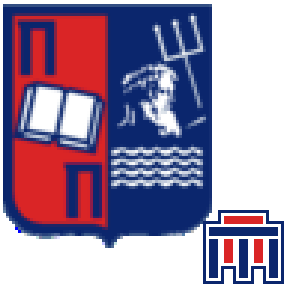
Πανεπιστήμιο Πειραιώς

Τμήμα Πληροφορικής



ΠΜΣ Κυβερνοασφάλεια και Επιστήμη Δεδομένων

Advanced Cryptographic and Security Technologies (CDS 211)

Ομαδική Απαλλακτική Εργασία

Χριστοφόρος	Κλάδης	ΜΠΚΕΔ 2316
Κωνσταντίνος	Κούφαλης	ΜΠΚΕΔ 2319
Θεόδωρος	Παπαγεωργίου	ΜΠΚΕΔ 2331

Περιεχόμενα

Εισαγωγή

Σκοπός της παρούσας εργασίας είναι η σχεδίαση και υλοποίηση μιας εφαρμογής ιχνηλάτησης φαρμακευτικών προϊόντων με βάση την τεχνολογία Blockchain και τη γλώσσα προγραμματισμού Solidity.

Οι παραδοσιακές μέθοδοι παρακολούθησης συχνά δεν επαρκούν λόγω της ευαισθησίας τους στην απάτη, τα σφάλματα και την αναποτελεσματικότητα. Η τεχνολογία blockchain παρουσιάζει μια επαναστατική λύση σε αυτές τις προκλήσεις, προσφέροντας ένα αμετάβλητο και διαφανές σύστημα τήρησης αρχείων για την παρακολούθηση των φαρμακευτικών προϊόντων από την κατασκευή έως την παράδοση.

Η αποκεντρωμένη φύση του blockchain εξασφαλίζει ότι καμία οντότητα δεν έχει τον έλεγχο ολόκληρης της αλυσίδας εφοδιασμού, μειώνοντας τον κίνδυνο παραποίησης και απάτης. Κάθε συναλλαγή ή αλλαγή φύλαξης στην αλυσίδα εφοδιασμού καταγράφεται με ασφαλή, αμετάβλητο τρόπο, παρέχοντας μια σαφή και ελέγξιμη παρακολούθηση της διαδρομής ενός προϊόντος. Τα έξυπνα συμβόλαια διευκολύνουν τις αυτοματοποιημένες και αναξιόπιστες συναλλαγές, διασφαλίζοντας ότι τα προϊόντα κινούνται μέσω της αλυσίδας εφοδιασμού σύμφωνα με προκαθορισμένους κανόνες και προϋποθέσεις.

Εφαρμόζοντας ένα σύστημα εντοπισμού με βάση το blockchain, οι φαρμακευτικές εταιρείες μπορούν να εγγυηθούν τη γνησιότητα των προϊόντων τους, να προφυλαχθούν από τα πλαστά φάρμακα και να ενισχύσουν τη συνολική αποτελεσματικότητα της αλυσίδας εφοδιασμού. Η τεχνολογία αυτή όχι μόνο προστατεύει τους καταναλωτές, αλλά και ενισχύει μεγαλύτερη εμπιστοσύνη και διαφάνεια στη φαρμακευτική βιομηχανία.

Αρχιτεκτονική αποκεντρωμένων εφαρμογών

Η αρχιτεκτονική των αποκεντρωμένων εφαρμογών (dApps) περιλαμβάνει διάφορες λειτουργίες οι οποίες συμβάλλουν στη σωστή και ασφαλή λειτουργία αυτών.

Αρχικά, το **Blockchain** είναι η βασική τεχνολογία πίσω από τέτοιου είδους εφαρμογές. Αποτελείται επιμέρους κόμβους (nodes). Κάθε κόμβος διατηρεί ένα αντίγραφο του ledger (αποθετήριο για όλες τις συναλλαγές και τις καταστάσεις των λογαριασμών). Η ακεραιότητα και η ασφάλεια στο δίκτυο διασφαλίζεται με τους μηχανισμούς συναίνεσης (Proof of Work, Proof of Stake).

Τα **έξυπνα συμβόλαια (Smart Contracts)** στα οποία περιέχεται ο κώδικας αυτών των εφαρμογών, αποτελούν τη βάση κάθε εφαρμογής, καθώς μέσω αυτών καθορίζονται η λογική και οι κανόνες που διέπουν τις λειτουργίες της. Τα έξυπνα συμβόλαια γράφονται σε συγκεκριμένες γλώσσες προγραμματισμού, όπως το Solidity (για Ethereum) και εκτελούνται στους κόμβους του blockchain.

Επιπλέον, τα dApps χρησιμοποιούν **αποκεντρωμένα συστήματα αποθήκευσης** για δεδομένα και αρχεία (π.χ., το IPFS – InterPlanetary File System), αυξάνοντας με αυτόν τον τρόπο την ασφάλεια και την ακεραιότητα των δεδομένων, καθώς τα δεδομένα δεν αποθηκεύονται σε έναν μόνο διακομιστή αλλά κατανέμονται σε πολλούς κόμβους.

Όσον αφορά το UI των dApps, αυτό επικοινωνεί με το Blockchain και τα Smart Contracts μέσω ειδικών βιβλιοθηκών (π.χ., Web3.js). Το front-end των dApps αναπτύσσεται με τη βοήθεια διαφόρων frameworks (π.χ., React, Angular), τα οποία δημιουργούν διαδραστικά UIs. Ταυτόχρονα, μέσω ειδικών wallets (π.χ., Metamask) μπορούν οι χρήστες να διαχειρίζονται τα κρυπτονομίσματά τους και να υπογράφουν συναλλαγές.

Σε επίπεδο επικοινωνίας, το RPC (Remote Procedure Call) είναι ένα πρωτόκολλο που χρησιμοποιείται για την επικοινωνία μεταξύ front-end και κόμβων του blockchain.

Τεχνολογία Blockchain - Ιχνηλασιμότητα στην εφοδιαστική αλυσίδα

Είναι γεγονός πως κάθε συναλλαγή που πραγματοποιείται και καταγράφεται στο Blockchain μπορεί να επαληθευτεί από όλους όσους συμμετέχουν σε αυτή. Παράλληλα, όλες οι εγγραφές στο blockchain είναι κρυπτογραφημένες, καθιστώντας τις συναλλαγές σε μεγάλο βαθμό ασφαλείς και διατηρώντας την ακεραιότητα των δεδομένων. Με αυτόν τον τρόπο ενισχύεται σε μεγάλο βαθμό η εμπιστοσύνη μεταξύ όλων των εμπλεκομένων στην εφοδιαστική αλυσίδα (κατασκευαστές, προμηθευτές, μεταφορείς, καταναλωτές).

Όσον αφορά την ιχνηλασιμότητα των προϊόντων στην εφοδιαστική αλυσίδα, η διαχείριση αυτής μπορεί να πραγματοποιηθεί ευκολότερα και με λιγότερο χρόνο και κόστος, το οποίο προφανώς οφείλεται στην αυτοματοποίηση των διαδικασιών. Επιπλέον, η παρακολούθηση της πορείας ενός προϊόντος σε πραγματικό χρόνο (από την παραγωγή στο εργοστάσιο, την προώθησή του μέσω του προμηθευτή και εν τέλει την κατανάλωσή του) εξυπηρετεί την καλύτερη διαχείρισή του. Παράλληλα, διευκολύνεται ο έλεγχος της ταυτότητας των προϊόντων με τη δημιουργία μοναδικών ψηφιακών ταυτοτήτων για κάθε προϊόν. Με αυτόν τον τρόπο, μπορεί να επαληθευτεί η προέλευση και η γνησιότητα αυτών των προϊόντων από τους συμμετέχοντες στην εφοδιαστική αλυσίδα.

Δομή και Λειτουργία Έξυπνου Συμβολαίου

1. Ρόλοι και Οντότητες

Καθορίζουμε τους ρόλους των χρηστών χρησιμοποιώντας τον τύπο δεδομένων **enum Role {Admin, Supplier, Logistic, Auditor}**.

Επιπλέον, καθορίζουμε τους τύπους των οντοτήτων που συμμετέχουν στην εφοδιαστική αλυσίδα χρησιμοποιώντας τον τύπο δεδομένων **enum EntityType {Supplier, Transportation, Manufacturer, Warehouse_Logistic, Distributor, Pharmacy}**.

```
contract Pharmacy {
    enum Role {Admin, Supplier, Logistic, Auditor}
     enum EntityType {Supplier, Transportation, Manufacturer, Warehouse_Logistic,
Distributor, Pharmacy}
```

2. Δομές Δεδομένων

Η δομή **“Product”** καθορίζει τα δεδομένα που σχετίζονται με ένα προϊόν (**id, name, quantity, currentScEntity**).

```
struct Product {
        uint256 id;
        string name;
        uint256 quantity;
        address currentScEntity;
    }
```

Η δομή **“Shipment”** καθορίζει τα δεδομένα που σχετίζονται με μια αποστολή (**shipment_id, origin, destination, date_of_departure, expected_date_of_arrival, product_id**).

```
struct Shipment {
        uint256 shipment_id;
        address origin;
        address destination;
        uint256 date_of_departure;
        uint256 expected_date_of_arrival;
        uint256 product_id;
    }
```

Η δομή **“ScEntity”** καθορίζει τα δεδομένα που σχετίζονται με μια οντότητα στην εφοδιαστική αλυσίδα (**entity_address, name, entity_type**).

```
    struct ScEntity {
        address entity_address;
        string name;
        EntityType entity_type;
    }
```

Η δομή **“User”** καθορίζει τα δεδομένα που σχετίζονται με έναν χρήστη (**user_address, name, role**).

```
   struct User {
        address user_address;
        string name;
        Role role;
    }
```

3. Mappings

Χρησιμοποιούνται τα ακόλουθα mappings για τη διαχείριση και την παρακολούθηση δεδομένων (χρήστες, οντότητες, προϊόντα και αποστολές) στο σύστημα εφοδιαστικής αλυσίδας.

```
mapping(address => User) public users;
mapping(address => uint256) public scEntities_index;
mapping(uint256 => uint256) public products_index;
mapping(address => uint256[]) public entities_products;
mapping(uint256 => uint256[]) public products_shipments;
```

Το **“mapping(address => User) public users”** αποθηκεύει πληροφορίες σχετικά με τους χρήστες στο σύστημα. Αντιστοιχεί μια διεύθυνση Ethereum σε μια δομή User, η οποία περιέχει πληροφορίες του χρήστη.

