Report

PHOTOBOOTH

August 26, 2025

Stage 2A 2023/2024

Arthur DEFORGE,
arthur.deforge@ecole.ensicaen.fr
Benjamin POIREAULT

University tutor:
Benjamin RICAUD

TABLE OF CONTENTS

PRESENTATION

This project, called Photo Booth, is an application that serves as a demonstration of the AI image generation process. It was designed and developed to be exhibited in the Nordnorsk Vitensenter in Tromsø, Norway. The basic principle of the application is to be a classic photo booth but with an AI-generated image as output. The user takes a picture of himself and chooses the generation style he wants. The app generates the image and the user can retrieve it on their phone.

1. Project framework

1.1. Machine Learning Group

The Machine Learning Group (ML Group) at the Norwegian Arctic University in Tromsø is behind the project. The Machine Learning Group at the Arctic University of Norway in Tromsø is a dynamic and innovative research center specializing in the development of machine learning solutions. The group is led by renowned experts in the field and has several researchers and engineers working on various research and development projects.

Figure 2 : Machine Learning Group logo

Figure 2 : Logo of the University of Tromsø

1.2. The Project of an internship

This application is the result of the work of two trainees in charge of producing demonstrations that the Machine Learning Group could exhibit at the Vitensenter. The two interns are both French interns studying in a French engineering school: there is POIREAULT Benjamin from the PHELMA / E3 engineering school of the GRENOBLE INP group in Grenoble in France and DEFORGE Arthur from the ENSI CAEN engineering school in Caen in France. This application is the result of two months of work.

Figure 4 : PHELMA and E3 logo of the GRENOBLE INP group

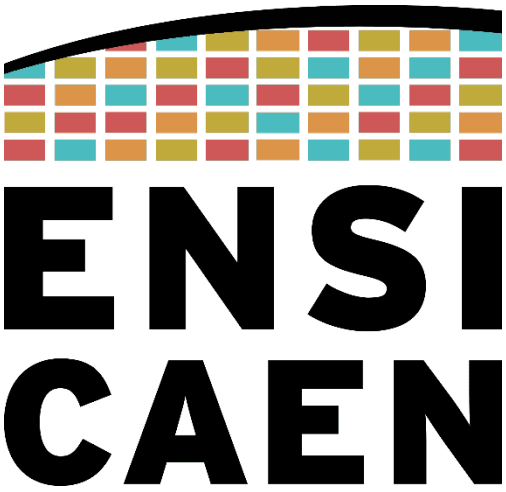

Figure 4 : ENSI CAEN logo

1.3. Collaboration with Tvibit

Tvibit is a multi-purpose production and resource center located in Tromsø, Norway. It supports young people and professionals in the fields of music, film, technology and youth culture, offering studios, equipment, training and funding. The center also incorporates a health station for the well-being of its community. Tvibit aims to foster talent, promote innovation and build a creative and inclusive community, being an essential hub for personal and professional development in the region.

As part of this project, TVIBIT, which is at the origin of many artistic directions, has produced

Figure 5 : TVIBIT logo

many graphic resources used within the application.

1.4. Contributors

The development of this project was made possible thanks to the collaboration and dedication of several people:

1.4.1. Project supervisors

- Benjamin RICAUD
    - Role: Tutor and supervisor of the project
    - Group: ITU Machine Learning Group
    - Position: Associate Professor
    - Contributions: Principal giver of artistic, technical and logistical guidelines. He also supervised the two interns, who were the main actors in the project.
- Samuel KUTTNER
    - Role: Project Supervisor
    - Group: ITU ML Group, UNN PET Imaging Center
    - Position: PhD, Medical Physicist
    - Contributions: Provided extensive artistic and technical guidance.
- Steffen AAGAARD SØRENSEN
    - Role: Project Supervisor
    - Group: Department of Geosciences, ITU Tromsø
    - Position: Researcher
    - Contributions: Provided artistic and logistical guidance.
- Erik HEGGELI
    - Role: Project Supervisor
    - Group: ITU Machine Learning Group
    - Position: Senior Engineer

1.4.2. Main actors (Interns)

- Benjamin POIREAULT

- Role: Project Intern
- Group: Machine Learning Group (PHELMA Engineering School / E3, Grenoble INP)
- Contributions: Responsible for all the AI and the creation of Comfy UI workflows. Involved in the design of the GUI.

- Arthur DEFORGE
  - Role: Project Intern
  - Group: Machine Learning Group (ENSI CAEN Engineering School)
  - Contributions: Designed the graphical interface, the interaction system between the PC and the Raspberry Pi, the communication system between the Raspberry Pi and a phone via a local Wi-Fi hotspot, and wrote most of the project documentation.

1.4.3. Other key contributors

- Cris K. B.
  - Role: Main actor of the artistic part
  - Group: TVIBIT
  - Contributions: Provided numerous graphic resources, as well as artistic and logistical guidelines for the realization of the application.

- Petter BJØRKLUND
  - Role: Communication Advisor
  - Group: ITU Machine learning group
  - Contributions: Provided language and translation data (English, Norwegian, Sami) and gave extensive artistic and logistical advice and guidance.

- Youssef WALLY
  - Role: PhD Fellow
  - Group: ITU Machine Learning Group
  - Contributions: Assisted in the design of the communication system between the Raspberry Pi and a phone via local Wi-Fi hotspot.

- Ismet B.
    - Role: Artistic Contributor
    - Group: TVIBIT
    - Contributions: Provided artistic direction.

2. Technical specifications

The Photo Booth app was developed to be exhibited at the Vitensenter. It was therefore in this context that it was developed and tested on the hardware already present at the Machine Learning Group's exhibition stand in Vitensenter. This part lists the hardware on which the application has been tested and how these elements are physically connected to each other.

2.1. Equipment used and tested

2.1.1.Main Machine

The application runs mainly on the "MSI Thin GF63 12UDX" gaming computer from MSI, it has the following hardware configuration:

- Intel Core i5, 4.4 GHz, 8 Core
- RAM: 16 GB, DDR4 3200 MHz
- NVIDIA GeForce RTX 3050, 6 GB, GDDR6
- 512 GB SSD

As for the operating system, this computer is running Windows 11 64-bit.

2.1.2. Display Screen

For the purposes of the presentation, the screen used is a large touch screen. The exact name of the screen used is "Elo 4303L". The hardware specifications are as follows:

- Touch, 40 touch points
- 42.5”
- Resolution1920 x 1080 @ 60hz

2.1.3. Camera

The Machine Learning Group's exhibition booths at the Vitensenter use webcams from the Logitech brand as a camera, the exact name is "Logitech StreamCam", with the following hardware specifications:

- Resolutions 1920x1080, 60 frames/second
- Supported resolutions: 1920x1080, 1280x720, 960x540, 848x480, 640x360, 320x240

2.1.4. Raspberry Pi

For the specific needs of this application, a Raspberry Pi has been added to the hardware already present at the Machine Learning Group's booth at Vitensenter. Raspberry Pi is a small single-board computer. The Raspberry Pi allows you to create, locally, a wireless network to allow users to retrieve the generated images.

The name of the Raspberry Pi used is "Raspberry Pi 4 Model B" with the following hardware specifications:

- 8GB RAM
- Gigabit Ethernet port
- 16 GB Micro-SD card

2.1.5. Physical interconnection between the different elements

All the elements are connected to each other via cables, the connections are established in the following way:

- Monitor – computer: 1 HDMI cable and 1 USB A cable
- Camera-computer: 1 USB C cable
- Raspberry Pi – computer: 1 Ethernet cable and 1 USB C to USB A cable to power the Raspberry Pi.
- Computer – AC: 1 computer power cable
- Display – mains: 1 display power cable.

The provision whose connection is illustrated by the Figure 6.

