Deliverable 3.1
Integrated INLIFE Open Framework Release R1.0
Abstract

This document describes the first release of the InLife Open Framework, providing details of the key components belonging to this release and all the communication data streams that link them together. The first release of the InLife platform provides all the key features needed to create and configure the gameplay of a serious game, personalizing the game experience according to the users’ needs and blending the in-game mechanics with real-life events coming from the integration with the IoT platform.

In addition, the document describes the iterative development process, including the linking of the basic building blocks, tests and validation of the deployment scenario of the architecture, and further steps needed to reach the first key milestone in the InLife architecture development: the alpha release with all the components and all the key integration tasks addressed.

This document also lists a set of links to access the components that provide a direct user access point and a section with pictures of the key graphical user interface screens involved in the use cases addressed by this release of the platform. As an appendix placed at the end, the document provides all the detailed technical information about the InLife components that have been developed and integrated so far, broadly described in the previous sections of the document.
Legal Disclaimer

This document contains confidential information in the form of the InLife project findings, work and products and its use is strictly regulated by the InLife Consortium Agreement and by Contract no. H2020 - ICT-732184.

Neither the InLife Consortium nor any of its officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein. The research leading to these results has received funding from the European Union’s Horizon 2020 Programme (H2020-ICT-24-2016) under grant agreement n° 732184. The contents of this document are the sole responsibility of the InLife consortium and can in no way be taken to reflect the views of the European Union.

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

This work is partially funded by the EU under grant ICT-H2020-732184
<table>
<thead>
<tr>
<th>DATE</th>
<th>COMMENT</th>
<th>REVISION</th>
<th>AUTHOR/EDITOR</th>
<th>AFFILIATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>04.07.2017</td>
<td>Frist draft and table of content</td>
<td>01</td>
<td>M. Cazzaniga</td>
<td>IMA</td>
</tr>
<tr>
<td>20.08.2017</td>
<td>Section 3.3</td>
<td>02</td>
<td>M. Cazzaniga, V. Nitti</td>
<td>IMA</td>
</tr>
<tr>
<td>30.08.2017</td>
<td>Section 3.5, Section 4.1, 4.2, Appendix 9.1, Appendix 9.2.3</td>
<td>03</td>
<td>S. Karachonztitis, Y. Oikonomidis</td>
<td>SYN</td>
</tr>
<tr>
<td>31.08.2017</td>
<td>Section 3.6 contributions to appendix 4.2 and appendix 9.2</td>
<td>04</td>
<td>B. Marcon</td>
<td>TRT</td>
</tr>
<tr>
<td>19.09.2017</td>
<td>Section 3.7</td>
<td>05</td>
<td>S. Morales, V. Ortega</td>
<td>FFM</td>
</tr>
<tr>
<td>20.09.2017</td>
<td>Section 3.2</td>
<td>06</td>
<td>P. Kosmides</td>
<td>ICCS</td>
</tr>
<tr>
<td>20.09.2017</td>
<td>Executive summary, introduction and conclusions</td>
<td>07</td>
<td>M. Cazzaniga</td>
<td>IMA</td>
</tr>
<tr>
<td>22.09.2017</td>
<td>Section 3.4</td>
<td>08</td>
<td>V. Nitti</td>
<td>IMA</td>
</tr>
<tr>
<td>25.09.2017</td>
<td>Appendix 8.4 integration test cards and overall revision of the document</td>
<td>09</td>
<td>M. Cazzaniga</td>
<td>IMA</td>
</tr>
<tr>
<td>01.10.2017</td>
<td>Review</td>
<td>09b</td>
<td>S. Karachonztitis, P. Kosmides</td>
<td>SYN, ICCS</td>
</tr>
<tr>
<td>3/10/2017</td>
<td>Modifications to address technical review feedbacks</td>
<td>10</td>
<td>M. Cazzaniga</td>
<td>IMA</td>
</tr>
<tr>
<td>13/10/2017</td>
<td>Merge of contributions to address the internal review feedback</td>
<td>11</td>
<td>M. Cazzaniga</td>
<td>IMA</td>
</tr>
</tbody>
</table>
# Authors

<table>
<thead>
<tr>
<th>PARTNER</th>
<th>NAME</th>
<th>E-MAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMA</td>
<td>Massimiliano Cazzaniga</td>
<td><a href="mailto:massimiliano.cazzaniga@i-maginary.it">massimiliano.cazzaniga@i-maginary.it</a></td>
</tr>
<tr>
<td></td>
<td>Vito Nitti</td>
<td><a href="mailto:vito.nitti@i-maginary.it">vito.nitti@i-maginary.it</a></td>
</tr>
<tr>
<td>TRT</td>
<td>Bruno Marcon</td>
<td><a href="mailto:Bruno.MARCON@thalesgroup.com">Bruno.MARCON@thalesgroup.com</a></td>
</tr>
<tr>
<td>ICCS</td>
<td>Pavlos Kosmides</td>
<td><a href="mailto:pkosmidis@cn.ntua.gr">pkosmidis@cn.ntua.gr</a></td>
</tr>
<tr>
<td></td>
<td>Konstantinos Demestichas</td>
<td><a href="mailto:cdemest@cn.ntua.gr">cdemest@cn.ntua.gr</a></td>
</tr>
<tr>
<td>FFM</td>
<td>Victor Ortega</td>
<td><a href="mailto:vortega@fiveflamesmobile.com">vortega@fiveflamesmobile.com</a></td>
</tr>
<tr>
<td></td>
<td>Sergio Morales</td>
<td><a href="mailto:smorales@fiveflamesmobile.com">smorales@fiveflamesmobile.com</a></td>
</tr>
<tr>
<td>SYN</td>
<td>Sotiris Karachontzitis</td>
<td><a href="mailto:karachontzitis@synelixis.com">karachontzitis@synelixis.com</a></td>
</tr>
<tr>
<td></td>
<td>Yannis Oikonomidis</td>
<td><a href="mailto:oikonomidis@synelixis.com">oikonomidis@synelixis.com</a></td>
</tr>
</tbody>
</table>
# Table of contents

Executive summary .................................................................................................................. 10  
1. Introduction .............................................................................................................................. 11  
2. Methodology ............................................................................................................................ 12  
2. Our approach ........................................................................................................................... 13  
3. InLife Open architecture ......................................................................................................... 15  
3.1. A2 - Authorization and authentication component ................................................................. 15  
3.1.1. Role in the architecture ....................................................................................................... 16  
3.1.2. Main APIs endpoints and data flows ................................................................................. 16  
3.1.3. Dependencies and modules .............................................................................................. 17  
3.1.4. Deployment ...................................................................................................................... 17  
3.2. User profile database .......................................................................................................... 17  
3.2.1. Role in the architecture ..................................................................................................... 17  
3.2.2. Main APIs endpoints and data flows ............................................................................... 18  
3.2.3. Dependencies and modules ............................................................................................. 18  
3.2.4. Deployment ...................................................................................................................... 18  
3.2.5. Deployment ...................................................................................................................... 21  
3.3. Game configuration engine .................................................................................................... 21  
3.3.1. Role in the architecture ..................................................................................................... 24  
3.3.2. Configuration data structures ........................................................................................... 25  
3.3.3. Game engine SDK ............................................................................................................ 25  
3.3.4. Dependencies and modules ............................................................................................. 27  
3.3.5. Deployment ...................................................................................................................... 28  
3.3.6. User interfaces and user experience ............................................................................. 28  
3.4. Learning Analytics .............................................................................................................. 30  
3.4.1. Role in the architecture ..................................................................................................... 31  
3.4.2. Game data tracking .......................................................................................................... 32  
3.4.3. Deployment ...................................................................................................................... 32  
3.4.4. Analysis and visualizations ............................................................................................. 33  
3.5. IoT component ..................................................................................................................... 34  
3.5.1. Role in the architecture ..................................................................................................... 34  
3.5.1.1. The IoT access API ......................................................................................................... 35  
3.5.1.2. The IoT cloud server ..................................................................................................... 35  
3.5.1.3. The IoT gateway .............................................................................................................. 38  
3.5.2. Main APIs endpoints and data flows ............................................................................... 39  
3.5.3. Dependencies and modules ............................................................................................. 40  
3.5.4. Deployment ...................................................................................................................... 41  
3.5.5. User interfaces and user experience ............................................................................. 41  
3.5.5.1. Define lists of sensing devices and actionEvents ........................................................... 42  
3.5.5.2. Update programming framework ................................................................................. 42  
3.6. Incentive server component ............................................................................................... 43  
3.6.1. Role in the architecture ..................................................................................................... 43  
3.6.2. Main APIs endpoints and data flows ............................................................................... 44  
3.6.3. Dependencies and modules ............................................................................................. 46  
3.6.4. Deployment ...................................................................................................................... 46  
3.6.5. User interfaces ................................................................................................................ 46  
3.7. Serious Games Application layer ........................................................................................ 47  
3.8. User portal .......................................................................................................................... 49  
3.8.1. Role in the architecture ..................................................................................................... 49  
3.8.2. Main APIs endpoints and data flows ............................................................................... 49  
3.8.3. Dependencies and modules ............................................................................................. 50  
3.8.4. User interfaces and user experience ............................................................................. 51
3.8.5. Deployment ................................................................................................................... 55
4. Integration test cases ........................................................................................................... 56
4.1. Integration testing methodology ..................................................................................... 56
4.2. Integration environment and tests .................................................................................. 56
5. Next steps and conclusions ............................................................................................... 59
6. References ......................................................................................................................... 60
7. List of abbreviations .......................................................................................................... 61
8. Appendix ............................................................................................................................ 62
8.1. Responses of endpoints of the IoT access API .................................................................. 62
8.1.1. POST an actionEvent ................................................................................................. 62
8.1.2. POST a sensorList ...................................................................................................... 63
8.1.3. POST deleting an actionEvent ................................................................................... 64
8.1.4. POST deleting a sensorList ......................................................................................... 64
8.1.5. GET the profiles of the active actionEvent of a SG ...................................................... 64
8.1.6. GET the sensing devices of all the existing Lists .......................................................... 65
8.1.7. GET the sensing types supported by the IoT component ............................................ 66
8.1.8. GET the registered sensing devices of a certain List .................................................... 66
8.1.9. Game Configuration Engine get a notification from the IoT platform when a new
      Action Event occurs ......................................................................................................... 67
8.2. Complex types used by the Incentive Server ................................................................. 68
8.3. Game Configuration files .............................................................................................. 69
8.3.1. Game json schema file descriptor .............................................................................. 69
8.4. Integration test cards ....................................................................................................... 69
8.4.1. Integration Test - 01: Create and activate actionEvents within a SG ........................... 69
8.4.2. Integration Test - 02: Game Configuration engine integration with A2 ....................... 70
8.4.3. Integration Test - 03: Configuration of the AKSION and ICEBERG serious game
      experience using the game configuration engine console ................................................. 70
8.4.4. Integration Test - 04: Integration of the Incentive Server in the architecture under the
      A2 single sign on component ............................................................................................. 71
List of tables

Table 1: List of the components of the first release of the architecture ................................................................................. 14
Table 2: Main services exposed and consumed by this component ................................................................................................. 16
Table 3: Main services exposed and consumed by User Profile database .......................................................................................... 18
Table 4: Game configuration actors and scope .......................................................................................................................... 23
Table 5: Types of IoT sensing devices supported in Release 1.0 of the InLife platform ........................................................................ 36
Table 6: High-level description of the main actionEvents considered in the ICEBERG and the AKSION SGs ........................................ 38
Table 7: Endpoints of the IoT access API exposed by the IoT component .......................................................................................... 39
Table 8: HTTP POST request sent to the Game configuration component by the IoT component upon actionEvent detection .......................................................................... 40
Table 9: The deployment of IoT modules ........................................................................................................................................ 41
Table 10: Incentive Server API endpoints ......................................................................................................................................... 45
Table 11: User Portal API endpoints ................................................................................................................................................ 49
Table 12: Services and corresponding integration tests in InLife platform Release 1.0 ........................................................................... 57
Table 13: Template for integration tests .......................................................................................................................................... 57
# List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alpha version of the InLife Open architecture</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>A2 applications proxy configuration console</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>User Profile Database</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>InLife GitHub repository</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>InLife Docker repository</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>InLife game configuration work pipeline</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>Configuration of a game scenario</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>Configuration of a mission inside a game scenario</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>Configuration of the &quot;objects&quot; composing a sample game scenario</td>
<td>29</td>
</tr>
<tr>
<td>10</td>
<td>Configuration of the all possible interactions with the game objects composing a sample scenario</td>
<td>30</td>
</tr>
<tr>
<td>11</td>
<td>Configuration of the steps of a sample mission</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>Learning Analytics Model definition workflow</td>
<td>31</td>
</tr>
<tr>
<td>13</td>
<td>Functional block of the IoT component</td>
<td>35</td>
</tr>
<tr>
<td>14</td>
<td>Interaction between actionEvent manager and resource manager of the IoT cloud server</td>
<td>37</td>
</tr>
<tr>
<td>15</td>
<td>Interactions between the Incentive Server and the others components</td>
<td>44</td>
</tr>
<tr>
<td>16</td>
<td>Interaction between the Game Engine and the SG layer</td>
<td>47</td>
</tr>
<tr>
<td>17</td>
<td>User portal - Home Screen</td>
<td>52</td>
</tr>
<tr>
<td>18</td>
<td>User portal - registration form</td>
<td>53</td>
</tr>
<tr>
<td>19</td>
<td>User portal - login form</td>
<td>53</td>
</tr>
<tr>
<td>20</td>
<td>User portal - add content</td>
<td>54</td>
</tr>
<tr>
<td>21</td>
<td>User portal - forum section</td>
<td>55</td>
</tr>
<tr>
<td>22</td>
<td>Topology of InLife integration environment</td>
<td>56</td>
</tr>
</tbody>
</table>
Executive summary

The aim of this document is to provide a description of all the components comprising the first release of the InLife platform (milestone M3.1) from the technical and user experience point of view. InLife has adopted the “lean” development approach for the implementation, meaning that development started with a very early prototyped version of the architecture with all the components (in their basic form) linked together. This helped identify, early on, all the typical technical issues related to the integration and communication between components that usually affect distributed and heterogeneous cloud architectures, instead of discovering and dealing with them at a late stage.

