Project Acronym:	NUTRISHIELD
Grant Agreement number:	818110 (H2020-SFS-2018-IA)
Project Full Title:	Fact-based personalised nutrition for the young

DELIVERABLE

D6.1 – Specifications of NUTRISHIELD software framework

Dissemination level	PU – Public
Type of Document	Report
Contractual date of delivery	30/10/2019
Deliverable Leader	INTRASOFT INTL (INTRA)
Status & version	Final
WP responsible	WP6 (INTRA)
Keywords:	

Deliverable Leader:	INTRA
Contributors:	Ioannis Daskalopoulos (INTRA), Stylianos Georgoulas (INTRA), Spyros Evangelatos (INTRA), Tasos Karydis (VER), Jennifer Karrer (QRT), Eirini Bathrellou (HUA), Geert Egghe (SWB), Fady Mohareb (CU), Arno Simon (ARGOS)
Reviewers:	ALPES, VER
Approved by:	ARGOS

Document History

Version	Date	Contributor(s)	Description
v0.1	20/09/2019	INTRA	Initial ToC and Draft version
v0.2	07/10/2019	INTRA, ARGOS, CU, VER, QRT	Further material and initial partner contributions
v0.3	18/10/2019	INTRA, SWB, HUA, VER, ALPES	Enhanced material
v0.4	25/10/2019	INTRA	Internal Review version
V0.5	31/10/2019	INTRA	Final version

Executive Summary

This report presents an overview of the technical components and the associated integration requirements, the fulfilment of which will result in a successful integration of the former into a unifying platform. Furthermore, details on the specifications of the components are provided along with specifications regarding interactions with other parts of the platform. In addition to the above, the technologies and approaches to be employed for the development of the NUTRISHIELD platform components are listed in the integration design section. The revised integration roadmap is finally presented, intensifying the consortium's efforts towards the successful demonstration of some of the platform's parts at a very early stage, providing confidence for the rapid uptake of the final form of the platform while ensuring adequate maturity of its offered functionality.

Table of Contents

Table of Figures

Definitions, Acronyms and Abbreviations

Acronym	Title
API	Application Programming Interface
CD	Continuous Delivery/Deployment
CI	Continuous Integration
CRUD	Create, Read, Update and Delete
DoA	Description of Action
FFQ	Food Frequency Questionnaire
FTIR	Fourier transform infrared (spectroscopy)
HDM	Human Donor Milk
HMAC	Hash Based Message Authentication
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
NoSQL	Not Only Structured Query Language
POPD	Protection of Personal Data
QCL	Quantum Cascade Laser
R	The R programming language
REST	Representational State Transfer
SIBO	Small Intestinal Bacterial Overgrowth
SQL	Structured Query Language
SSH	Secure Shell
SSHFS	SSH Filesystem
SSL	Secure Sockets Layer
STOMP	Simple Text Oriented Messaging Protocol
TDD	Test-Driven Development
TLS	Transport Layer Security
T2D	Type 2 Diabetes
URL	Uniform Resource Locator
VDP	Vertoyo Digital Platform
VCS	Version Control System
WP	Work Package
XML	eXtensible Markup Language

1. Introduction

This deliverable presents the core components of the NUTRISHIELD platform paving the way towards the successful system integration. It connects the Dashboard and Mobile app, Nutrition Algorithm, Databases and Machine Learning tools and the services that will be provided by the different analysers developed and implemented within WP4, 5 and 6 in a single point of reference, that is the NUTRISHIELD platform. The architectural approaches and technologies that will be used for the integration of the platform components are also presented followed by the specifications of the platform integration approach which follows the continuous integration/continuous development (CI/CD) paradigm, showcasing the flexibility that characterises the end-resulting solution. The NUTRISHIELD platform that will be developed following the guidelines within this document will be validated in terms of technology efficiency and methodology accuracy in the forthcoming clinical studies, further providing feedback that will iteratively fine tune the offered functionality.

The specific document is also applicable to a non-technical audience since a high-level overview of the components that are part of the architecture are also described at the beginning of the deliverable (Section 2).

1.1. Deliverable Overview and Report Structure

The chapters in this deliverable are organized as follows:

- Chapter 1 provides an introduction and overview of this report and its structure.
- Chapter 2 provides a high-level overview of each of the NUTRISHIELD components and how they fit in the overall architectural diagram. In addition, all the integration requirements together with the components' technical specifications are described in detail showcasing how these diverse entities can be connected and interact efficiently with each other.
- Chapter 3 describes the Microservices architectural approach and the detailed integration design followed by the CI/CD techniques used for the system development and deployment. The document concludes with the platform integration roadmap and the next steps towards the final version of the NUTRISHIELD platform.

2. The NUTRISHIELD Platform

In this chapter the latest high-level architecture is presented together with a brief description of the individual functional components the purpose of which is to seamlessly interoperate in order to provide the entire NUTRISHIELD Platform functionality.

2.1. Overview of NUTRISHIELD components

This section presents the architecture of the NUTRISHIELD platform which is depicted in Figure 1 below. The diagram represents the latest system architecture which is based on the users, components and applications identified in the course of the Project. At a high-level view, the NUTRISHIELD system consists of four major components, namely the Measuring Devices, the Dashboard (a web application), the Mobile Application and the Backend System. The Backend System is a multi-functional component including many other sub-components such as databases, web services, the nutrition algorithm service and machine learning modules. It interacts with both the dashboard and the mobile application to provide the necessary support to their functionality. It should be mentioned that the Measuring Devices are used in clinical or lab settings and may not have direct connectivity to the rest of the system. These devices are therefore not directly connected to the NUTRISHIELD platform. Respective measurements and analysis output originating from these devices is planned to be inserted to NUTRISHIELD by using the dedicated data entry forms which associate the input with the individuals it concerns.

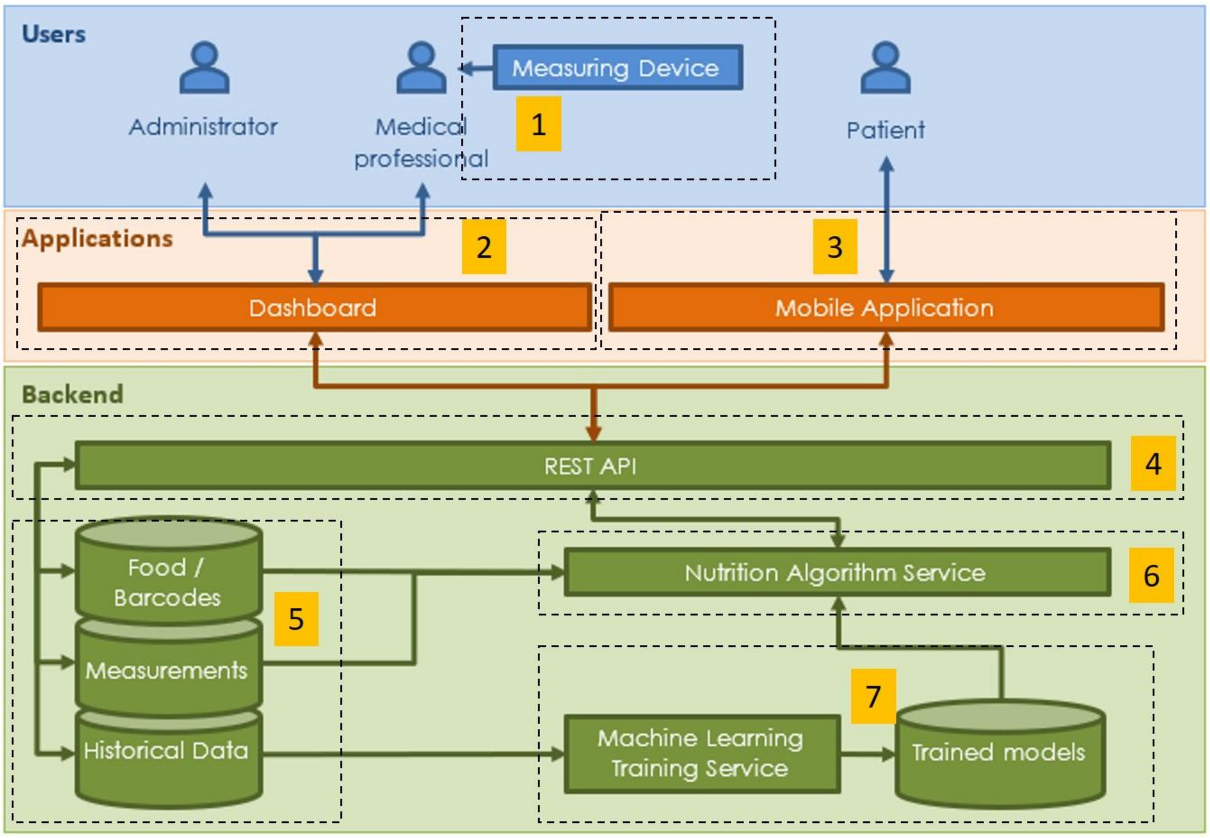


Figure 1: Architecture of the NUTRISHIELD platform

The **Measuring Devices (1)** consist of three prototypes for *urine*, *human milk and breath* analysis. The operator, measuring with the prototypes, will measure as instructed and will add the information from the prototypes to the NUTRISHIELD platform. Beside the concentration values for each analyzer, several additional info can be stored on the device, including measurement ID, Date & Time, Operator, Patient-ID/Sample ID, Absorbance and Status (Normal, Error, etc.).

