



Project no. 826278

SERUMS

Research & Innovation Action (RIA)

SECURING MEDICAL DATA IN SMART-PATIENT HEALTHCARE SYSTEMS

Software on the Refined Verified User Authentication Scheme D5.3

Due date of deliverable: 31st October 2020

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: Juliana Bowles (USTAN)

Bram Elshof (Accenture)

Wanting Huang (Accenture)

Version 1.0

Project co-founded 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
V0.01	03/02/2020	Marios Belk (UCY), Andreas Pitsillides (UCY)	Defined TOC and added initial Executive Summary
V0.02	04/03/2020	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Added refined general architecture and extended description on use-case scenarios
V0.03	09/04/2020	Marios Belk (UCY), Christos Fidas (UCY), Argyris Constantinides (UCY), Andreas Pitsillides (UCY)	Added initial description of new APIs
V0.04	14/05/2020	Thomas Given-Wilson (UCL), Marios Belk (UCY)	Added description on the verification properties
V0.05	15/05/2020	Marios Belk (UCY), Christos Fidas (UCY), Argyris Constantinides (UCY), Andreas Pitsillides (UCY)	Finalized use-case scenarios
V0.06	29/05/2020	Elias Athanasopoulos (UCY)	Added the architecture and implementation details of the credential hardening mechanism
V0.07	02/07/2020	Marios Belk (UCY), Christos Fidas (UCY), Argyris Constantinides (UCY), Andreas Pitsillides (UCY)	Finalized new APIs
V0.08	18/09/2020	Thomas Given-Wilson (UCL), Marios Belk (UCY)	Added description of the verification of the user authentication system
V0.1	09/10/2020	Marios Belk (UCY), Christos Fidas (UCY), Elias Athanasopoulos (UCY), Argyris Constantinides (UCY), Andreas Pitsillides (UCY), Thomas Given-Wilson (UCL)	First draft of the deliverable
V0.2	20/10/2020	Marios Belk (UCY), Christos Fidas (UCY), Elias Athanasopoulos (UCY), Argyris Constantinides (UCY), Andreas Pitsillides (UCY), Thomas Given-Wilson (UCL), Eduard Baranov (UCL)	Beta version of the deliverable for internal review
V0.3	27/10/2020	Juliana Bowles (USTAN), Bram Elshof (Accenture), Wanting Huang (Accenture)	Version after partners' comments
V0.4	30/10/2020	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Pre-final version for final check
V1.0	30/10/2020	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: Mark Mestrum Email: m.mestrum@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

<u>EXECUTIVE SUMMARY</u>	<u>7</u>
<u>1 INTRODUCTION</u>	<u>8</u>
1.1 ROLE OF THE DELIVERABLE	8
1.2 RELATIONSHIP TO OTHER SERUMS DELIVERABLES	8
1.3 STRUCTURE OF THIS DOCUMENT.....	8
<u>2 REFINED FLEXIBLE AND PERSONALIZED AUTHENTICATION PARADIGM</u>	<u>9</u>
2.1 OVERVIEW	9
2.2 SUGGESTED RETROSPECTIVE APPROACH IN GRAPHICAL PASSWORDS.....	9
2.3 NEW DEVELOPED CUED-RECALL GRAPHICAL AUTHENTICATION MECHANISM	10
<u>3 GENERAL ARCHITECTURE OF THE USER AUTHENTICATION SYSTEM.....</u>	<u>11</u>
<u>4 CREDENTIAL HARDENING</u>	<u>12</u>
4.1 IMPLEMENTATION	12
4.2 STORING TEXTUAL AND GRAPHICAL PASSWORDS.....	14
<u>5 USE-CASE SCENARIOS.....</u>	<u>17</u>
5.1 ADMINISTRATOR LOGIN.....	17
5.2 ADMINISTRATOR CREATES AND ACTIVATES A USER ACCOUNT	18
5.3 END-USER ACTIVATES ACCOUNT.....	18
5.4 CREATION OF THE GRAPHICAL AND TEXTUAL PASSWORD	19
5.5 ENABLE TWO-FACTOR AUTHENTICATION TYPE AND PAIR MOBILE DEVICE	21
5.6 TWO-FACTOR AUTHENTICATION LOGIN USING THE MOBILE APPLICATION	22
<u>6 VERIFICATION OF AUTHENTICATION PROPERTIES.....</u>	<u>24</u>
6.1 MODEL OF THE USER AUTHENTICATION SYSTEM.....	25
6.2 PROPERTIES VERIFICATION	30
6.3 NEXT STEPS.....	30
<u>7 CONCLUSIONS.....</u>	<u>31</u>
<u>REFERENCES</u>	<u>32</u>
<u>ABBREVIATIONS</u>	<u>34</u>
<u>APPENDIX A – REFINED PROTOTYPE DESIGNS OF THE USER INTERFACES.....</u>	<u>35</u>

UI OF THE FLEXPASS HOMEPAGE AND DEMONSTRATION PAGE	35
UI OF THE SYSTEM ADMINISTRATOR’S PAGE	36
UI OF THE USER ACCOUNT REGISTRATION PAGE	37
UI OF THE GRAPHICAL PASSWORD CREATION PAGE	37
UI OF THE TEXTUAL PASSWORD CREATION PAGE	38
UI OF THE TWO-FACTOR AUTHENTICATION ACTIVATION PAGE	39
UI OF THE USER LOGIN PAGE.....	40
UI OF THE TWO-FACTOR AUTHENTICATION LOGIN PAGE	42
UI OF THE MOBILE APPLICATION FOR TWO-FACTOR AUTHENTICATION	43

APPENDIX B – RESTFUL APPLICATION PROGRAMMING INTERFACE 49

CREATE ADMIN API TOKEN.....	49
REGISTER SERUMS USER.....	50
CHECK USERNAME	50
SET GRAPHICAL PASSWORD	51
RETRIEVE GRAPHICAL INFO	52
CREATE JWT	53
SET GRAPHICAL INFO	54
SET PASSPHRASE.....	55
SET SECOND FACTOR	56
CHECK PASSPHRASE SET	57
REFRESH JWT	57
CHECK SECOND FACTOR SET.....	58
STORE GRAPHICAL LOGIN ATTEMPT	59
STORE PASSPHRASE LOGIN ATTEMPT	60
REQUEST DEVICE ENROLL	61
POLL ENROLL STATUS.....	62
CHECK DEVICE ENROLLED.....	63
ENROLL DEVICE.....	64
MAP FCM TO DEVICE	66
SUBMIT TOTP.....	67
SEND PUSH NOTIFICATION	68
POLL AUTH PUSH STATUS	68
TWO FACTOR RESPONSE.....	69
VERIFY JWT	70

APPENDIX C – UPDATED DATABASE DESIGN (ENTITY-RELATIONSHIP DIAGRAM) 72

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 third 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 “*D5.3. Software on the Refined Verified User Authentication Scheme*” describes the outcome and overall methodology that has been applied for the design and development of the refined software of the user authentication scheme. A User-Centered Design methodology will be adopted for developing and finalizing the user authentication scheme through multiple iterations (three releases are anticipated; initial, refined, final software) that will be used for evaluation studies. This deliverable produced the second version (refined) software of the user authentication scheme.

1 Introduction

1.1 Role of the Deliverable

The role of this deliverable is to report the design and development of the refined software of the user authentication scheme. Specifically, it reports: *i)* the improved user authentication paradigm based on a novel retrospective approach in graphical passwords; *ii)* the updated architecture of the user authentication scheme; *iii)* the architectural design and development details of the credential hardening mechanism; *iv)* the sequence diagrams of new, refined authentication use-case scenarios; *v)* the description of the new, refined Application Programming Interface (API) and updated database design of the user authentication scheme; *vi)* the improved prototype designs of the user interfaces of the authentication system; and *vii)* results of the verification of the user authentication system. The outcome of the deliverable constitutes the basis for the development of the final Serums authentication system and the evaluation of the second Proof of Concept (PoC2).

1.2 Relationship to Other Serums Deliverables

Deliverable	Relation
D2.5: Report on Final Specification of Smart Patient Health Records	Specifications of D5.3 will be used as input in the final specification of the Smart Patient Health Records
D2.6: Final Software for Storage, Access, Blockchain and Metadata Extraction for Smart Patient Health Records	The refined API of D5.3 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 D5.3 will be used as input for data fabrication and semantic-preserving encryption
D5.4: Report on Final User Authentication System	The outcome of D5.3 will be used as input for further developing and improving the development of the third development cycle of the user authentication scheme
D6.2: Report on Refined Smart Health Centre System Software	The outcome of D5.3 will be used as input for the refined version of the integrated smart healthcare system software
D7.5: Report on Refined Use Cases and Evaluation	The second version of the prototype designs of the user authentication scheme that were produced in D5.3 will be used in the context of the evaluation studies of PoC2

1.3 Structure of this Document

The rest of the document is structured as follows: *Chapter 2* describes the refined authentication paradigm. *Chapter 3* describes the updated general architecture of the user authentication system. *Chapter 4* provides implementation details of the credential hardening mechanism. *Chapter 5* describes the sequence diagrams of the new and refined user authentication scenarios. *Chapter 6* describes the results of the user authentication component verification. *Chapter 7* concludes the deliverable. *APPENDIX A* presents the new and refined front-end designs of the user authentication screens, which were designed based on the primacy use-case scenarios defined in Chapter 5. *APPENDIX B and C* respectively describe the refined API of the user authentication system, and the refined design of the database.

2 Refined Flexible and Personalized Authentication Paradigm

In this chapter we present the refined flexible and personalized authentication paradigm that is based on a novel retrospective approach in graphical passwords.

2.1 Overview

Building on the first version of the user authentication system [2], and the feedback gathered from the first proof-of-concept evaluation cycle, we have refined the FlexPass authentication paradigm [3] with regards to the graphical authentication process. In particular, FlexPass allows users to create secret picture passwords in which users are required to remember a secret combination of three areas on an image by drawing them on the image. Building on existing work, which has shown that using images based on the users' prior experiences and activities improves security and memorability [3-8], FlexPass further provides images tailored to each user's existing daily life activities and experiences related to locations of their healthcare organizations (*e.g.*, cafeteria, reception hall, garden, etc.) to make them more memorable and secure. Users are also allowed to use textual passwords by creating a secret passphrase, which they can utilize to flexibly switch between the graphical password in order to login. Next, we present the suggested retrospective approach in graphical passwords for personalizing images to each user's prior activities and experiences.

2.2 Suggested Retrospective Approach in Graphical Passwords

An important user interface design factor that affects the security strength of user-chosen PGA passwords (Picture Gesture Authentication) is the background image used [9-11]. Research has shown that the selections of images can be predictable since users prefer clear *vs.* incoherent images [12], and choose images that illustrate people [13, 14] and sceneries [13]. In addition, users' choices are influenced by human attributes in an image (*e.g.*, race, age, gender [15]), image colors and type [16].

Prior works [11, 13, 17] investigated the use of image semantics and their effects on the security of user-chosen passwords. Images can be broadly categorized as *generic* (*i.e.*, not directly relevant nor familiar to the users, *e.g.*, abstract, nature, landscapes, etc.) or *personal* (*i.e.*, directly relevant and highly familiar to the users, *e.g.*, depicting people, objects, or scenes highly personal to users). Studies in [10, 11, 18] indicate that *generic images* are susceptible to hotspots (points on an image that attract users to select them), thus, leading to the creation of easily predictable passwords. Subsequently, several works focused on alleviating the hotspot issue, mainly by limiting the available choices during password creation to prevent users from making selections on hotspots [11, 19, 20].

The use of *personal images* also impacts the security of user-chosen passwords, since it may result to the creation of passwords easily guessable by someone who knows the user [17, 21, 22]. The use of images that are familiar to the user (*e.g.*, containing family members) increases the likelihood of certain areas on the image to be selected as part of the password [11]. Furthermore, the fact that many users often do not understand security features [23] may lead to the use of personal photos that violate the privacy of others depicted in the photo, as well as theirs, since private information is revealed [24].

Hence, the aforementioned state-of-the-art approaches embrace deficiencies; when image content is delivered randomly, the security of the graphical password is depreciated since users, in an attempt to scaffold memorability, tend to choose easy-to-remember and predictable hotspots [17]; when users are allowed to upload image content, security and privacy considerations also arise since users tend to create easily guessable passwords [17] and often violate the privacy of people depicted in the uploaded images [24]. Therefore, there is a need for a more sophisticated approach within PGA schemes to



Figure 2. A picture password illustrating the three gestures on the background image. The left image illustrates a generic image (not familiar to an end-user), the *right image* illustrates a personalized image (a public image of the ZMC hospital use-case).

On the right side, there is the background image on which users can create their passwords by drawing three gestures. After each gesture is drawn, the shape of the gesture is temporarily displayed on the screen at the corresponding location, to provide feedback to the user that the gesture has been captured by the mechanism. Users are required to redraw the three gestures to confirm their graphical password.

During the login task, the user is presented with the same page and the user is required to enter the graphical password by reproducing all three gestures. The mechanism compares the entered password with the stored one and login is 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.

3 General Architecture of the User Authentication System

In this chapter we present the refined architectural design of the developed user authentication system. **Figure 3** illustrates the refined high-level architectural design of the user authentication system, which now includes a password-hardening component (please see Section 3). The user authentication API is built and deployed using Docker (version: 19.03.13, API version: 1.40) and it is hosted at the University of Cyprus' (UCY) premises on a Kernel-based Virtual Machine (KVM) running CentOS Linux version 7 with 1GB of RAM and 40GB of disk space. The APIs have been implemented as a Django application in Python 3.7.4, using the Django REST Framework (DRF), which is an open-source Python and Django library intended for building Web APIs. The main benefits of using DRF include the feature of Web-browsable API, the support of broad categories of authentication schemes, and the powerful serialization for converting complex data into native Python data types. 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 WSGI (Web Server Gateway Interface) application. Furthermore, to support fast request-response cycles and deal with time-consuming tasks we use Celery, which is an asynchronous task queue based on distributed message passing. We also use RabbitMQ as the external message broker solution required by Celery. To store users' data, we use PostgreSQL which is an open-source Relational Database Management System (RDBMS) commonly used within Django applications.