Το **“mapping(address => uint256) public scEntities_index”** χρησιμοποιείται για την καταχώριση των οντοτήτων της αλυσίδας εφοδιασμού. Αντιστοιχεί μια διεύθυνση Ethereum σε έναν δείκτη uint256. Αυτός ο δείκτης αντιστοιχεί στη θέση της δομής ScEntity στον πίνακα scEntities_list**.**

Το **“mapping(uint256 => uint256) public products_index”** χρησιμοποιείται για την καταχώριση των προϊόντων. Αντιστοιχεί ένα product id (uint256) σε έναν δείκτη uint256. Αυτός ο δείκτης αντιστοιχεί στη θέση της δομής Product στον πίνακα products_list.

Το **“mapping(address => uint256[]) public entities_products”** παρακολουθεί τα προϊόντα που σχετίζονται με κάθε οντότητα της αλυσίδας εφοδιασμού. Αντιστοιχεί μια διεύθυνση Ethereum (που αντιπροσωπεύει μια οντότητα) σε έναν πίνακα product ids (uint256). Αυτός ο πίνακας περιέχει τα ids των προϊόντων που κατέχει τη δεδομένη στιγμή η οντότητα.

Το **“mapping(uint256 => uint256[]) public products_shipments”** παρακολουθεί τις αποστολές που σχετίζονται με κάθε προϊόν. Αντιστοιχεί ένα product id (uint256) σε έναν πίνακα shipment ids (uint256). Αυτός ο πίνακας περιέχει τα ids των αποστολών που έχουν πραγματοποιηθεί για το προϊόν.

4. **Modifiers**

Ο modifier **“only_for_role(Role role)”** χρησιμοποιείται για να περιορίσει την πρόσβαση σε χρήστες με συγκεκριμένο ρόλο.

```
modifier only_for_role(Role role) {
        require(users[msg.sender].role == role, "Not authorized");
        _;
    }
```

5. **Events**

Τα παρακάτω events χρησιμοποιούνται για την καταγραφή πληροφοριών σχετικά με συγκεκριμένες ενέργειες και αλλαγές κατάστασης στο σύστημα. (Τα events δεν χρησιμοποιούνται κατά την επικοινωνία με το frontend).

```
 // events for Users
    event UserCreated(address user_address, string name, Role role);
    event UserRemoved(address user_address);

    // events for Shipments
         event ShipmentCreated(uint256 shipment_id, address origin, address
destination, uint256 date_of_departure, uint256 expected_date_of_arrival, uint256
product_id);
```

```
    // events for Products
      event ProductCreated(uint256 id, string name, uint256 quantity, address
currentScEntity);
    event ProductRemoved(uint256 id);

    // events for ScEntities
       event ScEntityCreated(address entity_address, string name, EntityType
entity_type);
    event ScEntityRemoved(address entity_address);
```

Συγκεκριμένα έχουμε events για όταν:

- Δημιουργείται ένας νέος χρήστης **(event UserCreated)**.
- Αφαιρείται ένας χρήστης **(event UserRemoved)**.
- Δημιουργείται μια νέα αποστολή **(event ShipmentCreated)**.
- Δημιουργείται ένα νέο προϊόν **(event ProductCreated)**.
- Αφαιρείται ένα προϊόν **(event ProductRemoved)**.
- Δημιουργείται μια νέα οντότητα της εφοδιαστικής αλυσίδας **(event ScEntityCreated)**.
- Αφαιρείται μια οντότητα της εφοδιαστικής αλυσίδας **(event ScEntityRemoved)**.

6. **Constructor**

Ο constructor αρχικοποιεί το έξυπνο συμβόλαιο με προκαθορισμένο χρήστη. Ο χρήστης admin δημιουργείται με τη διεύθυνση του υπεύθυνου που κάνει deploy το συμβόλαιο.

```
constructor() {
        // Admin user
        User memory newUser = User({
            user_address: msg.sender,
            name: "Admin",
            role: Role.Admin
        });
        users[msg.sender] = newUser;
    }
```

7. **Συναρτήσεις**

Οι συναρτήσεις που χρησιμοποιήθηκαν για την διαχείριση των χρηστών είναι οι εξής:

- create_user -> Δημιουργία χρηστών
- remove_user -> Αφαίρεση χρηστών
- view_user -> Προβολή χρηστών

Ο κώδικας της συνάρτησης **create_user** φαίνεται παρακάτω:

```
function create_user(string memory name, Role role, address user_address) public
only_for_role(Role.Admin) {
```

```
        require(users[user_address].user_address != user_address, "User already exists");

        uint256 entityIndex = scEntities_index[user_address];
                        if (entityIndex < scEntities_list.length && scEntities_list[entityIndex].entity_address == user_address) {
            EntityType entityType = scEntities_list[entityIndex].entity_type;
            if (entityType == EntityType.Supplier) {
                    require(role == Role.Supplier, "User role must be Supplier for Supplier entity");
            } else if (entityType == EntityType.Warehouse_Logistic) {
                    require(role == Role.Logistic, "User role must be Logistic for Warehouse Logistic entity");
            } else {
                  require(role != Role.Supplier && role != Role.Logistic, "Invalid role for this entity");
            }
        }

        User memory newUser = User({
            user_address: user_address,
            name: name,
            role: role
        });
        users[user_address] = newUser;

        emit UserCreated(newUser.user_address,newUser.name,newUser.role);
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Μόνο εξουσιοδοτημένοι χρήστες (administrators) μπορούν να προσθέσουν νέους χρήστες μέσω του modifier **“only_for_role(Role.Admin)”**.
- Αποτρέπει την διπλή εγγραφή χρηστών μέσω της συνθήκης **“require(users[user_address].user_address != user_address)”** η οποία συγκρίνει την παρεχόμενη διεύθυνση user_address με την αποθηκευμένη διεύθυνση user_address στο users mapping. Εάν ο χρήστης υπάρχει επιστρέφεται μήνυμα "User already exists!".
- Αν το address έχει χρησιμοποιηθεί για την δημιουργία ενός Entity, τότε ο ρόλος του χρήστη πρέπει να είναι αντίστοιχος με το Entity Type. Δηλαδή, αν το Entity Type είναι Supplier τότε και ο χρήστης πρέπει να είναι Supplier, αν το Entity Type είναι Warehouse Logisitc τότε και ο χρήστης πρέπει να είναι Logisitic και αν το Entity Type είναι κάτι άλλο, τότε ο χρήστης δεν μπορεί να δημιουργηθεί.

- Αν ο χρήστης δεν υπάρχει ήδη, δημιουργείται μια νέα δομή User με τα στοιχεία που παρέχονται, και αυτός ο νέος χρήστης προστίθεται στη συνέχεια στην users mapping, με τη διεύθυνση user_address ως κλειδί.

Ο κώδικας της συνάρτησης **remove_user** φαίνεται παρακάτω:

```
function remove_user(address user_address) public only_for_role(Role.Admin) {
         require(users[user_address].user_address == user_address,"User does not
exist!");
        require(msg.sender != user_address,"Admin can't remove himself!");

        delete users[user_address];

        emit UserRemoved(user_address);
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Μόνο εξουσιοδοτημένοι χρήστες (administrators) μπορούν να αφαιρέσουν χρήστες μέσω του modifier **“only_for_role(Role.Admin)”**.
- Μέσω της συνθήκης **“require(users[user_address].user_address == user_address)”** γίνεται έλεγχος αν ο χρήστης με τη παρεχόμενη διεύθυνση user_address υπάρχει στο σύστημα. Εάν ο χρήστης δεν υπάρχει επιστρέφεται μήνυμα "User does not exist!".
- Μέσω της συνθήκης **“require(msg.sender != user_address)”** διασφαλίζεται ότι ένας διαχειριστής δεν μπορεί να αφαιρέσει το δικό του λογαριασμό.Εάν ο διαχειριστής προσπαθήσει να αφαιρέσει τον εαυτό του, θα επιστραφεί μήνυμα "Admin can't remove himself!".
- Η εντολή **“delete users[user_address]”** αφαιρεί τον χρήστη από το users mapping, διαγράφοντας τα δεδομένα του χρήστη από το σύστημα.

Ο κώδικας της συνάρτησης **view_user** φαίνεται παρακάτω:

```
function view_user(address user_address) public view returns (User memory) {
         require(users[user_address].user_address == user_address,"User does not
exist!");

        return users[user_address];
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Η συνθήκη **“require(users[user_address].user_address == user_address)”** ελέγχει αν ο χρήστης υπάρχει στο mapping. Εάν ο χρήστης δεν υπάρχει επιστρέφεται μήνυμα "User does not exist!".

- Εάν ο χρήστης υπάρχει, η εντολή **“return users[user_address]”** επιστρέφει τη δομή User που σχετίζεται με την παρεχόμενη διεύθυνση user_address. Τα δεδομένα ανακτώνται από το users mapping.

Οι συναρτήσεις που χρησιμοποιήθηκαν για την διαχείριση των αποστολών είναι οι εξής:

- create_shipment -> Δημιουργία αποστολών
- get_shipment -> Ανάκτηση στοιχείων μιας αποστολής
- get_shipments -> Ανάκτηση στοιχείων όλων των αποστολών
- get_product_shipments -> Ανάκτηση στοιχείων αποστολής για ένα συγκεκριμένο προϊόν

Ο κώδικας της συνάρτησης **create_shipment** φαίνεται παρακάτω:

```
  function create_shipment(address destination, uint256 product_id, uint256
expected_date_of_arrival) public {
            require(block.timestamp < expected_date_of_arrival,"Expected date of
arrival should be after date of departure!");

        uint256 p_index = products_index[product_id];
         require(p_index != 0 || products_list[0].id == product_id,"Product does
not exist!");

        uint256 destination_index = scEntities_index[destination];
                        require(destination_index < scEntities_list.length &&
scEntities_list[destination_index].entity_address == destination, "Entity does
not exist!");

        Product memory product = products_list[p_index];

        uint256 origin_index = scEntities_index[product.currentScEntity];
           require(uint256(scEntities_list[destination_index].entity_type) > 0 &&
uint256(scEntities_list[origin_index].entity_type)                              ==
uint256(scEntities_list[destination_index].entity_type)    -    1    ,    "Wrong
destination!");

        Shipment memory newShipment = Shipment({
            shipment_id: shipments_list.length,
            origin: product.currentScEntity,
            destination: destination,
            date_of_departure: block.timestamp,
            expected_date_of_arrival: expected_date_of_arrival,
            product_id: product_id
        });

        // Save shipment to list for this product
        products_shipments[product_id].push(newShipment.shipment_id);
```

