Project no. 826278

SERUMS

Research & Innovation Action (RIA)

SECURING MEDICAL DATA IN SMART-PATIENT HEALTHCARE SYSTEMS

Software on the Initial Verified User Authentication System D5.2

Due date of deliverable: 30th September 2019

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

Reviewers: Thomas Given-Wilson (UCL)
Euan Blackledge (SOPRA)

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	04/03/2019	Marios Belk (UCY), Andreas Pitsillides (UCY)	Defined TOC and added initial Executive Summary
V0.02	05/04/2019	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Added general architecture and initial description on use-case scenarios
V0.03	19/04/2019	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY), Cindy Wings (ZMC), Leon van de Weem (ZMC)	Extended description on use-case scenarios
V0.04	31/05/2019	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Added description of initial APIs
V0.05	28/06/2019	Thomas Given-Wilson (UCL), Marios Belk (UCY)	Added initial description on the verification properties
V0.06	26/07/2019	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Finalized use-case scenarios
V0.07	02/08/2019	Elias Athanasopoulos (UCY)	Added conceptual design of the credential hardening mechanism
V0.08	02/09/2019	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Finalized APIs
V0.09	13/09/2019	Thomas Given-Wilson (UCL), Marios Belk (UCY)	Improved description of the verification of the user authentication system
V0.1	13/09/2019	Marios Belk (UCY), Christos Fidas (UCY), Elias Athanasopoulos (UCY), Andreas Pitsillides (UCY), Thomas Given-Wilson (UCL), Ivo Buil (ZMC), Leon van de Weem (ZMC), SOPRA, IBM	First draft of the deliverable
V0.2	23/09/2019	Marios Belk (UCY), Christos Fidas (UCY), Elias Athanasopoulos (UCY), Andreas Pitsillides (UCY), Thomas Given-Wilson (UCL), Ivo Buil (ZMC), Mark Mestrum (ZMC), Leon van de Weem (ZMC), SOPRA, IBM	Beta version of the deliverable for internal review
V0.3	29/09/2019	Thomas Given-Wilson (UCL), Euan Blackledge (SOPRA)	Version after partners' comments

V0.4	30/09/2019	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY), Ivo Buil (ZMC), Mark Mestrum (ZMC), Leon van de Weem (ZMC)	Pre-final version for final check
V1.0	30/09/2019	Marios Belk (UCY), Christos Fidas (UCY), Andreas Pitsillides (UCY)	Release candidate

SERUMS Consortium

Partner 1	**University of St Andrews**
Contact Person	Name: Vladimir Janjic, Juliana Bowles Email: vj32@st-andrews.ac.uk, 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, Wanting Huang Email: bram.elshof@accenture.com, wanting.huang@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

Table of Contents

Executive Summary

Securing Medical Data in Smart Patient-Centric Healthcare Systems (Serums) is a research project supported by the European Commission (EC) under the Horizon 2020 program. This is the second 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 *"D5.2. Software on the Initial Verified User Authentication System"* describes the outcome and overall methodology that has been applied for the design and development of the initial 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 initial 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 initial software of the user authentication scheme. Specifically, it reports: *i)* the general architecture of the user authentication scheme; *ii)* the conceptual design for credentials hardening; *iii)* the activity flows of primary authentication use-case scenarios; *iv)* the description of the Application Programming Interface (API) and database design of the user authentication scheme; *v)* the initial prototype designs (wireframes and mockup designs) of the user interfaces of the authentication system; and *v)* the initial verification methodology for verifying the user authentication scheme. The outcome of the deliverable constitutes the basis for the development of the final Serums authentication system and its initial evaluation.

1.2 Relationship to Other Serums Deliverables

Deliverable	**Relation**
D2.2: Initial Software for Storage, Access, Blockchain and metadata Extraction for Smart Patient Health Records	The initial API of D5.2 will be used as input in the initial software of the Smart Patient Health Records
D2.3: Report on Refined Specification of Smart Patient Health Record Format	Specifications of D5.2 will be used as input in the refined specification of the Smart Patient Health Records
D4.1: Report on Initial Data Fabrication and Semantic-Preserving Encryption	Characteristics of the database schema of D5.2 will be used as input for data fabrication and semantic-preserving encryption
D5.3: Software on the Refined Verified User Authentication System	The outcome of D5.2 will be used as a basis for the development of the next development cycle of the user authentication scheme
D6.1: Report on Initial Smart Health Centre System Software	The outcome of D5.2 will be used as input for the integrated smart healthcare system software
D7.3: Initial Report on Use Cases and Evaluation	The initial prototype designs of the user authentication scheme that were produced in D5.2 will be used in the context of evaluation studies
D7.4: Refined Requirements Analysis and Success Metrics	The outcome of D5.2 and the evaluation of the initial software of the user authentication scheme will be used as input for the refined requirements analysis and success metrics

1.3 Structure of this Document

The rest of the document is structured as follows: *Chapter 2* describes the general architecture of the user authentication system. *Chapter 3* describes the conceptual design of the credentials hardening mechanism. *Chapter 4* describes the activity flows of important user authentication scenarios. *Chapter 5* presents initial prototype designs of front-end user authentication screens, based on the primacy use-case scenarios defined in Chapter 4. *Chapter 6* describes the verification methodology for validating the authentication scheme. *Chapter 7* concludes the deliverable. *APPENDIX A and B* respectively describe the API of the user authentication system, and the design and development of the database.

2 General Architecture of the User Authentication System

In this chapter we present an overview and the architectural design of the developed user authentication system.

2.1 Overview

Based on the state-of-the-art analyses reported in D5.1 [1], the main objective of WP5 in Serums is to propose a flexible and multi-factor user authentication solution that will combine knowledge-based user authentication types, along with token-based user authentication utilizing push notifications on smartphones and smartwatches. The vision of this work is to combine textual and graphical authentication schemes based on a new flexible user authentication paradigm, which allows us to move from current generic *"one-size-fits-all"* authentication systems to flexible, user-adaptable authentication systems. A first conceptual design of the proposed flexible user authentication paradigm, coined FlexPass [2], is depicted in **Figure 1**. Our approach attempts to provide a new user authentication paradigm that leverages upon theories in Cognitive Psychology (dual coding, episodic and semantic memory), which suggest that humans' episodic and semantic memories, represented as verbal and visual information, can be transformed into memorable and personal authentication secrets. Such secrets can be semantically similarly reflected on both textual and graphical password keys, and accordingly used complimentary based on user preference (**Figure 1**). The paradigm relies on a single, open-ended, user-selected secret that can be reflected as a textual key and a graphical key.

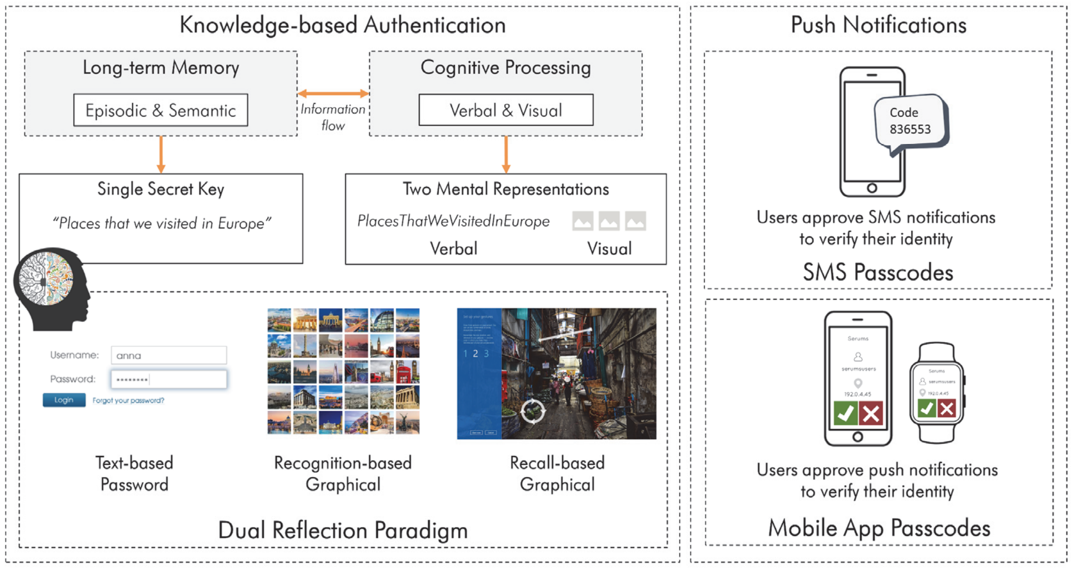


Figure 1. The flexible user authentication concept [1].

2.2 Architectural Design

Figure 2 illustrates the high-level architectural design of the user authentication system. The user authentication API is built and deployed using Docker (version: 18.09.2, API version: 1.39) 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 Apache HTTP Server 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.

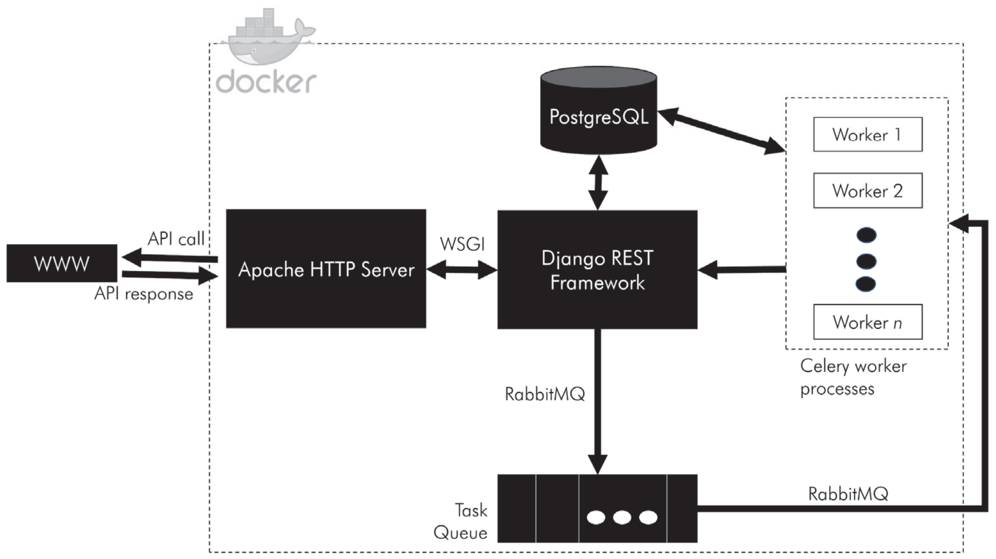


Figure 2. High-level architectural design and technologies used.

3 Credentials Hardening

The wide use of text-based passwords has several consequences, such as difficulties associated with handling a large set of passwords by the users themselves, but, also, and quite importantly, security implications that affect users but cannot be attributed to their faults. For instance, since Websites store user passwords, attackers can leverage site vulnerabilities to exfiltrate them.

Although it is rare to store text-based passwords in plain, but just the cryptographic digest of them, attackers can still use powerful infrastructures to crack the ones that are based on dictionary words (or combinations of them). Leaking the password database has affected quite a few Internet services, some of them being fairly established (Sony, LinkedIn, etc.) and, nowadays, it is estimated that leaked passwords are in the order of several millions.

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

3.1 The need for hardening

Text-based passwords are still the dominant form of user authentication in remote services. Beyond the many usability issues associated with handling several text-based passwords, security is also an important dimension. Through the years, a significant amount of online services has been compromised and their stored passwords have been leaked.

Once the database is compromised, it takes little time for a program to crack the cryptographically hashed (weak) passwords, no matter the algorithm used. In response to this problem, researchers have proposed cryptographic services for hardening all stored passwords. These services perform several sessions of cryptographic hashing combined with message authentication codes. The goal of these services is to coerce adversaries to use them while cracking the passwords. This essentially transforms offline password cracking to online.

In Serums, authentication can be carried out using different methodologies, including text-based passwords. Such credentials, if acquired, can give access to the entire infrastructure to external attackers. Stealing such credentials is often not hard. For instance, credentials can be leaked due to software vulnerabilities (not necessarily related to password handling, but to other parts of the Serums infrastructure), or even through social engineering. It is thus, vital, to ensure that in the unfortunate event of credential leaking, attackers will not be able to access sensitive information.