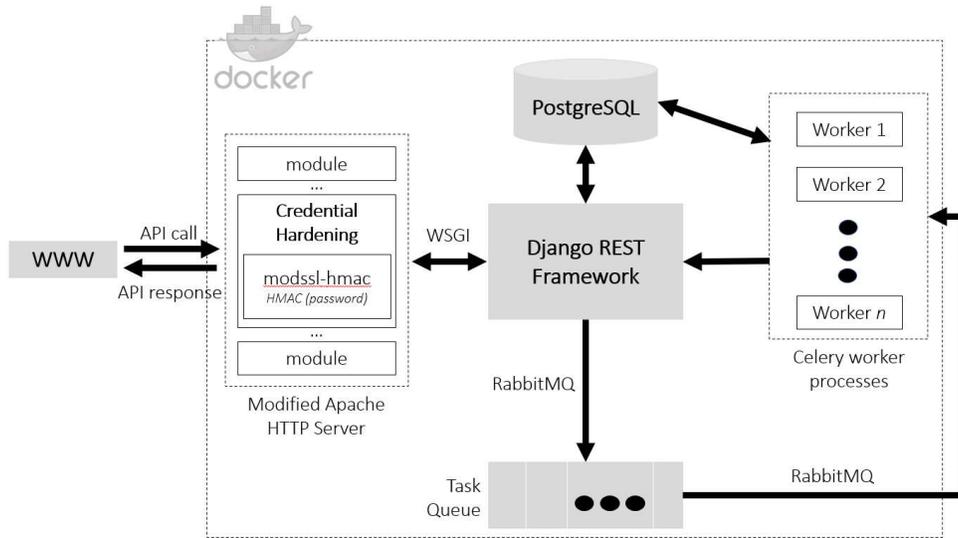


Figure 3. Refined high-level architectural design and technologies used.

4 Credential Hardening

In this section, we present how Serums employs additional countermeasures in order to defend against attacks that are based on cracking off-line leaked credentials.

4.1 Implementation

Recall that Serums secures a text-based password using a MAC, instead of a cryptographic hash function (please refer to D5.2 [2]). In particular, 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 (internally) the cryptographic hash. **Figure 4** illustrates an overview of how credential hardening works in the Serums server. In **Figure 4**, we assume that Web App is the front-end (User Interface) of the Serums infrastructure.

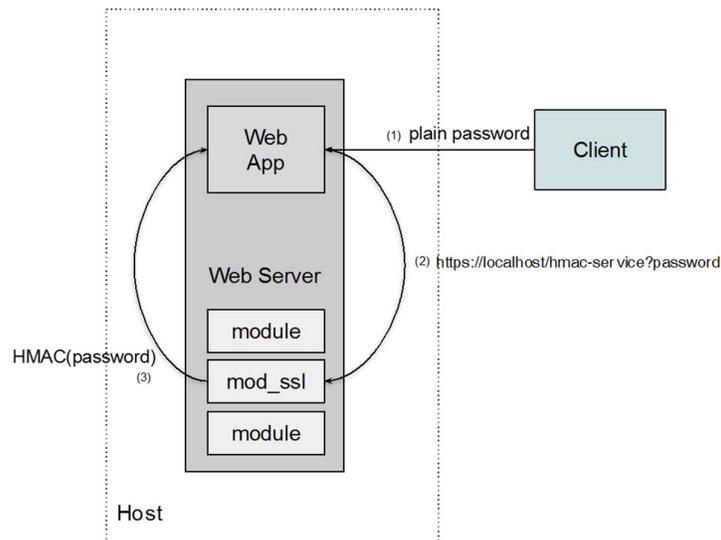


Figure 4. Architecture for credential hardening.

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, as 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 here the part 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);
    ...
}
...
```

Moreover, in **Figure 5**, 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. Thus, 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*)

