Grant Agreement no. 777167

BOUNCE

Predicting Effective Adaptation to Breast Cancer to Help Women to BOUNCE Back

Research and Innovation Action
SC1-PM-17-2017: *Personalised computer models and in-silico systems for well-being*

Deliverable: 5.1 BOUNCE Conceptual & Reference Architecture

Due date of deliverable: (10-31-2018)
Actual submission date: (16/11/2018)

Start date of Project: 01 November 2017 Duration: 48 months

Responsible WP: FORTH

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 777167		
Dissemination level		
PU	Public	x
PP	Restricted to other programme participants (including the Commission Service	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (excluding the Commission Services)	

0. Document Info

0.1. Author

Author	Company	E-mail
Lefteris Koumakis	FORTH	koumakis@ics.forth.gr
Haris Kondylakis	FORTH	kondylak@ics.forth.gr
Galatia Iatraki	FORTH	giatraki@ics.forth.gr
Maria Hatzimina	FORTH	hatzimin@ics.forth.gr
Panagiotis Argyropaidas	FORTH	parg@ics.forth.gr
Kostas Marias	FORTH	kmarias@ics.forth.gr
Kostas Perakis	SiLo	kperakis@ep.singularlogic.eu
Gianna Tsakou	SiLo	gtsakou@singularlogic.eu
Juha Salonen	Noona	juha.salonen@noona.com
Katerina Argyri	ICCS	kargyri@mail.ntua.gr
Nikolaos Christodoulou	ICCS	nikchris@mail.ntua.gr
Eleni Kolokotroni	ICCS	ekolok@mail.ntua.gr
Georgios Stamatakos	ICCS	gestam@mail.ntua.gr
Evangelos Karademas	FORTH	karademas@uoc.gr
Akis Simos	FORTH	akis.simos@gmail.com

0.2. Documents history

Document version #	Date	Change
V0.1	15/09/2018	Starting version, template
V0.2	20/09/2018	Definition of ToC
V0.3	17/10/2018	First complete draft
V0.4	22/10/2018	Integrated version (send to WP members)
V0.5	24/10/2018	Updated version (send to project internal reviewers)
Sign off	09/11/2018	Signed off version (for approval to PMT members)
V1.0	16/11/2018	Approved Version to be submitted to EU

0.3. Document data

Keywords	Software architecture
Editor Address data	Name: Lefteris Koumakis Partner: FORTH Address: N. Plastira 100, Bassilika Vuton, Heraklion Greece Phone: +30 2810 391424 Fax: E-mail: koumakis@ics.forth.gr
Delivery date	

1. Table of Contents

Table of Figures

2. Introduction

This deliverable describes the BOUNCE conceptual and reference architecture. An overview of the current state of the art in technologies relevant to the tools that will be developed and used within the project has been already reported in Deliverable D1.3. General decisions for the BOUNCE system architecture have been elaborated after validation and analysing the selected scenarios and user requirements described in deliverable D1.2 (Requirements & Usage Scenarios). This document reports on the components and specification of their interfaces, security and safety issues as well as semantic integration approach for data to be collected in BOUNCE. In addition, it documents a short summary on requirements and stakeholders and a description of the elaborated solution strategy including important design decisions.

2.1. Software architecture methodologies

A fundamental purpose of software architecture is to help manage the complexity of software systems and the modifications that systems inevitably undergo in response to external changes in the business, organizational, and technical environments. There is no single, industry-wide definition of software architecture. The term architecture in relation to ISO 42010 is defined[1] as: *"Fundamental concepts or properties of a software in its environment embodied in its **elements**, **relationships**, and in the **principles** of its design and evolution."*
The Software Engineering Institute (SEI) web site includes a long list of definitions for the term "software architecture"[2] such as: *"Software Architecture is a statement that addresses the key **concerns** of the software **stakeholders**.*

- A ***stakeholder*** in a software architecture is a person, group, or entity with an interest in or concerns about the realization of the architecture.
- A ***concern*** about an architecture is a requirement, an objective, an intention, or an aspiration a stakeholder has for that architecture."

2.1.1. Scrum

Scrum[3], one of the state of the art software architecture methodologies, is based on an iterative and incremental agile software development methodology for managing product development. It is a process framework that has been used to manage complex product development since the early 1990s. Scrum makes clear the relative efficacy of your product management and development practices so that you can improve. The Agile Manifesto does not provide concrete steps. Organizations usually seek more specific methods within the Agile movement. These include Crystal Clear, Extreme Programming, Feature Driven Development, Dynamic Systems Development Method (DSDM), Scrum, and others.
Scrum is a method for teams working together in order to develop a product. Product development, using Scrum, occurs in small pieces, with each piece building upon previously created pieces. The advantages of building products in small steps enables team to respond to feedback and changes, enhances creativity and reassures that only what is needed will be build.

[1] Architecture Descriptions in ISO/IEC/IEEE 42010: http://www.iso-architecture.org/ieee-1471/ads
[2] http://www.sei.cmu.edu/architecture/definitions.html
[3] https://www.scrum.org/

Scrum is a simple framework for effective team collaboration on complex projects and provides a small set of rules that create just enough structure for teams to be able to focus their innovation on solving what might otherwise be an insurmountable challenge.
Scrum's early advocates were inspired by empirical inspect and adapt feedback loops to cope with complexity and risk. It emphasizes decision making from real-world results rather than speculation. Time is divided into short work cadences, known as sprints, typically one week or two weeks long. The product is kept in a potentially shippable (properly integrated and tested) state at all times. At the end of each sprint, stakeholders and team members meet to see a demonstrated potentially shippable product increment and plan its next steps.
Scrum's incremental, iterative approach, as shown in Figure 1 and Figure 2 trades the traditional phases of "waterfall" development for the ability to develop a subset of high-value features first, incorporating feedback sooner.

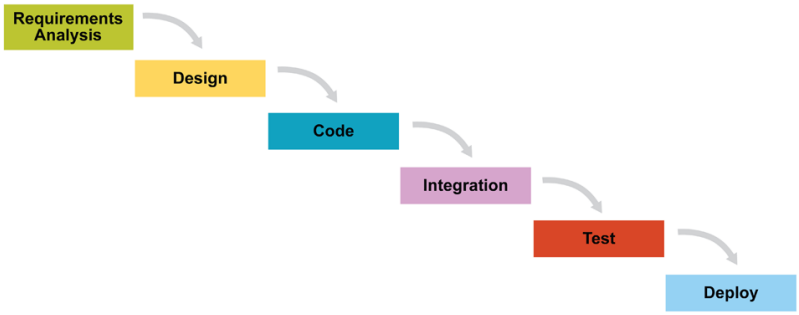

Figure 1: Traditional "waterfall" development

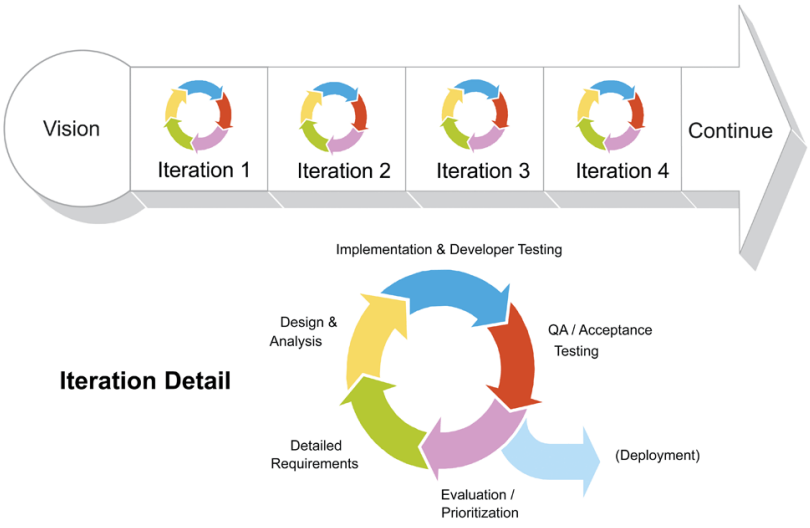

Figure 2: Scrum iterative process

For building complex products, Scrum provides structure to allow teams to deal with that difficulty. However, the fundamental process is incredibly simple, and at its core is governed by three primary roles:

1) **Product Owners** determine what needs to be built.
2) **Development Teams** build what is needed and then demonstrate what they have built. Based on this demonstration, the Product Owner determines what to build next.
3) **Scrum Masters** ensure this process happens as smoothly as possible, and continually help improve the process, the team and the product being created.

While this is an incredibly simplified view of how Scrum works, it captures the essence of this highly productive approach for team collaboration and product development. When interdependencies arise, Scrum's feature teams must learn to use team self-organization principles to coordinate with other teams.
Methodology to be used for software development will be scrum based on the high level (regular telcos between the development teams for the integration) while each development team can follow other software development methodologies internally.

2.2. Software Architecture Styles

Architecture styles[4] are high-level patterns and principles that provide an abstract framework for a family of systems. When many applications share the same structure and the relationships between the parts are very similar, we call it an "architecture style".
Advantages of understanding of architectural styles are:

- providing a common language
- providing of opportunities for technology independent higher level conversations, including patterns and principles, without getting into specifics

The following table lists the major focus areas for organizing of architectural styles and the corresponding architectural styles.

Category	Architectural styles
Communication	Service-Oriented Architecture (SOA), Message Bus
Deployment	Client/Server, N-Tier, 3-Tier
Structure	Component-Based, Object-Oriented, Layered Architecture

Table 1: Focus areas of architectural styles

In Table 2, descriptions of the common architectural styles identified in Table 1 are extended.

Architectural style	Description
Client/Server	Separates the system into two applications, client program initiates contact with a separate server program (usually on a different machine) for a specific function or purpose. The client exists in the position of the requester for the service provided by the server.
Component-Based Architecture	Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.
Layered Architecture	Partitions the concerns of the application into layers (stacked groups) so that changes can be made in one layer without affecting the others.
Message Bus	An architecture style that prescribes use of a software system that can receive and send messages using one or more communication channels, so that applications can interact without needing to know specific details about each other.
N-Tier / 3-Tier	3-Tier is a client–server/layered architecture in which the presentation, the application processing, and the data management are logically separate processes with each process being located on a physically separate computer. N-Tier is a generalization where more "tiers" (layers) are introduced.

[4] https://hasanalyazidissa.wordpress.com

Object-Oriented	A design paradigm based on division of responsibilities for an application or system into individual reusable and self-sufficient objects, each containing the data and the behaviour relevant to the object.
Service-Oriented Architecture (SOA)	Refers to applications that expose and consume functionality as a service using contracts and messages.
Representational State Transfer (REST)	This is a kind of client-server architectural style where the clients initiate requests to the servers and the servers return appropriate responses. Requests and responses convey the representations of resources where a resource can be anything that may be addressed.
Domain Driven Design	An object-oriented architectural style focused on modelling a business domain and defining business objects based on entities within the business domain.

Table 2: Common architectural styles

3. BOUNCE reference architecture

3.1. Stakeholders

The general groups of stakeholders involved or concerned about the BOUNCE project are the patients, the healthcare experts (oncologist, nurse, social-worker, psychologist), the software developers and the model providers. While each stakeholder might have different concerns or requirements over the BOUNCE platform, some of them might share in parallel more than one role. For example, a technical partner may both be a model provider and a software developer. A definition of the BOUNCE target groups and actors can be found in Deliverable D1.2.

3.2. User scenarios

Usage scenarios contribute a value in guiding the conversation during the design process, giving it context and scope. They indicate what to include, exclude, how wide, how deep to go, when to stop and they provide variations to test the design. User scenarios can be used during many stages of a system development, being associated with different objectives. Used at the analysis stage, they can prevent costly error corrections at later stages of the development. At the current stage, user scenarios will serve as a guiding tool to identify, preview and analyse the functionalities of the BOUNCE system, as well as to determine the technical requirements, both functional and non-functional, of the system being developed.

The interaction steps of BOUNCE end users with BOUNCE system throughout the breast cancer treatment continuum, e.g. the collection of different types of data and the resilience assessment after diagnosis and at regular visits, as identified during the user requirements procedure (reported in D1.2) are summarized in Figure 3.

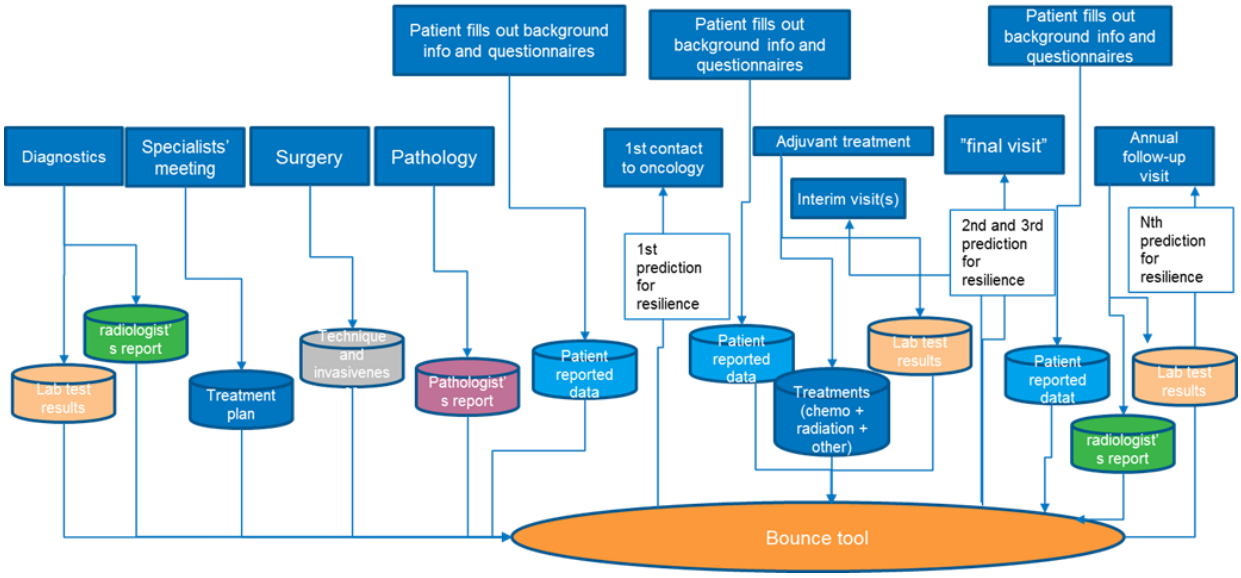

Figure 3: Steps in the BOUNCE care path

The following BOUNCE User Scenarios have been identified, and reported in detail in D1.2, per end users of the BOUNCE system (oncologist, nurse, social-worker, psychologist, patient, developer):

1. **Oncologist/nurse/social worker:** Assesses the need for referral to the psychological Team/Unit.
2. **Oncologist/nurse/psychologist/social worker:** Assesses patient progress on psychological functioning/well-being and resilience levels.
3. **Oncologist/social worker:** Assesses likely impact of patient biomedical and psychological characteristics and resilience levels on overall adaptation to illness.
4. **Psychologist/social worker:** Assesses patient resilience levels and/or psychological well-being in order to inform the patient and the medical team.
5. **Psychologist:** Assesses patient need for psychological/counselling intervention.
6. **Psychologist:** Design optimal intervention strategies, tailored to patient needs and current health status, and/or evaluate the progress of an ongoing psychological intervention.
7. **Patient:** Provide necessary information at first login and at predefined time intervals.

BOUNCE has identified six end users (oncologist, nurse, social worker, psychologist, patient, developer) and seven user scenarios that have been described in D1.2. From the technical point of view, user scenarios 1, 2, 3, 4, 5 and 6 refer to the BOUNCE Decision Support Tool. This tool produces (a) an overall “resilience predictor” score, and (b) scores for specific psychological variables that are important for resilience and adaptation to cancer. Since the six user groups (oncologist, nurse, social worker, psychologist, patient, developer) interact with the platform in the same way we group them in one user (“health professionals”). Apart from the health professional and the patient, three more end users are needed for the realization of the integrated BOUNCE platform. The first is the developer/modeler that can create, update or delete statistical, machine learning and mechanistic models. The second is the hospital administrator who must assure that data from patients, collected with the Noona tool, is properly anonymized by the Data Anonymization tool before it is pushed into the BOUNCE data lake. The third end user is the BOUNCE administrator that aspires to safeguard the (internal and external) datasets coming from the discrete and distributed data information sources are clean

and complete. For the three new end users we also introduce the IT user scenarios (following the same template as of the user scenarios in D1.2) in section 3.2.1.

3.2.1. Information technology user scenarios

3.2.1.1. Developer (modeler) user scenario

The following two technical use cases are identified for managing the content of Model Repository (MR).

3.2.1.1.1. Create content in MR

***Who** is the end-user of the MR:* **The modeler / developer**
***What** does the user want to accomplish with the MR?* **To *create* new content in the MR**. This includes (but is not limited to) model general information, pertinent files (executable, documentation, etc.) and input/output parameters.
***How** is the user going to achieve his/her goals?* **1.** The user accesses the MR either by clicking specific links available in the main BOUNCE web interface or directly, by entering the MR main URL into their web browser. **2.** The user successfully logins by using Username and Password (or creates a new account). **3.** The user begins the sequence for creating a new model by accessing the model general information input form. **4.** The user completes the fields appearing in the model information uploading form (title, description, comment, version, etc.), then submits the form to store its data in the MR **5.** The user opens the parameter information uploading form and sequentially adds all pertinent parameters. For each parameter, the user completes the pertinent fields (name, description, data type, data range, default value etc.), then submits the form to store its data in the MR. **6.** The user proceeds to the file uploading form that allows the linking of the model to a set of files e.g. the executable file. The user completes the fields pertaining to the title, description, type of file etc. and chooses the file to be uploaded from a browsing window produced by the OS, then submits the form to store its data in the MR (File is stored in a folder of a designated folder system in the OS). **7.** The user opens the form which allows the association of the model with a set of references and completes the corresponding fields, then submits the form to store its data in the MR.
Additional functionalities or interactions: N/A

3.2.1.1.2. View/update/delete/download existing content for MR

***Who** is the end-user of the MR:* **The modeler / developer**
***What** does the user want to accomplish with the MR?* **To *view/update/delete/download* the content in the MR.** This includes (but is not limited to) model general information, pertinent files (executable, documentation, etc.) and input/output parameters.
***How** is the user going to achieve his/her goals?* - **Common Steps** **1.** The user accesses the MR either by clicking specific links available in the main BOUNCE web interface or directly, by entering the MR main URL into their web browser.

2. The user successfully logins by using Username and Password (or creates a new account).
3. The user presses the button '*Browse the content of the Repository*' in order to view all the models that are stored in the repository, along with their parameters, corresponding references, etc.
4. The user selects the desired model.

- **Case:** *Download a model*

5. The user chooses the '*Download the model*' option.
6. The system creates a compressed zip containing all files represented by the objects associated with selected model.
7. The system sends the compressed zip to the user.

- **Case**: *View / update* a model (e.g. add a new file or parameter, etc.)

5. The user chooses the '*Show me the parameters of this model*' option.
6. The user chooses the '*Edit this parameter*' option and the pertinent edit form is presented.
7. The user fills in all necessary information in the corresponding fields (e.g. name, description, data type, default value etc.).
8. The user clicks on the '*Save parameter*' button.
9. The same procedure applies for all entities (descriptive information, files, etc.)

