

## Blockchain

### Report

## 1. Auditing

### - unSafe.sol

Introduction:

```
1 pragma solidity 0.4.17;
2
3 contract UnsafeBank {
4     mapping(address => uint256) public balances;
5
6     function deposit(address _for, uint value) public payable {
7         balances[_for] += value;
8     }
9
10    function withdraw(address from, uint256 amount) public {
11        if(balances[from] >= amount) {
12            balances[from] -= amount;
13            balances[msg.sender] += amount;
14        }
15    }
16 }
```

Please zoom in

### Problems Arising:

- Η συνάρτηση `deposit(address _for, uint value) public payable`, έχει οριστεί `payable`, που σημαίνει ότι μπορεί να δεχτεί Ether. Η λειτουργία της είναι να μπορεί ένας χρήστης να καταθέσει ένα ποσό σε ένα `address`, αλλά η `balances[_for]` δεν βασίζεται στο `msg.value` (το πραγματικό Ether που στέλνεται) αλλά στο `value` που δίνει ο χρήστης.

#### Vulnerability:

Λόγω της χρήσης του `value`, ένας κακόβουλος χρήστης μπορεί να βάλει οποιαδήποτε τιμή στο πχ. `value = 10 Ether` και στην συναλλαγή να αποστέλλεται μόνο 1 Ether. Άρα ο χρήστης μπορεί πανεύκολα να δηλώσει πολύ μικρότερο ποσό από αυτό που πραγματικά καταθέτει, εφ'όσον δεν υπάρχει και καν έλεγχος ισότητας του `value` (αυτό που δηλώνει ο χρήστης ότι θα καταθέσει) με το `msg.value`. Μπορεί να έχει οποιαδήποτε τιμή, ανεξάρτητα από το ποσό του Ether που αποστέλλεται. Πρέπει να αφαιρεθεί γενικά από την συνάρτηση το `value` και αυτό θα ανανεώνεται όπως χρειάζεται αυτόματα δεν θα ορίζεται.

- Η συνάρτηση `withdraw(address from, uint256 amount) public`, έχει σκοπό την ανάληψη Ether όμως δεν υπάρχει κανένας έλεγχος για το αν ο caller είναι ο ίδιος με

το from. Οποιοσδήποτε μπορεί να καλέσει τη συνάρτηση και να μεταφέρει από κάποιον άλλο λογαριασμό υπόλοιπο, προς τον δικό του. Άρα στην προκείμενη περίπτωση δεν έχουμε καν authorization, και πρέπει σίγουρα να συμπεριληφθεί κάτι σε:

require(msg.sender == from) για να διασφαλίσουμε ότι ο η μέθοδος όταν καλείται αφορά μόνο τον authorized sender και κανέναν άλλο.

3. Η έκδοση Solidity 0.4.17 δεν είναι πρόσφατη και αυτό από μόνο του μπορεί να μην καλύπτει πάρα πολλά κενά ασφαλείας που σε νεώτερα versions έχουν πλέον λυθεί. Για παράδειγμα, το SafeMath είναι ένας μηχανισμός που μπορεί να αναλάβει overflows/underflows.
4. Στην πραγματικότητα όλα αυτά δεν είναι καν vulnerabilities γιατί ούτως ή άλλως δεν υπάρχουν καν πραγματικά Ether transfers. Δεν γίνεται πραγματική καταβολή από το contract σε κάποιο address.  
Η συνάρτηση withdraw απλώς χειρίζεται τα εσωτερικά υπόλοιπα, αλλά το γεγονός ότι δεν χρησιμοποιεί τις μεθόδους address.transfer() ή msg.sender.transfer() την καθιστά ένα αναποτελεσματικό smart contract.

### Hardening Method:

Εκτός από τη διόρθωση των μεθόδων και των παραπάνω κενών ή vulnerabilities που σημειώθηκαν, μπορεί να προστεθεί και nonReentrant modifier στο smart contract ώστε να προστατέψουμε από επικίνδυνους actors να κάνουν exploit την recursive συμπεριφορά ενός contract (πχ. κάνοντας επαναλαμβανόμενες κλήσεις της συνάρτησης ανάληψης (withdraw) πριν το υπόλοιπο ενημερωθεί σωστά) και να αποσπάσει κεφάλαια.. ορίζοντας ένα κλείδωμα της συνάρτησης καθ' όσο εκτελείται και ξεκλείδωμα όταν ολοκληρωθεί.

Η χρήση της msg.sender.call{value: amount}("") επιτρέπει να προωθηθεί όλο το διαθέσιμο gas, δίνοντας μεγαλύτερη ευελιξία και δυνατότητα ελέγχου του αποτελέσματος της μεταφοράς (μέσω της επιστροφής ενός boolean). Αυτό επιτρέπει την ασφαλή επαναφορά της συναλλαγής σε περίπτωση που η μεταφορά αποτύχει, κάτι που δεν είναι εφικτό με την απλούστερη μέθοδο transfer.

Παρακάτω είναι η πρόταση:

```
pragma solidity ^0.8.0;

contract SafeNow{
    mapping(address => uint256) public balances;      bool
private locked;

    event deposited(address indexed user, uint256 amount);
    event withdrawal(address indexed account, uint256 amount, uint256
newBalance);

    modifier nonReentrant() {

```

```

        require(!locked, "Reentrancy detected");
locked = true;
    ;
locked = false;
}
function deposit() external payable {
    require(msg.value > 0, "Must send some
Ether..."); balances[msg.sender] += msg.value;
emit deposited(msg.sender, msg.value);
}
function withdraw(uint256 amount) external nonReentrant {
    require(balances[msg.sender] >= amount, "Insufficient balance");
balances[msg.sender] -= amount;
(bool success,) = msg.sender.call{value: amount}("");
require(success, "Transfer failed.");
emit withdrawal(msg.sender, amount, balances[msg.sender]);
}
}

```

## - victim.sol

### Introduction:

```

1 pragma solidity ^0.4.17;
2
3 contract Victim {
4     mapping (address => uint) public balances;
5
6     function deposit(uint value) public payable {
7         balances[msg.sender] += value;
8     }
9
10    function withdraw(uint value, address from ) public {
11        uint bal = balances[from];
12        require(bal > 0);
13        balances[from] -= value;
14    }
15
16
17 }
18

```

The annotations highlight several security flaws:

- No auditing! No emits**: Points to the lack of event emissions.
- msg.value not included = contract can add to the sender's balance irrelevant value compared to Ether value**: Points to the fact that the contract adds any value sent, regardless of the Ether amount.
- No authorization : It can be called by ANY address and reduce someone's balance**: Points to the lack of authorization, allowing any address to withdraw funds.
- Yes the balance should be above 0 to be able to withdraw, but is the value to be withdrawn less or equal to the balance?**: Points to the logical error where the withdrawal amount is not checked against the balance.

Please zoom in

### Problems Arising:

- Η συνάρτηση `deposit` δέχεται μια παράμετρο `uint value` και, αντί να χρησιμοποιεί το `msg.value` (δηλαδή το πραγματικό ποσό Ether που στάλθηκε), αυξάνει το εσωτερικό υπόλοιπο του αποστολέα με την τιμή που δόθηκε ως παράμετρος. Ένας κακόβουλος actor μπορεί να καλέσει τη συνάρτηση ορίζοντας ένα πολύ υψηλό `value` (πχ. 20 Ether) ενώ στέλνει στην συναλλαγή μόνο, για παράδειγμα, 1 Ether. Έτσι, το εσωτερικό mapping θα καταγράψει ένα υψηλό υπόλοιπο που δεν ισχυεί καν.