```

1  int hasher_handler(request_rec *r) {
2      if (strcmp(r->uri, "/hmac-service")==0 && r->args!=NULL &&
3          strcmp(ap_get_remote_host(r->connection, NULL,
4              REMOTE_NAME, NULL),
5              "127.0.0.1")==0) {
6          char * key; server_rec *s = r->server;
7          SSLSrvConfigRec *sc = mySrvConfig(s);
8          modssl_ctx_t *server = sc->server;
9          if (server == NULL || server->ssl_ctx == NULL)
10             return DECLINED;
11         else {
12             EVP_PKEY * evp = SSL_CTX_get0_privatekey(server->ssl_ctx);
13             if (evp) {
14                 size_t len = PRIVATE_KEY_SIZE; key = malloc(len);
15                 FILE *stringFile = fopen(key, "w");
16                 PEM_write_PrivateKey(stringFile, evp, NULL,
17                     NULL, 0, 0, NULL);
18                 fclose(stringFile);
19             } else return DECLINED;
20         }
21         char * plainPassword = getPasswordFromArgs(r->args);
22         int rounds = getRoundsFromArgs(r->args);
23         // wrong password format
24         char * dec=malloc(sizeof(char)*strlen(plainPassword)+1);
25         if (plainPassword==NULL || decode(plainPassword, dec)<0){
26             free(dec); free(key);
27             return DECLINED;
28         }
29         int rlen,i;
30         unsigned char * hashed = HMAC(EVP_sha256(),
31             key, strlen(key),
32             dec, strlen(dec), NULL, &rlen);
33         for (i=1;i<rounds;i++)
34             h = HMAC(EVP_sha256(), key, strlen(key), h, rlen, NULL, &rlen);
35         for (i = 0; i < rlen; i++) {
36             ap_rprintf(r, "%02X", h[i]);
37         }
38         free(key); free(dec); free(plainPassword);
39         return OK;
40     }
41     return DECLINED;
42 }

```

Figure 5. Implementation of credential hardening.

4.2 Storing Textual and Graphical Passwords

Regarding the storage of the textual password, we make a request to the Credential Hardening component, which converts the textual password string into a Hash Message Authentication Code 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.

For the processing of the gestures, the mechanism creates a grid of the image containing 100 squares (segments) on the longest side, and then divides the shortest side by the same scale¹. Rounding was not applied to any decimal segments, and we allowed .25 segments size overflow at the rightmost side of

¹ Microsoft™ Picture Password blog – bit.ly/2SajCDO

the image. The approach of creating a grid of 100 squares allows for storing the gestures based on their segment position on the grid rather than the coordinates in pixels. For each gesture, the following data are 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/counter-clockwise).

The mechanism allows for a tolerance distance in terms of the coordinates on the grid (36 segments around each initial selected segment are acceptable¹ [27], 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 will compare 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.

Although the approach adopted by Microsoft's PGATM for storing the graphical passwords remains undisclosed [14], 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 [14]. 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.

x1: The x coordinate of the tap inside the image grid.

y1: 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.

x1: The x coordinate of the circle's center inside the image grid.

y1: 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.

x1: The x coordinate of the line's starting point inside the image grid.

y1: The y coordinate of the line's starting point inside the image grid.

x2: The x coordinate of the line's ending point inside the image grid.

y2: 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 centre * 3 combinations for the radius), which correspond to the circle's centre (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 4096 combinations (16*16*16). 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 [31].

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.

5 Use-case Scenarios

The second version of the user authentication system builds on the first version of the system. For the tasks of the first version of the user authentication system, please refer to *D5.2 - Software on the Initial Verified User Authentication System* [2]. The new tasks are the following: *i)* administrator login; *ii)* administrator creates and activates a user account²; *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.

5.1 Administrator Login

The administrator login page aims to assure that an administrator³ (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 6**). 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.

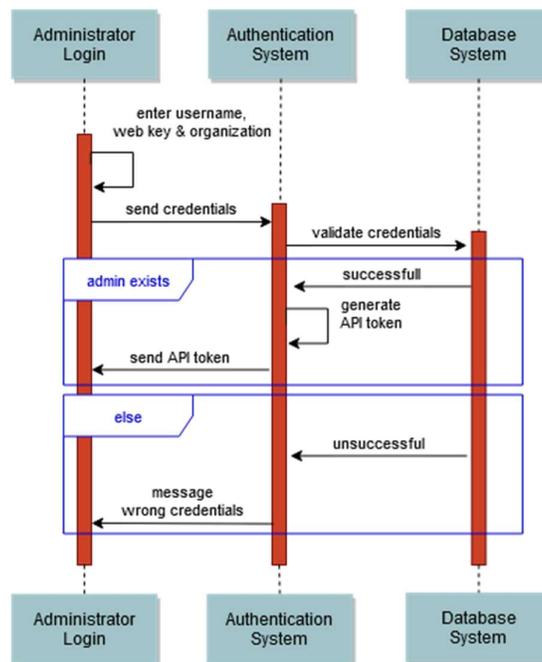


Figure 6. Administrator login.

² In *D5.2*, an end-user could create an account, however, due to requirements' changes, the user account is now created by the system administrator.

³ 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 7). 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.

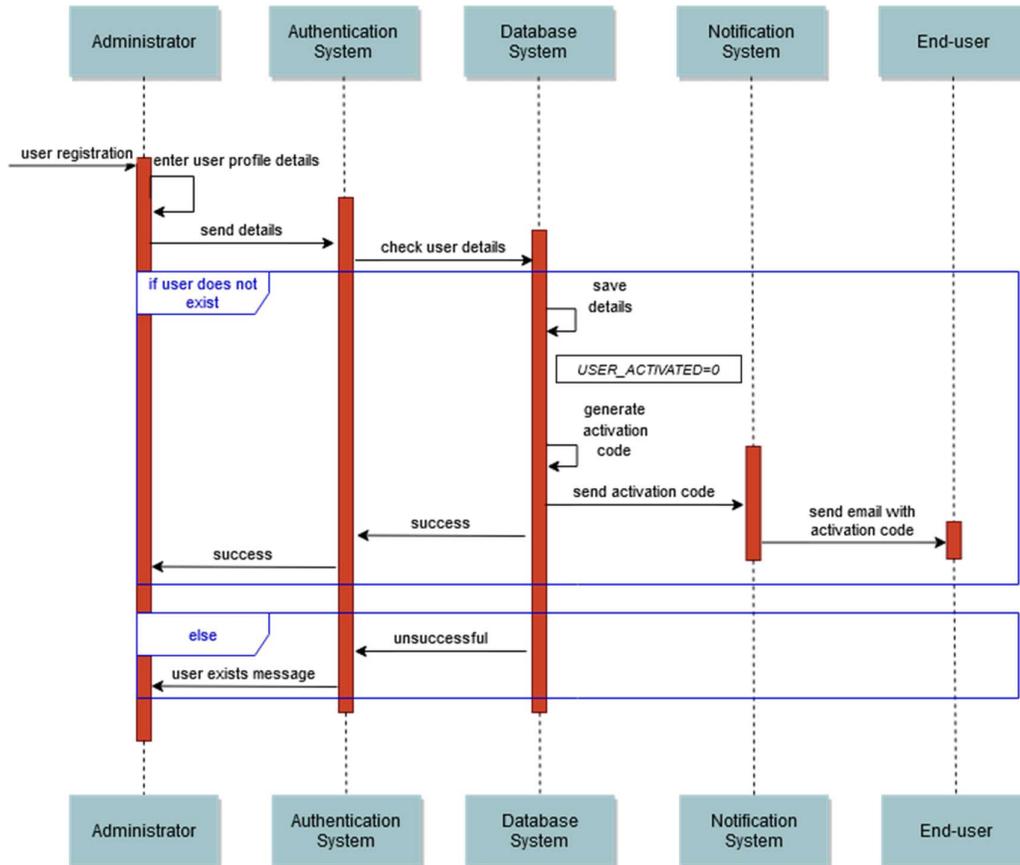


Figure 7. End-user registration and account activation by administrator.

5.3 End-user Activates Account

In this step, the end-user activates the account that was created by the administrator (Figure 8). 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.

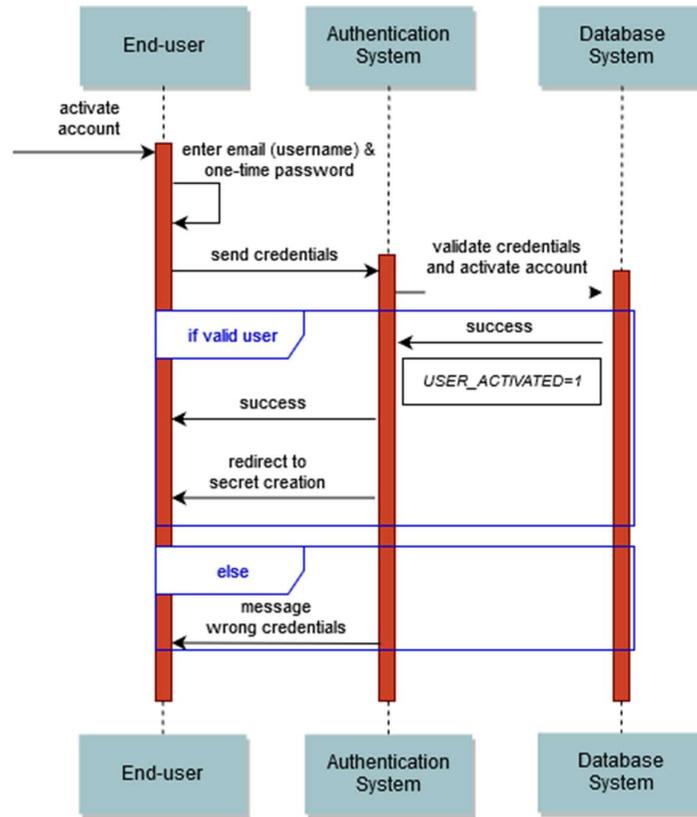


Figure 8. User account verification and activation.

5.4 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 the following 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 graphical password key, users may also (optionally) create a textual passphrase secret they wish (including minimum 16 characters). **Figure 9** illustrates the sequence diagram for the creation of graphical password, and **Figure 10** illustrates the sequence diagram for the creation of a textual password.

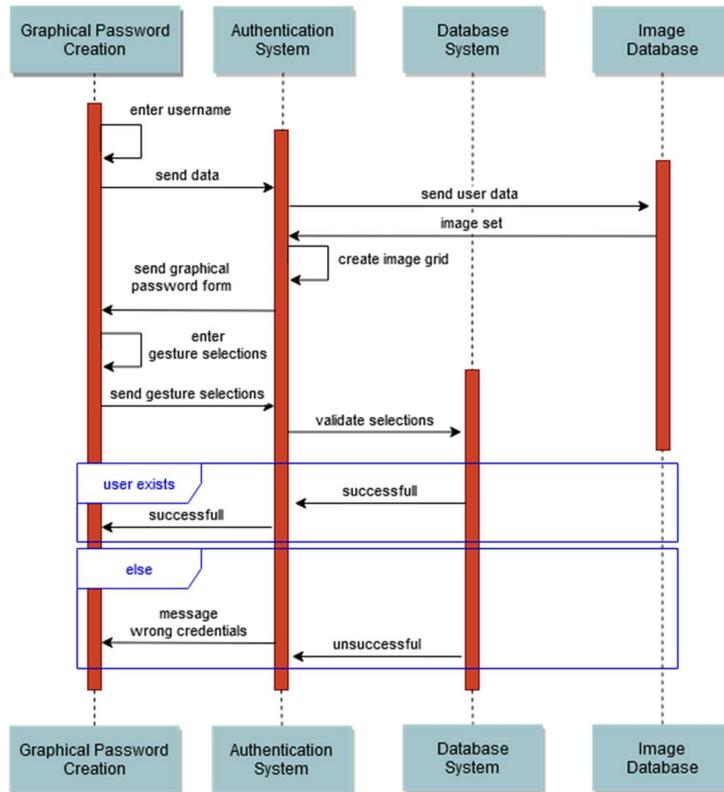


Figure 9. Creation of the graphical password.

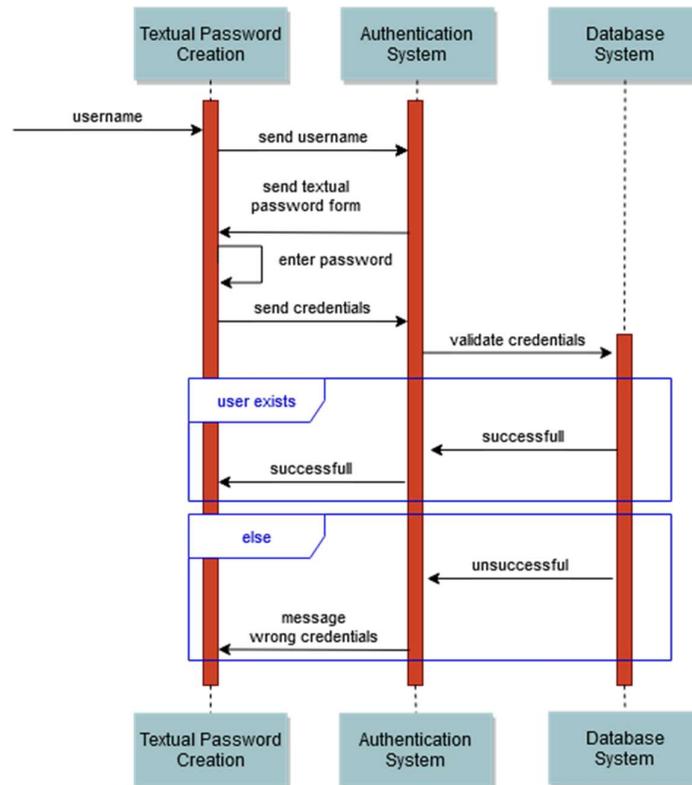


Figure 10. Creation of the textual password.

5.5 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 11** 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 12** illustrates the sequence of interactions for generating the enrollment and QR codes.

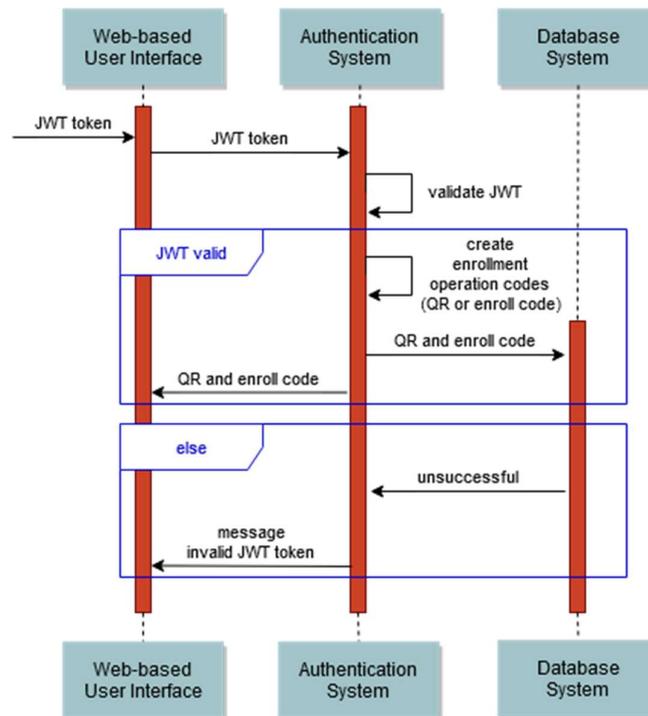


Figure 11. Creation of the textual password.

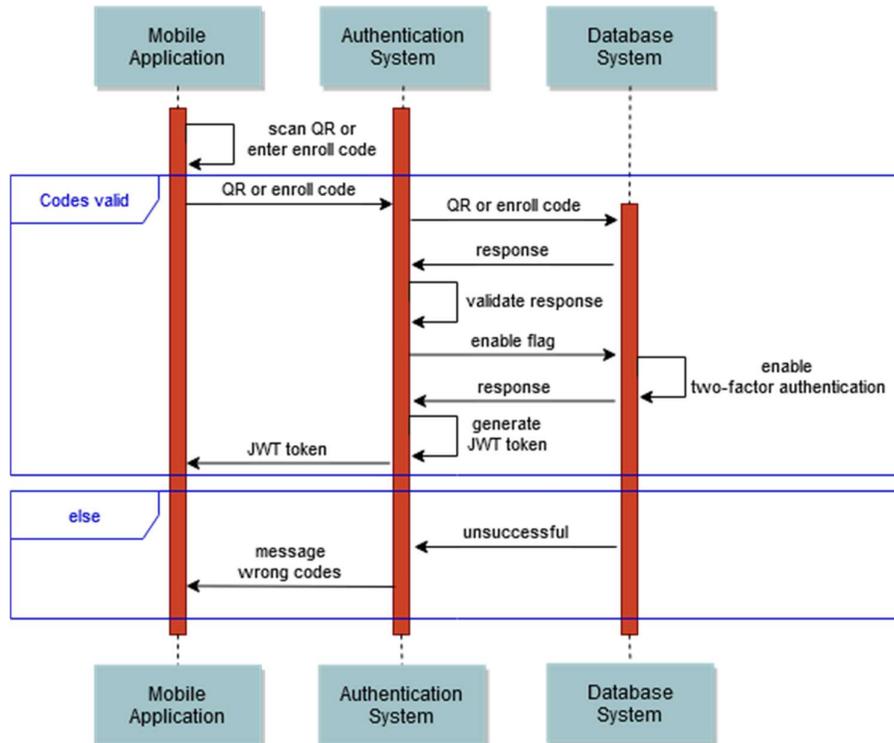


Figure 12. Pairing the user’s device based on the enrollment code or the QR code.