- **Case**: *Delete* a model related entity

5. The user chooses the '*Show me the parameters of this model*' option.
6. The user chooses the '*Delete this parameter*' option and the pertinent edit form is presented.
7. The same procedure applies for all entities (descriptive information, files, etc.)

- **Case**: *Delete* a model

5. The user chooses the '*Delete this model*' option.

Additional functionalities or interactions: N/A

3.2.1.2. Data anonymization from the hospital administrator user scenario

***Who** is the end-user of the Data Anonymization tool?* **The Hospital Administrator**
***What** does the user want to accomplish with the Data Anonymization tool?* Given that data received from the Noona tool are personalized, they need to be pseudonymized before storage in the BOUNCE central data repository. Information about the patient and the link between the pseudonyms and the patient stays in the hospital.
***How** is the user going to achieve his/her goals?* *Running the tool for the first time* 1. The hospital administrator receives example data from the Noona tool in a JSON or XML structure. The example data will have the exact structure that will use Noona for the subsequent data batches.

2. The hospital administrator identifies the data fields that should be anonymized and the anonymization algorithm to be used. The appropriate configuration parameters of the tool are saved for future usage.
3. The Data anonymization tool is executed and the example dataset is properly anonymized
4. The result file is examined by the hospital administrator to ensure that proper anonymization has been achieved.

Running the tool after the first time

1. The hospital administrator receives the data from the Noona tool in a JSON or XML structure.
2. The hospital administrator executes the Data Anonymization tool based on the configuration identified when running for the first time.
3. The administrator reviews the generated anonymized data to ensure that they are properly anonymized.
4. The data are uploaded to the BOUNCE central data repository

Additional functionalities or interactions: N/A

3.2.1.3. Data cleaning user scenario

***Who** is the end-user of the Data Cleaning tool?* **The BOUNCE Administrator**

***What** does the user want to accomplish with the Data Cleaning tool?*

Data Cleaning is the process that aspires to safeguard that the (internal and external) datasets coming from the discrete and distributed data information sources are (to the extent possible) clean and complete. It aims at providing the processes that will detect and correct (or remove) inaccurate or corrupted datasets containing incomplete, incorrect, inaccurate or irrelevant data elements with the purpose of replacing, modifying or deleting these data elements, also known as “dirty” data. Data cleaning as a process within the context of BOUNCE needs to take place prior to the check-in of the data in the BOUNCE data lake, since data need to be clean prior to the analysis. Towards this end, in order to safeguard data quality, proper curation and provenance, responsible for data cleaning is the BOUNCE administrator.

***How** is the user going to achieve his/her goals?*

Common steps:

1. Manually uploading datasets that s/he would like to clean
2. Defining and editing the structure of the datasets uploaded from different data providers.
3. Defining specific validation rules (e.g. data type, value range, etc.) per different variable and per different provider, thus being able to define different cleaning rules for different data providers
4. Defining corrective cleaning actions per constraint violation, per variable, and per data source / provider.
5. Defining corrective missing data handling actions per variable, and per data source / provider.

The BOUNCE administrator should also be able to have a quick visual reference (through proper logs and visualisations) of the cleaning actions that have been performed.

Additional functionalities or interactions: N/A

3.3. High Level architecture

BOUNCE aims to build an open architecture to maximise the benefits of combining technologies and data from different partners and organisation. The architecture will be constructed based on an iterative incremental process of software development. Short iterations will help keep quality under control by driving to a releasable state frequently, which will prevent the project from collecting a large backlog of defect correction work. Refinements of the architecture will take place during the whole lifetime of the project driven by the iterative feedbacks from all stakeholders. The general BOUNCE architecture is envisioned as a framework, which integrates several building blocks oriented to support/predict the resilience of women with breast cancer. The building blocks are organized in three tiers:

i. the semantic tier based on hybrid architecture which contains data extraction, transformation and serving the applications tier,
ii. the applications tier which contains the technology components such as biological and medical modelling, rule based decision support system and psycho-emotional models
iii. the security tier, a privacy framework able to handle user privacy and data security

The high-level architecture of BOUNCE is shown in Figure 4, while in the following sub-sections we describe the three tiers.

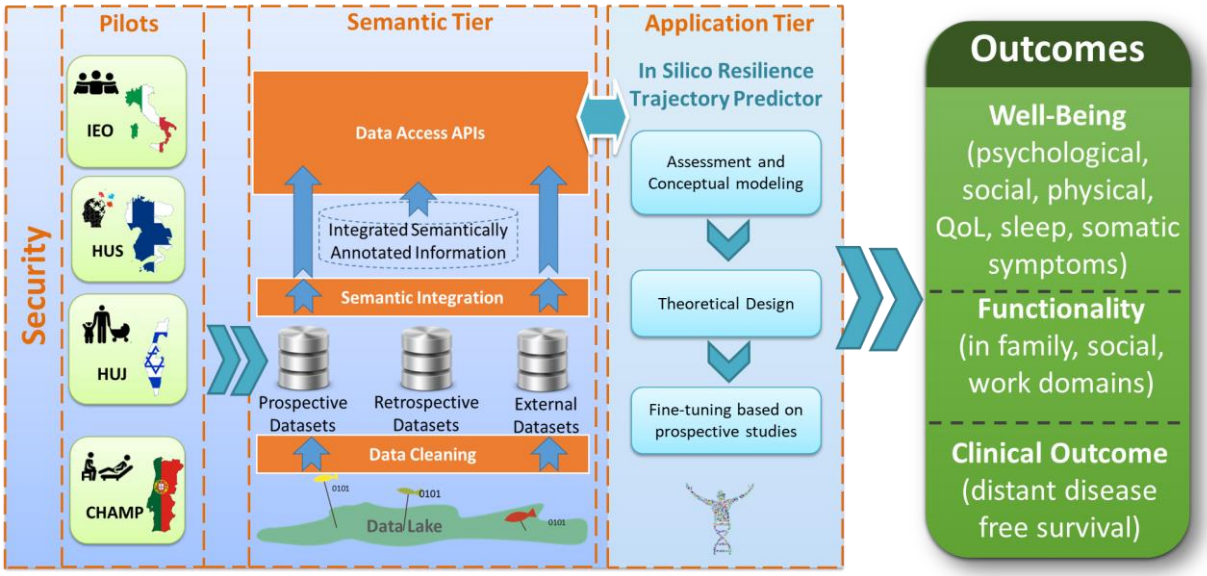

Figure 4: BOUNCE high-level architecture.

3.3.1. Semantic tier

The semantic tier of BOUNCE will provide a unified, homogenised view over the underlying data sources. A detailed view of the Semantic tier is shown in Figure 5 and consists of the following components: the data lake, the data cleaning tools, the cleaned databases, the semantic integration tool and the APIs. Below we will describe each one of those components in detail.

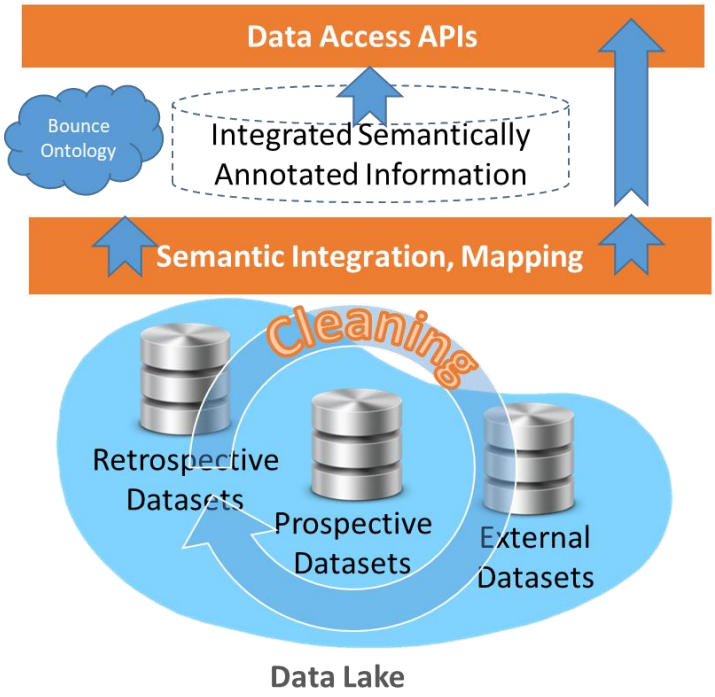

Figure 5: Initial Semantic Tier Architecture of BOUNCE

- **Data Lake:** At the bottom layer, all inserted data will be staged in a repository with multiple databases forming a data lake. Data will include the anonymized retrospective and prospective data provided by HUS, IEO, HUJ and CHAMP and imported data from external registries. The data will reside in the data lake in their original formatting to be further cleaned and processed. The data residing in the data lake will then be cleaned using appropriate data cleaning tools. Cleaning will entail missing value handling, identification of erroneous records etc. The cleaned data, will again be staged in the data lake.
- **Semantic Integration & Mapping**: Using the semantic model, i.e. the BOUNCE ontology that will be constructed within the BOUNCE project (WP3), all available data will be modelled, mapped and semantically uplifted to triples. There will be available the possibility to perform ETL, exporting selected data to an RDF triple store, or of the real time integration through a virtual data source.
- **Data Access APIs:** All data, both integrated and those residing at the data lake will be accessible through various APIs that will respect all necessary security and confidentiality requirements established by the security layer of the BOUNCE project.

3.3.2. Applications (models) Tier

Broadly speaking, statistical and artificial intelligence/machine learning modelling are the two distinct modelling approaches that have been recruited and serve as the basis for the modelling work of BOUNCE. These techniques are being used for the identification of potential correlations extracted from retrospective BOUNCE data (among various biomedical, psychological, functional and quality of life aspects and parameters) but are also to be used for the modelling needs of the prospective BOUNCE pilot study.

Obviously, combinations of the previous main approaches will also be used for particular problems reported in detail in deliverable D4.1 (served by the model fusion service of BOUNCE).

More precisely, the ultimate goal of the Application Tier is the development of an overall prediction model and a resilience trajectory predictive model that will serve as the core of a decision support system for predicting the resilience evolution in women with breast cancer throughout the cancer continuum. To this end, preliminary correlations between heterogeneous information sets related to resilience will be extracted and hypotheses to be used as input to the model will be defined. Psychological, clinical (including treatment information), socio-demographic and lifestyle data will be considered. Concrete hypotheses will be formulated based on the correlations to be extracted. Both retrospective data from the BOUNCE clinical partners and literature information will be exploited. Finally, mechanistic multiscale modelling will be used primarily for the development of the health literacy - educational tool *i.e.* the "Educational Oncosimulator" designed to raise awareness about the utmost importance of the patient's sticking to the medical treatment schemes and schedules prescribed by the clinician whenever this takes place.

Prediction models

Two models are proposed (i) an *overall/general*, and (ii) a *resilience-trajectory specific*.

The Overall Prediction Model: Based on the hypothesis that previous medical and psychological factors may determine or at least predict subsequent well-being and health outcomes, this overall model includes the following hypothesized significant relationships:

1. Outcomes may be predicted by
    - the variables (or their interactions) assessed at the immediately previous time-point
    - the factors (or their interactions) assessed at all previous time-points and baseline
    - the interactions between variables assessed at different time-points
2. Potential interplay between the medical and psychological variables of the study.
    - For example, it is possible for a medical event or changes in a significant biomedical index to lead to subsequent changes in illness self-regulation and psychological outcomes.
    - Likewise, it is possible and, therefore, will be examined whether there is an interaction between psychological variables, such as illness representations or self-efficacy, and crucial medical variables, such as therapy side-effects, regarding their impact on health outcomes.
3. The process of adaptation to illness is probably characterized by a choreography of dynamic changes in the several aspects of this process. In other words, it is possible that changes in the basic self-regulatory spiral (illness representations, coping behaviors, reappraisals etc.) are associated with corresponding changes to the ways that facilitating factors (such as, self-efficacy) change over time, and for both of these patterns of change to be associated with variations in health outcomes. Hence, the examination of the potential impact of the dynamic changes in different variables (or a set of the most important of them) on corresponding changes in health outcome scores is needed. Some paradigmatic pathways are illustrated below:
    - Medical events – (changes in) self-efficacy – (changes in) adherence to medical advice – (changes in) health outcomes
    - Medical events – (changes in) optimism – (changes in) coping behavior (e.g., positive attitude) – (changes in) health outcomes
    - Illness emotional representations – emotion regulation – social support – health outcomes

- Illness representation of treatment control – fear of recurrence – distress – health outcomes
- Sense of coherence – self-rated health – coping behavior – health outcomes

The Resilience Trajectory Prediction Model is supplementary to the previous main one and aims to identify:

a) the (different types of) trajectory over time (i.e., months 1 to 18) for the main outcomes (or the composite outcome indexes) in order to detect the time-point(s) that is (are) critical for inclusion in (or exclusion from) a specific type of trajectory (specifically, the resilience trajectory);
b) the possible transitions from one specific type of trajectory to another;
c) the critical factors that precede inclusion in a specific type of trajectory (e.g., changes in important variables; significant events).

3.3.2.1. Statistical Procedures and Data Mining Approaches

Temporal data mining, time-series prediction, sequence classification methods, clustering time-series data, and temporal association rules will be used to develop and validate the predictive model. Mediation, moderation and moderated mediation analyses will have a central role in the statistical methodology. In addition, any other methodology appropriate for time series prediction including autoregressive models, chaotic time series, Markov chains and deep learning will be considered for the optimal prediction of resilience in Bounce.

Because the performance of predictive models can be considerably deteriorated when non-relevant features (i.e. variables) are included during the training phase of the model, feature selection will take place prior to model building. Different feature selection techniques or combinations of them will be applied, namely filter methods, wrapper methods and embedded methods. Filter type methods that will be considered in the present study also include well-known statistical tests and procedures such as Student's t-test, Analysis of variance (ANOVA), Mann–Whitney U test, Kruskal-Wallis test, correlation, regression analyses etc. Given the temporal/sequential nature of data, BOUNCE will also make use of methodologies specific for time series analysis (e.g. repeated measures, autocorrelation analysis etc).

The methodological approach is described below:

- Univariate (e.g. t-test, chi-squared test, Mann–Whitney U test, Spearman's rank correlation coefficient etc.) and multivariate techniques (e.g. logistic regression, correlation-based feature selection, sequential forward selection, sequential backward elimination, decision trees, naive Bayes etc.) will be performed to identify features of importance.
- Intelligent pattern recognition analysis of an individual's context will be applied to allow the identification of established behaviors and eventually, cause and effect relationships.
- Given the sequential nature of recorded data, association analysis techniques, able to handle both co-occurrence and dynamic relationships in multivariate time series data, will be utilized.
- Temporal data mining will enable the identification of dynamic patterns or predictive rules in long-term trajectories and, eventually, will allow drawing conclusions regarding the associations between the patient's context - indicating resilience to BC - and the clinical health outcomes, and vice versa.
- The identification of groups of patients with similar characteristics will be investigated based upon classification and clustering analysis (e.g. random forests and supporting vector machines, hierarchical clustering, k-means,).

3.3.2.2. Other data analyses approaches

Descriptive statistics (n, mean, standard deviation, median, maximum and minimum, graphical representation) will be used to summarize the continuous data. Discrete measures will be summarized using counts, percentages and graphical representations. Bivariate charts will also be produced whenever desired. Descriptive statistics consist a standard preliminary step in data analysis which allow for a meaningful presentation of data and can complement the interpretation of the results of statistical analyses. Furthermore, they can help identify outliers, imbalance data and guide subsequent choices. Variables will be checked for normality, homoscedasticity etc. to guide the selection of most appropriate analysis technique. Variable transformation may also take place (e.g. log transformation or transformation of continuous variables into categorical based on recognized cut-off values) if needed/desired.

3.3.3. Security Tier & security by design

Security ensuring protection of personal / sensitive information in BOUNCE needs to be addressed through a two-fold procedure, which includes: 1) Data Access Control, and 2) Security of data across their whole lifecycle, from storage, to transit and to use. It should be noted however that the holistic security approach is not a standalone component, but rather a set of technologies and tools that are utilised within the components of the BOUNCE platform in order to enable cross-platform security.

3.3.3.1. Access Control

Access control in general includes authorization, authentication, access approval, and audit. Authentication and access control are often combined into a single operation, so that access is approved based on successful authentication, or based on an access token. Authentication methods and tokens include passwords, biometric scans, physical keys, electronic keys and devices, and other means. Within the context of BOUNCE, the Noona system which will be used for data collection at the pilot sites has already established the necessary data access control policies and mechanisms, according to which patient identity information is stored and shared with the research nurse and the treating doctor. Noona is hosted in the Amazon Web Services (AWS) platform with all the users' connections passing through the Web Application Firewall of the AWS while the databases are hosted in Amazon Relational Database Service not within the same server instance. Those data will be exported to the pilot sites, properly anonymized and sent to the central BOUNCE data management infrastructure. Thus, within the context of BOUNCE, access control needs to be realised only upon the central BOUNCE data management infrastructure, through granting access to the data to the corresponding tools illustrated in the forthcoming section.

3.3.3.2. Data Lifecycle Security

With regards to data security throughout the whole lifecycle of the data exploitation, BOUNCE consortium has preliminarily examined and as appropriate will develop and deliver safeguards regarding three main security aspects:

1) Security of data in storage
2) Security of data in transit, or data in motion
3) Security of "Data in Use"
4) Last but not least, security of technical interfaces (e.g. REST) amongst the various BOUNCE components will also be considered and adopted, minimising the risk associated

with the exploitation of the operation of various BOUNCE components from external malicious components.

Data-At-Rest/Data-In-Storage Security

Security of data in storage is the first of the three parts of the data lifecycle dealt with in the context of the project and is used as a complement to the terms data in use and data in transit which together define the three states of digital data. Data that falls under this category could include files stored on local or cloud hard drive. Data in storage security refers to the preservation of the security, privacy and integrity of data that is stored physically in any digital form. It deals with any type of security around the storage architecture and the data stored on it. BOUNCE will make use of checksum to ensure security and integrity control of the data in storage. Checksum is a small-sized datum derived as the outcome of the cryptographic hash function or checksum algorithm on a block of data or file. This outcome is utilised to identify data corruption errors or modifications and overall data integrity since even small changes will produce a different outcome. Several other solutions, like the use Symmetric Encryption Algorithms, Asymmetric Encryption Algorithms and Attribute-Based Encryption, have been primarily evaluated by the consortium in order to address the security, privacy and integrity of the data stored in the BOUNCE Data Store, however given that the data will already be transmitted and subsequently stored anonymised, these technologies do not seem to be ideal for the data-processing intensive BOUNCE ecosystems due to efficiency problems (e.g. significant latency) that they can introduce within the data analysis process.

Data-In-Transit Security Schemes

Data in transit, or data in motion, is data actively moving from one location to another such as across the internet or through a private network. Data protection in transit is the protection of this data while it's traveling from network to network or being transferred from a local storage device to a cloud storage device – wherever data is moving, effective data protection measures for in transit data are critical as data is often considered less secure while in motion. Concerning the security of data in transit or data in motion, BOUNCE will evaluate the provision of data encryption via Secure Sockets Layer (SSL) and Transport Layer Security (TLS) at the RPC layer.