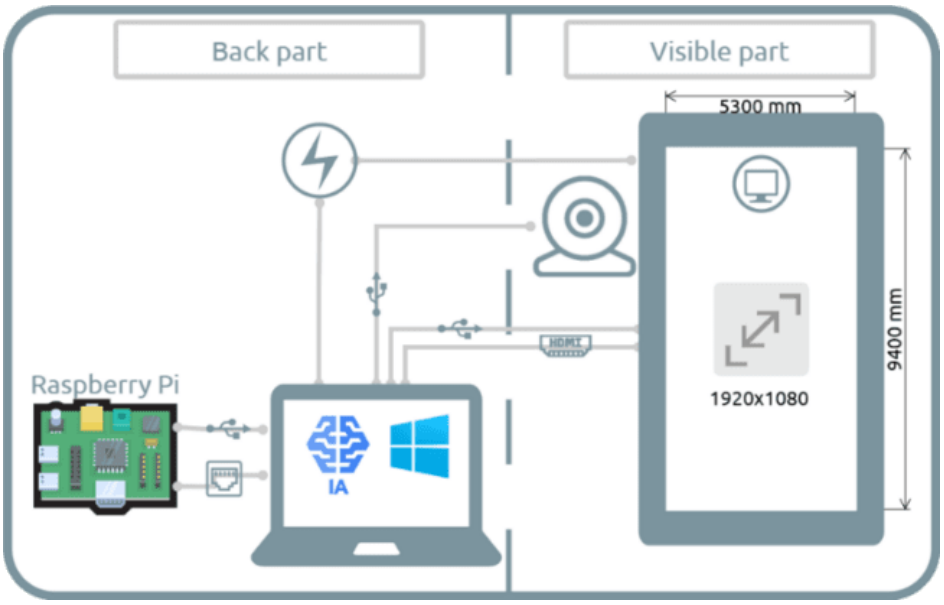

Figure 6 : Graphical representation of the hardware configuration

2.2. Technical constraints

The machine learning group imposed the following constraints: the application had to use completely open source and free tools so that anyone could use it. The application had to be entirely coded in python since it is the most widely used programming language in the school and research world.

As the Vitensenter demonstration facility did not allow access to the internet for the demonstrations, the application had to be fully functional, without internet access.

The TVIBIT has also imposed that the generation by artificial intelligence be done via the Comfy UI application, which is also an open source application. They have also imposed the easy addition of style or customization of the generation by artificial intelligence.

The photobooth application is based on the use of Comfy UI for photo generation but also on several other tools for sharing the image via phone or video capture.

3. The graphical interface

The graphical interface uses PySide6, in other words, the QT library remote from python. The choice of this graphical library was made since it is one of the most optimized libraries that allows you to make the lightest graphical interfaces while remaining entirely open-source.

The application has a specific need for optimization from a graphical point of view since the PC used for the demonstration at the Machine Learning Group booth is a PC with a low configuration. In view of the task of generating images with artificial intelligence to be performed, no resources should be wasted with the GUI.

Many other libraries and packages are used. They are all notified in the requirements.txt present in the app's GitHub repository.

3.1. Description of the views

3.1.1.Standby view

The standby view visible in the Figure 2, is the first one displayed when the app is launched. It has been designed to be extremely resource-intensive. The camera remains permanently on at this stage, but at low resolution with a refresh rate limited to around 15 FPS, which optimizes consumption.

This view also displays a slideshow of sample images generated by the app, with a refresh rate limited to around 25 FPS. This allows the user to discover the possibilities offered by the system.

It also contains a button to change the language of the application.

A simple tap of the screen allows you to exit the standby view and move to the next view.

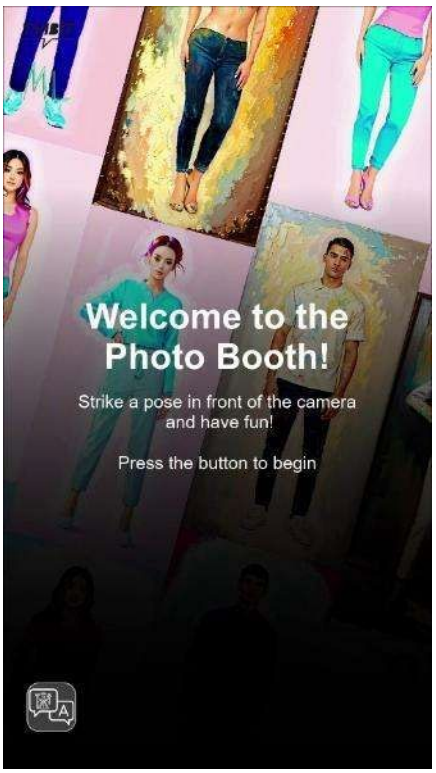

Figure 7 : Overview of the standby view

3.1.2. Selection view

The selection view visible in the Figure 8, allows the user to choose the image generation style and take a picture of themselves via the webcam.

The camera monitor is displayed in Full HD in the background. The display includes:

- A button to take a photo,
- Multiple buttons to select a generation style,
- A text box explaining the instructions to follow,
- The button to change the language.

The operation is as follows:

- If no style is selected and the photo button is clicked, nothing happens.
- If a style is selected, a 5-second countdown will start. Every second, a light flash is displayed; A more intense flash signals the photo taken at the end of the countdown. Taking a photo automatically switches to the next view.

Figure 8 : Overview of the selection view

3.1.3. Waiting view

This view indicates that AI image generation is in progress. This view can be seen in the Figure 9.

It displays:

- An informative message,
- A loading bar synchronized with the progress returned by the ComfyUI app.

The user does not need to do anything at this stage; The interface automatically moves to the next step once the build is complete.

Figure 9 : Overview of the Wait View

3.1.4. Validation and observation view

This view allows the user to view the generated result and choose from several options.

It includes:

- The generated image (in the background),
- A button to change the language,
- Four main buttons:
    - Validate the image to go to the recovery screen (and display the conditions of use),
    - Reject the image, which deletes the data from the memory (RAM and disk) and returns to the selection view,
    - Regenerate a new version from the same photo, with a new random seed,
    - Review the original photo to better understand the transformation carried out.

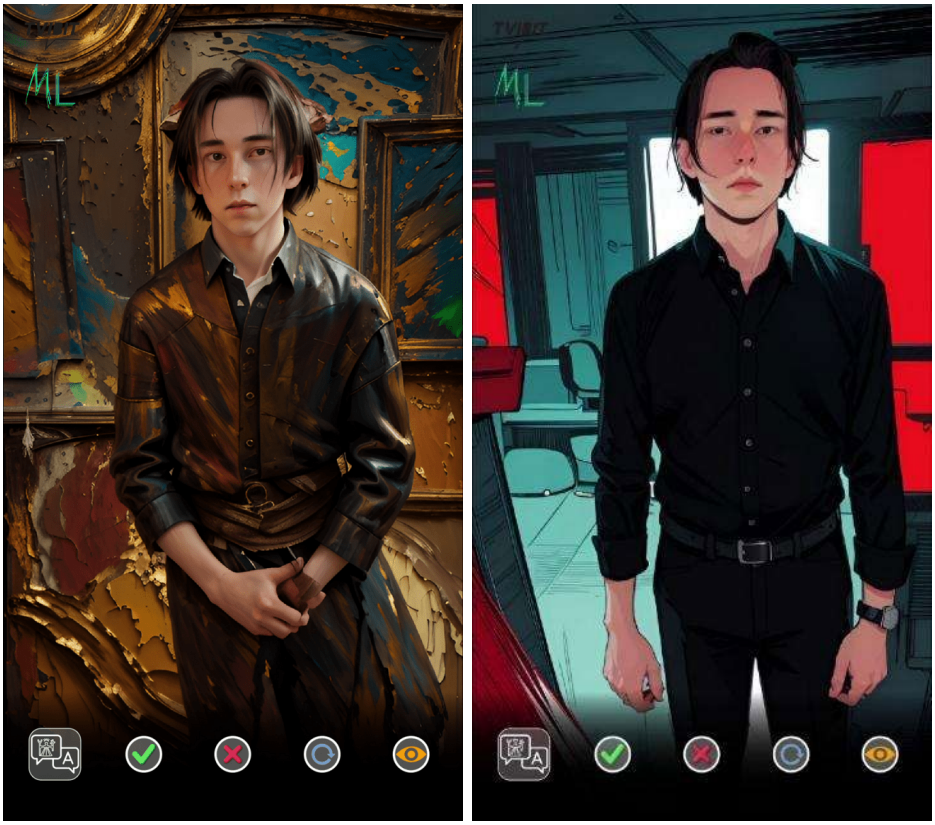

Figure 10 : Overview of the validation and observation view.

3.1.5. Terms and Usage Acceptance View

This view, visible in the Figure 11, displays the usage and rights rules associated with the generated images.