Protecting *leaked* credentials, *i.e.*, heavily constraint an attacker to crack a (stolen) hashed password, in the sense that is not computationally feasible, is the process we identify in this document as *credential hardening*.

3.2 Core Methodology

Text-based passwords need to be stored for validating future user logins. Although this sounds trivial, it is alarming that several services have failed multiple times to get it right. We leave out of the discussion services that do nothing special for password storage and we discuss other common mistakes.

A common misunderstanding is that using encryption should be enough for securing passwords. Unfortunately, the common attack vector is that passwords can be leaked, and keys used to encrypt passwords can be leaked, as well. Therefore, simply encrypting passwords will not make things better. Instead, a cryptographic hash function should be used, and not an encryption cipher, since the output of such function cannot be reversed by someone that has access to the key.

A second misunderstanding is how to validate an existing password digest. Some services allow the hash computation to be performed at the client-side (for instance, in the Webpage through JavaScript). Users may think that such practice is good, since their password never reaches the Web service, however, the described procedure is dangerous, since this allows attackers replaying password digests, without even trying to actually crack them.

Finally, just hashing the password is not enough, since equal passwords will produce equal digests. A common practice is to use a salt, a random and unique-per-password prefix that, if concatenated to the password, will make the final digest unique. The salt can be leaked as well, but it does not matter. The salt is not meant to protect passwords from cracking, but rather *hiding* known digests and common, between different users, passwords.

Based on the aforementioned observations, in Serums we leverage a password-hardening service which does not use cryptographic hashing for storing passwords, but rather an HMAC (Hash-based Message Authentication Code). The latter involves using the TLS (Transport Layer Security) private key of the Serums server. It is important to stress that:

- We do not prevent password leaks, however, cracking of passwords, once they are leaked, is *only* possible if the TLS private key of the Serums server is also leaked. In this case, we consider that the threat model of password leaks is no longer relevant, since an attacker that has access to the TLS private key can launch much stronger attacks, such as impersonating the server. In the case the TLS private key is leaked then passwords can be re-secured (*i.e.*, HMAC'ed with a new TLS private key) on demand: when a user logs in, the old key should be used to verify that the password is correct, and then the new key should be used to compute the new MAC (Message Authentication Code);

- We offer a high level of security against password cracking without using *any* external password-hardening service and with an overhead of the order of magnitude of adapting hashing (`scrypt` and `bcrypt`), however, without being vulnerable to weak passwords. With credential hardening, even simple passwords that are based on dictionary words *cannot* be cracked.

3.3 Preliminary Architecture

Serums secures a text-based password using a MAC, instead of a cryptographic hash function. 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 3 illustrates an overview of how credential hardening works in the Serums server. In **Figure 3**, we assume that Web App is the front-end (User Interface) of the Serums infrastructure.

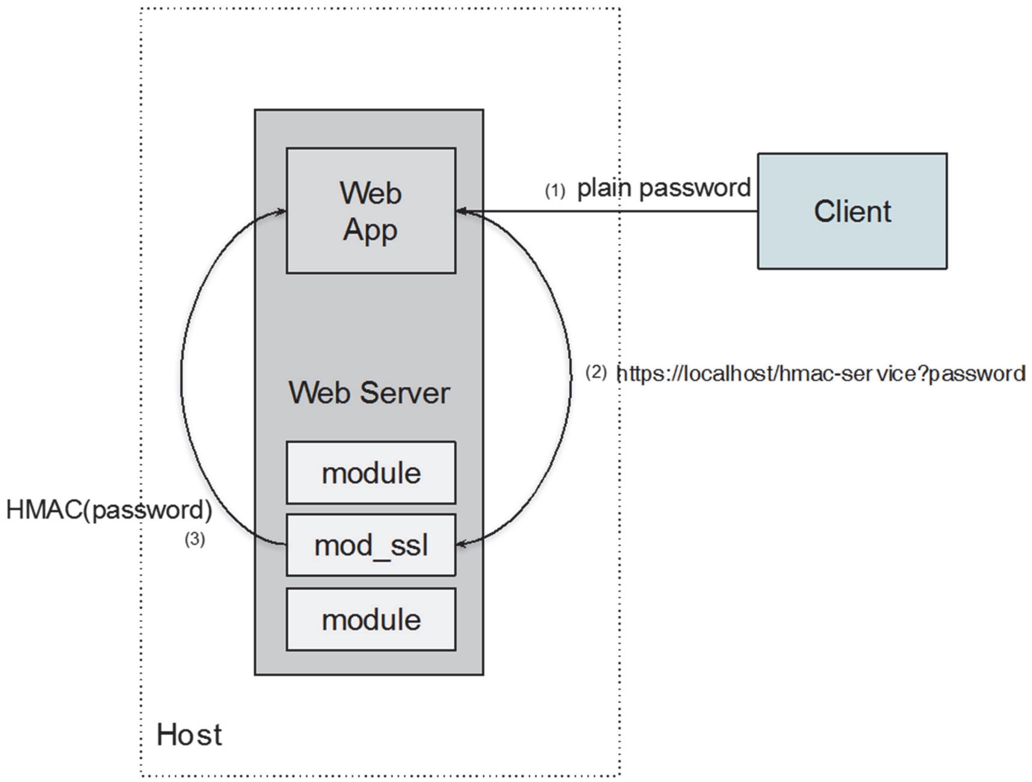


Figure 3. Architecture for credential hardening.

Web clients send their requests to the Serums server (1). Once a request is issued, then the Web application can use cryptographic operations, already available to the Apache process, through the credential-hardening component. These services can generate the HMAC of strings, and therefore the Web application can leverage strong HMACs of passwords (3), that are hard to be cracked offline. In the same fashion, the Serums server can validate an existing HMAC computation for checking existing credentials.

4 Use-case Scenarios

The user authentication system consists of various tasks. For the initial version of the user authentication system, we have focused on the following user tasks: *i)* initial user registration; *ii)* user account verification and activation; *iii)* request to reset secret; *iv)* reset secret; *v)* creation of the single secret and its two reflections (textual and graphical); and *vi)* user-adaptable authentication.

4.1 Initial User Registration

The first step of the user authentication process entails the user registration phase (**Figure 4**). In this phase, the user initially enters the profile details. 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.; *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.

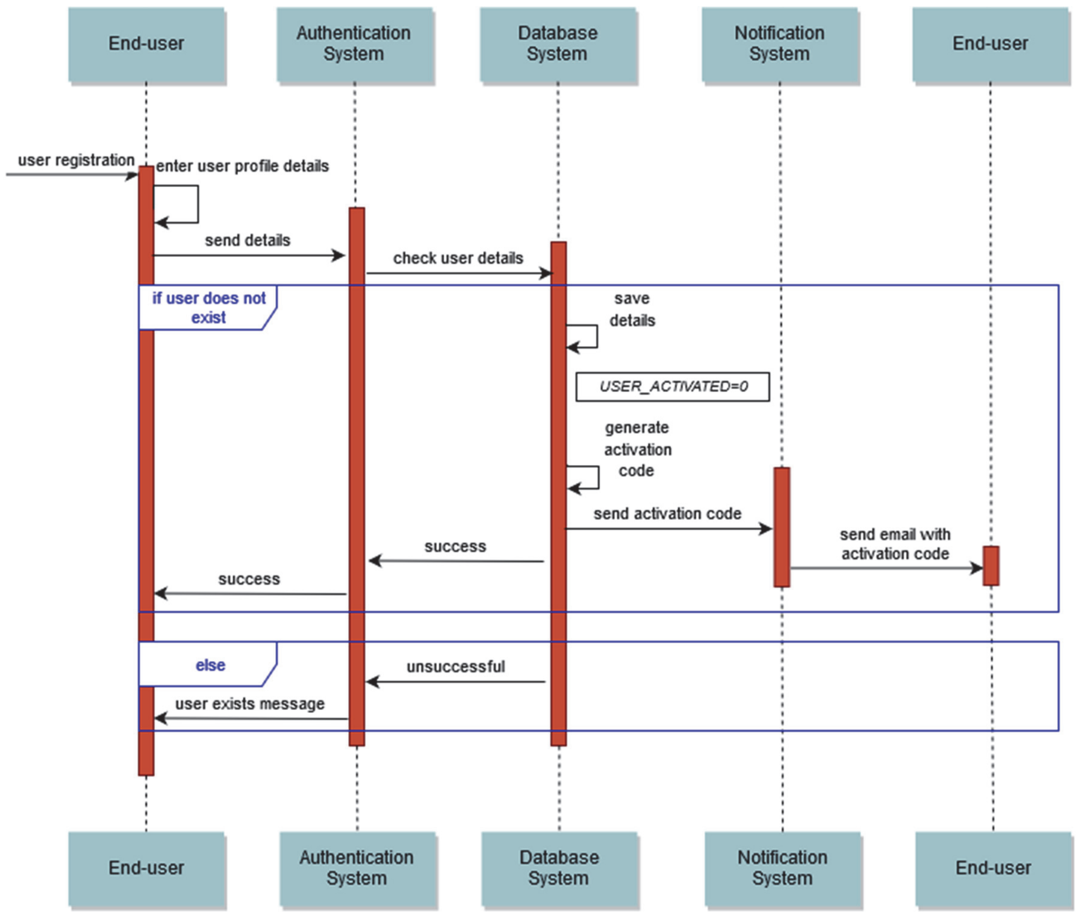


Figure 4. Initial user registration.

4.2 User Account Verification and Activation

The second step of the user authentication process entails the user account verification and activation phase (**Figure 5**). In this phase, 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.

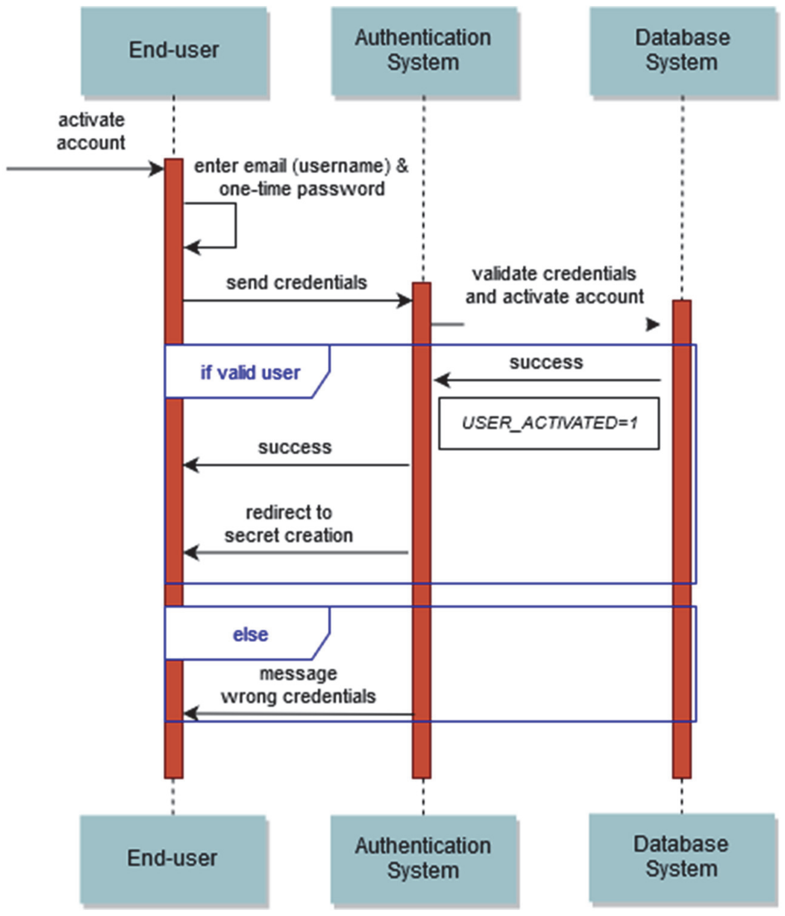


Figure 5. User account verification and activation.

