Test Scripts: REST

- Consent: Test Scripts: REST
- Identity & Access Management: Test Scripts: REST
- Therapeutic Exclusion: Test Scripts: REST
- Therapeutic Link: Test Scripts: REST

Execution summary

1. (Start the local application SUT - System Under Test)
2. Start the **ResponseRepository**
3. Execute **SeleniumRequest** / **AppiumRequest** to fetch data from the SUT
4. Stop the SUT
5. Start the **TokenGenerator**
   a. In a web browser, paste your IAM Connect access URL (e.g. https://api-acpt.ehealth.fgov.be/auth/realms/{myRealm}/protocol/openid-connect/auth?client_id={myClientId}&nonce={nonce}&redirect_uri=http://localhost:8080/login&response_type=code&scope=openid)
   b. Login with the same user than previously
6. Run the **Assertion** scripts

More information in projects' readme.md

1 Test architecture

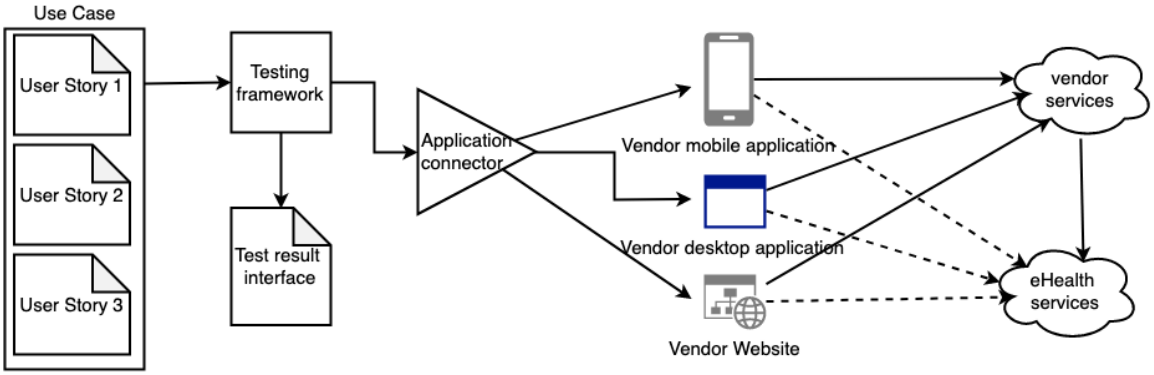
1.1 Logical view

The logical view provides a high-level description of the logical components of the architecture independently from any implementation. It eases the identification of the various components and mechanisms of the system to be implemented and the definition for each of them their technological role(s).

In the Figure below, the logical view of the test architecture is depicted. We describe this view from the left to the right. Each use-case is composed by one or multiple user story (ies). Each user story is used to (1) check the initial data, (2) to create user request and (3) to test if the result of the request is correct.