The **Dashboard (2)** is a web application used primarily by medical personnel to monitor patients, upload measurements and prepare the dietary and activity plans. It interacts directly with the backend over the internet. Dashboard component will act as the main web application which will be accessible by Administrators and Medical professionals. Dashboard component will be the gateway to NUTRISHIELD Platform for Administrators (Data Managers and NUTRISHIELD Admins) and Medical professionals (Doctors & Lab Technicians). Each type of user will be mapped into a distinct Role with certain level of functionality privileges and level of access on the NUTRISHIELD Data. Dashboard will be available to registered users only as a Web Portal/App and will support functionalities summarized in the following list:

- **NUTRISHIELD Admins**: CRUD operations on Medical professionals' user accounts.
- **Data Managers**: CRUD operations on Food entries.
- **Medical professionals**: CRUD operations on Patient user accounts and Patients' measurements
- **Doctors**: CRUD operations on nutrition/activity suggestions, nutrition/activity plans and nutrition/activity diaries. Ability to send/schedule notifications towards patients' mobile app.

The NUTRISHIELD **Mobile Application (3)** is used by patients/users for receiving notifications regarding their dietary and activity plan and logging the food consumed towards keeping a food journal. Running on a smartphone which must be connected to the internet, it interacts directly with the backend, having a camera and enough storage to store meal consumption images and other information.

The **REST API (4)** represents a collection of RESTful API endpoints used by the dashboard and the mobile application. These endpoints can be used to transfer data and commands between the applications and various backend subcomponents. The logical layer of Rest APIs should be accommodated by a component that can provide web service features such as better governance on the system integration among NUTRISHIELD components and allow the integration of heterogeneous systems that cannot be possibly integrated directly. VDP REST API will be utilized as Middleware in order to allow integration of the following flows of Data:

- Mobile App to Databases (read/write operations)
- Dashboard to Nutrition Algorithm and vice-versa
- Dashboard to any external service (such as Push Notification Service)

For the data persistence needs of the NUTRISHIELD platform both relational SQL **Databases (5)** (e.g. PostgreSQL) and NoSQL (MongoDB) approaches will be employed according to needs. Depending on the data and the related pre-defined or dynamic schemas where table-based or document-based approaches are necessary the appropriate database instance will be used. Dashboard Application along with VDP Rest API require the presence of an SQL Database to hold the out of the box Database entities (Tables, Views, Procedures, etc.). The specific DB schemas will have to be enhanced in order to hold customized entities where NUTRISHIELD data will be stored (such as patients' measurements, nutrition personalized plans, etc.).

The **Nutrition Algorithm Service (6)** component represents the service that implements and makes available the NUTRISHIELD nutrition algorithm to the rest of the system. By leveraging relevant data maintained in databases and by receiving input and triggers from the Dashboard it produces personalised nutrition suggestions for patients. This component does not maintain any internal state but rather receives the necessary input to execute the algorithm and return the output. Communication with the databases to retrieve relevant data (measurements) coupled with other input provided by the Dashboard are used as parameters in the algorithm execution. A request for a nutrition suggestion is performed via a well-defined RESTful API that this component exposes. Endpoints of this API are used for both triggering the algorithm and providing the necessary input for its execution, as well as the retrieval of the resulting nutritional suggestion.

2.2. Components details and integration requirements

2.2.1. Measuring Devices

The Measuring Devices of the NUTRISHIELD Platform are the Human Milk, Urine and Breath analyzers which represent one of the main sources of patient related measurements of appropriate analytes. The target analytes observed for the Human Milk Analyzer are the proteins casein, α-lactalbumin and lactoferrin as well as the total protein content. Regarding the analytes measured by the Urine Analyzer, these are phosphate and creatinine. The focus of detection for the Breath analyzer in this project is on Hydrogen Cyanide (HCN) and Methane (CH4). The Urine analyzer will measure phosphate and creatinine in urine in g/L and the human milk analyzer will measure total protein, lactoferrin, casein and α-lactalbumin in mg/mL. Besides the concentration values the following additional info could be an option to be stored on the device, however, the decision which information will be displayed on the prototype will be made later on in the process:

- Measurement ID
- Date & Time
- Operator
- Patient-ID/Sample ID
- Absorbance
- Status (Normal, Error…)

The Breath analyzer in turn, will analyze the concentration of methane, hydrogen cyanide and hydrogen in breath. The analysis data will be displayed on the screen of the analyzer and stored in the device. The data stored will include:

- Patient-ID (provided by medical personnel)
- Date and Time of measurement
- Measurement ID
- Concentration of Methane in ppm (e.g. 2,7)
- Concentration of Hydrogen Cyanide (e.g. 0,2)
- Concentration of Hydrogen (e.g. 0,5)

The data items above as well as their digital representation are subject to change in case there is a need to adapt to a specific format as the projects progresses. It should be noted that the main focus of WP4 lies with the development, performance and accuracy of the prototypes for Urine, Human Milk and Breath analysis. It has been decided among partners that initially the input of measurement data that originate from the analyzers should occur by using dedicated dashboard input forms made available to the laboratory staff. Therefore, the operator measuring with the prototypes, will measure as instructed and will proceed to enter the results of the measurement to the platform using the dashboard. At the time of writing, further analysis and information from WP4 as it is described in "D4.1 Report on chemical analysis methods" indicates that the prototype devices may potentially have network connectivity that allows the measurement data to be sent in an automatic manner. It should be made clear that although this lies outside of the main goals of WP4, the possibility of implementing an interface for the automatic transfer of the measured data will be investigated and may be considered. If applicable, the transmission of the data from the devices will follow the integration approaches that are applied throughout the NUTRISHIELD Platform as they are described in this document.

2.2.2. Dashboard

Dashboard component will be implemented by Vertoyo Digital Platform (VDP). VDP is a Java-Based Platform that can be utilized to support complex Web Applications and can offer out-of-the-box support for most of the required Dashboard features such as User Roles/Privileges, Entities Management using Web Forms, Entities Listing using Datatables. VDP comes with advanced functionalities of supporting BPM Workflows, Tasks Scheduling and can be esily interconnected with Push/Email Notifications Services as well as with any 3^rd^-party system by exposing or consuming REST or SOAP APIs. Considering that VDP is quite efficient on system integration, it is suggested that VDP REST API to encapsulate REST Services Component functionality. VDP adopts the 3-tier web architecture which is depicted in the following diagram:

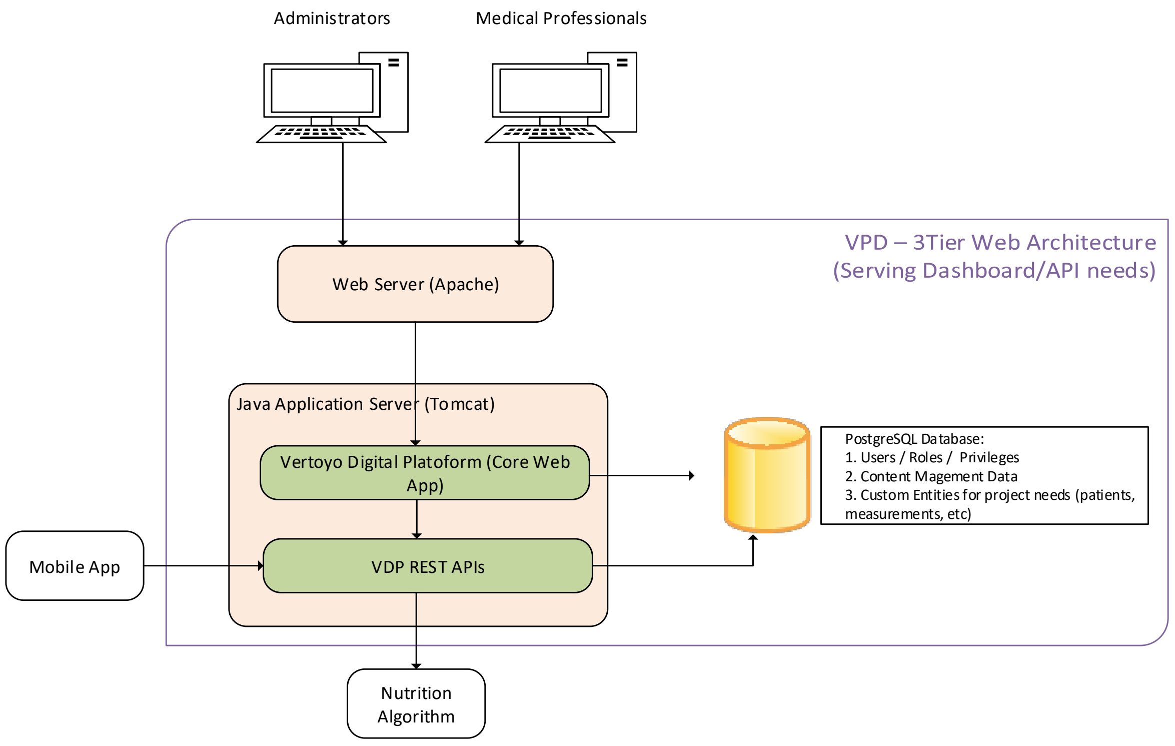

Administrators
Medical Professionals
VPD – 3Tier Web Architecture (Serving Dashboard/API needs)
Web Server (Apache)
Java Application Server (Tomcat)
Vertoyo Digital Platoform (Core Web App)
VDP REST APIs
Mobile App
Nutrition Algorithm
PostgreSQL Database:
1. Users / Roles / Privileges
2. Content Magement Data
3. Custom Entities for project needs (patients, measurements, etc)

Figure 2: VDP - 3 tier web architecture

Database Layer: VDP comes out-of-the-box with a single SQL schemas to store the needed Data for:

- User Management (encrypted data for sensitive fields)
- Content Management – static content of the Web App.
- Custom Entities Management – Encryption of data supported according to the project needs.

