# Project ~~Lezioni~~ Kutambua One-Login Authentication and Member Check

## General statements

Project Kutambua provides authentication services to other systems. Key requirements to fulfill:

- must provide human authentication interface
- must provide non interactive authentication request processing interface
  - web redirection
  - background processes
- should allow to edit Jurisdiction member data
- must not contain sensitive information about any member
- should allow to add bulk member data
- should be flexible to fulfill jurisdictional differences
- should be independent from any 3rd party dependencies (cloud or other providers)

### Security considerations:

All of the processes and technologies implemented are designed with these security concerns and considerations:

- no plain text data should every travel between the client and server-side: completed
- each session data flow must be unique and be protected against man-in-the-middle capture and replay attacks: completed
- client-server communication must use encryption: using only HTTPS

### Human authentication interface:

A web interface where any jurisdiction should be able to authenticate members. Key requirements:

- simple design: completed
- ability to select jurisdiction and display specific fields for that jurisdiction: completed
- ability to switch user interface language: in progress

## Jurisdictional differences:

Not all jurisdictions are using the same fields and method to authenticate the members. To overcome this a dynamic composition system is created in the validation method to fulfill all requirements with different fields.

The field composition algorithms are defined by jurisdiction.

## Validation composition:

The composition sequence can contain certain types of elements in an order. These elements are common, but the list can be extended very easily with specific ones.

Currently these elements are encoded in the POC system:

‘E’: Email address: converted to lower case then hashed with MD5 at client side.

‘P’: Password: The value is hashed with MD5 at client side.

‘U’: User / Member ID: converted to lower case then hashed with MD5 at client side.

‘T’: Static text: Everything after the letter ‘T’ will be appended to the composition.

Sample Data validation with composition sequence code: E, TSOME, P, TTHING

- Start with an empty string: S=”
- First part is ’E’: Take the MD5 hash of EMAIL field and append it to S.
- Second part is TSOME: Append the text ‘SOME’ to S.
- Third part is ‘P’: Take the MD5 hash of the PASSWORD field and append it to S.
- Fourth part is ‘TTHING’: Append the text ‘THING’ to S.
- Hash S with SHA1

# Extended member properties

These are optional property fields which are not taking part of the authentication processes but carry valuable extra information about the member:

- level: the levels the member currently at in various systems. A.M.O.R.C., T.M.O. or any other system.
- tags: a tagging based authorization system which controls which identity can access which services within Kutambua

## Universal Level number format:

Having a universal level identification system is key for IT systems to properly set these permission levels. In this proposal I was trying to think of various factors to include as:

- Have a universal but versatile numbering which could extend to other than just Monographs.
- Add some space for additions without the requirement to renumber everything.
- Must be easily comparable.

A level identification number shall be a numerical value with the following key parts:

- 1: Purpose identifier: currently it is 1 for the A.M.O.R.C. monograph system. We can have 2 for TMO.
- 3: Level Major: 001: Mandamus, 002: Atrium 1, 003: Atrium 2, 004: Atrium 3, 101: T1, 112: T12
- 4: Level minor: 0010: Monograph 1, 0220: Monograph 22, lest digit is for future use if we need to insert something between two.

Examples:

11080220: That is A.M.O.R.C. T8 Monograph 22.

20010010: Is TMO first level Monograph 1.

### Tags:

This extended field contains extra authorization information about a member. The filed contains a list of keywords separated by ‘,’ character.

Implemented tags:

- admin: if his tag exists then the member is authorized to administer the member data inside the jurisdiction.
- mcheck: if this tag exists then the member is authorized to do member checks in any jurisdiction.

# Implementation:

## Server side:

The server side stores several small database files in a native PHP file format but allows the bulk import to provide CSV imports.

### Scheme DB:

The first one is called the Scheme DB. This file stores the fields, and validation composition sequences for each Jurisdiction, along with the display name.

