chist-era SUCCESS

M4.2: Successful Pilot 1

Grant Agreement:	N/A
Project Acronym:	SUCCESS
Project Name:	SecUre aCCESSibility for the internet of things
Instrument:	CHIST-ERA Call 2015
Thematic Priority:	SPTIoT
Start Date:	1 December 2016
Duration:	36 months
Document Type[1]:	T (Technical Report)
Document Distribution[2]:	CO (Confidential)
Document Code[3]:	SUCCESS-MU-PR-001
Version:	v1.0
Editor (Partner):	J. Gimenez (MU)
Contributors:	J. Augusto, J. Gimenez, F. Kammüller
Workpackage(s):	WP4
Reviewer(s):	F. Kammüller (MU)
Due Date:	Month 24
Submission Date:	Nov 2018
Number of Pages:	28

Funded by EPSRC Engineering and Physical Sciences Research Council, ANR, NWO Netherlands Organisation for Scientific Research

[1]MD = management document; TR = technical report; D = deliverable; P = published paper; CD = communication/dissemination.
[2]PU = Public; PP = Restricted to other programme participants (including the Commission Services); RE = Restricted to a group specified by the consortium (including the Commission Services); CO = Confidential, only for members of the consortium (including the Commission Services).
[3]This code is constructed as described in the H2020 Project Handbook.

M4.2: Successful Pilot 1

J. Gimenez[1]

[1]MU

REVISION HISTORY

Date	Version	Author	Modification
10.12.2019	1.0	F. Kammüller	Started the document
27.1.2020	1.	F. Kammüller	Approval

APPROVALS

Role	Name	Partner	Date
Project Manager	F. Kammüller	MU	27.1.2020

Contents

1 Executive Summary

This is the documentation of the successful Pilot 1. An earlier first draft of this has has been provided in Milestone M3 (M4.1).

2 Tool Set

3 Environment

The Pilot development is carried on within the Smart Spaces lab at Middlesex University. Part of the Smart Spaces lab is set up as a smart room for experiments within IoT and for use of the GOODIES Team (Research GrOup On Development of Intelligent Environments http://ie.cs.mdx.ac.uk). Figure 3.1 shows an accurate map of the lab with hardware elements installed inside as server, some sensors used, smart hub and processing unit.

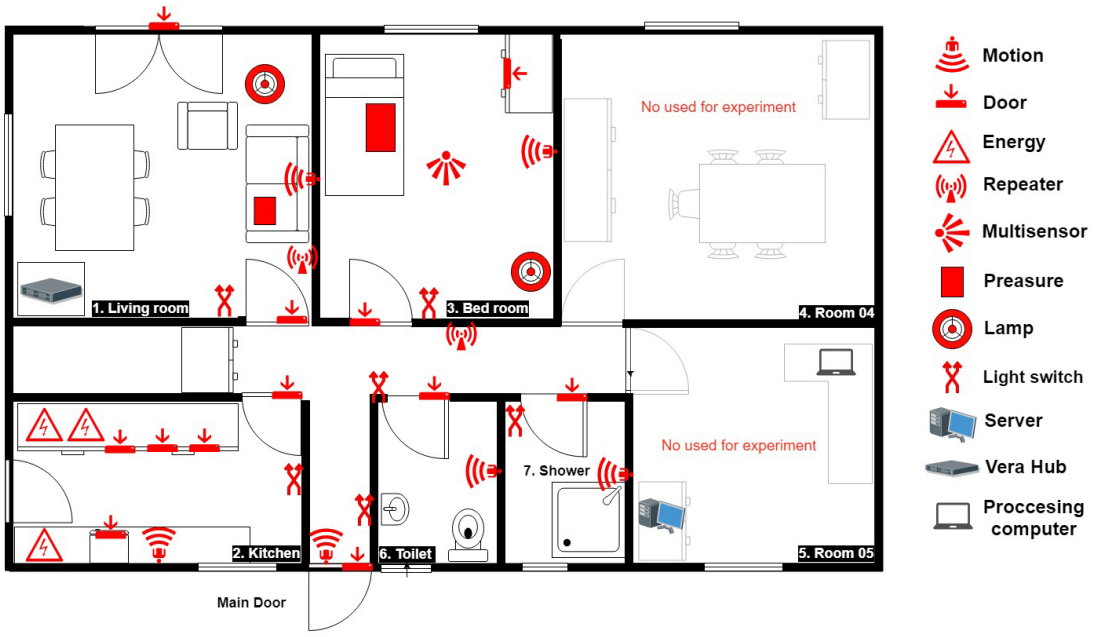

Figure 3.1: Lab map and hardware distribution.

4 Architecture

A first approach about initial Pilot 1 architecture is shown in figure 4.1 and next sections describe each element in more detail. This initial choice has been taken following first meetings to gather requirements.

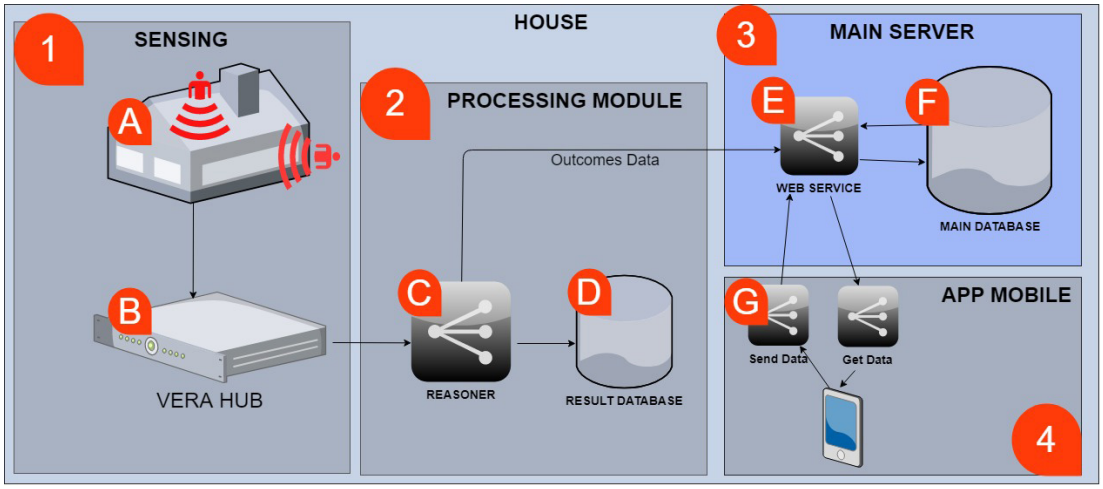

Figure 4.1: Initial Pilot 1 architecture

1 - The house sensing environment:

- A - Z-wave sensing network.
- B - Vera Hub.

2 - Processing module

- C - MReasoner tool
- D - MReasoner's database.

3 - Main Server

- E - MySQL Database.
- F - Web server and PHP API RESTful.

4 - Mobile Enviroment

- G - Android mobile APP

4.1 Network

The whole system is running within WLAN lab with DHCP management, although the server is configured with a static IP. The lab network (WAN and LAN) is provided by Middlesex University and we do not have control over it, but this net is closed and it is unavailable from outside. We should address to IT department for more details about net security and structure.

