**eWALL platform**



**This document presents an overview of eWALL that can be used by the eWALL Open Source Community. The document belongs to eWALL consortium and any use of the document requires the reference to eWALL Open Source (http://gitghub.com/ewallprojecteu)**

Table of Contents

[1 eWALL Platform Development Process and Environment 4](#_Toc510695440)

[1.1 Process and tools overview 4](#_Toc510695441)

[1.2 Basic aspects 6](#_Toc510695442)

[1.2.1 Programming language(s) 6](#_Toc510695443)

[1.2.2 Technology framework 7](#_Toc510695444)

[1.2.3 Documentation 7](#_Toc510695445)

[1.3 Development Environment 8](#_Toc510695446)

[1.3.1 Integrated development environment (Eclipse and Spring Tool Suite) 8](#_Toc510695447)

[1.3.2 Source code version control tool (Git) 9](#_Toc510695448)

[1.3.3 Source code repository manager (GitLab) 10](#_Toc510695449)

[1.3.4 Issue tracker (GitLab Issue Tracker) 10](#_Toc510695450)

[1.3.5 Automation build tool (Maven) 13](#_Toc510695451)

[1.3.6 Maven repository manager (Nexus) 15](#_Toc510695452)

[1.3.7 Inversion of control container server (Spring IoC Container) 16](#_Toc510695453)

[1.3.8 Deployment server (Tomcat) 16](#_Toc510695454)

[1.3.9 Continuous integration server (Jenkins) 18](#_Toc510695455)

[1.3.10 Server monitoring tool (Zabbix) 18](#_Toc510695456)

[1.3.11 Front-end JavaScript Framework (AngularJS) 19](#_Toc510695457)

[2 eWALL Platform within overall eWALL system architecture 20](#_Toc510695458)

[2.1 Data flows between home and cloud platform components 23](#_Toc510695459)

[3 eWALL Platform Implementation 26](#_Toc510695460)

[3.1 Remote Proxy 26](#_Toc510695461)

[3.1.1 Black box description 26](#_Toc510695462)

[3.1.2 Requirements 26](#_Toc510695463)

[3.1.3 Features 27](#_Toc510695464)

[3.2 Local Data Manager 28](#_Toc510695465)

[3.2.1 Black box description 28](#_Toc510695466)

[3.2.2 Requirements 28](#_Toc510695467)

[3.2.3 Features 29](#_Toc510695468)

[3.3 eWALL Portal 30](#_Toc510695469)

[3.3.1 Black box description 30](#_Toc510695470)

[3.3.2 Requirements 31](#_Toc510695471)

[3.3.3 Features 31](#_Toc510695472)

[3.4 Notification Manager 33](#_Toc510695473)

[3.4.1 Black box description 33](#_Toc510695474)

[3.4.2 Requirements 33](#_Toc510695475)

[3.4.3 Features 34](#_Toc510695476)

[3.5 Profiling Server 35](#_Toc510695477)

[3.5.1 Black box description 35](#_Toc510695478)

[3.5.2 Requirements 35](#_Toc510695479)

[3.5.3 Features 36](#_Toc510695480)

[3.6 Login service 37](#_Toc510695481)

[3.6.1 Black box description 37](#_Toc510695482)

[3.6.2 Requirements 38](#_Toc510695483)

[3.6.3 Features 38](#_Toc510695484)

[3.7 Cross-component functionalities 39](#_Toc510695485)

[3.7.1 Black box description 40](#_Toc510695486)

[3.7.2 Features 40](#_Toc510695487)

[3.8 Artefacts developed within other WP4 tasks 41](#_Toc510695488)

[3.8.1 Data Manager (T4.3) 41](#_Toc510695489)

[3.8.2 Cloud Gateway (T4.3) 41](#_Toc510695490)

[3.8.3 Intelligent Decision Support System (T4.2) 42](#_Toc510695491)

[3.8.4 Commons data model (T4.1) 42](#_Toc510695492)

[4 Conclusions 44](#_Toc510695493)

[Bibliography 45](#_Toc510695494)

[Abbreviations 46](#_Toc510695495)

[Appendix A: eWALL Platform RESTful interfaces 47](#_Toc510695496)

[Interface between Sensing Environments and eWALL Cloud (I1) 47](#_Toc510695497)

[Provisioning interface (I2) 49](#_Toc510695498)

[Notification interface (I4b) 51](#_Toc510695499)

[Profiling interface (I4a) 53](#_Toc510695500)

[Appendix B: eWALL Portal User manual 59](#_Toc510695501)

# eWALL Platform Development Process and Environment

This chapter describes the eWALL Platform development process, including the identification of the related phases (design, development, building, testing, deployment, runtime, etc.) and basic aspects such as programming language, technology framework and documentation.

## Process and tools overview

The identified development process for eWALL Platform covers the following phases of development:

1. Design – creating textual and UML specification of planned implementation; handled by tasks on WP2; define use cases, scenarios, requirements and system architecture
2. Implementation – performing actual implementation; usual tasks on WP3, WP4 and WP5;
3. Building – compiling the code and packaging into a final artefact;
4. Testing – mainly unit and integration testing, but also continuous integration; unit testing considered as part of development work, while integration testing as part of WP6;
5. Deployment – exposing the artefact to a repository shared by the partners;
6. Runtime – starting eWALL system components and running all needed eWALL components;
7. Issue Reporting – notifying the developers about any bugs or features which should be covered.

Phases are listed in their usual order in which they are performed during the development of eWALL Platform. The Figure 1 illustrates the eWALL Platform development process and tools that are used to help and support work in certain phases. Details about each tool are described in Section 4.3 about development environment.



Figure 1: Overview of eWALL Platform development process and tools

Continuous delivery workflow shown in Figure 2, building on top of the Continuous Integration, was applied in eWALL in order to keep the source code and whole setup in highly-releasable state.

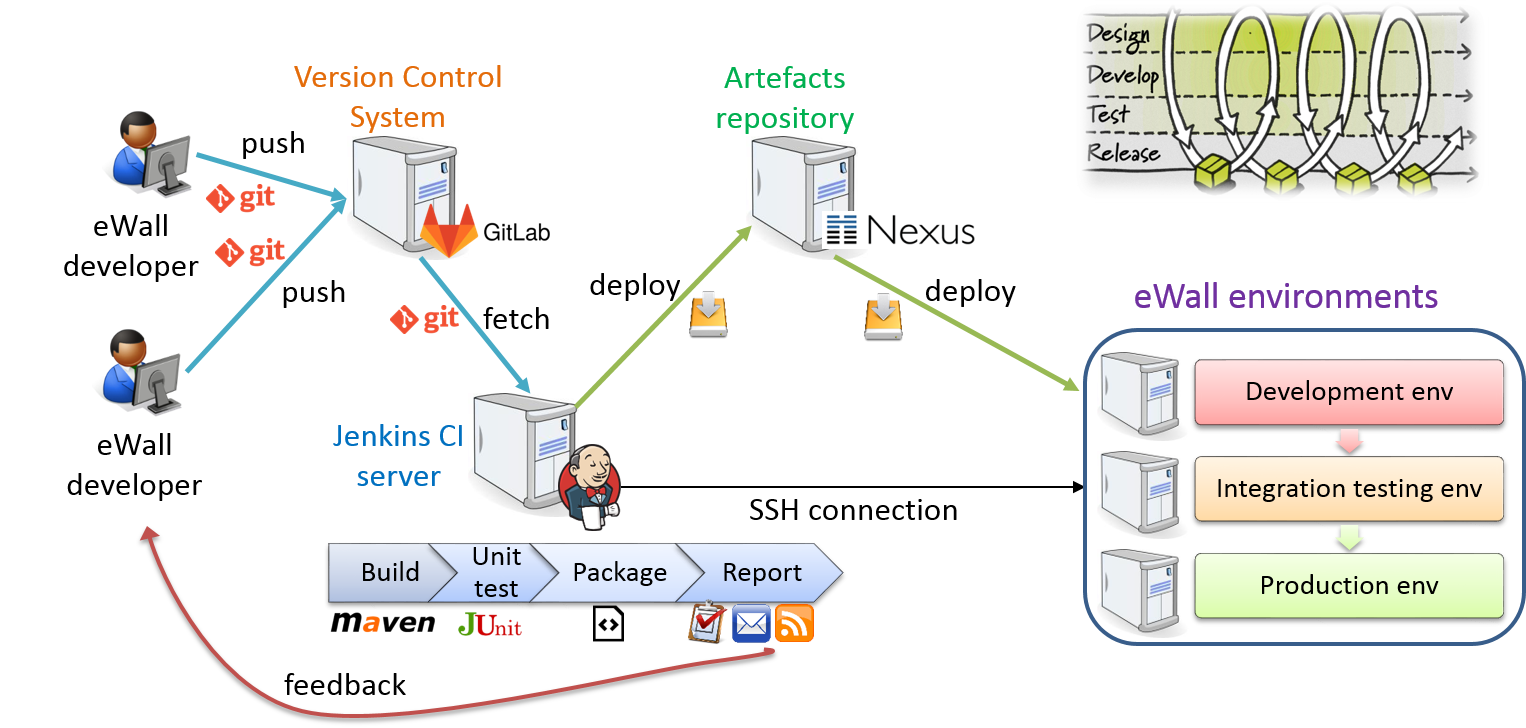


Figure 2 Continuous delivery workflow in eWALL

## Basic aspects

Besides the definition of the main development process, basic aspects related to fundamental choices such as the selected programming language, the technology platform and documentation is presented in this section.

### Programming language(s)

The main programming language for the development of the eWALL platform is Java. Java has many advantages over other languages and environments that make it suitable for eWALL Platform development. These are:

1. **Platform-independence:** ability to move easily from one computer system to another. Java is platform-independent at both the source and binary levels. Currently, Java programs can run on the Microsoft Windows, Mac OS X, Linux, and Solaris operating systems.
2. **Object-orientation:** allows creating modular programs and reusable code.
3. **Distribution**: Java makes distributed computing easy with the networking capability that is inherently integrated into it.
4. **Security**: Java language, compiler, interpreter, and runtime environment were each developed with security in mind.
5. **Robustness and reliability**: lot of emphasis on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages.
6. **Multithreaded:** the capability for a program to perform several tasks simultaneously within a program. In Java, multithreaded programming has been smoothly integrated into it, while in other languages, operating system-specific procedures have to be called in order to enable multithreading.

Because of Java's robustness, ease of use, cross-platform capabilities and security features, it has become a language universally used for complex, enterprise-level systems, thus also for the eWALL Platform.

### Technology framework

For the development of Java based enterprise systems there are two main frameworks that are commonly used: Java Enterprise Edition (JEE)[[1]](#footnote-2) and Spring Framework[[2]](#footnote-3). In this context JEE is referred to as reference implementation of JEE specification by Oracle since Spring Framework is actually built on top of many core JEE specification technologies.

The Spring Framework is an open source application framework and Inversion of Control (IoC) container for the Java platform focused on building modern enterprise applications. The framework decides, at a high-level, the flow of operations to perform a certain task and then the developer merely plugs-in the necessary specifics at each step. There are several advantages of using Spring Framework that were considered in selection for eWALL Platform:

1. Spring provides template to make using core JEE technologies easier (it does not replace them).
2. Spring does not require a heavy JEE application server, service component can be deployed in a web container such as Tomcat.
3. Spring offers more features which are not available as JEE standards.
4. Spring offers much more flexibility and power, e.g. aspect-oriented programming (AOP) is more powerful than JEE interceptors.
5. Because it does not go through long standardization process and there is only one vendor, the releases and the reaction to market requirements is much faster (for example features related to cloud, mobile, social computing or big data).

Considering the above advantages and based on the previous experience of the partners, it was decided on a project level that all eWALL Cloud components would be developed using Spring Framework.

### Documentation

#### Wiki-based documentation of software components

Concerning large projects it is important that the documentation is readily accessible for everyone involved in development process. For the eWALL Platform we followed a combined approach. A large amount of content, especially source code documentation and change logs, is easy to generate automatically using tools like Maven, Jenkins, Javadoc or similar tools. Furthermore, for installation instructions and general documentation about components we have used a wiki system supported (and hosted) by GitLab (explained in section 4.3.3).

#### Static (Javadoc-based) documentation of source code

Javadoc[[3]](#footnote-4) is a tool for software documentation from Oracle Corporation that is able to automatically generate API documentation in HTML format from Java source code. The HTML format is used to add the convenience of being able to hyperlink related documents together. The “doc comments” format used by Javadoc is the de facto industry standard for documenting Java classes. Some IDEs, such as Netbeans and Eclipse, automatically generate Javadoc HTML. Many file editors assist the user in producing Javadoc source and use the Javadoc info as internal references for the programmer.

#### Interactive (Swagger-based) documentation of API profiles

Swagger[[4]](#footnote-5) is a specification and complete web application framework implementation for describing, producing, consuming, and visualizing RESTful web APIs (collections of web resources). It enables interactive documentation of the RESTful APIs where developers can try out different API calls and get responses without even implementing a single line of code. Javadoc on the other hand offers only passive (read only) documentation and exposes the document provided by the developers of the software components.

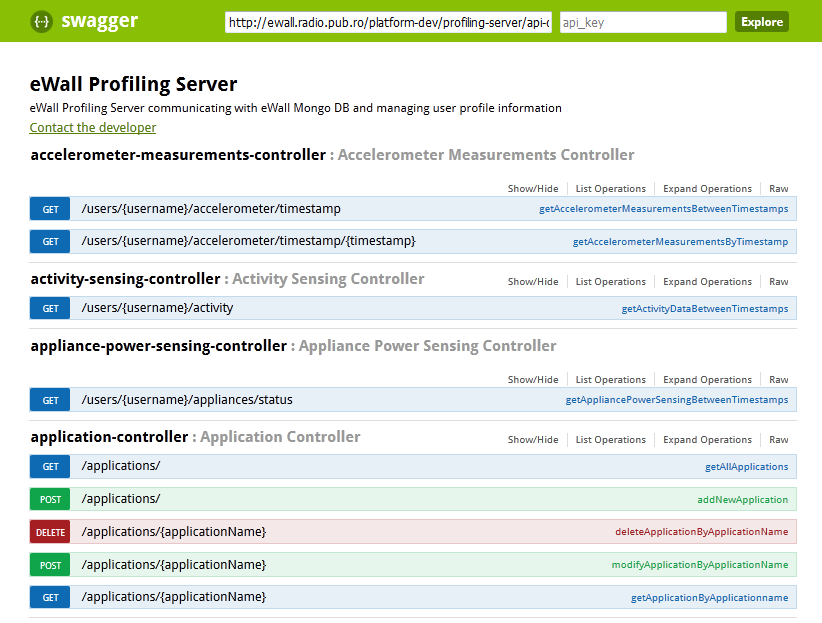


Figure 3 Screenshot of Swagger documentation for eWall Profiling Server component

## Development Environment

### Integrated development environment (Eclipse and Spring Tool Suite)

Eclipse is one of the most popular software development environments for Java. It is an open source project that provides an Integrated Development Environment (IDE) together with a large amount of components adding certain functionality, called plug-ins.

The Spring Tool Suite (STS) [6] provides the Eclipse-powered development environment for building Spring-based enterprise applications. STS supplies tools for all of the latest enterprise Java and Spring, and comes on top of the latest Eclipse releases. It provides a graphical real-time view of application performance metrics that lets developers identify and diagnoses problems from their desktops. STS supports application targeting to local, virtual and cloud-based servers. It is freely available for development and internal business operations use.

Because of the usage of Maven on eWALL Platform development, no specific IDE is mandated on project level. However, because of above features and usage of Spring, it is recommended to use STS for eWALL Cloud components development.

#### Setup and usage in eWall

At the beginning of the development STS3.6.3.SR1 was used but when new version was introduced some developers updated their environments (at the time of the writing of this document the version sts-3.7.3, based on Eclipse Mars 4.5.2 is the last one compatible with Java 7 which is used in eWall, STS 3.8.x needs Java 8). The updates to the newer version were neither obligatory nor needed to complete the implementation work but offered some polishing of the functionalities and non-critical updates.

### Source code version control tool (Git)

Software version control, also known as revision control and source control, is described as the management of changes to source code, documents, large web sites, and other collections of information.

Git [7] is free and open source distributed revision control and source code management tool that has complete history and revision tracking capabilities. Git has many advantages, such as resilience, very fast operations compared to other version control systems (such as CVS and Subversion), space (since compression can be done across repository not just per file which minimizes local size as well as push/pull data transfers), no single point of failure - avoids relying on single machine but permits centralized control of the "release version" of the project, allows offline work - most operations faster since no network is involved and as merging is essential to the workflow it is fast and tends to have far fewer conflicts than SVN.

#### Setup and usage in eWall

In eWALL, Git version 1.8.5.2 was used at the beginning although developers used new releases freely when made available (e.g. at the time of finalizing this deliverable some used v2.5.0). Following the overall organization of the eWALL project, the source code is organized by Work Packages. Each Work Package has a separate Git repository, organized into two branches:

* + - Master branch: this is the development branch, used to add new features during development cycles.
    - Stable branch: this branch is used only to fix bugs detected during the lifetime of a release.

Figure 4 shows the branch management strategy used in eWALL. At the end of each development cycle, when code in Master branch is considered to be mature enough, it is promoted and merged to Stable branch, while the version on Master is changed to the following release number. Then, a release is tagged and built from Stable branch. In this way, while on the Master branch a new development cycle is started (adding new features), the release on Stable branch is used for testing purposes. If bugs are found, their fixes are pushed directly to the Stable branch, then they are merged back to Master and a new bugfix release is built from stable.

