СОФИЙСКИ УНИВЕРСИТЕТ
„СВ. КЛИМЕНТ ОХРИДСКИ“
ФАКУЛТЕТ ПО
МАТЕМАТИКА И ИНФОРМАТИКА

SOFIA UNIVERSITY
St. KLIMENT OHRIDSKI
FACULTY OF
MATHEMATICS AND INFORMATICS

Курсов проект

по дисциплина
„Основи на разработването на блокчейн приложения“

Тема „PatientX“

Студенти:
Мартин Раванов, 62379
Георги Събев, 62380
Искрен Банински, 62456
Кристиан Какалов, 62459
Иван Лаков, 62505

гр. София
13.06.2022 г.

Съдържание

1. Описание на проблема

1.1. Въведение в темата

Във времена, в които медицината постоянно е поставена пред всевъзможни предизвикателства, възможността за настъпване на дезинформация, породена от грешно разтълкувани или непълно споделени с лекаря симптоми, е все по-голяма.

Винаги медицинските сведения и документите, които се пазят за всеки пациент, са били чувствителна тема, за чието съхранение и обработване са в действие все повече закони и нормативни уредби. Въпреки това повечето хора до ден днешен не знаят какви са точните процедури за това, преносът на информация между институциите е изключително тромав и в много случаи събраната информация със сведения за пациентите (какви заболявания ги съпровождат, какви лекарства са приемали и т.н.) от даден доктор или институция изобщо не достига до следващия. Съхранението на същите тези данни също е под въпрос, тъй като постоянно документи, написани на хартия изчезват или не се съхраняват според определените норми.

1.2. Задача на проекта

Основната идея на нашия проект е да дадем на пациентите сигурност, че техните медицински данни ще се съхраняват и ще може да се използват от всички лекари, на които те позволят.

Използвайки системата PatientX, пациентите също така имат възможността да получават възнаграждения под формата на crypto, в случай че дадат своето съгласие техните медицински данни и изследвания да бъдат използвани за анализ и създаване на прогнози на Медицинските институти, работещи с нашата ситемата. По този начин потребителите ще получат както сигурност и информация, ако имат лоши показатели, за това какви мерки трябва да вземат, за да не отключат дадено заболяване, така и тяхната информация също ще е от голяма полза за бъдещото развитие на медицината.

2. Спецификация на смарт контрактите

Системата PatientX е изградена върху два Smart contract-а: PatientX и InstituteInteraction. И двата contract-а имат library MyLibrary, съдържащо структури, нужни за реализацията на проекта.

```
library MyLibrary {
    struct Doctor {
        address doctorAddress;
        string name;
    }

    struct Patient{
        address patientAddress;
        string name;
    }

    struct MedicalInstitute{
        address instituteAddress;
        string name;
    }

    struct MedInstituteData{
        uint256 numOfMedExams;
        uint256 numOfPatientsSharingData;
    }

    struct MedicalExamination {
        address doctorAddress;
        address patientAddress;
        string data;
    }
}
```

Структурите *Doctor*, *Patient* и *MedicalInstitute* са подобни по съдържание: състоят се от адрес на потребителя и стринг с име. Структурата *MedicalExamination* е с цел по-лесно пазене на информацията на изследванията на пациента като запазва адреса на лекаря, направил изследването, адреса на пациента и стринг с детайлна информация за изследването. *MedInstituteData* е структура с цел запазване на информация за даден институт - колко пациента са дали съгласие техните изследвания да бъдат споделени с него и брой изследвания, до които има достъп съответният институт.

2.1. PatientX

В Smart contract-а PatientX се реализира цялата функционалност на проекта. В реализацията му са използвани следните полета:

```
address public owner;

address[] private patientAddresses;
mapping(address => MyLibrary.Patient) private allPatientsByAddress;
mapping(address => MyLibrary.MedicalExamination[]) private allMedicalExaminationsByPatientAddress;
mapping(address => mapping(address => MyLibrary.Doctor)) private allDoctorsByPatientAddress;
mapping(address => mapping(address => MyLibrary.MedicalInstitute)) private allMedicalInstitutesByPatientAddress;
mapping(address => address[]) private allMedicalInstituteAddressesByPatientAddress;

mapping(address => MyLibrary.Doctor) private allDoctorsByAddress;

mapping(address => MyLibrary.MedicalInstitute) private allMedicalInstitutesByAddress;
mapping(address => MyLibrary.MedInstituteData) public allInstituteDataByAddress;

mapping(address => MyLibrary.Doctor[]) private doctorsOfPatient;
mapping(address => MyLibrary.Patient[]) private patientsOfDoctor;
mapping(address => MyLibrary.MedicalInstitute[]) private medicalInstitutesOfPatient;

mapping(address => bool) public suchPatientExists;
mapping(address => bool) public suchDoctorExists;
mapping(address => bool) public suchMedicalInstituteExists;
```