4.3 Request to Reset Secret

The user can request a reset code in order to reset the secret (**Figure 6**). First, the user enters the email. Then, the Authentication System checks the provided input details, leading to one of the following cases: *i)* if the provided email exists in the Database, a reset code is generated and sent to the Notification System. Then, an email including the reset code is sent to the end-user and a success operation is returned.; *ii)* if the provided email does not exist in the Database, an unsuccessful operation is returned, along with a message notifying the user that the provided email is wrong.

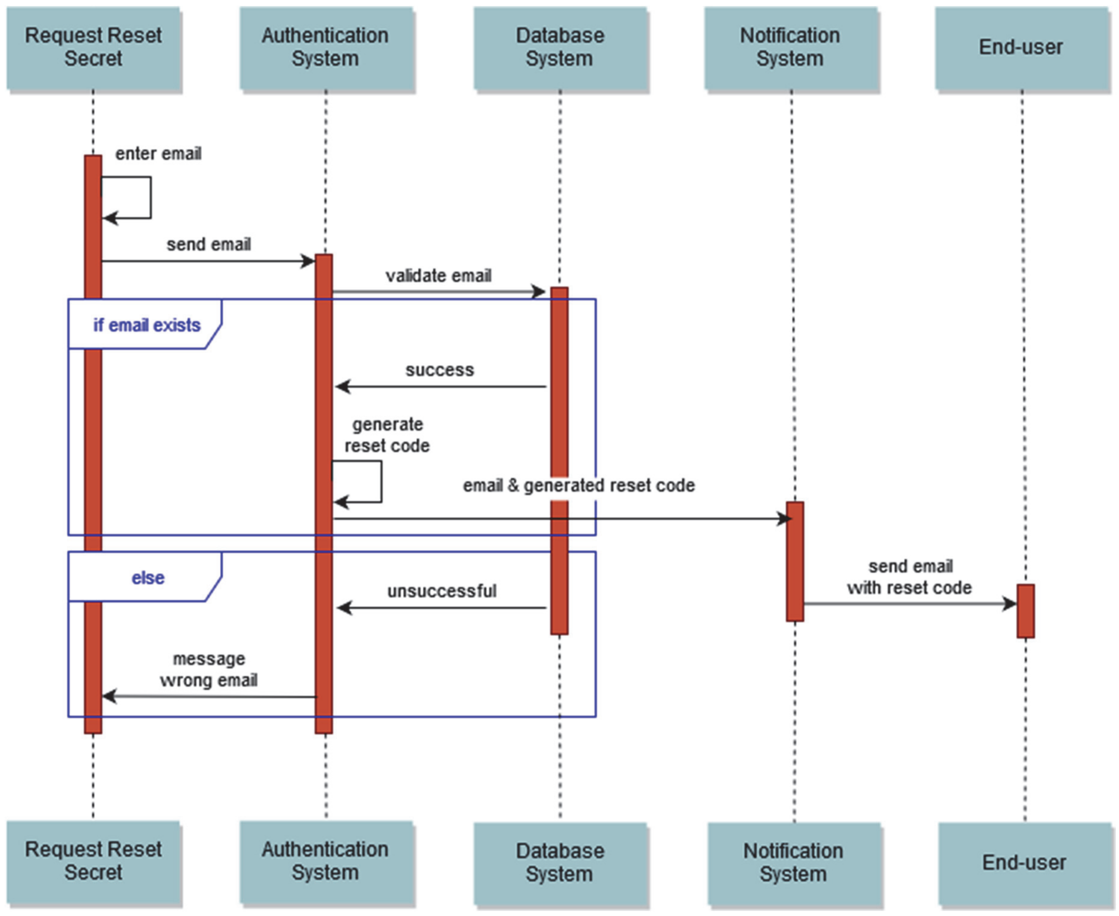


Figure 6. Request reset secret.

4.4 Reset Secret

The phase of reset secret is depicted in **Figure 7**. First, the user enters the email and the reset code received by email. Then, the Authentication System checks the provided input details, leading to one of the following cases: *i)* if the provided input details are valid, the user is allowed to reset the secret, a success operation is returned and the user is redirected to the secret creation page.; *ii)* if the provided input details are not valid, an unsuccessful operation is returned, along with a message notifying the user that the provided credentials are wrong.

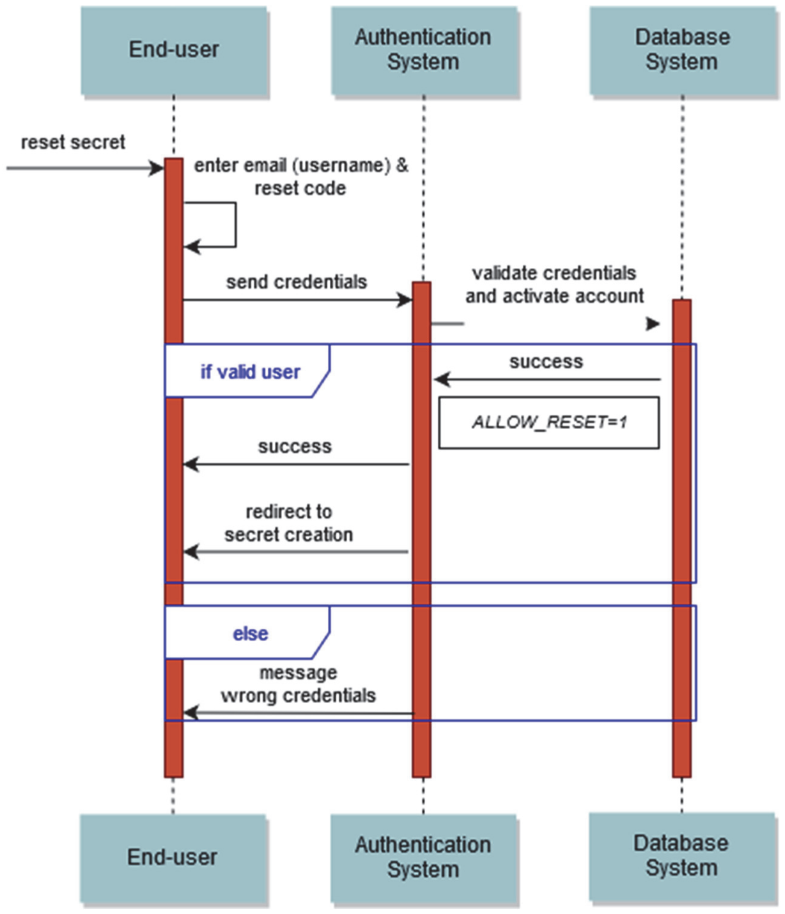


Figure 7. Reset secret.

4.5 Creation of the Single Secret and its Two Reflections

The secret creation phase is split in three main steps as follows. Users initially type a single secret they wish, *e.g.*, *"Places that I visited in Europe in my childhood"*. Next, the system generates a textual password key based on the user-given single secret, *e.g.*, *"PlacesThatIVisitedInEuropeInMyChildhood"*. Users are then allowed to slightly modify the text, *e.g.*, change upper- to lower-case letters, include special characters, etc. Finally, users create a graphical password key. For the first version of the user authentication system, we have implemented a recognition-based graphical authentication system which provides an image grid manager that can be filled with pictures related to the chosen secret. Through communication with external APIs (currently Google Custom Search has been implemented), FlexPass allows users to include existing pictures through search in Web-based engines. The users then select and create the graphical password key by selecting a set of images among decoys. Finally, users confirm their passwords which are further assigned to profile in the FlexPass database. **Figure 8** illustrates the activity flow diagram for the creation of a textual reflection secret, and **Figure 9** illustrates the activity flow diagram for the creation of graphical reflection secret.

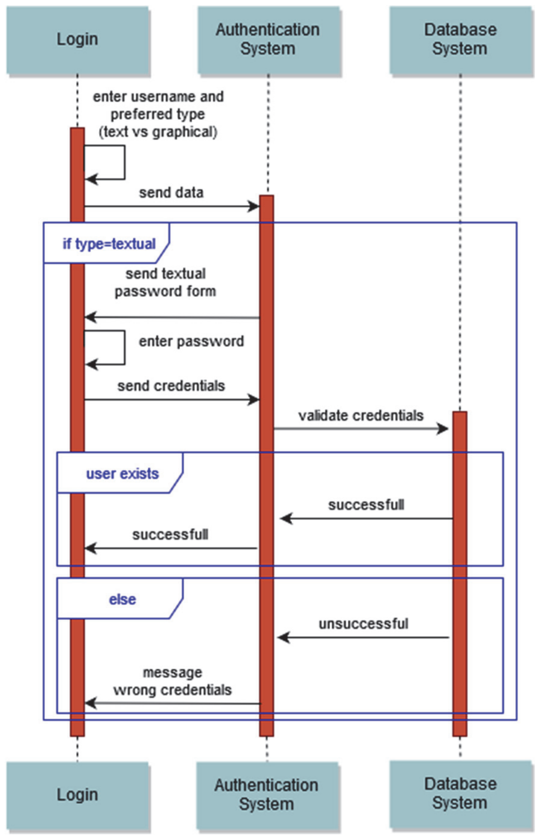


Figure 8. Creation of the single secret and its textual reflection.

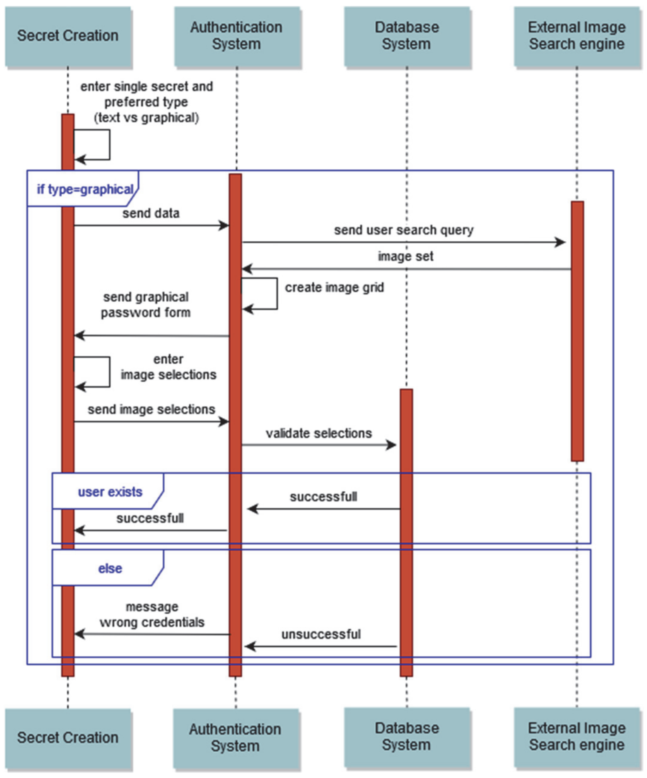


Figure 9. Creation of the single secret and its graphical reflection.

4.6 User-Adaptable Authentication

During user authentication, users can choose their preferred way to authenticate; either by entering the textual key or the graphical key. **Figure 10 (left)** illustrates a login scenario in which the user has selected a textual password as her preferred way to login, and **Figure 10 (right)** illustrates the same login process in which the user has chosen a graphical reflection. In each login session, the alternative option (*e.g.*, graphical password) is available to switch based on the user's preference. Entering the textual key follows the same process as traditional passwords. For entering the graphical key, a grid containing the user-selected and system-generated decoy images are presented. The image positions in the selection grid are randomly positioned in each login session. Thereafter, users have to select their images in the specific sequence, as entered in the enrolment phase to login.