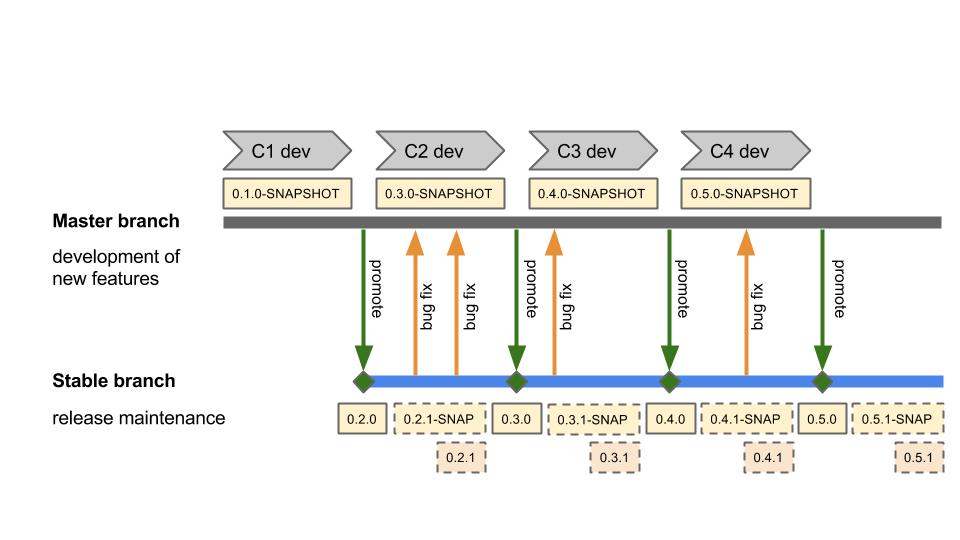


Figure 4: Branch management strategy in eWALL

### Source code repository manager (GitLab)

The software version control system is a repository of files with monitored access and where every change made to the source is tracked. Having such a version control system is essential to having a successful software project especially when developers are spread across countries and must share the code.

GitLab [8] is an open source Git management software, code collaboration platform, Git repository manager, issue tracker and code reviewer. It is used in the eWALL project as the main tool which allows to keep code secure on its own server, manage repositories, user and access permissions, communicate through issues, line-comments and wiki pages, and perform code review with merge requests.

#### Setup and usage in eWall

In eWall GitLab 6.9.0 was setup on a Unix server running at UPB premises. One administrator was assigned to create repositories and allow access to interested people (if people were developers then they were assigned with Developer role, if only observer then they were assigned with Reporter role). The leaders of the implementation in different parts were given Master access according to the GitLab access levels[[5]](#footnote-6).

### Issue tracker (GitLab Issue Tracker)

Issue tracker was based on the GitLab (described in the section 4.3.3). Issue tracker is a tool that developers use to maintain a list of software defects (bugs), new feature requests, and other issues that the project members must work to resolve.

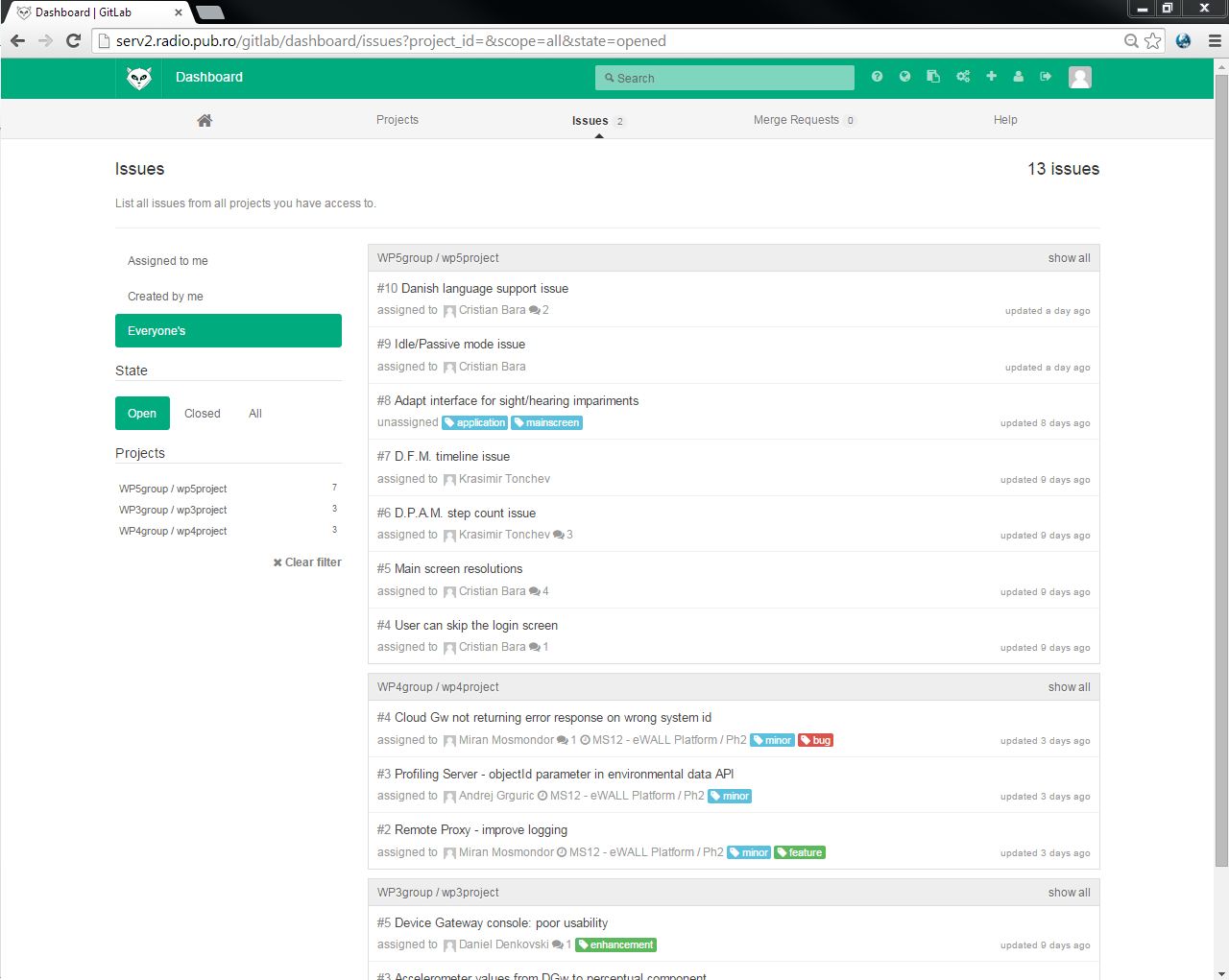


Figure 5 Screenshot of the eWALL Issue Tracker

#### Setup and usage in eWall

For setup info see section 4.3.1.1.

Usage in eWall (on all technical work packages) can be described in following steps:

1. step: submitting an issue (by anyone):

- Login to [GitLab](http://serv2.radio.pub.ro/gitlab/), select project that it related to (wp3project, wp4project, or wp5project), then in top menu select “Issues”, and then “+ New issue”. Direct links:

o for WP3: <http://serv2.radio.pub.ro/gitlab/wp3group/wp3project/issues>

o for WP4: <http://serv2.radio.pub.ro/gitlab/wp4/wp4project/issues>

o for WP5: <http://serv2.radio.pub.ro/gitlab/wp5group/wp5project/issues>

- Add title and description (title should contain name of the component that issue is related to (if this is known), description should preferably contain more details (exact steps how to reproduce the issue, a log trace, error message, etc.))

- Optionally you can select a milestone this issue is related to and add labels to easily classify the issue (see below for more details)

- For any question contact TM or WP leaders

2. step: WP leader assigns a person as responsible for resolving an issue

- If WP leader accepts the issue, he needs to set ‘Assign to’ field and inform assignee that it is responsible for solving it (the assignee may later discover that the issue is causing some other component. In that case assignee can request from WP leader to reassign the issue (or close the issue if it is discovered that it is related to another WP).

- If WP leader does not accept the issue (if he thinks issue is not related to his/hers WP) the leader can close the issue and suggest to submitter to open a new issue at another WP. Also, comments can be added stating the reason.

- In case of a conflicts, i.e. WP leaders and submitter cannot decide to who issue belongs to, the TM will decide.

3. step: When issue is resolved, the person who opened the issue must close it.

- Usually, when issue is resolved, the person who opened it must be notified of the fix, test it again and mark it as closed.

Regarding the labels: when you submit and issue you can add labels to easily classify the issue. Do not use some custom or very specific labels. Instead, it is recommended that you use one of the following:

For reporting software defects (bugs) add label “**bug**”, and one of the following:

- “**critical**” - failures and serious bugs which prevent component running (unhandled exceptions). Should be fixed immediately.

- “**major**” - anything that seriously hampers productivity but doesn't actually prevent work from being done. Fixed by next release.

- “**minor**” - these are "nuisance" bugs. A default setting not being applied, a read-only field showing as editable (or vice-versa), a misleading error message, etc. Fix for this release if there are no higher-priority issues, otherwise the following release.

It is also possible to use issue tracker for requesting new specific feature. For this use label “**feature**” if something new need to be implemented. If this is suggestion for major performance enhancement of existing functionality use “**enhancement**”.

The next figure shows the screenshot of the initial issues submitted to the GitLab Issue tracker.

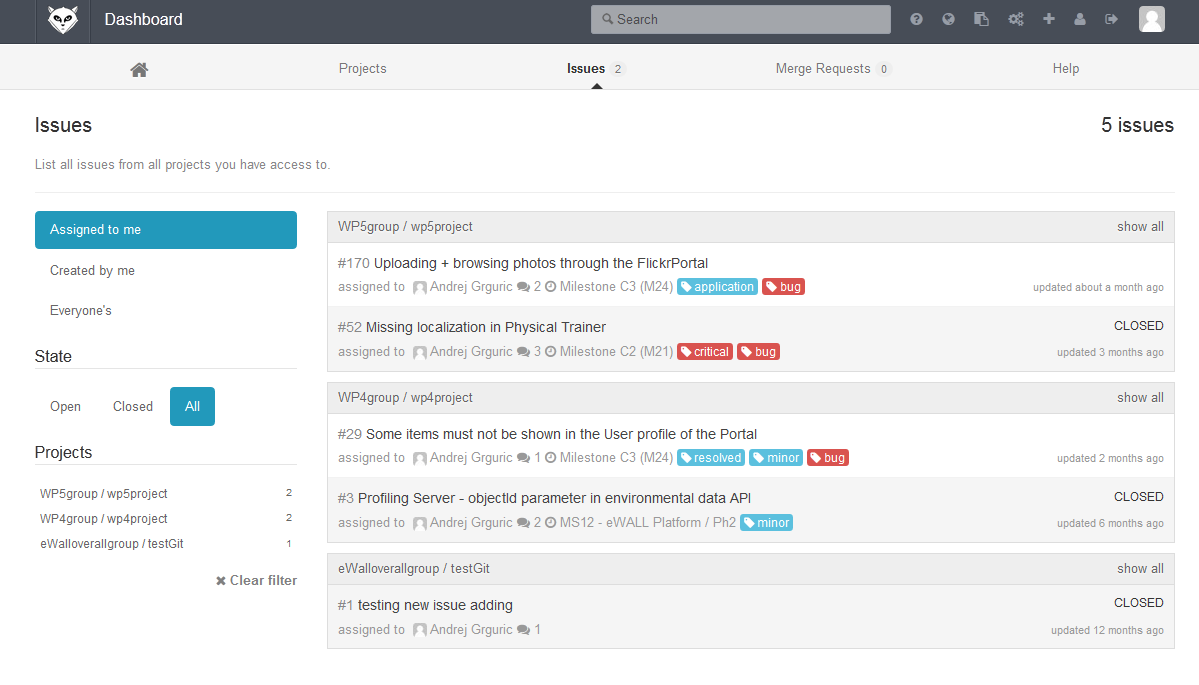


Figure 6 Screenshot of the eWall GitLab Issue Tracker

### Automation build tool (Maven)

Maven [9] is a software management and comprehension tool that is based on the concept of Project Object Model (POM) and which can manage project building, reporting, and documentation from a central piece of information. It provides:

* Easy build process,
* Uniform build system,
* Quality project information,
* Guidelines for best practices development,
* Achieved characteristics: visibility, reusability, maintainability.

Maven is not only a build-automation and project management tool but is also a set of guidelines to develop software projects. It uses a simple and generic structure, independent from programming language, and therefore it is easy to understand for any developer familiar with Maven. Maven’s main advantage is in that any piece of information related to the software project is stored in the POM file. This file stores metadata information like the name of the project, version, and developers that worked on it, as well as building information. POM files can define parent project relations and in this case child POM files will inherit most of the information from the parent project. This way child POM files only have to define the information that differentiates it from its father. The other relation between projects is dependency. When one project is dependent on other, the second one must be included in the compiler settings when building the first one. Maven will automatically configure the compiler with this information, so developers will always find the same building configuration regardless of the local environment (IDE, OS, JDK etc.). Furthermore, it is possible to automatically generate useful information about the software, e.g. change logs and testing results so the evolution can be tracked. All the knowledge on any given Maven project is described in a single file meaning that full understanding of the project is easier and learning curves for new developers are faster. The POM allows Maven to reference remote repositories and automatically download resources that are called artefacts. Additionally, it is possible to automatically deploy generated artefacts to remote repositories. It allows Maven to support not only the build process by also the deployment. This allows Continuous Integration approach (described in section 4.3.9). Used or locally compiled artefacts are cached locally and updated dynamically, ensuring that each developer has the same installed version. This means that a single POM file is sufficient to setup the IDE, download code and libraries, build the project and run it. The same POM file describes where to find test definition and continuous integration server; so development can start immediately. It also provides meta-information on who was involved in the development of the project and any other piece of information relevant to the software project cycle [10].

#### Setup and usage in eWall

Parent POM, extending spring-boot-starter-parent (v1.2.3-RELEASE) was used for all platform components. It defined all common properties for all the child projects and links to different eWall Nexus repositories (snapshots, releases, third-party).

### Maven repository manager (Nexus)

Nexus [11] is an open source artefact repository that is commonly used in continuous integration environments together with build automation tools such as Maven. Its purpose is storing artefacts such as JAR and WAR files produced by the build and exposing them for download via the HTTP protocol.

Artefact repositories are critical for modern build tools that allow declaring external dependencies (e.g. libraries) of a software project in a descriptor file, such as pom.xml in Maven. Since dependencies are clearly identified in the descriptor, developers do not need to store them into the source code versioning system, as it was commonly done in the past; this is beneficial for reducing the repository size and the duration of checkout operations. On the other hand, the build tool needs an artefact repository from which dependencies can be downloaded to a local cache upon first usage.

While Maven provides a public artefact repository, called Maven Central, that is constantly updated and contains all major open source libraries commonly used in Java projects, setting up a private repository such as Nexus provides several benefits. First of all, Nexus can work as a local proxy with respect to remote repositories such as Maven Central. This allows to speed up download times from the local network when the same artefact is downloaded multiple times, because Nexus fetches it from the remote source only once and then serves it from the local cache. Secondly, it is possible to set up private repositories to provide third party libraries that are not available on public repositories, such as commercial libraries that require a license. Thanks to the private repository these libraries can be downloaded by Maven just like the public ones, provided the build descriptor is configured to use the local Nexus. A third benefit of using a private Nexus is that it can be used to store artefacts produced by your own projects, such as JARs and WARs in a large Java EE project like eWALL. This means that a developer working on a certain library of the project that depends on other parts of the same project will not need to build every dependency on its own machine, consuming time and CPU power, but instead download a pre-built version of the library from Nexus. This works especially well in continuous integration environments in which the most recent versions of all components are constantly built by the CI server. Finally, being a centralized repository that stores all the dependencies used in internal projects, Nexus can be used as a check point for security-related assessments that verify whether vulnerable or out-of-date libraries are being used.

In addition to dependency resolution during the build phase, artefact repositories are also fundamental in continuous delivery processes, in which the software lifecycle is organized as a pipeline with multiple steps in which the build step is followed by integration testing, user acceptance testing and finally deployment to the production environment. All of these steps need to download a certain version of the application, which was built in the first step, and deploy it to different environments, while making sure that the same artefacts are being used across the pipeline without intermediate compilation steps that may alter them. Therefore, Nexus can be used to store the artefacts produced at the first step of the pipeline and provide them to further steps.

#### Setup and usage in eWall

In eWALL, a Nexus server was setup to exploit the aforementioned benefits. The Nexus version used was the 2.8.1-01. In order to install Nexus, the latest release (at that time) was downloaded as a .tar archive. Once the setup was finished, three repositories were configured: one for Releases, another one for Snapshots and, finally, one for third-party libraries (an example of third-party library is the one used to interface eWALL with lighting system Philips Hue). These repositories are the ones addressed by the pom.xml files, while the artefacts are uploaded directly by Jenkins.

### Inversion of control container server (Spring IoC Container)

Inversion of control (IoC) describes a design in which custom-written (task-specific) portions of a computer program receive the flow of control from a generic, reusable library (unlike in traditional procedural programming where the custom code that expresses the purpose of the program calls into reusable libraries to take care of generic tasks).

IoC is also known as dependency injection (DI). It is a process whereby objects define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean.

#### Setup and usage in eWall

For eWall Cloud platform components Spring Boot[[6]](#footnote-7) v1.2.3 was used which builds on top of the corresponding version of Spring Framework.

### Deployment server (Tomcat)

Apache Tomcat Bibliography entry is an open-source web server developed by the [Apache Software Foundation](https://en.wikipedia.org/wiki/Apache_Software_Foundation) (ASF). It implements some Java EE specifications such as Java Servlet, Java Server Pages (JSP), and WebSocket, and provides a “pure Java” HTTP web server environment for Java code to run in.

Apache Tomcat's components include: Catalina, Coyote, Jasper, Cluster, High availability, and Web application.