2. Χωρίς authorization: Η συνάρτηση withdraw δέχεται ως παραμέτρους ένα ποσό uint value και μία διεύθυνση from. Δεν ελέγχει αν αυτός που την καλεί (msg.sender) είναι ο ίδιος με το from. Κάποιος μπορεί να καλέσει τη συνάρτηση για λογαριασμό άλλου, μειώνοντας το υπόλοιπό του χωρίς να έχει δικαιώμα.
3. Ο έλεγχος require(bal > 0) δεν επαληθεύει ότι το ποσό που ζητείται (value) είναι μικρότερο ή ίσο με το υπόλοιπο. Λόγω και του Solidity (0.4.17), ένας κακόβουλος μπορεί να προκαλέσει underflow δηλαδή η αφαίρεση balances[from] -= amount; σκόπιμα να φέρει αρνητικό αριθμό, και αναγκάζει τον αλγόριθμο να πάει στον αμέσως μεγαλύτερο πιθανό αριθμό..
4. Η συνάρτηση withdraw δεν πραγματοποιεί πραγματικό transfer Ether (δεν καλεί transfer() ή call.value() για αποστολή Ether). Απλώς ενημερώνει τα mapping balances, χωρίς να εξασφαλίζει ότι το Ether που αντιστοιχεί στα υπόλοιπα αποστέλλεται στον χρήστη.

#### Hardening Method:

```
pragma solidity ^0.8.0;

contract NotAVictim {
    mapping(address => uint256) public balances;
    bool private locked;

    event deposited(address indexed user, uint256 amount);
    event withdrawal(address indexed account, uint256 amount, uint256 newBalance);

    modifier nonReentrant() {
        require(!locked, "Reentrant call detected!");
        locked = true;
        ;
        locked = false;
    }

    function deposit() external payable {
        require(msg.value > 0, "Must send some Ether");
        balances[msg.sender] += msg.value;
        emit deposited(msg.sender, msg.value);
    }

    function withdraw(uint256 amount) external nonReentrant {
        require(balances[msg.sender] >= amount, "Insufficient balance");
        balances[msg.sender] -= amount;
        (bool success, ) = msg.sender.call{value: amount}("");
        require(success, "Transfer failed");
        emit withdrawal(msg.sender, amount, balances[msg.sender]);
    }
}
```

## withdraw\_1.sol

- Introduction: completely ‘under construction’

```
1 pragma solidity 0.4.17;
2
3 contract withdrawBalance {
4     uint amount;
5     uint public value = 0 ether;
6     mapping (address => uint) balance;
7
8     function setBalance() public {
9         balance[msg.sender] = value;
10    }
11
12     function withdraw() public returns (uint) {
13         amount = balance[msg.sender];
14         balance[msg.sender] = 0 ether;
15         return amount;
16    }
17 }
```

Please zoom in

### Problems Arising:

1. Το version Solidity 0.4.17, είναι αρκετά παλιό, δεν παρέχει τα σύγχρονα μέτρα ασφαλείας (όπως έλεγχοι overflow/underflow που υπάρχουν στις νέες εκδόσεις).
2. Μεταβλητή value: ορίζεται ως uint public value = 0 ether; και χρησιμοποιείται στη συνάρτηση setBalance για να θέσει το υπόλοιπο του χρήστη. Επειδή η τιμή είναι 0, κάθε φορά που κάποιος καλεί το setBalance, το mapping balance[msg.sender] ορίζεται στο 0.
3. Η συνάρτηση setBalance αντίστοιχα απλώς αναθέτει το 0 (την τιμή της μεταβλητής value) στο υπόλοιπο του χρήστη. Δεν υπάρχει τρόπος να αλλάξει αυτό το ποσό ώστε να δημιουργηθεί ένα θετικό υπόλοιπο.
4. Η συνάρτηση withdraw αποθηκεύει το τρέχον υπόλοιπο στην amount. Θέτει το υπόλοιπο του στο 0 (χωρίς πρακτικά να πραγματοποιείται καμία μεταφορά Ether) και επιστρέφει το ποσό που ήταν αποθηκευμένο.  
Δηλαδή, η συνάρτηση δεν πραγματοποιεί κάποια πραγματική ανάληψη Ether αλλά απλώς "αναφέρει" το ποσό που είχε καταγραφεί στο mapping (το οποίο πάντα είναι 0 λόγω της setBalance...)

Δεν υπάρχει πραγματική χρηματική λειτουργικότητα στο smart contract για να προταθεί κάποιο hardening method..

## - withdraw\_2.sol

Introduction:

```

1 pragma solidity 0.4.17; ✘ It doesn't do what the naming implies, everything is actually set to 0.
2
3 contract withdrawBalance [ global vars for temporary data are risky
4     uint amount; this exceeds Ether supplies.
5     uint public value = 2**256-1 ether; +++ not enough, tracks no real ethers.
6     mapping (address => uint) balance;
7
8     function setBalance(address _victim) public { Everyone can literally set a whatever address and set the balance to value,
9         balance[_victim] = value; which was previously defined as maximum.
10    }
11
12    function withdraw(address _victim) public returns (uint) {
13        amount = balance[_victim];
14        balance[_victim] = 0 ether;
15        return amount; No authorization: Everyone can access these functions
16    }
17 }
```

Please zoom in

## Problems Arising:

- Δεν υπάρχει access control : καμία από τις δύο functions withdraw, setBalance δεν έχει ownership check ή modifier. Ο οποιοσδήποτε μπορεί να τις καλέσει και να χειριστεί άνευ ελέγχου οποιοδήποτε address.
- Χωρίς authorization: η withdraw δέχεται το argument \_victim που αφορά address πιθανώς ενός κυριολεκτικά θύματος, οπού ένας κακόβουλος actor μπορεί να αποσπάσει κεφάλαιο από το balance του.
- Ορίζονται αυθαίρετα τεράστια ποσά (π.χ. value = 2\*\*256 - 1 ether), τα οποία πλησιάζουν το ανώτατο όριο του τύπου uint256, κάτι που μπορεί να οδηγήσει σε προβλήματα óπως overflow ή underflow εάν δεν εφαρμοστεί σωστά ο χειρισμός των αριθμητικών πράξεων.
- Η έκδοση Solidity 0.4.17 δεν διαχειρίζεται το παραπάνω πρόβλημα σε περίπτωση που συμβεί, καταλήγουμε σε ευαλωτότητα, και γενικότερα μπορεί να καταλήγει σε άλλες μη άμεσα εμφανής ευαλωτότητες λόγω outdated version.
- Μη λειτουργικό smart-contract: Δεν υπάρχει πραγματικό trigger κάποιου Ether transfer.