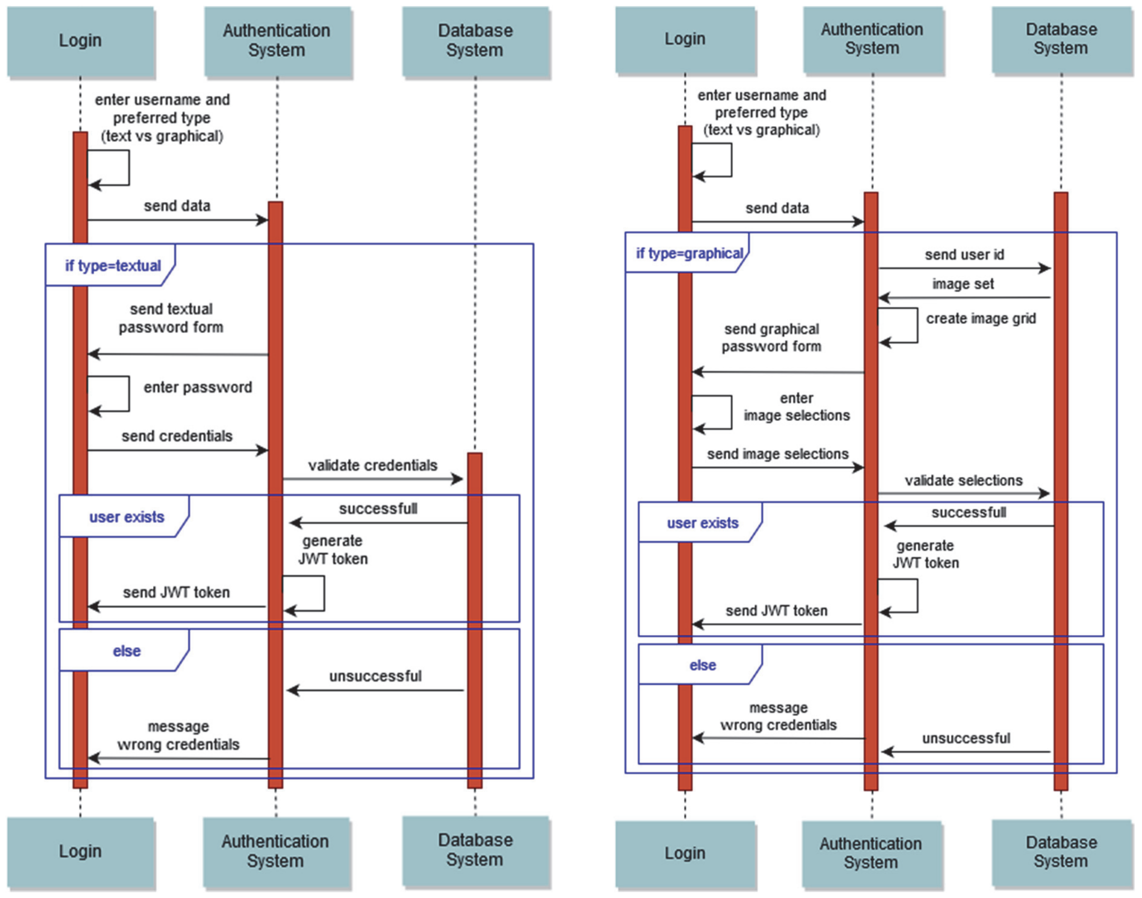


Figure 10. User-adaptable authentication for text login (left) and graphical login (right).

5 Initial Prototype Designs of the User Interfaces

In this section we provide mockups and initial prototypes of the main User Interfaces (UI) 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. Next we present the wireframes and initial mockups of each user task based on the aforementioned use-case scenarios.

5.1 General Layout

Since the user authentication system entails several processes (*i.e.*, password creation, login, password reset, etc.), we have designed a general layout (*i.e.*, a master template) in order to be consistent across the processes. **Figure 11** illustrates a wireframe of the general layout of the user authentication system. Following widely applied user interface design guidelines and heuristics, the general layout includes the system logo, a navigation bar in case the particular process entails menu items that redirect to specific resources, a notification section that illustrates specific system and user-related notifications, a user profile section which summarizes links that relate to the user's account, a navigation path (*"breadcrumps"*) of the current process and a section illustrating the main content of the particular process (*e.g.*, controls of the login page).

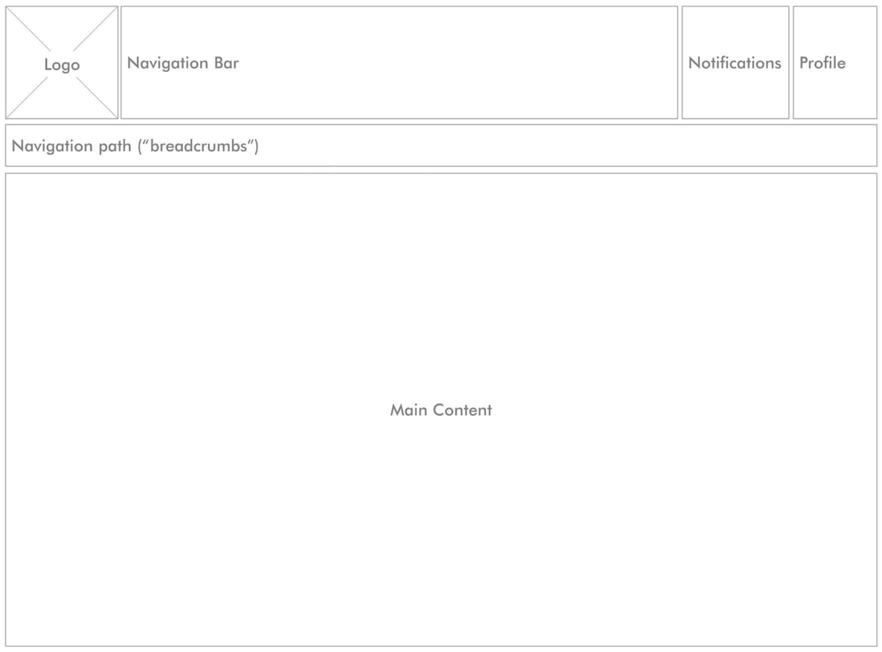


Figure 11. Wireframe of the general layout of the user authentication system.

5.2 UI of the User Secret Creation Process

Figure 12 and **Figure 13** respectively illustrate the wireframe and an initial prototype mockup of the secret creation process and its dual reflection. The process is split in three main steps in which the user initially enters a single secret (*i.e.*, pass phrase). An example secret can be *“Places that we visited in Europe”*. In the next step, users enter the textual reflection by articulating the single secret, *e.g.*, *PlacesthatwevisitedinEurope*. Finally, in the third step, users create a recognition-based graphical password key through an image grid manager that can be filled with pictures related to the chosen secret. In order to fill in the image grid with pictures, users will enter specific search keywords relevant to their secret and the system will communicate with third-party APIs (*e.g.*, Google Custom Search) in order to retrieve relevant images. Users then select and create their graphical password key by selecting a subset of those images. The same image cannot be selected multiple times in a single graphical key. After doing so, the system generates a set of decoy images that are semantically similar to the user-selected pictures to avoid predictability. For this purpose, the system will make use of state-of-the-art services for semantic comparison of images (*e.g.*, IBM Watson Visual Recognition service). Finally, the user confirms the user-selected and system-generated decoy images which are further assigned to the user’s profile in the Serums database.

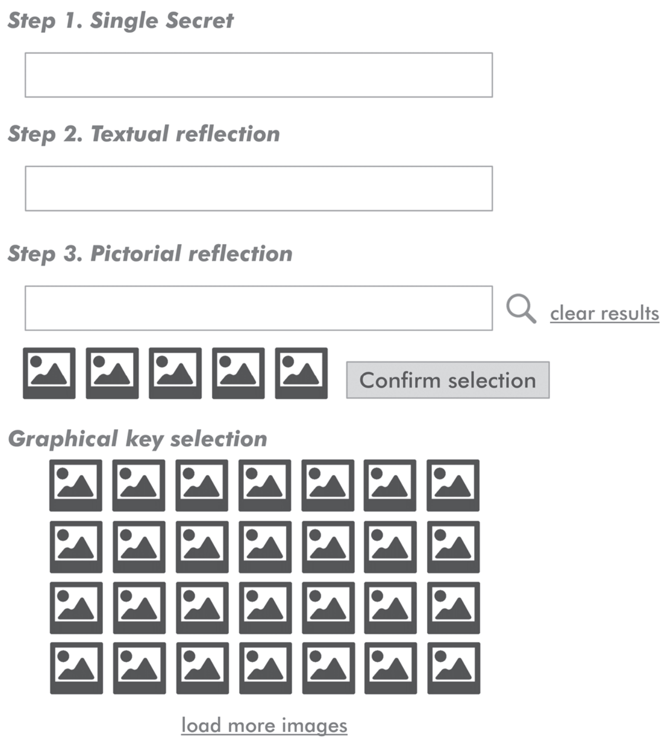
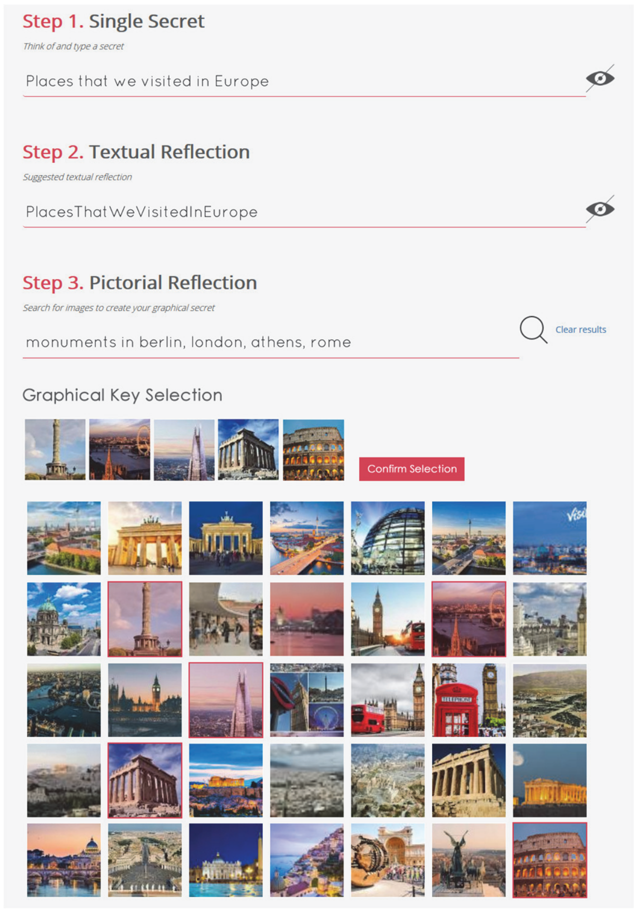


Figure 12. Wireframe of the secret creation and dual reflection.


Step 1. Single Secret

Think of and type a secret

Places that we visited in Europe

Step 2. Textual Reflection

Suggested textual reflection

PlacesThatWeVisitedInEurope

Step 3. Pictorial Reflection

Search for images to create your graphical secret

monuments in berlin, london, athens, rome

Figure 13. Initial prototype mockup of the secret creation and dual reflection.

5.3 UI of the Login Process

Figure 14 and **Figure 15** respectively illustrate the wireframe and an initial prototype mockup of the user authentication process. During this process, users can choose their preferred way to authenticate; either by entering the textual key or the graphical key. In each login session, the alternative option (*e.g.*, graphical password) is available to switch based on the user's preference. In the case of textual password selection, users are required to enter their textual secret, while in the case of graphical password selection, an image grid containing the user-selected and system-generated decoy images are presented. The image positions in the selection grid are randomly positioned in each login session. Thereafter, users have to select their images in the specific sequence, as entered in the creation phase to login.

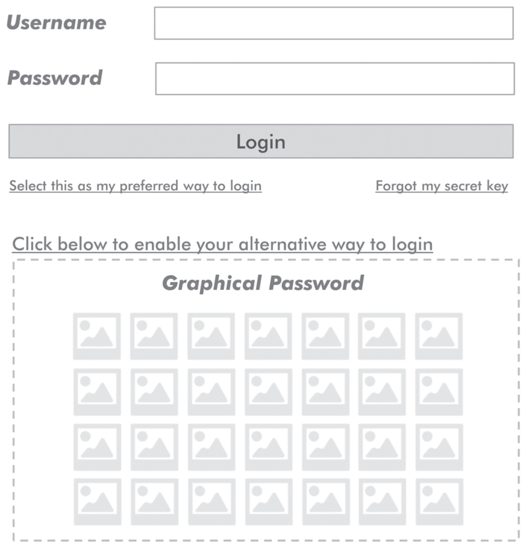

Username
Password
Login
Select this as my preferred way to login
Forgot my secret key
Click below to enable your alternative way to login
Graphical Password

Figure 14. Wireframe of the user authentication process.