* ***Catalina*** is Tomcat' servlet container that implements Sun Microsystems' specifications for servlet and JSP. In Tomcat, a Realm element represents a "database" of usernames, passwords, and roles (similar to Unix groups) assigned to those users. Different implementations of Realm allow Catalina to be integrated into environments where such authentication information is already being created and maintained, and then use that information to implement Container Managed Security as described in the Servlet Specification.
* ***Coyote*** is a Connector component for Tomcat that supports the HTTP 1.1 protocol as a web server. This allows Catalina, nominally a Java Servlet or JSP container, to also act as a plain web server that serves local files as HTTP documents.
* ***Jasper*** is Tomcat's JSP Engine. Jasper parses [JSP files](https://en.wikipedia.org/wiki/JSP_files) to compile them into Java code as servlets (that can be handled by Catalina). At runtime, Jasper detects changes to JSP files and recompiles them.
* ***Cluster*** has been added to manage large applications. It is used for [load balancing](https://en.wikipedia.org/wiki/Load_balancing_(computing)) that can be achieved through many techniques. Clustering support currently requires the JDK version 1.5 or later.
* ***High availability*** feature has been added to facilitate the scheduling of system upgrades (e.g. new releases, change requests) without affecting the live environment. This is done by dispatching live traffic requests to a temporary server on a different port while the main server is upgraded on the main port. It is very useful in handling user requests on high-traffic web applications.
* ***Web application*** has also added user- as well as system-based web applications enhancement to add support for deployment across the variety of environments. It also tries to manage sessions as well as applications across the network. In eWall, this component is the main focus in which the various Java Web Applications are deployed, including the cloud platform, service bricks, management portal and the web based Graphical User Interface (GUI) for the end users.

Tomcat 7.x, which is used in eWall, implements the Servlet 3.0, JSP 2.2, and EL 2.2 specifications. It requires Java version 1.6, although previous versions have run on Java 1.1 through 1.5. Native wrappers, known as "Tomcat Native", are also available for [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) and Unix for platform integration.

#### Setup and usage in eWall

In eWALL platform, Tomcat 7.0.54 was used as web-server and Java servlet container on which Java Web Applications are deployed. On each environment, a separate Tomcat server instance was installed and configured. The installation was done through one of the most widely used Linux package management tools, the *Advanced Packaging Tool* (“apt-get” tool).

The Tomcat web-server allows to launch, stop and deploy web applications independently. Among the other benefits provided by this approach, a remarkable one is the *hot deployment* of applications. Thanks to this fact, it is possible (cf. Figure 7) to deploy a new application/service (or an updated version of an already existing one) without stopping the whole server and consequently every other applications running on it.

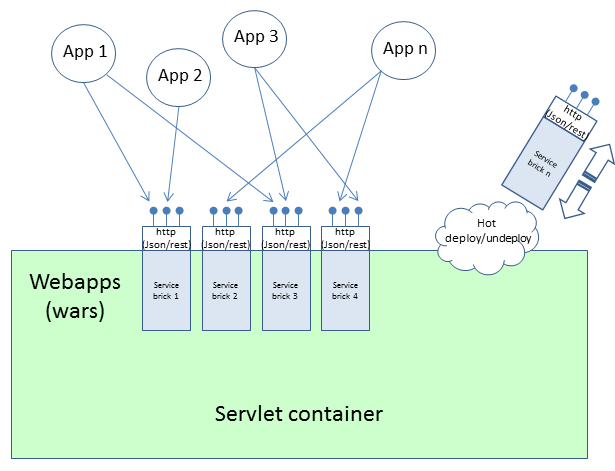


Figure 7: Deployment of service bricks (example of hot deployment of a micro service in eWALL)

Once a web application is deployed, the servlet container automatically activates it and the related endpoints are made available at the specified URLs for usage by the applications. If a previous version of the application was already present, the servlet container automatically un-deploys it and activates the new one. While Figure 7 depicts specifically the hot-deployment mechanism for eWALL service bricks, the same mechanism stands indeed for the deployment of all eWALL web applications and platform services, as the eWALL system is based on a composable micro-service architecture, where every micro-service is packaged as a web application archive.

### Continuous integration server (Jenkins)

Jenkins [12] is an open source software tool which allows applying *continuous integration* practices in software development projects. Continuous integration consists of a process in which software modules written by different developers are collected and built, and the resulting artefacts are assembled into an integrated platform, deployed on a runtime environment and tested. Both the build and testing phases happen automatically, according to policies that can be configured by an administrator.

Typically, Jenkins is connected to a source code repository which is periodically checked for updates. In eWALL, Jenkins is configured to monitor the code on GitLab repository for updates and, whenever a new code revision is pushed to it, Jenkins checks it out locally, builds it with the Maven build tool and, in case of successful build, deploys the software artefacts to a runtime environment in which they are executed. In addition to that, Jenkins functionalities can be extended adding custom jobs. This feature allows defining specific tasks to be executed at different phases of the whole continuous integration chain. In particular, tasks for running automated tests on the platform can be scheduled to be executed just after the deployment. This process allows for early detection of errors that cannot happen in the single modules, but show up only in the integrated system due to their interaction or interdependencies.

In addition to the above mentioned features, Jenkins can also send out notifications in various formats (e-mail, SMS, etc.) according to policies that can be configured by the administrator. Typically, e-mails are sent out whenever a build fails after a new revision of code has been pushed to the code repository. This allows to immediately make developers aware that what they have released must be fixed.

#### Setup and usage in eWall

In eWALL, Jenkins version 1.626 was used. Since Jenkins packages are available in a Linux package management repository, setup is pretty straightforward and can be performed through the Linux *Advanced Packaging Tool* (“apt-get” tool), available on Debian-based distributions (such as Ubuntu). Several Jenkins jobs were configured, in order to obtain the continuous integration chain as described above.

### Server monitoring tool (Zabbix)

Zabbix [13] is an open source software tool for monitoring the availability and performance of IT infrastructure components. It consists of a web application with a very rich web based interface, which offers unlimited possibilities for configuring the monitoring of remote components.

Based on the configurations created by users with administrator role, Zabbix connects to remote elements and gathers data into a local database. Data can be related to status, performance indicators, availability, usage quotas, filesystem status and so on, as well as custom indicators that can be defined via the configuration interface. In order to allow Zabbix to retrieve data, remote hosts must run a Zabbix agent, a software module that enables the communication between the Zabbix server and the monitored clients.

The data collected by Zabbix can be accessed and visualized from the web interface, either as raw values or represented as graphs. Zabbix also allows configuring a custom user-level dashboard with personal areas, where a user can set references to specific graphs and data sets.

Among the many configuration settings allowed by Zabbix, it is possible to define triggers, i.e. special conditions which require specific handling, and to associate events to triggers, so that actions can be taken whenever a trigger happens. This feature is commonly used to notify automatically (e.g. via e-mail) responsible actors about the malfunctioning or unavailability of some systems or services, or whenever thresholds related to resources usage are reached.

In the context of eWALL, two instances of the Zabbix monitoring system have been set up:

* the first instance is installed in the eWALL cloud itself, and is configured to monitor the availability and the system performance of the whole eWALL cloud infrastructure, including the physical server hosting the OpenStack platform and the virtual machines running on it.
* the second instance is installed at an external premise (currently in HPE Italy), since its main purpose is to monitor the availability of the physical server hosting all the eWALL services (including the first instance of Zabbix). This separation was also put in place to provide redundancy in the monitoring systems, to overcome situations where one of the two systems might suffer failures.

### Front-end JavaScript Framework (AngularJS)

AngularJS[[7]](#footnote-8) is an open-source framework used to create dynamic and flexible client-side applications. This framework extends traditional HTML and easily binds models with views (AngularJS is built according to MVC architecture), allowing developers to create single-page applications. This particular kind of web application provides a more fluid user experience, dynamically loading and changing the page content in response to user action.

# eWALL Platform within overall eWALL system architecture

The eWALL system involves an ecosystem of sensing devices, computing resources and display devices. The interaction within the elements of the ecosystem is enabled by the central component of the system, eWALL Platform with core running in cloud environment and which aggregates data coming from sensors, analyses them, transforms them into specific formats suitable for the user/system that needs them and infers new higher level knowledge by applying advanced reasoning on data. The objective of the ecosystem is providing added values services to the end users.

The purpose of this chapter is to describe eWALL Platform and its positioning within the overall eWALL system. In a broad sense of the word, the platform refers to as all developed software that is used as a base upon which all applications can run on top of any devices. However, the term eWALL Platform is also used as organizational unit of work under the responsibility of WP4. Based on this, we consider the eWALL Platform as a set of software artefacts developed within work on WP4 that are used as a base upon which other eWALL artefacts, namely devices and processing components from WP3 on one side, and personalized services and applications from WP5 on the other side, are implemented, integrated and deployed. The eWALL Platform provides core services to eWALL system which include secure and reliable communication between home and cloud environment, scalability, data management, security and privacy handling. Beside these core services, eWALL Platform also provides generic reusable set of services which ease the development of personalized services and applications, namely profiling, notification support, intelligent decision making, provisioning and configuration.

Based on eWALL multi-tier architecture described in eWALL Reference Model in D2.7 [5], the core functionality of eWALL Platform is positioned on Data Management and Business Services tiers. The Data Management tier provides global persistence services and various adapters that provide access, search and update services to databases and its data stored in a database management system. The Business Services tier provides enterprise-level services, and is also responsible for protecting the integrity of enterprise resources at the business logic level. Since eWALL Platform is also responsible for secure and reliable connection between home and cloud environment, the part of Data Sources tier, responsible for transfer of data to Data Management tier is also considered as part of eWALL Platform. The rest of Data Sources tier functionality, namely representing all sensors and gateways that aggregate and process data from sensing environment is related to the work of WP3.

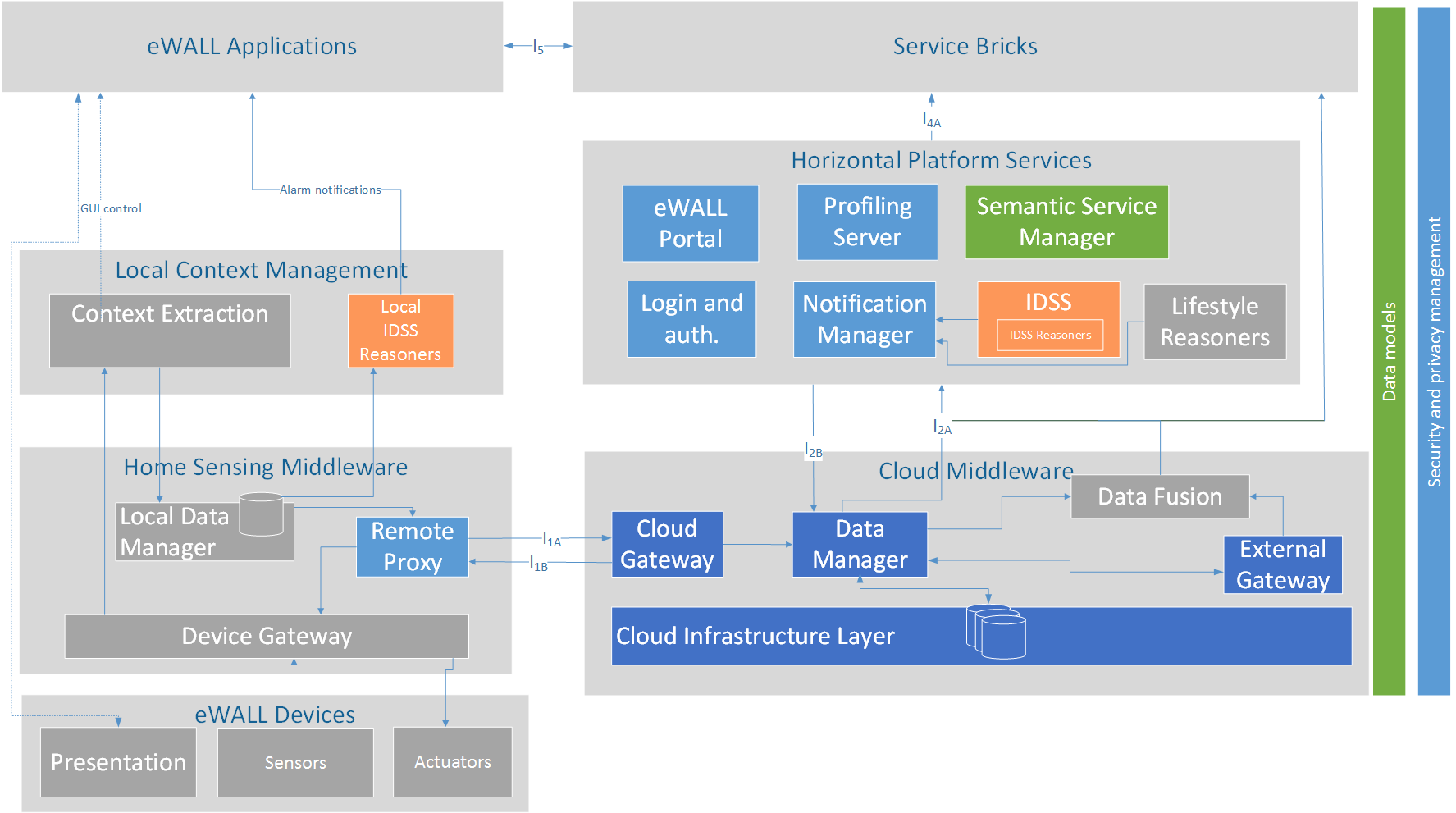


Figure 8: eWALL Platform and other WP4 components within the eWALL architecture

The basic eWALL Platform block diagram is presented in Figure 8. Main artefacts of the eWALL Platform implemented are:

* Remote Proxy (T4.4) - responsible for connecting home sensing environment with eWALL Cloud,
* Local Data Manager (T4.4) - responsible for retrieving sensing data from local storage and sending this data to eWALL Cloud using Remote Proxy if allowed by local privacy policy,
* eWALL Portal (T4.4) - responsible for providing interface and support for administrative and technical management of the eWALL Platform,
* Login and authentication service (T4.4) - responsible for authenticating users/clients and to verify access and permissions
* Notification Manager (T4.4) - processing entity that is capable of managing (i.e. storing, forwarding or deleting) all eWALL notifications,
* Profiling Server (T4.4) - manages user profile information represented by user profile ontology,
* Data Manager (T4.3) - responsible for basic data processing, format exchange, and input validation of received data,
* Cloud Gateway (T4.3) - responsible for interconnecting eWALL Cloud with Sensing Environments (Home and Mobile),
* Intelligent Decision Support System (T4.2) - responsible for decision making activities in eWALL system,
* Data Model (T4.1) - represents data model for eWALL Platform.

The functionality of these artefacts is based on the functional architecture of eWALL system, that was presented in deliverable D2.7 [5]. Details about current version of RESTful APIs are presented in *Appendix A: eWALL Platform RESTful interfaces*

The above Figure 8 presents also how the work on the eWALL Platform was divided among tasks and partners on WP4. As mentioned earlier, this deliverable describes implementation and integration process that lead to development of the second version of eWALL Platform, and implementation of artifact that have been implemented within the work of this deliverable (illustrated light blue), namely Remote Proxy, eWALL Portal, Login and authentication service, Notification Manager, Profiling Server and Security and privacy management. The other artefacts have been implemented in the scope of other WP4 tasks and implementation is described in related deliverables; D4.1.3 Semantic model of eWALL middleware services [1], D4.2.3 Intelligent support system for eWALL [2] (illustrated orange), and D4.3.3. Cloud middleware services for eWALL [3] (illustrated dark blue). Components illustrated grey are result of the work in WP3 and WP5.

## Data flows between home and cloud platform components

Basic eWALL system interaction flows that present most common interactions between different parts of eWALL system, such as system initialization flow, data push flow, data pull flow, notifications flow, are presented in D2.7 [5]. This section describes in more details only data flows between home and cloud platform components. The flow presented on Figure 9 show basic flow of data at local platform components initialization.



Figure 9 Data flow between home and cloud platform components at initialization

First, Remote Proxy sends credentials to login to the system and obtain authentication token that is then used in all subsequent HTTP requests towards eWALL Cloud. Afterwards, Remote Proxy registers sensing environment with Point of Contact with details on local platform version, status, expiration, etc. Cloud Gateway checks whether sensing environment is provisioned and enabled in the (via eWALL Portal), and if it is not already online. If successfully, Remote Proxy obtains devices configuration data, such as devices types, room name etc. This information is then used by Local Data Manager to create subscriptions for receiving data per each device from local database. Next, Remote Proxy performs synchronization sync check in case there was an unexpected system shutdown of Home PC. For pushing data from cloud to home environment, eWALL uses a technique call HTTP long polling. This is started at the end of initialisation phase.



Figure 10 Data push flow from local platform to cloud platform

After initialization, local platform components are ready to start sending data to cloud and to receive commands from cloud. As presented on Figure 10, when new processed data from WP3 is stored in local database, the data is immediately send to Local Data Manager via database continuous changes API. LDM adapts this data to commons data model (D4.1.3), checks privacy policy and if allowed sends data to cloud via Profiling Server and Cloud Gateway as designed in D2.7 data push communication model. Prior to that, if there is a Local Reasoner (D4.2.3) subscribed to this data type, the data will be process by Local Reasoner. If alarm needs to be send to cloud Local Reasoner uses Remote Proxy and Cloud Gateway to send notification to Notification Manager.

For receiving commands from cloud Remote Proxy uses a technique called HTTP long polling. In principle, the Cloud Gateway holds the request from Remote Proxy open until new event occurs. Once event occurs, the Cloud Gateway responds and sends the new data in response. When the RP receives the new data, it immediately sends another request, and the operation is repeated. This is used as server push feature and it is used to send actuator commands from cloud to home environment, as well as for remote configuration of local platform via eWall Portal, as presented in diagram in Figure 11.



