Serums

HORIZON 2020

Project no. 826278

SERUMS

Research & Innovation Action (RIA)

SECURING MEDICAL DATA IN SMART-PATIENT HEALTHCARE SYSTEMS

Report on Final User Authentication System
D5.4

Due date of deliverable: 30th April 2022

Start date of project: 1st January 2019

Type: Deliverable
WP number: WP5

Responsible Institution: UCY
Editor and editor's address: Marios Belk (belk@cs.ucy.ac.cy)
Partners Contributing: UCL, SOPRA, IBM, ZMC, FCRB

Reviewers:
Matthew Banton (USTAN)
Thais Webber (USTAN)

Version 1.0

Project co-funded by the European Commission within the Horizon H2020 Programme		
Dissemination Level		
PU	Public	X
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)	

Release History

Release No.	Date	Author(s)	Release Description/Changes made
V0.1	01/11/2021	Marios Belk (UCY), Andreas Pitsillides (UCY)	Defined TOC and added initial Executive Summary
V0.2	01/11/2021	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Added final general architecture and extended description on use-case scenarios
V0.3	12/12/2021	Marios Belk (UCY), Christos Fidas (UCY), Argyris Constantinides (UCY), Andreas Pitsillides (UCY)	Added description of new APIs
V0.4	15/02/2022	Elias Athanasopoulos (UCY), Argyris Constantinides (UCY)	Added credential hardening mechanism
V0.5	07/03/2022	Argyris Constantinides (UCY)	Added new front-end designs Added description on password strength meter Added description on image analysis tool
V0.6	21/04/2022	Eduard Baranov (UCL)	Added the verification of the authentication properties
V0.7	24/04/2022	Marios Belk (UCY), Argyris Constantinides (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Beta version of the deliverable for internal review
V0.8	28/04/2022	Matthew Banton (USTAN) Thais Webber (USTAN)	Version after partners' comments
V0.9	29/04/2022	Marios Belk (UCY), Argyris Constantinides (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Pre-final version for final check
V1.0	29/04/2022	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Release candidate

SERUMS Consortium

Partner 1	**University of St Andrews**
Contact Person	Name: Juliana Bowles Email: jkfb@st-andrews.ac.uk
Partner 2	**Zuyderland Medisch Centrum**
Contact Person	Name: Larissa Haen-Jansen Email: la.jansen@zuyderland.nl
Partner 3	**Accenture B.V.**
Contact Person	Name: Bram Elshof Email: bram.elshof@accenture.com
Partner 4	**IBM Israel Science & Technology Ltd.**
Contact Person	Name: Michael Vinov Email: vinov@il.ibm.com
Partner 5	**Sopra-Steria**
Contact Person	Name: Andre Vermeulen Email: andreas.vermeulen@soprasteria.com
Partner 6	**Université Catholique de Louvain**
Contact Person	Name: Axel Legay Email: axel.legay@uclouvain.be
Partner 7	**Software Competence Centre Hagenberg**
Contact Person	Name: Michael Rossbory Email: michael.rossbory@scch.at
Partner 8	**University of Cyprus**
Contact Person	Andreas Pitsillides Email: andreas.pitsillides@ucy.ac.cy
Partner 9	**Fundació Clínic per a la Recerca Biomèdica**
Contact Person	Name: Santiago Iriso Email: siriso@clinic.cat
Partner 10	**University of Dundee**
Contact Person	Name: Vladimir Janjic Email: vjanjic001@dundee.ac.uk

Table of Contents

Executive Summary

Securing Medical Data in Smart Patient-Centric Healthcare Systems (Serums) is a research project supported by the European Commission (EC) under the Horizon 2020 program. This is the fourth and final deliverable of *Work Package 5: “Authentication and Trust”*. The leader of this work package is UCY, with involvement from the following partners: UCL, SOPRA, IBM, ZMC, FCRB. The objective of this work package is focused on designing and developing a user-centric authentication system aiming to deliver a secure, personalized and usable authentication mechanism to each user’s preference and interaction device, in order to preserve security and improve usability. The primary goals are to: *i)* provide high levels of security to confirm the identity of each user and accordingly authorize access to certain parts of personal and/or medical data in the system; and *ii)* improve the usability levels of the user authentication mechanisms by increasing memorability of selected secrets and task execution efficiency and effectiveness.

This deliverable, entitled *“D5.4. Report on Final User Authentication System”* reports the implementation and verification of the final user authentication system. For the implementation of the Serums’ user authentication system, a User-Centered Design methodology has been adopted for developing and finalizing the user authentication scheme through multiple iterations (three prototypes of the system have been released throughout the course of the project; initial, refined, final software) that will be used for evaluation studies. This deliverable reports on the final software of the user authentication scheme.

Given that the outcome of this work package is a result of an iterative development process that refined the final Serums user authentication system throughout the course of the Serums project, for completeness, we include existing stable designs, modules, data-flow diagrams, scenarios, endpoints, and mechanisms from previous deliverables of this work package (Deliverable 5.2; Deliverable 5.3).

1 Introduction

1.1 Role of the Deliverable

The role of this deliverable is to report the design and development of the final software of the user authentication scheme. Specifically, it reports: *i)* the improved and final user authentication paradigm based on a novel retrospective and flexible approach in graphical and textual passwords; *ii)* the final architecture of the user authentication scheme; *iii)* the architectural design and development details of the credential hardening mechanism; *iv)* the sequence diagrams of the final authentication use-case scenarios; *v)* the description of the final Application Programming Interface (API) and database design of the user authentication scheme; *vi)* the final front-end design of the user interfaces in the authentication system; and *vii)* final results of the verification of the user authentication system. The outcome of the user authentication system constituted the basis for the evaluation of the third and final Proof of Concept (PoC3) of Serums.

1.2 Relationship to Other Serums Deliverables

Deliverable	**Relation**
D2.6: Final Software for Storage, Access, Blockchain and Metadata Extraction for Smart Patient Health Records	The user authentication API of D5.4 is used as input in the final software of the Smart Patient Health Records
D4.3: Report on Final Data Fabrication and Semantic-Preserving Encryption	Characteristics of the updated database schema of D5.4 is used as input for data fabrication and semantic-preserving encryption
D6.3: Report on Final Smart Health Centre System Software	The outcome of D5.4 is used as input for the final version of the integrated smart healthcare system software
D7.6: Report on Final Use Cases and Evaluation	The final version of the user authentication scheme of D5.4 is used in the context of the evaluation studies of PoC3
D7.7: Report on Technical Roadmap for Serums Technology	Outcomes of D5.4 is used as a basis for further elaborating ideas and areas for improvement for the user authentication system as part of the Serums technical roadmap

1.3 Structure of this Document

The rest of the document is structured as follows: *Chapter 2* describes the Serums authentication paradigm. *Chapter 3* describes the general architecture of the user authentication system, including details on new and extended modules, such as a password strength meter and semantic image analysis tool. *Chapter 4* provides implementation details of the credential hardening mechanism. *Chapter 5* describes the sequence diagrams of the final user authentication scenarios. *Chapter 6* describes the results of the user authentication component verification. *Chapter 7* discusses implications and the applicability of the proposed authentication paradigm within healthcare environments. *Chapter 8* concludes the deliverable including limitations of this research and future work. *APPENDIX A* lists the research publications in which activities of this work package have contributed to. *APPENDIX B* presents the final front-end design of the user authentication screens. *APPENDIX C and D* respectively describe the final Application Programming Interface of the user authentication system, and the design of the database.

2 A Flexible and Personalized Locimetric User Authentication Paradigm in Healthcare

In this section we describe the proposal of the user authentication method, coined *FlexPass*, which is based on a novel *"Single-Secret Two Reflections" (SS2R)* authentication paradigm. We first provide details on the underlying theory and conceptual design of the approach.

2.1 Research Motivation

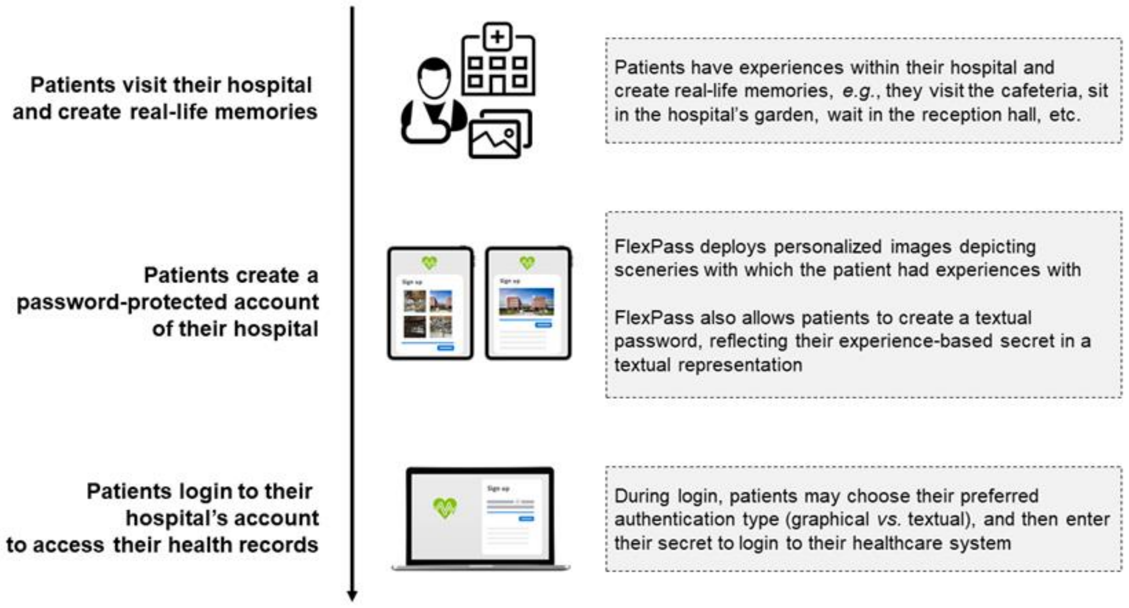
Healthcare organizations still rely on traditional knowledge-based authentication approaches, and specifically, on textual passwords and/or location-aware approaches (*e.g.*, Radio Frequency Identification - RFID). This is based on several reasons, *i.e.*, due to increased implementation and maintenance costs, due to immaturity of new authentication approaches, as well as known security and privacy issues of new user authentication paradigms (*e.g.*, biometrics) (Fidas et al., 2021), which negatively affect wide adoption of such technologies (Mason et al., 2020). Simultaneously, healthcare organizations' experts are aware that textual passwords negatively affect usability and security aspects due to complex policies, and therefore seek for novel and easy-to-adapt knowledge-based user authentication approaches as alternative solutions in order to avoid affecting the users' familiarity and existing practice.

Furthermore, the literature reveals that: *a)* a plethora of user authentication methods (knowledge-, token-, biometric-based) have been introduced for healthcare environments, each one having its own strengths and weaknesses with regards to security, privacy and user experience; *b)* it is estimated that knowledge-based authentication mechanisms will continue to prevail in the next decades (Leon and Boštjan, 2019), even in combination with other approaches (*e.g.*, token-based) or as fallback mechanisms, hence, new approaches need to partially rely on existing textual password approaches in order to support the technology transition of users; *c)* user authentication in healthcare environments entails a mixture of unique constraints and challenges related to the location and context in which interaction takes place (Constantinides et al., 2021; 2020; Eikey et al., 2015); and *d)* evidence has shown that user preference and task performance varies depending on the user (*e.g.*, age, abilities) and the context of use (*e.g.*, interaction device, screen size), suggesting that any specific solution might not please everyone (Mare et al., 2016).

Bearing in mind that user authentication in healthcare environments is performed by users with varying profiles, in different contexts of use and on multiple heterogeneous devices, this work investigates whether end-users would benefit from a flexible and personalized user authentication solution that would adapt and personalize different authentication mechanisms (graphical and textual) depending on their context of interaction, aiming to achieve a viable balance between security and usability. Our work is primarily driven by our vision to combine graphical and textual password mechanisms based on a new *"Single-Secret Two Reflections" (SS2R)* user authentication paradigm, which allows us to move from current generic "one-size-fits-all" authentication systems towards flexible, user-adaptable and personalized authentication systems. The aim is to provide a viable and flexible authentication solution, by following state-of-the-art practices in the healthcare domain, and applicable within current healthcare organizations.

2.2 Conceptual Design based on the Dual Coding Theory

User Scenario: From Location-based Memories towards Location-aware Passwords. Consider a scenario (**Figure 1**) in which a patient, Emma, visits her hospital for her weekly checkup at her doctor. Emma drives through the entrance of the hospital and then parks her car. She further walks from the car park through the hospital's garden, enters the building and goes to the reception hall. She then registers at the reception hall in which she confirms her appointment with her doctor. She is then asked to wait for fifteen minutes until her appointment. During these fifteen minutes, Emma walks to the hospital's cafeteria and orders a coffee and croissant until her appointment. Emma completes the checkup with her doctor, receives a prescription of medication and then leaves the hospital and drives back home.



Patients visit their hospital and create real-life memories
Patients have experiences within their hospital and create real-life memories, *e.g.*, they visit the cafeteria, sit in the hospital's garden, wait in the reception hall, etc.
Patients create a password-protected account of their hospital
FlexPass deploys personalized images depicting sceneries with which the patient had experiences with
FlexPass also allows patients to create a textual password, reflecting their experience-based secret in a textual representation
Patients login to their hospital's account to access their health records
During login, patients may choose their preferred authentication type (graphical *vs.* textual), and then enter their secret to login to their healthcare system

Figure 1. Use-case scenario of FlexPass.

During Emma's visit at the hospital, she created several real-life memories within the hospital (*e.g.*, walk through the garden, visit at the cafeteria, appointment with the doctor). Based on the dual coding theory (Paivio, 2006; Sternberg, 2003), Emma encrypted a series of visual and verbal stimuli within her long-term memory (Atkinson and Shiffrin, 1968; Baddeley, 1990), and more specifically with the episodic, semantic and autobiographical memories (Tulving, 2002; Squire, 1992; Williams et al., 2008), which entail information about certain events experienced in an individual's lifetime and the corresponding semantic information describing these events. Furthermore, according to the dual coding theory, the human brain consists of a visual cognitive sub-system, which is utilized by the human brain during processing, representation and recall of imagery information, as well as a verbal cognitive sub-system, which is utilized by the human brain during processing, representation and recall of verbal information (Paivio, 2006). For example, information such as the word "cappuccino" is represented in the human mind as a visual representation of a cappuccino coffee cup, as well as the word "cappuccino". During recall, individuals retrieve and process both representations simultaneously, or separately. **Figure 2** depicts the underlying idea of FlexPass.

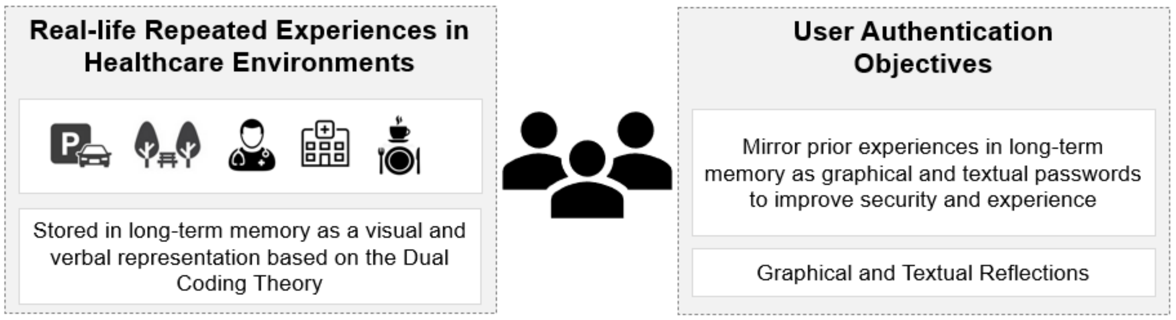

Real-life Repeated Experiences in Healthcare Environments
Stored in long-term memory as a visual and verbal representation based on the Dual Coding Theory
User Authentication Objectives
Mirror prior experiences in long-term memory as graphical and textual passwords to improve security and experience
Graphical and Textual Reflections

Figure 2. From location-based memories towards location-aware and flexible passwords.

2.3 FlexPass Authentication Paradigm

FlexPass aims to leverage on the dual coding theory based on a novel "Single-Secret Two Reflections" authentication paradigm. This enables patients to create a single conceptual secret leveraging upon their personal location-based memories they have built through their interactions in certain locations within the hospitals, and further reflect the secret on a graphical and/or textual password key. For creating the graphical password key, FlexPass presents location-aware images that depict image content of a certain location of a hospital, in which the patient had prior interaction with. In addition, FlexPass provides an additional option to the patient to create a textual password key that may be then utilized interchangeably with the graphical password based on user's preference. Our solution intentionally includes a textual password as an option to avoid changing the current state-of-the-art practice in the healthcare domain, and a method in which users are familiar with. Hence, we anticipate that FlexPass will be more easily transferable from the current state-of-the-art towards the new suggested approach, providing the option to users to switch to their preferred authentication type (graphical or textual). **Figure 3** illustrates the conceptual design of FlexPass.

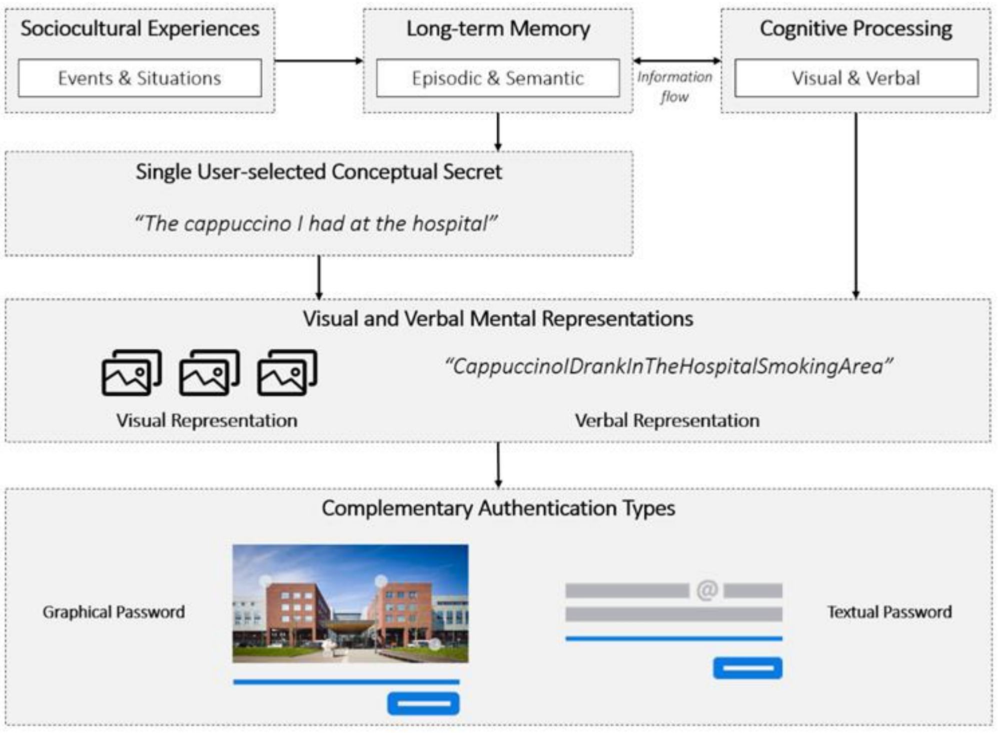

Sociocultural Experiences
Events & Situations
Long-term Memory
Episodic & Semantic
Information flow
Cognitive Processing
Visual & Verbal
Single User-selected Conceptual Secret
"The cappuccino I had at the hospital"
Visual and Verbal Mental Representations
"CappuccinoIDrankInTheHospitalSmokingArea"
Visual Representation
Verbal Representation
Complementary Authentication Types
Graphical Password
Textual Password

Figure 3. Conceptual design of FlexPass.

Graphical Passwords. The graphical password mechanism is based on cued-recall graphical authentication mechanisms (Biddle et al., 2012), which ask users to draw secret gestures on a background image that acts as a cue. For its implementation, we follow design and development guidelines of Microsoft's Picture Gesture Authentication (PGA)^{TM} mechanism (Johnson et al., 2014), introduced in Windows 8 (also available in most recent versions, *e.g.*, Microsoft Windows 10 and 11), which allows users to draw three types of gestures on the background image: taps (clicks), lines and circles. Free line gestures are automatically converted into one of the three allowed gestures.

Textual Passwords. FlexPass follows state-of-the-art security metrics and authentication policies with regards to the implementation of textual passwords (Komanduri et al., 2011; Burr et al., 2006; Belk et al., 2019). The textual password keys rely on a basic 16-character password policy, allowing the creation of dictionary words with no composition requirements, which is more usable and as secure as traditional complex 8-character policies (Komanduri et al., 2011) (the National Institute of Standards and Technology (NIST) predicts that both policies generate 30 bits of security entropy (Burr et al., 2006)).

In this context, FlexPass allows users to create a secret graphical and/or a textual password. During graphical password composition, FlexPass deploys images depicting popular sceneries of the hospital (*e.g.*, garden, reception hall, cafeteria, etc.). The user is asked to select an image of her preference and then create a graphical password by drawing secret gestures on certain regions of the image based on the experience she had with the depicted content in the image. For example, based on the aforementioned user scenario, a conceptual secret derived from Emma's episodic memory and experiences at the hospital would be: *"the cappuccino I drank at the hospital"*. Emma would reflect this secret on the graphical password by selecting for example a coffee cup and the exact table she sat for having her coffee in the hospital's cafeteria. As a next step, FlexPass also allows users to create a textual password by asking the patient to reflect the conceptual-based graphical secret as a textual representation by articulating the secret, *e.g.*, the textual version of the secret would be *"CappuccinoIDrankInTheHospitalsCafeteria"*.

Hence, the "Single-Secret Two Reflections" paradigm extends existing works in knowledge-based user authentication based on the dual coding theory aiming to: *a)* enhance security by enabling users to select regions on an image that are familiar to the users and not to the attacker; *b)* to enhance memorability through ownership, and prior experience and knowledge of each single user; and *c)* to support user authentication adaptability since users can choose their preferred way to login based on their needs and context of use. For example, users that are on the move might prefer to login through touch-based graphical password input on the tablet device *vs.* users that are in the office might prefer to login through a textual password input on the conventional desktop computer.

3 General Architecture of the User Authentication System

In this section, we present the architectural design of the developed user authentication system. **Figure 4** illustrates the high-level architectural design of the final user authentication system, which also includes the password-hardening component (please see *Section 4*). The user authentication system is hosted at the University of St. Andrews premises on a CentOS Linux version 7 machine with 47GB of RAM and 2T of disk space.

3.1 Backend user authentication system

The backend of the user authentication system is developed in Python 3.7.4 using the Django REST Web development framework, which is a powerful and flexible toolkit for building Web Application Programming Interfaces (APIs) in Python. The core component of the backend is the server-side Web API, which is a programmatic interface consisting of publicly exposed endpoints to a defined request–response message system, expressed in JavaScript Object Notation (JSON) format and exposed via the Web by means of a Web server that is based on the Hypertext Transfer Protocol (HTTP).

For the deployment of the Django application, we use a modified Apache HTTP Server with an extension of the *mod_ssl* module for credential hardening, and *mod_wsgi*, which is an Apache module that can host any Python Web Server Gateway Interface application. For certain heavy and time-consuming tasks, such as, storing of graphical passwords, ideally, we would like the request and response cycle to be fast, otherwise we would leave the user waiting for rather too long. Even worse, the Web server can only serve a certain number of users at a time. So, if this process is slow, it can limit the number of pages our application can serve at a time. To solve this problem, Celery is used, which is an asynchronous task queue based on distributed message passing. Celery is not used through the whole project, but only for specific tasks that are time-consuming. The idea here is to respond to the user as quickly as possible, pass the time-consuming tasks to the queue so to be executed in the background, and always keep the server ready to respond to new requests. Celery additionally requires an external solution for sending and receiving messages. For this purpose, RabbitMQ is used, which is an open-source message-broker software.

For the storage of the data, PostgreSQL is used, which is an open-source Relational Database Management System commonly used within Django applications. Finally, the whole user authentication system is packaged and runs as a lightweight, portable, and self-sufficient container through Docker (version: 19.03.13, API version: 1.40), which is a set of software products that use virtualization at the operating system level to deliver software in packages called containers.

3.2 Front-end Web-based graphical and textual password system

The FlexPass front-end is developed using the Django's template language, which contains variables that get replaced with values when the template file, *e.g.*, Hypertext Mark-up Language (HTML) file is rendered, and tags that control the logic of the template.

HTML is the primary mark-up language for the creation of Webpages on the World Wide Web. It provides a means to describe the structure of text-based information in a document through annotation of certain text as headings, paragraphs, etc., and to supplement that text with interactive forms, embedded images, and other objects. The main purpose of HTML is to display and format content, allowing very limited interaction with the Webpage. HTML also describes to some extent the semantics of a document and can include embedded scripting language code for manipulating at run-time the HTML elements of a document and the behavior of the Webpage. FlexPass front-end utilizes the latest version of HTML5, given its extended capabilities and to conform to the latest standards of today's HTML Web browsers.

To provide styling to the HTML elements, Cascade Style Sheets (CSS) are utilized, which provide a means for defining how HTML elements should be displayed. FlexPass front-end utilizes the latest version of CSS3 to take advantage of current state-of-the-art styling features and improvements for enhancing the Web presentation capabilities, as well as to conform to the latest World Wide Web Consortium design standards.

Today's Web-sites typically combine HTML, CSS, and client-side scripting for creating interactive pages. The most applied client-side scripting language on the World Wide Web (WWW) currently is JavaScript. Hence, FlexPass front-end utilizes JavaScript for the following purposes: *i)* for allowing users to create graphical passwords in an HTML canvas element that loads the background image; *ii)* for handling users' interactions (*e.g.*, time to create password, time to login); and *iii)* for communicating and exchanging data with the FlexPass back-end asynchronously without reloading the Webpage through Asynchronous JavaScript and XML.

The front-end designs of the Web application are available in three languages: English, Dutch, and Catalan.

3.3 Multi-factor authentication system

As an additional layer for security, we have implemented a smartphone application, available in both Android and iOS, using Flutter (*https://flutter.dev*). We coined the smartphone application as *Serums Authenticator*. APPENDIX B illustrates all the front-end user interface designs of the smartphone application. Serums Authenticator follows state-of-the-art practices with regards to multi-factor authentication solutions, *i.e.*, users can use their smartphone device as a second factor for authentication in which they can approve a successful login either through approval of a push notification and/or a time-based one-time passcode (TOTP). It implements the following functionalities: *i)* users initially pair their device with their Serums account using a QR code or a six-digit code; *ii)* users may use the TOTP displayed on the end-user's smartphone mobile application that is automatically reset every 30 seconds, aiming to approve their login; *iii)* users may also approve their login through an easy-to-use push notification; and *iv)* users have the option to remove the second factor for authentication if they wish.

The front-end designs of the smartphone application are available in three languages: English, Dutch, and Catalan.

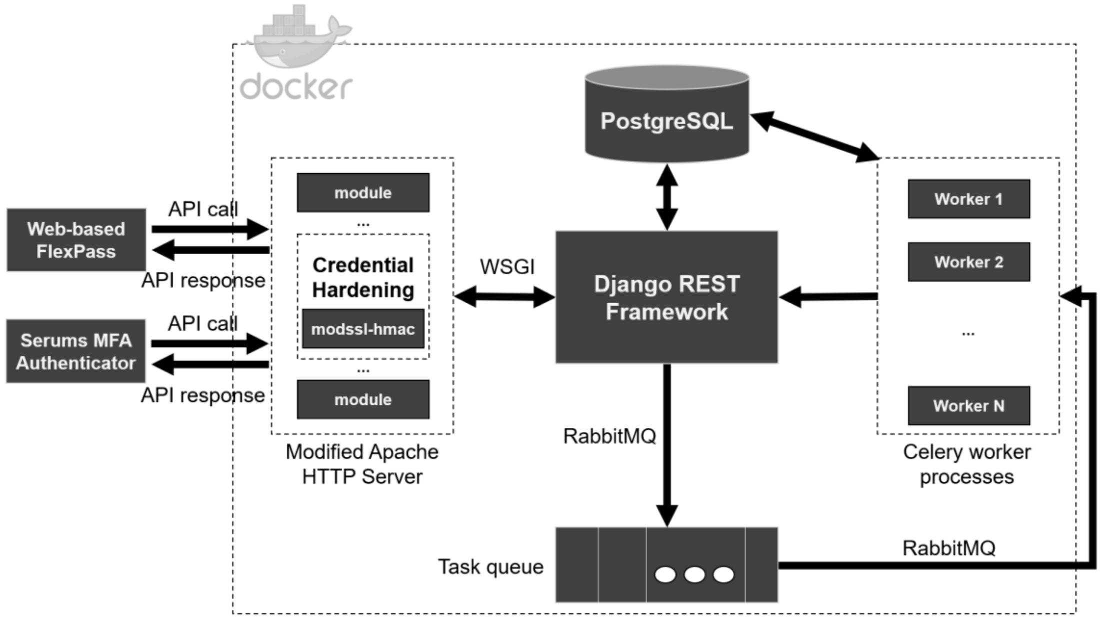


Figure 4. High-level architectural design and technologies used.

4 Credential Hardening

In this section, we present how Serums employs additional countermeasures in order to defend against attacks that are based on cracking offline leaked credentials. We further provide details on a password strength meter for assisting users in creating stronger passwords, and an intelligent image analysis tool for further assisting the quantification of graphical password strength.

4.1 Implementation

Recall that Serums secures a text-based password using a Message Authentication Code (MAC), instead of a cryptographic hash function (please refer to Serums Deliverables D5.2 and D5.3). In particular, Hash Message Authentication Code (HMAC) is used as provided by OpenSSL (Open Secure Sockets Layer); the aforementioned implementation uses internally SHA-256 (Secure Hash Algorithm) for hashing. The HMAC uses bits from the private key of the server to compute the cryptographic hash.

We have implemented the credential hardening mechanism by enhancing an Apache module, therefore it can be instantly enabled to all Web applications that run over Apache. Alternatively, it is straightforward to realize credential hardening to other Web infrastructures, so long as they support TLS connections. We now expand on all Apache-based modifications and then on all Web application modifications required for deploying credential hardening. Credential hardening builds on the existing *mod_ssl* module by adding a new hook. This can be done by modifying *mod_ssl.c*, where all the hooks needed to the Apache for serving TLS connections are set. Our hook is set as `APR_HOOK_FIRST` and thus it is executed as soon as possible in the request pipeline. We depict below the code excerpt where the hook is established.

```
#include "hasher.h"

static void ssl_register_hooks(apr_pool_t *p){

  ap_hook_handler(hasher_handler, NULL, NULL, APR_HOOK_FIRST);

}
```

Source Code. SSL register hook.

Moreover, we depict the core code of the entire credential hardening mechanism. Here, we reference lines of code for each of the basic steps credential hardening does, but reading the code is not necessary to understand the mechanics. The main handler of credential hardening does the following:

1. Declines any requests that are not local and that do not have arguments (*i.e.*, no password); *(lines 2-5)*
2. Checks that the connection uses TLS, and drops any non-encrypted one; *(lines 9-10)*
3. Reads the private key –used for TLS– from the SSL context and stores it to a buffer; if the private key is not available declines the request; *(lines 12-19)*
4. Decodes the argument (*i.e.*, password) from the request's URL; if the plain password is not correctly encoded, the request is declined; *(lines 24-28)*
5. Calls the HMAC function of the OpenSSL library with parameters: (a) the cryptographic hashing function (SHA256); (b) the private key as the key for the computed HMAC; and (c) the password to be hashed; *(lines 30-34)*
6. Returns the keyed digest to the client in the form of an encrypted HTTP response. *(lines 35-37)*

```
int hasher_handler(request_rec *r) {
  if (strcmp(r->uri,"/hmac-service")==0 && r->args!=NULL &&
    strcmp(ap_get_remote_host(r->connection, NULL,
    REMOTE_NAME, NULL),
    "127.0.0.1")==0) {
    char * key; server_rec *s = r->server;
    SSLSrvConfigRec *sc = mySrvConfig(s);
    modssl_ctx_t *server = sc->server;
    if (server == NULL || server->ssl_ctx == NULL)
      return DECLINED;
    else {
      EVP_PKEY * evp = SSL_CTX_get0_privatekey(server->ssl_ctx);
    if (evp) {
      size_t len = PRIVATE_KEY_SIZE; key = malloc(len);
      FILE *stringFile = fmemopen(key, len, "w");
      PEM_write_PrivateKey(stringFile, evp, NULL,
      NULL, 0, 0, NULL);
      fclose(stringFile);
    } else return DECLINED;
  }
  char * plainPassword = getPasswordFromArgs(r->args);
  int rounds = getRoundsFromArgs(r->args);
  // wrong password format
  char * dec=malloc(sizeof(char)*strlen(plainPassword)+1);
  if (plainPassword==NULL  || decode(plainPassword, dec)<0){
    free(dec); free(key);
  return DECLINED;
  }
  int rlen,i;
  unsigned char * hashed = HMAC(EVP_sha256(),
  key, strlen(key),
  dec, strlen(dec), NULL, &rlen);
  for (i=1;i<rounds;i++)
    h = HMAC(EVP_sha256(), key, strlen(key), h, rlen, NULL, &rlen);
    for (i = 0; i < rlen; i++) {
      ap_rprintf(r, "%02X", h[i]);
    }
  free(key); free(dec); free(plainPassword); return OK;
  }
  return DECLINED;
}
```

Source Code. Implementation of credential hardening.

To integrate the *modssl_hmac* library into the containerized Django application, we had to build and install from source the following: *i)* Apache HTTP server; *ii)* PHP; and *iii) mod_wsgi* for configuring Django to run based on the modified Apache version that includes the credential hardening component. Next, we present the script that builds and installs the aforementioned technologies, as well as configures the Django Web application of the user authentication system.

```
#!/bin/bash

set -e

touch /var/modssl/.cert;

echo "Downloading and installing apache";


# Install apache with mod_ssl_hmac

if [[ -e /var/modssl/.cert ]] && [[ ! -e /var/modssl/.apache ]]; then

    cd /var/modssl;

    rm -rf /var/modssl/httpd ;


    if [[ ! -d /var/modssl/httpd ]]; then

        git clone --depth 1 --branch 2.4.41 https://github.com/apache/httpd.git;

        sed -i '/#include "mod_ssl.h"/a #include "hasher.h"'
/var/modssl/httpd/modules/ssl/mod_ssl.c;

        sed -i '/ssl_io_filter_register(p);/a ap_hook_handler(hasher_handler, NULL, NULL,
APR_HOOK_FIRST);' /var/modssl/httpd/modules/ssl/mod_ssl.c;

        sed -i '/mod_ssl.lo dnl/a hasher.lo dnl/' /var/modssl/httpd/modules/ssl/config.m4;

        cp /var/modssl/lib/hasher.c /var/modssl/httpd/modules/ssl/hasher.c;

        cp /var/modssl/lib/hasher.h /var/modssl/httpd/modules/ssl/hasher.h;

        cd /var/modssl/httpd;

        svn co http://svn.apache.org/repos/asf/apr/apr/trunk srclib/apr;

        ./buildconf;

        chmod +x configure;

        ./configure --prefix=/usr/local/apache2;

    fi


    cp /var/modssl/lib/hasher.c /var/modssl/httpd/modules/ssl/hasher.c;

    cp /var/modssl/lib/hasher.h /var/modssl/httpd/modules/ssl/hasher.h;

    cd /var/modssl/httpd;

    make clean >>  /var/modssl/.logs;

    make >>  /var/modssl/.logs;

    make install >>  /var/modssl/.logs;

    touch /var/modssl/.apache;

fi

echo "Done apache";

echo "Downloading and installing php";

# Install php
```

```
if [[ -e /var/modssl/.apache ]] && [[ ! -e /var/modssl/.php ]]; then
    cd /var/modssl/;
    if [[ ! -d /var/modssl/php-7.2.16 ]]; then
        wget   -O   php.tar.gz   https://github.com/php/web-php-distributions/raw/master/php-
7.2.16.tar.gz >> /var/modssl/.logs;
        tar -zxvf php.tar.gz;
        rm php.tar.gz;
    fi
    cd /var/modssl/php-7.2.16;
    ./configure  --with-apxs2=/usr/local/apache2/bin/apxs  --with-curl  --with-mysqli  --enable-
mbstring --with-gd --with-jpeg-dir=/usr/lib64 --enable-opcache >> /var/modssl/.logs;
    make clean >> /var/modssl/.logs;
    make >> /var/modssl/.logs;
    make install >> /var/modssl/.logs;
    cd /var/modssl/;
    touch /var/modssl/.php;
fi
echo "Done php";
echo "Downloading and installing mod_wsgi for Django";
if [[ ! -e /var/modssl/.mod_wsgi ]]; then
    cd /var/modssl/;
    if [[ ! -d /var/modssl/mod_wsgi-4.7.1 ]]; then
        wget        https://github.com/GrahamDumpleton/mod_wsgi/archive/4.7.1.tar.gz        >>
/var/modssl/.logs;
        tar xvfz 4.7.1.tar.gz;
        rm 4.7.1.tar.gz;
    fi
    cd mod_wsgi-4.7.1/;
    ./configure --with-apxs=/usr/local/apache2/bin/apxs
    make >>  /var/modssl/.logs;
    make install >>  /var/modssl/.logs;
    cd /var/modssl/;
    touch /var/modssl/.mod_wsgi;
fi
echo "Done mod_wsgi";
echo "Applying apache configuration files";
# Apply configuration files
if [[ -e /var/modssl/.php ]] && [[ ! -e /var/modssl/.conf ]]; then
    cp /var/modssl/conf/httpd.conf /usr/local/apache2/conf/httpd.conf
    cp /var/modssl/conf/httpd-ssl.conf /usr/local/apache2/conf/extra/httpd-ssl.conf
    touch /var/modssl/.conf;
fi
```

```
if [[ -e /var/modssl/.started ]]; then
    rm /var/modssl/.started;
fi
# Start apache
echo "Start apache";

if [[ -e /var/modssl/.conf ]]; then
    /usr/local/apache2/bin/apachectl start;
    touch /var/modssl/.started;
fi
echo "Apache started";
echo "Container is ready";
cat;
```

Source Code. Integration of credential hardening.