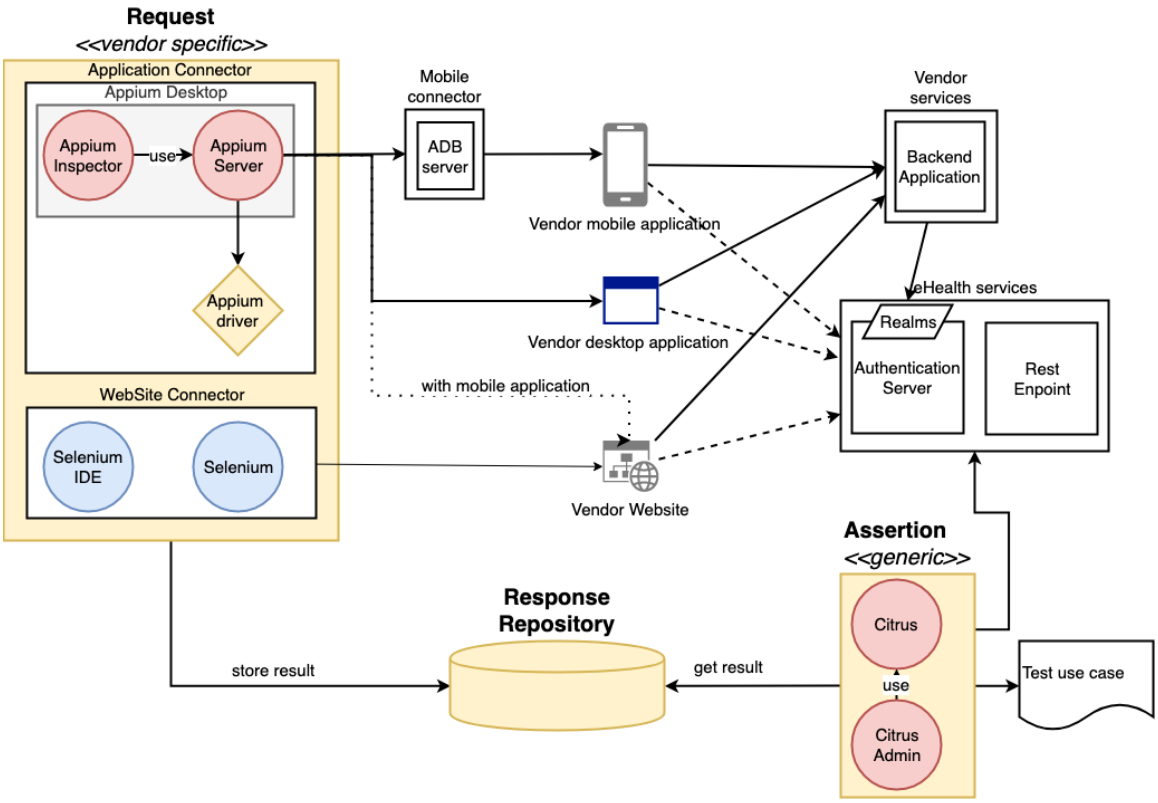
The application connector allows to make generic test scripts. The test script tests features and the device connector allows to run these scripts on any platform (Android version, iOS version, Windows desktop application,...).

The vendor application could be hosted on a website, a mobile application or a desktop application (application used by a practitioner). It is noteworthy that in this project only the mobile application will be implemented. The vendor applications get data from their own backend server or directly from the eHealth services.



1.2 Implementation view

The implementation view is used to describe the components of the architecture, focusing on the tools and technologies used to implement them (development environment, programming languages, protocols, third party software, …).



Request

The Request component is composed of two main parts: the Application connector and the Website connector. The Application connector integrat es Appium which is an open source test automation framework for use with native, hybrid and mobile web apps. It drives iOS, Android, and Windows apps using the WebDriver protocol. Appium receives HTTP requests from test scripts and translates these HTTP requests into actions on the application. Applications are controlled via the WebDriver protocol (formerly JSON Wire Protocol). There exist 2 versions of Appium:

- Appium Server: command line interface only
- Appium desktop: application for Mac/windows allowing to launch an Appium server and inspecting an application. While inspecting an ap plication, Appium generates a client code to automate action(s) of users. This client code could be imported into your test script.

Whereas the second part integrates with Selenium which is a portable framework for testing web applications. It provides a recording tool for authoring functional tests without the need to learn a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages. The tests can then run against most modern web browsers. Selenium deploys on Windows, Linux, and macOS platforms.

The Mobile connector component enables to connect the request component to the vendor mobile application. It integrates with Android Debug Bridge (ADB) which is a versatile command-line tool to communicate with a device. The ADB command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that you can use to run a variety of commands on a device. The ADB is included in the Android SDK Platform-Tools package. It allows you to connect to a physical device or virtual device (e.g. emulated by Android Studio).

Response repository

The response repository component makes the links between the generic tests (Assertion) and the vendor specific tests (Request). The request component runs and stores the results on the Response Repository component. The assertion component uses the data from the Repository component to make assertion with the results obtained from eHealth.

Assertion

The Citrus project is a Java project using the Citrus libraries.

