Virtual coaching to support a healthy and meaningful life of older adults and employees in their retirement process

Call: AAL-JP call 2018

Grant Agreement Number: aal-2018-5-92-CP

Deliverable

D3.6 AgeWell API Manual

Deliverable type:	Report
WP number and title:	WP3: Technical Development
Dissemination level:	Confidential
Due date:	Month 20 – 30 November 2020
Lead beneficiary:	AIT
Lead author(s):	Johannes Kropf (AIT), Elena Rostovtseva (Med)
Reviewers:	Niklas Hungerländer (AIT)

This project AgeWell has received funding from the AAL Joint Pgoramme under grant agreement No aal-2018-5-92-CP.

Document history

Version	Date	Author/Editor	Description
0.1	13.09.2020	Johannes Kropf	Initial version
0.2	18.10.2020	Elena Rostovtseva	Input Medrecord
0.3	20.10.2020	Cornelia Schneider	Input FHWN
1.0	25.10.2020	Johannes Kropf	Final version

Table of Contents

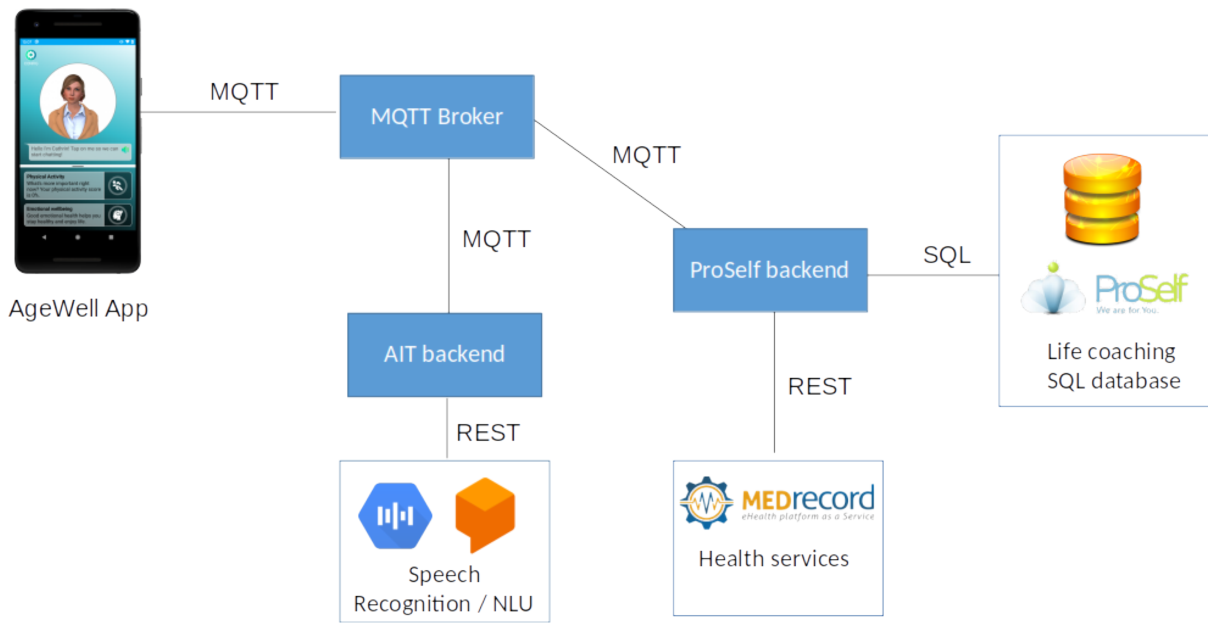
1 EXECUTIVE SUMMARY

AgeWell is a distributed system with components from 3 technical partners. Hence, it is not a monolithic solution which can be used by third parties out of the box. Nevertheless, third party developers are able to extend the functionalities in different ways. The core component from the end-user's perspective is the AgeWell Android app, hence, new functionalities or features are dedicated to this component. The App communicates with the backed services via a centralized messaging system (MQTT), hence there is no API (e.g. REST interface), but a messaging protocol is defined and documented in D3.5.