Although no images are kept (they are deleted systematically), this step helps to inform the user.

Two buttons are available:

- Accept: Proceed to the image recovery screen,
- Deny: Delete the data in memory and return to the selection view.

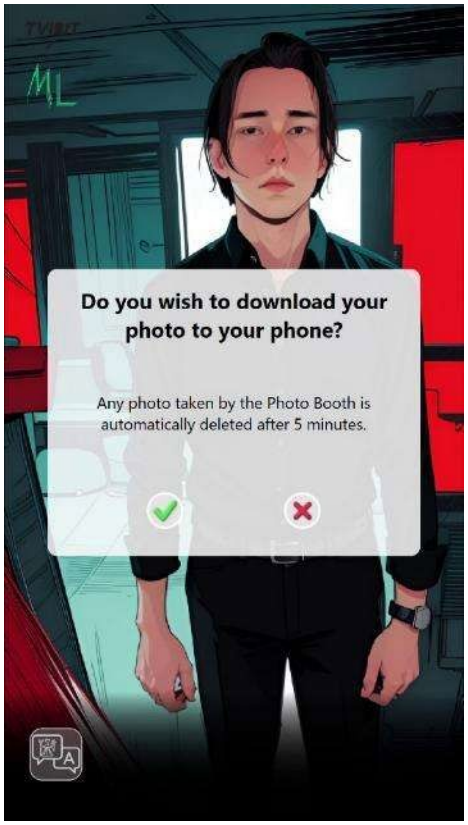

Figure 11 : Overview of the acceptance view of terms and uses

3.1.6. Image Recovery View with Phone

This view, visible in the Figure 12, allows the user to download the generated image to their phone.

It contains:

- A QR code that appears after a few seconds,
- Three buttons to display recovery instructions according to the type of phone (Android, iPhone, Samsung),
- A final button (cross) to confirm that the image has been recovered.

When the latter button is clicked:

- All images (RAM + disk) are deleted,
- The wifi LAN used for the transfer is disabled,
- The app automatically returns to the selection view.

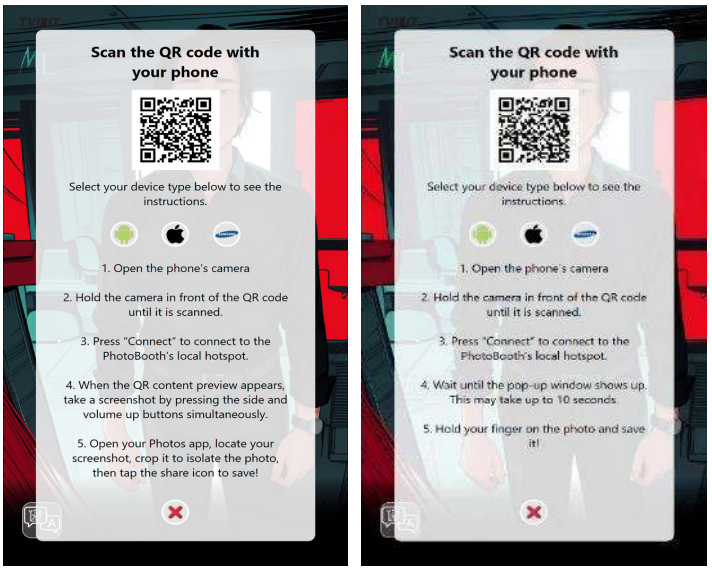

Figure 12 : Preview of the image recovery view with phone

3.1.7. Application language switching interface

This interface is accessible from each view via a dedicated button.

The user can choose the display language from the three available languages:

- French (default),
- English
- Norwegian Sami.

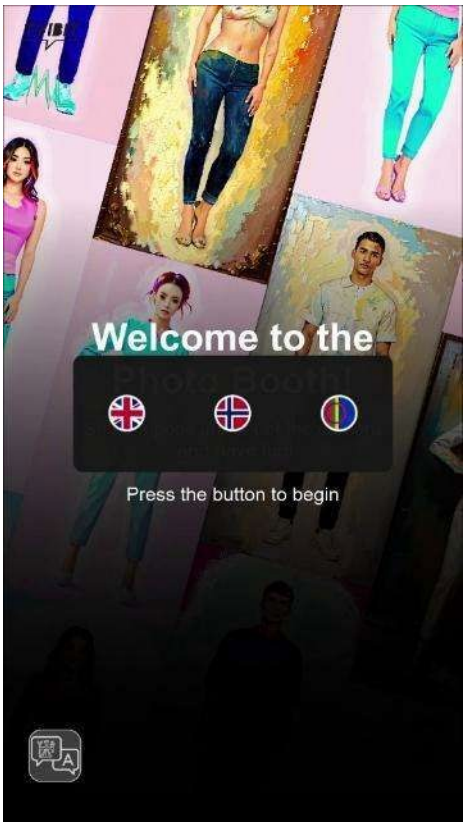

Figure 13 : Overview of the app's language switching interface

4. Interaction between the GUI and ComfyUI

Image generation by artificial intelligence

4.1. ComfyUI Overview

ComfyUI is an open-source tool designed to build, visualize, and run artificial intelligence image processing pipelines. The ComfyUI logo is featured in the Figure 7. It is based on a modular system in which each operation (image loading, model application, transformation, saving, etc.) is represented as a node. These nodes are connected to each other in a visual scheme called a workflow.

Figure 14 : Comfy UI logo

This workflow is a graphical processing chain, in which data (images, texts, prompts, etc.) circulate between the different stages of the process. An example workflow is shown Figure 8. The whole thing is configurable, saveable, reusable and exportable in a readable and editable JSON format, which makes it easy to integrate into external projects such as the Photo Booth

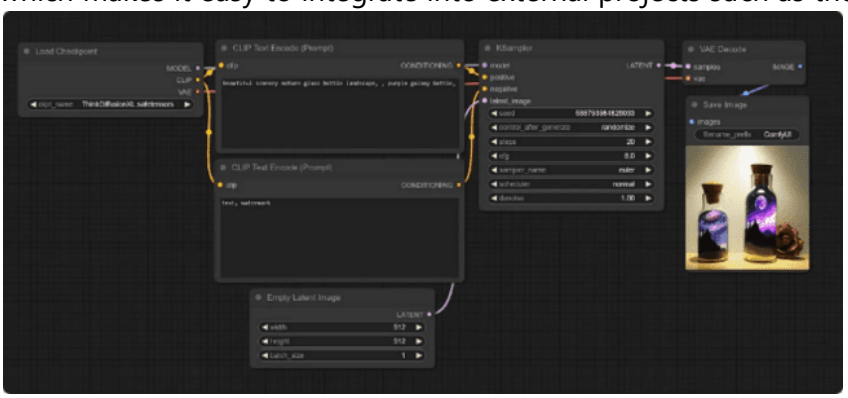

Figure 15 : Example of a workflow

application.

ComfyUI makes it possible to:

- Integrate compatible AI models (diffusion, upscaling, style transfer, etc.),

- Build complex workflows without coding,
- Execute workflows via a graphical interface or API, in local server mode.

As part of this project, ComfyUI plays a central role in generating images from the user's photo and the selected style. The application communicates with ComfyUI in the background, preparing a dynamic workflow, which is then sent to the ComfyUI engine for automatic processing.

This integration is based on two main protocols:

- HTTP, for sending the workflow,
- WebSocket, to track execution in real time.

The whole of this interaction is detailed in the following sections.

4.2. Artificial Intelligence Image Generation

The generation of images by artificial intelligence is based on ComfyUI, used here as the processing engine. For each build, the application loads a preconfigured ComfyUI workflow in JSON format, which is then dynamically modified before being sent to runtime.

Each build style selected by the user corresponds to a custom prompt, stored in an in-application dictionary. This prompt is automatically injected into the workflow, at the time of preparation, in order to guide the AI model towards a visual rendering consistent with the requested style (painting, sketch, cartoon, etc.).

The application uses a dynamic workflow selection system:

- If a JSON file named as the style (e.g. cartoon.json) is found in the workflows directory, this file is used as the base.
- Otherwise, a default file (default.json) is used.

The selected JSON file is then copied to RAM and modified with the following settings:

- The prompt of the chosen style,
- A random seed,
- The path to the input image (input.png),
- The name of the output image (output.png).

