|  |
| --- |
|  |
| Software Architecture and Design Patterns Assignment |
|  |

|  |
| --- |
| Konstantin Frolov November 2023 |

Contents

[1. Customized Scenario Selection or System Development 2](#_Toc152187370)

[1.1 Introduction 2](#_Toc152187371)

[1.2 Face recognition library 3](#_Toc152187372)

[1.2.1 What is Library 3](#_Toc152187373)

[1.2.2 Why so popular 3](#_Toc152187374)

[1.2.3 Benefits of using libraries 3](#_Toc152187375)

[1.2.4 Who makes libraries. 3](#_Toc152187376)

[1.2.5 Compare Library with Framework and Design Patterns 4](#_Toc152187377)

[1.2.3 The main features of face recognition library 4](#_Toc152187378)

[1.3. What needs to be changed. 6](#_Toc152187379)

[1.3.1. Separation of Concerns: 6](#_Toc152187380)

[1.3.2. Hardcoded Paths: 6](#_Toc152187381)

[1.3.3. Global Variables: 6](#_Toc152187382)

[1.3.4. Error Handling: 6](#_Toc152187383)

[1.3.5. Magic Numbers: 6](#_Toc152187384)

[1.3.6. Comments: 7](#_Toc152187385)

[1.3.7. Singleton Pattern for Webcam: 7](#_Toc152187386)

[1.3.8. Unit Testing: 7](#_Toc152187387)

[1.3.9. File Handling: 7](#_Toc152187388)

[2. Comprehensive Design Pattern Exploration 7](#_Toc152187389)

[2.1. Overview of Design Patterns 7](#_Toc152187390)

[2.1.1 Creational Design Pattern 7](#_Toc152187391)

[2.1.2. Behavioural Design Pattern 7](#_Toc152187392)

[2.1.3 Architectural Pattern 8](#_Toc152187393)

[2.2. The reasoning behind my selection. 8](#_Toc152187394)

[2.2.1. The use of Singleton 8](#_Toc152187395)

[2.2.2 The use of Template Method Pattern 8](#_Toc152187396)

[2.2.3. The use of Model-View-Controller (MVC) 9](#_Toc152187397)

[3. Integrated Design and Architecture Pattern Application 10](#_Toc152187398)

[3.1. UML diagram 10](#_Toc152187399)

[3.2. Code samples and Snippets 12](#_Toc152187400)

[3.2.1. The application starts at the main class. 12](#_Toc152187401)

[3.2.2. Main App Window 12](#_Toc152187402)

[3.2.3. The switching between frames 14](#_Toc152187403)

[3.2.4. Registration window 16](#_Toc152187404)

[3.2.5. Home Screen 17](#_Toc152187405)

[3.2.6. Using Singleton 18](#_Toc152187406)

[3.2.7. Using of Model View Controller (MVC) 19](#_Toc152187407)

[4. Presentation/Demonstration 22](#_Toc152187408)

[4.1. Screen Record 22](#_Toc152187409)

[4.2. Video Source 22](#_Toc152187410)

[5. Testing, Validation, and Code Quality 22](#_Toc152187411)

[5.1 Database Test 23](#_Toc152187412)

[5.2. File Manager Test 25](#_Toc152187413)

[5.2. Qodana 27](#_Toc152187414)

[6. Conclusions 28](#_Toc152187415)

[References 29](#_Toc152187416)

# Customized Scenario Selection or System Development

## 1.1 Introduction

While exploring YouTube, I came across a channel by a developer named Felipe, in one of his videos he gave an example of how you can use a facial recognition library, and for this, he wrote an attendance GUI application in Python.

(https://github.com/computervisioneng/face-attendance-system, 2023)

The user interface is simple and consists of two separate windows: the main window and the registration window, designed for different use cases - user registration and login.

After launching the application, the main window appears with two buttons “Login” and “Register”, as well as a live webcam feed. Activating the "Register" button causes a new window to appear on top of the Main View. This registration window includes two labels, two buttons, and an input field. The first label displays the latest webcam image, while the second prompts the user to enter their username. Below these labels is an input field that allows the user to enter a username, followed by two buttons - one to register and the other to return to the main view, giving you the opportunity to start over.