Any SQL Database can be used, however PostgreSQL DB is recommended and prefered for our purposes.
Application Layer: VDP Software, being a Java-based web application can be deployed and served by any modern Java Application Server. Apache Tomcat is recommended according to project needs.
Web Layer: A web server layer is also required in order to execute widely-used directives for boosting Dashborad component performance such as service static assets without any application layer inteference, allow browser-caching by adding the proper http header, enable gzip for serving compressed content etc.
In terms of hosting, VDP along with extra layers of Web Server & DB is recommended to be hosted in cloud virtual machines infrastructure with a Unix-based OS (we are mostly using Centos). In case of any project restriction or different approach selected for any purpose, compliance with any other hosting model will be feasible and fully compliant with our solution. The core component interactions will be towards the SQL Database which will integrate data in almost every use case. A few examples based on documented use cases (as per D2.6) are presented below:

- ***User Login***: For any user accessing Dashboard, login will be required. Credentials that will be posted will be checked versus the Database (USER table) and if validated the proper user role will be also retreived from the DB to define the next screen and menu options that will be allowed to the user.

- ***Patient Registration & Mobile App Login***: The doctor will have to fillin the required form. Once the data is provided and register button is pressed, the input data will be validated and then pesisted on the Database (MOBILE_APP_USER table). Once the patient registration is completed, the persisted data on MOBILE_APP_USER table will available via REST API to allow Mobile App Login for patients.

- ***Add Patient Measurment & Access Patient Measurement:*** The doctor will have to fillin the required form according to the measurement type. Once the data is provided and submit button is pressed, the input data will be validated and then pesisted on the Database (MEASUREMENT table). The specific data persisted on the Database will be available for the doctor to check patient’s history in terms of measurements and can be also available via the proper REST API to the Mobile App so that the patient himself can check his own past measurements.

Dashboard Application will also interact via VDP Rest API component with the Nutrition Algorithm which is one of the cornerstones of the NUTRISHIELD project. The Nutrition Algorithm will be triggered once the doctor (or medical personnel) has filled in via the Dashboard the full set of patient’s personal information and patient’s measurements. The outcome of the Nutrition algorithm could be either served as synchronous output of the REST API invoked or could be responded in an async way (depended on the algorithm execution time). Regardless of the way the Nutrition Alogirthm will provide the output, the final output will be finally persisted on the Database schema accessed by Dashboard, so that history data of the Nutrition Algorithm execution is avalaible to doctors via Dashboard UI.

For Database integration, direct access will be established towards the required databases via JDBC Adapter. All transactions will be executed over an established pool of Connection towards the Database and the proper configuration in terms of pool size, transaction time out will be fine-tuned according to volumes of traffic expected.

Regarding the integration with Nutrition Algorithm, REST APIs will be exposed both sides. Invocation will be enabled via VDP Rest API component. For Push Notifications purposes towards patient's smartphones, VDP will be integrated with any 3rd party Push Notification service (such as Urban Airship) via VDP Rest API component. For sending email notifications (needed for instance on reset password flow, registration, etc.), VDP will be integrated with a mail service hosted on the same Unix OS as VDP and will implemented the SMTP protocol.

2.2.3. Mobile Application

The main focus of the Mobile application will be in ensuring efficient dietary habits monitoring (food journal) in an engaging and simple way that adheres to best UI/UX practices while at the same time ensures that the patient and related information remains anonymous. The scope of the app will be to test the applicability of a Food Journal delivered to the platform by using it.

In a typical scenario the participants will login to the app by using credentials provided by their assigned nutrition specialist when they onboard the clinical study. The app will authenticate users by utilizing the NUTRISHIELD Backend services. Patients will usually have to record their food and drink intake for a few weekdays and a weekend day. The recording process includes the capture of a pre-consumption and post-consumption image of the meal/drink, together with a short text description provided by the user. This information will be subsequently and automatically sent to the NUTRISHIELD Platform by using the available RESTful APIs of the Backend. In addition to the images and text describing the meal, information regarding date and time of consumption will be automatically added and portion size information in numeric format with units - size category is also considered.

The data recorded as well as the user interactions are subject to fine tuning as the project progresses and will be revisited once initial feedback from users is received, following the focus groups before the initiation of the clinical studies. Some initial designs of the app screens can be seen below depicting the process described:

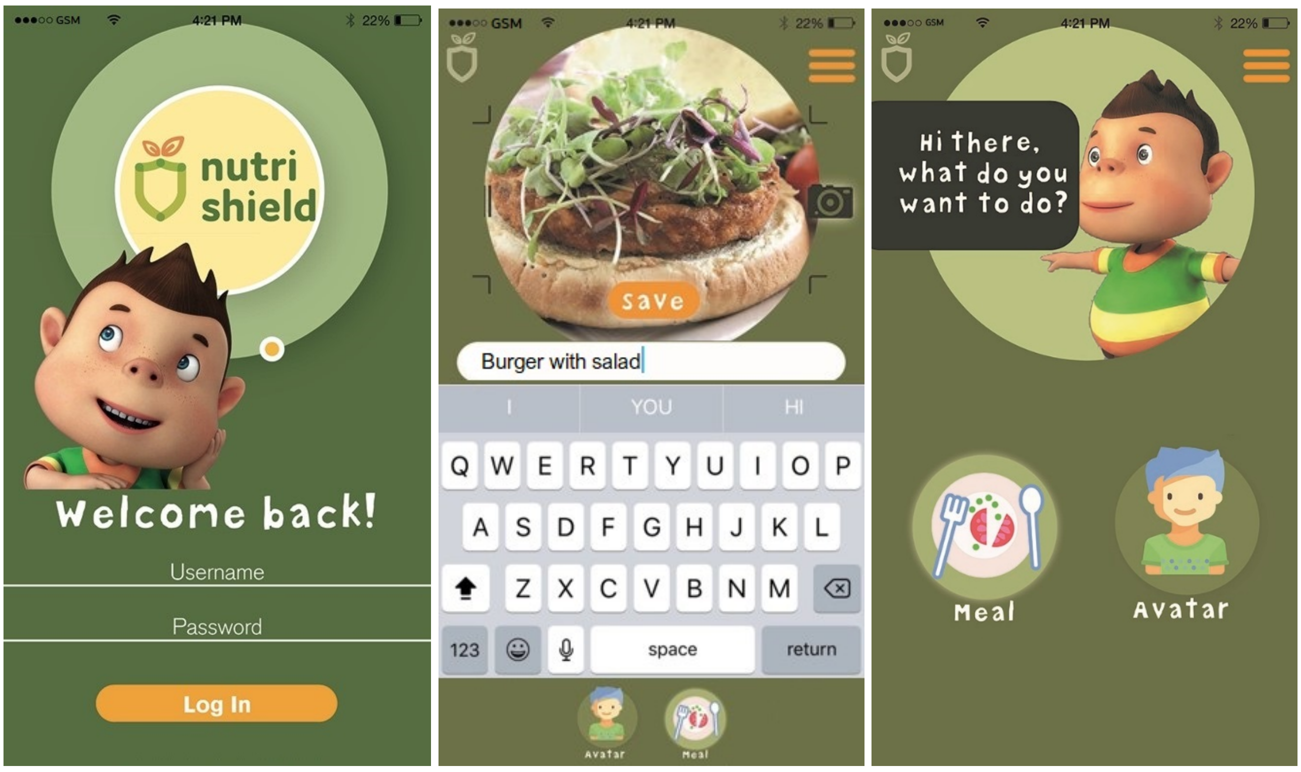


Figure 3: NUTRISHIELD APP sample screens

2.2.4. REST API

VDP Rest API component will mainly expose all the REST APIs that will be required by the Mobile App acting as Mobile Gateway / Middleware between the Mobile App and any NUTRISHIELD Databases. As described on Dashboard section, VDP Rest API component is part of VDP Software which is built on Java and can be deployed and served by any modern Java Application Server. Apache Tomcat is recommended according to project needs. In terms of hosting, VDP Rest API can be co-hosted in the same cloud virtual machines where rest VDP elements (Dashboard Application) will be hosted. Some of keys features of a Service Bus product which are also accomodated by VDP Rest API are listed below:

- Versioning: Once deploying a new version of a Rest API, the previous version can remain active too till all clients have transitioned to the latest version of the API.
- End-point Configuration & Load Balancing: Target end point of the can be easily configured without any downtime and load balancing (round robin algorithm) is also available in case that the target system has more than one target end-points for the same service.
- Read/Connect Timeout & Retry: Default timeouts are configured towards all target systems for all of the services exposed by VDP Rest API to avoid any thread starvation. The default configuration can be overriden in case this is required.
- Security: Security configuration per end-point exposed can be applied by whitelisting or blacklisting client IPs.