4.2 Smart hub and Sensors

The smart hub installed inside the Lab is a Vera Plus model UI7 (firmware 1.7.3232). Vera Plus has its own WIFI that allows users connection on port 80 through a friendly web interface as well as a range of URL commands to retrieve devices information and also allows the communication with Z-Wave sensors. The hub has a wire to connect to internet that is not necessary for the correct Vera work. The sensors used are Z-Wave mainly, which incorporates wireless 128-bit encryption, among other security measures. See more details in http://getvera.com/faq.

Vera hub does not have a database but it stores sensors configuration in JSON files and writes in a non-persistent log the information from sensors, e.g. the change of state of devices. This log can be accessible through 80 port by browsers and through on 22 port SSH encrypted using terminal server service to operating system of Vera, which is a Linux BusyBox v1.19.4, which needs to be provided with user and password. The method used by our reasoning tool (MReasoner) to get and sent information to Vera is using URL commands to port 80 interface.

Get further information in http://www.getvera.com

Figure 4.2: Smart hub Vera Plus installed in the lab.

4.3 Server

The computer used as server in this project is a DELL Latitude 5610 (Intel i5 dual-core at 2.17Ghz and 4GB of memory) which is placed in a lab's room that is locked with password. The server is connected to Ethernet and does not have emergency power energy system as UPS.

4.3.1 Operating System

The server has installed a Microsoft Windows 10 Enterprise 64. There is only a user with password created in the system which has system administrator role. It does not have any specific configuration since there are more research carrying out inside the lab using the server. The serves firewall is configured by default although with some modified rules to allow the correct work of other application used in the lab (HTTP, database and FTP connections) and for the correct Pilot work. The FTP is listening on port 22 by Filezilla Server and it is used during development stage as well as MySQL database is listening on 3306 port. The SO is not encrypted and there is not external backup at the moment.

Figure 4.3: Server in the lab.

4.3.2 Web server

The web server used is Microsoft Internet Information Services (IIS) v.10.1756. There are two web sites defined in IIS: 'default_site' which is block, and 'success' site which can only receive connections HTTPS through port 80. There is installed a self-signed certificate (use for deployment stages) required to allow the HTTPS connections, being self-signed it forces browsers to add a security exception when request. Also, the public certificate has to be included in the applications developed to connect with the server as well the appropriate code to manage the self-signed certificates and to control security exception generated. Other security restrictions are configured in IIS, as avoid uploading files from a client, forbid reading files from IIS that are not in the site and avoid reading content files from client. The rest IIS configuration is by default. Figure 4.4 shows a global view of the server and main flows through server layers as firewall, PHP CGI and API, MySQL and IIS.

Web API

The API CRUD RESTful running over IIS has been developed for this project using PHP 7.0.26. As stated in previous section it can only be accessed through HTTPS protocol by port 80 with the URL format: https://10.12.102.61/success A brief description of the API classes developed:

- **login.php** allows authenticate an user in the server. It returns an SID and a session cookie. Sessions management is achieved by PHP engine that has a default configuration. The request login require two parameters, 'username' and 'password', to start a session server and to allow subsequent requests. Sessions expire in one hour, whereupon the user cannot access to API and is forced to login again. See figure 4.5

- **server.php** is the main file to interact with user and resources. The URL request path determines what resource is accessed (extract from URL path) and request method determines the action on it. The methods allowed are GET (retrieve information, it can be all elements from a resource or just one element), POST(create new resource element), PUT(update one resource element), DELETE (delete all or just one element of a resource), they are the four methods developed following the CRUD standard. Also, this class checks whether the URL format is correct (e.g. the name of resource exist) and whether the JSON parameters format are corrects when they are required for the resource

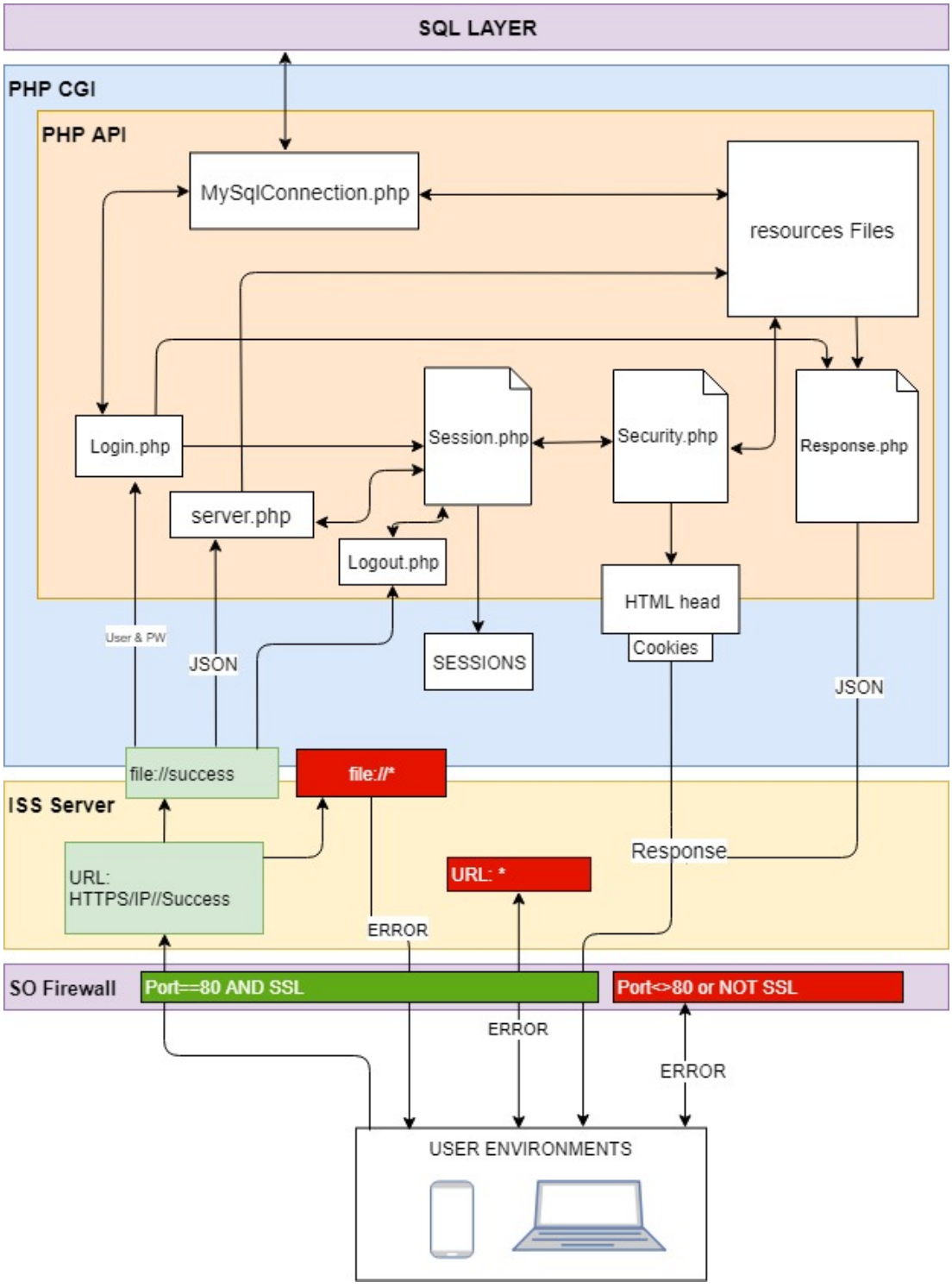

Figure 4.4: Overview system diagram. It shows the user connection route through logical layers server. Firstly, the firewall filters requests from users, blocking those not are address to 80 port using SSL. Next layer, ISS server, avoids the access to different paths of success site just allowing the URL format HTTPS://ip/success. PHP CGI managements PHP sessions and other security measures. Finally, the route reaches PHP API, which just allows connection to login.php, server.php and logout.php. Diagram shows a general picture about the relations between main PHP classes and the flows among them.

