreted neither as investment or legal advice, nor should its authors be held accountable for decisions made based on this document.

Vidma is a security audit company helping crypto companies ensure their code and products operate safely and as intended, enabling founders to sleep soundly at night. We specialize in auditing DeFi protocols, layer one protocols, and marketplace solutions. Our team consists of experienced and internationally trained specialists. Our company is based in Ukraine, known for its strong engineering, cryptography, and cybersecurity culture.

Website: **vidma.io**
Email: **security@vidma.io**

VIDMA