Data-In-Use

"Data in Use" is all data not in an at-rest state, which is kept only one particular node in a network (for example, in resident memory, or swap, or processor cache or disk cache, etc. memory). This data can be regarded as "secure" if and only if:

a) access to the memory is rigorously controlled (the process that accessed the data off of the storage media and read the data into memory is the only process that has access to the memory, and no other process can either access the data in memory, or man-in-the-middle the data while it passes through I/O)
b) regardless of how the process terminates (either by successful completion, or killing of the process, or shutdown of the computer), the data cannot be retrieved from any location other than the original at rest state, requiring re-authorization.

Although the BOUNCE consortium has already identified a list of candidate technologies, such as Homomorphic Encryption and Verifiable Computation, given that the data will already be transmitted and subsequently stored anonymised, these technologies do not seem to be ideal for the data-processing intensive BOUNCE ecosystem due to efficiency problems (e.g. significant latency) that they can introduce within the data analysis process. The approach to be adopted is currently under discussion.

Security of Technical Interfaces

The holistic security approach also covers the security aspects for the technical interfaces (e.g. REST) provided by the platform. This includes the interfaces provided by the components of the platform in regards to the authorisation, authentication and access approval mechanisms. Through the preliminary analysis, the consortium decided to introduce a token-based authentication with JSON Web Token (JWT)[5]. JWT is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

The following table presents the holistic security approach of BOUNCE platform for the data lifecycle security as described above.

Security Aspect	Proposed Approach	Adopted Approach	Remarks
Access Control	Several approaches safeguarding access control including DAC, MAC, RAC, RBAC and more	Customised RBAC for access control.	Policies will need to be defined on different access rights based upon the users' roles.
Data in Storage	Checksums for data integrity, Symmetric Encryption Algorithms, Asymmetric Encryption algorithms and Attribute-Based Encryption	Usage of checksums for data integrity.	Encryption Algorithms seem to not be ideal for the data-processing intensive BOUNCE ecosystem because of performance and efficiency issues that they may introduce.
Data in Transit	Secure Sockets Layer (SSL) and Transport Layer Security (TLS)	Provision of SSL and TLS data encryption and authentication at the RPC layer.	SSL and TLS encryption are the de-facto standard in the security of data in transit.
Data in Use	Homomorphic Encryption, Verifiable Computation	Currently none of the technologies has been adopted	Encryption Algorithms seem to not be ideal for the data-processing intensive BOUNCE ecosystem because of performance and efficiency issues that they may introduce.
Technical Interfaces	Token-based authentication and authorisation mechanism with JSON Web Token	JSON Web Token will be introduced.	The implementation and integration of JSON Web Token mechanism is an ongoing activity.

Table 3. Holistic Security Approach summary

[5] JSON Web Tokens, https://jwt.io

4. BOUNCE components, interfaces and diagrams

4.1. Procedures

4.1.1. Components and interfaces reporting

In order to standardize documenting of the development process of the BOUNCE tools, two templates has been created for the description of the components and their programming interfaces:

- A template for description of the tools (Appendix A – Template for component description), which should contain mainly an information about provided and expected interfaces for each application as well as third party frameworks used for implementation (incl. their licences for usage).
- A template for detailed specifications of the provided REST interfaces (Appendix B – Template for REST service specification).

4.1.2. Sequence reporting

Behaviour and interaction of the BOUNCE components in time sequence is presented as a set of sequence diagrams, which show how processes operate with one another and in what order. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development. For identifying the use cases for BOUNCE platform, the scenarios described in the deliverable D1.2 were analysed. The sequence diagrams depict the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenarios. Sequence diagrams show, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner[6].

4.1.3. Deployment reporting

The *physical* deployment of BOUNCE components and details about required hardware and software are presented in deployment diagrams. A typical deployment diagram[7] shows what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI). The nodes appear as boxes, and the artefacts allocated to each node appear as rectangles within the boxes. Nodes may have sub-nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

Below we report the descriptions of all components based on the template (Appendix A – Template for component description), sequence interfaces and deployment interfaces (including UML component diagrams). Furthermore, the mature components and the under development components will report the detailed rest interfaces (Appendix B – Template for REST service

[6] https://en.wikipedia.org/wiki/Sequence_diagram

[7] https://en.wikipedia.org/wiki/Deployment_diagram

specification). Since the implementation of some components (e.g. decision support system) has not started yet, we cannot provide details about their REST APIs. Nevertheless, the Appendix B – Template for REST service specification will be used during the development of all the components ensuring a smooth final integration. For that reason, we consider the deliverable D5.1 of BOUNCE a live document that will be updated during the software lifecycle.

4.2. Personal Health System

The Personal Health System Noona is a platform already established in Finland and currently expanding to other countries as well. Within BOUNCE the platform will be made available for the collection of the data in the pilots.

4.2.1. Component and interfaces

Name	Noona		
Related use cases	All		
Due Date	Implemented.		
Location source code	Confidential.		
Responsibilities/ Functionality	Noona is responsible for collecting the questionnaire data and enabling the clinic to be able to export the data. The main goal is to provide the psychological scales and other relevant eCRF forms to be filled in by the patients.		
Provided Interfaces [Interfaces implemented by the component]	**Interface**	**Type**	**Description**
	Patient UI	GUI	Patient can track their questionnaires
	Clinic UI	GUI	Clinic can request a data export
Required Interfaces [Interfaces used by the component]	**Interface**	**Type**	**Description**
	Data Anonymizer API	REST	REST API to (semi-)automatically anonimyze and push data to the BOUNCE data lake/semantic tier
Implementation	Java, Kotlin, Angular, Typescript		

Third party Libraries

BACKEND	
Library	**License**
Apache Avro	Apache-2.0
Jersey	CDDL Version 1.1
JAX-RS API	CDDL Version 1.1
libphonenumber	Apache-2.0
java-apns	BSD-3-Clause
Jackson JSON Processor	Apache-2.0
Jackson Databind	Apache-2.0
Joda Time	Apache-2.0
Eclipse Link	Eclipse Public License - v 1.0, Eclipse Distribution License - v 1.0
PostgreSQL JDBC driver	BSD-3-Clause

Apache Commons Beanutils	Apache-2.0
jsonrpc4java	MIT
jade4j	MIT
Apache PDFBox	Apache-2.0
Apache Commons FileUpload	Apache-2.0
Metadata Extractor	Apache-2.0
Amazon AWS Java SDK KMS	Apache-2.0
GeoIP2 Java API	Apache-2.0
FRONTEND	
Library	**License**
Angular: common, compiler, core, forms, http, platform-browser, platform-browser-dynamic, router, upgrade	MIT
AngularJS	MIT
AngularJS Animate	MIT
AngularJS Cookies	MIT
AngularJS JSON-RPC	WTFPL
AngularJS Moment	MIT
AngularJS Route	MIT
AngularJS Sanitze	MIT
AngularJS Scroll	MIT
Angular Tooltips	MIT
AngularJS Touch	MIT
AngularJS UI-router	MIT
Angular-toastr	MIT
Bootstrap	MIT
Bootstrap datepicker	Apache-2.0
core-js	MIT
D3	BSD-3-Clause
JQuery	MIT
JQuery transform	MIT
Moment	MIT
Moment timezone	MIT
ng-showdown	BSD-3-Clause
@ngrx/core	MIT
@ngrx/store	MIT
@ngrx/store-devtools	MIT
Photoswipe	MIT
RxJS	Apache-2.0
Showdown	BSD-3-Clause
stacktrace-js	The Unlicense
loadsh	MIT
Zone.js	MIT
cookie-storage	MIT
fastclick	MIT
tinygradient	MIT

ngrx-store-logger	MIT
url-join	MIT
Cordova:cordova-plugin-camera,cordova-android,cordova-ios, cordova-plugin-device, cordova-plugin-file, cordova-plugin-media, cordova-plugin-media-capture, cordova-plugin-network-information, cordova-plugin-splashscreen, cordova-plugin-statusbar, cordova-plugin-vibration, cordova-plugin-whitelist	Apache-2.0
Cordova-hot-code-push	MIT
Cordova ActionSheet	MIT
Cordova AppVersion plugin	MIT
Cordova Badge Plugin	Apache-2.0
Cordova Browser-Sync Plugin	Apache-2.0
Cordova DatePicker	MIT
Cordova Firebase	MIT
Cordova-plugin-ios-disableshaketoedit	MIT
Cordova Screen Orientation Plugin	Apache-2.0
Cordova SecureStorage	MIT
Cordova SSL Certificate Checker	MIt
Cordova ES6-Promise	MIT

4.2.2. Sequence diagram

Noona provides different user interfaces for different roles. Figure 6 provides the sequence diagram of the patient, while Figure 7 provides the sequence diagram of the healthcare professional.

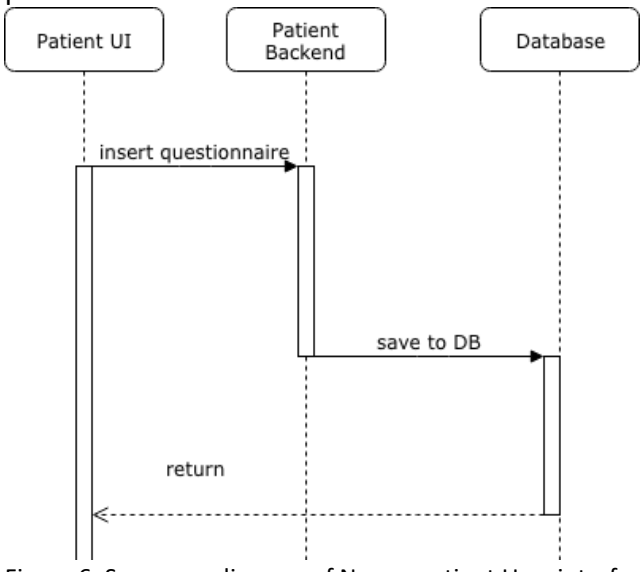

Figure 6: Sequence diagram of Noona patient User interface

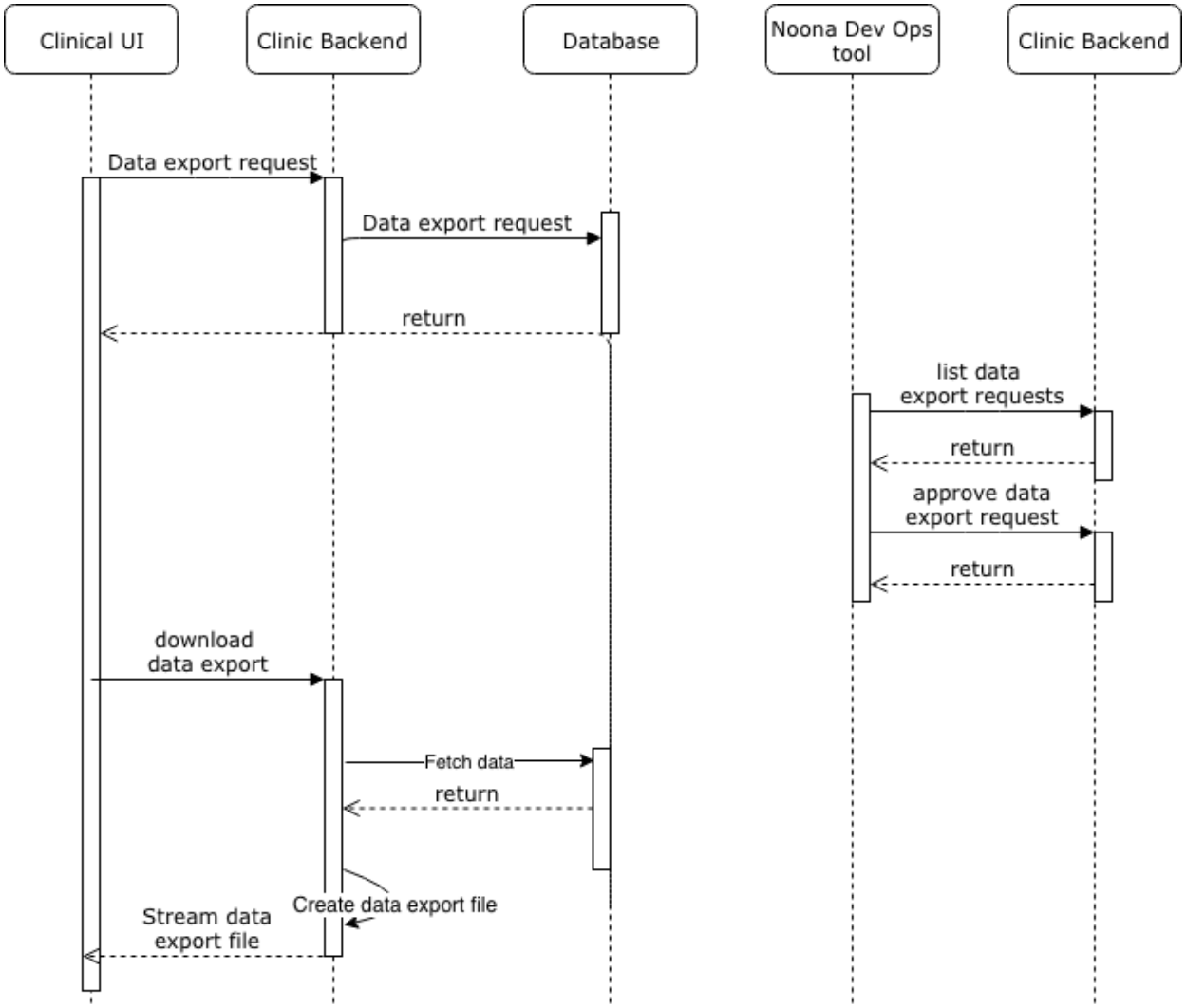

Figure 7: Sequence diagram of Noona healthcare professionals User interface

4.2.3. Deployment diagram

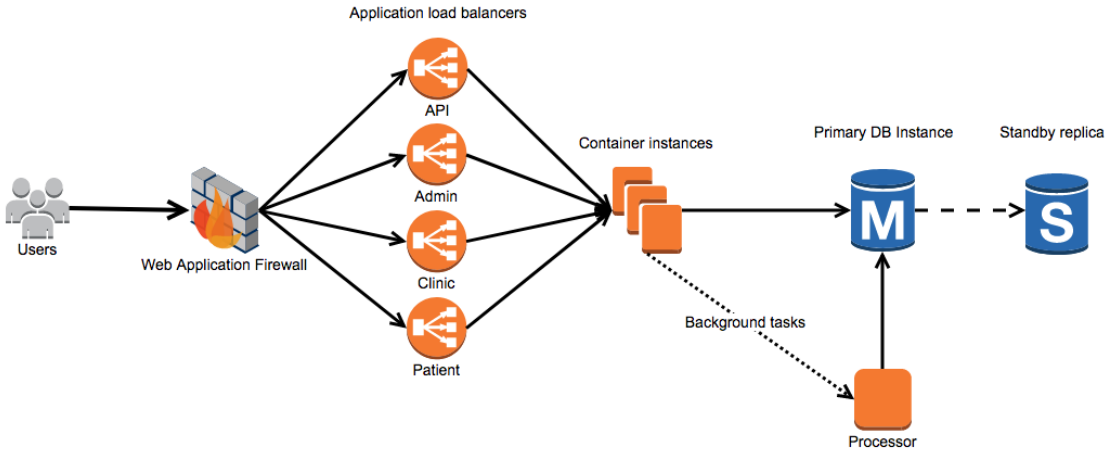

Figure 8: Deployment diagram of Noona

4.3. Authentication, authorization and role management

4.3.1. Component and interfaces

The Access Controller is the component responsible for controlling the access to the different resources of the platform, like the platform's datasets, added value services and internal components offered through the platform such as the decision support. The Access Controller functionality follows the main principles defined by the Attribute-Based-Access-Control[8] (ABAC) paradigm, which is a logical access control model, where access to objects is controlled by evaluating rules (policies) against the attributes of the entities (subject and object) actions and the environment relevant to a request.
More specifically, the Access Controller has two main functionalities:

- It controls the access to the platform's datasets, added value services and internal components.
- It manages the process of requesting and granting access to the platform's aforementioned resources.

The implementation of the Access Controller which follows the ABAC paradigm comprises of three main modules:

- The Policy Enforcement Point (PEP), which acts as the endpoint that receives the access requests to the different resources.
- The Policy Information Point (PIP), which retrieves the required attributes of the resources and the active policies.
- The Policy Decision Point (PDP), which evaluates the access requests based on the resources' attributes and the policies and produces a decision to grant or deny access.

4.3.2. Sequence diagram

The execution flow of examining the access rights to a resource is described as follows:

- The Policy Enforcement Point receives a request for accessing a resource.
- The type of the request is examined and the suitable function of the Policy Decision Point is called to resolve the request.
- The Policy Decision Point examines which resources are being requested and by whom and calls the Policy Information Point to get the required attributes and the policies for this kind of resource.
- The Policy Information Point gathers all the information from the platform's storage and the stored policies.
- The Policy Decision Point evaluates the policies one-by-one, according to the received attributes. If a policy evaluation is successful the access is granted, otherwise, if none of the policy evaluations is successful, then the access is denied.
- The Policy Enforcement Point receives the decision and grant or deny access accordingly.

[8] https://en.wikipedia.org/wiki/Attribute-based_access_control

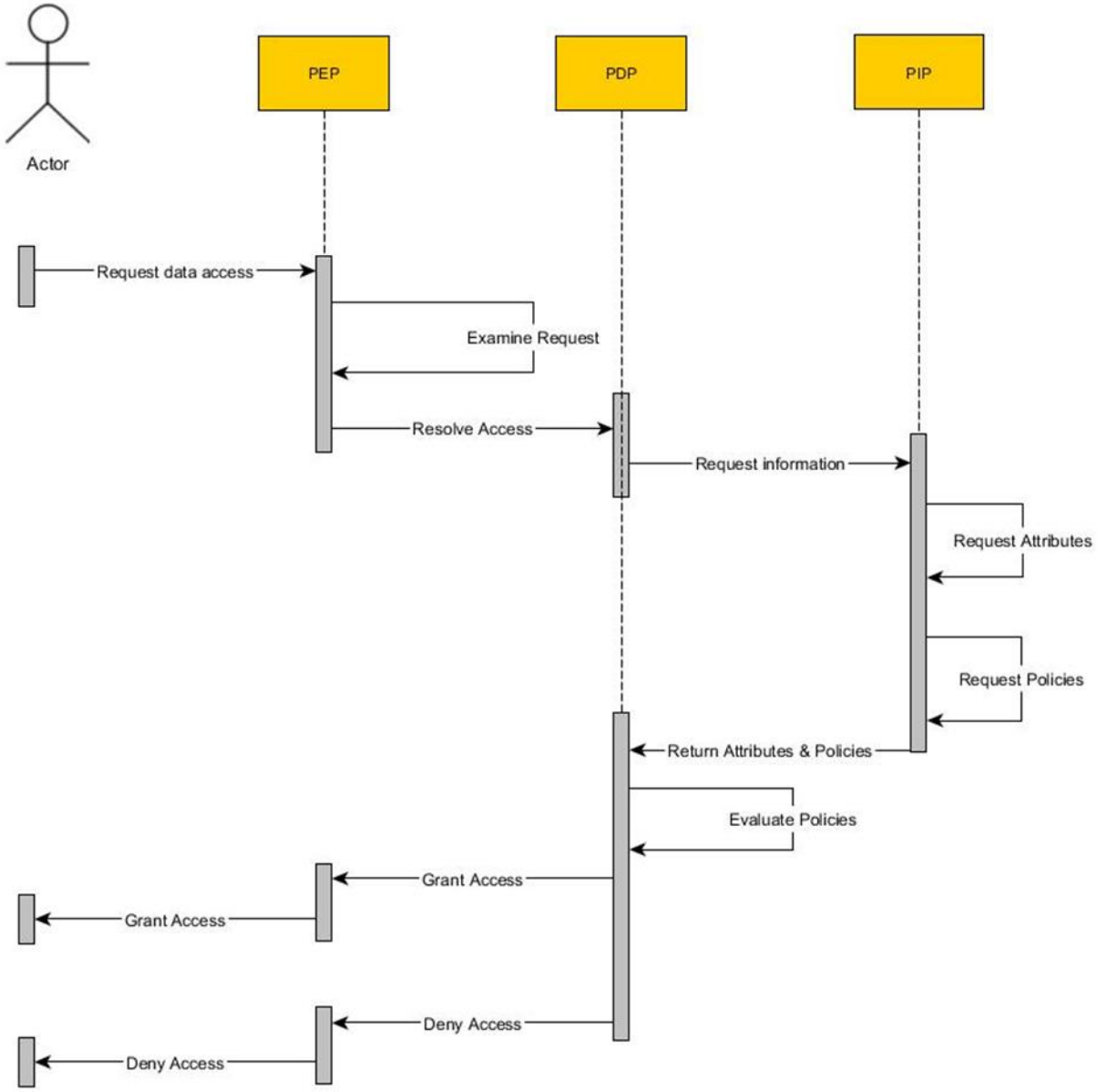

Figure 9: Access Controller sequence diagram

Figure 9 displays the execution flow for the functionality of the Access Controller.