При deploy на contract-а в тестовата среда, в owner се запазва информация за това кой е собственикът на контракта. За реализиране на функционалностите, свързани с

пациентите, са използвани масив с адресите на всички пациенти (*allPatientsByAddress*), map-ове с адресите на пациентите, връщащи структурата *Patient* (*allPatientsByAddress*), масив от всички негови изследвания (*allMedicalExaminationsByPatientAddress*), map с адресите на лекаря, връщащи структурата *Doctor* (*allDoctorsByPatientAddress*), map с адресите на институтите, връщащи структурата *MedicalInstitute* (*allMedicalInstitutesByPatientAddress*) и масив от адреси на всички институти, на които пациентът е разрешил да имат достъп до изследванията му. Информацията за лекарите и институтите пазим с map от адреси, връщащи структурата *Doctor* (*allDoctorsByAddress*), структурата *MedicalInstitute* (*allMedicalInstitutesByAddress*), структурата *MedInstituteData* (*allInstituteDataByAddress*). За достъпване на всички лекари и институти, на които пациентът е позволил да обработват неговите данни, имаме map-овете *doctorsOfPatient* и *medicalInstitutesOfPatient*, а за да може лекарят да види всичките си пациенти, имаме map-а *patientsOfDoctor*. За проверка дали даден потребител вече се е регистрирал използваме map-овете *suchPatientExists*, *suchDoctorExists*, *suchMedicalInstituteExists*, връщащи bool. Също така са налични 2 event logs (извикван при *fallback* и *receive* функциите) и *OwnerSet* (при промяна на *owner*).

В конструктора задаваме кой е собственикът на контракта и се извиква събитието *OwnerSet*. В *receive* и *fallback* функциите извикваме събитието *Log* и при успешно изпратено крипто, го приемаме като donation.

Функциите *enlistPatient*, *enlistDoctor*, *enlistMedicalInstitute* са сходни по предназначение – добавят и запазват информация на потребителя в зависимост от неговата роля.

Функцията *addNewMedicalExaminationOfPatientByDoctor* служи за добавяне на изследване на пациент като актуализира данните на институтите, които имат достъп до изследванията на пациента с този адрес (*updateInstitutesDataNewMedRecord*).

Функцията *getAllPatientsOfDoctor* служи за получаване на масив от всички пациенти, които посещават този лекар.

Функцията *addDoctorToPatient* служи за добавяне на нов лекар към списъка с лекари на даден пациент.

Ако пациент прецени, че няма да посещава вече даден лекар, може да го премахне от списъка си с лекари чрез функцията *removeDoctorFromPatient*.

За да получи масив от всички лекари, които посещава, пациент може да използва функцията *getAllDoctorsOfPatient*.

Фунцията *addInstituteToPatient* служи за добавяне на нов институт към списъка с доверени институти на даден пациент.

Ако пациент реши, че вече не иска даден институт да има достъп до данните му, може да го премахне от списъка чрез функцията *removeInstituteFromPatient*, при което актуализира данните на институтите, които имат достъп до изследванията на пациента с този адрес (*updateInstitutesDataPatientRemoved*).

За да получи масив от всички институти, които имат достъп до данните му, може да използва функцията *getAllInstitutesOfPatient*.

Фунцията *getMedicalExaminationsShareable* връща масив от всички изследвания на пациенти, дали достъп на този институт до данните им. При извикването ѝ се създава масив от адреси на пациентите, за да могат да получат крипто. Също така собственикът получава крипто при извикване на тази функция.

