Project Identification	
Project number	AAL-2013-6-091
Duration	1st June 2014 – 30th November 2016
Coordinator	Univ. Prof. Dr. Manfred Tscheligi
Coordinator Organization	AIT Austrian Institute of Technology GmbH, Austria
Website	www.pearl-project.eu

Platform for Ergonomic and motivating, ICT-based Age-friendly woRkpLaces (PEARL)

Document Identification	
Deliverable ID:	D-4.3.2 Programmable Interfaces and Configuration Management Tools
Release number/date	V1.0 02.08.2016
Checked and released by	Anton Katov/AAU

Key Information from "Description of Work"	
Deliverable Description	This deliverable describes the final prototype implementation of the programmable interfaces and configuration management tools.
Dissemination Level	PU = Public
Deliverable Type	P = Prototype, R = Report
Original due date	Project Month 24 / 01.June.2016

Authorship& Reviewer Information	
Editor	Anton Katov (AAU)
Partners contributing	AAU, SiLO, SENSAP, RRD
Reviewed by	Athanasios Moralis (SENSAP)

Abbreviations

Abbrev.	*Description*
AAL	Ambient Assisted Living
API	Application Programmable Interface
ATL	Ambient Tuning Layer
BIL	Business Intelligence Layer
CBR	Case-Based Reasoning
DB	Database
DSS	Decision Support System
HATEOS	Hypermedia as the Engine of Application State
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MVC	Model-View-Controller
REST	Representational State Transfer
RFID	Radio-Frequency IDentification
RSA	public-key cryptosystem (Rivest, Shamir, and Adleman)
SHA	Secure Hash Algorithm
SSO	Single Sign-On
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Table of Contents

List of Tables

List of Figures

Executive Summary

The integrated PEARL platform consists of a set of loosely coupled software components, deployed and managed in a distributed manner across the facilities of the different partners, thus introducing large variety in terms of utilized technologies and operation principles. For the provision of a unified service platform in such a scenario, the development of an efficient configuration management mechanism and standardized programmable interfaces becomes imperative. For the needs of PEARL we have chosen to implement a set of RESTful web services coupled with a collection of configuration management tools, which enable agile and robust exchange of configurations and application data. In order to fully cover the configurations management capabilities of the platform we describe the functionalities of the various tools in the context of four major scenarios – account creation and user registration, cross-authentication, task switching and ambient configurations deployment and preference editing. The principle of operation of the major software components, involved in the configurations management process is further elaborated by examining the related dataflow sequence diagrams. The main modules related directly or indirectly to the configurations management are indicated to be the cross authentication mechanism, the decision support system, the preference and user profile editor, the configuration manager and the message broker, which will be described in the context of the aforementioned scenarios. This deliverable describes the second version of the prototype. One major addition as compared to the first prototype version of the platform setup is the message broker application, which was developed in order to resolve the connectivity issue between the ATL (Task Switcher / Sensor – Actuator Manager) and the Workspace & Ambient Configurator and provide unique identification of user's workstation by pairing public and private IPs.

Furthermore, we focus on the description of the implementation techniques and the operation parameters of the major programmable interfaces, developed for the needs of PEARL. In order to present the design principles of the PEARL programmable interface, we have described in details the autologin and single sign-on mechanism, account creation and user registration APIs of the PEARL Task and Time Management and the Physical Wellbeing modules, the decision support system (DSS) and configurations manager invocation APIs, the task switcher and the trigger manager programmable interface. Furthermore, we summarize here the web services, developed for the needs of PEARL, which enable full control over the Task and Time Management module functionalities. The aforementioned were primarily developed to enable the communication with the digital paper calendar prototype – Calendula (see D2.3.2 and D3.2.2). Finally, we offer summary of our contribution and some concluding remarks.

1 About this Document

1.1 Role of the deliverable

The scope of D4.3.2 Programmable Interfaces and Configuration Management Tools is to provide a summary of the final version of the configurations management functionalities of the relevant PEARL platform components and to offer a detailed description of the major programmable interfaces that enable reliable exchange of configurations and application data. As such its main purpose is to serve as a reference point for debugging and improvement of the current software components or development of novel components and functionalities.

1.2 Relationship to other PEARL deliverables

The deliverable is related to the following PEARL deliverables:

Deliv:	***Relation***
D4.1	System Architecture Specification and Implementation: Deliverable D4.1 provides an anatomy of the PEARL platform through presenting its architecture and main components. D4.1 serves therefore as a basis for identifying the placement and basic functionality within the PEARL architecture.
D2.2	Use Cases, Scenarios and Integrated Functionalities: This deliverable presents integrated scenarios and use cases and describes the functional specifications of the PEARL applications. D2.2 serves therefore as a basis for the scenarios that will be used.
D4.2	User, Tasks and Workspaces Databases, Ontologies and Knowledge Bases. This deliverable provides the information for the data structures used in PEARL centralized database.
D2.3.2	Final PEARL User Interfaces. This deliverable serves as a basis to guidelines so as to achieve a user-friendly interface for the components needs.

1.3 Summary of changes over the previous version

When considering the previous prototype version of PEARL's programmable interfaces and configuration management tools several major points have to be noted. One of the major new additions is the design and implementation of the message broker application that aims to resolve the connectivity issues between the Ambient Tuning Layer and the Workspace and Ambient Configurator. The principles of operation, the newly developed message broker APIs and the resulting changes impacting the overall configuration management system architecture will be described in the following sections. Furthermore, all relevant components descriptions will be updated in order to accurately describe the final prototype of the PEARL system. Another major addition as compared to the first document is the detailed description of the existing as well as the newly developed APIs for data exchange between the PEARL Task and Time Management module and the digital paper user interface – Calendula (see D2.3.2 and D3.2.2). The description of the remaining programmable interfaces has also been consolidated and extended in order to provide a solid reference point for future debugging or development efforts.

2 Role within PEARL Architecture and System Overview

In the current chapter the major requirements of the PEARL project will be outlined and a general system overview will be presented, putting the focus on the configurations deployment and the necessary interfaces.

2.1 Requirements of PEARL project

The implementation of an efficient, agile and scalable configuration management mechanism and unified programmable interfaces, deployed across the various application layers, is of primary concern for the fluent operation of the integrated PEARL platform, because of its modular loosely coupled design, presented in D4.1 System Architecture Specification and Implementation. The major requirement of the project is to define and expose a set of standardized APIs, which will facilitate the development of an efficient decision support system and robust configuration management mechanism. The configuration management tools should further enable the integrated configurations deployment across the various application layer modules and should assist in the integration, development and configuration of new turn-key PEARL solutions. The development should focus on the following main aspects:

- Enabling unified account creation and user registration to all PEARL modules, transparent to the user.
- Enabling Single Sign-On (SSO) for accessing all applications with single set of credentials.
- Implementing web-based APIs that enable action triggers and data exchange between the various PEARL platform components, where needed.
- Enabling configurations retrieval, distribution and deployment across the holistic PEARL platform.

2.2 System Overview

The PEARL platform has been designed as a holistic entity that consists of a number of modules, deployed and managed in a distributed manner across the facilities of the different partners, resulting in a great variety in terms of development and implementation techniques and utilized technologies. In this distributed environment a standardized approach for configurations deployment and data exchange is of primary importance. The first step towards defining a unified approach that can incorporate the variety of PEARL application modules was the definition of the PEARL System architecture in “*D4.1 System Architecture Specification and Implementation”.* In the current document, we redefine the interactions of the major components focusing on the configuration management features and in the following sections we define the main programmable interfaces, developed for the needs of PEARL.

The general diagram of the architecture is presented on figure 1 below. It consists of three major entities each comprised of a set of logical components. The bottom horizontal layer of the PEARL architecture is the PEARL Data Layer which among others integrates the PEARL database schema and database access and the user management mechanism. From the view point of the configuration management mechanism the Data Layer is a key entity as it holds the user profile data and the corresponding configuration plans that define all major configuration settings of the workspace environment and the PEARL application layer modules.

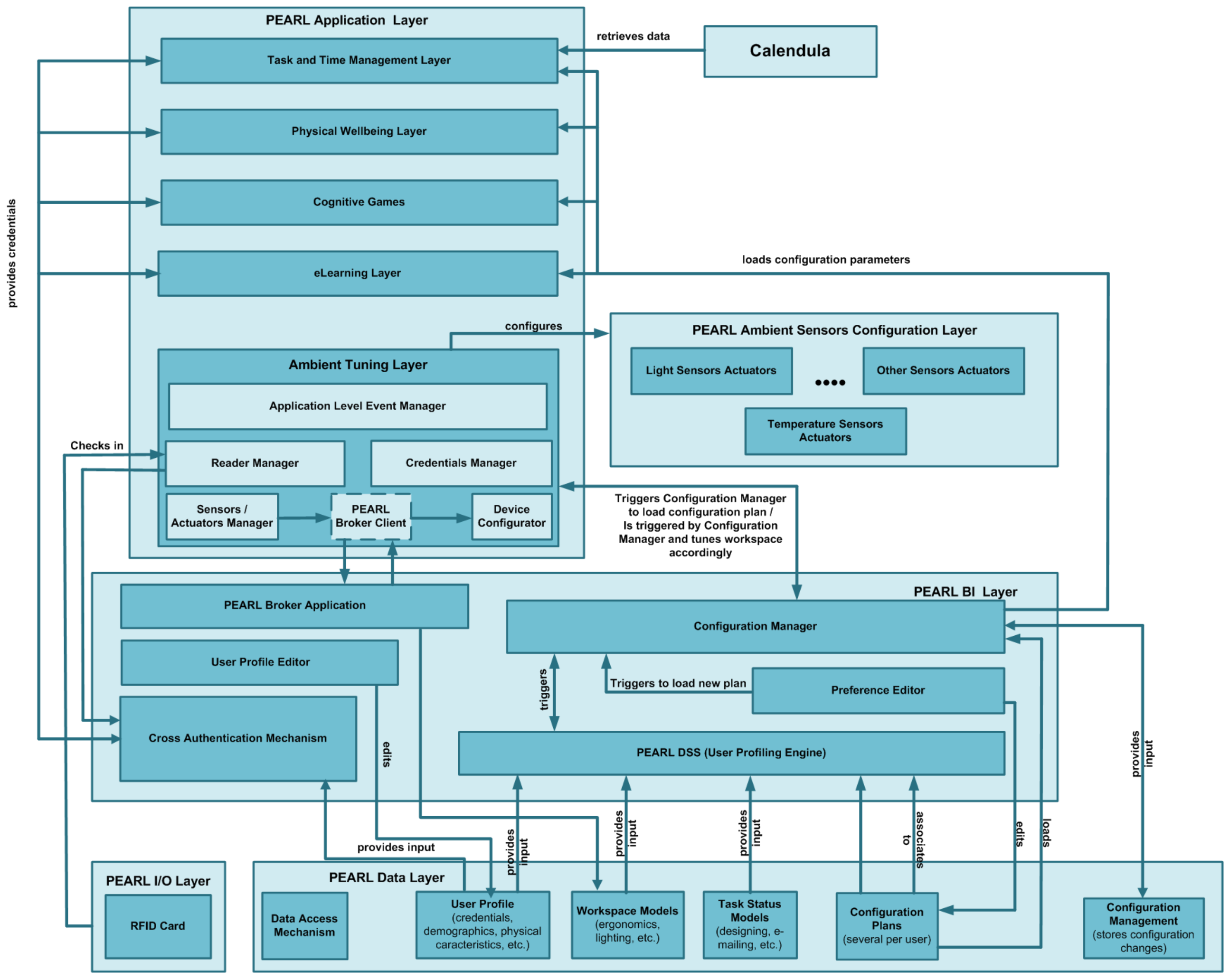

PEARL Application Layer
Task and Time Management Layer
Physical Wellbeing Layer
Cognitive Games
eLearning Layer
retrieves data
Calendula
loads configuration parameters
provides credentials
configures
PEARL Ambient Sensors Configuration Layer
Light Sensors Actuators
Other Sensors Actuators
Temperature Sensors Actuators
Ambient Tuning Layer
Application Level Event Manager
Checks in
Reader Manager
Credentials Manager
Sensors / Actuators Manager
PEARL Broker Client
Device Configurator
Triggers Configuration Manager to load configuration plan / Is triggered by Configuration Manager and tunes workspace accordingly
PEARL BI Layer
PEARL Broker Application
Configuration Manager
User Profile Editor
triggers
Triggers to load new plan
Preference Editor
Cross Authentication Mechanism
edits
PEARL DSS (User Profiling Engine)
provides input
provides input
provides input
provides input
associates to
edits
loads
provides input
PEARL I/O Layer
RFID Card
PEARL Data Layer
Data Access Mechanism
User Profile (credentials, demographics, physical caracteristics, etc.)
Workspace Models (ergonomics, lighting, etc.)
Task Status Models (designing, e-mailing, etc.)
Configuration Plans (several per user)
Configuration Management (stores configuration changes)

Figure 1. PEARL System Overview

On top of the Data Layer is the second horizontal layer - the PEARL Business Intelligence Layer (BIL), which implements most of the tools, related either directly or collaterally to the configuration management process – the cross authentication mechanism, the decision support system, the preference and user profile editor, the configuration manager and the PEARL Broker Application. The cross authentication mechanism is responsible for providing SSO for the users across all PEARL application domains. The decision support system defines customized configuration plans upon registration of new users, thus facilitating the configurations deployment and improving the overall user experience. The preference and user profile editors provide ergonomic user interface (UI) for direct management of the user profile characteristics and configuration settings and are responsible for triggering the configuration manager and the ambient tuning layer. The configuration manager's main function is to distribute and trigger the deployment of new configuration settings across the various application modules. The PEARL Broker Application takes care of the parameters mismatch between the Ambient Tuning Layer and the Workspace and Ambient Configurator, providing an automated mechanism for mapping public and private IP addresses to a set of configurations, so as to provide seamless integration between the centralized PEARL data layer and the locally distributed Workspace and Ambient Configurator agents.