```
            // Adds product_id to Logistic employee (if entity is a Logistic
warehouse )
                        if(   scEntities_list[destination_index].entity_type   ==
EntityType.Warehouse_Logistic ){
            entities_products[destination].push(product_id);
        }

        products_list[p_index].currentScEntity = destination;
        shipments_list.push(newShipment);

              emit  ShipmentCreated(newShipment.shipment_id,  newShipment.origin,
newShipment.destination,                          newShipment.date_of_departure,
newShipment.expected_date_of_arrival,  newShipment.product_id);
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Η συνθήκη **“require(block.timestamp < expected_date_of_arrival)”** ελέγχει εάν η αναμενόμενη ημερομηνία άφιξης είναι μεταγενέστερη σε σχέση με την τρέχουσα χρονοσήμανση (timestamp). Εάν η συνθήκη δεν ικανοποιείται, επιστρέφεται μήνυμα: “Expected date of arrival should be after date of departure!”.
- Ανακτά το δείκτη του προϊόντος χρησιμοποιώντας το **products_index[product_id]**. Στη συνέχεια, η συνθήκη **“require(p_index != 0 || products_list[0].id == product_id)”** ελέγχει αν το προϊόν υπάρχει στη λίστα **products_list**. Εάν το προϊόν δεν υπάρχει, επιστρέφεται μήνυμα: "Product does not exist!".
- Ανακτά το δείκτη της οντότητας προορισμού χρησιμοποιώντας το **scEntities_index[destination]**. Η συνθήκη **“require(destination_index < scEntities_list.length && scEntities_list [destination_index].entity_address == destination”** ελέγχει αν η οντότητα προορισμού υπάρχει στη λίστα **scEntities_list** και ταιριάζει με τη δεδομένη διεύθυνση. Εάν όχι, επιστρέφεται μήνυμα: "Entity does not exist!"
- Ανακτά τα στοιχεία του προϊόντος από τη λίστα **products_list** χρησιμοποιώντας το δείκτη προϊόντος. Επίσης, ανακτά το δείκτη της οντότητας προέλευσης χρησιμοποιώντας το **scEntities_index[product.currentScEntity]**.
- Η συνθήκη **“require(uint256(scEntities_list[destination_index].entity_type) > 0 && uint256(scEntities_list[origin_index].entity_type) == uint256(scEntities_list[destination_index] .entity_type) - 1”** ελέγχει ότι ο τύπος οντότητας προορισμού είναι κατάλληλος με βάση τον τύπο οντότητας προέλευσης. Συγκεκριμένα, ο τύπος οντότητας προορισμού πρέπει να είναι ακριβώς ένας τύπος μεγαλύτερος από τον τύπο οντότητας προέλευσης. Εάν αυτή η συνθήκη δεν ικανοποιείται, η συνάρτηση επιστρέφει με: "Wrong destination!"
- Δημιουργείται μια νέα εγγραφή αποστολής με τα στοιχεία (shipment_id, origin, destination, date_of_departure, expected_date_of_arrival, product_id). Το νέο shipment_id προστίθεται στη λίστα **products_shipments** για το συγκεκριμένο προϊόν.

- Η συνθήκη **if(scEntities_list[destination_index].entity_type == EntityType.Warehouse_Logistic)”** ελέγχει εάν η οντότητα προορισμού είναι η Warehouse Logistic, τότε το product_id προστίθεται στη λίστα **entities_products** για τον προορισμό.
- Η τρέχουσα θέση του προϊόντος (**currentScEntity**) ενημερώνεται στη διεύθυνση προορισμού με την χρήση του **products_list[p_index].currentScEntity = destination**.
- Η νέα αποστολή προστίθεται στη λίστα **shipments_list** με την χρήση του **shipments_list.push(newShipment)**.

Ο κώδικας της συνάρτησης **get_shipment** φαίνεται παρακάτω:

```
function get_shipment(uint256 shipment_id) public view returns (Shipment memory)
{
        require(shipment_id < shipments_list.length,"Shipment does not exist!");

        return shipments_list[shipment_id];
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Η συνθήκη **“require(shipment_id < shipments_list.length)”** διασφαλίζει ότι το **shipment_id** που παρέχεται βρίσκεται εντός του εύρους της λίστας **shipments_list**. Εάν είναι εκτός ορίων, επιστρέφεται μήνυμα "Shipment does not exist!".
- Εάν η συνθήκη require περάσει, η συνάρτηση ανακτά και επιστρέφει τα στοιχεία της αποστολής που αντιστοιχούν στο δεδομένο shipment_id από την λίστα shipments_list.

Ο κώδικας της συνάρτησης **get_shipments** φαίνεται παρακάτω:

```
function get_shipments() public view returns (Shipment[] memory) {
        return shipments_list;
    }
```

Η παραπάνω συνάρτηση επιστρέφει έναν πίνακα από δομές Shipment (**shipments_list**). Η λέξη-κλειδί memory δηλώνει ότι αυτός ο πίνακας είναι προσωρινό αντίγραφο και υπάρχει μόνο κατά τη διάρκεια της εκτέλεσης της συνάρτησης.

Ο κώδικας της συνάρτησης **get_product_shipments** φαίνεται παρακάτω:

```
   function   get_product_shipments(uint256   product_id)   public   view   returns
(Shipment[] memory) {
            require(products_index[product_id] != 0 || products_list[0].id ==
product_id, "Product does not exist!");

        User memory user = users[msg.sender];
        require(user.user_address == msg.sender, "Not authorized");

        uint256[] memory shipment_ids = products_shipments[product_id];
```

```
        if( user.role == Role.Supplier || user.role == Role.Logistic){
            bool product_belongs_to_user = false;
            uint256[] memory user_products = entities_products[msg.sender];
            for (uint256 i = 0; i < user_products.length; i++) {
                if (user_products[i] == product_id) {
                    product_belongs_to_user = true;
                    break;
                }
            }
                require(product_belongs_to_user, "Product does not belong to the
user");

            uint256 max_shipments = user.role == Role.Supplier ? 2 : 3;
                 uint256 shipment_count = shipment_ids.length < max_shipments ?
shipment_ids.length : max_shipments;

            Shipment[] memory _shipments = new Shipment[](shipment_count);
            for (uint256 i = 0; i < shipment_count; i++) {
                _shipments[i] = shipments_list[shipment_ids[i]];
            }

            return _shipments;
        }

        Shipment[] memory shipments = new Shipment[](shipment_ids.length);
        for (uint256 i = 0; i < shipment_ids.length; i++) {
            shipments[i] = shipments_list[shipment_ids[i]];
        }

        return shipments;
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Με την συνθήκη **“require(products_index[product_id] != 0 || products_list[0].id == product_id)”** ελέγχει αν το προϊόν με **product_id** υπάρχει. Εάν το προϊόν δεν υπάρχει, επιστρέφεται μήνυμα "Product does not exist!".
- Ανακτά τη δομή User για τον καλούντα (msg.sender). Με την συνθήκη **“require(user.user_address == msg.sender”** ελέγχεται αν η διεύθυνση user_address του ανακτηθέντος χρήστη ταιριάζει με το msg.sender για να διασφαλίσει ότι ο καλών είναι εξουσιοδοτημένος. Εάν ο καλών δεν είναι εξουσιοδοτημένος, επιστρέφεται μήνυμα: "Not authorized".
- Ανακτά τον πίνακα shipment_ids που σχετίζονται με το δεδομένο product_id από το products_shipments.

- Με την συνθήκη **“if(user.role == Role.Supplier || user.role == Role.Logistic)”** ελέγχεται αν ο χρήστης είναι ο Supplier ή ο υπάλληλος Logistic. Εάν είναι, τότε ελέγχει αν το προϊόν ανήκει στον χρήστη (**product_belongs_to_user**) με προσπέλαση του πίνακα user_products που σχετίζεται με τον αποστολέα msg.sender. Εάν το προϊόν δεν ανήκει στον χρήστη, επιστρέφεται μήνυμα: "Product does not belong to the user".
- Με την χρήση του **“max_shipments = user.role == Role.Supplier ? 2 : 3”** περιορίζεται ο αριθμός των αποστολών που επιστρέφονται βάσει του ρόλου του χρήστη: Οι Suppliers μπορούν να δουν έως και 2 αποστολές, ενώ οι υπάλληλοι Logistic μπορούν να δουν έως και 3 αποστολές.
- Δημιουργείται ένας πίνακα αποστολών Shipment και γεμίζει με τα στοιχεία της αποστολής από τη λίστα shipments_list.
- Εάν ο χρήστης δεν έχει ρόλο Supplier ή Logistic, δημιουργείται ένας πίνακα αποστολών Shipment και τον γεμίζει με τα στοιχεία της αποστολής από τη λίστα shipments_list.

Οι συναρτήσεις που χρησιμοποιήθηκαν για την διαχείριση των προϊόντων είναι οι εξής:

- create_product -> Δημιουργία προϊόντων
- remove_product -> Αφαίρεση προϊόντων
- get_product -> Ανάκτηση στοιχείων ενός προϊόντος
- get_products -> Ανάκτηση στοιχείων όλων των προϊόντων
- get_product_entity -> Ανάκτηση στοιχείων οντότητας για ένα συγκεκριμένο προϊόν

Η συνάρτηση **create_product** φαίνεται παρακάτω:

```
        function create_product(string memory name, uint256 quantity, address
currentScEntity) public {
        uint256 index = scEntities_index[currentScEntity];
                                require(index    <    scEntities_list.length    &&
scEntities_list[index].entity_address == currentScEntity, "Cannot create product
because Entity does not exist!");
              require(scEntities_list[index].entity_type == EntityType.Supplier,
"Cannot create product because Entity is not a Supplier!");

        Product memory newProduct = Product({
            id: products_ids++,
            name: name,
            quantity: quantity,
            currentScEntity: currentScEntity
        });

        products_index[newProduct.id] = products_list.length;
        products_list.push(newProduct);

        // adds product to supplier
        entities_products[currentScEntity].push(newProduct.id);
```

```
        emit ProductCreated(newProduct.id, newProduct.name, newProduct.quantity,
newProduct.currentScEntity);
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Η συνθήκη “**require(index < scEntities_list.length && scEntities_list[index].entity_address == currentScEntity, "Cannot create product because Entity does not exist!")**” χρησιμοποιείται για να ελέγξει αν η συγκεκριμένη διεύθυνση υπάρχει στη λίστα με τις οντότητες. Εάν δεν υπάρχει η συνάρτηση επιστρέφει μήνυμα “Cannot create product because Entity does not exist!".
- Η συνθήκη “**require(scEntities_list[index].entity_type == EntityType.Supplier, "Cannot create product because Entity is not a Supplier!")**” χρησιμοποιείται για να ελέγξει αν η οντότητα που θα δημιουργήσει το προϊόν είναι τύπου Supplier. Αν δεν είναι, επιστρέφεται μήνυμα “Cannot create product because Entity is not a Supplier!".
- Στη συνέχεια δημιουργείται ένα νέο προϊόν, τύπου Product, και αποθηκεύεται στη μεταβλητή newProduct. Τα δεδομένα που αποθηκεύονται σε κάθε καινούργιο προϊόν θα είναι: id, name, quantity, currentScEntity.
- Το νέο προϊόν προστίθεται στη λίστα “**products_list**” και ενημερώνεται ο δείκτης “**products_index**”.
- Στη συνέχεια, το νέο προϊόν προστίθεται στη λίστα προϊόντων του προμηθευτή, με την εντολή “**entities_products[currentScEntity].push(newProduct.id)”.**

Η συνάρτηση **remove_product** φαίνεται παρακάτω:

```
 function remove_product(uint256 id) public {
          require(products_index[id] != 0 || products_list[0].id == id,"Product
does not exist!");

        uint256 index = products_index[id];
        products_list[index] = products_list[products_list.length - 1];

        products_index[products_list[index].id] = index;
        delete products_index[id];

        products_list.pop();

        emit ProductRemoved(id);
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Η συνθήκη **“require(products_index[id] != 0 || products_list[0].id == id, "Product does not exist!")”** χρησιμοποιείται για να ελέγξει αν το προϊόν με το συγκεκριμένο id υπάρχει στη λίστα των προϊόντων. Αν το προϊόν δεν υπάρχει στη λίστα, η συνάρτηση τερματίζει με μήνυμα “Product does not exist!”.