The document explains the structure and configuration of the InLife platform, describing the role, modules, dependencies, application programming interfaces (APIs), graphic user interfaces (if applicable), and deployment choice for each major platform component. The seven (7) major platform components include:

- Authorization and authentication (A2), serving as a single sign-on solution that provides a unique login experience and secure roles management for all platform users.
- User profile database, designed as an extensible database for storing any additional information about the user to enhance his/her experience (e.g. avatar).
- Game configuration engine, which is the heart of the platform, responsible for providing extensive configuration capabilities to game developers and learning designers for enhancing the experience of players.
- Learning analytics, providing a database for storing in-game data and allowing the creation of meaningful dashboards for educators and other stakeholders.
- IoT platform, responsible for providing real-life action events to the game engine in order to blend game mechanics with actions done in the real world.
- Incentive server, which provides adaptive, non-deterministic incentives to the game players, reasoning over the in-game data statistics stored in the Learning Analytics system.
- User portal, which represents the main web access point for potential players to access information about the latest game releases, and for developers to contribute with new games, plugins and graphic themes.

The implementation of the first two components has been based on the open-source outputs of the EU project RAGE, which serves as an open ecosystem of reusable technological assets for serious game developers. Details for the implementation blueprint and status of each component are given inside the document.

The document also describes the structured approach followed for the integration tests of the platform. InLife specified detailed templates or “test cards” for defining an integration test, namely: Test ID, Test description, Test location, Partner(s) responsible, Components involved, Features tested, relevant Requirements or use cases, Test environment, Preparation process, Dependencies (if applicable), Testing procedure steps, Pass/Fail criteria. This approach was followed to successfully carry out the following necessary integration tests: 1) Creation and activation of IoT-based action events within a serious game, 2) Game configuration engine integration with A2, 3) Configuration of the AKSION and ICEBERG serious games’ experience using the game configuration engine console, 4) Integration of the Incentive Server in the architecture under the A2 single-sign-on component.

The activities performed resulted in the InLife Platform Release 1.0, successfully integrated and deployed in a distributed manner using the cloud server infrastructures of partners ICCS, IMA and TRT. Next steps towards the 2.0 release include the enhancement of learning analytics and game incentives, the extension of user profile datasets, and the definition of even more complex action (IoT-based) events.
I. Introduction

The InLife Open framework architecture undergoes an iterative development approach, starting from an early prototype version and reaching its final shape within two main iterations (plus potentially others) in order to address feedback raised by pilot activities.

One of the biggest challenges of this technical process was that the game design of the two serious game scenarios was scheduled in the DoA after the platform architecture design and development tasks (T4.1 and T4.2 scheduled to start at M7, T2.2 scheduled to start at M1 and T3.3 scheduled to start at M4).

At the same time this challenge represents also an opportunity to think the architecture modules and links as much open and flexible as possible and this is absolutely compliant with the goals of the InLife project of providing a platform able to fit with several different serious game scenarios and game mechanics.

Due to the innovation nature and the time constraints of the InLife project, the technical partners agreed on leveraging on some ready-to-go open source components made available by another European project (RAGE [1]). Those components are the single sign on module and the Learning Analytics system: the first provides a unique authentication and authorization (A2) module used by all InLife components, while the second provides a database to store in-game data and allow the creation of meaningful dashboards for educators and stakeholders.

The goal of the RAGE project is to provide an ecosystem of open source modules to speed up the development of serious games, so the InLife technical partners decided to leverage on these open source technologies in order to focus more on the core features of the InLife components (e.g. personalization and configurability of the games, blending of the in-game mechanics with real life actions) while relying on stable and production ready assets addressing the basic features of the InLife platform such as the user identification and authorization, or the learning analytics platform.

Apart from the A2 and Learning Analytics components, transferred and adapted from the RAGE project, the other key components of the InLife Open Framework are:

- the Game Configuration Engine: this is the component responsible for providing a flexible configuration framework used by game developers and learning designers to personalize the game experience tailored to their users' needs;
- the User Profile Module: technically, it constitutes an appendix of the user’s dataset natively provided by the A2 component, and is meant to store additional information about the user, in order to enrich user experience (e.g. avatar thumbnail, nickname etc.);
- the IoT Platform: responsible for providing real-life action events to the game engine in order to blend game mechanics with actions done in the real world;
- the Incentive Server: responsible for providing non-deterministic incentives to the games, reasoning over the in-game data statistics stored in the Learning Analytics system;
- the User Portal: the main web access point for all the InLife stakeholders where it is possible to access information about the latest game releases (as a potential player) or contribute with new games, plugins or graphic themes (as a developer).
Structure of the document

This document is structured as follows: starting with the description of the methodology used to design and implement the first release of the architecture following the "lean" approach (Section 2), then each component of the architecture is broadly described with details about the key sub-components, the communication flows, the role in the architecture, the graphical user interfaces (where available), and the URL where it is possible to access the service for the components that provide a user interface (Section 3).

After the description of the components, there is a section (Section 4) about integration, analyzing the integration test cases that have been designed in order to validate the release.

The document concludes with a summary (Section 5) presenting further development milestones in order to reach the final release of the platform.

Finally, at the end of the document there are four technical Appendix sub-sections, where the API endpoints documentation is available together with the data structures involved in the configuration workflow of the games, as well as the details of the main integration tests performed.
2. Methodology

As described in the architectural deliverable D2.2 [6] the InLife Open framework architecture is a distributed ecosystem of server side components, some hosted on partners premises and others on cloud infrastructures. One of the greatest challenges of such deployment scenario is to ensure that all the components will be able to effectively communicate with no lag, transmitted data are secure and each component should be able to be up and running even in case of failure of another sub component. In addition, this approach will definitely facilitate the Framework Integration (Task 3.5) since it will allow the detection of deployment issues at an early stage, making them easier to fix and cope with. This is due to the fact that when components are almost in their final shape, it is more complex to fix any integration issues without spending further effort in refactoring parts of the software or even changing from scratch parts of the architecture backbone.

The idea was to follow a "lean approach" [2] applied to the InLife Open Framework architecture development using the approach described in the next paragraph.

Our approach

Following the Lean approach manifesto, we anticipated all the integration tasks developing a very early prototyped version of the architecture with all the components (in their easiest shape), linked together.

This approach has been even more useful since a couple of the components have originated (properly adapted) from another project (A2 for single sign on and the learning analytics component), so it was crucial to test them inside the InLife scope to verify if they really fit in as expected. Following this technical path, we reached quite soon the release of a working version of the whole environment, where all the components were linked together and the basic operations such as login, logout, sign-in and game data tracking were in place.

We called this first prototyped version of the InLife platform "Hello World Architecture". This name takes its origin from a very popular way of saying "hello world" between developers and technical people, by using the easiest piece of running software able to at least print the "hello world" sentence on the user’s screen.

In order to test and demonstrate the reliability of the "Hello World Architecture", we developed also a simple prototyped version of the game clients that used the basic functionalities made available by the "Hello World Architecture".

Once this first "early prototyped" version of the architecture was released, and all the key communication flows have been assessed and validated, the technical partners started to build the components according to the technical guidelines tested within the "Hello World Architecture" to reach the release of the "alpha version" of the platform.

The alpha version of the platform, coupled with the first release of the games will be tested and validated during the pilots’ activities, and all the relevant feedback raised by the stakeholders will be analyzed and inserted in the development roadmap that will end with the release of the final version of the InLife Open framework.
In the table below, a list of all the preliminary integration activities done to release the "Hello World Architecture" is summarized.

<table>
<thead>
<tr>
<th>Module</th>
<th>Feature</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2 (see D2.2 [4] for details)</td>
<td>authentication, authorization, roles creation</td>
<td>§ section 3.1</td>
</tr>
<tr>
<td>User profile module</td>
<td>extend the basic user’s data set provided by the A2 component</td>
<td>§ section 3.2</td>
</tr>
<tr>
<td>Game configuration engine</td>
<td>creation of the A2 users also at Game Configuration backend level using the A2 APIs</td>
<td>§ section 3.3</td>
</tr>
<tr>
<td>Learning Analytics</td>
<td>setup of the system and integration with A2.</td>
<td>§ section 3.4</td>
</tr>
<tr>
<td>IoT component</td>
<td>simple creation, from the educator console provided by the configuration backend, of one &quot;action event&quot; transmitted to the IoT platform through API calls.</td>
<td>§ section 3.5</td>
</tr>
</tbody>
</table>
3. InLife Open architecture

Figure 1 displays a high-level diagram of the architecture, as released in the alpha version, containing all the components described in the sections that follow. This diagram also depicts the actual deployment approach of the components.

![Diagram of the InLife Open architecture](image)

**Figure 1: Alpha version of the InLife Open architecture**

### 3.1. A2 - Authorization and authentication component

This component plays a key role in the whole infrastructure, since it represents the single sign on module that provides to the users a unique login experience while keeping the communication between all the components secure using OAuth2 tokens exchange.

This component, as mentioned in Section 1, represents a technology transfer of an open source module developed within the scope of the RAGE project. The RAGE project is a European funded project aimed at providing a set of reusable technological assets useful to address the typical use cases involved in the serious game development, representing a production ready and validated toolbox to speed up and foster the development activities of the games.

The key responsibilities of the A2 module are the following:

- sign up of players from the game Apps;
- players’ login from the game clients;
- creation of new developers or educators from the Game Configuration Engine backoffice;
- login of developers and educators from the Game Configuration Engine backoffice;
- user login details data storage (username, password, role, email);
- roles and applications management to enable secure proxy links between them.
3.1.1. Role in the architecture

The role of this component in the architecture is to provide a centralized unique access point to all the clients to authenticate only once towards the system and to gain access to all the components of the InLife Open framework with the right roles and authorizations.

Leveraging on the authenticated proxy feature of the A2 component, it is actually possible to configure in the A2 console proxy URLs for all the authorized components so that all the calls to those components will be proxied by A2 and the authentication token will be automatically verified each time.

Figure 2 below shows a snippet of the screenshot of the developers’ user interface to configure the proxy URLs of the services under the A2 authentication and authorization framework.

Figure 2: A2 applications proxy configuration console

3.1.2. Main APIs endpoints and data flows

The table below presents the main authentication and authorization API REST endpoints used by the InLife game clients and game configuration backoffice.

<table>
<thead>
<tr>
<th>No</th>
<th>Service</th>
<th>Endpoint</th>
<th>Parameters</th>
<th>Response</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>signup</td>
<td>/api/signup</td>
<td>username, password, email, role, prefix</td>
<td>HTTP/1.1 200 OK { &quot;msn&quot;: &quot;Users registered&quot;, &quot;errors&quot;: [], &quot;errorCount&quot;: 0 }</td>
<td>POST</td>
</tr>
<tr>
<td>2</td>
<td>login</td>
<td>/api/login</td>
<td>username, password</td>
<td>HTTP/1.1 200 OK { &quot;user&quot;: { &quot;_id&quot;: &quot;559a447831b7acec185bf513&quot;, &quot;username&quot;: &quot;root&quot;, &quot;email&quot;: &quot;<a href="mailto:yourmail@ucm.es">yourmail@ucm.es</a>&quot;, &quot;roles&quot;: [&quot;admin&quot;], &quot;token&quot;: &quot;eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...&quot; } }</td>
<td>POST</td>
</tr>
<tr>
<td>3</td>
<td>add user role</td>
<td>/api/users/:userId/roles</td>
<td>userId, roles</td>
<td>HTTP/1.1 200 OK { &quot;message&quot;: &quot;Success.&quot; }</td>
<td>POST</td>
</tr>
</tbody>
</table>
3.1.3. Dependencies and modules

This component represents the centralized Single Sign On module used by all the InLife clients to authenticate their requests, therefore it is supposed to be linked with all the InLife components that need to perform authenticated requests to other sub components, as well with all the sub components that need to retrieve user data.

A permanent link is established with the user profile database, the module of the InLife user management system in charge for hosting all the user data that A2 natively does not supports.

3.1.4. Deployment

The test version of this component has been deployed on ICCS premises and the link to reach and configure this component is the following:

http://sso-inlife.cn.ntua.gr/login

3.2. User profile database

This component is responsible for storing users’ profile data by extending the basic user’s data set provided by the A2 component. Specifically, the A2 user’s data set will be extended to allow users, once logged in toward the A2 single sign on component, to fill or update their profile, providing a set of personal data such as the avatar thumbnail or other personal data that will compose the personal profile. Specifically, as presented in Figure 3 user profile database is a part of the Users Database that is created by the A2 component.

3.2.1. Role in the architecture

The role of this component in the architecture is to extend the A2 user dataset, managing users’ profile data including personal information, as well as roles and achievements the users reached during the game play.

Due to the “noSQL” nature of the database used by this component (MongoDB [4]), it is relatively easy, in the future, to expand the dataset, adding any other information that will be required by the game scenarios.
3.2.2. Main APIs endpoints and data flows

The User Profile database will extend the API provided by the A2 component in order to take advantage of the authentication and authorization mechanism provided by the single sign on, and also include the necessary users' profile data.

Using the endpoints listed in the table below, it will be possible, using the "userId" information retrieved by the login, to load all the additional information linked to the user's account.

Table 3: Main services exposed and consumed by User Profile database

<table>
<thead>
<tr>
<th>No</th>
<th>Service</th>
<th>Endpoint</th>
<th>Parameters</th>
<th>Response</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>getUserProfile</td>
<td>/api/users/:userid</td>
<td>userId</td>
<td>HTTP/1.1 200 OK { &quot;id&quot;: &quot;59ac2d0073ee620042156641&quot;, &quot;username&quot;: &quot;pavlos&quot;, &quot;email&quot;: &quot;<a href="mailto:pavlos@email.com">pavlos@email.com</a>&quot;, &quot;profile&quot;: { &quot;address&quot;: &quot;home&quot;, &quot;picture&quot;: &quot;pavlos.jpeg&quot; }, timeCreated: &quot;2017-09-03T16:25:36.406Z&quot;, &quot;verification&quot;: { &quot;complete&quot;: false }, &quot;name&quot;: { &quot;last&quot;: &quot;Kosmidis&quot;, &quot;middle&quot;: &quot;P&quot;, &quot;first&quot;: &quot;Pavlos&quot; } }</td>
<td>GET</td>
</tr>
<tr>
<td>2</td>
<td>updateUserProfile</td>
<td>/api/users/:userid</td>
<td>userId, updated schema</td>
<td>HTTP/1.1 200 OK { &quot;id&quot;: &quot;59ac2d0073ee620042156641&quot;, &quot;username&quot;: &quot;pavlos&quot;, &quot;email&quot;: &quot;<a href="mailto:pavlos@email.com">pavlos@email.com</a>&quot;, &quot;profile&quot;: { &quot;address&quot;: &quot;home&quot;, &quot;picture&quot;: &quot;pavlos.jpeg&quot; }, timeCreated: &quot;2017-09-03T16:25:36.406Z&quot;, &quot;verification&quot;: { &quot;complete&quot;: false }, &quot;name&quot;: { &quot;last&quot;: &quot;Kosmides&quot;, &quot;middle&quot;: &quot;P&quot;, &quot;first&quot;: &quot;Pavlos&quot; } }</td>
<td>POST</td>
</tr>
</tbody>
</table>

3.2.3. Dependencies and modules

This component, since it is an extension of the A2 module, it will be linked with all InLife components that need to have access to users’ profile data, as described in Section 3.1.3. A necessary prerequisite will be for the user to be logged in to the game.