The final major entity of the PEARL holistic platform is the PEARL Application Layer, which hosts the various application modules. In the context of configuration management a leading role here has the Ambient Tuning Layer, which is responsible for delivering the personalisation of the workplace, and for integrating tools that can assist the employee in tuning the ambient environment based on a combination of software and hardware utilities.

3 Scenarios and Configurations Management

Based on the holistic system architecture and the general platform lifecycle, described in "*D2.2 Detailed Use Cases and Scenarios"*, we have indentified the major scenarios that will be used for describing the different aspects of the configuration management mechanism. To elaborate further on the application logic of each of the tools we fill further describe the major data flows in terms of sequence diagrams.

3.1 Account Creation and User Registration

In order for a user to have access to PEARL platform and all its services, s/he must first create an account and register at the platform. The creation of a user account / registration of a user can take place in two main ways:

1) The user can manually navigate to the PEARL platform login / registration page and follow the instructions to create an account or,
2) The user can scan his/her RFID card, which – if the user does not exist and the specific RFID card has not been associated with any user and is valid - will automatically redirect the user to the PEARL platform registration page. It must be noted that in this case, for security purposes, the Tag_ID is encrypted before it is actually sent to the browser, using a public-key cryptosystem, described below in chapter 4 of the current deliverable, and thus can only be decrypted by the authentication mechanism. After a successful registration process, a user can log-in to the PEARL platform either using the credentials provided or by using the RFID card.

Upon the first login the user is requested to fill out a number of personal preferences that directly influence the configurations of his/her workplace environment. In order to enhance the overall user experience, the PEARL platform implements a decision support system (DSS), which provides customized configuration plans to the new users, based on their profile characteristics. The DSS consists of two main submodules: the matchmaker and the rule engine. The matchmaker is based on the Case-Based Reasoning (CBR) technique and is powered by myCBR[1]. It is responsible for matching a set of user preferences to a similar existing profile and retrieving the relevant configuration plan. The PEARL Rule Engine is powered by Drools[2] and incorporates the case adaptation logic required whenever a retrieved solution does not satisfy fully the initial user requirements and needs to be altered. The case adaptation logic consists of predefined IF-THEN rules that will ultimately customize the retrieved configuration plan to match as closely as possible the corresponding user profile. For further information regarding the PEARL DSS implementation and underlying technologies, please refer to "*D4.4.2 Decision Support System and Rule Engines"*.

[1] http://www.mycbr-project.net/
[2] http://www.drools.org/

In order to provide maximum flexibility during the initial platform configuration phase a semi-automated operation model will be utilized, allowing the user to either accept the suggested configuration plan or further modify the settings, based on his/her preferences. The modification will be possible via the Preference Editor, which provides an ergonomic UI, adapted for the needs of the elderly employees. The corresponding data flow is presented on figure 2 and the main steps are summarized below:

1. **Initial data input** – the new user is prompted to fill out a number of forms defining his account credentials and a set of personal characteristics.
2. **DSS trigger** – the input data are stored in the PEARL database and the DSS is triggered by sending an HTTP GET request with a path variable containing the ID of the newly created user.
3. **Problem retrieval** – the user profile characteristics of the newly created user are retrieved by the DSS by executing a database query based on the user ID. The resulting data set constitutes the input problem definition for the CBR engine.
4. **Case base construction**– the CBR case base is constructed by retrieving from the PEARL database all existing user profiles that have a full set of configuration settings already defined. An additional parameter, “*setConfigurations”,* that indicates the validity of a given database entry is introduced.
5. **Case retrieval** – the best matching user profile is selected from the case base, based on predefined similarity functions, provided by myCBR engine. The corresponding configuration settings are then retrieved from the database.
6. **Case adaptation** – the retrieved configurations data set is then adapted by the rule engine based on predefined rules.
7. **Adapted solution** – the adapted configurations are then passed to the Preference editor in JavaScript Object Notation (JSON) format and displayed to the user.
8. **Solution revision** – the user is then prompted to either accept or modify the suggested configurations via the Preference Editor UI.
9. **Case retention** – when the configurations are adapted/accepted by the user, the resulting set of parameters is stored to the database and the *setConfigurations* variable is changed, thus indicating that the new case can be included in the case base in the future.
10. **Configurations deployment** – the configurations manager and the ambient tuning layer trigger manager are activated in order to deploy the new configurations.

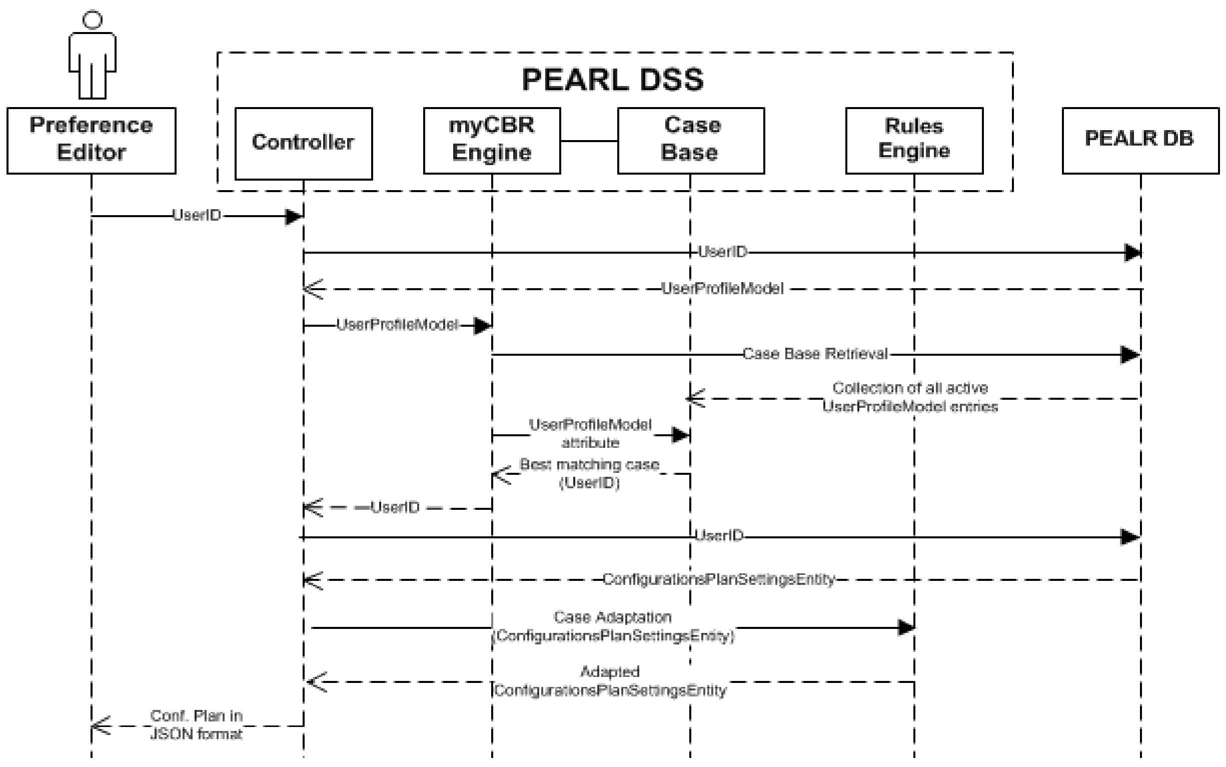

PEARL DSS
Preference Editor
Controller
myCBR Engine
Case Base
Rules Engine
PEALR DB
UserID
UserID
UserProfileModel
UserProfileModel
Case Base Retrieval
Collection of all active UserProfileModel entries
UserProfileModel attribute
Best matching case (UserID)
UserID
UserID
ConfigurationsPlanSettingsEntity
Case Adaptation (ConfigurationsPlanSettingsEntity)
Adapted ConfigurationsPlanSettingsEntity
Conf. Plan in JSON format

Figure 2. DSS-based configurations retrieval

3.2 Cross-Authentication

The PEARL holistic platform integrates a number of services, which can be either hosted and provided centrally, or remotely. The reason for this is so as to enable the platform to integrate and deliver to PEARL users, services from third party providers. In the current version of the system, these service providers are partners of the consortium (e.g. RRD and COMARG), but after the commercialisation of the platform, these could be external service providers who could provide added value services through the PEARL platform, hosted however in remote, own premises. Towards this end, the PEARL platform incorporates a cross authentication mechanism that is responsible for authenticating the users in all platforms, providing to the users a seamless sign in experience. The secure authentication mechanism is analysed in chapter 4 of the current deliverable. It relies on a set of secure REST web services, one to add a user to the platform, and one to delete the user from the platform, the same web services that are employed during account creation and user registration. This set of web services safeguards that a synchronised list of users is always available across all platforms integrated in PEARL.

3.3 Task Switching and Ambient Configurations Deployment

One of the most distinct services, provided by the PEARL platform is the per task configuration of the workspace environment, provided by the Ambient Tuning Layer. Through this service, the user is able to define different configurations settings for the different daily tasks he is involved in (e.g. email communication, text editing, teleconferencing, etc.) and transform his ambient workspace environment with a click of a button on the PEARL Task Switching UI. The task switching and the deployment of the relevant configurations are managed by the task switcher manager, the trigger manager and the sensors/actuators manager.

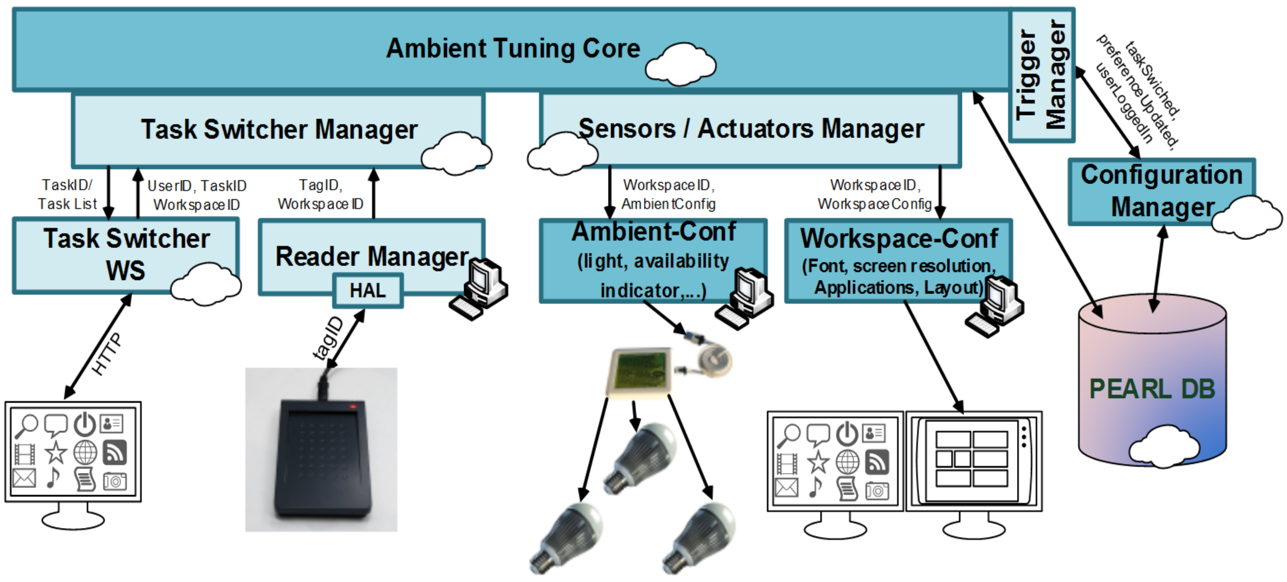

Ambient Tuning Core
Trigger Manager
taskSwitched, preferenceUpdated, userLoggedIn
Task Switcher Manager
Sensors / Actuators Manager
Configuration Manager
TaskID/ Task List
UserID, TaskID WorkspaceID
TagID, WorkspaceID
WorkspaceID, AmbientConfig
WorkspaceID, WorkspaceConfig
Task Switcher WS
Reader Manager
HAL
Ambient-Conf (light, availability indicator,...)
Workspace-Conf (Font, screen resolution, Applications, Layout)
PEARL DB
HTTP
tagID

Figure 3. Ambient Tuning Layer architecture

Task switcher manager is the component responsible to interact with the user so as to provide him with the ability of switching tasks. It provides appropriate interfaces so as to inform the PEARL platform (see Figure above), either directly the Ambient Tuning Core or the rest of the PEARL components by the use of the Trigger manager interface, for a change of a task. The Trigger Manager (Figure 3 above), provides the appropriate interfaces so as to enable the components that interacts with the User (Reader Manager, Task Switcher) to inform the rest of the PEARL platform that a change has occurred (i.e. preferences has been updated, a tasks has been switched or a user has logged in).

The User is capable to initiate a different task by hitting the equivalent graphical button on the dedicated touch screen. The Ambient Tuning Layer (ATL) receives the trigger and identifies the unique task ID assigned to this specific task, the UserID and the WorkspaceID. Using this information, The ATL queries the PEARL DB to retrieve the User's personal configuration related with the new task which includes general preferences (e.g., in terms of applications to be used, environment light, availability indicator etc.), but also of the task he is carrying out (Task configuration plan) and the User Info (id, name). The ATL passes the TaskID and UserID to the Configuration Manager through the trigger manager I/F (Figure unterhalb) so as to inform the rest of the PEARL components for the task change. The ATL undertakes the task to configure the workspace environment by controlling the available ambient sensors and actuators based on the User Preferences. Finally the ATL updates the User Task Switcher UI by highlighting the current task.