The input and output files are always placed in the input/ and output/ directories of ComfyUI, respectively.

4.3. Communication via HTTP and WebSocket

ComfyUI can be used without a GUI, in local server mode, accessible via a pre-configured IP address. The application then interacts with this server through two channels:

- The HTTP protocol to send the workflow to be executed,
- The WebSocket protocol to receive real-time information on the progress of the build.

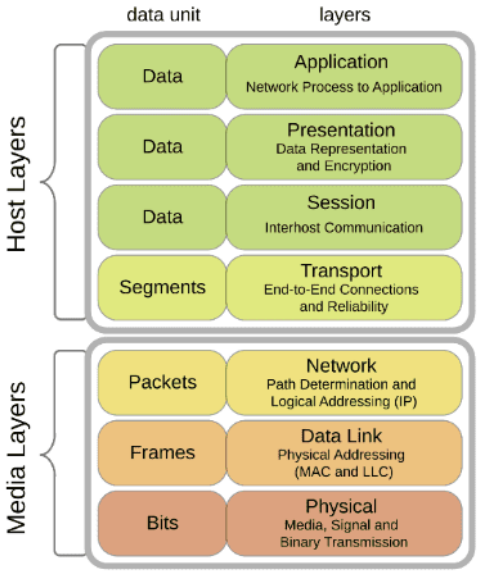

Figure 16 : Representation of the OSI model

4.3.1. Communication HTTP

The HTTP protocol (application layer of the OSI model — see Figure 10) is used to send the main query containing, the communication is shown Figure 9 :

- The customer ID,
- The modified workflow,
- Output parameters,
- An indication to force the execution.

ComfyUI then executes this workflow asynchronously.

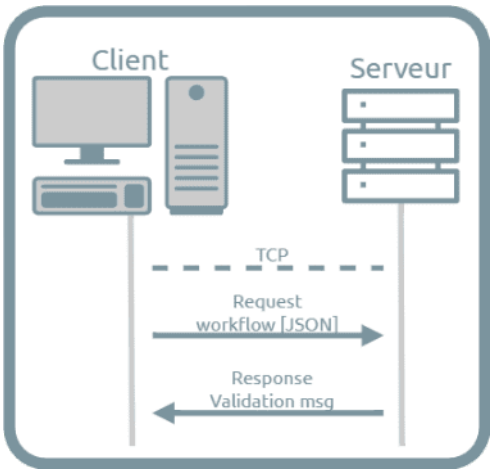

Figure 17 : HTTP Communication

4.3.2. Communication WebSocket

At the same time, a WebSocket connection is opened with the server. This bidirectional protocol (see Figure Y) allows the server to continuously send information about:

- The status of the nodes (ongoing steps),
- Quantitative progression (percentage progression values),
- Any error or success messages.

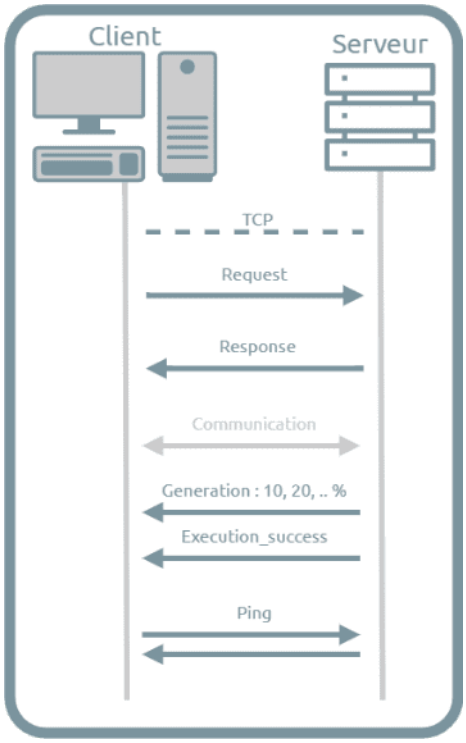

Figure 18 : WebSocket Communication

The ImageGeneratorAPIWrapper class, used by a dedicated thread, is responsible for all of this communication. It also maintains a secondary thread that regularly pings to check the status of the WebSocket connection.

4.4. Progress tracking and interface update

Each message received through WebSocket is processed to extract the progress of the different nodes in the workflow. This data is aggregated and normalized to the total number of steps, in order to derive an overall percentage of completion. This percentage is then emitted via a Qt signal and used to update the loading bar displayed on the screen during the standby view.

4.5. File recovery and cleanup

When the build is complete, the server sends a completion message (done, execution_success, and so on). The program:

- Closes WebSocket connections,
- Retrieves the generated image from the output/ folder,
- RAM load (in the form of QImage),
- Then deletes the input and output files from the disk (input.png and the images in the output/ folder), so that only a temporary RAM copy remains.

4.6. Failure Management

If a failure is detected (error message received, timeout, or no ping response), the application:

- Closes open connections,
- Deletes temporary files,
- And displays the original photo in the validation view, as if it had been generated.

This approach allows the user to still access the validation screen and retry a shot with the same snapshot.

5. Interaction between the PC and the Raspberry Pi

The interaction between the Photo Booth application (on the PC side) and the Raspberry Pi allows the generated image to be transferred to the Pi, in order to make it accessible via a temporary Wi-Fi network. This communication is local, secure and isolated, without any connection to the Internet. The whole process is triggered by the PC, which plays a master role in the exchange.

5.1. Local communication and protocol used

The PC is connected to the Raspberry Pi via a direct Ethernet cable, over a network interface configured with a fixed IP address assigned to the Pi (e.g. 192.168.10.2). No routers, switches, or internet access are required for this exchange. The Raspberry Pi runs a secure HTTP server (Flask) listening on port 5000, which is automatically launched at startup.

The application on the PC launches a dedicated thread (ThreadShareImage) that uses a HotspotClient object to:

- Send the generated image (in PNG format) to the Raspberry Pi via an HTTP POST request to the /share URL,
- Receive in return a QR code in binary format (base64) as well as the identifiers of a Wi-Fi network (SSID + password).

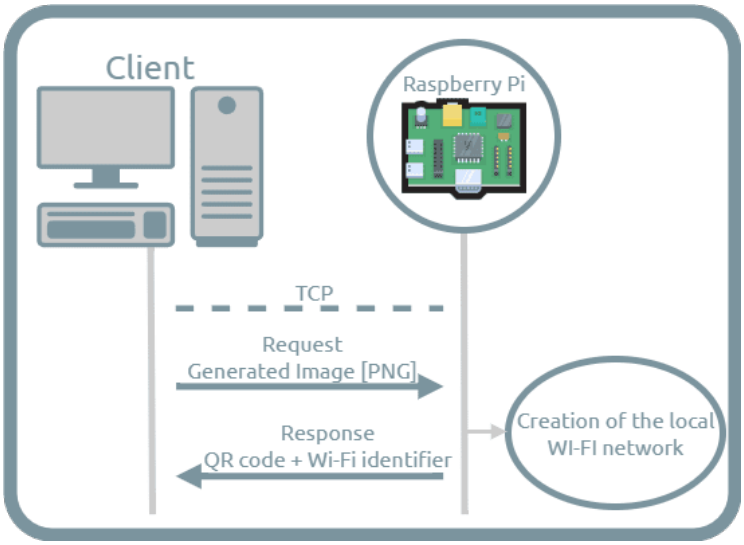

Figure 19 : Communication Application - Raspberry Pi

This request is made asynchronously so that the UI doesn't crash, even if it's slow or fails.

5.1.1. Secure HTTP server (Flask) on the Raspberry Pi

The server running on the Raspberry Pi is a Python application based on the Flask micro-framework, widely used to develop simple and lightweight web APIs. As part of this project, Flask allows you to manage HTTP requests sent by the PC to initiate the image sharing process.

The Flask server is configured to:

- Listen on the local IP address of the Raspberry Pi (e.g. 192.168.10.2),
- Use port 5000,
- Operate in HTTPS mode, using a self-signed local SSL certificate (cert.pem and key.pem), ensuring that exchanges cannot be intercepted or modified by a third party,
- Expose a single POST /share route to:
    - Receive an image sent by the PC,
    - Trigger the generation of the hotspot and QR code,
    - Return a JSON response containing the Wi-Fi IDs and the encoded QR code.

