Build TRUSTID solution locally

This document includes step-by-step instructions for building the TRUSTID solution (server, client applications, mobile application) locally. It requires Docker technology (https://www.docker.com/).

First, you need to clone the repository of the TRUSTID solution:

```
git clone https://github.com/cognitiveux/trustid.git
```

TRUSTID Server

First, go to the source code directory of the final version:

```
cd trustid/second_period/
```

Then, extract the **trustid-server.zip** to get the source code of the backend server:

```
unzip trustid-server.zip
```

Then, go to the directory of the backend server:

```
cd trustid-server/server/
```

Then, build the Docker image as follows:

```
docker-compose -f docker-compose.yml build --no-cache
```

Then, create the Docker network as follows:

```
docker network create web
```

To start the Docker containers of the server, issue the following command:

```
docker-compose -f docker-compose.yml up
```

Wait until you see on the terminal the following:

```
Starting development server at http://0.0.0.0:10000/
```

Interactive Web API: http://localhost:10000/backend/demo/
Documentation: http://localhost:10000/backend/doc/

In another terminal, start the Docker container of the face recognition library as follows:

```
docker-compose -f docker-compose-face-lib.yml up
```

Furthermore, you will need to create an App password in your Gmail account. Follow the instructions here: https://support.google.com/mail/answer/185833 to create an app password that will be used to send email via the server. Once you have an App password, modify the following file:

```
trustid/second_period/trustid-server/server/django_variables.env
```

and update the values of the following variables accordingly:

```
SERVER_EMAIL=YOUR_EMAIL@DOMAIN.COM
SERVER_EMAIL_ALIAS=YOUR_EMAIL_ALIAS@DOMAIN.COM
SERVER_EMAIL_PASSWORD=THE_CREATED_APP_PASSWORD
```

Additional Notes

Create an Instructor account

Fill in the required fields under the /register_user endpoint through the Interactive Web API or through an API testing tool (*e.g.*, Postman)

Add a new examination

You could either:

- Use the /instructor/add_exam endpoint (requires first to obtain a JWT token via the /login endpoint and pass it as a Bearer token in the authorization header) or
- Login through the TRUSTID client application, go to Management, Add Exam

Enroll students to examination

- In the case of UC and UPAT:
    - Upload a .csv file with the students' information exported from other LMS in the following format:

Csv file format for Windows:

Surname	Name	E-mail	Identity	Username	User Group
User_Surname	User_Name	user@example.com	123456	username	-

Csv file format for macOS:

Course Name	user@example.com	User_Name	User_Surname

- In the case of UCY (includes Moodle LMS integration):

Setup Moodle base url in the following file:

```
trustid/second_period/trustid-server/server/trustid_project/settings.py
    MOODLE_BASE_URL = "YOUR_MOODLE_BASE_URL_HERE"
```

Create a Moodle course and manually enroll students from within Moodle.

TRUSTID Windows Client Application

First, go to the source code directory of the final version:

```
cd trustid/second_period/
```

Then, extract the **trustid-windows.zip** to get the source code of the Windows client application:

```
unzip trustid-windows.zip
```

Then, go to the directory of the Windows client application:

```
cd trustid-windows/
```

Then, open the solution **trustid.sln** in Visual Studio.

Point the server base url to localhost:

Go to **trustid-windows/trustid/Globals.cs** and change the **BASE_API_URL** to http://localhost:10000/backend/

Finally, start the solution in debug mode.

TRUSTID macOS Client Application

First, go to the source code directory of the final version:

```
cd trustid/second_period/
```

Then, extract the **trustid-macos.zip** to get the source code of the macOS client application:

```
unzip trustid-macos.zip
```

Then, go to the directory of the macOS client application:

```
cd trustid-macos/
```

Then, go to **TrustId/TrustId.xcodeproj/** and open the project **project.pbxproj** in Xcode.

Point the server base url to localhost:

Go to **trustid-macos/TrustId/TrustId/Domain/Api/TrustIdApi.swift** and set the environment to localhost as follows:

```
environment:Environment = .localhost
```

Start the project in debug mode.

TRUSTID Privacy-preserving Mobile Application Wallet

First, go to the source code directory of the final version:

```
cd trustid/second_period/
```

Then, extract the **trustid-wallet.zip** to get the source code of the wallet mobile application:

```
unzip trustid-wallet.zip
```

Then, go to the directory of the wallet mobile application:

```
cd trustid/wallet/
```

Point the server base url to localhost:

Go to **trustid-wallet/ServerIP.js** and change the **server_ip** variable to the IP address provided by Expo during application start:

http://LOCAL_URL_HERE:10000/backend/

Open the project in Visual Studio Code.

You need to install:

- Install **npm** (https://docs.npmjs.com/downloading-and-installing-node-js-and-npm)
- Install **node.js** (https://nodejs.org/en/download)
- Setup **Expo Go** development (https://reactnative.dev/docs/environment-setup)

To install and run the project in a USB-connected device, run the following:

```
npm install
npm start
```

Scan the QR code from the Expo Go application (Android) or the Camera app (iOS)

(Optional) To create an Android .apk, run the following:

```
eas build -p android --profile preview
```

Notes on the Moodle LMS Integration

Obtain a Moodle token

Send a POST request to **MOODLE_BASE_URL + "/login/token.php"** using the following:

```
payload = {
        "username": moodle_username,
        "password": password,
        "service": "moodle_mobile_app"
}
headers = {
        'Content-Type': "application/x-www-form-urlencoded",
}
```

Upon successful response, grab the Moodle token from the response.text json and then the **token** variable.

Get Moodle courses

Use the moodle token obtained previously, and send a POST request to **MOODLE_BASE_URL + "/webservice/rest/server.php"** using the following:

```
payload = {
        "wstoken": moodle_token,
        "wsfunction": "core_course_get_courses_by_field",
        "moodlewsrestformat": "json"
}
headers = {
        'Content-Type': "application/x-www-form-urlencoded",
}
```

Upon successful response, grab the Moodle courses of the instructor from the response.text json and then the **courses** variable. For each moodle course in the **courses** variable, grab the course **id** and the course **fullname**.

Get enrolled users in each Moodle course
Use the moodle token obtained previously, and for each course id from the previous step, send a POST request to MOODLE_BASE_URL + "/webservice/rest/server.php" using the following:

```
payload = {
        "wstoken": moodle_token,
        "wsfunction": "core_enrol_get_enrolled_users",
        "moodlewsrestformat": "json",
        "courseid": course_id
}
headers = {
        'Content-Type': "application/x-www-form-urlencoded",
}
```

Upon successful response, grab the roles from the roles variable, and for that role check if its shortname variable is equal to "student", and then grab the "firstname", the "lastname", and the "email".