- Tracing: Tracing of each request/response by masking any sensitive personal data is available. Tracing will allow effective troubleshooting.

In terms of hosting, VDP Rest API is recommended to be hosted in cloud virtual machines infrastructure with a Unix-based OS (CentOs preffered). In case of any project restriction or different approach selected for any purpose, compliance with any other hosting model will be feasible and fully compliant with our solution. The specific component will mainly receive requests from Mobile App and in order to serve them will have to access the proper Database. A few examples based on documented use cases (as per D2.6) are presented below:

- User Login: For any user accessing Mobile App, login will be required. Credentials that will be posted towards the proper VDP Rest API will be checked versus the Database (USER table) and if validated the service will respond with success and user will idenitifed as logged in by the app.

- View Plan: Once a Logged in user requests to view the plan created by his/her doctor, a service call will be performed from Mobile App to the proper VDP Rest API. The Service will query the Database, retrieve the plan the doctor created for the specific user and the plan will be responded to the Mobile App for the user to read it.

- Upload Food Image: Every time a logged in user wishes to upload an image of the plate he is going to consume (or already consumed), the proper VDP Rest API will be invoked. The uploaded image will be stored by VDP Rest API to the database and upon successful upload the service will respond with success to Mobile App so that the user is displayed with the proper confirmation screen.

VDP Rest API will also implement the Middleware between Dashboard Application and Nutrition Algorithm by exposing the proper APIs for both systems/components. The Nutrition Algorithm will be triggered once the doctor has filled in via the Dashboard the full set of patient's personal info and patient's measurements. The outcome of the Nutrition algorithm could be either served as synchronous output of the REST API invoked or could be responded in an async way (depended on the algorithm execution time). Regardless of the way the Nutrition Alogirthm will provide the output, the final output will be finally persisted on the Database schema accessed by Dashboard, so that history data of the Nutrition Algorithm execution is avalaible to doctors via Dashboard UI.

Acting like a Gateway to the Mobile App, VDP Rest API component will expose the required APIs to serve the following operations of Mobile App:

- Login
- Logout
- Change Password
- Retrieve Plan
- Add/Edit/Delete meal info
- Add/Edit/Delete activity info
- Upload/Delete image

For security purposes, any Rest API exposed for Mobile App usage will have to be invoked using a security token. VDP Rest API will have to implement a token creation mechanism (encapsulated within Login API) and token pool management mechanism. Upon successful Login (valid credentials send by the Mobile App), the API will have to create a unique token and respond it back to the Mobile App. Mobile App will have to use the unique token in every call performed post login. VDP Rest API will keep the pool of tokens up to date and associate each token with a specific user. Tokens will automatically expire if not used for 30 mins (adjustable value) and mobile app user will have to perform login again. Upon logout action, the token will expire immediately.

Regarding the integration with Nutrition Algorithm, REST APIs will be exposed both sides. Invocation will be enabled via VDP Rest API component.

2.2.5. Databases

VDP Solution requires the integration with an SQL Database. Any SQL-technology Database can be utilized such as MySQL, Oracle, etc. however it is recommended to utilize PostgreSQL [1] (version 9 or higher). PostgreSQL, also known as Postgres, is a free and open-source relational database management system (RDBMS) emphasizing extensibility and technical standards compliance. It is designed to handle a range of workloads, from single machines to data warehouses or Web services with many concurrent users.

In terms of Database Architecture, any of the following installation options are available:

- Standalone Instance: Install Postgres as a standalone instance that will be taken back-up once per day (during low-traffic hours). The specific type of installation will require a simpler installation and less H/W needs, guaranteeing that in case of any fault, data up to 24h could be lost.

- High Availability (Master-Slave model): Such an architecture enables maintenance of a master database with one more standby server ready to take over operations if the primary server fails. These standby databases will remain synchronized (or almost synchronized) with the master. To allow real-time replication, Postgres uses a stream of write-ahead log (WAL) records to keep the standby databases synchronized. If the main server fails, the standby contains almost all of the data of the main server and can be quickly made the new master database server.

In terms of hosting, VDP Database is recommended to be hosted in cloud virtual machines infrastructure with a Unix-based OS (we are mostly using Centos). In case of any project restriction or different approach selected for any purpose, compliance with any other hosting model will be feasible and fully compliant with our solution.

Integration towards VDP Database Schema will be allowed with direct JDBC access for read & write operations from VDP (Dashboard) & VDP Rest API. All transactions will be executed over an established pool of Connection towards the Database and the proper configuration in terms of pool size, transaction time out will be fine-tuned according to volumes of traffic expected.

For any other component/layer of NUTRISHIELD Platform, read & write operations towards VDP Database schema will be allowed only via an API served by VDP Rest API layer.

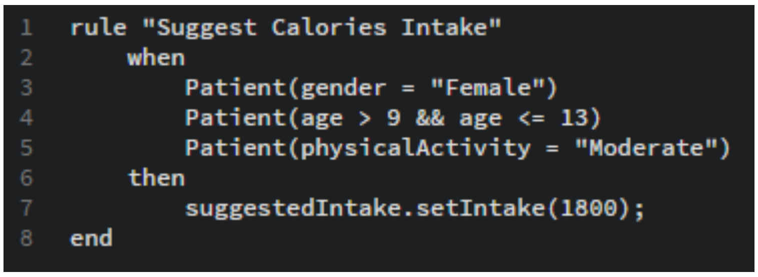
2.2.6. Nutrition Algorithm Service

By adhering to the Microservices architectural approach which is presented in detail in Section 3.1 of this report, the design of the Nutritional Algorithm Service aims at producing a NUTRISHIELD platform component which is modular, easy to integrate and use and provides the necessary nutritional suggestions by taking in account the available patient information. The algorithm by leveraging relevant data stored in databases and input provided from the Dashboard it produces personalised nutrition suggestions for patients.

Integrating this service to the rest of the system is easily achieved as the operations it offers are made available via a well-defined and documented set of endpoints which receive and return data serialised in JSON format. Communication with the databases to retrieve relevant data (measurements) coupled with other input provided by the Dashboard are used as parameters in the algorithm execution. A request for a nutrition suggestion is performed via a well-defined RESTful API that this component exposes. Endpoints of this API are used for both triggering the algorithm and providing the necessary input for its execution, as well as getting hold of the resulting nutritional suggestion.

This component is designed so as to not maintain any internal state but rather execute using the input provided by the Client consuming the services it offers, resulting this way in minimal coupling. For the implementation of the API this component offers the RESTful approach already described is adopted and the programming language of choice is Java and the Spring Framework [2]. While Java and Spring enable the implementation of the endpoints necessary that will wrap around and expose the algorithm, the actual implementation of the nutritional recommendation engine is based on Drools [3]. The rule engine provided by Drools can process facts in an attempt to produce output which results from the execution of rules based according to the facts. The set of applicable nutritional rules backed by relevant literature and provided by HUA can be written in an easy to comprehend format and furthermore allow the necessary flexibility for finetuning.

The figure below demonstrates the approach and syntax of the Drools Rule Language in defining associations between the input (Facts) and the corresponding action. It should be noted that the Rule described below is simplistic and for demonstration purposes only, used to display a Rule structure and format. Therefore, its actual calorie suggestion although it remains reasonable, it should be ignored by the reader:



```
rule "Suggest Calories Intake"
    when
        Patient(gender = "Female")
        Patient(age > 9 && age <= 13)
        Patient(physicalActivity = "Moderate")
    then
        suggestedIntake.setIntake(1800);
end
```

Figure 4: An example Rule written in Drools Rule Language (for displaying syntax)

The Basic concepts of Drools can be summarised as follows:

1. **Facts:** Are the actual input to the system which will be fed into the rules.
2. **Working Memory:** Represents the space where all the facts are temporarily stored while the rules engine is executing and searches for matching rules given the input.
3. **Rule:** An association between given Facts with the corresponding actions. The Drools Rule Language is commonly used to express rules but also decision tables in Excel are an option.
4. **Knowledge Session:** It is a session that maintains all resources necessary for firing the applicable rules. The available input data (Facts) are inserted when a session is initiated and subsequently the rules matching the facts are executed.
5. **Knowledge Base:** Maintains the information regarding resources such as where the Rules reside and can create the Knowledge Session.
6. **Module:** A Drools module can hold many Knowledge Bases which in turn can create different Knowledge Sessions.

In the NUTRISHIELD algorithm implementation, a Drools Module will hold a Knowledge Base that will be used to create Knowledge Sessions. The Session will be used whenever a Client (API caller) requests a nutritional recommendation. The Facts will be provided by the Client in JSON format via dedicated endpoints offered by the RESTful API. Once a call has been performed, the Session and Working Memory will be initialised, which is equivalent to initialising variables with the input provided by the Client. The set of Rules of the algorithm provided by the experts and expressed accordingly in Drools Rule Language clearly dictate the associations between a patient's characteristics (Facts) and the corresponding actions (nutritional recommendations). Once the rule engine has evaluated the rules given the provided input, the nutritional recommendations will be returned in JSON format for further use.