The MedRecord care plan backend is not connected via MQTT, but via an API over the ProSelf backend server. In this document, mainly this API is described.

2 INTRODUCTION

The components of the AgeWell system are connected via the MQTT protocol, which is a centralized message queuing instance. Only the existing MedRecord platform and third party applications like Google Dialog Flow or Firebase Messaging are connected via RESTful APIs. In xxx an overview is given how the AgeWell components interact with each other. In the following sections of the documents, the MQTT Broker and the Medrecord API is described.



3 MQTT BROKER

In the AgeWell project, the Mosquitto MQTT Broker with version 1.6.9 is used (https://mosquitto.org/). For connecting to the broker, certain client libraries are used. For Java, as used by AIT and ProSelf), the Eclipse Paho MQTT client library is used (https://www.eclipse.org/paho/).

MQTT is a light weight publish/subscribe messaging protocol, originally created by IBM and Arcom (later to become part of Eurotech) around 1998. MQTT is an OASIS standard. The latest version is 5.0 and is available in a variety of formats. MQTT 3.1.1 is also an ISO standard (ISO/IEC 20922).

PUBLISH/SUBSCRIBE

The MQTT protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a broker and subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. The broker and MQTT act as a simple, common interface for everything to connect to.

TOPICS/SUBSCRIPTIONS

Messages in MQTT are published on topics. There is no need to configure a topic, publishing on it is enough. Topics are treated as a hierarchy, using a slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a filesystem. For example, multiple computers may all publish their hard drive temperature information on the following topic, with their own computer and hard drive name being replaced as appropriate:

- sensors/COMPUTER_NAME/temperature/HARDDRIVE_NAME

Clients can receive messages by creating subscriptions. A subscription may be to an explicit topic, in which case only messages to that topic will be received, or it may include wildcards. Two wildcards are available, + or #.

+ can be used as a wildcard for a single level of hierarchy. It could be used with the topic above to get information on all computers and hard drives as follows:

- sensors/+/temperature/+

As another example, for a topic of "a/b/c/d", the following example subscriptions will match:

- a/b/c/d
- +/b/c/d
- a/+/c/d
- a/+/+/d
- +/+/+/+

The following subscriptions will not match:

- a/b/c
- b/+/c/d
- +/+/+

# can be used as a wildcard for all remaining levels of hierarchy. This means that it must be the final character in a subscription. With a topic of "a/b/c/d", the following example subscriptions will match:

- a/b/c/d
- #
- a/#

- a/b/#
- a/b/c/#
- +/b/c/#

Zero length topic levels are valid, which can lead to some slightly non-obvious behaviour. For example, a topic of "a//topic" would correctly match against a subscription of "a/+/topic". Likewise, zero length topic levels can exist at both the beginning and the end of a topic string, so "/a/topic" would match against a subscription of "+/a/topic", "#" or "/#", and a topic "a/topic/" would match against a subscription of "a/topic/+" or "a/topic/#".

QUALITY OF SERVICE

MQTT defines three levels of Quality of Service (QoS). The QoS defines how hard the broker/client will try to ensure that a message is received. Messages may be sent at any QoS level, and clients may attempt to subscribe to topics at any QoS level. This means that the client chooses the maximum QoS it will receive. For example, if a message is published at QoS 2 and a client is subscribed with QoS 0, the message will be delivered to that client with QoS 0. If a second client is also subscribed to the same topic, but with QoS 2, then it will receive the same message but with QoS 2. For a second example, if a client is subscribed with QoS 2 and a message is published on QoS 0, the client will receive it on QoS 0.

Higher levels of QoS are more reliable, but involve higher latency and have higher bandwidth requirements.

- 0: The broker/client will deliver the message once, with no confirmation.
- 1: The broker/client will deliver the message at least once, with confirmation required.
- 2: The broker/client will deliver the message exactly once by using a four step handshake.