3.2.4. Deployment

A first implementation of this component has been made by extending the A2 open source code that can be found at the following link: https://github.com/e-ucm/a2

In order to be able to support InLife’s User Profile component, an extension on the user schema was made in order to be able to store necessary profile data. Specifically, the basic skeleton of users’ schema is now formulated as follows:
module.exports = function (app, mongoose) {
  var userSchema = new mongoose.Schema({
    email: {
      type: String,
      unique: true,
      lowercase: true,
      required: true,
      validate: [validateEmail, 'Invalid email!']
    },
    name: {
      first: {
        type: String,
        default: ''
      },
      middle: {
        type: String,
        default: ''
      },
      last: {
        type: String,
        default: ''
      }
    },
    resetPassword: {
      token: {
        type: String
      },
      expires: {
        type: Date
      }
    },
    verification: {
      complete: {
        type: Boolean,
        default: false
      },
      token: {
        type: String
      }
    },
    timeCreated: {
      type: Date,
      default: Date.now
    },
    profile: {
      picture: {
        type: String,
        default: ''
      },
      address: {
        type: String,
        default: ''
      },
      age: {
        type: String,
        default: ''
      },
      contactNumber: {
        type: String,
        default: ''
      },
      email: {
        type: String,
        default: ''
      }
    }
  });
According to the schema above, the data that, at the moment, can be stored for the users’ profile are:

- profile picture,
- user’s address,
- user’s age,
- user’s contact number (if available),
- user’s email.

The above schema will be further adjusted once the users’ profile information that will be used by the games will be finalized.

In order to make the necessary changes to the A2 module, an InLife forked branch of the e-uCM/a2 repository was created at https://github.com/inlife-h2020/a2 as shown in Figure 4.

![GitHub repository](https://github.com/inlife-h2020/a2)
In addition, as shown in Figure 5, a corresponding InLife Docker repository was also created and is accessible at the following link https://hub.docker.com/r/inlife/a2/.

![Inlife Docker repository](image)

**Figure 5: Inlife Docker repository**

### 3.2.5. Deployment

The runtime endpoint for this component is the same of the main single sign on entry point for the InLife platform that is the following:

http://sso-inlife.cn.ntua.gr/login

### 3.3. Game configuration engine

This component represents the core of the game backend logic. It allows InLife game developers to create a configuration file that, once uploaded to the Game Configuration Engine system, will automatically generate a web configuration console for game designers and educators to let them personalize the game mechanics, objectives, levels of difficulty and scope of the game play.

The work of game developers and educators is split into a game configuration pipeline (see picture below) where each actor of the InLife games creation need to access and work on it at different times.
The main entities that can be configured through the game configuration engine, as shown in the picture above are:

- **game scenario (or environment)**: this entity represents an environment part of the overall game experience, it can be a room, a planet, a fantasy world or other settings specific for the game. Each scenario is a part of the game that the player has to experience and succeed in it. The rules and the missions are configured and personalized using the game configuration console.

- **mini games**: inside a scenario, one or more available mini games can be configured by the educator in order to focus the attention of the player on specific tasks or challenges, detached from the overall game experience.

- **themes**: a set of game graphics that the educator can configure. The idea is to have pre-made sets of graphics to be chosen instead of giving the educator the freedom to choose each colour or graphic, to avoid final visual issues in the user interface.

Basically, through the Game Configuration Engine web console, the involved actors can perform the activities summarized in the following table.
Table 4: Game configuration actors and scope

<table>
<thead>
<tr>
<th>Actor</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform manager</td>
<td>• create new educators</td>
</tr>
<tr>
<td></td>
<td>• edit educator data: username, password</td>
</tr>
<tr>
<td>Game Developer</td>
<td>• upload the json schema game descriptor and generate the game specific configuration console.</td>
</tr>
<tr>
<td></td>
<td>• visually design the game mechanics (in-game objects, missions, steps) using the generated visual artefacts listed in the game json descriptor file.</td>
</tr>
<tr>
<td></td>
<td>• translate in all the supported languages the pieces of configuration that will be loaded by the game client.</td>
</tr>
<tr>
<td>Educator</td>
<td>• assign players to himself</td>
</tr>
<tr>
<td></td>
<td>• create different game versions selecting one or more game environment for each of them.</td>
</tr>
<tr>
<td></td>
<td>• assign different game versions to different players groups. (e.g. one easier game version, with less game objects in it, can be assigned to more impaired kids while the default one can be played by all the rest of the players.</td>
</tr>
<tr>
<td></td>
<td>• configure one or more game scenarios selecting rules, difficulty level, mini games, game objects and game objects interactions.</td>
</tr>
<tr>
<td></td>
<td>• translate in all the supported languages the pieces of configuration that will be loaded by the game client.</td>
</tr>
</tbody>
</table>

A basic work pipeline scenario is the following: once the game configuration file descriptor has been uploaded by the developer to the system, and the configuration console has been automatically generated, the game designer can start creating the game experience visually, creating missions, steps and levels inside the game environment. When the developer and the game designer finish their job, the educator can log into the game configuration engine console, select the game, check if everything is in place and configure any missing setting of the game and assign it to the players or to a group of them.

Paragraph 3.3.6 includes some screenshots of the graphical user interface used by game developers and educators to fully configure and customize an InLife game experience.

The Game Configuration Engine data model has been designed keeping in mind the high level of reusability and configurability asked by the InLife platform requirements.

In fact, since this tool is used by third party game designers, it should be able to design and personalize the majority of their game design ideas. The configuration data model is so abstract that it is possible to consider each configuration "artefact" in a different way according to the specific game scenario. For example, an "Object" inside a mission can be any in-game object or can be an IoT action event, a "Mission" can be a set of steps to be done inside an environment (as for the AKSION game) or a full level completion in another game.
The taxonomy of the key artifacts composing the Game Configuration Engine data model is the following:

- Game environment
  - Objects list: game digital objects as well as IoT physical objects with their properties and settings.
- Mission: a set of steps to be done by the player in order to succeed in a section of the game.
  - Steps: steps to succeed in a mission.
    - Actions: set of actions that the player has to do to complete a step.
      - Groups: actions are “groupable”.
      - Consequence: each action can imply a consequence in the game (e.g. this action brings the player to the next step in the mission.
      - Score: a score for each action.
      - Feedback: a feedback for the player associated to this action: can be a deterministic reward, a message, a popup.

Each mission can be translated (by the developer or the educator) in all the available languages using the game configuration console. When the game begins the mission is loaded in the default language (the one used by the game developer) but, in a second step, other educators (for example, belonging to other pilots) can provide the translations working on the same mission configuration console simply clicking on the flag corresponding to the desired language and providing the translated text for the configuration items.

3.3.1. Role in the architecture

The role of this component in the architecture is to provide a centralized game engine and game configuration engine, integrated both with the Incentive Server and the IoT platform, to create personalized and fully configurable game experiences.

This component is composed by the following sub-components:

- configuration engine: is the framework that allows the automatic generation if the configuration console starting from the game json descriptor file.
- game engine: the component responsible to implement the personalized game logic. This component is coupled with the game engine SDK to provide easy access to the backend resources at game development time.
- game engine SDK: described in section §3.3.3 represents a useful tool for InLife game developers to access low level game engine resources directly from Unity3D, without the complexity of using low level http APIs.

In more detail, the "game configuration engine" is meant to be the framework and the console to create personalized game experience, while the "game engine" is the sub-component responsible to store and make the personalized game logic to the game Apps.
The main links are:

1. Bidirectional data exchange through REST APIs to configure IoT action events in the IoT platform and retrieve notification about those events to be loaded by the games to behave accordingly.

2. Full integration with the A2 component for authentication, authorization and developer and educator users' creation.

3. Integration with the Incentive Server to configure "incentives" to be loaded by the games to blend the game play with non-deterministic users' feedback and "prizes" that make the overall gaming experience more fun and engaging.

3.3.2. Configuration data structures

In this paragraph, the main configuration files are described along with their role and their main structure.

The first one is the "schema json" file, which is created by the developer and uploaded to the Game Configuration Engine backend in order to automatically generate all the configuration dashboards. A detailed structure of this file is available as a link in the Appendix 8.3.

Also, there are the "json configuration" missions files generated by the educator using the Game Configuration Engine console and loaded at runtime by the game Apps. These files are automatically generated by the system during the authorization.

These data structures are the following:

- **Game scenario (or environment):** a data structure containing all the configurable objects of a specific environment (e.g. a room where several missions have to be done by the player, a planet setting or other scenarios)

- **Mission:** a list of steps that have to be done by the player in order to progress in a specific game environment exploration.

- **Game plan:** a composition of one or more scenarios that will be composing the personalized overall game experience for a single player or for a group of players.

3.3.3. Game engine SDK

The only descriptor file that developers need to write is the "game schema", together with the list of all the artifacts that will be composing the game play. Then, once the missions have been authored and assigned to the players, the game Apps need to synchronize with the game engine backend in order to load and provide the game user experience according to the selected configuration.

In order to hide from developers the high complexity of the fine grained Game Configuration Engine REST API layer, an alpha version of an InLife game SDK has been developed and released.

At this time, the SDK is available in the InLife project files repository at the following link:

http://research.cn.ntua.gr/twiki/pub/Main/GameEngine/StatusManager.zip

This SDK is going to be available to the developers of the InLife portal, and the game developers will be able to download it and easily access the game engine features during the development of the games.
The package contains a Unity3D specific SDK that exposes to the game developers all the classes needed to handle the game configuration artifacts: game plan, missions, steps, actions and consequences.

With the SDK, the game Apps can load at runtime the data structures described in §3.3.2. Specifically, the objects that can be accessed through the SDK are:

**Game Plan**: A plan is just a container of games identified by a session id, which can contain several options in key value format. Moreover, there is the attribute "results" that can store information about the plan, defined by the developer.

- **Game**: a game is defined by:
  - *pg_id*: id of the game in the plan.
  - *game_id*: id of the game/environment (independent by the plan, defines the id of the game itself)
  - *name*: name of the game
  - *description*: description of the game
  - *parameters*: is an hashtable that contains all game parameters (if the game is a minigame) by key value
  - *resource_URL*: contains a reference to a thumbnail of the game
  - *executable*: the reference to the scene name
  - *version*: the version of the game, useful to distinguish between the different versions of the game
  - *status*: the status of the game can be "toplay","complete","userexit" or "fail"
  - *score*: is the score obtained in the game, previously saved (if the game was completed)
  - *actualLevel*: a reference to the Level selected
  - *levels*: a list of "levels" contained in the sequence (e.g. a list of mission)
  - *seqMode*: mode of the sequence of levels, it is used to define how the app shall treat the levels sequence and it can be:
    - *auto*: flag to define if the levels will be advanced by the system.
    - *unlocked*: the level is unlocked and available; the player can select all the levels that are lower than the reached level
    - *free*: all the levels in the sequence are available
    - *suggestion*: the level is the one selected by the developer/educator in the plan; however, the level choice in the plan provides information about the user status related to the level. (e.g. if the user has completed the level and should change level or not)
  - *dependencies*: a list of Dependencies that shall be loaded to play the game (if they exist, and are useful for the game)
  - *mission*: actual Mission object related to the Level (a mission is in fact a Level for an environmental game)
• Level
  o A level stores the following properties:
    ▪ id: the id of the level
    ▪ name: the name of the level
    ▪ progress: the amount of progress at this level (if the level is inside a sequence); a value from 0 to 100 where 100 means that the level is completed.
    ▪ parameters: if the game is a minigame, the parameters are pairs of key value.
    ▪ mission: if the game is environmental the level is identified by a mission.

• Dependency: a game dependency is another object that has to be loaded in order to play that game
  o name: name of the dependency
  o type: right now there can be two types of dependency:
    o scene: maps a Unity scene that has to be loaded in a multi scene environment
    o resource: maps whatever resource is useful to load
    o resource URL: a reference to the actual resource that has to be loaded
    o version: the version of the resource (used for caching purposes)

• Mission: The mission is a one to one C# mapping of the data structures of the authored "mission.json" file.

3.3.4. Dependencies and modules

As described in Figure 1, this component is linked to the A2 module for authentication and authorization and has access to the Incentive Server, providing to the configuration console a way to define non-"static" or pre-determined feedbacks by the educator. Specifically, feedbacks for a game scenario are generated by the algorithms of the incentive server.

Another key integration with the Configuration Engine is the IoT platform. The IoT platform allows game designers and educators to configure game scenarios by using not only game objects, but also real IoT objects. Following this approach, missions can be configured in such a way, that the necessary steps for succeeding are based on IoT object events.

Specifically, it will be possible to create an IoT "action event" object, through the configuration console, by having the same user experience as creating any in-game object and select for each action event the necessary sensors that will be used. This information will be saved in the IoT platform using a REST APIs, which is described in detail at the Appendix 8.1. Once an IoT event takes place, the IoT layer informs the game configuration engine. Since the game clients are always synchronized with the game backend, they are also informed about this event and adjust the game logic accordingly.
3.3.5. Deployment

Two different instances of the Game Configuration Engine were created, one for each game that will be developed using the InLife platform. This means that two environments are available, one for the AKSION game and one for the ICEBERG game as shown below:

- [http://aksion-inlife.cn.ntua.gr](http://aksion-inlife.cn.ntua.gr)
- [http://iceberg-inlife.cn.ntua.gr](http://iceberg-inlife.cn.ntua.gr)

The creation of a new environment is a task that will be performed on demand by the platform manager, while multiple instances can be deployed using the same virtual machine.