- Αν το προϊόν υπάρχει στη λίστα, βρίσκει τον δείκτη του προϊόντος στη λίστα “products_list”, χρησιμοποιώντας το “products_index.
- Με την εντολή **“products_list[index] = products_list[products_list.length - 1]”** αντικαθιστάται το προϊόν που πρόκειται να αφαιρεθεί με το τελευταίο προϊόν της λίστας “**products_list”** και στη συνέχεια ενημερώνει το δείκτη του τελευταίου προϊόντος με την εντολή **“products_index[products_list[index].id] = index”.**
- Με την εντολή **“delete products_index[id]”** διαγράφεται ο δείκτης του προϊόντος που αφαιρέθηκε από τη λίστα, ενώ με την εντολή **“products_list.pop()”** αφαιρείται το τελευταίο στοιχείο από τη λίστα “products_list”.

Η συνάρτηση **get_product** φαίνεται παρακάτω:

```
   function get_product(uint256 id) public view returns (Product memory) {
        require(products_index[id] != 0 || products_list[0].id == id,"Product does
not exist!");
         if( users[msg.sender].role == Role.Supplier || users[msg.sender].role ==
Role.Logistic){
            for(uint256 i=0; i < entities_products[msg.sender].length ; i++ ){

if(products_list[products_index[entities_products[msg.sender][i]]].id == id){
                    return products_list[products_index[id]];
                }
            }

            require(false,"Product does not exist!");
        }

        return products_list[products_index[id]];
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Η συνθήκη **“require(products_index[id] != 0 || products_list[0].id == id, "Product does not exist!")”** χρησιμοποιείται για να ελέγξει αν το προϊόν με το συγκεκριμένο id υπάρχει στη λίστα των προϊόντων. Αν δεν υπάρχει, η συνάρτηση τερματίζει και επιστρέφει το μήνυμα “Product does not exist”.
- Αν ο user είναι Supplier ή Logistic, θα πάρει μόνο τα προϊόντα που του αντιστοιχούν.
- Αν το προϊόν υπάρχει, η συνάρτηση επιστρέφει τις πληροφορίες του προϊόντος από τη λίστα “products_list”, χρησιμοποιώντας το “products_index” για να βρει τη θέση του στη λίστα.

Η συνάρτηση **get_products** φαίνεται παρακάτω:

```
    function get_products() public view returns (Product[] memory) {
```

```
        if( users[msg.sender].role == Role.Supplier || users[msg.sender].role ==
Role.Logistic){
                                    Product[]   memory   _products_list   =   new
Product[](entities_products[msg.sender].length);

            for(uint256 i=0; i < entities_products[msg.sender].length ; i++ ){
                                                                _products_list[i]    =
products_list[products_index[entities_products[msg.sender][i]]];
            }

            return _products_list;
        }
        return products_list;
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Με την εντολή "**if (users[msg.sender].role == Role.Supplier || users[msg.sender].role == Role.Logistic)**" ελέγχεται αν ο ρόλος του χρήστη που καλεί τη συνάρτηση είναι Supplier ή Logistic.
- Αν ο χρήστης είναι Supplier ή Logistic, δημιουργείται ένας νέος πίνακας προϊόντων "**Product[]**" με όνομα **"_products_list".** Ο πίνακας αυτός έχει μήκος όσο και ο αριθμός των προϊόντων που ανήκουν στο συγκεκριμένο χρήστη. Αυτό το μήκος καθορίζεται από τη λίστα **"entities_products[msg.sender]".**
- Με την εντολή **"for (uint256 i = 0; i < entities_products[msg.sender].length; i++) { _products_list[i] = products_list[products_index[entities_products[msg.sender][i]]]"** γεμίζει ο πίνακας **"_products_list"** με τα προϊόντα που ανήκουν στο χρήστη χρησιμοποιώντας τους δείκτες των προϊόντων, ενώ κάνοντας χρήση της λίστας **"entities_products[msg.sender]"** βρίσκει τα id των προϊόντων της λίστας **"product_index",** ώστε να βρει τη θέση τους στη λίστα **"products_list".**
- Με την εντολή **"return _products_list;"** επιστρέφεται ο πίνακας **"_products_list"** που περιέχει τα προϊόντα του χρήστη (αν είναι Supplier ή Logistic).
- Αν ο χρήστης δεν είναι Supplier ή Logistic, η συνάρτηση επιστρέφει όλα τα προϊόντα που υπάρχουν στη λίστα **"products_list".**

Η συνάρτηση **get_product_entity** φαίνεται παρακάτω:

```
function get_product_entity(uint256 id) public view returns (ScEntity memory) {
       require(products_index[id] != 0 || products_list[0].id == id,"Product does
not exist!");

        if( users[msg.sender].role == Role.Supplier || users[msg.sender].role ==
Role.Logistic){
          for(uint256 i=0; i < entities_products[msg.sender].length ; i++ ){
```

```
if(products_list[products_index[entities_products[msg.sender][i]]].id == id){
                                                                    return
get_ScEntity(products_list[products_index[id]].currentScEntity);
                }
            }
            require(false,"Product does not exist!");
        }

        return get_ScEntity(products_list[products_index[id]].currentScEntity);
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Η συνθήκη **“require(products_index[id] != 0 || products_list[0].id == id, "Product does not exist!")”** ελέγχει αν το προϊόν με το συγκεκριμένο id υπάρχει στη λίστα των προϊόντων. Αν δεν υπάρχει, η συνάρτηση τερματίζει και εμφανίζει μήνυμα “Product does not exist!”.
- Αν το προϊόν υπάρχει, η εντολή **“address entityAddress = products_list[products_index[id]].currentScEntity”** ανακτά τη διεύθυνση της οντότητας που κατέχει το συγκεκριμένο προϊόν, και βρίσκει τη θέση του στη λίστα “products_list” χρησιμοποιώντας το “products_index”.
- Επιστρέφονται στοιχεία για το Entity του προϊόντος αν το προϊόν ανήκει στο user (στην περίπτωση που ο user είναι **Supplier** ή **Logistic**)
- Στη συνέχεια, με την εντολή **“return get_ScEntity(entityAddress)”** καλείται η συνάρτηση “get_ScEntity” με τη διεύθυνση της οντότητας αυτής, ώστε να ανακτήσει τις πληροφορίες της συγκεκριμένης οντότητας.

Οι συναρτήσεις που χρησιμοποιήθηκαν για την διαχείριση των οντοτήτων είναι οι εξής:

- create_ScEntity -> Δημιουργία οντότητας
- remove_ScEntity -> Αφαίρεση οντότητας
- get_ScEntity -> Ανάκτηση στοιχείων μιας οντότητας
- get_ScEntities -> Ανάκτηση στοιχείων όλων των οντοτήτων

Η συνάρτηση **create_ScEntity** φαίνεται παρακάτω:

```
function create_ScEntity(address entity_address, string memory name, EntityType
entity_type) public only_for_role(Role.Admin) {
        require(scEntities_index[entity_address] == 0 && (scEntities_list.length
==  0  ||  scEntities_list[scEntities_index[entity_address]].entity_address  !=
entity_address), "Entity already exists!");

        // Check if address belongs to an existing user
        if (users[entity_address].user_address == entity_address) {
            Role userRole = users[entity_address].role;
```

```
            // If user role is Supplier, entity type must be Supplier
            if (userRole == Role.Supplier) {
                  require(entity_type == EntityType.Supplier, "Entity type must be
Supplier for Supplier role");
            }
            // If user role is Logistic, entity type must be Warehouse Logistic
            else if (userRole == Role.Logistic) {
                 require(entity_type == EntityType.Warehouse_Logistic, "Entity type
must be Warehouse Logistic for Logistic role");
            }
            // If user role is Admin or Auditor, entity cannot be created
            else if (userRole == Role.Admin || userRole == Role.Auditor) {
                revert("Cannot create entity with Admin or Auditor role address");
            }
        }

        ScEntity memory newScEntity = ScEntity({
            entity_address: entity_address,
            name: name,
            entity_type: entity_type
        });

        scEntities_index[entity_address] = scEntities_list.length;
        scEntities_list.push(newScEntity);

              emit  ScEntityCreated(newScEntity.entity_address,  newScEntity.name,
newScEntity.entity_type);
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Ο modifier **“only_for_role(Role.Admin)”** εξασφαλίζει ότι μόνο εξουσιοδοτημένοι χρήστες (Admins) μπορούν να δημιουργήσουν μια νέα οντότητα.
- Η συνθήκη **“require(scEntities_index[entity_address] == 0 && (scEntities_list.length == 0 || scEntities_list[scEntities_index[entity_address]].entity_address != entity_address), "Entity already exists!")”** ελέγχει αν η οντότητα με τη συγκεκριμένη διεύθυνση υπάρχει στη λίστα με τις οντότητες. Εάν η οντότητα υπάρχει, επιστρέφει μήνυμα **"Entity already exists!".**
- Το Entity Address αν έχει χρησιμοποιηθεί για κάποιον χρήστη, τότε ο ρόλος αυτού του χρήστη πρέπει να ταιριάζει με το Entity Type. Δηλαδή, αν ο χρήστης είναι **Supplier** τότε και το **Entity Type** πρέπει να είναι **Supplier**, αν ο χρήστης είναι **Logistic** τότε το **Entity Type** πρέπει να

είναι **Warehouse Logistic** και αν είναι **Admin** ή **Auditor**, τότε δεν θα πρέπει να μπορεί να δημιουργηθεί αυτό το Entity.

- Δημιουργείται μια νέα οντότητα (τύπου ScEntity) (**entity_address, name, entity_type**) και προστίθεται στη λίστα **"scEntities_list"**, καθώς επίσης ενημερώνεται ο δείκτης **"scEntities_index"**.