Figure 11 Actuator control command and remote configuration data push from cloud to local platform

# eWALL Platform Implementation

This chapter describes the implementation of eWALL Platform artefacts. As already mentioned, the eWALL Platform is composed of artefacts that were developed in scope of this deliverable (which are in more detail presented in this chapter) but also of artefacts that were developed within other WP4 tasks (these are just briefly listed in this chapter with a reference to the related deliverable with more implementation details).

Each artefact has its own ID card (table) with links to source, documentation, deployment location, version, responsible partners and task, and black box description. Additionally, all T4.4 artefacts also have description of requirements and features.

All eWALL Platform cloud components are built as Web Application Archive (WAR file) and deployed and integrated at Apache Tomcat Web servers running on eWALL OpenStack IaaS (see D4.3.3[3] for more details).

## Remote Proxy

|  |  |
| --- | --- |
| Artefact: Remote Proxy | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/remote-proxy/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/remote-proxy |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.remoteproxy$remote-proxy/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/remoteproxy/remote-proxy/1.0.0/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.remoteproxy$remote-proxy/javadoc/ |
| Version | 1.0.0 |
| Responsible partner(s) | ENT, UPB |
| Related to task | T4.4 |

### Black box description

The Remote Proxy (RP) is responsible for connecting the Home Sensing Environment with the eWALL Cloud. It provides necessary interfaces for remote access and configuration of sensor and actuators in sensing environment, and is responsible for transmission of sensor data to the eWALL Cloud. It is responsible for sending processed and indexed measured sensor data, to receive control and configuration data to Sensing Environments and for bidirectional application data. It directly communicates with Cloud Gateway component from Cloud Middleware and supports both *message based* and *pull and push* communication. Currently, RP is implemented as standalone Java component that can run on any JVM.

### Requirements

List of related requirements from D2.7 [5]:

* Gen\_001: Flexibility - ability to support a variety of market available or eWALL developed user and network devices
* Gen\_002: Scalability - ability to easily scale the eWALL platform to all envisioned use-cases
* Gen\_003: Traceability - ability to log and track the taken actions throughout the platform operation
* Gen\_004: Extensibility - ability to easily integrate novel devices in the platform (transparent protocol formats and protocol messages)
* Gen\_005: Reliability - ability to provide reliable communication within the platform and always on-time reaction
* Gen\_006: Compatibility - ability to integrate various information from various devices in a user transparent manner
* Gen\_018: Maintainability and configurability - ability to easily maintain and configure the system after deployment
* Spec\_009: Remote accessibility – ability to provide remote access to the eWALL platform
* Spec\_023: Communication **–** ability to enable inter-component message-based (or event-based) and call-based communication between distributed components
* Fun\_003: Continuous reporting – ability to provision continuous communication of user data among all envisioned interfaces – continuous message communication (including communication to/from cloud environment)

### Features

Main features of Remote Proxy currently implemented are:

* Performs registration and configuration of Remote Proxy components and Sensing Environment at eWALL Cloud via REST interface.
* Sending point of contact information from Sensing Environment via REST interface that is used for reaching Sensing Environment from remote.
* Responsible for realization of communication protocol and adaptation, especially in regards to the quality, security, capacity and reliability of the communication between the home and cloud environments. Sending all measurements data coming from WP3 device components, including accelerometer data, temperature, humidity, luminance, movement, bed sensor etc. via REST interface and AMQP messages.
* Besides implemented RESTful interface for message exchange between the Sensing Environment and eWALL Cloud, a second communication mechanism, based on Advanced Message Queuing Protocol (AMQP) has been developed for high volume/critical sensor data upload. Registration and sending of measurements from WP3 devices is currently implemented. The measurements are sent to a RabbitMQ server in different queues (one for each measurement type) and with different routing keys. The routing keys will enable future differentiation between messages coming from different eWALL systems, sensing environments or devices. More about AMQP/RabbitMQ can be found in D4.3.2 [3].
* Responsible for adaptation of data being transferred between Sensing Environment and eWALL Cloud.
* Interfaces Device Gateway to request raw/unprocessed data directly from sensor and control actuators.
* Provides security features such as token based authentication and authorization, and encryption of sent data
* Support for basic remote host system monitoring

Features that are planned to be implemented in next cycle:

* Improved support for remote administration and monitoring (alarm generation)
* Support for subscription/notification based communication

## Local Data Manager

|  |  |
| --- | --- |
| Artefact: Local Data Manager | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/stable/local-data-manager/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/local-data-manager |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.local.datamanager$local-data-manager/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/local/datamanager/local-data-manager/1.0.0/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.local.datamanager$local-data-manager/javadoc/ |
| Version | 1.0.0 |
| Responsible partner(s) | ENT, UPB |
| Related to task | T4.4 |

### Black box description

The Local Data Manager (LDM) is a component that provides the trust and privacy support of the local data being processed or kept at the Home Sensing Environment (namely Home PC). The passing, receiving and managing the data products is handled by the LDM based on a set of ethical, privacy and security requirements and policies described addressing the requirements in D2.7 [14] and D2.4 The LDM will be mainly driven by the locally running eWALL sensor data processing outputs for local context extraction and the local IDSS reasoners, applications and services (see Figure 8). The communication of the LDM with the cloud will go through to the Remote Proxy component. The functionality also includes trust monitoring and key or certificate management as well as authentication and authorization for local running eWALL components.

### Requirements

List of related initial requirements from D2.7 [5]:

|  |  |
| --- | --- |
| Gen\_014 | **Privacy** – ability to keep personal information from being disclosed and shared with unauthorized parties |
| Gen\_020 | **User data separation** – ability to create pseudo identifiers for privacy protection supported by eWALL user and network devices |
| Gen\_022 | **Anonymity** – ability to switch off eWALL sensors and devices and manage the deletion of raw data from these sensors |
| Spec\_009 | **Remote accessibility** – ability to provide remote access to the eWALL platform |
| Spec\_010 | **Priorities management** – ability to handle different simultaneous requests and messages with different priority levels |
| Spec\_017 | **Identification, authentication and authorization** – ability of system components to identify, authenticate and authorize an entity (human users and other system components) that wants to use them before allowing them access to resources |
| Spec\_018 | **Confidentiality** - ability to maintain confidentially (the way in which the information disclosed or managed by the system is treated) of identifiable data, including controls on storage, handling, and sharing of data |
| Spec\_019 | **Integrity** – ability to detect data modifications and prevent unauthorized modifications, especially related to service user data, sensor data and commands sent to actuators. |
| Spec\_020 | **Non-repudiation** – ability to trace back every action on sensitive assets to the person or system component that performed it |
| Spec\_021 | **Auditing** – ability of a system to log all actions on sensitive assets, including failed access attempts |
| Spec\_022 | **Consent specification** – ability to provide a usable interface to capture the consent of the end-user about sharing data with services |
| Spec\_023 | **Communication –** ability to enable inter-component message-based (or event-based) and call-based communication between distributed components |

### Features

The LDM is built based on the security and privacy components, defined in D2.4 [5]:

* User Identification – Responsible for recognizing a valid user's identity
* Authentication – This component provides authentication for entitles which send inbound data or which consume data. Authentication is the process of verifying the claimed identity of a user.
* Authorization – Provides authorization for an entity (user or component) with specific role for what component or data are allowed to be sent or consumed. Basically, it checks that the given entity has the right permissions to access a certain component or data.
* Consent – It can be described as privacy management module that allows user to manage access to user related data to other entities (users)
* Encryption - Provides encryption/decryption for both data transmission (e.g. SSL) and stored data
* Audit logger - Provides functions for security audits.
* Configuration – Responsible for handling all security and privacy related configuration.

## eWALL Portal

|  |  |
| --- | --- |
| Artefact: eWALL Portal | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/stable/ewall-portal/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/ewall-portal |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.portal$ewall-portal/ |
| Deploy | http://ewall.radio.pub.ro/platform-dev/ewall-portal/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/portal/ewall-portal/1.0.0/ |
| Version | 1.0.0 |
| Responsible partner(s) | HP, ENT |
| Related to task | T4.4 |

### Black box description

The eWALL Portal is responsible for providing interface and support for administrative and technical management of the eWALL Platform. It is designed for:

* Healthcare System Administrators: personnel belonging to or representing the healthcare system administrative staff, which manage the policies for the provisioning of the healthcare services to end users (e.g. for defining which caregivers must/can assist end users) in a specific geographical or administrative region
* Technical Administrators: personnel trained for managing the technical aspects of the system (e.g. technical configuration, initial addition of users and assignment of roles, etc.)

From the administrative or technical personnel’s perspective, the system allows them to manage policies or technical aspects of the platform. It will have a mechanism where:

* the responsible actors can be registered into the system and the relationships among them can be set
* the applications/services related to a specific user can be configured
* the sensing environment (set of sensing devices installed at a target end user’s home) can be associated to a specific user in the system
* the relevant events related to a specific user can be monitored

It will also have specific dashboards showing aggregated information, which depends on the role of the active user, e.g. related to system status for administrators, to target end user events for caregivers, etc. The ideal way to provide the above mentioned functionality is by means of a web portal, which allows access from everywhere in a modality which is independent from the client operating system used. If properly designed it might also offer access via mobile.

In the following we provide a description of the overall envisioned functionalities, which will be then defined at a finer level in terms of technical specifications.

### Requirements

List of related requirements from D2.7 [5]:

* Gen\_001: Flexibility - ability to support a variety of market available or eWALL developed user and network devices
* Gen\_003: Traceability - ability to log and track the taken actions throughout the platform operation
* Gen\_006: Compatibility - ability to integrate various information from various devices in a user transparent manner
* Gen\_007: Responsiveness – ability to dynamically react and/or reconfigure eWALL platform elements
* Gen\_008: Multiuser capability – ability to support multiple eWALL users with guaranteed profiling
* Gen\_011: Security – ability to secure the eWALL users’ data from obtrusive and accidental eavesdropping
* Gen\_018: Maintainability and configurability - ability to easily maintain and configure the system after deployment
* Spec\_009: Remote accessibility – ability to provide remote access to the eWALL platform

### Features

Currently implemented features:

* Basic Support for eWALL system provisioning of sensing environments, users, devices, sensors and actors via REST interface
* Web based access for provision of sensing environments and associated primary users
* Creation and management of “Regions”: The eWALL system will be adopted in different Countries, each one with specific regulations and policies with regard to the healthcare services provisioning. The portal provides means for creating so-called “Regions”, each one representing a healthcare system specific to a geographical or administrative area. Such kind of initial, top level configuration is made by a Super User, defined at the time of the deployment of the platform by the eWALL technical staff.
* Region Administration: Every Region needs administration; therefore, within each Region the Portal makes possible to add users with administrative roles. The Region Administration section of the portal provides means for creating and managing (add/delete/modify) actors with the “region administrator” role for a specific healthcare system and creating and managing (add/delete/modify) actors with the “technical administrator” role for a specific healthcare system
* End User registration: Every end user (both target end users and caregivers) must be properly registered into the system by setting adequate profile information, and the relationship among them (e.g. “who are the caregivers who are assisting a specific target end user?”) must also be defined. In fact, all the back-end components which perform behavioural patterns analysis, messaging, notification triggering, etc. must know information about specific users and their profile, and also the relationship between the caregivers and the end users, to know e.g. who must be notified about events, when needed. Therefore, the Portal provides the following functionalities:
  + Creating and managing (add/delete/modify) new target users
  + Creating and managing (add/delete/modify) new caregivers (formal or informal)
  + Associating a target end user to one or more caregivers
  + Associating a target end user with a “sensing environment”. This step enables the platform to accept data coming from a remote home

In addition, for each user the Portal provides the management of profile data.

* Applications setting: every application interacts with a set of web service endpoints (“micro services”), each one providing specific functionality or data retrieval. The Portal allows, for each application, to configure information like the name and the endpoint for access and, most importantly, to specify the exact set of back end services required. This information is crucial because enables the token-based authentication and authorization mechanism implemented in eWALL (c.f. sections 6.6 and 6.7).
* Services setting: the Portal allows to configure the back end services deployed on the system, by specifying name and endpoint.
* Application assignment: For each target end user, a responsible actor must be able to define the set of applications that the user is entitled to use. Therefore, for each user, there is a related area in the portal where all the required configuration and settings are filled in and saved into the central platform. This area, which is accessed from the user profile page, allows:
  + Visualization of the list of applications available in the platform,
  + For each target end user, visualization of the list of the applications in the platform that can be assigned to him (e.g. because he/she has a specific set of sensors in the home),
  + Assignment of applications to an end-user.
* Event monitoring and history: Relevant events must be tracked within the platform and the actors in the system must be able to monitor them. Depending on their role, they might be able to see different subsets of such data. While the “event tracking” is dedicated to visualization of instant (recent) data, all the events history should be saved and made available for later access. This feature is under evaluation and will be considered for the next project cycle.

## Notification Manager

|  |  |
| --- | --- |
| Artefact: Notification Manager | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/stable/notification-manager/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/notification-manager |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.notificationmanager$notification-manager/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/notification-manager/1.0.0/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.notificationmanager$notification-manager/javadoc/ |
| Documentation (Swagger) | http://ewall.radio.pub.ro/platform-dev/notification-manager/index.html |
| Version | 1.0.0 |
| Responsible partner(s) | UKIM |
| Related to task | T4.4 |

### Black box description

The notification manager is a processing entity that is capable of managing (i.e. storing, forwarding or deleting) all eWALL notifications. It is a single central block where all notifications are being managed, so the user is not overloaded by all information coming randomly from independent notification sources. The main role of the notification manager is to manage the notifications (alarms) and messages that originate from a specific set of processing blocks, e.g. service bricks and reasoners.

### Requirements

The notification manager exhibits the following system requirements:

* Gen\_001: Flexibility - ability to support a variety of market available or eWALL developed user and network devices
* Gen\_002: Scalability - ability to easily scale the eWALL platform to all envisioned use-cases
* Gen\_003: Traceability - ability to log and track the taken actions throughout the platform operation
* Gen\_004: Extensibility - ability to easily integrate novel devices in the platform (transparent protocol formats and protocol messages)
* Gen\_005: Reliability - ability to provide reliable communication within the platform and always on-time reaction
* Gen\_006: Compatibility - ability to integrate various information from various devices in a user transparent manner
* Gen\_018: Maintainability and configurability - ability to easily maintain and configure the system after deployment
* Spec\_009: Remote accessibility – ability to provide remote access to the eWALL platform
* Spec\_023: Communication **–** ability to enable inter-component message-based (or event-based) and call-based communication between distributed components
* Fun\_003: Continuous reporting – ability to provision continuous communication of user data among all envisioned interfaces – continuous message communication (including communication to/from cloud environment)
* Gen\_007: Responsiveness - ability to dynamically react and/or reconfigure eWALL platform elements
* Gen\_008: User mobility input: ability to track the position and the movement of users in-house (important for various eWALL services)
* Spec\_018: Confidentiality: ability to maintain confidentially (the way in which the information disclosed or managed by the system is treated) of identifiable data, including controls on storage, handling, and sharing of data

### Features

The Notification Manager acts as a communication gateway/interface that forwards all necessary notifications (alarms) and messages, which are triggered by the respective processing blocks, to the specific notification user interface(s) UI(s). The Notification Manager is designed to combine and prioritize notifications based on their nature origin and destination. For the highest priority notifications and alarms the Notification Manager sends the notification/alarm content via email to the respective caregivers and primary user. Moreover, it can decide to discard specific ones in case of possible information overload. Additionally, the Notification Manager is capable of sending the required information to a specific, i.e. target, UI device based on the user’s (patient’s) location. The Notification Manager also handles caregiver specific notifications that are fed to the eWALL caregiver application.

Currently implemented features:

* Receiving notifications from IDSS reasoners and distributing them towards the corresponding user interface.
* Prioritization of notifications (prioritized scheduling and sending of notifications) based on the priority coefficient carried in the notification message. *Comment: Higher priority coefficients reflect a lower overall notification priority.*
* Sending highest priority notifications/alarms via email. The recipients of these notifications are the primary user, informal and formal caregivers. The e-mail addresses of all recipients are gathered from the corresponding user profile via the profiling server.
* Sending notifications to a target UI device of interest based on the user’s location. The user location is acquired from the user’s measurement data via the profiling server. *Example: If the user is in the same room as the eWALL main screen (i.e. the living room), then the Notification Manager will send all notifications to it. If the user leaves the room (and has a mobile device with an installed notification UI), then the Notification Manager will send all future notifications to the hand held device. In this manner, the user can be always fed with important information regardless of its actual whereabouts.*
* Cross-component artefacts compatibility of the Notification Manager. (i.e. Token-based authentication and authorization compatible interfaces*,* component configuration compatibility, automatic REST API documentation generation compatibility).
* Management (storing and handling) of the notifications in the cloud data base
* Handling caregiver based notifications
* Interfacing and sending notifications with caregivers’ application
* Policy based reasoning for sending notifications (e.g. send only specific notifications to the underlying caregiver/user based on the caregiver’s/user’s preferences)

## Profiling Server

|  |  |
| --- | --- |
| Artefact: Profiling Server | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/stable/profiling-server/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/profiling-server |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform$profiling-server/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/profiling-server/1.0.0/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform$profiling-server/javadoc/ |
| Documentation (Swagger) | [http://ewall.radio.pub.ro/platform-dev/profiling-server/index.html](https://www.google.com/url?q=http://ewall.radio.pub.ro/platform-dev/profiling-server/index.html&usd=2&usg=ALhdy29I0XtTV5QIAIqrofhjTPPVHAKmwA) |
| Version | 1.0.0 |
| Responsible partner(s) | ENT |
| Related to task | T4.4 |

### Black box description

Profiling Server manages user profile information represented by user profile ontology, described in Profile ontology section within eWALL ontology and implemented within commons-data-model artefact, both described in eWALL deliverable D4.1.2 [1].