In the current release, the two game configuration environments mentioned above are installed in the same server, since the technology of this component is a LAMP 1 stack, lightweight and not demanding in terms of server resources consumption. In this way, it is possible to deploy several game environments on the same server without affecting the overall platform performance.

3.3.6. User interfaces and user experience

In the following pictures, screenshots of the game configuration console with the key steps to configure a game are presented.

Figure 7 depicts the section where the game developer can create a new game (see scenario) for AKSION. In this case "school planet" is the name of the first scenario.

---

1 LAMP: abbreviation for Linux - Apache - MySQL database and PHP
Figure 8, depicts the first mission of the "school planet" scenario where it is possible to open and edit all the missions.

An example of the configuration screen with all the available "game objects" that can be used to create a mission is presented below.
All the available actions to interact with the "game objects" can be found in the following figure.

![Figure 10: Configuration of the all possible interactions with the game objects composing a sample scenario](image)

Finally, a screen with all the steps (set of actions) to complete a mission is presented in Figure 11.

![Figure 11: Configuration of the steps of a sample mission](image)

### 3.4. Learning Analytics

This component represents another example of reuse and exploitation of a module developed within the context of the EU H2020 RAGE Project [1] aimed to collect, analyze and visualize data obtained from in-games and real-life activities within the InLife platform, with purposes such as revealing players' progress and how they actually act and learn while playing games and at the same time providing all this information to educators and stakeholders.

As described in the architectural deliverable D2.2 the Learning Analytics platform leverages upon the Experience API (xAPI) data exchange protocol developed within the ADL initiative by xAPI developers part of an open community.
3.4.1. Role in the architecture

The role of this component in the architecture is to provide a centralized system to store game data and provide a centralized and customized reporting system to let developers create meaningful dashboard and visualizations to analyze and assess players' behaviors over the time.

In the picture below there is a diagram showing the actors and the information flows involved in the definition of a Learning Analytics Model (LAM).

![Diagram of Learning Analytics Model definition workflow]

For each InLife game, it is necessary to define clear learning goals. This will lead to the definition of a Learning Analytics Model (LAM). The LAM is responsible for describing which data is to be collected for that particular educational InLife serious game (also called Serious Game or SG) and how it should be processed, reported, and interpreted (for example, in terms of success or failure).

The LAM is an integral part of building usable dashboards for the different stakeholders, and is a key component of Learning Analytics for SGs, also termed Game Learning Analytics, or GLA [5]. Figure 12 illustrates the role of a LAM within GLA.
The key steps to design a LAM are the following:

1. **Definition of the learning goals of a serious game**: the first step is to define a set of learning goals (e.g. specific knowledge, procedures, tasks) that are to be achieved in the game and result in a specific learning design. This task is supposed to be achieved at game development time with a tight cooperation between educators and game developers.

2. **Define game goals**: the previously defined learning goals should be reflected in game mechanics (e.g. tasks and levels design) resulting in the game design. The easier the learning goals are identified in game, the easier the tracking process will be.

3. **Definition of the xAPI traces to be sent**: once the learning and the game design have been defined, the game developers, with the help of the educators, will define the information that should be traced and delivered by the game in order to analyze and visualize them at a later stage. The traces sent by the game should contain the learning information and they must follow the xAPI-SG Model.

4. **Definition of the analysis model**: it is mandatory to have a well-designed analysis model that is able to extract the meaning from the xAPI traces, sent to determine if the game is actually meeting its expected goals or not. This model is responsible of defining how the traces are going to be analyzed and interpreted according to instructions from the game developers and game designers in cooperation with educators. Based on this model it will be possible for the developers to implement and design data visualizations that fit and exploit the above-mentioned analysis model.

5. **Definition of the visualizations**: together with the analysis model, the required visualizations need to be defined to adequately represent and visualize the traces and information already analyzed to help stakeholders understand the learning objectives achievement.

### 3.4.2. Game data tracking

The analytics process starts together with the game. While a user plays, the serious game is responsible of observing the actions the student is performing and sending the information of interest to the learning analytics component. This first step is key to the whole LAM: the system cannot infer any information that has not been obtained and sent by the game.

The tool embedded in the InLife serious games to send the information is called "tracker" and is accessible at the following link as a sub-component of the Rage Learning Analytics system: https://github.com/e-ucm/unity-tracker.

If in the future other serious games will be developed using the InLife Open framework, the open-source RAGE suite offers to developers implementations of the tracker compliant with the most common technologies (e.g javascript tracker for HTML5 games).

However, any tracker that sends traces that conform to the standard xAPI Profile for Serious Games is acceptable to be used for tracking purposes in the InLife platform.

### 3.4.3. Deployment

The Learning Analytics component is currently deployed on ICCS premises at the following address:

http://sso-inlife.cn.ntua.gr/api/proxy/afront/
It exposes a set of RESTful APIs, under A2 authentication (broadly documented at the following link: http://e-ucm.github.io/rage-analytics-backend/) that allows the game configuration engine to perform a set of operations on the Learning Analytics system such as:

- **Games management**: adding a game or game version, removing a game, changing game attributes or returning the list of available public games.

- **Classes' management**: returning a class information or deleting classes (with class is intended a group of students, but a class can even be the whole set of players if there is no need to manage different groups of players with different game configurations.

- **Analysis management**: adding, deleting or returning analysis for a given version of a game.

- **Tracker's management**: starting a tracking session for a client.

- **Visualizations management**: adding or removing visualizations for a game.

- **Kibana management**: adding visualizations or dashboards in Kibana.

- **Templates management**: adding or returning templates in an Elasticsearch index.

3.4.4. Analysis and visualizations

The learning analytics system expects to receive statements following the xAPI-SG model. Any other format received may be stored but no default analysis or visualization can be ensured unless the model is met. Some analysis and visualizations are provided by default from the system.

Those default visualizations are able to display, among others, chart types with the maximum score achieved by each player in each completable\(^2\), the number of correct and incorrect answers in each alternative, bar charts with minimum, mean and maximum times of completion in each completable, line chart with progress in each completable along time or alternatives selected in each alternative.

If any other specific analysis or visualizations are required, they should be provided and developed by game developers according to the specific analysis needs using the flexible Kibana dash-boarding system embedded in the Learning Analytics platform. In this first release of the InLife platform, there are no visualizations and dashboards available since the serious game Apps are still under development and a complete LAM has not been yet designed.

The work that has been done so far went in the direction of assessing and testing the Learning Analytics component offered by the Rage project to understand its fit within the InLife requirements framework, and, once got a positive feedback install and deploy it inside the first release of the InLife platform.

In parallel with this task, during the game design activities of ICEBERG and AKSION done so far, the LAM requirements described above have been taken into account in order to enrich the game design documents of the two serious games with the learning analytics tracking requirements.

---

\(^2\) Class is a group of players. It is possible to have several classes or one class with all the players.

\(^3\) a "completable" is any game object that can be completed (level, scene, mission). See https://github.com/e-ucm/xapi-seriousgames#2-completable
3.5. IoT component

This component is responsible to integrate the IoT paradigm into the InLife framework. The IoT component aims to deliver a flexible and scalable data and resource management framework guaranteeing consistency, security, and reliability in the development of gaming-driven, smart IoT applications. In InLife, the IoT component is implemented as a cloud solution, where the sensing devices expose application-agnostic elementary functionality and the application logic is entirely deployed in the cloud environment. Leveraging on the provided services, it is possible for a serious game hosted and/or developed by the InLife platform to merge versatile application logic into its gameplay and mechanics on top of the conditions and players’ actions taking place in the physical environment.

3.5.1. Role in the architecture

In the InLife architecture, the primal role of the IoT component is to provide a reliable and scalable cloud-enabled IoT environment to effectively manage data acquired from the physical space and support the implementation of smart IoT applications. Towards this direction, the IoT component instruments the proper management of actionEvents, i.e. events that drive business logic and trigger specific rules in the serious game based on input from the available sensing devices [6]. Overall, the mission of the IoT component is to:

- Introduce a reliable and secure data management and data adaptation framework that effectively and transparently aggregates measurements from the sensing devices and processes them to produce higher-level information (i.e. actionEvents introduced in [6]) and timely notify the InLife game server and its components.

- Deliver a flexible and technology-agnostic, cloud-based resource management and programming framework enabling the extension of the IoT component and the introduction of additional features over the deployed infrastructure, thus further promoting synergies between the IoT and the gamification concepts. Typical cases are the development of smart processing applications on top of acquired data, the connection with 3rd parties’ applications/services, the installation of heterogeneous sensing devices etc.,

- Provide the necessary software endpoints allowing educators and/or developers to define, configure and (de)activate actionEvents and lists of sensing devices and continuously probe potential occurrences of the activated actionEvents.

The functional block diagram of the IoT component is shown in Figure 13. As shown, the IoT component is implemented in a modular way and it consists of three main modules:

- the IoT access API,
- the IoT cloud server
- the (local) IoT gateways.

The roles of these three modules are presented in the following subsections.
3.5.1.1. The IoT access API

The IoT access API is a RESTful API exposed by the IoT component to the game configuration engine of the InLife game server. It provides the means to educators and developers to define, configure and (de)activate actionEvents in the context of the deployed mini games, as well as to retrieve detection history of those events. Also, it enables the creation/management of lists of sensing devices (things) as a means to simplify the definition of actionEvents by outlining separate sensing environments, e.g. environments devoted to specific serious game deployments. A thorough description of the endpoints exposed by the IoT access API is given in §3.5.2.

3.5.1.2. The IoT cloud server

The IoT cloud server, shown in Figure 13, is responsible to aggregate and process measurements from the sensing devices in a consistent, vendor-agnostic and transparent way, and manage the lifecycle of the introduced actionEvents and lists of sensing devices. The IoT cloud server consists of three main blocks, i.e. the resource manager, the actionEvent manager and the IoT database system.

The resource manager: The resource manager is a Java-based module running within a Tomcat container, which provides the necessary back-end functionality to manage the connected sensing devices and their measurements. The resource manager introduces a series of virtual resources/sensors to organize data acquisition from the physical sensing devices in a consistent and uniform way and to address any vendor-related potential constraint about their configuration and/or operation [9]. Each virtual resource/sensor provides a common data abstraction schema for all the physical sensing devices of a certain type from those which are supported by the IoT component. The types of sensing devices supported...
in the release 1.0 of the InLife platform are shown in Table 5. More specifically, each virtual resource/sensor establishes a transparent data management mechanism which tracks, stores and synchronizes the measurements received by the sensing devices of a certain type. Also, it responds to the requests of the actionEvent manager, each time the latter checks whether an actionEvent has been occurred or not, by respond to it with the most recent measurements from the sensing devices which are bound to that actionEvent.

Table 5: Types of IoT sensing devices supported in Release 1.0 of the InLife platform

<table>
<thead>
<tr>
<th>No</th>
<th>Sensing Device Type</th>
<th>IoT non-functional requirement in D21</th>
<th>Type of local IoT gateway into the IoT component</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Temperature</td>
<td>NREQ_ IoT14</td>
<td>System on Chip (RPI)</td>
</tr>
<tr>
<td>2</td>
<td>Light</td>
<td>NREQ_ IoT15</td>
<td>Android app. and System on Chip (RPI)</td>
</tr>
<tr>
<td>3</td>
<td>Current (Smart plug)</td>
<td>NREQ_ IoT16</td>
<td>Desktop app. consuming API exposed by a 3rd party</td>
</tr>
<tr>
<td>4</td>
<td>Proximity</td>
<td>NREQ_ IoT17</td>
<td>Android app. and System on Chip (RPI)</td>
</tr>
<tr>
<td>5</td>
<td>Occupancy/vacancy/motion</td>
<td>NREQ_ IoT18</td>
<td>System on Chip (RPI)</td>
</tr>
<tr>
<td>6</td>
<td>Switch</td>
<td>-</td>
<td>System on Chip (RPI)</td>
</tr>
<tr>
<td>7</td>
<td>Water Level</td>
<td>-</td>
<td>System on Chip (RPI)</td>
</tr>
</tbody>
</table>

Apart from virtual sensors/resources, the resource manager is also responsible to orchestrate a message mediator over the IoT gateways and the IoT cloud server. This mediator is responsible for the proper message validation, transformation and routing of the data generated from/to the physical sensing devices. The mediator decouples the communication between the applications running in the IoT gateways, minimizing the mutual awareness they should have of each other to exchange messages. The message broker in the IoT component of the Release 1.0 of the InLife platform is based on the queuing system implemented by the Kaa IoT platform [10].

The actionEvents manager: The actionEvents manager is a cloud-enabled IoT application implemented as a Java servlet which undertakes the lifecycle management of all the actionEvents that have been activated in the serious games hosted by the InLife platform. As defined in [6], each serious game developed or hosted by the InLife platform can merge into its gameplay numerous actionEvents of interest, i.e. events that trigger specific rules in the serious game based on the input from the available sensing devices. From the perspective of the IoT component, each actionEvent defines a combination of values/statuses for a specific set of sensing devices. When this combination holds, an occurrence of the actionEvent is detected. Thus, the main responsibilities of the actionEvent manager is to continually probe all the activated actionEvents (for every serious game hosted by the InLife platform) and send timely HTTP POST requests to the Game configuration component (i.e. notifications to the corresponding serious game) whenever an occurrence is detected.