Η συνάρτηση **remove_ScEntity** φαίνεται παρακάτω:

```
function remove_ScEntity(address entity_address) public only_for_role(Role.Admin)
{
        uint256 index = scEntities_index[entity_address];
                                require(index    <    scEntities_list.length    &&
scEntities_list[index].entity_address   ==   entity_address,   "Entity   does   not
exist!");

        if (index != scEntities_list.length - 1) {
            scEntities_list[index] = scEntities_list[scEntities_list.length - 1];
            scEntities_index[scEntities_list[index].entity_address] = index;
        }

        scEntities_list.pop();
        delete scEntities_index[entity_address];

        emit ScEntityRemoved(entity_address);
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Ο modifier **"only_for_role(Role.Admin)"** εξασφαλίζει ότι μόνο εξουσιοδοτημένοι χρήστες (Admins) έχουν τη δυνατότητα να αφαιρέσουν μια οντότητα.
- Η συνθήκη **"require(index < scEntities_list.length && scEntities_list[index].entity_address == entity_address, "Entity does not exist!")"** ελέγχει αν η οντότητα με τη συγκεκριμένη διεύθυνση υπάρχει στη λίστα με τις οντότητες. Εάν η οντότητα δεν υπάρχει, επιστρέφει μήνυμα **"Entity does not exist!".**
- Με την συνθήκη **if (index != scEntities_list.length - 1)** ελέγχεται αν η οντότητα που πρόκειται να αφαιρεθεί δεν είναι η τελευταία στη λίστα **"scEntities_list"**, ώστε να αντικατασταθεί με την τελευταία οντότητα της λίστας. Στη συνέχεια ενημερώνεται ο δείκτης **"scEntities_index"**.

Η συνάρτηση **get_ScEntity** φαίνεται παρακάτω:

```
    function get_ScEntity(address entity_address) public view returns (ScEntity
memory) {
        uint256 index = scEntities_index[entity_address];
                                require(index    <    scEntities_list.length    &&
scEntities_list[index].entity_address   ==   entity_address,   "Entity   does   not
exist!");
```

```
        return scEntities_list[index];
    }
```

Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Ο modifier **“only_for_role(Role.Admin)”** εξασφαλίζει ότι μόνο εξουσιοδοτημένοι χρήστες (Admins) έχουν τη δυνατότητα να δουν τις πληροφορίες μιας συγκεκριμένης οντότητας.
- Η συνθήκη **“require(index < scEntities_list.length && scEntities_list[index].entity_address == entity_address, "Entity does not exist!")”** ελέγχει αν η οντότητα με τη συγκεκριμένη διεύθυνση υπάρχει στη λίστα με τις οντότητες. Εάν η οντότητα δεν υπάρχει, επιστρέφει μήνυμα **"Entity does not exist!".**
- Αν η οντότητα υπάρχει, η συνάρτηση επιστρέφει τα στοιχεία της από τη λίστα **“scEntities_list”.**

Η συνάρτηση **get_ScEntities** φαίνεται παρακάτω:

```
    function get_ScEntities() public view returns (ScEntity[] memory) {
        return scEntities_list;
    }
```

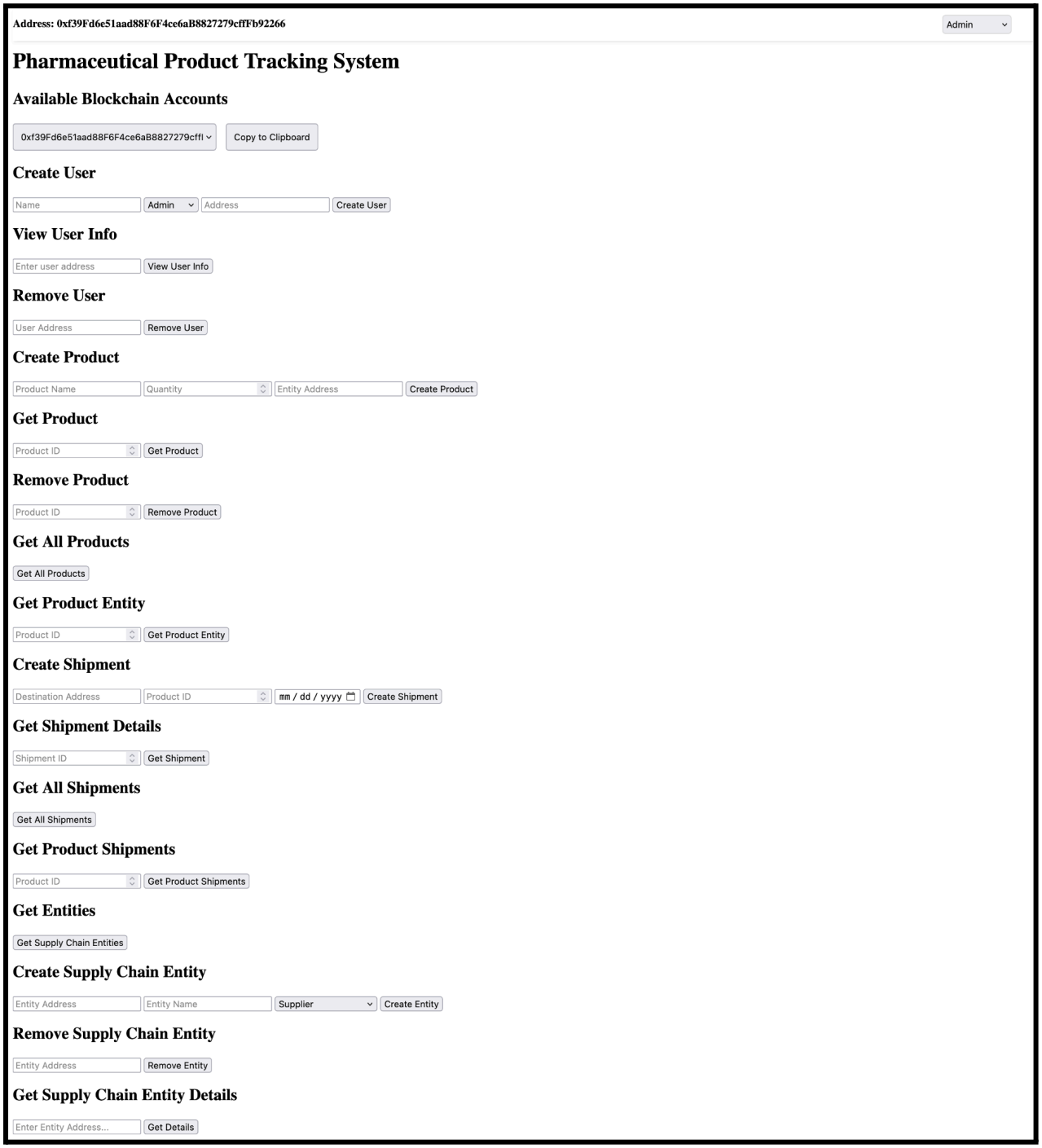
Με την παραπάνω συνάρτηση εξασφαλίζεται ότι:

- Ο modifier **“only_for_role(Role.Admin)”** εξασφαλίζει ότι μόνο εξουσιοδοτημένοι χρήστες (Admins) έχουν τη δυνατότητα να δουν λίστα με όλες τις οντότητες στην εφοδιαστική αλυσίδα.
- Η συνάρτηση επιστρέφει τη λίστα με όλες τις οντότητες στην εφοδιαστική αλυσίδα. Η λίστα αυτή αποθηκεύεται στο “**scEntities_list”.**

Ανάπτυξη UI και διασύνδεση με Έξυπνο Συμβόλαιο

Υλοποιήσαμε User Interface (UI) για το Έξυπνο Συμβόλαιο χρησιμοποιώντας το περιβάλλον ανάπτυξης **Hardhat** ως backend και απλής html-js για το frontend .

Παρακάτω φαίνεται η αρχική σελίδα του UI όπου οι χρήστες μπορούν να αλληλεπιδρούν με το Έξυπνο Συμβόλαιο και οι ενέργειες θα αλληλεπιδρούν απευθείας με το blockchain μέσω του αναπτυγμένου έξυπνου συμβολαίου.



1. Γενικές Παρατηρήσεις

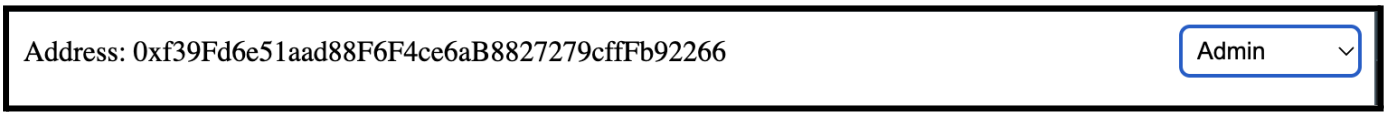
Πάνω σε αυτό το UI έχουν υλοποιηθεί όλες οι διαθέσιμες συναρτήσεις του συμβολαίου.

Για τον καλύτερο έλεγχο του συστήματος, προσθέσαμε μια dropdown λίστα, με τα accounts μέσα στο blockchain.

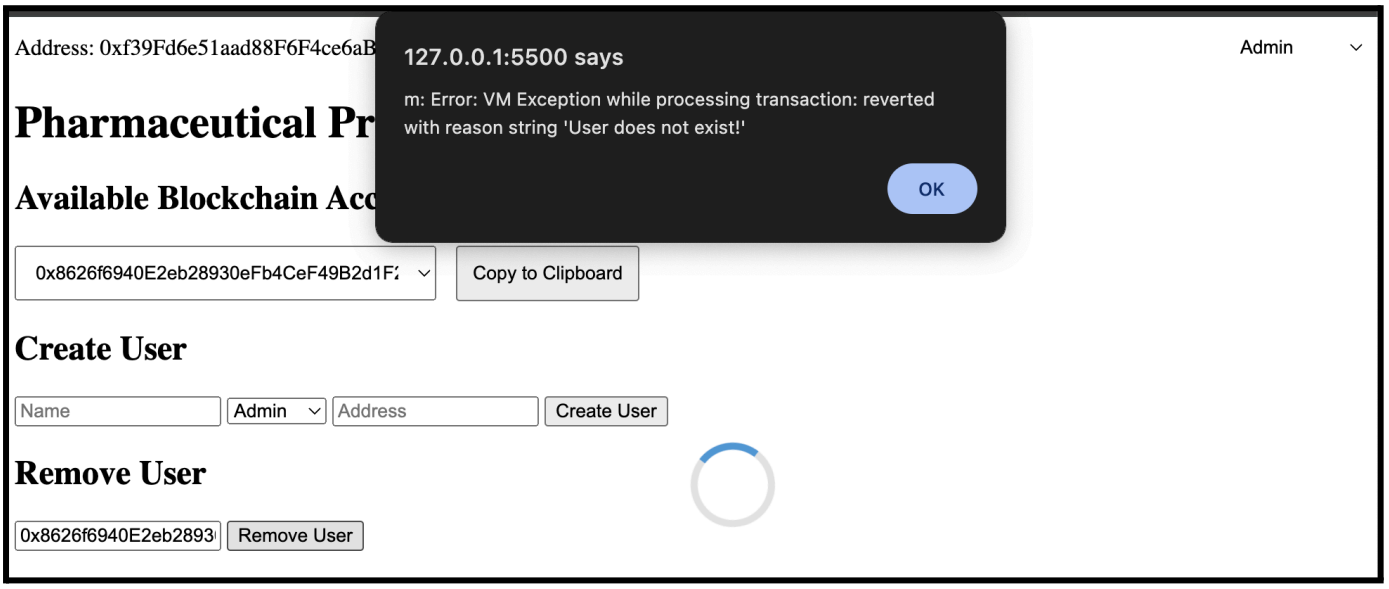
Επίσης, έχει προστεθεί μια dropdown λίστα, αποτελούμενη από τους default χρήστες που έχουμε υλοποιήσει, στα δεξιά του header της σελίδας και στην αριστερή πλευρά έχουμε το address του εκάστοτε επιλεγμένου χρήστη.

Όταν επιλέγουμε ένα χρήστη από την λίστα, ισοδυναμεί με το να αλλάξουμε το account που χρησιμοποιούμε μέσα στο blockchain.

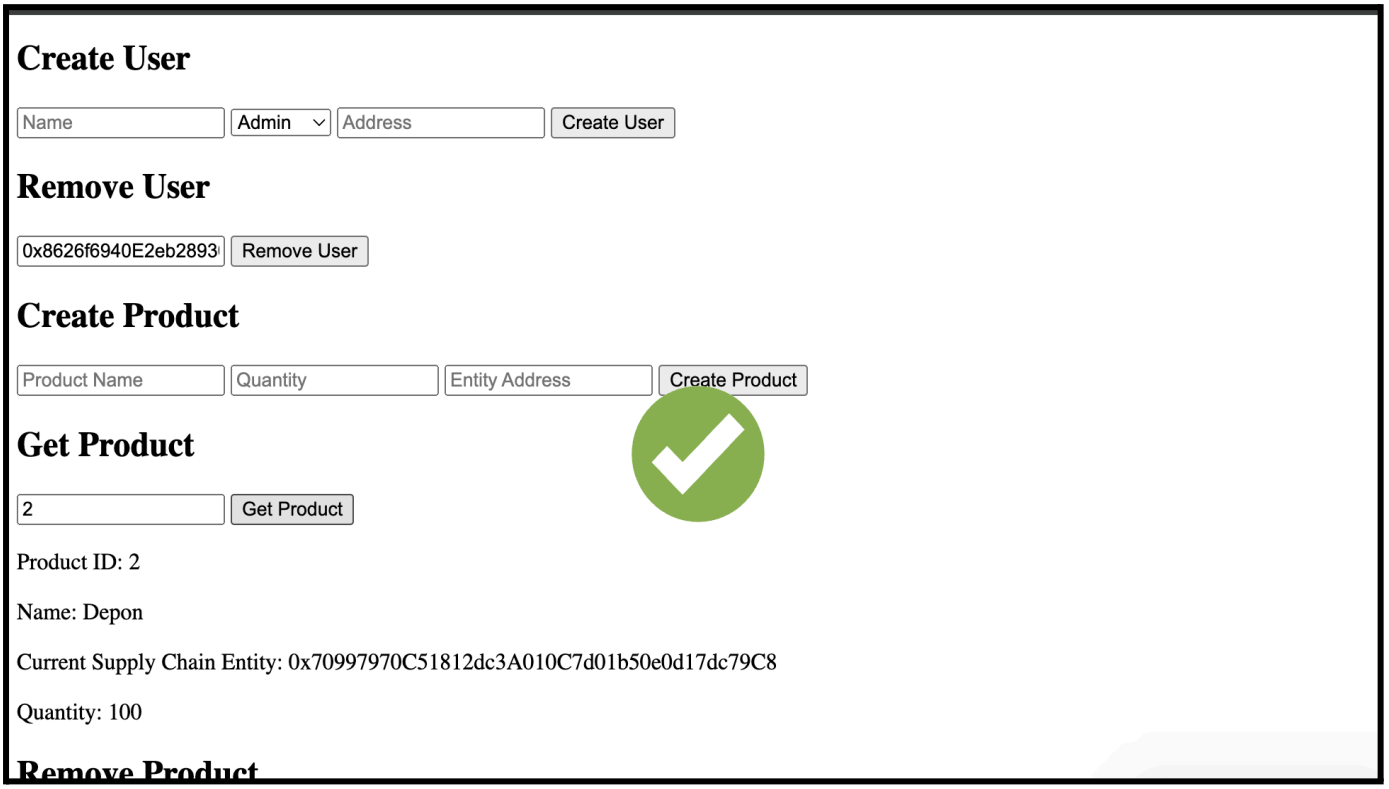
Η προσθήκη ή η αφαίρεση χρηστών από την λίστα **δεν** ανανεώνει το σχετικό menu. Οι χρήστες προστίθενται και αφαιρούνται κανονικά στο blockchain.



Στην περίπτωση κάποιου σφάλματος κατά την διάρκεια διαχείρισης του συστήματος, ένα alert μας ειδοποιεί για το σχετικό error.



Ενώ στην περίπτωση επιτυχίας περάτωσης μιας διαδικασίας, θα εμφανίζεται στο κέντρο της σελίδας για λίγα δευτερόλεπτα ένα checkmark.



Επίσης, τα αποτελέσματα από functions που επιστρέφουν πληροφορίες θα εμφανίζονται ακριβώς από κάτω από τα πεδία του συγκεκριμένου function μέσα στην σελίδα.

2. Αρχικά Objects μέσα στο blockchain

Πέρα από τον user admin που ορίζεται μέσα στον constructor του συμβολαίου, τα υπόλοιπα ζητούμενα objects δημιουργούνται μέσα από συναρτήσεις javascript.

Συγκεκριμένα, στο αρχείο **app.js** (frontend/js/app.js), έχουν δημιουργηθεί συναρτήσεις για την δημιουργία των αρχικών χρηστών, των προϊόντων και μερικών Entities.

Οι συναρτήσεις αυτές είναι οι:

- init_users
- init_products
- init_entities

Τα αντικείμενα αυτά έχουν υλοποιηθεί με βάση ορισμένα jsons αρχεία που υπάρχουν μέσα στον φάκελο **objects** (frontend/objects).

Συγκεκριμένα τα αρχεία αυτά είναι τα:

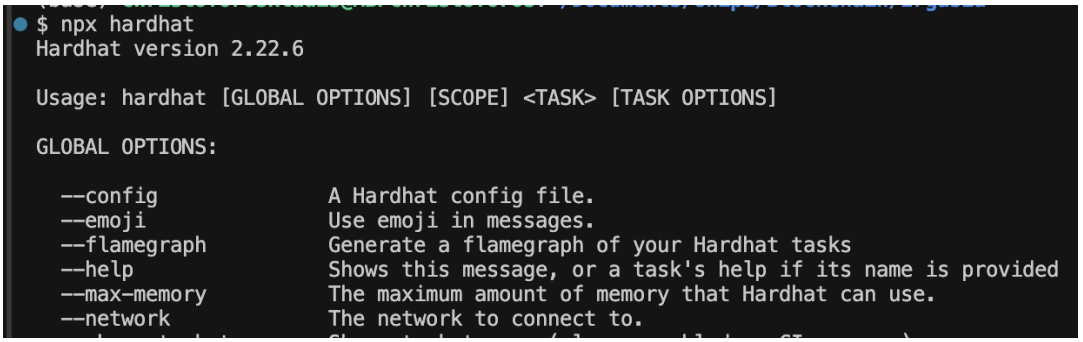
- users.json για τους users
- products.json για τα products και
- entities.json για τα entities

3. Οδηγίες εκκίνησης της εφαρμογής

Για την εκκίνηση της εφαρμογής ακολουθήστε τα παρακάτω βήματα.

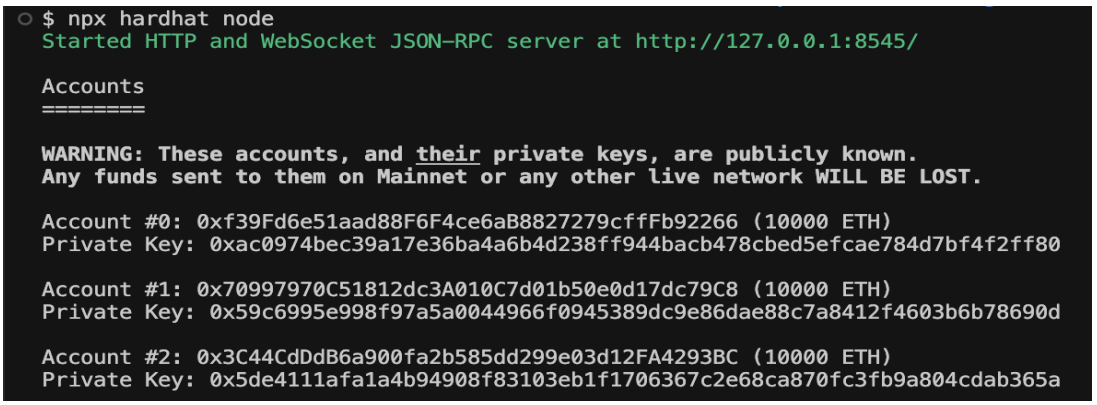
1. Κατεβάστε το project
2. Στο root του project, μέσα από terminal, τρέχετε την εξής εντολή:
   npm install

   Αν όλα πάνε καλά, τότε αν τρέξετε την εντολή:
   npx hardhat
   θα σας εμφανιστεί μια λίστα από εντολές για το hardhat.



3. Στο ίδιο terminal τρέχετε την εντολή:
   npx hardhat compile

   Με αυτή την εντολή μετατρέπεται ο κώδικας του αρχείου Pharmacy.sol (contracts/Pharmacy.sol) σε json αρχείο, από το οποίο μπορούμε να πάρουμε το abi.
4. Στο ίδιο terminal τρέχετε την εντολή
   npx hardhat node
   Αυτή η εντολή ξεκινά το blockchain και δημιουργεί κάποια accounts addresses.



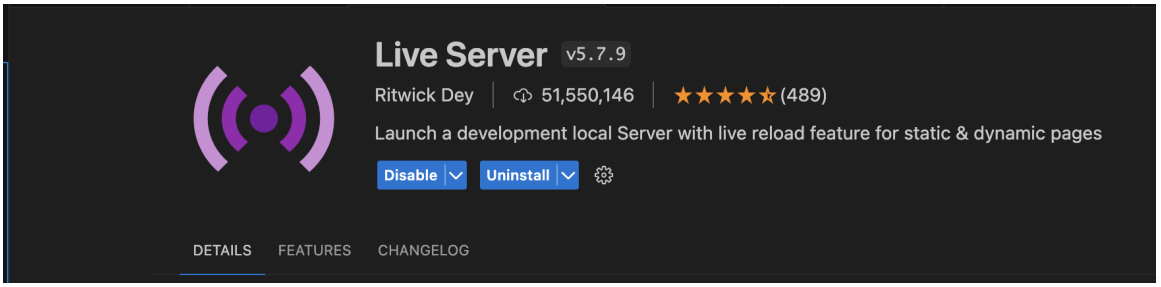
5. Σε άλλο terminal, τρέχετε την εντολή
   npx hardhat run scripts/deploy.js --network localhost
   η οποία κάνει deploy το contract στο blockchain.