Citrus is a test framework written in Java that enables automated integration testing of message-based EAI applications. The tool can easily simulate surrounding systems across various transports and protocols (e.g. JMS, SOAP WebServices, Http, TCP/IP, …) in order to perform end-to-end use case testing. Citrus provides strong validation mechanisms for XML message contents and allows to build complex testing logic.

The structure of the Java project will be split into use cases and each use case will be split into user stories. A user story defines a functionality that must be tested.

Vendor services

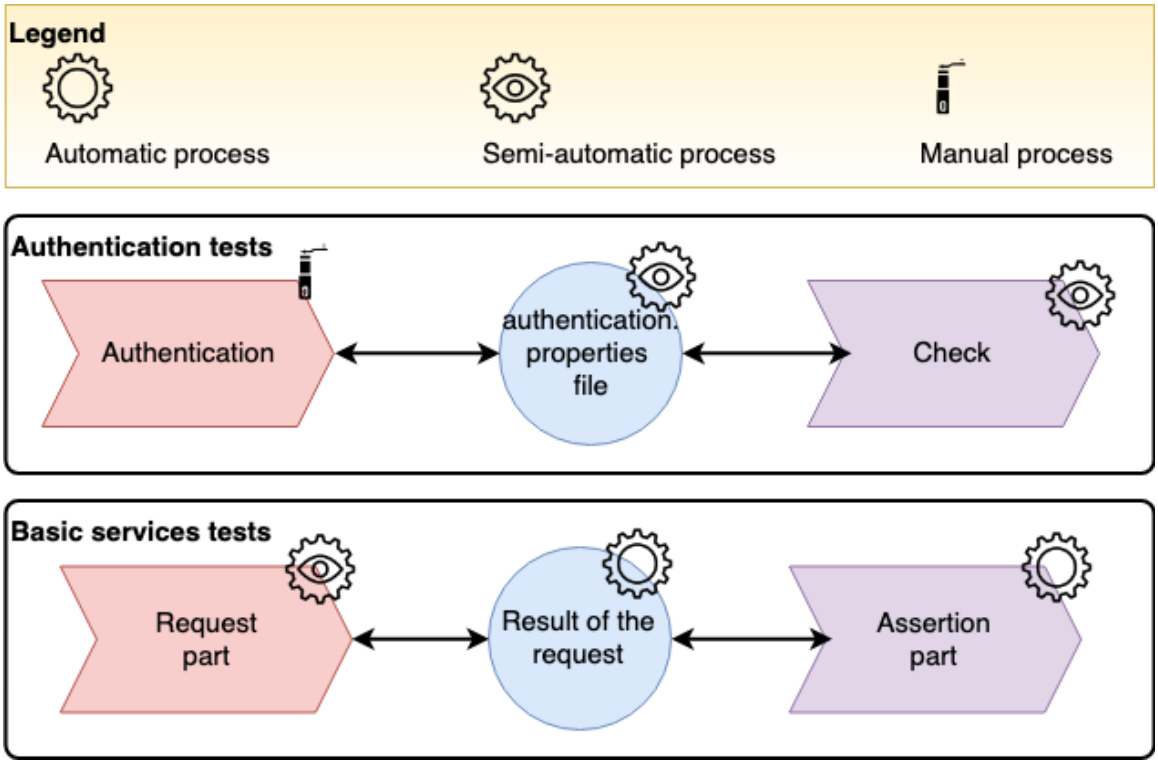
This part presents an optional component that may play the role of the backend between a frontend and the eHealth Platform. Vendor services are generally used to merge data coming from different sources/protocols.

eHealth services

This part presents the eHealth platform which integrates with two main components: the authentication server and the basic services. The authentication server authenticates a user and defines access rights to the application (with a configured Realm). If the access is authorized the user can invoke to the basic services (e.g. consent, therapeutic link, therapeutic exclusion, etc.) with a REST interface.

2 Test methodology

In order to test and validate how the vendor mobile applications use the services of the eHealth platform, we defined a test methodology based on two steps. The first step denotes the test and the validation of the authentication. The second step represents the test and the validation of the use of the eHealth basic services.