### Requirements

List of related requirements from D2.7 [5]:

* Gen\_001: Flexibility - ability to support a variety of market available or eWALL developed user and network devices
* Gen\_002: Scalability - ability to easily scale the eWALL platform to all envisioned use-cases
* Gen\_003: Traceability - ability to log and track the taken actions throughout the platform operation
* Gen\_004: Extensibility - ability to easily integrate novel devices in the platform (transparent protocol formats and protocol messages)
* Gen\_005: Reliability - ability to provide reliable communication within the platform and always on-time reaction
* Gen\_006: Compatibility - ability to integrate various information from various devices in a user transparent manner
* Gen\_018: Maintainability and configurability - ability to easily maintain and configure the system after deployment
* Spec\_023: Communication **–** ability to enable inter-component message-based (or event-based) and call-based communication between distributed components
* Gen\_007: Responsiveness - ability to dynamically react and/or reconfigure eWALL platform elements
* Gen\_008: User mobility input: ability to track the position and the movement of users in-house (important for various eWALL services)
* Spec\_018: Confidentiality: ability to maintain confidentially (the way in which the information disclosed or managed by the system is treated) of identifiable data, including controls on storage, handling, and sharing of data

### Features

User profile information is stored in central place, eWALL cloud database (MongoDB[[8]](#footnote-9)) and is being exposed by the Profiling Server via REST interface. Except the data contained within (user) profile ontology different user related measurements are also exposed. Alongside data retrieval minimal user information storing functionality is also exposed via REST interface as a start towards more profound provisioning.

Main currently implemented features are:

* Receiving requests on REST interface for storing, modifying, obtaining and deleting user profile information,
* Receiving requests on REST interface for storing, modifying, obtaining and deleting eWALL applications’ information,
* Receiving requests on REST interface for storing, modifying, obtaining and deleting eWALL services information,
* Receiving requests on REST interface for storing, modifying, obtaining and deleting user related measurements data (e.g. health, visual)
* Receiving requests on REST interface for storing, modifying, obtaining and deleting environment related measurements data (e.g. gases, humidity, illuminance, mattress pressure, temperature, movement)
* Receiving requests on REST interface for storing and deleting user credentials data needed for the authentication with the eWALL system. The passwords are hashed and stored to the separate part of the eWALL MongoDB database.
* Receiving requests on REST interface for checking if the given user is allowed/authenticated to access the eWALL system.
* Communicating with the eWALL cloud (MongoDB) database containing the data,
* Receiving requests on REST interface and storing of user profile information.

Features that are planned for implementation in the next period:

* More elaborate support for profile data (both storing and retrieving) in the sense making available different API calls etc.
* Connection with Data Manager component (described both this document and in D4.3.2 [3]) and clear separation of concerns.

## Login service

|  |  |
| --- | --- |
| Artefact: eWALL Platform Login | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/stable/ewall-platform-login/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/ewall-platform-login |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform$ewall-platform-login/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/ewall-platform-login/1.0.0/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform$ewall-platform-login/javadoc/ |
| Documentation (Swagger) | http://ewall.radio.pub.ro/platform-dev/ewall-platform-login/ |
| Maven artefact | 1.0.0/ |
| Responsible partner(s) | HP |
| Related to task | T4.3 |

### Black box description

The Login service allows clients to authenticate to the eWALL platform via credentials, and in case they are authenticated returns them a token, a self-contained data structure which describes the role and the permissions that a client has on the system. Such token must be transmitted by the client in every subsequent request in form of http header. Since the token represents the operations that the client is allowed to do on the system, it is also used by the applications to create/delete dynamically GUI elements. If the user profile is changed by an operator, e.g. adding or removing permissions for using an application, the token contents change accordingly. Based on the change, the GUI reacts by redefining the elements which allow to access applications.

### Requirements

List of related requirements from D2.7 [5]:

* Gen\_001: Flexibility – ability to support a variety of market available or eWALL developed user and network devices
* Gen\_002: Scalability – ability to easily scale the eWALL platform to all envisioned use-cases
* Gen\_003: Traceability - ability to log and track the taken actions throughout the platform operation
* Gen\_004: Extensibility – ability to easily integrate novel devices in the platform (transparent protocol formats and protocol messages)
* Gen\_006: Responsiveness – ability to dynamically react and/or reconfigure eWALL platform elements
* Gen\_007: Multiuser capability – ability to support multiple eWALL users with guaranteed profiling
* Gen\_011: Security – ability to secure the eWALL users’ data from obtrusive and accidental eavesdropping
* Gen\_015: Context information - ability to provide context information that is useful for services to adapt themselves according to the needs, preferences and situation of the user
* Gen\_016: Service orientation – ability of a system to ensure reusability and composability of services and service components
* Gen\_017: Semantic interoperability – ability to enable semantic interoperability between applications and services for ensuring the highest degree of decoupling (enables an open system and facilitates reuse of existing services and applications)
* Gen\_019: Multi-modal user interaction – ability to support multi-modal user interaction
* Spec\_009: Remote accessibility – ability to provide remote access to the eWALL platform
* Spec\_017: Identification, authentication and authorization – ability of system components to identify, authenticate and authorize an entity (human users and other system components) that wants to use them before allowing them access to resources

### Features

The eWALL Login service is the entry point for all the eWALL users to the eWALL platform. Every user which is entitled to access the platform is assigned a username and password. Before accessing an eWALL service, users must authenticate by providing their credentials toward the Login service. Then, the Login service checks for the validity of the credentials (existence of the user and matching with a hashed password). After this preliminary verification, the service retrieves the user profile from the Profiling Server and extracts from it the user role and the list of the applications which such user is authorized to access. This setting (definition of the set of applications per user) is made from the portal by an administrator when inserting a new user (or modifying an existing one) into the system. From the list of applications, the Login service retrieves the list of all the needed backend services. Finally, the Login service builds a “token”, a string containing a set of information which identify the end user and the set of services he/she can access. The token is compliant with the JSON Web Token (JWT) specification[[9]](#footnote-10), which defines a standard structure for representing claims to be transferred between two parties. The token contains information such as the username, the expiration date (the token has limited lifetime), the set of services accessible, and the user role. After creation, the token is encoded and signed using the content of the token itself and a secret key, which is known only by the login service and by the backend services which builds up the eWALL platform. The token is then sent back to the client over https. In all following requests towards every eWALL service endpoint, the client must provide in an http header the token obtained after login. The service providers will extract the token from the request and check it for validation before serving the request. The signing mechanism grants that the token, if modified by a client (to get more permissions than stated by the claims), will be considered invalid by the service providers. In fact, the change will be immediately detected, due to the mismatch between the signature in the token and the signature built for verification by the server by applying the secret key to the token contents received by the client.

Since the token has an expiration time, the Login service provides to clients an endpoint which allows them to renew an existing token before its expiration.

In summary, the Login service provides two REST endpoints for:

* Authenticating a client, taking as input username and password, and returning a token in case of successful authentication
* Renewing a valid token, taking as input a valid token and returning a new token with updated contents and a new expiration date

Thanks to the renewal mechanism, a client can maintain the token live for the time required by requesting a renewal before the expiration time. Moreover, since the token is dynamically generated based on the user profile, if a modification in the profile happens, at the next renewal the client can detect the change and react accordingly, e.g. by modifying an application GUI to reflect the current set of applications that a user can access.

## Cross-component functionalities

|  |  |
| --- | --- |
| Artefact: eWALL Common Libraries | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/stable/ewall-common-libs/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/ewall-common-libs/ |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-stable/eu.ewall.platform$ewall-common-libs/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/application/ewall-common-libs/1.0.0/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/jenkins/job/wp4-stable/eu.ewall.platform$ewall-common-libs/javadoc/ |
| Maven artefact | 1.0.0/ |
| Responsible partner(s) | HP |

### Black box description

The eWALL REST services are provisioned by software components which share the same high-level structure:

* They are based on java servlets that:
  + are packaged in wars
  + leverage the Spring framework, and in particular the Spring boot features
  + load configuration properties from files
* They expose REST endpoints which must be invoked by other components, hence need to provide clear API definition
* They must share the same authentication/authorization mechanism, which allows to detect if a request can be served depending on the requestor’s role and permissions
* They should share the same configuration and documentation generation mechanisms, so that they are easily maintainable and have a uniform documentation

Therefore, a set of cross-component functionalities have been implemented and released in form of libraries, packaged into a “ewall-common-libs” artefact, to be included in the eWALL components.

### Features

The “ewall-common-libs” artefact provides the following features:

* ***Component configuration***: a uniform way to configure components and services, based on a Spring framework ”@PropertySource” annotation which allows to load automatically and transparently a set of property files, without the need for a developer to specify anything else in the code.
* ***Token-based authentication and authorization verification***: Automatically adds a servlet filter to the service, so that every request received is checked against the auth token to allow/deny access. A developer can use this feature by just setting a specific Spring ”@Import” annotation for a JwtAuthConfig class of the library
* ***Token-based requests:*** Provides an *ewallRestClient* object to make authenticated requests towards internal components (micro-services interaction, e.g. to the Profiling Server) in a transparent way for the developers. A developer can use this feature by just setting a specific Spring ”@Import” annotation for a CommonConfig class of the library
* ***Memory object caching system:*** enable caching of http requests and related response on the eWALL Memcached server (cf. [3]) via annotation. Requires only to set a specific Spring ”@Import” annotation for a CacheConfig class of the library, on top of the methods which implement REST endpoints. All the communication with the Memcached server and the related connection management and data storage are transparent to the developers.
* ***Automatic documentation generation*** for REST API: adds configuration for the Swagger framework (a tool for generating documentation on REST endpoints and to test them interactively, cf. Section 4.2.3.3). By just adding a Spring ”@Import” annotation for a SwaggerConfig class of the library, web documentation for all the endpoints of the service are automatically generated at build time and deployed together with the service

## Artefacts developed within other WP4 tasks

### Data Manager (T4.3)

The Data Management (DM) is responsible for basic data processing, format exchange, and input validation of received data. It uses Cloud Infrastructure Layer to seamlessly access physical storage facilities (databases) for data persistency. More details about implementation of this artefact can be found in *D4.3.3 Cloud middleware services for eWALL* [3].

|  |  |
| --- | --- |
| Artefact: Data Manager (DM) | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/stable/data-manager/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/data-manager |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.middleware.datamanager$data-manager/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/middleware/datamanager/data-manager/1.0.0/ |
| Maven artefact | 1.0.0 |
| Responsible partner(s) | ENT, UPB |
| Related to task | T4.3 |

### Cloud Gateway (T4.3)

The Cloud Gateway (CGw) is responsible for interconnecting eWALL Cloud with Sensing Environments (Home and Mobile). It is responsible for receiving processed and indexed measured sensor data, to send control and configuration data to Sensing Environments and for bidirectional application data. It directly communicates with Remote Proxy and supports both message based pull and push communication. More details about implementation of this artefact can be found in *D4.3.2. Cloud middleware services for eWALL* [3].

|  |  |
| --- | --- |
| Artefact: Cloud Gateway (CGw) | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/stable/cloud-gateway/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/cloud-gateway |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.middleware.cloudgateway$cloud-gateway/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/middleware/cloudgateway/cloud-gateway/1.0.0/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.middleware.cloudgateway$cloud-gateway/javadoc/ |
| Documentation (Swagger) | http://ewall.radio.pub.ro/platform-dev/cloud-gateway/index.html |
| Version | 1.0.0 |
| Responsible partner(s) | ENT, UPB |
| Related to task | T4.3 |

### Intelligent Decision Support System (T4.2)

The Intelligent Decision Support System (IDDS) is responsible for decision making activities in eWALL system. It uses inference to discover new relationships, automatically analyses the content of the data, and manages knowledge (i.e. perform reasoning based on corresponding or appropriate rules). More details about implementation of this artefact can be found in *D4.2.3 Intelligent support system for eWALL* [2].

|  |  |
| --- | --- |
| Artefact: Intelligent Decision Support System (IDSS) | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/master/idss-core/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/idss-core |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.idss$idss-core/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/idss/idss-core/1.0.0/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform.idss$idss-core/javadoc/ |
| Version | 1.0.0 |
| Responsible partner(s) | RRD, UOZ |
| Related to task | T4.2 |

### Commons data model (T4.1)

The commons data model artefact represents data model for eWALL (cloud) platform. It is used by Profiling Server, Data Manager, Cloud Gateway and all other software artefacts (also service bricks from WP5) that store, read and use eWALL data. More details about implementation of this artefact can be found in *D4.1.2 Semantic model of eWALL middleware services* [1].

|  |  |
| --- | --- |
| Artefact: Commons data model | |
| Maven artefact | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/blob/stable/commons-data-model/pom.xml |
| Source (Git address) | http://serv2.radio.pub.ro/gitlab/wp4/wp4project/tree/stable/commons-data-model |
| Build (Jenkins) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform$commons-data-model/ |
| Binaries (Nexus) | http://ewall.radio.pub.ro/nexus/content/repositories/releases/eu/ewall/platform/commons-data-model/1.0.0/ |
| Documentation (Javadoc) | http://ewall.radio.pub.ro/jenkins/job/wp4-release/eu.ewall.platform$commons-data-model/javadoc/ |
| Version | 1.0.0 |
| Responsible partner(s) | ENT |
| Related to task | T4.1 |

# Conclusions

This document provides a report about software developed within the Task 4.4 *“eWALL platform implementation”*. This deliverable is primarily a software deliverable, but it also contains a brief description of the development process in the eWALL project ranging from the very inception of the process, selection of existing technologies to the component development and implementation, unit testing and deployment.

All planned features of software artefacts, namely of Remote Proxy, Notification Manager, eWALL Portal, Login service, Cross component functionalities and Profiling Server, have been successfully implemented, tested in integrated environment and used in production environment during running of the pilots. The main implemented features include reliable and secure communication and data transfer between Sensing Environment with eWALL Cloud, data handling, provision and administration, second version of notification management, and improved user profiling support.

# Bibliography

1. eWALL project deliverable *D4.1.3: Semantic model of eWALL middleware services*, January 2016
2. eWALL project deliverable *D4.2.3: Intelligent support system for eWALL*, October 2016
3. eWALL project deliverable *D4.3.3: Cloud middleware services for eWALL*, January 2016
4. eWALL project deliverable *D4.4.2 eWALL Platform,* October 2014
5. eWALL project deliverable *D2.7 Final user and system requirements and architecture,* February 2015
6. Spring Tool Suite (STS), online at <http://spring.io/tools/sts>
7. Git, online at <http://git-scm.com/>
8. GitLab - Open source software to collaborate on code, online at <https://about.gitlab.com/>
9. Maven - Welcome to Apache Maven, online at <http://maven.apache.org/>
10. universAAL project deliverable *D2.3-D: universAAL integration and testing strategy and issue tracker*, October 2012
11. Nexus - Sonatype.com, online at <http://www.sonatype.com/nexus>
12. Jenkins - an extendable open source continuous integration server, online at <http://jenkins-ci.org/>
13. Homepage of Zabbix : An Enterprise-Class Open Source Distributed Monitoring Solution, online at <http://www.zabbix.com/>
14. eWALL project deliverable *D2.7 Final user and system requirements and architecture, January 2015*
15. eWALL project deliverable *D2.4 Ethics, Privacy and Security,* November 2014

# Abbreviations

AOP Aspect-oriented Programming

AMQP Advance Message Queuing Protocol

CGw Cloud Gateway

CIL Cloud Infrastructure Layer

DM Data Manager

IaaS Infrastructure as a Service

IDSS Intelligent Decision Support System

IoC Inversion of Control

LDM Local Data Manager

JEE Java Enterprise Edition

JWT Json Web Token

NM Notification Manager

POM Project Object Model

PS Profiling Server

REST Representational State Transfer

RP Remote Proxy

SSL Secure Socket Layer

STS Spring Tool Suite

UI User Interface

UML Unified Modeling Language

# Appendix A: eWALL Platform RESTful interfaces

This section provides summary list of main eWALL Platform RESTful interfaces related to:

* Interface between Sensing Environments and eWALL Cloud (Remote Proxy - Cloud Gateway) (I1)
* System provisioning interface (Cloud Gateway – eWALL Portal) (I2)
* Notification interface (Notification Manager – other eWALL components) (I4b)
* Profiling interface (Profiling Server – other eWALL components) (I4a)

The complete and interactive documentation of the last version of all eWALL Platform RESTful APIs is accessible via Swagger tool (described Section 4.2.3.3) with link is provided in each component ID card.