```
$ npx hardhat run scripts/deploy.js --network localhost
Deploying contracts with the account: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
Pharmacy contract deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

6. Ελέγξτε αν το **Pharmacy contract deployed** από την προηγούμενη εντολή είναι το 0x5FbDB2315678afecb367f032d93F642f64180aa3. Αν είναι διαφορετικό, τότε προσθέστε την νέα address στο αρχείο **app.js** (frontend/js/app.js) , στην γραμμή 6, στην μεταβλητή **contractAddress.**

```
1  const web3 = new Web3('http://127.0.0.1:8545/'); // Change to your node URL
2  let accounts = [];
3  let users = {};
4  let suppliers = [];
5  let contractABI;
6  let contractAddress = "0x5FbDB2315678afecb367f032d93F642f64180aa3";
7  let contract;
```

7. Τέλος, ανοίξτε σε μια σελίδα το αρχείο index.html μέσω ενός http-server. Για την εργασία, χρησιμοποιούμε το Live Server extension της εφαρμογής Visual Code.



Στιγμιότυπα Λειτουργίας Έξυπνου Συμβολαίου

1. Δημιουργία και έλεγχος στοιχείων χρήστη

Από τα available accounts κάνουμε copy ένα account και με αυτό συμπληρώνουμε την φόρμα Create User. (Όπως αναφέρθηκε και παραπάνω, μόνο ο admin μπορεί να δημιουργήσει χρήστες).

Σε περίπτωση επιτυχίας θα εμφανιστεί το checkmark στην μέση της οθόνης.

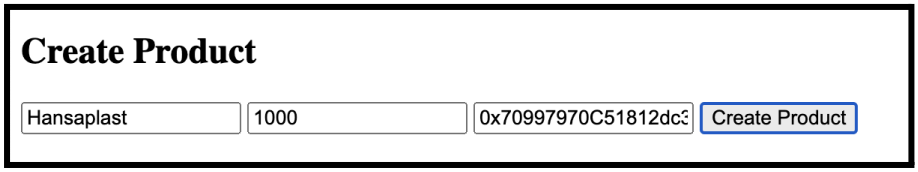
Για τον έλεγχο των στοιχείων του χρήστη, συμπληρώνουμε στην φόρμα **View User Info** την διεύθυνση account που θέλουμε να ελέγξουμε.

Για την διαγραφή του user συμπληρώνουμε την φόρμα **Remove User** με το account του user που θέλουμε να σβήσουμε. Ανάλογα με το αν είναι επιτυχής ή όχι η διαδικασία, θα πάρουμε κάποιο alert με ένα μήνυμα σφάλματος ή ένα checkmark. (Όπως αναφέρθηκε και παραπάνω, μόνο ο admin μπορεί να διαγράψει χρήστες).

2. Δημιουργία και έλεγχος στοιχείων προϊόντων

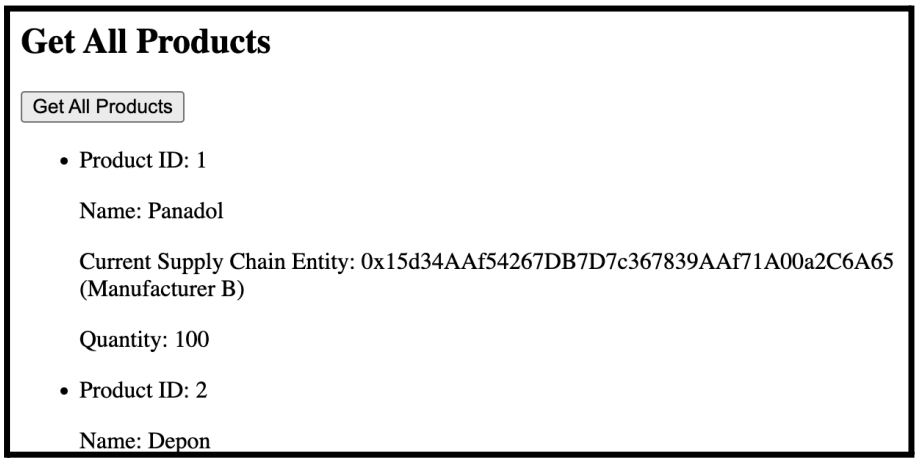
Για την δημιουργία νέων προϊόντων συμπληρώνουμε την φόρμα **Create Product.**

Στο πεδίο **Entity Address** πρέπει να συμπληρώσουμε μια valid διεύθυνση ενός **Supply Chain Entity** (ScEntity.)



Αν θέλουμε να ελέγξουμε ότι φτιάχτηκε το προϊόν, μπορούμε να επιλέξουμε την συνάρτηση **Get All Products,** η οποία θα μας φέρει όλα τα προϊόντα του συστήματος.

Αυτό γίνεται στην περίπτωση που ο χρήστης είναι **Admin** ή **Ελεγκτής.** Αν ο χρήστης είναι **Supplier** τότε μπορεί να δει μόνο τα προϊόντα που έχουν ξεκινήσει από τον ίδιο, ενώ αν είναι **Logistic** μπορεί να δει τα προϊόντα που έχουν περάσει στον ίδιο μέσω κάποιου shipment.

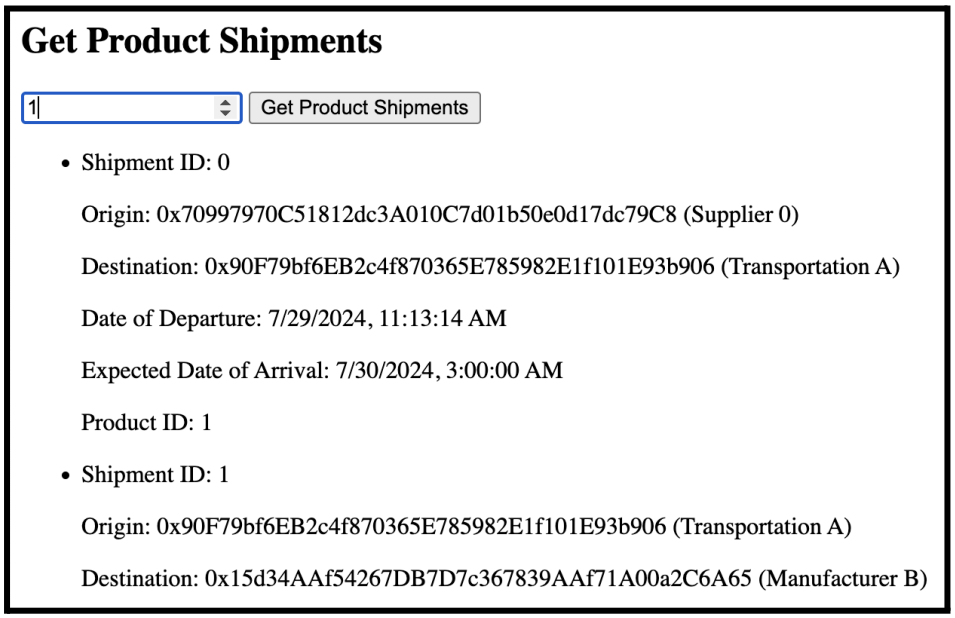


Αν θέλουμε να εξετάσουμε τις πληροφορίες ενός προϊόντος, μπορούμε να χρησιμοποιήσουμε την συνάρτηση **Get Product.** Ισχύουν οι ίδιοι περιορισμοί με την προηγούμενη συνάρτηση.

Αν θέλουμε να ελέγξουμε συγκεκριμένα το Current Supply Chain Entity ενός προϊόντος, χρησιμοποιούμε την συνάρτηση **Get Product Entity.** Ισχύουν οι ίδιοι περιορισμοί με την προηγούμενες συναρτήσεις.

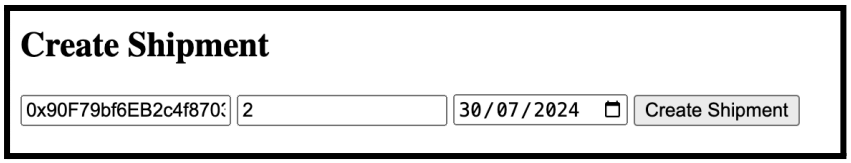
Αν θέλουμε να διαγράψουμε κάποιο προϊόν, μπορούμε να χρησιμοποιήσουμε την συνάρτηση **Remove Product**. Αν γίνει διαγραφή ενός προϊόντος, τα shipments αυτού θα παραμείνουν.

Αν, τέλος, θέλουμε να πάρουμε τα shipments ενός προϊόντος, χρησιμοποιούμε την συνάρτηση **Get Product Shipments**. Η συγκεκριμένη συνάρτηση επιστρέφει όλα τα shipments του προϊόντος για τους χρήστες **Admin** και **Auditor.** Για τους **Supplier** και **Logistic**, επιστρέφει τα shipments μέχρι το entity **Manufacturer** και **Warehouse Logistic** αντίστοιχα.