RETAINED MESSAGES

All messages may be set to be retained. This means that the broker will keep the message even after sending it to all current subscribers. If a new subscription is made that matches the topic of the retained message, then the message will be sent to the client. This is useful as a "last known good" mechanism. If a topic is only updated infrequently, then without a retained message, a newly subscribed client may have to wait a long time to receive an update. With a retained message, the client will receive an instant update.

Structure of AgeWell MQTT messages

In general, MQTT messages are structured in the JSON format. In AgeWell, all messages follow a common structure with mandatory fields as shown below:

```
{

"topic": "eu/agewell/event/reasoner/<TOPIC>/<SUBTOPIC>",
"properties": {
      "LANGUAGE_CODE": "<LANGUAGE CODE>",
      "SOURCE_ID": "<PACKAGE NAME>",
      "DIMENSION_ID": <DIMENSION ID>,
      "CLIENT_ID": "<CLIENT ID>",
      "TIMESTAMP": <TIMESTAMP>
      }
}
```

The <SUBTOPIC> indicates if a messages is intended to be sent to or sent by the user interface. By convention, for messages sent from the GUI to a backend service, the subtopic REQUEST is used, for responses on a request sent back the GUI RESPONSE is used and for messages, which are sent without a request from the GUI (or the user), the subtopic MESSAGE is used.

The other mandatory fiels are:

LANGUAGE_CODE: the two digit language code the content es provided (e.g. en for Englisch, it for Italian, de for German and nl for Dutch.

SOURCE_ID: An id indicating the source of the message. By convention, the package name is used. e.g. at.ac.ait.hbs.agewell.demo.server.package.

DIMENSION_ID: The targeted dimension. By convention, 1...physical activities, 2...mental health, 3...social activities, 4...health literacy, 5….for retirement. The dimension id corresponds to the sections of the App dashboard.

CLIENT_ID: This is a unique id of the MQTT client, e.g. a UUID

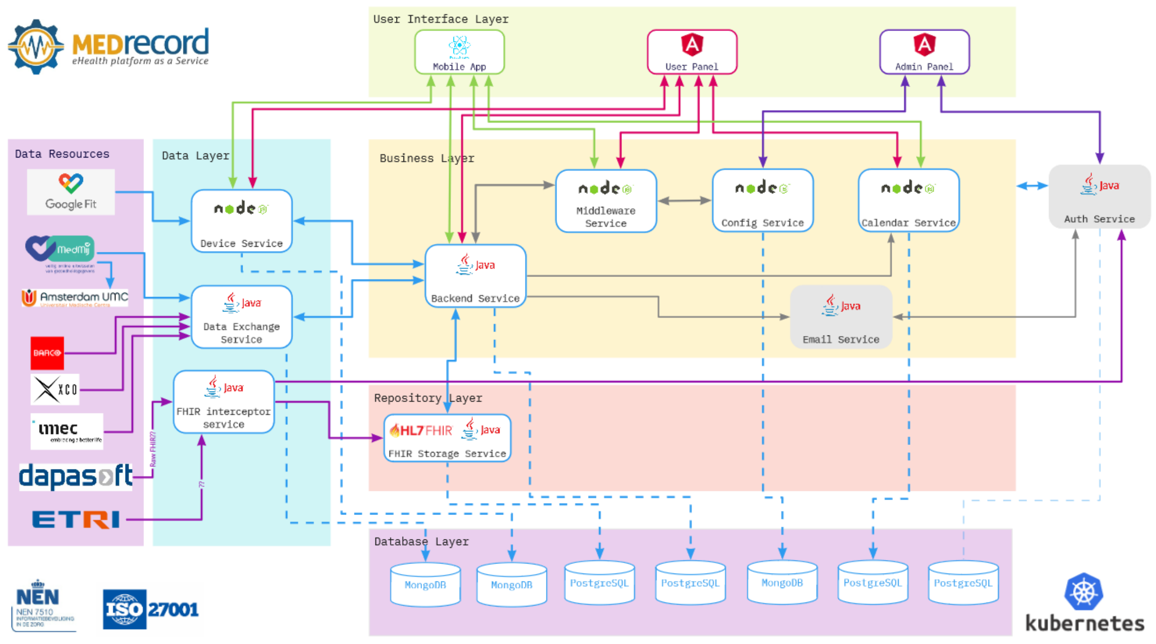
TIMESTAMP: The timestamp the message has been generated in milliseconds since 1.1.1970 (Unix Epoche time).