2.2. InstitutesInteraction

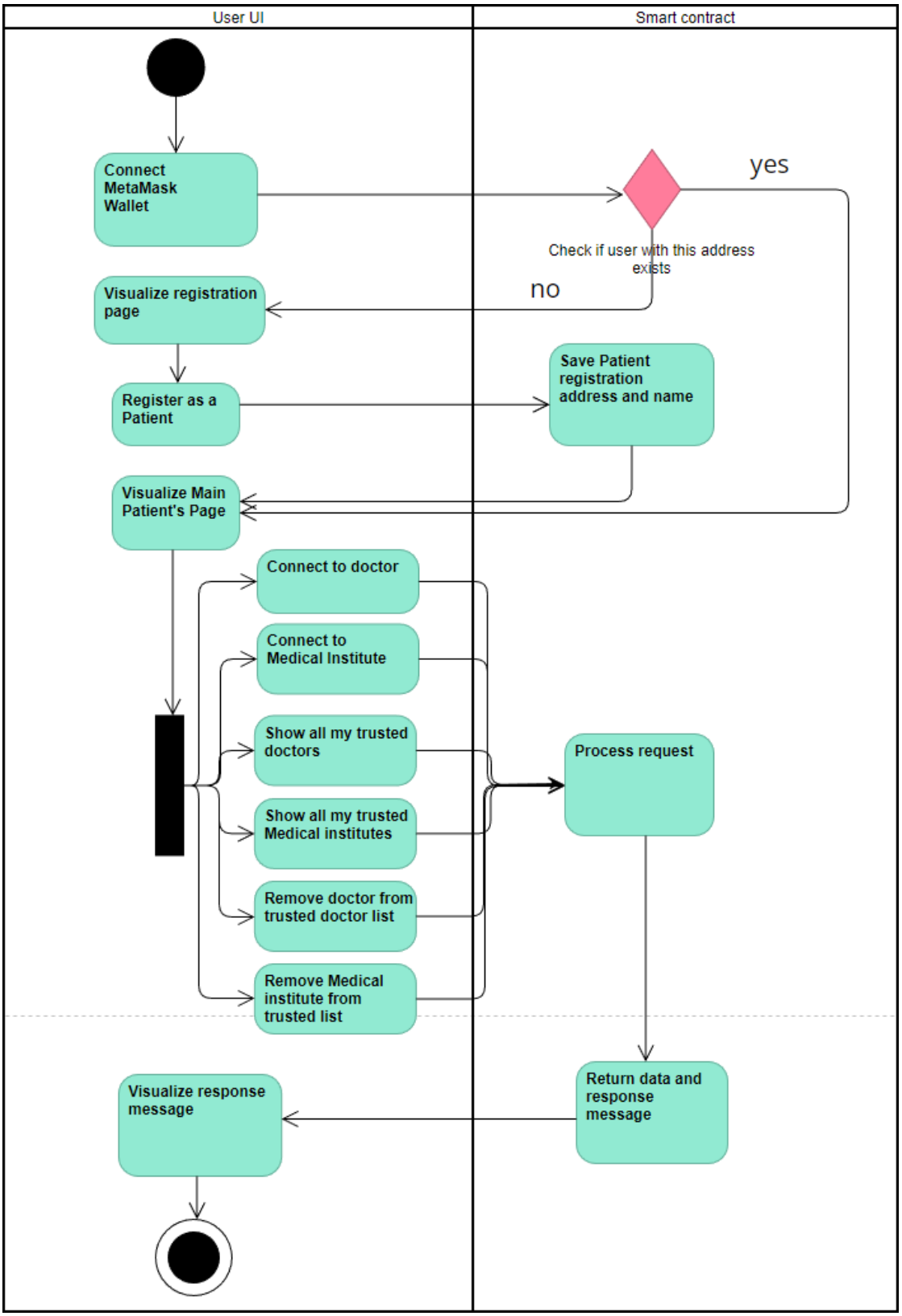
Втория contract изпълнява функционалностите, до които институтите имат достъп. За да може да ограничим достъпа до методите на контракта PatientX, сме създали интерфейса *IPatientX* с функциите *getMedicalExaminationsShareable* и *enlistMedicalInstitute*. Целта му е да демонстрира как два Smart contract-а могат да комуникират помежду си.

```
interface IPatientX {
    function getMedicalExaminationsShareable() payable external returns (MyLibrary.MedicalExamination[] memory);

    function enlistMedicalInstitute(string calldata _name) external;
}

contract InstitutesInteraction {
    function enlistMedicalInstitute(IPatientX _address, string calldata _name) external {
        _address.enlistMedicalInstitute(_name);
    }

    function getMedicalExaminationsShareable(address _address) external payable returns(MyLibrary.MedicalExamination[] memory)
        IPatientX patientXContract = IPatientX(_address);
        return patientXContract.getMedicalExaminationsShareable{value: msg.value}();
    }
}
```