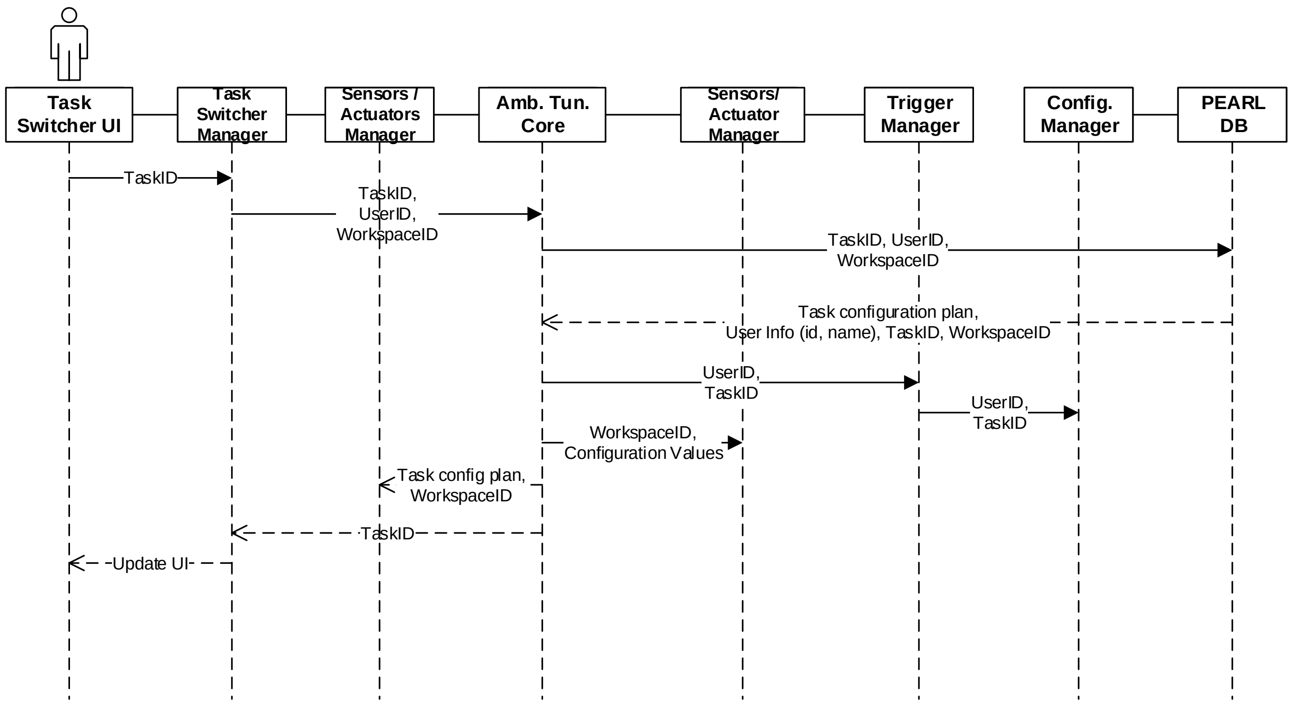

Task Switcher UI
Task Switcher Manager
Sensors / Actuators Manager
Amb. Tun. Core
Sensors/ Actuator Manager
Trigger Manager
Config. Manager
PEARL DB
TaskID
TaskID, UserID, WorkspaceID
TaskID, UserID, WorkspaceID
Task configuration plan, User Info (id, name), TaskID, WorkspaceID
UserID, TaskID
UserID, TaskID
WorkspaceID, Configuration Values
Task config plan, WorkspaceID
TaskID
Update UI

Figure 4. Task switching sequence diagram

3.4 Message Broker Application

As it was identified during the pre-pilot of the first integrated prototype, network configurations can impose limitations on the communication between the PEARL local client integrating the reader manager, the workspace and ambient configurator, and the rest of the “remotely hosted” components of the PEARL platform. Towards this end, and because not all personal computers and laptops could have fixed IPs so as to directly facilitate communication between the reader manager and the ambient and workspace configurators, the concept of an intermediator, namely a broker was introduced. The main task of the broker application would be to retrieve the configuration file that was originally transmitted from the Task Switcher and Actuator Manager to the Workspace and Ambient Configurator, and properly forward it to the appropriate client (user PC) “hidden” behind a router forwarding network traffic. In order to facilitate this, the broker application utilises a unique pair of public and private IPs which uniquely characterise a client. Both the public and private IPs are retrieved and sent to the broker application through the locally installed Reader Manager. The conceptual architecture of the Broker Application is graphically illustrated in the figure below. In section 4.3.5, the authors provide information regarding the broker REST API used to post the information included in the JSON file from the Actuator Manager, while the implementation details of the broker application are provided within the context of deliverable D4.5.2.

In addition to the conceptual approach, the corresponding data flow is presented on the sequence diagram on figure 6.

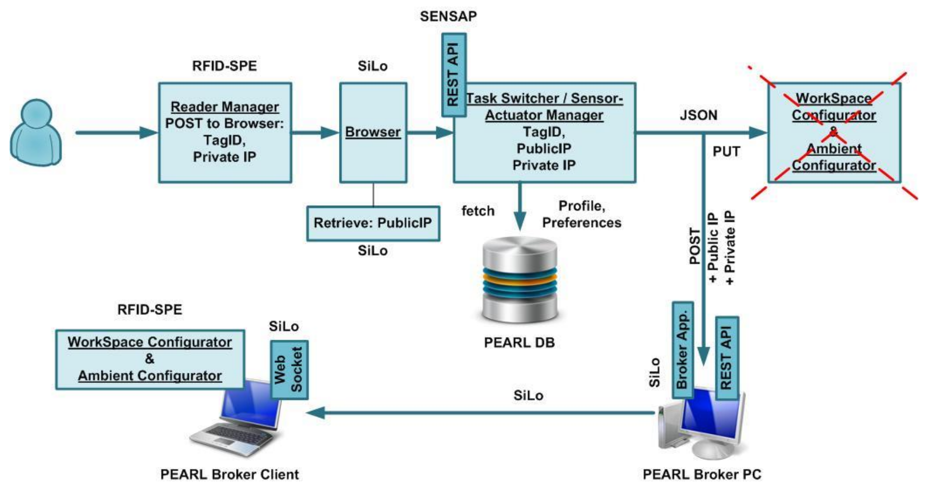


Figure 5. Ambient Tuning Layer Message Broker Application

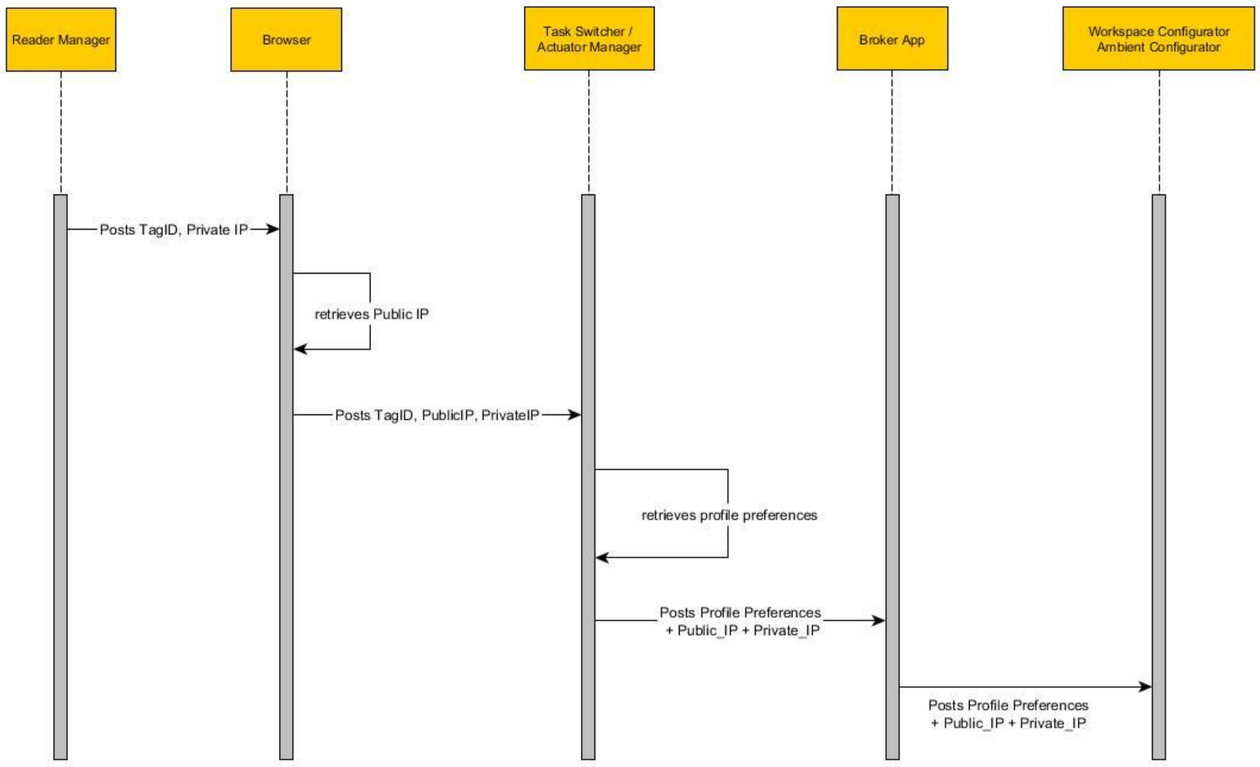
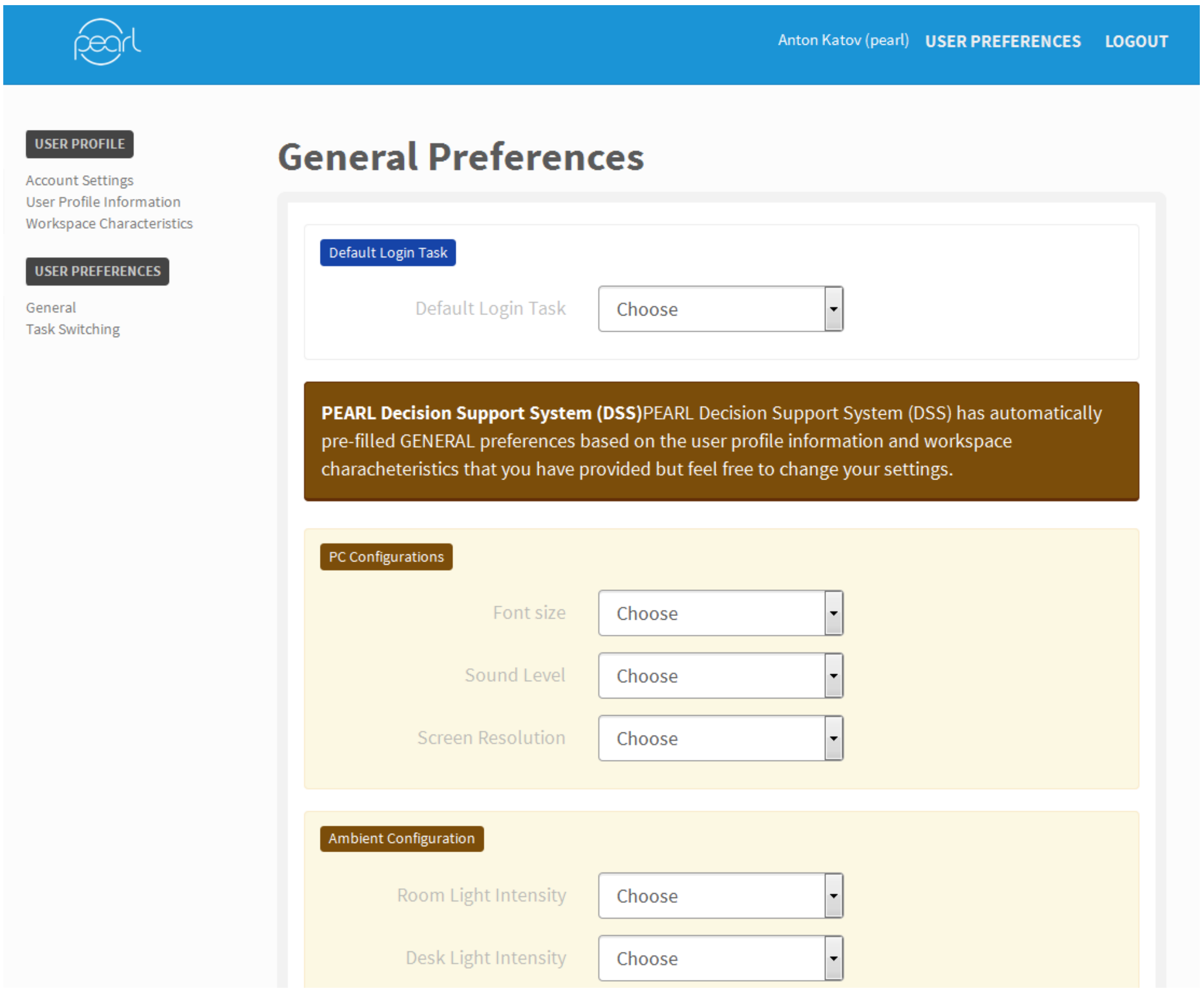

Reader Manager
Browser
Task Switcher / Actuator Manager
Broker App
Workspace Configurator Ambient Configurator
Posts TagID, Private IP
retrieves Public IP
Posts TagID, PublicIP, PrivateIP
retrieves profile preferences
Posts Profile Preferences + Public_IP + Private_IP
Posts Profile Preferences + Public_IP + Private_IP

Figure 6. Preference editing sequence diagram

3.5 Preference Editing

After the initial registration the user has defined his/her preferred configuration settings and the corresponding values have been stored in the database and are available for retrieval by the DSS and Ambient Tuning Core. During the platforms running phase the user can access and modify the stored configuration settings via the Preference Editor UI. The Preference Editor UI is presented on figure 7, for further details please refer to "*D2.3.2 Final PEARL User Interfaces*".