2.2.7. Machine Learning

Essentially, this component will be a set of mathematical models which takes the patient data as input in order to classify it in a certain group (e.g. risk groups, disease type and stage, etc.). The group classification will be determined based on the results from the clinical studies and serve as a basis to assess which diet is the best for each particular individual. The main part of the component will consist of an R model. Different input formats will be used for the prediction: the genotyping, microbiome and biomarkers.

Due to their size, the raw genotyping data, in fastq format, will be stored directly within the servers at CU and processed there. The group's hardware infrastructure includes a dedicated server components specifically aquired for this purpose. The group's HPC facility will be used for performing the computationally intensive genotype (SNP and InDEL calling) analysis for the chosen population during clinical studies. The output will be vcf files (variant calling format), which will contain the variants found for a specific patient. This information will be combined with the other data (Microbiome, Biomarkers…) to classify the patient in a certain group. The final algorithm will deliver personalised nutrition advice based on biomarkers and their time-dependent evolution. Generating the advice will require the integration of mechanisms identified from the analysis of the data collected during the study as well as established nutritional rules. The set of nutritional rules integrated in the final algorithm will need to be identified and given a raking based on priority. A process that will be performed in collaboration with the rest of the NUTRISHIELD consortium.

Fusing of biomarker data with the algorithm optimised in T5.2 will be performed by using the biomarker data as "training set". The biomarker data collected during the first 6 month of observation period of the NUTRISHIELD study will be used as training set for the personalised nutrition algorithm. The result will be the first version of the algorithm that will be used to make predictions. These prediction need to be evaluated from a nutritional perspective before being transmitted to study subjects. If necessary, the algorithm needs to be tuned to generate plausible advice. The subjects will then follow the predictions of the algorithm for the second 6-month intervention phase of the study, as more biomarker data will be collected. After this phase, the data collected will be used as a second training set, to refine the algorithm to generate advice in the last 6-month intervention phase of the NURISHIELD study. The adaptive nature of machine learning algorithms will allow the integration of biomarker data with well-accepted nutritional rules. This component will interact with the rest of the NUTRISHIELD App via an API that will be set up at CU. The API will be able to accept requests, with the patient data (metabolics, biomarkers, age etc…) which will also launch the prediction algorithm. The output will be the classification of this patient into a specific group which will then be linked to a personnalised diet. The output will be sent back to the NUTRISHIELD platform backend in json format.

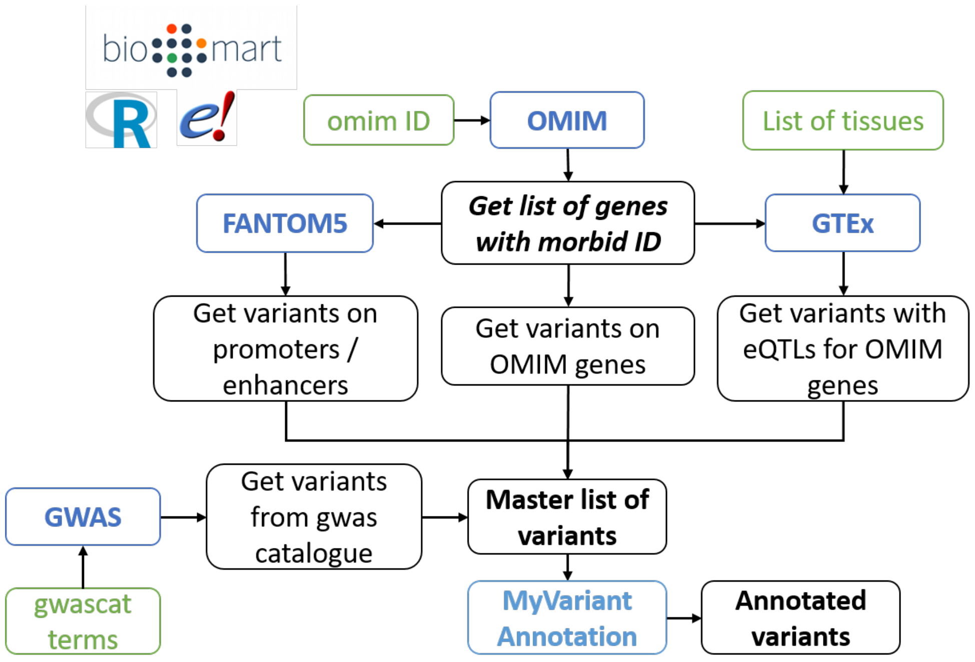

biomart
omim ID
OMIM
List of tissues
FANTOM5
Get list of genes with morbid ID
GTEx
Get variants on promoters / enhancers
Get variants on OMIM genes
Get variants with eQTLs for OMIM genes
GWAS
Get variants from gwas catalogue
Master list of variants
gwascat terms
MyVariant Annotation
Annotated variants

Figure 5: The VarGen workflow: This pipeline is centred on the genes linked to the disease of interest in the Online Mendelian Inheritance in Man (OMIM).

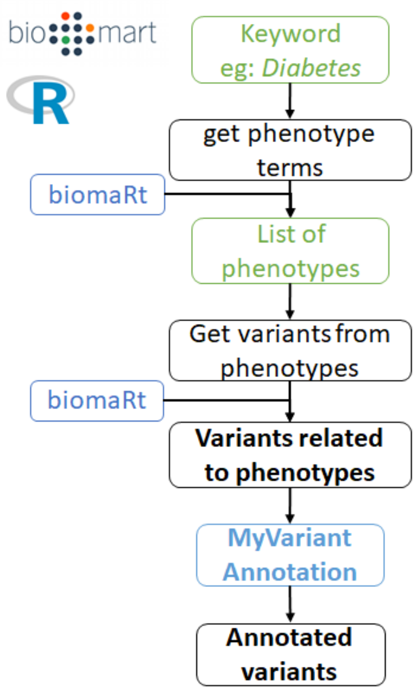


Figure 6: The VarPhen workflow. A more specific, alternative pipeline is available as part of this package, called "VarPhen", it outputs a smaller list of variants, but directly related to the disease of interest.

3. Detailed Integration Design

This section aims to highlight the main approaches and related technologies that will be adopted in order to facilitate interactions and communications between platform components, resulting to the seamless integration of the NUTRISHIELD platform.

3.1. The Microservices architectural approach

As an alternative to the somewhat traditional Monolithic architectural approach for developing applications, which under certain conditions can still remain a good choice, the Microservices architectural approach represents an option which not only addresses limitations and issues of the former, but also is a good fit for large and complex applications.

The Microservices approach achieves the above by adopting a strategy of putting together a large and complex application from small individual building blocks. These are distinct components that perform a specific function, examples of which include among others processing of data, login services and persistence of information. These individual and discrete components (microservices) can be considered as separate software components which have their own code and resources. The entire functionality of the system is therefore realized and composed by the microservices available, which work together as a whole, communicating among them and clients of the system to fulfil requests.

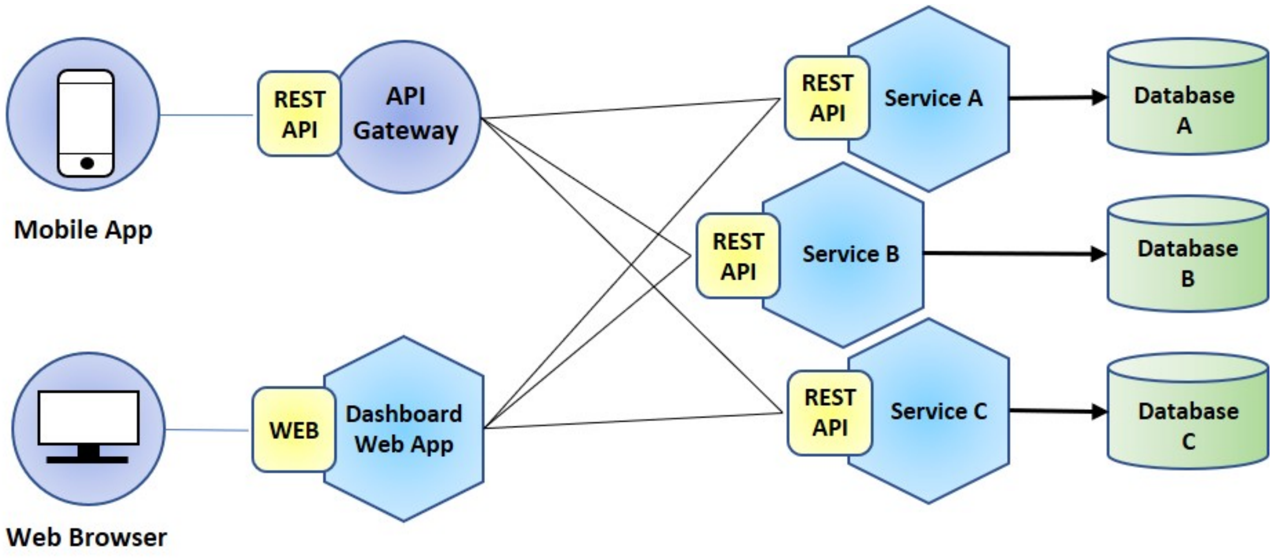


Figure 7: An example of the Microservices Architecture

The overall aim is to structure an application as a collection of services that are: Highly maintainable and testable, loosely coupled, independently deployable, organized around business capabilities and each owned by a small team [4]. It is expected that each small service is running in its own process and communicates with other services and clients using lightweight mechanisms such as well-defined HTTP resource APIs (web services). These APIs frequently and depending on the case employ authenticated and encrypted communications that follow the RESTful paradigm (further discussed in later sections of this document). The services are implemented in ways that allows for them to be deployed independently from each other while always having in mind fully automated deployment approaches as they are described in the CI/CD related section (see Section 3.3). This particular approach further allows for minimum centralized management of the available services that can also be implemented by different teams, using different programming languages and potentially employ different data storage approaches if necessary.