## Hardening Method:

Κανονικά, σε αντίθεση με τα παραπάνω hardening methods, δεν χρειάζεται στην πραγματικότητα να εφαρμόσουμε nonReentrant modifier επειδή πρόκειται για μια πάρα πολύ απλή λειτουργικότητα.. παραπάνω ήταν ενδεικτική η χρήση οπότε εδώ παραλείπεται.

```
pragma solidity ^0.8.0;

contract WithdrawSetBalance {
    mapping(address => uint256) private balances;

    event deposited(address indexed user, uint256 amount);
    event withdrawal(address indexed account, uint256 amount, uint256 newBalance);

    function deposit() external payable {
        require(msg.value > 0, "Must deposit an amount > 0");
        balances[msg.sender] += msg.value;
        emit deposited(msg.sender, msg.value);
    }

    function withdraw(uint256 amount) external {
        require(balances[msg.sender] >= amount, "Insufficient balance");
        balances[msg.sender] -= amount;
        (bool success, ) = msg.sender.call{value: amount}("");
        require(success, "Ether transfer failed");
        emit withdrawal(msg.sender, amount, balances[msg.sender]);
    }

    function getBalance(address account) external view returns (uint256) {
        return balances[account];
    }
}
```

## 2. Smart contracts

### First Smart Contract

3. Ο ιδιοκτήτης (owner) έχει ισχύ να ορίζει τους καθηγητές.
4. Κάθε καθηγητής μπορεί να εγγράψει (register) φοιτητές ή να ενημερώσει τους βαθμούς τους.
5. Κάθε φοιτητής μπορεί να εγγράψει μόνο τον εαυτό του, εφόσον δεν είναι ήδη εγγεγραμμένος, αν δεν τον γράψει κάποιος καθηγητής πρώτα.
6. Οποιοσδήποτε χρήστης μπορεί να δίνει ή να ανακαλεί πρόσβαση στα δικά του δεδομένα μέσω των grantAccess/revokeAccess.
7. Το getUserGrade και το getUserData έχουν ελέγχους ώστε να επιστρέφουν πληροφορίες μόνο σε:
  - Τον ιδιοκτήτη των δεδομένων (ίδιο msg.sender),
  - Σε έναν καθηγητή,
  - Ή σε όποιον έχει δοθεί ρητά πρόσβαση (allowedAccess).
8. Η συνάρτηση isAllowed παροσιάζει μια τιμή bool για το checking του allowedAccess.

Υλοποίηση:

Το συμβόλαιο δηλώνει την έκδοση του Solidity (**pragma solidity ^0.8.0;**).

Ορίζεται ένα enum Occupation με τρεις πιθανές τιμές:

1. None (κανένα επάγγελμα)
2. Student (μαθητής/φοιτητής)
3. Teacher (καθηγητής/διδάσκων)

Το smart contract UserData είναι υπεύθυνο για την καταχώρηση και διαχείριση των στοιχείων ενός User.

Περιέχει το struct User που περιλαμβάνει τα στοιχεία διεύθυνση, όνομα, επώνυμο, ηλικία, βαθμό και επάγγελμα (όπου το occupation είναι "student" ή "teacher"). Επιπλέον, περιέχει έναν εσωτερικό mapping (allowedAccess) που δηλώνει σε ποιες διευθύνσεις έχει δοθεί δικαίωμα πρόσβασης στα δεδομένα του συγκεκριμένου χρήστη.

Ο Owner αυτόματα θέτεται εκείνος που κάνει deploy το smart contract: address public owner: Καθορίζει τον κάτοχο του συμβολαίου. mapping(address => User) private users;

Αποθηκεύει όλα τα προφίλ χρηστών, με κλειδί τη διεύθυνση (address) του χρήστη και τιμή ένα struct User.

## Modifiers

### onlyOwner

Επιτρέπει την εκτέλεση της συνάρτησης μόνο εάν αυτός που την καλεί (msg.sender) είναι ο owner του συμβολαίου.

### onlyTeacher

Επιτρέπει την εκτέλεση μόνο εάν ο χρήστης που την καλεί έχει occupation = Teacher.

### onlyOwnerOrTeacher

Επιτρέπει την εκτέλεση μόνο εάν ο msg.sender είναι ο ιδιοκτήτης ή αν έχει occupation = Teacher.

Στην συνέχεια ορίζονται events τα οποία χρησιμοποιούνται στα καταλληλά σημεία εντός των αντίστοιχων functions.

## Functions

### registerTeacher()

1. Δηλώνεται ως **external** onlyOwner

2. Εγγράφει έναν νέο καθηγητή.
  1. Ελέγχει ότι η διεύθυνση \_teacher δεν είναι 0.
  2. Αποθηκεύει τα στοιχεία σε ένα User με occupation = Teacher.
  3. Χρήση του event UserRegistered.

Μόνο ο owner έχει δικαίωμα να δηλώσει νέους καθηγητές.

#### **registerStudent()**

1. Δηλώνεται ως external, και περιέχει έλεγχο για το ποιος μπορεί να την καλέσει.
2. Επιτρέπει είτε σε έναν ήδη καταχωρημένο καθηγητή είτε στον ίδιο το φοιτητή να καταχωρηθεί ως μαθητής (ο μαθητής θα πρέπει να δηλώσει στην μέθοδο το address account του).
3. Ελέγχει ότι ο χρήστης δεν είναι ήδη εγγεγραμμένος (occupation == Occupation.None).
4. Θέτει occupation = Student και κάνει emit event UserRegistered.

#### **grantAccess (address\_grantee)**

1. Ο τρέχων χρήστης (msg.sender) προσθέτει τη διεύθυνση \_grantee στον χάρτη allowedAccess.
2. Αυτό σημαίνει ότι η \_grantee θα μπορεί να δει τα δεδομένα (βαθμό, προσωπικά στοιχεία) του χρήστη.
3. Emit event AccessGranted.

#### **revokeAccess (address\_grantee)**

1. Ο τρέχων χρήστης αφαιρεί τη διεύθυνση \_grantee από την access control list.
2. Emit event AccessRevoked.

#### **updateGrade(address\_student, uint\_newGrade)**

1. Δηλώνεται ως public onlyTeacher.
2. Επιτρέπει μόνο σε καθηγητή να ενημερώσει τον βαθμό ενός μαθητή.
3. Ελέγχει ότι ο \_student είναι πραγματικά μαθητής.
4. Ενημερώνει το grade χρήση του event GradeUpdated.
5. \*\* όλες οι μέθοδοι που είναι public θα μπορούσαν μελλοντικά να χρησιμοποιηθούν και εσωτερικά από functions.

#### **isAllowed(address\_user, address\_requester) view**

function.

Επιστρέφει true αν:

1. Ο \_requester είναι ο ίδιος ο \_user (μπορεί πάντα να δει τα δικά του δεδομένα)
2. Ή αν το allowedAccess[\_requester] είναι true.

#### **getUserGrade(address\_user)**

1. Επιτρέπει στον msg.sender να πάρει τον βαθμό ενός χρήστη μόνο εάν:

- msg.sender == \_user,
- αν ο msg.sender είναι Teacher,
- αν το isAllowed(\_user, msg.sender) είναι true (έχει δοθεί ρητή πρόσβαση προηγουμένως).

2. Αν περάσει κάποιος από τους παραπάνω ελέγχους, ο βαθμός επιστρέφεται.

#### **getUserOccupation(address\_user)** επιστρέφεται το επάγγελμα ("Student", "Teacher", ή "None") με βάση την ιδιότητα του χρήστη.

#### **getUserData(address\_user)**

Επιστρέφει όλα τα στοιχεία ενός χρήστη (διεύθυνση, όνομα, επώνυμο, ηλικία, βαθμό, επάγγελμα).

Μόνο αν ο msg.sender είναι:

- owner,
- ο ίδιος ο \_user,
- καθηγητής,
- ή περιλαμβάνεται στο allowedAccess του \_user.