3. Процесни диаграми

3.1. Patient workflow диаграма

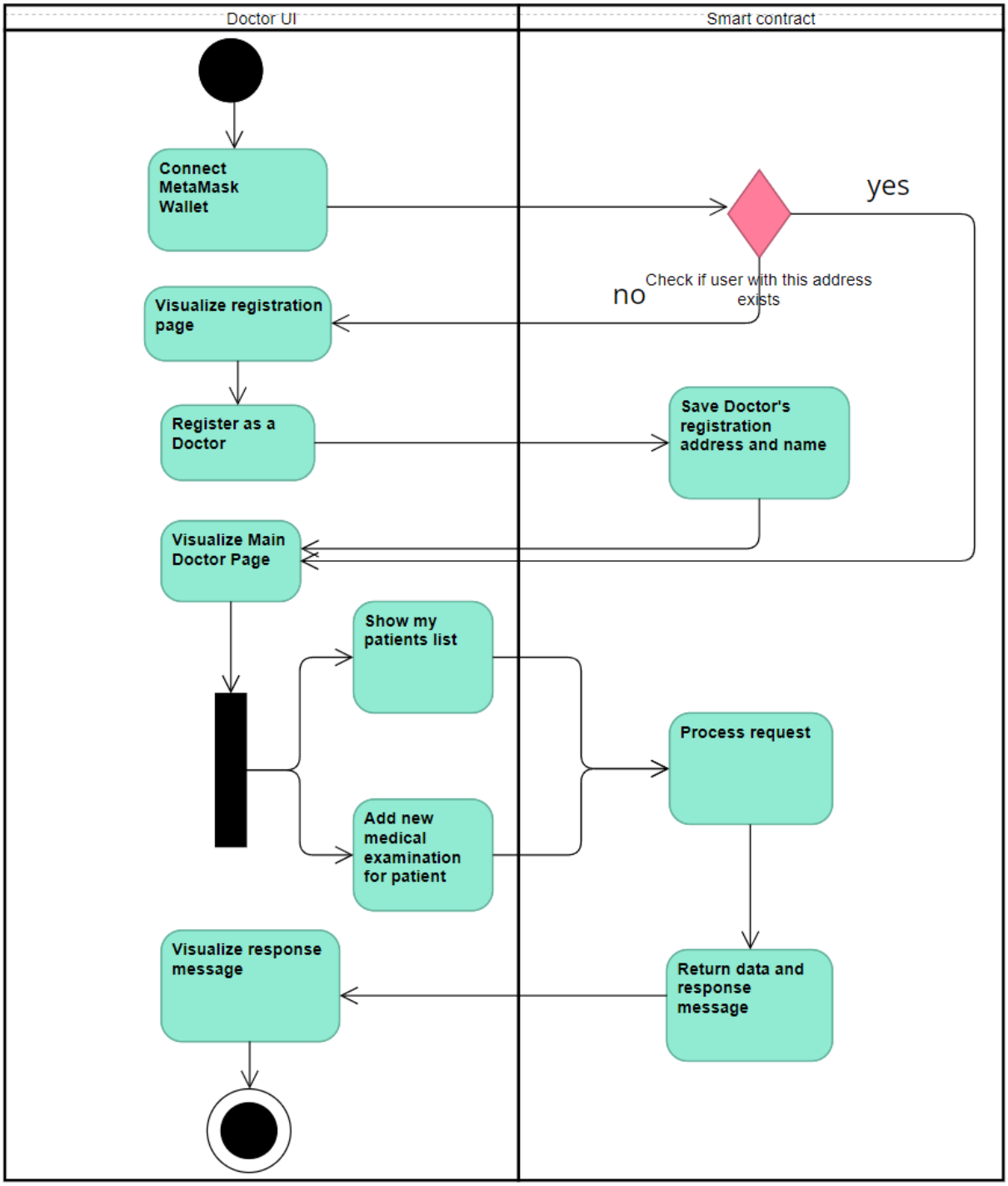
Диаграмата по-долу показва какви са дейностите, които един потребител регистриран като пациент, може да извършва. Първото и задължително нещо при използване на системата още преди регистрация е потребителят да добави своя MetaMask wallet към системата и акаунта си. Като след това пред него се визуализира страницата за регистрация и той си избира да се регистрира като Пациент, като за това се въвежда само Потребителско име, като системата ползва за идентификатор адреса на предоставяния wallet. При влизането в системата и акаунта на пациента пред него се визуализират 6 бутона, като всеки от тях прави нещо специфично и за на натискането на бутоните за добавяне на доктор или медицински институт към списъка на доверените ни такива и премахването на доктор, медицински институт от списъка ни се въвеждат и съответните техни адреси (идентификатори). Имаме и още два бутона за показване на списък на доктори и медицински институти, на които сме дали доверие. След натискане на бутон пред нас се визуализират данните и съобщение отговор на нашата заявка.