Apart from the seven types shown there, the IoT component can support up to 20 different types of sensing devices, with a large number of sensing devices per type. The number of supported types can be further scaled up by reconfiguring and restarting the backend services of the resource manager.
Figure 14 illustrates how the actionEvent manager interacts with the resource manager to detect actionEvents. Specifically, their interaction defines two levels of processing: In the lower level, the resource manager orchestrates the data acquisition process from the physical sensing devices and the management of both physical and virtual resources/sensors (the role of IoT gateways is omitted in Figure 14 for clarity reasons). Note that data acquisition from a physical sensing device to the corresponding virtual sensor may be either event-driven or periodic. In the first case, the sensing device sends data upon a status change or change in the measured value (wrt. the previous measured value), while in the second case it sends data per fixed time intervals. In the upper level, the actionEvent manager focuses exclusively on handling the activated actionEvents, being unaware of any networking, connectivity or other issue related to the physical sensing devices and their status. To check whether an activated actionEvent has occurred, the actionEvent manager makes requests to the virtual resources/sensors which control the sensing devices composing the specific actionEvent. These requests are periodic with a configurable period value per actionEvent. Clearly, by the usage of virtual resources/sensors controlled by the resource manager, the network resources are better utilized, the data traffic that incomes into the IoT cloud server side is significantly reduced and the actionEvent manager is released from the tasks to control and direct communicate with the sending devices. Overall, the provable benefit from the deployment of this two-level data management framework is the separation of data messaging and resources handling concern (i.e. the lower level) from the cloud-enabled smart processing and actionEvent control concern (i.e. the upper level). This separation improves the utilization of the available cloud computing and networking resources, makes the deployment of smart IoT applications transparent to the individual specifications of the physical devices and reduces the signaling overhead which is related to the data traffic and messaging within the cloud computing.

The use of actionEvents is the means for the educators to merge IoT events into their serious games to blend the in-game experience with real life actions. As mentioned before, each actionEvent should be bound with a set of sensing devices at the time of its activation, an action which requires educators to have knowledge about the unique IDs of the sensing devices. To avoid forcing them to have such knowledge (which may be impossible in some cases), the IoT component introduces a mechanism to create lists of registered sensing devices. Typically, these lists are created by the developers (or any person) who undertake the deployment of the sensing devices in the physical space. Apart from the registered sensing devices, every such list may also contain some end-user oriented description per device, e.g. what is the role of the device, where it has been located, what is the range of the values it returns etc. Thus, in one respect, each such list models the sensing environment tailored to a specific real use case or serious game deployment. Note that all the defined lists of registered devices are controlled by the actionEvent manager, which does not pose any restriction on their number or the sensing devices that can be registered per list. The only restriction is that each actionEvent should be linked with an existing list at the time of its activation and that the sensing devices which will be bound to this actionEvent should be members of that list. The high-level description of some typical actionEvents considered by the ICEBERG and the AKSION serious games are shown in Table 6.
Summarizing the above discussion, the actionEvent manager undertakes the following responsibilities:

- The periodical probing of all activated actionEvents for all the serious games hosted by the InLife platform. The period value per actionEvent is a configurable parameter and it is set by the educator at the time of activation.
- The mapping of each defined actionEvent with the corresponding sensing devices. The consistency control over each actionEvent is based on the lists of the registered devices.
- The proper handling of requests received via the IoT access API to configure/update/(de)activate actionEvents and the timely messaging to the InLife game server whenever an actionEvents’ is detected.
- The proper storage and recall of (the past) detected actionEvents’ into/from the DBs maintained by the IoT component

**The IoT Database** is a MySQL supporting multiple schemas to store the following types of information:

- The profiles of the activated actionEvents
- The profiles of the lists of registered sensing devices
- The detected actionEvents

The data stored in the DB are available the internal modules/components of the InLife platform via the IoT access API.

### 3.5.1.3. The IoT gateway

The IoT gateways (and the sensing devices which are connected to them) are not considered as parts of the cloud server core architecture. The role of the IoT gateway is to assist data acquirement at the IoT cloud server side by hosting messaging endpoints that translate and transfer measurements of the connected sensing devices. Such endpoints abstract the communication protocols used by the devices and provide functionalities to facilitate the communication to/from the resource manager. Typically, an IoT gateway is accessible from any other deployed endpoint or internal service of the IoT component. As shown in Table 5, three different types of (local) IoT gateways are supported in the Release 1.0 of the InLife platform. Specifically, data from sensing devices are transferred to the IoT cloud server via

**Table 6: High-level description of the main actionEvents considered in the ICEBERG and the AKSION SGs**

<table>
<thead>
<tr>
<th>No.</th>
<th>actionEvent high-level description</th>
<th>Type(s) of sensing devices</th>
<th>Serious Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A player leaves a room and the light is still on/off</td>
<td>Proximity, light</td>
<td>ICEBERG/AKSION</td>
</tr>
<tr>
<td>2</td>
<td>A player approaches a specific recycling bin and throws material in a trash can</td>
<td>Proximity, switch</td>
<td>ICEBERG</td>
</tr>
<tr>
<td>3</td>
<td>The window is open while the air-condition is on</td>
<td>Switch, temperature</td>
<td>ICEBERG</td>
</tr>
<tr>
<td>4</td>
<td>A player leaves his office but the PC is still turned on</td>
<td>Proximity, smart plug</td>
<td>ICEBERG</td>
</tr>
<tr>
<td>5</td>
<td>A player uses the stairs instead of the elevator</td>
<td>Proximity</td>
<td>ICEBERG</td>
</tr>
<tr>
<td>6</td>
<td>A player refills the printer’s tray</td>
<td>Proximity, switch</td>
<td>ICEBERG</td>
</tr>
<tr>
<td>7</td>
<td>A player uses the car-pooling service instead of his car</td>
<td>Proximity</td>
<td>ICEBERG</td>
</tr>
<tr>
<td>8</td>
<td>A player flushes the toilet after usage</td>
<td>Proximity, water level</td>
<td>AKSION</td>
</tr>
</tbody>
</table>
i) A system on chip (SoC) acting as data aggregator which can support several physical sensing devices located at a close distance to it. To support the sensing devices shown in Table 5, three different images have been implemented over Raspberry Pi3 SoC for 1) light, 2) proximity (RFID), and 3) motion, switch, temperature, water level sensing devices, respectively.

ii) A desktop application consuming a private/public API exposed by a 3rd party which supports data acquisition from sensing devices of certain type (e.g. a smart plug). By using this type of gateway, the IoT component is able to support variant smart processing application on top of wearables and smart objects.

iii) An android application running as a background service of the serious game application (e.g. the beacon). This way, it is possible for an android application to interact and exchange data directly with the environment of the SG.

For all the above cases, an application-driven software code is implemented (in C++, Java) at the IoT gateway to facilitate the real-time bi-directional data exchange between the connected sensing devices and the resource manager. Thus, on the one hand, this application subscribes to the messaging system implemented by the IoT component and, on the other, tailors to the individual measuring specifications of the connected physical sensing devices (e.g. digital or analog values etc.).

3.5.2. Main APIs endpoints and data flows

The IoT component exposes the IoT access API to the Game Configuration Engine component: this API is consumed by the IoT configuration service (but also other internal services of the Game configuration component) to allow the educator and/or the SG developer to define, update, and delete lists of sensing devices and actionEvents. The endpoints of the IoT access API are summarized in Table 7. The endpoints #1, 3, 5, 6 are referred to the actionEvent management, while the endpoints #2, 4, 7, 8, and 9 are referred to the management of the connected sensing devices and their lists. The parameters/response of the endpoint #5 will be defined in release 2.0 of the InLife platform.

Table 7: Endpoints of the IoT access API exposed by the IoT component

<table>
<thead>
<tr>
<th>No</th>
<th>Service</th>
<th>Endpoint</th>
<th>Parameters</th>
<th>Response</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Activate an actionEvent</td>
<td>/inlife/activateActionEvent</td>
<td>inLifeEnableEvents: Complex body</td>
<td>(presented in Appendix §8.1.1)</td>
<td>POST</td>
</tr>
<tr>
<td>2</td>
<td>Create a list of (registered) sensing devices or add one (or more) to a certain list</td>
<td>/inlife/addToSensorList</td>
<td>sensorList: Complex body</td>
<td>(presented in Appendix §8.1.2)</td>
<td>POST</td>
</tr>
<tr>
<td>3</td>
<td>Delete an actionEvent</td>
<td>/inlife/deleteActionEvent</td>
<td>guid: Integer number</td>
<td>(presented in Appendix §8.1.3)</td>
<td>POST</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>aevent: Integer number</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D3I: Integrated INLIFE Open Framework Release R1.0
The parameters of the HTTP POST request which is sent to the Game configuration component whenever an actionEvent is detected are shown in Table 8.

Table 8: HTTP POST request sent to the Game configuration component by the IoT component upon actionEvent detection

<table>
<thead>
<tr>
<th>No</th>
<th>Service</th>
<th>Endpoint</th>
<th>Parameters</th>
<th>Response Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pass a detected actionEvent to the Game configuration engine</td>
<td>/v1/api.php?action=saveservice=iot</td>
<td>aeID: action event id: number, giD: game id: number, timestamp: timestamp: date, uID: anonymous user token, h= hash, details: body of the event: json</td>
<td>TBD POST</td>
</tr>
</tbody>
</table>

3.5.3. Dependencies and modules

The modules of the IoT component and their dependencies from other technological frameworks and/or components of the InLife platform are summarized as:

- ActionEvent manager: It is a Java executable running on Java version 1.8.0 or newer version. The database system where the profiles of the actionEvents, the lists of sensors and the detected actionEvents are stored is MySQL. The ActionEvent manager communicates with the resource manager using REST requests.
Resource manager: It is implemented by using Spring framework for J2EE services. The message brokering is based on the queuing mechanism and related SDK provided by the open-source Kaa IoT platform (version 0.10.1) [10].

IoT access API: It is implemented based on the RESTfull architectural style. The API is consumed by the game engine configuration component of the InLife game server though the IoT configuration tab of the GUI used to configure the SG.

IoT gateways: Raspberry PI3 model B running Raspbian has been used as SoC-based gateway. The android version for android application gateways should be newer than 4.3. In all cases, TCP/IP and networking access should be guaranteed for each IoT gateway. Message brokering and SDK of Kaa IoT platform are used to enable the development of application specific software codes supporting data aggregation from the sensing devices and transfer to the IoT cloud server.

Sensing devices: The IoT component is technology agnostic wrt the sensing devices which are not considered as a module of the IoT component. The integration of each individual sensing device requires an application-driven code that properly adapts measurements and communication management to the schema introduced by the resource manager. This code is typically written in Java/Python/C++ and it is executed by the IoT gateway.

The IoT cloud server runs on Ubuntu 64 16.04, and communicates with the game configuration component of the InLife game server and the IoT gateways (and the physical sensing devices connected to them) over the internet using HTTP requests on top of TCP/IP protocols.

3.5.4. Deployment

The deployment of the software modules of the IoT component to hardware infrastructure is shown in Table 9.

<table>
<thead>
<tr>
<th>No</th>
<th>Software module of the IoT component</th>
<th>Hosting hardware infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IoT cloud server side (includes the actionEvent, the resource manager and the IoT access API)</td>
<td>A virtual machine (VM) running in ICCS premises (4 vCPUs, 8GB Memory and 32GB disk, Ubuntu 64) IP domain: iot-inlife.cn.ece.ntua.gr</td>
</tr>
<tr>
<td>2</td>
<td>SDK and application specific code providing data adaptation</td>
<td>SoCs (i.e. raspberries) acting as local IoT gateways for (groups) of sensing devices</td>
</tr>
<tr>
<td>3</td>
<td>Android IoT application</td>
<td>Android devices (e.g. tablets, smartphones) running also the SGs</td>
</tr>
</tbody>
</table>

3.5.5. User interfaces and user experience

From business perspective, the interaction an end-user may have with the IoT component is of two kinds: i) define lists of sensing devices and actionEvents, and ii) update programming framework by connecting further sensing devices, installing further sensing types or update internal (backend) services. These two options are discussed below:
3.5.5.1. Define lists of sensing devices and actionEvents

The IoT component does not expose a direct user interface. In the SGs hosted by the InLife platform, the lists of sensing devices and the actionEvents are defined by the educators and/or developers through the configuration console provided by the game configuration engine. The rules of the IoT component that should be merged into the workflow of this GUI are summarized in the steps below:

1) The educator and/or the SG developer selects, updates or creates one or more lists of sensing devices on top of which one or more actionEvents will be defined in the context of a mini game. Note that upon the definition of a new list of sensing devices, the IDs of the sensing devices which are included to that should be given as input by the educator and/or the developer.

2) One or more actionEvents are activated on top of a selected list by specifying the sensing devices which will be linked with each actionEvent.

3) Repeat step 1) and/or step 2) as many times as needed.

3.5.5.2. Update programming framework

The actions a developer may perform wrt. the programming framework of the IoT component are:

- Update the configuration settings of an already connected sensing device,
- Connect an additional sensing devices of a supported type by the IoT component,
- Install an additional sensing devices of a (currently) not supported type by the IoT component.
- Update/modify internal services of the cloud server side.

In the first two actions, the developer has to update the application-driven software code running in the IoT gateway which handles data adaptation and communication between the sensing device and the resource manager. For example, the following source code shows how the measurements from a digital temperature sensor with ID 000111222 are collected by a SoC IoT gateway and are transferred to the IoT cloud server.

```cpp
#include <boost/asio.hpp>
#include <boost/algorithm/string.hpp>
#include <kaa/Kaa.hpp>
#include <kaa/IKaaClient.hpp>
#include <kaa/configuration/manager/IConfigurationReceiver.hpp>
#include <kaa/configuration/storage/FileConfigurationStorage.hpp>
#include <kaa/log/strategies/RecordCountLogUploadStrategy.hpp>
#include <kaa/event/gen/EventFamilyFactory.hpp>
#include <kaa/event/gen/InlifeSensorEventClassFamily.hpp>
#include <kaa/event/gen/InlifeSensorEventClassFamilyGen.hpp>
#include <kaa/event/registration/IUserAttachCallback.hpp>

+ class SimpleUserAttachCallback : public kaa::IUserAttachCallback {
  ...
}

+ class ConfigurationCollection : public kaa::IConfigurationReceiver {
  ...
}

int main()
{
  ConfigurationCollection configurationCollection;
  try {
    // It does control of the transmit and receive data
```
configurationCollection.run();
} catch (std::exception & e) {
    std::cout << "Exception: " << e.what();
}

std::cout << "Start listening for Sensor Events" << std::endl;

while (true) {
    // Retrieve the sensor id
    std::string deviceId = "000111222";
    // Retrieve the sensor value (returned by the sensor I/O)
    double value = temperatureValue();
    // Retrieve the sensor type. It has to be one of the available sensor types offered by the platform.
    nsInlifeSensorEventClassFamily::InlifeSensorType sensorType = nsInlifeSensorEventClassFamily::InlifeSensorType::TEMPERATURE;
    configurationCollection.update(sensorType, deviceId, value);
}

return 0;

To extend the types of the sensing devices supported by the IoT component, the InLife developers have to update the source code of the resource manager by implementing further virtual sensors queues and updating its internal addressing and messaging management structures. To update/modify the internal backend services of the IoT cloud server, the platform developer has to intervene to its source code implementation. Details and examples of the last two cases will be described in the manual of the Release 2.0 of the InLife platform.

