Acronym:	vINCI
Name:	Clinically-validated INtegrated Support for Assistive Care and Lifestyle Improvement: the Human Link
Call:	AAL 2017 “AAL Packages / Integrated Solutions”
Contract nr:	AAL-2017-63-vINCI
Start date:	01 June 2018
Duration:	36 months

D2.4. Report on technical specifications of the vINCI platform

Nature[1]:	R
Dissemination level [2]:	CO
Due date: Month:	Month 27
Date of delivery:	Month 27
Partners involved (leader in bold):	ICI, UNRF, **NIT**, CTR

Project Co-Funded by:

Project Partners:

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

INSTYTUT ŁĄCZNOŚCI
PAŃSTWOWY INSTYTUT BADAWCZY

[1] L = legal agreement, O = other, P = plan, PR = prototype, R = report, U = user scenario

[2] PU = Public, PP = Restricted to other programme participants (including the Commission Services), RE = Restricted to a group specified by the consortium (including the Commission Services), CO = Confidential, only for members of the consortium (including the Commission Services)

Partner list:

No.	Partner name	Short name	Org. type	Country
1	National Institute for Research and Development in Informatics	ICI	R&D	Romania
2	Marche Polytechnic University	MPU	R&D	Italy
3	University of Nicosia Research Foundation	UNRF	R&D	Cyprus
4	National Institute of Telecommunications	NIT	R&D	Poland
5	Connected Medical Devices	CMD	SME	Romania
6	Automa Srl	AUT	SME	Italy
7	Optima Molliter (f. Salvatelli) Srl	SAL	SME	Italy
8	National Institute of Gerontology and Geriatrics "Ana Aslan"	NIGG	R&D	Romania
9	Comtrade Digital Services	CTR	Large enterprise	Slovenia

Revision History

Rev.	Date	Partner	Description	Name
1	31.12.2019	NIT	Created the template, added the sections and content to the section 2	Waldemar Latoszek, Piotr Krawiec
2	27.04.2019	NIT	ToC update	W.Latoszek, P.Krawiec
3	09.06.2020	ICI	Contribution from ICI	ICI
4	20.08.2020	NIT	Contribution from NIT	NIT
5	28.08.2020	NIT	Editorial checking	W. Latoszek, P. Krawiec

Disclaimer:
The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

All rights reserved
The document is proprietary of the vINCI consortium members. No copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights.
This document reflects only the authors' view. The European Community is not liable for any use that may be made of the information contained herein.

Table of Contents

1. Introduction 5
2. The vINCI platform architecture 6
2.1 Main modules 6
2.2 Data storage 10
2.3 vINCI API 12
3. Security 16
3.1 vINCI Infrastructure Security 17
3.2 vINCI Server Platform Security 17
3.3 Access control 18
3.3.1 Release#1: data security ensures by using classic mechanisms 19
3.3.2 Release#2: blockchain-based approach for access rights management 19
3.3.3 Data privacy protection 20
4. Blockchain-based data access management concept 22
4.1 Implementation of the blockchain platform 23
4.1.1 Blockchain network description 23
4.1.2 Blockchain platform implementation 24
4.1.3 Fabric proxy 27
4.2 Blockchain platform validation tests 27
5. Open platform approach 28
5.1 Technical aspects for accessing patient data from the vINCI platform by 3rd Party entities 29
5.2 Technical aspects of integration with 3rd Party data providers 31
Bibliography 32
6. Annex I – HyperLedger terms definition 33
7. Annex II – Classes and methods of fabric-proxy sub-module. 35
8. Annex III - Blockchain platform validation tests results 37

9. Annex IV - Installation/configuration guides of the vINCI platform 43
9.1 Installation of the vINCI platform on NIT infrastructure 43
9.2 Network infrastructure........ 43
9.3 HTTP proxy – NGINX server 44
9.4 vINCI server platform........ 45
9.5 Access to the vINCI platform 49

1. Introduction

This report presents the deployment details of the vINCI platform. The platform aims to gather patient's data using different vINCI Kits, and next use the data to monitor patient's health.

The vINCI platform has been implemented in multi-module architecture. The implementation was done based on Java JHipster - a development platform to generate, develop and deploy web applications with microservice architectures. Microservices are JHipster applications that handle REST (Representational state transfer) requests. They are stateless, and several instances of them can be launched in parallel to handle heavy loads.

The detailed description of the vINCI platform architecture, jointly with all implementation aspects, is presented in other Project deliverables: *D2.5 Report on technologies integration and lab technical validation of kits* (section 2 *The vINCI Architecture* and section 4 *Platform Services, Data Orchestration and Integration* of [D2.5]) and *D3.3 Open data and model repository* (section *4 vINCI Open Data and Model Repository*). Consequently, this report in the next section presents only a brief description of the vINCI platform architecture and modules, and next focuses on the platform's data security mechanisms, and blockchain-based data access management developed within the scope of the project. The last section provides information on integration 3rd Party solutions with the vINCI platform.

Provided appendixes present implementation details of blockchain platform jointly with related validation tests results, and an installation guide for vINCI platform software.

2. The vINCI platform architecture

The vINCI platform allows gathering data from external sensing devices (watch, shoe/insole, camera) using REST calls. Each device type is mapped to dedicated platform's microservice. The main microservice – Gateway, handles Web traffic, and it acts as the entrance to all microservices and provides HTTP routing and load balancing, quality of service, security functions and API router for the all microservices. Moreover, the Gateway stores information about the patients (platform users) and their devices, together with user alerts and device alerts. On the other hand, the IOServer microservice stores the actual data that comes from sensing devices (in separate tables for each device type).

The brief description of functionalities of each module is presented in subsections below. Detailed specifications of the modules, their functionalities and implementation details are provided in deliverables [D2.5] and [D3.3].

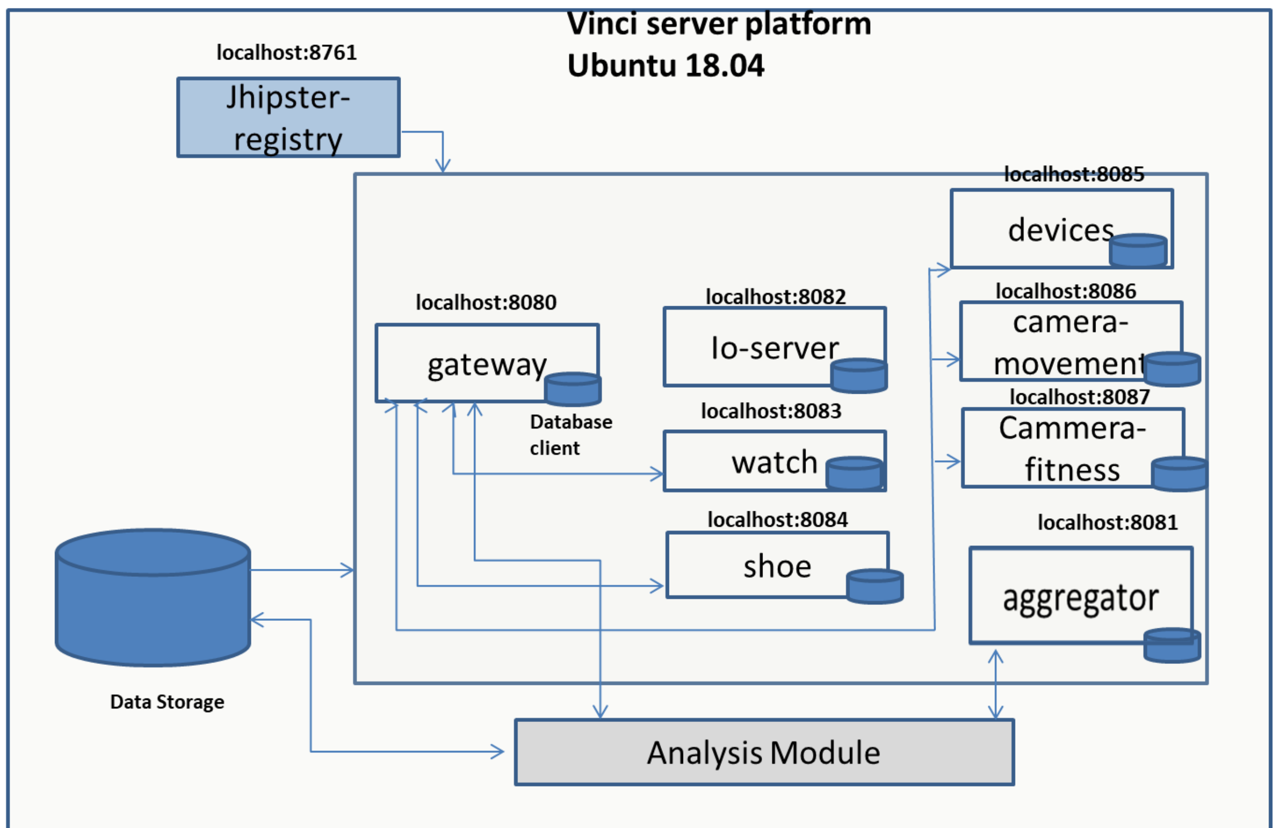


Figure 1 vINCI platform deployment architecture.

2.1 Main modules

The platform components/modules are run as Java applications in production mode or docker machines. The Figure 1 shows the architecture of the platform: all components and communication between them. All components have been implemented based on the Java Spring framework using a JHipster

generator. The Data Storage module is responsible for collecting data and is implemented as a distributed PostgreSQL database. Individual modules write and read data to the database using PostgreSQL clients.

JHipster Registry

The JHipster Registry is a runtime open source Apache application. Its source code is available on GitHub under the JHipster organization at jhipster/jhipster-registry.

The JHipster Registry has the following functionality:

- It is an Eureka server, that serves as a discovery server for Gateway and other microservices. This is how JHipster handles routing, load balancing and scalability for all applications.
- It is a Spring Cloud Config server, that provide runtime configuration to all microservices.
- It is an administration server, with dashboards to monitor and manage gateway/microservices.

Gateway

The gateway contains both the front-end part implemented using React library and the Java Spring microservice that manages the access to the vINCI platform backend.

It manages the following entities:

- User – main information about the user (login, password, name etc.),
- UserAlert – information about user alerts,
- UserExtra, - addition information about the user (address, gender etc.),
- UserImage – user profile picture;
- Device – information about user devices;
- DeviceAlert – information about device alerts.

Gateway front-end has the following main functionalities:

- user registration;
- authentication using JSON Web Tokens;
- sensor data storage by communicating with the storage stack;
- user, application data and sensor data retrieval;
- user interface, consisting of:
    - editing account data;
    - adding standalone sensors (by defining names, UUID, type, description, etc.);
    - adding virtual sensors (for the mobile application);
    - selecting applications (defining associated sensors and parameters);
    - visualizing per-application data;
    - visualizing data from any application defined in a customizable client dashboard;
- admin interface

In the backend, the Gateway is connected to all microservices and redirects all requests from client application to the microservices, using their application name: for example, when microservices *watch* is registered, it is available on the gateway on the */watch* URL. The gateway also provides additional functionalities like: rate limiting, access control and tests of microservices API.

Dashboard

The dashboard is a ReactJS client-side application which is deployed together with the gateway. The basic functionalities of the dashboard include:

- creating and managing user accounts;
- entity management;
- server administration and server operation monitoring.

In the case of managing user accounts, it is possible to register various types of users: administrators, patients, organizations, families.

The dashboard panel enables operations that allow managing entities implemented in gateway and IOServer modules. The following operations can be performed for each of the entities:

- creating new records;
- modify existing records;
- editing existing records;
- delete records.

The above-defined actions translate into calls to the appropriate REST API methods, which next perform operations on the database.

Administration and monitoring enable a set of functionalities such as:

- checking the currently running microservices;
- checking resources used by applications and statistics related to generated traffic;
- checking real-time user activities;
- configuring the appropriate login level in individual microservices.

Microservice IO-server

The microservice IOServer is a module that integrates user data with sensor data and questionnaires filled in by users. This microservice communicates on the one hand with Watch, Shoe, Cameras and Survey microservices in order to download current data, on the other hand with the gateway microservice in order to associate this data with users/patients by checking the identifiers of their devices. Intra-module communication has been implemented on the basis of Feign Client classes. In the case of data from Watch, Shoe and Cameras microservices, the data is stored in the same types of entities composed of the following parameters:

- *Id* – data identifier
- *data* - contains data in the form of JSON string,
- *timestamp* - time of data receipt,
- *deviceID* - client device ID.

Microservice Survey

The microservice Survey is responsible for managing the data obtained from the patient questionnaires. The microservice has implemented *SurveyData* and *UserExtra* entities along with REST API controllers. There are Many-to-One relationships between entities, which means that a single user/patient can complete several, different types of questionnaires.

The data stored in the survey microservice do not contain the entire questionnaires in the form of JSON strings, but only important information from the questionnaire completed by the user (see Figure 4).

Microservice devices

This microservice collects in the entity Device generic information about user device/devices and device user ID.

Microservice Shoe

The basic functionality of the Shoe microservice is processing and collecting data generated by vINCI Shoe/Insole Kits. The microservice has implemented database client connected constantly to the platform database and implements REST methods: POST,GET, PUT and DELETE. Calling the appropriate REST method causes the appropriate action in the database (*read, write, update, delete*). In the processes of calling POST and PUT methods the validation of the data in JSON format is performed.