3.2. Doctor workflow диаграма

Диаграмата по-долу описва процеса на работа на докторите в нашето приложение. За достигане до страницата за регистриране процесът е същият като при описанието на горната диаграма. След това докторите в нашето приложение имат следните действия - да визуализират списък с всичките пациенти и да добавят

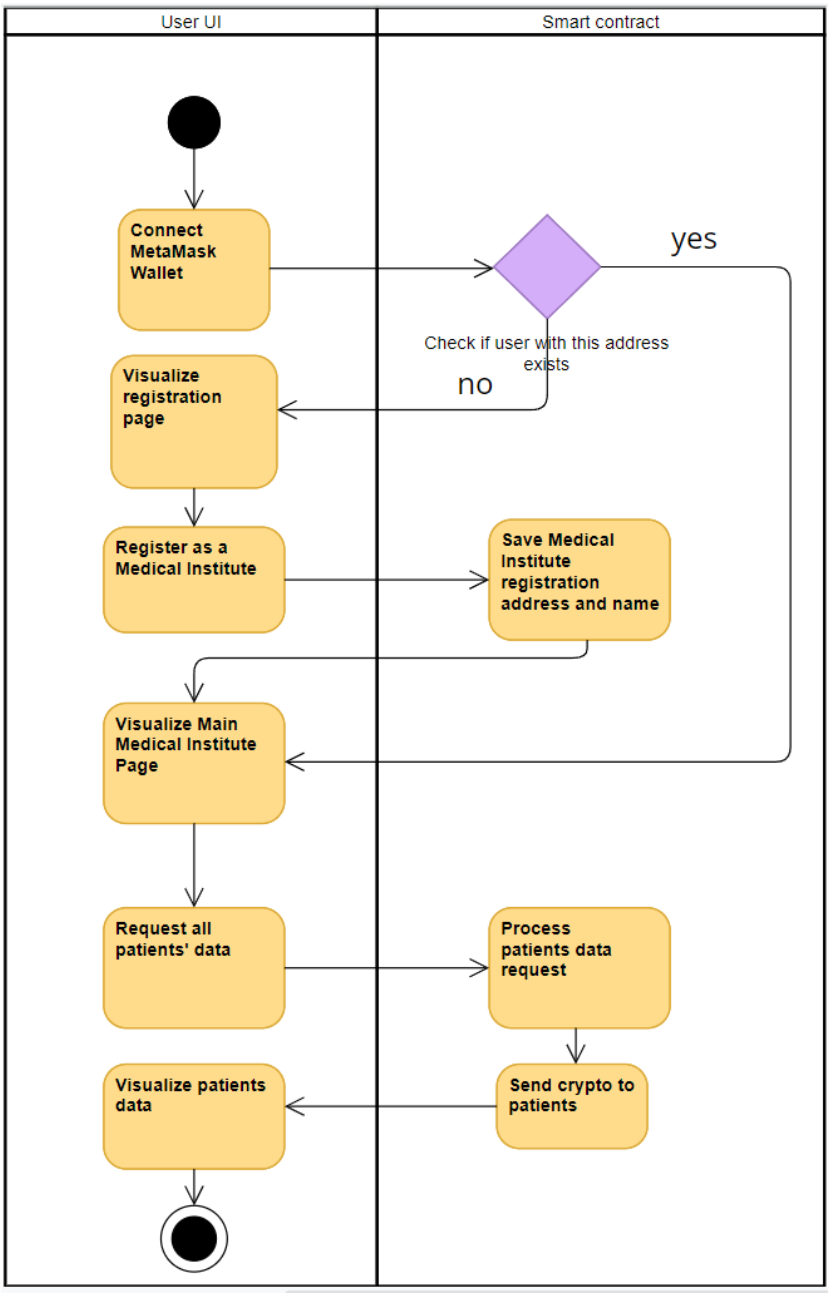
изследвания за даден пациент като се въвеждат адресът на пациента и информация за изследванията под формата на свободен текст. Като след това заявките се обработват и се визуализират съобщения и информация пред докторите.