3.6. Incentive server component

This component is responsible of computing rewards for players in order to incentivize them to meet the Educator’s training goals. It uses intensive machine learning techniques and artificial intelligence (AI) principles in order to choose the best reward to incentivize the players. An Educator’s goal could be as an example that players have an energy consumption responsible behavior. The Educator will more precisely explain how to reach this objectives: perhaps he wants players accomplished actions in a balance way and avoid that a player accomplishes all the time the same action or perhaps the Educator wants players get a minimum number of points each days.

Each time the game engine needs to suggest or assign a reward to a player for an action in the game, it requests the incentive in order to define this reward. If the incentive server “thinks” the action going to be realized by the player will help him to reach the expected behavior, the component will return a reward he “thinks” it will incent the player to realize the action. For example, when the game engine asks to the incentive server which reward to give to a player for switching the lights, the incentive server will return a high reward if this action helps the user to reach the behavioral goal defined by the educators, or a lesser reward otherwise.

3.6.1. Role in the architecture

The role of this component in the architecture is to answer all requests of the game engine in order to return the better reward possible to incent players. Furthermore, this component will retrieve all players’ data from the backend Analytics in order to learn from players’ actions, then build incentive rules policies and store analysis results. These results will help the Incentive server to choose the better reward strategy to incent players.
3.6.2. Main APIs endpoints and data flows

The Incentive Server is a REST service. The table below shows the Incentive Server API consumed by:

1. The game engine in order to request rewards computation for players
2. The incentive administration WEB console used by Educators to configure the incentives options
3. The game Apps (like ICEBERG or AKSION) in order to initialize the Incentive Server with data concerning rewards and actions

This API contains POST methods that enable to initialize the Incentive server for a game and a GET method in order to retrieve a reward for a player related to an action of the game. The GET reward method can return JSON or XML format, depending of the “Accept” header tag ("application/json" or "application/xml").
Table 10: Incentive Server API endpoints

<table>
<thead>
<tr>
<th>No</th>
<th>Service</th>
<th>Endpoint</th>
<th>Parameters</th>
<th>Response Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>createActionsTable</td>
<td>api/proxy/incentive/actions/{Id}</td>
<td>Id = GameId</td>
<td>201: created</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ActionsTable: complex type JSON object in body</td>
<td>POST</td>
</tr>
<tr>
<td>2</td>
<td>setGlobalBehavioralObjectives</td>
<td>api/proxy/incentive/behaviors/{Id}/global</td>
<td>Id = GameId, objectives: String</td>
<td>201: created</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>POST</td>
</tr>
<tr>
<td>3</td>
<td>setBehavioralObjectives</td>
<td>api/proxy/incentive/behaviors/{Id}</td>
<td>Id = GameId, user: String, objectives: String</td>
<td>201: created</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>POST</td>
</tr>
<tr>
<td>4</td>
<td>setGlobalRewardsPolicyMode</td>
<td>api/proxy/incentive/rewards/{Id}/global</td>
<td>Id = GameId, mode: String</td>
<td>201: created</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>POST</td>
</tr>
<tr>
<td>5</td>
<td>setRewardsPolicyMode</td>
<td>api/proxy/incentive/rewards/{Id}</td>
<td>Id = GameId, user: String, mode: String</td>
<td>201: created</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>POST</td>
</tr>
<tr>
<td>6</td>
<td>getJSONReward</td>
<td>api/proxy/incentive/rewards/{Id}</td>
<td>Id = GameId, user: String, action: String</td>
<td>Reward (JSON)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GET</td>
</tr>
<tr>
<td>7</td>
<td>getXMLReward</td>
<td>api/proxy/incentive/rewards/{Id}</td>
<td>Id = GameId, user: String, action: String</td>
<td>Reward (XML)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GET</td>
</tr>
</tbody>
</table>

First of all, for each serious game of the InLife platform, the incentive server must be initialized with a table of actions (either real life actions or in-game actions) that will contain all the actions allowed and handled by the game. This table will be sent as a JSON or XML object in the body of the request (see Game Configuration Engine data structures described in section § 3.3.2)

When the actions list available for a game has been defined, educators must define and tell to the incentive server what behavioral objectives they expect for each user. A global behavioral objective for all players of the game can also be defined. Behavioral objectives describe/explain how players must accomplish actions in order to reach the Educator’s objectives of the game. They are configured by an Educator using the web administration console.

The setRewardsPolicyMode() method enable to set the rewards computation mode of the Incentive Server for a game (or for a specific player), with two possible values:

- Static: the Incentive server, for this game, will use fixed rewards or number of points (defined by Educators) each time it must rewards a player. Educators can change these values during the game.

- Dynamic: the Incentive server uses AI algorithms (inspired from reinforcement learning mechanism based on Ant Colony Optimization [11]) to determine the best reward for actions completed by a player. These values evolved continuously with player’s behavior.

The GET requests allow the game engine to retrieve the best reward to give to a specific player when he accomplishes an action in the game or in the real life.
3.6.3. Dependencies and modules

The Incentive server, in the final release of the InLife platform, will fully communicate with the following components:

- The learning analytic platform in order to retrieve players’ data and reason over that in order to provide non-deterministic incentives to the games.
- The A2 component for authenticated and authorized access.
- The Game Engine that will request incentives for players.
- A web administration console that will allow educators or game developers to configure the incentives with behavioral objectives (at initialization time and during the game lifecycle)
- The InLife serious games that will initialize the Incentive Server with their actions table (at initialization time and at any time during the game)

In this first release of the platform, since the game design of AKSION and ICEBERG is still under development, and the incentives rules and mechanics have not been fully yet identified, the Incentive Server will be only integrated in the platform through the A2 integration to allow educators and game developers to access the administrative console.

When the game design of AKSION and ICEBERG will be defined and the configuration data structures will be available, the Incentive Server database will be initialized and first incentives rules will be created through the administration console, so that as soon as the Learning Analytics system will have stored enough users’ data, also the non-deterministic and machine learning computed incentives could be provided to the players as well.

3.6.4. Deployment

This component is a self-executable REST service that is deployed in the TRT private cloud, into a virtual machine with 4 cores and 32 Gb of memory. These values can be easily scaled up to support more computational power as well as more storage for players' in-game data.

3.6.5. User interfaces

The Incentive Server web administration console allows educators to configure the incentive strategy, specify the behavioral objectives and specify the Reward's policy mode. The web console is part of the Incentive Server platform and will be accessible under A2 authentication.
3.7. Serious Games Application layer

The next diagram shows how the InLife SG App layer interacts with the game server:

Main components in the application layer are:

- Unity3D Learning Analytics Tracker SDK. This component, provided by the RAGE project put together two different functionalities:
  
  - User management: This component encapsulates all the functionality related with the creation, authorization and authentication of users against the A2 server in order to be able to send authenticated xAPI calls to the Learning Analytics component. As a prerequisite the user should be registered and logged in the platform.

```csharp
Tracker.T.username = username;
Tracker.T.password = password;
Tracker.T.Connect((bool resultExt, string data) => {
...
});
```
• Learning Analytics: This component encapsulates all the functionalities related with the learning analytics system. Translate the game logic objects into the xAPI format and send the information to the learning analytics platform to allow the educators later on extract meaningful information from the visualizations available in Kibana.

```java
Tracker.T.completable.Initialized(Application.productName,CompletableTracker.Completable.Game);
```

• Inlife SDK
  o Load the game configuration: the Inlife SDK encapsulates the functionality to load the main game configuration that the educator and the game developer have set in the game configuration console. As explained in section §3.3 this configuration schema contains all the game objects (with its associated options and actions) and the missions configured by the educator. This information will be loaded using the Inlife data model that has to be mapped to real game objects in Unity Engine.

```java
StatusManager.instance.Load<Game>(loadHandler,parameters);
```

  o Load current game status: each logged user that has previously played an Inlife game wants to recover its previous game status. The Inlife SDK allows the storage and restore of a game status to provide the user with a seamlessly cross-device game experience: the player can continue playing in the same device or a different one once it has logged in. If the player has never played the game previously, the developer will provide a default starting point.

  o Game lifecycle management: Inlife SDK allows tracking the game lifecycle sending all the missions that have been completed by the user, retrieving the deterministic (set by the educator) rewards, and providing almost real-time in-game feedback to the player.

```java
StatusManager.instance.Log ("status-game",{"pg_id":+g.pg_id+,"session_id":+p.session_id+,"results":{"test":true}}).hashtableFromJson());
```

• Game Logic: acts as a gateway between the InlifeSDK objects and the Unity3D engine. Translate missions, plans, objects and options in game elements.

• Assets and user interfaces: it’s the layer that provides the user interface and the game experience (characters, storyboards, events, music, sound).
3.8. User portal

This component is in charge for the management of the different stakeholders that can interact with the InLife platform with different roles. User portal allows managing user, roles, games and components that are available to build games using the InLife SDK and acts also as a showcase for the games and plugins developed inside InLife ecosystem.

The main objectives of the user portal are:

- be the main marketing tool for the InLife project to raise awareness in stakeholders belonging to different market segments
- be the main entry point for developers that want to use InLife tools to create serious games
- build a community for games, plugins and themes developers that generates a collaborative environment.
- acts as a marketplace to share components, plugins, themes and games developed with the InLife tools.

3.8.1. Role in the architecture

The role of this component in the architecture is mainly for dissemination and marketing purposes providing links to the ecosystem of games and tools developed by the InLife community. It represents the main tool to publish InLife tools and to attract developers to the InLife ecosystem. It also collects KPIs and metrics regarding how many games are available and how many users are linked to the InLife ecosystem as simple registered users or as active players.

3.8.2. Main APIs endpoints and data flows

In the table below the main endpoints provided by this component are described.

<table>
<thead>
<tr>
<th>No</th>
<th>Service</th>
<th>Endpoint</th>
<th>Parameters</th>
<th>Response</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inlife News</td>
<td>/news/feed</td>
<td>None</td>
<td>RSS 2.0</td>
<td>GET</td>
</tr>
<tr>
<td>2</td>
<td>Game List</td>
<td>/games/list</td>
<td>None</td>
<td>An array of Game object with the following properties:</td>
<td>GET</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• description</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• thumbnail</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• download url</td>
<td></td>
</tr>
</tbody>
</table>
3.8.3. Dependencies and modules

This component is in relationship with the following components:

- **A2**: sign-in and sign-up to the user portal will be made available through the A2 component.

- **Learning Analytics**: user portal will query the learning analytics system to retrieve the main game usage statistics that could be useful to be presented in the public portal.

- **Gitlab server**: a code repository will be part of the InLife architecture as well, to provide the developers with the opportunity to access and download the latest versions of the tools and games developed for the InLife ecosystem.

The main modules of the InLife user portal are:

- **User module**: allows to register, modify, delete or block registered users. Both public and private functionalities will be made available:
  
  - Public features available for all the users that access the web portal:
    
    - **Register a new user**: allows a new user to be registered toward the platform providing the following information: username, email and password. It also provides the option to select the type of user between game or plugins developer.
    
    - **Edit profile**: after login edit information about the user profile (e.g. avatar, preferences etc)
    
    - **Login**: login with user credentials
    
    - **Reset password**: send an email with a URL that allows the user to reset and restore the password.

  - Private features available for the InLife administrators.
    
    - list, search and filter all the registered users.
    
    - read-only access to the users' profiles.
    
    - block or unblock users.

- **Games / Plugins / Themes management**: Allows the developers and the administrators to create/modify/delete plugins and themes and all its related information such as screenshots, descriptions, availability and so on. Also they can upload an notify the portal users for new versions of a specific product.

- **Forums**: Allows the community to share information, ask questions and solve issues related to the InLife components. The forums will be categorized as follows and only registered users can access it:
  
  - Game development
  
  - Plugin development
  
  - Theme development.
• **Web Portal**: Is the main public section of the user portal and shows any information related with the InLife ecosystem: news, new games, SDK documentation, new versions available for games and tools, and any information that can be useful for the dissemination and the promotion of the InLife project.

3.8.4. User interfaces and user experience

This section provides a description of the main screens that compose the user portal together with the functionalities included in each one of them.

**Home screen**

Main screen from which you can access the rest of the functionalities and which shows the main information of the portal in a summarized way to allow you to see all the contents at a glance.

From the top menu, we can access each of the main sections of the portal: Games, Assets, Forum or Login, our shopping cart or the portal content search engine.

Below is the header of the portal with different promotion images that show the new characteristics of the portal.

Then a grid with the latest games and content displayed on the portal is shown, a small top menu allows us to switch between the latest and most popular or free contents for example. For each of the contents, an advertising image is shown, the name of the content and the author, the publication date and the price if any.

Finally, we have the link to download the platform SDK as well as access to the necessary help information for its use.
User registration form and Login screen

Login and registration screens to access or create if necessary our profile in the portal. The process has been designed to be as simple and quick as possible, without large information requirements that make the process longer than necessary.
To register, you only need the user’s email address, your username, an identification name and password, allowing in any case the automatic registration through external accounts already existing in Google or Facebook. Once the user has registered, you will receive a confirmation email in order to validate the account.

The login screen will allow the user to enter the platform by entering their email and password or through the associated Google or Facebook account. In this way, the user will have access to all its contents, purchases made, content uploaded, etc.

**New game registration form**

This form allows users registered in the portal to upload content (games, themes, plugins) so that they can be accessed by other users.

The data needed to register a content are:

- Name of the content
- Content type (Game, Theme, plugin)
• Version of the uploaded content

• Brief description of the content, what it does, what it is used for, how it can be used....

• Price of the content, which can be 0 if it is free.

• The file with the required assets.

• Two promotional images, of different sizes and proportions.

• Screenshot of the content

Finally, before uploading any type of content, the user must accept the terms of use and privacy policy provided by the portal, checking the appropriate checkbox.

**Forums list**

Forums allow registered users to share information with the community, ask questions or resolve questions from other users about InLife platform components. The behavior of this section is the standard for this type of forums, where users will be able to create the topics they want to talk about within any of the predefined categories.
3.8.5. Deployment

This component will be deployed over a LAMP stack composed by an Apache server with PHP (v7 or greater) and connected to a MySQL (v5.6 or later) or MariaDB (v10.0 or later).