4.3.3. Deployment diagram

As aforementioned, the BOUNCE Access Controller follows the ABAC paradigm and comprises of three main modules:

- The Policy Enforcement Point (PEP). The PEP receives the requests for accessing the different resources and also receives the decision to grant or to deny access to these resources accordingly.
- The Policy Information Point (PIP). The PIP retrieves all the required attributes of the resources and the active policies according to the request made to the PEP and provides it to the Policy Decision Point.
- The Policy Decision Point (PDP). The PDP holds the business intelligence of the Access Controller and is responsible for receiving all the required attributes of the resources and

the active policies according to the request made to the PEP, from the PIP, and to evaluate the access requests based on the resources' attributes and the policies in order to make the decision to grant or deny access and forward it to the PEP.

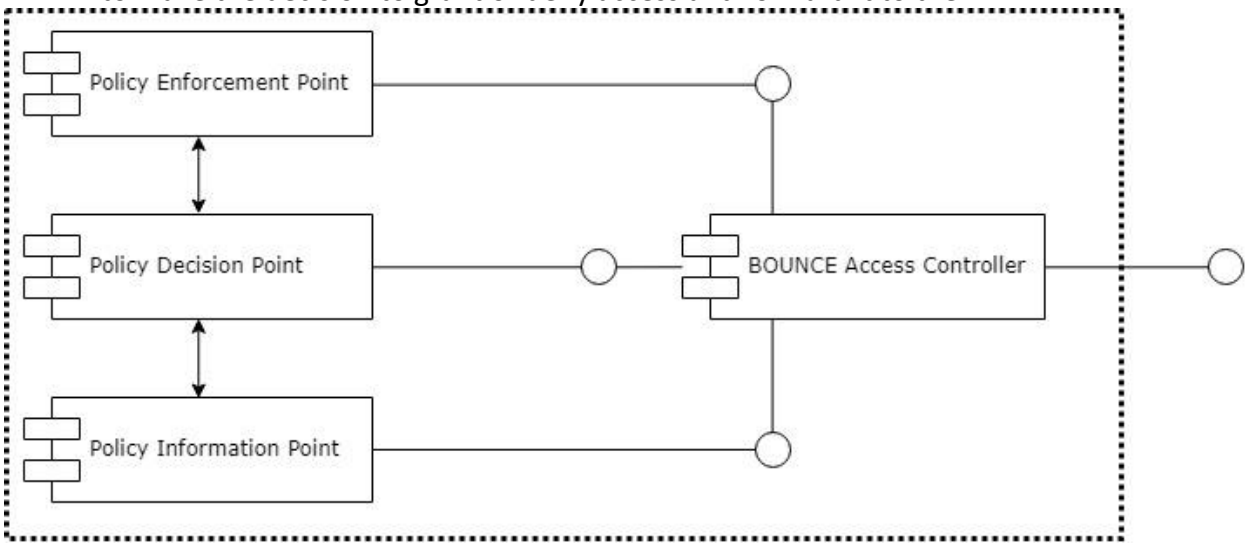

Figure 10: Access Controller deployment diagram

4.4. *Data Anonymizer*

The data anonymizer will anonymize the data collected from Noona before pushing them into the data layer/lake.

4.4.1. Component and interfaces

Name	**Data Anonymizer**		
Related use cases	User Scenarios: 1, 2, 3, 4, 5, 6		
Due Date	Month 24		
Location source code	Currently under development, FORTH's internal repository		
Responsibilities/ Functionality	The data anonymizer will anonymize the data collected from Noona before pushing them into the data layer/lake.		
Provided Interfaces [Interfaces implemented by the component]	**Interface**	**Type**	**Description**
	Login Interface	GUI	User is required to login to the system using their credentials
	Data go through anonymization process	PROCESS	A process will provide anonymization of the patients' data.
Required Interfaces [Interfaces used by the component]	**Interface**	**Type**	**Description**
	Authentication API	REST	An API for ensuring user authorization to use Temporary Research Supporting Tool
	Noona Repository API	REST	API for patient's data management
Implementation	Java Web Application		

Third party frameworks	• Spring Framework (Apache 2.0 license). • Hibernate (GNU Lesser General Public License).

4.4.2. Sequence diagram

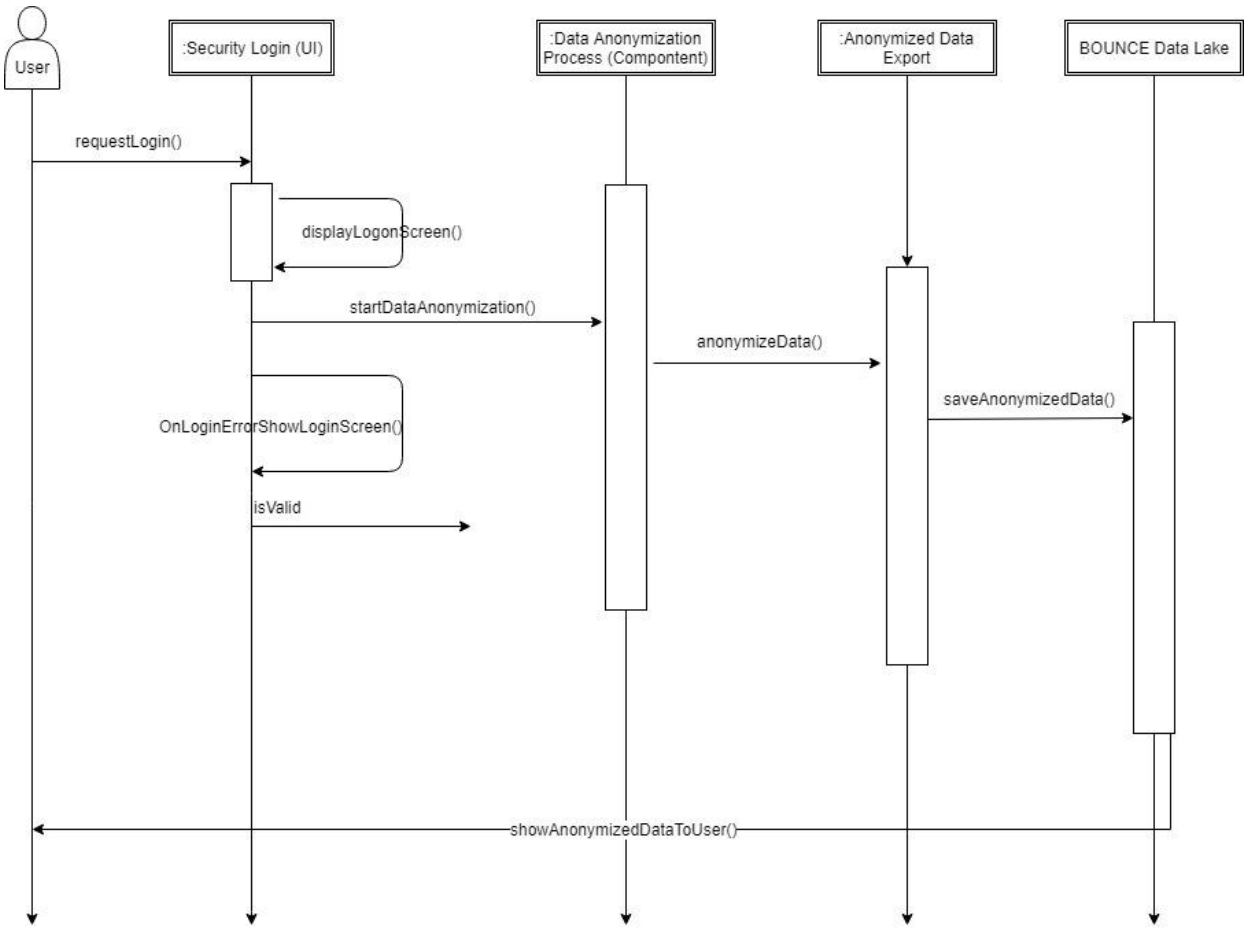

Figure 11: Data Anonymizer Sequence Diagram

4.4.3. Deployment diagram

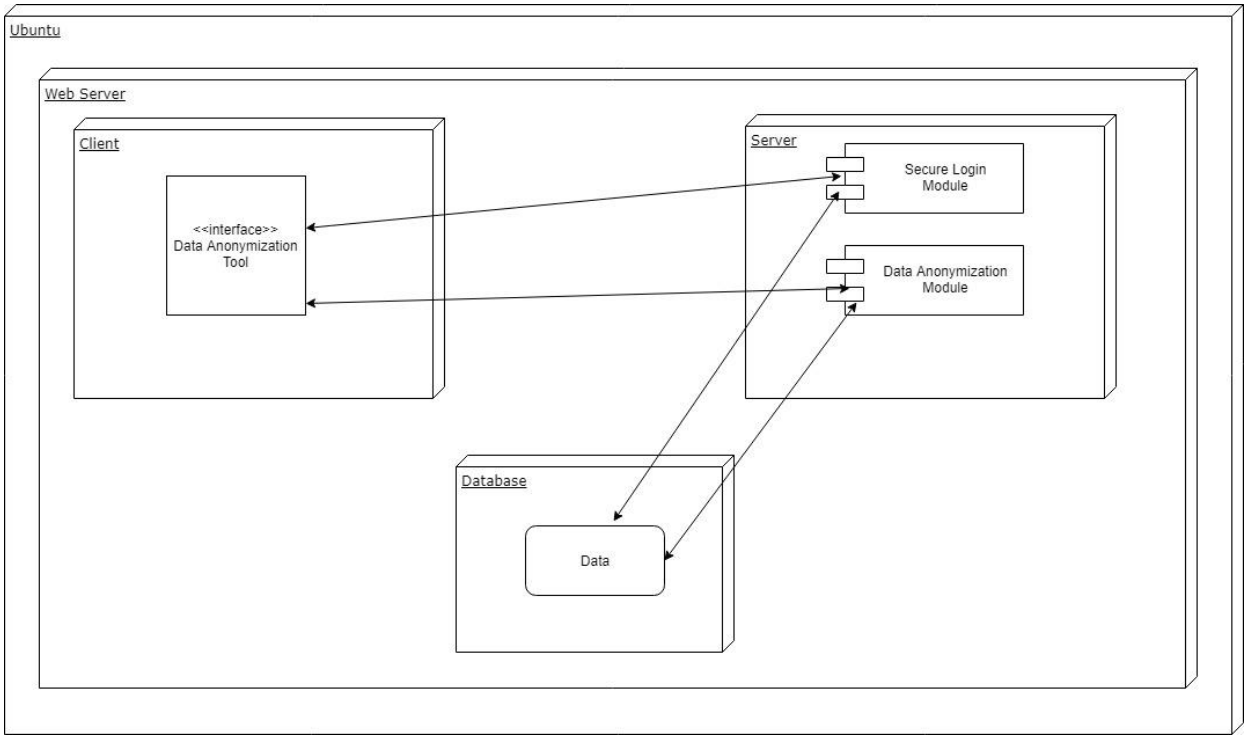

Figure 12: Data Anonymizer Deployment Diagram

4.5. Data Aggregator & Harmonizer

4.5.1. Component and interfaces

Name	Data Aggregator & Harmonizer		
Related use cases	All		
Due Date	First Prototype M24, Final Version M36,		
Location source code	FORTH's internal repository		
Responsibilities/ Functionality	This component will be responsible for providing access to the available, semantically uplifted data of the BOUNCE project. The input of this component will be the BOUNCE ontology (D3.2 and D3.3), the available data sources and the corresponding mappings of the sources' schemata to the ontology. Then it will provide a SPARQL endpoint for querying the integrated data. Available choices of this component is whether the integrated data should form a materialized or a virtual database with the integrated data.		
Provided Interfaces [Interfaces implemented by the component]	**Interface**	**Type**	**Description**
	SPARQL endpoint	REST API	An interface for accepting a SPARQL endpoint and answering it over the available integrated and harmonized data.
Required	**Interface**	**Type**	**Description**

Interfaces [Interfaces used by the component]	Data source APIs	REST API or java code API	An interface should be available by the underlying databases in order to be able to query them.
Implementation	This component will be developed using JAVA.		
Third party frameworks	In addition, it will relly on D2RQ[9] and OntoQ[10] ontology-based data access frameworks. Both the tools are using the Apache License V2.0		

As the implementation of this component matures, more details will be described in this deliverable about the exact API calls and the services provided.

4.5.2. Sequence diagram

The sequence diagram of eventually using the data aggregator and harmonizer is shown in Figure 13. In essence, Data Aggregator & Harmonizer accepts queries from the application tier and answers them. To answer them it either produces the required subqueries, forwards them to the underlying data sources to be answered and formulates the final results (on-line integration) or has already materialized the necessary data (off-line integration/ETL) and returns the answers based on the transformed, existing data.

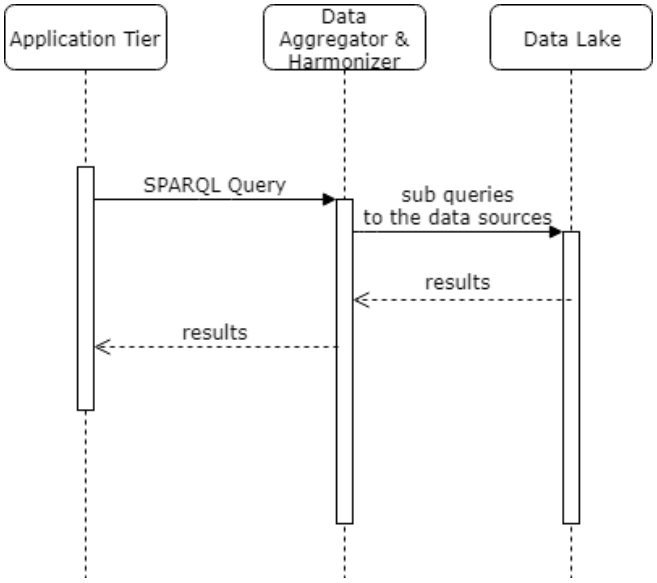

Figure 13: Data Aggregator & Harmonizer sequence diagram

4.5.3. Deployment diagram

The deployment diagram of the data aggregator and harmonizer is shown in Figure 14. Ideally, the Data Aggregator and Harmonizes should be placed in the same server with the data lake to minimize network transfer time. Then applications can be placed in another server. The applications that require access to the integrated data can query the data aggregator and harmonized, whereas the applications that require access only to a limited amount of non-integrated data can directly access them through the data lake.

[9] http://d2rq.org/

[10] http://ontop.inf.unibz.it/

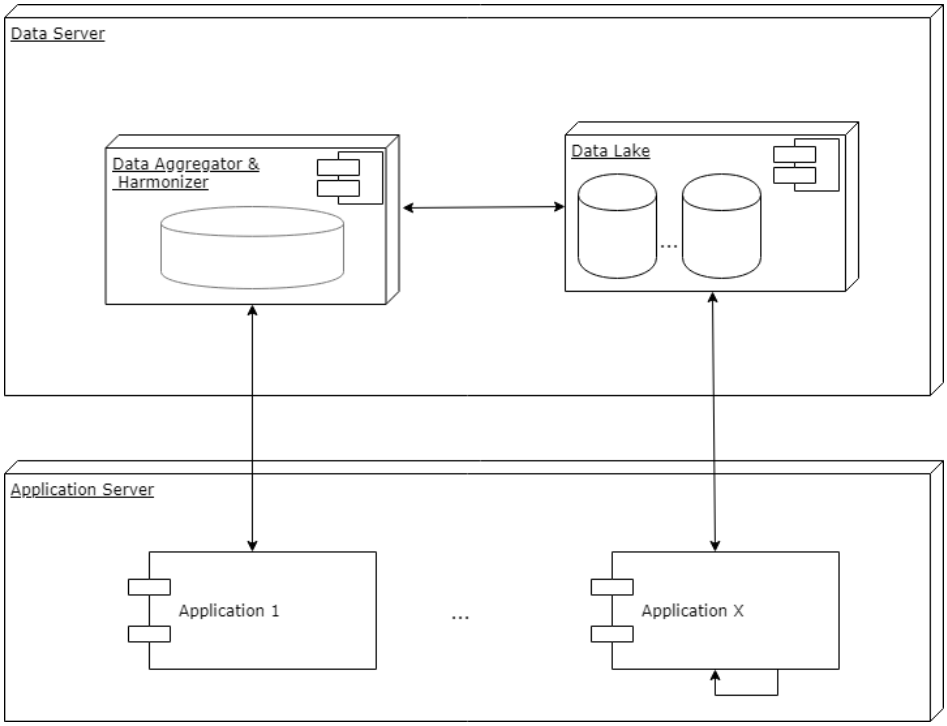

Figure 14: Data Aggregator & Harmonizer deployment diagram

4.6. Data Cleanser

4.6.1. Component and interfaces

The scope of the BOUNCE Data Cleaner is to deliver a software able to provide the assurance that the BOUNCE datasets coming from the discrete and distributed data information sources will be clean and complete, to the extent possible. The BOUNCE Data Cleaner is aiming at providing the processes that will detect and correct (or remove) inaccurate or corrupted datasets containing incomplete, incorrect, inaccurate or irrelevant data elements with the purpose of replacing, modifying or deleting these data elements, also known as “dirty” data.
The BOUNCE Data Cleaner will implement the data cleaning workflow providing all the necessary actions towards safeguarding the storage and delivery of consistent datasets across the BOUNCE platform. The data cleaning workflow comprises of the following steps as illustrated also in Figure 15:

1. **Validate Data:** Data validation ensures that a program operates on clean, correct and useful data. Errors are reported for data elements that do not comply with the specified rules.
2. **De-Cleanse Data:** Data de-cleansing ensures that the data elements for which validation errors are raised (according to pre-defined rules and conditions) are properly corrected or removed.
3. **Check Data Completeness:** The data completeness checks ensure the existence of the required/mandatory data elements on the dataset. It is mainly associated with handling missing values and filling in values according to pre-defined rules.
4. **Verify Data**: Data verification ensures that the data elements of a dataset are checked for accuracy and inconsistencies after steps like validation, de-cleansing and data completeness are done.