Microservice Watch

The microservice is responsible for collecting data from vINCI Watch Kits.The microservice has implemented a function in the REST API controller to import data from the CMD platform by the POST method call (https://vinci.il-pib.pl/watch/api/import). This method calls the functions of the *retrieveAndPersist ()* of the *ImportResource* class through which it communicates with the CMD platform. As a result the received data are collected in the local storage of the watch microservice. The microservice has also implemented REST Client (based on Spring Feign Client class) to get the data from the Gateway *Device* entity and send the data to the IOServer microservice.

Microservice Camera Movement and Camera Fitness

For camera movement and fitness, two separate microservices (without their own databases) are deployed on the platform. The main function of these microservices is to receive data captured by camera devices and send them to the IOServer microservice.

Analysis Module

The analysis module is implemented in the Aggregator microservice. This module is intended for performing analyses based on data determining the patient's static profile and dynamic profile (see section *4.3.3. Patient profile* of [D3.3]).

Aggregator module communicates with the following modules: Gateway, IOServer and Survey through the use of classes implemented on the basis of Feign Client libraries, which are responsible for intra-module communication.

The microservice periodically takes all the data from the database (from the last processing onwards), and extracts a Machine Learning model to detect anomalies in patient health/behaviour. This gives the probability, given the data collected by kits, that the patient suffers a clinically-significant deterioration in

his condition(s) associated with old age (and how these conditions reflect over her/his subjective perception of QoL).

At this moment, the module implements mechanisms for extraction of collected data from the database. It's analytical logic will be implemented in parallel with the results obtained in work package *WP3 Data Analytics & Governance*, which aims to develop anomaly detection algorithm based on, among others, vINCI pilot studies.

2.2 Data storage

Data storage of the vINCI platform is implemented based on PostgreSQL database. The detailed description of the platform data storage is presented in section 4.3 *Data repository structure* of [D3.3].

Current implementation of the vINCI platform contains three logical data repository:

- Gateway database, with the structure presented in Figure 2;
- IOServer database, with the structure presented in Figure 3;
- Survey database, with the structure presented in Figure 4.

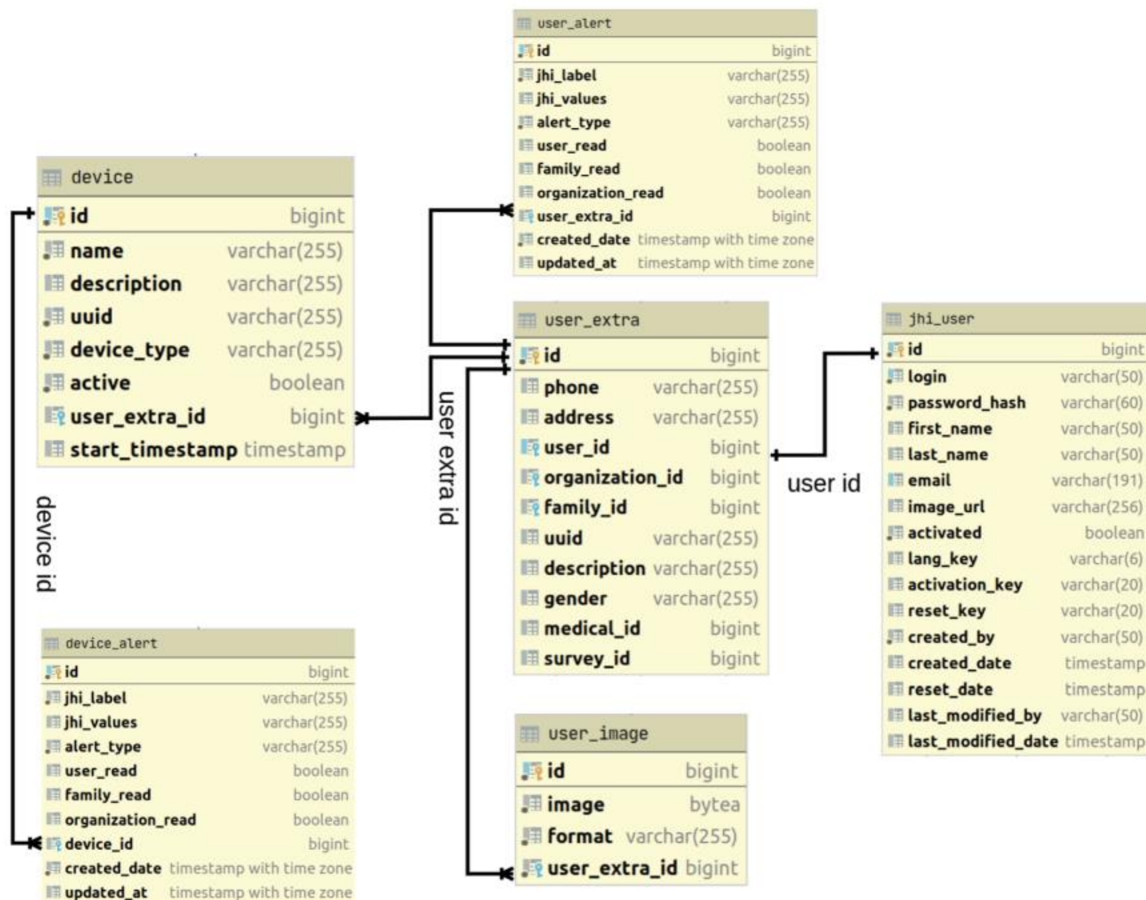


Figure 2 The structure of the Gateway database.

Please note that in the final solution the Survey database will be removed, and all its data will be migrated to IOService database (*survey_data* entity). As a result, IOService database will be the central point that provides data collected from different vINCI data sources to further analysis.

Figure 5 illustrates the relation between IOServer database and Gateway database. The anchor point is generic *device* entity that represents each device that belongs to the user.

Although PostgreSQL is an efficient and recognized database engine, high frequency of data transmissions performed by vINCI Kits may results in its performance degradation. We consider this aspect as a potential platform's bottleneck and we will monitor access delays to the database during pilots. If monitoring indicates database deterioration, we will migrate sensor data storing service to the high performance NoSQL database.

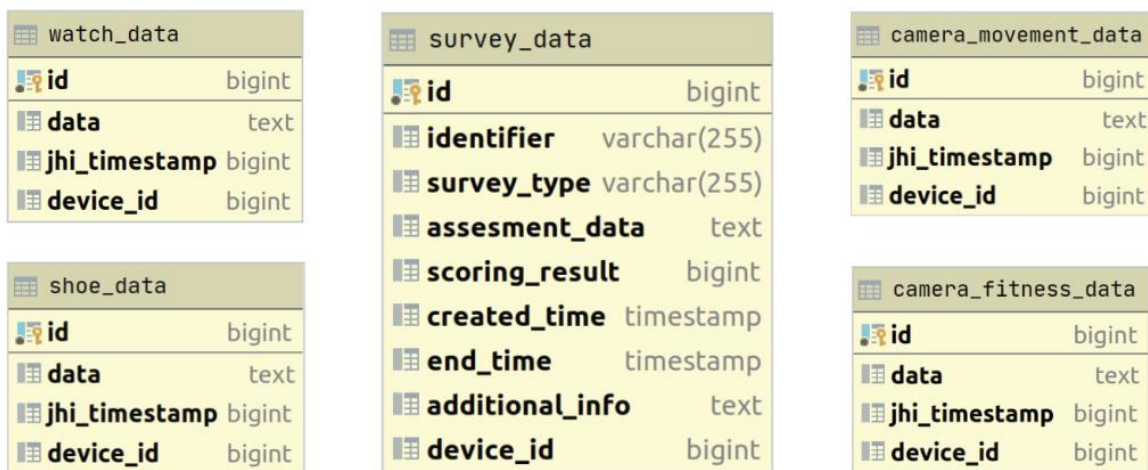


Figure 3 The structure of the IOServer database.

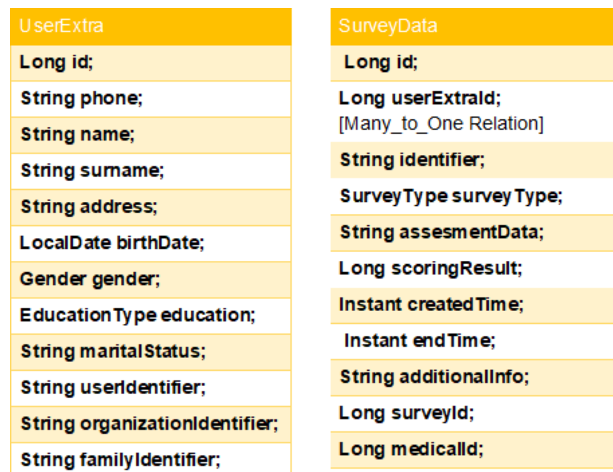


Figure 4 The structure of the Survey database.

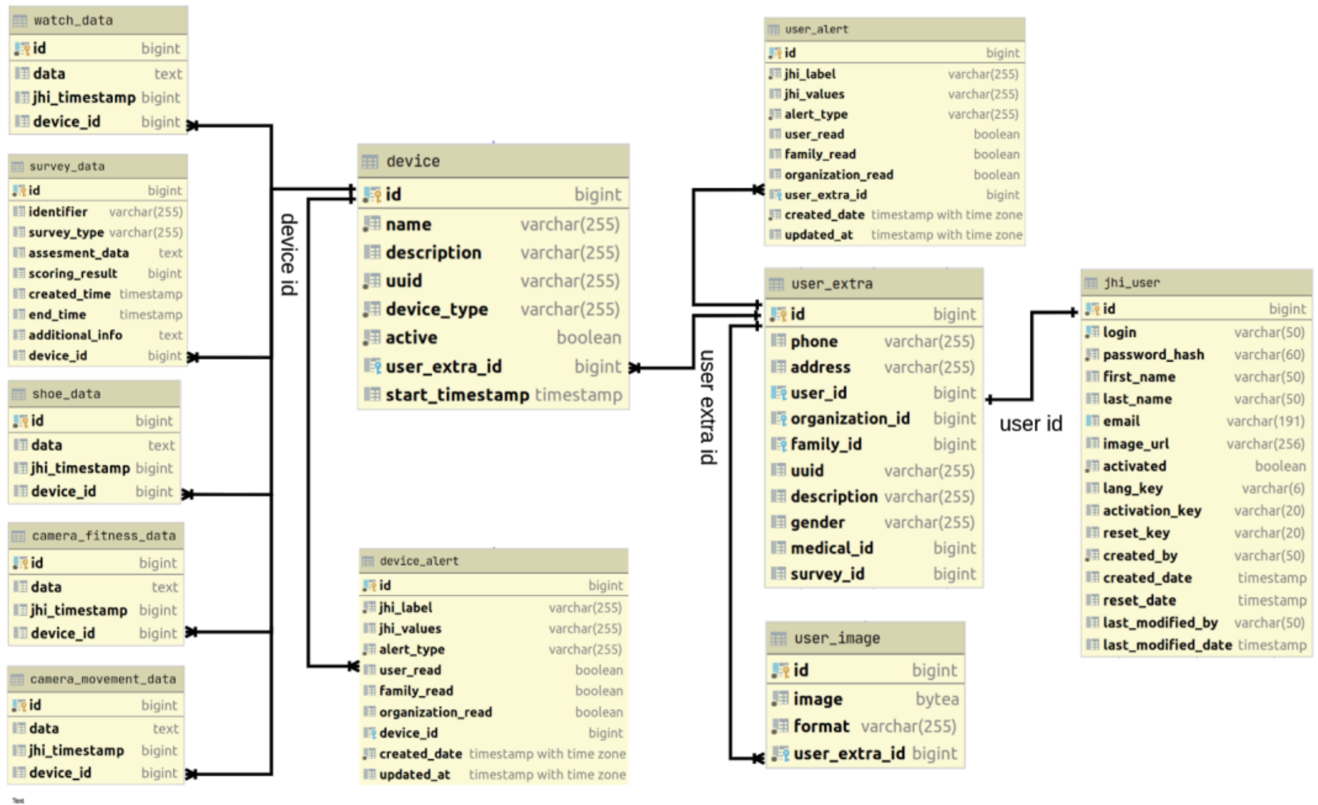


Figure 5 The important connections between the tables from the IOServer and Gateway databases.

2.3 vINCI API

The table below contains summary of all REST interfaces currently implemented in the vINCI platform.