5.6 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 13** 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 14**). 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 15**).

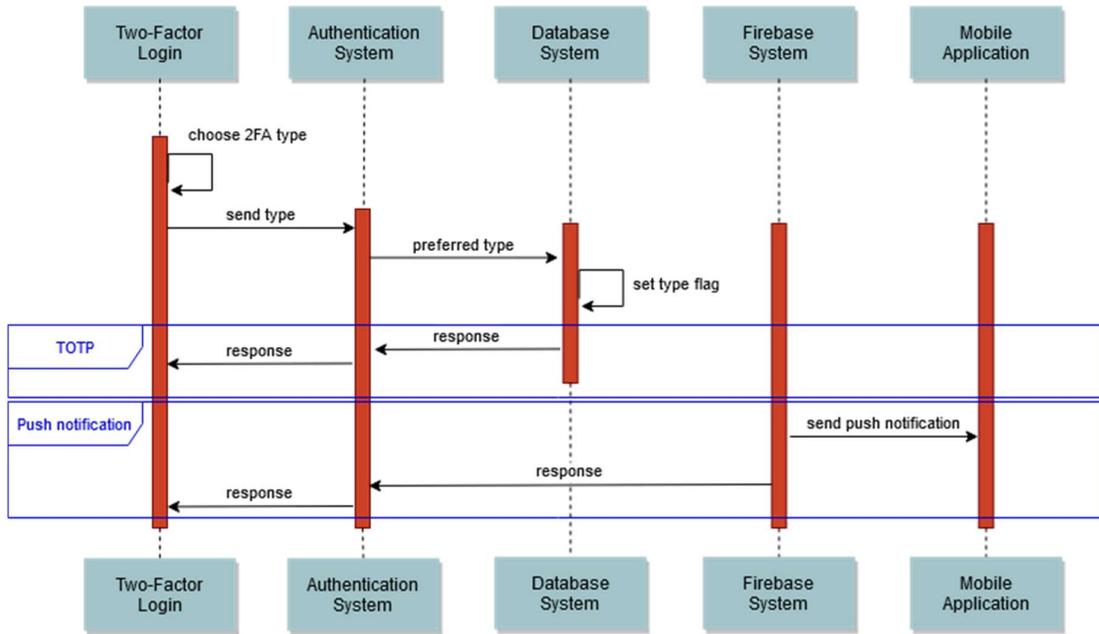


Figure 13. Generation of two-factor authentication login types.

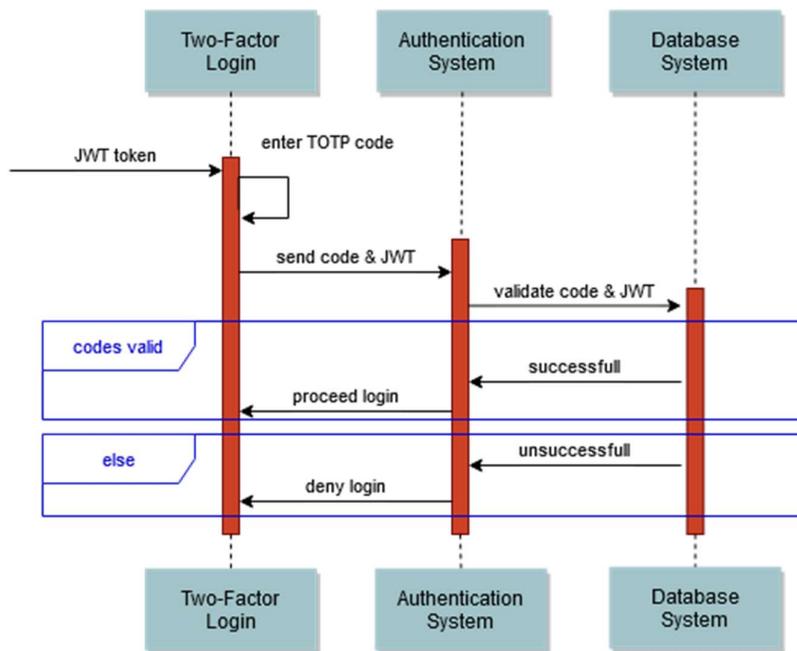


Figure 14. Two-factor authentication login through a time-based one-time password.

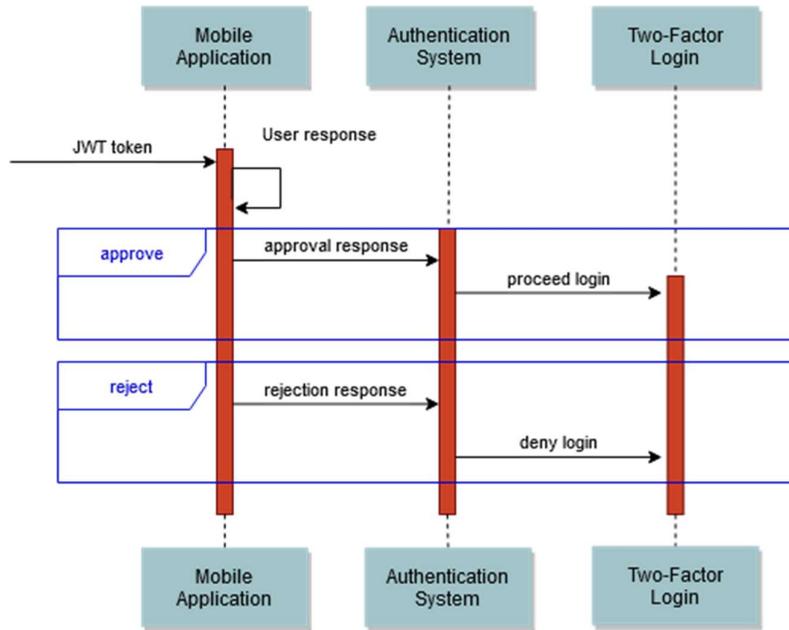


Figure 15. Two-factor authentication login through a mobile-based push notification.

6 Verification of Authentication Properties

For the verification of the User Authentication System, we built a model in a formal language 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 allows us to check that the behavior meets the specification. This is also a significant part of *WP6 - "Integration and Testing"* that will combine the software here with other work packages (particularly WP3 and WP4).

We are using the Uppaal tool [32] that provides an expressive modelling formalism and model checkers allowing us to verify that the properties hold on the model. Uppaal has been used in multiple projects to address similar challenges, for example [33, 34].

Uppaal models are defined as networks of timed automata. An example of an automaton is shown in **Figure 16**.

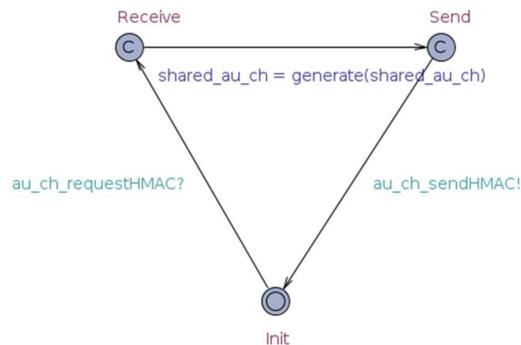


Figure 16. Credential Hardening model.

An automaton can be considered as a graph where nodes are states of the system (e.g., 'Receive' state in **Figure 16** 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 on **Figure 16**; 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 16** 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 16**, 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.

6.1 Model of the User Authentication System

The current version of the model is based on the implementation developed for the second Proof-of-Concept (PoC2). The model of the User Authentication System consists of four automata: Backend, Frontend, and two small automata for Credential Hardening. In addition, we have a model of a patient or doctor communicating with the system.

The model of the Backend is shown in **Figure 17**. 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. All requests are checked for correctness, in particular that the required parameters send via the shared variable are present. Some requests require checking that the user is in a database, some require checking the passphrase or JWT. In case of an incorrect input the component returns into the initial state notifying the requestor through the corresponding channel. In case of the correct input, the automaton performs the required actions and return the result to the requestor. For example, if the component receive a request to create JWT (middle-right part of the model) via *au_pga_create_jwt?* channel (? indicates that Backend is not an initiator of the synchronization) it takes the value from the shared variable and parse it in order to obtain 3 parameters: username, type of passphrase (graphical or text base) and the passphrase.

If the parsing has failed, the component takes the transition with *!checkInput()* guard, which notifies the requestor via *au_pga_incorrect_request!* channel (! indicates that Backend initiates the notification) and returns to the idle state. Similarly, if the username is not in the database, the requestor is notified via *au_pga_no_user!* channel. If both checks are passed (guard *checkInput()* && *hasIdInDb()*), the passphrase is sent to the Credential Hardening component 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 parameter) and either return wrong credentials message or a generated JWT. The API for the second factor authentication is not yet included in the model.

The model for Credential Hardening component (**Figure 16**) 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 generate in the model is returning the input.

The model for the Frontend component is shown in **Figures 18** and **19**. The Signup and Login behaviors are included in this model. Signup starts from the bottom right state by getting the username followed by creation of the graphical password, textual passphrase and a placeholder for the second factor authentication. In **Figure 18**, 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 login procedure also starts with receiving of the username. The Backend component shows which type of identification is available for the user and requesting a passphrase or a graphical password afterwards. In **Figure 19**, the procedure starts from the bottom left state and moves in a counterclockwise direction. In case of successful password check, the Frontend returns a JWT to the user. The Frontend component interacts with both Backend and Users.

Figure 20 illustrates the interaction of a patient with the Authentication system. A sequence of actions showed in the right-top part of the **Figure 20** 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 counter-clockwise direction from the bottom right state.

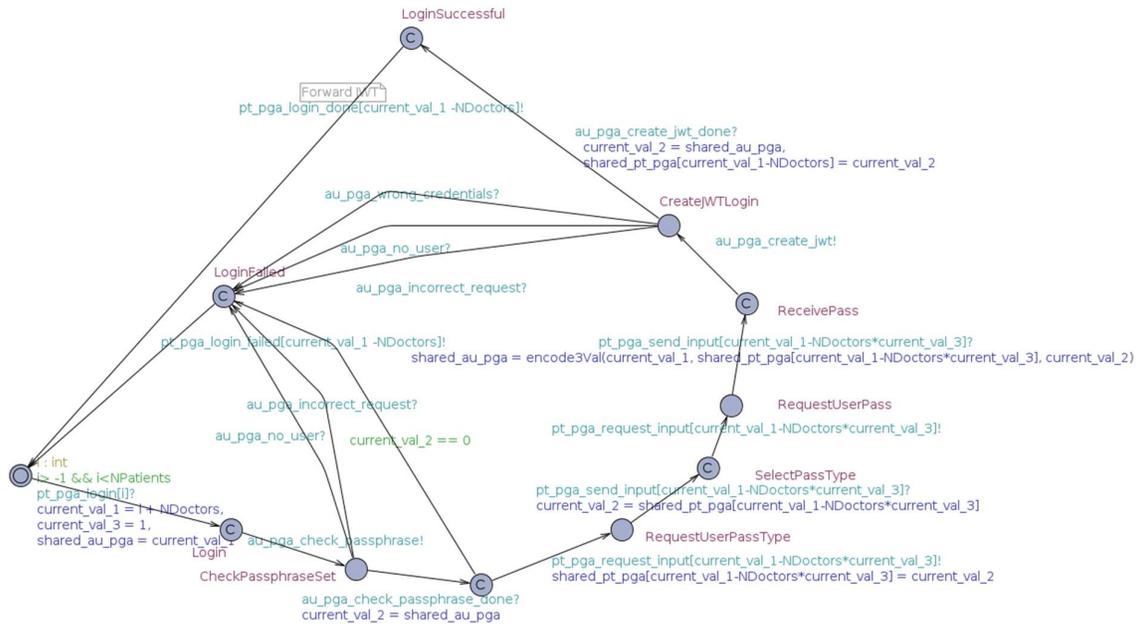


Figure 19. Authentication front-end model: Login procedure.

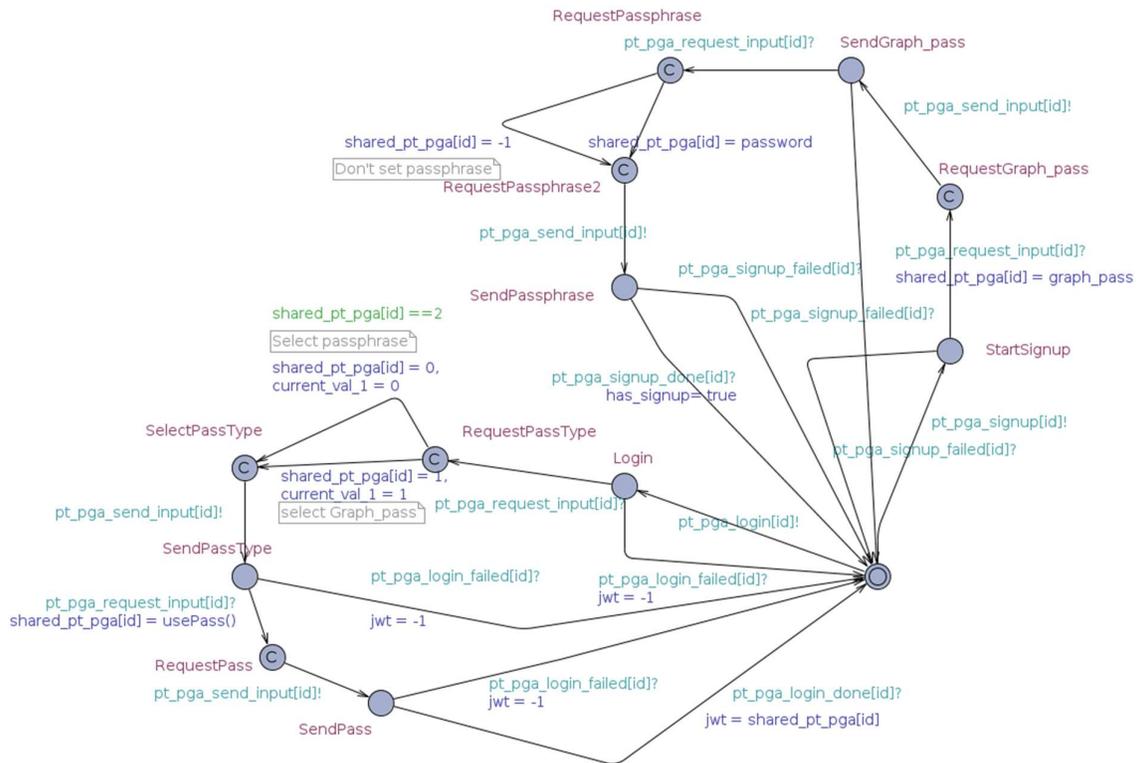


Figure 20. Patient model for authentication system.

6.2 Properties Verification

For the verification of a model, the properties are checked with the Uppaal model checker. The properties shall be expressed in the Uppaal query language based on a simplified version of Timed Computation Tree Logic (TCTL). In the following queries we would use an expression of a form $A[] p$ which means that the property p is required to hold in all states on all execution paths. At this stage we have successfully verified several simple safety properties like:

- 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: $A[] !\text{deadlock}$.
- The user cannot login and receive a JWT if he has not signed up. The property is checked with the following query: $A[] ((\text{Patient.JWT} \neq -1) \Rightarrow \text{Patient.has_signedup})$. 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. A JWT variable is equal to -1 (an initial value) if the patient is not logged in. At the end of the successful login procedure, the Authentication component sends a JWT to the patient that is stored in this variable. In this and the following query, we assume that the patient does not attempt to update the JWT variable without receiving the JWT from the Authentication component.