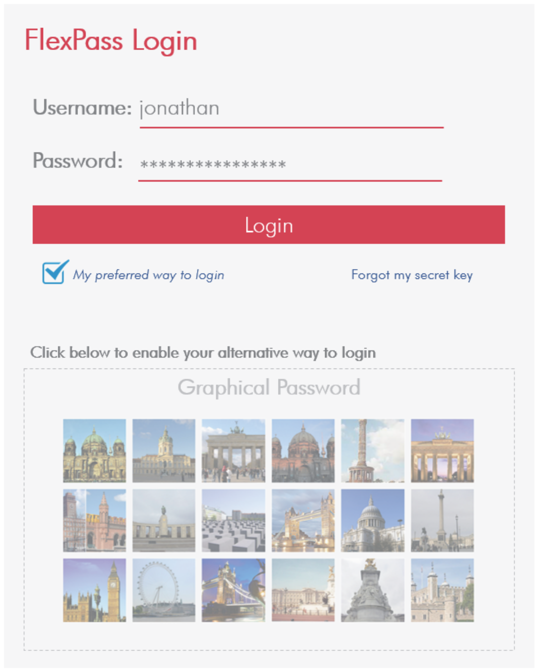

FlexPass Login
Username: jonathan
Password: ****************
Login
My preferred way to login
Forgot my secret key
Click below to enable your alternative way to login
Graphical Password

Figure 15. Initial prototype mockup of the user authentication process.

5.4 UI of the Request Reset Code Process

Figure 16 and **Figure 17** respectively illustrate the wireframe and an initial prototype mockup of the request reset code process. During this process, users are requested to enter their username and then click the "Submit" button in order to receive a reset code (in the email associated with this username) which is required for entering the reset password process.

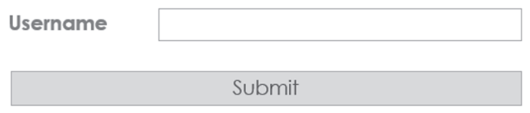
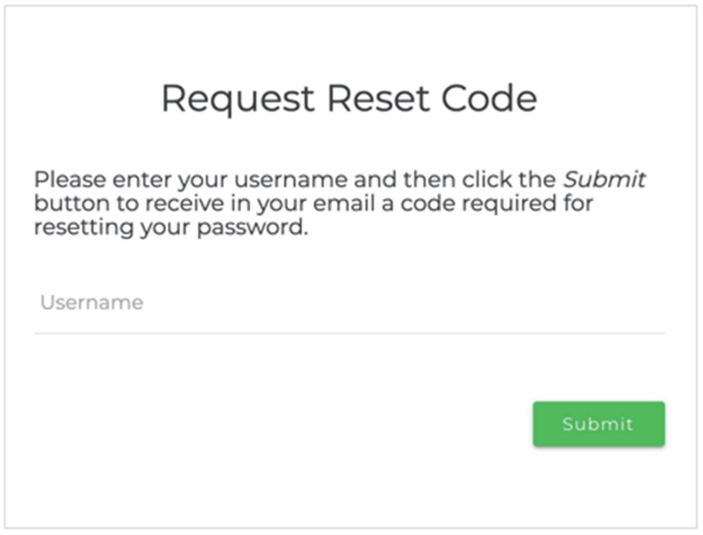


Figure 16. Wireframe of the request reset code process.


Request Reset Code

Please enter your username and then click the *Submit* button to receive in your email a code required for resetting your password.

Figure 17. Initial prototype mockup of the request reset code process.

5.5 UI of the Reset Secret Process

Figure 18 and **Figure 19** respectively illustrate the wireframe and an initial prototype mockup of the request reset secret process. During this process, users are requested to enter their username and the reset code received in the request reset code process and then click the "Submit" button to finalize the process. As part of this process, users also have the ability to request a new reset code in case they haven't received it by clicking the "Resend Reset Code" button. Thereafter, if the reset code provided by the user matches the current active reset code associated to that user, the user will be granted permission to reset the secret and will be redirected to the user secret creation process (**Figure 13**).

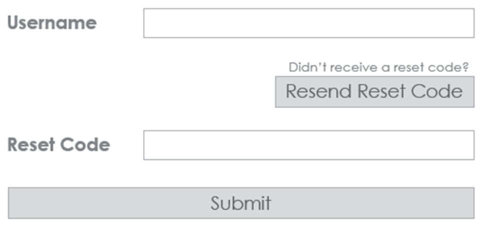
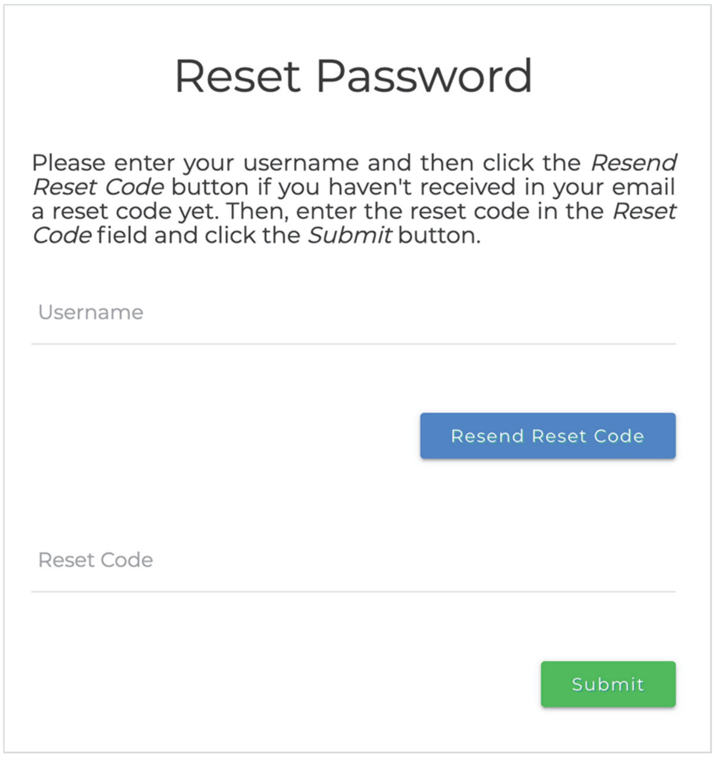


Figure 18. Wireframe of the reset secret process.


Reset Password

Please enter your username and then click the *Resend Reset Code* button if you haven't received in your email a reset code yet. Then, enter the reset code in the *Reset Code* field and click the *Submit* button.

Figure 19. Initial prototype mockup of the reset secret process.

6 Verification of Authentication Properties

As described in *WP5 - T5.4 "Verification of Authentication Properties"* as part of the Serums project we will verify that the authentication policies and scheme satisfy classical authentication properties such as, secrecy or weak agreement. Also, as in D5.1 we will ensure that the implemented software for the *"Verified User Authentication Scheme"* meets various security related measurements and metrics.

For the verification of classical authentication properties, the implementation initially described here will be modelled in a formal language that allows for checking the behavior meets the specification. This will exploit attack models to explore how to violate the correct behaviors of the system and find potential vulnerabilities such as in [3]. These is also a significant part of *WP6 - "Integration and Testing"* that will combine the software here with other work packages (particularly WP3 and WP4).

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 [4]. Note that the approaches used in [4] also incorporate testing of the smart contracts that use the authentication from WP5.

The validation of the metrics described in D5.1 and above in Section 5 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.

Note that at this stage the software is being developed with these verification and validation goals in mind. This deliverable describes the metrics and approaches that will be used. However, due to the requirements to model and validate the implementation, verification and validation results are expected to appear in detail in deliverables D5.3 and D5.4, along with the refinements of the software as influenced by these results.

7 Conclusions

The aim of this deliverable *D5.2. - "Software on the Initial Verified User Authentication System"* is to report the outcome and overall methodology that has been applied for the design and development of the initial software of the user authentication scheme. This includes the general architecture design, the conceptual design for credentials hardening mechanism, the activity flows of primary use-case scenarios of the user authentication scheme, the design of initial prototypes (wireframes and mockup designs) of the user authentication system, and the verification methodology and models of the authentication properties.

This deliverable is an essential first phase within the User-Centered Design software development methodology that was adopted for the implementation of the Serums' user authentication system. The outcome of this deliverable will be used as an essential input for other tasks and deliverables in Serums. Specifically, the *initial API and the underlying database* will be used as input in D2.2 and D2.3 for the software and refined specifications of the Smart Patient Health Records. The authentication *architecture, APIs and database* will be used as an essential input in D6.1 for integrating the authentication system in the overall Serums' smart healthcare system software. The *user interface designs of the primary user authentication tasks* will be evaluated as part of D7.3 aiming to receive initial user feedback and opinions on the proposed authentication paradigm (FlexPass), the mockup designs 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 D5.2 as a whole (and the forthcoming results of the first evaluation studies in WP7) will be used as a basis for the next development cycle of the user authentication scheme in *D5.3. - "Software on the Refined Verified User Authentication Scheme"*.

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] Belk, M., Fidas, C., Pitsillides, A. (2019). Flexpass: Symbiosis of seamless user authentication schemes in IoT. In ACM CHI 2019, ACM Press, 2019.

[3] 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

[4] B. Jiang, Y. Liu, and W. K. Chan. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. CoRR, abs/1807.03932, 2018.

ABBREVIATIONS

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
RDBMS	Relational Database Management System
SHA-256	Secure Hash Algorithm
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UCD	User Centered Design
UI	User Interface
UX	User Experience
WSGI	Web Server Gateway Interface

APPENDIX A – RESTful Application Programming Interface

Base url: http://serums.cs.ucy.ac.cy/ua

Demo: http://serums.cs.ucy.ac.cy/ua/demo

Documentation: http://serums.cs.ucy.ac.cy/ua/doc

Register User - HTTP 201 (Created)

Endpoint	/register_user/
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) [1 .. 50] characters
password *	string (Password) [1 .. 500] characters
single_secret *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource	string (A numerical value associated with that resource)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/register_user/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@ucy.ac.cy\", \"organization\": \"UCY\", \"password\": \"123\", \"single_secret\": \"MySingleSecret\"}"
Response	
Schema	application/json
Description	User has been created and id is returned in resource as <Integer>
Status Code	201

Body	{
	"resource": 2,
	"message": "User has been created",
	"resource_name": "user"
	}

Register User - HTTP 400 (Already Exists Field)

Endpoint	/register_user/
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) [1 .. 50] characters
password *	string (Password) [1 .. 500] characters
single_secret *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/register_user/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"organization\": \"UCY\", \"password\": \"123\", \"single_secret\": \"MySingleSecret\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400

Body	{
	  "missing_required_fields": [],
	  "already_exists_fields": [
	    "username"
	  ],
	  "bad_formatted_fields": [],
	  "message": "username already exists"
	}

Register User - HTTP 400 (Bad Formatted Field)

Endpoint	/register_user/
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) [1 .. 50] characters
password *	string (Password) [1 .. 500] characters
single_secret *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)

Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/register_user/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"argyris.co\", \"organization\": \"UCY\", \"password\": \"123\", \"single_secret\": \"MySingleSecret\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400
Body	{ "missing_required_fields": [], "already_exists_fields": [], "bad_formatted_fields": ["username"], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Register User - HTTP 400 (Missing Required Field – Case 1)

Endpoint	/register_user/
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) [1 .. 50] characters
password *	string (Password) [1 .. 500] characters
single_secret *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/register_user/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"argyris.co@hotmail.com\", \"organization\": \"\", \"password\": \"123\", \"single_secret\": \"MySingleSecret\"}"
Response	
Schema	application/json

Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400

Body	{
	"missing_required_fields": [
	"organization"
	],
	"already_exists_fields": [],
	"bad_formatted_fields": [],
	"message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"
	}

Register User - HTTP 400 (Missing Required Field – Case 2)