Microservice Gateway	
Entity **User**	
	POST /users : Creates a new user
	PUT /users : Updates an existing User.
	GET /users : get all users.
	GET /users/authorities return a string list of the all of the roles
	GET /users/:login get the "login" user
	DELETE /users/:login delete the "login" User
	GET /users/aggregated get all the associated users' info plus their status
	GET /users/id-for-bracelet get the user id associated with the bracelet color and device id
	GET /users/images get all user images
	POST /users/camera-data Send camera data on the websocket
	POST /users/face-recognition : Send camera recognition on the websocket
UserExtra	

	POST /user-extras : Create a new userExtra
	PUT /user-extras : Updates an existing userExtra.
	GET /user-extras : get all the userExtras
	GET /user-extras/:id : get the "id" userExtra
	DELETE /user-extras/:id : delete the "id" userExtra
UserAlert	
	POST /user-alerts : Create a new userAlert.
	PUT /user-alerts : Updates an existing userAlert
	GET /user-alerts : get all the userAlerts
	GET /user-alerts/:id : get the "id" userAlert
	DELETE /user-alerts/:id : delete the "id" userAlert
Device	
	POST /devices : Create a new device
	PUT /devices : Updates an existing device.
	GET /devices : get all the devices
	GET /devices/type : get all watch the devices.
	GET /my-devices : get a list of devices corresponding to the logged user
	GET /devices/:id : get the "id" device.
	GET /devices/uuid/:uuid : get the "uuid" device (used by ioserver feign client).
	GET /devices/:id : get the "id" device (used by ioserver feign client)
	DELETE /devices/:id : delete the "id" device.
	GET /devices : get all the devices.
DeviceAlert	
	POST /device-alerts : Create a new deviceAlert
	PUT /device-alerts : Updates an existing deviceAlert
	GET /device-alerts : get all the deviceAlerts
	GET /device-alerts/:id : get the "id" deviceAlert
	DELETE /device-alerts/:id : delete the "id" deviceAlert

Microservice IOServer	
SurveyData	
	POST /survey-data : Create a new surveyData
	PUT /survey-data : Updates an existing surveyData
	GET /survey-data : get all the surveyData
	GET /survey-data/count : count all the surveyData
	GET /survey-data/:id : get the "id" surveyData
	DELETE /survey-data/:id : delete the "id" surveyData.
WatchData	
	POST /watch-data : Create a new watchData
	PUT /watch-data : Updates an existing watchData
	GET /watch-data : get all the watchData.
	GET /watch-data/count : count all the watchData
	GET /watch-data/:id : get the "id" watchData
	DELETE /watch-data/:id : delete the "id" watchData
	DELETE /watch-data :delete all watchData entities
ShoeData	
	POST /shoe-data : Create a new shoeData

	PUT /shoe-data : Updates an existing shoeData
	GET /shoe-data : Get the list of data for a shoe device id
	GET /shoe-data/count : count all the shoeData
	GET /shoe-data/:id : get the "id" shoeData
	DELETE /shoe-data/:id : delete the "id" shoeData
CameraFitness	
	POST /camera-fitness-data : Create a new cameraFitnessData
	PUT /camera-fitness-data : Updates an existing cameraFitnessData
	GET /camera-fitness-data : get all the cameraFitnessData
	GET /camera-fitness-data/count : count all the cameraFitnessData
	GET /camera-fitness-data/:id : get the "id" cameraFitnessData
	DELETE /camera-fitness-data/:id : delete the "id" cameraFitnessData
CameraMovement	
	POST /camera-movement-data : Create a new cameraMovementData
	PUT /camera-movement-data : Updates an existing cameraMovementData
	GET /camera-movement-data : get all the cameraMovementData
	GET /camera-movement-data/count : count all the cameraMovementData
	GET /camera-movement-data/:id : get the "id" cameraMovementData
	DELETE /camera-movement-data/:id : delete the "id" cameraMovementData

Microservice Survey	
SurveyData	
	POST /survey-data : Create a new surveyData
	PUT /survey-data : Updates an existing surveyData
	GET /survey-data : get all the surveyData
	GET /survey-data/:id : get the "id" surveyData
	DELETE /survey-data/:id} : delete the "id" surveyData
	GET /survey-data : get all the surveyData for userExtraId
	GET /survey-data : get all the surveyData for surveyId
	GET /survey-data : get all the surveyData for medicalId
UserExtra	
	POST /user-extras : Create a new userExtra
	PUT /user-extras : Updates an existing userExtra.
	GET /user-extras : get all the userExtras
	GET /user-extras/:id : get the "id" userExtra
	DELETE /user-extras/:id : delete the "id" userExtra
	GET /user-extras/user/{userIdentifier} GET userExtra for user
	GET /user-extras/user/family/{familyIdentifier} GET get all the userExtras by family
	GET /user-extras/user/organization/{organizationIdentifier} GET get all the userExtras by organization

Microservice Watch	
Watch	
	POST /import : Records import API

Microservice Shoe	
Shoe	POST /records : Create a new record
	POST /import : Import a list of records
	PUT /records : Updates an existing record
	GET /records : get all the records
	GET /records/count : count all the records
	GET /records/:id : get the "id" record
	DELETE /records/:id : delete the "id" record

3. Security

vINCI platform handles sensitive patient's data, therefore a very important aspect is to ensure the security of data transfer, data storage, data access as well as data privacy. Set of requirements related to this issue have been presented in [D3.1] (requirements Req#1 – Req#18 described in section 4.2 of [D3.1]). The implementation of the platform was aimed at meeting all the specified requirements.

The figure below shows the overall security architecture of the vINCI platform deployed in the NIT, using the PL-LAB[1] experimental facility. It provides three levels of security provisioning:

- level 1 – NIT firewall secure functions;
- level 2 - HTTP Proxy: the NGINX server with security mechanisms;
- level 3 - vINCI server platform security mechanisms.

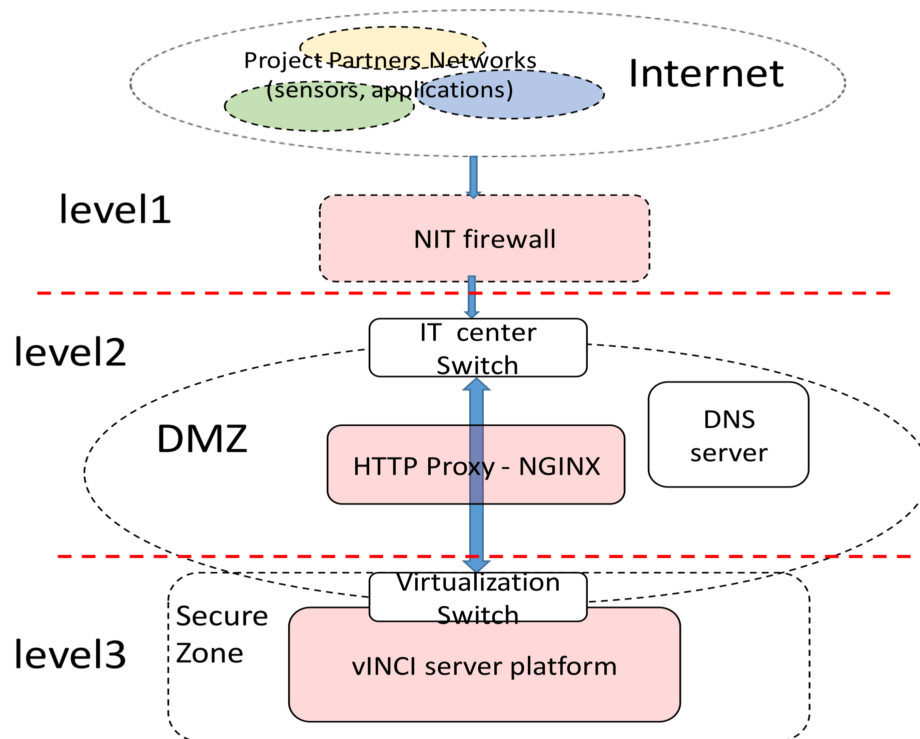


Figure 6 vINCI platform overall security architecture.

[1] PL-LAB2020 is the Polish network testbed developed under the project financed by the European Union through the European Regional Development Fund under the Operational Programme 'Innovative Economy' for the years 2007-2013. Project number: POIG.02.03.01-00-104/13-00

3.1 vINCI Infrastructure Security

The PL-LAB infrastructure is located in a dedicated server room with restricted access, ensuring physical security.

NIT Firewall

The firewall provides protection against threats coming from the Internet and network protection against intrusion. The firewall configuration includes:

- configuration the external public IP address for the vINCI platform;
- transition (NAT) the local address of vINCI platform to the public IP address;
- limiting the ports exposed externally - only ports for ssh (22) and https (443) services should be opened (note: for testing purposes during system development, also http ports are opened);
- restricting access to the vINCI platform only for the network of Project partners.

DMZ (Demilitarized Zone)

The DMZ includes the following:

- **Switch network** configured into one common VLAN with KVM switch installed on the vINCI server platform;
- **DNS server -** The DNS server is maintained by the NIT IT center. Domain names of the vINCI platform entered into the DNS are:
  - www.vinci.il-pib.pl
  - vinci.il-pib.pl
- **HTTP proxy – NGINX server.** The NGINX server is a proxy for the system and it is responsible for the following security mechanisms:
  - SSL termination and proxy to the vINCI platform gateway module;
  - maintaining certificates for HTTPS operation for domain names;
  - maintaining certificates for user authentication based on the private key;
  - limiting access to Proxied HTTPS Resources;
  - restricting access to Proxied HTTPS Resources;
  - dynamic blacklisting of IP Addresses;

Using DMZ approach ensures physical and logical protection of remote diagnostic and devices' configuration ports.

3.2 vINCI Server Platform Security

The security mechanisms used in the implementation of the vINCI system are described below.

JWT (JSON Web Token) authentication

JWT (JSON Web Token) is an industry standard, easy-to-use method for securing applications in a microservices architecture [jwt].

The authentication process is as follows:

1) the client sends a request with user and password;
2) the gateway redirect them to the dedicated microservice, which is a specialized authorization component;

3) the gateway receives a token with the user's ID and role and send it to the client;
4) for each subsequent request, the client includes the token, which the API gateway checks using the AUTH microservice.

To ensure security, a JWT secret token must be shared between all microservices. The tokens are self-sufficient: they have both authentication and authorization information, so microservices do not need to query a database or an external system. This is important in order to ensure a scalable architecture.

The JWT authentication implies that access to individual applications must require a user ID and authentication.

User roles

The Gateway front-end (login view) allows the access for different types of users:

- "**SystemUser"**, who is mainly used by our audit logs, when something is done automatically;
- "**AnonymousUser**", who is given to anonymous users when they do an action;
- **"User",** who is a normal user with "ROLE_USER" authorization; the default password is "user";
- "**Admin",** who is an admin user with "ROLE_USER" and "ROLE_ADMIN" authorizations; the default password is "admin".

HTTP configuration security

Basic options for HTTP controlling access to the microservices resources are defined in the *SecurityConfiguration.java* file (for each microservice). The following rights are defined in the file:

- related to HTTP user authentication;
- resource access rights for users;
- management access for users.

Security of inter-service-communication using Feign clients

The communication between modules is based on Feing clients. This communication also has an appropriate level of security. To accomplish this, some request interceptor for Feign has been implemented, which implements the client credentials flow from OAuth to authorize the current service for requesting the other service. In JHipster, it is used *@AuthorizedFeignClients* instead. This is an annotation provided by JHipster, which performs this functionality.

Testing of the vulnerability to attacks requirement, as recommended in the Open Web Application Security Project - OWASP Top 10, is fulfilled through application of the OWASP dependency-check-gradle plugin in the compilation of vINCI platform microservices.

3.3 Access control

Ensuring the security and confidentiality of data in the vINCI system requires the implementation of appropriate mechanisms to manage permissions and access control to data and/or functions.

In order to facilitate system implementation, we have introduced two phases in the development of security mechanisms for the vINCI platform.

The first phase is that security of access to data is realized by using classical security mechanisms, according to the current state of the art and guidelines of relevant bodies and organizations (NIST,

OWASP etc.). This enables the quick development of the platform (we name it as *Release#1*), which is ready to accept the data and carry out the first tests of the vINCI system.

In parallel to the development and maintenance of the vINCI platform Release#1, we are working on a new data access management solution based on blockchain technology. The vINCI platform integrated with blochkchain-based access management is called *Release#2*.

3.3.1 Release#1: data security ensures by using classic mechanisms

We use the following mechanism to secure access to the vINCI data (Figure 7):

- Access to service is possible only for authorized users/devices. By default, the authentication and authorization mechanism relies on client SSL authentication certificates. However, mainly for testing purposes, the platform allows also for login/password authentication.
- Communication security is provided by using HTTPS protocol.