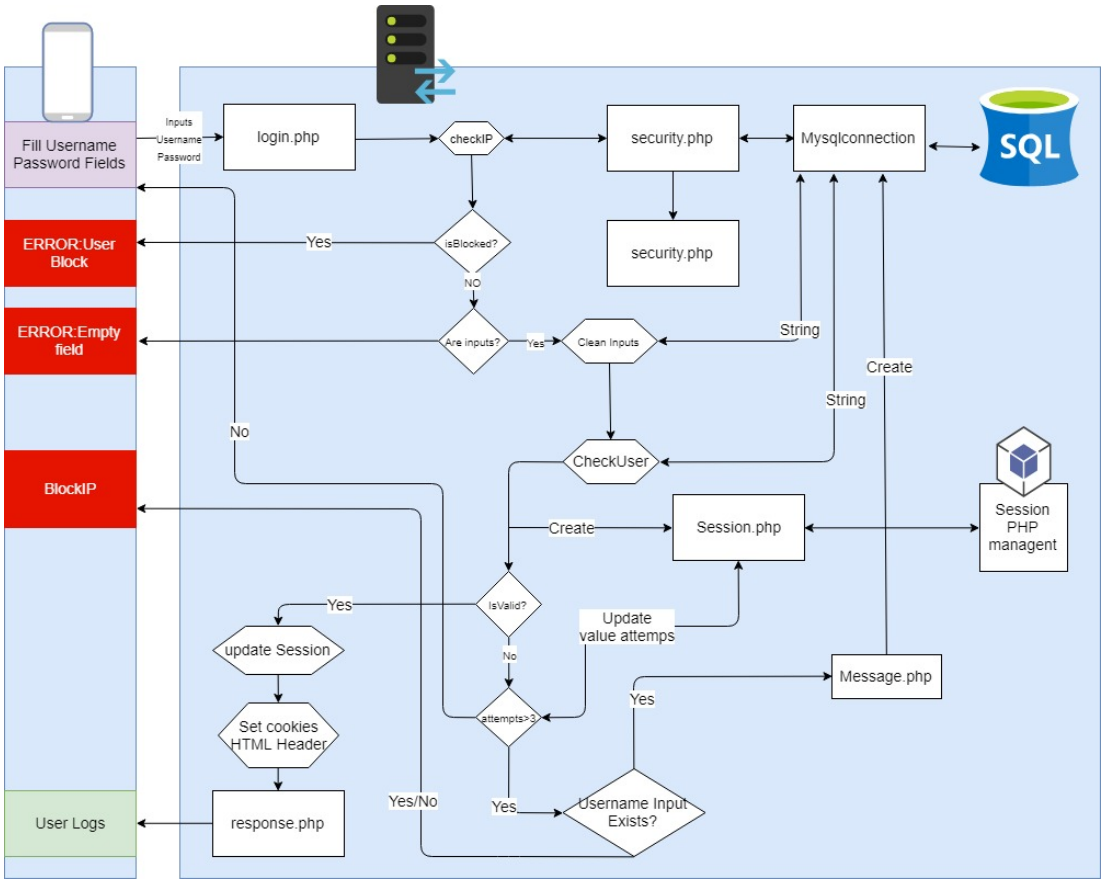

Figure 4.5: Login diagram. It describe the login action by a user. The followed step once the user side connect with the server are: checking if the request comes from a ip blocked previously, if not report an error; if inputs are corrects (exists two parameters: user and password) these are cleaned (avoiding sql-injection attacks), a session is created, and the parameters are verified in DB(checkUser).Finally, if user exist and password is correct the system update the session and create an appropriate HTML/JSON response, if not, the user can try to log twice times more, if user run out the attempts, system adds the ip a ipblock table in DB and check if the user name exists in order to send a message alerting him.

and method requested. Figure 4.6 shows a diagram of processes related with this class. Some examples of all possible requests over a resource are:

  - Method GET. https://10.12.102.61/success/outcomes
  - Method GET. https://10.12.102.61/success/outcomes/2
  - Method PUT. https://10.12.102.61/success/outcomes
  - Method PUT. https://10.12.102.61/success/outcomes/2
  - Method POST. https://10.12.102.61/success/outcomes
  - Method POST. https://10.12.102.61/success/outcomes/2 (INVALID)
  - Method DELETE. https://10.12.102.61/success/outcomes
  - Method DELETE. https://10.12.102.61/success/outcomes/

  PUT and POST methods have to include a JSON string in the request to have success. Depending on the resource requested and method the JSON object must have different input values that correspond with the table resource fields, however, each resource class filters and gets only the necessary inputs as well as checks them.

  POST JSON parameter example to create a new row in outcome resource table:

  {”iduser”:”1”,”datetime”:”2018-05-14 14:21:40.984”,”type”:”wandering”,”value”:”0”}

- **logout.php** deletes the current user session from the server avoiding later connections until new log in successful. See figure 4.7.

- **Security.php** contains methods related with security: as checking the user role using session.php, checking if the ip is usual for the user or blocking in case the user attempt to login wrong three times. Also implements a system to increment the waiting time after a wrong login attempt in order to avoid brutal force Attacks. This file builds the response header to the client where security measures as no-cache, expired connection, etc, are included to avoid subsequent attacks from sniffing.

- **Session.php** manages session objects, although no necessary because PHP engine manages that, this class allow a session manage clear in the code getting it been cleaner and understandable.

- mysqlconection.php is the bridge between different resources classes and queries to database. Also, it implements a method (using a native Mysql function)to clean the string from JSON in order to avoid SQL injection attacks.

- **jsonResponse.php** manages the response from server.php, ensuring that the server always returns a JSON string understandable by requester devices.

- **resourceAbstract.php** is an abstract class. Each resource defined in the application has their own PHP class that hierarchy from resourceAbstract.php and renders a DB object which is need to manage through the API. Each class manages and controls the security as whether a resource detects an user request without correct rights or the number of input parameters (JSON object) is incorrect, being, thereby, a invalid request and returning the appropriate error response. The different resource classes are loaded dynamically depending on the request. The resources accessible in the applications are:

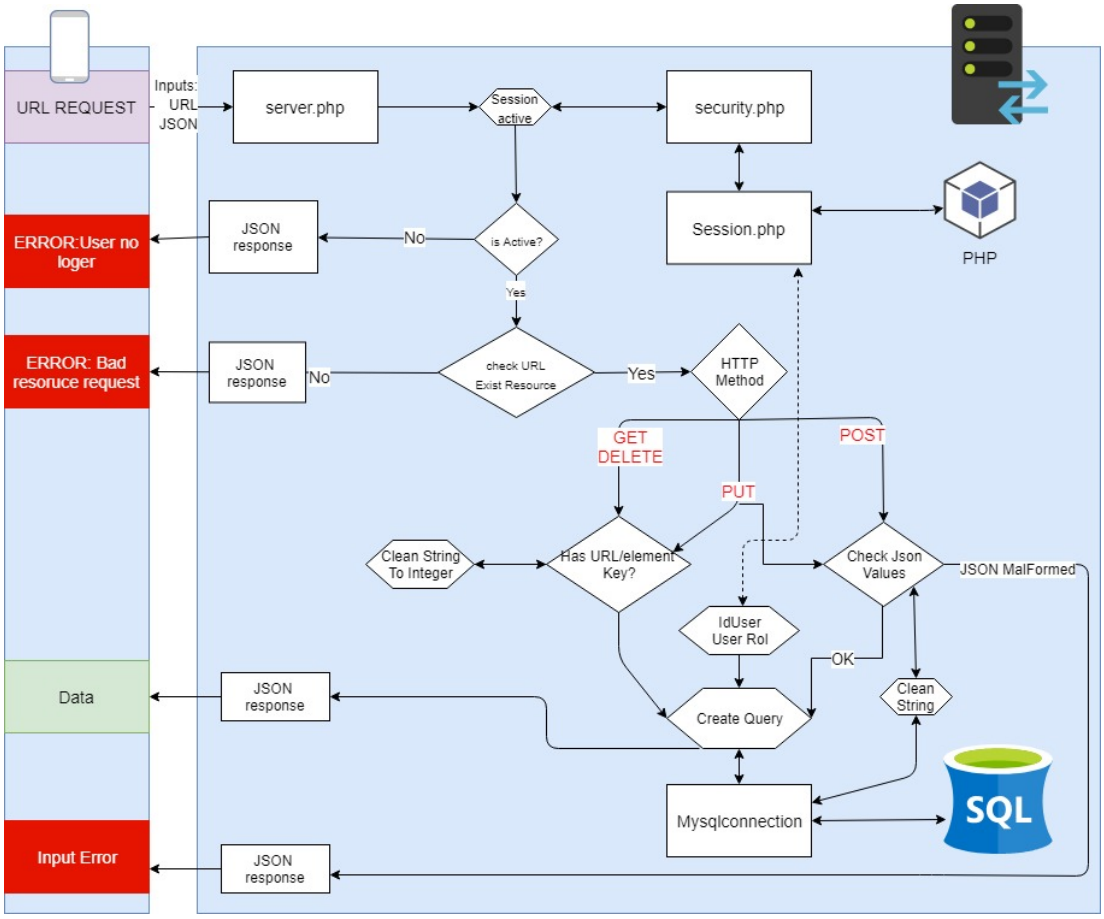

Figure 4.6: Server flow process diagram. All information resources request are managed by server.php. Firstly it checks if the session exists for the current user requests and if is so, it checks the request url with resource name, but if either of the processes are wrong, the server return a JSON message error. Next, system calls the appropriate resource class (extract from URL) and function (from method parameter) requested, clean the inputs and depending on sort of method it checks the URL id resource. As all DB user information are associated with an user id, the resources classes extract the id from the object session and generate a query to run in Database. Finally, Server returns a JSON response regardless the response from SQL engine.

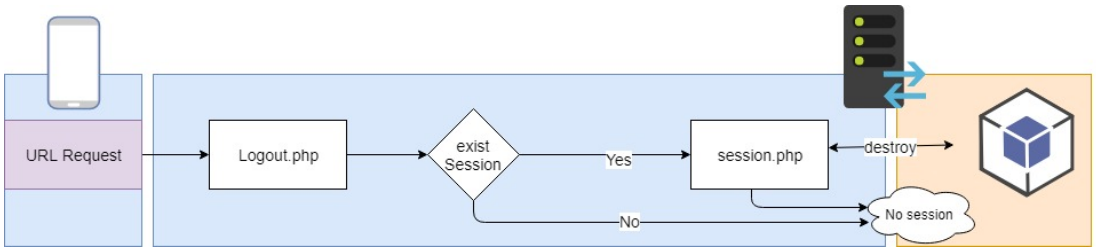

Figure 4.7: Logout just destroys the associated user session in PHP server, avoiding future connection to server.php at least a new login be succeeded.

- **Outcomes:** Represents the results from user processes, as results from Mreasoner or other result desired stored in database as Emotional Test which is an example of result input from mobile APP launching by the user.
- **Access:** Renders access of users to view primary users (PU) outcomes. See figure 4.8.
- **AccessRequest:** Manages the access request from users to PU. See figure 4.8.