3.3. Medical institute диаграма

Диаграмата по-долу описва действията извършвани от медицинските институти. Както при всички потребители, тук също имаме задължителното връзване към MetaMask wallet. Ако нямаме акаунт се регистрираме като медицински институт като пак се въвежда име и се взима адреса. След това се показва съответната страница, като там има опция за визуализиране на данните на всичките пациенти, които са предоставили

съгласие. При изпълнението на тази заявка се взима и изпраща крипто до MetaMask wallet-ите на всички пациенти, на които медицинския институт използва данните.



4. Данни за проекта, необходими за свързване

4.1. Адреси на Smart contract-ите

4.1.1. PatientX:

0xD26438C0282F30252674aE37298442c888DB0C18

4.1.2. InstitutesInteraction:

0x5c74B41Ca7ceb8a90e299a115044eDCdCe4E2C3c

4.2. ABI-та на Smart contract-ите

4.2.1. PatientX:

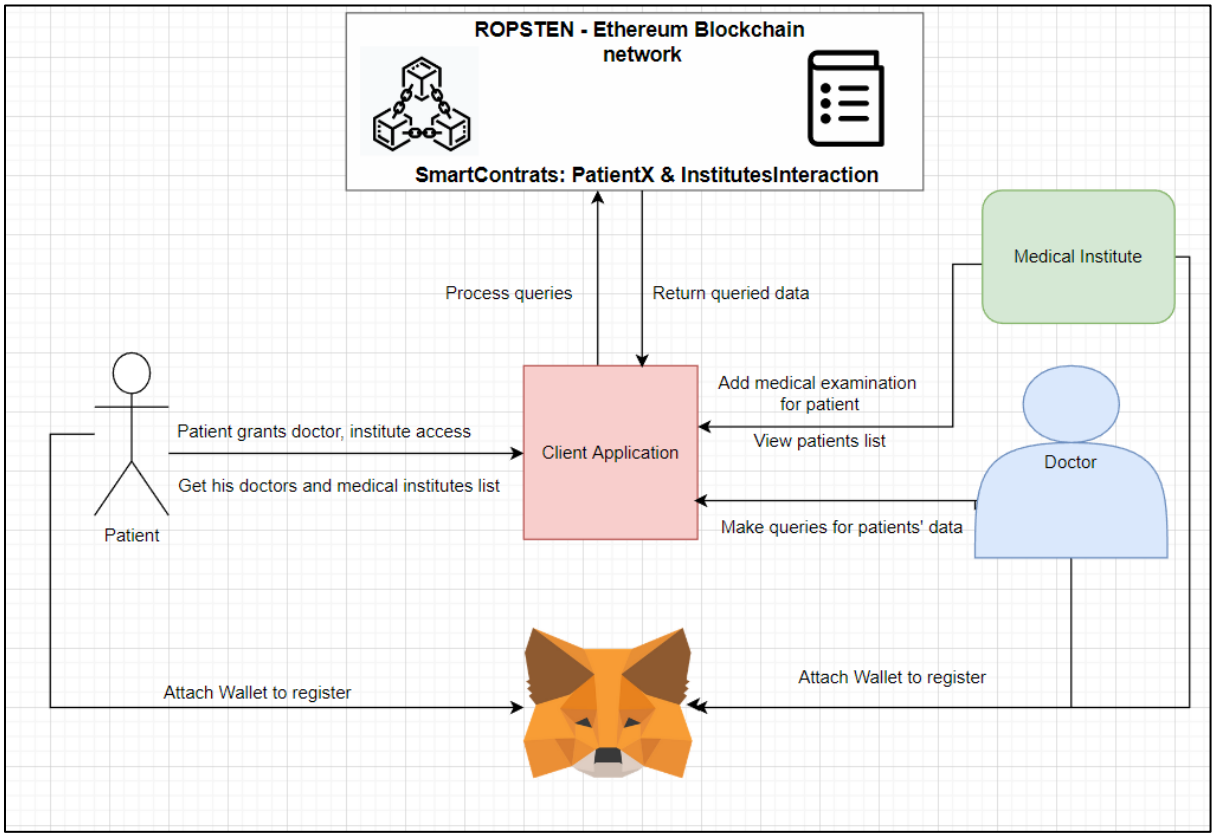
const patientxABI = [{ "inputs": [], "stateMutability": "nonpayable", "type": "constructor" }, { "anonymous": false, "inputs": [{ "indexed": false, "internalType": "string", "name": "func", "type": "string" }, { "indexed": false, "internalType": "address", "name": "sender", "type": "address" }, { "indexed": false, "internalType": "uint256", "name": "value", "type": "uint256" }, { "indexed": false, "internalType": "bytes", "name": "data", "type": "bytes" }], "name": "Log", "type": "event" }, { "anonymous": false, "inputs": [{ "indexed": true, "internalType": "address", "name": "oldOwner", "type": "address" }, { "indexed": true, "internalType": "address", "name": "newOwner", "type": "address" }], "name": "OwnerSet", "type": "event" }, { "stateMutability": "payable", "type": "fallback" }, { "inputs": [{ "internalType": "address", "name": "_patientAddress", "type": "address" }, { "internalType": "address", "name": "_doctorAddress", "type": "address" }], "name": "addDoctorToPatient", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "_patientAddress", "type": "address" }, { "internalType": "address", "name": "_medicalInstituteAddress", "type": "address" }], "name": "addInstituteToPatient", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "_doctorAddress", "type": "address" }, { "internalType": "address", "name": "_patientAddress", "type": "address" }, { "internalType": "string", "name": "_data", "type": "string" }], "name": "addNewMedicalExaminationOfPatientByDoctor", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "", "type": "address" }], "name": "allInstituteDataByAddress", "outputs": [{ "internalType": "uint256", "name": "numOfMedExams", "type": "uint256" }, { "internalType": "uint256", "name": "numOfPatientsSharingData", "type": "uint256" }], "stateMutability": "view", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "_newOwner", "type": "address" }], "name": "changeOwner", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "string", "name": "_name", "type": "string" }], "name": "enlistDoctor", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "string", "name": "_name", "type": "string" }], "name": "enlistMedicalInstitute", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "string", "name": "_name", "type": "string" }], "name": "enlistPatient", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "_patientAddress", "type": "address" }], "name": "getAllDoctorsOfPatient", "outputs": [{ "components": [{ "internalType": "address", "name": "doctorAddress", "type": "address" }, { "internalType": "string", "name": "name", "type": "string" }], "internalType": "struct MyLibrary.Doctor[]", "name": "", "type": "tuple[]" }], "stateMutability": "view", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "_patientAddress", "type": "address" }], "name": "getAllInstitutesOfPatient", "outputs": [{ "components": [{ "internalType": "address", "name": "instituteAddress", "type": "address" }, { "internalType": "string", "name": "name", "type": "string" }], "internalType": "struct MyLibrary.MedicalInstitute[]", "name": "", "type": "tuple[]" }], "stateMutability": "view", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "_doctorAddress", "type": "address" }], "name": "getAllPatientsOfDoctor", "outputs": [{ "components": [{ "internalType": "address", "name": "patientAddress", "type": "address" }, { "internalType": "string", "name": "name", "type": "string" }], "internalType": "struct MyLibrary.Patient[]", "name": "", "type": "tuple[]" }], "stateMutability": "view", "type": "function" }, { "inputs": [], "name": "getMedicalExaminationsShareable", "outputs": [{ "components": [{ "internalType": "address", "name": "doctorAddress", "type": "address" }, { "internalType": "address", "name": "patientAddress", "type": "address" }, { "internalType": "string", "name": "data", "type": "string" }], "internalType": "struct MyLibrary.MedicalExamination[]", "name": "", "type": "tuple[]" }], "stateMutability": "payable", "type": "function" }, { "inputs": [], "name": "owner", "outputs": [{ "internalType": "address", "name": "", "type": "address" }], "stateMutability": "view", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "_patientAddress", "type": "address" }, { "internalType": "address", "name": "_doctorAddress", "type": "address" }], "name": "removeDoctorFromPatient", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "_patientAddress", "type": "address" }, { "internalType": "address", "name": "_medicalInstituteAddress", "type": "address" }], "name": "removeInstituteFromPatient", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "", "type": "address" }], "name": "suchDoctorExists", "outputs": [{ "internalType": "bool", "name": "", "type": "bool" }], "stateMutability": "view", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "", "type": "address" }], "name": "suchMedicalInstituteExists", "outputs": [{ "internalType": "bool", "name": "", "type": "bool" }], "stateMutability": "view", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "", "type": "address" }], "name": "suchPatientExists", "outputs": [{ "internalType": "bool", "name": "", "type": "bool" }], "stateMutability": "view", "type": "function" }, { "stateMutability": "payable", "type": "receive" }];