General Preferences

Default Login Task

Default Login Task: Choose

PEARL Decision Support System (DSS)PEARL Decision Support System (DSS) has automatically pre-filled GENERAL preferences based on the user profile information and workspace characteristics that you have provided but feel free to change your settings.

PC Configurations

Font size: Choose

Sound Level: Choose

Screen Resolution: Choose

Ambient Configuration

Room Light Intensity: Choose

Desk Light Intensity: Choose

Figure 7. Preference editor UI

After the user edits his preferences, the altered settings will be stored in the database and will be deployed by the Ambient Tuning Core or the Configurations Manager, depending on what has been changed. If the altered settings concern the ambient environment configurations or the list of the available tasks, the Ambient Tuning core will be involved. If the changes involve any of the other application layer modules, the deployment will be undertaken by the Configurations Manager. The corresponding data flow is presented on the sequence diagram on figure 8.

The user is selecting the new configurations via the Preference Editor UI. The configurations are stored in the PEARL database by the Preference Editor and the configurations manager is triggered by sending the UserID, the TaskID (if applicable) and the edited configurations parameter. If the changes concern the ambient workspace environment, the configurations manager initiates the Ambient Tuning Core by sending a REST call to the Trigger Manager, containing the UserID and the TaskID of the altered task. The trigger manager provides the necessary interfaces to the Ambient Tuning Core. The Ambient Tuning Core then retrieves the most recent task configuration directly from the PEARL database and triggers the Task Switcher Manager and the Sensors/Actuator manager which the deploy the new configurations.

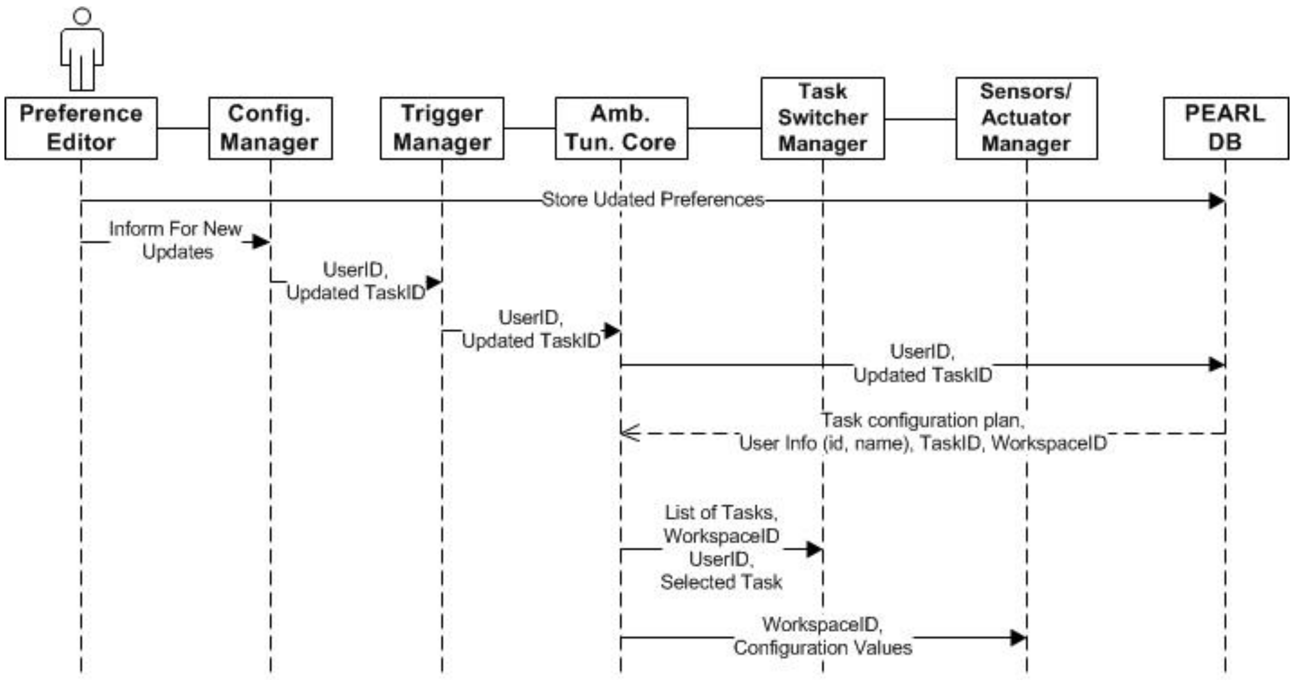
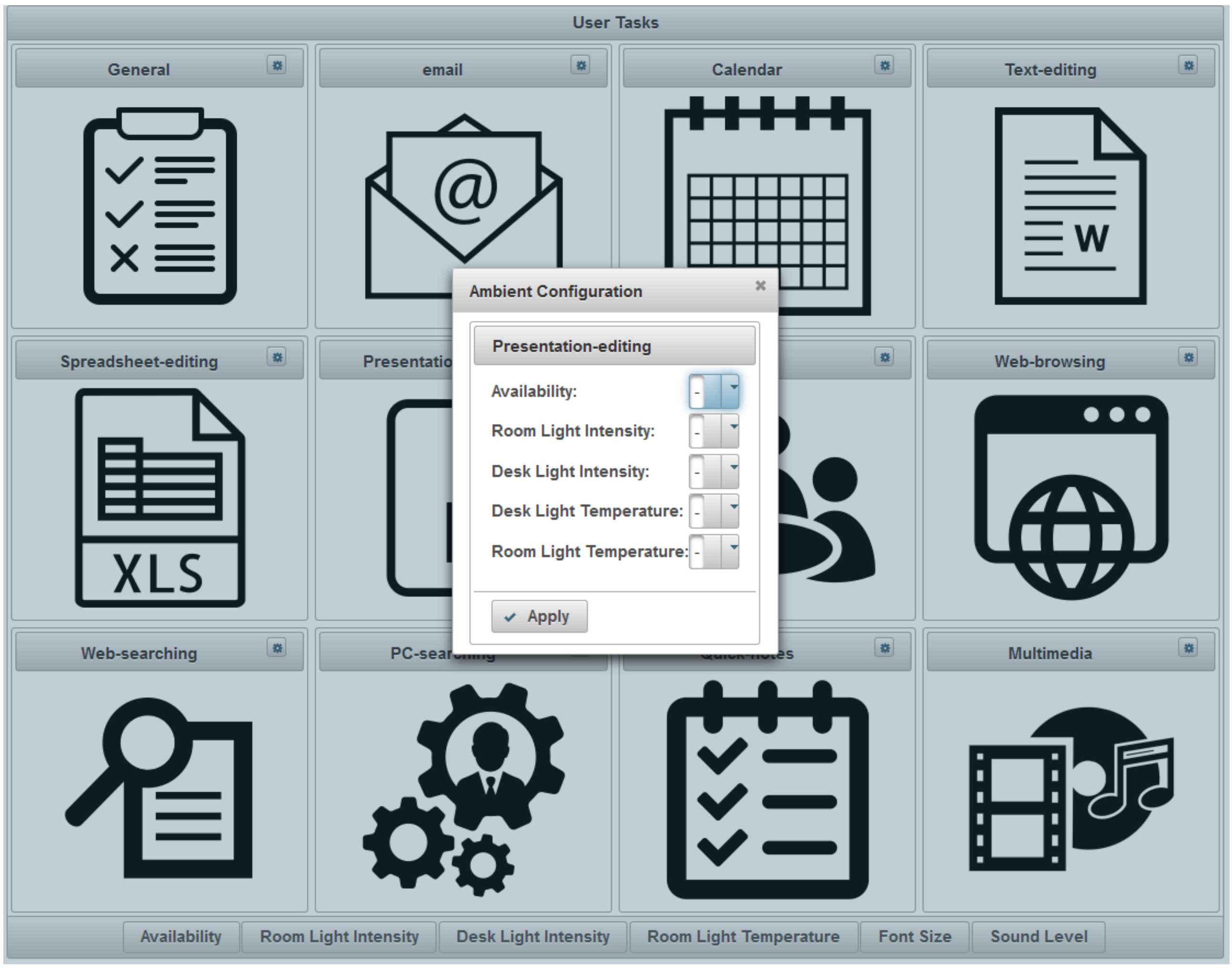


Figure 8. Preference editing sequence diagram

For improving the platform's usability and facilitating the users interactions with PEARL an additional UI for preference editing has been added as a part of the task switching UI. This offers a direct access for configurations adjustments both related to the general set of preferences and the specific task related preferences. This allows the user to directly alter the configurations while switching between tasks. The requested changes will be directly applied by the Sensors-Actuators Manager via the PEARL Broker Application and saved to the PEARL database. For further details, regarding the underlying data flow please refer to section 3.4 of the current document. The UI for direct editing of the workspace configurations is presented in Figure 9. For more detailed description please refer to "*D2.3.2 Final PEARL User Interfaces*".

If the changes concern any of the other application modules, the configuration manager issues a REST call to the respective application layer modules.



Ambient Configuration

Presentation-editing

Availability: -

Room Light Intensity: -

Desk Light Intensity: -

Desk Light Temperature: -

Room Light Temperature: -

Apply

Figure 9. Task Switcher UI for Preference Editting

4 Programmable Interfaces and APIs

In the current chapter we summarize the web-based APIs that have been developed and deployed over the various PEARL modules to enable reliable and secure exchange of configuration and application data. The current chapter will serve as reference for the future implementation activities and will be periodically updated upon the development of new APIs.

4.1 Account Creation and User Registration

In order for a user to have access to PEARL platform and all its services, s/he must first create an account and register at the platform. The creation of a user account / registration of a user can take place in two main ways:

1) The user can manually navigate to the PEARL platform login / registration page and follow the instructions to create an account. In this case, the user navigates to the login page (http://pearl.euprojects.net/login), selects “create account”, and fills in all the required information.
2) The user can scan his/her RFID card, which – if the user does not exist and the specific RFID card has not been associated with any user and is valid - will automatically redirect the user to the PEARL platform registration page.

When the user scans his/her RFID card, there are three main scenarios that can take place:

- Scenario 1 (Successful Login): A user has already registered to the PEARL platform and the ID of his/her RFID card is associated with his/her account. In this scenario, when the user passes the RFID card over the RFID scanner, the user's' preferred browser opens, and an auto login request is sent to the PEARL platform. The authentication mechanism authenticates the user and redirects the request to the home page of the PEARL platform.

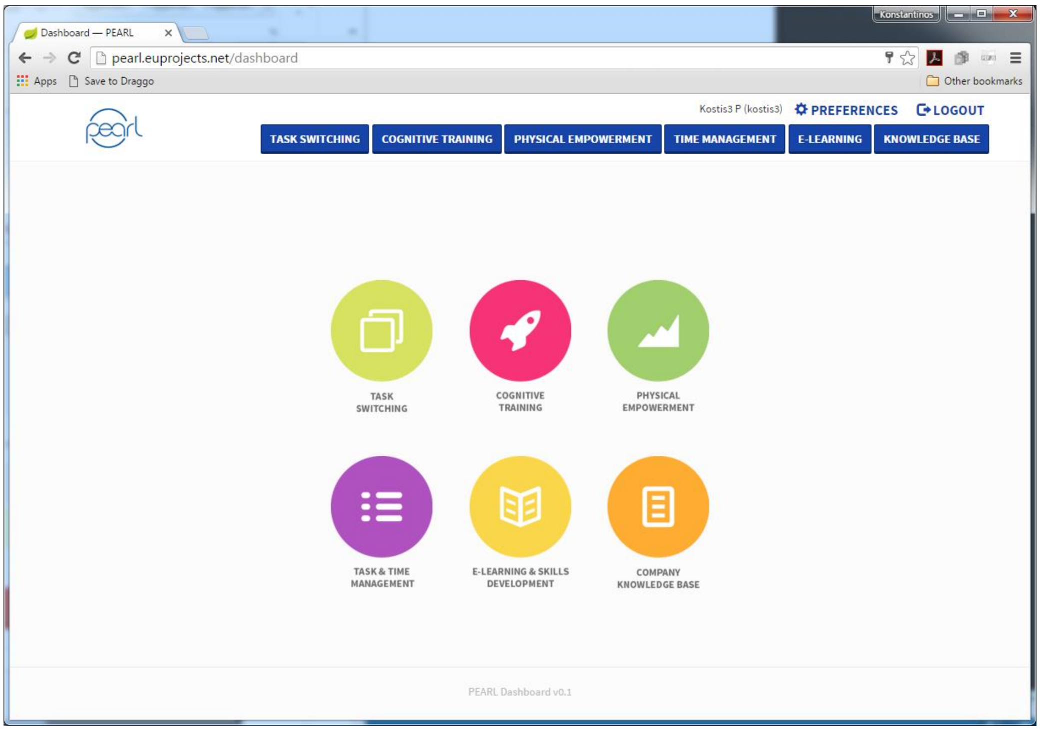


Figure 10. Successful login - 1

- Scenario 2 (Invalid Tag_ID): A user has already registered to the PEARL platform and an RFID card has been assigned to his/her account, however an invalid RFID Tag_ID is sent. In this case, the scenario authentication process fails, because an invalid or a corrupted RFID Tag_ID is sent. The auto login request is redirected to the login page in order for the user to manually login to the PEARL platform.