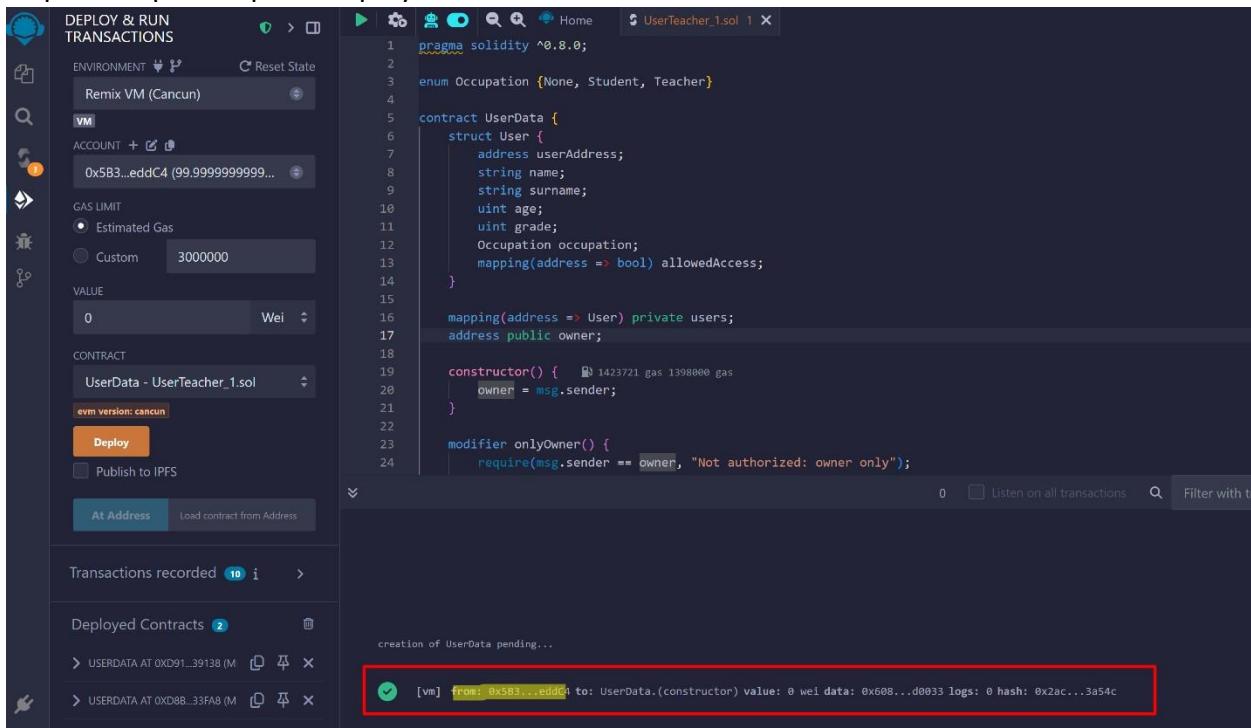
#### **Βήματα για τεστ του κώδικα**

1. Ο Owner αυτόματα θέτεται εκείνος που κάνει deploy το smart contract. Επιλέγω συνήθως το πρώτο account από τα προτεινόμενα του environment που μου δίνονται.
2. Ο owner καταχωρεί έναν καθηγητή καλώντας registerTeacher.
3. Ο καθηγητής στη συνέχεια μπορεί να καταχωρίσει φοιτητές με τη registerStudent.
4. Ο ίδιος ο φοιτητής μπορεί να καταχωρίσει τον εαυτό του αν δεν το κάνει ο καθηγητής (και εφόσον δεν είναι ήδη καταχωρημένος).
5. Όταν ο φοιτητής εγγραφεί, ο καθηγητής μπορεί να ορίσει έναν βαθμό με το updateGrade(address\_of\_student, newGrade).
6. Για πρόσβαση σε προσωπικά δεδομένα, ο ίδιος ο φοιτητής μπορεί να χρησιμοποιήσει grantAccess(address) ώστε να επιτρέψει σε κάποιον άλλο χρήστη (π.χ. σε άλλο φοιτητή) να δει τα δεδομένα/τον βαθμό του.

7. Οι καθηγητές έχουν πάντα δικαίωμα να δουν τους βαθμούς όλων των φοιτητών, ενώ οι φοιτητές μπορούν να δουν μόνο τον δικό τους βαθμό ή του συμμαθητή τους αν αυτός τους έδωσε πρόσβαση.

## Παραδείγματα Εκτέλεσης

Παρακάτω βλέπουμε το deploy και το owner account.



```

1 pragma solidity ^0.8.0;
2
3 enum Occupation {None, Student, Teacher}
4
5 contract UserData {
6     struct User {
7         address userAddress;
8         string name;
9         string surname;
10        uint age;
11        uint grade;
12        Occupation occupation;
13        mapping(address => bool) allowedAccess;
14    }
15
16    mapping(address => User) private users;
17    address public owner;
18
19    constructor() {
20        owner = msg.sender;
21    }
22
23    modifier onlyOwner() {
24        require(msg.sender == owner, "Not authorized: owner only");
25    }
26
27 }
```

The screenshot shows the Remix IDE interface with the Solidity code for the `UserData` contract. The code defines a `User` struct with fields for address, name, surname, age, grade, occupation, and a mapping of address to boolean allowedAccess. It also includes a `mapping(address => User) private users;` and a `address public owner;`. The `constructor()` sets the `owner` to `msg.sender`. A `modifier onlyOwner()` ensures only the owner can call certain functions. The `Deploy` button is highlighted in orange. Below the code editor, a transaction history shows two deployed contracts: `USERDATA AT 0xD91...39138 (M)` and `USERDATA AT 0xD88...33FAB (M)`. The transaction for the second contract is highlighted with a red box and shows a green checkmark indicating success. The log message is `[vm] from: 0x5B3...eddC4 to: UserData.(constructor) value: 0 wei data: 0x608...d0033 logs: 0 hash: 0x2ac...3a54c`.

Επόμενο βήμα να φτιάξουμε έναν teacher και να τον κάνουμε register. Επιλεγώ από τα accounts το δεύτερο που μου προτείνεται, copy, και συμπληρώνω τα πεδία στο registerTeacher

```

27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
    modifier onlyTeacher() {
        require(users[msg.sender].owner == true);
    }

    modifier onlyOwnerOrTeacher() {
        require(
            msg.sender == owner || users[msg.sender].owner == true
        );
    }

    event UserRegistered(address user);
    event GradeUpdated(address index);
    event AccessGranted(address index);
}

```

```

[vm] from: 0x5B3...edc4 to: UserData.registerTeacher(address,string,string,uint256) 0xd91...39138 value: 0 wei data: 0x75f...0000 logs: 1
status 0x1 Transaction mined and execution succeed
transaction hash 0xb1cace00e8c2f512831809bfa115cbf24b254b43ad03e61fd1cb17773b139792
block hash 0x67c017bacd12b6530a0f474009a75ed878b47eee8cfcc30930f84ced411abc
block number 2

```

```

...
decoded output {}
logs []
[
    {
        "from": "0xd9145CCE52D386f254917e481e844e9943F39138",
        "topic": "0x92822564ba8864c3a47b34e8d23fbce5c46234eb5da261f94087b995ac0f33b",
        "event": "UserRegistered",
        "args": {
            "0": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",
            "1": "Stavrakas",
            "2": "Adamantios",
            "user": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",
            "name": "Stavrakas",
            "surname": "Adamantios"
        }
    }
]