## Interface between Sensing Environments and eWALL Cloud (I1)

Initial consideration[[10]](#footnote-11):

{HOST\_URI} = i.e. serv2.radio.pub.ro, or ewallproject.eu

<ROOT\_URI>= https://{HOST}/cloud-gateway

<SENSING\_ENV\_URI>=<ROOT\_URI>/sensingenvironments/{Sensing\_Environment\_ID}

Registering/deregistering Sensing Environment:

|  |  |
| --- | --- |
| Method | POST |
| Url | < SENSING\_ENV\_URI>/register, < SENSING\_ENV\_URI>/deregister, |
| Parameters | ewallsystem\_id, sensingenvironment\_id , |
| Request body | Point of Contact JSON object (T4.1) |
| Response | 200 OK, Errors 40x |

Updating, deleting Point of Contact (PoC) of Sensing Environment:

|  |  |
| --- | --- |
| Method | POST, DELETE |
| Url | < SENSING\_ENV\_URI>/poc |
| Parameters | ewallsystem\_id, sensingenvironment\_id |
| Request body | Point of Contact JSON object |
| Response | 200 OK, Errors 40x |

Remote getting and setting configuration properties of Remote Proxy:

|  |  |
| --- | --- |
| Method | GET, POST |
| Url | < SENSING\_ENV\_URI>/configuration |
| Parameters | ewallsystem\_id, sensingenvironment\_id |
| Response | 200 OK, Errors; 40x |

Push environmental data:

|  |  |
| --- | --- |
| Method | POST |
| Url | <SENSING\_ENV\_URI>/devices/{device\_id}/environmental/add |
| Request body | single or array of JSON measurement objects (T4.1) |
| Responses | Success; 200 Ok, Errors;40x |

Push activity related data:

|  |  |
| --- | --- |
| Method | POST |
| Url | <SENSING\_ENV\_URI>/devices/{device\_id}/activity/add |
| Request body | single or array of JSON measurement objects (T4.1) |
| Responses | Success; 200 Ok, Errors;40x |

Push vital sensing data:

|  |  |
| --- | --- |
| Method | POST |
| Url | <SENSING\_ENV\_URI>/devices/{device\_id}/vitals/add |
| Request body | single or array of JSON measurement objects (T4.1) |
| Responses | Success; 200 Ok, Errors;40x |

Push visual sensing data:

|  |  |
| --- | --- |
| Method | POST |
| Url | <SENSING\_ENV\_URI>/devices/{device\_id}/visual/add |
| Request body | single or array of JSON measurement objects (T4.1) |
| Responses | Success; 200 Ok, Errors;40x |

Push mattress pressure sensing data:

|  |  |
| --- | --- |
| Method | POST |
| Url | <SENSING\_ENV\_URI>/devices/{device\_id}/furniture/add |
| Request body | single or array of JSON measurement objects (T4.1) |
| Responses | Success; 200 Ok, Errors;40x |

Push speaker sensing data:

|  |  |
| --- | --- |
| Method | POST |
| Url | <SENSING\_ENV\_URI>/devices/{device\_id}/speaker/add |
| Request body | single or array of JSON measurement objects (T4.1) |
| Responses | Success; 200 Ok, Errors;40x |

Push appliance power sensing data:

|  |  |
| --- | --- |
| Method | POST |
| Url | <SENSING\_ENV\_URI>/devices/{device\_id}/power/add |
| Request body | single or array of JSON measurement objects (T4.1) |
| Responses | Success; 200 Ok, Errors;40x |

Sending notification from local environment:

|  |  |
| --- | --- |
| Method | POST |
| Url | <SENSING\_ENV\_URI>/notifications  <SENSING\_ENV\_URI>/caregiverNotifications |
| Request body | JSON notification objects (T4.1) |
| Responses | Success; 200 Ok, Errors;40x |

Sending actuator control command to sensing environment:

|  |  |
| --- | --- |
| Method | PUT |
| Url | <ROOT\_URI>/users/{user\_name}/actuators/{actuator\_id}/sendCommand  <ROOT\_URI>/users/{user\_name}/actuators/lights/setColor  <ROOT\_URI>/users/{user\_name}/actuators/sendCommand  <ROOT\_URI>/users/{user\_name}/actuators/sendCommandParameters |
| Request body | JSON actuator command objects (T4.1) |
| Responses | Success; 200 Ok, Errors;40x |

## Provisioning interface (I2)

Initial consideration:

<HOST\_URI> = i.e. http://ewall.radio.pub.ro/platform-dev, or ewallproject.eu

<ROOT\_URI>= <HOST\_URI>/cloud-gateway

<SENSING\_ENV\_URI>=<ROOT\_URI>/sensingenvironments/{Sensing\_Environment\_ID}

Getting all provisioned Sensing Environments in certain eWALL System

|  |  |
| --- | --- |
| Method | GET |
| Url | <ROOT\_URI>/sensingenvironments/ |
| Responses | Success; 200 Ok, Errors; 40x |
| Response body | list of Sensing Environments JSON objects (T4.1) |

Getting info about certain Sensing Environments

|  |  |
| --- | --- |
| Method | GET |
| Url | <ROOT\_URI>/sensingenvironments/{sensingenvironment\_id} |
| Parameters | sensingenvironment\_id |
| Responses | Success; 200 Ok, Errors; 40x |
| Response body | Sensing Environment JSON object (T4.1) |

Adding/updating information about certain Sensing Environment

|  |  |
| --- | --- |
| Method | PUT, POST |
| Url | <ROOT\_URI>/sensingenvironments/{sensingenvironment\_id} |
| Parameters | sensingenvironment\_id |
| Request body | Sensing Environment JSON object (T4.1) |
| Responses | Success; 200 Ok, 201 Created, Errors; 40x |

Getting all devices in Sensing Environment

|  |  |
| --- | --- |
| Method | GET |
| Url | <SENSING\_ENV\_URI>/devices |
| Responses | Success; 200 Ok, Errors; 404 Not found |
| Response body | list of device JSON objects (T4.1) |

Get device description

|  |  |
| --- | --- |
| Method | GET |
| Url | <SENSING\_ENV\_URI>/devices/{device\_Id} |
| Parameters | sensingenvironment\_id, device\_id, |
| Responses | Success; 200 Ok, Errors; 404 Not found |
| Response body | device JSON object (T4.1) |

Adding/updating device info

|  |  |
| --- | --- |
| Method | POST, PUT |
| Url | <SENSING\_ENV\_URI>/devices/{device\_ID} |
| Parameters | sensingenvironment\_id, device\_id, |
| Request body | device JSON object (T4.1) |
| Responses | Success; 201 Created, 201 Sensor Updated, Errors; 400 Invalid device ID, 400 Device exists, 404 Not Found |

Removing a device from sensing environment

|  |  |
| --- | --- |
| Method | DELETE |
| Url | <SENSING\_ENV\_URI>/devices/{device\_ID} |
| Responses | Success; 204 Deleted, Errors; 404 Not Found, 400 Bad Request |

Get actuator details

|  |  |
| --- | --- |
| Method | GET |
| Url | <ROOT\_URI>/users/{user\_name}/actuators |
| Parameters | user\_name, type, room |
| Responses | Success; 200 Ok, Errors; 404 Not found |
| Response body | List of actuators JSON objects (T4.1) |

## Notification interface (I4b)

Initial consideration:

<HOST\_URI> = i.e. http://ewall.radio.pub.ro/platform-dev, or ewallproject.eu

 Notification Manager – IDSS interface (primary user related notifications):

|  |  |
| --- | --- |
| Method | POST |
| Url | <HOST\_URI>/Notif?user={user\_name}&&date={date}&time={time}&type={type}&title={title}&content={content}&prior={priority}&source={source} |
| Parameters | **user** – user name (e.g. “Bob”), **date** – Date of the trigger (e.g. "08-05-2014"), **time** – Time of the trigger (e.g. "12.30.21"), **type** – The design type of the notification, in regards to its representation to the UI (e.g. "two-buttons"), **title**- The title of the notification shown on the UI (e.g. " Congratulations!"), **content** – The information content of the notification shown on the UI (a JSON formatted information), **prior** – the priority value for the given type of notification. This facilitates the possibility to sort the notifications based on priority (left for future use), **source –** the url address of the component that sends the notification |
| Response | 200 OK, Errors; 40x |

Notification Manager – IDSS interface (caregiver related notifications):

|  |  |
| --- | --- |
| Method | POST |
| Url | <HOST\_URI>/{caregiverUsername}/notifications?user={user\_name}&&date={date}&time={time}&type={type}&title={title}&content={content}&prior={priority}&source={source} |
| Parameters | **caregiverUsername –** caregiver user name, **user** – user name (e.g. “Bob”), **date** – Date of the trigger (e.g. "08-05-2014"), **time** – Time of the trigger (e.g. "12.30.21"), **type** – The design type of the notification, in regards to its representation to the UI (e.g. "two-buttons"), **title**- The title of the notification shown on the UI (e.g. " Congratulations!"), **content** – The information content of the notification shown on the UI (a JSON formatted information), **prior** – the priority value for the given type of notification. This facilitates the possibility to sort the notifications based on priority (left for future use), **source –** the url address of the component that sends the notification |
| Response | 200 OK, Errors; 40x |

Notification Manager – Caregiver Application interface:

|  |  |
| --- | --- |
| Method | GET |
| Url | <HOST\_URI>/{caregiverUsername}/notifications?from={from}&to={to} |
| Parameters | **caregiverUsername –** caregiver user name, **user** – user name (e.g. “Bob”), **from** - timestamp in milliseconds, **to** - timestamp in milliseconds |
| Response | 200 OK, Errors; 40x |

Notification Manager - UI interface:

|  |  |
| --- | --- |
| Method | GET |
| Url | <HOST\_URI>/ getNot?user={user}&userID={ user }&device={device} |
| Parameters | **user** – user name (e.g. “Bob”), , **device** - Type of UI device (e.g. main screen, mobile phone); values {mobile phone=1, main screen=0} |
| Response | 200 OK, Errors; 40x |

Notification Manager - UI interface (feedbacks):

|  |  |
| --- | --- |
| Method | POST |
| Url | <HOST\_URI>/userFeedback?notID={notID}&feedback={feedback} |
| Parameters | **notID** – the notification ID , **feedback** – The feedback content |
| Response | 200 OK, Errors; 40x |

NM -> UI string query:

|  |  |
| --- | --- |
| Method | POST |
| Url | <HOST\_URI>/Position?user={user}& notID ={ notID }&position={position}&lang={lang}&date={date}&time={time}&type={type}&title={title}&content={content} |
| Parameters | **user** – user name (e.g. “Bob”), notID – unique notification ID based on the mongodb object ID format (e.g. “507f1f77bcf86cd799439011”), **position** - Location of the user (e.g. outdoor, indoor); values {outdoor=1, indoor=0}, lang - Language type (parameter), **date**, **time**, **type** – The design type of the notification, in regards to its representation to the UI (e.g. "two-buttons"), **title** - The title of the notification shown on the UI (e.g. " Congratulations!"), **content** |
| Response | 200 OK, Errors; 40x |

## Profiling interface (I4a)

Initial consideration:

<HOST\_URI> = i.e. http://ewall.radio.pub.ro/platform-dev, or ewallproject.eu

Get user profile data (all users or by username):

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username} |
| Parameters | username (optional), if not present, list of all users is returned |
| Response | 200 OK with, Errors; 40x |
| Response body | array list or single JSON user data (T4.1) |

Update, delete user profile data:

|  |  |
| --- | --- |
| Method | POST, DELETE |
| Url | < HOST\_URI>/profiling-server/users/{username} |
| Parameters | username |
| Response | 200 OK with, Errors; 40x |
| Request body | User profile JSON object (T4.1) |

Setting user credentials:

|  |  |
| --- | --- |
| Method | POST |
| Url | < HOST\_URI>/profiling-server/users/{username}/credentials |
| Parameters | username |
| Response | 200 OK with, Errors; 40x |
| Request body | Password String object |

Get application data (all, or by application name):

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/applications/{applicationName} |
| Parameters | applicationName (optional), if not present, data of all applications is returned |
| Response | 200 OK with, Errors; 40x |
| Response body | array list or single JSON application data (T4.1) |

Update, delete application data:

|  |  |
| --- | --- |
| Method | POST, DELETE |
| Url | < HOST\_URI>/profiling-server/applications/{applicationName} |
| Parameters | applicationName |
| Response | 200 OK with, Errors; 40x |
| Request body | Application JSON object (T4.1) |

Get services data (all, or by service name):

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/services/{serviceName} |
| Parameters | serviceName (optional), if not present, data of all services is returned |
| Response | 200 OK with, Errors; 40x |
| Response body | array list or single JSON service data (T4.1) |

Update, delete services data:

|  |  |
| --- | --- |
| Method | POST, DELETE |
| Url | < HOST\_URI>/profiling-server/regions/{serviceName} |
| Parameters | serviceName |
| Response | 200 OK with, Errors; 40x |
| Request body | Service JSON object (T4.1) |

Get region data (all, or by region name):

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/ regions /{regionName} |
| Parameters | regionName (optional), if not present, data for all regions is returned |
| Response | 200 OK with, Errors; 40x |
| Response body | array list or single JSON region data (T4.1) |

Update, delete region data:

|  |  |
| --- | --- |
| Method | POST, DELETE |
| Url | < HOST\_URI>/profiling-server/regions /{regionName} |
| Parameters | regionName |
| Response | 200 OK with, Errors; 40x |
| Request body | Region JSON object (T4.1) |

Get accelerometer data in certain time interval:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/accelerometer/timestamp?from={timestamp}&to={timestamp} |
| Parameters | username, timestamp |
| Response | 200 OK with, Errors; 40x |
| Response body | array list of JSON accelerometer data (T4.1) |

Get activity data from accelerometer in certain time interval:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/activity?from={from}&to={to} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds) |
| Response | 200 OK with, Errors; 40x |
| Response body | array list of JSON activity data (T4.1) |

Get user heart rate measurements in certain time interval:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/vitals/heartrate?from={from}&to={to} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds) |
| Response | 200 OK with, Errors; 40x |
| Response body | array list of JSON heart rate data (T4.1) |

Get user oxygen saturation measurements in certain time interval:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/vitals/oxygensaturation?from={from}&to={to} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds) |
| Response | 200 OK with, Errors; 40x |
| Response body | array list of JSON oxygen saturation data (T4.1) |

Get user blood pressure measurements in certain time interval:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/vitals/bloodpressure?from={from}&to={to} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds) |
| Response | 200 OK with, Errors; 40x |
| Response body | array list of JSON blood pressure data (T4.1) |

Get visual sensing data in certain time interval:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/visual?from={from}&to={to} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds) |
| Response | 200 OK with, Errors; 40x |
| Response body | array list of JSON visaul data (T4.1) |

Get mattress pressure data in certain time interval per room:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/environment/ mattress\_pressure?from={from}&to={to}&room={roomName} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds), roomName (optional) |
| Response | 200 OK, Errors; 40x |
| Response body | JSON mattress pressure data (T4.1) |

Get temperature data in certain time interval per room:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/environment/ temperature?from={from}&to={to}&room={roomName} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds), roomName (optional) |
| Response | 200 OK, Errors; 40x |
| Response body | JSON temperature data (T4.1) |

Get humidity data in certain time interval per room:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/environment/ humidity?from={from}&to={to}&room={roomName} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds), roomName (optional) |
| Response | 200 OK, Errors; 40x |
| Response body | JSON humidity data (T4.1) |

Get illuminance data in certain time interval per room:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/environment/ illuminance?from={from}&to={to}&room={roomName} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds), roomName (optional) |
| Response | 200 OK, Errors; 40x |
| Response body | JSON illuminance data (T4.1) |

Get movement data in certain time interval per room:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/environment/ movement?from={from}&to={to}&room={roomName} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds), roomName (optional) |
| Response | 200 OK, Errors; 40x |
| Response body | JSON movement data (T4.1) |

Get door status data in certain time interval per room:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/environment/ doors?from={from}&to={to}&room={roomName} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds), roomName (optional) |
| Response | 200 OK, Errors; 40x |
| Response body | JSON door status data (T4.1) |

Get gases level data in certain time interval per room:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/environment/ gases/(lpg|co|no)?from={from}&to={to}&room={roomName} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds), roomName (optional) |
| Response | 200 OK, Errors; 40x |
| Response body | JSON gases data (T4.1) |

Get speaker sensing data in certain time interval:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/speaker?from={from}&to={to}&room={roomName} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds) |
| Response | 200 OK, Errors; 40x |
| Response body | JSON speaker sensing data (T4.1) |

Get appliances power sensing data in certain time interval:

|  |  |
| --- | --- |
| Method | GET |
| Url | < HOST\_URI>/profiling-server/users/{username}/appliances/status?from={from}&to={to}&room={roomName}&name ={applianceName}&type={applianceType} |
| Parameters | username, from (timestamp in milliseconds), to (timestamp in milliseconds), roomName (optional), applianceName (optional), applianceType (optional) |
| Response | 200 OK, Errors; 40x |
| Response body | JSON appliance sensing data (T4.1) |

# Appendix B: eWALL Portal User manual

Note: Following figures are screenshots from the development environment.

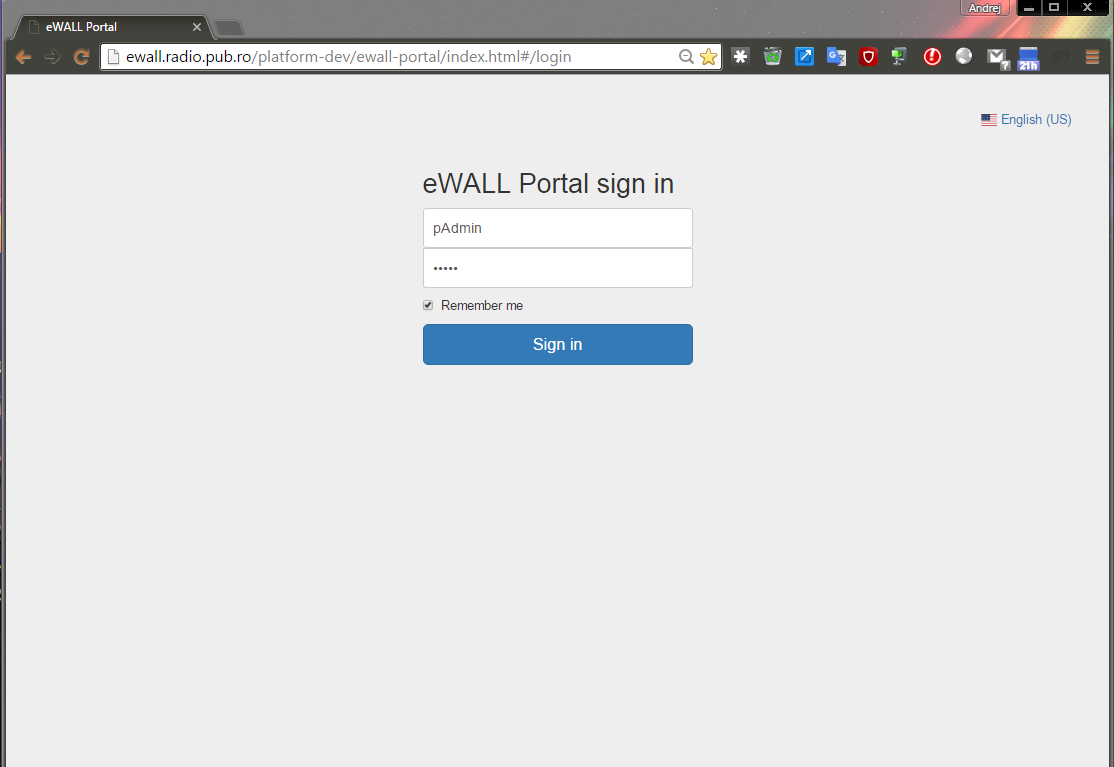


Figure 12 Screenshot of the eWall Portal login screen

Figure 12 shows login screen of eWall Portal application. It is used for entering user credentials (username and password). There is option for remembering user (“Remember me” checkbox). Button “Sign in” signs in the user if provided credentials are matched with data stored in database.