The authentication tests step includes the following elements:

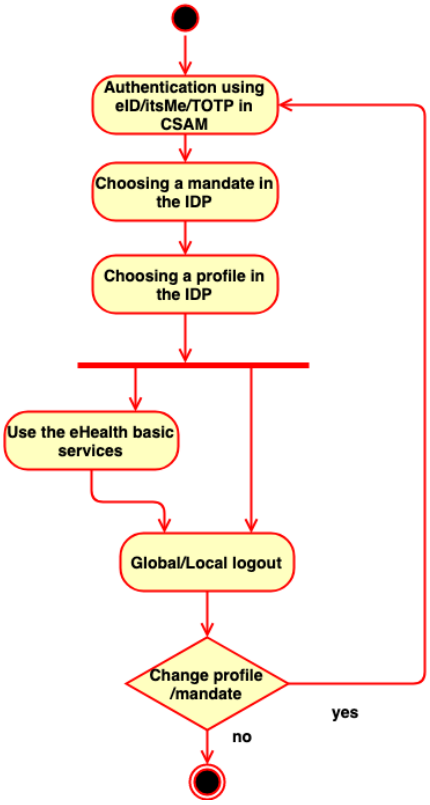
- **Authentication**: The authentication consists to authenticate a user to a given vendor application. This latter must be emulated on a test computer and the authentication process must be done with the Chrome browser.
- **IDP Check:** The Check component enables to get the cookie "shib_idp_session" created by the IDP in order to check if the application is correctly logged in the eHealth platform.
- **Authentication.properties" file:** This file contains required information (i.e. first name, the last name, and the desired profile) in order to check and validate the authentication of the user to the IDP.

The basic services tests step integrates with the following elements:

- **Request part:** It is specific to the application to test and must be implemented by the vendor. The request part could be implemented with on Selenium or Appium framework. It contains the request enabling to get the data which must be validated (e.g. getting the value of the textbook containing data about the patient consent).
- **Result of the request:** The results of the request part are stored in the response repository (e.g. storing the result of the patient consent).
- **Assertion part:** The assertion part integrates a set of generic scripts enabling to compare data coming from the request part with the data given by the eHealth platform.

3 Test workflow

The following schema defines how to sequence tests.



3.1 Testing the integration of the authentication

Considered test scripts: *TS-0XX* (Identity & Access Management (IAM) Test scripts)

Scope: The aim of these test scripts is to check if the application opens correctly an IDP and IAM session when the user tries to authenticate him /herself via one of the following ways: eID, itsMe or TOTP.

Test environment configuration

Module Name	Used	Configuration needed
Assertion	X	citrus.properties (IAM Configuration section)
ResponseRepository		
SeleniumRequest		
AppiumRequest		
tokenGenerator		

An instance of your application must be available. (backend + gui)

Manual actions: These tests should be semi-automatic because the user has to identify him/herself on the CSAM portal via eID, TOTP or itsme and on the IDP (e.g. choosing a profile, choosing a mandate, etc.).

Operations

Test script	Operation
TS-001 partially automatic	1. Before running this test, configure the "citrus.properties" file with the right credentials of the user (i.e. first name and the last name) and the desired profile 2. Start the application and identify yourself depending on the information in the "citrus.properties" file. 3. Update the "pom" of assertion module with "M2-IAM-Authentication.xml" test suite and run the project (mvn package)

TS-002 partially automatic	1. Before running this test, configure the "citrus.properties" file with the right credentials of the user (i.e. first name and the last name) and the desired profile 2. Start the application and identify yourself depending on the information in the "citrus.properties" file. 3. Logout locally from the application 4. Update the "pom" of assertion module with "M2-IAM-LocalLogout.xml" test suite and run the project (mvn package)
TS-003 partially automatic	1. Start the application and identify yourself with respect to the information in the "citrus.properties" file. 2. Logout globally from the application 3. Update the "pom" of assertion module with "M2-IAM-GlobalLogout.xml" test suite and run the project (mvn package)
TS-004 manual	Same as TS-001 but following the profile chosen the application must display a welcome page depending of the chosen profile
TS-005 deduction	Test succeeded if TS-001, TS-003 and TS-004 passed
TS-006 manual	Test succeeded if TS-001 and TS-004 passed with a mandate.
TS-007 deduction	Test succeeded if test TS-006 and TS-003 passed
TS-008 partially automatic	1. Start the application and identify yourself with a revoked mandate. (it must fail) 2. Update the "pom" of assertion module with "M2-IAM-FailedAuthentication.xml" test suite and run the project (mvn package)