4.2 Storing Textual and Graphical Passwords

Regarding the storage of the textual password, we initiate a request to the credential hardening component, which converts the textual password string into a HMAC using the TLS key. The final HMAC'ed textual password string is stored in the database.

With regards to the graphical password system, three types of gestures are allowed: taps (clicks), lines and circles. Free line gestures are not permitted; hence, they are automatically converted into one of the three permitted gestures.

To process the gestures, the mechanism creates a grid of the image containing 100 squares (segments) on the longest side[1], and then divides the shortest side by the same scale. Rounding is not applied to any decimal segments and the mechanism allows 0.25 segments size overflow at the rightmost side of the image. The approach of creating a grid of squares allows for storing the gestures based on their segment position on the grid rather than the coordinates in pixels (**Figure 5**). The following data is stored: for taps, the *(x, y)* coordinates of a point, for lines the *(x, y)* coordinates of the starting and ending point, and for circles the *(x, y)* coordinates of the center, the radius and the directionality (clockwise/counterclockwise).

The mechanism allows for a tolerance distance in terms of the coordinates on the grid (36 segments around each initial selected segment are acceptable[1] (Katsini et al., 2018), thus, building a circle of 3 segments radius). This tolerance allows better accuracy of users' selections during login. However, there is no tolerance regarding ordering, type, directionality of the gestures. During the login, the mechanism compares the entered password with the stored one and login will be considered successful if *(a)* all three gestures (ordering, type, and directionality) match with the stored ones; and *(b)* the tolerance distance between the entered gestures and the stored ones fit in the predefined tolerance threshold.

[1] *https://docs.microsoft.com/en-us/archive/blogs/b8/signing-in-with-a-picture-password*

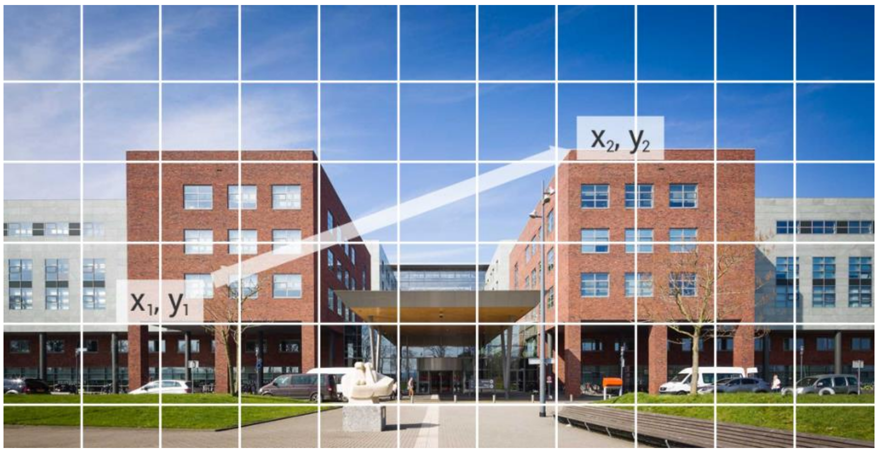


Figure 5. The graphical password mechanism creates a grid of the image containing 100 squares (segments) on the longest side, and then divides the shortest side by the same scale.

Although the approach adopted by Microsoft's PGA^{TM} for storing the graphical passwords remains undisclosed (Zhao et al., 2013), we follow state-of-the-art research on PGA for the scenario in which all the passwords that fall into the vicinity (as defined by the threshold) of chosen passwords are stored in a file with hashes on the server (Zhao et al., 2013). To be able to calculate the hash of the graphical password and store it in the file, we first need to represent the graphical password as a string. To do so, we use the following string representation for each gesture type:

Tap: "#N|T|x1,y1"

#: Denotes the start of the gesture representation.

N: The order of the gesture. Can be any integer number in the range [1-3].

|*:* The first vertical bar separates the order of the gesture and the gesture type.

T: The letter "T" refers to the gesture type tap (click).

|*:* The second vertical bar separates the gesture type and the remaining data, *i.e.*, (x1, y1) coordinates.

x_1*:* The x coordinate of the tap inside the image grid.

y_1*:* The y coordinate of the tap inside the image grid.

Circle: "#N|C|x1,y1,r,c"

#: Denotes the start of the gesture representation.

N: The order of the gesture. Can be any integer number in the range [1-3].

|*:* The first vertical bar separates the order of the gesture and the gesture type.

C: The letter "C" refers to the gesture type circle.

|*:* The second vertical bar separates the gesture type and the remaining data, *i.e.*, (x1, y1) coordinates, radius, and directionality.

x_1*:* The x coordinate of the circle's center inside the image grid.

y_1*:* The y coordinate of the circle's center inside the image grid.

r: The radius of the circle.

c: Boolean value that denotes whether the directionality is clockwise (True) or counter-clockwise (False)

Line: "#N|C|x1,y1,x2,y2"

#: Denotes the start of the gesture representation.

N: The order of the gesture. Can be any integer number in the range [1-3].

|*:* The first vertical bar separates the order of the gesture and the gesture type.

L: The letter "L" refers to the gesture type line.

|*:* The second vertical bar separates the gesture type and the remaining data, *i.e.*, (x1, y1) coordinates and (x2, y2) coordinates.

x_1*:* The x coordinate of the line's starting point inside the image grid.

y_1*:* The y coordinate of the line's starting point inside the image grid.

x_2*:* The x coordinate of the line's ending point inside the image grid.

y_2*:* The y coordinate of the line's ending point inside the image grid.

Combinations based on the threshold. The final string representation of the graphical password is the concatenation of the three strings, where each string refers to the corresponding gesture. Due to the introduction of the tolerance, for each graphical password string we also compute all the possible combinations that will match the initial graphical password string. After applying the tolerance to each segment, we end up with the following combinations for each gesture type:

- *Tap:* A total of 4 combinations, which correspond to the tap's *(x, y)* pairs of coordinates that will match the initial graphical password string during the comparison.
- *Circle:* A total of 12 combinations (4 combinations for the center × 3 combinations for the radius), which correspond to the circle's center *(x, y)* pairs of coordinates combined with 3 radii (initial, increased, decreased) that will match the initial graphical password string during the comparison.
- *Line:* A total of 16 combinations (4 combinations for the line's starting point × 4 combinations for the line's ending point), which correspond to the starting point's *(x, y)* pairs of coordinates combined with the ending point's *(x, y)* pairs that will match the initial graphical password string during the comparison.

Due to the differences in total combinations across gestures, and aiming to avoid revealing any information about the gesture type, we take an extra step by generalizing to the most complex combination, *i.e.*, as in having 3 lines, which would yield $16^3 = 4096$ combinations. Therefore, in case of taps and circles, for the remaining combinations, we also generate dummy password string combinations, so we always create the maximum number of 4096 combinations. To do so, we generate the remaining dummy password strings as 50-character strings (Komanduri et al., 2011). The 50-character dummy password strings are generated by combining the following: *i)* a 32-character lowercase hexadecimal string generated using Python's *uuid* module (*e.g.*, *uuid4* function); and *ii)* a

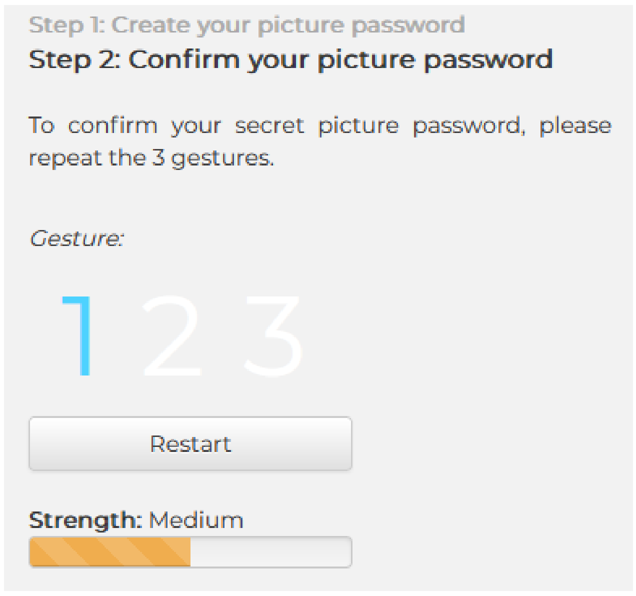
18-character random string generated using Python’s *secrets* module (*e.g.*, *choice* function, which accepts as input a non-empty sequence consisting of ascii letters and digits, and returns a randomly-chosen element). Finally, for each of the generated combination, we make a request to the credential hardening component, which converts the password string into a Hash Message Authentication Code using the TLS key. The final HMAC’ed graphical password string is stored in a file and contains all the possible matching combinations for a particular user. The content of this file is used during the login process, in which the input graphical password string is first converted to an HMAC via the credential hardening component and is then compared to the HMACs contained in the file for comparison. Below, we present an example of initiating a request to the credential hardening component that generates the HMAC of all combinations of a graphical password.

```
def generate_graphical_password_hashes(password_data):
    ''' Generates all possible hash combinations that match the input password data.
        Returns:
            (list of str): A list that contains all the hashes
    '''
    final_combinations = []
    first_gesture_items = password_data[0]
    second_gesture_items = password_data[1]
    third_gesture_items = password_data[2]

    url = "https://127.0.0.1/hmac-service"

    for gesture_1 in first_gesture_items:
        for gesture_2 in second_gesture_items:
            for gesture_3 in third_gesture_items:
                password_string = gesture_1 + gesture_2 + gesture_3
                params = {
                    'password': password_string
                }
                r = requests.get(url, params=params)
                final_combinations.append(r.text)
    return final_combinations
...
graphical_password_hashes = generate_graphical_password_hashes(password_data)
```

Source Code. Request to the credential hardening component that generates the HMAC of all combinations of a graphical password.

4.3 Password Strength Meter

Aiming to assist users in creating stronger passwords, we have implemented a password strength meter, which provides a run-time estimation about the strength of the user-created graphical password and textual password. **Figure 6** illustrates the password strength meter of the graphical user authentication system.


Step 1: Create your picture password

Step 2: Confirm your picture password

To confirm your secret picture password, please repeat the 3 gestures.

Gesture:

1 2 3

Strength: Medium

Figure 6. Password strength meter of the graphical user authentication system.

For measuring the graphical password strength, we used a heuristic approach by considering state-of-the-art knowledge on picture gesture authentication[1]. Taking into consideration that the tap (click) is the least complex gesture, while the line is the more complex gesture, we set our complexity heuristic as depicted in **Table 1**.

Table 1. Complexity heuristics based on combination of gestures, disregarding order.

Combination of gestures	**Complexity**
3 taps	40%
3 circles	80%
3 lines	100%
# Disregarding order	
1 tap & 2 circles	50%
2 taps & 1 circle	50%
1 tap & 1 line & 1 circle	70%
2 taps & 1 line	70%
1 tap & 2 lines	70%
2 circles & 1 line	70%
2 lines & 1 circle	80%

Taking into consideration that the proximity of different gestures impacts the overall complexity of the password (*e.g.*, different gestures on the same *(x, y)* segment on the grid are less secure), we take an extra step to either penalize (-20%) password combinations that include gestures that are in close proximity (as defined by the threshold of a circle of 3 segments radius), or reward (+20%) password combinations that do not include gestures in close proximity. The minimum and maximum value of textual password complexity is 0% and 100% respectively. The higher the score, the more complex the password is.

Furthermore, for measuring textual password complexity, we have implemented Dropbox's *zxcvbn* (Wheeler, 2016), which is a widely applied and realistic password strength estimator. *zxcvbn* applies pattern matching and conservative estimation, recognizing and weighing 30,000 common passwords, names, surnames according to US census data, popular English words from Wikipedia and television and movies. Commonly used patterns by users are also considered, such as, dates, repeated characters, sequential characters, keyboard patterns, etc. *(https://github.com/dropbox/zxcvbn)*. The source code below depicts an example of password strength estimation through zxcvbn.

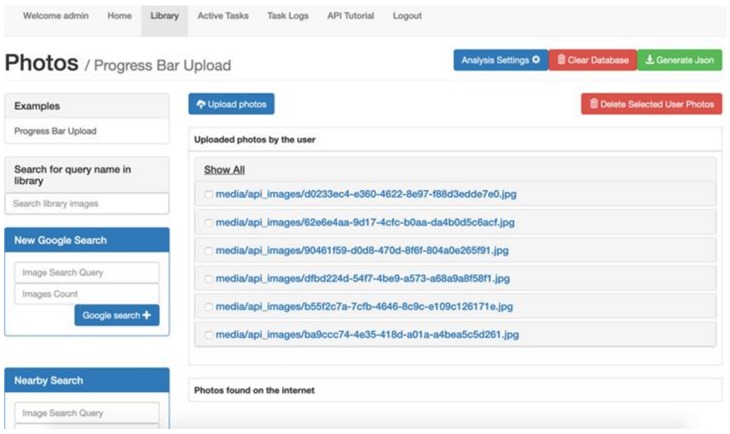
```
function get_passphrase_strength_results(passphrase) {

var result = zxcvbn(passphrase, user_inputs=[]);

var crack_time_seconds = result.crack_times_seconds;

var crack_times_display = result.crack_times_display;

var res_obj = {};

res_obj.guesses = result.guesses;

res_obj.guesses_log10 = result.guesses_log10;

res_obj.online_throttling_100_per_hour=crack_time_seconds['online_throttling_100_per_hour'];

res_obj.online_no_throttling_10_per_second=crack_time_seconds['online_no_throttling_10_per_se
cond'];

res_obj.offline_slow_hashing_1e4_per_second=crack_time_seconds['offline_slow_hashing_1e4_per_
second'];

res_obj.offline_fast_hashing_1e10_per_second=crack_time_seconds['offline_fast_hashing_1e10_pe
r_second'];

res_obj.crack_times_display_online_throttling_100_per_hour=crack_times_display['online_thrott
ling_100_per_hour'];

res_obj.crack_times_display_online_no_throttling_10_per_second=crack_times_display['online_no
_throttling_10_per_second'];

res_obj.crack_times_display_offline_slow_hashing_1e4_per_second=crack_times_display['offline_
slow_hashing_1e4_per_second'];

res_obj.crack_times_display_offline_fast_hashing_1e10_per_second=crack_times_display['offline
_fast_hashing_1e10_per_second'];

res_obj.score = result.score;;

return res_obj;

}

res_obj = get_passphrase_strength_results(passphrase);
```

Source Code. Example of password strength estimation through *zxcvbn*.

4.4 Intelligent Image Analysis for Quantification of Graphical Password Strength

Aiming to further assist the quantification of graphical password strength, we have implemented an image analysis service, which performs Artificial Intelligence-driven object detection and points of interest detection on an image. Such knowledge can be then fed into the graphical password strength estimator for calculating the graphical password strength based on whether users make selections on certain regions of an image that attract their attention and might be leveraged by attackers during a guessing attack.

The service allows service providers to upload their own images or fetch images from the Internet through Google's Custom Search API *(https://developers.google.com/places/web-service/search)* and location-based search through Google's Place Search API *(https://developers.google.com/places/web-service/search)*. The service implements state-of-the-art computer vision algorithms to run along with the training dataset they wish to be used. The current prototype was developed in Python through the Django Web framework. Specifically, for the detection of objects, we integrated state-of-the-art object detection algorithms, such as, the Mask R-CNN trained on COCO (Tsung-Yi et al., 2014), YOLOv3 trained on the Open Images dataset (Kuznetsova et al., 2020), and Faster R-CNN trained on PASCAL VOC (Everingham et al., 2010) and Cityscapes (Cordts et al., 2016) datasets. **Figure 7** and **Figure 8** respectively illustrate the front-end user interface in which the user may upload an image for semantic analysis, and the results of an image analysis in which objects that are detected trough the image analysis tool are visually annotated on the image.



Photos / Progress Bar Upload

Examples

Progress Bar Upload

Search for query name in library

New Google Search

Uploaded photos by the user

media/api_images/d0233ec4-e360-4622-8e97-f88d3edde7e0.jpg
media/api_images/62e6e4aa-9d17-4cfc-b0aa-da4b0d5c6acf.jpg
media/api_images/90461f59-d0d8-470d-8f6f-804a0e265f91.jpg
media/api_images/dfbd224d-54f7-4be9-a573-a68a9a8f58f1.jpg
media/api_images/b55f2c7a-7cfb-4646-8c9c-e109c126171e.jpg
media/api_images/ba9ccc74-4e35-418d-a01a-a4bea5c5d261.jpg

Nearby Search

Photos found on the internet

Figure 7. Front-end user interface in which the user may upload an image for semantic analysis.

The image analysis tool has been designed to be extensible, by allowing easy adjustments to the existing algorithms, as well as easy addition of new algorithms. The aforementioned algorithms are implemented in Tensorflow (Abadi et al., 2016) and PyTorch (Paszke et al., 2019). The pre-trained weights of the models are stored on the server and are loaded in each request, so that the model can predict the objects present in the image along with their respective locations and bounding boxes. The interested reader may refer to Constantinides et al. (2021) for a comparative study among the aforementioned computer vision algorithms for assisting users in graphical password composition.

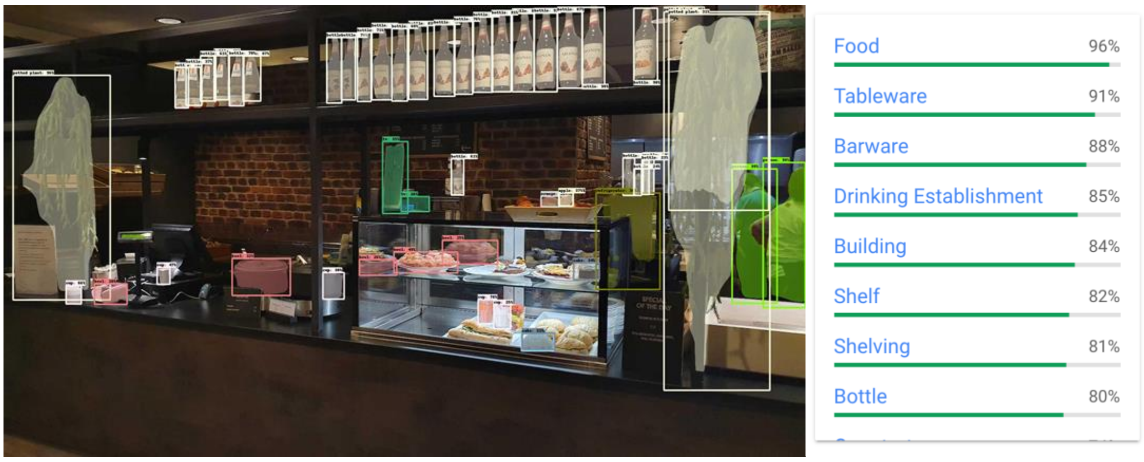


Figure 8. Objects detected through the image analysis tool.

5 Use-case Scenarios

The final version of the user authentication system builds on the third version of the system. For the tasks of the previous versions of the user authentication system, please refer to *D5.2 - Software on the Initial Verified User Authentication System* (Serums Deliverable 5.2) and *D5.3 - Software on the Refined Verified User Authentication System* (Serums Deliverable 5.3). The new tasks are the following: *i)* administrator login; *ii)* administrator creates and activates a user account; *iii)* updated activation page; *iv)* set two-factor authentication during registration; *v)* download mobile application and enroll user device; and *vi)* two-factor authentication approval page. The following tasks from the first version of the system remain the same: *i)* user-adaptable authentication; *ii)* request to reset secret; and *iii)* reset secret. *For completeness, we include existing stable use-case scenarios from previous deliverables of this work package (Deliverable 5.2; Deliverable 5.3).*

5.1 Administrator Login

The administrator login page aims to assure that an administrator[2] (*e.g.*, administrator from the end-user organizations) has the right to access the Serums' authentication administration page, which is primarily used to create end-user (*e.g.*, patient) accounts (**Figure 9**). In this phase, administrators enter their credentials, which consist of a unique username, a secret Web-based key and their organization. Then, the Authentication System validates the provided input details, leading to one of the following cases: *i)* if the administrator does not exist in the Database, an error message is communicated to the user interface with an informational message that the credentials are not correct; and *ii)* if the credentials' validation is successful, then an expiring API token is generated, and sent back to the administrator's user interface.

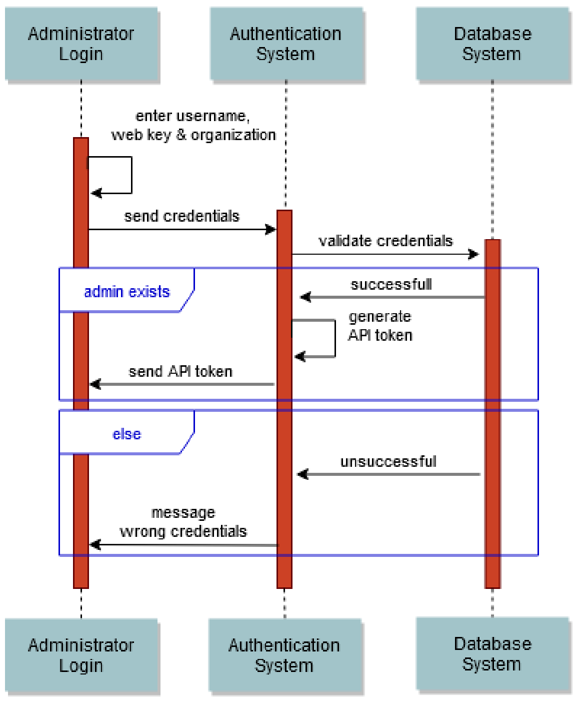


Figure 9. Administrator login sequence diagram.

[2] *To create system administrator accounts, we run a helper script that generates the accounts directly in the Database. This is a special type of user that can enrol a user of any of the following types: hospital_admin; medical_staff; and patient.*

5.2 Administrator Creates and Activates a User Account

In this step, an administrator creates a new user account for an end-user (*e.g.*, patient, doctor, etc.) (**Figure 10**). In this phase, the user initially enters the account details of the end-user. Then, the Authentication System checks the provided input details, leading to one of the following cases: *i)* if the user does not exist in the Database, the provided input details are stored in the Database, and an activation code is generated and sent to the Notification System. Then, an email including the activation code is sent to the end-user and a success operation is returned to the administrator; *ii)* if the user already exists in the Database, an unsuccessful operation is returned, along with a message notifying the user that the provided user profile already exists.

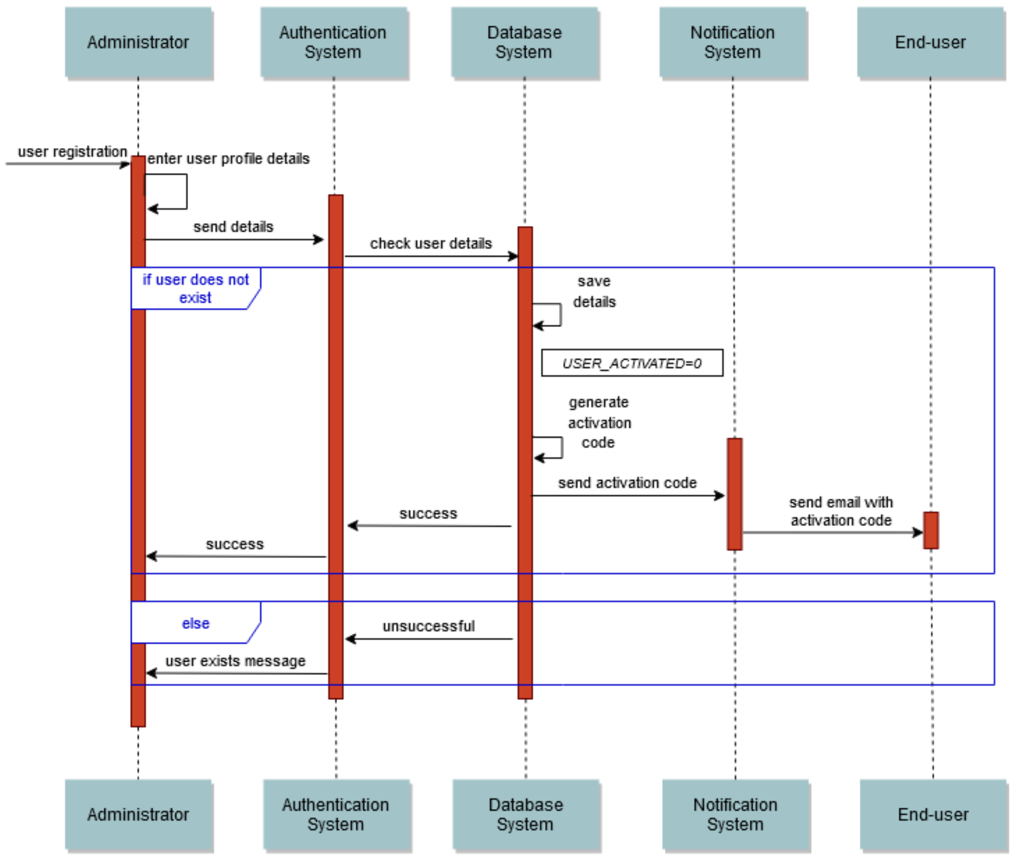


Figure 10. End-user registration and account activation by administrator.

5.3 Administrator Sends an Activation Code to User for Account Verification

In this step, a system administrator can send an activation code to an end-user's email (**Figure 11**). Then, the Authentication System checks the provided input details (*i.e.*, username), leading to one of the following cases: *i)* if the user does not exist in the Database, an unsuccessful operation is returned, along with a message notifying the system administrator that the provided user profile does not exist; *ii)* if the user exists in the Database and his/her account is already activated, a successful operation is returned, along with a message notifying the system administrator that the provided user profile is already activated; and *iii)* if the user exists in the Database and his/her account is not activated yet, an activation code is generated and sent to the Notification System. Then, an email including the activation code is sent to the end-user and a success operation is returned to the administrator. Accordingly, the end-user can use the code received in their email during account verification.

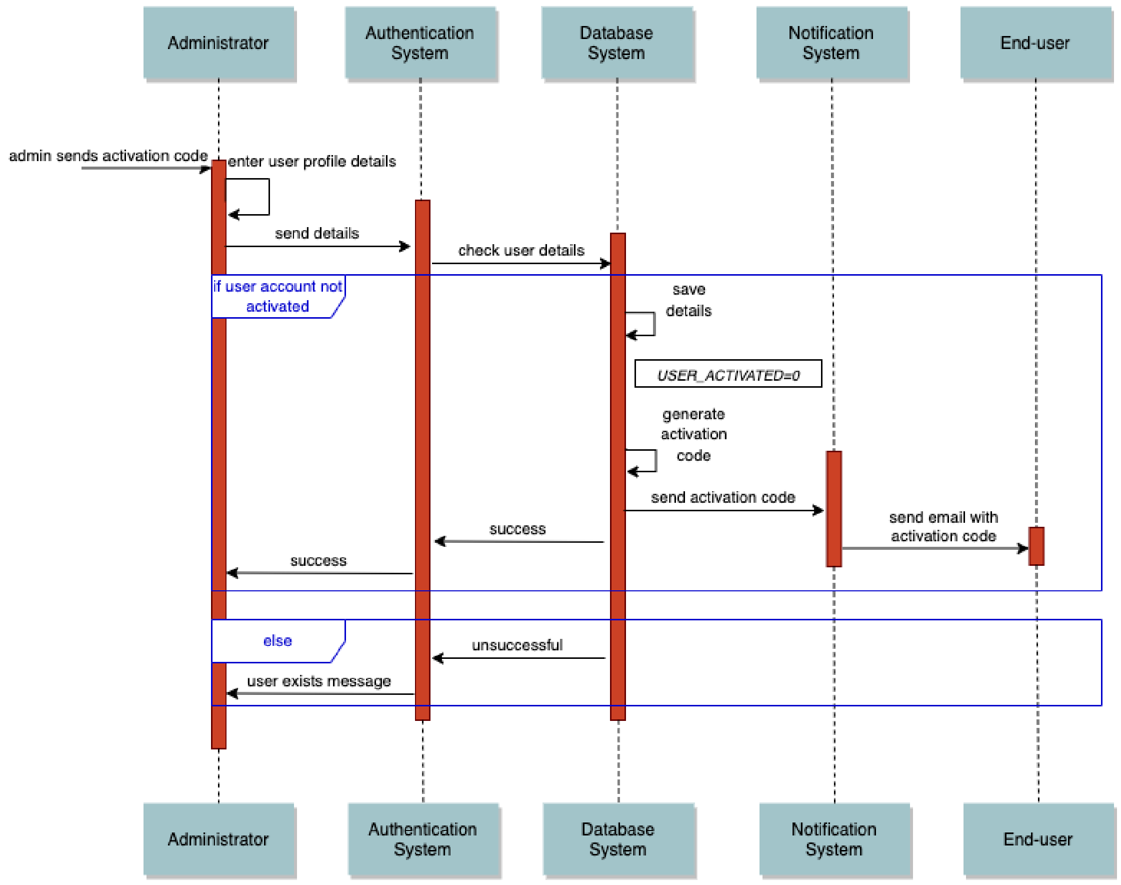


Figure 11. Administrator sends an activation code to end-user.

5.4 Administrator Sends a Reset Code to User for Account Reset

In this step, a system administrator can send a reset code to an end-user's email (**Figure 12**). Then, the Authentication System checks the provided input details (*i.e.*, username), leading to one of the following cases: *i)* if the user does not exist in the Database, an unsuccessful operation is returned, along with a message notifying the system administrator that the provided user profile does not exist; and *ii)* if the user exists in the Database and his/her account is not activated yet, an unsuccessful operation is returned, along with a message notifying the system administrator that the provided user profile is not activated yet; and *iii)* if the user exists in the Database and his/her account is already activated, a reset code is generated and sent to the Notification System. Then, an email including the reset code is sent to the end-user and a success operation is returned to the administrator. Accordingly, the end-user can use the code received in their email for the account reset.

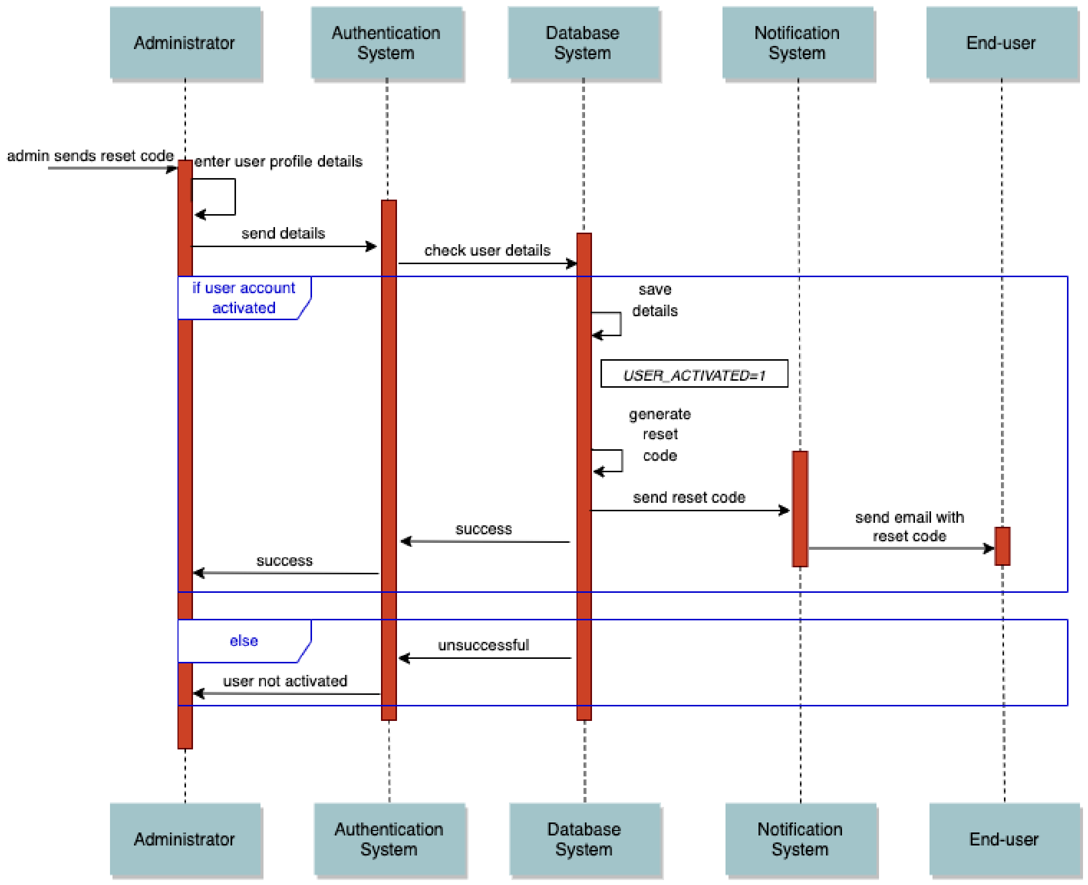


Figure 12. Administrator sends a reset code to end-user.

5.5 End-user Activates Account

In this step, the end-user activates the account that was created by the administrator (**Figure 13**). The user enters the email and the one-time password (activation code) received in the email. Then, the Authentication System checks the provided input details, leading to one of the following cases: *i)* if the provided details are valid, the user account is activated, a success operation is returned, and the user is redirected to the secret creation page; *ii)* if the provided details are not valid, an unsuccessful operation is returned, along with a message notifying the user that the provided credentials are wrong.

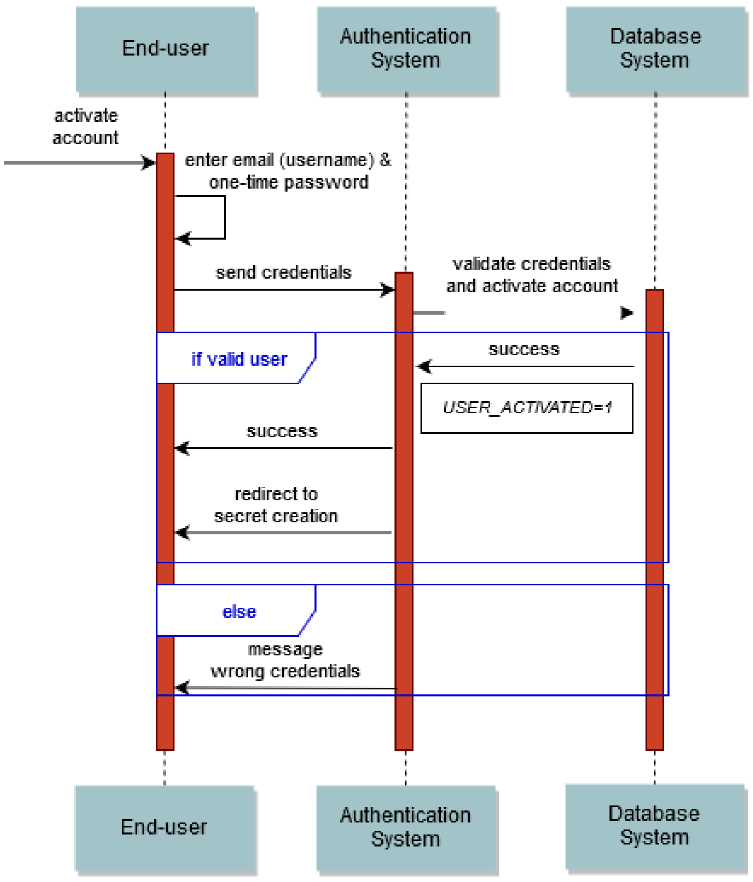


Figure 13. User account verification and activation.