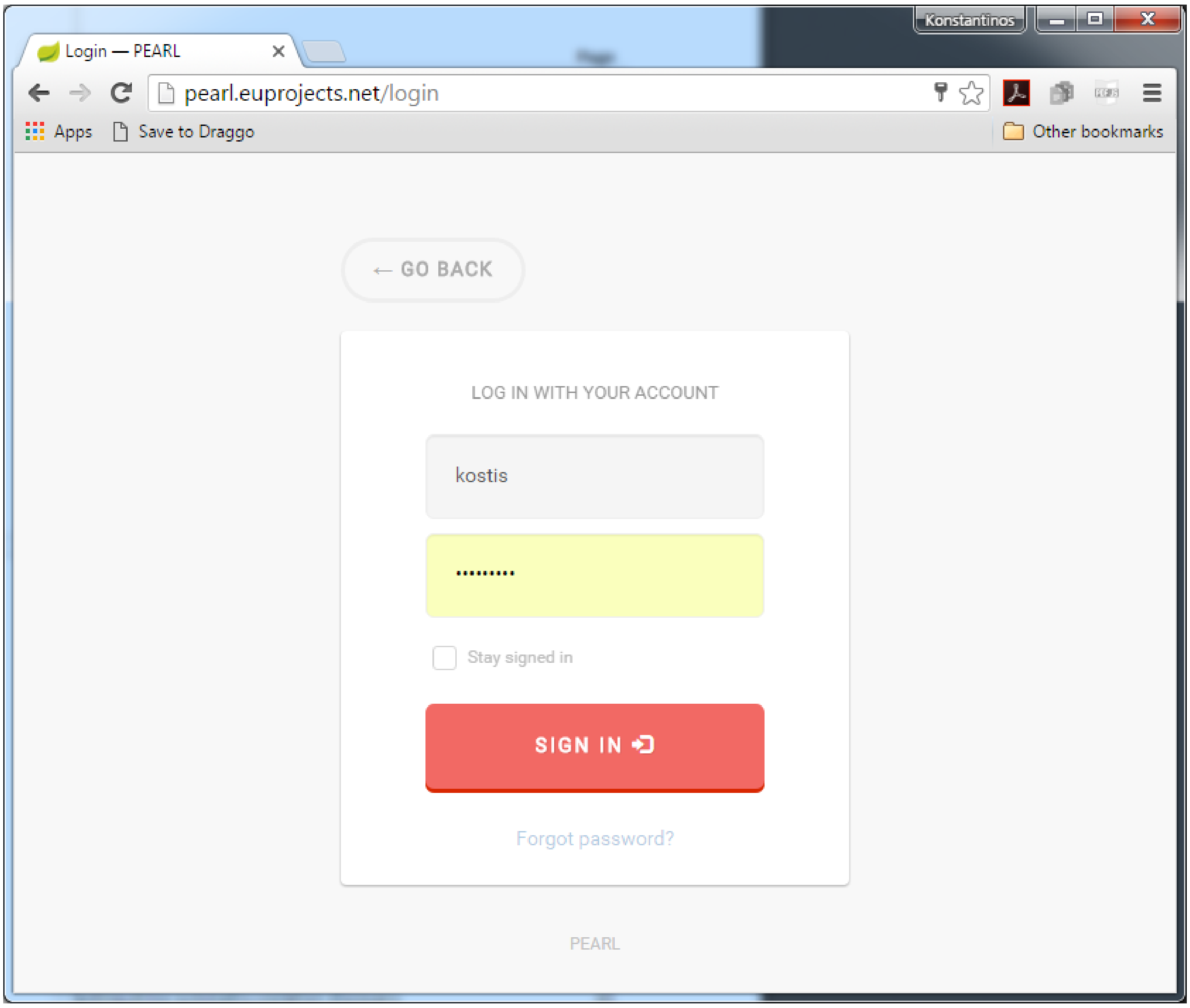


Figure 11. Successful login - 2

- <u>Scenario 3 (User Registration):</u> A user is not registered to the PEARL platform however a valid RFID Tag_ID is sent. The authentication mechanism cannot match the RFID with a registered user, thus the auto login request is redirected to the registration page, so that the user can register to the PEARL platform.

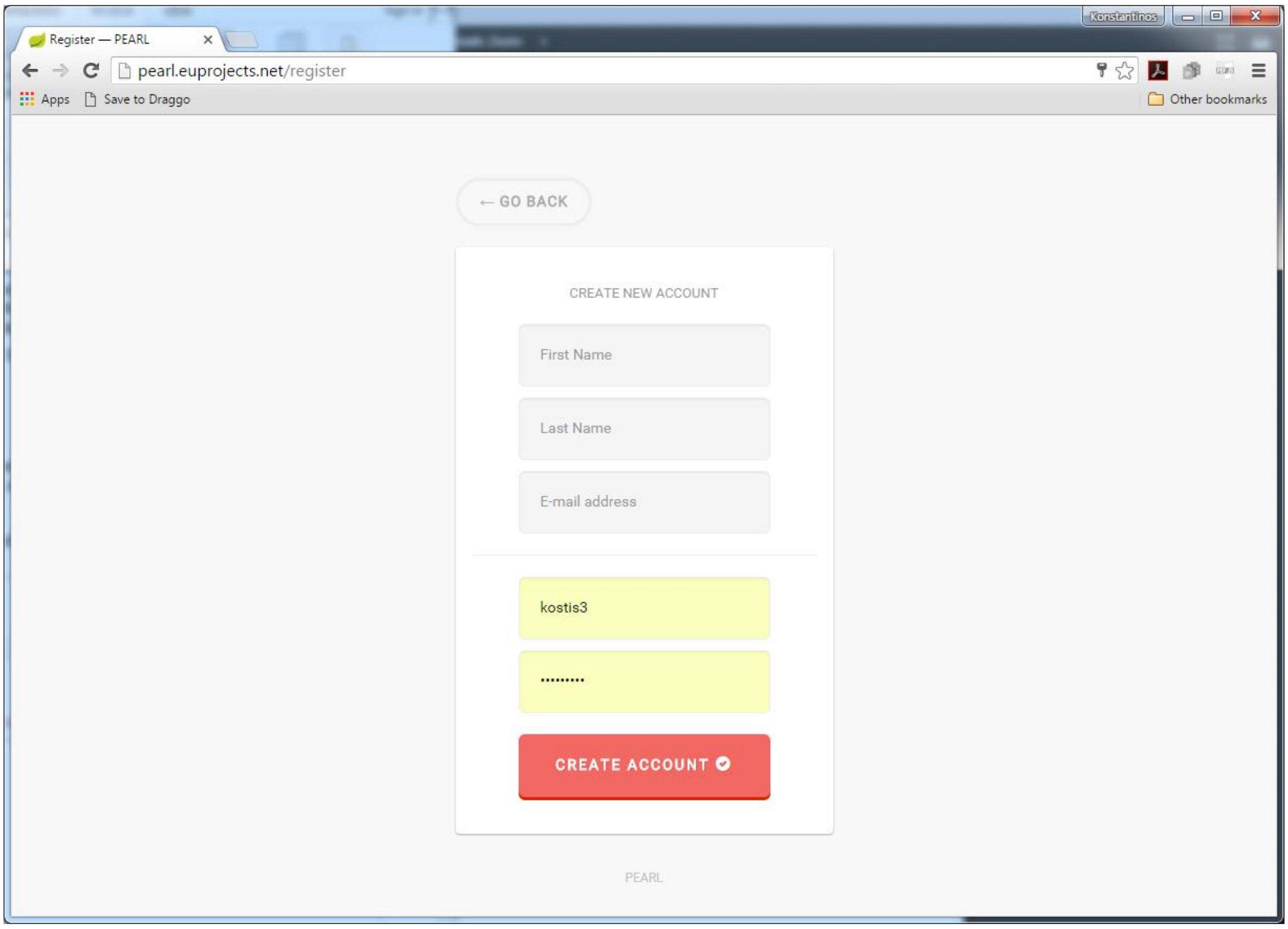


Figure 12. Successful login - 3

The autologin URI required information is provided below:

Method: GET

Parametres: tagID (String), ip (String)

Endpoint: http://pearl.euprojects.net/autologin

***tagID** Is BASE64 encoded string containing an encrypted information using asymmetric key. In addition, the ip parameter is encoded using a basic url encoder.

The aforementioned scenarios can be visualized in the following user authentication flowchart:

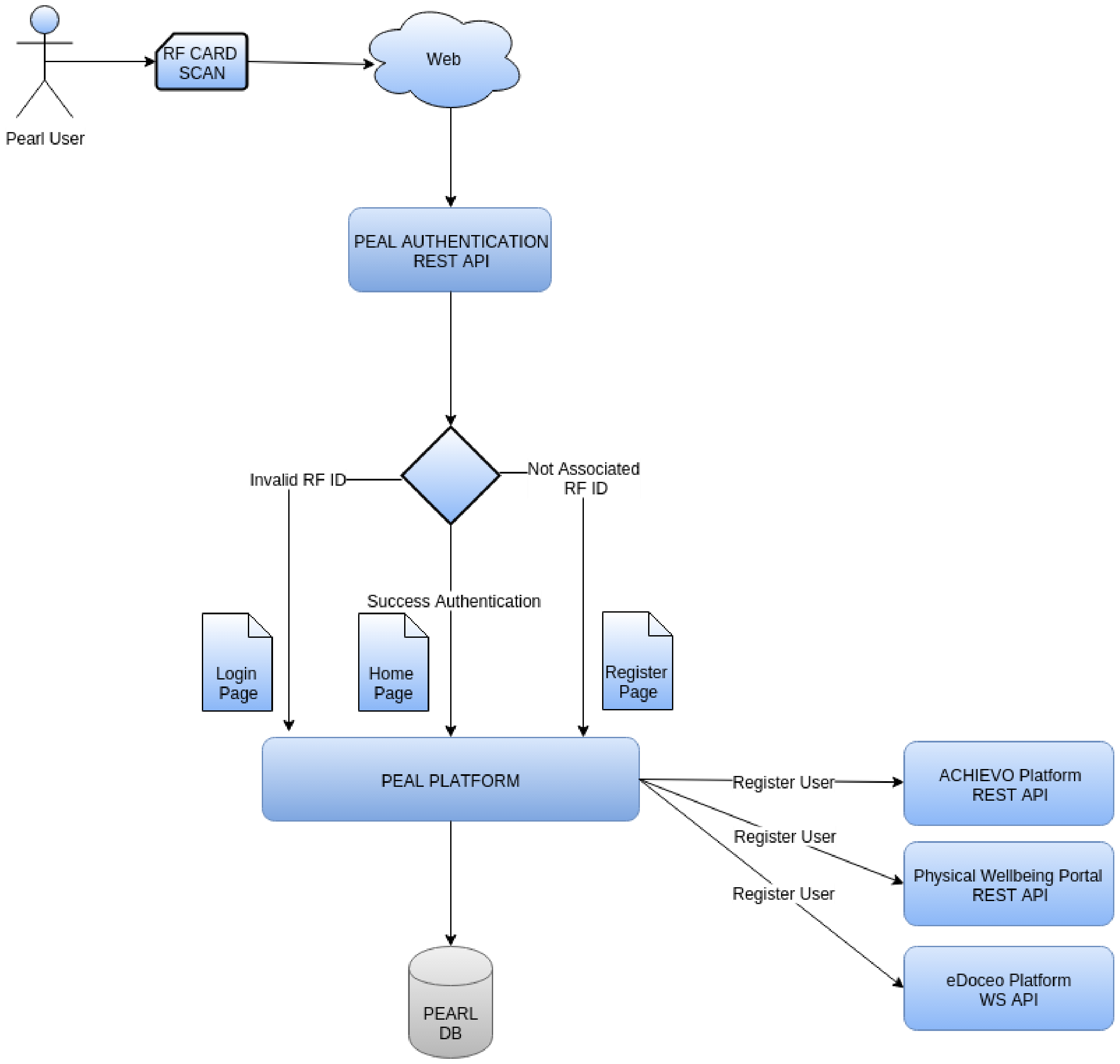
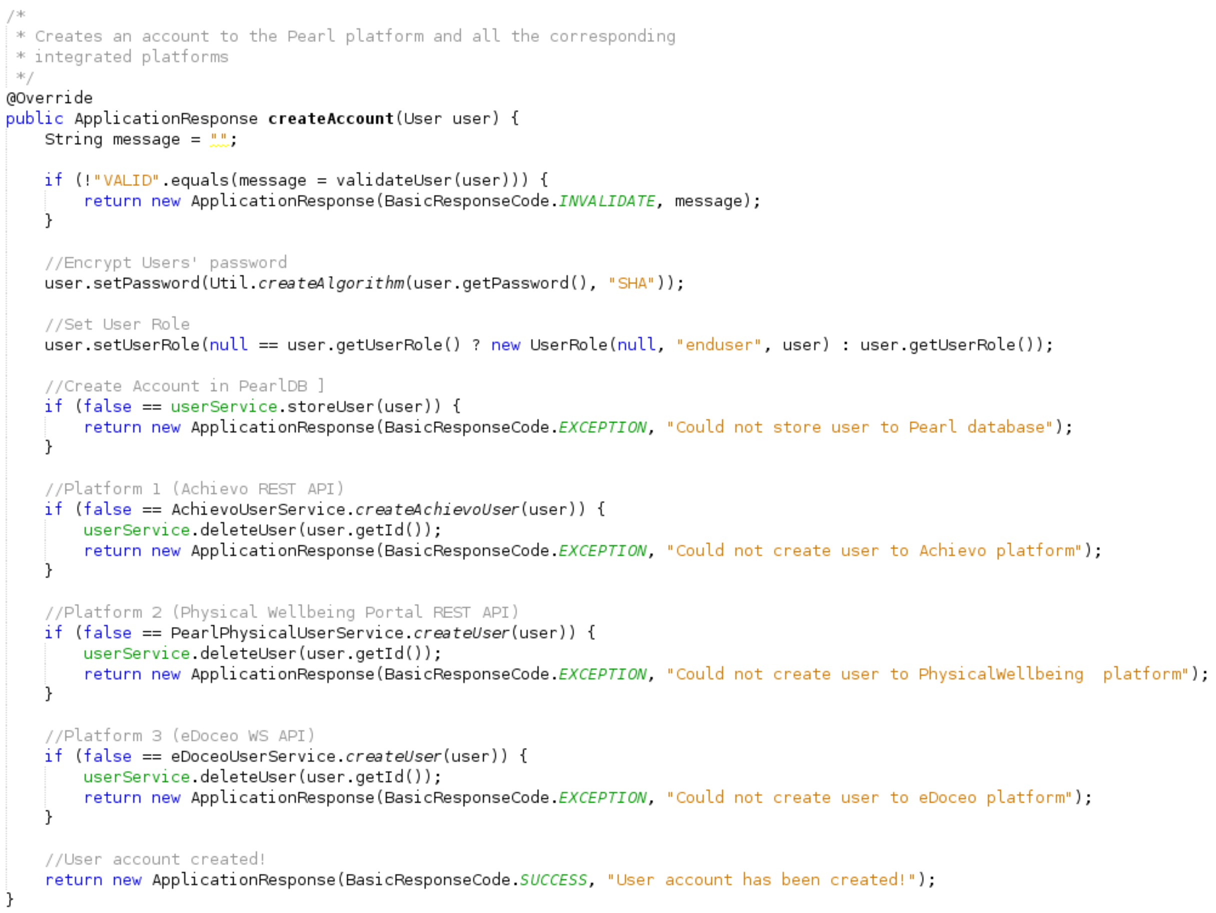