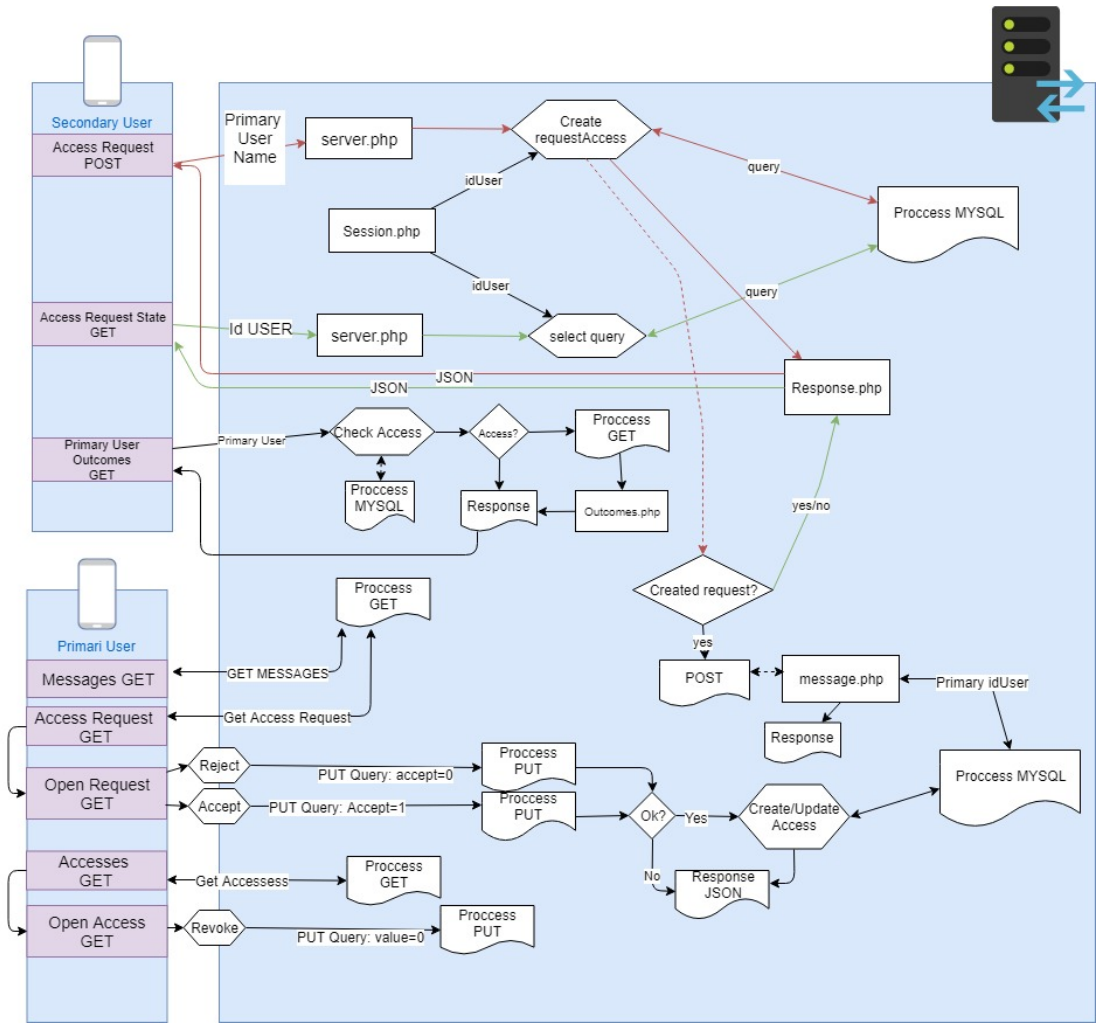

Figure 4.8: Access and AccessRequest management flow process diagram. Basically, an user with role different of 1 (no PU) can create an access request in order to view the requested user outcomes. Once, the application is done, the PU receive a message and also can checks in his app the pending requests where he can accept or deny the access. Once accepted, the user who requested will be able to access the user outcomes, but this PU can revoke the access through the app. Although, this kind of process could be considered internal to the server, the used methodology to create or modify these resources is through CRUD API which purpose is simplify and standardize the code.

- **Messages:** Renders the message object. See figure 4.9
- **Users:** Although this element is not accessed by any outer device, it helps to simplify the code and standardize it.

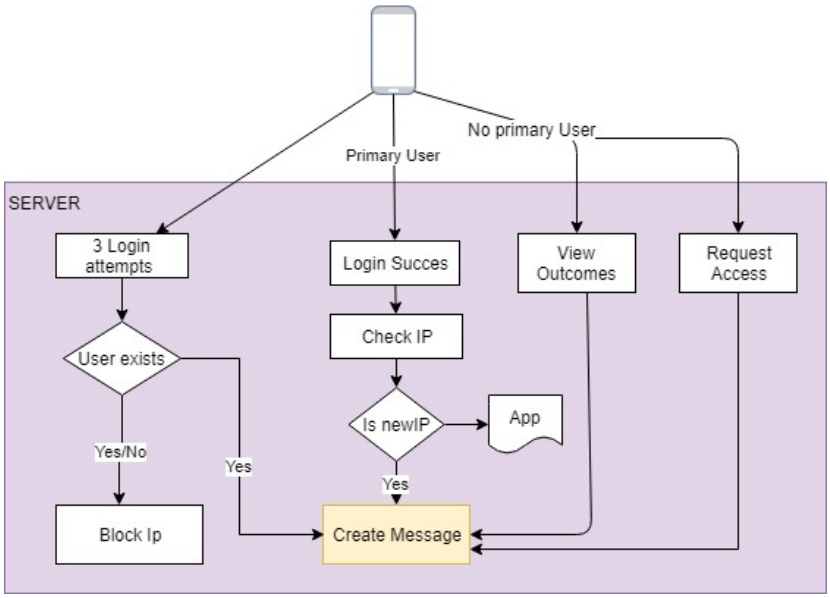

Figure 4.9: Messages process diagram. There is three events that create messages for PU: if someone try to log in the server with wrong credentials and the username matches with PU who will receive the message; if PU logs in the server using a different IP (each successful PU login generates a entry in DB associating IP and User); when a no PU views PU overcomes or create a access request for the user.

Each user has assign a role in order to allow a general management over the security dealing with users. Thus PUs have role 1, secondary users have role 2, so on. Thereby the security control just has to mind in consideration the role and differentiating between role 1 and other role in order to request resources. Table 4.1 summarizes next description about users authorisations on each resource depending on the requested method:

- Outcomes
  - GET: Only PUs can retrieve their own outcomes. Other users (rol¡¿1) can retrieve the PU outcomes when they have permissions, that is, if it exists a row in access table which relates PU with other and the value field is true.
  - PUT: No one can update/modify.
  - POST: Just PUs can create outcomes.
  - DELETE: Only PUs can delete their results.
- Access
  - GET: Only PUs can check who has access to theirs data.
  - PUT: PUs can modify access to revoke.
  - POST: No one is allowed use this method. Just the system can create this resource when access request is accepted by a PU.
  - DELETE: No one.
- AccessRequest

  - GET: PUs can retrieve the access requests related to themselves.
  - PUT: Only PUs can modify one of this sort of resource in order to accept or denied the request.
  - POST: Only users with a role¡¿1 can create.
  - DELETE: No one.

- Messages
  - GET: Only PUs can get messages.
  - POST: No one, the system create the appropriate messages.
  - PUT: Only PUs can modify the message status to read.
  - DELETE: No one.

- Users
  - GET: No one. Just the system manages this table.
  - POST: No one.
  - PUT : No one.
  - DELETE: No one.

Resource \ Method	GET	POST	PUT	DELETE
Outcomes	PUs/OUA	None	PUs	PUs
Messages	PUs	System	PUs	None
Access	PUs	System	PUs	None
AccessRequest	PUs	OU	PUs	None
Users	System	None	None	None

PUs: Primary Users
OUA: No primary Users with authorization from PUs
OU: No primary users
None: No one external to system
System: internal code management

Table 4.1: Resources accessed by HTTP methods.

4.3.3 Database-management system

The database engine is running in server is MySql 5.7.16 where is placed ‘cloud_server’ database. The engine accepts connection to 3306 port from out of the server. There are two database roles created: administrator user who has full control on database and a specific user to access only to ‘cloud_server’ database scheme with restriction. This user can create, modify, update and delete rows from public scheme, having no access to meta-data which is used by API to connect to database. Database scheme is very simple on stand by for future complex requirements. There are created primary and foreigner keys but there are not complex relations. or complex rules, between tables, and database information is not encrypted at the moment. Figure 4.10 shows the ER diagram database scheme.