5.6 Creation of the Graphical and Textual Password

The first version of the password creation phase (*D5.2 - Software on the Initial Verified User Authentication System*) has been adapted and includes two steps as follows. First, a grid of personalized images to limit to the set of images linked to their hospital is illustrated to the users, which illustrate sceneries from their hospitals. For the personalization of the images, we currently limit the set of images to a predefined set that contains images highly relevant to the participants' everyday activities and experiences within their healthcare environments. The users then select their preferred image, which is then used as a background image on which the users create a secret gesture-based password. Three types of gestures are allowed: taps, lines and circles. After creating the secret graphical password they wish, users may also (optionally) create a textual passphrase (including minimum 16 characters). **Figure 14** illustrates the sequence diagram for the creation of graphical password, and **Figure 15** illustrates the sequence diagram for the creation of a textual password.

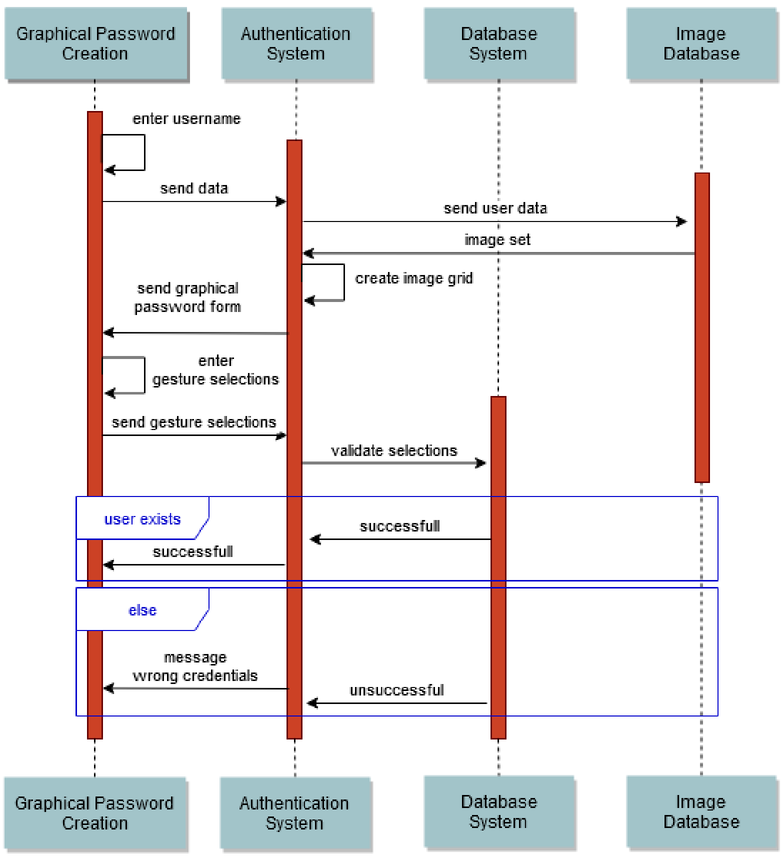


Figure 14. Creation of the graphical password.

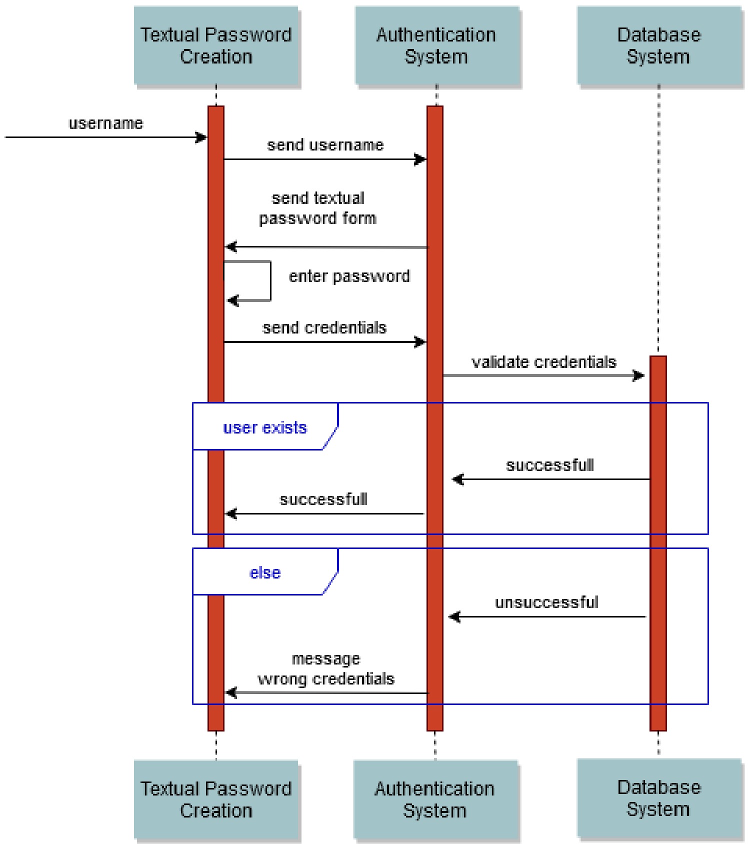


Figure 15. Creation of the textual password.

5.7 Enable Two-Factor Authentication Type and Pair Mobile Device

After successful creation of the graphical and/or textual password, the user may choose to set a second factor for authentication for increased security. For this purpose, a mobile application has been developed, which is utilized by the user to login. The mobile application is downloaded and installed by the user and then the user needs to pair the device with his/her Serums account. Before pairing the device, an enrollment code or a QR code is generated and sent to the Web-based user interface. **Figure 16** illustrates the sequence of interactions for generating the enrollment and QR codes.

Next, to pair the device, the user enters the enrolment code or scans the QR code through the mobile phone. When codes are valid, the 2FA feature is enabled and the mobile device of the user is successfully paired with the Serums account. Otherwise, an error message is communicated to the user. **Figure 17** illustrates the sequence of interactions for generating the enrollment and QR codes.

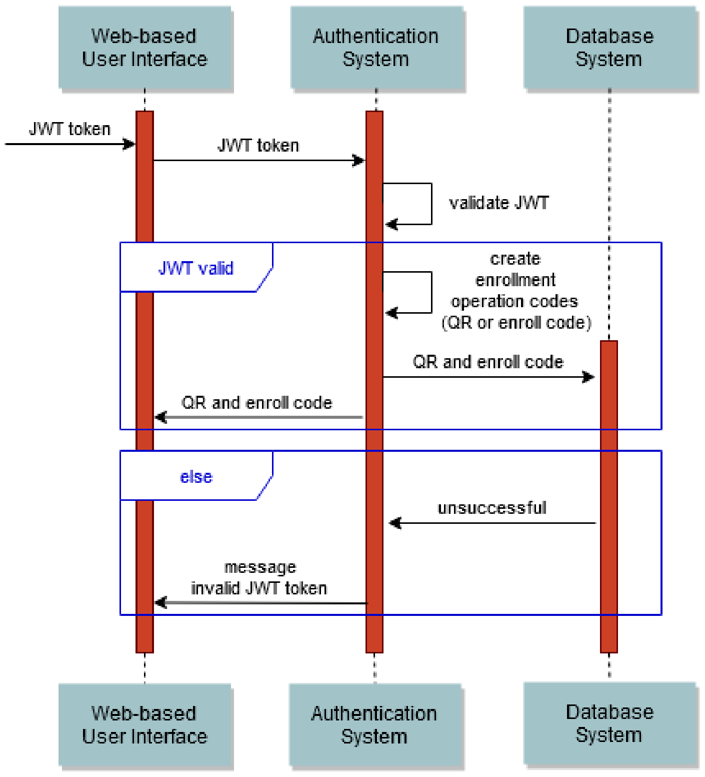


Figure 16. Creation of the textual password.

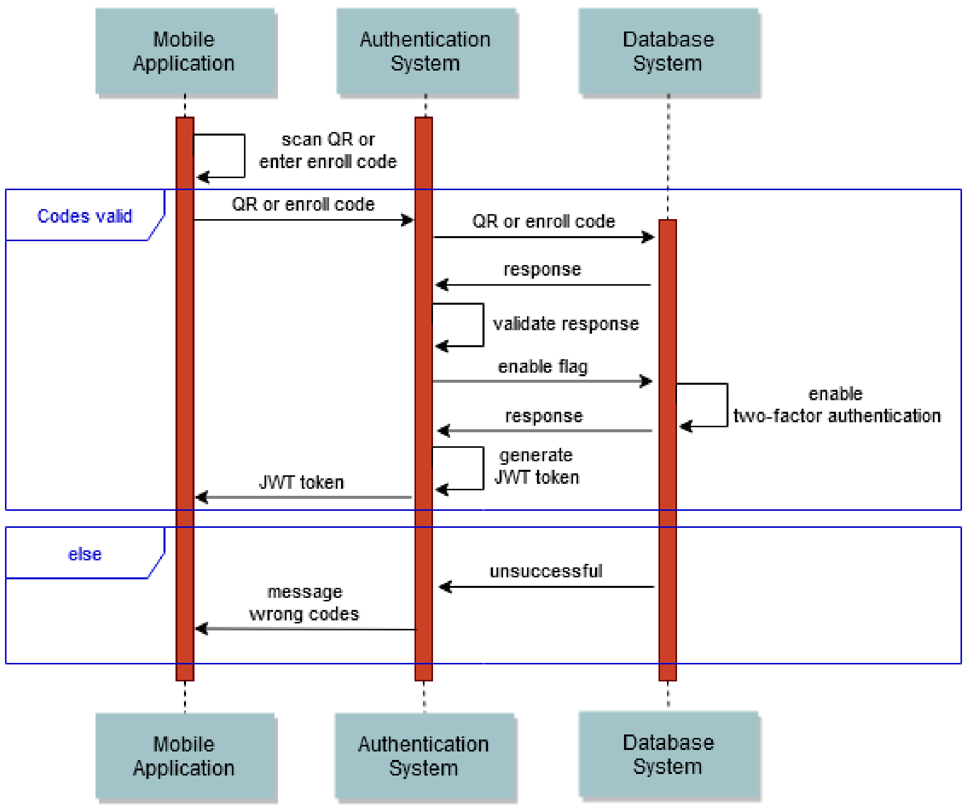


Figure 17. Pairing the user's device based on the enrollment code or the QR code.

5.8 Two-Factor Authentication Login using the Mobile Application

In this page, a user approves or rejects the two-factor authentication (2FA) login request through his/her smartphone's mobile application. Two types for 2FA are supported; login through a Time-based One-Time Password (TOTP), or through a mobile-based push notification. The user initially selects the preferred 2FA login type (TOTP or push notification). **Figure 18** illustrates the sequence of interactions for the generation of the two-factor authentication login types.

In case the TOTP option is selected, the login screen enables a textbox, waiting for the user to enter the code that can be found on the mobile application (**Figure 19**). In case the push notification option is selected, a request is made to the Google's Firebase Cloud Messaging Platform, which then sends the appropriate notification to the end-user's mobile application (**Figure 20**).

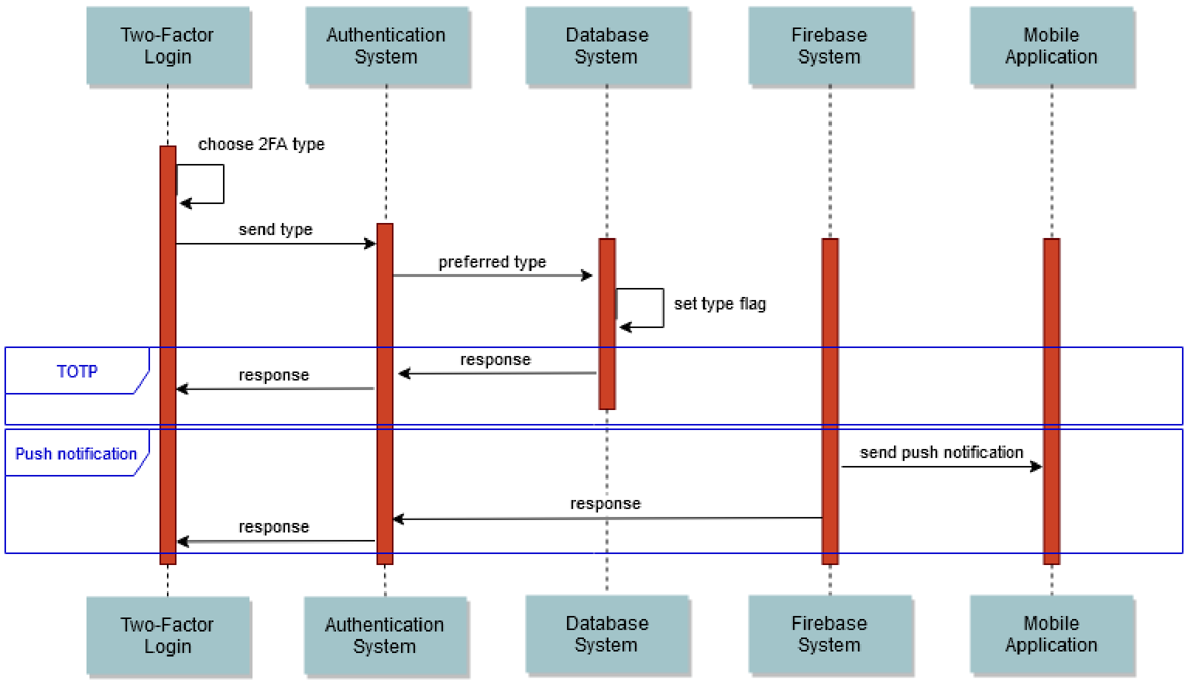


Figure 18. Generation of two-factor authentication login types.

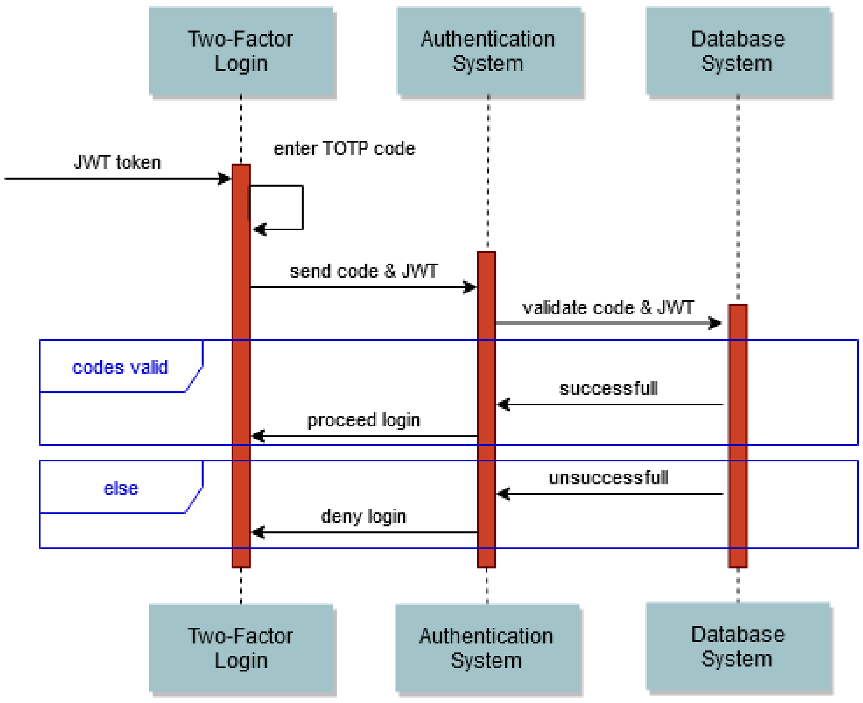


Figure 19. Two-factor authentication login through a time-based one-time password.

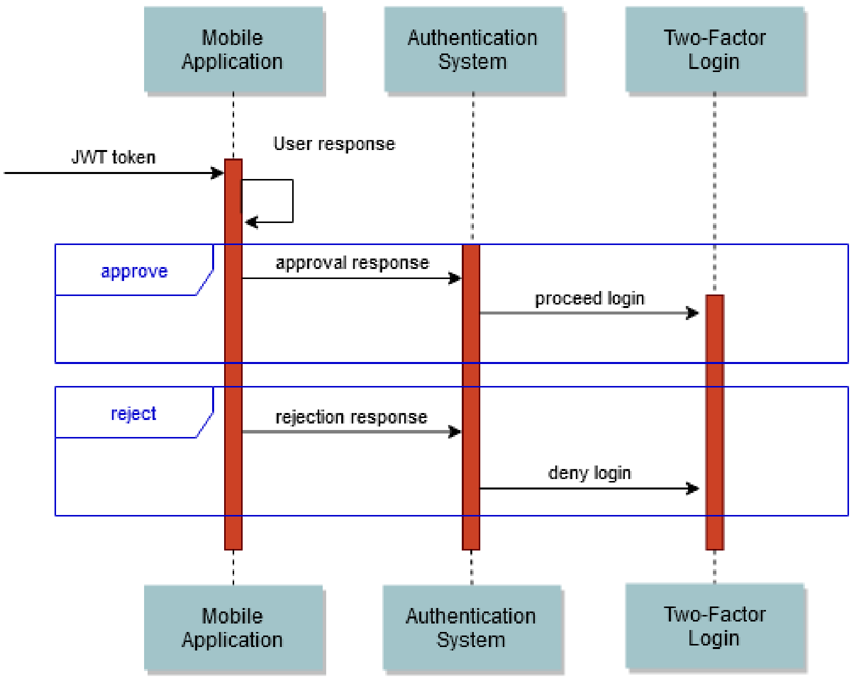


Figure 20. Two-factor authentication login through a mobile-based push notification.

5.9 Unpair Mobile Device from Two-Factor Authentication

Users have the option to opt out from two-factor authentication if they wish to do so. This can be performed via the "Remove" option in the main screen of the mobile application. After successful removal of the two-factor authentication option, users will login using only their graphical or textual password, without the additional layer of two-factor authentication. **Figure 21** illustrates the sequence of interactions for unpairing the mobile device from the two-factor authentication.

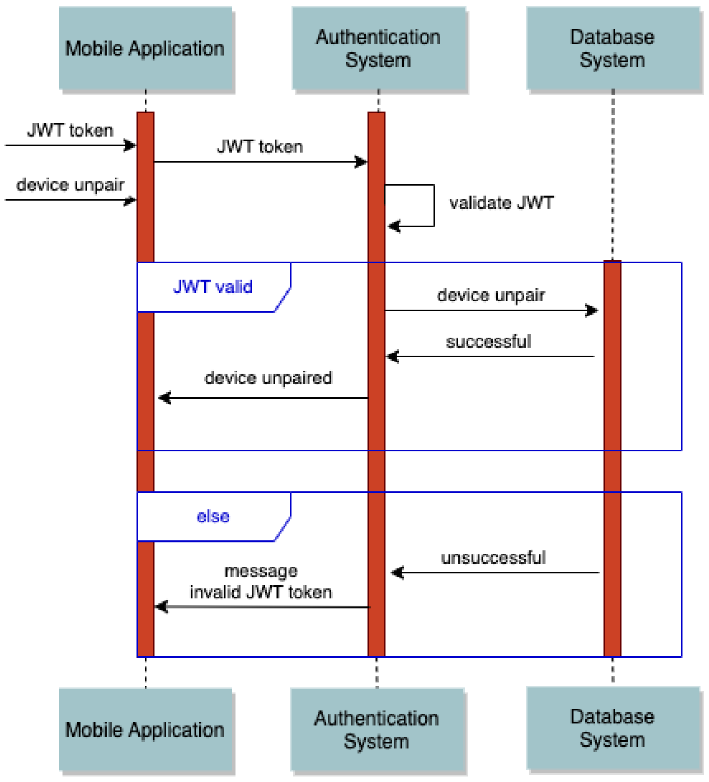


Figure 21. Unpair mobile device from two-factor authentication.

6 Verification of the Authentication System

6.1 Statistical Model Checking

The first approach being used for the verification of the Authentication System is based on formal methods. It requires a formal representation of both the system and the properties under verification. One of the most commonly used system representations is transition systems where system behavior is modelled with a set of states and relations between them describing how the system change states. In formal methods, properties are represented with temporal logic.

Model Checking is one of the techniques applicable to transition systems and temporal logic. The approach performs an exhaustive exploration of the state space of the system and, as a result, can guarantee that the property is satisfied. Unfortunately, model checking is subject to 'state space explosion': the size of the state space of non-trivial system can be extremely large and infeasible for the exploration. Statistical Model Checking (SMC) (Legay et al., 2010) is an alternative approach that combines ideas from model checking with statistics. The core idea of SMC is to run a large number of simulations of the system checking the property on each simulation and then to use statistical methods to decide the probability of the property to be satisfied. SMC has been broadly applied in different projects, *e.g.* (Gu et al., 2020' Basile et al., 2017; Noomene Ben Henda, 2014)

We have built a formal model of the Authentication System representing the developed software. The model also includes other components of the Serums system as well as other actors allowing us to reason about interactions between different components. Modelling of the whole Serums system is also a significant part of *WP6 - "Integration and Testing"*.

We used the Uppaal tool (Uppaal, 2022) that provides an expressive modelling formalism and model checkers allowing us to verify that the properties hold on the model. Uppaal models are defined as networks of timed automata. An example of an automaton is shown in **Figure 22**.

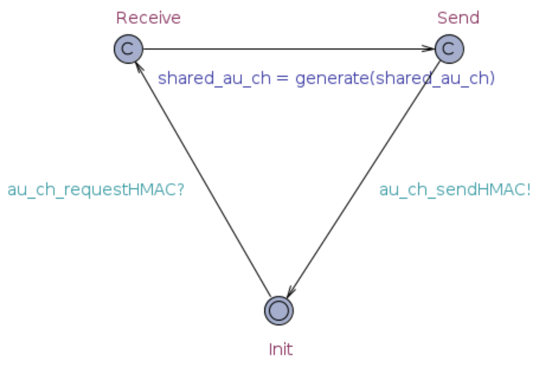


Figure 22. Credential Hardening timed automata model.

An automaton can be seen as a graph where nodes are states of the system (*e.g.*, 'Receive' state in **Figure 22** corresponds to the moment when Credential Hardening component receives a passphrase) and edges are transitions defining how the system change states (*e.g.*, the edge between 'Receive' and 'Send' states models the generation of an HMAC by the component and the preparation to send it). Each transition has a set of optional labels. A guard is a Boolean expression controlling the enablement of the transition (there is no guard depicted on **Figure 22**; an example could be a guard on the transition from the 'Init' state to the 'Receive' modelling the reception of a passphrase that blocks the transition if the passphrase is empty). The second label is an update, a sequence of actions that modify the variables of the model (a blue label on the transition between 'Receive' and 'Send' modifies the value of a variable *shared_au_ch*). The updates are defined with a subset of C language. The third label is a channel allowing automata to synchronize actions. Each channel is defined by a specific variable and transitions of two automata labelled with the same channel are synchronized, *i.e.*, they must be taken simultaneously. For example, the transition from the 'Init' state of **Figure 22** to the 'Receive' state is synchronized over the channel *au_ch_requestHMAC*. Another automaton sending the passphrase to the Credential Hardening component shall have a transition with the same channel and the two actions would be performed together. It is important to note that if two transitions are synchronized and one of the automata cannot take the transition the second one cannot take the transition either (*e.g.*, Credential Hardening component cannot receive a second passphrase before it finishes processing the first one). One of the two transitions shall be an initiator of the synchronization (indicated with '!') and the other is a receiver (indicated with '?'). In **Figure 22**, sending a generated HMAC is initialized by the Credential Hardening component while reception of the passphrase shall be initialized by another component.

Several timed automata are combined into a network via synchronizations and shared variables. At each point of time the network has three options to evolve to the next state: 1) by passing time 2) by one automaton making a transition that is not synchronized with other automata 3) by several automata making a simultaneous transition synchronized over the same channel.

The Authentication System is modelled with 4 automata: Back-end, Front-end, Credential Hardening and an auxiliary automaton for JWT refresh procedure. The model also includes automata for the 'environment' of the Authentication System, including Smart Health Center System (a front-end component of the SERUMS system interacting with users), patients, doctors, and administrators. The model of the Backend is shown in **Figure 23**. The central state is the initial state of the automaton. This automaton does not initiate any action: it waits for inputs from other components. After receiving a request, the component performs a set of actions modelling to the implemented software. Request procession is represented by one of the petals in the model. Following the software code, requests are checked for correctness. Some requests verify only the presence of required parameters (in the model they are sent via shared variables), while other require additional checks, for example user being registered, passphrase, and JWT. These checks are located in the transition guards. If any of the check fails, the corresponding transition is taken that returns an error message via the corresponding channel. In case of all checks are passed, the automaton performs the required actions and return the result to the requestor. For example, if the component receives a request to create a JWT (middle-right part of the model) via *au_pga_create_jwt?* channel (the symbol '?' indicates that Backend is not an initiator of the synchronization) it takes the value from the shared variable and parses it in order to obtain three parameters: username, type of passphrase (graphical or text base) and the passphrase. The outgoing transitions from the state has guards *!checkInput(),* and *!checkInput() && !hasInDbVerif.* The *checkInput* function returns whether the parsing has been successful while the *hasInDbVerif* function checks the presence of a user in the database. If any of the functions return *false*, the automaton notifies the requestor via *au_pga_incorrect_request!* channel (where '!' indicates that the Backend initiates the notification) or *au_pga_no_user!* channel. If both checks are passed, the automaton selects the next transition based on the value of *current_val_3* variable that is updated during the parsing of the authentication type. The graphical password directly checks the hash while text-based passphrase interacts with the Credential Hardening automaton via *au_ch_requestHMAC!* channel and wait for the reply from the Credential Hardening component. The reply is compared with the correct hashed passphrase (textual or graphical depending on the authentication type parameter) and it returns a JWT in case of successful match, otherwise, it returns a wrong credentials message.

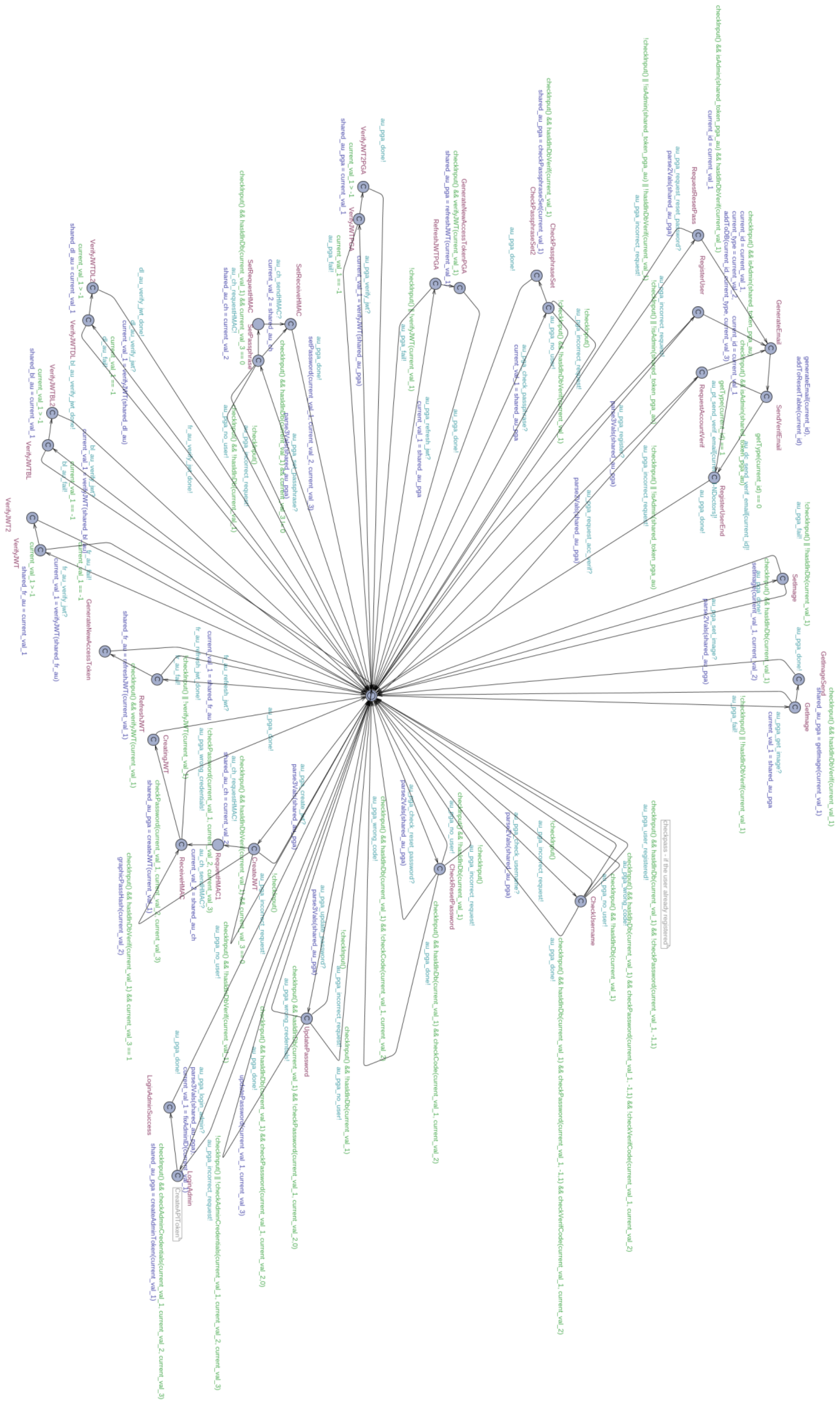


Figure 23. Authentication Back-end timed automata model

We do not model the task queue and the database as separate automata. The design of the model ensures that a new task cannot arrive before the end of processing of the previous task while the database is incorporated inside the automata variables.

The model for Credential Hardening component (**Figure 22**) is simple: it gets the passphrase from the User Authentication component, creates a HMAC and sends it back. Note that the model being an abstraction of the original system does not include full implementation of the HMAC procedure. Since the implementation of the function is taken from the standard *openssl* library, we assume the implementation to be correct and, in the model, we assume that the original passphrase cannot be recovered from the HMAC. The function generated in the model is returning the input.

The model for the front-end component is split into three parts for the ease of presentation. These parts are shown in **Figures 24, 25** and **26**. First part represents the signup procedure. It starts from the bottom right state by getting the username followed by creation of the graphical password, and textual passphrase. In **Figure 24**, the procedure follows the circle in a counterclockwise direction and in case of any failure the model moves to the state in the center followed by patient notification. The second part models the login procedure, which starts with receiving the username. The Backend component shows which type of identification is available for the user and requests a passphrase or a graphical password afterwards. In **Figure 25**, the procedure starts from the bottom left state and moves in a counterclockwise direction. In case of successful password check, the Front-end returns a JWT to the user. The Front-end component interacts with both Backend and Users. The final part models interactions with administrators: login, logout, creation of a new user, and password reset.

Figure 27 illustrates the interaction of a patient with the Authentication system. A sequence of actions showed in the right-top part of the **Figure 27** shows the actions required by a Sign-up procedure. The left part of the figure shows the Login actions. Both procedures follow the circle in a counterclockwise direction from the bottom right state. The model of a doctor is similar. Administrators are modelled with an automaton shown in **Figure 28**.

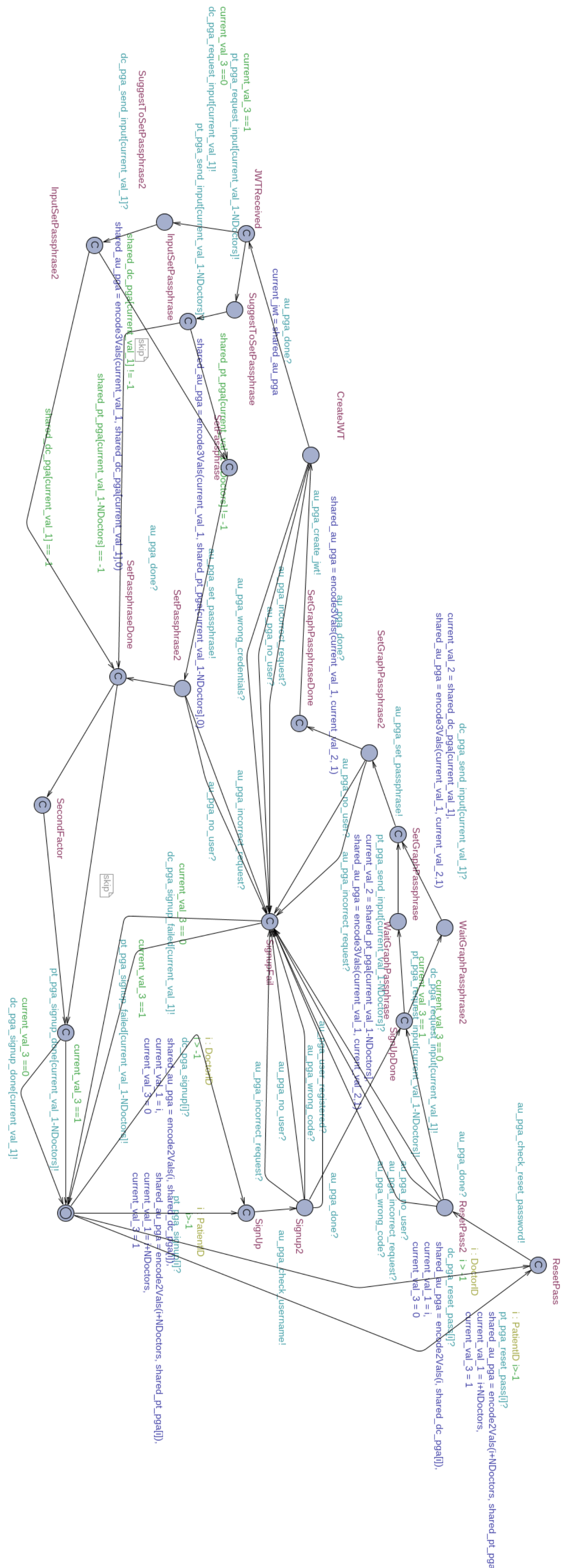


Figure 24. Authentication front-end: Sign up procedure model.