Figure 13. User Authentication Flowchart Diagram

The PEARL platform registration process code snippet is also described in the following figure:

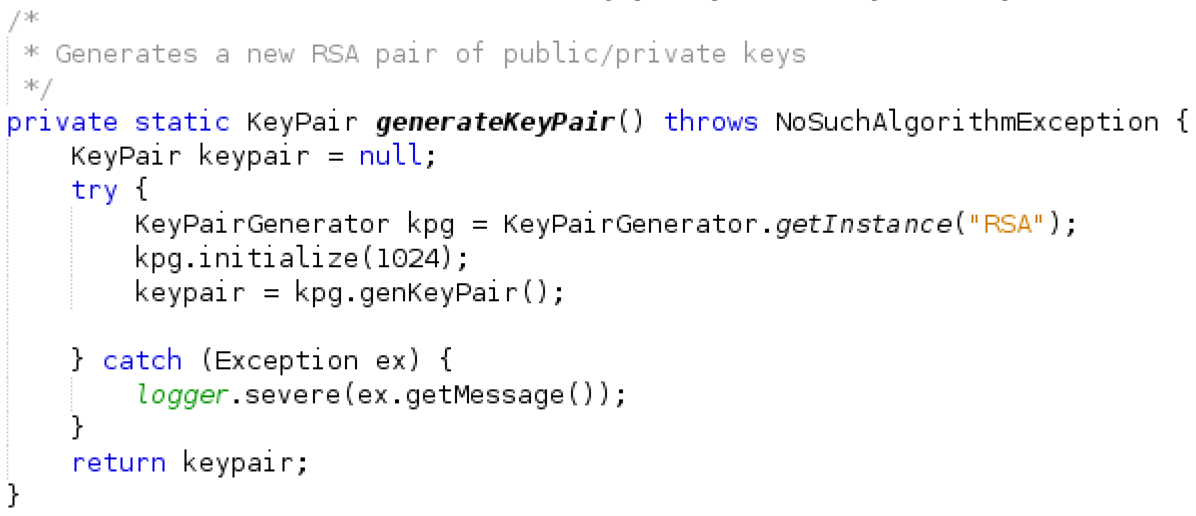


```java
/*
 * Creates an account to the Pearl platform and all the corresponding
 * integrated platforms
 */
@Override
public ApplicationResponse createAccount(User user) {
    String message = "";

    if (!"VALID".equals(message = validateUser(user))) {
        return new ApplicationResponse(BasicResponseCode.INVALIDATE, message);
    }

    //Encrypt Users' password
    user.setPassword(Util.createAlgorithm(user.getPassword(), "SHA"));

    //Set User Role
    user.setUserRole(null == user.getUserRole() ? new UserRole(null, "enduser", user) : user.getUserRole());

    //Create Account in PearlDB ]
    if (false == userService.storeUser(user)) {
        return new ApplicationResponse(BasicResponseCode.EXCEPTION, "Could not store user to Pearl database");
    }

    //Platform 1 (Achievo REST API)
    if (false == AchievoUserService.createAchievoUser(user)) {
        userService.deleteUser(user.getId());
        return new ApplicationResponse(BasicResponseCode.EXCEPTION, "Could not create user to Achievo platform");
    }

    //Platform 2 (Physical Wellbeing Portal REST API)
    if (false == PearlPhysicalUserService.createUser(user)) {
        userService.deleteUser(user.getId());
        return new ApplicationResponse(BasicResponseCode.EXCEPTION, "Could not create user to PhysicalWellbeing  platform");
    }

    //Platform 3 (eDoceo WS API)
    if (false == eDoceoUserService.createUser(user)) {
        userService.deleteUser(user.getId());
        return new ApplicationResponse(BasicResponseCode.EXCEPTION, "Could not create user to eDoceo platform");
    }

    //User account created!
    return new ApplicationResponse(BasicResponseCode.SUCCESS, "User account has been created!");
}
```

Figure 14. PEARL platform registration process

In addition, we also provide the description of the PEARL Platform Public-key Cryptosystem used for securing the information exchange.

4.1.1 Generation of an RSA Key pair (one time process)



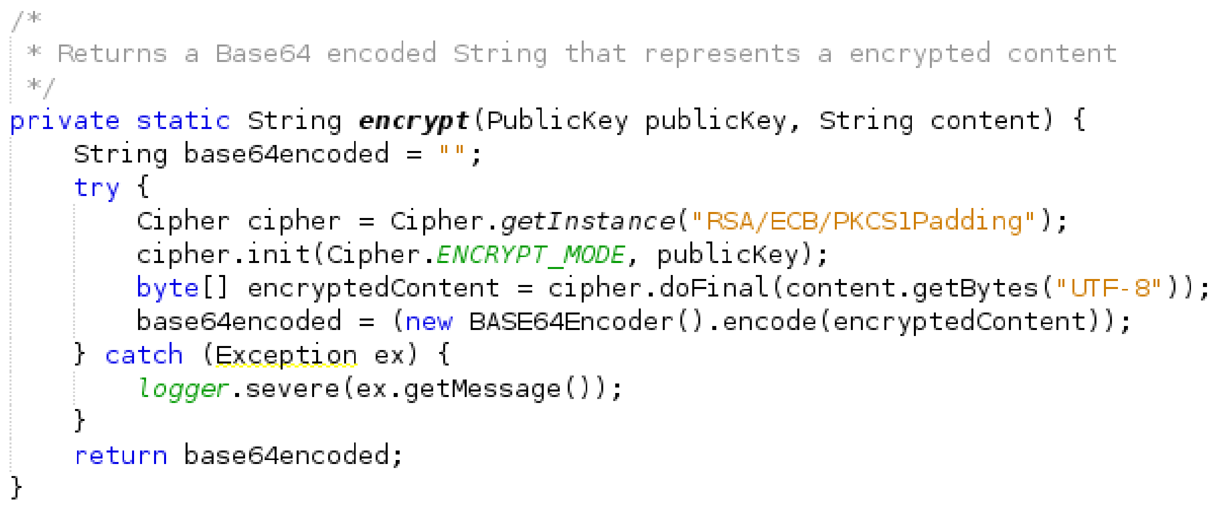
```java
/*
 * Generates a new RSA pair of public/private keys
 */
private static KeyPair generateKeyPair() throws NoSuchAlgorithmException {
    KeyPair keypair = null;
    try {
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(1024);
        keypair = kpg.genKeyPair();

    } catch (Exception ex) {
        logger.severe(ex.getMessage());
    }
    return keypair;
}
```

Figure 15. Generation of an RSA Key pair

The encryption process is a three-step process. The first step involves the generation of a new pair of assymetric RSA (Rivest-Shamir-Adleman) keys, a public and a private one, based on the RSA cryptosystem, one of which is used by the RFID infrastructure for the encryption of the information sent to the PEARL platform (including the TagID and the public IP of the PC of the user), and the other is used by the PEARL platform in order to decrypt the received information. The generation of the RSA Key pair is illustrated in Figure 15.

4.1.2 Encrypt content using the public key

Section 4.1.1 describes the process for the generation of the RSA Key pair. Figure 16 below illustrates the process employed for the actual encryption of the information sent to the PEARL platform (including the TagID and the public IP of the PC of the user), and for safeguarding the validity of the content.

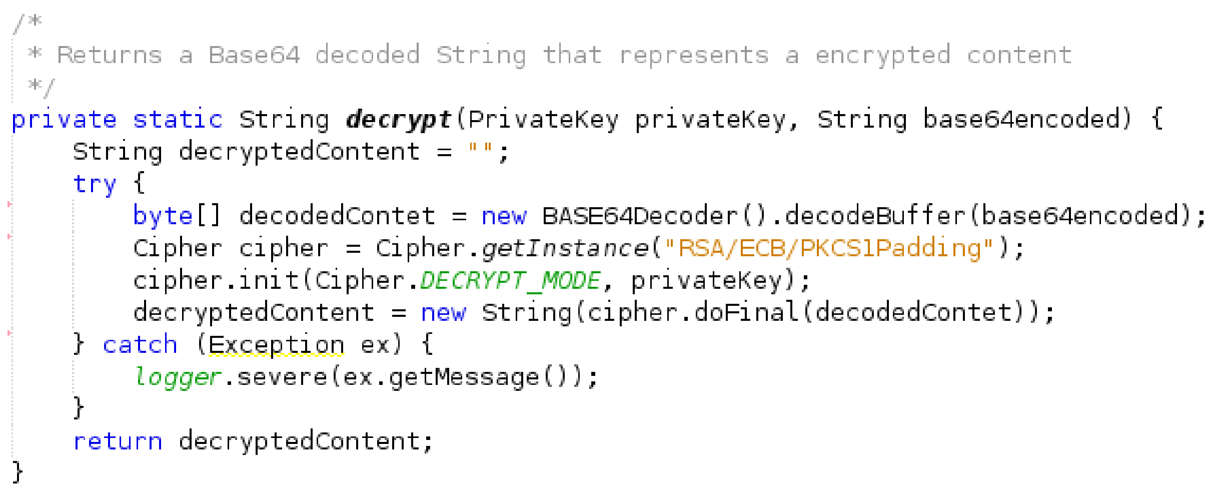


```
/*
 * Returns a Base64 encoded String that represents a encrypted content
 */
private static String encrypt(PublicKey publicKey, String content) {
    String base64encoded = "";
    try {
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedContent = cipher.doFinal(content.getBytes("UTF-8"));
        base64encoded = (new BASE64Encoder().encode(encryptedContent));
    } catch (Exception ex) {
        logger.severe(ex.getMessage());
    }
    return base64encoded;
}
```

Figure 16. Content encryption using the public key

4.1.3 Decrypt content using the private key

Section 4.1.1 describes the process for the generation of the RSA Key pair. Figure 17 below illustrates the process employed for the actual decryption of the information received by the PEARL platform (including the TagID and the public IP of the PC of the user).



```
/*
 * Returns a Base64 decoded String that represents a encrypted content
 */
private static String decrypt(PrivateKey privateKey, String base64encoded) {
    String decryptedContent = "";
    try {
        byte[] decodedContet = new BASE64Decoder().decodeBuffer(base64encoded);
        Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        decryptedContent = new String(cipher.doFinal(decodedContet));
    } catch (Exception ex) {
        logger.severe(ex.getMessage());
    }
    return decryptedContent;
}
```

Figure 17. Content decryption using the private key

4.2 Single Sign-On mechanism for cross-authentication

Within the context of the implementation of the PEARL platform, and based on the need to facilitate seamless login for the PEARL users throughout the various platforms integrated in the holistic PEARL platform, the project consortium has implemented a single sign-on mechanism. This mechanism allows the user to login to the platform once, either using his/her RFID card, or his/her platform credentials (user name and password), and have access to all of the integrated platforms (currently the Task and Time Management module and RRD's Physical Wellbeing Platform), without requiring additional logins. The idea behind the SSO mechanism is that upon a user registration via the PEARL platform, an identical user with the exact credentials (username, password) is created among the integrated platforms (Task and Time Management and Physical Wellbeing Portal) via the provided APIs of each one. The credentials are embedded within the link of each integrated platform, thus when a user is logged-in to the PEARL platform and clicks a link which opens a page on an integrated platform, no additional credentials are required. It must be noticed that for security purposes the credentials which are embedded within the link of each integrated platform are protected using a secure hashing algorithm (SHA-1).

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST. SHA-1 produces a 160-bit (20-byte) hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long. SHA-1 is a member of the Secure Hash Algorithm family. The four SHA algorithms are structured differently and are named SHA-0, SHA-1, SHA-2, and SHA-3. SHA-0 is the original version of the 160-bit hash function published in 1993 under the name SHA: it was not adopted by many applications. Published in 1995, SHA-1 is very similar to SHA-0, but alters the original SHA hash specification to correct weaknesses that were unknown to the public at that time. SHA-2, published in 2001, is significantly different from the SHA-1 hash function[3].

In the following table, a sample „Create User Request" and „Create User Response" are illustrated. Within the context of PEARL, one API (preferably RESTful API following JSON format for the requests and responses) per platform integrated is required.