5. **Consolidate & Store Errors' logs:** Error logging provides history records of the errors identified and the executed corrective actions.

Figure 15. Data Cleaning workflow

The BOUNCE Data Cleaner also exposes a user interface to the administrator who will be able to perform a number of activities, as described in the table below
In terms of the scenario, the BOUNCE administrator is responsible for:

1. Manually uploading datasets that s/he would like to clean
2. Defining and editing the structure of the datasets uploaded from different data providers.
3. Defining specific validation rules (e.g. data type, value range, etc.) per different variable and per different provider, thus being able to define different cleaning rules for different data providers
4. Defining corrective cleaning actions per constraint violation, per variable, and per data source / provider.
5. Defining corrective missing data handling actions per variable, and per data source / provider.

The BOUNCE administrator should also be able to have a quick visual reference (through proper logs and visualisations) of the cleaning actions that have been performed.

Name	**BOUNCE Data Cleaner**		
Related use cases	N/A		
Due Date	First version M18, Final version M24		
Location source code	Not available yet		
Responsibilities/ Functionality	Safeguards that the (internal and external) datasets coming from the discrete and distributed data information sources are (to the extent possible) clean and complete. It provides the processes that will detect and correct (or remove) inaccurate or corrupted datasets containing incomplete, incorrect, inaccurate or irrelevant data elements with the purpose of replacing, modifying or deleting these data elements, also known as "dirty" data.		
Provided Interfaces [Interfaces implemented by the component]	**Interface**	**Type**	**Description**
	Initial Interface	GUI	Greet user and prompt them to go to login screen to enter their credentials
	Login Interface	GUI	User is required to login to the system using their credentials
	Upload Dataset	GUI	User is prompted to upload dataset (and define the provider)
	Edit Data Structure	GUI	User is prompted to define and edit the structure of the datasets uploaded from different data providers.

	Define Validation Rules	GUI	User is prompted to define specific validation rules (e.g. data type, value range, etc.) per different variable and per different provider
	Define Cleaning Rules	GUI	User is prompted to define corrective cleaning actions per constraint violation, per variable, and per data source / provider
	Define Missing Values Handling Rules	GUI	User is prompted to define corrective missing data handling actions per variable, and per data source / provider
	Dashboard	GUI	User can have a quick visual reference (through proper logs and visualisations) of the cleaning actions that have been performed
Required Interfaces [Interfaces used by the component]	**Interface**	**Type**	**Description**
	Authentication API	REST	An API for ensuring user authorization to use the system
	Data repository API	REST	API for data management (retrieve data)
Implementation	N/A		
Third party frameworks	N/A		

4.6.2. Sequence diagram

Figure 16 illustrates the data cleaning workflow along with the procedures and processes supported by the BOUNCE Data Cleaner. Every step of the sequence flow is elaborated in the forthcoming paragraphs.

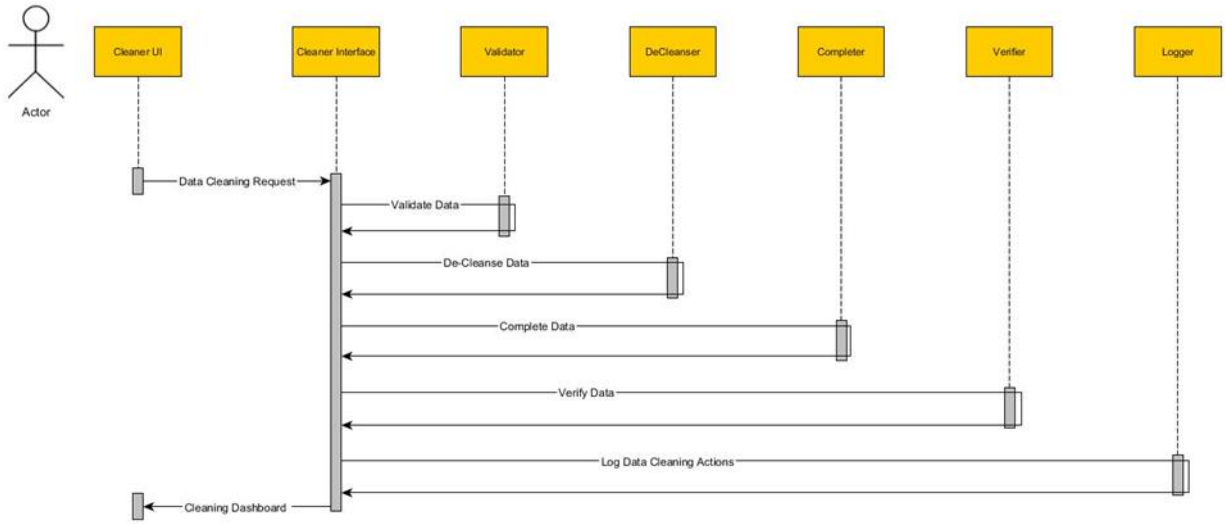

Figure 16: Data Cleaner Component Sequence Diagram

As depicted in Figure 16, the BOUNCE Cleaner is exposing one interface to the rest of the components of the BOUNCE platform, the Data Cleaner interface (IDataCleaner). Through this

interface, the data cleaning workflow can be initiated by pulling / pushing the incoming information data retrieved / received from / to the BOUNCE Data Cleaner. Once the BOUNCE Data Cleaner receives the incoming data, it will initiate the first step of the workflow which is the data validation step. Through the IValidator internal interface, the Data Validator service is invoked to perform the error detection due to lack of conformance to the specified set of constraints and in response the list of errors is returned.
Following the data validation step, the Data De-Cleanser service is invoked via the IDeCleanser internal interface. The Data De-Cleanser service is receiving the incoming data along with the list of errors identified by the Data Validator service. The Data De-Cleanser service performs the necessary actions to correct or remove the data elements of the incoming data marked with errors. As a result, the updated incoming data is returned along with the list of actions performed on each data element.
In the next step, the Data Completer service is invoked by the ICompleter internal interface. The Data Completer service performs automated filling of missing values according to pre-defined rules and conditions, and returns the updated data and the list of actions performed on the corresponding data elements.
Upon performing the data completion step, the Data Verifier service is triggered via the IVerfiier internal interface. The Data Verifier service ensures that the updated data is now error free, accurate and consistent as required by the BOUNCE platform. The results of the verification are returned in response to the verification request.
Finally, once all previous steps have been executed towards the completion of the data cleaning workflow, the Errors' Logger Service is triggered via the ILogger internal interface. The Errors' Logger service stores the information in records. These records will give the BOUNCE moderator (should such a role eventually be supported) the opportunity to see exactly which errors were identified by the BOUNCE Data Cleanser, and what were the corrective / cleansing actions performed by the rest of the internal services on the incoming data. Once the record for the incoming data is returned, the BOUNCE Data Cleaner returns the updated (cleaned) information data to the BOUNCE repository as a response to the invocation of the IDataCleaner interface.

4.6.3. Deployment diagram

The BOUNCE Data Cleaner comprises of several internal sub-components, each one providing an interface to support the interconnection and collaboration with the rest of the internal sub-components or other internal components of the BOUNCE platform. Figure 17 illustrates the internal architecture of the Data Cleaner component, displaying all internal sub-components along with their interfaces.

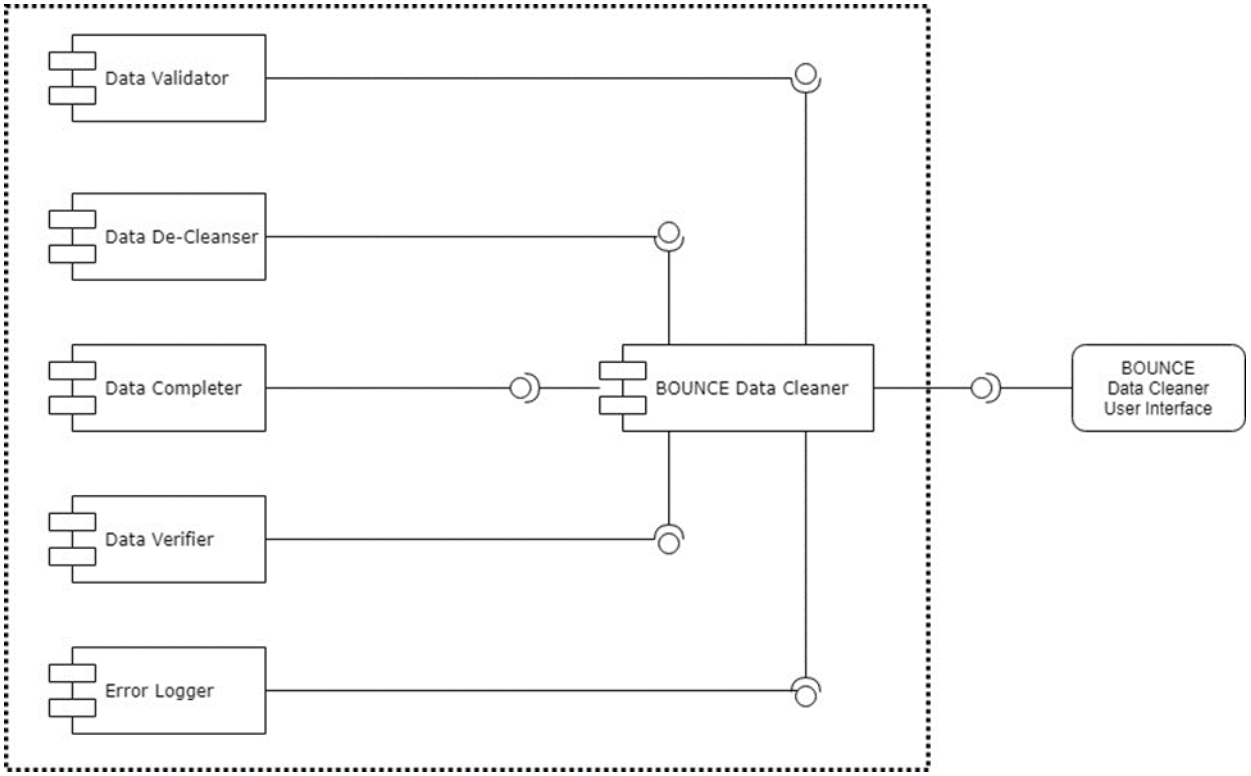

Figure 17: Data Cleaner Component design

As depicted in the internal architecture diagram, the BOUNCE Data Cleaner Service comprises of the following main services:

1. Data Validator Service
2. Data De-Cleanser Service
3. Data Completer Service
4. Data Verifier Service
5. Errors' Logger Service

The **Data Validator** will perform data validation of the incoming information data with the purpose of identifying errors associated with the conformance to specific set of constraints. This service acts as a safeguard that the data measures compare to defined business rules or constraints set.

The **Data De-Cleanser** service will perform the de-cleansing step of the data cleaning workflow. In particular, this service performs the necessary corrections or removals of errors identified by the Data Validator Service, and depending on the nature of the error, automated cleansing of the information will be performed based on a predefined set of rules, or a moderator (should such a role eventually be supported) is notified to take further actions.

The **Data Completer** service will safeguard the appropriateness and completeness of the incoming information data. This service safeguards conformance to mandatory fields and required attributes of the dataset based again on a predefined set of rules or the desired configuration. On these rules and configuration parameters the actions taken by the service are determined for either automated filling of the missing values by interpolation or extrapolation techniques or notification of a moderator (should such a role eventually be supported) to take the necessary actions.

The **Data Verifier** service will ensure that all data will accurately be corrected or completed and the dataset will eventually be error free.

The **Errors' Logger** Service will undertake the responsibility of keeping in records the reported error logs from the rest of the services and the actions taken towards the data cleaning of the incoming information data. Due to the fuzzy nature of the data cleansing workflow it is mandatory that all error events thrown or actions performed during the processing of the incoming information data are held in records. The BOUNCE moderator (should such a user role eventually be supported) should be able to check over the log entries created, the results of the data cleansing workflow along with detailed information on the error or the action that occurred.

4.7. Temporary Research Supporting Tool

4.7.1. Component and interfaces

Name	**Temporary Research Supporting Tool**		
Related use cases	User scenarios: 1, 2, 3, 4, 5, 7		
Due Date	First version M18, Final version M24		
Location source code	Not available yet		
Responsibilities/ Functionality	Temporary Research Supporting Tool is the short-term internal project tool to facilitate data exploration and visualization of data and scales. The tool should be able to retrieve and visualize anonymized patient data, data about the individual scores on each scale and the combination of scores in different biomedical and psychosocial variables (coming from the current and/or possible previous assessments).		
Provided Interfaces [Interfaces implemented by the component]	**Interface**	**Type**	**Description**
	Initial Interface	GUI	Greet user and prompt them to go to login screen to enter their credentials
	Login Interface	GUI	User is required to login to the system using their credentials
	Entering Data	GUI	A user interface used by the expert for selecting data.
	Visualization of Results	GUI	A user interface will provide an overview of the analysis' results.
Required Interfaces [Interfaces used by the component]	**Interface**	**Type**	**Description**
	Authentication API	REST	An API for ensuring user authorization to use the system
	Data repository API	REST	API for data management (retrieve data)
Implementation	JavaScript, Java Web Application		
Third party frameworks	• Spring Framework (Apache 2.0 license). • Hibernate (GNU Lesser General Public License).		

As the implementation of this component matures, more details will be described in this deliverable about the exact API calls and the services provided.

4.7.2. Sequence diagram

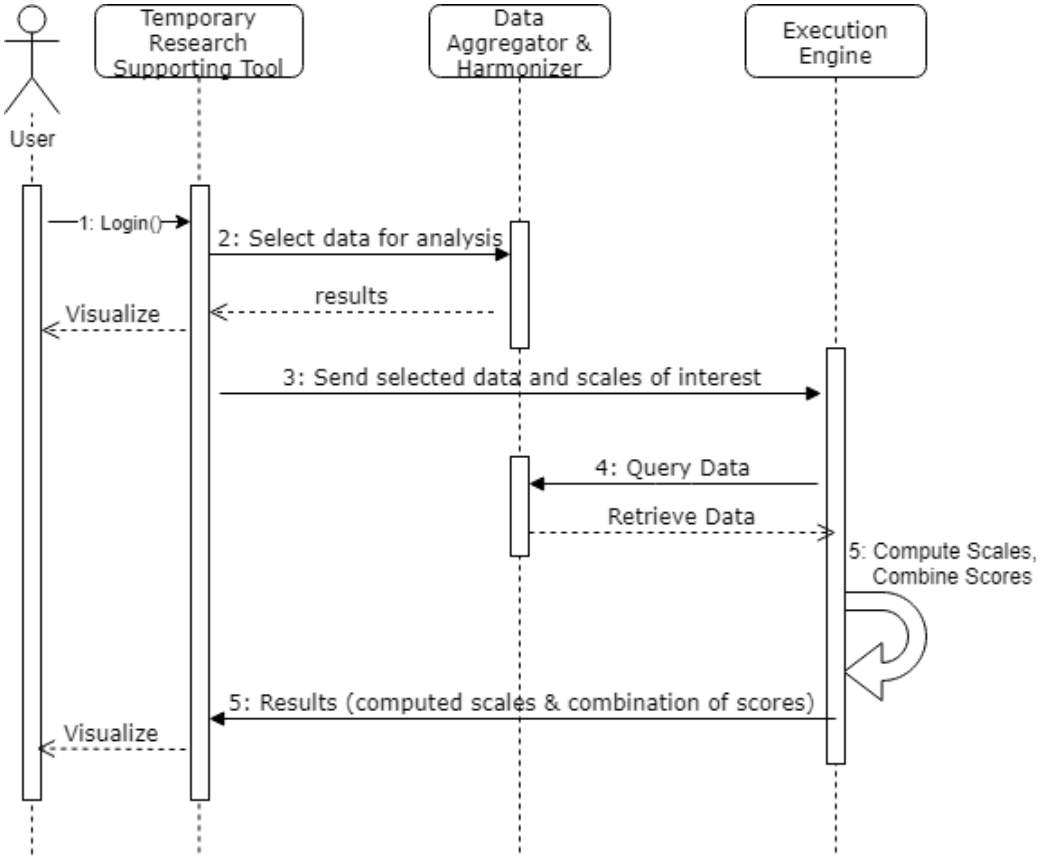

Figure 18: Temporary Research Tool Sequence Diagram

4.7.3. Deployment diagram

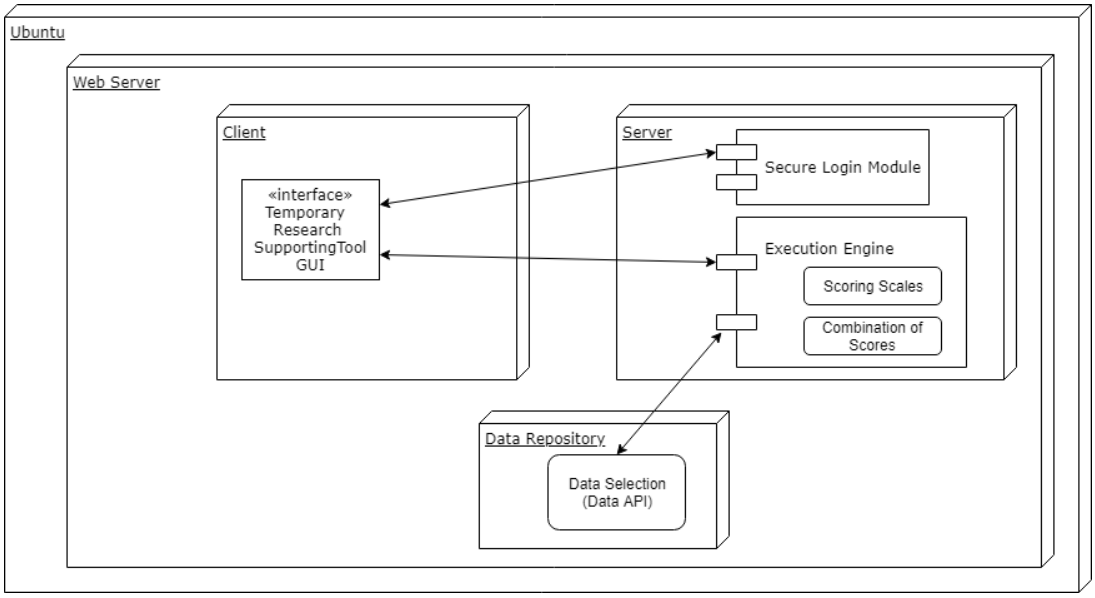

Figure 19: Temporary Research Tool Sequence Diagram

4.8. Model Repository

4.8.1. Component and interfaces

The web based BOUNCE Model Repository (MR) will store the overall prediction model and the resilience trajectory prediction model.
The key entities of the MR are the model, the parameters, the properties and the files (see Figure 20). The model entity includes all the descriptive information of a model, the parameters entity contains all the information regarding the input parameters needed for the execution of the model (data type, units, description etc.) as well as the output data of a model (description, type etc.), the property entity contains the properties that could characterize a model (e.g. statistical and machine learning techniques that were used) and the file entity contains the files linked to the model (e.g. several versions of binaries).