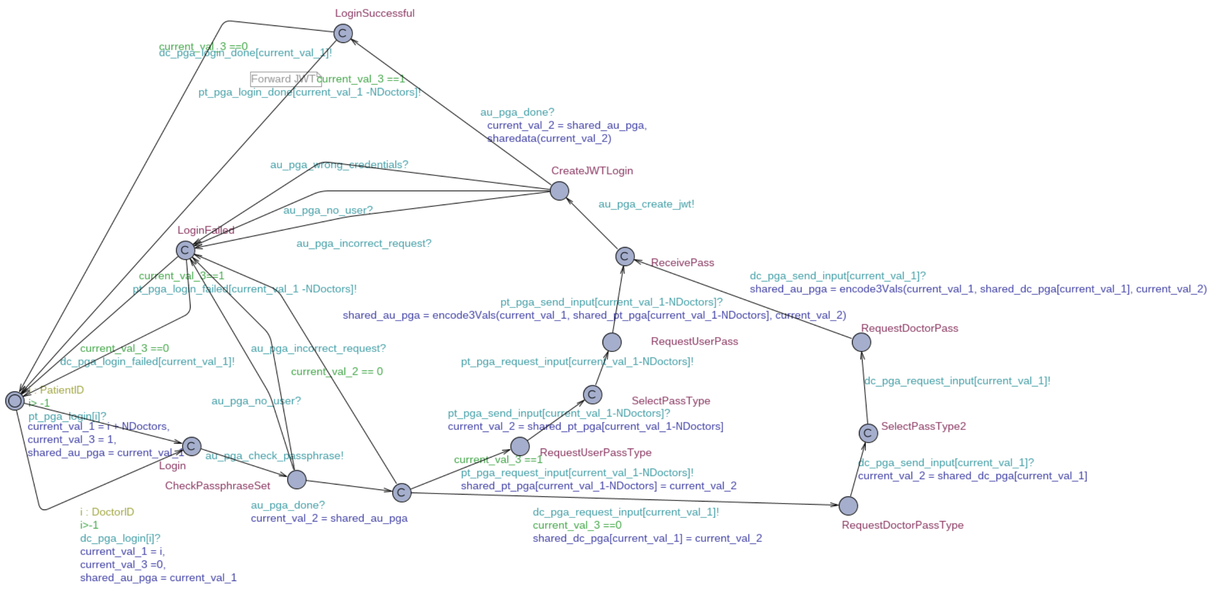

LoginSuccessful
current_val_3 ==0
dc_pga_login_done[current_val_1]!
Forward JWT
current_val_3 ==1
pt_pga_login_done[current_val_1 -NDoctors]!
au_pga_done?
current_val_2 = shared_au_pga,
sharedata(current_val_2)
au_pga_wrong_credentials?
CreateJWTLogin
au_pga_no_user?
au_pga_create_jwt!
LoginFailed
au_pga_incorrect_request?
ReceivePass
dc_pga_send_input[current_val_1]?
shared_au_pga = encode3Vals(current_val_1, shared_dc_pga[current_val_1], current_val_2)
current_val_3==1
pt_pga_login_failed[current_val_1 -NDoctors]!
pt_pga_send_input[current_val_1-NDoctors]?
shared_au_pga = encode3Vals(current_val_1, shared_pt_pga[current_val_1-NDoctors], current_val_2)
RequestDoctorPass
RequestUserPass
current_val_3 ==0
dc_pga_login_failed[current_val_1]!
au_pga_incorrect_request?
current_val_2 == 0
pt_pga_request_input[current_val_1-NDoctors]!
dc_pga_request_input[current_val_1]!
i : PatientID
i>-1
au_pga_no_user?
SelectPassType
pt_pga_send_input[current_val_1-NDoctors]?
current_val_2 = shared_pt_pga[current_val_1-NDoctors]
SelectPassType2
pt_pga_login[i]?
current_val_1 = i + NDoctors,
current_val_3 = 1,
shared_au_pga = current_val_1
Login
au_pga_check_passphrase!
current_val_3 ==1
RequestUserPassType
pt_pga_request_input[current_val_1-NDoctors]!
shared_pt_pga[current_val_1-NDoctors] = current_val_2
dc_pga_send_input[current_val_1]?
current_val_2 = shared_dc_pga[current_val_1]
CheckPassphraseSet
au_pga_done?
current_val_2 = shared_au_pga
dc_pga_request_input[current_val_1]!
current_val_3 ==0
shared_dc_pga[current_val_1] = current_val_2
RequestDoctorPassType
i : DoctorID
i>-1
dc_pga_login[i]?
current_val_1 = i,
current_val_3 =0,
shared_au_pga = current_val_1

Figure 25. Authentication front-end model: Login procedure model.

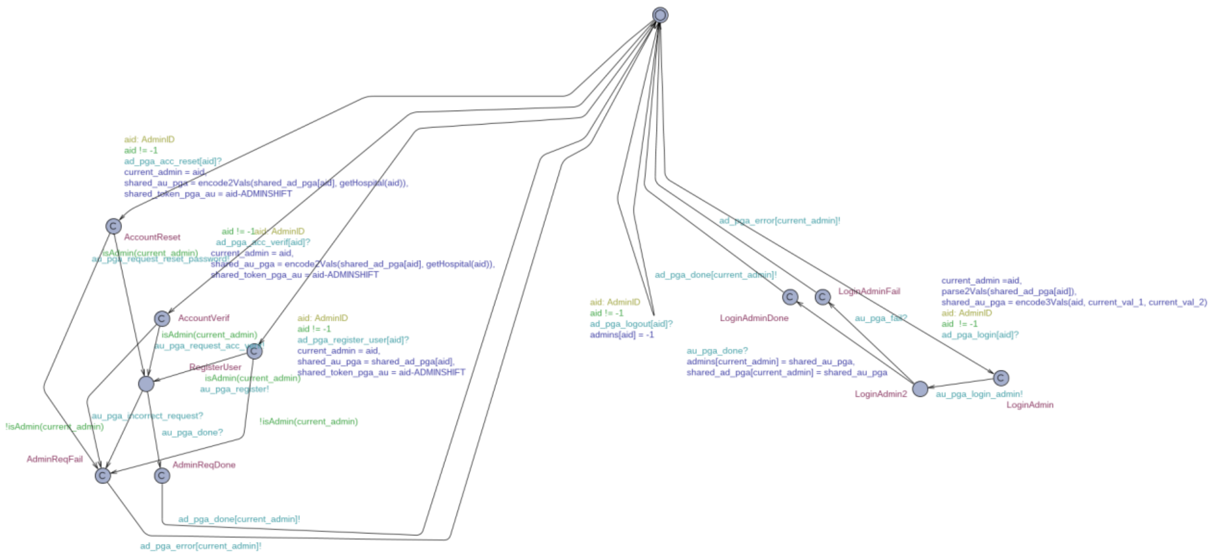

aid: AdminID
aid != -1
ad_pga_acc_reset[aid]?
current_admin = aid,
shared_au_pga = encode2Vals(shared_ad_pga[aid], getHospital(aid)),
shared_token_pga_au = aid-ADMINSHIFT
AccountReset
aid != -1
aid: AdminID
ad_pga_acc_verif[aid]?
current_admin = aid,
shared_au_pga = encode2Vals(shared_ad_pga[aid], getHospital(aid)),
shared_token_pga_au = aid-ADMINSHIFT
isAdmin(current_admin)
au_pga_request_reset_password!
AccountVerif
isAdmin(current_admin)
au_pga_request_acc_verif!
aid: AdminID
aid != -1
ad_pga_register_user[aid]?
current_admin = aid,
shared_au_pga = shared_ad_pga[aid],
shared_token_pga_au = aid-ADMINSHIFT
RegisterUser
isAdmin(current_admin)
au_pga_register!
au_pga_incorrect_request?
!isAdmin(current_admin)
au_pga_done?
!isAdmin(current_admin)
AdminReqFail
AdminReqDone
ad_pga_done[current_admin]!
ad_pga_error[current_admin]!
ad_pga_error[current_admin]!
ad_pga_done[current_admin]!
aid: AdminID
aid != -1
ad_pga_logout[aid]?
admins[aid] = -1
LoginAdminDone
LoginAdminFail
au_pga_fail?
current_admin =aid,
parse2Vals(shared_ad_pga[aid]),
shared_au_pga = encode3Vals(aid, current_val_1, current_val_2)
aid: AdminID
aid != -1
ad_pga_login[aid]?
au_pga_done?
admins[current_admin] = shared_au_pga,
shared_ad_pga[current_admin] = shared_au_pga
LoginAdmin2
au_pga_login_admin!
LoginAdmin

Figure 26. Authentication front-end model: Administrative procedures model.

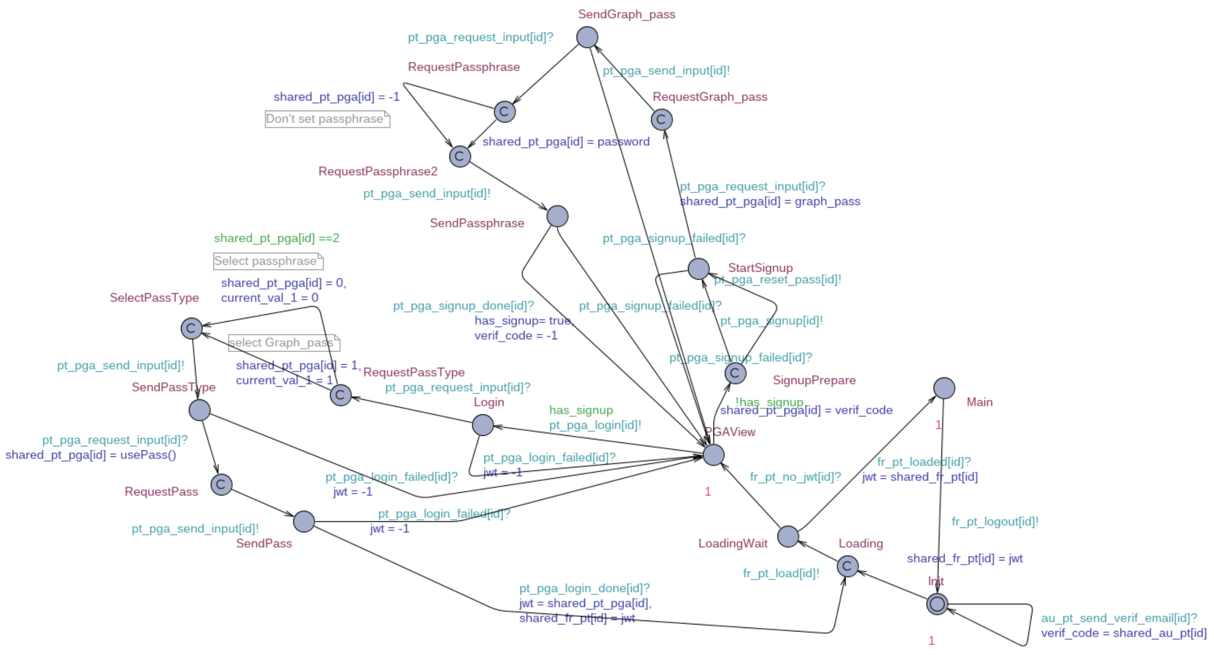


Figure 27. Patient model for the authentication system.

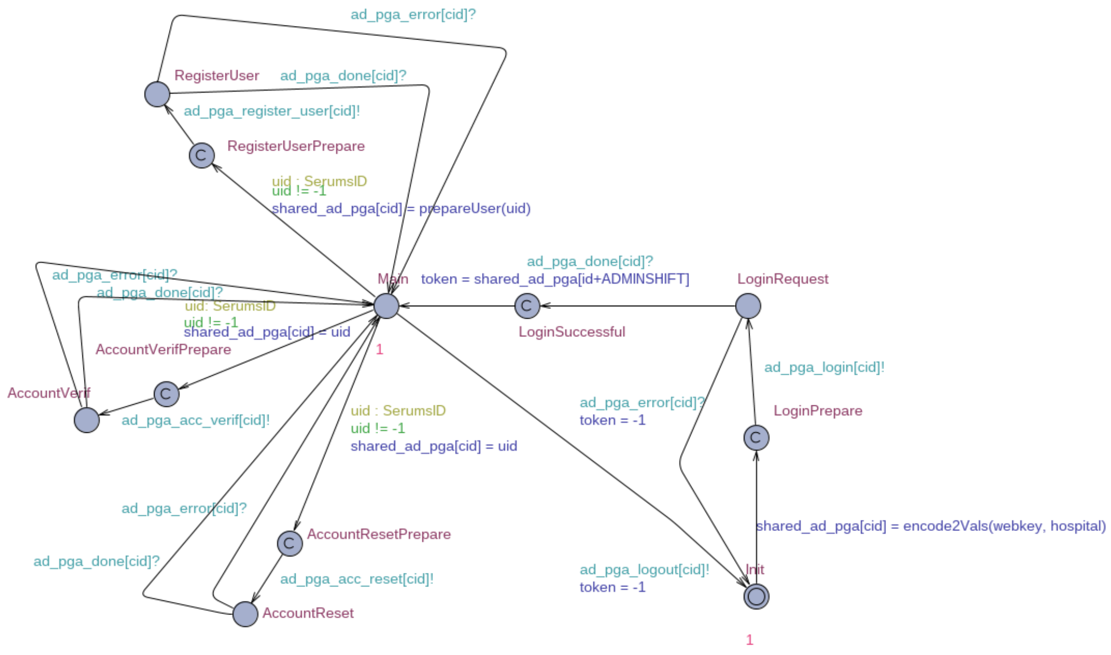


Figure 28. Administrator automaton.

6.2 Properties Verification

For the verification of a model, the properties are checked with the Uppaal statistical model checker. The properties shall be expressed in the Uppaal query language based on a simplified version of Metric Interval Temporal Logic (MITL). Basic temporal operators in temporal logics are *[]p* and *<>p*. The former checks that *p* holds in all states, and the latter checks that *p* holds in at least one future state. The properties has the following format: *Pr[# <= N] F*, where *F* is the property to verify, *N* is the maximal length of traces, and # indicates that we consider the number of transitions. Intuitively, the property checks the probability of *F* to be satisfied on traces containing at most *N* transitions. Results returned by the SMC checker have the meaning depending on a temporal operator used in *F*. For <> operator, the property is considered satisfied if the probability is not close to 0 and unsatisfied otherwise. For *[]* operator, the property is satisfied if the probability is close to 1 and unsatisfied otherwise.

For the verification of the Authentication System properties, we have removed the automata that are not interacting with the Authentication System and simplified the remaining automata from the environment in order to focus on the properties related to the system under verification. We limited the trace length to 10000 steps: such length is sufficient to involve hundreds of interactions with the Authentication System. Following, we list the properties with their description and their encoding in MITL that we have verified. We omit *Pr[# <= 10000]* from the formulas in the list. We use *-1* as *null* or *empty* value for variables.

- The model does not deadlock, i.e. the model does not have a state from which it cannot progress. This is checked with the following query: *[] !deadlock*.
- Users can login to the system provided they use the correct graphical password or passphrase. The query is: *<>(Doctor.Main),* where *Main* is the state of Doctor automaton reached after successful login. The query for patients is similar. An extended version of this query is *<>(Doctor.Main && Doctor.jwt != -1),* that in addition checks that at the state where the user is logged in the user also has a JWT.
- Users cannot login to the system without providing the correct graphical password or passphrase. For this query we modified the users automaton forcing to provide an incorrect graphical password or passphrase. *[](!PatientWithoutPassword.Main)*. An alternative option checks that such patient cannot receive a JWT: [](*PatientWithoutPassword.jwt* == *-1*)
- The user cannot login and receive a JWT if he has not signed up. The property is checked with the following query: *[] ((Patient.jwt != -1) => Patient.has_signedup)*. Each Patient has a Boolean variable *has_signedup* initialized to *False*. After successful completion of the Sign up procedure, the variable is set to *True*.
- An issued JWT can be verified by the Authentication System. This property can be checked by adding an additional transition synchronized with the Backend on a channel *au_pga_verify_jwt* and adding a Boolean variable that is set to *True* if the JWT verification fails. The model checker verifies that the property holds: *[] (! Backend.jwt_unverified)*.
- A dual to the previous property: a fake JWT fails verification by the Authentication system. The Front-end component sends a fake JWT to the Backend with the assumption that the key used to sign the JWT is unknown to the creator of the fake JWT. A Boolean variable is set to *True* if the JWT verification succeeds: *[] (! Backend.jwt_approved)*.

- A user cannot sign up without being registered by an admin. For this query we force admins to not register a *Patient0* and check *[](!Patient0.has_signedup)*.
- A user can sign up provided that there is an admin registering users. <> *(Patient.has_signedup)*. Note that during simulation, admins may never register a particular user and therefore the query cannot be satisfied with the probability close to 1, but for queries with <> temporal operator we check the probability being not close to 0.
- A user cannot change password without being reset by an admin. We extend the behavior of one of the patients with password change option. Admins are forced not to reset the patient password. At the end of the password change procedure, there is a state *PassChanged* reachable in case of successful password change and the query is: *[] (! Patient.PassChanged).*
- A dual property that a user can change password when admins are allowed to reset passwords. <> *(Patient.PassChanged).*
- An admin can login to the Authentication System provided usage of a correct password. <>*(Admin.LoginSuccessful).*
- A dual property – admin cannot login without correct password. *[](!AdminWithoutPassword.LoginSuccessful).*
- The following group of properties is constructed via a single scheme. We generate an automaton of a user that tries to interact directly with the Backend automaton. It is done with an assumption that such user is an attacker, and the attacker has knowledge of expected request parameters but not passwords of other users. If the Backend returns successful result, the attack reach the state *Success*. The query is *[] (!Attacker.Success)*. The property has not been verified by several backend requests. Further exploration showed that while some of the request parameters are checked in the request processing code, authorization parameters are added as an annotation to the function and processed by Django Framework. Therefore, we had to update the model in order to take into account these annotations. The query has been checked on a modified model and the property failed with *set_graphical_password* request. Exploration of the deployed system showed that indeed this request can be executed without proper authorization, and return a success message in the case a password reset has been requested but not performed, at the moment of attack, which limits its application.

6.3 Fuzzing

The second approach used for the verification of Authentication System is fuzzing. Fuzzing (short for fuzz testing) is an effective and widely used technique for finding security bugs and vulnerabilities in software. It inputs irregular test data into a target program to try to trigger a vulnerable condition in the program execution (Chen et al., 2018). Technically, fuzzing tests a system with the continuous processing of test cases generated by another program. At the same time, the system is monitored to expose any defects revealed by processing this input.

Fuzzing techniques can be divided into three kinds: black box, white box, and gray box depending on how much information they require from the target program at runtime (Jääskelä, 2016). Black-box fuzzers are unaware of the internal program structure, that is, their target is a black-box, no feedback other than what is directly observable is provided (Van Rooij et al., 2021). One of their main advantages is their low overhead which allows them to exercise the program under test with millions of inputs. In this way, their chances of triggering a bug increase. On the other hand, their lack of knowledge on the program's structure comes with a cost.

White-box fuzzing is based on the knowledge of internal logic of the target program (DeMott, 2006). It uses a method which in theory can explore all execution paths in the target program. Unlike black-box fuzzing, white-box fuzzing requires information from the target program and uses the required information to guide test case generation. Specifically, starting execution with a given concrete input, a white-box fuzzer first gathers symbolic constraints at all conditional statements along the execution path under the input.

Gray-box fuzzing stands in the middle of black-box fuzzing and white-box fuzzing to effectively reveal software errors with partial knowledge of the target program. By this means, a gray-box fuzzer can obtain code coverage of the target program at runtime; then, it utilizes this information to adjust its mutation strategies to create test cases which may cover more execution paths or find bugs faster. Gray-box fuzzing only uses the acquired information to guide test case generation, but it cannot guarantee that using this piece of information will surely generate better test cases to cover new paths or trigger specific bugs. By contrast, white-box fuzzing utilizes source codes or binary codes of the target program to systemically explore all the execution paths.

Although several fuzzing tools have been developed over the years (**Figure 29**), they are mostly designed for local executables and not usable for web applications like our targets.

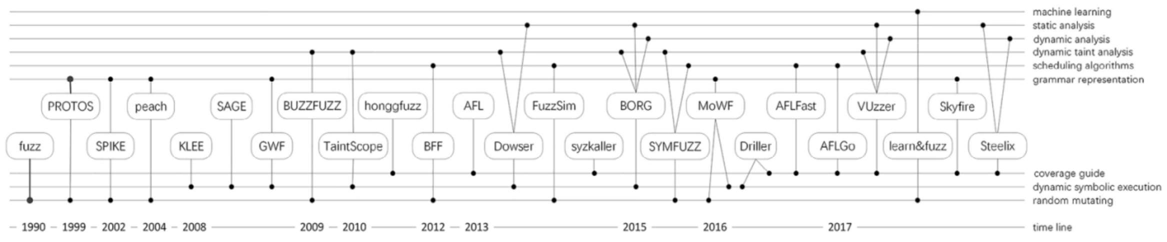
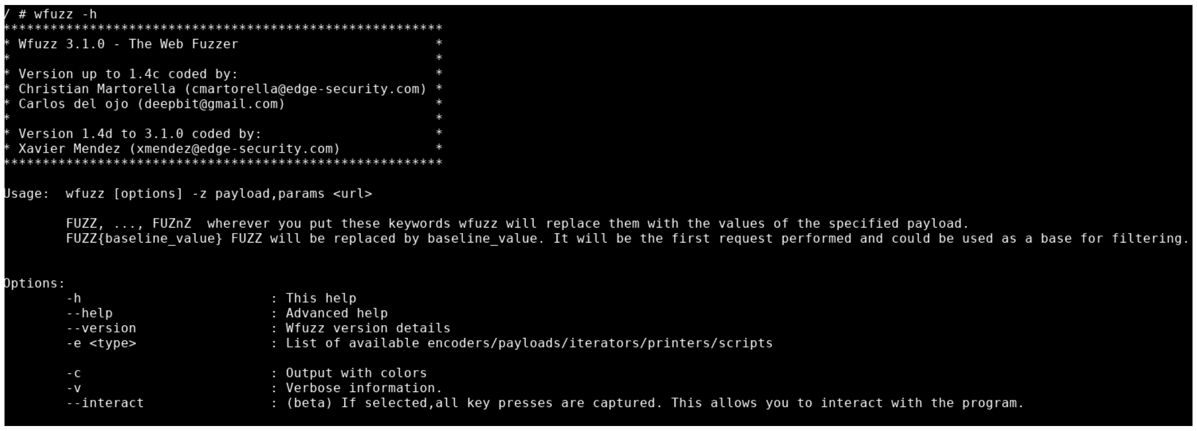


Figure 29. Fuzzing Technique Evolution Diagram (Chen et al., 2018).

A black-box fuzzer: wfuzz (*http://wfuzz.io* – **Figure 30**), is one of few appropriate fuzzers publicly available for our targets, *i.e.,* front and back-end web applications for the Authentication System. The wfuzz allows any input to be injected in any field of an HTTP request since it is based on a simple concept: it replaces any reference to the FUZZ keyword by the value of a given payload.



```
/ # wfuzz -h
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
*                                                      *
* Version up to 1.4c coded by:                         *
* Christian Martorella (cmartorella@edge-security.com) *
* Carlos del ojo (deepbit@gmail.com)                   *
*                                                      *
* Version 1.4d to 3.1.0 coded by:                      *
* Xavier Mendez (xmendez@edge-security.com)            *
********************************************************

Usage:  wfuzz [options] -z payload,params <url>

        FUZZ, ..., FUZnZ  wherever you put these keywords wfuzz will replace them with the values of the specified payload.
        FUZZ{baseline_value} FUZZ will be replaced by baseline_value. It will be the first request performed and could be used as a base for filtering.


Options:
        -h                        : This help
        --help                    : Advanced help
        --version                 : Wfuzz version details
        -e <type>                 : List of available encoders/payloads/iterators/printers/scripts

        -c                        : Output with colors
        -v                        : Verbose information.
        --interact                : (beta) If selected,all key presses are captured. This allows you to interact with the program.
```

Figure 30. WFuzz – A black-box fuzzer for testing.

We performed the black-box fuzzing along with pre-loaded http sessions in order to mimic the gray-box fuzzing throughout our target web applications using the wfuzz.

6.4 Fuzzing Checks

We performed the fuzzing to the front and backend web applications for the Authentication System. More specifically, we scanned vulnerable paths within or without a user session and tested user login procedure.

To scan for the paths to access the system we used the word list provided by wfuzz and collected accessible paths in front- and back-ends.

```
for f in wfuzz/wordlist/general/* #for all wordlist files in general category

do
  #Runnning wfuzz with hiding the response of code: 404 and 403 along two different common directories
  wfuzz -w $f --hc 404,403 https://localhost:9001/web_app/login_admin/FUZZ

  wfuzz -w $f --hc 404,403 https://localhost:9001/web_app/FUZZ

done
```

In addition to these, we performed fuzzing on values of random cookie parameters and recursive fuzzing of the paths up to the depth of three. In order to fuzz within a user session, we first obtained the cookies: *csrftoken*, *sessionid*, *jwt_access* and *jwt_refresh*, using the commands listed below, and then injected those cookies into the fuzzing requests.

```
#Access to login page and get csrftoken cookie
curl -k 'https://localhost:9001/web_app/login_username/' -o login_username.html -c cookie.txt

csrftoken=$(grep token cookie.txt |cut -f 7)

#Feed username : test@test.com and get session cookie

curl --data-urlencode username=test@test.com -k https://localhost:9001/web_app/check_passphrase_set -b cookie.txt -c cookie.txt -e https://localhost:9001/web_app/login_username -X POST -H "X-CSRFToken: $csrftoken"

sessionid=$(grep sessionid cookie.txt |cut -f 7)

 #Feed passpharase and get jwt_access and jwt_refresh cookies

curl -k https://localhost:9001/web_app/passphrase_login -b "csrftoken=$csrftoken; sessionid=$sessionid" -c cookie.txt -e https://localhost:9001/web_app/text_login.html -d 'username=test%40test.com&passphrase=00ph0120541pin63c70m9&total_time_until_submit=13565&total_time_until_submit_since_page_load=13285&time_interaction_started=0&is_reset=false' -H "X-CSRFToken: $csrftoken"

jwt_access=$(grep jwt_access cookie.txt |cut -f 7)

jwt_refresh=$(grep jwt_refresh cookie.txt |cut -f 7)
```

Commands for getting into a user session (username: test@test.com, passphrase: 00ph0120541pin63c70m9)

Front-end

Our scanning for accessible pages at front-end server led to the following pages. Note that the presence of the page is not a vulnerability but a potential entry point for further search. We found each of those pages is not associated with just one fixed url but with keywords from the same categories.

Without user session

Below shows the discovered pages without user session.

https://localhost:9001/web_app/add_user.php
Empty json string {} as the below image.

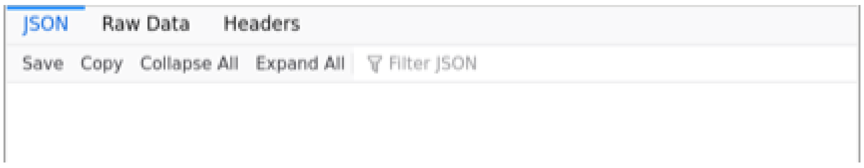
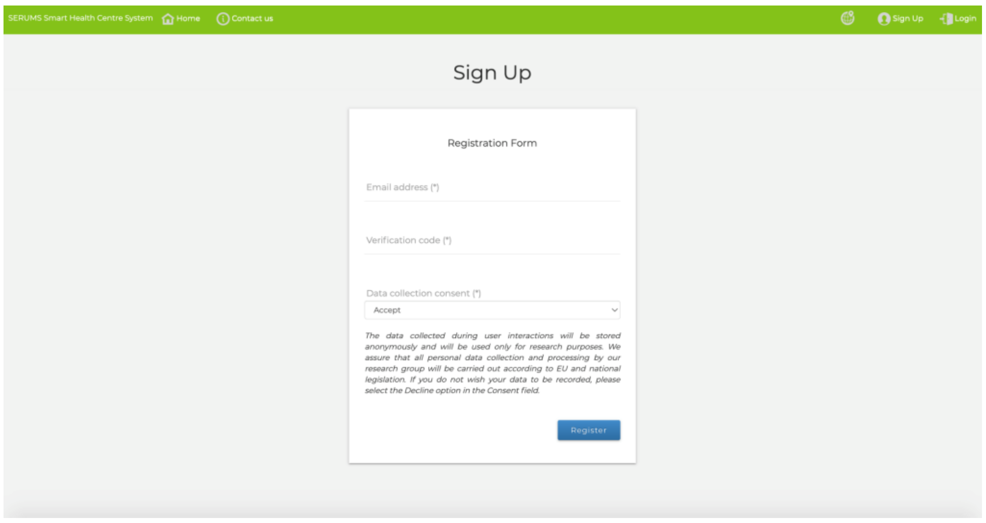


Figure 31. Empty json string {} as the below image.

https://localhost:9001/web_app/cgi-sys/realsignup.cgi
Registration form to sign up a user as the below image.



Sign Up

Registration Form

Email address (*)

Verification code (*)

Data collection consent (*)
Accept

The data collected during user interactions will be stored anonymously and will be used only for research purposes. We assure that all personal data collection and processing by our research group will be carried out according to EU and national legislation. If you do not wish your data to be recorded, please select the Decline option in the Consent field.

Register

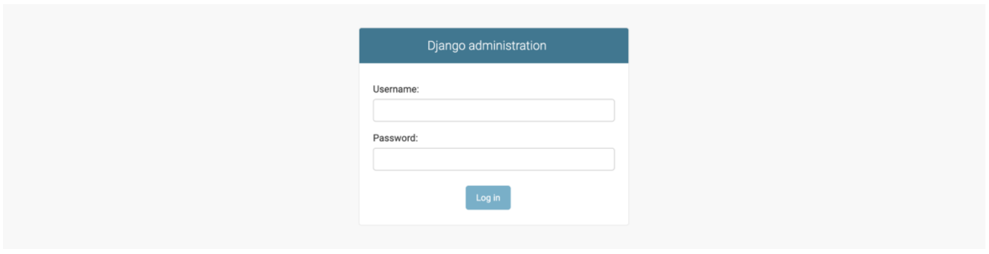
Figure 32. Registration form to sign up a user.

https://localhost:9001/web_app/login_admin/email
Login page of System Administrator

https://localhost:9001/web_app/login_admin/demo
The first page of Demonstration

https://localhost:9001/web_app/login_admin/index
Welcome page of Serums

https://localhost:9001/admin/ redirecting to *https://localhost:9001/admin/login/?next=/admin/*
Login page of Django administrator as the below image. Django administrator login/password does not coincide with the Serums administrator credentials, neither vulnerable standard login/password pairs work.



Django administration

Username:

Password:

Log in

Figure 33. Django administrator page.

https://localhost:9001/web_app/login_admin/images

redirects to https://localhost:9001/web_app/login_username

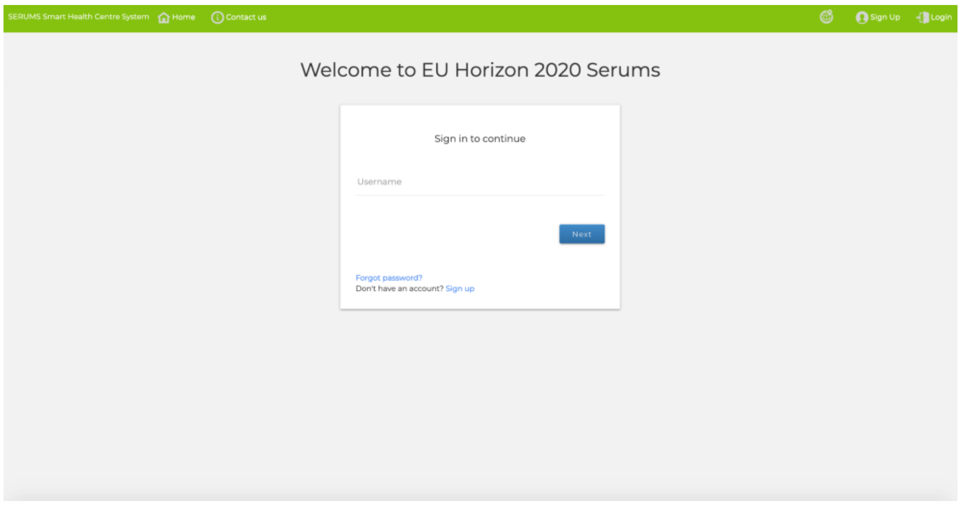
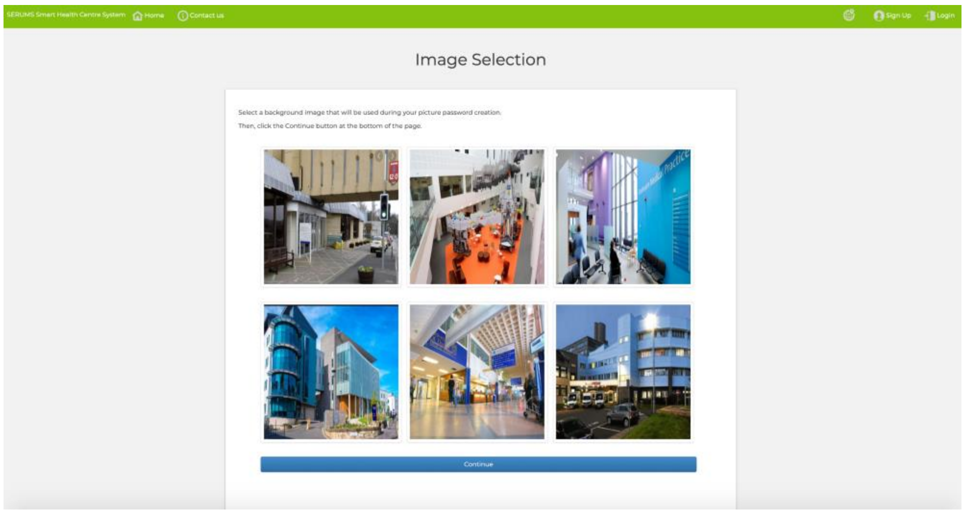

SERUMS Smart Health Centre System
Home
Contact us
Sign Up
Login
Welcome to EU Horizon 2020 Serums
Sign in to continue
Username
Next
Forgot password?
Don't have an account? Sign up

Figure 34. Sign in page.

Within a user session

In addition to the pages discovered without user session shown above, the following page was found by fuzzing within a user session.

https://localhost:9001/web_app/images
The image selection page for the gesture pass. This page shall be accessible only during creation or update of the graphical password; however it is accessible at any point of time from a user session. Further steps of the password update cannot be reached from this page: after pressing the continue button, the system checks that the password update has not been initiated and redirects to the login page.



Select a background image that will be used during your picture password creation.
Then, click the Continue button at the bottom of the page.

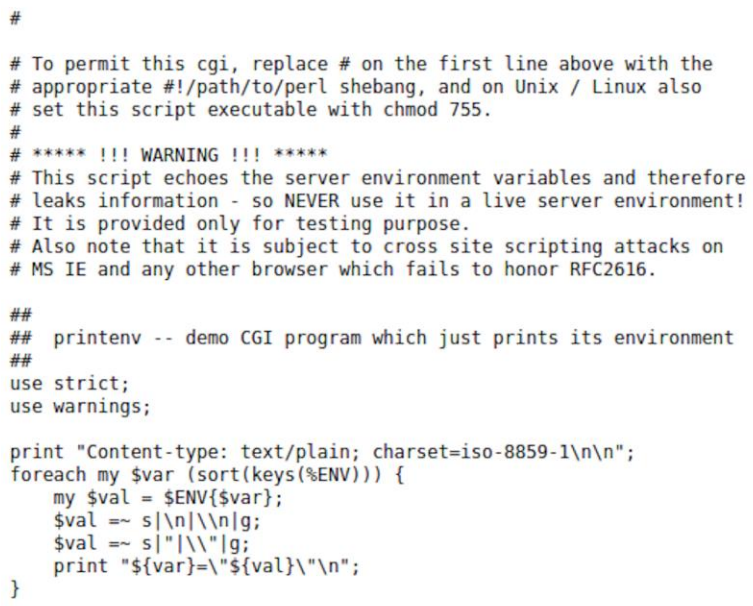
Figure 35. Image selection for creating a graphical password.