- The user cannot receive a JWT if he uses incorrect passphrase or graphical pass. The property is checked by minor modification of the Patient model forcing to use an incorrect password. The model checker verifies that such patient never receives JWT: $A[] (\text{Patient.JWT} \neq -1)$. As in the previous property, the JWT variable is updated only after the successful login procedure, therefore the query checks that all login attempts are unsuccessful.
- 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: $A[] (!\text{Backend.jwt_unverified})$.
- A dual to the previous property: a fake JWT fails verification by the Authentication system. The Frontend 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: $A[] (!\text{Backend.jwt_approved})$.

6.3 Next Steps

The model will be further developed incorporating second factor authentication and the developed components will be integrated in the global Serums model. That would allow us to verify properties related to authentication for the whole Serums system, in particular that the generated JWT can be used to authenticate user by other Serums components. For the security properties, attacker will be modelled on the Authentication component level and on the global Serums system level. This will exploit attack models to explore how to violate the correct behaviors of the system and find potential vulnerabilities such as in [35].

The software developed in WP5 will also be tested using fuzzing techniques that can identify unexpected behaviors that would violate correctness and be beyond the capability of some formal methods [36]. Note that the approaches used in [36] also incorporate testing of the smart contracts that use the authentication from WP5.

The validation of the metrics described in D7.4 “*Guessability*” and “*Password cracking rate*” will also be checked by implementation of the defined algorithms to ensure conformance. These will be validated to be implemented correctly and that authentication requires adequate security metrics from users of the Serums software developed in WP5.

7 Conclusions

The aim of this deliverable *D5.3. - “Software on the Refined Verified User Authentication Scheme”* is to report the outcome of the design and development of the refined software of the user authentication scheme. This includes the improved authentication paradigm based on a novel retrospective approach in graphical passwords, the refined general architecture design, the development details of the credential hardening component, the sequence diagrams of new use-case scenarios of the user authentication scheme, the design of the front-end prototypes of the second user authentication system, the results of the verification of the authentication properties, 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 *refined API and the underlying database* will be used as input in D2.5 and 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 refined authentication *architecture, APIs and database* will be used as an essential input in D6.2 for integrating the authentication system in the overall Serums’ smart healthcare system software. The refined *user interface designs of the user authentication tasks* will be evaluated as part of D7.5 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 D5.3 as a whole (and the forthcoming results of the second evaluation studies in WP7) will be used for the final development cycle of the user authentication scheme for *D5.4. - “Report on Final User Authentication System”*.

References

- [1] 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)
- [2] Deliverable 5.2 - Software on the Initial Verified User Authentication System (2019). Deliverable of EU Horizon 2020 Grant 826278 "Securing Medical Data in Smart Patient-Centric Healthcare Systems" (Serums)
- [3] 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
- [4] 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
- [5] 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
- [6] 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
- [7] 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
- [8] Constantinides, A., Fidas, C., Belk, M., 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
- [9] Thorpe, J., Al-Badawi, M., MacRae, B. and Salehi-Abari, A., 2014. The presentation effect on graphical passwords. In proceedings of the SIGCHI conference on human factors in computing systems (pp. 2947-2950)
- [10] Van Oorschot, P.C. and Thorpe, J., 2011. Exploiting predictability in click-based graphical passwords. *Journal of Computer Security*, 19(4), pp.669-702
- [11] Bulling, A., Alt, F. and Schmidt, A., 2012. Increasing the security of gaze-based cued-recall graphical passwords using saliency masks. In *Conference on Human Factors in Computing Systems* (pp. 3011-3020)
- [12] Aydın, Ü.A., Acartürk, C. and Çağıltay, K., 2013. The role of visual coherence in graphical passwords. In *Proceedings of the Annual Meeting of the Cognitive Science Society* (Vol. 35)
- [13] Alt, F., Schneegass, S., Shirazi, A.S., Hassib, M. and Bulling, A., 2015. Graphical passwords in the wild: Understanding how users choose pictures and passwords in image-based authentication schemes. In *Proceedings of the International Conference on Human-Computer Interaction with Mobile Devices and Services* (pp. 316-322)
- [14] 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).
- [15] Davis, D., Monrose, F. and Reiter, M.K., 2004. On user choice in graphical password schemes. In *USENIX Security Symposium* (Vol. 13, No. 2004, pp. 11-11)
- [16] Mihajlov, M., Jerman-Blažič, B. and Ciunova Shuleska, A., 2016. Why that picture? discovering password properties in recognition-based graphical authentication. *International Journal of Human-Computer Interaction*, 32(12), pp.975-988
- [17] Tullis, T.S. and Tedesco, D.P., 2005. Using personal photos as pictorial passwords. In *CHI'05 extended abstracts on Human factors in computing systems* (pp. 1841-1844)

- [18] Thorpe, J. and van Oorschot, P.C., 2007. Human-Seeded Attacks and Exploiting Hot-Spots in Graphical Passwords. In *USENIX Security Symposium* (Vol. 8, pp. 1-8).
- [19] Chiasson, S., Van Oorschot, P.C. and Biddle, R., 2007. Graphical password authentication using cued click points. In *European Symposium on Research in Computer Security* (pp. 359-374). Springer, Berlin, Heidelberg
- [20] Chiasson, S., Forget, A., Biddle, R. and Oorschot, P.V., 2008. Influencing users towards better passwords: persuasive cued click-points. *People and Computers XXII Culture, Creativity, Interaction* 22, pp.121-130
- [21] Wiedenbeck, S., Waters, J., Birget, J.C., Brodskiy, A. and Memon, N., 2005. Authentication using graphical passwords: Effects of tolerance and image choice. In *Symposium on Usable privacy and security* (pp. 1-12)