{
   "username":"user1",
   "password":"Fffew#!2eD33aa",
   "firstName":"Pearl",
   "lastName":"User",
   "email":"testuser@pearl-platform"
}
	{
	   "status":"OK",
	   "data":{
	      "username":"user1",
	      "lastname":"User",
	      "firstname":"Pearl",
	      "email":"testuser@pearl-platform",
	      "id":"10"
	   }
	}

Sample Create User Request	**Sample Create User Response**

Table 1. Sample Create User Request & Response

[3] https://en.wikipedia.org/wiki/SHA-1

To illustrate the implementation across the various application modules the RESTful web services for account creation and deletion of the Task and Time Management module are presented in Tables 2 and 3 and of the Physical Wellbeing module in Tables 4 and 5.

a) Create new user (Task and Time Management module)

Method	**POST** *{root}/ api/user*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username assigned to the application that issues the REST call.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**

addNewUser	{
	"userid":"the username of the new user",
	"lastname":"…",
	"firstname":"…",
	"email":"…",
	"password":"…"
	}
		Status code, Object *$person*	The input is sent in the body of the request in JSON format.

Response representations			
Response code	**Response message**	**Response body**	

201	Created	{
		"status":"OK",
		"data":{
		"userid":"cska",
		"lastname":"john",
		"firstname":"rest2",
		"email":"john@pear.com",
		"password":"john","id":"10"
		}
		}	

409	Conflict	{	

		"status":"ERROR", "messages":{ … } }

Table 2. PEARL Task and Time Management Create User Request and Response

b) Delete existing user (Task and Time Management module)

Method	**POST** *api/deleteuser/{userid}*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username assigned to the application that issues the REST call.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
deleteUser	String userID	Status code	
Response representations			
Response code	**Response message**	**Response body**	
200	OK	{"status":"OK"}	
409	Conflict	{ "status":"ERROR", "messages":{ … } }	

Table 3. PEARL Task and Time Management Create User Request & Response

c) Create new user (Physical Well-Being Module)

Method	**POST** *{websiteroot}/api/json-rpc.php*		
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**

adduser	{
	"id": request ID,
	"method": "adduser",
	"jsonrpc": "2.0",
	"params": {
	"username": …,
	"securityToken": <fixed token>,
	"firstname": …,
	"lastname": …,
	"email": …,
	"password": …,
	"lang": <language code>
	}
	}
		Status (success/ error)	Follows JSON-RPC 2.0 standard.

Response representations			
Response code	**Response message**	**Response body**	

201	Success	{
		"jsonrpc": "2.0",
		"id": "request ID,
		"result": {
		code: "SUCCESS",
		message: "User added"
		}
		}	

201	Error	{
		"jsonrpc": "2.0",
		"id": "request ID,
		"error": {
		code: <numeric code>,
		message: <error message>
		}
		}
		Error code and message can be one of:
		1: No method defined
		2: No id (request ID) defined
		3: Authentication failed
		4: User already exists
		5: User does not exist
		6: Unknown method …
		7: Parameter … not defined	

Table 4. PEARL Physical Well-Being Layer, Create User Request and Response

d) Delete existing user (Physical Well-Being Module)

Method	**POST** *{websiteroot}/api/json-rpc.php*		
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
deleteuser	{ "id": request ID, "method": "deleteuser",	Status (success/ error)	Follows JSON-RPC 2.0 standard.

	“jsonrpc”: “2.0”,
	“params”: {
	“username”: …,
	“securityToken”: <fixed token>,
	}
	}		

Response representations		
Response code	**Response message**	**Response body**

201	Success	{
		“jsonrpc”: “2.0”,
		“id”: “request ID,
		“result”: {
		code: “SUCCESS”,
		message: “User deleted”
		}
		}

201	Error	{
		“jsonrpc”: “2.0”,
		“id”: “request ID,
		“error”: {
		code: <numeric code>,
		message: <error message>
		}
		}
		See adduser for a list of error codes and messages.

Table 5. PEARL Physical Well-Being Layer, Delete User Request and Response

4.3 Configurations retrieval and deployment

The process of retrieving and deploying the relevant configuration plans varies depending on whether the user is already registered to the platform or he/she is a new user, being introduced to the platform services for the first time. As a result some of the components, such as the DSS, are invoked only when a new registration is taking place. In the current chapter the major APIs involved in the process of configuration retrieval and deployment will be described.

4.3.1 Registration

As the transparent account creation over the various application modules was already described in the previous sections, here we will focus on the DSS-enabled retrieval of a customized configuration plan upon first login, based on the user profile characteristics.In order to comply with the requirements of the other PEARL platform modules, the PEARL DSS was designed and implemented as a RESTful web service. Spring annotation-based Model-View-Controller (MVC) framework[4] was chosen for the process of implementation. For further details regarding the underlying technologies and implementation logic, please refer to “*D4.4.2 Decision Support System and Rule Engines”*.

[4] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

PEARL DSS web service is called by the Preference Editor upon the completion of the User Profile Characteristics by the newly registered user. The *userID* of the user is sent as a GET request URL query string parameter. The PEARL DSS web service details are presented in table 6 below.

Method	**GET** *{root}/pearl_bil/rest/retrieveconf/{user_id}*		
Headers	Content-Type	application/json	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
retrieveConfPlanByUserId	Integer *user_id*	ConfSetEntityResource *res*	The output will be a success/error message/code and JSON representation of the configuration plan. *user_id* is the ID of the newly created user profile. *res* is the REST representation of the *ConfigurationSettingsEntity* class, created using Spring HATEOS resource assembler.
Response representations			
Response code	**Response message**	**Response body**	

200	OK	{
		"user_id": Integer,
		"tasksSettings": [
		{
		"taskSettingsMap": {
		"taskName": String,
		"soundLevel": String,
		"screenResolution": String,
		"roomLightIntensity": String,
		"deskLightIntensity": String,
		"lightTemperature": String,
		"defaultTask": String,
		"preferredSoftware": String,
		"availabilityIndicator": String,
		"taskWorkspace": Integer
		}
		},
		{	

		"taskSettingsMap": {
		          "taskName": String,
		          "soundLevel": "low",
		          "screenResolution": String,
		          "roomLightIntensity": String,
		          "deskLightIntensity": String,
		          "lightTemperature": String,
		          "preferredSoftware": String,
		          "availabilityIndicator": String,
		          "taskWorkspace": Integer
		        }
		      },
		      {
		        "taskSettingsMap": {
		          "taskName": String,
		          "availabilityIndicator": String,
		          "preferredSoftware": String
		        }
		      }
		    ],
		    "links": []
		}

404	NOT_FOUND	-

Table 6. PEARL DSS REST API definition

4.3.2 Preference Editing

As described in section 3.4 above, the user has the possibility to modify any of the configuration settings via the Preference editor. Depending on the parameter that has been changed either the Ambient Tuning Core or the Configurations Manager will be responsible for the deploying the new settings and therefore will be triggered by the Preference Editor. If the altered settings concern the ambient environment configurations or the list of the available tasks, the Ambient Tuning Core will be triggered via the Trigger Manager. The relevant task switcher and trigger manager APIs are presented in tables 9, 10, 11 and 13, 14, 15 respectively.

If the changes involve any of the other application layer modules, the deployment will be undertaken by the Configurations Manager. It will be triggered by the preference editor by sending a REST call, containing the corresponding *user_id* and the ID of the relevant application layer module. The configuration manager will then undertake the task of retrieving the configuration from the database and forwarding them to the module of interest. The web service that will be used for triggering the Configurations Manager is described in table 7 below. It should be noted that this service is currently not utilized due to limitation of the application layer modules.

Method	**GET** *{root}/pearl_bil/rest/setconfigurations/{user_id}/{module_id}*

Headers	Content-Type		application/json
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
setModuleConfigurations	String *user_id*, String *module_id*	Status code	The output will be a success/error message/code. *user_id* is the ID of the user. *module_id* indicates, which module is influenced by the change of the configurations.
Response representations			
Response code	**Response message**	**Response body**	
200	OK		
404	NOT_FOUND	-	

Table 7. PEARL Configurations Manager REST API definition

4.3.3 Task Switcher Manager API

An overview of the Task Switching API API is provided in Table 8 below. The other tables below describe each operation as a RESTful Web Service.

<<interface>> TaskSwitcherManagerInterface --- taskSwiched(userID:String, taskID:String, workspaceID:String): String userLogIn(tagID:String, workspaceID:String): String userWebLogIn (userID:String, workspaceID:String): String

Table 8. List of primitives comprising the Switcher Manager API

Method	**GET** *{root)/taskSwicherManager/taskSwitched/{userId}/{taskId}/{workspaceId}*		
Headers	Content-Type		text/plain
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**

taskSwitched	String userID, String taskID, String workspace ID	String	This service is used whenever a new task is chosen from the User. It takes as input the User ID, the chosen Task ID and the Workspace ID that the User has made the choice from. The output will be a success/error message/code. The workspaceID is the IP of the local machine..
Response representations			
Response code	**Response message**	**Response body**	
200	0		
404	NOT_FOUND	-	

Table 9. Switcher Manager Task Switched API definition

Method	**GET** *{root)/taskSwicherManager/userLogIn/{tagId}/{workspaceId}*		
Headers	Content-Type	text/plain	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
userLoggedIn	String tagID, String workspace ID	String	This service is used whenever a User logs in to the system by using his RFID card. It takes as input the User's tag ID, which is provided from the RFID reader, and the Workspace ID that the User has logged in from. The output will be a success/error message/code. The workspaceID is the IP of the local machine
Response representations			
Response code	**Response message**	**Response body**	
200	0		

404	NOT_FOUND	-

Table 10. Switcher Manager Tag User Login API definition

Method	**GET** *{root)/taskSwicherManager/userWebLogIn/{userId}/{workspaceId}*		
Headers	Content-Type	text/plain	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
userWebLogIn	String tagID, String workspace ID	String	This service is used whenever a User logs in to the system by using the web interface of the platform. It takes as input the User ID and the Workspace ID that the User has logged in from. The output will be a success/error message/code. The workspaceID is the IP of the local machine.
Response representations			
Response code	**Response message**	**Response body**	
200	0		
404	NOT_FOUND	-	

Table 11. Switcher Manager Web User Login API definition

4.3.4 Trigger Manager API

The Trigger Manager API is provided in Table 12. The other tables in this section provide the description of each operation as a RESTFul Web Service.

<<interface>>
TriggerManagerInterface
---
taskSwiched(userID:String, taskID:String):String
preferenceUpdated(userID:String, taskID:String): String
userLoggedIn(userID:String): String

Table 12. List of primitives comprising the Trigger Manager API

Method	**GET** *{root)/pearl-trigger-manager/triggerManager/taskSwitched/{userId}/{taskId}*		
Headers	Content-Type		text/plain
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
taskSwiched	String userID, String taskID	String	This service is used to inform the PEARL components whenever a User has switched a task. It takes as input the User ID and the task ID that the User has switched to. The output will be a success/error message/code
Response representations			
Response code	**Response message**	**Response body**	
200	OK		
404	NOT_FOUND	-	

Table 13. Trigger Manager Task Switched API definition

Method	**GET** *{root)/pearl-trigger-manager/triggerManager/preferenceUpdated/{userId}/{taskId}*		
Headers	Content-Type		text/plain
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
preferenceUpdated	String userID, String taskID	String	This service is used to inform the PEARL components whenever a User has updated one of the platform preferences. It takes as input the User ID and the task ID that the User is currently using. The output will be a

			success/error message/code

Response representations		
Response code	**Response message**	**Response body**
200	OK	
404	NOT_FOUND	-

Table 13. Trigger Manager Preference Updated API definition

Method	**GET** *{root)/pearl-trigger-manager/triggerManager/* userLoggedIn/*{userId}*		
Headers	Content-Type	text/plain	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
userLoggedIn	String userID, String taskID	String	This service is used to inform the PEARL components whenever a User has logged in to the system. It takes as input the ID of the user that has logged in. The output will be a success/error message/code

Response representations		
Response code	**Response message**	**Response body**
200	OK	
404	NOT_FOUND	-

Table 14. Trigger Manager User Logged-In API definition

4.3.5 Message Broker API

The MessageBroker API definition is provided in the tables below.

<<subclass>> CustomStompSessionHandler ---

void afterConnected(StompSession session, StompHeaders connectedHeaders)

Table 15. List of public methods exposed in MessageBroker API

Service Name	Input	Output	Info
afterConnected	StompSession session, StompHeaders connectedHeaders	void	A contract for client STOMP session lifecycle events including a callback when the session is established and notifications of transport or message handling failures. Invoked when the session is ready to use.