Endpoint	/register_user/
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) [1 .. 50] characters
password *	string (Password) [1 .. 500] characters
single_secret *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/register_user/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"argyris.co@hotmail.com\", \"password\": \"123\", \"single_secret\": \"MySingleSecret\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400
Body	{ "missing_required_fields": ["organization"], "already_exists_fields": [], "bad_formatted_fields": [], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"

	}

Register User - HTTP 405

Endpoint	/register_user/
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) [1 .. 50] characters
password *	string (Password) [1 .. 500] characters
single_secret *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/register_user/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@ucy.ac.cy\", \"organization\": \"UCY\", \"password\": \"123\", \"single_secret\": \"MySingleSecret\"}"
Response	
Schema	application/json
Description	Method not allowed
Status Code	405
Body	{ "message": "Method not allowed" }

Register User - HTTP 415

Endpoint	/register_user/
Method	POST
Headers	
accept	application/json
Content-Type	text/plain
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
organization *	string (Organization) [1 .. 50] characters
password *	string (Password) [1 .. 500] characters
single_secret *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json

Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/register_user/" -H "accept: application/json" -H "Content-Type: text/plain" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"organization\": \"UCY\", \"password\": \"123\", \"single_secret\": \"MySingleSecret\"}"
Response	
Schema	application/json
Description	Unsupported Media Type
Status Code	415
Body	{ "message": "Unsupported Media Unsupported media type \"text/plain\" in request." }

Register User - HTTP 500 (Mocked function call to raise Exception)

Endpoint	/register_user/
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) [1 .. 50] characters
password *	string (Password) [1 .. 500] characters
single_secret *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/register_user/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@ucy.ac.cy\", \"organization\": \"UCY\", \"password\": \"123\", \"single_secret\": \"MySingleSecret\"}"
Response	
Schema	application/json
Description	Internal server error
Status Code	500
Body	{ "message": "Internal server error" }

Request Account Verification - HTTP 200 (User not already activated)

Endpoint	/request_account_verification/
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	integer (1 if resource is already activated)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\"}"
Response	
Schema	application/json
Description	Account verification code email sent successfully or User is already activated
Status Code	200
Body	{ "message": "Account verification code email sent successfully", "resource_name": "account_verification_code", "resource_already_activated": 0 }

Request Account Verification - HTTP 200 (User already activated)

Endpoint	/request_account_verification/
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	integer (1 if resource is already activated)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\"}"
Response	
Schema	application/json
Description	Account verification code email sent successfully or User is already activated
Status Code	200
Body	{

	"message": "User is already activated", "resource_name": "user", "resource_already_activated": 1 }

Request Account Verification - HTTP 400 (Bad Formatted Field)

Endpoint	/request_account_verification/
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)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400

Body	{
	"missing_required_fields": [],
	"already_exists_fields": [],
	"bad_formatted_fields": [
	"username"
	],
	"message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"
	}

Request Account Verification - HTTP 400 (Missing Required Field – Case 1)

Endpoint	/request_account_verification/
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)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400
Body	{ "missing_required_fields": ["username"], "already_exists_fields": [], "bad_formatted_fields": [], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Request Account Verification - HTTP 400 (Missing Required Field – Case 2)

Endpoint	/request_account_verification/
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)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: application/json" -d "{}"

Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400

Body	{
	  "missing_required_fields": [
	    "username"
	  ],
	  "already_exists_fields": [],
	  "bad_formatted_fields": [],
	  "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"
	}

Request Account Verification - HTTP 404 (User not found)

Endpoint	/request_account_verification/
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 "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12345@cs.ucy.ac.cy\"}"
Response	
Schema	application/json
Description	User not found
Status Code	404

Body	{
	  "resource_name": "user",
	  "message": "User not found"
	}

Request Account Verification - HTTP 405

Endpoint	/request_account_verification/
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)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\"}"
Response	
Schema	application/json
Description	Method not allowed
Status Code	405
Body	{ "message": "Method not allowed" }

Request Account Verification - HTTP 415

Endpoint	/request_account_verification/
Method	POST
Headers	
accept	application/json
Content-Type	text/plain
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)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: text/plain" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\"}"
Response	
Schema	application/json
Description	Unsupported Media Type
Status Code	415
Body	{ "message": "Unsupported Media Unsupported media type \"text/plain\" in request." }

Request Account Verification - HTTP 422 (Request Limit Exceeded)

Endpoint	/request_account_verification/
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	string (A numerical value associated with that resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\"}"
Response	
Schema	application/json
Description	Request limit exceeded. Try again in <Integer> minutes
Status Code	422
Body	{ "resource": 59, "resource_name": "account_verification_code", "message": "Request limit exceeded. Try again in 59 minutes" }

Request Account Verification - HTTP 500 (Mocked function call to raise Exception)

Endpoint	/request_account_verification/
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)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/request_account_verification/" -H "accept: application/json" -H "Content-Type: text/plain" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\"}"
Response	
Schema	application/json
Description	Internal server error
Status Code	500
Body	{ "message": "Internal server error" }

Verify Account - HTTP 200 (Account activated)

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_already_activated	integer (1 if resource is already activated)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"verification_code\": \"c7a974\"}"
Response	
Schema	application/json
Description	User is now activated or User is already activated
Status Code	200
Body	{ "message": "User is now activated", "resource_name": "user", "resource_already_activated": 0 }

Verify Account - HTTP 200 (Account already activated)

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource_already_activated	integer (1 if resource is already activated)
Example Call	
Request	
Schema	application/json

Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"verification_code\": \"c7a974\"}"
Response	
Schema	application/json
Description	User is now activated or User is already activated
Status Code	200
Body	{ "message": "User is already activated", "resource_name": "user", "resource_already_activated": 1 }

Verify Account - HTTP 400 (Bad Formatted Field)

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http:// serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12\", \"verification_code\": \"c7a974\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400
Body	{ "missing_required_fields": [], "already_exists_fields": [], "bad_formatted_fields": ["username"], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Verify Account - HTTP 400 (Missing Required Field – Case 1)

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400
Body	{ "missing_required_fields": ["username"], "already_exists_fields": [], "bad_formatted_fields": [], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Verify Account - HTTP 400 (Missing Required Field – Case 2)

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)

missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400
Body	{ "missing_required_fields": ["username", "verification_code"], "already_exists_fields": [], "bad_formatted_fields": [], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Verify Account - HTTP 404 (User not found)

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] 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 "http://serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12345@cs.ucy.ac.cy\"}"
Response	
Schema	application/json
Description	User not found
Status Code	404
Body	{ "resource_name": "user", "message": "User not found" }

Verify Account - HTTP 405

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"verification_code\": \"c7a974\"}"
Response	
Schema	application/json
Description	Method not allowed
Status Code	405
Body	{ "message": "Method not allowed" }

Verify Account - HTTP 415

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	text/plain
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: text/plain" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"verification_code\": \"c7a974\"}"
Response	
Schema	application/json
Description	Unsupported Media Type
Status Code	415

Body	{
	   "message": "Unsupported Media Unsupported media type \"text/plain\" in request."
	}

Verify Account - HTTP 422 (Request Limit Exceeded)

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource	string (A numerical value associated with that resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"verification_code\": \"c7a974\"}"
Response	
Schema	application/json
Description	Request limit exceeded. Try again in <Integer> minutes
Status Code	422

Body	{
	   "resource": 59,
	   "resource_name": "account_verification_code",
	   "message": "Request limit exceeded. Try again in 59 minutes"
	}

Verify Account - HTTP 500 (Mocked function call to raise Exception)

Endpoint	/verify_account/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
verification_code *	string <email> (Username) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json

Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/verify_account/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"verification_code\": \"c7a974\"}"
Response	
Schema	application/json
Description	Internal server error
Status Code	500

Body	{
	"message": "Internal server error"
	}

Get Token - HTTP 201 (Created)

Endpoint	/get_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
password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource	string (A numerical value associated with that resource)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/get_token/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@ucy.ac.cy\", \"password\": \"123\"}"
Response	
Schema	application/json
Description	JWT Token has been created
Status Code	201
Body	{ "resource": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIj oxNTY5MDYyMzQyLCJqdGkiOiI2NGNjZDY0ZmM0OTE0YzExYjBjMTk5NzkwZjI1ODE xNSIsInVzZXJfaWQiOjEsInVzZXJuYW1lIjoiYWNvbnN0MTJAY3MudWN5LmFjLmN5Ii wic3ViIjoiYWNvbnN0MTJAY3MudWN5LmFjLmN5Iiwic2VydW1zR3JvdXBzIjpbIm51 cnNlIiwiZG9jdG9yIl0sImlzcyI6IlNlcnVtc0F1dGhlbnRpY2F0aW9uIiwiaWF0IjoxNTY5 MDYyMDQyLCJzZXJ1bXNPcmdJZCI6ImFjb25zdDEyQGNzLnVjeS5hYy5jeSIsImF1ZCI6 Imh0dHBzOi8vdXJsZGVmZW5zZS5wcm9vZnBvaW50LmNvbS92Mi91cmw_dT1odH RwLTNBX193d3cuc2VydW1zLmNvbSZkPUR3SURhUSZjPWVJR2pzSVRmWFBfeS1ET ExYMHVFSFhKdlU4bk9IclVLOElyd05LT3RrVlUmcj11VGZONXVRMWtod2JSeV9UZ0tI NmFVZDAtQmJtMEc4Sy1WYWprelpteTk4Jm09MmlVTm4yOUZTYWY3LTAzeHU5eE 1CcmNuNHQ2VV8zdzN1cUxpTHl0VGZUNCZzPTVqQjJqbXFoc05BX2cxU1Z5WmdVR

	lJGOW9FUDhfQVFhLWxpY1lXM0l1ZncmZT0ifQ.p1MHe2zzXsEY3sOtX3i8qBQSvF8BiEamFcIspNdY-n8",
	    "message": "JWT Token has been created",
	    "resource_name": "jwt"
	}

Get Token - HTTP 400 (Bad Formatted Field)

Endpoint	/get_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
password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/get_token/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12\", \"password\": \"123\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400

Body	{
	    "missing_required_fields": [],
	    "bad_formatted_fields": [
	        "username"
	    ],
	    "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"
	}

Get Token - HTTP 400 (Missing Required Field – Case 1)

Endpoint	/get_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

password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/get_token/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"\", \"password\": \"123\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400

Body	{
	 "missing_required_fields": [
	 "username"
	],
	 "bad_formatted_fields": [],
	 "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"
	 }

Get Token - HTTP 400 (Missing Required Field – Case 2)

Endpoint	/get_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
password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/get_token/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned

Status Code	400

Body	{
	  "missing_required_fields": [
	    "password"
	  ],
	  "bad_formatted_fields": [],
	  "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"
	}

Get Token - HTTP 401

Endpoint	/get_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
password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/get_token/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"password\": \"12345\"}"
Response	
Schema	application/json
Description	Unauthorized
Status Code	401

Body	{
	  "message": " Unauthorized "
	}

Get Token - HTTP 405

Endpoint	/get_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
password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	

Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/get_token/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"password\": \"123\"}"
Response	
Schema	application/json
Description	Method not allowed
Status Code	405
Body	{ "message": "Method not allowed" }

Get Token - HTTP 415

Endpoint	/get_token/
Method	POST
Headers	
accept	application/json
Content-Type	text/plain
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/get_token/" -H "accept: application/json" -H "Content-Type: text/plain" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"password\": \"123\"}"
Response	
Schema	application/json
Description	Unsupported Media Type
Status Code	415
Body	{ "message": "Unsupported Media Unsupported media type \"text/plain\" in request." }

Get Token - HTTP 500 (Mocked function call to raise Exception)

Endpoint	/get_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
password *	string (Password) [1 .. 500] characters

Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/get_token/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@ucy.ac.cy\", \"password\": \"123\"}"
Response	
Schema	application/json
Description	Internal server error
Status Code	500
Body	{ "message": "Internal server error" }

Update Password - HTTP 200 (OK)

Endpoint	/update_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
current_password *	string (Password) [1 .. 500] characters
new_password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/update_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"current_password\": \"123\", \"new_password\": \"12345\"}"
Response	
Schema	application/json
Description	Password updated successfully
Status Code	200
Body	{ "message": "Password updated successfully", "resource_name": "password" }

Update Password - HTTP 400 (Bad Formatted Field)