Back-end

Our fuzzing against the back-end server led to the following three pages.

https://localhost:9000/cgi-bin/printenv

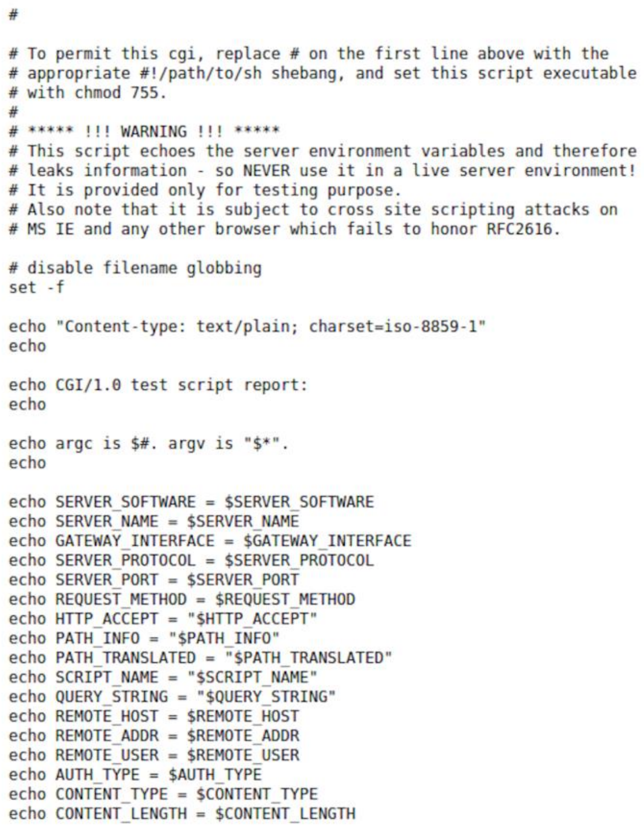
A testing script of the apache server. The script if allowed to be executed could show internal information from the server. This output and additional checks showed that the testing functionality is disabled and the vulnerability is not present.



```
#

# To permit this cgi, replace # on the first line above with the
# appropriate #!/path/to/perl shebang, and on Unix / Linux also
# set this script executable with chmod 755.
#
# ***** !!! WARNING !!! *****
# This script echoes the server environment variables and therefore
# leaks information - so NEVER use it in a live server environment!
# It is provided only for testing purpose.
# Also note that it is subject to cross site scripting attacks on
# MS IE and any other browser which fails to honor RFC2616.

##
##  printenv -- demo CGI program which just prints its environment
##
use strict;
use warnings;

print "Content-type: text/plain; charset=iso-8859-1\n\n";
foreach my $var (sort(keys(%ENV))) {
    my $val = $ENV{$var};
    $val =~ s|\n|\\n|g;
    $val =~ s|"|\\"|g;
    print "${var}=\"${val}\"\n";
}
```

*https://localhost:9000/cgi-bin/test-cgi?/**

Another testing script of Apache server.



```
#

# To permit this cgi, replace # on the first line above with the
# appropriate #!/path/to/sh shebang, and set this script executable
# with chmod 755.
#
# ***** !!! WARNING !!! *****
# This script echoes the server environment variables and therefore
# leaks information - so NEVER use it in a live server environment!
# It is provided only for testing purpose.
# Also note that it is subject to cross site scripting attacks on
# MS IE and any other browser which fails to honor RFC2616.

# disable filename globbing
set -f

echo "Content-type: text/plain; charset=iso-8859-1"
echo

echo CGI/1.0 test script report:
echo

echo argc is $#. argv is "$*".
echo

echo SERVER_SOFTWARE = $SERVER_SOFTWARE
echo SERVER_NAME = $SERVER_NAME
echo GATEWAY_INTERFACE = $GATEWAY_INTERFACE
echo SERVER_PROTOCOL = $SERVER_PROTOCOL
echo SERVER_PORT = $SERVER_PORT
echo REQUEST_METHOD = $REQUEST_METHOD
echo HTTP_ACCEPT = "$HTTP_ACCEPT"
echo PATH_INFO = "$PATH_INFO"
echo PATH_TRANSLATED = "$PATH_TRANSLATED"
echo SCRIPT_NAME = "$SCRIPT_NAME"
echo QUERY_STRING = "$QUERY_STRING"
echo REMOTE_HOST = $REMOTE_HOST
echo REMOTE_ADDR = $REMOTE_ADDR
echo REMOTE_USER = $REMOTE_USER
echo AUTH_TYPE = $AUTH_TYPE
echo CONTENT_TYPE = $CONTENT_TYPE
echo CONTENT_LENGTH = $CONTENT_LENGTH
```

https://localhost:9000/admin/login
Django admin login page

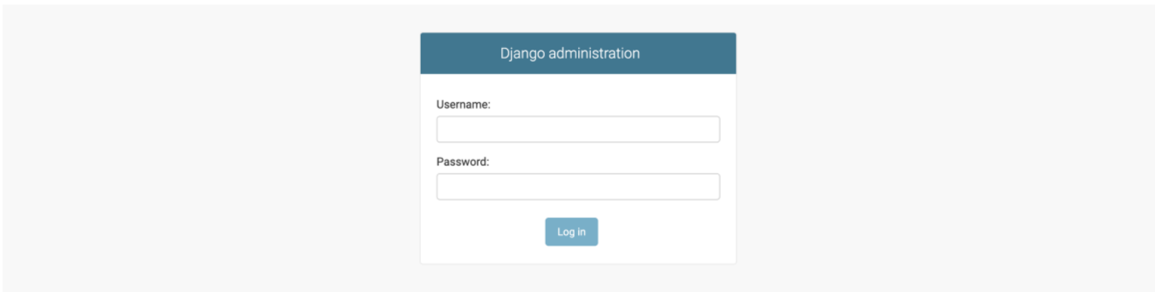


Figure 36. Django admin login page.

User login

We performed fuzzing of the user login procedure. We registered a user: test@test.com with a passphrase that can found in a word list of SecLists git repository and considered to be unsafe.

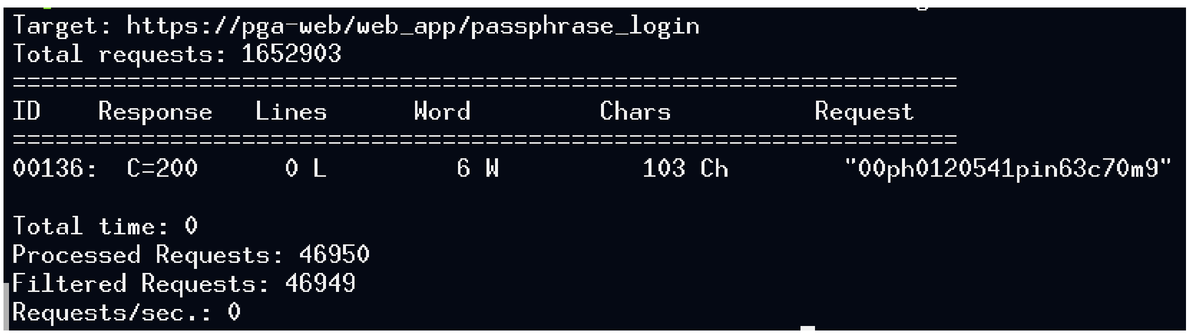
The login procedure consists of three steps:

1. Load login page and token cookie
2. Send username along with the token cookie through a POST request and get session id cookie
3. Send username, passphrase and the other fixed parameters along with token and session id.

Therefore, we first processed the step 1 and 2 with curl commands, and then fuzzed the step 3 using the token and session id cookies.

```
#Step 1
curl -k https://localhost:9001/web_app/login_username/ -o login_username.html -c cookie.txt
token=$(grep token cookie.txt |cut -f 7)

 #Step 2
curl --data-urlencode username=test@test.com -k
https://localhost:9001/web_app/check_passphrase_set -b cookie.txt -c cookie.txt -e

https://localhost:9001/web_app/login_username -X POST -H "X-CSRFToken: $token"

session=$(grep sessionid cookie.txt |cut -f 7)

 #Step 3
wfuzz   -w   /wordlist/bt4-password.txt   --hc   405,403   --hh   104   -b   "csrftoken=$token;
sessionid=$session"   -H   "Referer:   https://localhost:9001/web_app/text_login.html"   -H   "X-
CSRFToken:                                    $token"                                    -d
"username=test%40test.com&passphrase=FUZZ&total_time_until_submit=13565&total_time_until_subm
it_since_page_load=13285&time_interaction_started=0&is_reset=false"
https://localhost:9001/web_app/passphrase_login
```

Commands for fuzzing the login procedure with a known username. As shown below, the passphrase is identified at the 136th candidate of the word list.



```
Target: https://pga-web/web_app/passphrase_login
Total requests: 1652903
==================================================================
ID    Response   Lines      Word         Chars          Request
==================================================================
00136:  C=200      0 L        6 W         103 Ch        "00ph0120541pin63c70m9"

Total time: 0
Processed Requests: 46950
Filtered Requests: 46949
Requests/sec.: 0
```

Figure 37. Output from wfuzz.

The server allowed unlimited number of attempts (in our case, approx. 47k attempts) to test passphrase within the same session, which seems a potential risk.

Above, we fuzzed the procedure with the known username, however, there is also a way to find registered usernames using the fuzzing. At the #Step 2, the server responds either the request succeeds or not as below.

```
{"success": false, "set": false, "errors": "User not found in the system"}

{"success": false, "set": false, "errors": "Bad request - Please provide a valid email"}

{"success": false}
```

Examples of server responses for the failure of #Step 2: Username

And therefore, it is also possible to search registered usernames by fuzzing with approx. 9M usernames across four wordlists from wfuzz and SecLists. If the username is found, then the associated password can be fuzzed as above with approximately 50M candidate passwords. And again, such cracking can be avoided by introducing the user locking system.

The login with a graphical password can be fuzzed around the different parameters for gestures in a similar manner.

We performed the fuzzing to discover vulnerable paths and crack a user login. We invite the developers and service providers who will use the FlexPass system to judge if the discovered paths introduce vulnerabilities. However, for the user login, we found a potential risk to be broken by fuzzing, and thus recommend introducing a locking function after a certain number of login attempt failures.

7 Implications

7.1 FlexPass Applicability in the Healthcare Domain

In this section we elaborate on the applicability of FlexPass in the broader healthcare domain and provide guidelines and prototypes that can serve as a basis for implementing an adaptation and personalization system based on the suggested authentication paradigm. **Figure 38** illustrates the envisioned FlexPass system within healthcare environments. At a first stage, an organization would need to identify mainstream spatial areas of the hospital, *i.e.*, areas that visited by the majority of individuals (medical staff, patients, relatives, visitors, etc.). Next, the spatial relevance of each mainstream area should be identified in order to create a neighborhood/relationship map among the diverse mainstream spatial areas identified, *e.g.*, the mainstream spatial area "reception hall" is related to the hospital's "cafeteria", hence, a relationship rule would be created connecting the two areas. Finally, the system administrator would need to prepare and upload relevant images depicting sceneries for each of the identified mainstream of the hospital. These images would then be processed through an adaptation and recommendation engine that would recommend best-fit images to end-users aiming to improve memorability and security of passwords. For doing so, the recommendation engine would also receive as input the end-users visitation record in order to extract the relevant experiences and visits the end-users had in specific mainstream spatial areas of the hospital. In this respect, FlexPass will leverage on the existing user authentication infrastructure that exists in the healthcare organization for retrieving the user's visitation record.

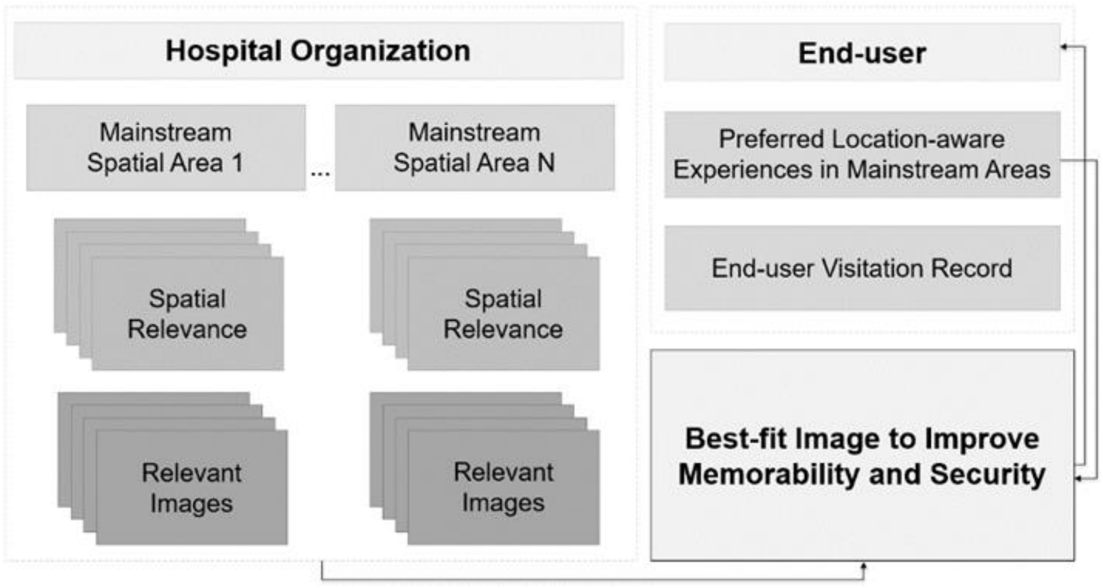

Hospital Organization
Mainstream Spatial Area 1
...
Mainstream Spatial Area N
Spatial Relevance
Spatial Relevance
Relevant Images
Relevant Images
End-user
Preferred Location-aware Experiences in Mainstream Areas
End-user Visitation Record
Best-fit Image to Improve Memorability and Security

Figure 38. The envisioned FlexPass system within healthcare environments.

The following scenarios are anticipated: *i) Enrolment Scenario:* During user enrolment, the system would retrieve (based on the username and a unique enrolment code) the user's visitation record within the hospital. Based on the semantic similarity of the user visits and the mainstream spatial areas of the hospital, the system recommends three relevant images to choose from for creating their graphical password. Note that the three images would have the same level of complexity and hotspots to avoid scenarios in which the user would create predictable passwords; *ii) Login Scenario:* During login, the system would illustrate two options for authentication (graphical *vs.* textual), and accordingly the user would enter their secret credentials to login; and *iii) Reset Scenario:* Password reset could be initiated either by the user (*e.g.*, in case they forget their password) or by the system based on the organization's applied policy. In this case, the same procedure would follow as in the enrolment scenario, considering however the previous image selections of the user, in order to avoid users selecting the same password.

7.2 FlexPass Personalization Workflow and Recommendation Rules

We envision that FlexPass may be deployed as a standalone user authentication system within healthcare organizations, which would consist of the following modules (**Figure 39**): *i)* the System Administration module; *ii)* the User Modeling module; *iii)* the Recommendation module; and *iv)* the Flexible User Authentication module. The System Administration module would allow administrators to upload and maintain images that depict sceneries of various locations of the hospital (*e.g.*, reception hall, main rooms of the hospital, garden, etc.). The system's image database would also be filled by end-users, would be able to upload their own images taken within the hospital, once approved by the system administrator by following internal policies and requirements of the organization. The User Modeling module would analyze the existing health record of the patients based on their activity and visits at the hospital (*e.g.*, patient may visit doctors of cardiology department, etc.). Based on the analysis, the module would infer the patient's frequent visits and important locations within the hospital, which would be then provided as input to the Recommendation module to recommend images depicting sceneries from the patient's most common visits. The Recommendation module would be further enhanced with image analysis technologies aiming to annotate the images semantically and automatically with the depicted content, which may be used during password creation for recommendation and user guidance for the creation of more memorable and secure passwords. Finally, the Flexible User Authentication module would be responsible for authenticating users based on an easy-to-use and a flexible authentication paradigm that would be based on the recommended and/or user-adaptable graphical passwords.

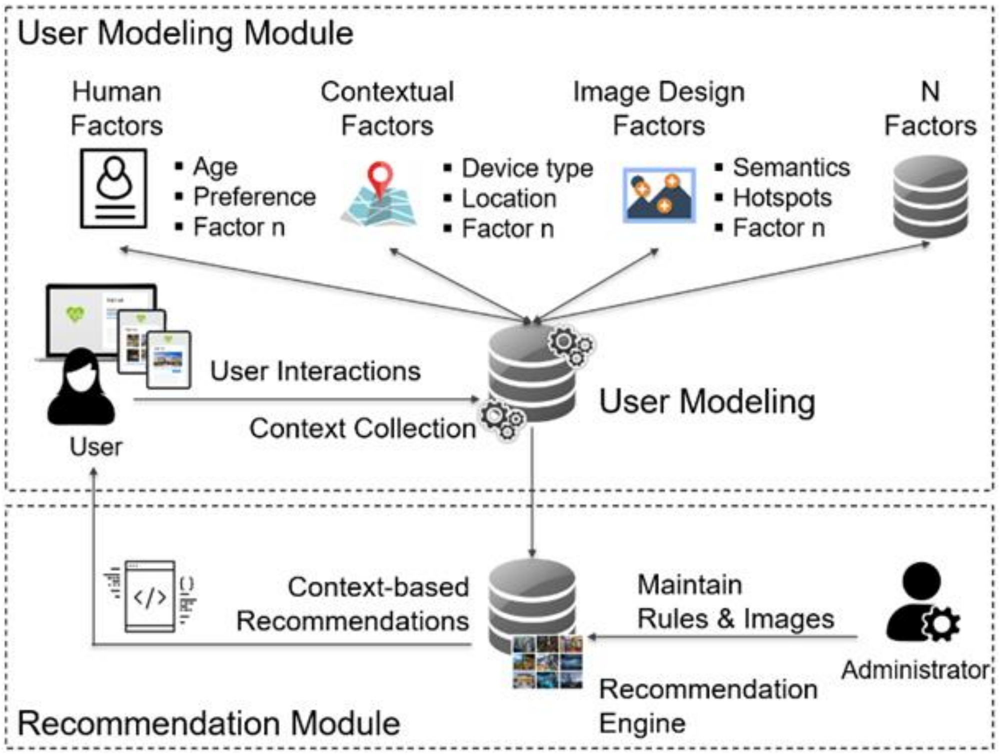


Figure 39. FlexPass personalization workflow – building on Fidas et al. (2015).

Algorithm #1 presents our content-based recommendation algorithm that will recommend relevant images during password creation/reset based on the rules of mainstream spatial areas and user's visitation records and experiences within the hospital.

Algorithm 1. Algorithm for image recommendation during password creation/reset.

```
Algorithm #1: Image recommendation during password creation/reset
Input: A set of user models that describe the users' frequent visits and important
locations at the hospital, filtered to contain relevant information based on the
relationship map between the mainstream areas um = {$um_1$, $um_2$, …, $um_m$} and a set of
candidate images that depict the mainstream spatial areas of the hospital, provided by the
system administrator and the end-users ci = {$ci_1$, $ci_2$, …, $ci_k$}.
Output: The top N images that are recommended to the end-user based on the semantic
similarity scores.
 procedure Mainstream_Map()
     ma = identify_mainstream_areas();
     mm = create_relationship_map(ma);
     return mm;
 end procedure
 procedure Candidate_Images()
     ci_set = upload_images();
     for i := 1 to k do begin
         $eit_i$ = explicit_image_tags(); # Annotated by system administrator
         $iit_i$ = implicit_image_tags(); # Annotated by computer vision techniques
         $ci_i$ = $eit_i$ U $iit_i$;
         $ci_i$ = clean_text($ci_i$);
         append_to_set(ci_set, $ci_i$);
     end for
     return ci_set;
 end procedure
 procedure Recommend_Images(mm, ci)
     for i := 1 to m do begin
         semantic_ranking = {};
         $fv_i$ = frequent_visits();
         $il_i$ = importan_locations();
         $fmm_i$ = filter_mainstream_map(mm, $fv_i$, $il_i$);
         $um_i$ = $fv_i$ U $il_i$ U $fmm_i$;
         $um_i$ = clean_text($um_i$);
         for j := 1 to k do begin
             $ss_{ij}$ = semantic_similarity($um_i$, $ci_j$); # through NLP techniques
             semantic_ranking[i][j] = $ss_{ij}$;
         end for
         sort_by_value(semantic_ranking);
         recommend_top_N($um_i$, semantic_ranking);
     end for
 end procedure
 mm = Mainstream_Map();
 ci = Candidate_Images();
 Recommend_Images(mm, ci);
```

8 Conclusions

The aim of this deliverable *D5.4. - "Report on Final User Authentication System"* is to report the outcome of the design, development, verification and evaluation of the final software of the user authentication scheme. This includes the final suggested authentication paradigm based on a novel retrospective approach in graphical passwords, the general architecture design, the development details of the credential hardening component, the sequence diagrams of use-case scenarios of the user authentication scheme, the design of the front-end prototypes of the final user authentication system, the results of the verification of the authentication properties, the results of the user evaluation of the user authentication system and the description of the core endpoints of the Application Programming Interface.

The outcome of this deliverable will be used as an essential input for other tasks and deliverables in Serums. Specifically, the *API and the underlying database* will be used as input in D2.6 for the final specifications and final software of the Smart Patient Health Records, and in D4.3 for the final data fabrication and semantic-preserving encryption. The authentication *architecture, APIs and database* will be used as an essential input in D6.3 for integrating the authentication system in the overall Serums' smart healthcare system software. The *user interface designs of the user authentication tasks* will be evaluated as part of D7.6 aiming to further evaluate the likeability aspects of FlexPass, its security and usability characteristics, the design of the user authentication system front-end, measure the users' acceptance, as well as the users' perceptions on aspects such as usability, memorability, security and trust. Finally, the outcome of D5.4 will be used as a basis for further elaborating ideas and areas for improvement for the user authentication system as part of the Serum technical roadmap.

Limitations of this research work are related to the fact that certain background images were used to control the factors of the user evaluation studies (as part of WP7's activities). Nevertheless, we provided the most commonly used image categories (representing landscapes, sceneries) and images of similar complexity. Aiming to increase external validity of this research work, we plan to explore a wider variety of image categories to triangulate findings. Furthermore, we stress that FlexPass is also susceptible to shoulder-surfing attacks, similar to the majority of graphical password systems (Tari et al., 2006; Chiasson et al., 2007). To minimize the threats of shoulder-surfing attacks, expansion of this research will consider adopting fake cursors (De Luca et al., 2013), decoy techniques (Zakaria et al., 2011) during graphical password creation. Likewise, the aforementioned countermeasures could also be used in the case of stealth attacks, during which the adversary takes a video from a distance while the user enters their credentials (Yue et al., 2014).

In addition, future work includes investigating the impact of other intrinsic human factors (*e.g.*, emotional parameters, cognitive styles, etc.) in personalized interactive graphical user authentication schemes (Fidas et al., 2021). Considering that emotions correlate with long-term memory (Tyng et al., 2017), and that events associated with emotions are more likely to be remembered (Christianson, 1992), future work entails investigating whether differences exist in individuals' emotions triggered when utilizing personalized images in FlexPass and when utilizing non-personalized images in other graphical user authentication schemes.

Furthermore, FlexPass requires users to remember only one password at a time, similar to the majority of graphical user authentication schemes. Hence, there is limited knowledge regarding memory interference (Biddle et al., 2012), which refers to *"the impaired ability to remember an item when it is similar to other items stored in memory"* (Anderson and Neely, 1996). Future research entails investigating whether memory interference occurs in cases in which individuals have created graphical passwords on similar images across multiple accounts in FlexPass.

Finally, evidence suggests that individuals tend to reuse the same or similar passwords across multiple accounts, aiming to reduce the memory load of remembering multiple passwords (Biddle et al., 2012). Such approaches have a negative impact on the security. Hence, another future research direction entails investigating whether differences exist in password reuse between individuals who utilize personalized images in FlexPass and individuals who utilize non-personalized/generic images in other graphical user authentication schemes.

We anticipate that the suggested approach will have a positive impact on both healthcare organizations and end-users. From the organization's perspective, the flexible approach will assist healthcare organizations to easily adjust their policies to the varying roles of their end-users (patients, doctors, administrators), in which current practice indicates that the "one-size-fits-all" approach is not adequate in the highly dynamic and heterogenous contexts of use in the healthcare domain. From the end-user's perspective, the suggested flexible and personalized paradigm and supported results open new directions for considering novel knowledge-based user authentication mechanisms to assist end-users to choose the "best-fit" authentication scheme depending on preference, unique characteristics and the context of interaction (*e.g.*, interaction in the office, in the emergency room, off the network, and so on).

Within nowadays information era, patients and medical staff interact in highly dynamic healthcare environments and contexts, and tend to use multiple devices to authenticate themselves, it is obvious that the current widely deployed "one-size-fits-all" text-based authentication paradigm might soon become obsolete. Hence, we believe that approaches like FlexPass provide an alternative solution to current state-of-the-art research and practice, and have the potential to be easily adopted with a rather inexpensive solution compared to other token-based (*e.g.*, smartcards) and biometric-based (*e.g.*, fingerprint) solutions, which necessitate increased implementation effort and maintenance costs.

References

Abadi, M., Barham, P., Chen, J. et al. 2016. Tensorflow: A system for large-scale machine learning. In 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), 265-283.

Anderson, M. C., Neely, J. H. Interference and inhibition in memory retrieval. In Memory. Elsevier, 1996, pp. 237–313

Atkinson, R.C., Shiffrin, R.M. (1968). Human memory: a proposed system and its control processes. In: Spence, K.W., Spence, J.T. (eds.), The psychology of learning and motivation (Volume 2), Academic Press, 89-195

Baddeley, A. (1990). Human memory: theory and practice. Lawrence-Erlbaum, London

Basile, D., Giandomenico, F.D., Gnesi, S. Statistical Model Checking of an Energy-Saving Cyber-Physical System in the Railway Domain. In Proceedings of the Symposium on Applied Computing, 2017

Belk, M., Fidas, C., Germanakos, P. and Samaras, G., 2017. The interplay between humans, technology and user authentication: A cognitive processing perspective. Computers in Human Behavior, 76, pp. 184-200

Belk, M., Fidas, C., Pitsillides, A. (2019). Flexpass: Symbiosis of seamless user authentication schemes in IoT. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI 2019)*, ACM Press, 2019

Biddle, R., Chiasson, S., van Oorschot, P. (2012). Graphical passwords: Learning from the first twelve years. ACM Computing Surveys, 44(4), 41 pages

Biddle, R., Chiasson, S., Van Oorschot, P. C. Graphical passwords: Learning from the first twelve years. ACM Computing Surveys (CSUR) 44, 4 (2012), 1–41

Burr, W.E., Dodson, D.F., Polk, W.T. (2006). Electronic authentication guideline. National Institute of Standards and Technology, Technical report

Chen, C., Cui, B., Ma, J., Wu, R., Guo, J., & Liu, W. (2018). A systematic review of fuzzing techniques. Computers and Security, 75, 118–13

Chiasson, S., Van Orschot, P. C., Biddle, R. Graphical password authentication using cued click points. In European Symposium on Research in Computer Security (2007), Springer, 359–374.

Christianson, S.-A. Emotional stress and eyewitness memory: a critical review. Psychological bulletin 112, 2 (1992), 284

Constantinides, A., Belk, M., Fidas, C., Beumers, R., Vidal, D., Huang, W., Bowles, J., Webber, T., Silvina, T., Pitsillides, A. (2021). Security and Usability of a Personalized User Authentication Paradigm: Insights from a Longitudinal Study with Three Healthcare Organizations. ACM Transactions on Computing for Healthcare *(submitted; major revision; second round of reviews)*

Constantinides, A., Belk, M., Fidas, C., Pitsillides, A. (2020). Design and Development of a Patient-centric User Authentication System. In *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization (Adjunct UMAP 2020)*, 201-203

Constantinides, A., Belk, M., Fidas, C., Pitsillides, A. 2019. On the accuracy of eye gaze-driven classifiers for predicting image content familiarity in graphical passwords. In *Proceedings of the ACM Conference on User Modeling, Adaptation and Personalization (UMAP 2019)*. ACM Press, 201-205

Constantinides, A., Belk, M., Fidas, C., Pitsillides, A. 2020. An eye gaze-driven metric for estimating the strength of graphical passwords based on image hotspots. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI 2020)*, ACM Press, 33–37

Constantinides, A., Fidas, C., Belk, M., Pietron, A., Han, T., Pitsillides, A. (2021). From hot-spots towards experience-spots: Leveraging on users' sociocultural experiences to enhance security in cued-recall graphical authentication, International Journal of Human-Computer Studies, 149

Constantinides, A., Fidas, C., Belk, M., Pitsillides, A. 2019. "I Recall this Picture": Understanding Picture Password Selections based on Users' Sociocultural Experiences. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI 2019)*, ACM Press, 408-412

Constantinides, A., Pietron, A., Belk, M., Fidas, C., Han, T., Pitsillides, A. 2020. A Cross-cultural Perspective for Personalizing Picture Passwords. In *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization (UMAP 2020)*, ACM Press, 43-52

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B. 2016. The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3213-3223.

Daniel Lowe Wheeler. 2016. Zxcvbn: low-budget password strength estimation. In Proceedings of the 25th USENIX Conference on Security Symposium (SEC'16). USENIX Association, USA, 157–173.

De Luca, A., Von Zezschwitz, E., Pichler, L., Hussmann, H. Using fake cursors to secure on-screen password entry. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (2013), 2399–2402.

DeMott, J. "The evolving art of fuzzing," in Proc. DEF CON Conf., vol. 14, 2006, pp. 1–25.

Eikey, E. V., Murphy, A.R., Reddy, M.C., Xu, H. (2015). Designing for privacy management in hospitals: Understanding the gap between user activities and IT staff's understandings. International Journal of Medical Informatics, 84(12), 1065-1075

Everingham, M., Van Gool, L., Williams, C., Winn, J., Zisserman, A. 2010. The pascal visual object classes (voc) challenge. International journal of Computer Vision, 88(2), 303-338.

Fidas, C., Belk, M., Portugal, D., Pitsillides, A. (2021). Privacy-preserving biometric-driven data for student identity management: Challenges and approaches. ACM User Modeling, Adaptation and Personalization (UMAP 2021 Adjunct), ACM Press, 368-370, doi: 10.1145/3450614.3464470

Fidas, C., Belk, M., Constantinides, C., Constantinides, A., Pitsillides, A. (2021). A field dependence-independence perspective on eye gaze behavior within affective activities. IFIP TC13 Human-Computer Interaction (INTERACT 2021), Springer-Verlag, 63-72, doi:10.1007/978-3-030-85623-6_6

Fidas, C., Hussmann, H., Belk, M., Samaras, G. (2015). iHIP: Towards a User Centric Individual Human Interaction Proof Framework. CHI Extended Abstracts 2015, ACM Press, 2235-2240

Gu, R., Enoiu, E., Seceleanu, C. TAMAA: UPPAAL-based mission planning for autonomous agents. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, 2020

Jääskelä, E. Genetic algorithm in code coverage guided fuzz testing, Dept. Comput. Sci., Univ. Oulu, 2016.

Johnson, J., Seixeiro, S., Pace, Z., van der Bogert, G., Gilmour, S., Siebens, L., Tubbs, K., Microsoft Corp (2014). Picture gesture authentication. U.S. Patent 8,650,636. Retrieved from https://google.com/patents/US8910253

Katsini, C., Fidas, C., Raptis, G.E., Belk, M., Samaras, G. and Avouris, N., 2018. Influences of human cognition and visual behavior on password strength during picture password composition. In Proceedings of the 2018 CHI conference on human factors in computing systems (pp. 1-14)

Komanduri, S., Shay, R., Kelley, P., Mazurek, M., Bauer, L., Christin, N., Cranor, L., Egelman, S. (2011). Of passwords and people: Measuring the effect of password-composition policies. ACM Conference on Human Factors in Computing Systems (CHI 2011), ACM Press, 2595-2604

Kuznetsova, A., Rom, H., Alldrin, N. et al. 2020. The open images dataset v4. International Journal of Computer Vision 128, 1956-1981. DOI: https://doi.org/10.1007/s11263-020-01316-z

Legay, A., Delahaye, B., Bensalem, S. Statistical model checking: An overview. In: International Conference on Runtime Verification. pp. 122–135. Springer (2010)

Leon, B., Boštjan, B. (2019). Rejecting the death of passwords: Advice for the future. Computer Science and Information Systems, 16(1), 313-332

Mare, S., Baker, M., Gummeson, J. (2016). A study of authentication in daily life. Symposium on Usable Privacy and Security (SOUPS 2016), USENIX Association, 189-206

Mason, J., Dave, R., Chatterjee, P., Graham-Allen, I., Esterline, A., Roy, K. (2020). An investigation of biometric authentication in the healthcare environment. Array 8, 100042

Noomene Ben Henda. 2014. Generic and efficient attacker models in SPIN. In Proceedings of the 2014 International SPIN Symposium on Model Checking of Software (SPIN 2014). ACM, New York, NY, USA, 77-86. Doi: http://dx.doi.org/10.1145/2632362.2632378

Paivio, A. (2006). Mind and its evolution: A dual coding theoretical approach. Lawrence-Erlbaum, Mahwah, NJ

Paszke, A., Gross, S., Massa, F. et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems 32, 8024-8035.

Serums Deliverable 5.1 - Initial Report on Security Metrics and Authentication Policies (2019). Deliverable of EU Horizon 2020 Grant 826278 *“Securing Medical Data in Smart Patient-Centric Healthcare Systems”*

Serums Deliverable 5.2 - Software on the Initial Verified User Authentication System (2020). Deliverable of EU Horizon 2020 Grant 826278 *“Securing Medical Data in Smart Patient-Centric Healthcare Systems”*

Serums Deliverable 5.3 - Software on the Refined Verified User Authentication Scheme (2021). Deliverable of EU Horizon 2020 Grant 826278 *“Securing Medical Data in Smart Patient-Centric Healthcare Systems”*

Squire, L (1992). Declarative and nondeclarative memory: Multiple brain systems supporting learning and memory. Journal of Cognitive Neuroscience, 4(3), 232-243

Sternberg, R.J. (2003). Cognitive theory. Thomson Wadsworth, Belmont, CA

Tari, F., Ozok, A. A., Holden, S. H. A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. In Proceedings of the second symposium on Usable privacy and security (2006), 56–66.

Tsung-Yi, L., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan D., Dollár, P., Zitnick, C.L. 2014. Microsoft coco: Common objects in context. In European Conference on Computer Vision. Springer, 740-755

Tulving, E. (2002). Episodic memory: From mind to brain. Annual Review of Psychology, 53, 1-25

Tyng, C. M., Amin, H. U., Saad, M. N., Malik, A. S. The influences of emotion on learning and memory. Frontiers in psychology 8 (2017), 1454

Uppaal (2022). http://www.uppaal.org

Van Rooij, O., Charalambous, M. A., Kaizer, D., Papaevripides, M., & Athanasopoulos, E. (2021). webFuzz: Grey-Box Fuzzing for Web Applications. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

Williams, H. L., Conway, M. A., Cohen, G. (2008). Autobiographical memory. In Cohen, G., Conway, M.A. (Eds.), Memory in the Real World (3rd ed.), 21-90, Hove, UK: Psychology Press

Yue, Q., Ling, Z., Fu, X., Liu, B., Ren, K., Zhao, W. Blind recognition of touched keys on mobile devices. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (2014), 1403–1414

Zakaria, N. H., Griffiths, D., Brostoff, S., Yan, J. Shoulder surfing defence for recall-based graphical passwords. In Proceedings of the seventh symposium on usable privacy and security (2011), pp. 1–12.

Zhao, Z., Ahn, G.J., Seo, J.J. and Hu, H., 2013. On the security of picture gesture authentication. In Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13) (pp. 383-398).