The remaining part of the JSON message is specific to the topic. The full list of topics and its properties is described in detail in Deliverable 3.5.

4 THE MEDRECORD API

The MEDrecord platform is completely based on a microservice architecture, we offer them as separate services. For clinical or healthcare related data we are offering two microservices that query exactly the same backend, so the result will be the same:

1. Data exchange service: simplified clinical REST based API
2. Interceptor service: FHIR API, these are the more complex FHIR queries that you can certainly use.



Here you can find technical information on how the MEDrecord API works: https://sites.google.com/medrecord.io/helpcenter/home?authuser=1

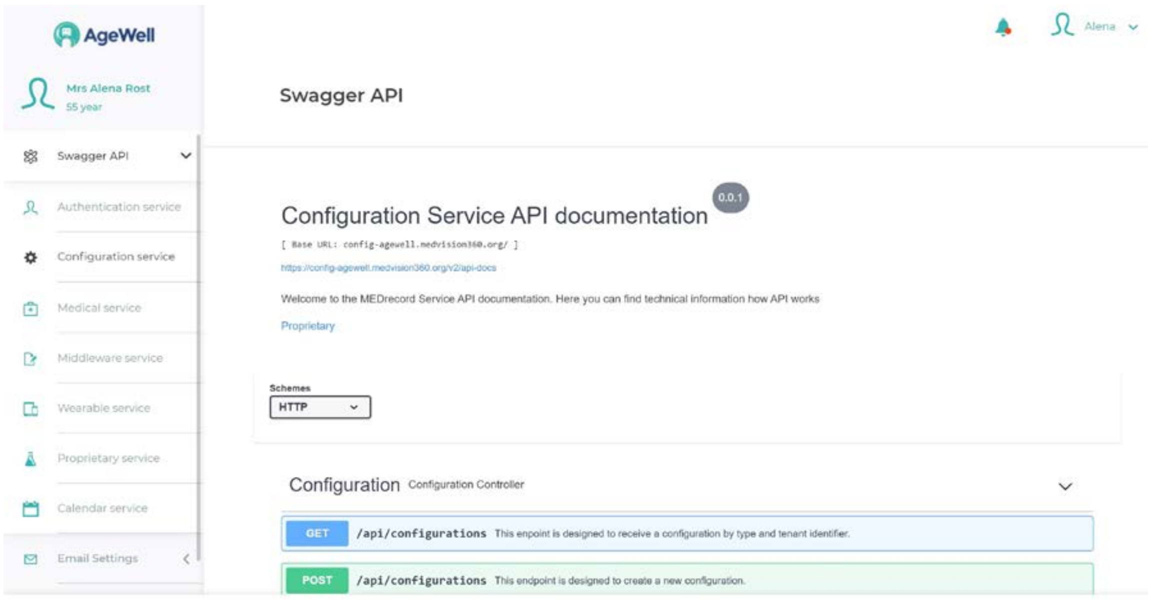
Most of the information can be found inside the Admin panel https://admin-agewell.medvision360.org/

In order **to become admin** and access the MEDrecord APIs please:

- Create an account on the patient portal
- Ping Jan-Marc/Edin that you did and you have access
- We will make you admin (manually)
- Only after that you are able to login at the admin panel.

Please note that you have to **choose first** if you would like to use the REST based API's inside the admin panel, or use our FHIR based interceptor service.

For the API we have created a GUI for the easy management of all data which can be accessed at the admin panel. For each that is shown inside the GUI there is also an API endpoint.