Endpoint	/update_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
current_password *	string (Password) [1 .. 500] characters
new_password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/update_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12\", \"current_password\": \"123\", \"new_password\": \"12345\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400

Body	{
	"missing_required_fields": [],
	"bad_formatted_fields": [
	"username"
	],
	"message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"
	}

Update Password - HTTP 400 (Missing Required Field – Case 1)

Endpoint	/update_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
current_password *	string (Password) [1 .. 500] characters
new_password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
Example Call	

Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/update_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12\", \"current_password\": \"\", \"new_password\": \"12345\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400

Body	{
	"missing_required_fields": [
	"current_password"
	],
	"already_exists_fields": [],
	"bad_formatted_fields": [],
	"message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"
	}

Update Password - HTTP 400 (Missing Required Field – Case 2)

Endpoint	/update_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
current_password *	string (Password) [1 .. 500] characters
new_password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/update_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12\", \"new_password\": \"12345\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned or User already exists
Status Code	400
Body	{ "missing_required_fields": ["current_password"],

	"bad_formatted_fields": [], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Update Password - HTTP 405

Endpoint	/update_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
current_password *	string (Password) [1 .. 500] characters
new_password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/update_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"current_password\": \"123\", \"new_password\": \"12345\"}"
Response	
Schema	application/json
Description	Method not allowed
Status Code	405

Body	{
	"message": "Method not allowed"
	}

Update Password - HTTP 415

Endpoint	/update_password/
Method	POST
Headers	
accept	application/json
Content-Type	text/plain
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
username *	string <email> (Username) [1 .. 50] characters
current_password *	string (Password) [1 .. 500] characters
new_password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json

Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/update_password/" -H "accept: application/json" -H "Content-Type: text/plain" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"current_password\": \"123\", \"new_password\": \"12345\"}"
Response	
Schema	application/json
Description	Unsupported Media Type
Status Code	415
Body	{ "message": "Unsupported Media Unsupported media type \"text/plain\" in request." }

Update Password - HTTP 500 (Mocked function call to raise Exception)

Endpoint	/update_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
current_password *	string (Password) [1 .. 500] characters
new_password *	string (Password) [1 .. 500] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/update_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"username\": \"aconst12@cs.ucy.ac.cy\", \"current_password\": \"123\", \"new_password\": \"12345\"}"
Response	
Schema	application/json
Description	Internal server error
Status Code	500
Body	{ "message": "Internal server error" }

Request Reset Password - HTTP 200 (OK)

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

Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/request_reset_password/?serums_user_id=1" -H "accept: application/json"
Response	
Schema	application/json
Description	Password reset code email sent successfully
Status Code	200
Body	{ "message": "Password reset code email sent successfully", "resource_name": "password_reset_code" }

Request Reset Password - HTTP 400 (Bad Formatted Field)

Endpoint	/request_reset_password/
Method	GET
Headers	
accept	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/request_reset_password/?serums_user_id" -H "accept: application/json"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400
Body	{ "missing_required_fields": [], "already_exists_fields": [], "bad_formatted_fields": ["serums_user_id"],

	"message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Request Reset Password - HTTP 400 (Missing Required Field – Case 1)

Endpoint	/request_reset_password/
Method	GET
Headers	
accept	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/request_reset_password/" -H "accept: application/json"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400

Body	{
	  "missing_required_fields": [
	    "serums_user_id"
	  ],
	  "already_exists_fields": [],
	  "bad_formatted_fields": [],
	  "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned"
	}

Request Reset Password - HTTP 400 (Missing Required Field – Case 2)

Endpoint	/request_reset_password/
Method	GET
Headers	
accept	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)

already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/request_reset_password/some_random_param=1" -H "accept: application/json"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400
Body	{ "missing_required_fields": ["serums_user_id"], "already_exists_fields": [], "bad_formatted_fields": [], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Request Reset Password - HTTP 404 (User not found)

Endpoint	/request_reset_password/
Method	GET
Headers	
accept	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/request_reset_password/?serums_user_id=2" -H "accept: application/json"
Response	
Schema	application/json
Description	User not found
Status Code	404
Body	{ "resource_name": "user", "message": "User not found" }

Request Reset Password - HTTP 405

Endpoint	/request_reset_password/
Method	GET
Headers	

accept	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/request_reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"serums_user_id\": \"1\"}"
Response	
Schema	application/json
Description	Method not allowed
Status Code	405
Body	{ "message": "Method not allowed" }

Request Reset Password - HTTP 422 (Request Limit Exceeded)

Endpoint	/request_reset_password/
Method	GET
Headers	
accept	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
resource	string (A numerical value associated with that resource)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/request_reset_password/?serums_user_id=1" -H "accept: application/json"
Response	
Schema	application/json
Description	Request limit exceeded. Try again in <Integer> minutes
Status Code	422
Body	{ "resource": 59, "resource_name": "password_reset_code", "message": "Request limit exceeded. Try again in 59 minutes" }

Request Reset Password - HTTP 500 (Mocked function call to raise Exception)

Endpoint	/request_reset_password/
Method	GET
Headers	

accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/request_reset_password/?serums_user_id=1" -H "accept: application/json"
Response	
Schema	application/json
Description	Internal server error
Status Code	500
Body	{ "message": "Internal server error" }

Reset Password - HTTP 200 (OK)

Endpoint	/reset_password/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
password *	string (Password) [1 .. 500] characters
reset_code *	string (Password) [1 .. 6] 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 "http://serums.cs.ucy.ac.cy/ua/reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"serums_user_id\": \"1\", \"password\": \"12345\", \"reset_code\": \"63a7dc\"}"
Response	
Schema	application/json
Description	Password updated successfully
Status Code	200
Body	{ "message": "Password updated successfully", "resource_name": "user" }

Reset Password - HTTP 400 (Bad Formatted Field)

Endpoint	/reset_password/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
password *	string (Password) [1 .. 500] characters
reset_code *	string (Password) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"serums_user_id\": \"test_user\", \"password\": \"12345\", \"reset_code\": \"63a7dc\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400
Body	{ "missing_required_fields": [], "already_exists_fields": [], "bad_formatted_fields": ["serums_user_id"], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Reset Password - HTTP 400 (Missing Required Field – Case 1)

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

password *	string (Password) [1 .. 500] characters
reset_code *	string (Password) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"serums_user_id\": \"1\", \"password\": \"\", \"reset_code\": \"63a7dc\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400
Body	{ "missing_required_fields": ["password"], "already_exists_fields": [], "bad_formatted_fields": [], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Reset Password - HTTP 400 (Missing Required Field – Case 2)

Endpoint	/reset_password/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
password *	string (Password) [1 .. 500] characters
reset_code *	string (Password) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
bad_formatted_fields	Array of strings (Any field that is not in the correct format will be returned in the list)
missing_required_fields	Array of strings (The missing required fields are returned as a list)
already_exists_fields	Array of strings (Any field that is unique and already exists, will be returned in the list)
Example Call	
Request	

Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"serums_user_id\": \"1\", \"reset_code\": \"63a7dc\"}"
Response	
Schema	application/json
Description	Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned
Status Code	400
Body	{ "missing_required_fields": ["password"], "already_exists_fields": [], "bad_formatted_fields": [], "message": "Bad Request - Invalid Data. Any missing, already existing or bad formatted fields will be returned" }

Reset Password - HTTP 404 (User not found)

Endpoint	/reset_password/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
password *	string (Password) [1 .. 500] characters
reset_code *	string (Password) [1 .. 6] 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 "http://serums.cs.ucy.ac.cy/ua/reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"serums_user_id\": \"3\", \"password\": \"12345\", \"reset_code\": \"63a7dc\"}"
Response	
Schema	application/json
Description	User not found
Status Code	404
Body	{ "resource_name": "user", "message": "User not found" }

Reset Password - HTTP 405

Endpoint	/reset_password/

Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
password *	string (Password) [1 .. 500] characters
reset_code *	string (Password) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"serums_user_id\": \"1\", \"password\": \"12345\", \"reset_code\": \"63a7dc\"}"
Response	
Schema	application/json
Description	Method not allowed
Status Code	405
Body	{ "message": "Method not allowed" }

Reset Password - HTTP 415

Endpoint	/reset_password/
Method	POST
Headers	
accept	application/json
Content-Type	text/plain
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
password *	string (Password) [1 .. 500] characters
reset_code *	string (Password) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/reset_password/" -H "accept: application/json" -H "Content-Type: text/plain" -d "{ \"serums_user_id\": \"1\", \"password\": \"12345\", \"reset_code\": \"63a7dc\"}"
Response	
Schema	application/json
Description	Unsupported Media Type
Status Code	415

Body	{
	 "message": "Unsupported Media Unsupported media type \"text/plain\" in request."
	 }

Reset Password - HTTP 422 (Request Limit Exceeded)

Endpoint	/reset_password/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
password *	string (Password) [1 .. 500] characters
reset_code *	string (Password) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
resource_name	string (The name of the resource)
reason	string (The reason behind this message)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"serums_user_id\": \"1\", \"password\": \"12345\", \"reset_code\": \"63a7dc\"}"
Response	
Schema	application/json
Description	Password reset code is invalid or Password reset code has expired
Status Code	422

Body	{
	 "reason": "expired_token",
	 "resource_name": "password_reset_code",
	 "message": "Password reset code has expired"
	 }

Reset Password - HTTP 500 (Mocked function call to raise Exception)

Endpoint	/reset_password/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Input Parameters (* required)	**Type <Format> (Field Model) [MinLength .. MaxLength]**
serums_user_id *	integer (serums_user_id)
password *	string (Password) [1 .. 500] characters
reset_code *	string (Password) [1 .. 6] characters
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	

Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/reset_password/" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"serums_user_id\": \"1\", \"password\": \"12345\", \"reset_code\": \"63a7dc\"}"
Response	
Schema	application/json
Description	Internal server error
Status Code	500
Body	{ "message": "Internal server error" }

Sample Authenticated - HTTP 200 (OK)

Endpoint	/sample_authenticated/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/sample_authenticated/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNTY5MDYyMzQyLCJqdGkiOiI2NGNjZDY0ZmM0OTE0YzExYjBjMTk5NzkwZjI1ODExNSIsInVzZXJfaWQiOjEsInVzZXJuYW1lIjoiYWNvbnN0MTJAY3MudWN5LmFjLmN5Iiwic3ViIjoiYWNvbnN0MTJAY3MudWN5LmFjLmN5Iiwic2VydW1zR3JvdXBzIjpbIm51cnNlIiwiZG9jdG9yIl0sImlzcyI6IlNlcnVtc0F1dGhlbnRpY2F0aW9uIiwiaWF0IjoxNTY5MDYyMDQyLCJzZXJ1bXNPcmdJZCI6ImFjb25zdDEyQGNzLnVjeS5hYy5jeSIsImF1ZCI6Imh0dHBzOi8vdXJsZGVmZW5zZS5wcm9vZnBvaW50LmNvbS92Mi91cmw_dT1odHRwLTNBX193d3cuc2VydW1zLmNvbSZkPUR3SURhUSZjPWVJR2pzSVRmWFBfeS1ETExYMHVFSFhKdlU4bk9IclVLOElyd05LT3RrVlUmcj11VGZONXVRMWtod2JSeV9UZ0tINmFVZDAtQmJtMEc4Sy1WYWprelpteTk4Jm09MmlVTm4yOUZTYWY3LTAzeHU5eE1CcmNuNHQ2VV8zdzN1cUxpTHl0VGZUNCZzPTVqQjJqbXFoc05BX2cxU1Z5WmdVRlJGOW9FUDhfQVFhLWxpY1lXM0l1ZncmZT0ifQ.p1MHe2zzXsEY3sOtX3i8qBQSvF8BiEamFcIspNdY-n8" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Success
Status Code	200
Body	{ "message": "Authenticated" }

Sample Authenticated - HTTP 401 (Unauthorized – Token expired)