This way it can be possible to decompose an otherwise complex monolithic application into a set of services, each of which can be developed in a shorter amount of time, independently and by a dedicated to that service team and also in parallel with others. Furthermore, since the software produced has specific purposes depending on the implemented service, it becomes more understandable and maintainable with the added benefit of being able to deploy each service independently to the existence or state of others. The above results in rapid, frequent and reliable delivery of large, complex applications with the ability to integrate new services and existing monolithic applications into one efficient, secure and flexible feature rich application.

By adopting the Microservices approach, developers and developer teams of the NUTRISHIELD project will be able to develop and deploy the necessary services independently and therefore be able to introduce functionality to the system simultaneously with minimal dependency between them once the interfaces of the components have been identified and agreed upon. The technologies chosen for implementation can also in each case be the most suited for the service being developed. The implemented functionality can further be deployed in a containerized fashion which can minimize overhead and increase the portability between environments while allowing the realization of CI/CD pipelines. Another added benefit of the above approach is the inherent ability to have a more systematic approach when identifying and debugging failing components which in most cases do not tend to affect other microservices which can continue to function.

3.1.1.1. REST Architectural Approach

The Representational State Transfer (REST) architectural approach is a design pattern for implementing HTTP resource APIs. Clients of the available APIs represent browsers, developers or other software who make use of the service available via an API, consuming thus such services. In general, APIs provide information about Resources (specific objects). In particular, RESTful web endpoints (web services) expose information about its resources and allow clients to perform specific actions on those resources including creation or alteration of the latter. An example of this can be the creation of a new NUTRISHIELD patient record or the alteration of the information related to them etc. The Representational State Transfer occurs when a Client uses a specific RESTful API endpoint to e.g. fetch information about a specific NUTRISHIELD patient. This result in the server transferring to the Client a representation of the state of the requested resource where in this example could be e.g. the related doctor, meals consumed and other information according to applicable security policies.

While the response containing the representation can be in HTML or XML format, it is usually the case that the information is returned in JSON format. The typical interaction between a Client and a RESTful API begins with the former providing the URL related to the resource of interest, together with the operation that the server should perform on that resource. The operation is in the form of an HTTP method such as GET, POST, PUT and DELETE. In the mentioned example the patient info would require a

GET of the related patient resource. In order to change e.g. the doctor related to a specific patient a PUT request would be necessary in which the new doctor would be provided and usually in JSON format. It should be mentioned that in general the JSON format for exchanging data is the most popular approach in RESTful APIs where information is maintained in structured and organized name-value pairs. The entire set of operations in a RESTful API can effectively compose a Microservice as mentioned above. There exist specific constraints when implementing APIs which aim to make them more usable and efficient and these should be adhered to if an API is to be considered RESTful. These constraints are related to the fact that an API should have a uniform interface, feature client - server separation, be stateless, be a layered system, be cacheable and facilitate code-on-demand, further information about which can be found in [5].

3.1.1.2. REST Authentication and Authorization

Authentication and authorization concerns are also inherent and relevant to the use of REST APIs. Authentication deals with whether a Client is allowed to connect to a specific server providing the API while authorization is concerned with whether a particular Client after connection is allowed to perform the task in question or otherwise have access to a specific resource. The Basic authentication approach is the simplest one where an HTTP header with username and password information is included in base64 encoding. Due to the fact that this approach contains the credentials unencrypted it should only be used with encrypted SSL/TLS connections. To avoid the overhead of sending the credentials with each request and avoid tampering of the headers or body of the request, Hash Based Message Authentication (HMAC) represents an option where the Client sends a digest of the request and a nonce that based on a shared secret between Client and Server it can be verified. A more common approach used is the OAuth 2.0 token-based authentication where Clients after registering with a provider are given a token to be used with every request. Requests that have been altered in any way or contain an invalid token are rejected. In general requests should be made over SSL/TLS otherwise the token can be stolen. Tokens can also be made to expire after a certain period or be revoked to restrict access when necessary.

For the data persistence needs of the NUTRISHIELD platform both relational SQL databases (e.g. PostgreSQL) and NoSQL (MongoDB) approaches will be employed. Depending on the data and the related pre-defined or dynamic schemas where table-based or document-based approaches are necessary the appropriate database instance will be used. The data will originate from both the dashboard and mobile application and can include patient measurements, user account data, food diary entries, etc. In general, anything that needs to be persisted will be stored in the system databases and will be accessed by using the appropriate database drivers for each technology. The Java Database Connectivity (JDBC) API is an available Java API for SQL databases used to connect to a database, send queries and commands to it and handle the result sets returned once queries get executed. For NoSQL MongoDB database instances there exists the equivalent MongoDB Java Driver which provides the necessary functionality for connecting, administering, querying against collections and handling result sets. Furthermore, several additional REST interfaces are available for MongoDB instances, implemented in various programming languages including Java and Python that can be utilised in the course of the project to provide a front end to a database instance, effectively creating a database service for other components.

Regarding communications between system components it is necessary to enable secure and encrypted only approaches (HTTPS using SSL/TLS). The hosts used for deploying the various NUTRISHIELD platform components and services will be adequately secured employing encryption at rest (encrypted volumes), adequate firewall rules configuration that allows remote login and connections only from other trusted hosts, while in the cases that a volume of a remote host needs to be mount locally on another, SSHFS will be used. For notification purposes between backend systems to the dashboard and to the mobile app, Stomp and Push notifications can be used respectively.

3.2. System development and deployment

3.3. Repositories, building and deployment tools (CI/CD)

A crucial part of complex software systems development and delivery lifecycle is Continuous Integration (CI) and Continuous Delivery/Deployment (CD) – jointly mentioned as CI/CD. CI, in software development, is a practice of building/integrating and testing all developer working code frequently in a shared code repository. A common practice in CI is to integrate the changed code at least daily. The frequent integration helps the contributors to notice any arising errors and correct them instantly.

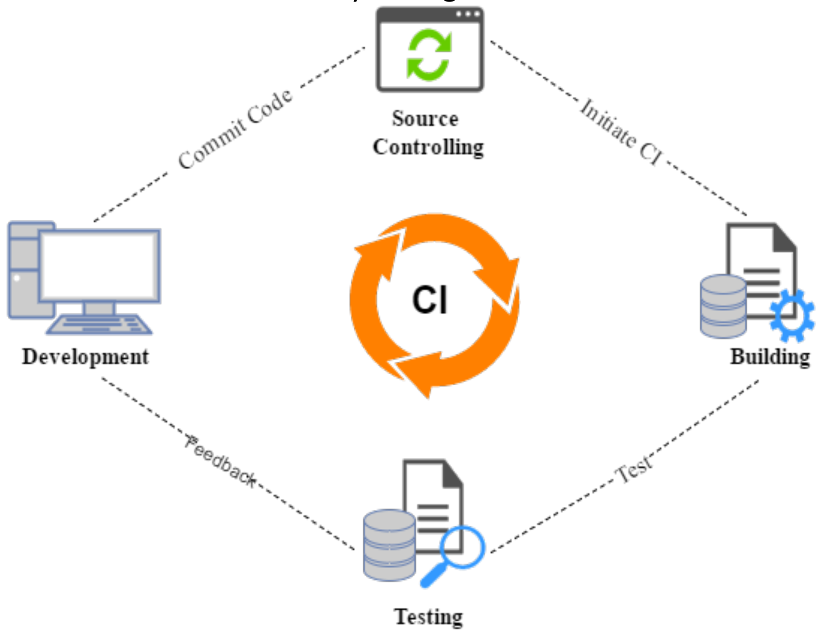


Figure 8: The Continuous Integration Lifecycle

Continuous Integration offers <u>significant advantages</u> such as:

- reduction of risk in the development, since it reveals any possible incompatibility between software components early during the development phase
- facility of instant bug fixing
- availability of current product version at any moment, as the code is frequently integrated

In order to enable the CI process and benefit from the aforementioned advantages we have to perform extensive testing of each software component/module but also of the integrated platform as a whole. Automated per component tests and combined integration tests that are performed on each new build (version) of a component shall be executed and in case of success the integrated platform will be updated with the new version of the component. Otherwise the developers will be notified so as to take proper action to fix the problem that caused the failure. A Test-Driven Development (TDD) approach may be followed, starting from unit tests per software component, upon specified test cases for each component, proceeding to integration tests that validate the correct functionality of the integrated platform that involves two or more components in an automated manner with the use of a CI server, such as Jenkins [6]. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

In case that the resulting software system is composed of several software components/modules, developed by diverse development teams, a collaborative software development approach can be followed, during which it is important to ensure data integrity and availability. To support this, a distributed version control system (VCS) is necessary for the efficient system software delivery. For this purpose, GitLab [7] could be utilized, a source code management and VCS that aims mostly at data integrity and full version tracking. GitLab is an easy yet powerful and intuitive git VCS. Multiple developers can concurrently create, merge and delete parts of the code they are working on independently, at their local system before applying the changes to the shared GitLab repository. A DevOps framework could be adopted.