It will be deployed under ICCS premises but the architecture allows, if needed, an easy porting toward an on-premises deployment.
4. Integration test cases

A major principle for InLife’s effective integration process, relates to the data models and interfaces that have been introduced. Data models define classes coupled with the related methods and attributes, while interfaces define the specifications of the communication and the information flow among the various software components. In InLife, a RESTful architectural style was adopted to establish cross-component communication and interaction. Thus, the introduction of well-defined data models and specifications for the interfaces was a major effort during both the architecture design and development phases to avoid significant incompatibilities and facilitate platform’s integration. Apart from this, numerous unit tests were performed during the development process of the individual components (and their provided services), not only to ensure that the requirements set in D21 are properly addressed but also to find out bugs and logical errors and fix them before their effects are propagated to the other stages of integration and testing procedures. As typically happens, these unit tests were performed by the developing teams of the corresponding components following the white-box unit testing approach at the level of the source code [12].

In this section, we describe the integration process of the various software components and services implemented in the context of the InLife platform. First, the methodology for integration testing is presented, including description about the integration environment and the software tools that have been introduced to support both integration and testing. Then the implemented integration tests are presented in a unified way. As the InLife platform will be extended/updated towards the Release 2.0, enhancements and modifications of existing integration tests will also take place, by considering any potential update and modification of the software architecture, as well as the used data models and interfaces.

4.1. Integration testing methodology

First, the integration environment and the components to be integrated and tested are identified, and their physical and logical (functional) boundaries are defined. Note that the term “components” refers to any software or hardware element which can perform autonomous functions and whose operation can be summarized by a set of functional features. Next, the services of the InLife platform are identified requiring the interaction of two or more components. The validation of these services is performed by integration tests executed on the test environment.

4.2. Integration environment and tests

The InLife project introduces an integration and testing environment to facilitate complete integration and testing of the development efforts delivered by the individual developing teams involved in the InLife project. It aims at verifying in runtime platform’s services, updates and integrations before they are uploaded to the “production environment” supporting the four pilots of the project. The topology of the integration environment is shown in Figure 22.
As shown in Figure 23, the ICCS cloud premises provide three VMs hosting the following components:

i) the IoT cloud server of the IoT component,

ii) the A2 and user profiling components,

iii) the management portal and SG application layer components.

iv) the Learning Analytics module.

Further, the game configuration component is hosted in a VM belonging to IMA AWS cloud (both AKSION and ICEBERG configuration environment are hosted in the same VM), while the incentive server is hosted in a VM belonging to TRT private cloud premises. The components which are hosted in different cloud environments communicate with each other over the internet by using HTTPS requests, while internal communication in the ICCS cloud is performed using VPN over the local IP LAN.

The main services of the InLife platform Release 1.0 requiring cross-component interaction and the corresponding integration tests that have been introduced to validate them, are shown in Table 12.

<table>
<thead>
<tr>
<th>No</th>
<th>Service</th>
<th>Integration test ID</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create and activate actionEvents within a SG</td>
<td>IT - 01</td>
<td>§ Appendix 8.4.1</td>
</tr>
<tr>
<td>2</td>
<td>Game Configuration engine integration with A2</td>
<td>IT - 02</td>
<td>§ Appendix 8.4.2</td>
</tr>
<tr>
<td>3</td>
<td>Configuration of the AKSION and ICEBERG serious game experience using the game configuration engine console.</td>
<td>IT - 03</td>
<td>§ Appendix 8.4.3</td>
</tr>
<tr>
<td>4</td>
<td>Integration of the Incentive Server in the architecture under the A2 single sign on component</td>
<td>IT - 04</td>
<td>§ Appendix 8.4.4</td>
</tr>
</tbody>
</table>

The integration tests referred in Table 12 are described in Appendix 8.4 by using the template shown in Table 13. Each integration test validates a certain group of functional features of two or more individual components of the InLife platform, which are working together to deliver the corresponding service.

<table>
<thead>
<tr>
<th>Field</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test ID</td>
<td>The unique ID of the test</td>
</tr>
<tr>
<td>Test description</td>
<td>Brief test objective description (3-4 sentences) mentioning the services of the InLife platform under test</td>
</tr>
<tr>
<td>Test location</td>
<td>Where the test is going to take place (physical resources)</td>
</tr>
<tr>
<td>Partner(s) responsible</td>
<td>Partners who are responsible to perform the test</td>
</tr>
<tr>
<td>Components</td>
<td>The components which are involved in the test</td>
</tr>
<tr>
<td><strong>Features</strong></td>
<td>The functional features under test</td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td>The requirements and/or use cases which are validated by the test case.</td>
</tr>
<tr>
<td><strong>Test environment</strong></td>
<td>List of elements needed for the test execution. May refer to a general test environment.</td>
</tr>
<tr>
<td><strong>Preparation</strong></td>
<td>Short list of preconditions which the test environment must meet before test execution</td>
</tr>
<tr>
<td><strong>Dependencies (optional)</strong></td>
<td>List of codes of the test cases which need to be verified before this test case can be started. This shall reflect that the test case at hand depends on features previously tested by other test cases.</td>
</tr>
<tr>
<td><strong>Steps</strong></td>
<td>Testing procedures, i.e. list of actions needed for test execution</td>
</tr>
<tr>
<td><strong>Pass/Fail criteria</strong></td>
<td>Expected (measurable) results, allowing to unambiguously judge if the test is passed or not passed.</td>
</tr>
</tbody>
</table>
5. Next steps and conclusions

The release 1.0 of the platform described in this document represents only the first milestone in the development path that will reach the final version on M16.

This release, although it is not complete, allows the creation and personalization of the serious games AKSION and ICEBERG game play through the game configuration engine, blended with a simple action event coming from the IoT platform.

Several further steps will have to be done in order to reach the final version of the platform that will also take in consideration feedback and any further needs coming from the pilot activities.

The next key steps will mainly affect the following activities:

- definition of an InLife serious games xAPI vocabulary and LAM, adequately generic and adaptable to several serious game designs, able to let developers send meaningful and semantic xAPI streams to the Learning Analytics component in order to be able to create, through the Kibana console, meaningful reports for all the stakeholders.

- Improved analysis and definition of the learning goals and metrics of the games (highlighted during the pilots’ activities) in order to provide an improved version of the in-game data tracking that will allow stakeholders and educators to retrieve more information and feedback from the games’ usage.

- Update and extension of the user data set handled by the user profile database.

- Optimization of the integration with the Incentive Server reasoner, leveraging on the above mentioned InLife serious games vocabulary in order to provide machine learning generated incentives to the players.

- Define and integrate more complex IoT action events (or series of action events) in the games’ final release.

- Development of a set of custom learning analytics dashboards with metrics and KPIs coming from the AKSION and ICEBERG serious games players’ data compliant with the LAM definitions.

- Perform another set of integration tests in order to assess and validate the components' updates available in the final release of the platform.
6. References

[1] RAGE EU project: http://rageproject.eu/
[8] Opensource sub-component belonging to the Learning Analytics module: https://www.elastic.co/
[10] https://www.kaaproject.org/
7. List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>DoA</td>
<td>Description of the Action</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>LAM</td>
<td>Learning Analytics Model</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency Identification</td>
</tr>
<tr>
<td>RSS</td>
<td>Rich Site Summary</td>
</tr>
<tr>
<td>SDK</td>
<td>Service Development Kit</td>
</tr>
<tr>
<td>SG</td>
<td>Serious Game</td>
</tr>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>UI/UX</td>
<td>User Interface/User eXperience</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
8. Appendix

8.1. Responses of endpoints of the IoT access API

8.1.1. POST an actionEvent

**Endpoint URL:** /inlife/activateActionEvent

**Description of the parameter InLifeEnableEvents:**

sensor***: (applies to every sensor type available, i.e. sensorLight, sensorTemperature, sensorCurrent, sensorProximity, sensorWaterLevel, sensorMotion, sensorSwitch, sensorType8,…,sensorType20)

<table>
<thead>
<tr>
<th>Type</th>
<th>Complex body, {}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Structures the attributes that define the behaviour of each sensing device (of the specific type) that participate in the actionEvent.</td>
</tr>
</tbody>
</table>

**Internal Complex type fields:**

- sensorId: Type: String
  - Description: the unique id of the sensing device

- sensorThreshold:
  - [optional]
  - Type: double
  - Description: the value which has to be reached (for sensor types Proximity, WaterLevel, Motion, Switch) or exceeded (for every other sensor type, downwards/upwards depending on “ascending” field) for the sensor to be regarded as triggered. Default value is 0.

- exceeding:
  - [optional]
  - Type: boolean
  - Description: indicates whether the specific sensing device is triggered when the sensor value is lower or higher than the sensorThreshold value specified. It does not affect sensor types Proximity, WaterLevel, Motion, and Switch. Default value is false, i.e. the sensing device is considered triggered when the sensor value is below the sensorThreshold specified.

**Example:** Define an actionEvent with ID 101 for the SG with ID 25. The actionEvent will combine a light sensing device (set threshold to indicate that the light is on at 100 lux) and a switch sensing device detecting closed position. The actionEvent will be checked every 5 sec.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inLifeEnableEvents</td>
<td>{}</td>
</tr>
</tbody>
</table>

```json
{
  "aeuId": 101,
  "guid": 25,
  "interval": 5,
  "sensorCurrent": [{}],
  "sensorLight": [{
"sensorId":"3211AO54S","sensorThreshold":"100","exceeding":"true"}],
  "sensorListId": 32,
  "sensorMotion": [{}],
```

D31: Integrated INLIFE Open Framework Release R1.0
Response Code

200

Response Body

{
  "reason": "Successfully enabled sensors for an existing game"
}

8.1.2. POST a sensorList

Endpoint URL: /inlife/activateAddToSensorList

Description of the parameter sensorList:

<table>
<thead>
<tr>
<th>Type</th>
<th>Complex body, {}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Structures the information needed to define/update the list of registered sensing devices</td>
</tr>
<tr>
<td>Internal Complex type fields:</td>
<td></td>
</tr>
<tr>
<td>sensorDescriptionList:</td>
<td></td>
</tr>
<tr>
<td>Type: array of complex types, {}...{}</td>
<td></td>
</tr>
<tr>
<td>Internal complex type fields: sensorId: {}</td>
<td></td>
</tr>
<tr>
<td>Internal complex type fields:</td>
<td></td>
</tr>
<tr>
<td>Internal complex type fields:</td>
<td></td>
</tr>
<tr>
<td>Type: tuple, {&quot;sensorType&quot;:&quot;String&quot;, &quot;sensorDescription&quot;:&quot;String&quot;}</td>
<td></td>
</tr>
<tr>
<td>Description: the tuple that contains user generated information describing the sensing device with the unique sensorId</td>
<td></td>
</tr>
<tr>
<td>sensorListId:</td>
<td></td>
</tr>
<tr>
<td>Type: long</td>
<td></td>
</tr>
<tr>
<td>Description: [Optional] The ID of the list under definition. If not defined, the sensors are added to the default list (i.e. list with ID -1)</td>
<td></td>
</tr>
</tbody>
</table>

Example: Define a list (with ID 32) that registers two sensing devices and provide corresponding high-level description

Parameters

Parameter | Value
sensorList
{
    "sensorDescriptionList":
    {
        "3211AO54S":{"sensorType":"sensorLight","sensorDescription":"Office 12"},
        "FFFS658475A1":{"sensorType":"sensorSwitch","sensorDescription":"Recycling bin in room 2"}
    },
    "sensorListId":32
}

Response Code
200

Response Body
{
    "reason": "Successfully added new Sensor Set with id: 32"
}

8.1.3. POST deleting an actionEvent

Endpoint URL: /inlife/deleteActionEvent

Example: Delete the actionEvent with ID 101 of the SG with ID 25

Response Code
200

Response Body
{
    "reason": "Successfully deleted Action Event for an existing game"
}

8.1.4. POST deleting a sensorList

Endpoint URL: /inlife/deleteFromSensorList

Example: Delete sensor with ID FFFS658475A1 (sensorSwitch) from the list with ID 32

Response Code
200

Response Body
{
    "reason": "Successfully deleted Sensor Id FFFS658475A1 from Sensor Set with id: 32"
}

8.1.5. GET the profiles of the active actionEvent of a SG

Endpoint URL: /inlife/getActiveActionEvents

Example: Get the profiles of the actionEvents for the SG with ID 25

Response Code
200
Response Body

```json
{
  "guid": 25,
  "aeuid": 101,
  "sensorListId": 32,
  "interval": 5,
  "sensorLight": [
    {
      "sensorId": "3211AO54S",
      "sensorThreshold": "100",
      "exceeding": "true"
    }
  ],
  "sensorMotion": []
}
```

8.1.6. GET the sensing devices of all the existing Lists

**Endpoint URL:** /inlife/getAllSensorLists

**Example:**

**Response Code**

200

**Response Body**

```json
{
  "guid": 25,
  "aeuid": 101,
  "sensorListId": 32,
  "interval": 5,
  "sensorLight": [
    {
      "sensorId": "3211AO54S",
      "sensorThreshold": "100",
      "exceeding": "true"
    }
  ],
  "sensorMotion": []
}
```
"sensorDescription": "Office 12"
}
}
,"45": {
"sensorListId": 45,
"sensorDescriptionList": {
"1wer45sdf": {
"sensorType": "sensorWaterLevel",
"sensorDescription": "WC in room 2"
},
"AAARTDFS": {
"sensorType": "sensorCurrent",
"sensorDescription": "PC135-John Malkovic"
}
}
}

8.1.7. GET the sensing types supported by the IoT component

Endpoint URL: /inlife/getAvailableSensorTypes

Example:

Response Code
200

Response Body
[
"sensorLight",
"sensorMotion",
"sensorCurrent",
"sensorProximity",
"sensorWaterLevel",
"sensorTemperature",
"sensorSwitch",
"sensorSensorType8",
"sensorSensorType9",
"sensorSensorType10"
"sensorSensorType11"
"sensorSensorType12"
"sensorSensorType13"
"sensorSensorType14"
"sensorSensorType15"
"sensorSensorType16"
"sensorSensorType17"
"sensorSensorType18"
"sensorSensorType19"
"sensorSensorType20"
]