This server is launched automatically when the Raspberry Pi starts. It operates independently of the internet and does not require any external access, making it ideally suited for use in isolated environments or in contexts where the security of personal data is crucial.

5.2. Raspberry Pi Side Processing

Upon receiving the image, the Raspberry Pi:

1. Checks the received file to make sure it is a valid image.
2. Generates random Wi-Fi credentials (SSID and password).
3. Configures and starts the services needed to create a secure local hotspot:
    a. hostapd (creation of the Wi-Fi network),
    b. dnsmasq (IP address distribution),
    c. nodogsplash (automatically redirect users to a custom page).
4. Copies the received image to the /etc/nodogsplash/htdocs web directory.
5. Dynamically modifies a splash.html page to include the image and an automatic download link.

6. Generates a QR code containing Wi-Fi credentials.

7. Starts a 5-minute timer to automatically mute the hotspot and clear the image after the time is up.

5.3. Reception of the QR code by the PC

Once the process is complete, the Pi returns to the PC a JSON response containing:

- The SSID of the Wi-Fi network created,
- The associated password,
- The base64-encoded QR code, ready to be displayed to the user.

The PC-side thread saves this data and makes it accessible to the GUI.

In the event of an error (failed connection, inaccessible server, invalid image, etc.), an error icon is used as a replacement QR code, without displaying an explicit message: the visual symbol is enough to indicate the problem to the user.

5.4. Resetting the share

When the user clicks the final close button (cross) in the GUI, a reset() method is called. This:

- Sends an error image to the Raspberry Pi again,
- Triggers a hotspot regeneration with this empty image,
- Replaces any previously shared images, ensuring that no user images remain available.

This step ensures data security and privacy, since any images still on the Pi are overwritten and the temporary network is disabled.

6. Communication between the Raspberry Pi and a phone via local Wi-Fi hotspot

6.1. Overview

To allow users to easily retrieve the image generated by the artificial intelligence, the application sets up a local sharing system via a Wi-Fi network emitted by the Raspberry Pi. This temporary network is fully isolated, with no internet access, and does not require any third-party apps to be installed on phones.

The connection is made by scanning a QR code displayed on the graphical interface. Once logged in, users are automatically redirected to a page containing the image, which they can view or download depending on their device's behavior.

The operation is based on several software components installed and configured on the Raspberry Pi:

- hostapd to transmit a local Wi-Fi network;
- dnsmasq to manage DHCP and DNS;
- nodogsplash for automatic redirection to a local page (captive portal);

A Flask server to receive the image, configure the services, and generate the QR code.

This device works completely autonomously, without Internet, and allows several users to connect simultaneously.

6.2. The principle of the captive portal

A captive portal is a mechanism used to intercept the first web browsing attempt of a device connected to a network, in order to redirect it to a specific locally hosted page. This technique is used in public places such as hotels, universities or cafes to force authentication, present conditions of use or offer different access offers.

In a network with a captive portal:

- All DNS queries are redirected to a single IP address (often that of the local gateway);
- HTTP traffic is intercepted and redirected by a firewall rule or local proxy server;

- A specific web page (called a splash.html page) is then automatically displayed to the user.

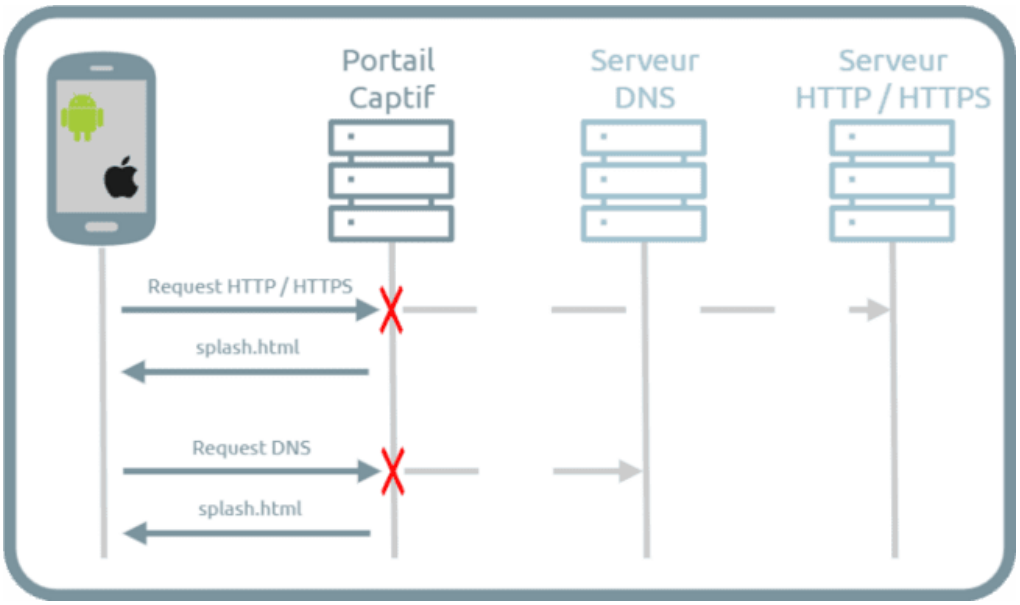

Figure 20 : Principle of the captive portal

This method is technically based on the interception of network packets, especially those related to HTTP and sometimes HTTPS protocols. In the latter case, the redirect may fail or generate a security alert because of the lack of a valid certificate. For this reason, captive portals primarily redirect HTTP traffic. Specifications such as RFC 7710 also provide for the ability to report the existence of the captive portal directly via DHCP or RA (Router Advertisement), but not all systems support them.

On modern operating systems, such as Android or iOS, automatic mechanisms detect the presence of a captive portal by testing access to external servers. If this access fails and a redirection service is detected, the device automatically displays the local page hosted by the network.

6.3. Implementation on the Raspberry Pi

6.3.1. Creating the Wi-Fi Network

The Raspberry Pi uses hostapd to emit a temporary Wi-Fi network, dynamically configured for each sharing session. The network name (SSID) and password are randomly generated and applied to the /etc/hostapd/hostapd.conf configuration file. This network is deliberately isolated: it does not give access to any external service and is only used for the local transfer of the image.

These identifiers are integrated into a Wi-Fi-compliant QR code, which is then displayed on the graphical interface. This QR code contains only the SSID, the password, and a hidden network transmission indication.

6.3.2. IP address assignment and DNS management

The dnsmasq service is responsible for automatically assigning an IP address to clients via DHCP in the range 192.168.5.10 to 192.168.5.100. It also acts as the local DNS server, with the following settings:

- All DNS queries are redirected to the Pi's local address (192.168.5.1);
- An A record is added for the photobooth.ml name, so that the user can access the portal even by typing this URL into a browser.

This behavior simulates a fully functional network, while intercepting requests as soon as they are made.

6.3.3. Captive Portal with NoDogSplash

The nodogsplash software is used to intercept HTTP connections and redirect users to the splash.html page, hosted on the Raspberry Pi. This page contains the image transferred by the PC, a download link, and a script allowing automatic download if the browser allows it.

The /etc/nodogsplash/nodogsplash.conf configuration file is adjusted to:

- Listen on the wlan0 interface;
- Allow only access to the redirect page;
- Automatically redirect all HTTP connections to `http://192.168.5.1/splash.html` ;
- Apply a session timeout of 300 seconds.

As soon as a phone connects to the network, the system detects the lack of internet connectivity and automatically launches the captive portal page.

6.3.4. How the Flask server works

The Flask server is responsible for receiving the image to be shared. It exposes a /share entry point that can only be accessed via the wired network between the PC and the Raspberry Pi. This server:

- Receives the image via a multipart POST request;
- Checks the validity of the image;

- Copy the image to nodogsplash's local web directory;
- Dynamically updates the HTML splash.html file to embed the image;
- Generates the QR code with the network credentials;
- Restarts services (hostapd, dnsmasq, nodogsplash);
- Returns a JSON response that contains the necessary information, as well as the system logs.

A timer is also launched to automatically stop services after five minutes and delete the temporary image.

6.3.5. Behavior on different mobile systems

The behavior varies depending on the operating system of the phone used:

Newer Android devices (e.g. Google Pixel): The system automatically detects the captive portal. A notification offers to connect. Clicking on it will automatically open the page, the image will be displayed, and the download will begin without additional intervention.