```

Ο teacher δημιουργήθηκε με επιτυχία. Μπορούμε αλλάζοντας το address στο account του teacher, να δημιουργήσουμε με τον ίδιο τρόπο (αντιγράφοντας ένα νέο account για τον student, να συμπληρώσουμε τα πεδία) και να γίνει register ένας student. Υπενθύμιση: Ένας owner μπορεί να χρησιμοποιήσει αυτή τη μέθοδο αλλά πχ. δεν μπορεί να βάλει βαθμό:

```
[vm] from: 0x4B2...C02db to: UserData.registerStudent(address,string,string,uint256) 0xd91...39138 value: 0 wei data: 0x20a...00000 logs: 1
hash: 0x9ca...b3e6f
transact to UserData.updateGrade pending ...

✖ [vm] from: 0x4B2...C02db to: UserData.updateGrade(address,uint256) 0xd91...39138 value: 0 wei data: 0x63c...0000d logs: 0 hash: 0xc8d...2980e
transact to UserData.updateGrade errored: Error occurred: revert.

revert
The transaction has been reverted to the initial state.
Reason provided by the contract: "Not authorized: teacher only".
If the transaction failed for not having enough gas, try increasing the gas limit gently.
```

O teacher:

```
output          0x ⓘ
decoded input  {
    "address _student": "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db",
    "uint256 _newGrade": "13"
} ⓘ
decoded output {} ⓘ
logs          [
    {
        "from": "0xd9145CCE52D386f254917e481eB44e9943F39138",
        "topic": "0x12d55b4ae074602c87a39bb97ac8bf557e91259797ae295100c5852c54744d7",
        "event": "GradeUpdated",
        "args": {
            "0": "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db",
            "1": "13",
            "student": "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db",
            "newGrade": "13"
        }
    }
] ⓘ
```

Χρησιμοποιώντας την getGrade για τον συγκεκριμένο student:

```
CALL  [call] from: 0xAB8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: UserData.getUserGrade(address) data: 0xff8...c02db

from          0xAB8483F64d9C6d1EcF9b849Ae677dD3315835cb2 ⓘ
to           UserData.getUserGrade(address) 0xd9145CCE52D386f254917e481eB44e9943F39138 ⓘ
execution cost 5280 gas (Cost only applies when called by a contract) ⓘ
input         0xff8...c02db ⓘ
output        0x0000000000000000000000000000000000000000000000000000000000000000d ⓘ
decoded input {
    "address _user": "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db"
} ⓘ
decoded output {
    "0": "uint256: 13"
```

getUserOccupation

getUserGrade	getUserOccupation
User: 0x4B20993Bc481177ec7E8f571c	User: 0x4B20993Bc481177ec7E8f571c
Calldata	Calldata
Parameters	Parameters
call	call
0: uint256: 13	0: string: Student
isAllowed	
address: user, address	
owner	
Low level interactions	

Αλλάζοντας με τον ίδιο τρόπο το account σε μια διεύθυνση student, μπορούν να χρησιμοποιηθούν οι παρακάτω μέθοδοι:

Ένας student , όπως περιγραφεται και παραπανω μπορει να κανει register τον εαυτο του, εννοειται πλην του να αναθεσει grade στον εαυτο του:

The screenshot shows a blockchain development interface. On the left, there are input fields for student, name, surname, and age. On the right, the decoded input and output are displayed. The decoded input shows the transaction details, including the event "UserRegistered" and args. The decoded output shows the logs, which include the transaction hash and the event data.

```

student: 0x78731D3Ca6b7E34aC0F824c4
_name: Irini
_surname: Loli
_age: 10

decoded input
{
    "address _student": "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabab",
    "string _name": "Irini",
    "string _surname": "Loli",
    "uint256 _age": "10"
}

decoded output
{}

logs
[{"from": "0xd9145CE5D3B6f254917e481e844e9943f39138", "topic": "0x9282256ab884ac3a47b34e8d23fbce5c46234eb5da261f94887b995ac0f33b", "event": "UserRegistered", "args": {"0": "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabab", "1": "Irini", "2": "Loli", "user": "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabab", "name": "Irini", "surname": "Loli"}]
}

```

UpdateGrade:

The screenshot shows a blockchain development interface. A transaction for "updateGrade" is being processed. The logs show an error message: "Reason provided by the contract: 'Not authorized: teacher only'". This indicates that the user does not have the necessary permissions to update a grade.

Ένας θελει student να δωσει access σε έναν άλλο student – μεσω της grantAccess:

The screenshot shows a blockchain development interface. A transaction for "grantAccess" is being processed. The logs section shows the event data for the granted access, including the grantee's address and the access type.

```

transaction cost      45772 gas
execution cost       24348 gas
input                0x0se...cabab
output               0x
decoded input
{
    "address _grantee": "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabab"
}

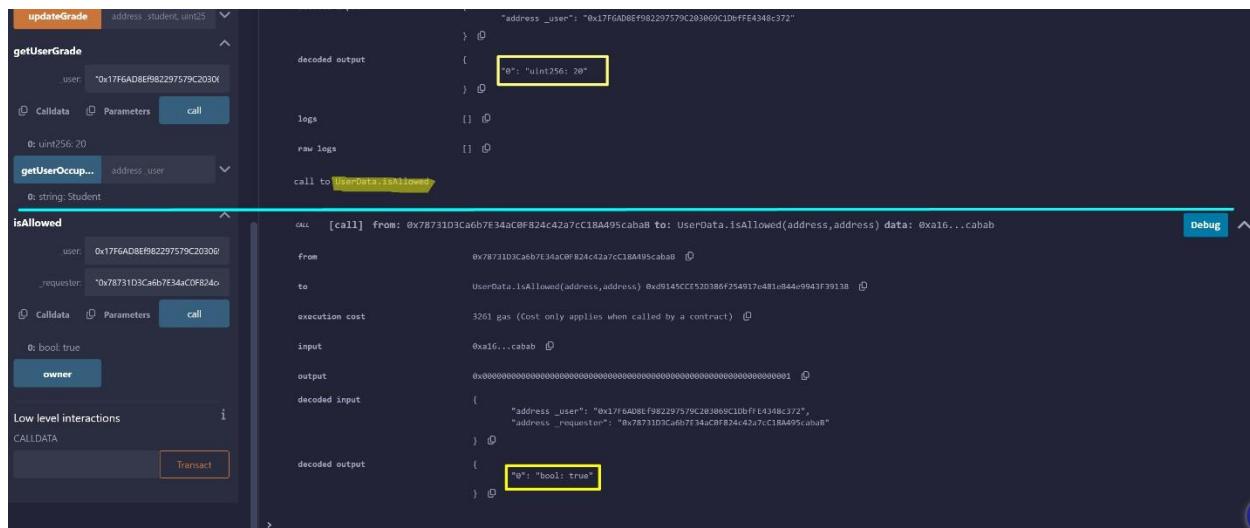
decoded output
{}

logs
[{"from": "0xd9145CE5D3B6f254917e481e844e9943f39138", "topic": "0x9282256ab884ac3a47b34e8d23fbce5c46234eb5da261f94887b995ac0f33b", "event": "AccessGranted", "args": {"0": "0x17f6ADEfF982297579c20386c10fFE4348c372", "1": "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabab", "user": "0x17f6ADEfF982297579c20386c10fFE4348c372", "grantee": "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabab"}]
}

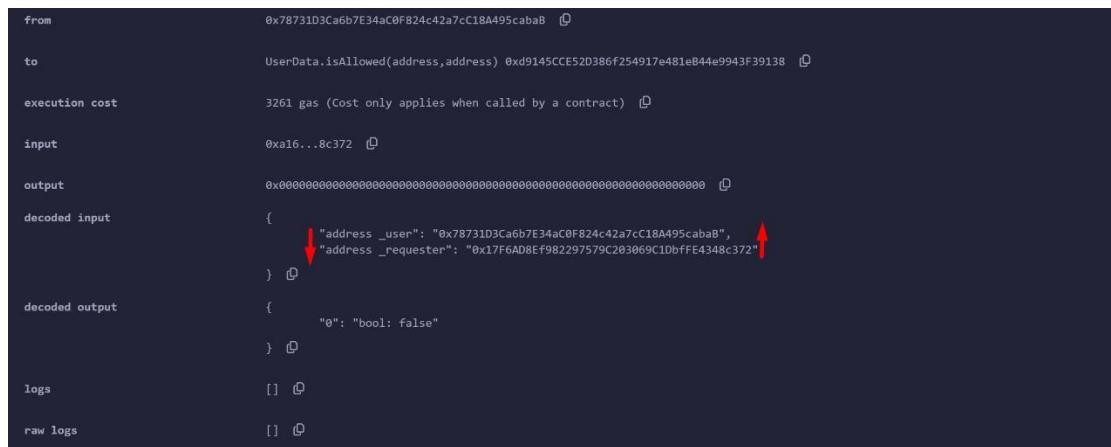
raw logs
[{"0": "0x17f6ADEfF982297579c20386c10fFE4348c372", "1": "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabab", "user": "0x17f6ADEfF982297579c20386c10fFE4348c372", "grantee": "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabab"}]