ABBREVIATIONS

2FA	Two-Factor Authentication
API	Application Programming Interface
CSS	Cascade Style Sheets
HMAC	Hash-based Message Authentication Code
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MAC	Message Authentication Code
PGA	Picture Gesture Authentication
POC	Proof-of-Concept
QR	Quick Response
RFID	Radio Frequency Identification
SHA-256	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TOTP	Time-based One-Time Password
UI	User Interface
UX	User Experience
WSGI	Web Server Gateway Interface

APPENDIX A – Contributions to Research Publications based on Activities within Work Package 5

Belk, M., Fidas, C., Katsi, E., Constantinides, A., Pitsillides, A. (2021). An empirical study of picture password composition on smartwatches. IFIP TC13 Human-Computer Interaction (INTERACT 2021), Springer-Verlag, 655-664, doi: 10.1007/978-3-030-85610-6_37

Constantinides, A., Fidas, C., Belk, M., Pitsillides, A. (2021). Understanding insider attacks in personalized picture password schemes. IFIP TC13 Human-Computer Interaction (INTERACT 2021), Springer-Verlag, 722-731, doi: 10.1007/978-3-030-85610-6_42

Fidas, C., Belk, M., Constantinides, C., Constantinides, A., Pitsillides, A. (2021). A field dependence-independence perspective on eye gaze behavior within affective activities. IFIP TC13 Human-Computer Interaction (INTERACT 2021), Springer-Verlag, 63-72, doi:10.1007/978-3-030-85623-6_6

Leonidou P., Constantinides A., Belk M., Fidas C., Pitsillides A. (2021). Eye gaze and interaction differences of holistic versus analytic users in image-recognition human interaction proof schemes. Human-Computer Interaction International (HCII 2021) – HCI for Cybersecurity, Privacy and Trust, LNCS, Springer-Verlag, 66-75, doi: 10.1007/978-3-030-77392-2_5

Constantinides, C., Constantinides, A., Belk, M., Fidas, C., Pitsillides, A. (2021). A comparative study among different computer vision algorithms for assisting users in picture password composition. ACM User Modeling, Adaptation and Personalization (UMAP 2021 Adjunct), ACM Press, 357-362, doi: 10.1145/3450614.3464474

Fidas, C., Belk, M., Portugal, D., Pitsillides, A. (2021). Privacy-preserving biometric-driven data for student identity management: Challenges and approaches. ACM User Modeling, Adaptation and Personalization (UMAP 2021 Adjunct), ACM Press, 368-370, doi: 10.1145/3450614.3464470

Constantinides, A., Fidas, C., Belk, M., Pietron, A., Han, T., Pitsillides, A. (2021). From hot-spots towards experience-spots: Leveraging on users' sociocultural experiences to enhance security in cued-recall graphical authentication. International Journal of Human-Computer Studies, 149, Elsevier. doi: 10.1016/j.ijhcs.2021.102602

Constantinides, A., Belk, M., Fidas, C., Pitsillides, A. (2020). Design and Development of a Patient-centric User Authentication System. Adaptive and Personalized Privacy and Security Workshop (APPS 2020), UMAP (Adjunct Publication).doi: 10.1145/3386392.3399564

Constantinides, A., Pietron, A., Belk, M., Fidas, C., Han, T., Pitsillides, A. (2020). A Cross-cultural Perspective for Personalizing Picture Passwords. ACM User Modeling, Adaptation and Personalization (UMAP 2020), ACM Press, 43-52, doi: 10.1145/3340631.3394859

Costi, A., Belk, M., Fidas, C., Constantinides, A., Pitsillides, A. (2020). CogniKit: An Extensible Tool for Human Cognitive Modeling based on Eye Gaze Analysis. ACM Intelligent User Interfaces (IUI Companion 2020), ACM Press, 130-131, doi: 10.1145/3379336.3381460

Constantinides, A., Belk, M., Fidas, C., Pitsillides, A. (2020). An eye gaze-driven metric for estimating the strength of graphical passwords based on image hotspots. ACM Intelligent User Interfaces (IUI 2020), ACM Press, 33-37, doi: 10.1145/3377325.3377537

Janjic, V., Bowles, J.K.F., Vermeulen, A. F., Silvina, A., Belk, M., Fidas, C., Pitsillides, A., Kumar, M., Rossborry, M., Vinov, M., Given-Wilson, T., Legay, A., Blackledge, E., Arredouani, R., Stylianou, G., Huang, W. (2019). The SERUMS tool-chain: ensuring security and privacy of medical data in smart patient-centric healthcare systems. (IEEE Big Data), Los Angeles, December 2019, IEEE Press. doi: 10.1109/BigData47090.2019.9005600

Fidas, C. (2019). Eye tracking based cognitive-centered user models. ACM Conference on Web Intelligence (WI 2019), ACM Press, 433-437. doi: 10.1145/3350546.3352563

Constantinides, A., Fidas, C., Belk, M., Pitsillides, A. (2019).“I Recall this Picture”: Understanding Picture Password Selections based on Users’ Sociocultural Experiences. ACM Web Intelligence (WI 2019), ACM Press, 408-412. doi: 10.1145/3350546.3352557

Fidas, C., Belk, M., .Hadjidemetriou,G., Pitsillides A. (2019). Influences of Mixed Reality and Human Cognition on Picture Passwords: An Eye Tracking Study Published by Springer Nature Switzerland AG 2019 D. Lamas et al. (Eds.): (INTERACT 2019), LNCS 11747, pp. 304–313, 2019. doi: 10.1007/978-3-030-29384-0_19

Diomedous, C., Athanasopoulos E. (2019). Practical Password Hardening Based on TLS.Springer Nature Switzerland AG 2019 R. Perdisci et al. (Eds.): (DIMVA 2019), LNCS 11543, pp. 441–460, 2019. doi: 10.1007/978-3-030-22038-9_21

Constantinides, A., Belk, M., Fidas, C., Pitsillides, A. (2019). On the accuracy of eye gaze-driven classifiers for predicting image content familiarity in graphical passwords. ACM User Modeling, Adaptation and Personalization (UMAP 2019), ACM Press, 201-205. doi: 10.1145/3320435.3320474

Janjic, V., Bowles, J.K.F., Belk, M., Pitsillides. A. (2019). Security And Privacy Of Medical Data: Challenges For Next-Generation Patient-Centric Healthcare Systems. (UMAP 2019) Adjunct: Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization June 2019 Pages 213–214. doi: 10.1145/3314183.3326364

Constantinides, A., Fidas, C., Belk, M., Pitsillides, A. (2019). On the Personalization of Image Content in Graphical Passwords based on Users’ Sociocultural Experiences: New Challenges and Opportunities. Adaptive and Personalized Privacy and Security Workshop (APPS 2019), UMAP (Adjunct Publication), 199-202. doi: 10.1145/3314183.3324966

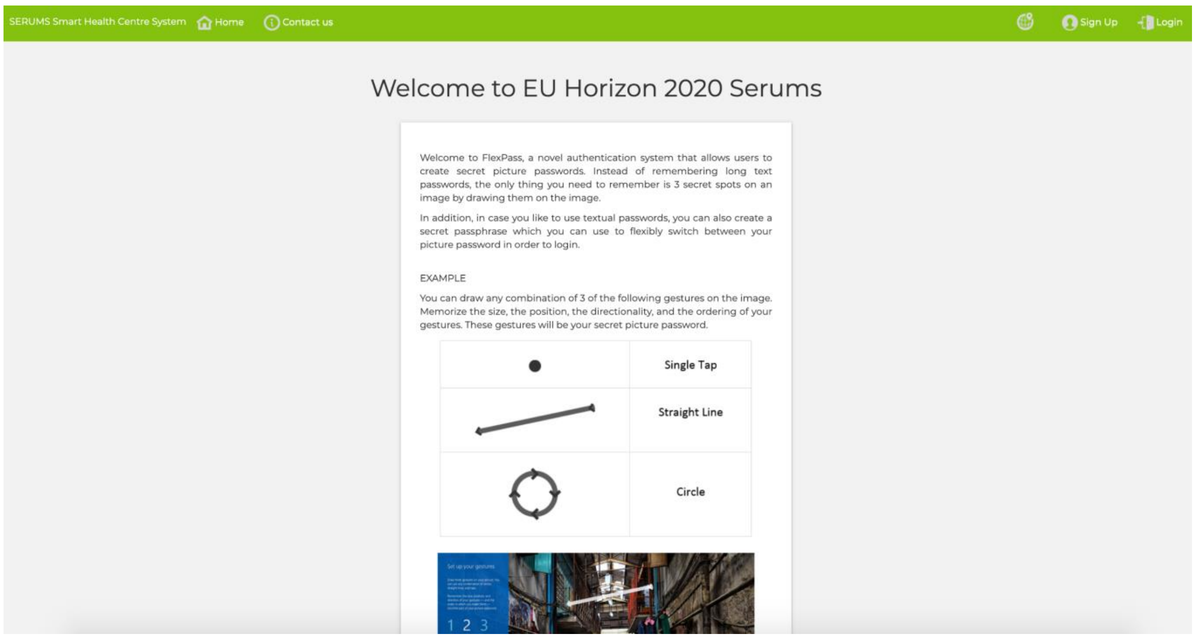
Belk, M., Fidas,C., Pitsillides. A. (2019). Flexpass: Symbiosis of seamless user authentication schemes in Iot. In Extended Abstracts of the (CHI 2019). doi: 10.1145/3290607.3312951

Hadjidemetriou, G., Belk, M., Fidas, C., Pitsillides, A. (2019). Picture passwords in mixed reality: Implementation and evaluation (2019). ACM SIGCHI Human Factors in Computing Systems (CHI 2019), ACM Press. doi: 10.1145/3290607.3313076

APPENDIX B – Prototype Design of the User Interfaces

In this section, we provide prototypes of the main User Interfaces (UI) of the final version of the user authentication system according to User Experience (UX) principles, heuristics and trends. Aiming to build an easy to use and usable user authentication system that can be deployed on heterogenous devices, fundamental UX principles were considered for the design of the UI interfaces. Focus will be given on using a simple language for communicating information and feedback to the end-users, avoiding technical terms. The UIs have been designed focusing on functional and hedonic aspects.

UI of the FlexPass Homepage and Demonstration Page



Welcome to EU Horizon 2020 Serums

Welcome to FlexPass, a novel authentication system that allows users to create secret picture passwords. Instead of remembering long text passwords, the only thing you need to remember is 3 secret spots on an image by drawing them on the image.

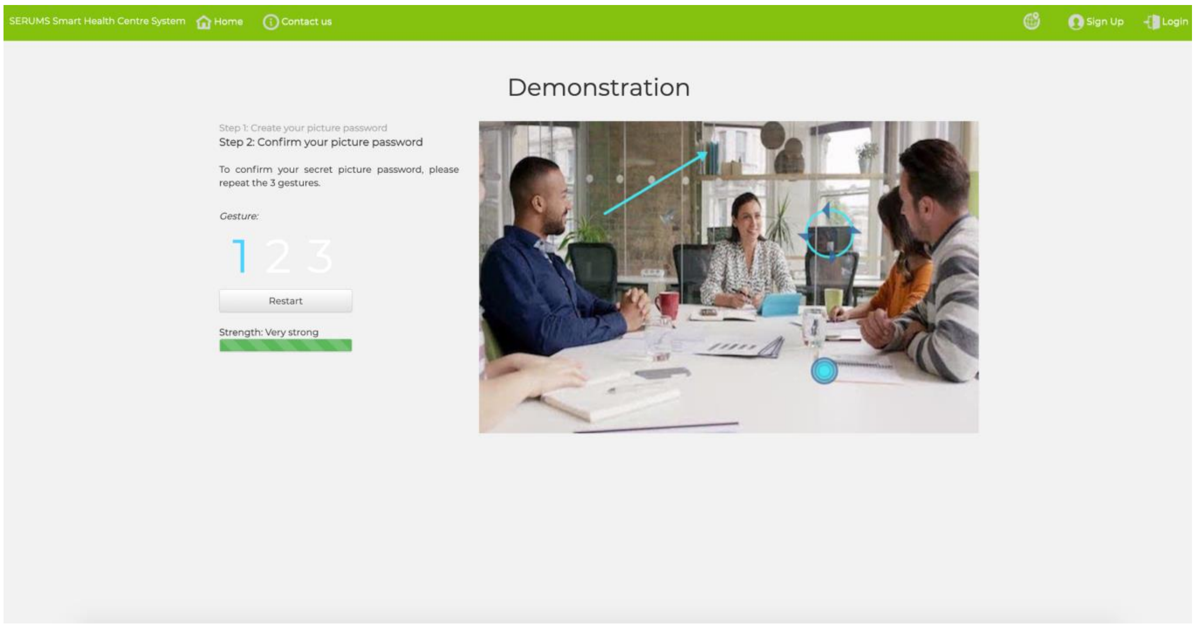
In addition, in case you like to use textual passwords, you can also create a secret passphrase which you can use to flexibly switch between your picture password in order to login.

EXAMPLE

You can draw any combination of 3 of the following gestures on the image. Memorize the size, the position, the directionality, and the ordering of your gestures. These gestures will be your secret picture password.

●	Single Tap
	Straight Line
	Circle

Figure 40. Homepage screen introducing the FlexPass paradigm.



Demonstration

Step 1: Create your picture password

Step 2: Confirm your picture password

To confirm your secret picture password, please repeat the 3 gestures.

Gesture:

1 2 3

Strength: Very strong

Figure 41. Demonstration page in which users can familiarize with the graphical password creation in the FlexPass system.

UI of the System Administrator's Page

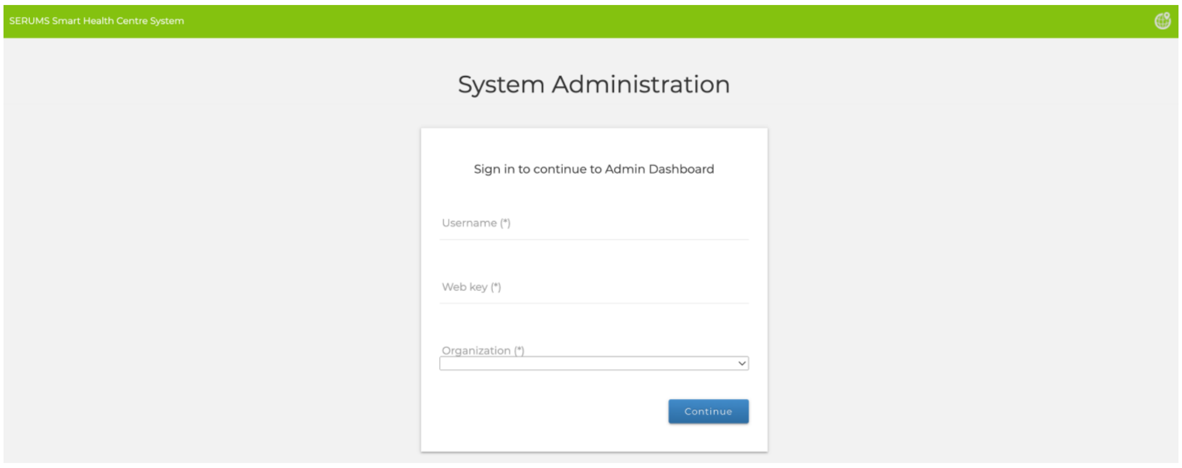

SERUMS Smart Health Centre System
System Administration
Sign in to continue to Admin Dashboard
Username (*)
Web key (*)
Organization (*)
Continue

Figure 42. System administrator's login page.

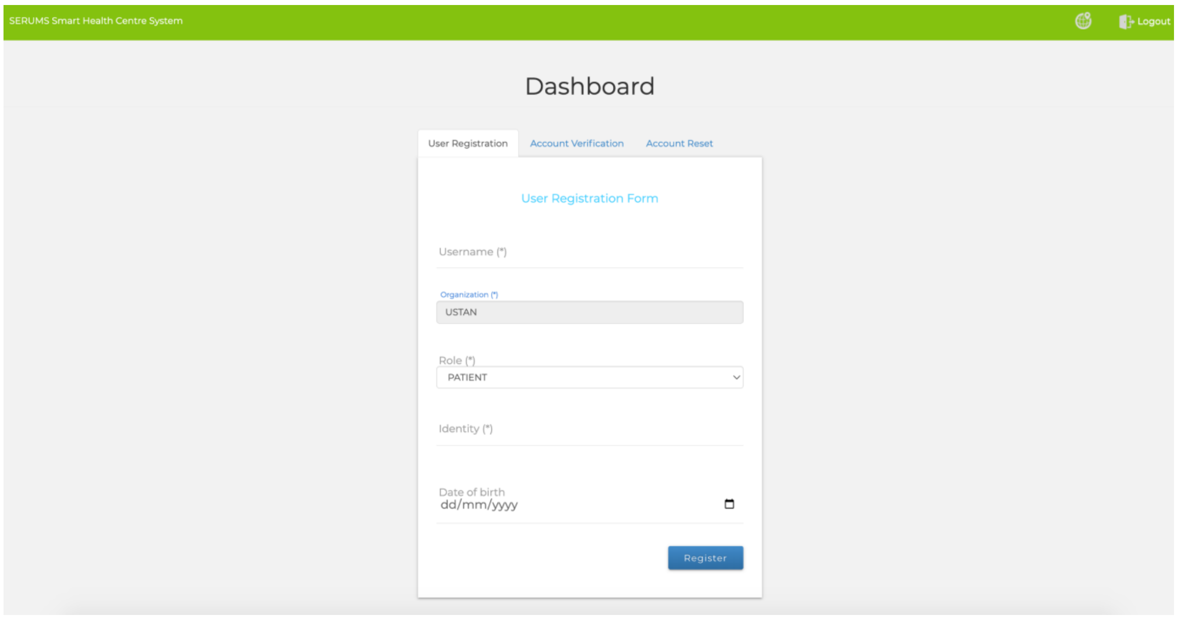


Figure 43. Administrator's user account creation page in which system administrators create new accounts for end-users of their organization and their corresponding role (*i.e.*, patient, medical staff, system administrator).

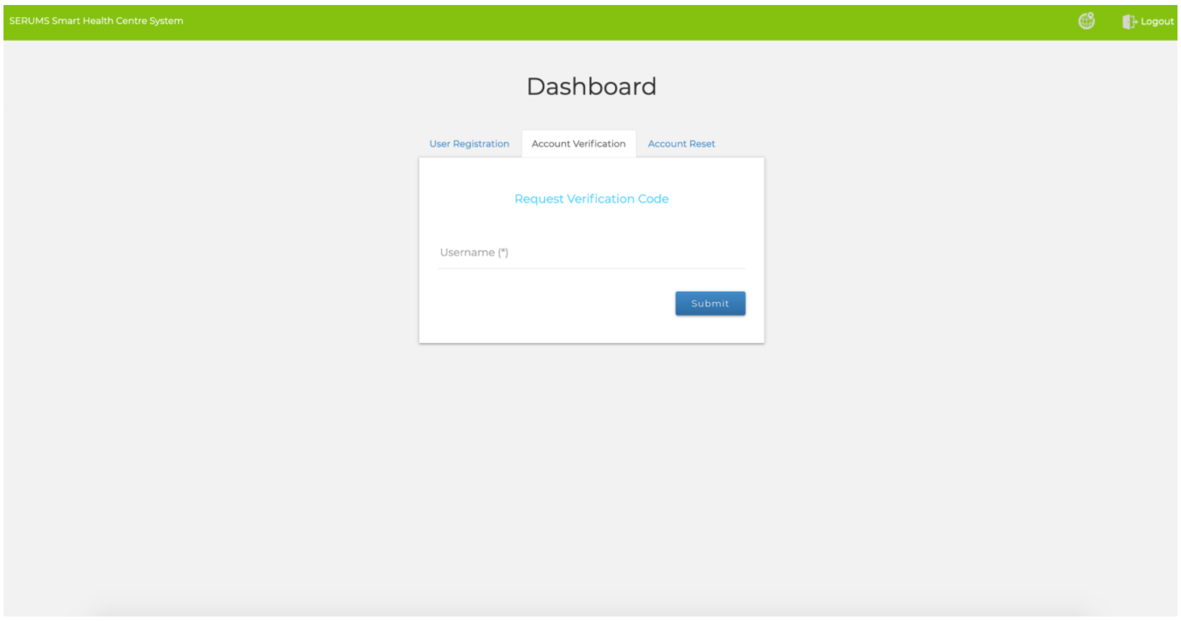


Figure 44. End-users can contact helpdesk and request from system administrators to get a new account verification code on their email. Accordingly, administrators can use the above UI to send an account verification code to a particular end-user's email.

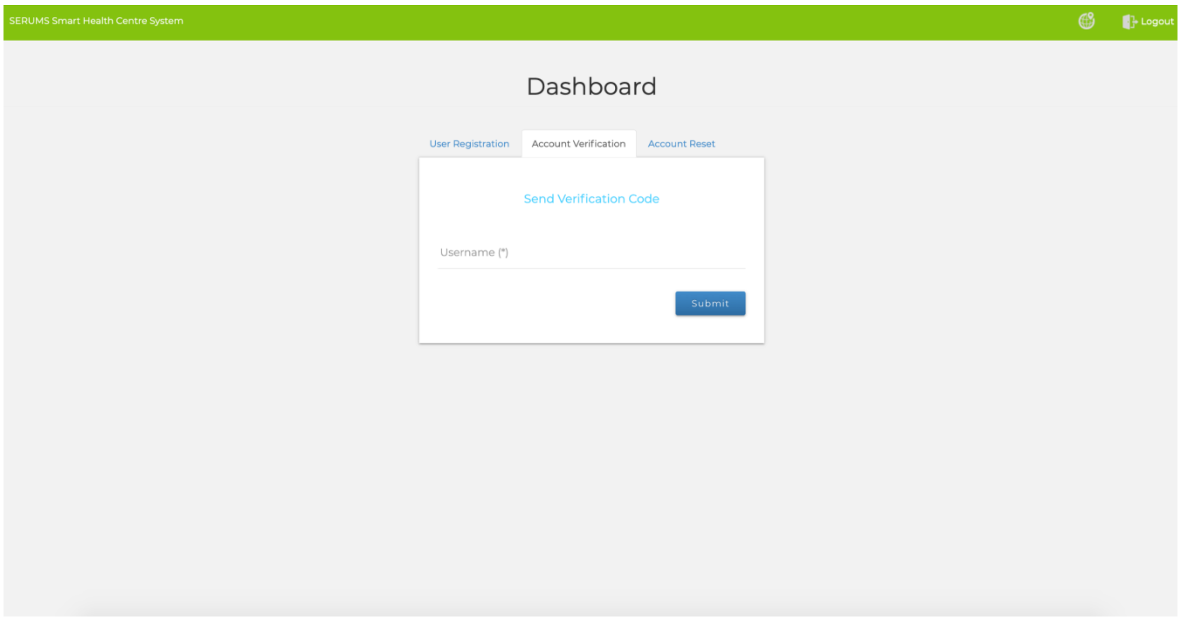
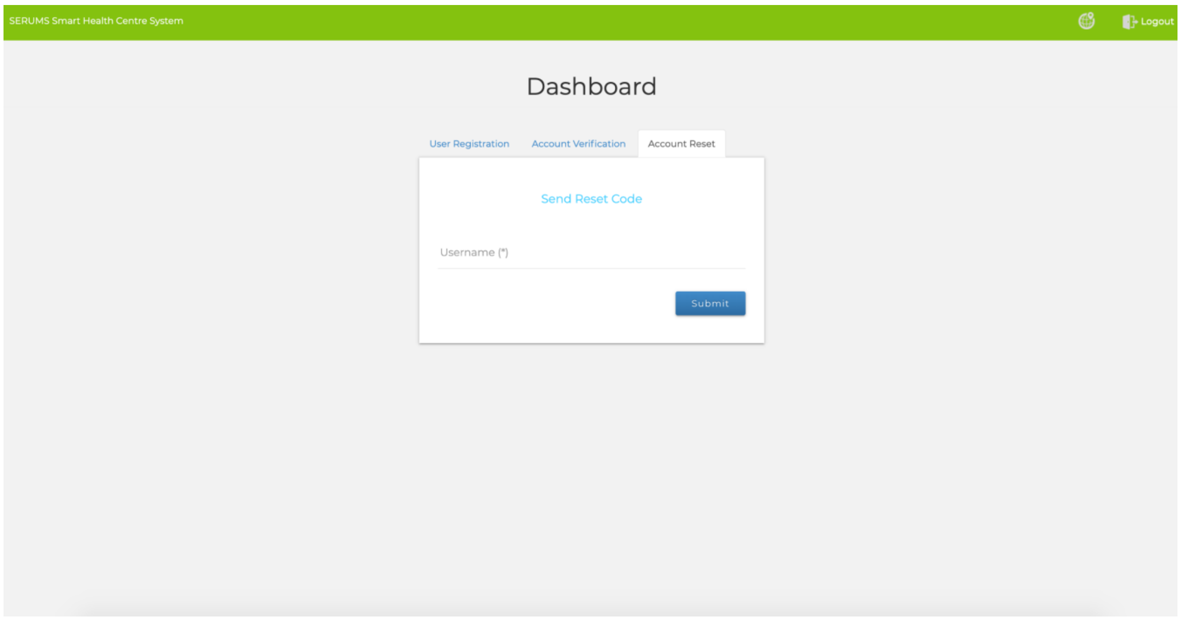


Figure 45. System administrators can send a verification code to an end-user's email. Accordingly, the end-user can use the code received in their email during account verification.


SERUMS Smart Health Centre System

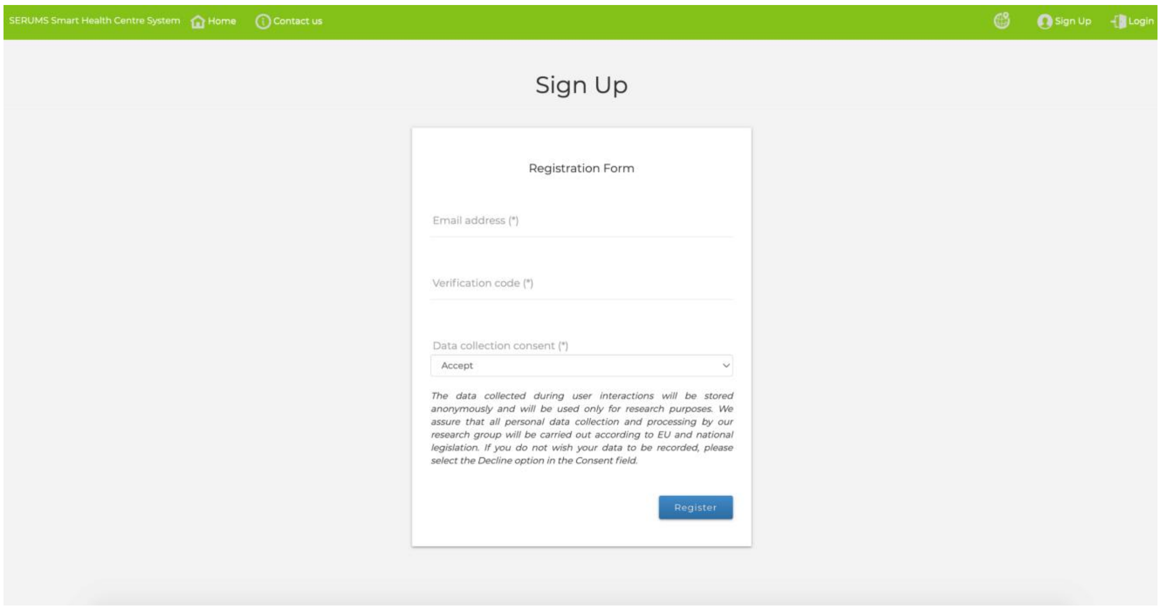
Dashboard

Send Reset Code

Username (*)

Figure 46. System administrators can send a reset code to an end-user's email. Accordingly, the end-user can use the code received in their email during reset password of their account.

UI of the User Account Registration Page


SERUMS Smart Health Centre System

Sign Up

Registration Form

Email address (*)

Verification code (*)

Data collection consent (*)

Accept

The data collected during user interactions will be stored anonymously and will be used only for research purposes. We assure that all personal data collection and processing by our research group will be carried out according to EU and national legislation. If you do not wish your data to be recorded, please select the Decline option in the Consent field.

Figure 47. User account registration page. Once a user account has been created by the system administrator, an email is sent to the end-user along with an activation page in which the user is redirected to start creating his/her password.

UI of the Graphical Password Creation Page

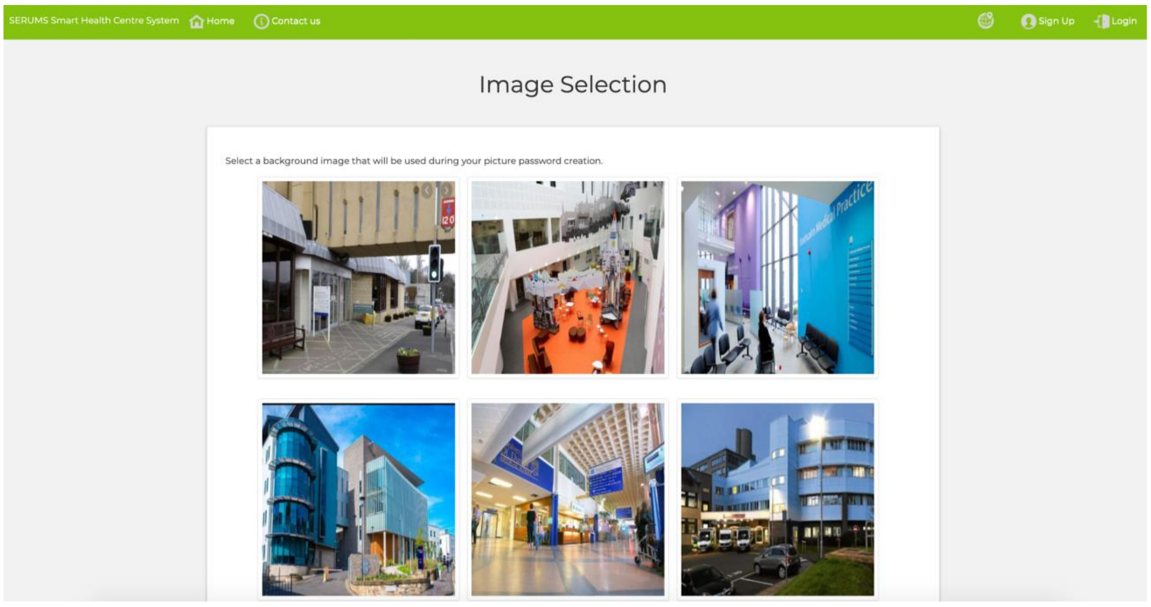
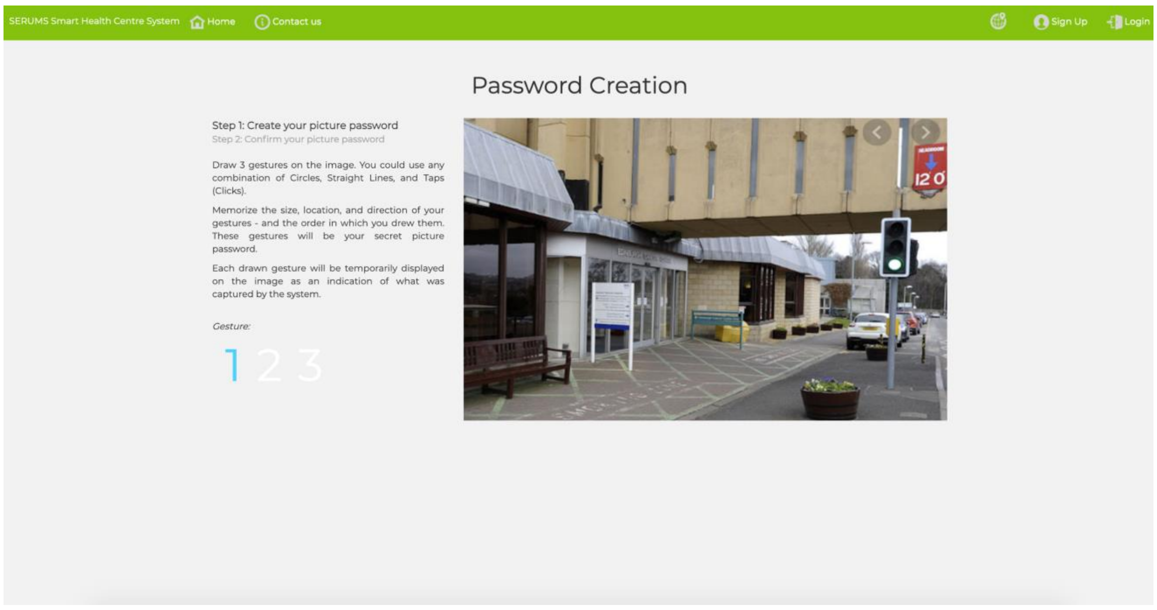


Image Selection

Select a background image that will be used during your picture password creation.

Figure 48. Image selection for graphical password creation. This page illustrates a set of images illustrating content that is highly relevant to the users' everyday activities and experiences within their healthcare environments. End-users select their preferred image, which is used to create their graphical password.



Password Creation

Step 1: Create your picture password
Step 2: Confirm your picture password

Draw 3 gestures on the image. You could use any combination of Circles, Straight Lines, and Taps (Clicks).

Memorize the size, location, and direction of your gestures - and the order in which you drew them. These gestures will be your secret picture password.

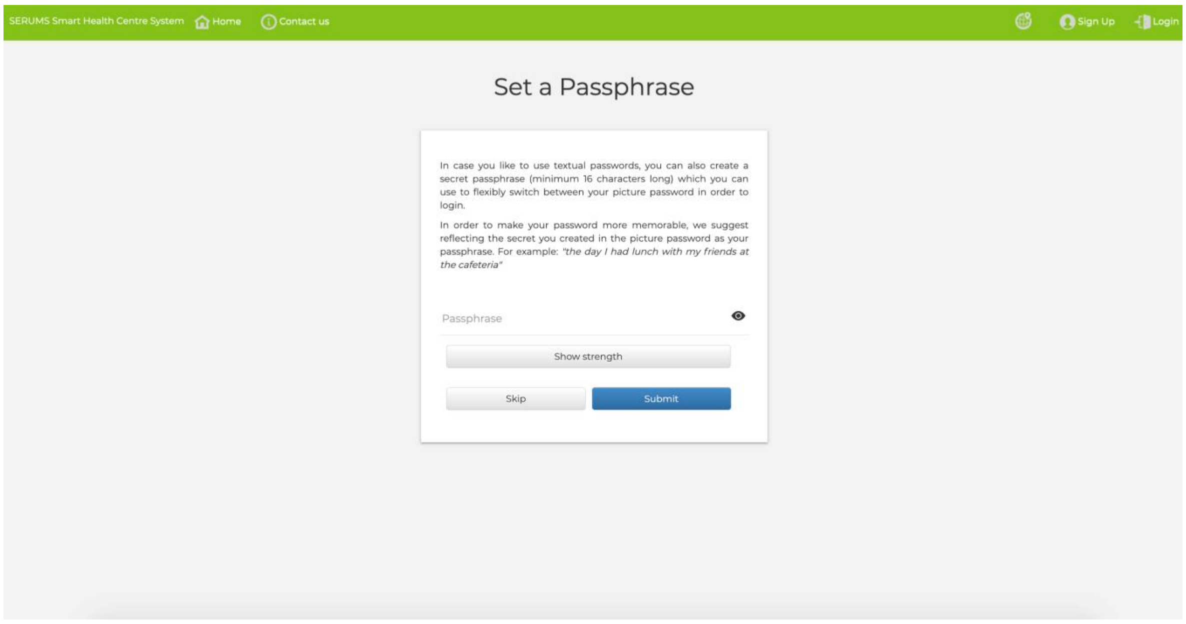
Each drawn gesture will be temporarily displayed on the image as an indication of what was captured by the system.

Gesture:

1 2 3

Figure 49. Graphical password creation. End-users create their graphical password by creating a set of secret gestures on the image (gestures can be a combination of tabs, lines and circles).

UI of the Textual Password Creation Page



Set a Passphrase

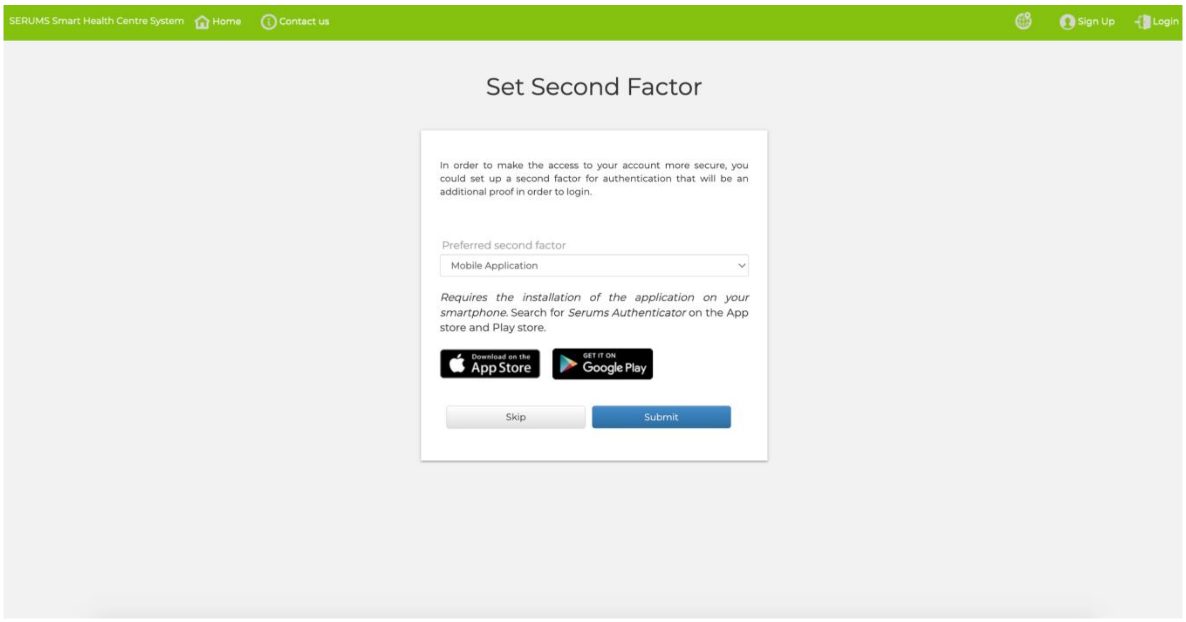
In case you like to use textual passwords, you can also create a secret passphrase (minimum 16 characters long) which you can use to flexibly switch between your picture password in order to login.

In order to make your password more memorable, we suggest reflecting the secret you created in the picture password as your passphrase. For example: *"the day I had lunch with my friends at the cafeteria"*

Passphrase

Figure 50. Passphrase creation page. End-users can optionally create a textual passphrase as an alternative type for authentication by reflecting their secret used in the graphical password creation, which can then be used to switch between types of passwords (graphical *vs.* textual) during login.