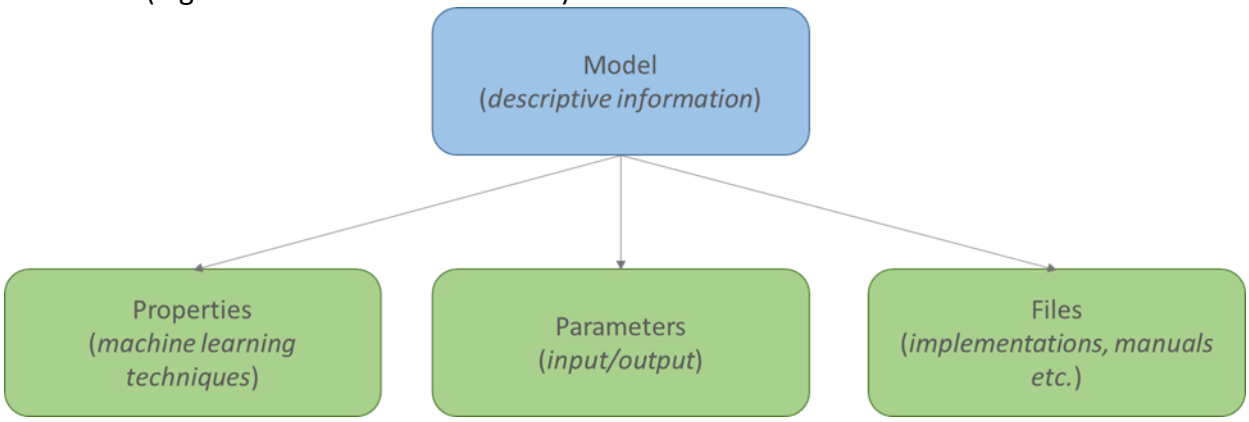

Figure 20: Key entities composing MR

The basic principles of the MR are:

- Each model has basic descriptive information. This information uniquely defines the model and differentiates it from other models.

- Each model can have one or more properties that further describes or/and classifies it.
- Each model is associated with a set of parameters.
- Each model may be associated with a set of references, which provide direct or indirect links to additional material, extending in this way the knowledge base related to the specific model.
- Every model can be accompanied by a set of files. The corresponding DB table only holds the descriptive information of the file and not the actual file data, which is stored in a designated set of folders within the file system.

Name	Model Repository (MR)		
Related use cases	User scenarios 1-6		
Due Date	*prototype*: M21, *first version*: M24, *second version*: M32 *first implementation*: M38		
Location source code	N/A		
Responsibilities/ Functionality	This is the web-based component that will permanently host the models that will be developed in the context of the BOUNCE project.		
Provided Interfaces [Interfaces implemented by the component]	**Interface**	**Type**	**Description**
	Initial Interface	GUI	Greet user and prompt them to go to login screen to enter their credentials
	Login Interface	GUI	User is required to login to the system using their credentials
	Incorrect Login Interface	GUI	User is notified that their login attempt was unsuccessful and prompted to retry
	Main Interface	GUI	User is presented with the necessary choices for CRUD procedures on the model components
	Creation Interface	GUI	For each model component (files, parameters, etc.) and/or administrative data (users, roles, etc.) the user is required to complete a list of fields in a corresponding screen and submit the form.
	Content List Interface	GUI	User is presented with a set of screens containing the MR contents. By clicking on specific buttons in each row the user can edit (and update) or delete the row.
	Edit Interface	GUI	For each model component (files, parameters, etc.) and/or administrative data (users, roles, etc.) the user is required to correct a list of fields in a corresponding screen and submit the form.
	Authentication API	REST	An API for ensuring user authorization to use MR
	DB API	REST	API for data management of MR content (data CRUD functions, file download)
Required	**Interface**	**Type**	**Description**

Interfaces			
Implementation	Python, MySQL, Javascript, HTML, CSS		
Third party frameworks	Django, jQuery		

A web-based user interface is being developed in order to allow users to interact with the repository. MR makes use of RESTful web services for its integration with the other components. The aforementioned integration will facilitate the retrieval of the MR information (model executables, descriptive information of the models, etc.). Appropriate authentication and authorization mechanisms are being implemented in order to ensure that only authorized persons have access to the content of the repository.

Component API	Model Repository API		
Version	v0.1	**Date**	12/10/18
Notes	Under development		

Method	storeModel		
POST	**Description**	Stores the basic descriptive information of the model and returns the id	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*title*: Title of the model (Required) description: Description of the model (Not required) *comment*: Comments on the model (Not required) *semtype*: URL representing semantic information about this model (not required)
		JSON Object	The JSON object required in the request body must have all the above parameters (at least the required ones)
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned by method storeModel has one key i.e. *id*, and one value, which is associated with this key.
		Example	
Method	deleteModelById		
DELETE	**Description**	Deletes the descriptive information, the files, the parameters, and property values of a model.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: Model id (required)

		JSON Object	The JSON object required in the request body must have the *id* parameter
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	N/A
		Example	
Method **GET**	getModelById		
	Description	Returns the descriptive information stored under the id (title, description, comment, semtype) and null when not existing	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: Model id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned by method getModelById has eight keys i.e. title, description, comment, semtype, created_on, created_by, modified_on and modified_by, and eight values associated with those keys.
		Example	
Method **POST**	storeParameter		
	Description	Stores the parameter information of a model and returns the id	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*model_id*: id of the model to which the parameter belongs (required) *name*: parameter name (required) *description*: Parameter description (not required) *data_type*: Parameter type e.g. number, string, file (required) *unit*: Units in which the parameter is represented (only applicable if the parameter is a number) (not required) *data_range*: Data range of the parameter (required) *default_value*: Value to be used if a parameter value is not provided to the tool *is_output*: 1 if the parameter is output, 0 if it is input (required)

			comment: Comments on the parameter (not required) *semtype*: url Representing semantic information about this parameter (not required)
		JSON Object	The JSON object required in the request body must have all the above parameters (at least the required ones)
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has one key i.e. id, and one value associated with this key.
		Example	
Method **DELETE**	deleteParameter		
	Description	Deletes a certain parameter	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: Model id (required)
		JSON Object	The JSON object required in the request body must have *id* parameter
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	N/A
		Example	
Method **GET**	getParametersByModelId		
	Description	Returns the information of all the parameters of a given model	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*model_id*: The id of the model to which the parameters belong (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The keys of the JSON object returned are as many as the different parameters belonging to the model. Each value associated with a specific key is represented by a nested JSON object. Each key of the aforementioned nested JSON object represents the column name of the parameter entity and each value of the nested JSON object

			represents the information of the corresponding column.
		Example	
Method	storeProperty		
POST	**Description**	Stores the basic descriptive information of a property and returns the id	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*name*: Name of the property (required) *description*: Description of the property (not required) *comment*: Comments on the property (not required) *semtype*: url representing semantic information about this property (not required)
		JSON Object	The JSON object required in the request body must have all the above parameters (at least the required ones)
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned by method storeProperty has one key, named id, and one value that is associated with this key.
		Example	
Method	getAllProperties		
GET	**Description**	Returns all the properties and the corresponding descriptive information stored (id, name, description, comment, semtype)	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	No parameters required
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The keys of the JSON object returned are as many as the different properties stored in the MR. Each value associated with a specific key, is represented by a nested JSON object.
		Example	
Method	getPropertyById		
GET	**Description**	Returns the descriptive information stored under the property id (name, description, comment)	

	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: Property id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned by method getPropertyById has four keys i.e. name, description, comment, semtype, and four values associated with those keys.
		Example	
Method	storePropertyValue		
	Description	Stores the value of a property for a model and returns the id	
POST	**Request**	**Authentication**	
		Content Type	JSON
		Query Parameters	*model_id*: Model id (required) *property_id*: Property id (required) *value*: Property value (required)
		JSON Object	The JSON object required in the request body must have all the above parameters
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned by method storePropertyValue has one key, named id, and one value which is associated with this key
		Example	
Method	deletePropertyValue		
	Description	Deletes the property value for a certain model	
DELETE	**Request**	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: The id of the record which holds the property value (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	
		Example	
Method	getPropertyValuesByModelId		

GET	**Description**	Retrieves all the property – value pairs for a given model	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*model_id*: The id of the model that the property-value pairs are associated with (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The keys of the JSON object returned by method getPropertyValuesByModelId are as many as the different properties that describe or/and classify the given model. Each value associated with a specific key is represented by a nested JSON object. The keys of the nested JSON object are the name, description, comment, value, semtype.
		Example	
Method	deletePropertyById		
	Description	Deletes the property of the given id and the corresponding values	
DELETE	**Request**	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: The id of the record which holds the descriptive information of the property (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	
		Example	
Method	StoreReference		
	Description	This method stores information of the reference. The reference is associated with a model.	
POST	**Request**	**Authentication**	
		Content Type	JSON
		Query Parameters	*model_id*: The id of the model with which the reference is associated (required) *title*: Title of the reference (required) *type*: Type of the reference (book, journal article, etc.) (required) *author*: Author(s) of the resource (required) *issued*: Date of formal issuance (required) *citation*: Bibliographic citation of the

			resource (not required) *doi*: Digital Object Identifier of the resource *pmid*: PubMed Identifier (not required)
		JSON Object	The JSON object required in the request body must have all the above parameters (at least the required ones)
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has one key i.e. id, and one value associated with this key.
		Example	
Method **DELETE**	deleteReferenceById		
	Description	This method deletes a specific reference	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: The id of the reference (required)
		JSON Object	The JSON object required in the request body must have the *id* parameter
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	
		Example	
Method **GET**	getReferencesByModelId		
	Description	Returns all the references of a given model	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*model_id*: The id of the model with which the references are associated (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The keys of the JSON object returned are as many as the different references that are associated with the given model. Each value associated with a specific key is represented by a nested JSON object.
		Example	
Method	storeFile		

POST	**Description**	Stores the file information and returns the id	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*model_id*: The id of the model with which the file is associated (required) *title*: Title of the file (required) *description*: Description of the file (not required) *kind*: Defines what the file is (document, source code, binary, etc.) (not required) *license*: License associated with the file (not required) *Sha1sum*: Sha1 checksum of the file (not required) *comment*: comments on the file(not required) *engine*: Engine for executing this file (not required) *file*: The actual file (blob) (required)
		JSON Object	The JSON object required in the request body must have all the above parameters (at least the required ones)
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned by method storeFile has one key i.e. *id*, and one value that is associated with this key.
		Example	
Method	deleteFile		
DELETE	**Description**	Deletes a certain file	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: File id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	
		Example	
Method	getFileById		
GET	**Description**	Returns the file.	
	Request	**Authentication**	
		Content Type	JSON

		Query Parameters	*id*: File id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	File
		JSON Object	
		Example	

Requirements that selected software should meet:

- Be free and open source.
- Have an active community that supports it by building plug-ins and extensions.
- Follow the Model-View-Controller (MVC) paradigm.

4.8.2. Sequence diagram

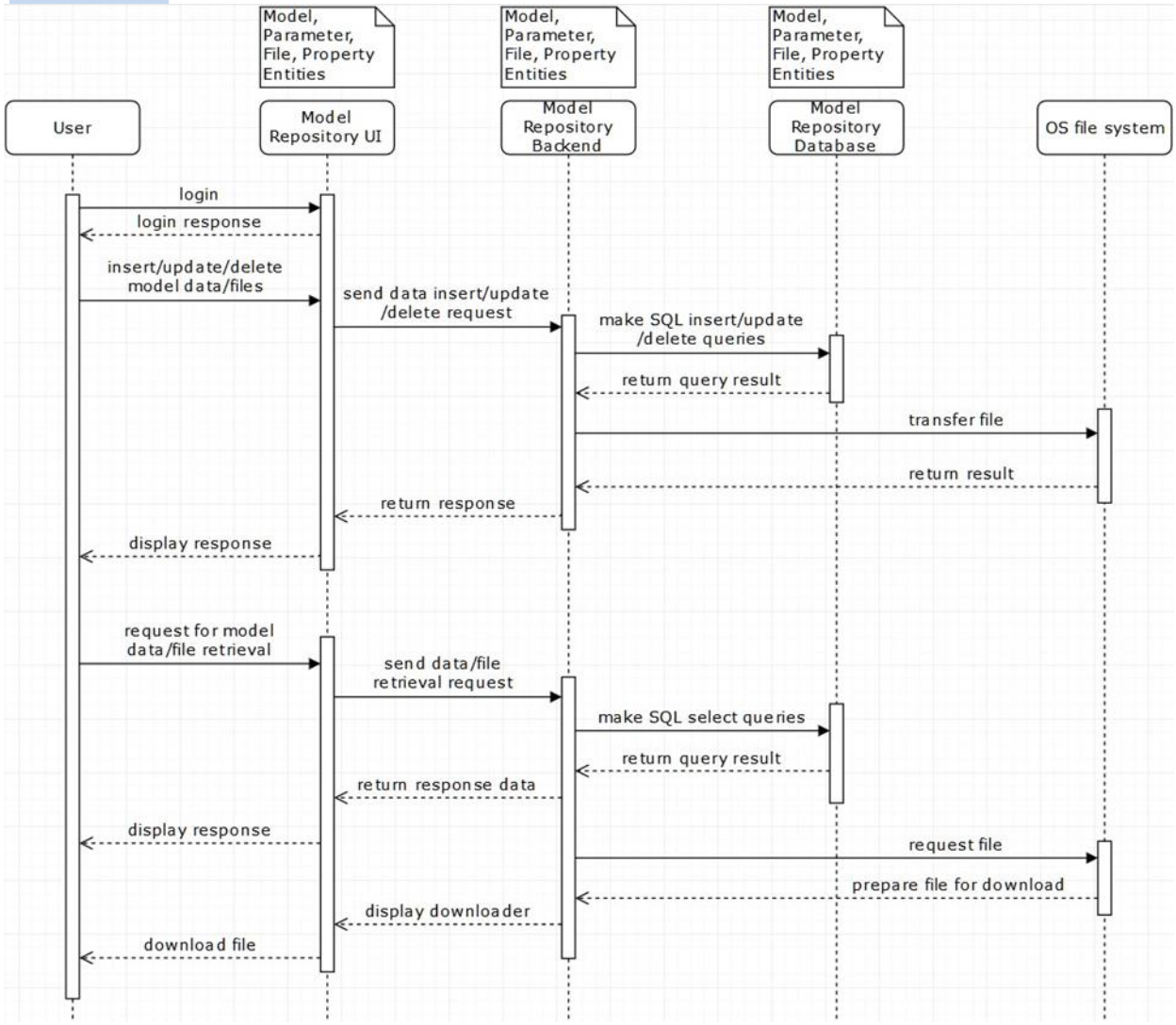

Figure 21: Model's repository sequence diagram

4.8.3. Deployment diagram

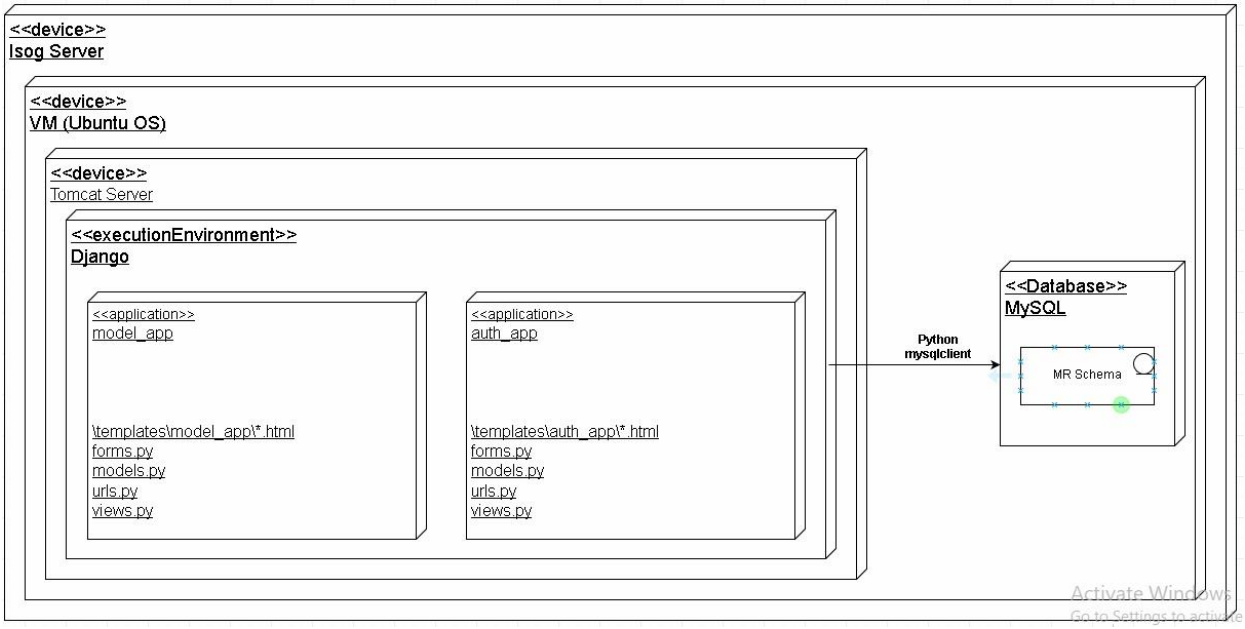

Figure 22: Model's repository deployment diagram

4.9. In Silico Prediction Repository

4.9.1. Component and interfaces

The BOUNCE *In Silico* Prediction Repository (ISPR) will be a web-based application, capable of persistently storing the predictions of the models developed within the BOUNCE project. Since the *in silico* predictions may require many computational resources, especially when the simulations involve multiscale data, the development of an ISPR in the context of the BOUNCE project is of utmost importance. The input data of each simulation (biological status, medical information sets, clinical information sets, contextual and psychosocial information sets, etc.), the model used in the simulation, and the output data will be stored persistently after the completion of the simulation scenario. Information related to the input (biological markers, medical imaging, lifestyle, psychological status, etc.) and the output (predicted psychological status, biological status or level of resilience of women with breast cancer) of all the simulations conducted using the *overall prediction model* and the *resilience trajectory prediction model* will be readily available through the ISPR for evaluation, comparison and validation. Consequently, since all predictions will be stored in the ISPR, there will be no need for executing the same simulation twice.

The key entities of the ISPR are the model prediction log, the experiment, and the subject. The model prediction log refers to multiple experiments. The experiments are performed with the same model each time, which is defined in the prediction log.

The basic principles of the ISPR are:

- The subject entity actually represents an instance of a subject e.g. a patient. Every instance of a subject can be accompanied by a set of files. The data of the files are internally stored in a file based repository.
- The *in silico* experiment entity consists of triples of input data – model information – output data. The model that is used in an in *silico* experiment is not stated in the

experiment entity, but in *the* model prediction log entity in which the experiment belongs.

- The *in silico* experiments are grouped per model in the model prediction log entity. All *in silico* experiments that are part of the same model prediction log entity use the same model.