```

Δοκιμάζοντας από τα παραπάνω args – “grantee” ως account για να δούμε τα στοιχεία του “user”:



Αν δοκιμάσουμε το αντίστροφο : Δηλαδή συνδεδεμένοι ως user (με βάση τα παραπάνω args) προσπαθήσουμε να δούμε πχ τον βαθμό του grantee:



**Σημείωση:** Στην εργασία αναφέρεται «όσοι έχουν την ιδιότητα teacher **εντός του δικτύου**», αυτό σημαίνει ότι κανονικά χρειάζεται ένα νέο smart contract το οποίο θα ονομάζεται TeacherRegistry.sol και θα αναλαμβάνει την εγγραφή ενός teacher από τον owner του συστήματος. Υπέρα το smart contract ονόματι UserTeacher.sol θα μπορεί ανακτώντας τους teachers από εξωτερικά, να πραγματοποιεί τα ιδιά με τα παραπάνω transactions:

Example: require(teacherRegistry.isTeacher(msg.sender), "Not authorized: registered teacher only");

```

pragma solidity ^0.8.0;
enum Occupation {None, Student, Teacher}
contract UserData {
    struct User {
        address userAddress;
        string name;
        string surname;
        uint age;
        uint grade;
        Occupation occupation;
        mapping(address => bool) allowedAccess;
    }
    mapping(address => User) private users;
    address public owner;
}

TEACHERREGISTRY AT 0xD91...
USERDATA AT 0xF8E...9FBEB0

```

Εδώ βλέπουμε έναν teacher που έγινε add από το TeacherRegistry.sol από τον owner μόνο ως διεύθυνση και επεξεργάστηκε στο παρόν contract για να καταχωρηθεί στους UserData.

**Σημαντικό:** η onlyTeacher() άλλαξε και για να υπάρχει συνοχή και αποτελεσματικότητα μεταξύ των 2 συστημάτων και θα πρέπει να ισχύει και **isTeacher = true** και ότι ο teacher(msg.sender) έχει καταχωρηθεί στο σύστημα (δηλαδή έχουμε διαθέσιμες πληροφορίες για το πρόσωπο αυτό).

```

modifier onlyTeacher() {
    require(teacherRegistry.isTeacher(msg.sender) && users[msg.sender].occupation
== Occupation.Teacher, "Not authorized: registered teacher only");
}

```

Για τους ελέγχους, έχει υλοποιηθεί το interface το οποίο περιλαμβάνει μόνο την ενεργοποίηση / απενεργοποίηση ενός teacher account από τον owner. Η δυσκολία για ασφάλεια και συνοχή πλέον έχει αυξηθεί, και θα έπρεπε αυτή η αλλαγή να έρθει ταυτόχρονα με προσαρμογή και όλων των προηγούμενων υλοποίήσεων στο UserData (πχ. οι τρέχοντες χρήστες να ανήκουν στο δίκτυο – όχι στο smart contract αποκλειστικά- για να μπορούν να χρησιμοποιούν τις μεθόδους).

Στον φάκελο με τον κώδικα, περιλαμβάνεται το σενάριο με το Interface που δουλεύει κανονικά (θέτοντας για deploy στο address\_teacherRegistry κάποιο address) αλλά δεν έχει διασφαλιστεί η ακεραιότητα των δεδομένων.

Επομένως, θα δείτε δύο φακέλους για το συγκεκριμένο smart contract

- = TeacherStudent\_v1 που περιλαμβάνει την απλή εκδοχή
- = TeacherStudent\_v2 που έχει την προσπάθεια υλοποίησης του δικτύου όπου επιχειρήθηκε να εισάγονται εξωτερικά τα addresses των teachers.

## Second Smart Contract

Το smart contract προσομοιώνει μια ψηφοφορία που εκλέγεται ένας πρόεδρος (President). Κεντρικό σημείο της εφαρμογής είναι η διαχείριση των υποψηφίων (που υλοποιούνται ως struct **Student**), μαζί με την ασφαλή καταμέτρηση των ψήφων και την αποφυγή διπλής ψηφοφορίας για κάθε χρήστη. Ο κώδικας χωρίζεται σε δύο βασικές λειτουργίες: την εγγραφή υποψηφίων και την ψηφοφορία. Όλοι έχουν τη δυνατότητα να εκλεγούν.

### *Struct*

Η struct Student περιέχει:

- **candidateAddress:** Η διεύθυνση του υποψηφίου (μοναδική).
- **name και surname:** Το όνομα και το επίθετο του υποψηφίου.
- **age:** Η ηλικία του υποψηφίου.
- **voteCount:** Ο counter, η καταμέτρηση ψήφων που λαμβάνει ο υποψήφιος (αρχικοποιείται σε 0 (by default γενικά)).
- **registered:** Ένα flag που επιβεβαιώνει ότι ο υποψήφιος έχει ήδη καταχωρηθεί, για να αποτραπεί η διπλή εγγραφή.

*Mapping candidates:* Κάνει map κάθε διεύθυνση σε μια struct Student για ανάκτηση στοιχείων.

*candidateList:* Ένας πίνακας που διατηρεί ένα dictionary με τις διευθύνσεις των υποψηφίων, επιτρέποντας έτσι την πρόσβαση στα δεδομένα τους με βάση τη θέση στον πίνακα.

*Mapping hasVoted:* Εξασφαλίζει ότι κάθε χρήστης (address) ψηφίζει μόνο μία φορά, αποτρέποντας την διπλή ψηφοφορία.

### *Functions*

Το contract παρέχει δύο τρόπους εγγραφής υποψηφίων: registerCandidateBySelf και registerCandidateByOwner. Εξηγούνται γενικά τα functions του contract:

### registerCandidateBySelf

Επιτρέπει σε έναν χρήστη να εγγραφεί ως υποψήφιος χρησιμοποιώντας τη δική του διεύθυνση. Ελέγχει αν ο χρήστης έχει ήδη εγγραφεί (μέσω του flag **registered**) και αν όχι, δημιουργεί μία νέα εγγραφή στον mapping **candidates** και προσθέτει το address του στον πίνακα **candidateList**. Καλεί το event CandidateRegistered.

### registerCandidateByOwner

1. Ο owner του contract (η διεύθυνση που το έκανε deploy) μπορεί να εγγράψει έναν υποψήφιο για λογαριασμό του, εισάγοντας τα αντίστοιχα δεδομένα.
2. Ελέγχεται το δικαίωμα πρόσβασης (μόνο ο owner μπορεί να καλέσει αυτή τη συνάρτηση) και επίσης αποτρέπει την διπλή εγγραφή του ίδιου υποψηφίου.