Swagger API

Configuration Service API documentation 0.0.1

```
[ Base URL: config-agewell.medvision360.org/ ]
```

https://config-agewell.medvision360.org/v2/api-docs

Welcome to the MEDrecord Service API documentation. Here you can find technical information how API works

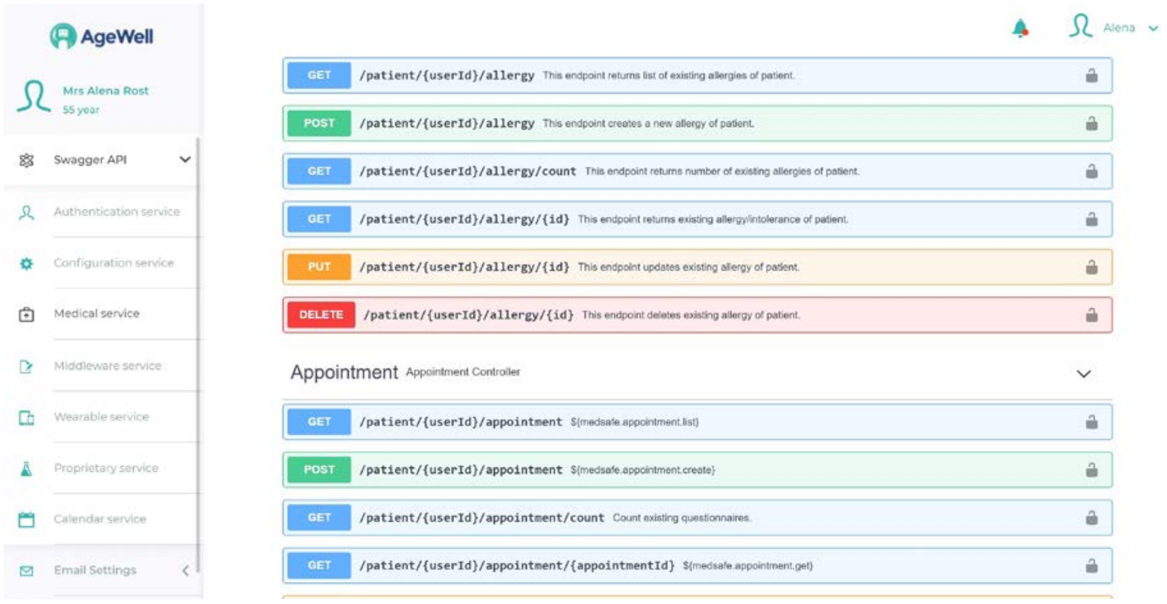
Proprietary

Schemes: HTTP

Configuration Configuration Controller

GET /api/configurations This enpoint is designed to receive a configuration by type and tenant identifier.

POST /api/configurations This endpoint is designed to create a new configuration.



GET /patient/{userId}/allergy This endpoint returns list of existing allergies of patient.

POST /patient/{userId}/allergy This endpoint creates a new allergy of patient.

GET /patient/{userId}/allergy/count This endpoint returns number of existing allergies of patient.

GET /patient/{userId}/allergy/{id} This endpoint returns existing allergy/intolerance of patient.

PUT /patient/{userId}/allergy/{id} This endpoint updates existing allergy of patient.

DELETE /patient/{userId}/allergy/{id} This endpoint deletes existing allergy of patient.

Appointment Appointment Controller

GET /patient/{userId}/appointment ${medsafe.appointment.list}

POST /patient/{userId}/appointment ${medsafe.appointment.create}

GET /patient/{userId}/appointment/count Count existing questionnaires.

GET /patient/{userId}/appointment/{appointmentId} ${medsafe.appointment.get}

Swagger API screen

For our whole API, we provide a Swagger approach of describing RESTful APIs. It serves a great purpose in the Admin Panel visualizing all the endpoints.

5 CONCLUSIONS

The AgeWell solution is developed as a modular and expandable solution. By using the MQTT protocol, every component can be extended or replaced by another one and the connection layer of each component does not deeply affect the services behind (encapsulation paradigm). For example, it is possible to use alternative UIs instead of the AgeWell app. One could, e.g. implement an interface to Amazon’s Alexa as the main UI without the need of changing the other components. In a similar way, the GUI could be placed on the robot for future use.