Recent iOS devices (e.g. iPhone): Portal detection works correctly, the page opens automatically and the image is displayed. However, the image is blocked from downloading. Attempting to save the image cuts off the network connection. The only solution is to take a screenshot.

Newer Samsung devices: An alert is displayed when you log in, indicating that there is no internet access. The user must manually validate the connection, then open a browser and enter the http://photobooth.ml URL to access the page.

6.3.6. Limitations and security

No internet access is possible during the sharing session, which is voluntary to ensure network isolation.

Multiple phones can connect simultaneously, up to the 250 user limit set in the configuration.

The session is automatically terminated after five minutes, after which all services are stopped and files are deleted.

The generated QR code contains only the network credentials, without any personal information.

All Raspberry Pi services are run locally, and do not require any exchange to the outside world.

7. Application architecture

The PhotoBooth application is designed as a modular system with several interdependent components: a PySide6 graphical interface for user interaction, a communication engine with ComfyUI for image generation, and a set of network services for sharing results via the Raspberry Pi.

It manages both the hardware flows (camera, touch screen), software processing (preparation and sending of JSON workflows, real-time monitoring of the generation via WebSocket), and the distribution of the generated images (QR code, captive portal).

The software structure is based on a hierarchy of modular classes that facilitate the reuse of code, the management of transitions between views (standby, capture, display of results) and the smooth integration of animations and content. The multi-threaded architecture allows interactive tasks (interface) to be separated from network or processing operations, ensuring a smooth and responsive user experience.

The different interactions between the different elements are illustrated in the Figure 21.

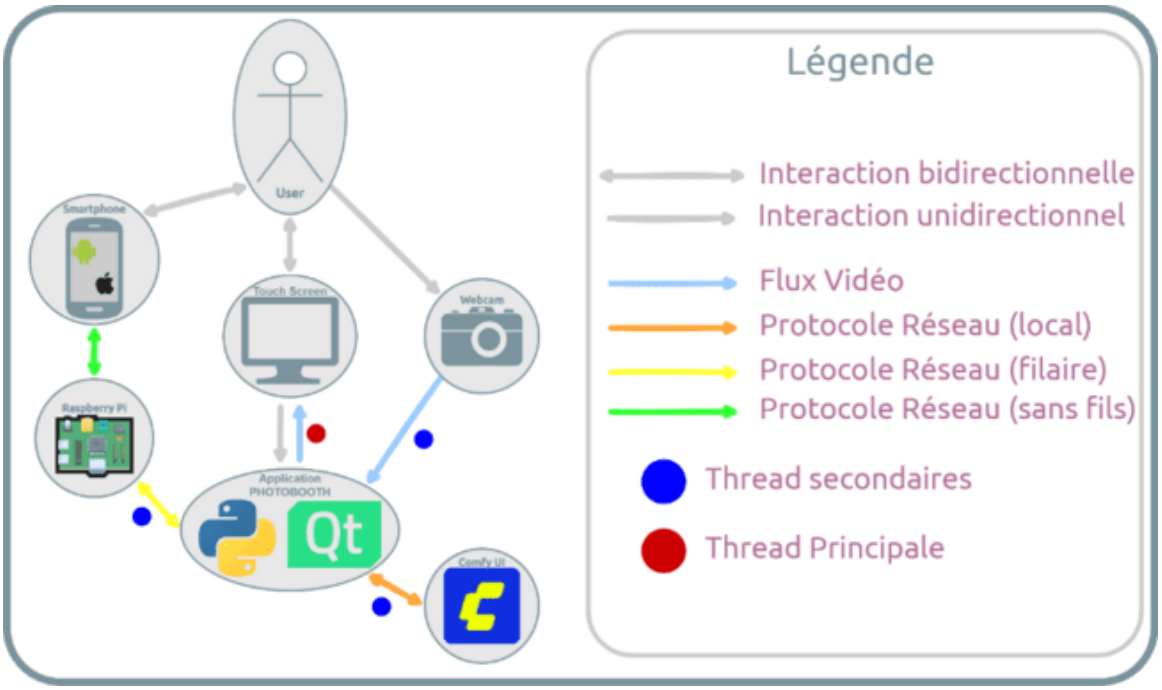

Figure 21 : Representation of the interactions between the different elements of the system

7.1. BaseWindow: Interface Foundation

At the heart of this architecture is the BaseWindow class, inherited from QWidget, which serves as the parent class for the two main views of the application: SleepScreenWindow (screensaver) and MainWindow (active screensaver). This class abstracts common behaviors and interface elements, such as:

- The transparency of the window and its full-screen positioning, always in the foreground.
- The management of a grid system (QGridLayout) on which buttons, labels or overlays are dynamically placed.
- Integration of custom interactive buttons (including the language selection button).
- The management of overlays (visual overlays), such as a QR code, rules of use, a loading overlay or a language selector.
- The ability to dynamically display headers, temporary messages (QToolTip), or a loading bar.
- A mechanism for systematic cleaning of the active elements each time the window state changes (on_leave, cleanup).

This provides a robust and reusable graphics toolbox for all other windows.

Two main windows: SleepScreenWindow and MainWindow

The two main interfaces of the application inherit directly from BaseWindow:

- SleepScreenWindow represents the pending state. It is very uncluttered, with only a message and a language change button. It also displays a scroll-generated animated background (managed via a specific overlay).
- MainWindow is the main interface used when taking photos and generating images. It uses a background from either the camera feed or the AI-generated image (handled by the BackgroundManager class). This window has three display modes:
    - Default mode: display of style buttons, photo button, etc.
    - Wait mode: Display the photo taken and disable other interactions.
    - Validation mode: display of the generated image and validation/rejection buttons.

These modes are defined by specific methods that dynamically change the visible elements, the active buttons, and the displayed background.

7.2. Custom Buttons

The application defines a hierarchy of classes for button management:

- Btn is a custom class inheriting from QPushButton, on which all the buttons in the interface are based.
- Two styles of buttons derive from this base:
    - Style 1 buttons: round buttons with icon (e.g. photo button, validation, return, etc.).
    - Style Buttons 2: Rectangular buttons with centered text, white border, and background texture (e.g., style selection). The text displayed depends on the language selected through the LangManager, and the background texture is related to the name of the style it represents.

All buttons are organized via a Btns class and inserted into the BaseWindow grid.

7.3. Window Manager: WindowManager

The transition between the two main views is provided by the WindowManager, which:

- Initially displays SleepScreenWindow, with the animated background overlay active (based on scrolling columns of images).
- When a user interaction is detected, it switches to MainWindow, and reverses the order of the windows, playing a smooth transition animation (visual scrolling up or down).
- Also manages the display of the scrolling overlay, either in the background during sleep mode, or in the foreground temporarily during the transition, before disappearing.

This behavior is driven by a TimerUpdateDisplay class, which runs continuously at a configurable frequency (by default 20 or 80 FPS depending on the phase), and notifies registered objects (including the overlay) to update their display.

7.4. Ancillary Components

Several other modules complete this architecture:

- LangManager: manages the translation files, the display of text according to the selected language, and the updating of all the textual elements of the interface.
- BackgroundManager: Controls the background of the main interface, between the camera feed (displayed live) and the generated image.
- ScrollOverlay: An overlay containing a system of automatically scrolling columns of animated images.

- Multiple threads (ThreadCamera, ThreadGeneration, and so on): Initiate actions such as taking a photo or generating an image without blocking the GUI.

INSTALLING THE APP ON THE BUILD PC

The PhotoBooth graphics application must be installed on a computer dedicated to photo capture and image generation via artificial intelligence. It works in tandem with ComfyUI, an open-source workflow-based image generation tool. The GUI and ComfyUI must coexist in the same parent directory.

8. Prerequisite

Before installing the PhotoBooth app, it is imperative to properly install and configure ComfyUI, which is the artificial intelligence image generation engine used by the app.

8.1. Installing ComfyUI

ComfyUI is an open-source software for creating and executing image generation or transformation workflows. It can be downloaded from its official deposit

It is recommended to use the Electron version of ComfyUI on Windows, which provides a standalone interface that is easily executed.

8.2. Dedicated environment

ComfyUI must be installed in its own virtual environment, separate from that of PhotoBooth. This helps avoid dependency conflicts between the two applications.