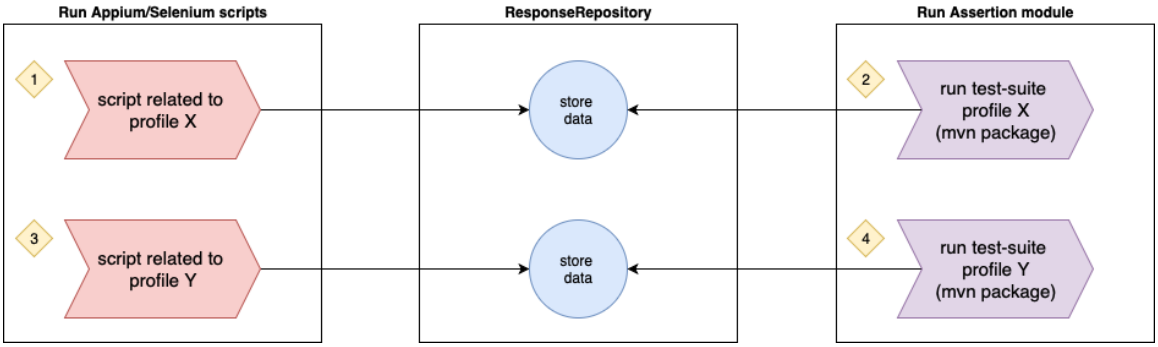
3.2 Checking eHealth basic services (consent, therapeutic exclusion and therapeutic links)

Considered test scripts: TS-1XX (consent), TS-2XX (therapeutic link), TS-3XX (therapeutic exclusion)

Scope: These tests aim to validate the basic functionalities of eHealth. The aim of these test scripts is to check if the application displays the correct information.

These tests are grouped on test-suites. Each test suite relevant to one specific user profile and run tests on the 3 health basic services (consent, therapeutic link and therapeutic exclusion)

Following the test methodology described before, executing a test-case needs 2 steps: getting data from the application and making the assertion.



For each profile, the combination of Appium/Selenium scripts and the Assertion module must be run separately.

3.2.1 Getting data from the request component (relative to one profile)

Test environment configuration

Module Name	used	configuration needed
Assertion		
ResponseRepository	X	
SeleniumRequest	X	Generally you have to use Selenium or Appium, the choice depends on the nature of the application to test.
AppiumRequest	X	
tokenGenerator		

An instance of your application must be available. (backend)

The SeleniumRequest and the AppiumRequest modules contain scripts that are specific to the application to test. These 2 modules contain script examples that get data from our ' eHealth MockApplication' and populate a data model. This data model will be used for the assertions.

Your scripts have to collect data from your application user interface and sent the collected data to the ResponseRepository module.

Your scripts must be grouped by user profile.

3.2.2 Running the assertions

Test environment configuration

Module Name	Used	Configuration needed
Assertion	X	
ResponseRepository	X	
SeleniumRequest		
AppiumRequest		
tokenGenerator	X	Simulate a backend to give a token to the Assertion Module. The port of this module must be linked to the callback of your Realm (default 8080)

Your application doesn't run. Your backend neither. The tokenGenerator is used instead.

Operations

Test suite	Profile tested (flow)	Basic eHealth service tested	Test script run
M2-BasicService-PatientFlow.xml	Patient	consent, therapeutic exclusion, therapeutic links	TS-101, TS-202, TS-302
M2-BasicService-Physician.xml	Physician	consent, therapeutic links	TS-102, TS-203