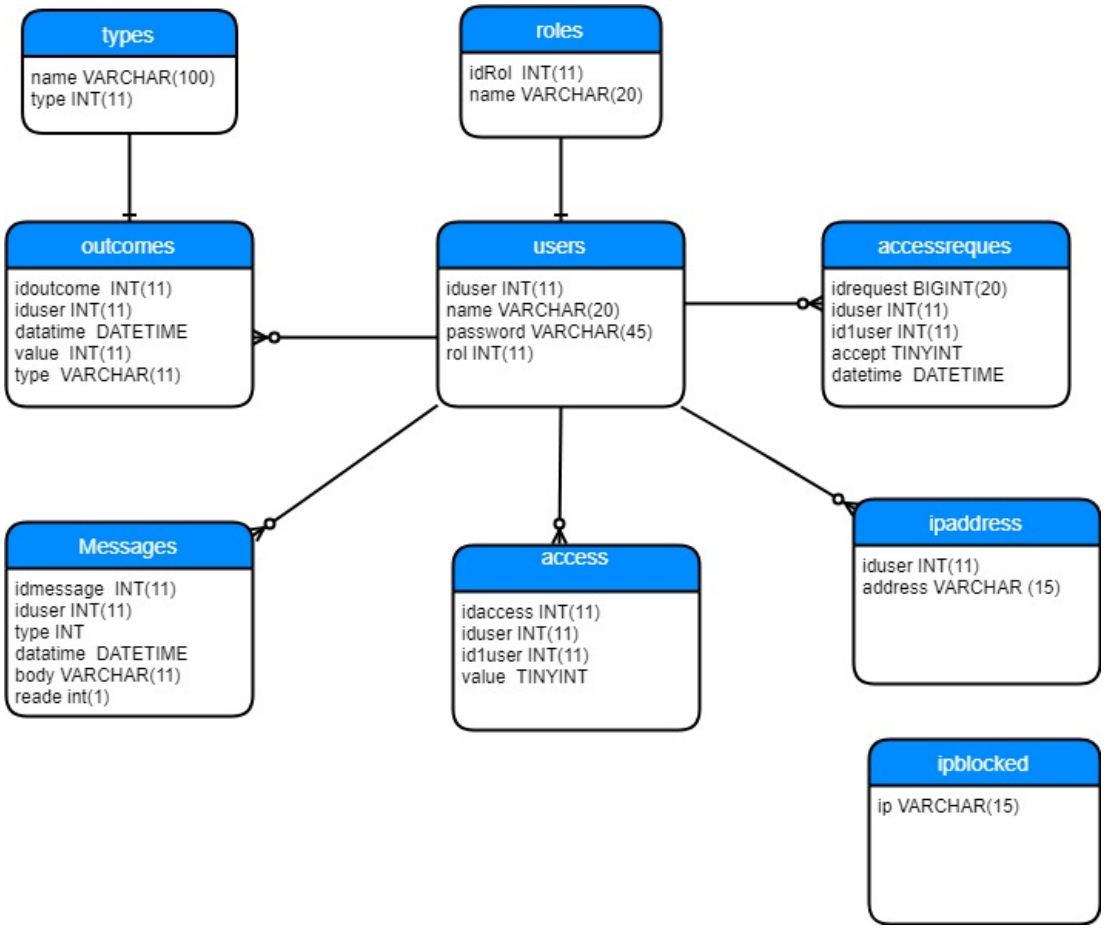

Figure 4.10: Scheme Cloud_server database ERR diagram. The main entity is user and all other entities, which represent a application resource, are related with it.

4.4 Processing Module

The processing module consists in a laptop Dell Latitude 5580 (Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz, 2901 Mhz, 2 Core(s), 4 Logical Processor(s) and 16GB of memory) where is running the reasoning tool MReasoner. The SO is a Microsoft Windows 10 Enterprise 2016 LTSB with Bit-locker activated and there is a unique SO user with administrator role. All security laptop configuration were done by MDX IT department as it is University property. As MReasoner needs a database to run there is a PostgreSQL database management installed in the laptop.

Figure 4.11: Laptop working as processing module.

MReasoner

MReasoner is built with JavaSE JRE 1.8 and developed by GOODIES team. Next it is given a brief description about how it works without technical details. The application MReasoner runs over a set of timing rules written for each action it wanted to detect. These rules are based in a logical inference in one or more binary states. Mreasoner polls Vera hub each second checking whether the sensors, defined into the code as states and called independent states, change (e.g.: motion sensor is represented by an independent state which set its value in function of the real value of motion sensor 1= detect movement, 0= no movement). The rules get as result new values for other defined states, termed dependent states, e.g: 'wandering state in the house' is a dependent state searched. All changes in the defined states are stored in MReasoner the database, and, those dependent states defined as searched action, sent to the cloud server. There is technical documentation available about the application if necessary.

Database-management

The database manager used by Mreasoner is a PostgreSQL 9.6. The access from outside the laptop is blocked and there is only a database scheme in the system that it is only used by Mreasoner which connects to database using the unique role created to access since MReasoner needs this grant to work.

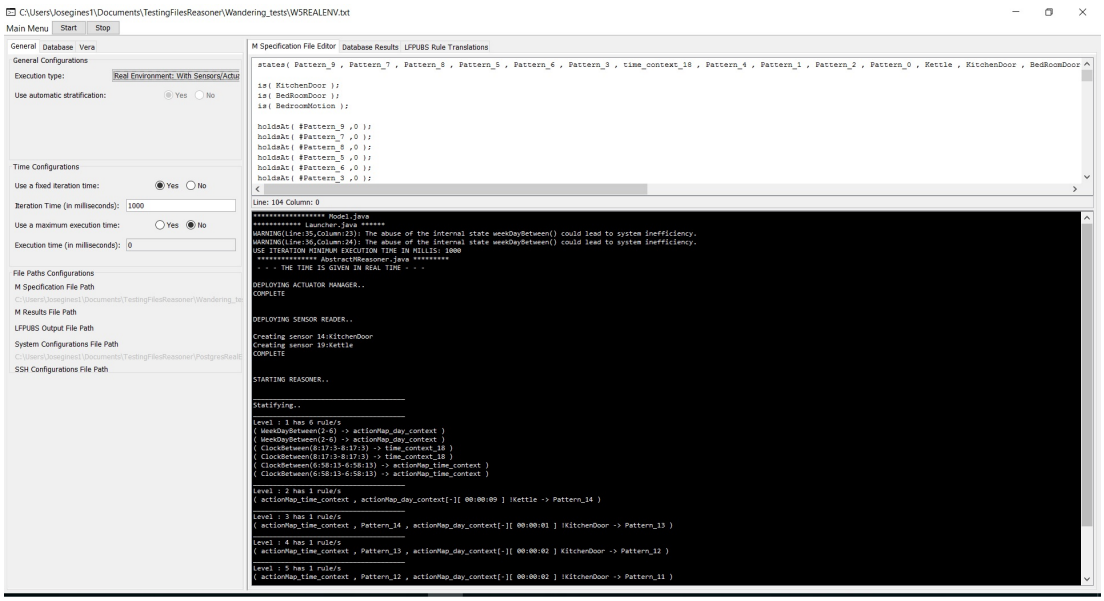

Figure 4.12: Mreasoner running, screen capture.

4.5 Mobile

The mobile used in Pilot 1 is a Tablet Nexus 7 model with Android 6.0.1 with any special configuration except the initial unlocking screen pattern. The App is developed with Android Studio Java 1.6 and Kotlin 1.1. It is a simple application to show the interaction between mobile user and server. Note that the communication of the application with the server is encrypted (HTTPS) and use the cookies to keep sessions after user successfully logs in, so the app just requires permission to 'Internet access' from the Android system. The current android applications just works on android 6 or higher. Whether more information about the code is needed, it can be provided. Figure 4.14 shows the different interface screens and application flow .

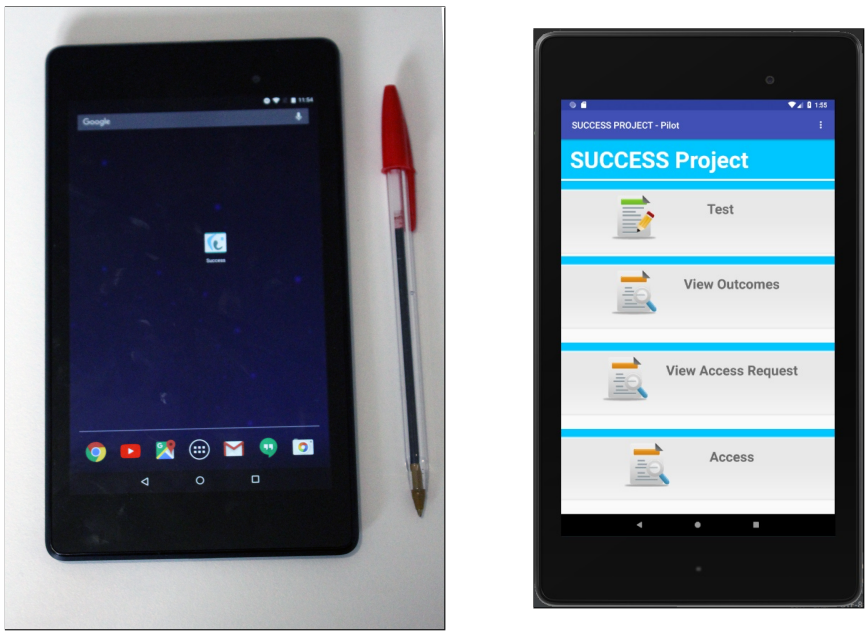

Figure 4.13: Tablet Nexus 7 used in Pilot.