8.3. Additional dependencies

To work properly with PhotoBooth, ComfyUI needs to include some custom nodes. These must be installed in the custom_nodes/ folder of ComfyUI.

A later section of this report will describe specifically:

- What nodes are required,
- How to install them,
- And how to check that they are working properly with the workflows provided.

9. Recommended Directory Structure

The working inventory should be structured as follows:

```
<parent folder>/
├─── ComfyUI/
│   ├─── input/
│   ├─── output/
│   ├─── ...
├─── PhotoBooth/
│   ├─── gui_classes/
│   ├─── comfy_classes/
│   ├─── workflows/
│   ├─── ...
```

This organization allows the PhotoBooth application to directly access ComfyUI's input and output directories, without additional configuration.

10. Installing PhotoBooth

Here are the steps to install and launch the PhotoBooth app:

10.1.1. Clone the repository to the correct directory

```
cd <path/to/folder/parent>
git clone https://github.com/ADEFORGE/PhotoBooth.git
```

10.1.2. Create a virtual environment dedicated to the application

Linux / macOS

```
sudo apt update && sudo apt install -y python3-venv
cd PhotoBooth
python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

Windows (PowerShell)

```
cd PhotoBooth
python -m venv .venv
.\.venv\Scripts\Activate.ps1
pip install --upgrade pip
pip install -r requirements.txt
```

10.1.3. Install the necessary ComfyUI dependencies

ComfyUI must be properly configured with all the nodes required to run the workflows used by the application. A dedicated section later in this report details the steps required to set up this configuration.

The default workflow requires 5 custom node packs. To install a package, simply use the ComfyUI manager:

Figure 22 : Node installation buttons, Comfy UI

Then simply search for and install:

- ComfyUI_IPAdapter_plus (github.com/cubiq/ComfyUI_IPAdapter_plus)
- comfyui_controlnet_aux (github.com/Fannovel16/comfyui_controlnet_aux)
- ComfyUI-Custom-Scripts (github.com/pythongosssss/ComfyUI-Custom-Scripts)

- efficiency-nodes-comfyui (github.com/jags111/efficiency-nodes-comfyui)
- was-node-suite-comfyui (github.com/ltdrdata/was-node-suite-comfyui)

These nodes can also be installed manually by going to the `ComfyUI\custom_nodes` directory and then making a `'git clone'` of each of the GitHub repositories.

ComfyUI must then be restarted. To test if the workflow works, import the default.json workflow into ComfyUI.

The required templates are as follows:

- models/controlnet:

```
huggingface.co/lllyasviel/control_v11f1p_sd15_depth
huggingface.co/lllyasviel/control_v11e_sd15_ip2p
```

- models/checkpoints: (can be replaced by any SD 1.5 model)

```
civitai.com/models/4384/dreamshaper
```

Other models are required for the IP-Adapter, the procedure is detailed in the github.com/cubiq/ComfyUI_IPAdapter_plus extension documentation.

11. Automatic app launch

(Windows only)

ComfyUI and PhotoBooth must be launched in their respective environments. To make this process easier, a customizable .bat script allows both to be launched automatically on Windows.

Example of a .bat script:

```
@echo off
del /f /q "C:\AI Demos\PhotoBooth\app.log"
start "" "C:\Users\vitensenteret_ml3\AppData\Local\Programs\@comfyorgcomfyui-electron\ComfyUI.exe"
cd /d "C:\AI Demos\PhotoBooth"
python .\main.py
pause
```

- This script does the following:
- Deletes the previous log file.
- Launches the ComfyUI executable (via ComfyUI Electron).
- Starts the PhotoBooth app.
- Leaves the window open (useful for checking for any error messages).

It is necessary to adapt this .bat file to your own configuration, including the paths to ComfyUI.exe and the PhotoBooth folder.

12. General configuration and aesthetics of the application

The constante.py file contains all the technical and aesthetic constants necessary for the operation of the PhotoBooth application. It is automatically imported by several modules of the GUI and the image generation API.

This file contains, in particular:

- Interface appearance settings (colors, fonts, button styles, dimensions),
- The messages displayed on the screen,
- Paths to working folders (input/output),
- ComfyUI connection settings,
- The styles and prompts associated with each AI transformation.

It is possible to fine-tune the appearance and behavior of the application by modifying this file.

Important note: Some constants present in this file are no longer used in the current version of the interface, but have been retained to ensure compatibility with older imports and avoid runtime errors.

12.1. Customization possible

Here are the main elements that you can edit in this file:

- Colors: The COLORS dictionary centralizes all the colors used.
- Text and button styles: Constants like BUTTON_STYLE, TITLE_LABEL_STYLE, DISPLAY_LABEL_STYLE, etc., precisely define the visual rendering.
- Size and layout: Ratios (DISPLAY_SIZE_RATIO, HUD_SIZE_RATIO) or margins (GRID_MARGIN_TOP, etc.) allow you to adjust the display on screens of various sizes.
- Custom AI prompt: The dico_styles variable allows you to associate each style with a unique prompt used in the build via ComfyUI.
- Timeout settings: SLEEP_TIMER_SECONDS, COUNTDOWN_START, etc.

12.2. Enable debug logs

There are two constants that control the amount of logs displayed in the console or saved in log files:

```
DEBUG = False
DEBUG_FULL = False
```

DEBUG = True: Enables standard logs that are useful for monitoring normal execution.

DEBUG_FULL = True: Enables all logs, including those related to frequent operations (e.g., refreshing the image on the screen), resulting in a very verbose output.

12.3. Change connection addresses (ComfyUI, Flask, WebSocket)

If you have changed the network configuration or launched the services on different ports/addresses, it is necessary to update the following constants:

```
WS_URL = "ws://127.0.0.1:8188/ws"
HTTP_BASE_URL = "http://127.0.0.1:8188"
HOTSPOT_URL = "https://192.168.10.2:5000/share"
```

WS_URL: The URL of the WebSocket to receive build notifications from ComfyUI.

HTTP_BASE_URL: HTTP URL to send requests to ComfyUI.

HOTSPOT_URL: local address of the Raspberry Pi Flask server (see dedicated section).

12.4. Disable hotspot image sharing

If you don't have a Raspberry Pi connected or you don't want to enable Wi-Fi sharing (captive portal), it's imperative to disable this option in constante.py:

```
ShareByHotspot = True
```

Replace it with:

```
ShareByHotspot = False
```

13. Installing on Raspberry Pi

13.1. Preconfigured per-frame method

Advance warning

The provided image was compressed using Pishrink to reduce its size to approximately 9 GB. It contains an already configured version of the system with all the services and dependencies necessary for the operation of the PhotoBooth.

13.1.1. Uploading the image

The image is available in .img format via a Google Drive link (provided separately). Download this file and keep it in an accessible location on your PC.

13.1.2. Preparing the micro-SD card

Insert the micro-SD card into your computer's drive.

Identify the device that corresponds to the adapter (for example, /dev/sdb) using the command:

```
LSBLK
```

Check the device name carefully to avoid overwriting a disk with important data.

Copy the image to the micro-SD card by adapting the paths:

```
sudo dd if=~/photobooth_rpi.img of=/dev/sdX bs=4M status=progress
```

Replace `~/photobooth_rpi.img` with the path to the uploaded image.

- Replace `/dev/sdX` with the device you detected earlier (e.g., /dev/sdb).

13.1.3. Default Login Information

- Hostname: `uitml`
- Username: `uitml`
- Password: `ITU123`

By default, the Raspberry Pi can be reached on the wired network at 192.168.10.2.

You can therefore connect directly to it via SSH from a PC:

```
ssh uitml@192.168.10.2
```

This IP configuration is valid only for the Ethernet connection.

13.1.4. End of installation

Once the command is complete and the data sync is complete, cleanly eject the micro-SD card, insert it into the Raspberry Pi, and then boot it up.

Basic network and software configuration is already built into the image.

13.2. Step-by-step configuration method

13.2.1. Installing the prerequisites (Raspberry Pi)

Before configuring the hotspot and the Flask server, it is necessary to install a number of packages and software components. This step assumes that the Raspberry Pi is temporarily connected to the Internet, ideally via USB tethering with a phone (so as not to use Wi-Fi, which will be reserved for the hotspot). The use of an Ethernet cable for connection to the PC is also to be avoided at this stage, as it will later be used for the direct link to the generation computer.

Temporary Internet connection

Connect the Raspberry Pi to the phone via USB and activate tethering.

Verify that Internet access is functional, for example with:

```
ping -c 4 google.com
```

Installing System Packages

Update the package list and install the necessary dependencies:

```
sudo apt update
sudo apt install -y hostapd dnsmasq iptables-persistent git libmicrohttpd-dev
build-essential python3 qrcode
sudo apt install -y python3-pip python3-flask python3-pil
```

Installing NoDogSplash

```
git clone https://github.com/nodogsplash/nodogsplash.git
cd nodogsplash
make
sudo make install
```

Installing Python dependencies

```
pip install requests Pillow
```

13.2.2. Configuration de la Raspberry Pi

This section describes the configuration required to turn the Raspberry Pi into a Wi-Fi access point with a captive portal and a Flask server for image exchange.

The entire configuration was tested under Debian 12 (Bookworm) in ARM64 architecture.

General File Structure

The system and portal configuration files are located in the standard Raspberry Pi directories:

Path	*Role*
`/etc/hostapd/hostapd.conf`	Wi-Fi Access Point Setup
`/etc/dnsmasq.conf`	Local DHCP/DNS service for connected clients
`/etc/nodogsplash/nodogsplash.conf`	Setting up the captive portal
`/etc/nodogsplash/htdocs/splash.tmpl`	HTML page automatically displayed to customers
`/etc/nodogsplash/htdocs/splash.css`	Related Style Sheet
`/home/uitml/flask_rpi/app.py`	Flask server for image exchange
`/home/uitml/flask_rpi/cert.pem`	Self-signed SSL certificate (to be created by the user)
`/home/uitml/flask_rpi/key.pem`	Associated private key (to be created by the user)

A copy of the portal files (splash.tmpl, splash.css, load.png, error.png...) is also present in the application directory:

```
hotspot_classes/in_py/configuration_files
```

This folder serves as a backup in case of reinstallation.

Network configuration

The Raspberry Pi uses a static IP address on the wlan0 Wi-Fi interface and has no internet connection.

The configuration is defined in `/etc/dhcpcd.conf` :

```
interface eth0
nogateway
static domain_name_servers=8.8.8.8 1.1.1.1
interface wlan0
static ip_address=192.168.5.1/24
nohook wpa_supplicant
```

Network and NAT commands

After applying the configuration files, run the following commands:

```
# Static IP on wlan0
sudo ip addr flush dev wlan0
sudo ip addr add 192.168.5.1/24 dev wlan0
sudo ip link set dev wlan0 up

# Enable IPv4 forwarding
sudo sysctl -w net.ipv4.ip_forward=1
```

```
 # NAT Rules and Forwarding
 sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
 sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state
RELATED,ESTABLISHED -j ACCEPT
 sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

Services to be activated

The necessary services are:

```
 sudo systemctl enable hostapd dnsmasq nodogsplash flask_rpi.service
 sudo systemctl restart hostapd dnsmasq nodogsplash flask_rpi.service
```

5. Verification and debugging

To check the status of a service:

```
 systemctl status hostapd
 systemctl status dnsmasq
 systemctl status nodogsplash
 systemctl status flask_rpi.service
```

To view the latest logs:

```
 journalctl -u hostapd -n 30 --no-pager
 journalctl -u dnsmasq -n 30 --no-pager
 journalctl -u nodogsplash -n 30 --no-pager
 journalctl -u flask_rpi.service -n 30 --no-pager
```

To track logs in real time:

```
 journalctl -u hostapd -f
 journalctl -u dnsmasq -f
 journalctl -u nodogsplash -f
 journalctl -u flask_rpi.service -f
```

14. Add a new style

The PhotoBooth app makes it easy to add new image transformation styles, without changing the internal logic of the interface. This extensibility is based on a dynamic architecture based on the constants declared in the constante.py file.

Adding a style can be as simple as a statement, or it can be enhanced with optional elements such as a custom texture, translation, or a dedicated workflow.

14.1. Mandatory elements

In order for the app to recognize and display a new style, it is imperative to perform at least the following steps:

14.1.1. Add style to the dico_styles dictionary

In the constante.py file, locate the dictionary named dico_styles, and add a new entry with:

- The key: the name of the style (in lowercase, without special spaces),
- The value: a prompt in English, describing precisely the transformation to be applied to the image.

Example:

```
 dico_styles = {
     "comic": "Turn this person into a bold comic book style character...",
     "oil paint": "Render the subject as a classical Renaissance oil
painting...",
     "mythic": "Transform this person into a figure from Norse mythology with
runes and ancient armor..."
 }
```

The more detailed the prompt, the more faithful the generation will be to the desired style. It is strongly recommended to draw inspiration from existing ones.

14.1.2. Default behavior

Once this entry is added:

- A new style button automatically appears in the interface.
- The text of the button corresponds to the name of the key (e.g., "mythic").
- A default texture is applied to the button if no custom image is provided.
- The workflow used for the build will be the default.json file.

14.2. Optional Elements (Highly Recommended)

For better aesthetic, linguistic, and functional integration, you can add the following.

14.2.1. Custom button texture

To make the new brush button look clean, add a .png-format image to the textures folder, with the same name as the style key.

Location:

```
gui_template/btn_textures
```

Example:

For a "mythic" key → add mythic.png in the folder above.

14.2.2. Style translation

The interface is multilingual. To ensure that the style name is displayed correctly in each language, you must:

- Open the translation files in `PhotoBooth/language_file/`, e.g. `/norway.json` for Norwegian.
- Add an entry in the JSON dictionary with:

```
{
  "comic": "Tegneserie",
 "oil paint": "Olje-målet",
  "mythic": "Norrøn Mytologi"
}
```

The name of the key must be the same as the one used in dico_styles.

14.2.3. Custom workflow (advanced)

By default, all styles use the default.json file as the build pipeline.

However, it is possible to create a style-specific workflow to tailor processing to their needs.

Create a JSON file in the PhotoBooth/workflows/ folder with the same name as the style key.

```
Example: mythic.json
```

This file must imperatively respect the structure expected by the application, i.e.:

- An image type input (photo taken),

- A text input (prompt),
- A single image output.

If the workflow does not respect this interface, the application may not work properly with this style.

15. Multilingual management: Edit or add translations

The PhotoBooth app supports multiple languages via JSON files bundled in the language_file/ folder. These files contain all the text displayed in the interface, including titles, instructional messages, warnings, style names, etc.

15.1. Language File Structure

Each file is named according to language or region (e.g., norway.json, sami.json, uk.json) and follows a nested dictionary structure.

Example of a partial file (language_file/uk.json):

```
{
  "OverlayRules": {
    "title": "Do you wish to download your photo to your phone?",
    "message": "Any photo taken by the Photo Booth is automatically deleted
after 5 minutes."
  },
  "main_window": {
    "message": "Choose a filter, then press the button and pose!"
  },
  "style": {
    "comic": "Comic",
    "statue": "statue"
  }
}
```

15.2. Add or edit a translation

To customize a text displayed by the interface in a given language, all you have to do is modify the corresponding files.

The keys used in these files must be exactly the same as those used in the source code, including:

- "`OverlayRules`", "`OverlayQrcode`", "`OverlayLoading`", "`main_window`", "`WelcomeWidget`" for components,
- "`style`" for the names of the styles.

15.3. Add a Translated Style

When adding a new style to constante.py, it is highly recommended that you add its translation to each language file, in the "`style`" section.

Example to add in the "style" dictionary:

- In norway.json:
  - `"mythic": "Norrøn Mytologi"`
- In sami.json:
  - `"mythic": "Árbevirolaš Mythologiija"`
- In uk.json:
  - `"mythic": "mythical"`

If no translation is provided, the text displayed will simply be the raw name of the key (e.g. "mythic").

15.4. Line Breaks: Use \n

To force a line break in a multi-line message, use \n.

Correct example:

```
"message": "Open the camera\nThen scan the code\nSave the image"
```

15.5. Where to find the files

The language files are located in the directory:

```
PhotoBooth/language_file/
```

Each file corresponds to a language, for example:

`norway.json` : Norwegian

`sami.json` : Northern Sami

`uk.json` : English

TABLE OF FIGURES