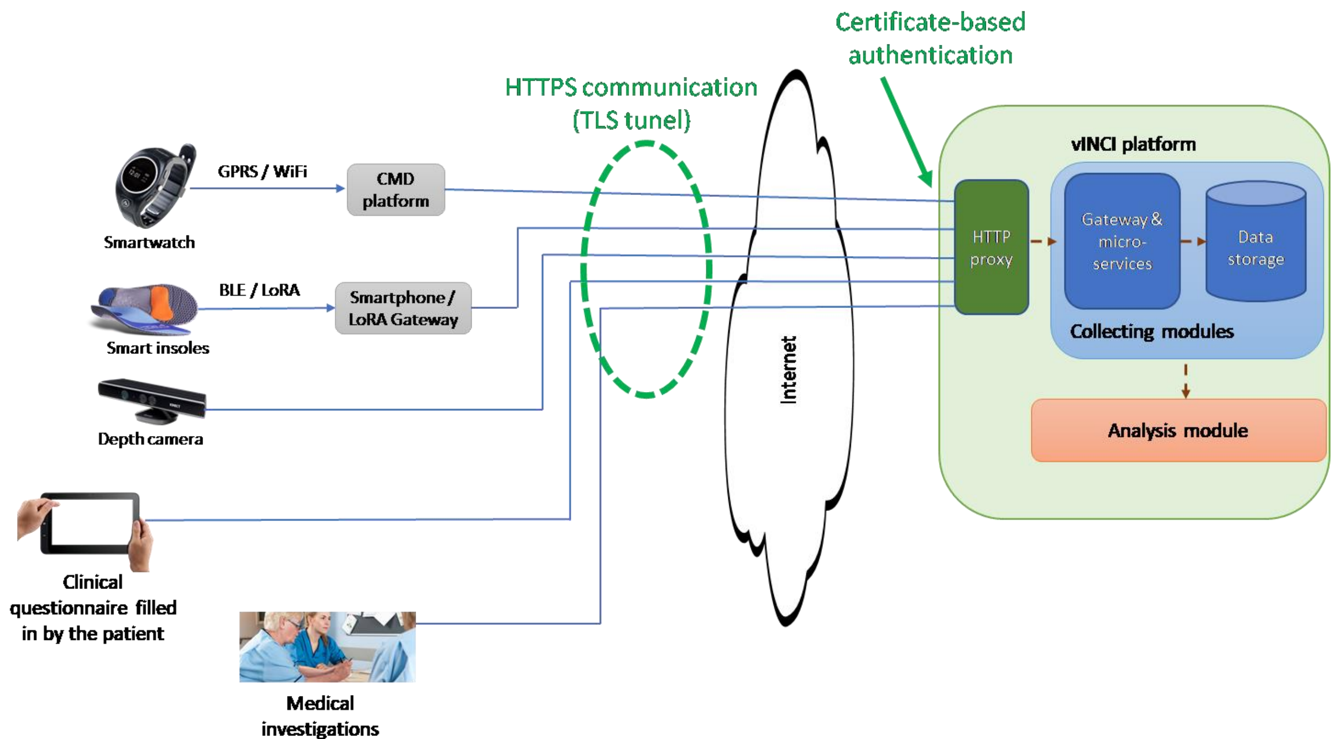


Figure 7 The data security mechanisms (Release#1) of the vINCI platform

3.3.2 Release#2: blockchain-based approach for access rights management

In Release#2 of the vINCI platform, management of permissions and access control to data is performed using blockchain (Figure 8). Blockchain is a technology that supports sharing of values, in case of vINCI: medical data. Blockchain is a digital ledger where there are stored all the executed transactions. It uses a distributed, peer-to-peer network to make a continuous growing list of ordered records called blocks. Every block contains a set of signed transactions and is validated by the network itself, by means of a consensus mechanism. Copies of the blockchain are distributed on each participating node in the

network. Blockchain can be considered as a permanent database because the implemented algorithms prevent alteration of the already stored information.

Blockchain provides unified, secure and user-controlled access to patient's health data. It allows users to easily grant, modify or revoke access to their data. Thanks to blockchain, patients' data are:

- tamper-proof,
- shareable and retrievable for carers/healthcare providers who have been granted access to it,
- secure - entities that have granted permissions to data from data owners, can access health records only when their identities and cryptographic keys have been verified by blockchain.

We use open source Hyperledger Fabric framework as an implementation platform. The prototype application will implement the main components of a blockchain based software such as blockchain display, blockchain query, adding new transaction, transactions validation, creating blocks and appending them to blockchain, broadcast blockchain, and blockchain integrity check.

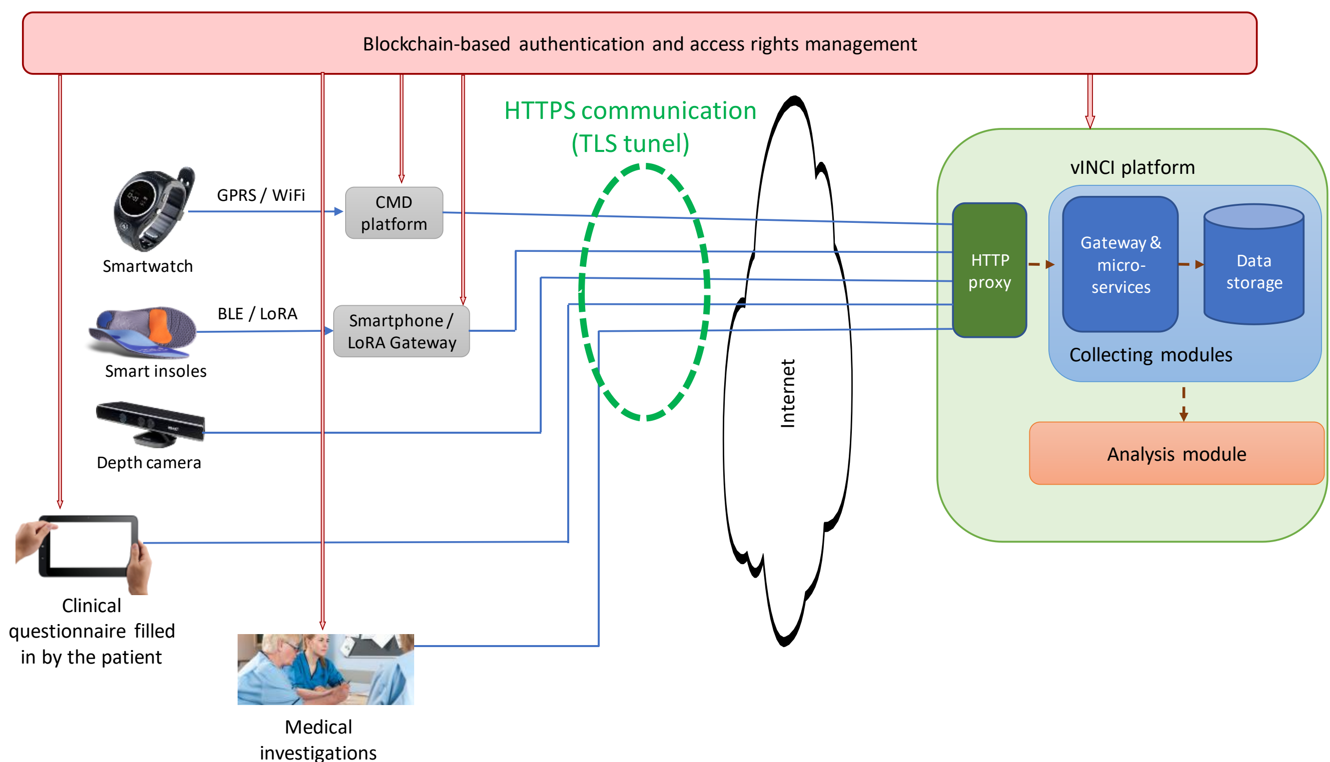


Figure 8 The data security mechanisms (Release#2) of the vINCI platform

More details of blockchain-based data access management are presented in Section 4 of this report.

3.3.3 Data privacy protection

In order to increase the security and privacy of personal data, our system uses pseudonymization mechanisms to remove the characteristics of personal data. Data containing personal data are kept based on data separation model, investigating two databases: *identification* database and *medical* database.

The identification database contains only identification data, without medical data - although it is still personal data, it does not contain any additional information apart from pure data identifying, and above all, do not contain sensitive information about the patient's health.

The identification data set contains two entities from Gateway module: User and UserExtra (see Figure 2). All other entities from Gateway, IOServer and Survey microservices are considered as medical data set, that do not contain identification information (see Figure 3). The linking of the relevant data records from the identification data set with health related sensitive data is made by using UserExtra Id, which is a random attribute disguising an individual's identity (see Figure 5).

Moreover, fields of the UserExtra entity are encrypted (using PostgreSQL encrytpion options) to hide relationship between User Id used for user identification and UserExtra Id used in medical data storage (see Figure 5).

In case of the analysis module, we consider entities “Contact Information” and “Socio-cultural data” of the patient's static profile (see section 4.3.3 of [D3.3]) as an identification data set, which is separated from a medical data. In turn, medical data set includes all entities related with dynamic profile, as well as medical information from static profile.

It should be noticed, however, that simultaneous, unauthorized access to both databases (identification and medical) immediately causes the loss of data confidentiality - as a consequence of the existence of an explicit relationship. Nevertheless, such a solution is a tradeoff between high security demands and high performance requirements, which is also important for a system that handles large amount of data.

To enhance security, the two data sets are physically separated, so that each of them is stored in a different database, on a different virtual server.

4. Blockchain-based data access management concept

As mentioned in section 3.3.2, the vINCI platform (Release#2) provides blockchain-based access management to patient's data as an additional feature that increases data security and data access flexibility.

Our concept, as presented in [D3.1], assumes that patients are recognized as owners of their own health data and have full control over it. They can apply various security policies, such as sharing data with specific clinics or institutions, and can contribute anonymously to certain statistics. These policies are stored securely in blockchain network which ensures a high level of guarantees that such data access policies will not be modified or injected into the system in an unauthorised manner.

Security policies will be created by patients in a simplified way by using patient's app. This process requires twofold trustiness:

- The system's trust in the user who defines a new policy for accessing (her/his) medical data;
- The user's trust in the entity to which he grants access rights to her/his medical data (doctor, hospital etc.).

The first point can easily be met by traditional security mechanisms (access control), as the vINCI system stores data about its patients (users) and thus verifies their identity.

It is more complicated in the second case. It can often turn out that an entity wishing to access a patient's medical data, is not a user of the vINCI system (e.g. a doctor that a patient has applied to during a holiday trip). In such a case, the verification of the identity of this type of entity falls exclusively on the patient, as the vINCI system has no information about this entity.

A helpful solution in the latter case is Self-Sovereign Identity (SSI), i.e. the idea that a digital identity can be created and used without dependencies on central or hierarchical authorities. In the concept of SSI, Verifiable IDs and Verifiable Attestations (both types of "Verifiable Credentials") enable entities (i.e. natural and legal persons: doctors, hospitals, insurance companies) to claim certain things about themselves or others in a way that these claims can be regarded as proofs for certain attributes (i.e. verify their identity against the vINCI system and vINCI users).

An example that supports those principles is ESSIF (European Self-Sovereign Identity Framework) ID Service – a blockchain ID service developed in the framework of EBSI (European Blockchain Services Infrastructure). This service is designated to provide unified identity verification of European level, based on the background on different national Trusted Issuers that confirms Verifiable ID within its scope.

In this way, the entity to which patient wants to grant access will not have to be a vINCI registered entity, but any legal entity which is credible to ESSI. This will significantly facilitate cooperation with institutions such as clinics, doctors etc., to which the vINCI client will be able to share data whenever he/she wishes, without the need for complex, direct interaction with the vINCI system (to register the entity in the vINCI platform). In particular, this solution facilitates cross-border interaction with vINCI data and keeps patient's health data interoperable on European level.

As a result, patients are recognized as owners of their medical data and analysis results, have full control over them and can apply access policies any time for whomever they want.

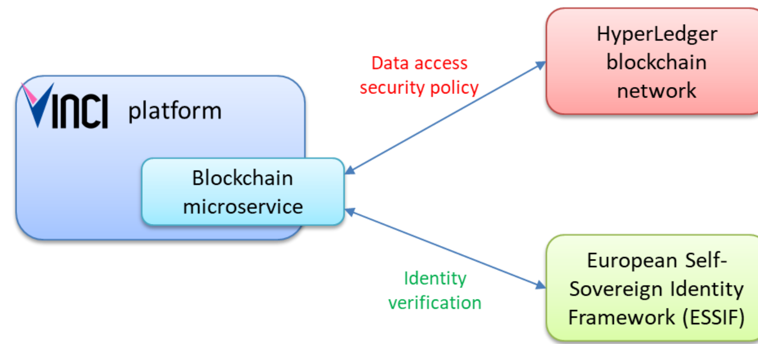


Figure 9. Concept of blockchain-based access data management with ESSIF identify verification.

Current Release#2 implementation includes blockchain network that stores data access policies. The upcoming work will include the integration of the vINCI system with the module implementing the interfaces specified so far under EBSI Technical Group.

4.1 Implementation of the blockchain platform

The implementation of the blockchain platform is based on an open source software - HyperLedger Fabric version 1.4 [hyf].

The blockchain platform consists of the following three components:

- **blockchain microservice**
- **fabric-proxy**
- **blockchain network**

The blockchain microservice is the vINCI platform module that is integrated with the Gateway module. The fabric-proxy is the module responsible for connecting the blockchain network with the vINCI platform. The communication with the blockchain network is based on the *fabric-gateway-java* [fgj] libraries that allows applications to interact with a fabric blockchain network.

The blockchain network is a set of components launched as docker machines that are implemented based on *fabric-samples* software [hfs].

The following subsection presents the main components of the blockchain platform and implementation issues.

4.1.1 Blockchain network description

For the project purposes, the blockchain network consists of two organizations[2], each maintaining two peer nodes. There was deployed a "kafka" ordering service by default. Each organization has its own

[2] HyperLedger terms definitions are presented in Annex I.