- [22] Schaub, F., Walch, M., Könings, B. and Weber, M., 2013. Exploring the design space of graphical passwords on smartphones. In *Proceedings of the Ninth Symposium on Usable Privacy and security* (pp. 1-14)
- [23] Furnell, S., 2005. Why users cannot use security. *Computers & Security*, 24(4), pp.274-279
- [24] Ahern, S., Eckles, D., Good, N.S., King, S., Naaman, M. and Nair, R., 2007. Over-exposed? Privacy patterns and considerations in online and mobile photo sharing. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 357-366).
- [25] 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
- [26] Biddle, R., Chiasson, S. and Van Oorschot, P.C., 2012. Graphical passwords: Learning from the first twelve years. *ACM Computing Surveys (CSUR)*, 44(4), pp.1-41.
- [27] 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)
- [28] Erez, M., Gati, E., 2004. A dynamic, multi-level model of culture: from the micro level of the individual to the macro level of a global culture. *Applied Psychology*, 53(4), pp.583-598
- [29] Constantinides, A., Fidas, A., Belk, M., Pietron, A.M., Han, T., Pitsillides, A. (2020). 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*, Elsevier (under review)
- [30] Johnson, J., Seixeiro, S., Pace, Z., van der Bogert, G., Gilmour, S., Siebens, L. and Tubbs, K., Microsoft Corp, 2014. Picture gesture authentication. U.S. Patent 8,650,636. Retrieved from <https://google.com/patents/US8910253>
- [31] Komanduri, S., Shay, R., Kelley, P. G., Mazurek, M. L., Bauer, L., Christin, N. & Egelman, S. (2011, May). Of passwords and people: measuring the effect of password-composition policies. In *Proceedings of the sigchi conference on human factors in computing systems* (pp. 2595-2604).
- [32] Uppaal. <http://www.uppaal.org/>
- [33] R. Gu and E. Enoiu and C. Seceleanu. TAMAA: UPPAAL-based mission planning for autonomous agents. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020
- [34] D. Basile, F.D. Giandomenico, and S. Gnesi. Statistical Model Checking of an Energy-Saving Cyber-Physical System in the Railway Domain. In *Proceedings of the Symposium on Applied Computing*, 2017
- [35] 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>
- [36] B. Jiang, Y. Liu, and W. K. Chan. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. *CoRR*, abs/1807.03932, 2018.

ABBREVIATIONS

2FA	Two-Factor Authentication
API	Application Programming Interface
DRF	Django REST Framework
HMAC	Hash-based Message Authentication Code
HTTP	HyperText Transfer Protocol
KVM	Kernel-based Virtual Machine
MAC	Message Authentication Code
PGA	Picture Gesture Authentication
POC	Proof of Concept
QR	Quick Response
RDBMS	Relational Database Management System
SHA-256	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security
TOTP	Time-based One-Time Password
UCD	User Centered Design
UI	User Interface
UX	User Experience
WSGI	Web Server Gateway Interface

APPENDIX A – Refined Prototype Designs of the User Interfaces

In this section, we provide prototypes of the main User Interfaces (UI) of the second 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 both functional and hedonic aspects.

UI of the FlexPass Homepage and Demonstration Page

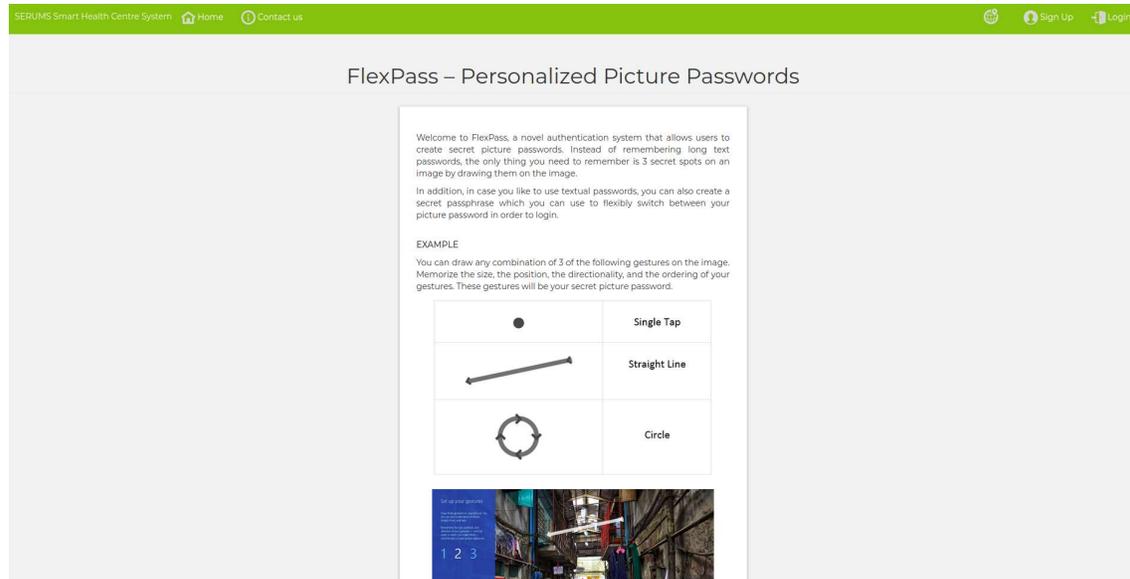


Figure 21. Homepage screen introducing the FlexPass paradigm.

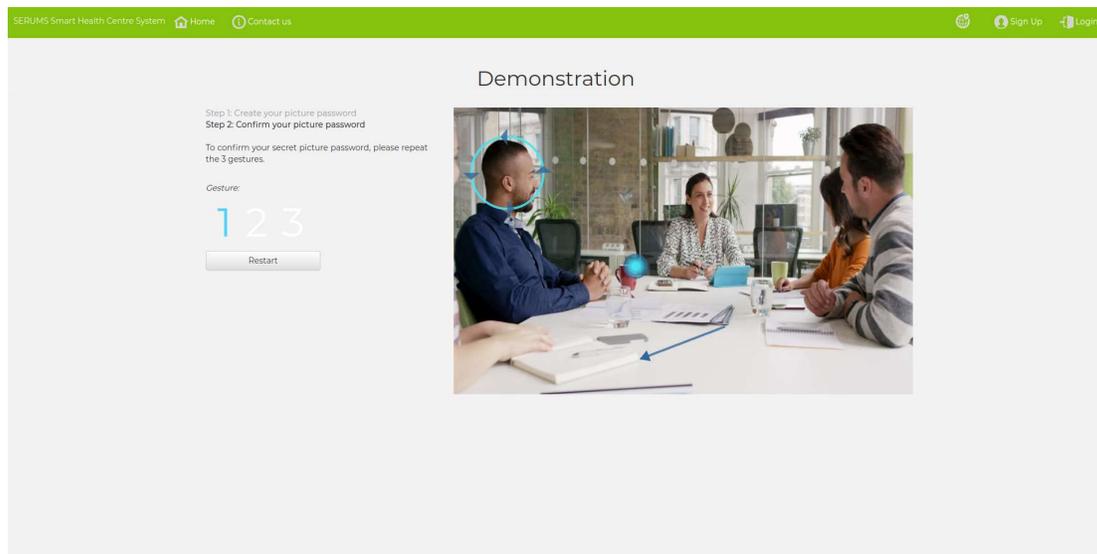
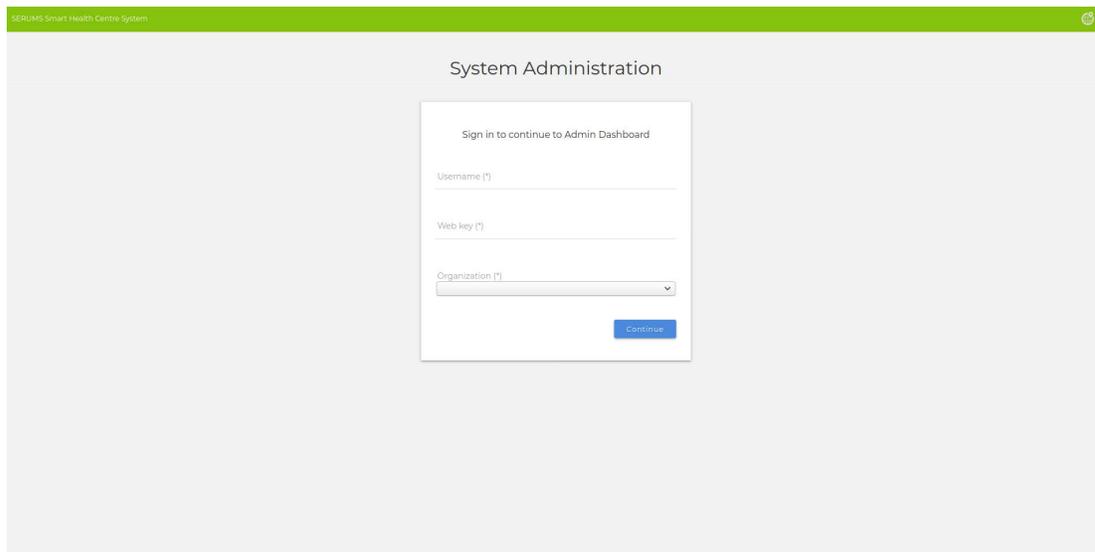


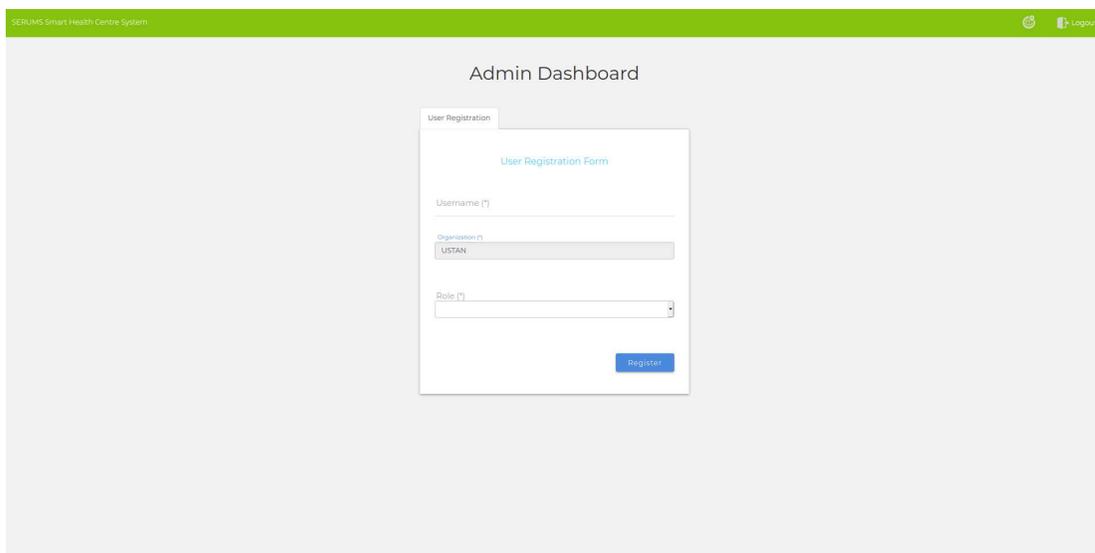
Figure 22. Demonstration page in which users can familiarize with the graphical password creation in the FlexPass system.

UI of the System Administrator's Page



The screenshot shows the 'System Administration' page of the 'SERUMS Smart Health Centre System'. The page features a central login form with the following fields: 'Username (*)', 'Web key (*)', and 'Organization (*)' (a dropdown menu). A blue 'Continue' button is located at the bottom right of the form. The page has a green header bar with the system name and a user icon.

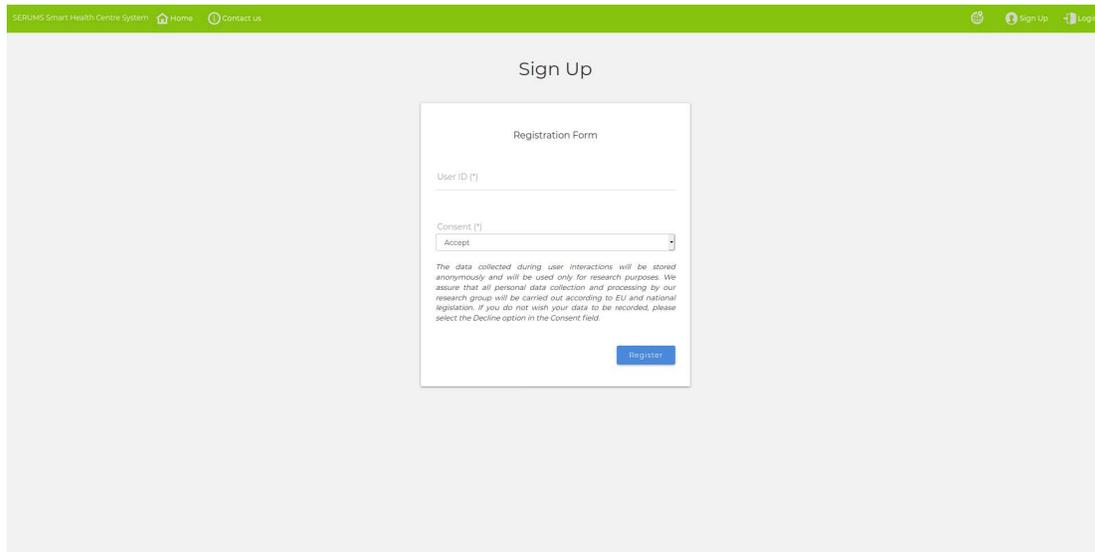
Figure 23. System administrator's login page.



The screenshot shows the 'Admin Dashboard' page of the 'SERUMS Smart Health Centre System'. The page features a 'User Registration' form with the following fields: 'Username (*)', 'Organization (*)' (with 'USTAN' selected), and 'Role (*)' (a dropdown menu). A blue 'Register' button is located at the bottom right of the form. The page has a green header bar with the system name, a user icon, and a 'Logout' link.

Figure 24. 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, hospital administrator).

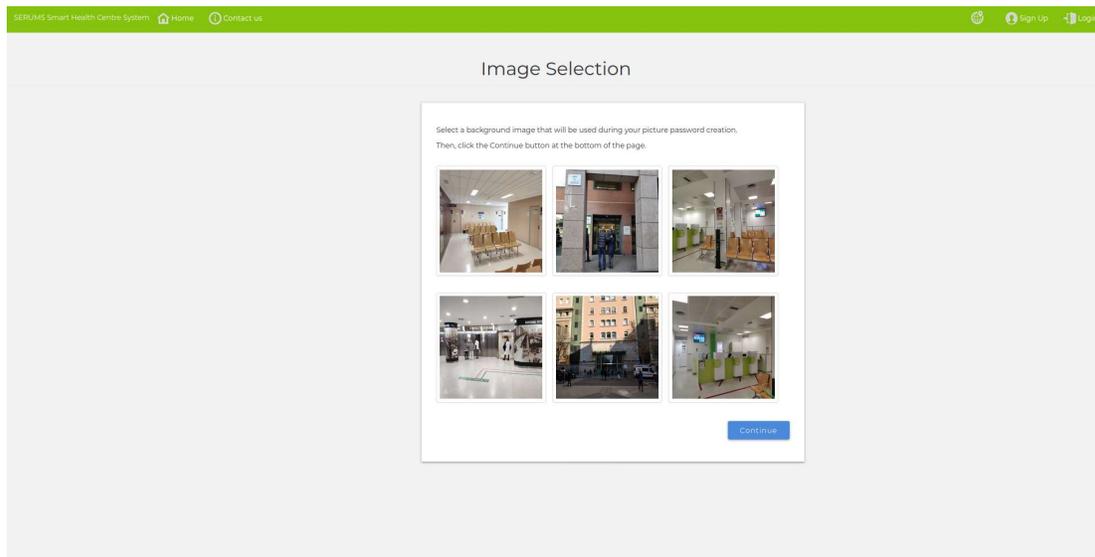
UI of the User Account Registration Page



The screenshot shows a web page titled "Sign Up" with a green header bar containing navigation links for "SERUMS Smart Health Centre System", "Home", and "Contact us", along with "Sign Up" and "Login" buttons. The main content area is a "Registration Form" with a "User ID (*)" text input field and a "Consent (*)" dropdown menu currently set to "Accept". Below the form is a paragraph of text: "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." A blue "Register" button is located at the bottom right of the form.

Figure 25. User account registration page. Once a user account has been created by the system administrator (see **Figure 24**), 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



The screenshot shows a web page titled "Image Selection" with a green header bar containing navigation links for "SERUMS Smart Health Centre System", "Home", and "Contact us", along with "Sign Up" and "Login" buttons. The main content area is a white box with the text: "Select a background image that will be used during your picture password creation. Then, click the Continue button at the bottom of the page." Below the text is a grid of six small images showing various healthcare environments: a waiting area with chairs, a reception desk, a hallway, a pharmacy, a laboratory, and a hospital ward. A blue "Continue" button is located at the bottom right of the image grid.

Figure 26. 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.

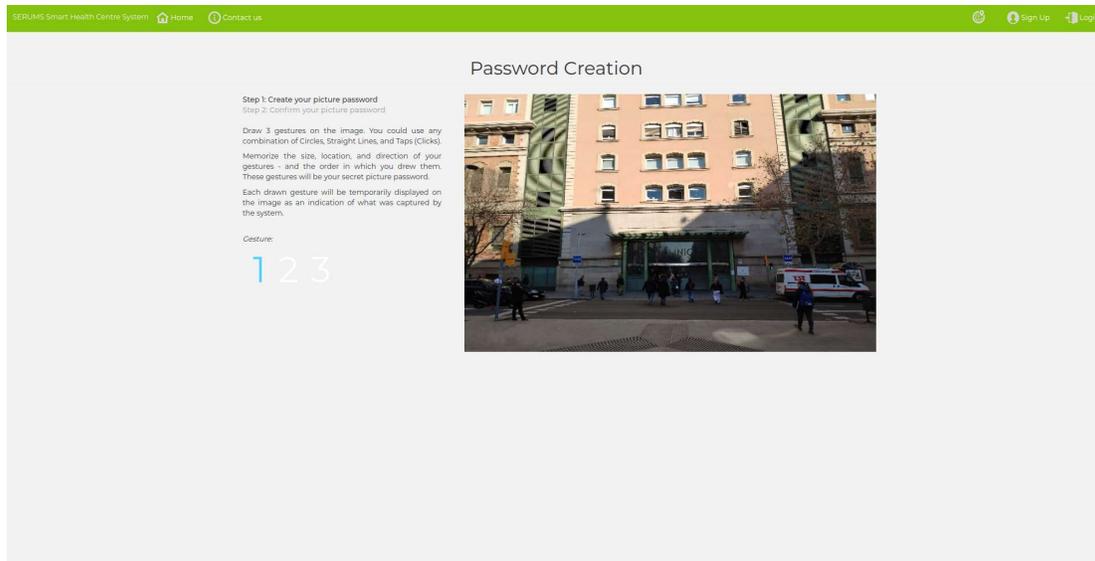


Figure 27. Graphical password creation. End-users create their graphical password by creating a set of secret gestures on the image (gestures can be a combination of tabs, lines and circles).

UI of the Textual Password Creation Page

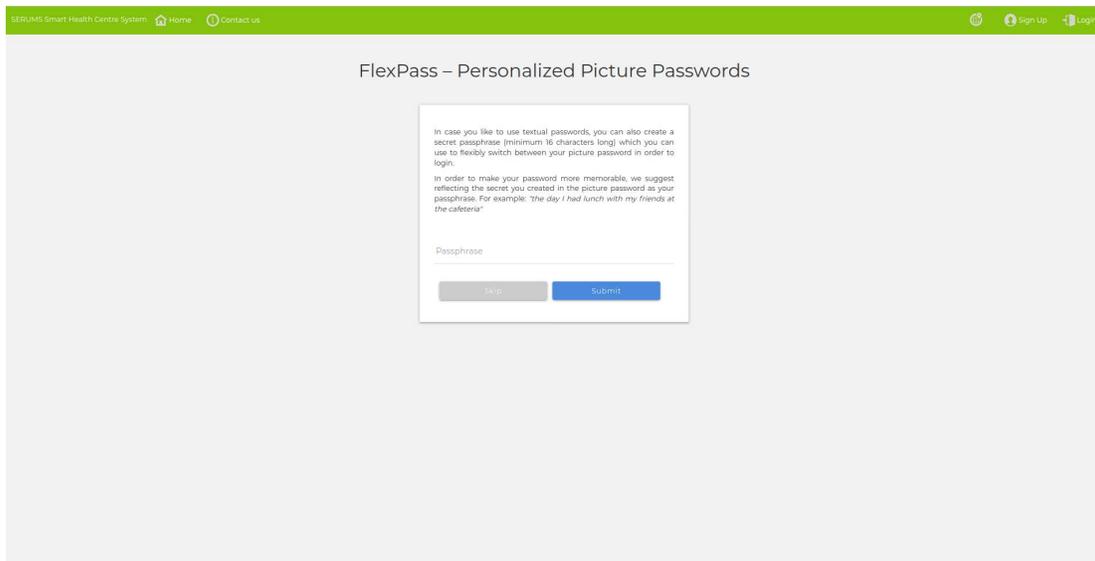