4.2.2. InstitutesInteraction:

const institutesInteractionABI = [{ "inputs": [{ "internalType": "contract IPatientX", "name": "_address", "type": "address" }, { "internalType": "string", "name": "_name", "type": "string" }], "name": "enlistMedicalInstitute", "outputs": [], "stateMutability": "nonpayable", "type": "function" }, { "inputs": [{ "internalType": "address", "name": "_address", "type": "address" }], "name": "getMedicalExaminationsShareable", "outputs": [{ "components": [{ "internalType": "address", "name": "doctorAddress", "type": "address" }, { "internalType": "address", "name": "patientAddress", "type": "address" }, {

"internalType": "string", "name": "data", "type": "string" }], "internalType": "struct MyLibrary.MedicalExamination[]", "name": "", "type": "tuple[]" }], "stateMutability": "payable", "type": "function" }];

5. Архитектура на dApp-а на PatientX



Архитектурата на PatientX е от тип Web3 Application. Смарт контрактите са деплойнати на тестовата среда Ropsten Test Network като за целта е използван Truffle. При този тип архитектура е необходимо само да се качат смарт контрактите на децентрализирана машина на състоянието (Blockchain network) и да имаме потребителски интерфейс (клиентска част) за по-лесна комуникация с нея.

6. Инструкции за стартиране на клиентското приложение

За да се стартира локално проектът, е необходимо да имате инсталирани Node.js и npm. При първоначално сваляне на проекта трябва да се изпълни скриптът npm install, за да се инсталират всички необходими библиотеки и пакети на npm. След успешното инсталиране на пакетите, трябва да се изпълни скриптът npm run dev, за да се стартира react-ското приложение на localhost.

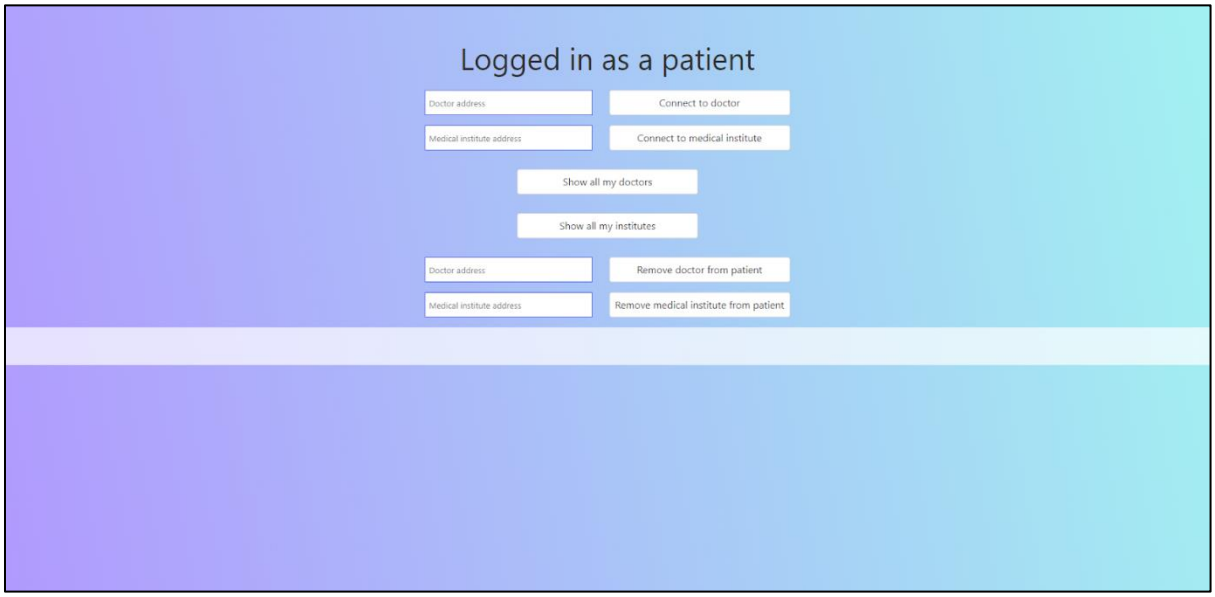
7. Използвани технологии

- **Solidity** – използван за имплементация на Smart contract-ите;
- **React** – използван за имплементация на потребителския интерфейс на приложението;
- **Bulma** – използван за стилизация на отделните React компоненти, изграждащи потребителския интерфейс.

8. User interface на dApp-а на PatientX

8.1. Landing page

8.2. Main page, Patient



8.3. Main page, Doctor

8.4. Main page, Medical institute