8.1.8. GET the registered sensing devices of a certain List

Endpoint URL: /inlife/getSensorList

Example: Retrieve the sensing devices of the list with ID 32

Parameters

Parameter	Value
8.1.9. Game Configuration Engine get a notification from the IoT platform when a new Action Event occurs

**Endpoint URL:** http://aksion-inlife.cn.ntua.gr/v1/api.php?

**Example:** IoT platform send an ActionEvent with ID 17, referred to game ID 3 (e.g. aksion) to the Game Configuration Server at the timestamp 20171006170256 referred to user 125678156253456

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>save</td>
</tr>
<tr>
<td>service</td>
<td>iot</td>
</tr>
<tr>
<td>aeID</td>
<td>17</td>
</tr>
<tr>
<td>gID</td>
<td>3</td>
</tr>
<tr>
<td>timestamp</td>
<td>20171006170256</td>
</tr>
<tr>
<td>uID</td>
<td>125678156253456</td>
</tr>
<tr>
<td>h</td>
<td>6869b92f288fcd99c4afa7826e2fa23a</td>
</tr>
<tr>
<td>details</td>
<td>{test:12345}</td>
</tr>
</tbody>
</table>

**Response Code**

200

**Response Body**

```json
{
  "ActionEvent_ID": "27",
  "Game_ID": "3",
  "User_ID": "125678156253456",
  "ActionEvent_Time": {
    "date": "2017-10-06 17:02:56.000000",
    "timezone_type": 3,
    "timezone": "Europe/Berlin"
  },
  "Details": "{test:12345}",
  "errors": []
}
```

**Response Code**

200

**Response Body**

```json
{"success":false,"message":"","errors":["something went wrong"]}
or
{"success":false,"message":"","errors":["illegal hash"]}
or
{"success":false,"message":"","errors":["required field missing"]}
```
8.2. Complex types used by the Incentive Server

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Complex body</td>
</tr>
<tr>
<td>Description</td>
<td>Contains the reward value to assign to a player for an action</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Internal Complex type fields:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>userId:</td>
<td></td>
</tr>
<tr>
<td>Type: String</td>
<td></td>
</tr>
<tr>
<td>Description: the unique id of the player specified</td>
<td></td>
</tr>
</tbody>
</table>

| actionId:                    |     |
| Type: String                 |     |
| Description: the unique id of the action specified |

| rewardId:                    |
| Type: int                    |
| Description: the score of the player, a value between 0 and 100 points than can be translated into game points (each game having its own points currency). |

**Example (JSON)**

```
{
  "userId": "PLAYER_1",
  "actionId": "TURN_OFF_WC_LIGHT",
  "rewardValue": "90",
}
```

**Example (XML)**

```
<reward>
  <userId>PLAYER_1</userId>
  <actionId>TURN_OFF_WC_LIGHT</actionId>
  <rewardValue>90</rewardValue>
</reward>
```

<table>
<thead>
<tr>
<th>Object Name</th>
<th>ActionsTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Complex body</td>
</tr>
<tr>
<td>Description</td>
<td>Contains the complete list of all actions available in the game</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Internal Complex type fields:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>actions:</td>
<td></td>
</tr>
<tr>
<td>Type: complex body</td>
<td></td>
</tr>
<tr>
<td>Description: the complete list of actions handled by the game</td>
<td></td>
</tr>
</tbody>
</table>

| id:                          |     |
| Type: String                 |     |
| Description: the unique id of an action |

| type: [Optional]             |
| Type: String                 |
| Description: the type of the action. This information could be used in the future by the Incentive Server to compute rewards taking into account actions type. |

**Example (JSON)**

```
{
  "actions": [
    {
      "id": "ACTION_1",
      "type": "ACTION_TYPE_1",
    },
  ]
}
```
## 8.3. Game Configuration files

### 8.3.1. Game json schema file descriptor

<table>
<thead>
<tr>
<th>Object Name</th>
<th>serious game json schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>JSON format</td>
</tr>
<tr>
<td>Description</td>
<td>Contains the complete list of all objects and actions available in the game</td>
</tr>
<tr>
<td>Example (JSON)</td>
<td><a href="http://research.cn.ntua.gr/twiki/pub/Main/GameEngine/schema.json">http://research.cn.ntua.gr/twiki/pub/Main/GameEngine/schema.json</a></td>
</tr>
</tbody>
</table>

## 8.4. Integration test cards

### 8.4.1. Integration Test - 01: Create and activate actionEvents within a SG

<table>
<thead>
<tr>
<th>Test ID</th>
<th>IT-01: Create and activate actionEvents within a SG (e.g. ICEBERG)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test description</td>
<td>Ensure that the requests which are sent by the game configuration console to the IoT platform are properly handled by the last. Also, that HTTP POST requests which are sent by the actionEvent manager (of the IoT platform) to the Game engine are properly handled by the SG environment to activate a mini game rule. The test introduces four actionEvents, which aim to detect two specific real-life conditions (i.e. that a PC is still on while the player is leaving its working station and that a window in the space is open) and corresponding player’s expected behaviour (i.e. turning off the PC and closing the window, respectively).</td>
</tr>
</tbody>
</table>
| Test location         | - Integration environment of the InLife platform  
 - IoT gateway and sensors are placed in SYN premises  
 - The player accesses the ICEBERG SG in SYN premises using Galaxy Tab A |
| Partner(s) responsible| SYN, ICCS, FFM, IMA |
| Components            | - IoT cloud server  
 - Game configuration engine  
 - User profile DB  
 - ICEBERG environment  
 - Sensing devices: switch, RFID, smart plug, beacon |
| Features              | - Metering services (by connected sensing devices)  
 - Data acquisition (by IoT gateways of the three supported types)  
 - Data adaptation and management (by resource manager running in the IoT |
ActionEvent configuration (by configuration engine console within ICEBERG environment)
ActionEvent detection (by actionEvent manager running in the IoT server)
ActionEvent handling (by game engine)

Requirements
- Granularity of sensing measurements and relative settings
- Configurable groups of things/ sensing devices
- Resources configuration
- Time-stamped measurements responses
- Configure a thing/sensing device
- Retrieve current measurement of a thing
- IoT component modularity
- IoT component transparency

Test environment
Sensing devices of certain types (from those supported Release 1.0) are connected to the IoT platform. Specifically,
1. a switch is attached to a window
2. a beacon is placed near to the external door
3. a smart plug is connected to a PC of the computer lab and an RFID is placed near to the door of the same room

The resource manager and the actionEvent manager of the IoT component are running in VM1 of ICCS cloud, the RAGE A2 is running in VM2 of ICCS cloud and the game configuration engine is running in IMA cloud. The physical sensing devices are placed at SYN premises, and the player (also in SYN premises) uses a Samsung Tab A (2016) to access the ICEBERG hosted in the InLife platform.

Preparation
The ICEBERG SG is onboarded in the InLife platform and the educator has already configured its baseline mechanics and he has created a group of players. Sensing devices have been placed on SYN premises and they have been connected to the IoT platform (i.e. the corresponding application driven software has been onboarded).

Dependencies (optional)
- The mechanics of the ICEBERG have been set
- The educator and the player have been registered in the user profiling DB. In the player’s record, there is info about the RFID tag and the tablet android ID he uses. This info can be retrieved by the ICEBERG
- The player should scans his RFID card each time he is passing the door of the computer room

Steps
1) The educator accesses the game configuration console of the ICEBERG to configure a mini game which make use of the IoT platform. The educator defines a list of sensing devices by including a beacon, an RFID, a smart plug, and a switch sensor.
2) The educator activates the following four actionEvents:
   - AE1: The smart plug is on and the RFID detects presence
   - AE2: The smart plug is off
   - AE3: The switch (attached in a window) is open and the beacon detects presence
   - AE4: The switch (attached in a window) is closed and the RFID detects presence
3) The actionEvent manager verifies the proper definition of the activated actionEvent(s).
4) The educator creates the following SG rules on top of the activated actionEvent(s).
   - If AE1 is detected, a notification is sent to the player who poses the RFID tag that the device (where the smart plug is attached) is on. If AE2 is detected within some short period of time (e.g. 1 min) after the notification sent, the player takes some points.
If AE3 is detected, notifications are sent to all players who are within the range of the beacon urging them to close the window. If AE4 is detected within a short period of time, the player who poses the tag takes some points.

5) The player launches the ICEBERG.
6) Physical space in SYN premises are modified to meet certain conditions. Specifically, the window (with attached switch) is opened and the PC (connected to the smart plug) is turned on.
7) The player passes the computer room door by scanning his RFID card.
8) The player is ready to leave the building by passing the external door.
9) Repeat step 6) at least 10 times

HTTP POST requests are received by the ICEBERG environment (indicating an occurrence of the activated actionEvent(s)). The related SG rules are activated and expected notifications are sent to the player.

### Test ID
IT-02: Game configuration engine integration with A2

### Test description
Access the Game configuration console, create educators and developers users able to create and configure a serious game.

### Test location
- Serious game configuration environment on IMA cloud: http://aksion-inlife.cn.ntua.gr or http://iceberg-inlife.cn.ntua.gr

### Partner(s) responsible
IMA

### Components
- A2 server
- Game configuration engine

### Features
- creation of educators or developers users

### Requirements
- A2 component up and running
- Game Configuration Server for a specific serious game up and running.

### Test environment
For this test will be taken into consideration the creation of AKSION users using the following game specific configuration environment: http://aksion-inlife.cn.ntua.gr

### Preparation
Launch a web browser and access the following link: http://aksion-inlife.cn.ntua.gr

### Dependencies (optional)
- A2 available

### Steps
1. Access the following web link: http://aksion-inlife.cn.ntua.gr
2. Login as a platform manager.
3. Create, using the game configuration engine console, one educator user providing all the mandatory information.
4. Logout as platform manager.
5. Login as educator (using the same account that has been created at the step 3)
6. Create the players that will have to play to your game
7. Associate the players to yourself creating a class.

### Pass/Fail criteria
The above mentioned users have been created also at A2 level. The test is to login with those users directly to the A2 console available at the following link: sso-inlife.cn.ntua.gr
## 8.4.3. Integration Test - 03: Configuration of the AKSION and ICEBERG serious game experience using the game configuration engine console.

<table>
<thead>
<tr>
<th>Test ID</th>
<th>IT-03: Configuration of the AKSION and ICEBERG serious game experience using the game configuration engine console.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test description</td>
<td>The goal of this test case is to test the automatic generation of the game configuration dashboard and the configuration process to create and personalize the game play of the AKSION and ICEBERG serious games.</td>
</tr>
</tbody>
</table>
| Test location | • AKSION Serious game configuration environment: http://aksion-inlife.cn.ntua.gr available on IMA cloud  
• ICEBERG Serious game configuration environment: http://iceberg-inlife.cn.ntua.gr available on IMA cloud |
| Partner(s) responsible | IMA, FFM |
| Components | • A2 server  
• Game configuration engine |
| Features | • Automatic generation of the game configuration dashboard  
• Personalization of the game experience for AKSION and ICEBERG. |
| Requirements | • A2 component up and running  
• Game Configuration Server for a specific serious game up and running.  
• Internet connection available to allow game clients to synchronize with the game engine to retrieve the game configuration. |
| Test environment | • http://iceberg-inlife.cn.ntua.gr  
• http://aksion-inlife.cn.ntua.gr |
| Preparation | • Launch a web browser and access the following link: http://aksion-inlife.cn.ntua.gr  
• Launch a web browser and access the following link: http://iceberg-inlife.cn.ntua.gr |
| Dependencies (optional) | • The game developer must have written the specific game configuration json schema file.  
• A2 available |
| Steps | 1. Access the AKSION game configuration console as game developer  
2. Create a new game uploading the json schema file to the system, using the "upload" button functionality.  
3. Create a first game environment  
4. Create one or more missions that use one or more game object listed in the json schema.  
5. Configure for each game object all the available parameters.  
6. For each mission define a set of steps, and for each steps the preliminary actions and consequences.  
7. Assign the game to a single player or to a group of players  
8. Repeat from step 1 using the ICEBERG environment. |
| Pass/Fail criteria | The game App, after the user’s login, will synchronize with the game backend and will load the personalized game experience. |
8.4.4. Integration Test - 04: Integration of the Incentive Server in the architecture under the A2 single sign on component

<table>
<thead>
<tr>
<th>Test ID</th>
<th>IT-04: Integration of the Incentive Server in the architecture under the A2 single sign on component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test description</td>
<td>Ensure that the Incentive Server, in the TRT private cloud, is reachable for InLife components and educators through the A2 Gateway proxy of the platform.</td>
</tr>
</tbody>
</table>
| Test location | • Integration environment of the InLife platform  
• Incentive Server is deployed in TRT premises |
| Partner(s) responsible | TRT |
| Components | • Incentive Server  
• A2 services |
| Features | • User authentication/authorization using A2 gateway  
• Incentive server reachability for outside components  
• Incentive Server configuration |
| Requirements | • A2 component deployed and configured to include the Incentive server app  
• Incentive server deployed in TRT private cloud  
• Web client tester to test API of the Incentive Server |
| Test environment | The Incentive server is deployed on a virtual machine hosted in the TRT private cloud. The A2 gateway is deployed in ICCS premises and is configured in order to redirect incoming requests to the Incentive server. A Web client tester is started outside of TRT premises and sends requests through Internet to the A2 gateway at destination of the Incentive server. |
| Preparation | A user test user is created with a role of “developer” in order to access all API of the Incentive server. |
| Dependencies (optional) | |
| Steps | 9. The Web client tester application is started and send a request through Internet to login to the A2 gateway, using the user TestUser credentials  
10. The A2 Gateway receive the login request, proceed and return an authorization token that will use to submit requests to the Incentive Server.  
11. The web client sends a new request using this token to the A2 Gateway that checks the token then redirect the request to the Incentive Server  
12. The Incentive server receives the request then computes and returns a reward for PLAYER_1 and action TURN_OFF_WC_LIGHT |
| Pass/Fail criteria | The Web Client receives the reward and displays it and the reward must contain a user ID PLAYER_1, an action ID TURN_OFF_WC_LIGHT and a reward value with a random number between 0 to 100, for example:  
```json  
{  
   "userId": "PLAYER_1",  
   "actionId": "TURN_OFF_WC_LIGHT",  
   "rewardValue": "90",  
}  
```