Certificate Authority (ca_peerOrg1, ca_peerOrg2). Individual nodes are the docker virtualization system machines and are configured based on *docker-compose* file. The Peer0 in Org1 and Org2 are designated as the anchor Peer. All transactions saved in the blockchain of individual peers are finally entered into the distributed *CouchDB* database – a document-oriented NoSQL database where document fields are stored as key-value maps.

Moreover, a CLI container is launched to execute scripts that will join peers to a channel, deploy a chaincode and drive execution of transactions against the deployed chaincode. The following nodes have started in the blockchain network (Figure 10):

- *ca_peerOrg1*
- *ca_peerOrg2*
- *order.example.com*
- *peer0.org1.example.com*
- *peer1.org1.example.com*
- *peer0.org2.example.com*
- *peer1.org2.example.com*
- *cli*
- *couchdb0,*
- *couchdb1*
- *couchdb2*
- *couchdb3*

```
4defecc2eee8        hyperledger/fabric-tools:latest
days                                                             cli
b487d601052c        hyperledger/fabric-peer:latest
days           0.0.0.0:8051->8051/tcp                            peer1.org1.example.com
b2ecb5e4bb51        hyperledger/fabric-peer:latest
days           0.0.0.0:9051->9051/tcp                            peer0.org2.example.com
37317b5d3cd3        hyperledger/fabric-peer:latest
days           0.0.0.0:7051->7051/tcp                            peer0.org1.example.com
21d7214b52a8        hyperledger/fabric-peer:latest
days           0.0.0.0:10051->10051/tcp                          peer1.org2.example.com
0280a06baff2        hyperledger/fabric-couchdb
days           4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp        couchdb2
5c19b687aeef        hyperledger/fabric-ca:latest
days           7054/tcp, 0.0.0.0:8054->8054/tcp                  ca_peerOrg2
07520cba07b4        hyperledger/fabric-couchdb
days           4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp        couchdb3
d834379a1abd        hyperledger/fabric-ca:latest
days           0.0.0.0:7054->7054/tcp                            ca_peerOrg1
7c4960870ffc        hyperledger/fabric-couchdb
days           4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp        couchdb0
e76f83979628        hyperledger/fabric-orderer:latest
days           0.0.0.0:7050->7050/tcp                            orderer.example.com
4ad377769eab        hyperledger/fabric-couchdb
days           4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp        couchdb1
```

Figure 10 Installed components of the blockchain network

4.1.2 Blockchain platform implementation

The blockchain platform implementation includes three modules: microservice blockchain, fabric-proxy and blockchain network. The Figure 11 shows the integrated architecture of the platform.

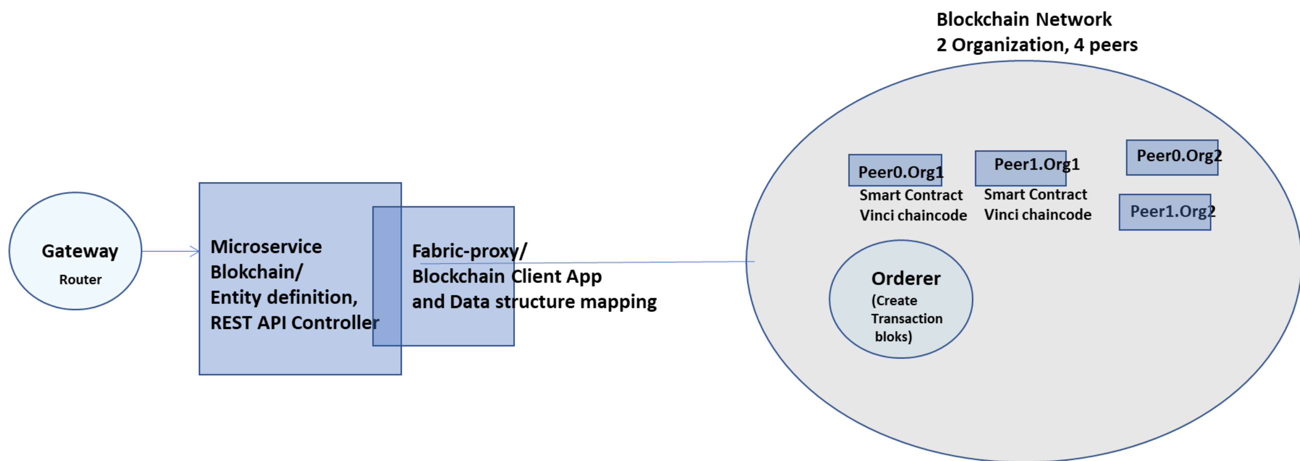


Figure 11 Integrated architecture of the blockchain platform

Blockchain microservice

The general idea of using blockchain platform has been presented in [D3.1], section 4.3. This chapter describes the implementation based on the above concept and adapted to the implementation of the vINCI platform. The main assumption of using blockchain technology is managing access to patients' medical data. Thanks to this, it is possible to make diagnostic analyzes available only to specific entities/organizations (doctors, medical institutions and other users). Data access rights are defined in transaction blocks of the blockchain network, while health data (medical analyzes) are stored in the vINCI database.

The interaction of the vINCI system with the blockchain network is based on the blockchain microservice and its sub-module fabric-proxy (see Figure 11). The microservice defines two main entities whose data is saved as transactions in the blockchain network: **Storage** and **Policy**.

The storage entity specifies information about a single medical analysis for a patient. A *Policy* entity is defined for each *Storage* transaction and specifies data access policy by calling the appropriate REST API methods defined in the microservice.

The scenario for creating a *Storage* and *Policy* transaction is as follows:

1. Storage transactions are recorded periodically and are performed automatically by the analysis module running on the vINCI platform. The recording is done via REST API communication between the analysis modules and microservice blockchain and the frequency of transaction recording depends on the configuration of the analysis module.
2. The patient defines the list of entities authorized to access by calling the corresponding REST API functions of the blockchain microservice, as a result of which the Policy transaction associated with the Storage transaction is saved. At this step, verification of the entity that wants to get access to patient’s data will be performed based on ESSIF service.
3. The concerned entity queries the appropriate Policy transaction entity to retrieve the keys for accessing patient data.
4. Based on the received security access key, it is possible to call the REST API function to obtain patient diagnostic results saved in the blockchain database.

The object definitions of the **BlockchainStorage and BlockchainPolicy** entities and REST API functions defined in the blockchain microservice are presented below.

*public class **BlockchainStorageDTO** {*

private Long TransactionID;
private Long UserID;
private Long MedicalID ;
Instant Timestamp;
}

*public class **BlockchainPolicyDTO** {*

private Long TransactionID;
private Long organizationID;
private String Signature_of_the_granted_organization;
private String Users_ signature;
private String Access_key;
Instant Timestamp;
}

REST API Controller methods:

BlockchainStorage entity:

- **POST /blockchainstorages :** Create a new userBlockchain , the function calls the method *createStorageRecord() of* Blockchain.java class.
- **PUT / blockchainstorages:** Updates an existing userBlockchain , the function calls the method *updateStorageRecord() of* Blockchain.java class
- **GET / blockchainstorages:** get all the userBlockchains, the function calls the method *getAllStorageRecords() of* Blockchain.java class.
- **GET / blockchainstorages /:id :** get the "id" userBlockchain, the function calls the method *getStorageRecord () of* Blockchain.java class.
- **DELETE / blockchainstorages /:id :** delete the "id" userBlockchain, the function calls the method *deleteStorageRecord() of* Blockchain.java class.

BlockchainPolicy entity

- **POST /blockchainpolicy:** Create a new userBlockchain , the function calls the method *createPolicyRecord() of* Blockchain.java class.
- **PUT / blockchainpolicy:** Updates an existing userBlockchain , the function calls the method *updatePolicyRecord() of* Blockchain.java class
- **GET / blockchainpolicy:** get all the userBlockchains, the function calls the method *getAllPolicyRecords() of* Blockchain.java class.
- **GET / blockchainpolicy /:id :** get the "id" userBlockchain, the function calls the method *getPolicyRecord () of* Blockchain.java class.

- **DELETE / blockchainpolicy /:id :** delete the "id" userBlockchain, the function calls the method *deletePolicyRecord() of* Blockchain.java class.

4.1.3 Fabric proxy

The fabric-proxy module is responsible for the integration of the blockchain vINCI microservice with the blockchain network components. The main functions of the fabric-proxy module are as follows:

- User management (creating and granting permissions to blockchain network through certificates);
- Ensuring connection to the blockchain network, calling transactions and queries;
- Mapping functions and data formats from the REST API controller microservice to functions of the blockchain network components.

The fabric-proxy interacts with the blockchain network by calling functions implemented in chaincode (smartcontract). The chaincode is a program running on specified peers on the blockchain network. Implementation of the vINCI chaincode was instantiated on the common communication channel and installed on two endorsing peers (peer0.org1.example.com and peer0.org2.example.com).

Annex II presents implementation details of the fabric-proxy.

4.2 Blockchain platform validation tests

The preliminary validation tests have been performed to check correctness of blockchain service implementation. The tests covers:

- integration of blockchain platform with the vINCI platform;
- blockchain microservice REST API function tests for the *Userblockchain* tests entity.

The blockchain microservice is a component of both the vINCI platform and the blockchain platform. To integrate both platforms, it is necessary to import the blockchain microservice entity into the Gateway module. Thanks to this, the data saved on the blockchain platform will be reached from the Gateway front-end. The performed integration test confirms the correct integration of both platforms (see Annex III).

The REST API tests were designed to verify the basic API functions in an integrated environment that includes blockchain microservice, fabric-proxy module and blockchain platform. The tests are related to operations on an example data entity defined in the blockchain microservice.

The following tests were performed based on python script as a client application using JSON encoder and decoder and *request* library to generate HTTP method calls. The test results obtained (presented in Annex III) confirm proper data exchange between three engaged entities.

5. Open platform approach

Communication with vINCI platform relies on commonly used HTTPS protocol. HTTPS provides adequate security for data transfer (data encryption, validation of data sender and receiver, etc.), and also high platform accessibility since HTTPS port (443) are usually not blocked by firewalls. vINCI platform implements external interfaces that base on REST API with open standard JSON data format. Using this API, 3rd party entities can interact with the platform in two ways:

a) by consuming data from vINCI platform;
b) by extending vINCI platform functionality with their own data sources/sensors (*3rd Party Kit*).

Therefore, we distinguish two categories of 3rd Party entities (see Figure 12):

1. **Consumer** - doctors, research entities, medical units: they are mainly interested in obtaining health/diagnostic data of patients, statistics for a specific group of patients, history for a given patient taking into account reported alarms, etc.
2. **Data Provider**- sensor providers: they aim to integrate their own sensors/devices with the vINCI system.

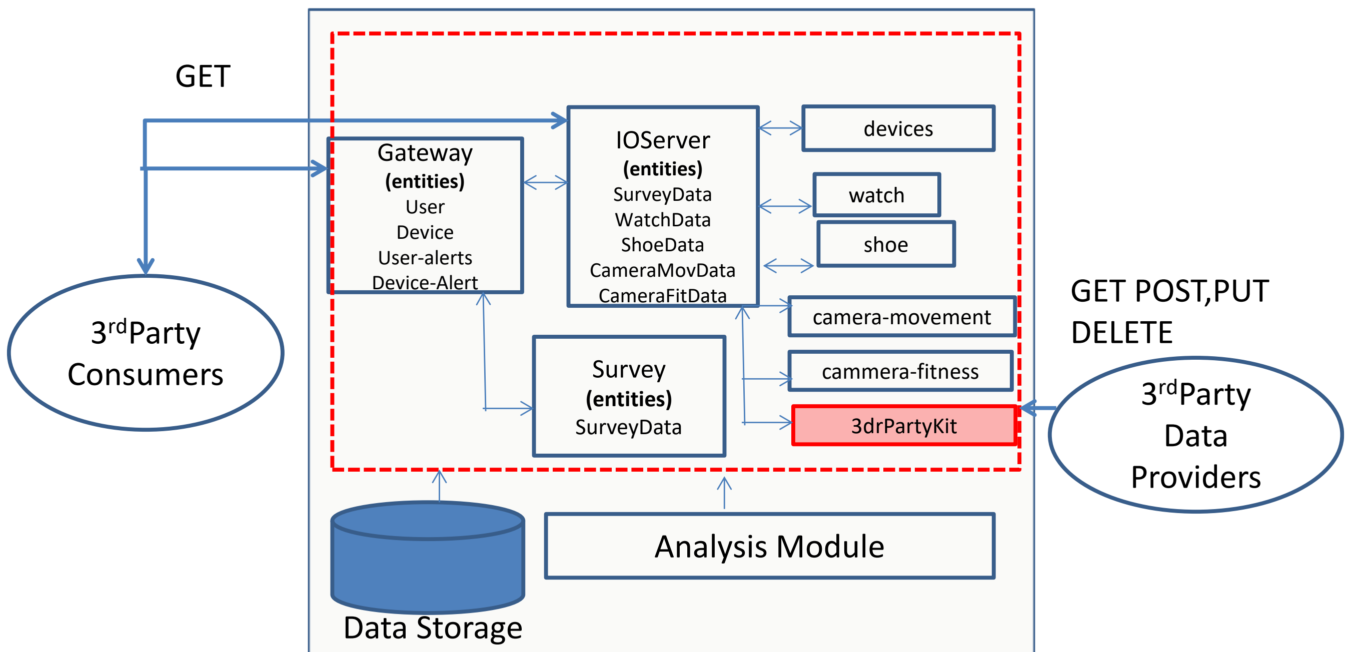


Figure 12 The concept of the integration of the vINCI system with 3rd Parties services

We assume that *Consumer* entities uses the REST API limited mainly to GET calls. In few justified cases it will be possible to allow using other methods, for example POST to provide own survey data to vINCI system. *Consumer* will use the REST API for the Gateway, IOServer and Survey entities as well as extended API provided by Analysis module that will be implemented in the further phase of the Project.