### vote

Η συνάρτηση vote παρέχει τον ασφαλή μηχανισμό για την ψηφοφορία:

1. Επιβεβαιώνει ότι ο υποψήφιος για τον οποίο ψηφίζεται είναι εγγεγραμμένος και ελέγχει ότι ο ψηφοφόρος δεν έχει ήδη ψηφίσει, χρησιμοποιώντας το mapping **hasVoted**.
2. Σε περίπτωση που οι έλεγχοι είναι επιτυχείς, αυξάνεται το **voteCount** του επιλεγμένου υποψηφίου και θέτει το flag στο mapping για να καταγράψει ότι ο ψηφοφόρος ψήφισε ήδη.
3. Καλείται το event **VoteCast**, που βοηθά στο logging και στην παρακολούθηση της διαδικασίας ψηφοφορίας.

### getVoteCount

1. Επιστρέφεται το πλήθος ψήφων για έναν συγκεκριμένο υποψήφιο. (χρήσιμο για εμφάνιση των αποτελεσμάτων σε πραγματικό χρόνο).

### getCandidatesCount

Επιστρέφει τον συνολικό αριθμό των εγγεγραμμένων υποψηφίων.

### getCandidate

Με τη χρήση ενός index, η συνάρτηση επιτρέπει την ανάκτηση όλων των στοιχείων ενός υποψηφίου μέσω της διεύθυνσης που βρίσκεται στην αντίστοιχη θέση του πίνακα **candidateList** (δηλαδή ανάλογα με την σειρά που εισήχθησαν ως **candidates** κατά το register process). Αρχικά γίνεται έλεγχος εγκυρότητας του index για να διασφαλιστεί ότι δεν γίνεται προσπάθεια πρόσβασης σε μη υπάρχοντα στοιχεία. Είναι μια συνάρτηση που μπορεί μελλοντικά να χρησιμοποιηθεί από κάποιο ui εξωτερικά / εσωτερικά για να εμφανίζει σε μια λίστα όλους τους υποψηφιους μαζί με τα στοιχεία τους.

Επόμενα βήματα:

Υπάρχουν 2 διαφορετικά σενάρια για να δούμε 1. Τη λίστα των υποψηφίων της candidateList ταξινομημένους (με ή χωρίς fixed size length της νέας ταξινομημένης λίστας) και 2. μόνο τον νικητή σε πραγματικό χρόνο candidate.

Η συνάρτηση winnerCandidate αναζητεί μόνο τον υποψήφιο με τις περισσότερες ψήφους. Για πιο σύνθετο reporting, μπορεί να προστεθεί μία function όπως πχ. getSortedCandidates που θα επιστρέφει μια ταξινομημένη λίστα βάσει των ψήφων και του αρχικού index = σε περίπτωση που έχουμε ίδιο αριθμό ψήφων.

Για μια τέτοια λειτουργία που απαιτεί ταξινόμηση μεγάλων δεδομένων (όπως η πλήρης κατάταξη όλων των υποψηφίων), μια υλοποίηση off-chain θα ήταν κατάλληλη, για να αποφευχθεί το υψηλό κόστος gas και να παρέχονται ενημερωμένα results στο ui.

Στην προκείμενη περίπτωση θα υλοποιηθεί η *winnerCandidate*: **winnerCandidate**

Η συνάρτηση ελέγχει αν υπάρχει τουλάχιστον ένας υποψήφιος. Θεωρείται ότι ο πρώτος υποψήφιος είναι νικητής και στη συνέχεια διατρέχεται η candidateList. Αν βρούμε κάποιον με περισσότερες ψήφους, ενημερώνουμε τον νικητή.

### getVotingSummary

Επιστρέφεται ένας συνοπτικός πίνακας των αποτελεσμάτων της ψηφοφορίας, συγκεκριμένα:

- **voted**: Ο αριθμός των ατόμων που έχουν ήδη ψηφίσει (δηλαδή, πόσοι έχουν ψηφίσει και καταχωρηθεί μέσω της votedCount μεταβλητής).
- **notVoted**: Ο αριθμός των ατόμων που δεν έχουν ψηφίσει, ο οποίος υπολογίζεται ως η διαφορά μεταξύ του συνολικού αριθμού των εγγεγραμμένων υποψηφίων (δηλαδή candidateList.length) και του votedCount. (με την προυπόθεση ότι όλοι οι candidates είναι eligible. Διαφορετικά θα προσαρμόζαμε την μέθοδο).

(Μπορεί να προστεθεί ακόμα και περιορισμός στο πόσοι candidates θα είναι εκλέξιμοι, την οποία function θα την ελέγχει μόνο ο owner. ) – υπάρχει σαν σχόλιο στο source code.

```
function setTotalEligibleVoters(uint _total) public {
    require(msg.sender == owner, "Only owner can set total eligible voters");
    totalEligibleVoters = _total;
}
```

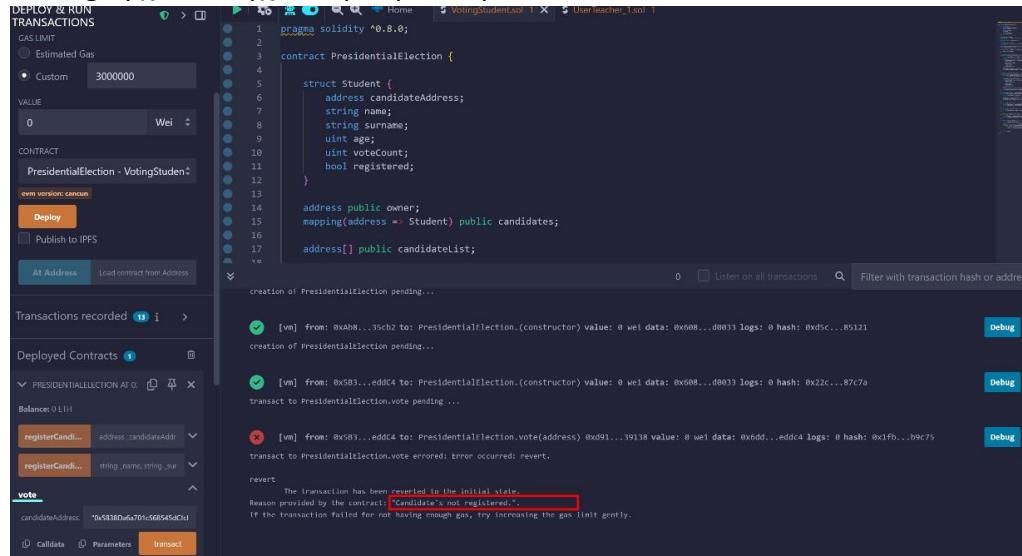
Έτσι μετά θα είχαμε

```
uint public totalEligibleVoters;
///
function getVotingSummary() public view returns (uint voted, uint notVoted) {
    voted = votedCount;
```

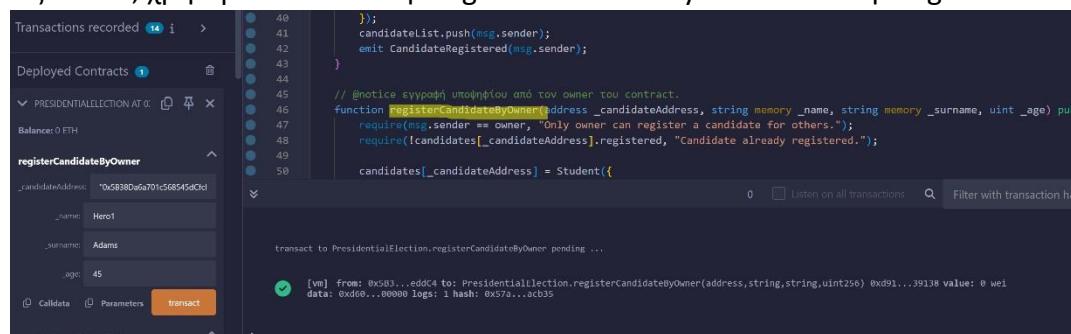
```
    notVoted = totalEligibleVoters - votedCount;
}
```