In the upper-right corner is button with country flag for setting application localization. Available languages are: Bulgarian, English (US), Macedonian, Danish, Croatian, Dutch, Serbian, German (Austria), Italian and Romanian.

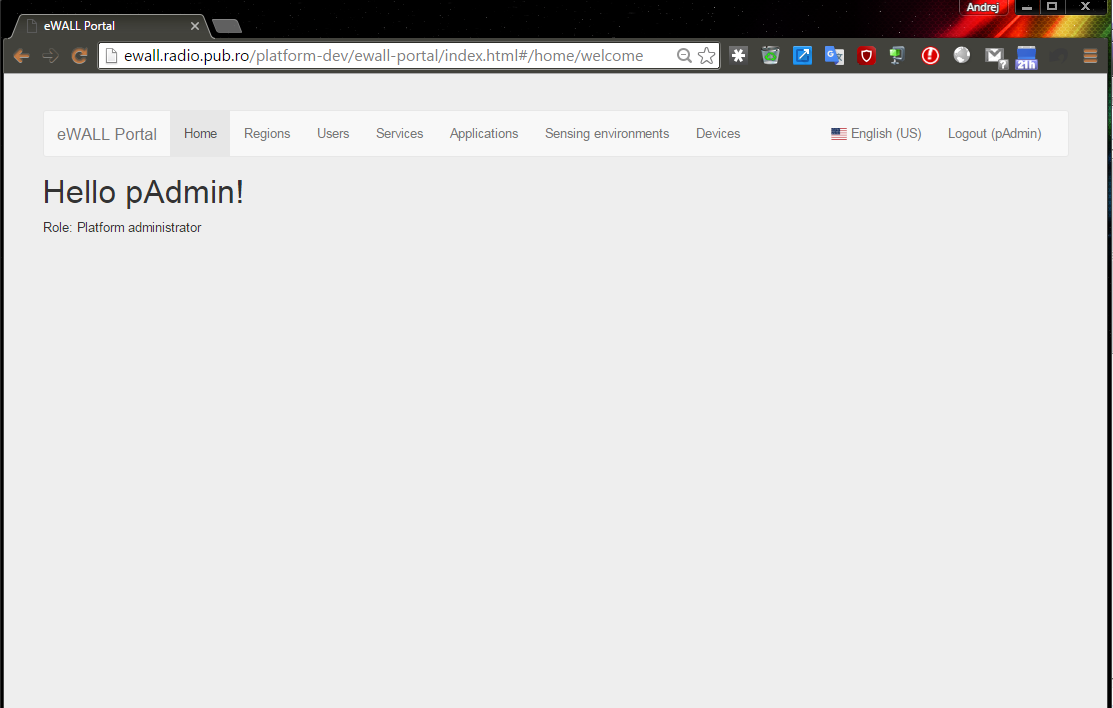


Figure 13 Screenshot of the home screen

Figure 13 shows home screen of eWall application. There are greetings for user and user role is marked. On the top of the page is navigation bar with tabs:

* Home (open in this figure),
* Regions,
* Users,
* Services,
* Applications,
* Sensing environments and
* Devices.

Every tab, except Home, displays data stored in database associated with that tab. Those tabs are going to be explained in more details in the following figures.

On the right of the navigation bar is language button explained in Figure 12 and button for logging out user (username of logged in user is written inside parenthesis of Logout label).

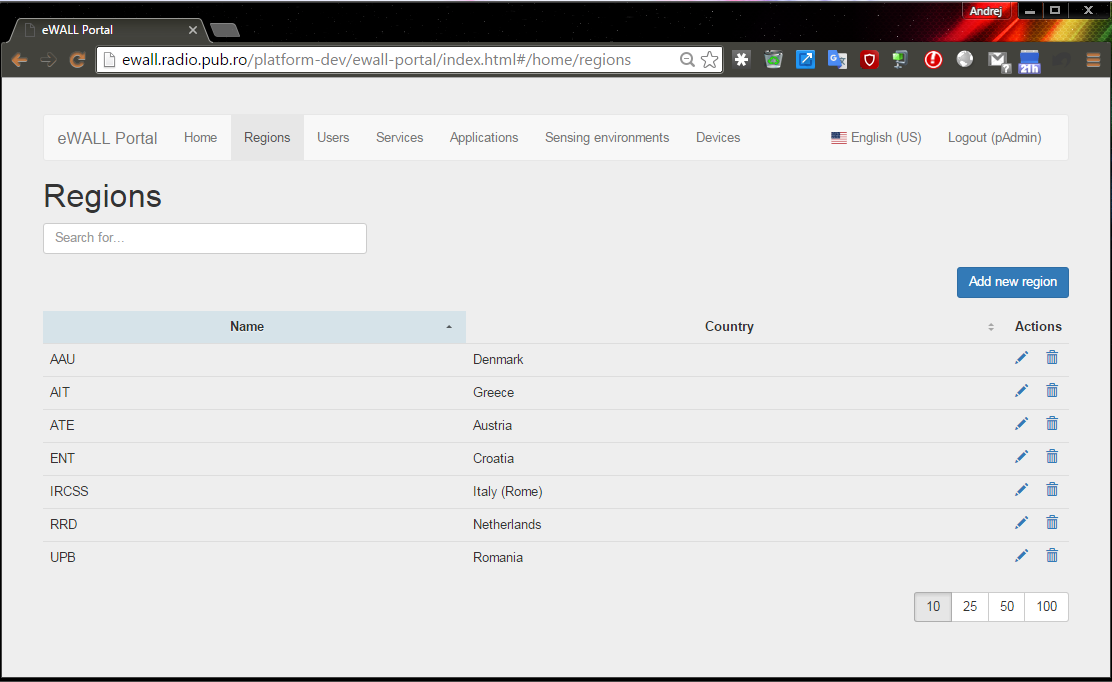


Figure 14 Screenshot of the “Regions” tab

Figure 14 shows open “Regions” tab which contains stored regions inside eWall system database which are handled inside *profiling-server* platform application.

Regions are shown in the table. Columns “Name” and “Country” show homonymous properties of regions. Column “Actions” has buttons for editing region (labeled with pencil icon) and for deleting region (labeled with trash icon). Click on column captions (Name and Country) sort regions by chosen property (column caption). Button for editing region opens new page with region properties which can be modified. Button for deleting region opens confirmation pop up dialog.

In the lower-right corner is pagination bar used for setting how many rows are going to be shown in one page of regions table (default option is 10 rows). If there is more rows (regions) than set in pagination bar, navigation bar will show up in the lower-right corner for navigating through pages. Just below “Regions” caption is placed textbox for filtering regions by values of all region properties shown in this table. There is also blue button “Add new region” for adding new region – page which opens after pressing that button is the same as for editing existing region.

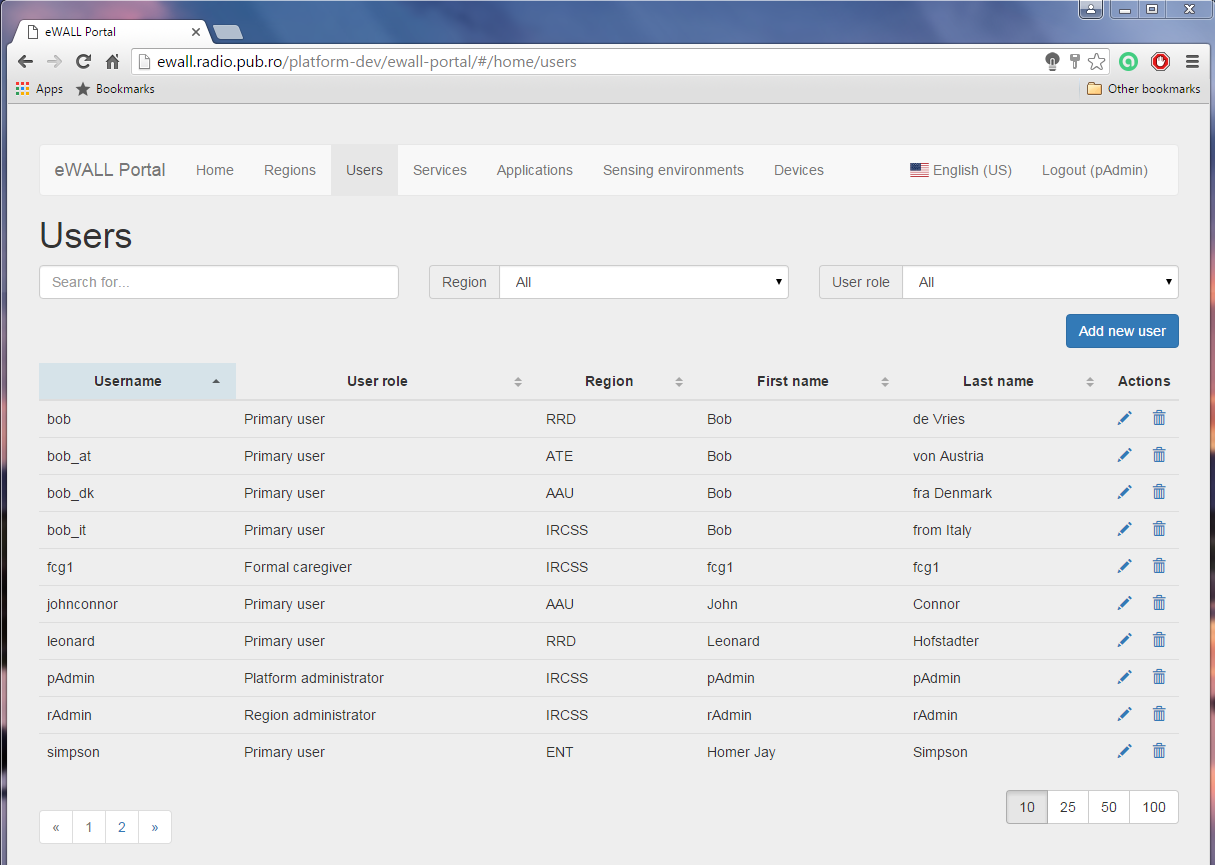


Figure 15 Screenshot of the "Users" tab

Figure 15 shows open “Users” tab which contains stored users inside eWall system database which are handled inside *profiling-server* platform component (described previously in section 6.5).

Existing users are shown in the table. Columns “Username”, “User role”, “Region”, “First name” and “Last name” show homonymous properties of users. Column “Actions” has buttons for editing user (labeled with pencil icon) and for deleting user (labeled with trash icon). Click on column captions (Username, User role, Region…) sort users by chosen property (column caption). Button for editing user opens new page with user properties which can be modified. Button for deleting user opens confirmation pop up dialog.

On the bottom of page are pagination and navigation bars explained in Figure 14. Just below “Users” caption is placed textbox for filtering users by values of all user properties shown in this table. Blue button “Add new user” is used for adding new user – page which opens after pressing that button is the same as for editing existing user (explained in Figure 16 and Figure 17).

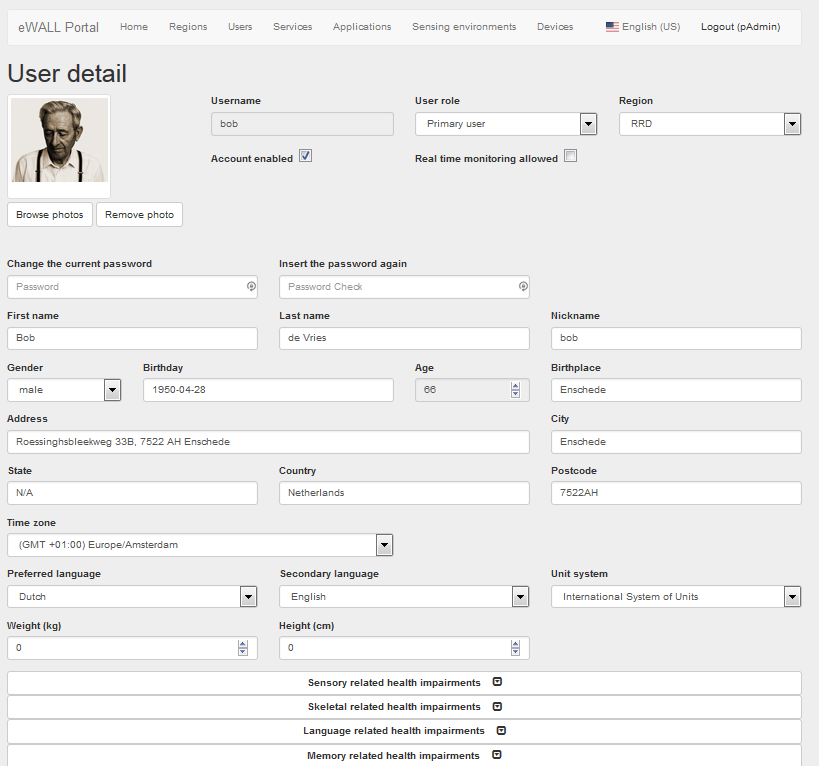


Figure 16 Screenshot of the (top part of) “User details” edit page for selected test user Simpson

Figure 16 shows top part of page for adding or editing users.

On this page are shown values of user properties which can be edited, except username. For user properties view are used common form input types: textboxes, checkboxes, drop-down lists, datepicker and listboxes. Just below “User detail” caption is placed control for editing user profile photo. This control has main area where is selected photo previewed and buttons for changing photo (labeled “Browse photos”) and removing photo (labeled “Remove photo”).

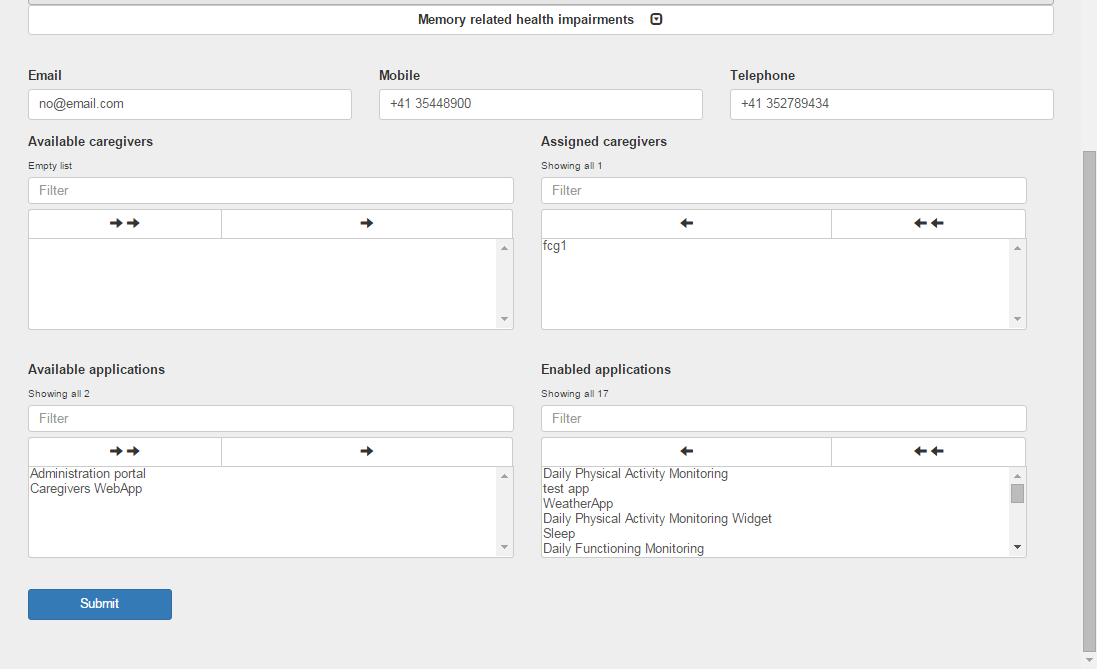


Figure 17 Screenshot of the (bottom part of) “User details” edit page for selected test user Simpson

Figure 17 shows bottom part of page for adding or editing users which is extension to top part of page described in Figure 16.

There are used listboxes for setting caregivers and applications associated to chosen user. These listboxes have auxiliary arrows for moving items from one box to another. On the bottom of this page is “Submit” button for submitting created or modified data.