To ensure quality of software development, build automation is additionally pursued. Build automation is considered to be the act of automating processes that are associated with software building. Such processes might include various parts like source code compiling into binary code, packaging binary code and automated test running but also the delivery/deployment and documentation parts. Maven [8] is a useful tool for build automation and project management for projects that are written in numerous programming languages (Java, Ruby, C# and other). This process can be applied for components software shared and residing in the software source code repository.

It is also essential in collaborative software development to distribute the software efficiently. Among others, Nexus [9] is a popular repository manager that manages the required software artifacts. It allows developers to distribute their software easily but also to proxy, publish and collect the necessary project dependencies. Actually, the Nexus Repository offers a standardized way for cataloging and storing developers' artifacts. Once a new library is developed, it is handed out to the repository manager. After that, other developers can efficiently access these software components by using a standardized procedure. Clearly, it is possible to control centrally the development of all artifacts and the access to them. Nexus can be used for pre-built software components, the source code of which is not available or shared – however, updated versions of software components need to be frequently built and released in new versions at the Nexus repository to support continuous integration and delivery. The CI server will monitor the repository and will initiate a new platform test and deployment cycle after each new version upload.

To further support automated delivery (**Continuous Delivery - CD**) in software development, Docker [10] has been chosen, an open-source software containerization platform, as a straightforward way to provide isolated running environments with pre-set configuration. Docker works with software containers in order to allow the software to run always the same, independently of the deployment environment. Actually, it wraps up the software in a complete file system, along with any necessary tools or software resources, such as libraries, code and runtime. This way, multiple docker containers can run on a single Linux instance, without any overhead for managing several virtual machines. Moreover, by using Docker, the software system deployment is simplified at a great extent, set up is minimal and uniform across all component projects. Each component is contained in a different Docker image ready to be executed in a Docker hosting machine as a separate container. These images can be published in a shared repository, such as the Docker registry, and through the Docker Compose functionality these images can be retrieved from the Docker registry and deployed together via a single configuration file. Containerization thus provides OS level virtualization. This means that multiple applications running in containers on a single host, access the same OS kernel. Hence, it is faster and more lightweight than isolating applications using VMs. Containers have an initial configuration which does not affect the configuration of other containers, even though they share the same host OS. This eliminates errors due to unexpected conflicts or missing dependencies, which are common when applications are installed on a single host without isolation. In more demanding installations due to increased load of the system, Docker is perfectly suitable to be configured with load balancing mechanisms that can scale up the performance of the system.

The following figure depicts the CI/CD workflow with all the tools mentioned above, and summarized below:

- GitLab for source control, acting as code repository and allowing code versioning
- Jenkins for automated build and testing
- Docker for containerization of services and components
- Docker Registry for easy deployment at different infrastructures

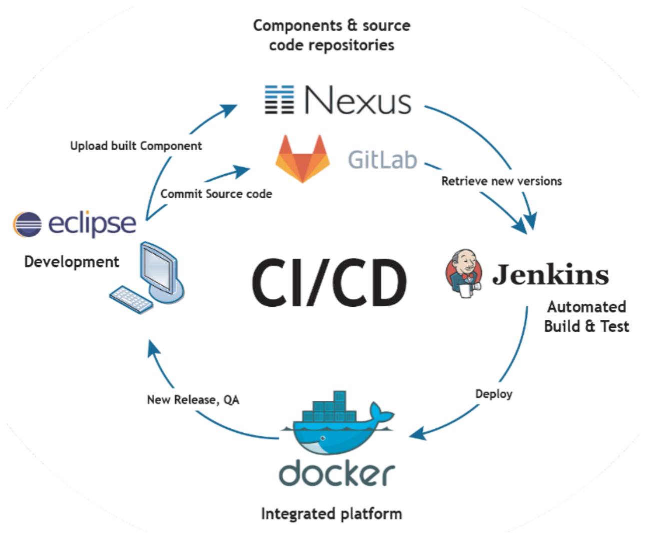


Figure 9: Continuous Integration & Continuous Delivery process

Using the CI/CD environment and tools, when developers implement new component features or integration endpoints, they *push* their code to GitLab, the central source code repository in this environment, which is then compiled, built and tested using Jenkins, while Docker is finally creating a Docker image, that is pushed to the Docker Registry.
Once components have been built and their images have been pushed to the docker registry, they are available to be pulled from *any server* which has access to the docker registry. The deployment to Deployment servers can be carried out via a script which automates the entire process, as shown below:

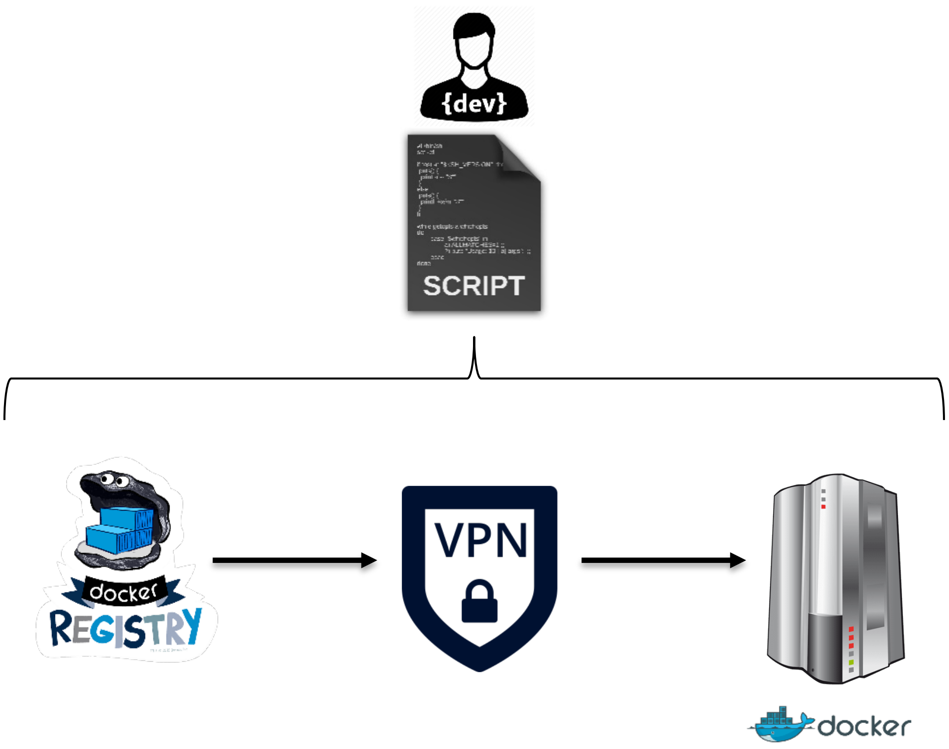


Figure 10: Flow from the docker registry to the Deployment server

Whenever code changes are required, component developers push their source code to GitLab. Afterwards, Jenkins takes over from that point automatically performing the steps described above. Eventually, the component is built into an image and *pushed* into the Docker registry. Then, a new version of the system is specified, and the deployment script is rerun to install the updated software system at the Deployment server. The Figure below illustrates the flow from code changes (e.g. bug fix, implementation of new features) to re-deployment on any server.