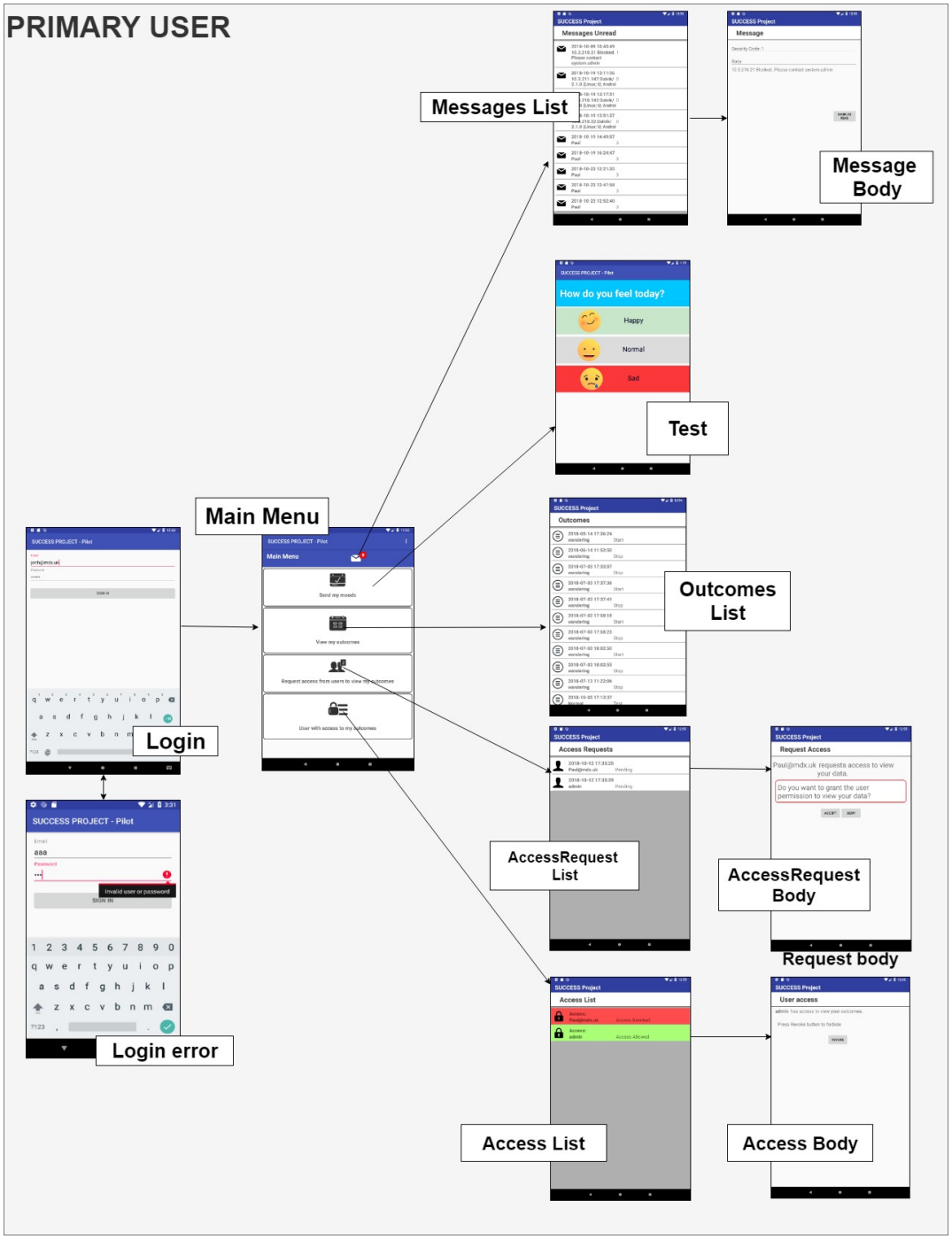

Figure 4.14: PUs Android Success Application flow. After a successful login, a PU can choose between among resources defined in the system. Each menu entry shows a resource list that allows different actions in each list item, except "Emotional test" which represents the data sent from mobile and just send the test result or choice.

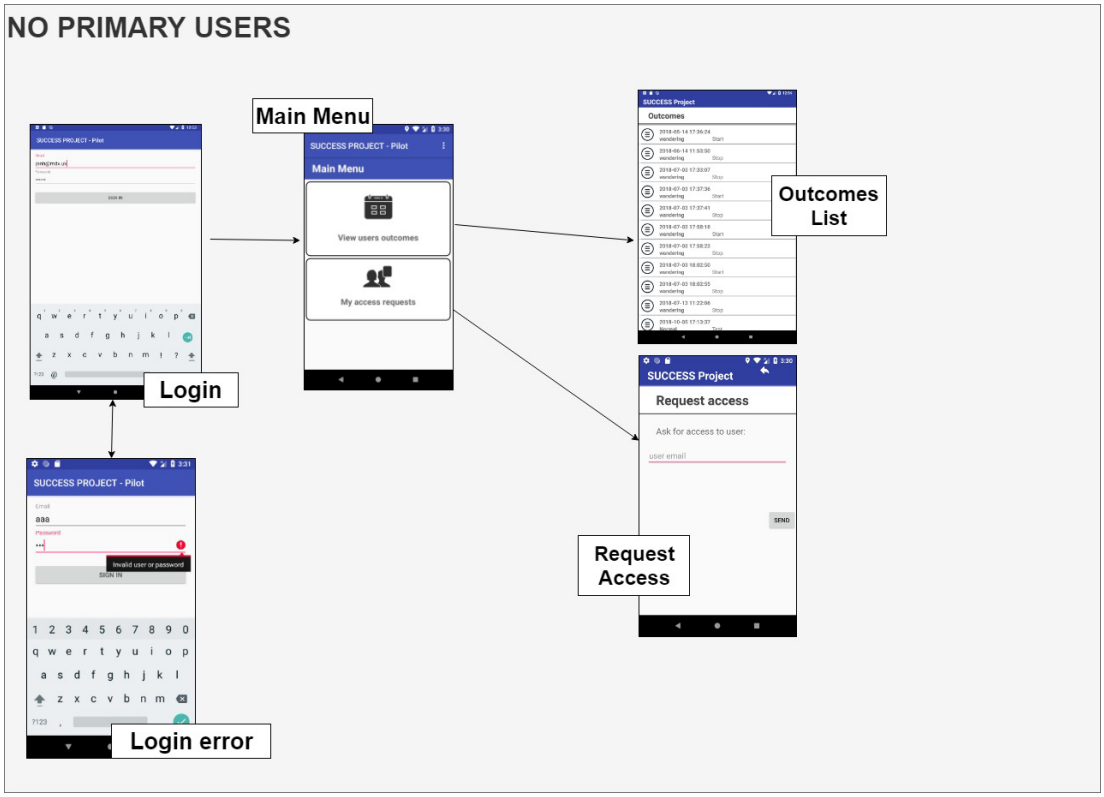

Figure 4.15: Android Success Application Flow for other users. No PUs just have the possibility to view PU outcomes previous PU authorization, and to request access to PU for the authorisation to view the outcomes.