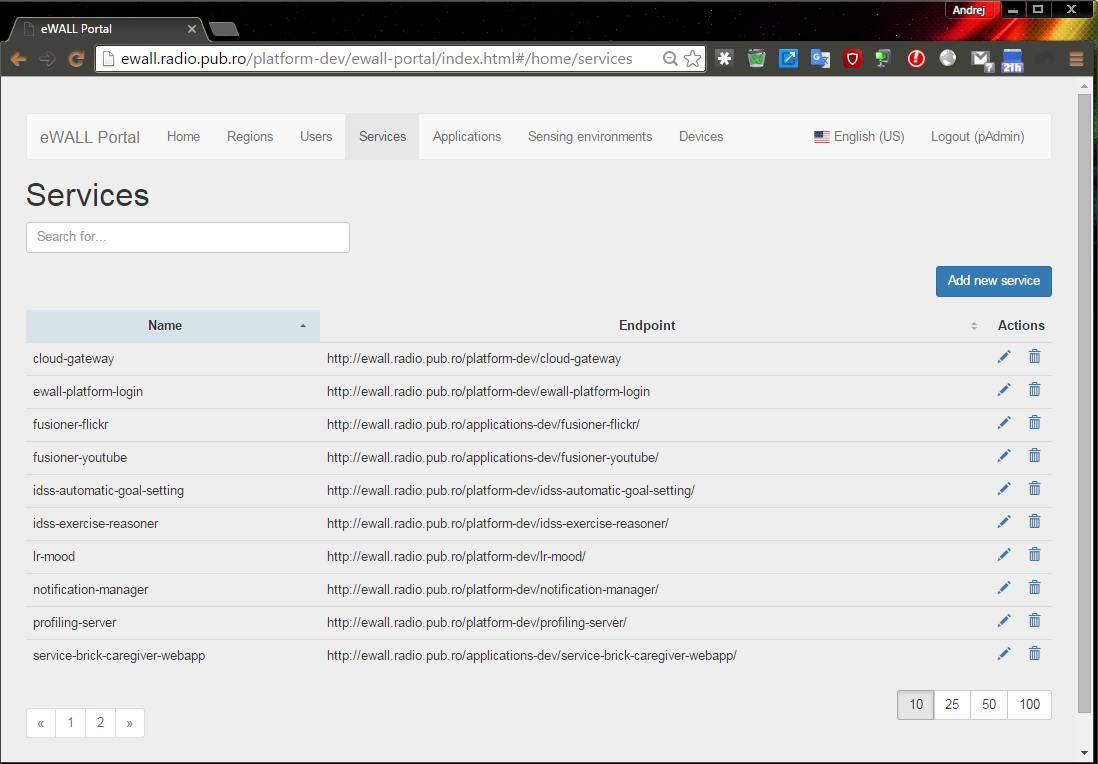


Figure 18 Screenshot of the “Services” tab

Figure 18 shows open “Services” tab which contains stored services inside eWall system database which are handled inside *profiling-server* platform application.

Services are shown in the table. Columns “Name” and “Endpoint” show homonymous properties of services. Column “Actions” has buttons for editing service (labeled with pencil icon) and for deleting service (labeled with trash icon). Click on column captions (Name and Endpoint) sort services by chosen property (column caption). Button for editing region opens new page with service properties which can be modified. Button for deleting service opens confirmation pop up dialog.

On the bottom of page are pagination and navigation bars explained in Figure 14. Just below “Services” caption is placed textbox for filtering services by values of all service properties shown in this table. Blue button “Add new service” is used for adding new service – page which opens after pressing that button is the same as for editing existing service.

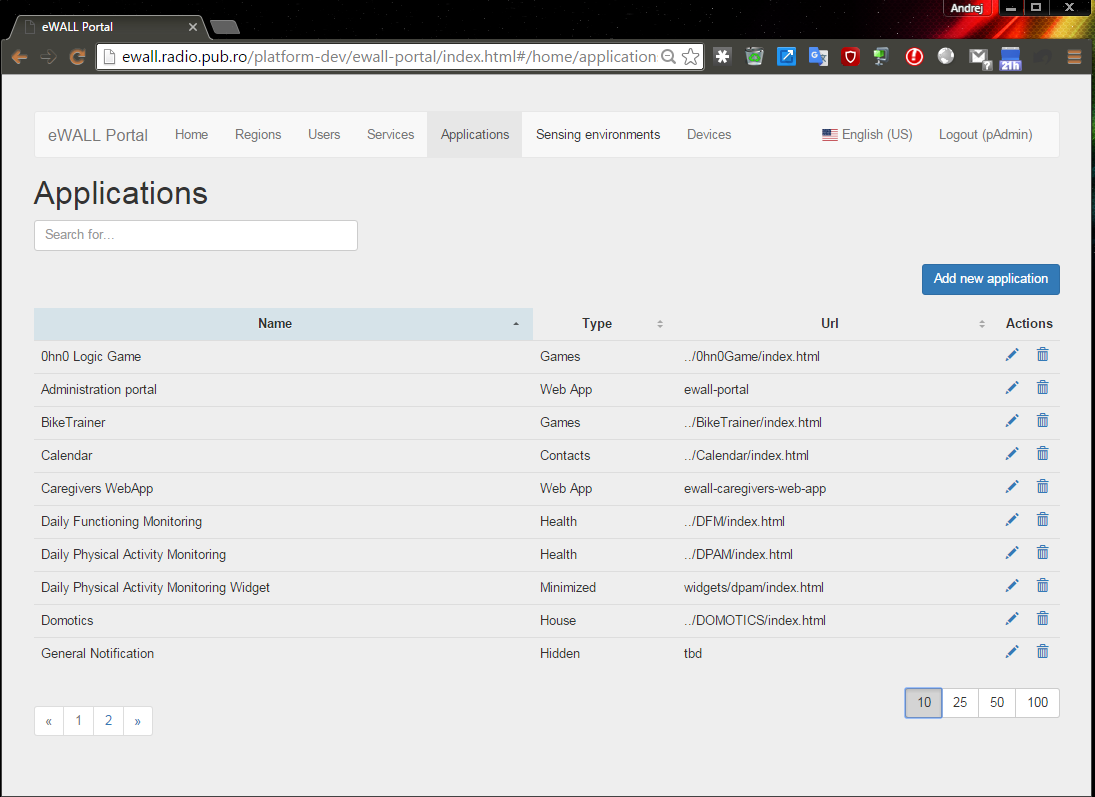


Figure 19 Screenshot of the “Applications” tab

Figure 19 shows open “Applications” tab which contains stored applications inside eWall system database which are handled inside *profiling-server* platform application.

Applications are shown in the table. Columns “Name”, “Type” and “Url” show homonymous properties of applications. Column “Actions” has buttons for editing application (labeled with pencil icon) and for deleting application (labeled with trash icon). Click on column captions (Name, Type and Url) sort applications by chosen property (column caption). Button for editing application opens new page with application properties which can be modified. Button for deleting application opens confirmation pop up dialog.

On the bottom of page are pagination and navigation bars explained in Figure 14. Just below “Applications” caption is placed textbox for filtering applications by values of all application properties shown in this table. Blue button “Add new application” is used for adding new application – page which opens after pressing that button is the same as for editing existing application.

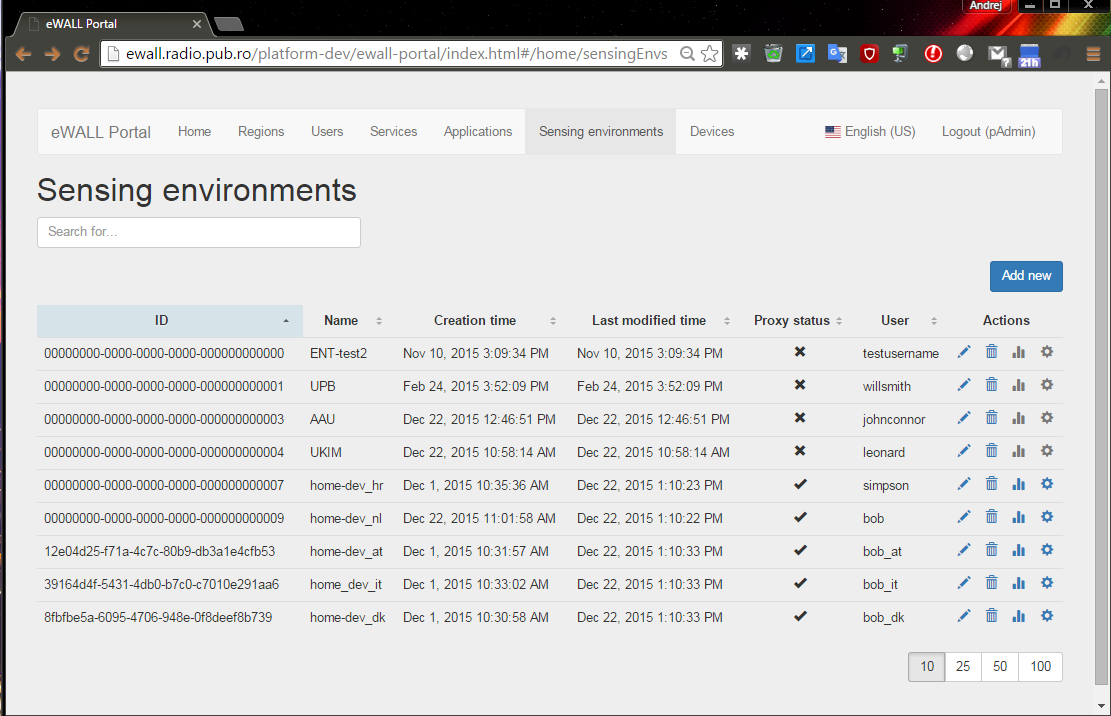


Figure 20 Screenshot of the “Sensing environments” tab

Figure 20 shows open “Sensing environments” tab which contains stored sensing environments inside eWall system database which are handled inside *cloud-gateway* platform application.

Sensing environments are shown in the table. Columns “ID”, “Name”, “Creation time”, “Last modified time”, “Proxy status”, “User” and “Actions” show homonymous properties of sensing environments. Column “Actions” has buttons for editing sensing environment (labeled with pencil icon), for deleting sensing environment (labeled with trash icon), for getting sensing environment status information (labeled with chart icon) and for editing configuration properties (labeled with cog icon). Click on column captions (ID, Name, Creation time, …) sort sensing environments by chosen property (column caption). Button for editing sensing environment opens new page with sensing environment properties which can be modified. Button for deleting sensing environment opens confirmation pop up dialog. Button for getting sensing environment status information opens page where can be seen load of CPU, used RAM memory and used disc root partition related to chosen sensing environment. Button for editing configuration properties opens page where can be edited properties loaded from *local-platform* platform application. Buttons for getting sensing environment status information and editing configuration properties are enabled only if sensing environment is registered.

On the bottom of page are pagination and navigation bars explained in Figure 14. Just below “Sensing environments” caption is placed textbox for filtering sensing environments by values of all sensing environment properties shown in this table. Blue button “Add new” is used for adding new sensing environment – page which opens after pressing that button is the same as for editing existing sensing environment.

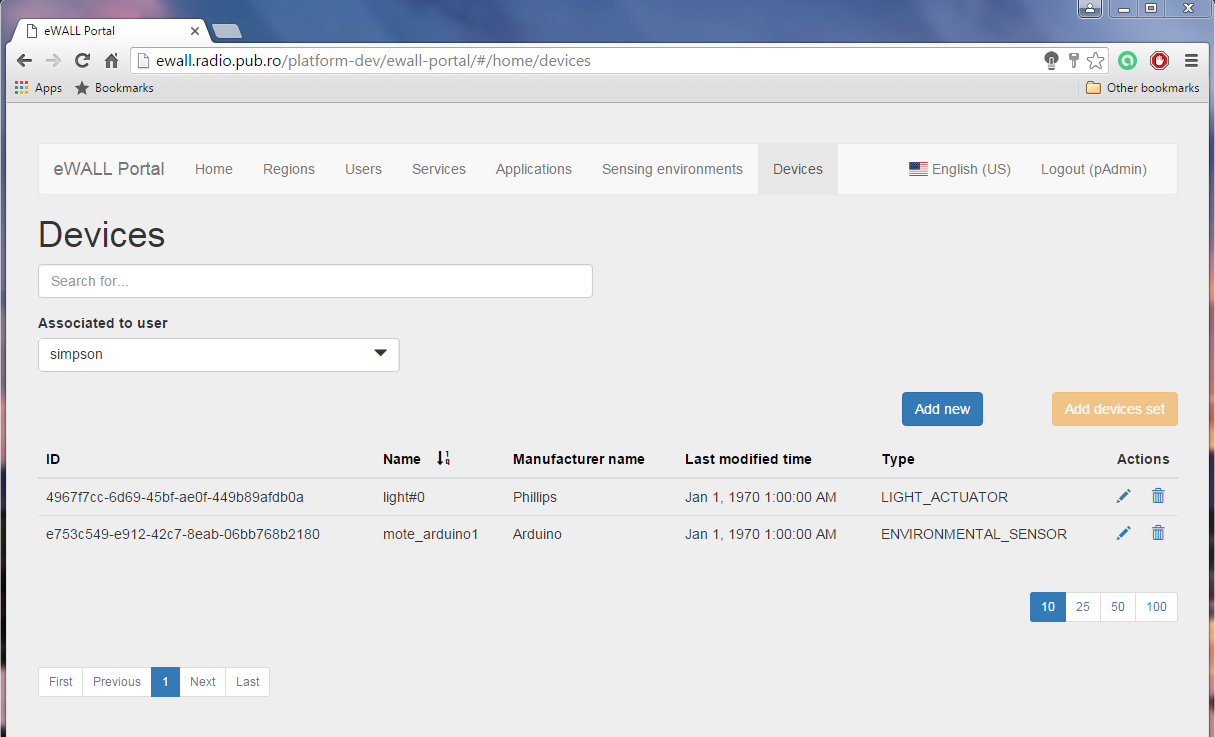


Figure 21 Screenshot of the “Devices” tab

Figure 21 shows open “Devices” tab which contains stored devices inside eWall system database which are handled inside *cloud-gateway* platform application.

Devices are shown in the table. Columns “ID”, “Name”, “Manufacturer name”, “Last modified time” and “Type” show homonymous properties of devices. Column “Actions” has buttons for editing device (labeled with pencil icon), for deleting device (labeled with trash icon). Click on column captions (ID, Name, Manufacturer name, …) sort devices by chosen property (column caption). Button for editing device opens new page with device properties which can be modified. Button for deleting device opens confirmation pop up dialog.

On the bottom of page are pagination and navigation bars explained in Figure 14. Just below “Devices” caption is placed textbox for filtering devices by values of all device properties shown in this table.

Blue button “Add new” is used for adding new device – page which opens after pressing that button is the same as for editing existing device (explained in Figure 22).

Orange button “Add devices set” is used for adding default devices set stored in default properties of *cloud-gateway* platform application (button is enabled only if there are no devices associated with selected sensing environment of user – in this figure button is disabled because there are 2 devices associated with simpson’s sensing environment). Click on button “Add devices set” opens confirmation pop up dialog.

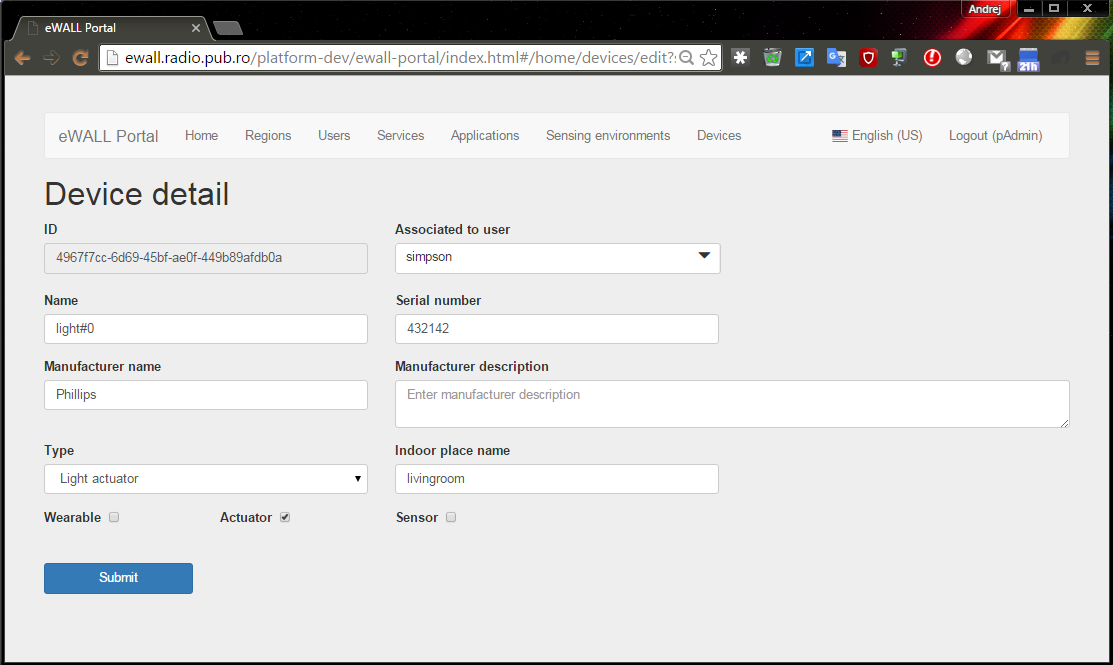


Figure 22 Screenshot of the “Device detail” edit page

Figure 22 shows page for adding or editing device.

On this page are shown values of device properties which can be edited, except ID. For device properties view are used common form input types: textboxes, checkboxes, and drop-down lists. On the bottom of this page is “Submit” button for submitting created or modified data.

1. http://www.oracle.com/technetwork/java/javaee/overview/index.html [↑](#footnote-ref-2)
2. http://spring.io/ [↑](#footnote-ref-3)
3. http://docs.oracle.com/javase/7/docs/technotes/guides/javadoc/ [↑](#footnote-ref-4)
4. http://swagger.io/ [↑](#footnote-ref-5)
5. https://gitlab.com/gitlab-org/gitlab-ce/blob/master/doc/permissions/permissions.md [↑](#footnote-ref-6)
6. <http://projects.spring.io/spring-boot/> [↑](#footnote-ref-7)
7. https://angularjs.org/ [↑](#footnote-ref-8)
8. http://www.mongodb.org/ [↑](#footnote-ref-9)
9. http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html [↑](#footnote-ref-10)
10. In production environments host web server provides SSL (https). All endpoints are invoked passing the “*X-Auth-Token*” header parameter, containing the token returned by the login. [↑](#footnote-ref-11)