UI of the Two-Factor Authentication Activation Page



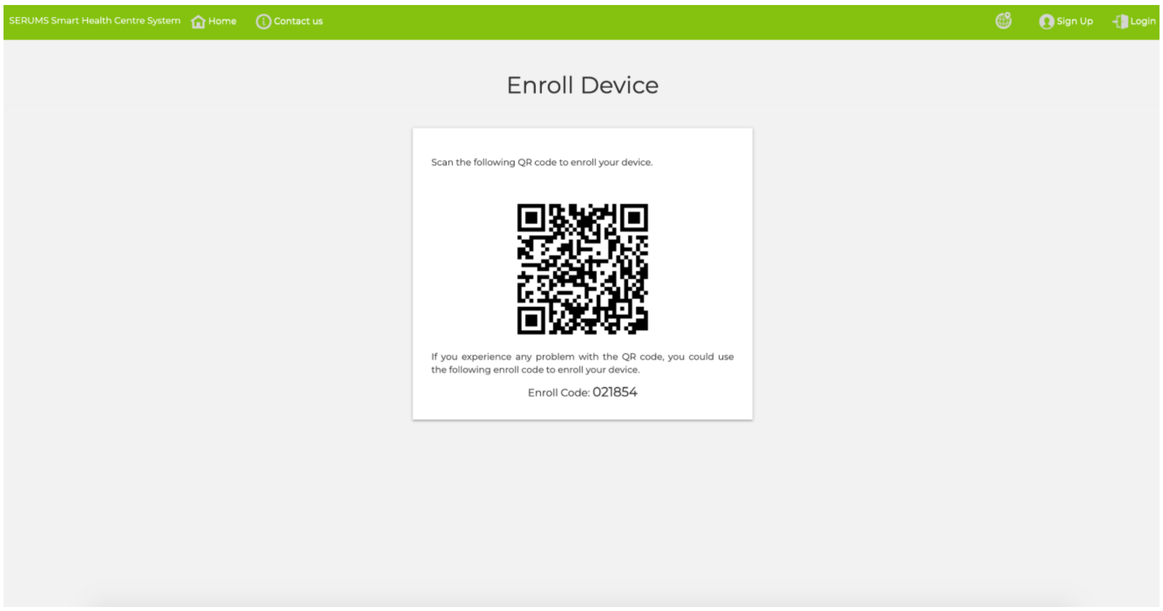
Set Second Factor

In order to make the access to your account more secure, you could set up a second factor for authentication that will be an additional proof in order to login.

Preferred second factor: Mobile Application

Requires the installation of the application on your smartphone. Search for *Serums Authenticator* on the App store and Play store.

Figure 51. Two-factor authentication activation page. In order to add an additional factor for authentication, end-users can setup two-factor authentication by downloading and installing a mobile application on their smartphone that has been developed for this purpose. The smartphone's mobile application can then be used as a second factor for authentication during login.



Enroll Device

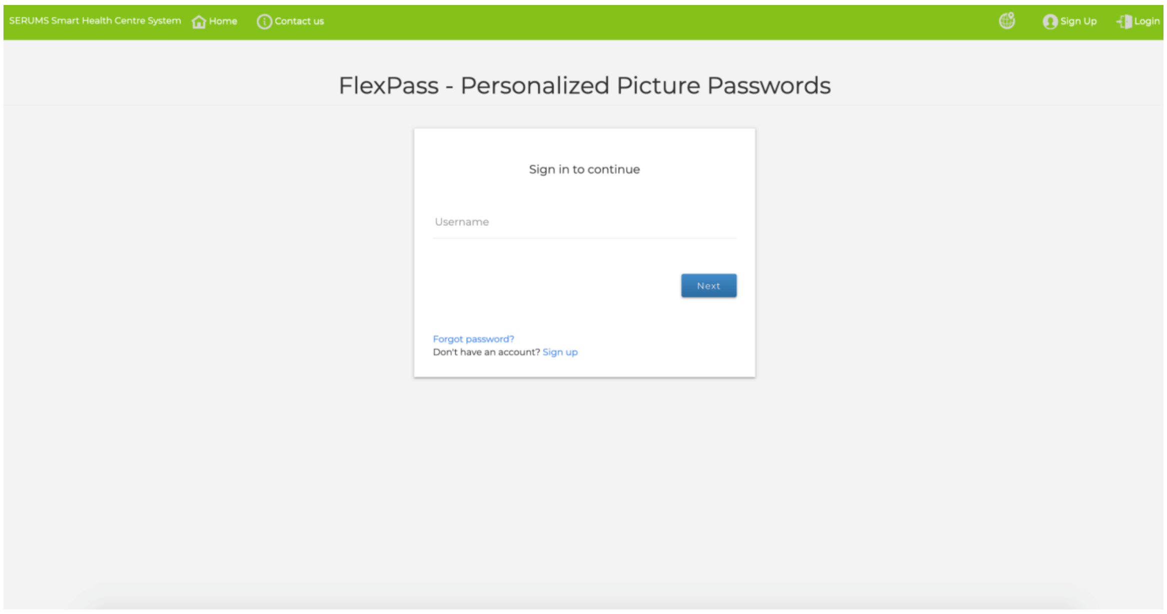
Scan the following QR code to enroll your device.

If you experience any problem with the QR code, you could use the following enroll code to enroll your device.

Enroll Code: 021854

Figure 52. QR code for enrolling the user's smartphone device for two-factor authentication.

UI of the User Login Page



FlexPass - Personalized Picture Passwords

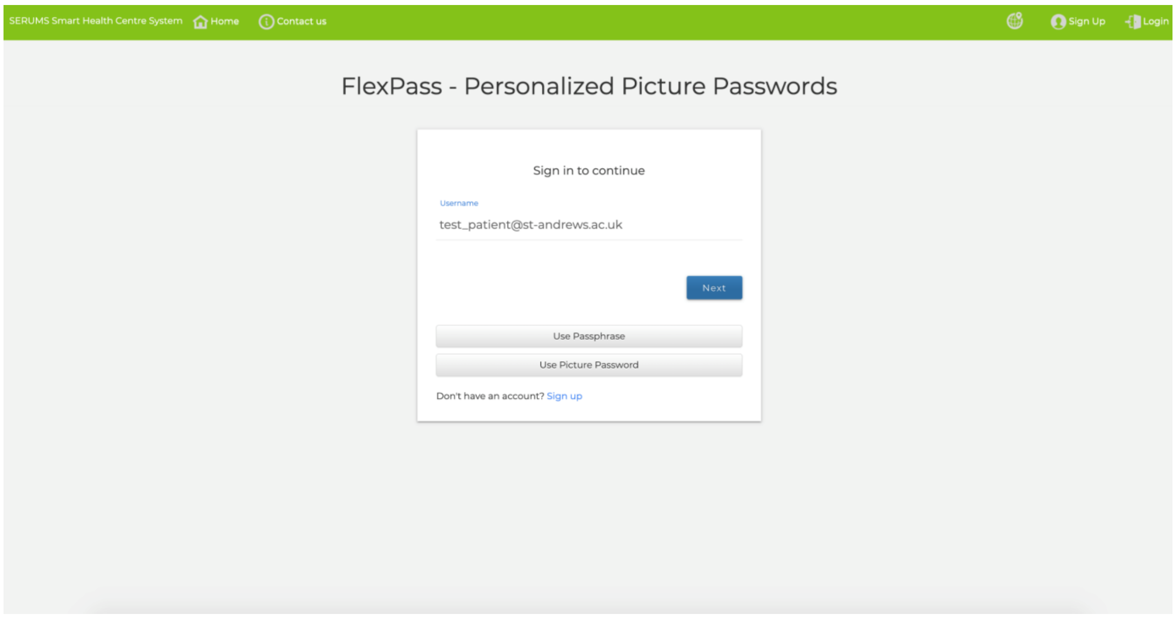
Sign in to continue

Username

Next

Forgot password?
Don't have an account? Sign up

Figure 53. Sign in page based on the end-user's username.


SERUMS Smart Health Centre System Home Contact us Sign Up Login

FlexPass - Personalized Picture Passwords

Sign in to continue

Username

test_patient@st-andrews.ac.uk

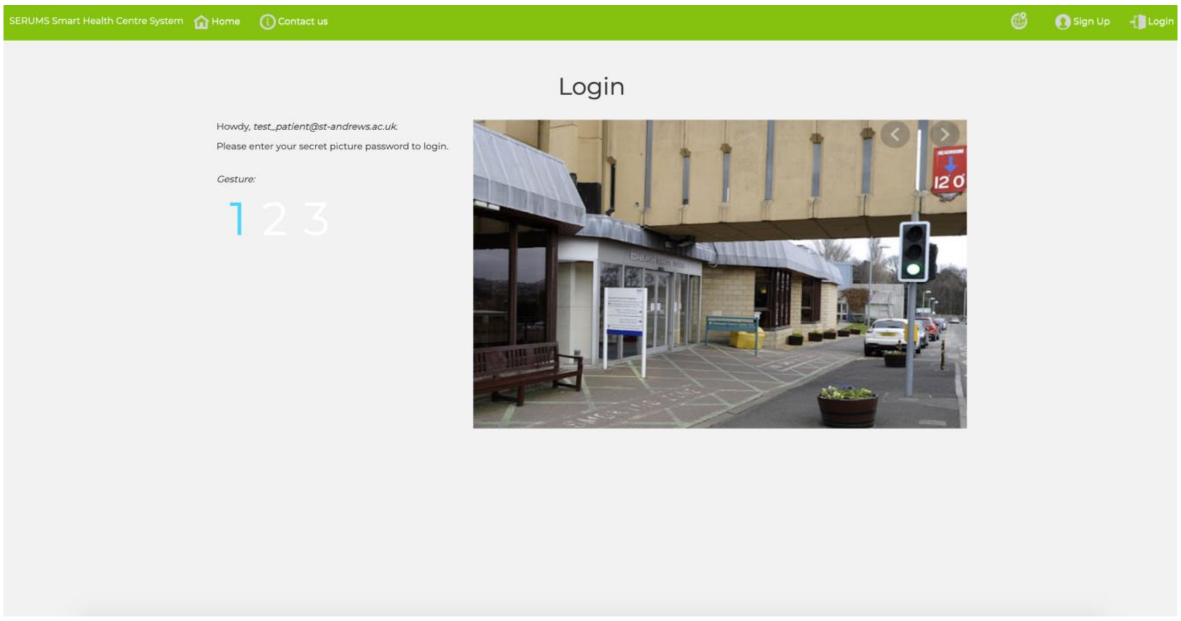
Next

Use Passphrase

Use Picture Password

Don't have an account? Sign up

Figure 54. Sign in page in which the users select their preferred authentication type (graphical or textual).


SERUMS Smart Health Centre System Home Contact us Sign Up Login

Login

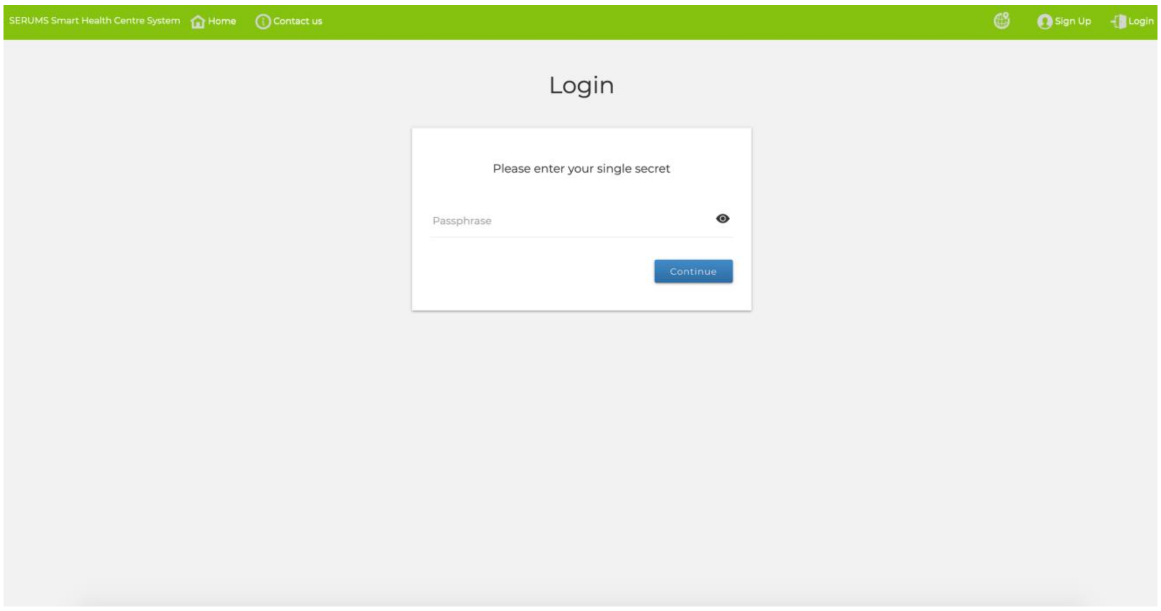
Howdy, *test_patient@st-andrews.ac.uk*

Please enter your secret picture password to login.

Gesture:

1 2 3

Figure 55. User graphical password login page. End-users enter their graphical password by creating gestures on the image that were setup during the graphical password creation phase.

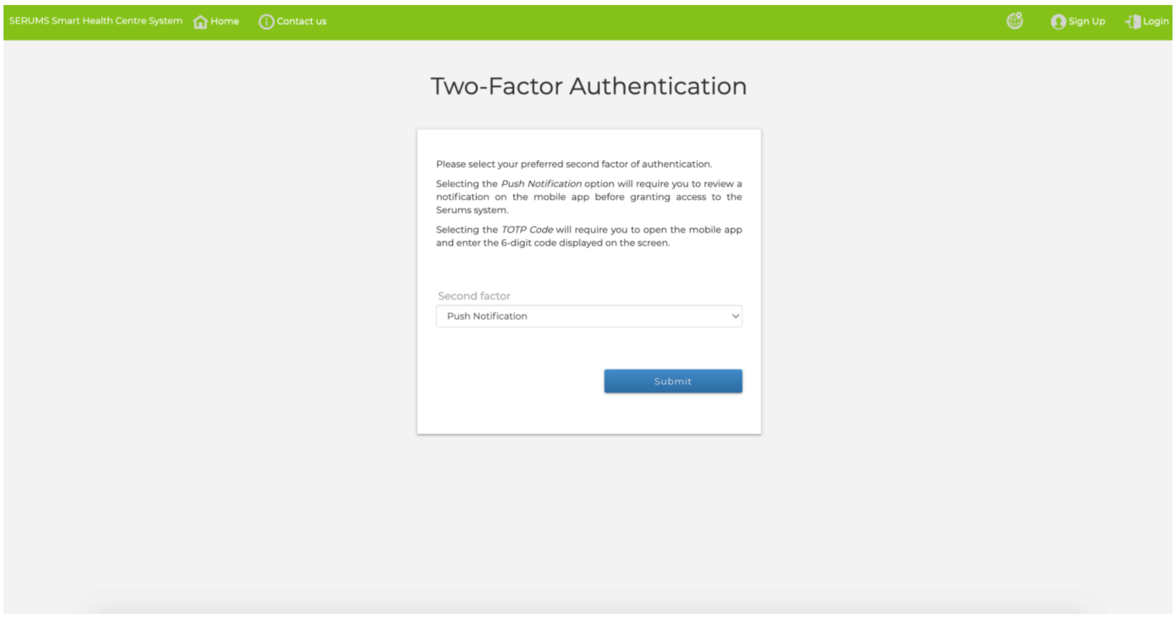


Please enter your single secret

Passphrase

Figure 56. User textual password login page for end-users that have selected the textual password type to login.

UI of the Two-Factor Authentication Login Page



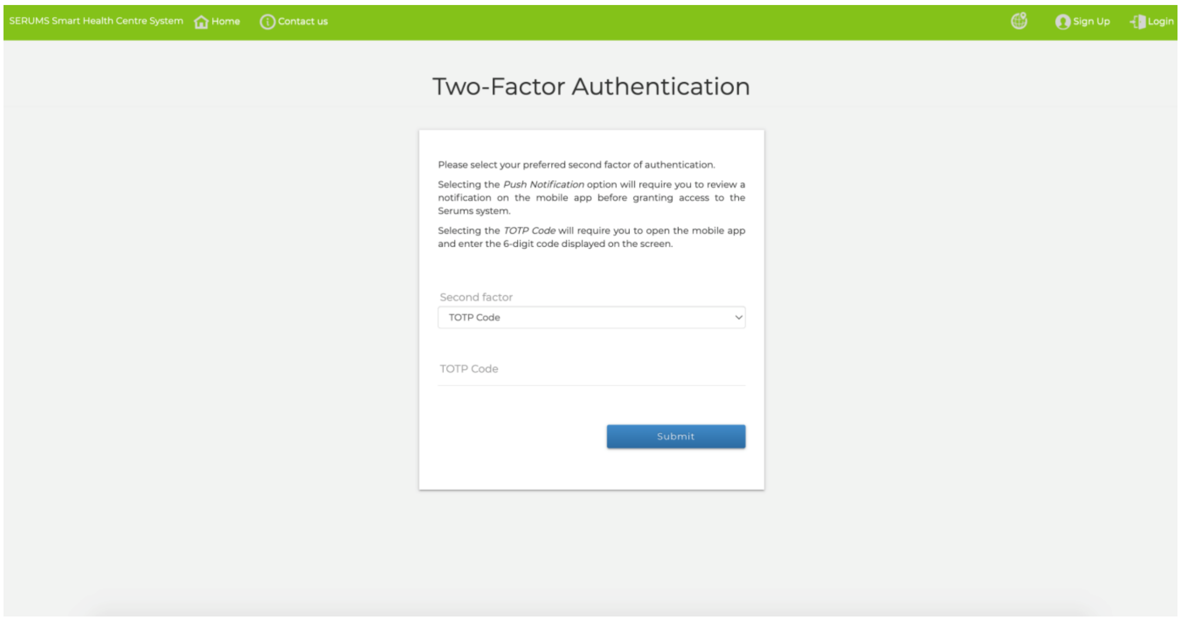
Please select your preferred second factor of authentication.

Selecting the *Push Notification* option will require you to review a notification on the mobile app before granting access to the Serums system.

Selecting the *TOTP Code* will require you to open the mobile app and enter the 6-digit code displayed on the screen.

Second factor: Push Notification

Figure 57. Two-factor authentication with push notification in which a push notification is sent to the end-user's mobile application for approval.



Two-Factor Authentication

Please select your preferred second factor of authentication.

Selecting the *Push Notification* option will require you to review a notification on the mobile app before granting access to the Serums system.

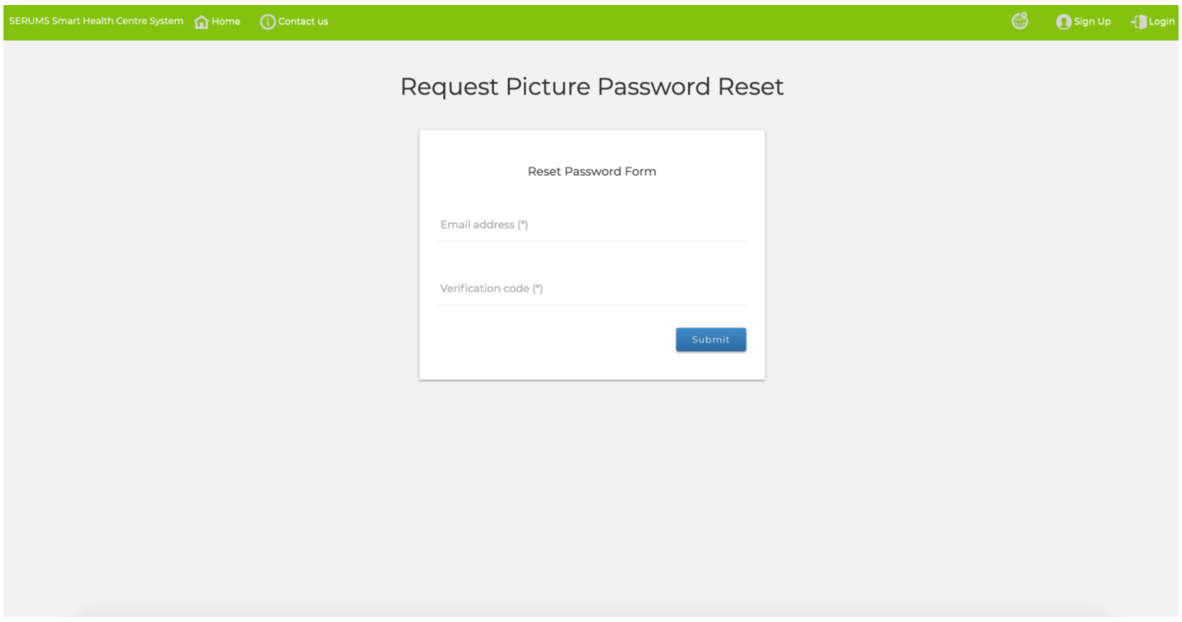
Selecting the *TOTP Code* will require you to open the mobile app and enter the 6-digit code displayed on the screen.

Second factor: TOTP Code

TOTP Code

Submit

Figure 58. Two-factor authentication with a Time-based One-Time Password. The end-user provides a one-time password code that can be found on the smartphone's mobile application.



Request Picture Password Reset

Reset Password Form

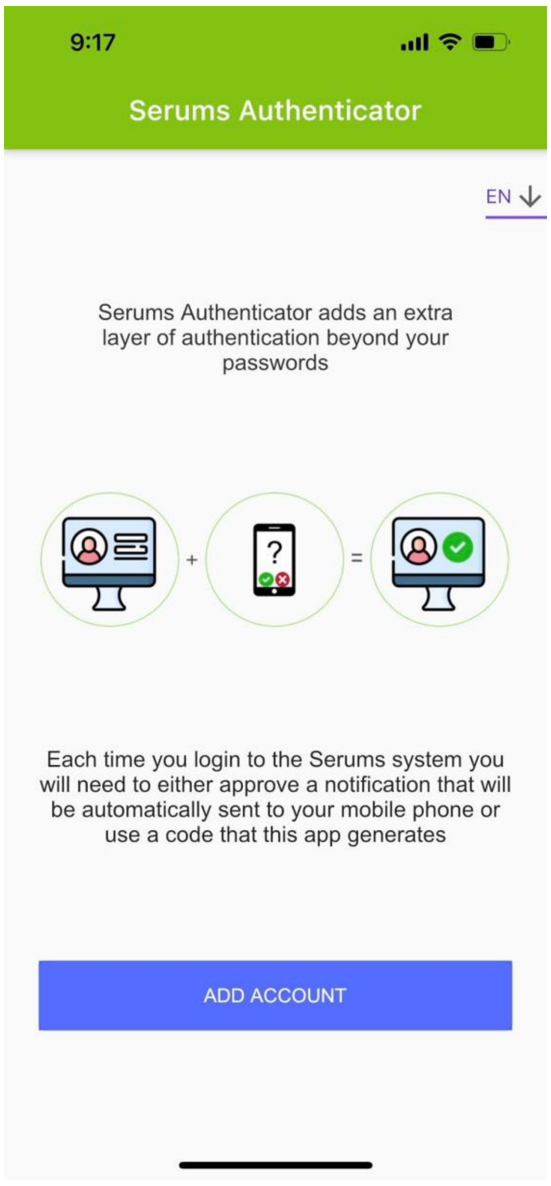
Email address (*)

Verification code (*)

Submit

Figure 59. End-users can request a reset code in case they forgot their password. The reset code will be sent in their associated email.

UI of the Mobile Application for Two-Factor Authentication


9:17

Serums Authenticator

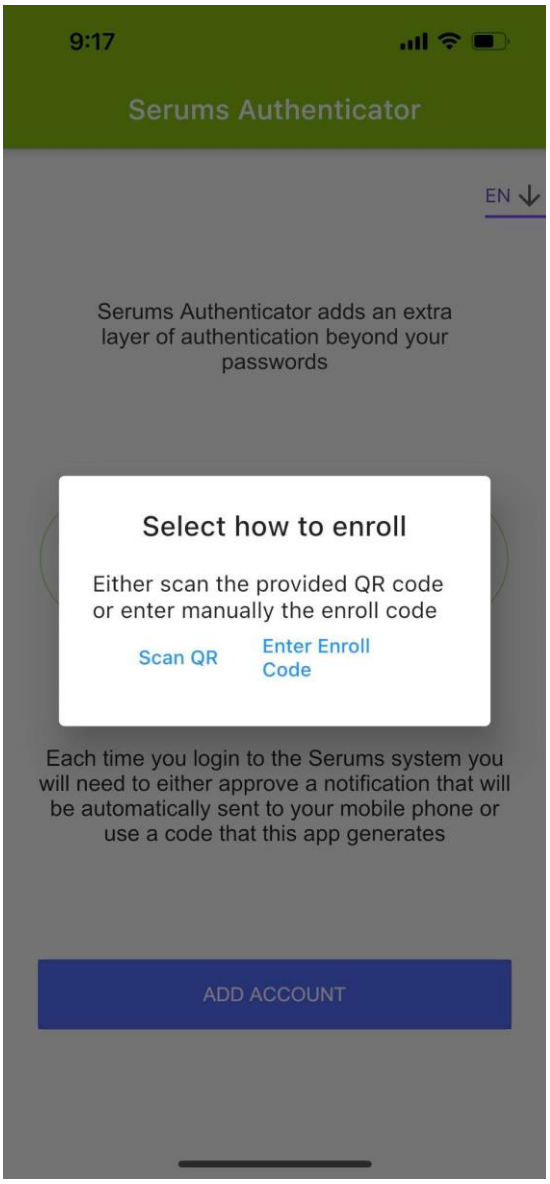
EN

Serums Authenticator adds an extra layer of authentication beyond your passwords

Each time you login to the Serums system you will need to either approve a notification that will be automatically sent to your mobile phone or use a code that this app generates

ADD ACCOUNT

Figure 60. User account creation and enrolment of the end-user's device for two-factor authentication.


9:17

Serums Authenticator

EN

Serums Authenticator adds an extra layer of authentication beyond your passwords

Select how to enroll

Either scan the provided QR code or enter manually the enroll code

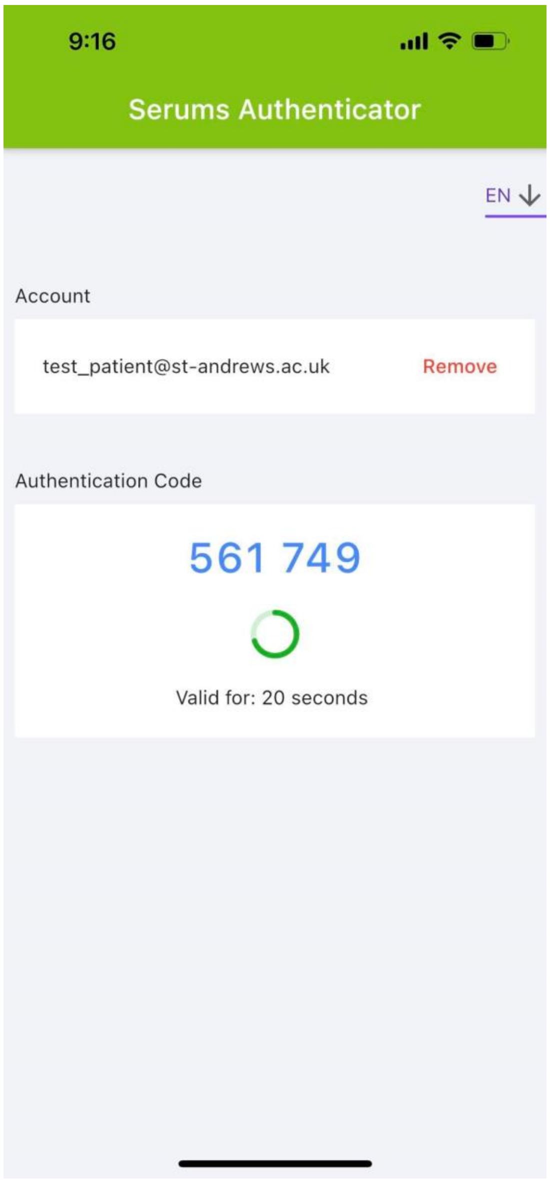
Scan QR

Enter Enroll Code

Each time you login to the Serums system you will need to either approve a notification that will be automatically sent to your mobile phone or use a code that this app generates

ADD ACCOUNT

Figure 61. Enrolment with QR code or enrollment code. In case the user selects the QR code option, the mobile application is ready to scan the QR code that is illustrated on the end-user's Web-based registration system of FlexPass. In case the user selects the enrollment code option, the user has to enter the secret code that is also available on the end-user's Web-based registration system of FlexPass.


9:16

Serums Authenticator

EN

Account

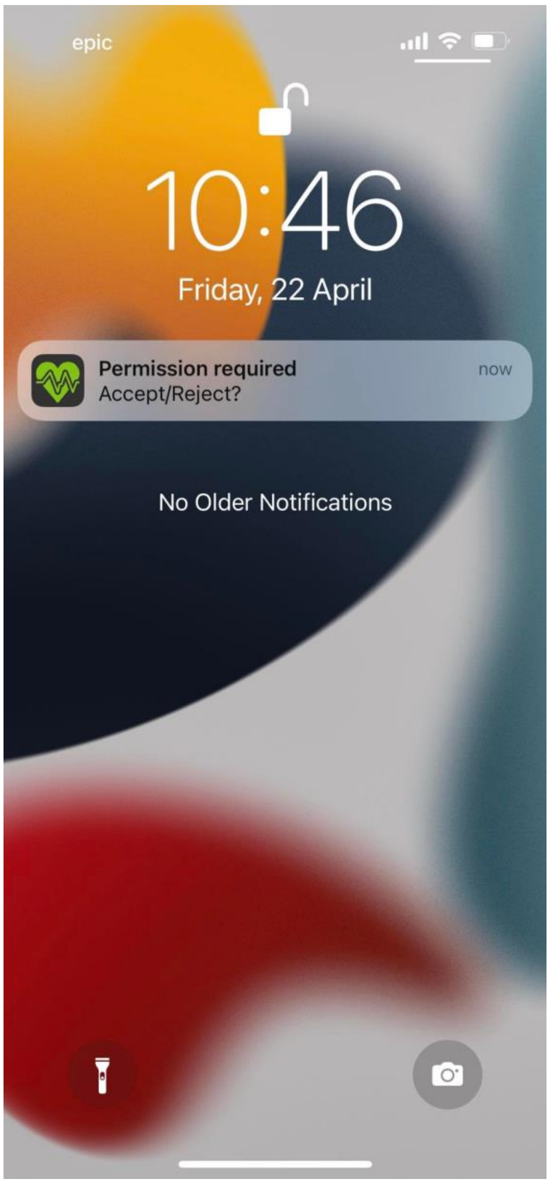
test_patient@st-andrews.ac.uk Remove

Authentication Code

561 749

Valid for: 20 seconds

Figure 62. Time-based One-Time Password on the end-user's smartphone mobile application that is automatically reset every 30 seconds. The one-time password can be used by the user during two-factor authentication login.



Permission required
Accept/Reject?

Figure 63. Push notification for two-factor authentication approval to login.

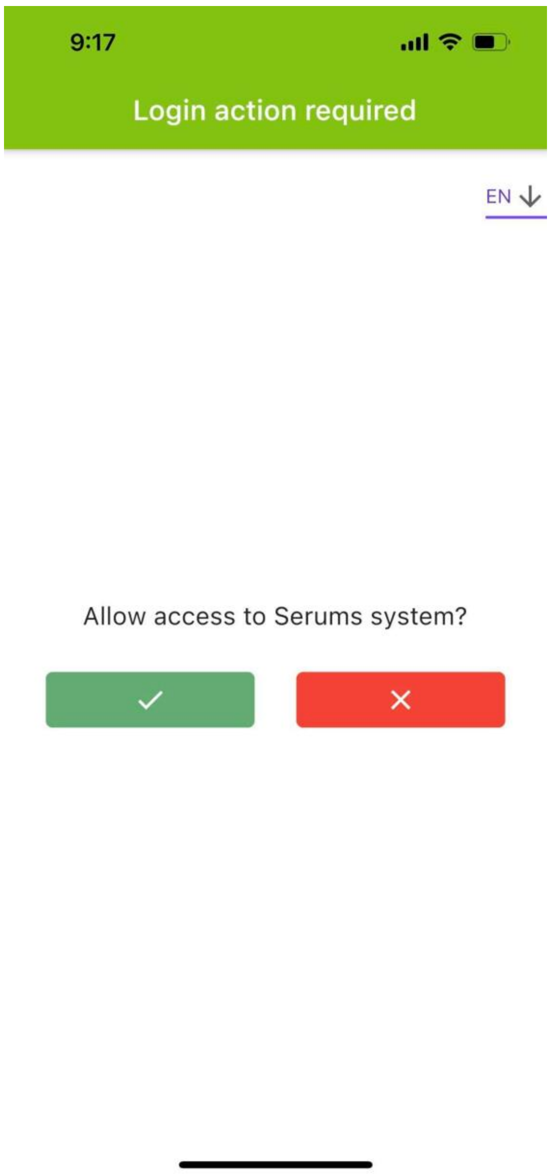
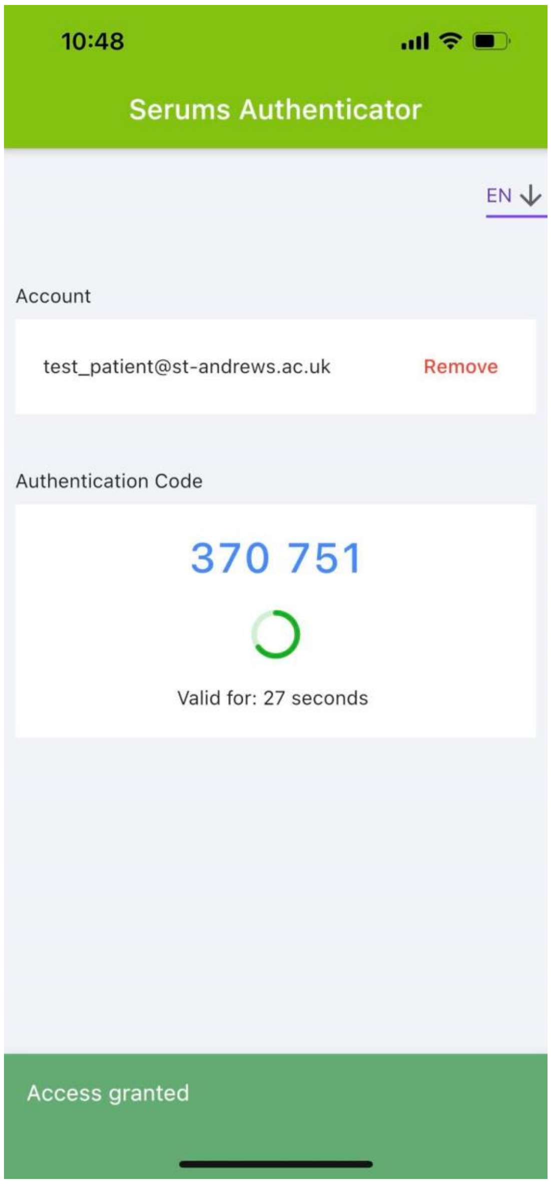


Figure 64. Two-factor authentication approval page. The user either approves or rejects the push notification of the login attempt.