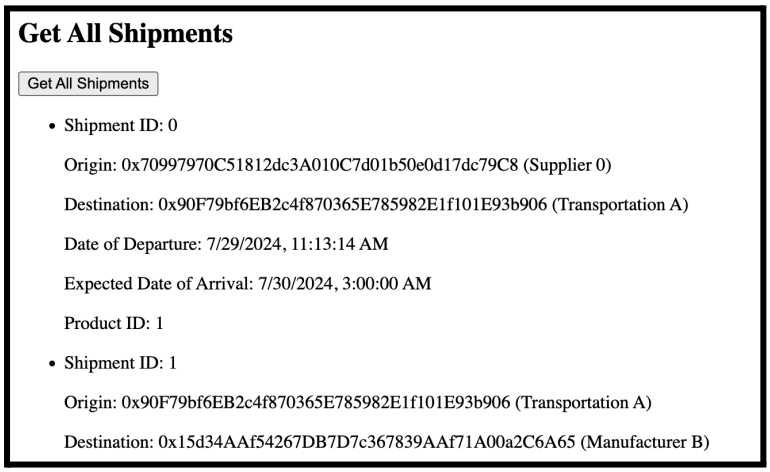
3. Δημιουργία και έλεγχος στοιχείων μεταφορών (shipments)

Για την δημιουργία μιας μεταφοράς (στο εξής shipment), χρησιμοποιούμε την συνάρτηση **Create Shipment.**

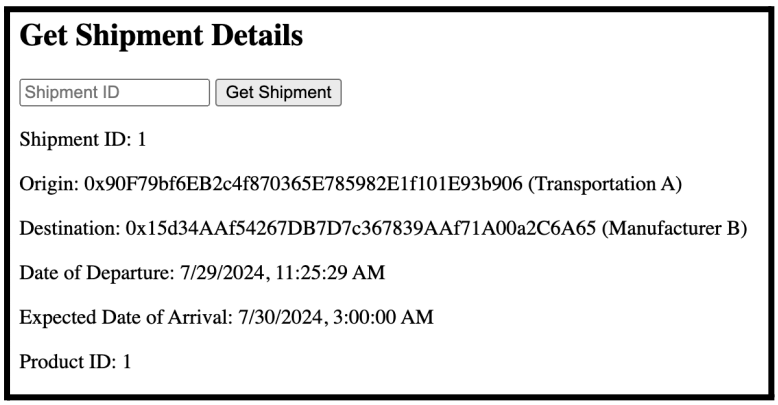


Για την επιτυχή εκτέλεση της συνάρτησης είναι αναγκαίο να βάλουμε σωστά τα στοιχεία στην φόρμα. Δηλαδή, να υπάρχει το προϊόν, η εκτιμώμενη ημερομηνία άφιξης να είναι ύστερα από την ημερομηνία αναχώρησης και το destination Entity πρέπει να είναι **μια οντότητα ακριβώς μετά** το current Entity του προϊόντος. Δηλαδή, αν το current Entity είναι ένας **Supplier**, τότε το destination πρέπει οπωσδήποτε να είναι ένα **Transportation**.

Αν θέλουμε να ελέγξουμε τα shipments που δημιουργήθηκαν, μπορούμε να εκτελέσουμε την συνάρτηση **Get All Shipments**. Η συγκεκριμένη συνάρτηση φέρνει τα ίδια αποτελέσματα σε όλους τους χρήστες.



Αν θέλουμε να ελέγξουμε ένα συγκεκριμένο shipment, μπορούμε να χρησιμοποιήσουμε την συνάρτηση **Get Shipment Details**. Η συγκεκριμένη συνάρτηση φέρνει τα ίδια αποτελέσματα σε όλους τους χρήστες.



4. Δημιουργία και έλεγχος συμμετεχόντων οντοτήτων (ScEntities ή entities)

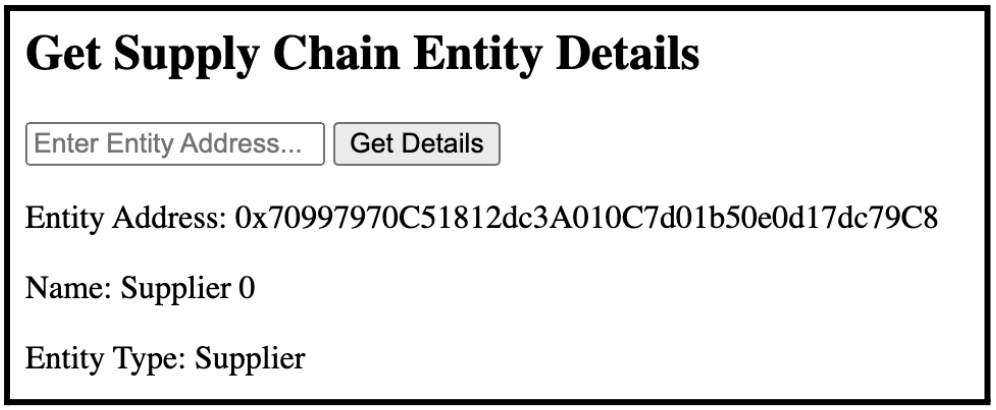
Αν θέλουμε να δημιουργήσουμε μια καινούργια συμμετέχουσα οντότητα μέσα στο Supply Chain (στο εξής entity) χρησιμοποιούμε την φόρμα **Create Supply Chain Entity**.

Προϋποθέσεις για να εκτελεστεί σωστά η συνάρτηση είναι ο τωρινός χρήστης να είναι admin και το Entity Address να μην έχει χρησιμοποιηθεί για την δημιουργία άλλου entity. Επίσης, το Entity Address αν έχει χρησιμοποιηθεί για κάποιον χρήστη, τότε ο ρόλος αυτού του χρήστη πρέπει να ταιριάζει με τον Entity Type. Δηλαδή, αν ο χρήστης είναι **Supplier** τότε και το **Entity Type** πρέπει να είναι **Supplier**, αν ο χρήστης είναι **Logistic** τότε το **Entity Type** πρέπει να είναι **Warehouse Logistic** και αν είναι **Admin** ή **Auditor**, τότε δεν θα πρέπει να μπορεί να δημιουργηθεί αυτό το Entity.

Αν θέλουμε να ελέγξουμε τα entities που δημιουργήθηκαν, μπορούμε να εκτελέσουμε την συνάρτηση **Get Entities.** Η συγκεκριμένη συνάρτηση φέρνει τα ίδια αποτελέσματα σε όλους τους χρήστες. Τα Supply Chain Entities εμφανίζονται σε τυχαία σειρά.

Η διαδρομή της Εφοδιαστικής Αλυσίδας (δηλαδή, τα Supply Chain Entities) είναι η εξής:
Supplier -> Transportation -> Manufacturer -> Warehouse_Logistic -> Distributor -> Pharmacy

Αν θέλουμε να ελέγξουμε συγκεκριμένο Entity, χρησιμοποιούμε την φόρμα **Get Supply Chain Entity Details.**



Τέλος, αν θέλουμε να διαγράψουμε κάποιο Entity, τότε χρησιμοποιούμε την φόρμα **Remove Supply Chain Entity.** Αυτή την συνάρτηση μπορεί να την τρέξει μόνο ο Admin.

Συμβολή της Τεχνολογίας Blockchain στην ιχνηλασιμότητα φαρμακευτικών προϊόντων

Η τεχνολογία blockchain προσφέρει μια ισχυρή λύση για την παρακολούθηση φαρμακευτικών προϊόντων, ενισχύοντας την ιχνηλασιμότητα, τη διαφάνεια και την ασφάλεια. Παρακάτω συνοψίζεται η εφαρμογή ενός συστήματος παρακολούθησης φαρμακευτικών προϊόντων με τη χρήση blockchain που υλοποιήσαμε.

Το blockchain είναι ένα αποκεντρωμένο σύστημα τήρησης αρχείων που καταγράφει τις συναλλαγές αμετάβλητα. Τα χαρακτηριστικά του blockchain το καθιστούν ιδανικό για τη διαχείριση της φαρμακευτικής αλυσίδας εφοδιασμού, εξασφαλίζοντας την εμπιστοσύνη και την επαλήθευση.

Η εφαρμογή μας περιλαμβάνει τη δημιουργία ενός έξυπνου συμβολαίου Solidity για τη διαχείριση της φαρμακευτικής αλυσίδας εφοδιασμού με τα ακόλουθα βασικά χαρακτηριστικά:

- Οντότητες: Supplier, Transporter, Manufacturer, Logistic Warehouse, Distributor, Pharmacy.
- Χρήστες (με συγκεκριμένα δικαιώματα πρόσβασης): Administrator, Supplier, Logistic Employee, Controller.
- Διαχείριση προϊόντων: Τα προϊόντα παρακολουθούνται από τη δημιουργία έως την παράδοση με μοναδικά αναγνωριστικά, ποσότητες και τοποθεσίες.
- Διαχείριση αποστολών: Οι αποστολές καταγράφουν τις μεταφορές προϊόντων μεταξύ οντοτήτων, συμπεριλαμβανομένων των δεδομένων αναχώρησης και άφιξης.
- Ιχνηλασιμότητα και ακεραιότητα: Οι αμετάβλητες εγγραφές blockchain διασφαλίζουν ότι τα δεδομένα δεν μπορούν να αλλοιωθούν. Τα ιστορικά δεδομένα μεταφοράς είναι προσβάσιμα για έλεγχο και επαλήθευση.

Τα οφέλη της τεχνολογίας blockchain από την χρήση στην φαρμακευτική αλυσίδα εφοδιασμού είναι:

- Ενισχυμένη ιχνηλασιμότητα: Διαφανής διαδρομή του προϊόντος από την κατασκευή έως την πώληση. Το επαληθεύσιμο ιστορικό του προϊόντος διασφαλίζει τη γνησιότητα.
- Βελτιωμένη ασφάλεια: Οι αμετάβλητες εγγραφές αποτρέπουν την απάτη και την πλαστογράφηση. Η ασφαλής αποθήκευση δεδομένων διασφαλίζει την ακεραιότητα του προϊόντος.
- Αυξημένη διαφάνεια: Το κοινόχρηστο σύστημα τήρησης αρχείων παρέχει διαφάνεια σε όλους τους συμμετέχοντες στην αλυσίδα εφοδιασμού. Ο άμεσος εντοπισμός και η επίλυση προβλημάτων βελτιώνουν την αποδοτικότητα.
- Αποτελεσματικότητα στους ελέγχους και τη συμμόρφωση: Τα απαραβίαστα αρχεία συναλλαγών απλοποιούν την κανονιστική συμμόρφωση. Οι ελεγκτές μπορούν να έχουν πρόσβαση στο αμετάβλητο ιστορικό διακίνησης προϊόντων.

Η τεχνολογία blockchain ενισχύει σημαντικά την ιχνηλασιμότητα, τη διαφάνεια και την ασφάλεια της φαρμακευτικής αλυσίδας εφοδιασμού. Το εφαρμοζόμενο σύστημα διασφαλίζει τη γνησιότητα και την ακεραιότητα των φαρμάκων από την παρασκευή έως την παράδοση, παρέχοντας μια αξιόπιστη λύση κατά της παραποίησης και διασφαλίζοντας την ασφάλεια των φαρμάκων.

Βιβλιογραφικές Πηγές

1. Architecture of Decentralized Applications (dApps): https://medium.com/@blocktorch/architecture-of-decentralized-applications-dapps-d583db198a6f
2. Hardhat development environment for Ethereum software: https://hardhat.org/tutorial
3. Remix Integrated Development Environment (IDE): https://remix-ide.readthedocs.io/en/latest/
4. Solidity Programming Language: https://docs.soliditylang.org/en/v0.8.26/