## Παραδείγματα Εκτέλεσης

Πχ. Θέλουμε να εγγράψουμε τον εαυτό μας ως candidate: Deploy το smart contract και για testing αρχικά ελέγχω αν μπορώ να με κάνω vote:



Ως owner, χρησιμοποιώ είτε την registerCandidateByOwner είτε την registerCandidateBySelf.



```

output
decoded input
[{"address _candidateAddress": "0x58380a6a701c568545dCfcB83FcB875f56beddC4",
 "string _name": "Hero1",
 "string _surname": "Adams",
 "uint256 _age": "45"}]

decoded output
logs
[{"from": "0xd9145CE52D386f254917e481e844e9943F39138",
 "topic": "0xa0a4c4f95ab0bbddc4b80883add9e055e02c796b2cde2ed80fdh11a38f77e5c9",
 "event": "CandidateRegistered",
 "args": {
     "0": "0x58380a6a701c568545dCfcB875f56beddC4",
     "candidate": "0x58380a6a701c568545dCfcB83FcB875f56beddC4"
   }
}]

raw logs
[{"logIndex": "0x1",
 "blockNumber": "0x3",
 "blockHash": "0x7cd26352232351fb0525fe280923c43d2ad7fc0d60adbe86f2560dc5828419d",
 "transactionHash": "0x57ada44c3b94780e727550b7a8e388334beef2d39c067d7ff73a041e7aacb35",
 "transactionIndex": "0x0"}]

```

Μπορώ να κάνω vote τον εαυτό μου:

Deployed Contracts

- PRESIDENTIALELECTION AT 0:
- Balance: 0 ETH
- registerCandidate** address \_candidateAddress
- registerCandidate** string \_name, string \_surname
- vote**
- \_candidateAddress: **0x58380a6a701c568545dCfc**
- Calldata Parameters **transact**
- candidateList uint256

43

execution cost 49246 gas

input 0x6dd...eddc4

output ex

decoded input

```

{
    "address _candidateAddress": "0x58380a6a701c568545dCfcB83FcB875f56beddC4"
}
```

decoded output

logs

```

[{"from": "0xd9145CE52D386f254917e481e844e9943F39138",
 "topic": "0x111ed0b18008fe2035a16f110674645020920ff1015abfe08dbe9a2971a7e2",
 "event": "VoteCast",
 "args": {}}
]
```

Έλεγχος για το πόσους ψήφους έχω:

getCandidate...

getVoteCount 0x58380a6a701c568545

0: uint256: 1

hasVoted address

owner

winnerCandidate...

decoded input

```

{
    "address _candidateAddress": "0x58380a6a701c568545dCfcB83FcB875f56beddC4"
}
```

decoded output

```

{
    "0": "0x000000000000000000000000000000000000000000000000000000000000000"
}
```

logs

raw logs

Δημιουργώ νέο candidate χρησιμοποιώντας νέο address, προσπαθώ να χρησιμοποιήσω την registerCandidateByOwner:

Deployed Contracts

- PRESIDENTIALELECTION AT 0:
- Balance: 0 ETH
- registerCandidateByOwner**
- candidateAddress: **0xAb483F64d9C6d1Ecf9b845**
- \_name: Hero2
- \_surname: Christoforou
- age: 67
- Calldata Parameters **transact**

logs []

raw logs []

transact to PresidentialElection.registerCandidateByOwner pending ...

[x] [vm] from: 0xAb8...35cb2 to: PresidentialElection.registerCandidateByOwner(address,string,string,uint256) 0x7EF...8CB47 value: 0 w data: 0xd60...00000 logs: 0 hash: 0x817...8de4e transact to PresidentialElection.registerCandidateByOwner errored: Error occurred: revert.

revert The transaction has been reverted to the initial state.  
Reason provided by the contract: "Only owner can register a candidate for others.".  
If the transaction failed for not having enough gas, try increasing the gas limit gently.

## Check :

At Address Load contract from Address

Transactions recorded 1 >

Deployed Contracts 1

PRESIDENTIALELECTION AT 0:

Balance: 0 ETH

registerCandidateBySelf

registerCandidateBySelf

name: Hero2

surname: Xristoforou

age: 67

Calldata Parameters transact

status 0x1 Transaction mined and execution succeed

transaction hash 0xc039da0abcf...004d2

block hash 0xc039da0abcf...3d315835cb2

block number 9

from 0xAb8483F64d9C6d1EcF9b849Ae677d02315835cb2

to PresidentialElection.registerCandidateBySelf(string,string,uint256) 0x7ef2e0048f5bd4e0e46f68f797943da...04ED08C47

gas 192251 gas

transaction cost 167174 gas

execution cost 144962 gas

input 0x20...00000

output 0x

decoded input { "string \_name": "Hero2", }

... και

decoded input { "string \_name": "Hero2", "string \_surname": "Xristoforou", "uint256 \_age": "67" }

Θελω να δω τις πληροφοριες του candidate με address 0x..C4:

candidates 0x803FcB875f56beddC4

0: address: candidateAddress 0xAb8483F64d9C6d1EcF9b849Ae677d02315835cb2

1: string: name Hero2

2: string: surname Xristoforou

3: uint256: age 67

4: uint256: voteCount 0

5: bool: registered true

getCandidate uint256 index

decoded input { "address": "0x5838Da6a701c568545dFcB03FcB875f56beddC4" }

decoded output { "0": "address: candidateAddress 0x5838Da6a701c568545dFcB03FcB875f56beddC4", "1": "string: name Hero1", "2": "string: surname Papanikolh", "3": "uint256: age 45", "4": "uint256: voteCount 2", "5": "bool: registered true" }

Ή του πρώτου που έκανε εγγραφή

candidateList uint256

candidates address

0: address: candidateAddress 0x00000000000000000000000000000000

1: string: name

2: string: surname

3: uint256: age 0

4: uint256: voteCount 0

5: bool: registered false

getCandidate 0

decoded input { "uint256 index": "0" }

decoded output { "0": "tuple(address,string,string,uint256,uint256,bool): 0x5838Da6a701c568545dFcB03FcB875f56beddC4(Hero1,Papanikolh,45,2,true)" }

logs []

raw logs []

...Ή εκείνου που σε πραγματικό χρόνο, τη στιγμή που κάνω την κλήση, έχει τους περισσοτέρους ψήφους – έρχονται όλες οι πληροφορίες του, και οι ψήφοι που έχει :



Αν προσπαθήσω να βρω πόσες ψήφους έχει ένα candidate address που δεν έχει καταχωρηθεί για το smart contract

getVotingSummary :

O President Student:

Εννοείται αυτό το σενάριο smart contract μπορεί να επεκταθεί με πάρα πολλούς τρόπους: Μπορούν να προστεθούν τα εξής για παράδειγμα

1. Ορισμός περιόδου ψηφοφορίας
2. Περισσότεροι ρόλοι - επίπεδα πρόσθιασης με πιο αυστηρά καθορισμένες λειτουργικότητες για τον καθένα.