The *Data Provider* entities have access to API functions of all microservices. In addition, a separate microservice need to be developed for the device/sensor supplied, along with a definition of specific API functions (if necessary).

5.1 Technical aspects for accessing patient data from the vINCI platform by 3rd Party entities

Patient data can be accessed by the 3rd Party using any HTTPS client software. The data download process is based on REST API communication using HTTPS GET method calls.

To ensure communication security the authentication is performed via the NGINX HTTP-proxy server (Release#1) or by blockchain service (Release#2).

The functions implemented on the vINCI platform in the REST API controller enable a number of options for downloading patient data, including:

- obtaining data for a specified period of time – an example of a request is:

  GET / users -extra/id/ medicalId / start = start_date & end = end_date – medical reports for the specified user for the time period

- obtaining data for appropriate groups of patients, different in terms of sex and age – an example of a request is:

  GET / users -extra / medicalId / fromage = start_age & toage = end_age – medical reports for the users in a given age range

Currently, the platform provides the basic set for the REST controller GET method (presented below). Depending on the needs of 3rd Parties an additional methods with different parameters (enabling data collection with detailed options) will be implemented.

The basic REST API available for 3rd Parties

GET call: Server_url / resource_path

Microservice Gateway

Entity Users

GET /users : *get all users*.
GET /users/authorities *return a string list of the all of the roles*
GET /users/login : *get the "login" user*
GET /users/aggregated: *get all the associated users' info plus their status*
GET /users/images: *get all user images*

Entity UserExtra

GET /user-extras - *get all the userExtras*
GET /user-extras/id -*get the "id" userExtra*
GET /user-alerts : -*get all the userAlerts*
GET /user-alerts/:id - *get the "id" userAlert*

Entity UserAlert
GET /user-alerts - *get all the userAlerts*
GET /user-alerts/:id -*get the "id" userAlert*

Entity Devices
GET /devices - *get all the devices*
GET /devices/type - *get all watch the devices.*
GET /my-devices -*get a list of devices corresponding to the logged user*
GET /devices/:id - *get the "id" device.*

Entity DeviceAlert
GET /device-alerts - *get all the deviceAlerts*
GET /device-alerts/:id - *get the "id" deviceAlert*

Microservice IOServer

Entity WatchData
GET /watch-data - *get all the watchData.*
GET /watch-data/count - *count all the watchData*
GET /watch-data/:id - *get the "id" watchData*

Entity ShoeData
GET /shoe-data - *get the list of data for a shoe device id*
GET /shoe-data/count - *count all the shoeData*
GET /shoe-data/:id : *get the "id" shoeData*

Entity cameraFitnessData
GET /camera-fitness-data - *get all the cameraFitnessData*
GET /camera-fitness-data/count - *count all the cameraFitnessData*
GET /camera-fitness-data/id -*get the "id" cameraFitnessData*

Entity cameraMovementData
GET /camera-movement-data - *get all the cameraMovementData*
GET /camera-movement-data/count - *count all the cameraMovementData*
GET /camera-movement-data/:id - *get the "id" cameraMovementData*

Microservice Survey
SurveyData
GET /survey-data - *get all the surveyData*
GET /survey-data/id - *get the "id" surveyData*
GET /survey-data/userExtra/{userExtraId}: *get all the surveyData for userExtraId*
GET /survey-data/userExtra/surveyId/{surveyId} - *get all the surveyData for surveyId*
GET /survey-data/userExtra/medicalId/{medicalId}- *get all the surveyData for medicalId*

5.2 Technical aspects of integration with 3rd Party data providers

A basic requirement for 3rd Party sensor vendor that wants to integrate with vINCI is to provide the sensor/device with the REST API client that is capable to send data in JSON format.

Moreover, the following implementation work is required on the vINCI platform:

- Implementation of a new "*3rdPatryKit"* JHipster microservice on the vINCI platform;
- Update of the IOServer microservice by adding a new entity for storing data provided by the *3rdPatryKit* microservice (if the new data does not match the existing entities);
- Implementation of the internal communication between microservices IOServer and *3rdPatryKit* using REST API (if the API already implemented by IOServer does not match new module functionality);
- Update of the Gateway microservice for the data presentation of the *3rdPatryKit* microservice data in the gateway dashboard (if the new data does not match the existing entities);
- Update of the Analysis module for the new *3rdPartyKit* data extraction and next proper handle by analytical logic (if the new data does not match the existing entities).

We assume that the 3rd Party will have full access to the REST API functions (methods GET, POST, PUT, DELETE) for the entity created for their microservice. This access will be based on the same security rules as for *Consumer* type 3rd Parties.

Bibliography

[D2.5] D2.5: Report on technologies integration and lab technical validation of kits. vINCI Technical Report
[D3.1] D3.1: Data Privacy Regulations and Security Requirements. vINCI Technical Report.
[D3.2] D3.3: Open data and model repository. vINCI Technical Report
[fgj] Java Fabric-proxy gateway. Webpage: https://github.com/hyperledger/fabric-gateway-java
[hfs] HyperLedger source code github repository: https://github.com/hyperledger/fabric-samples Report
[hyf] Hyperledger Fabric – open source blockchain framework. Project webpage: https://www.hyperledger.org/projects/fabric
[jwt] JSON Web Tokens. Webpage: https://www.jhipster.tech/security/#securing-jwt)

6. Annex I – HyperLedger terms definition

Orderer

Orderer (Ordering service) provides a shared communication channel to clients and peers, offering a broadcast service for messages containing transactions. It's primary goal is to provide total order for transactions published, cut blocks with ordered transactions.

Organization

Organizations are invited to join the blockchain network by a blockchain service provider. An organization is joined to a network by adding its Membership Service Provider to the network. The MSP defines how other members of the network may verify that signatures (such as those over transactions) were generated by a valid identity, issued by that organization.

Peer

A network element that hosts a ledger and runs chaincode containers in order to perform read/write operations to the ledger.

Channel

A channel is a private blockchain entity which allows for data isolation and confidentiality. A channel defines specific ledger that is shared across the peers in the channel. In order to interact with the channel transacting parties (peers, client applications) must be properly authenticated.

Smart Contract

The chaincode manages the ledger state through transactions invoked by client applications. In Hyperledger Fabric, smart contracts are referred to as chaincode. Smart contract chaincode is installed onto peer nodes and instantiated to one or more channels.

The process of placing a chaincode on a peer's file system.

- **Instantiate -** The process of starting and initializing a chaincode application on a specific channel. After instantiation, peers that have the chaincode installed can accept chaincode invocations.
- **Invoke** - Used to call chaincode functions. A client application invokes chaincode by sending a transaction proposal to a peer. The peer will execute the chaincode and return an endorsed proposal response to the client application.

Chain

The ledger's chain is a transaction log structured as hash-linked blocks of transactions. Peers receive blocks of transactions from the ordering service, mark the block's transactions as valid or invalid based

on endorsement policies and concurrency violations, and append the block to the hash chain on the peer's file system.

Block

A block contains an ordered set of transactions. It is cryptographically linked to the preceding block, and in turn it is linked to be subsequent blocks. The first block in such a chain of blocks is called the genesis block. Blocks are created by the ordering system, and validated by peers.

Transaction

Invoke or instantiate results that transaction are submitted for ordering, validation, and commit. Invokes are requests to read/write data from the ledger. Instantiate is a request to start and initialize a chaincode on a channel. Application clients gather invoke or instantiate responses from endorsing peers and package the results and endorsements into a transaction that is submitted for ordering, validation, and commit.

7. Annex II – Classes and methods of fabric-proxy sub-module.

Blockchain.java class (blockchain platform client) implements the following methods:

- ***createStorage()*** – the function builds new contract instance of the **vinci** chaincode and invokes the function *createStorageBlock ()* of the chaincode. It returns the entered record in the JSON object format, if successful .
- ***getAllStorage()–*** the function builds new contract instance of the **vinci** chaincode and invokes the function *querryAllStorageBlock()* of the chaincode. It returns the JSON array of objects with all record in the ledger, if successful .
- ***getStorage ()*** – the function builds new contract instance of the **vinci** chaincode and invokes the function *queryStorageBlock()*of the chaincode. It returns one record in the JSON object format, if successful
- ***updateStorage()*** – the function builds new contract instance of the **vinci** chaincode and invokes the function *changeStorageBlock(),* if successful.
- ***deleteStorage()*** – the function builds new contract instance of the **vinci** chaincode and invokes the function *deleteStorageBlock(),* that delete one record.

- ***createPolicy ()*** – the function builds new contract instance of the **vinci** chaincode and invokes the function *createPolicyBlock()* of the chaincode. It returns the entered record in the JSON object format, if successful .
- ***getAllPolicy()–*** the function builds new contract instance of the **vinci** chaincode and invokes the function *querryAllPolicyBlock()* of the chaincode. It returns the JSON array of objects with all record in the ledger, if successful .
- ***getPolicy()*** – the function builds new contract instance of the **vinci** chaincode and invokes the function *queryPolicyBlock()*of the chaincode. It returns one record in the JSON object format, if successful
- ***updatePolicy()*** – the function builds new contract instance of the **vinci** chaincode and invokes the function *changePolicyBlock(),* if successful.
- ***deletePolicy ()*** – the function builds new contract instance of the **vinci** chaincode and invokes the function *deletePolicyBlock(),* that delete one record.

StorageMapper.java class (mapping of data structures inside of fabric-proxy) implements the following methods:

- ***getAllStorageMap()*** – maps the JSON array of the returned records from ledger to the microservices ArrayList< **BlockchainStorageDTO** >
- ***postStorageMap()*** – maps the JSON object of the record entered to the ledger to the microservices **BlockchainStorageDTO** structure

- ***getStorageMap() -*** maps the JSON object of the record read from the ledger to the microservices **BlockchainStorageDTO** structure
- ***putStorageMap() -*** maps the JSON object of the record updated in the ledger to the microservices **BlockchainStorageDTO** structure

PolicyMapper.java class (mapping of data structures inside of fabric-proxy) implements the following methods:

- ***getAllPolicyMap()*** – maps the JSON array of the returned records from ledger to the microservices ArrayList< **BlockchainStorageDTO** >
- ***postPolicyMap()*** – maps the JSON object of the record entered to the ledger to the microservices **BlockchainStorageDTO** structure
- ***getPolicyMap() -*** maps the JSON object of the record read from the ledger to the microservices **BlockchainStorageDTO** structure
- ***putPolicyMap() -*** maps the JSON object of the record updated in the ledger to the microservices **BlockchainStorageDTO** structure

Design of vINICI chaincode covers two data structures (smartcontracts) that are saved as transaction blocks. They are as follows:

SmartContract structure

```
type Storage struct {
      TransactionID       string `json:"transactionid "`
      UserID       string `json:"userid"`
      MedicalID            string `json:"medicalid "`
      Timestamp       string `json:"timestamp"`
}
```

SmartContract structure

```
type Policy struct {
      TransactionID       string `json:"transactionid "`
      OrganizationID      string  ‘json””organiztionid”
      Org_signature ;     string `json:"orgsignature"`
      Users_ signature;   string `json:"usersignature “
      Access_key;       string `json:"access_key "
      Timestamp       string `json:"timestamp"`
}
```

8. Annex III - Blockchain platform validation tests results

Integration test

Objective

The purpose of the test is to check the correctness of data (records stored in a ledger of the blockchain platform) imported to the vINCI gateway front-end level.

Test procedure

The test procedure is as follows:

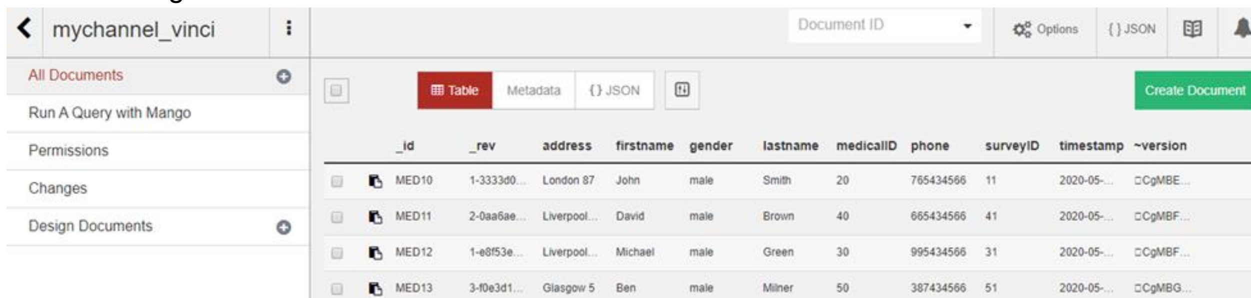
- Data initialization in the blockchain platform by calling the *initLedger ()* function in the blockchain network from the CLI docker container.
- Running the front-end gateway API from the administrator account and checking the data in the UserBlockchain entity.