Fields:

- JCode: The Jurisdiction internal code.
- Fields: The login field list string. Each letter represents a field:
  - E: Email address field
  - U: User / Member ID field
  - P: Password field
- VMethod: The validation composition sequence string.
- MFields: Member Check fields which are used for the Member Check function.
- MMethod: Member Check validation method. The same as the VMethod excluding the password.
- Name: The display name of the Jurisdiction

### Member DB:

The rest are the Authentication DBs by Jurisdiction which contains the actual login validation hashes and any data needed for the application, like a display name.

Fields in CSV:

- Hash: The pre-compiled HASH of the user authentication data
- DisplayName: A display name which can be anything and here just for display purposes when successfully authenticated the user.
- Level: A data field containing level information if it is needed for level checks.
- Tags: system flags which control certain features, like access privileges. In this POC I use the following tags:

- Admin: the user with this tag can manage their Jurisdiction data
- MCheck: the user with this tag can execute Memebr Checks
- CHash: The generated hash for Member Check funcion

The server-side code provides data security with session variables which exist only on the server side.

### Jurisdiction scheme example:

```
JCode;Fields;VMethod;MFields;MMethod;Name
HU;EUP;TAM,E,TOR,U,TC,P;E;CME,E,CMBER;Magyar Teszt Nagypáholy
EN;UP;TAM,U,TOR,P,TC;U;CTEM,U,CP;American Test Jurisdiction
FI;EP;E,TAMO,P,TRC;E;CFIN,E,CNISH;Finnish Test Jurisdiction
```

### Auth DB example:

```
Hash;DisplayName;Level;Tags;CHash
5f1b29cd324701dcf90e~;NVL Teszt;L1;Admin,MCheck;5f1b29cd324701dcf90e~
5f1b29cd324701dcf91e~;Member1;L2;MCheck;5f1b29cd324701dcf91e~
```

5f1b29cd324701dcf91e~;Member2;L3;;5f1b29cd324701dcf91e~

### Server side system requirements:

The server-side requirements for this POC are a web server with PHP scripting engine.

## Client side:

On the client computer the member opens the authentication page and selects the jurisdiction. This can be preset with the language selector in the URL itself. That way a direct link can be shared with the language already selected.

The login fields are automatically displayed based on the jurisdiction requirements.

The member fills in the form then clicks on the ‘Login’ button.

A client-side JavaScript code takes over and hashes all field values then submits the form. That way the data is securely obfuscated.

### Client side requirements:

A standard web browser with JavaScript engine enabled (that is default nowadays)

# Processes:

## Jurisdiction data requirements:

Each Jurisdiction must provide:

- Authentication fields, like e-mail or Member ID or any other text field and password.
- Validation compilation sequence code: Discussed earlier.
- Member hash list.

### Jurisdiction addition: generate database entries:

To add a Jurisdiction into the system these steps must be completed:

- First, a scheme must be created. This needs the following information:
  - Pick a Jurisdiction code. We recommend this to be a 2 characters country code where the center is located. Example: HU
  - Define which fields you are using to identify your membership. Predefined fields are E-mail address, Member ID, Password. This list can be amended with new types of values.
  - Validation Method: Define what will be the validation concatenation code of input data to create the hash from. Check the Validation composition section in this document for details.
  - Define which fields you are using to do a Member Check. Predefined fields are E-mail address (E) and Member ID (U). This list can be amended with new types of values if needed.
  - Member Check Validation Method: Define what will be the validation concatenation code of input data to create the hash from. Check the Validation composition section in this document for details.
  - Pick a display name which will identify your Jurisdiction in the list.
- Second, create the hash entries from your authentication database. This means that you must take each member's entry and generate the following fields:
  - Execute the Validation steps on your key fields and generate the Hash.
  - Pick a display name. This can be a surname or serial number of any text.
  - Generate the level code (still under development …)
  - Set tags for system privileges or any application related function.
  - Execute the Validation steps on your Member Check fields and generate the Member Check Hash.