Name	*In Silico* Prediction Repository		
Related use cases	User scenarios 1-6		
Due Date	*prototype*: M21, *first version*: M24, *second version*: M32 *first implementation*: M38		
Location source code	N/A		
Responsibilities/ Functionality	This is the web-based component that will permanently host the predictions of the models developed within the BOUNCE project.		
Provided Interfaces [Interfaces implemented by the component]	**Interface**	**Type**	**Description**
	Initial Interface	GUI	Greet user and prompt them to go to login screen to enter their credentials
	Login Interface	GUI	User is required to login to the system using their credentials
	Incorrect Login Interface	GUI	User is notified that their login attempt was unsuccessful and prompted to retry
	Main Interface	GUI	User is presented with the necessary choices for CRUD procedures on the prediction log / subject / experiment components
	Creation Interface	GUI	For each prediction log / subject / experiment component (models, parameters, etc.) and/or administrative data (users, roles, etc.) the user is required to complete a list of fields in a corresponding screen and submit the form.
	Content List Interface	GUI	User is presented with a set of screens containing the ISPR contents. By clicking on specific buttons in each row the user can edit (and update) or delete the row.
	Edit Interface	GUI	For each prediction log / subject / experiment component (models, parameters, etc.) and/or administrative data (users, roles, etc.) the user is required to correct a list of fields in a corresponding screen and submit the form.
	Authentication API	REST	An API for ensuring user authorization to use ISPR.
	DB API	REST	API for data management of ISPR content (data CRUD functions, file download)
Required	**Interface**	**Type**	**Description**

Interfaces			
Implementation	Python, MySQL, Javascript, HTML, CSS		
Third party frameworks	Django, jQuery		

Just like the BOUNCE MR, a user-friendly web interface and appropriate web services is being developed for the ISPR in order to expose its content to the users or other software components. Furthermore, pertinent authorization and authentication mechanisms will deny any unauthorized access. Information related to the predictions of all the models developed within the BOUNCE project will be stored.

Component API	*In Silico* Prediction Repository API		
Version	v0.1	**Date**	12/10/18
Notes	Under development		

Method	storePredictionLog		
POST	**Description**	Stores the basic descriptive information of the model prediction log, and returns the id of the model prediction log	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*model_id*: id of the model used in the prediction log (required) *description*: Description of the prediction log (required) *model_url*: The url where the model is located (required) *comment*: Comments on the prediction log (not required)
		JSON Object	The JSON object required in the request body must have all the above parameters (at least the required ones)
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has one key i.e. *id*, and one value which is associated with this key.
		Example	
Method	getAllPredictionLogs		
GET	**Description**	Returns the descriptive information of all the prediction logs stored in ISPR (prediction log ids, description of the prediction log, comments, etc.).	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	No parameter required

		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The keys of the JSON object returned are as many as the different prediction logs stored in the ISPR. Each value associated with a specific key is represented by a nested JSON object.
		Example	
Method	getPredictionLogById		
GET	**Description**	Returns the descriptive information (description of the prediction log, comments, etc.), of the given prediction log.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: Prediction log id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has nine keys i.e. id, description, model_id, model_url, comment, created_on, created_by, modified_on and modified_by, and nine values associated with those keys.
		Example	
Method	getPredictionLogByModelId		
GET	**Description**	Returns the information related to the prediction log in which the given model is used (prediction log id, description of the prediction log, comments, etc.). The argument is the id of the model used in the MR.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: Id of the model corresponding to the Prediction log (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has nine keys i.e. id, description, model_id, model_url, comment, created_on, created_by, modified_on and

			modified_by, and nine values associated with those keys.
		Example	
Method **DELETE**	deletePredictionLogById		
	Description	Deletes the prediction log, the experiments included in the prediction log and the reference links	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: Prediction log id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	
		Example	
Method **POST**	storeExperiment		
	Description	Stores the necessary and descriptive information of an experiment. It returns the id of the stored experiment.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*PredictionLog_id*: Id of the prediction log with which the new experiment is associated (required) *description*: the description of the new experiment (required) *subject_id_in*: The id of the subject that is used as an input to the new *in silico* experiment (required) *subject_id_out*: The id of the subject that is used as an output to the new *in silico* experiment (required) *status*: the status of the *in silico* experiment (NOT STARTED, ON PROGRESS, FINISHED SUCCESSFULLY, FINISHED ERRONEOUSLY) (not required)
		JSON Object	The JSON object required in the request body must have all the above parameters (at least the required ones)
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has one key named id, and one value which is associated with this key.

		Example	
Method	getAllExperimentsByPredictionLogId		
GET	**Description**	Returns information of all the experiments which belong to a given prediction log.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*PredictionLog_id*: Id of the prediction log that the new experiment is associated with (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The keys of the JSON object returned are as many as the different experiments that belong to the given prediction log. Each value associated with a specific key is represented by a nested JSON object.
		Example	
Method	getExperimentById		
GET	**Description**	Returns the experiment and the related information stored under the id (description, subject_id_in, subject_id_out, status, comment, etc.)	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id*: Experiment id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object has eleven keys named id, PredictionLog_id, description, subject_id_in, subject_id_out, status, comment, created_on, created_by, modified_on and modified_by, and eleven values associated with those keys.
		Example	
Method	getExperimentStatusById		
GET	**Description**	Returns the status of the experiment.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id: Experiment id* (required)
		JSON Object	N/A

		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has one key named status, and one value is associated with this key.
		Example	
Method	getExperimentsByStatus		
	Description	Returns all the experiments that are on a given status	
GET	**Request**	**Authentication**	
		Content Type	JSON
		Query Parameters	*status:* the status of the *in silico* experiment (NOT STARTED, ON PROGRESS, FINISHED SUCCESSFULLY, FINISHED ERRONEOUSLY) (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The keys of the JSON object returned are as many as the different experiments that are on a given status. Each value associated with a specific key is represented by a nested JSON object.
		Example	
Method	updateExperimentStatus		
	Description	Updates the status of a given experiment.	
PUT	**Request**	**Authentication**	
		Content Type	JSON
		Query Parameters	*id:* Experiment id (required) *status:* the status of the *in silico* experiment (NOT STARTED, ON PROGRESS, FINISHED SUCCESSFULLY, FINISHED ERRONEOUSLY) (required)
		JSON Object	The JSON object required in the request body must have the *id* and *status* parameters.
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	
		Example	
Method	deleteExperimentById		

DELETE	**Description**	Deletes the experiment.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id: Experiment id* (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has one key named status, and one value is associated with this key.
		Example	
Method	storeSubject		
POST	**Description**	Stores information related to a subject. Returns the id of the created subject	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*description:* Description of the state of the subject (required) *subject_external_id:* External id of the subject (not required) *comment*: Comments on the subject (not required)
		JSON Object	The JSON object required in the request body must have all the above parameters (at least the required ones)
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has one key named id, and one value is associated with this key.
		Example	
Method	deleteSubjectById		
DELETE	**Description**	Deletes a subject.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id:* Subject id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON

		JSON Object	
		Example	
Method	getAllSubjects		
	Description	Returns all the subjects.	
GET	**Request**	**Authentication**	
		Content Type	JSON
		Query Parameters	No parameters required
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The keys of the JSON object returned are as many as the different subjects that are stored in the *ISPR*. Each value associated with a specific key is represented by a nested JSON object.
		Example	
Method	getSubjectById		
	Description	Returns the subject and the related information stored under the id.	
GET	**Request**	**Authentication**	
		Content Type	JSON
		Query Parameters	*id:* Subject id (required)
		JSON Object	The JSON object required in the request body must have the *id* parameter.
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned by method getSubjectById has eight keys named id, description, subject_external_id, comment, created_on, created_by, modified_on and modified_by, and eight values associated with those keys.
		Example	
Method	storeFile		
	Description	Stores the file information and returns the id.	
POST	**Request**	**Authentication**	
		Content Type	JSON
		Query Parameters	*subject_id: Id of the subject with which the file is associated* (required) *title*: Title of the file (required) *description*: Description of the file (not required)

			kind: Defines what the file is (document, spreadsheet, csv, etc.) (not required) *Sha1sum*: Sha1 checksum of the file (not required) *Comment*: Comments on the file (not required)
		JSON Object	The JSON object required in the request body must have all the above parameters (at least the required ones)
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	The JSON object returned has one key i.e. id and one value associated with this key.
		Example	
Method **DELETE**	DeleteFile		
	Description	Deletes a certain file.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id:* File id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	JSON
		JSON Object	
		Example	
Method **GET**	getFileById		
	Description	Returns the requested file.	
	Request	**Authentication**	
		Content Type	JSON
		Query Parameters	*id:* File id (required)
		JSON Object	N/A
		Example	
	Response	**Status Codes**	200: OK, 400: BAD REQUEST, 404: DOESN'T EXIST, 500: SERVER ERROR
		Content Type	File
		JSON Object	
		Example	

Requirements that selected software should meet:

- Be free and open source.
- Have an active community that supports it by building plug-ins and extensions.
- Follow the Model-View-Controller (MVC) paradigm.

4.9.2. Sequence diagram

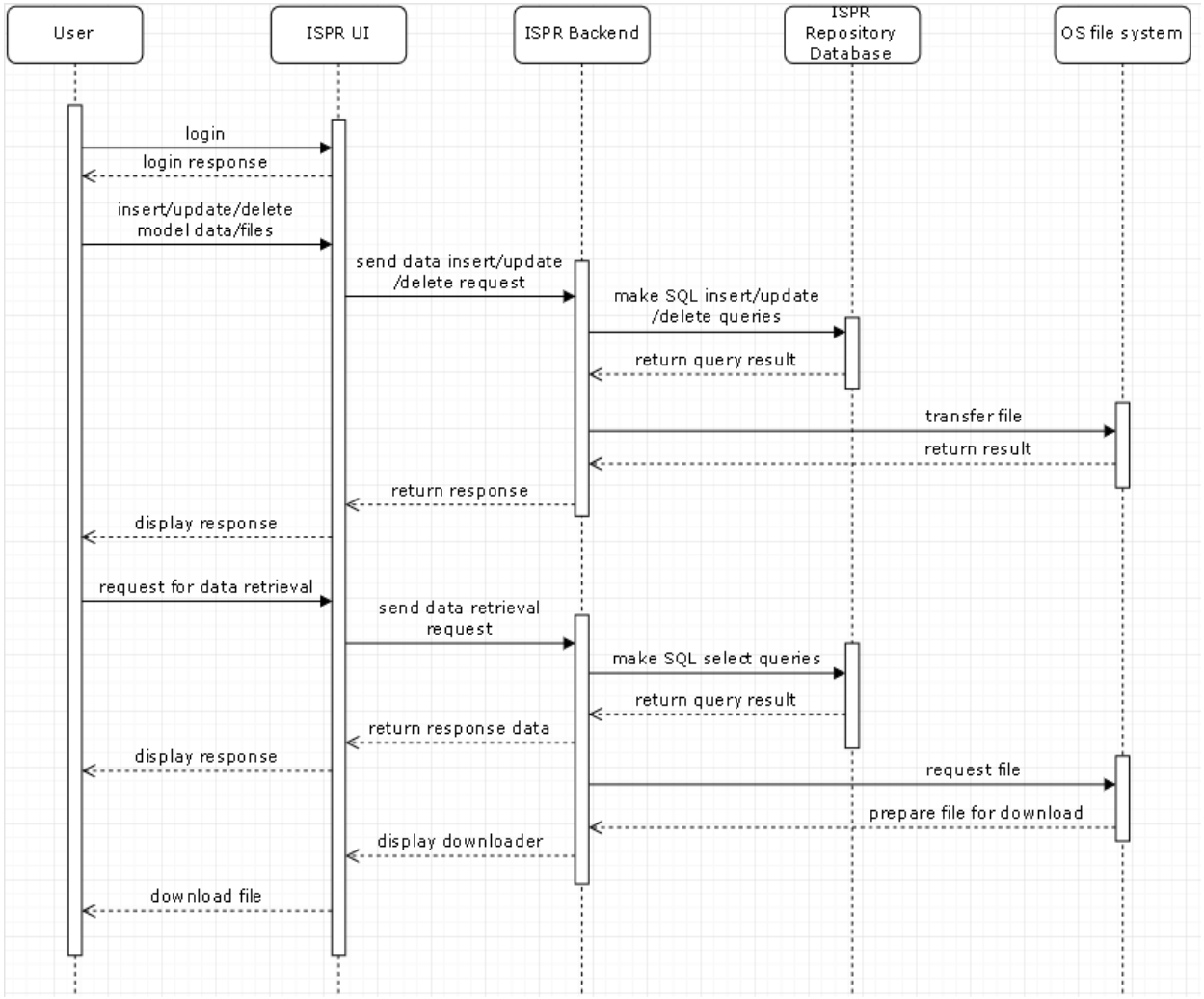

Figure 23: In Silico Prediction repository sequence diagram

4.9.3. Deployment diagram

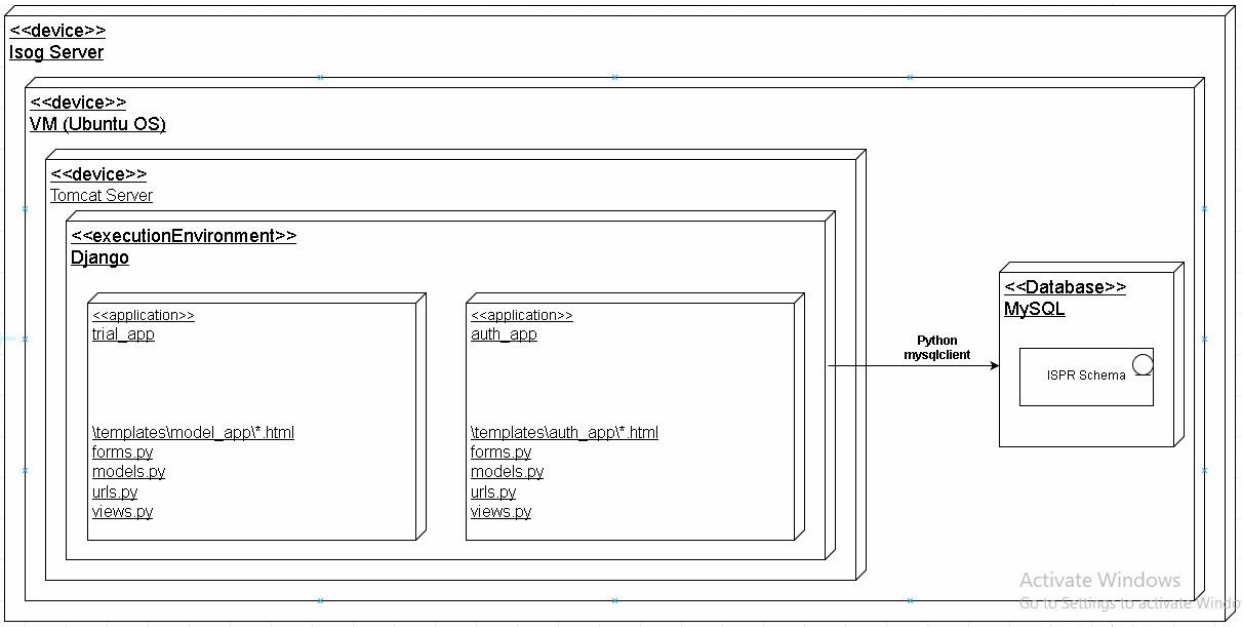

Figure 24: In Silico Prediction Repository deployment diagram

4.10. Execution Engine

4.10.1. Component and interfaces

Name	Execution Engine		
Related use cases	User scenarios 1-6		
Due Date	First version M24, Final version M40		
Location source code	Not available yet		
Responsibilities/ Functionality	The execution engine is responsible for the execution of a specific model over a selected set of data. Models to be supported must be written as scripts in python3 or R version 3.5 (supporting the Bioconductor library) or executables from C, C++ (gcc version 7.3.0) or Java v8 for Ubutnu 18.04 64 bit. It will provide an API (through the models API) to request an execution and monitor the progress of an execution. The execution engine will interact with the Data API to retrieve data, with the Models API to retrieve a model and with the In Silico prediction repository to store the results of an execution.		
Provided Interfaces [Interfaces implemented by the component]	**Interface**	**Type**	**Description**
	Executing Models	GUI	A programmatic interface used by the decision support tool for execution of model(s).
	Visualization of Results	GUI	A user interface will provide an overview of the analysis' results.
Required	**Interface**	**Type**	**Description**

Interfaces [Interfaces used by the component]	Data repository API	REST	API for the retrieval of the data
	Models repository API	REST	API to communicate with the models repository (retrieve model)
	In Silico Prediction Repository API	REST	API to store results
Implementation	Java and bash commands		
Third party frameworks			