10:48

Serums Authenticator

EN

Account

test_patient@st-andrews.ac.uk Remove

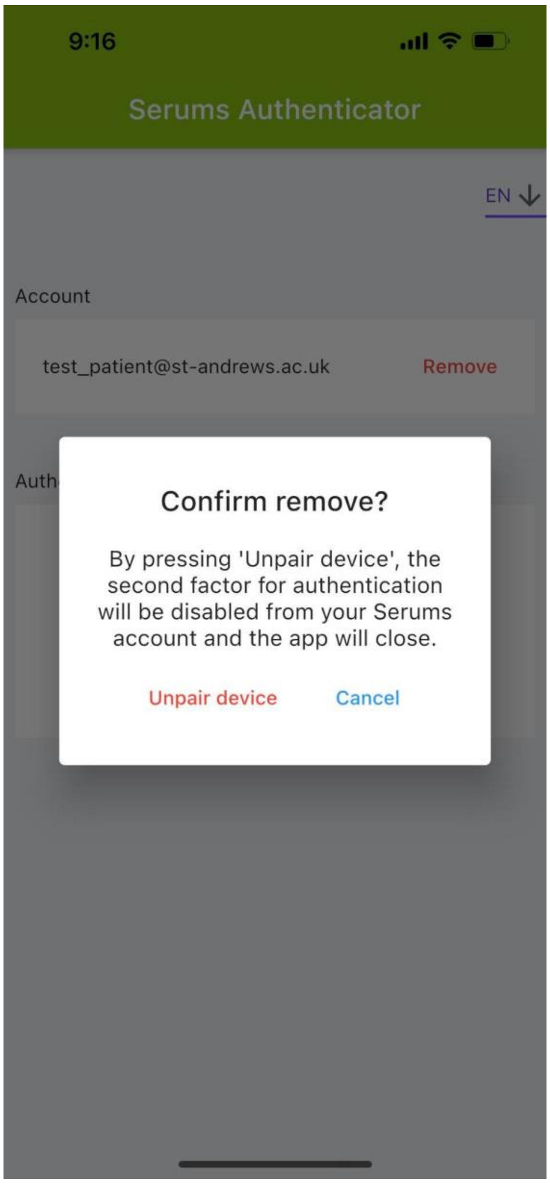
Authentication Code

370 751

Valid for: 27 seconds

Access granted

Figure 65. Notification after the user accepts the push notification.


9:16

Serums Authenticator

EN

Account

test_patient@st-andrews.ac.uk Remove

Auth

Confirm remove?

By pressing 'Unpair device', the second factor for authentication will be disabled from your Serums account and the app will close.

Unpair device Cancel

Figure 66. End-users can remove the second factor for authentication if they wish.

APPENDIX C – RESTful Application Programming Interface

This section lists all the endpoints of the final version of the user authentication system. Note that this section provides all the successful scenarios and their respective responses (*i.e.*, 200, 201). The full list of responses (including for *e.g.*, 400 – Bad Request; 401 – Unauthorized; 500 – Internal Server Error, etc.), is available at the Serums' development and testing server.

Base url: https://authentication.serums.cs.st-andrews.ac.uk/ua

Demo: https://authentication.serums.cs.st-andrews.ac.uk/ua/demo

Documentation: https://authentication.serums.cs.st-andrews.ac.uk/ua/doc

Create Admin API Token

Description: Creates an expiring API token that must be used by the web application to authorize the requests to the secured endpoints

Endpoint	/create_api_token/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
organization *	string (Organization) Enum ["ZMC", "USTAN", "FCRB"]
web_key *	string (Web key) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_str	string (A string value associated with the resource_name)
resource_expires_in_sec	float (The expiration time in seconds)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/create_api_token/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"admin@test.com\", \"organization\": \"USTAN\", \"web_key\": \"6HRrEPetK6UadiSnmtHFLJmnw5CN1Hi9su9LQvpF7peR8hBuOa\"}"
Response	
Schema	application/json
Description	Expiring API Token has been created successfully. The value is returned in resource_str and expires in resource_expires_in_sec seconds.
Status Code	201
Body	{ "message": "Expiring API Token has been created successfully. The value is returned in \`resource_str\` and expires in \`resource_expires_in_sec\` seconds.", "resource_name": "token", "resource_str": " 68b9d94424b118c6d3606320d20da2ac8721c297", "resource_expires_in_sec": 601126.755196}

Register Serums User

Description: Creates a new Serums user instance

Endpoint	/register_user/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Token: <Expiring API Token>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
organization *	string (Organization) Enum ["ZMC", "USTAN", "FCRB"]
role *	string (Role) Enum ["HOSPITAL_ADMIN", "MEDICAL_STAFF", "PATIENT"]
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_int	integer (An integer value associated with the resource_name)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/register_user/" -H "accept: application/json" -H "Authorization: Token 68b9d94424b118c6d3606320d20da2ac8721c297" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\", \"organization\": \"USTAN\", \"role\": \"PATIENT\"}"
Response	
Schema	application/json
Description	User has been created successfully. The value is returned in resource_int.
Status Code	201
Body	{ "message": "User has been created successfully. The value is returned in \`resource_int\`.", "resource_name": "user", "resource_int": 371 }

Check Username

Description: Checks whether the provided username can be activated or not based on the provided verification code

Endpoint	/check_username/
Method	POST
Headers	
accept	application/json
Content-Type	application/json

Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string (Verification code) [1 .. 50] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/check_username/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\", \"verification_code\": \"30ad3f1839e2478cad17d42fc02aa6bc\"}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Success", "resource_name": "user" }

Set Graphical Password

Description: Sets the graphical password data for the provided user

Endpoint	/set_graphical_password/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
graphical_password *	string (Graphical password) [1 .. 200] characters
time_started_creation *	Integer (Time started creation) [0 .. 9223372036854776000]
time_finished_creation *	Integer (Time finished creation) [0 .. 9223372036854776000]
time_first_gesture_started *	Integer (Time first gesture started) [0 .. 9223372036854776000]
time_first_gesture_fin *	Integer (Time first gesture finished) [0.. 9223372036854776000]
time_second_gesture_started *	Integer (Time second gesture started) [0.. 9223372036854776000]
time_second_gesture_fin *	Integer (Time second gesture finished) [0 .. 9223372036854776000]
time_third_gesture_started *	Integer (Time third gesture started) [0 .. 9223372036854776000]
time_third_gesture_fin *	Integer (Time third gesture finished) [0 .. 9223372036854776000]
total_time_creation *	Integer (Total time creation) [0 .. 9223372036854776000]
total_time_creation_with_confirm *	Integer (Total time creation with confirm) [0 .. 9223372036854776000]
total_time_first *	Integer (Total time first gesture) [0 .. 9223372036854776000]
total_time_second *	Integer (Total time second gesture) [0 .. 9223372036854776000]
total_time_third *	Integer (Total time third gesture) [0 .. 9223372036854776000]

total_failed_attempts *	Integer (Total failed attempts) [0 .. 9223372036854776000]
total_restart_attempts *	Integer (Total restart attempts) [0 .. 9223372036854776000]
total_time_creation_task *	Integer (Total time creation task) [0 .. 9223372036854776000]
timestamp_page_load *	Integer (Timestamp page load) [0 .. 9223372036854776000]
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/set_graphical_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\", \"graphical_password\": \"#1|T|55,29#2|T|67,22#3|T|93,47\", \"time_started_creation\": 195, \"time_finished_creation\": 13244, \"time_first_gesture_started\": 6598, \"time_first_gesture_fin\": 7451, \"time_second_gesture_started\": 8536, \"time_second_gesture_fin\": 9758, \"time_third_gesture_started\": 11724, \"time_third_gesture_fin\": 12782, \"total_time_creation\": 6184, \"total_time_creation_with_confirm\": 9315, \"total_time_first\": 853, \"total_time_second\": 1222, \"total_time_third\": 1058, \"total_failed_attempts\": 0, \"total_restart_attempts\": 0, \"total_time_creation_task\": 14894, \"timestamp_page_load\": 1650548204}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Success", "resource_name": "user" }

Retrieve Graphical Info

Description: Retrieves the graphical information (*image_id* and *image_type*)

Endpoint	/retrieve_graphical_info/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
image_id	integer (The ID of the image)
image_type	string (The type of the image)

Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/retrieve_graphical_info/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\"}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Success", "resource_name": "image", "image_id": 1, "image_type": "retrospective" }

Create JWT

Description: Creates a JSON Web Token if the provided credentials are correct

Endpoint	/create_jwt/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) non-empty
password *	string (Password) non-empty
login_type *	string (The type of login) Enum ["TEXT", "GRAPHICAL"]
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_obj	object (A dictionary that contains the JWT in the form of key-value pairs. The key access is a string that corresponds to the JWT access token and the key refresh is a string that corresponds to the JWT refresh token.)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/create_jwt/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\", \"password\": \"#1|T|55,29#2|T|67,22#3|T|93,47\", \"login_type\": \"GRAPHICAL\"}"
Response	
Schema	application/json
Description	JSON Web Token has been created successfully. The value is returned in resource_obj.

Status Code	201
Body	{ "message": "JSON Web Token has been created successfully. The value is returned in \`resource_obj\`.", "resource_name": "jwt", "resource_obj": { "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0", "refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTY4MjA4NDk0NSwianRpIjoiY2IyNTUwZWVkNmM5NDYwZjk0OTY4ZjQ3ZGI3OTM4NTAiLCJ1c2VySUQiOjM3MSwiaXNzIjoiU2VydW1zQXV0aGVudGljYXRpb24iLCJpYXQiOjE2NTA1NDg5NDUsInN1YiI6InRlc3RfcGF0aWVudEBzdC1hbmRyZXdzLmFjLnVrIiwiZ3JvdXBJRHMiOlsiUEFUSUVOVCJdLCJvcmdJRCI6IlVTVEFOIiwiZGVwdElEIjpudWxsLCJkZXB0TmFtZSI6bnVsbCwic3RhZmZJRCI6bnVsbCwibmFtZSI6bnVsbCwiYXVkIjoiaHR0cHM6Ly9zaGNzLnNlcnVtcy5jcy5zdC1hbmRyZXdzLmFjLnVrLyJ9.SO5YVEQtBl4-KZGTTjDsV-ognylun93VWJjRC5rF3cE" } }

Set Graphical Info

Description: Sets the *image_id* and the *image_type* for the user

Endpoint	/set_graphical_info/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
image_type *	string (Image type) [1 .. 13] characters
image_id *	integer
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/set_graphical_info/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIi

	wiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{ \"image_type\": \"retrospective\", \"image_id\": 1}"
Response	
Schema	application/json
Description	Success
Status Code	200

Body	{
	  "message": "Success",
	  "resource_name": "user"
	}

Set Passphrase

Description: Sets the *single_secret* passphrase for the provided user

Endpoint	/set_passphrase/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
single_secret *	string (Single secret) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/set_passphrase/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{ \"single_secret\": \"The day I had lunch at the hospital\"}"
Response	

Schema	application/json
Description	Success
Status Code	200

Body	{
	  "message": "Success",
	  "resource_name": "user"
	}

Set Second Factor

Description: Sets the *second_factor* for the user

Endpoint	/set_second_factor/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
second_factor *	string (The type of login) Enum ["MOBILE", "TOTP"]
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/set_second_factor/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{ \"second_factor\": \"MOBILE\"}"
Response	
Schema	application/json
Description	Success
Status Code	200

Body	{
	  "message": "Success",
	  "resource_name": "user"
	}

Check Passphrase Set

Description: Checks whether the passphrase was set or not for the provided username

Endpoint	/check_passphrase_set/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_already_activated	boolean (True if resource is already activated, else False.)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/check_passphrase_set/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\"}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Success", "resource_name": "passphrase", "resource_already_activated": true }

Refresh JWT

Description: Uses the longer-lived refresh token to obtain another access token

Endpoint	/refresh_jwt/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
refresh *	string (Refresh) non-empty
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_str	string (A string value associated with the resource_name)
Example Call	
Request	

Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st- andrews.ac.uk/ua/refresh_jwt/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"refresh\": \"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoicmVmcm VzaCIsImV4cCI6MTY4MjA4NDk0NSwianRpIjoiY2IyNTUwZWVkNmM5ND YwZjk0OTY4ZjQ3ZGI3OTM4NTAiLCJ1c2VySUQiOjM3MSwiaXNzIjoiU2Vy dW1zQXV0aGVudGljYXRpb24iLCJpYXQiOjE2NTA1NDg5NDUsInN1YiI6InR lc3RfcGF0aWVudEBzdC1hbmRyZXdzLmFjLnVrIiwiZ3JvdXBJRHMiOlsiUEF USUVOVCJdLCJvcmdJRCI6IlVTVEFOIiwiZGVwdElEIjpudWxsLCJkZXB0TmF tZSI6bnVsbCwic3RhZmZJRCI6bnVsbCwibmFtZSI6bnVsbCwiYXVkIjoiaHR0 cHM6Ly9zaGNzLnNlcnVtcy5jcy5zdC1hbmRyZXdzLmFjLnVrLyJ9.SO5YVEQ tBl4-KZGTTjDsV-ognylun93VWJjRC5rF3cE\"}"
Response	
Schema	application/json
Description	JSON Web Token has been created successfully. The value is returned in resource_str.
Status Code	201

Body	{
	  "message": "JSON Web Token has been created",
	  "resource_name": "jwt",
	  "resource_str":
	"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzI
	iwiZXhwIjoxNjUzMTQxNDQ2LCJqdGkiOiI1MjkwYTlmNDg3ZTU0NDk4YjB
	mNmIxZTcwOTAxNTBlNSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXR
	oZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wY
	XRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJ
	RU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW
	1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHR
	wczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.c
	N25G4kb-Nycl0HVct4h5bZVlwbmB2BnGM8T5KcWi98"
	}

Check Second Factor Set

Description: Checks whether the second factor was set or not by the user

Endpoint	/check_second_factor_set/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_str	string (A string value associated with the resource_name)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st- andrews.ac.uk/ua/check_second_factor_set/" -H "accept:

	application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Success
Status Code	200

Body	{
	  "message": "Success",
	  "resource_name": "second_factor",
	  "resource_str": "MOBILE"
	}

Store Graphical Login Attempt

Description: Stores the graphical login attempt

Endpoint	/store_graphical_login_attempt/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) non-empty
total_failed_attempts *	integer (Total failed attempts) [0 .. 2147483647]
is_reset *	boolean (Is reset)
is_reset_from_main_page *	boolean (Is reset from main page)
total_time_until_submit *	integer (Total time until submit) [0 .. 9223372036854776000]
total_time_until_successful_login *	integer (Total time until successful login) [0 .. 9223372036854776000]
time_interaction_started *	integer (Time interaction started) [0 .. 9223372036854776000]
total_time_since_page_load *	integer (Total time since page load) [0 .. 9223372036854776000]
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/store_graphical_login_attempt/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIi

	wiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YW U4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBd XRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9 wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQ VRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHRO YW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJo dHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvI n0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\", \"total_failed_attempts\": 0, \"is_reset\": false, \"is_reset_from_main_page\": false, \"total_time_until_submit\": 16792, \"total_time_until_successful_login\": 1254, \"time_interaction_started\": 108, \"total_time_since_page_load\": 22755}"
Response	
Schema	application/json
Description	Success
Status Code	200

Body	{
	  "message": "Success",
	  "resource_name": "GRAPHICAL"
	}

Store Passphrase Login Attempt

Description: Stores the passphrase login attempt

Endpoint	/store_passphrase_login_attempt/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) non-empty
is_failed_attempt *	boolean (Is failed attempt)
is_reset *	boolean (Is reset)
total_time_until_submit *	integer (Total time until submit) [-9223372036854776000 .. 9223372036854776000]
total_time_until_submit_since_page_load *	integer (Total time until submit since page load) [-9223372036854776000 .. 9223372036854776000]
time_interaction_started *	integer (Time interaction started) [-9223372036854776000 .. 9223372036854776000]
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/store_passphrase_login_attempt/" -H "accept: application/json" -H "Authorization: Bearer

	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIi wiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YW U4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBd XRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9 wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQ VRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHRO YW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJo dHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvI n0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content- Type: application/json" -d "{ \"username\": \"test_patient@st- andrews.ac.uk\", \"is_failed_attempt\": false, \"is_reset\": false, \"total_time_until_submit\": 4568, \"total_time_until_submit_since_page_load\": 12786, \"time_interaction_started\": 1208}"
Response	
Schema	application/json
Description	Success
Status Code	200

Body	{
	  "message": "Success",
	  "resource_name": "TEXTUAL"
	}

Request Device Enroll

Description: Returns an activation code as a QR code PNG image for the user

Endpoint	/request_device_enroll/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_obj_qr	object (A dictionary that contains information about the QR code in the form of key-value pairs. The key img_byte_str is a base64 encoded string that corresponds to the image bytes. The key qr_img_id is a string that corresponds to the image id. The key enroll_text_id is a string that corresponds to the enroll id as text.)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st- andrews.ac.uk/ua/request_device_enroll/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIi wiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YW U4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBd XRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9 wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQ VRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHRO

	YW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	QR code has been created successfully. The value is returned in resource_obj_qr.
Status Code	201
Body	{ "message": "QR code has been created successfully. The value is returned in \`resource_obj_qr\`.", "resource_name": "QR", "resource_obj_qr": { "qr_img_byte_str": "iVBORw0KGgoAAAANSUhEUgAAAcIAAAHCAQAAAABUY/ToAAADiUlEQVR4nO2cW4rcSBBFT0wK6jMLegG9FGlnwyxpdiAtxQswZH4aJMIf+ZBUbWMw1e7q8o0PgZAOkiDIuPFImfN7tvzzmyCIFClSpEiRIkU+HmnVBsxsALKZTWxmE8By3cym3O6aPvhtRT4Wibu7M7q7ewruc9xPwd1XKgjGkzn6GtUnd6psjnIIebc4OwGmyDjwlsTC8Ul7rfM0U+OTmmzSAP3XM2e/dninwKMrr7DLBcgwPBKe4Tiwhym9jl0X2eKfKpyMXMzK7AmOrB/v1yccgDNuUBYCtp2ce/rciHIoseOjQ8lmvAYa2H5fWb0VakOz1T5HORh9y+LkHuNZaNvtIOLdIRlduLvLXiQ8QVn+NaFE+pFJX6UArlanGfLo/kQyK7VR/qVhej7lKJg0YCgnxI5Nm6HtoGiAkjrgPkK0as5SLIYbWih/LLR76tyEckux5a8RnqoUQwgvscqx5iTO2C9JDIk1UfSntUC11TH4Nc86YVxTKRZ6seAfhM7bkWf6k52C62W1tWPiTyZLumLu5TvaSGthbB2tVZmlrkGzusQ56gZfl99iOFVjhKofiQ9JDIsx1iVPGcpnj2GmNdkYpu0jok8tZ8t5l94emN16KMqjySHhL5U3IxM5tibcofsvw5D1UUzfd+psgnIXtuH2oeP6bj6nPQ2UVOJ5AeEnm2YxjrAqiq696HTX10KLpimcgb2zX1Tchqxer9FFC/TORb94emN16KMqjySHhL5U3IxM5tibcofsvw5D1UUzfd+psgnIXtuH2oeP6bj6nPQ2UVOJ5AeEnm2YxjrAqiq696HTX10KLpimcgb2zX1Tchqxer9FFC/TORbO9WHel/D+4jHIVdr7RD5kMiT3daH9pRsHx3qt4DqQyJvrfTtjfh1gOgY1LFGXwycfC0HRgcjpjb0+Lm+U+T7kW32l18c8svqy+sKsA0/+iGIk4c2lP+5vlPkHyHNXlf2bUE2xRWza3CbCO7/lXH8TXulRZ6txTIwyFYClY1fLm4QnGXaDOK3spvDxv/Dah/3tiIfkTz2OmqGdiwl1kIjp1PlZSJPtuf21PGz47DHPnRWEjb1XEX+jCwpfKo1xqaMzEq/jGxW+x+ATfd5psgnIQ+zsHWl8b4FKPWaUV+q+sys1iGR3W58qDbN0mmPIvRJWdUYRf6SLH/7iCvl10OLDcWRbKJeqBXrh3hbkQ9AvvkPWm2L7dP5XSi1LYuKZSJPVv9htZSiT8DGGQw2q/sW6YXpfHEDyi2f7TtFvh9p+se5SJEiRYoUKflvJ78D2wJlcBrhmcIAAAAASUVORK5CYII=" , "qr_img_id": "dae881b5c555464192bb11ec5e0410af", "enroll_text_id": "BdEpTU" } }

Poll Enroll Status

Description: Polls the qr_img_id to detect whether the device was enrolled or not

Endpoint	/poll_enroll_status/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**

qr_img_id *	string (QR img id) [1 .. 50] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_already_activated	boolean (True if resource is already activated, else False)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/poll_enroll_status/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{ \"qr_img_id\": \"dae881b5c555464192bb11ec5e0410af\"}"
Response	
Schema	application/json
Description	Device is already activated
Status Code	200

Body	{
	  "message": "Device is already activated",
	  "resource_name": "device",
	  "resource_already_activated": false
	}

Check Device Enrolled

Description: Checks whether the device is already enrolled or not

Endpoint	/check_device_enrolled/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_already_activated	boolean (True if resource is already activated, else False)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/check_device_enrolled/" -H "accept:

	application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Device is already activated
Status Code	200

Body	{
	  "message": "Device is already activated",
	  "resource_name": "device",
	  "resource_already_activated": false
	}

Enroll Device

Description: Enrolls the device to the user's account

Endpoint	/enroll_device/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
device_name *	string (Device name) non-empty
device_id *	string (Device id) non-empty
enroll_id *	string (Enroll id) non-empty
operation *	string (Operation) non-empty
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_obj_device	object (A dictionary that contains information about the enrolled device in the form of key-value pairs. The key totp is a string that will be used for the time-based one-time passwords. The key username is a string that corresponds to the username set for the enrolled device. The key jwt_access is a string that corresponds to the JWT access token that will be saved on the device. The key jwt_refresh is a string that corresponds to the JWT refresh token that will be saved on the device.)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/enroll_device/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"device_name\": \"Xioami Mi 9T Pro\", \"device_id\": \"123456\", \"enroll_id\": \"eaa45e70-0fa5-

	11eb-a8c0-0242ac170005\", \"operation\": \"QR\"}"
Response	
Schema	application/json
Description	Device has been created successfully. The value is returned in resource_obj_device.
Status Code	201

Body	{
	  "message": "Device has been created successfully. The value is returned in `resource_obj_device`.",
	  "resource_name": "device",
	  "resource_obj_device": {
	    "totp": "d65c042c9b034080",
	    "username": "test_patient@st-andrews.ac.uk",
	    "jwt_access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQ1OTc5LCJqdGkiOiI5MGUwY2JmMzdjOTQ0ZjJiOTJiYWM1NDNhMjVhYmEzNCIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU1Mzk3OSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.X3r9l_Plh_y_VI9kjeFIkohDuygNiwW_Wl-nMpuXJH0",
	    "jwt_refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTY4MjA4OTk3OSwianRpIjoiZTY2YjkxMjlkNDBiNGNjOTkyMzI1NWY5ODk3ZDQ1YjciLCJ1c2VySUQiOjM3MSwiaXNzIjoiU2VydW1zQXV0aGVudGljYXRpb24iLCJpYXQiOjE2NTA1NTM5NzksInN1YiI6InRlc3RfcGF0aWVudEBzdC1hbmRyZXdzLmFjLnVrIiwiZ3JvdXBJRHMiOlsiUEFUSUVOVCJdLCJvcmdJRCI6IlVTVEFOIiwiZGVwdElEIjpudWxsLCJkZXB0TmFtZSI6bnVsbCwic3RhZmZJRCI6bnVsbCwibmFtZSI6bnVsbCwiYXVkIjoiaHR0cHM6Ly9zaGNzLnNlcnVtcy5jcy5zdC1hbmRyZXdzLmFjLnVrLyJ9.wK1vM0es9Aai2siGfyFiJiisGSIF_wdjL-99N350iIE"
	  }
	}

Map FCM to Device

Description: Maps the Firebase Cloud Messaging token to the user's device.

Endpoint	/map_fcm_to_device/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
device_id *	string (Device id) non-empty
fcm_token *	string (Fcm token) [1 .. 255] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)

Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/map_fcm_to_device/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQ1OTc5LCJqdGkiOiI5MGUwY2JmMzdjOTQ0ZjJiOTJiYWM1NDNhMjVhYmEzNCIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU1Mzk3OSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.X3r9l_Plh_y_VI9kjeFIkohDuygNiwW_Wl-nMpuXJH0" -H "Content-Type: application/json" -d "{ \"device_id\": \"123456\", \"fcm_token\": \"d0hnVaEW7AI:APA91bE0Hw-u78mkhvr0Vk61Rs3zop5Q2J8UL1xvFT-qLbqeT6xE48ulq_R_ZDmNnEfUHW4UAlrt6xg1IiVF-4DP1QzfMNRNF3sLNvcJsQEFRQ7iehAxud1QgRkA9cJQgQz0RSmDkInV\"}"
Response	
Schema	application/json
Description	Success
Status Code	200

Body	{
	"message": "Success"
	}

Submit TOTP

Description: Checks whether the provided TOTP code is verified or not

Endpoint	/submit_totp/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
totp_token *	string (Totp code) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_bool	boolean (Returns True/False that is associated with the resource_name)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/submit_totp/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIi

	wiZXhwIjoxNjUzMTQ1OTc5LCJqdGkiOiI5MGUwY2JmMzdjOTQ0ZjJiOTJiYWM1NDNhMjVhYmEzNCIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU1Mzk3OSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.X3r9l_Plh_y_VI9kjeFIkohDuygNiwW_Wl-nMpuXJH0" -H "Content-Type: application/json" -d "{ \"totp_code\": \"187542\"}"
Response	
Schema	application/json
Description	Authentication response
Status Code	200

Body	{
	  "message": "Authentication response",
	  "resource_name": "Authentication response",
	  "resource_bool": false
	}

Send Push Notification

Description: Sends a push notification to the enrolled device associated with the user

Endpoint	/send_push_notification/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_str	string (A string value associated with the resource_name)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/send_push_notification/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Success

Status Code	200

Body	{
	  "message": "Success",
	  "resource_name": "authentication_id",
	  "resource_str": "1ae4d876-0fab-11eb-a8c0-0242ac170005"
	}

Poll Auth Push Status

Description: Polls to detect whether the response received from user's device for the authentication_id is True or False.

Endpoint	/poll_auth_push_status/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
authentication_id *	string (Authentication id) non-empty
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_bool	boolean (Returns True/False that is associated with the resource_name)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/poll_auth_push_status/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{ \"authentication_id\": \"1ae4d876-0fab-11eb-a8c0-0242ac170005\"}"
Response	
Schema	application/json
Description	Authentication response
Status Code	200

Body	{
	  "message": "Authentication response",
	  "resource_name": "Authentication response",
	  "resource_bool": false
	}

Two Factor Response

Description: Receives the response from the second factor attempt initiated from the push notification

Endpoint	/two_factor_response/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
device_id *	string (Device id) non-empty
response *	boolean (Response)
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/two_factor_response/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQ1OTc5LCJqdGkiOiI5MGUwY2JmMzdjOTQ0ZjJiOTJiYWM1NDNhMjVhYmEzNCIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU1Mzk3OSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.X3r9l_Plh_y_VI9kjeFIkohDuygNiwW_Wl-nMpuXJH0" -H "Content-Type: application/json" -d "{ \"device_id\": \"123456\", \"response\": true}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Success" }

Verify JWT

Description: Returns the JWT payload {*userID*, *username*, *groupIDs*, *orgID*, *deptID*} if the JWT is successfully verified

Endpoint	/verify_jwt/
Method	POST
Headers	
accept	application/json

Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
groupIDs	Array of strings (The group IDs associated with the SERUMS userID)
orgID	string (The organization ID associated with the SERUMS userID)
userID	integer (The SERUMS userID)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/verify_jwt/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0"
Response	
Schema	application/json
Description	Success
Status Code	200

Body	{
	  "message": "Success",
	  "userID": 371,
	  "groupIDs": [
	    "PATIENT"
	  ],
	  "orgID": "USTAN"
	}

Check Reset Password

Description: Checks whether the provided username can be reset or not based on the provided reset code

Endpoint	/check_reset_password/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
reset_code *	string (Reset code) [1 .. 50] characters
Output Parameters	**Type (Description)**
message	string (A general message description)

organization	string (The organization name)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/check_reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\", \"reset_code\": \"d15dd2d79b694cc4bc6b493815a41db2\"}"
Response	
Schema	application/json
Description	Success
Status Code	200

Body	{
	"message": "Success",
	"organization": "USTAN",
	"resource_name": "user",
	}

Map User Info

Description: Maps the identity within the organization to the existing user information

Endpoint	/map_user_info/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
identity *	string (Identity) [1 .. 200] characters
organization *	string (Organization) Enum ["ZMC", "USTAN", "FCRB"]
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/map_user_info/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvI

	n0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{ \"identity\": \"123456789\", \"organization\": \"USTAN\"}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Success", "resource_name": "user", }

Poll Graphical Status

Description: Polls to detect whether the graphical password creation status has finished or not

Endpoint	/poll_graphical_status/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
is_pending	boolean (Returns True/False that is associated with the task status)
resource_bool	Returns True/False that is associated with the resource_name
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/poll_graphical_status/" -H "accept: application/json" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Graphical status poll returned successfully
Status Code	200
Body	{ "message": "Success", "resource_name": "Graphical status", "is_pending": false, "resource_bool": true }

Remove Second Factor

Description: Removes the second factor for authentication (e.g., paired mobile app) from the user's account

Endpoint	/remove_second_factor/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/remove_second_factor/" -H "accept: application/json" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Success" }

Request Account Verification

Description: Request a verification code for account activation via email

Endpoint	/request_account_verification/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Token: <Expiring API Token>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
organization *	string (Organization) Enum ["ZMC", "USTAN", "FCRB"]
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_already_activated	boolean (True if resource is already activated, else False)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "https://authentication.serums.cs.st-andrews.ac.uk/ua/request_account_verification/" -H "accept: application/json" -H "Authorization: Token 68b9d94424b118c6d3606320d20da2ac8721c297" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\", \"organization\": \"USTAN\"}"
Response	

Schema	application/json
Description	User is already activated
Status Code	200
Body	{ "message": "Account verification code email sent successfully", "resource_name": "user", "resource_already_activated": true }

Request Reset Verification

Description: Requests a reset token for password reset via email

Endpoint	/request_reset_verification/
Method	GET
Headers	
accept	application/json
Content-Type	application/json
Authorization	Token: <Expiring API Token>
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
organization *	string (Organization) Enum ["ZMC", "USTAN", "FCRB"]
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "https://authentication.serums.cs.st-andrews.ac.uk/ua/request_reset_verification/" -H "accept: application/json" -H "Authorization: Token 68b9d94424b118c6d3606320d20da2ac8721c297" -H "Content-Type: application/json" -d "{ \"username\": \"test_patient@st-andrews.ac.uk\", \"organization\": \"USTAN\"}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Account verification code email sent successfully", "resource_name": "password_reset_code" }

Retrieve ID Info

Description: Retrieves the information about the identity of the user

Endpoint	/retrieve_id_info/
Method	GET

Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_obj_id	object (A dictionary that contains information about the identities of the user in the form of key-value pairs. Each key corresponds to the organization (string) and the associated value corresponds to the identity (string) possessed within the organization.)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "https://authentication.serums.cs.st-andrews.ac.uk/ua/retrieve_id_info/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	The values of identities are returned in resource_obj_id.
Status Code	200

Body	{
	 "message": "Success",
	 "resource_name": "user",
	 "resource_obj_id": {
	   "USTAN": "123456789"
	 }
	}

Retrieve User Info

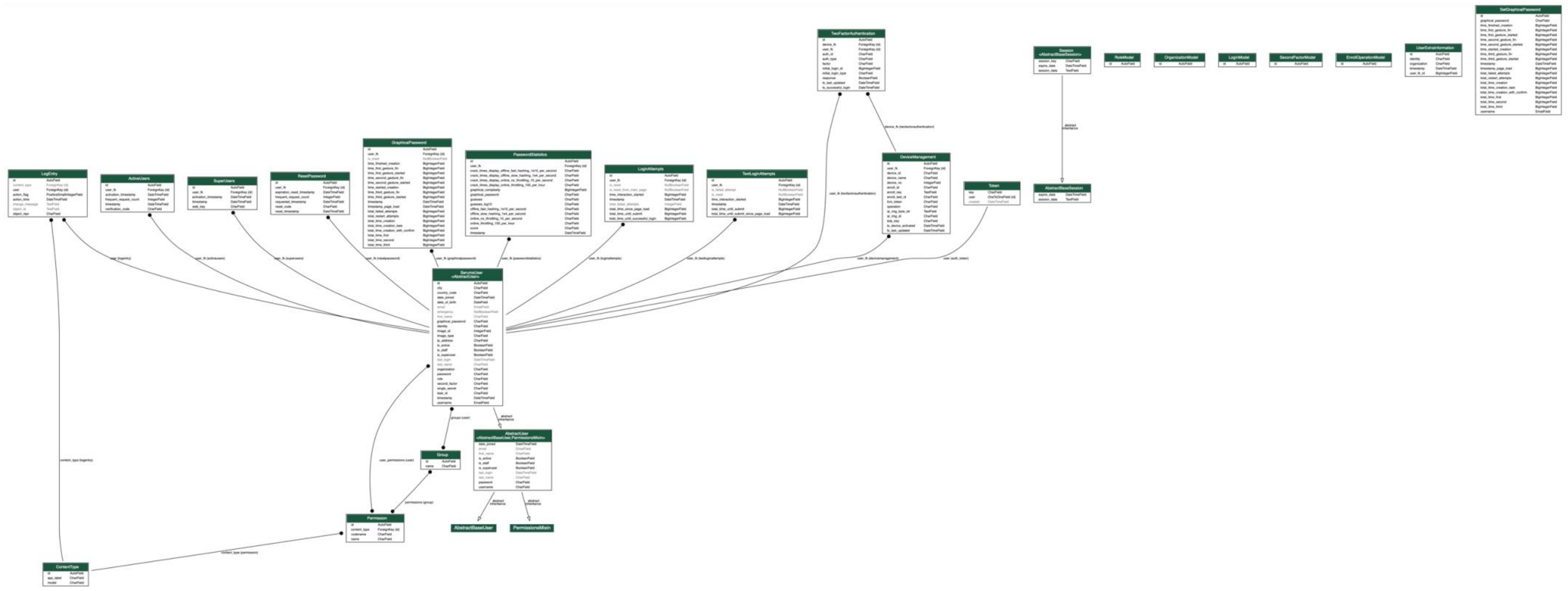
Description: Retrieves the information about the user (*i.e., Serums ID*, *date of birth*, etc.) from a specific organization (*e.g., patient ID*, *staff ID*, etc.).

Endpoint	/retrieve_user_info/
Method	GET
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**

identity *	string (Identity) [1 .. 50] characters
organization *	string (Organization) Enum ["ZMC", "USTAN", "FCRB"]
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_obj_info	object (A dictionary that contains information about user ID from a specific organization (e.g., patient ID, staff ID etc) in the form of key-value pairs. The key serums_id is an integer that corresponds to the associated Serums user ID. The key dob is a date object (None if not available) that corresponds to the date of birth of the associated user ID.)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "https://authentication.serums.cs.st-andrews.ac.uk/ua/retrieve_user_info/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjUzMTQwOTQ1LCJqdGkiOiI5MzE4NjNjZTQyNjU0NmE4YWU4MzNmYjZmZmNiYzkyMSIsInVzZXJJRCI6MzcxLCJpc3MiOiJTZXJ1bXNBdXRoZW50aWNhdGlvbiIsImlhdCI6MTY1MDU0ODk0NSwic3ViIjoidGVzdF9wYXRpZW50QHN0LWFuZHJld3MuYWMudWsiLCJncm91cElEcyI6WyJQQVRJRU5UIl0sIm9yZ0lEIjoiVVNUQU4iLCJkZXB0SUQiOm51bGwsImRlcHROYW1lIjpudWxsLCJzdGFmZklEIjpudWxsLCJuYW1lIjpudWxsLCJhdWQiOiJodHRwczovL3NoY3Muc2VydW1zLmNzLnN0LWFuZHJld3MuYWMudWsvIn0.qXgQp0AqhhK0ob97WX69kjaIZmCxCc0X-m_rLYfqrP0" -H "Content-Type: application/json" -d "{ \"identity\": \"123456789\", \"organization\": \"USTAN\"}"
Response	
Schema	application/json
Description	The values of identities are returned in resource_obj_id.
Status Code	200

Body	{
	  "message": "Success",
	  "resource_name": "user",
	  "resource_obj_info": {
	    "serums_id": "371",
	    "dob": ""
	  }
	}

APPENDIX D – Database Design (Entity-Relationship Diagram)


LogEntry
ActiveUsers
SuperUsers
ResetPassword
GraphicalPassword
PasswordStatistics
LoginAttempts
TextLoginAttempts
TwoFactorAuthentication
DeviceManagement
Token
Session <<AbstractBaseSession>>
AbstractBaseSession
RoleModel
OrganizationModel
LoginModel
SecondFactorModel
EnrollOperationModel
UserExtraInformation
SetGraphicalPassword
SerumsUser <<AbstractUser>>
Group
AbstractUser <<AbstractBaseUser,PermissionsMixin>>
AbstractBaseUser
PermissionsMixin
Permission
ContentType