Figure 28. Passphrase creation page. End-users can optionally create a textual passphrase as an alternative type for authentication by reflecting their secret used in the graphical password creation, which can then be used to switch between types of passwords (graphical vs. textual) during login.

UI of the Two-Factor Authentication Activation Page

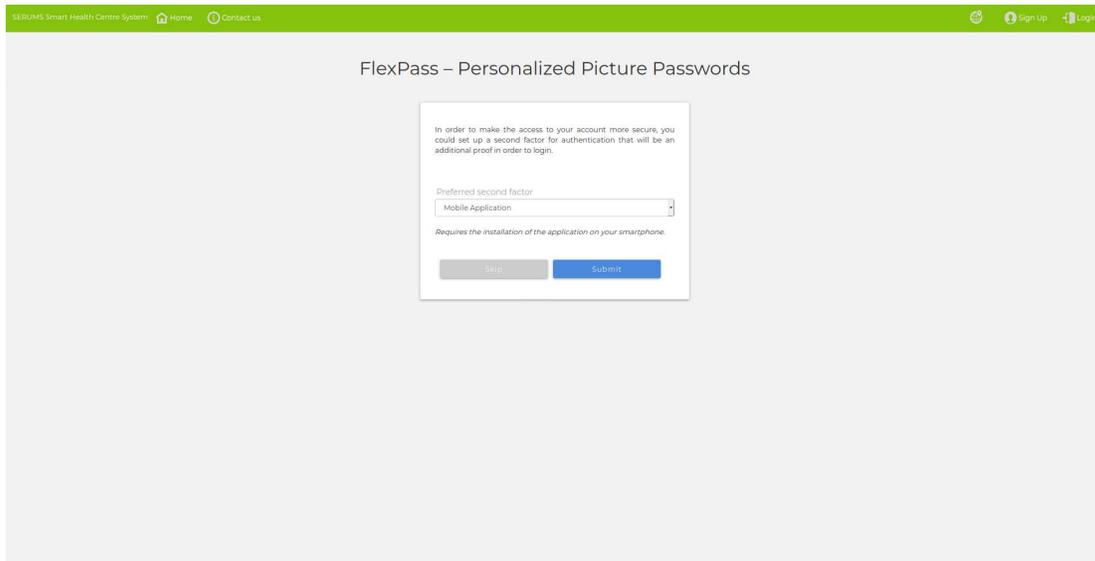


Figure 29. Two-factor authentication activation page. In order to add an additional factor for authentication, end-users can setup two-factor authentication by downloading and installing a mobile application on their smartphone that has been developed for this purpose. The smartphone's mobile application can then be used as a second factor for authentication during login.

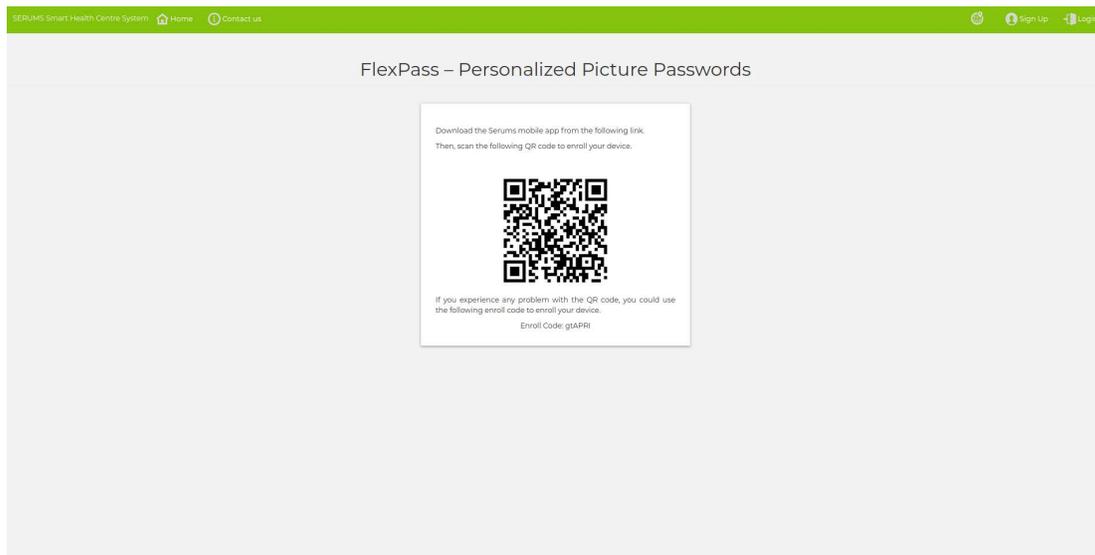


Figure 30. QR code for enrolling the user's smartphone device for two-factor authentication.

UI of the User Login Page

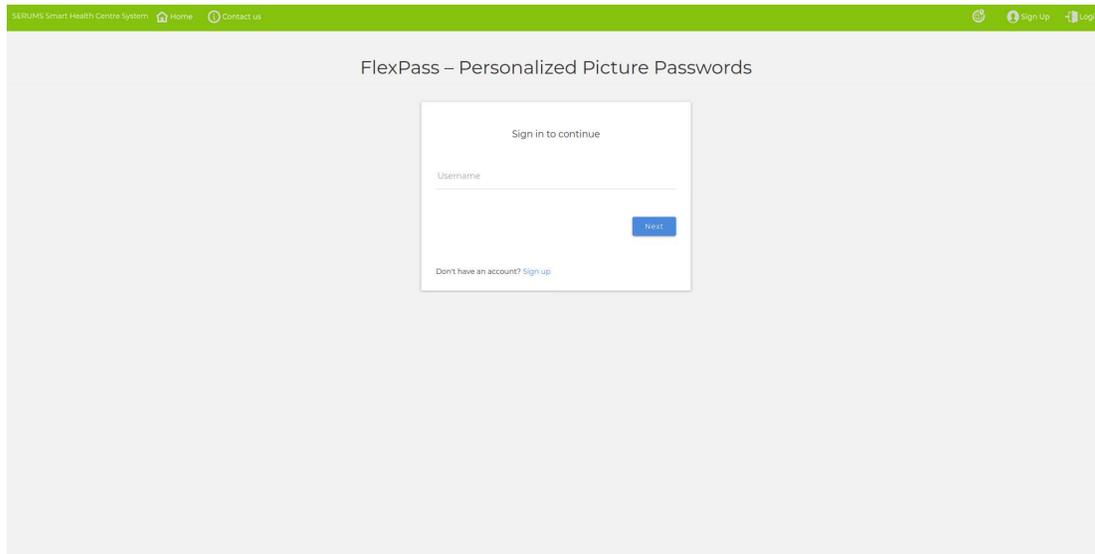


Figure 31. Sign in page based on the end-user's username.

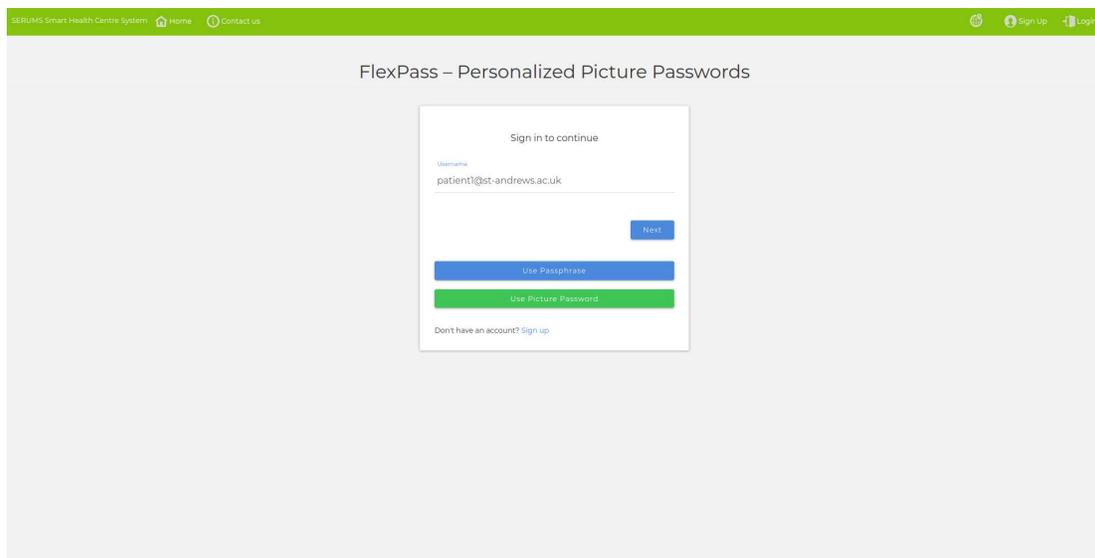


Figure 32. Sign in page in which the users select their preferred authentication type (graphical or textual).

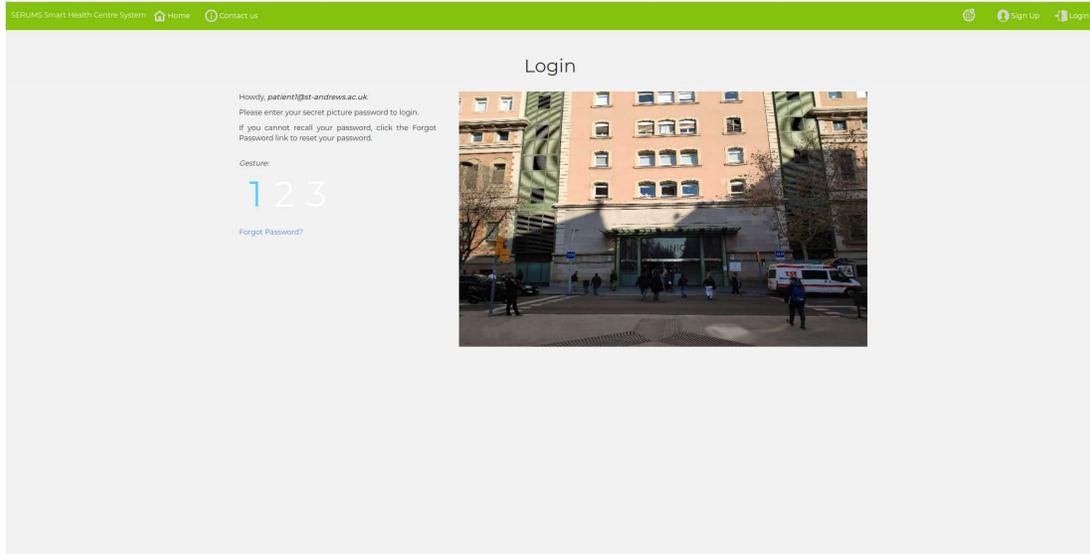


Figure 33. User graphical password login page. End-users enter their graphical password by creating gestures on the image that were setup during the graphical password creation phase.

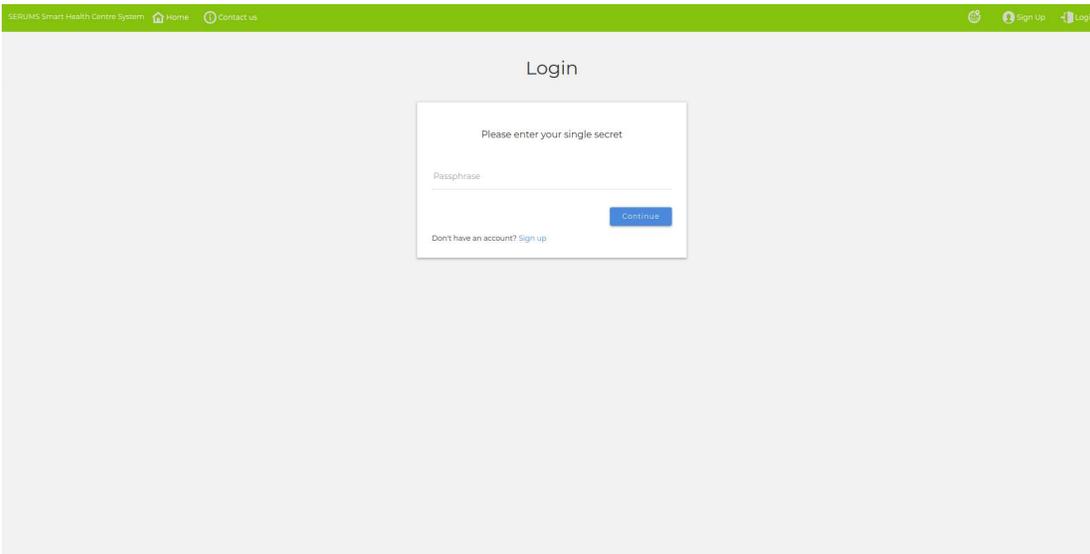
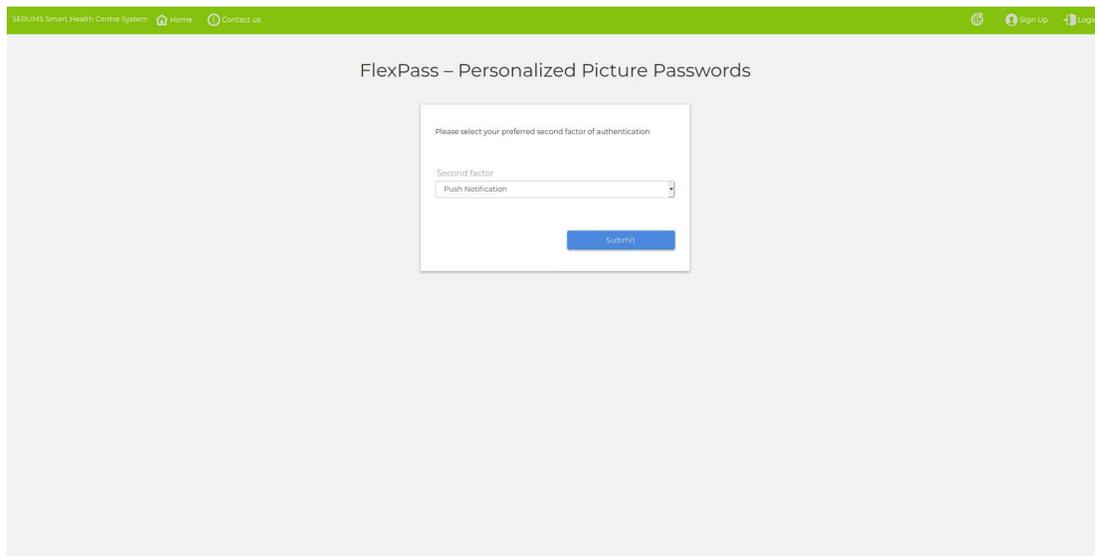


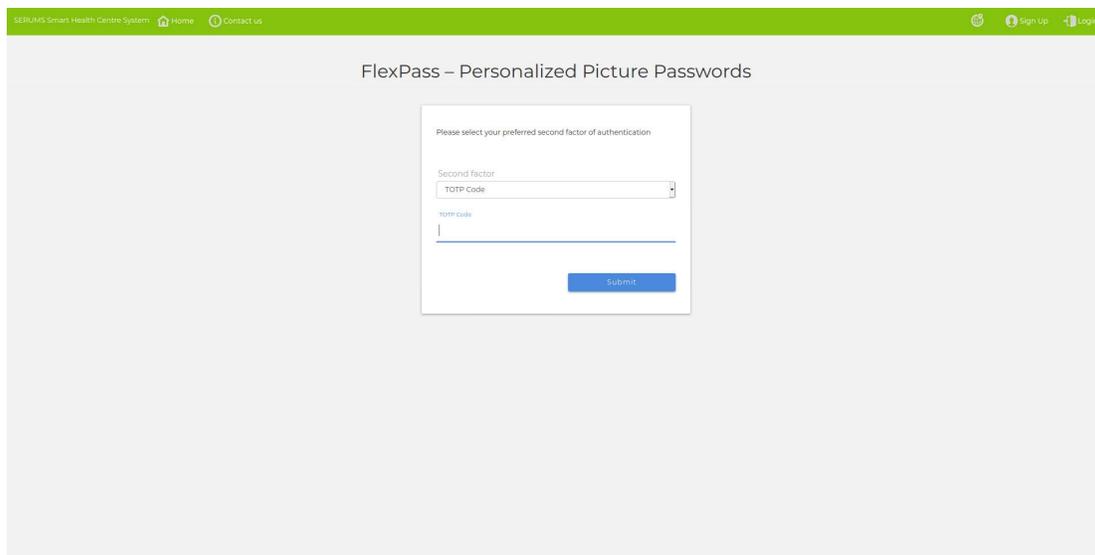
Figure 34. User textual password login page for end-users that have selected the textual password type to login.

UI of the Two-Factor Authentication Login Page



The screenshot shows a web browser window with a green header bar. The header contains the text "SERUMS Smart Health Centre System" on the left and "Sign Up" and "Login" on the right. Below the header, the page title is "FlexPass – Personalized Picture Passwords". The main content area features a white box with the instruction "Please select your preferred second factor of authentication". Inside this box, there is a dropdown menu labeled "Second factor" with "Push Notification" selected. Below the dropdown is a blue "Submit" button.

Figure 35. Two-factor authentication with push notification in which a push notification is sent to the end-user's mobile application for approval.



The screenshot shows the same web browser window as Figure 35. The page title is "FlexPass – Personalized Picture Passwords". The main content area features a white box with the instruction "Please select your preferred second factor of authentication". Inside this box, there is a dropdown menu labeled "Second factor" with "TOTP Code" selected. Below the dropdown is a text input field with a blue underline and a blue "Submit" button.

Figure 36. 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.

UI of the Mobile Application for Two-Factor Authentication

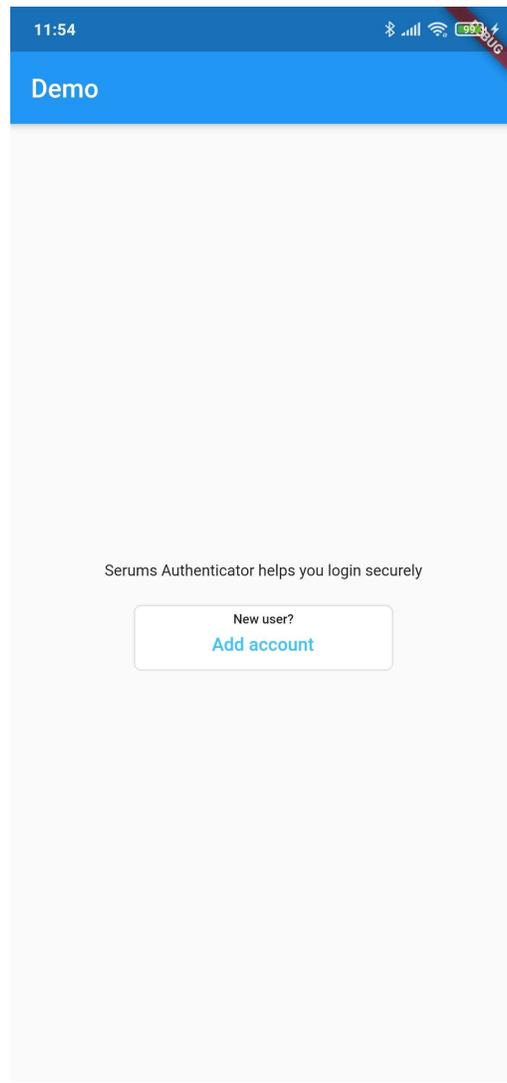


Figure 37. User account creation and enrolment of the end-user's device for two-factor authentication.

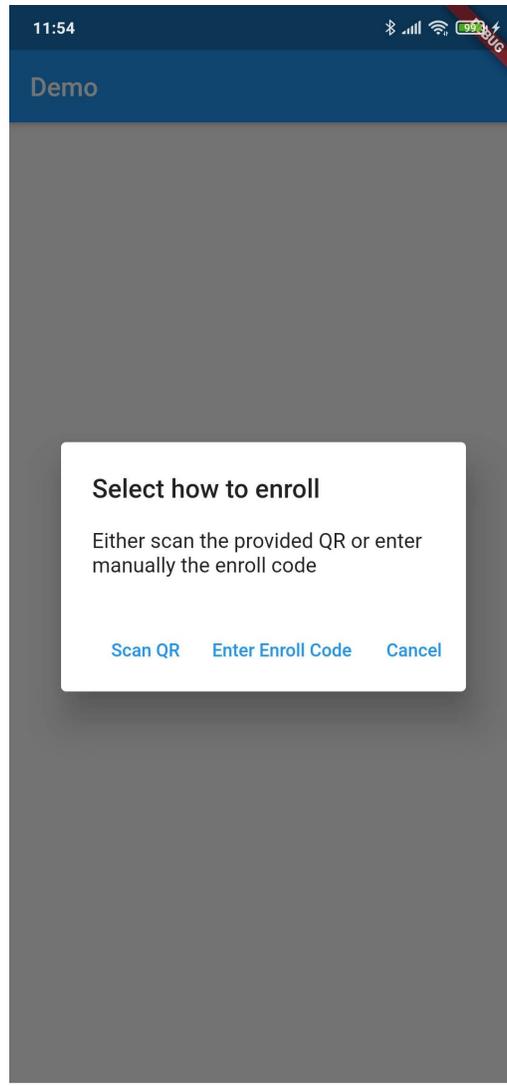


Figure 38. Enrolment with QR code or enrollment code. In case the user selects the QR code option, the mobile application is ready to scan the QR code that is illustrated on the end-user’s Web-based registration system of FlexPass (see **Figure 30**). In case the user selects the enrollment code option, the user has to enter the secret code that is also available on the end-user’s Web-based registration system of FlexPass (see **Figure 30**).

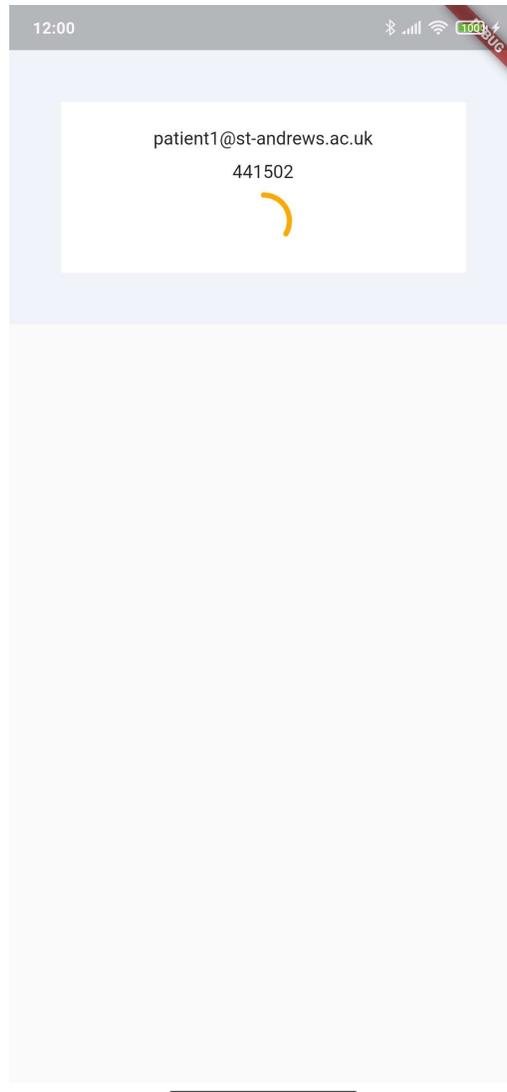


Figure 39. Time-based One-Time Password on the end-user's smartphone mobile application that is automatically reset every 30 seconds. The one-time password can be used by the user during two-factor authentication login.



Figure 40. Push notification for two-factor authentication approval to login.

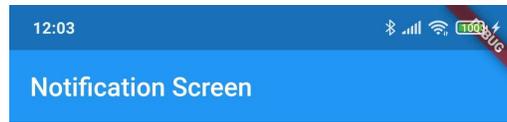


Figure 41. Two-factor authentication approval page. The user either approves or rejects the push notification of the login attempt.

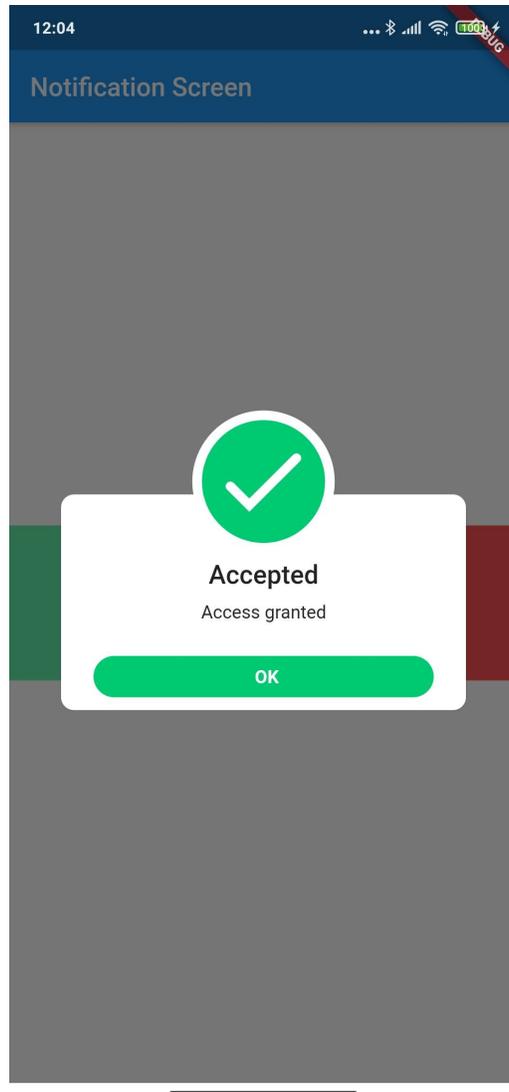


Figure 42. Notification after the user accepts the push notification.

APPENDIX B – RESTful Application Programming Interface

This section lists all the endpoints of the second 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

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": "5604c407f727fedd38e60ddafd2b870a28fe129d", "resource_expires_in_sec": 344944.185673 }

Register Serums User

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 5604c407f727fedd38e60ddafd2b870a28fe129d" -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": 53 }

Check Username

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
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"}'
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Success", "resource_name": "user" }

Set Graphical Password

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) Enum [1 .. 200] characters
time_started_creation *	Integer (Time started creation) [-9223372036854776000 .. 9223372036854776000]
time_finished_creation *	Integer (Time finished creation) [-9223372036854776000 .. 9223372036854776000]
time_first_gesture_started *	Integer (Time first gesture started) [-9223372036854776000 .. 9223372036854776000]
time_first_gesture_fin *	Integer (Time first gesture finished) [-9223372036854776000 .. 9223372036854776000]
time_second_gesture_started *	Integer (Time second gesture started) [-9223372036854776000 .. 9223372036854776000]
time_second_gesture_fin *	Integer (Time second gesture finished) [-9223372036854776000 .. 9223372036854776000]
time_third_gesture_started *	Integer (Time third gesture started) [-9223372036854776000 .. 9223372036854776000]
time_third_gesture_fin *	Integer (Time third gesture finished) [-9223372036854776000 .. 9223372036854776000]
total_time_creation *	Integer (Total time creation) [-9223372036854776000 .. 9223372036854776000]
total_time_creation_with_confirm *	Integer (Total time creation with confirm) [-9223372036854776000 .. 9223372036854776000]
total_time_first *	Integer (Total time first gesture) [-9223372036854776000 .. 9223372036854776000]
total_time_second *	Integer (Total time second gesture) [-9223372036854776000 .. 9223372036854776000]

total_time_third *	Integer (Total time third gesture) [-9223372036854776000 .. 9223372036854776000]
total_failed_attempts *	Integer (Total failed attempts) [-9223372036854776000 .. 9223372036854776000]
total_restart_attempts *	Integer (Total restart attempts) [-9223372036854776000 .. 9223372036854776000]