Expected results

It is expected that the data presented front-end gateway are consistent with blockchain platform data (stored in CouchDB).

Test results

The Figure 13 and Figure 14 shows that the data were imported correctly . The records stored in the blockchain database (couchDB) as key-value have id - key in the form of a string, which is stored in objects of the vINCI platform as a numeric value. The value after the MED string is mapped, e.g. MED12 string to the number 12.



_id	_rev	address	firstname	gender	lastname	medicalID	phone	surveyID	timestamp	~version
MED10	1-3333d0...	London 87	John	male	Smith	20	765434566	11	2020-05-...	□CgMBE...
MED11	2-0aa6ae...	Liverpool...	David	male	Brown	40	665434566	41	2020-05-...	□CgMBF...
MED12	1-e8f53e...	Liverpool...	Michael	male	Green	30	995434566	31	2020-05-...	□CgMBF...
MED13	3-f0e3d1...	Glasgow 5	Ben	male	Milner	50	387434566	51	2020-05-...	□CgMBG...

Figure 13 Blockchain platform CouchDB database data

Vinci | Home | Entities | Administration | English | Account

User Blockchains

+ Create a new User Blockchain

ID	First Name	Last Name	Gender	Phone	Address	Medical ID	Survey ID	Timestamp	
10	John	Smith	male	765434566	London 87	20	11	18/05/20 16:56	View Edit Delete
11	David	Brown	male	665434566	Liverpool 25	40	41	18/05/20 18:56	View Edit Delete
12	Michael	Green	male	995434566	Liverpool 87	30	31	19/05/20 18:56	View Edit Delete
13	Ben	Milner	male	387434566	Glasgow 5	50	51	16/05/20 18:56	View Edit Delete

Figure 14 Data in gateway front-end (entity UserBlockchain)

REST API method tests

The purpose of the test is to verify the basic API functions in an integrated 3-module environment.

Test case 1 - GET all blockchain users

Tested method -GET/user-blockchains

Expected results

1. HTTPResponse with table of JSON object of all users.
2. Correct logs only from endorsing peers - the GET method called in the above scenario is mapped to the *queryAllMed()* method of the **fabric-proxy** module, which calls a query to the blockchain platform without creating a new transaction block. For this reason, only confirmation by endorsing peers without validation logs are expected (there is no validation process).

Test results

The body of the HTTPResponse is compliant with the user's records saved on the blockchain platform:

[{"id":10,"firstName":"John","lastName":"Smith","gender":"male","phone":"765434566","address":"London 87","medicalID":20,"surveyID":11,"timestamp":"2020-05-18T14:56:06.157Z"},{"id":11,"firstName":"David","lastName":"Brown","gender":"male","phone":"665434566","address":"Liverpool 25","medicalID":40,"surveyID":41,"timestamp":"2020-05-18T16:56:06.157Z"},{"id":12,"firstName":"Michael","lastName":"Green","gender":"male","phone":"995434566","address":"Liverpool 87","medicalID":30,"surveyID":31,"timestamp":"2020-05-19T16:56:06.157Z"},{"id":13,"firstName":"Ben","lastName":"Milner","gender":"male","phone":"387434566","address":"Glasgow 5","medicalID":50,"surveyID":51,"timestamp":"2020-05-16T16:56:06.157Z"}]

The Figure 15 shows correct logs from the endorsing peer - peer0.org1.example.com that informs about processing the chaincode **vinci**. The same logs were registered in the endorsing peer0.org2.example.com representing organization 2 of the network.

```
2020-05-25 13:16:14.724 UTC [endorser] callChaincode -> INFO 4a2 [mychannel][da665a74] Entry chaincode: name:"vinci"
2020-05-25 13:16:14.785 UTC [endorser] callChaincode -> INFO 4a3 [mychannel][da665a74] Exit chaincode: name:"vinci"  (60ms)
2020-05-25 13:16:14.785 UTC [comm.grpc.server] 1 -> INFO 4a4 unary call completed grpc.service=protos.Endorser grpc.method=ProcessProposal grpc.peer_address=172.26.
1:39192 grpc.code=OK grpc.call_duration=62.17826ms
```

Figure 15 Logs from peer0.org1.example.com

Test case 2 - GET the specific user method test

Tested method - GET/user-blockchains/13

Expected results

1. HTTPResponse with an JSON object of the user with specific *id.*
2. Correct logs from endorsing peers - The GET method called in the above scenario is mapped to the *queryMed()* method of the **fabric-proxy** module.

Test results

The body of the HTTPResponse is compliant with the user's record saved on the blockchain platform:

{"id":13,"firstName":"Ben","lastName":"Milner","gender":"male","phone":"387434566","address":"Glasgow 5","medicalID":50,"surveyID":51,"timestamp":"2020-05-16T18:56:06.157Z"}

The logs from endorsing peers: *peer0.org1.example.com* and *peer0.org2.example.com* indicate correct information about vinci chaincode processing.
Moreover, the correct result of the same method call from the gateway front-end (Figure 16)

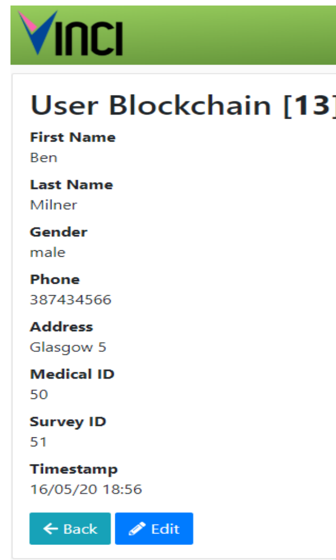


Figure 16 The result of the GET method received in the gateway front-end

Test case 2 – POST method –create new user

Tested method – POST /user-blockchains/

Expected results

1. HTTPResponse with an JSON object of the user with specific *id.*

2. Correct logs from endorsing and validating peers. In this test scenario the transaction was performed. For this reason, appropriate logs are expected from both endorsing peers and validating peers informing about adding a transaction block.

Test results:

HTTPResponse

As a result, a message was received containing a JSON object compatible with the generated JSON object:

{"id":14,"firstName":"Adam","lastName":"Clark","gender":"male","phone":"587434566","address":"Dover 57","medicalID":60,"surveyID":61,"timestamp":"2020-05-19T16:56:06.157Z"}.

The Figure 17 shows correct logs from the endorsing peer - peer0.org1.example.com that informs about processing the chaincode vinci and validation on the new block. The peer1.org1.example.com signalized also correct validation logs (Figure 18) . As a result, a new block (27) was created and saved in blockchains of all the peers in the network.

```
[mychannel][d555de93] Entry chaincode: name:"vinci"
[mychannel][d555de93] Exit chaincode: name:"vinci"  (4ms)
unary call completed grpc.service=protos.Endorser grpc.method=ProcessProposal grpc.peer_address=172.26.0.
1:40396 grpc.code=OK grpc.call_duration=6.891199ms
[mychannel] Received block [27] from buffer
[mychannel] Validated block [27] in 11ms
[mychannel] Committed block [27] with 1 transaction(s) in 299ms (state_validation=97ms block_and_
pvtdata_commit=19ms state_commit=174ms) commitHash=[65a2e5675f6d5c11a319c6446cf25d1506ad3f7aa5c4793aab6b91052cd510f3]
[mychannel][d9979008] Entry chaincode: name:"vinci"
[mychannel][d9979008] Exit chaincode: name:"vinci"  (5ms)
unary call completed grpc.service=protos.Endorser grpc.method=ProcessProposal grpc.peer_address=172.26.0.
1:40396 grpc.code=OK grpc.call_duration=6.287866ms
```

Figure 17 Validation and processing logs from the endorsing peer - peer0.org1.example.com

```
2020-05-25 19:52:37.865 UTC [gossip.privdata] StoreBlock -> INFO 122 [mychannel] Received block [27] from buffer
2020-05-25 19:52:37.867 UTC [committer.txvalidator] Validate -> INFO 123 [mychannel] Validated block [27] in 1ms
2020-05-25 19:52:38.073 UTC [kvledger] CommitWithPvtData -> INFO 124 [mychannel] Committed block [27] with 1 transaction(s) in 205ms (state_validation=6ms block_and_p
vtdata_commit=15ms state_commit=180ms) commitHash=[65a2e5675f6d5c11a319c6446cf25d1506ad3f7aa5c4793aab6b91052cd510f3]
```

Figure 18 Validation logs from the– peer1.org1.example.com

A new record (*id=14)* has been added to the database(Figure 19) and is noticed in the front-end gateway(Figure 20).

User Blockchains

+ Create a new User Blockchain

ID	First Name	Last Name	Gender	Phone	Address	Medical ID	Survey ID	Timestamp	
10	John	Smith	male	765434566	London 87	20	11	18/05/20 16:56	View Edit Delete
11	David	Brown	male	665434566	Liverpool 25	40	41	18/05/20 18:56	View Edit Delete
12	Michael	Green	male	995434566	Liverpool 87	30	31	19/05/20 18:56	View Edit Delete
13	Ben	Milner	male	387434566	Glasgow 5	50	51	16/05/20 18:56	View Edit Delete
14	Adam	Clark	male	587434566	Dover 57	60	61	19/05/20 18:56	View Edit Delete

Figure 19Information about new record in the gateway front-end

Table | Metadata | {} JSON | Create Document

_id	_rev	address	firstname	gender	lastname	medicalID	phone	surveyID	timestamp	~version
MED10	1-3333d0...	London 87	John	male	Smith	20	765434566	11	2020-05-...	□CgMBE...
MED11	2-0aa6ae...	Liverpool...	David	male	Brown	40	665434566	41	2020-05-...	□CgMBF...
MED12	1-e8f53e...	Liverpool...	Michael	male	Green	30	995434566	31	2020-05-...	□CgMBF...
MED13	3-f0e3d1...	Glasgow 5	Ben	male	Milner	50	387434566	51	2020-05-...	□CgMBG...
MED14	3-f3e932...	Dover 57	Adam	male	Clark	60	587434566	61	2020-05-...	□CgMBG...

Figure 20 Information about new record in the blockchain database

Test case 4 – PUT method –update medicalID

Tested method – PUT /user-blockchains/

Expected results

1. HTTPResponse - just like for the POST method, with the medicalID field updated.
2. Correct logs from endorsing and validating peers informing about adding a transaction block.

Test results

HTTPResponse

{"id":14,"firstName":"Adam","lastName":"Clark","gender":"male","phone":"587434566","address":"Dover 57","medicalID":100,"surveyID":61,"timestamp":"2020-05-19T16:56:06.157Z"}

The endorsing peer - peer0.org1.example.com that informs about processing the chaincode vinci and validation on the new block -28 (Figure 21). In the case of validating peer1.org1.example.com correct validation block logs were received (Figure 22) .

```
2020-05-25 20:52:09.235 UTC [endorser] callChaincode -> INFO 4d4 [mychannel][dd18d030] Entry chaincode: name:"vinci"
2020-05-25 20:52:09.279 UTC [endorser] callChaincode -> INFO 4d5 [mychannel][dd18d030] Exit chaincode: name:"vinci"  (44ms)
2020-05-25 20:52:09.280 UTC [comm.grpc.server] 1 -> INFO 4d6 unary call completed grpc.service=protos.Endorser grpc.method=ProcessProposal grpc.peer_address=172.26.0.
1:40662 grpc.code=OK grpc.call_duration=45.43975ms
2020-05-25 20:52:11.329 UTC [gossip.privdata] StoreBlock -> INFO 4d7 [mychannel] Received block [28] from buffer
2020-05-25 20:52:11.332 UTC [committer.txvalidator] Validate -> INFO 4d8 [mychannel] Validated block [28] in 2ms
2020-05-25 20:52:11.661 UTC [kvledger] CommitWithPvtData -> INFO 4d9 [mychannel] Committed block [28] with 1 transaction(s) in 328ms (state_validation=104ms block_and
_pvtdata_commit=17ms state_commit=201ms) commitHash=[18c36c28ac181fda7c4256f10f527b160b8bfd30a0c718874b7e5d933cac6ab3]
```

Figure 21 Processing logs from the peer peer0.org1.example.com

```
2020-05-25 20:52:11.345 UTC [gossip.privdata] StoreBlock -> INFO 128 [mychannel] Received block [28] from buffer
2020-05-25 20:52:11.347 UTC [committer.txvalidator] Validate -> INFO 129 [mychannel] Validated block [28] in 1ms
2020-05-25 20:52:11.691 UTC [kvledger] CommitWithPvtData -> INFO 12a [mychannel] Committed block [28] with 1 transaction(s) in 343ms (state_validation=130ms block_and
_pvtdata_commit=38ms state_commit=171ms) commitHash=[18c36c28ac181fda7c4256f10f527b160b8bfd30a0c718874b7e5d933cac6ab3]
```

Figure 22 Validation logs from the peer1.org1.example.com

The Figure 23 shows updated value of medicalID field in the record 14 noticed in the front-end gateway and blockchain CouchDB.