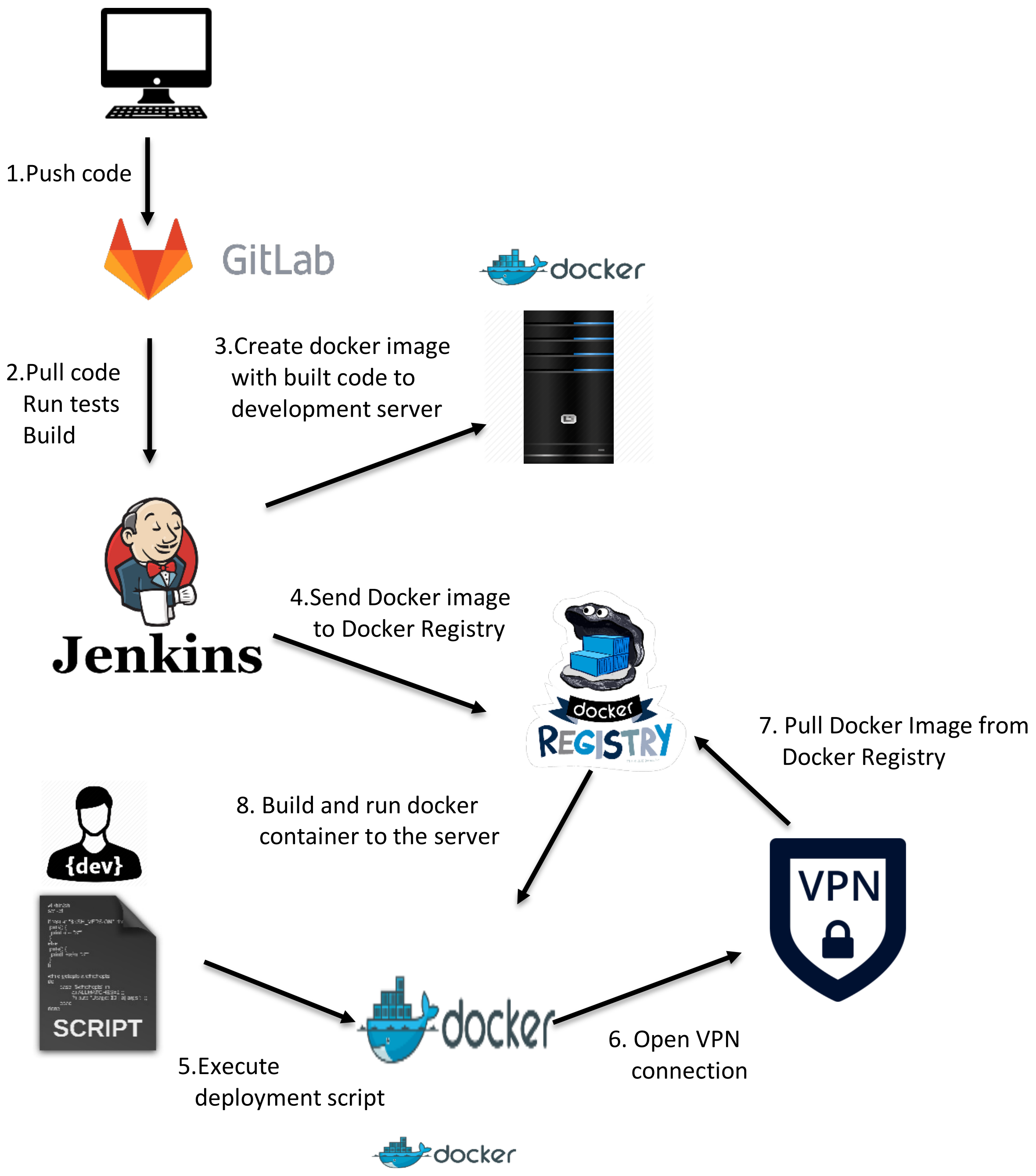
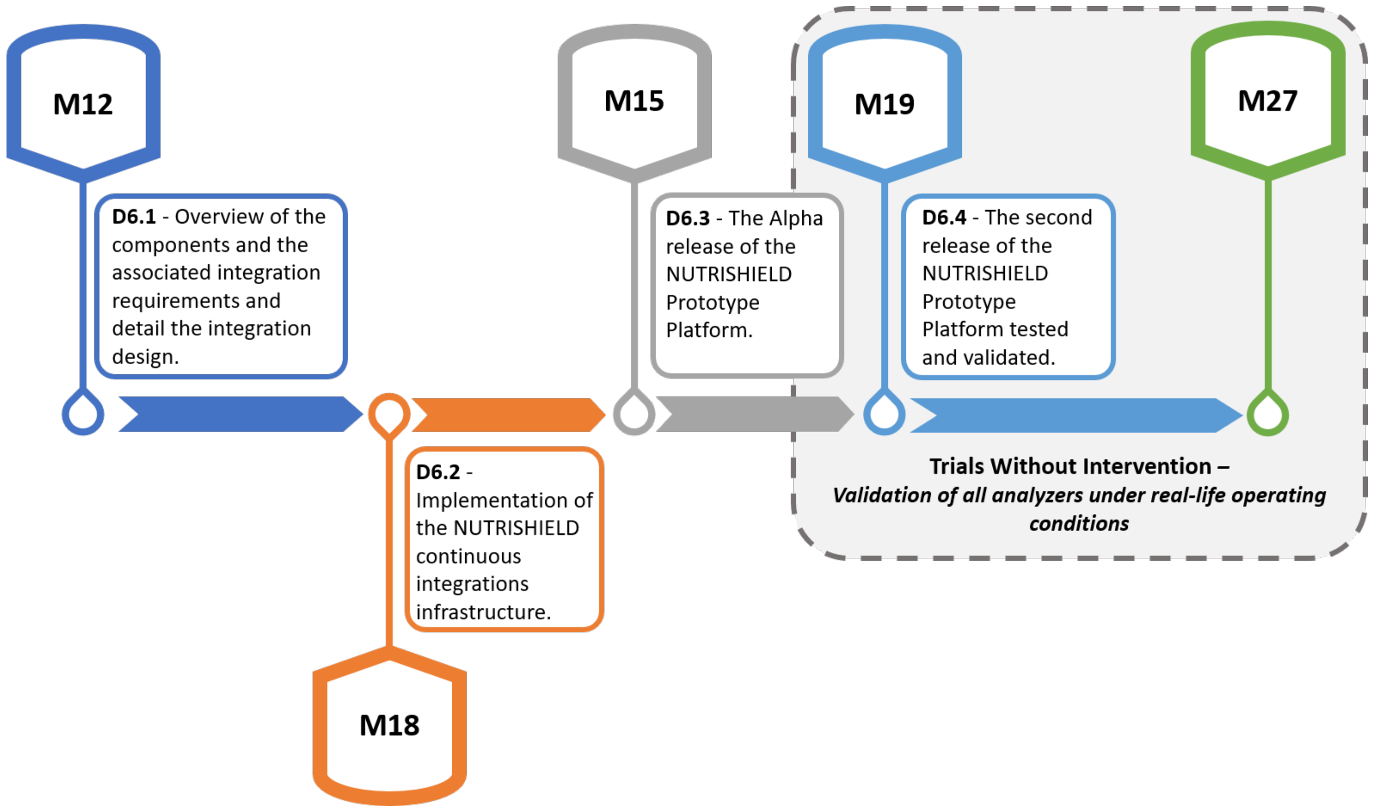


Figure 11: Code changes workflow

Docker containers are ephemeral. This means that when a container is removed, its state (and data) is also lost. To remedy this, docker offers named volumes on which containers mount. These containers exist independently from the containers that use them.

3.4. Platform Integration Roadmap


M12
D6.1 - Overview of the components and the associated integration requirements and detail the integration design.
M15
D6.3 - The Alpha release of the NUTRISHIELD Prototype Platform.
M19
D6.4 - The second release of the NUTRISHIELD Prototype Platform tested and validated.
M27
Trials Without Intervention – Validation of all analyzers under real-life operating conditions
D6.2 - Implementation of the NUTRISHIELD continuous integrations infrastructure.
M18

Towards the forthcoming clinical study in Italy that will focus on the children's obesity, the consortium partners decided to shift their efforts on implementing the NUTRISHIELD mobile application and dashboard six months earlier than it was initially planned and documented within the DoA. As a result, these two components of the NUTRISHIELD system will be ready to be tested at M15 (alpha release) and M19 (final version). This way, valuable feedback regarding the described functionalities of the components will be gained, through the testing activities performed in real-life conditions under the supervision of the appropriate clinical experts.

Conclusions

This deliverable provided a detailed overview of the components that constitute the NUTRISHIELD platform. Each component, together with the associated integration requirements was described, highlighting the integration design not only in high-level but also in terms of technical specifications. In addition, the proposed modular approach and the production grade product prototyping CI/CD techniques that will be used throughout the project's lifespan are also described allowing the extension of the platform and thus providing flexible opportunities for NUTRISHIELD to be adapted to different use cases. Further modifications and additions to this report will be considered if necessary and applicable as the project progresses and components' maturity level increases. These changes will be documented in the forthcoming deliverables of WP6 which will accompany the demonstrators, i.e., the NUTRISHIELD prototype platform.

4. References

[1] "PostgreSQL: The World's Most Advanced Open Source Relational Database," The PostgreSQL Global Development Group, 2019. [Online]. Available: https://www.postgresql.org/. [Accessed 13 Sep 2019].

[2] "Spring: the source for modern java," Pivotal Software Inc., 2019. [Online]. Available: https://spring.io. [Accessed 11 Oct 2019].

[3] "Drools -Business Rules Management System," Red Hat Inc, 2019. [Online]. Available: https://www.drools.org/. [Accessed 23 Sep 2019].

[4] C. Richardson, "What are microservices?," 13 Sep 2019. [Online]. Available: https://microservices.io/.

[5] S. B. Avraham, "What is REST — A Simple Explanation for Beginners, Part 2: REST Constraints," Medium, 5 Sep 2017. [Online]. Available: https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-2-rest-constraints-129a4b69a582. [Accessed 10 Oct 2019].

[6] "Jenkins - Build great things at any scale," Creative Commons, 2019. [Online]. Available: https://jenkins.io/. [Accessed 26 Sep 2019].

[7] "GitLab," GitLab, Inc, 2019. [Online]. Available: https://about.gitlab.com/. [Accessed 7 Oct 2019].

[8] "Maven – Welcome to Apache Maven," The Apache Software Foundation, 2019. [Online]. Available: https://maven.apache.org/. [Accessed 14 Sep 2019].

[9] "Nexus Repository Pro," Sonatype Inc, 2008. [Online]. Available: https://www.sonatype.com/product-nexus-repository. [Accessed 13 Oct 2019].

[10] "Docker: Enterprise Container Platform," Docker Inc, 2019. [Online]. Available: https://www.docker.com/. [Accessed 30 Sep 2019].