total_time_creation_task *	Integer (Total time creation task) [-9223372036854776000 .. 9223372036854776000]
timestamp_page_load *	Integer (Timestamp page load) [-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/set_graphical_password/" -H "accept: application/json" -H "Content-Type: application/json" -d '{"username": "user3@ustan.com", "graphical_password": "\#1 T 50,22#2 T 69,32#3 T 95,37", "time_started_creation": 177, "time_finished_creation": 14234, "time_first_gesture_started": 6979, "time_first_gesture_fin": 7311, "time_second_gesture_started": 8039, "time_second_gesture_fin": 8353, "time_third_gesture_started": 10327, "time_third_gesture_fin": 10678, "total_time_creation": 3699, "total_time_creation_with_confirm": 7306, "total_time_first": 332, "total_time_second": 314, "total_time_third": 351, "total_failed_attempts": 0, "total_restart_attempts": 0, "total_time_creation_task": 15710, "timestamp_page_load": 1603879802171}'
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Success", "resource_name": "user" }

Retrieve Graphical Info

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": 5, "image_type": "retrospective" }

Create JWT

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 27,40#2 T 27,40#3 T 27,40\", \"login_type\": \"GRAPHICAL\"}"
Response	
Schema	application/json
Description	JSON Web Token has been created successfully. The value is returned in resource_obj.

Description	Success
Status Code	200
Body	{ "message": "Success", "resource_name": "user" }

Check Passphrase Set

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

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)

total_time_since_page_load *	integer (Total time since page load) [-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_graphical_login_attempt/" -H "accept: application/json" -H "Content-Type: application/json" -d "{\n\"username\": \"test_patient@st-andrews.ac.uk\", \n\"total_failed_attempts\": 0, \"is_reset\": false, \n\"is_reset_from_main_page\": false, \"total_time_until_submit\": 1000, \"total_time_until_successful_login\": 100, \n\"time_interaction_started\": 10, \n\"total_time_since_page_load\": 1100}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{\n \"message\": \"Success\", \n \"resource_name\": \"passphrase\" \n}

Store Passphrase Login Attempt

Endpoint	/store_passphrase_login_attempt/
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
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

	4Siztj_MbTTT6qKKRlp8nGed-j1M" -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	<pre>{ "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/ToA AADiUIEQVR4nO2cW4rcSBBFT0wk6jMLegG9FGInwypdiAtxQs wZH4ajMlf+ZBUbWMw1e7q8o0PgZAOkiDluPFImfN7tvzzmyCI FCISpEiRikU+HmnVBsxsALKZTWxmE8By3cym3O6aPvhtRT4Wib u7M7q7ewruc9xPwd1XgOCMKfjh5vzfafI9ydzX1+yGcsVWF7d WezijGkzn6GtUnd6psjnIlebc4OwGmyDjwlsTC8UI7rfMOU+OTm mzSAP3XM2e/dninwKMrr7DLBcgwPBKe4Tiwhym9jI0X2eKfKpy MXMzK7AmOrB/v1yccgDNuUBYCTp2ce/rciHloseOjQ8lmvAYa2 H5fWb0VakOz1T5HORh9y+LkHuNZaNvtiOLdiRlduLvLXiQ8QVn +NaFE+pFJX6UArIlanGfLo/kQyK7VR/qVhej7IKJg0YCgnxI5Nm6H toGiAkjrgPkK0as5SLIYbWih/LLR76tyEckux5a8RnqoUQwgvscqx 5iT02C9JDik1UfSntUC11TH4Nc86YVxTKRZ6seAfhM7bkWf6k52 C62W1tWPiTyZLumLu5TvaSGthbB2tVZmlrkGzusQ56gzfI99iOF VjhKofiQ9JDIsx1iVPGcnpj2GmNdkYpu0jok8tZ8t5I94emN16KM qjySHhL5U3lxM5tibcofsvw5D1UUzfd+psgnIXtuH2oeP6bj6nPQ 2UVOJ5AeEnm2YxjrAqiq696HTX10KLpimcgb2zX1Tchqxer9FFC/ TORbO9WHel/D+4jHIVdr7RD5kMiT3daH9pRsHx3qt4DqQyJvrf Ttjfh1gOgY1LFGXwycfC0HRgcjpb0+Lm+U+T7kW32118c8svqy+ sKsA0/+iGik4c2IP+5vIPkHyHNXlf2bUE2xRWza3CbCO7/IXH8TXu IRZ6txTIwyFYCIY1fLm4QnGXaDOK3spvDxv/Dah/3tilfkTz2OmQ Gdiw11kljp1PIZSJptuf21PGz47DHPnRWEjb1XEX+jCwpfKo1xqqa MzEq/jGxW+x+ATfd5psgnIQ+zsHWI8b4FKPWaUV+q+sys1iGR3 W58qDbN0mmPlvRJWdUYRf6SLH/7iCvI10OLDcWRbKJeqBXRh3 hbkQ9AvvkPWm2L7dP5XSi1LYuKZSJpVv9htZSiT8DGGQw2q/s W6YXpfHEDyi2f7TtFvh9p+se5SJEiRYoUKflvJ78D2wJlcBrhmclAA AAASUVORK5CYII=", "qr_img_id": "dae881b5c555464192bb11ec5e0410af", "enroll_text_id": "AzqpCp" } }</pre>

Poll Enroll Status

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.eyJ0b2t1b190eXBlljoiYWNjZXRzZiwiZXhwIjoxNjAyODUxMTg2LjQdGkiOiJiOTY3MzRlMTkyM2Y0ZjBhODRkYzg2MGIwN2NiNjY0MSIsInVzZXJJRCI6NTMslmIzcyI6IiNlcnVtc0F1dGhlnRnRyY2F0aW9uIiwiaWF0IjoxNjAyODQyNzI1LCJzdWliOiJ0ZXN0X3BhdGllbnRac3QtYW5kcmV3cy5hYy51aylslmIybnVzSURzljpbllBBVElFTlQixSwib3JnSUQiOiJlVU1RBTiIsImF1ZC16Imh0dHBzOi8vdXJsZGVmZW5zZS5wcm9vZnBvaW50LmNvbS92Mi91cmw_dT1odHRwLTNBX193d3cuc2VydW1zLmNvbSZkPUR3SURhUSZjPWVJR2pzSVRmWFBfeS1ETExYMHVFSFhKdlU4bk9icVLOElyd05LT3RrVIUmcj11VGZONXVRMwto d2JSeV9UZ0tINmFVZDAtQmJtMEc4Sy1WYWpIprelpteTk4Jm09MmVtM4yOUZTYWY3LTAzeHU5eE1CcmNuNHQ2V8zdzN1cUxpTHI0VGZUNCZzPTVqQjJqbXFoc05BX2cxU1Z5WmdVRIJGOW9FUDhfQVFlWxpY1lXM0I1ZncmZT0ifQ.jkE-3Yx9V9vdcyaHYx_n2esgc3RPIZGzIReqe5wvvt4" -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

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.eyJ0b2t1b190eXBlljoiYWNjZXRzIiwiaXhwIjoxNjAyODUxMTg2LjQdGkiOiJiOTY3MzRlMTkyM2Y0ZjBhODRkYzgzMGlwN2NiNjY0MSIsInVzZXJJRCI6NTMslmZcyI6IiNlcnVtc0F1dGhlbnRpY2F0aW9uliwiaWF0IjoxNjAyODQyNzI1LCJzdWliOiJ0ZXN0X3BhdGllbnRac3QtYW5kcmV3cy5hYy51ayIsImdyb3VwSURzljpbllBBVElFTlQiXSwib3JnSUQiOiJlVU1RBTiIsImF1ZCI6Imh0dHBzOi8vdXJsZGVmZW5zZS5wcm9vZnBvaW50LmNvbS92Mi91cmw_dT1odHRwLTNBX193d3cuc2VydW1zLmNvbSZkPUR3SURhUSZjPWVWJR2pzSVRmWFBfeS1ETExYMHVFSFhKdlU4bk9icVLOElyd05LT3RrVlUmcj11VGZONXVRMwto d2JSeV9UZ0tInMhVZDAtQmJtMEc4Sy1WYWprelpteTk4Jm09MmlVTm4yOUZTYWY3LTazeHU5eE1CcmNuNHQ2V8zdN1cUxpTHl0VGZUNCZzPTVqQjJqbXFoc05BX2cxU1Z5WmdVRlJGOW9FUDhfQVFlWxpY1lXMOl1ZncmZT0ifQ.jkE-3Yx9V9vdcyaHYx_n2esgc3RPIZGzIReqe5wvvt4" -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

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

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.eyJ0b2t1b190eXBlljoiYWVWQ3NGU2NDc0M2NIOTZkOGRiMGE0YjlxYzYzMTNiLnVzZXJRCi6NTMslmlzcyI6IiNlcnVtc0F1dGhlnRyY2F0aW9uIiwiaWF0IjoxNjY5ODQyNzI1LCJzZWVhOiJ0ZXN0X3BhdGllbnRac3QtYW5kcmV3cy5hYy51aylslmdyb3VwSURzljpbllBBVElFTlQiXSwib3JnSUQiOiJVU1RBTiIsImF1ZCI6Imh0dHBzOi8vdXJsZGVmZW5zZS5wcm9vZnBvaW50LmNvbS92Mi91cmw_dT1odHRwLTNBX193d3cuc2VydW1zLmNvbSZkPUR3SURhUSZjPWVJR2pzSVRmWFBfeS1ETExYMHVFSFhkdIU4bk9icVLOElyd05LT3RrVlUmcj11VGZONXVRMwto d2JSeV9UZ0tINmFVZDAtQmJtMEc4Sy1WYWprelpteTk4Jm09MmVtM4yOUZTYWY3LTAzeHU5eE1CcmNuNHQ2VV8zdN1cUxpTHI0VGZUNCZzPTVqQjJqbXFoc05BX2cxU1Z5WmdVRlJGOW9FUDhfQVFlWxpY1lXM0l1ZncmZT0ifQ.TbDJ_qvTle8rFEUBVb_Dl2XeJM4LmDVaWpW8vyaqOn8" -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

Endpoint	/verify_jwt/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	
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