4.10.2. Sequence diagram

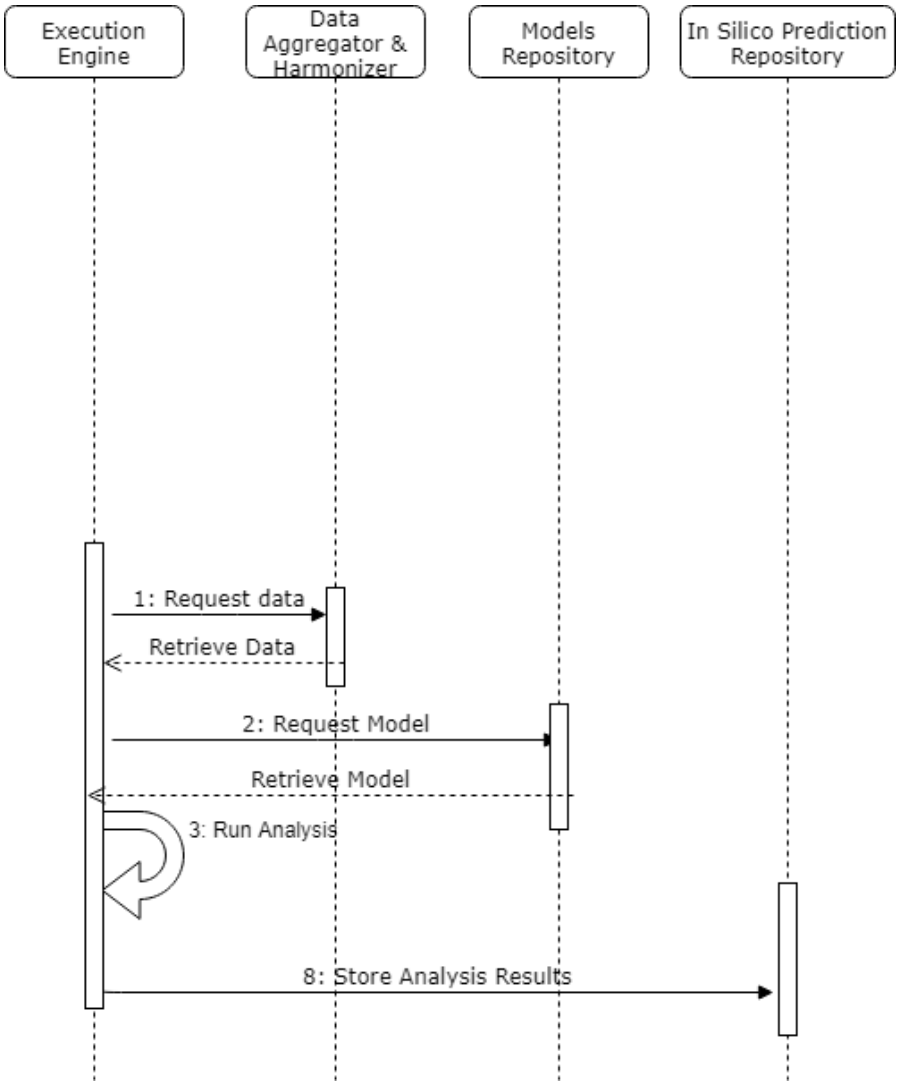

Figure 25: Execution Engine sequence diagram

4.10.3. Deployment diagram

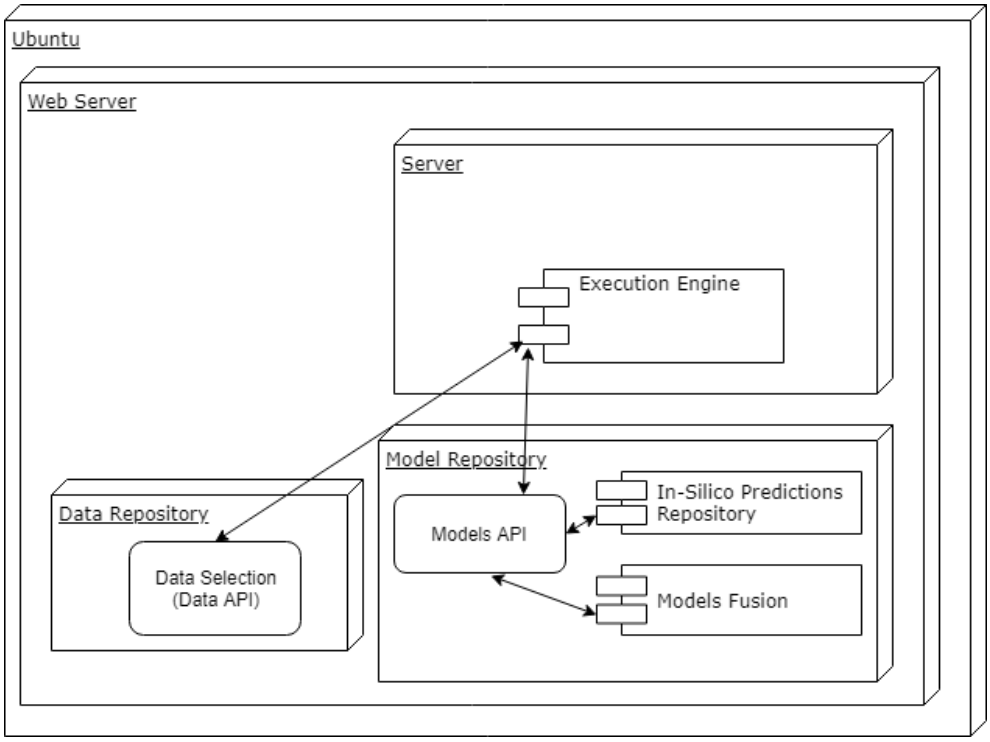

Figure 26: Execution Engine deployment diagram

4.11. Decision Support System

4.11.1. Component and interfaces

Name	Decision Support System		
Related use cases	All		
Due Date	First version M24, Final version M40		
Location source code	Not available yet		
Responsibilities/ Functionality	Decision Support System is the final online tool that produces (a) an overall "resilience predictor" score, and (b) scores for specific psychological variables that are important for resilience and adaptation to cancer. The tool will replace the Temporary Research Tool and will be able to retrieve data about the individual scores on each scale and the combination of scores in different biomedical and psychosocial variables (coming from the current and/or possible previous assessments). The tool should also be able to produce (a) an overall "resilience predictor" score, and (b) scores for specific psychological variables that are important for resilience and adaptation to cancer.		
Provided Interfaces	**Interface**	**Type**	**Description**
	Initial Interface	GUI	Greet user and prompt them to go to login screen to enter their credentials

[Interfaces implemented by the component]	Login Interface	GUI	User is required to login to the system using their credentials
	Entering Data	GUI	A user interface used by the expert for entering data.
	Executing Models	GUI	A user interface used by the expert for selection and execution of model(s).
	Visualization of Results	GUI	A user interface will provide an overview of the analysis' results.
	Store Results	GUI	A user interface will provide the option to the user to store the results (at the In Silico Prediction Repository)
Required Interfaces **[**Interfaces used by the component]	**Interface**	**Type**	**Description**
	Authentication API	REST	An API for ensuring user authorization to use the system
	Data repository API	REST	API for data management (retrieve data)
	Models repository API	REST	API to communicate with the models repository (retrieve model)
	Execution Engine API	REST	API to request an execution of the selected model over the selected data. The API will also provide information about the status of the execution
	In Silico Prediction Repository API	REST	API to store results
Implementation	JavaScript, Java Web Application		
Third party frameworks	• Spring Framework (Apache 2.0 license). • Hibernate (GNU Lesser General Public License).		

As the implementation of this component matures, more details will be described in this deliverable about the exact API calls and the services provided.

4.11.2. Sequence diagram

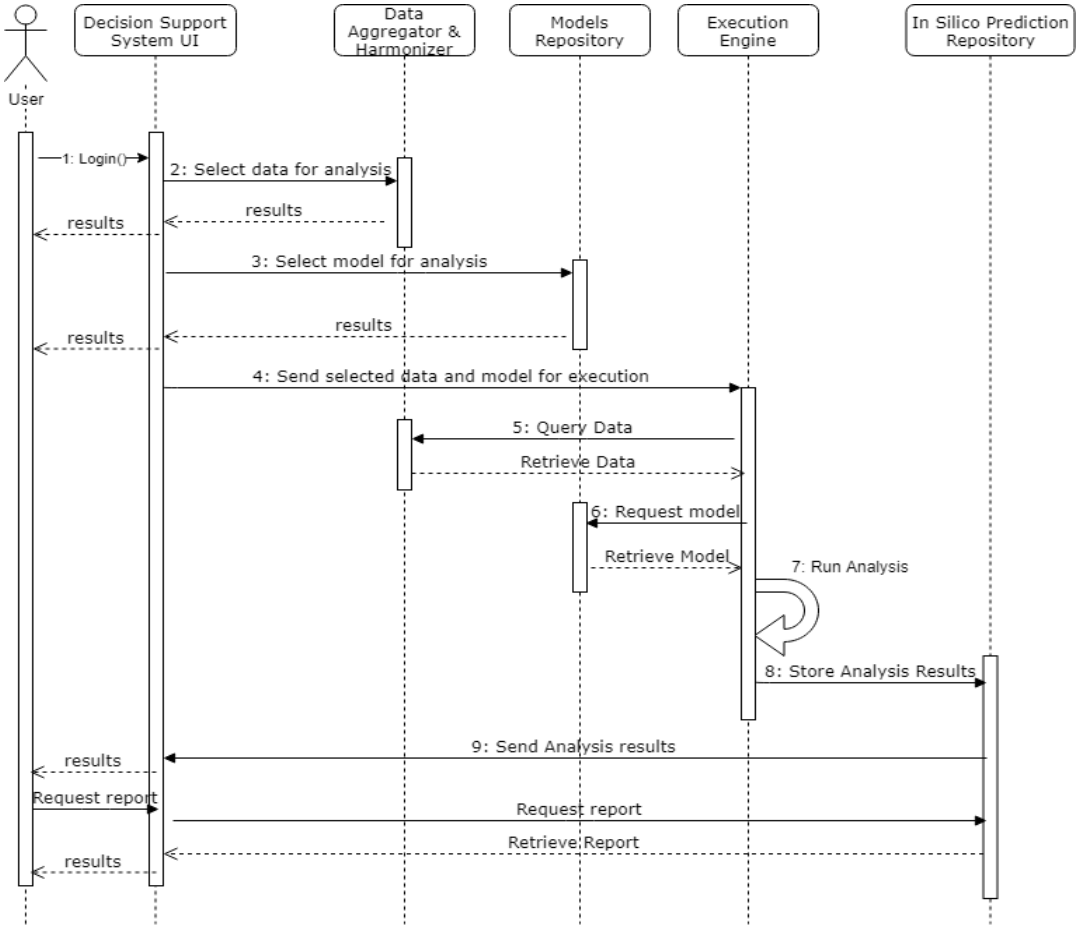

Figure 27: Decision Support System sequence diagram

4.11.3. Deployment diagram

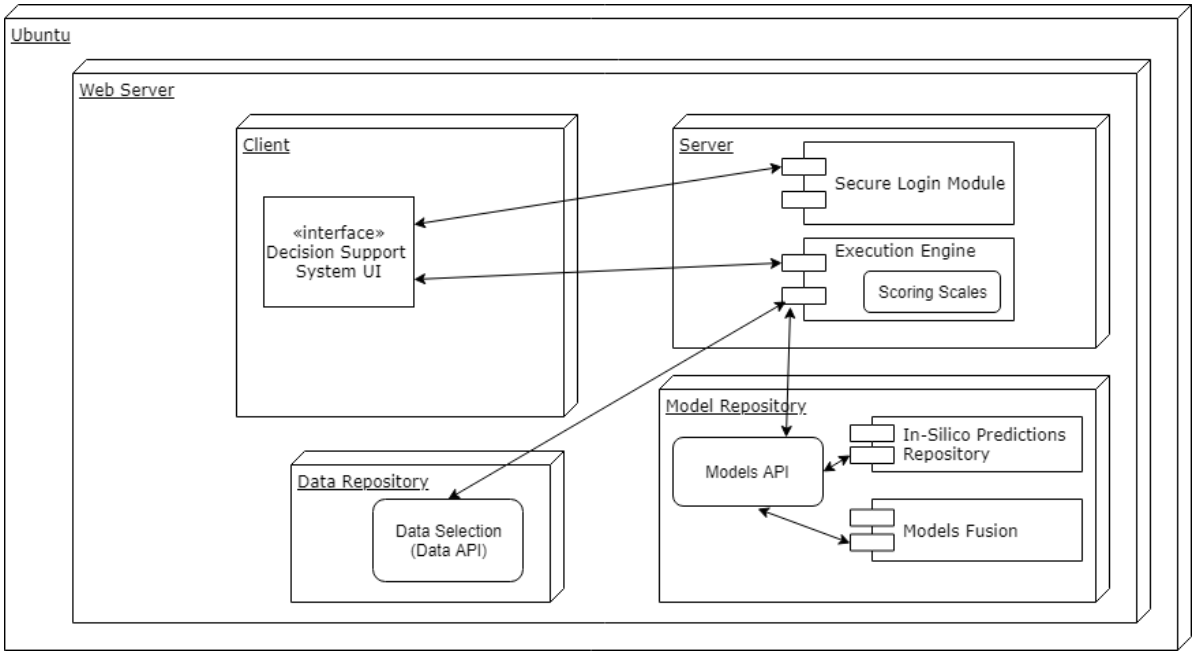

Figure 28: Decision Support System deployment diagram

5. Initial BOUNCE reference Architecture

The main challenge of the BOUNCE architecture is the interoperability of systems, tools and services that are made available to the users of the environment with the ultimate goal of secure, transparent, and unobtrusive sharing of data and functionality. Most of the currently identified scenarios in the project are focused on data access and processing of data but there are also tasks involving computational jobs and visualization. In order to fulfil the requirements imposed by these scenarios a scalable and flexible environment is needed and the following technologies, which have gained momentum in the recent years, will be adopted: (i) Web/REST Services technologies and (ii) Semantic Web technologies.

The BOUNCE platform is designed using a multi-tier architecture (security tier, semantics tier and applications tier). Every component/service designed within BOUNCE can be mapped to one of these layers (or spanned over multiple layers). Modules and components designed and built within the project should seamlessly operate through well-defined interfaces on different levels (i.e. interoperability on the level of IT-protocol, data format, information content, etc.).

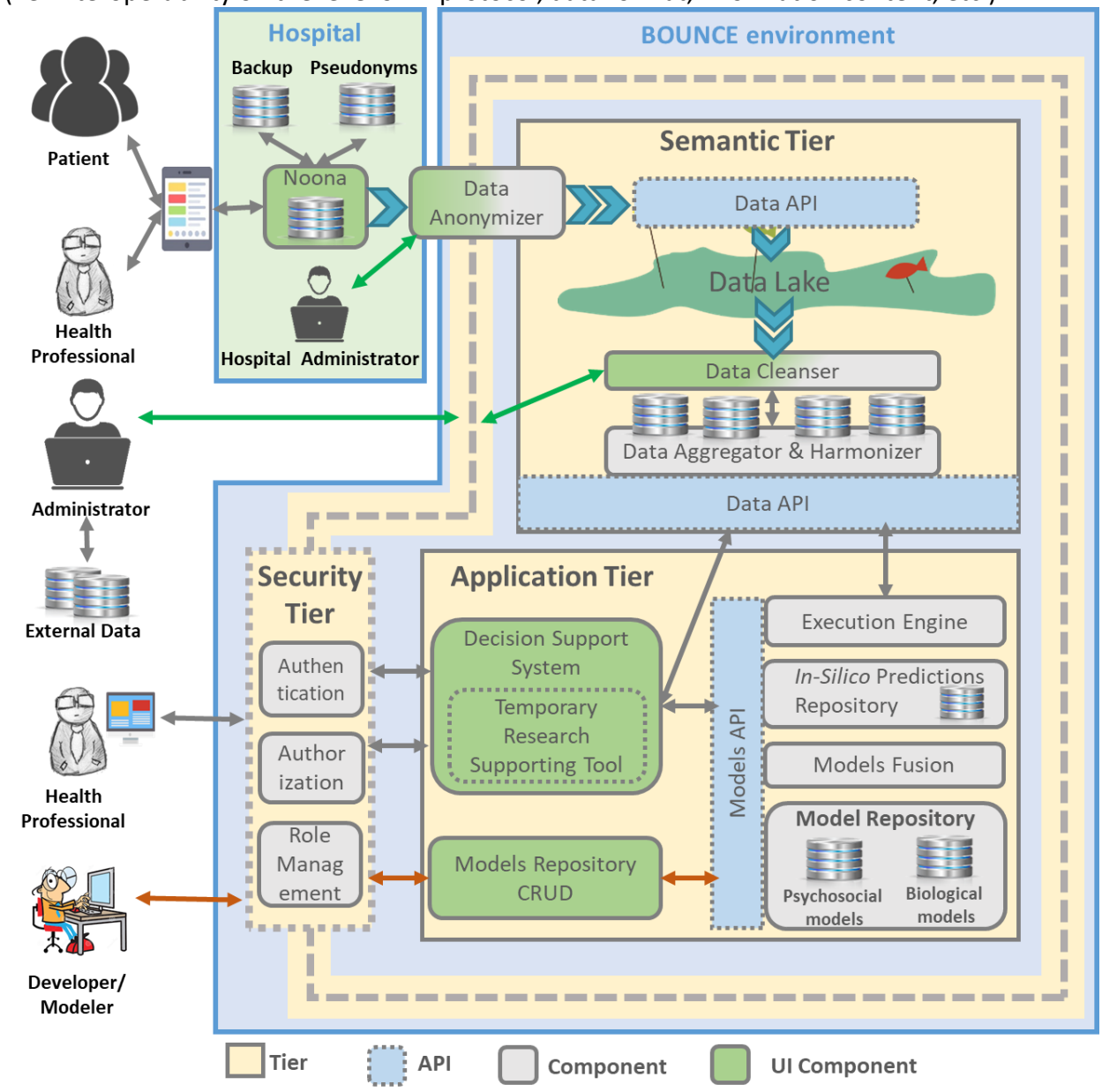

Figure 29: Initial BOUNCE Architecture

Figure 29 shows the main components of BOUNCE architecture and their interconnections. As shown in figure, patient's data will be collected via the Noona tool and stored within the hospital premises. Healthcare professionals, such as physicians, research nurses and psychologists, will be able to access the Noona tool and provide clinical assessment data for the patients. The data anonymizer module anonymizes the collected data and pushes it into the BOUNCE data lake (via the Data API). Then the semantic tier modules clean and harmonize the data that is accessible with the Data API. Health professionals have also access to the decision support tool and in the first stages of the development to the temporary research-supporting tool. Temporary Research Supporting Tool is the short-term internal project tool to facilitate data exploration and visualization of data and scales. Decision Support System is the final online tool that produces (a) an overall "resilience predictor" score, and (b) scores for specific psychological variables that are important for resilience and adaptation to cancer. The Decision Support System will replace the Temporary Research Tool and will be able to retrieve data, apply prediction model(s) and view/store the results of the analysis. Furthermore, model developers can upload models to the models repository.

6. Conclusions

This document aims to provide recommendations and guidelines that will ensure the integration and interoperability of software components within the BOUNCE technological environment. Software integration is a laborious and challenging task and therefore analysis of the user requirements is necessary in order to identify the objectives of the system under development and how these can be accomplished. In this sense, this document has been prepared based on the input given by the "Description of Annex" document, the Deliverable D1.3. (BOUNCE methodology) and deliverable D1.2 (Requirements & Usage Scenarios).

BOUNCE aims to provide a framework of tools, which can be easily interconnected in different configurations, tailored to the needs of different environments (e.g. hospitals) and end-users. To obtain high flexibility, loose coupling and service-orientated approaches are chosen. On the implementation level, it was decided that BOUNCE would rely on REST APIs as communication protocol. Focus is on interoperability and interfacing in the architectural description and for that reason modules and components designed and built within the project should seamlessly operate through well-specified interfaces on different levels (i.e. interoperability on the level of IT-protocol, data format, information content, etc.).

Deliverable 5.1 reports in detail the BOUNCE components using uniform templates for description (Appendix A – Template for component description), sequence interfaces and deployment interfaces (including UML component diagrams). Furthermore, the mature components and the under development components reported the detailed rest interfaces (Appendix B – Template for REST service specification). Since the implementation of some components (e.g. decision support system) has not started yet, we cannot currently provide details about their REST APIs. Nevertheless, the Appendix A – Template for component description will be used during the development of all the components ensuring a smooth final integration.

This document is a snapshot of the evolving BOUNCE architecture as the project progresses. It should be noted that BOUNCE takes an iterative approach towards the design of the architecture, so this first iteration does not by any means offers the final solution. We consider the deliverable D5.1 of BOUNCE a live document that will be updated during the software lifecycle.

Appendixes

Appendix A – Template for component description

Name			
Related use cases	[Use the identifiers of the use cases of D1.2]		
Due Date	[Due dates for prototype, first version etc.]		
Location source code	[Link to source code]		
Responsibilities/ Functionality	[Brief summary of the responsibilities and functionality]		
Provided Interfaces [Interfaces implemented by the component]	**Interface**	**Type**	**Description**
		[Type of interface e.g. GUI, Rest etc.]	[Brief description of interface, if the interface is a programmatic interface please provide a detailed specification in Appendix …]
Required Interfaces [Interfaces used by the component]	**Interface**	**Type**	**Description**
Implementation	[Which technologies will be used to implement the component?]		
Third party frameworks	[List any third party software frameworks / systems that you which to integrate into your component including the license under which it is provided]		

Appendix B – Template for REST service specification

Component API			
Version		**Date**	
Notes			
Method			
	Description		
	Request	**Authentication**	
		Content Type	
		Query Parameters	
		JSON Object	
		Example	
	Response	**Status Codes**	
		Content Type	
		JSON Object	
		Example	
Method			
	Description		
	Request	**Authentication**	
		Content Type	
		Query Parameters	
		JSON Object	
		Example	
	Response	**Status Codes**	
		Content Type	
		JSON Object	
		Example	
Method			
	Description		
	Request	**Authentication**	
		Content Type	
		Query Parameters	
		JSON Object	
		Example	
	Response	**Status Codes**	
		Content Type	
		JSON Object	
		Example	