Endpoint	/sample_authenticated/
Method	POST

Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/sample_authenticated/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNTY5MDYyMzQyLCJqdGkiOiI2NGNjZDY0ZmM0OTE0YzExYjBjMTk5NzkwZjI1ODExNSIsInVzZXJfaWQiOjEsInVzZXJuYW1lIjoiYWNvbnN0MTJAY3MudWN5LmFjLmN5Iiwic3ViIjoiYWNvbnN0MTJAY3MudWN5LmFjLmN5Iiwic2VydW1zR3JvdXBzIjpbIm51cnNlIiwiZG9jdG9yIl0sImlzcyI6IlNlcnVtc0F1dGhlbnRpY2F0aW9uIiwiaWF0IjoxNTY5MDYyMDQyLCJzZXJ1bXNPcmdJZCI6ImFjb25zdDEyQGNzLnVjeS5hYy5jeSIsImF1ZCI6Imh0dHBzOi8vdXJsZGVmZW5zZS5wcm9vZnBvaW50LmNvbS92Mi91cmw_dT1odHRwLTNBX193d3cuc2VydW1zLmNvbSZkPUR3SURhUSZjPWVJR2pzSVRmWFBfeS1ETExYMHVFSFhKdlU4bk9IclVLOElyd05LT3RrVlUmcj11VGZONXVRMWtod2JSeV9UZ0tINmFVZDAtQmJtMEc4Sy1WYWprelpteTk4Jm09MmlVTm4yOUZTYWY3LTAzeHU5eE1CcmNuNHQ2VV8zdzN1cUxpTHl0VGZUNCZzPTVqQjJqbXFoc05BX2cxU1Z5WmdVRlJGOW9FUDhfQVFhLWxpY1lXM0l1ZncmZT0ifQ.p1MHe2zzXsEY3sOtX3i8qBQSvF8BiEamFcIspNdY-n8" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Unauthorized
Status Code	401

Body	{
	"message": "Unauthorized"
	}

Sample Authenticated - HTTP 401 (Unauthorized – Credentials were not provided)

Endpoint	/sample_authenticated/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/sample_authenticated/" -H "accept: application/json" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Unauthorized
Status Code	401

Body	{
	"message": "Unauthorized"
	}

Sample Authenticated - HTTP 405

Endpoint	/sample_authenticated/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X GET "http://serums.cs.ucy.ac.cy/ua/sample_authenticated/" -H "accept: application/json" -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNTY5MDYy MzQyLCJqdGkiOiI2NGNjZDY0ZmM0OTE0YzExYjBjMTk5NzkwZjI1ODExNSIsInVzZXJfaWQiOjEsInV zZXJuYW1lIjoiYWNvbnN0MTJAY3MudWN5LmFjLmN5Iiwic3ViIjoiYWNvbnN0MTJAY3MudWN5L mFjLmN5Iiwic2VydW1zR3JvdXBzIjpbIm51cnNlIiwiZG9jdG9yIl0sImlzcyI6IlNlcnVtc0F1dGhlbnRpY2 F0aW9uIiwiaWF0IjoxNTY5MDYyMDQyLCJzZXJ1bXNPcmdJZCI6ImFjb25zdDEyQGNzLnVjeS5hYy5j eSIsImF1ZCI6Imh0dHBzOi8vdXJsZGVmZW5zZS5wcm9vZnBvaW50LmNvbS92Mi91cmw_dT1odH RwLTNBX193d3cuc2VydW1zLmNvbSZkPUR3SURhUSZjPWVJR2pzSVRmWFBfeS1ETExYMHVFSFh KdlU4bk9IclVLOElyd05LT3RrVlUmcj11VGZONXVRMWtod2JSeV9UZ0tINmFVZDAtQmJtMEc4Sy1 WYWprelpteTk4Jm09MmlVTm4yOUZTYWY3LTAzeHU5eE1CcmNuNHQ2VV8zdzN1cUxpTHl0VGZ UNCZzPTVqQjJqbXFoc05BX2cxU1Z5WmdVRlJGOW9FUDhfQVFhLWxpY1lXM0l1ZncmZT0ifQ.p1M He2zzXsEY3sOtX3i8qBQSvF8BiEamFcIspNdY-n8" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Method not allowed
Status Code	405

Body	{
	  "message": "Method not allowed"
	}

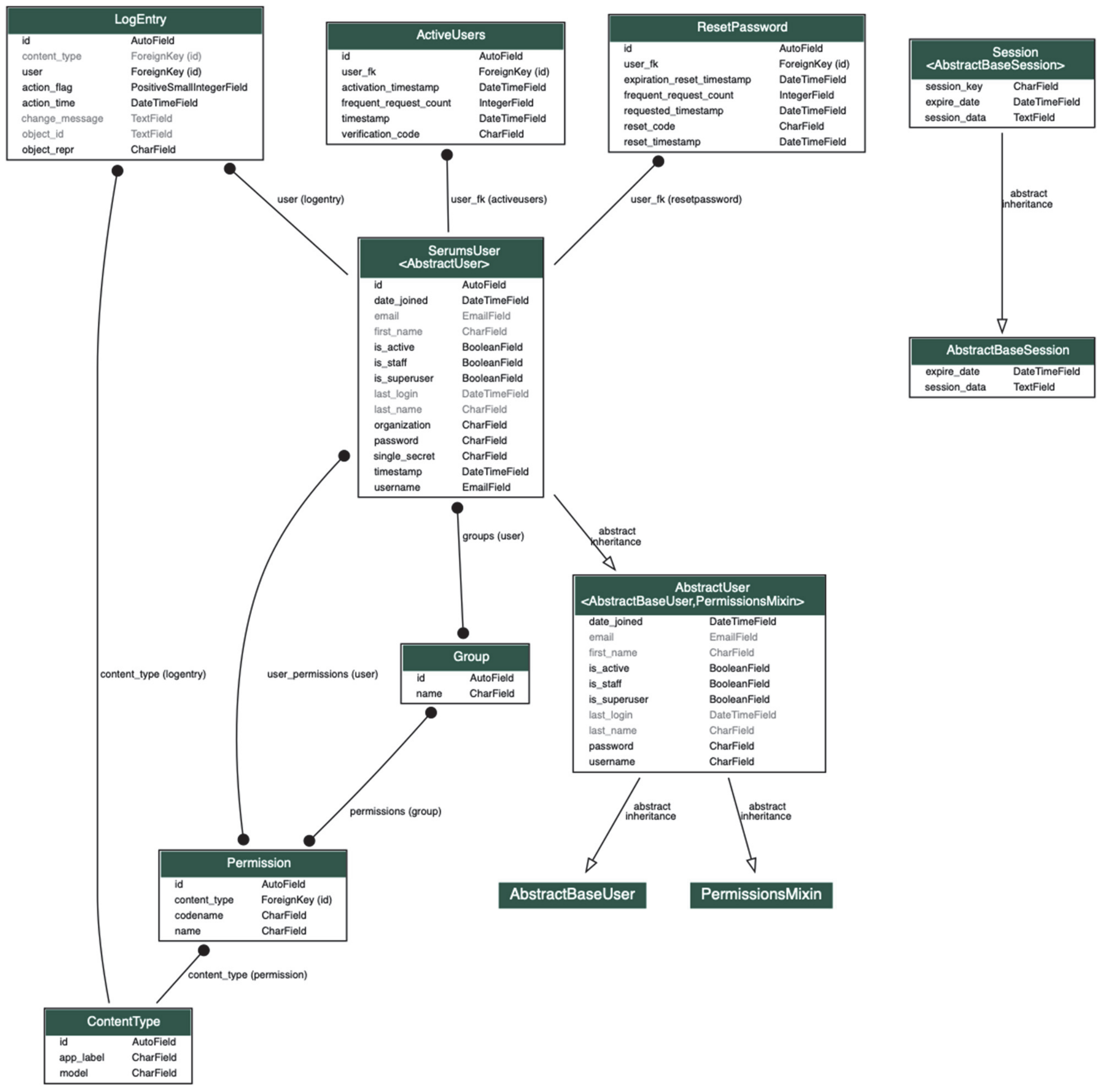
Sample Authenticated - HTTP 500 (Mocked function call to raise Exception)

Endpoint	/sample_authenticated/
Method	POST
Headers	
accept	application/json
Content-Type	application/json
Authorization	Bearer: <JWT>
Output Parameters	**Type (Description)**
message	string (A general message description)
Example Call	
Request	
Schema	application/json
Curl command	curl -X POST "http://serums.cs.ucy.ac.cy/ua/sample_authenticated/" -H "accept: application/json" -H "Authorization: Bearer

	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNTY5MDYy MzQyLCJqdGkiOiI2NGNjZDY0ZmM0OTE0YzExYjBjMTk5NzkwZjI1ODExNSIsInVzZXJfaWQiOjEsInV zZXJuYW1lIjoiYWNvbnN0MTJAY3MudWN5LmFjLmN5Iiwic3ViIjoiYWNvbnN0MTJAY3MudWN5L mFjLmN5Iiwic2VydW1zR3JvdXBzIjpbIm51cnNlIiwiZG9jdG9yIl0sImlzcyI6IlNlcnVtc0F1dGhlbnRpY2 F0aW9uIiwiaWF0IjoxNTY5MDYyMDQyLCJzZXJ1bXNPcmdJZCI6ImFjb25zdDEyQGNzLnVjeS5hYy5j eSIsImF1ZCI6Imh0dHBzOi8vdXJsZGVmZW5zZS5wcm9vZnBvaW50LmNvbS92Mi91cmw_dT1odH RwLTNBX193d3cuc2VydW1zLmNvbSZkPUR3SURhUSZjPWVJR2pzSVRmWFBfeS1ETExYMHVFSFh KdlU4bk9IclVLOElyd05LT3RrVlUmcj11VGZONXVRMWtod2JSeV9UZ0tINmFVZDAtQmJtMEc4Sy1 WYWprelpteTk4Jm09MmlVTm4yOUZTYWY3LTAzeHU5eE1CcmNuNHQ2VV8zdzN1cUxpTHl0VGZ UNCZzPTVqQjJqbXFoc05BX2cxU1Z5WmdVRlJGOW9FUDhfQVFhLWxpY1lXM0l1ZncmZT0ifQ.p1M He2zzXsEY3sOtX3i8qBQSvF8BiEamFcIspNdY-n8" -H "Content-Type: application/json" -d "{}"
Response	
Schema	application/json
Description	Internal server error
Status Code	500

Body	{
	    "message": "Internal server error"
	}

APPENDIX B – Database Design (Entity-Relationship Diagram)



LogEntry
id AutoField
content_type ForeignKey (id)
user ForeignKey (id)
action_flag PositiveSmallIntegerField
action_time DateTimeField
change_message TextField
object_id TextField
object_repr CharField
ActiveUsers
id AutoField
user_fk ForeignKey (id)
activation_timestamp DateTimeField
frequent_request_count IntegerField
timestamp DateTimeField
verification_code CharField
ResetPassword
id AutoField
user_fk ForeignKey (id)
expiration_reset_timestamp DateTimeField
frequent_request_count IntegerField
requested_timestamp DateTimeField
reset_code CharField
reset_timestamp DateTimeField
Session <AbstractBaseSession>
session_key CharField
expire_date DateTimeField
session_data TextField
abstract inheritance
AbstractBaseSession
expire_date DateTimeField
session_data TextField
user (logentry)
user_fk (activeusers)
user_fk (resetpassword)
SerumsUser <AbstractUser>
id AutoField
date_joined DateTimeField
email EmailField
first_name CharField
is_active BooleanField
is_staff BooleanField
is_superuser BooleanField
last_login DateTimeField
last_name CharField
organization CharField
password CharField
single_secret CharField
timestamp DateTimeField
username EmailField
groups (user)
abstract inheritance
AbstractUser <AbstractBaseUser,PermissionsMixin>
date_joined DateTimeField
email EmailField
first_name CharField
is_active BooleanField
is_staff BooleanField
is_superuser BooleanField
last_login DateTimeField
last_name CharField
password CharField
username CharField
content_type (logentry)
user_permissions (user)
Group
id AutoField
name CharField
permissions (group)
abstract inheritance
abstract inheritance
AbstractBaseUser
PermissionsMixin
Permission
id AutoField
content_type ForeignKey (id)
codename CharField
name CharField
content_type (permission)
ContentType
id AutoField
app_label CharField
model CharField