User Blockchain [14]

First Name
Adam

Last Name
Clark

Gender
male

Phone
587434566

Address
Dover 57

Medical ID
100

Survey ID
61

Timestamp
19/05/20 18:56

mychannel_vinci > MED14

Save Changes Cancel


```
{
  "_id": "MED14",
  "_rev": "4-8d3d99f8705b64c4cf6f3283c3601d0b",
  "address": "Dover 57",
  "firstname": "Adam",
  "gender": "male",
  "lastname": "Clark",
  "medicalID": "100",
  "phone": "587434566",
  "surveyID": "61",
  "timestamp": "2020-05-19T18:56:06.157Z",
  "~version": "\u0000CgMBHAA="
}
```


Figure 23 The fields of the record 14 noticed in the front-end gateway and blockchain CouchDB after the PUT method

9. Annex IV - Installation/configuration guides of the vINCI platform

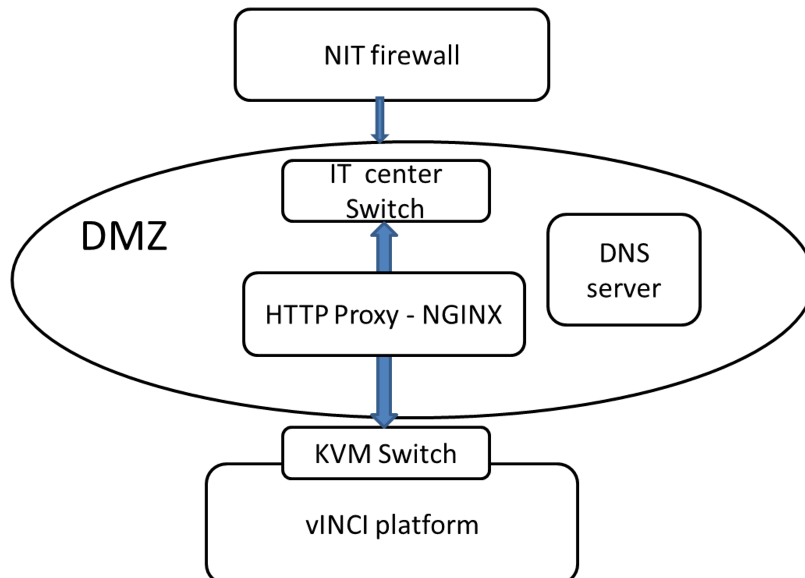
9.1 Installation of the vINCI platform on NIT infrastructure

The document describes Installation/configuration guides of the vINCI platform, based on deployment in NIT.

NIT's deployment includes:

1. Network infrastructure (firewall, switches, DNS)
2. HTTP proxy – NGINX server (version 1.14.0)
3. vINCI platform
   - modules implemented by ICI
   - database – PostgreSQL (version 10.14)

The following figure shows the overall hardware architecture of the vINCI platform located in the NIT (for development/testing purposes, platform modules and the database can be launched on the same virtual machine).



9.2 Network infrastructure

Note: these modules are needed to make the platform available from the Internet – they are not necessary for internal tests

NIT Firewall (maintained by the NIT IT center)

The basic firewall configuration includes:

- Configuration the external public IP address for the vINCI platform.
- Transition (NAT) the local address of vINCI platform to the public IP address.
- Limiting the ports exposed externally (only ports for ssh -22 and https -443 services are opened)

DMZ (maintained by the NIT IT center)

The DMZ Includes the following:

- Switch network configured into one common VLAN with KVM switch installed on the vINCI server platform
- DNS server - Domain names of the vINCI platform entered into the DNS are:
  - www.vinci.il-pib.pl
  - vinci.il-pib.pl
  - www.vinci.itl.waw.pl
  - vinci.itl.waw.pl

9.3 HTTP proxy – NGINX server

The NGINX server is responsible for the following security functions:

- SSL termination and proxy to the vINCI platform gateway module
- Maintaining certificates for HTTPS operation for domain names
- Level1: User Authentication based on the private key (SSL certificates)
- Level2: User Authentication based on login/password

Appendix 2 describes the data required to generate client SSL certificates.

For development work/testing it is suggested to limit authentication only to login/password option (level2).

NGINX Installation and configuration at NIT:

Ubuntu Nginx installation:

sudo apt update
sudo apt install nginx

Check the installation status:

systemctl status nginx

Nginx configuration: configuration file - */etc/nginx/site-available/default*

```
upstream vinci_platform {
   server 193.110.137.63:8080;
      }
```

```
server {
   listen          443 ssl default_server;
   listen          [::]:443 ssl default_server;

   server_name     vinci.il-pib.pl;
 #   access_log      /var/log/nginx/example.com_access.log combined;
 #   error_log       /var/log/nginx/example.com_error.log error;

   ssl_certificate         /etc/ssl/vinci/vinci.il-pib.pl.pem;
   ssl_certificate_key     /etc/ssl/vinci/vinci.il-pib.pl.key;
   ssl_client_certificate  /etc/ssl/auth/ca.pem;
   ssl_verify_client on;


   location / {
      auth_basic "Restricted";
      auth_basic_user_file /etc/nginx/.htpasswd;
     # proxy_redirect     off;
      proxy_set_header X-Forwarded-User $remote_user;
      proxy_set_header   Host              $http_host;
      proxy_set_header   X-Real-IP         $remote_addr;
      proxy_set_header   X-Forwarded-For   $proxy_add_x_forwarded_for;
      proxy_set_header   Authorization     "";
      proxy_pass         http://vinci_platform;
}

}
```

Starting in service mode command

nginx service restart

9.4 vINCI server platform

The vINCI platform (**software modules developed by ICI** + PostgreSQL database) has been installed on servers with Linux Ubuntu 18.04 and KVM virtualization system.

The vINCI platform can be run as a single virtual machine with the following characteristics:

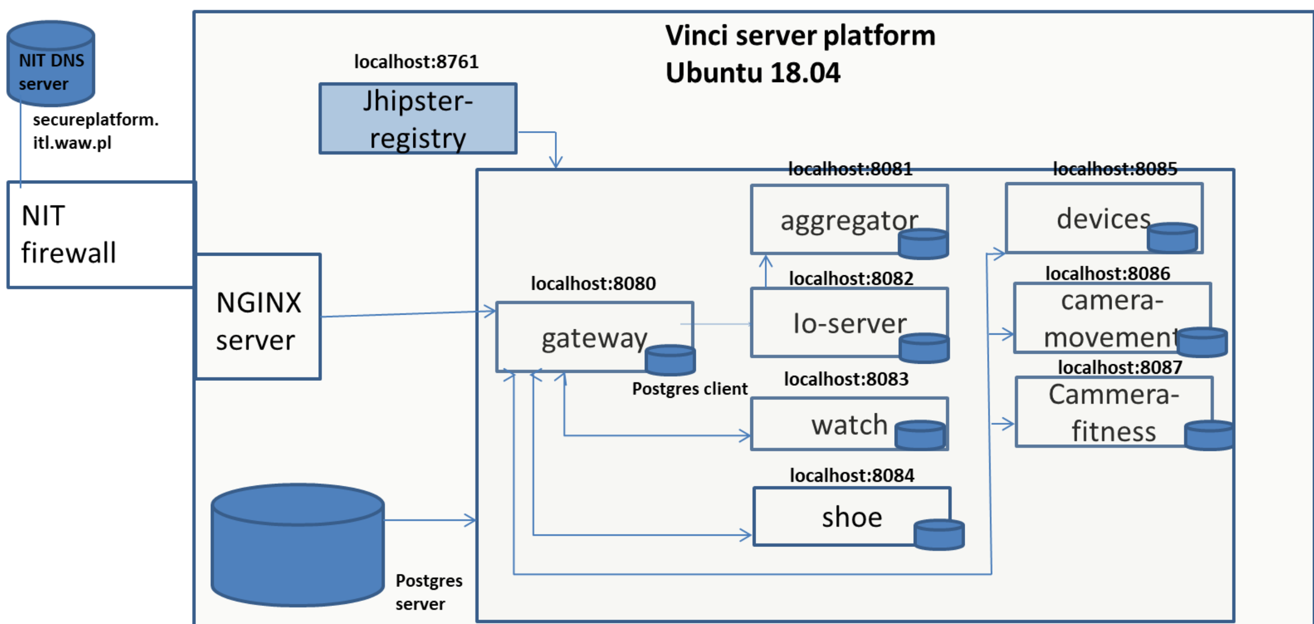
- disk storage size - 40 GB
- memory size - 32 GB
- operating system - Linux Ubuntu 18.04 Bionic

Other required software

- Java – (Java version in NIT - OpenJDK Runtime Environment (build 1.8.0_242-8u242-b08-0ubuntu3 ~ 18.04-b08)
- Node (Node version in NIT - v8.17.0)
- PostgreSQL – (PostgreSQL version in NIT 10.14)
- Docker - Docker (version in NIT - 19.03.0)
- The jhipster-registry - (https://www.jhipster.tech/jhipster-registry/)

The figure below shows the NIT's vINCI platform. The modules developed by ICI are available at: https://gitlab.com/vinci-aal

At the moment, the following ICI modules (microservices) have been installed at NIT testbed: camera-movement, io-server, aggregator, survey, camera-fitness, shoe, watch, devices, gateway



Installation and launch of jhipster-registry

Clone source code from the GitHub jhipster / jhipster-registry repository (https://github.com/jhipster/jhipster-registry).

Running

./mvnw (development mode)

./mvnw -Pprod package (production mode)

docker-compose -f src/main/docker/jhipster-registry.yml up (as docker container)

Installing the PostgreSQL database

sudo apt update
sudo apt install postgresql postgresql-contrib

For each of the above modules, it is required to create a user (with the password) and a PostgreSQL database.

Configuration of PostgreSQL database (an example for watch microservice/module)

Step # 1: Add a Linux / UNIX user called watch
commands:
adduser watch
passwd watch
Step # 2: Becoming a superuser and connect to database server
Commands:
# su - postgres
$ psql template1

Step # 3: Add a user called watch / password watch
command
template1 = # CREATE USER watch WITH PASSWORD 'watch';

Step # 4: Add a database called watch
commands:
template1 = # CREATE DATABASE watch;
Now grant all privileges on database
template1 = # GRANT ALL PRIVILEGES ON DATABASE watch to watch;

Step # 6: Test watch user login
In order to login as *watch* you need to type following commands:
$ su - watch
$ psql -d watch -U watch

Configuration of PostgreSQL authentication
Configuration of **pg_hba.conf** file that enables client authentication between the PostgreSQL server and the client application:
In Ubuntu system the file location is: /etc/postgresql/10/main/pg_hba.conf
1.Modification - change the method to md5 for all users
2.Restart the postgresql service

ICI modules installation and configuration

1. Download the source codes of the modules from vinci repository https://gitlab.com/vinci-aal/
2. Information about installation procedure can be found in the README.md file in the main directory of each module. If the module uses the PostgreSQL database, then you must run PostgreSQL as the docker container machine.

Basic configuration: settings of the database and port number (the same for all modules)

For the production mode:

File /src/main/resources/config/aplication-prod.yml

```
spring:
  datasource:
        (settings database name, user and password)
server:
   port:
        (port number on which the service is running)
```

Configuration of data access - security.

It allows you to unblock access to data without user authentication

directory - *src/main/java/default package folder/config/SecurityConfiguration.java*

configure function – modification rules of HTTPRequest

Installation/boot using gradle

The following sequence for starting modules is required

1. jhipster-registry
2. gateway
3. other modules

Running individual modules is possible in 3 modes:

1. (development mode)

npm install (only for gateway module)

./gradlew

2.(production mode)

./gradlew –Pprod clean bootWar

java -jar build/libs/*.war – command to run in production mode

3.(docker container)

./gradlew bootWar -Pprod jibDockerBuild

docker-compose -f src/main/docker/app.yml up –d

9.5 Access to the vINCI platform

In the NIT configuration, access to the platform is possible using the HTTPS protocol, and it is provided through the NGINX proxy server, which:

1. requires client certificates (user.pem, user.key)
2. requires entering username and password

and next removes authentication header and redirects requests to the gateway module.

The data for generating client certificates is as follows:

- Country Name (2 letter code) []
- State or Province Name (full name) []
- Locality Name (eg, city) []:
- Organization Name (eg, company) [Internet Widgits Pty Ltd]
- Organizational Unit Name (eg, section) []
- Common Name (e.g. server FQDN or YOUR name) []
- Email Address []:

Logins and passwords should be configured after issuing certificates.

Simple test script in Python (GET and POST request to the platform) to test authentication mechanism:

```
#!/usr/bin/env python3
import requests
import json

with open("watch_record.json", "r") as read_file:
   data = json.load(read_file)
jdata=json.dumps(data)
headers = {'Accept' : 'application/json', 'Content-Type' : 'application/json'}

r = requests.post('https://vinci.il-pib.pl/watch/api/records', data=jdata, headers=headers,
cert=('user.pem','user.key'), verify=True,  auth=('login','password'))
url = r.text
p = requests.get('https://vinci.il-pib.pl/api/users',cert=('user.pem',user.key'), verify= True,
auth=('login','password))
```