5 Annex

5.1 Pilot 1 security measures

Here is presented some general and usual security measures carried out in each system platform within the global system. Some of them are not implemented yet for development reason to facility the deployment, but it is interesting to add in order to get a guideline of Pilot 1 security. The document is divided in different sections for each element where it is possible setting up security measures, which have been gathered from different developers' references [2] [1] [3] .

- Gateway (router- SO firewall)
  - Block unused and dangerous ports and services in router, SO firewall and IIS Server.
  - Force connection to use SLL connections with HTTPS and digital certificate.
  - Avoid/block HTTP connections in web server, allowing only HTTPS.
  - Create ACL system and configure it according with the system functionality.
  - Block url access to folder/files in the server that are not related with the application.
- PHP Server: general security recommendations to configure PHP.ini

Setting	Description
allow_url_fopen=Off allow_url_include=Off	Disable remote URLs (which may cause code injection vulnerabilities) for file handling functions.
register_globals=Off	Disable register_globals.
open_basedir="c:/inetpub"	Restrict where PHP processes can read and write on a file system.
safe_mode=Off safe_mode_gid=Off	Disable safe mode.
max_execution_time=30 max_input_time=60	Limit script execution time.
memory_limit=16M upload_max_filesize=2M post_max_size=8M max_input_nesting_levels=64	Limit memory usage and file sizes.
display_errors=Off log_errors=On error_log="C:/path/of/your/choice"	Configure error messages and logging.
fastcgi.logging=0	Internet Information Services (IIS) FastCGI module will fail the request when PHP sends any data on stderr by using FastCGI protocol. Disabling FastCGI logging will prevent PHP from sending error information over stderr, and generating 500 response codes for the client.
expose_php=Off	Hide presence of PHP.

- PHP API development
  - A unique main class publicly accessible and able to manage the other classes that access to different resources.
  - Managing only the CRUD methods request blocking another kind of requests and redirecting in order to the resource requested.

  - Check the user roles in each CRUD method request.
  - Activate session and cookies management.
  - Add in responses a security header in order to use by sniffing users the session or cookies.
  - Check inputs: JSON well-formed, check correct field value (whether request expects a integer check that is an integer), clear inputs avoiding script embedded in inputs and check the length with the max length a data needs.

- Database Server
  - Create different user roles.
  - Review ACL users.
  - Encrypt password.
  - Forbidden connection from outside of server.

- Vera
  - Create username and password.
  - Encrypt sensor communication with smart hub.

5.2 Initial requirements achievement

Following the initial list of requirements checked by all teams, Pilot 1 has been developed under these terms. Although some of them could be improve or modify in order to new needs or system changes from other teams needs. At the moment, mostly are developed under generic standards of security awaiting for new requirements reviews. Some requirements belong to Pilot 2 but for different reasons, as design or functionality, they have already been implemented, although they can be improved.

- System architecture requirements
  - SA1: All platforms should support connectivity. All platforms have internet access through MDX connection: server, mobile app and laptop.
  - SA2: Support data availability and integrity. Information from the process are centralized and data store in a database in the main server. This server complies with availability (except in very specific cases, e.g. general power failure) and integrity with secure measures in different levels.
  - SA3: Servers should Support web services APIs and manage databases. Through Apache server and PHP engine gateway the sever satisfies this.
  - SA4: All platforms and components must have user authentication. Actually, phone, laptop and server have the default user authentication system specific of each system, that it is enough at the moment. These kind of requirements could be improved in the future.
  - SA5: Support the environment runs home-based data processing unit. Laptop has installed the reasoning application and it has been tested within whole environment.

- SA6: Support integration of Attack Tree (AT) tool. We need information about AT tools which is not yet been provided.
- SA7: Support patients to view their information (Pilot 2). A user can request his/her information from the server through a developed mobile APP or web browser.
- SA8: Support Multiple Users with different roles (Pilot 2). PHP API has authentication system for user signed and a user/role management to control the access
- SA9: Should define ACL and user's roles for each platform. (Pilot 2). Should define ACL and user's roles for each platform. All platforms have a system to manage user roles and security, some of them are implemented and other are ready to do it.
- SA10: Should establish an SSL protocol between all component connections (Pilot 2).The connections between server-laptop-mobile are encrypted by HTTPS.
- SA11: Primary user can configure data access (Pilot 2). The system includes a simple management for primary users to manage this feature.

- Functional requirements
  - Fun1a: The system should be able to detect unhealthy wandering. It is developed and working.
  - Fun1b: The system should be able to detect unhealthy sleeping. It is developed and working.
  - Fun1c: The system should be able to detect unhealthy eating. It is developed and working.
  - Fun1d: The system should be able to detect frequent forgetting. This requirement is not accomplish due the complexity, however it is possible add as many user behaviours to the system as we need. At the moment, we consider the previous ones enough.
  - Fun2a: Each time a meaningful event is detected should be stored. The reasoning tool in laptop stores the information and send to server.
  - Fun2b: Each time the activity log is updated the Cloud server should be informed through web Service. Previous gather information is send to the server once an event happen where is stored too
  - Fun4: Forbid the Primary user access to the smart hub. Is needed username and password.
  - Fun5: Forbid Primary user to change access configuration and stored outcomes. All platform has authentication system. The result only can be written by application or database administrator user.
  - Fun6: Avoid different users retrieve information. In addition to default security measures of each platform, a non-administrator user cannot access to information unrelated with him/her or without the express permission of the primary user (primary users can manage grants for second users to allow or deny access to user data).

- No-functional requirements
  - NF1: Support web services API RESTful and manage database. The API is developed, tested and working.

  - NF3: Only admin user has access to OS. Each element have an authentication system.

- Design requirements: Primary user can configure access rights. It is developed in mobile phone, but the API allow to use any browser too.
  - D1: It has already been an initial prototype to do that. Primary user allow or deny access to another users who requested access previously.
  - D2: Mobile/web Interface should be accessible and easy to browse. The current version is very simple.

- Hardware requirements
  - H1: The system should include at least one laptop and an Android based smart-phone in the patient's house. This requirement is satisfied.

6 Conclusion

Pilot 1 is finished and complies with initial requirements, thus it is ready to carry on whit SUCCESS proposed aims. A remaining practical challenge is a closer integration with the earlier phases of the SUCCESS process as described in M2.2.

Bibliography

[1] M. S. Chris Snyder, Thomas Myer. *Pro PHP Security*. Apress, Berkeley, CA, second edition, 2010.

[2] P. group. Php manual: Security, 2002. Location: https://secure.php.net/manual/en/security.php.

[3] T. Smith. Secure php with configuration settings, 2002. Location: https://docs.microsoft.com/en-us/iis/application-frameworks/install-and-configure-php-on-iis/secure-php-with-configuration-settings.