Table 16. MessageBroker API definition

<<CLASS>> AbstractWebSocketMessageBrokerConfigurer – void registerStompEndpoints(StompEndpointRegistry registry) <<CLASS>> WebController --- void ambientConfigManager(@PathVariable(value = "workspaceid") String workspaceid, @RequestBody String requestBody) void workspaceConfigManager(@PathVariable(value = "workspaceid") String workspaceid, @RequestBody String requestBody)

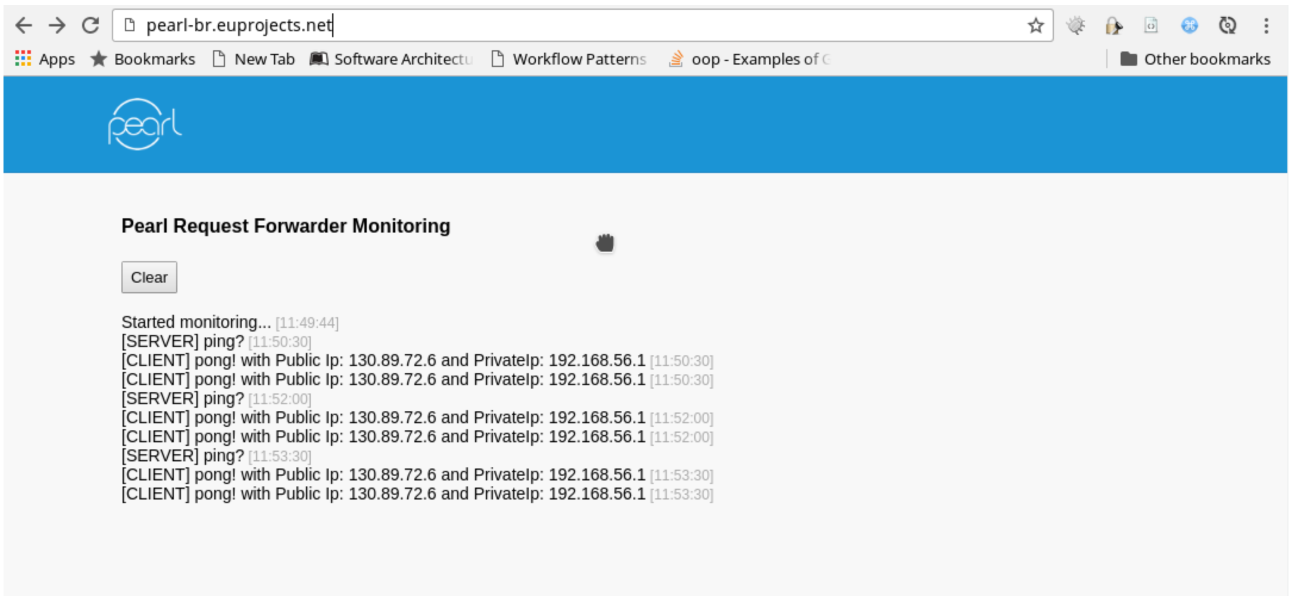
Table 17. List of public methods exposed in MessageBroker REST API

WebService Name	Protocol	URL	Info
registerStompEndpoints	WebSocket	ws://pearl-br.euprojects.net/stomp/	The exposable public endpoint of the MessageBroker. STOMP clients can subscribe to

			broker via this endpoint.
ambientConfigManager	HTTP (PUT)	http://pearl-br.euprojects.net/api/v1/ambient/	The exposable public endpoint which SENSAP webservice invokes on ambient configuration trigger.
workspaceConfigManager	HTTP (PUT)	http://pearl-br.euprojects.net/api/v1/workspace/	The exposable public endpoint which SENSAP webservice invokes on workspace configuration trigger.

Table 18. MessageBroker REST API definition

In addition to the Broker API, in the figure below we also provide the PEARL Message Broker monitoring UI.



Pearl Request Forwarder Monitoring

```
Started monitoring... [11:49:44]
[SERVER] ping? [11:50:30]
[CLIENT] pong! with Public Ip: 130.89.72.6 and PrivateIp: 192.168.56.1 [11:50:30]
[CLIENT] pong! with Public Ip: 130.89.72.6 and PrivateIp: 192.168.56.1 [11:50:30]
[SERVER] ping? [11:52:00]
[CLIENT] pong! with Public Ip: 130.89.72.6 and PrivateIp: 192.168.56.1 [11:52:00]
[CLIENT] pong! with Public Ip: 130.89.72.6 and PrivateIp: 192.168.56.1 [11:52:00]
[SERVER] ping? [11:53:30]
[CLIENT] pong! with Public Ip: 130.89.72.6 and PrivateIp: 192.168.56.1 [11:53:30]
[CLIENT] pong! with Public Ip: 130.89.72.6 and PrivateIp: 192.168.56.1 [11:53:30]
```

Figure 18. PEARL Message Broker monitoring UI.

4.4 Caledula APIs

The following set of APIs is specific for the Task and Time Management module. In relation to the development of the digital paper analogue user interface – Calendula (see D2.3.2 and D3.2.2) – a full set of RESTful web services has been developed to allow full control over the Task and Time Management module functionalities. Calendula will then work autonomously and retrieve data from the backend on demand to reduce management complexity on the backend. The developed web services can be divided into two main categories – web services for calendar events management and web services for tasks management. The details about all of the developed programmable interfaces will be described in the following sections.

4.4.1 Calendar Management APIs

The Calendar Management APIs include methods for events insertion, update and deletions, as well as methods for retrieving a list of available users, inviting new users to existing events, responding to and cancelling invitations.

1) Retrieve all events for a given time period

Method	**GET** *{root}/api/events/{owner}/{start_date}/{end_date}*	
Headers	Content-Type	application/json
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.
	User	Username of a registered PEARL user.
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.

API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
retrieveEvents	{owner} – the userID of the user of interest {start_date} – the starting date of the period of interest {end_date} - the end date of the period of interest	Status code, Object *$event*	The input is sent as URL encoded variables.

Response representations		
Response code	**Response message**	**Response body**

200	OK	{
		"eventid":"2ls4lms77l65fae8n3u2d7m7e0",
		"text":"organio rest test 1",
		"start_time":"2016-02-02 11:35:00",
		"end_time":"2016-02-02 12:35:00",
		"participants":"pearl"
		}

409	Conflict	{
		"status":"ERROR",
		"messages":{ … }
		}

Table 19. PEARL Task and Time Management Retrieve Calendar Events

2) Add new calendar event

Method	**POST** *{root}/api/events*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username of a registered PEARL user.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
addNewEvent	{ "userid": "pearl", "start_date": "2016-02-02 11:35", "end_date": "2016-02-02 12:35", "text": "organio rest test 1", “private”: “1” *(optional)* }	Status code, ID of the newly Created event.	The input is sent in the body of the request in JSON format.
Response representations			
Response code	**Response message**	**Response body**	
201	Created	{ "status":"OK", "eventid": “…” }	
409	Conflict	{ "status":"ERROR", "messages":{ … } }	

Table 20. PEARL Task and Time Management Add New Calendar Event

3) Update existing event

Method	**POST** *{root}/api/changeevent/{eventid}*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username of a registered PEARL user.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**

updateEvent	{eventid} – ID of the event that will be updated
	{
	"userid": "pearl",
	"start_date": "2016-02-02 11:35",
	"end_date": "2016-02-02 12:35",
	"text": "organio rest test 1",
	“private”: “1” *(optional)*
	}
		Status code	The *eventid* is sent as URL encoded variables.
			The input is sent in the body of the request in JSON format.

Response representations			
Response code	**Response message**	**Response body**	
200	Updated	{ "status":"OK"}	

409	Conflict	{
		"status":"ERROR",
		"messages":{
		…
		}
		}	

Table 21. PEARL Task and Time Management Update Calendar Event

4) Delete existing event

Method	**POST** *{root}/api/deleteevent/{userid}/{eventid}*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username of a registered PEARL user.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
deleteEvent	{userid} – the userID of the owner of the event {eventid} – the ID of the event that will be deleted	Status code, Object *$event*	The input is sent as URL encoded variables.
Response representations			
Response code	**Response message**	**Response body**	
200	OK	{ "status":"OK"}	

409	Conflict	{
		"status":"ERROR",
		"messages":{
		…
		}
		}	

422	Unprocessable Entry	{
		"status":"ERROR",
		"messages":{
		"Unsuccessful deletion! Please check your input parameters and try again! If the problem persists, please contact your PEARL administrator!"
		}	

Table 22. PEARL Task and Time Management Delete Calendar Event

5) Retrieve all registered users

Method	**GET** *{root}/api/listofusers*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username of the user registered to the PEARL platform.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
retrieveListOfUsers		Status code, Array *$users*	
Response representations			
Response code	**Response message**	**Response body**	

200	OK	{
		"data":{
		"userid":"cska",
		"name":"John Smith"
		},
		…
		}	

Table 23. PEARL Task and Time Management Retrieve All Registered Users

6) Invite a user to an event

Method	**POST** *{root} /api/respondtoinvitation/{eventid}*	
Headers	Content-Type	application/json
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.
	User	Username of a registered PEARL user.
	Hash	SHA256 hash generated by the combination of

		the current time stamp, the username and the secret password.

API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
inviteUserToEvent	{eventid} – ID of the event, for which the invitation will be sent {userid} – ID of the user that will be invitated	Status code	The *eventid* is sent as URL encoded variables.

Response representations		
Response code	**Response message**	**Response body**
201	Invited	{ "status":"OK"}

409	Conflict	{
		"status":"ERROR",
		"messages":{
		…
		}
		}

Table 24. PEARL Task and Time Management - Invite User to an Existing Calendar Event

7) Respond to invitation

Method	**POST** *{root}/api/respondtoinvitation/{eventid}*	
Headers	Content-Type	application/json
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.
	User	Username of a registered PEARL user.
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.

API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
respondToInvitation	{eventid} – ID of the event of interest {	Status code	The *eventid* is sent as URL encoded variable. The *userid* and the

	"userid": "pearl", "status": "tentative" }		*status* are sent in the body of the request in JSON format.

Response representations		
Response code	**Response message**	**Response body**
201	OK	{ "status":"OK"}
409	Conflict	{ "status":"ERROR", "messages":{ … } }

Table 25. PEARL Task and Time Management – Respond to Invitation to an Existing Calendar Event

8) Cancel invitation

Method	**POST** *{root} /api/cancelinvitation/{eventid}/{userid}*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username of a registered PEARL user.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
cancelInvitation	{eventid} – ID of the event of interest {userid} – ID of the user that will be removed	Status code	The *eventid and userid* are sent as URL encoded variables.
Response representations			
Response code	**Response message**	**Response body**	
201	OK	{ "status":"OK"}	

409	Conflict	{ "status":"ERROR", "messages":{ … } }

Table 26. PEARL Task and Time Management – Cancel Invitation to an Existing Calendar Event

4.4.2 Task Management APIs

The Task Management APIs include methods for communication with the PEARL task management application, including tasks retrieval per user, adding, deleting and completing tasks.

1) Retrieve list of open tasks

Method	**GET** *{root}/api/tasks/{userid}*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username of a registered PEARL user.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
retrieveEvents	{userid} – the userID of the user of interest	Status code, Object *$tasks*	The input is sent as URL encoded variable.
Response representations			
Response code	**Response message**	**Response body**	
200	OK	{ "taskid":"46", "task_title":"new test item 1", "task_category":"test", "task_description":"test 1", "priority":"high", "due_date":"2016-02-12" }	

409	Conflict	{ "status":"ERROR", "messages":{ … } }

Table 27. PEARL Task and Time Management Retrieve Tasks

2) Add new task

Method	**POST** *{root}/api/tasks*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username of a registered PEARL user.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
addNewTask	{ "userid":"pearl", //required "task_title":"rest test 1", //optional "task_category":"api", //optional "task_description":"test 1", //optional "priority":"high", //optional "due_date":"2016-02-12" //optional }	Status code, ID of the newly created task.	The input is sent in the body of the request in JSON format.
Response representations			
Response code	**Response message**	**Response body**	
201	Created	{ "status":"OK", "taskid": "…" }	

409	Conflict	{ "status":"ERROR",
		"messages":{ … }
		}

Table 28. PEARL Task and Time Management Add New Task

3) Update existing task

Method	**POST** *{root}/api/changetask/{taskid}*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username of a registered PEARL user.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**

updateTask	{taskid} – ID of the task that will be updated
	{
	"userid":"pearl", //required
	"task_title":"rest test 1", //optional
	"task_category":"api", //optional
	"task_description":"test 1", //optional
	"priority":"high", //optional
	"due_date":"2016-02-12" //optional
	}
		Status code	The *taskid* is sent as URL encoded variables.
			The input is sent in the body of the request in JSON format.

Response representations

Response code	**Response message**	**Response body**
200	Updated	{ "status":"OK"}

409	Conflict	{
		"status":"ERROR",
		"messages":{
		…
		}

		}

Table 29. PEARL Task and Time Management Update Existing Task

4) Complete existing task

Method	**POST** *{root}/api/completetask/{taskid}*		
Headers	Content-Type	application/json	
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.	
	User	Username of a registered PEARL user.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
completeTask	{taskid} – ID of the task that will be completed	Status code	The *taskid* is sent as URL encoded variables.
Response representations			
Response code	**Response message**	**Response body**	
200	Completed	{ "status":"OK"}	
409	Conflict	{ "status":"ERROR", "messages":{ … } }	

Table 30. PEARL Task and Time Management Complete Existing Task

5) Delete Task

Method	**POST** *{root}/api/deletetask/{taskid}*	
Headers	Content-Type	application/json
	Time	Unix timestamp used to generate the HMAC hash, expires after 10 min.

	User	Username of a registered PEARL user.	
	Hash	SHA256 hash generated by the combination of the current time stamp, the username and the secret password.	
API definition and request query parameters			
Service Name	**Input**	**Output**	**Info**
deleteTask	{taskid} – ID of the task that will be deleted	Status code	The *taskid* is sent as URL encoded variables.
Response representations			
Response code	**Response message**	**Response body**	
200	OK	{ "status":"OK"}	

409	Conflict	{
		"status":"ERROR",
		"messages":{
		…
		}
		}	

Table 31. PEARL Task and Time Management Delete Existing Task

5 Summary and Conclusions

The distributed nature of the PEARL application layer modules has posed a number of challenges in terms of reliable and secure exchange of configuration and application data, fluent and uninterrupted user access to the variety of PEARL services and provision of customized user experience. In order to address the aforementioned challenges and to enable seamless inclusion of additional application layer in the future we have developed a set of configuration management tools and programmable interfaces, such as transparent cross-platform account creation and user registration, single sign-on cross authentication, DSS for automated configuration plan customization, message broker application for automated and robust mapping of mismatching parameters at the local client agents, reliable interfaces and mechanisms for configuration retrieval and deployment and secure application data transfer. The current document described the major configurations management capabilities of the platform by presenting the relevant functionalities of the underlying platform components in the context of four major scenarios, thus aiming to fully illustrate the adaptability of the holistic PEARL platform. To further elaborate on the configurations management process, we presented the major data flows related to each of the scenarios and we have offered detailed description of the programmable interfaces, developed for inter-module data exchange. The purpose of this deliverable is to serve as a summary of the implemented programmable interfaces and configuration management tools, which will enable the future development of extended functionalities and will facilitate the inclusion of third-party application layer modules.