The registration process involves creating a binary data file or pickle file based on the user's selfie. The file accepts the name entered in the input field, ensuring uniqueness. However, if the user is already registered, a message box notifies him of the existing registration status. Returning to the main view, clicking the Login button brings up a personalized welcome message for registered users; Unregistered users are prompted in a message box to register before logging in.

Fig. Figure 1 shows a visual representation of this user flow.

A person looking at a computer screen

Description automatically generated

Figure 1 Screenshot from a YouTube

## 1.2 Face recognition library

### 1.2.1 What is Library

A software library or so-called code library is a collection of pre-written code or modules that perform specific tasks or provide functionality. In other words, the library is a written code by another programmer who saved your time, and to do a specific task you don't need to write the code yourself, it is already there, all you do is import this library and use that specific library that already has that task covered. As I mentioned it really saves you time and effort, allowing you to benefit from the work of others in your own projects.

### 1.2.2 Why so popular

Libraries are popular in Python for several reasons. Python has a rich ecosystem of libraries covering a wide range of areas, from scientific computing (NumPy, SciPy) to web development (Django, Flask), machine learning (TensorFlow, PyTorch), and even more. The Python Package Index (PyPI) makes it easy for developers to share and distribute their libraries. The simplicity and readability of Python code also contribute to the popularity of using language libraries. Also, the libraries are thoroughly tested and used by a large community, which increases their reliability.

### 1.2.3 Benefits of using libraries

As I mentioned above it saves time and effort, for example: Libraries offer pre-implemented functions and classes, saving developers time and effort by providing ready-made solutions to common problems. Here is another example in this project I was using a face recognition library, and if I need to work on another similar project, I possibly will need to rewrite the code again to use face recognition, but thanks to this library instead of writing it from the scratch all I need is just import it.

### 1.2.4 Who makes libraries.

Libraries can be created by individual developers, open-source communities, or organizations. Many libraries are the result of the collaborative efforts of authors from all over the world. Some libraries are developed and maintained by companies to support their products or services.

### 1.2.5 Compare Library with Framework and Design Patterns

#### 1.2.5.1 Library

Using simple words, a library is a collection of different pieces of code that perform specific tasks or add certain functions to their projects. Libraries are thoroughly tested which increases their reliability. It can be generated not only by organizations or open-source communities but also by developers like you and me.

#### 1.2.5.2 Design Patterns

I think this is the best example that can explain this in simple words, imagine you're building a house, and you face some common design challenges, for example where to put the doors and windows. Design patterns are like blueprints for solving these common design problems in software. They provide a guide or template for developers to follow when they encounter similar issues, promoting smart and proven ways of building software. In other words, why create a new house plan if there are already ready solutions for you, I am pretty sure you will find maybe even a better house plan than you had from the beginning.

#### 1.2.5.3 Frameworks

Now, picture a construction framework for building entire houses. A framework in software is a set of tools, rules, and guidelines that help developers build entire applications. It not only provides ready-made tools (libraries) but also defines how everything should be structured. It's like having a plan or skeleton for the entire building. Developers use this framework to create customized applications more efficiently.

### 1.2.3 The main features of face recognition library

Felipe used a facial recognition library developed by another contributor, Adam, available at [GitHub](https://github.com) (ageitgey, 2018). With 99.38% accuracy in the Labelled Faces in the Wild test, this library allows for both face recognition and manipulation.

The main features of the library include:

Identification of all faces in a picture (see Figure 2).

A person standing next to a flag

Description automatically generated

Figure 2 Example of an image with more than one face

Find and outline facial features such as eyes, nose, mouth, and chin (see Figure 3) to manipulate them according to the requirements of a specific task.

A collage of a person in a suit

Description automatically generated

Figure 3 Example of detection and outline facial features.

## 1.3. What needs to be changed.

## 1.3.1. Separation of Concerns:

The code currently combines the GUI setup (tkinter), webcam handling (cv2), face recognition (face\_recognition), and business logic all within the App class. It would be beneficial to separate these concerns into different classes or modules. In other words Apply any variation of Model View Controller (MVC) Design Patterns, and make classes bit shorter.

## 1.3.2. Hardcoded Paths:

The code contains hardcoded paths for model directories ('/home/phillip/Desktop/todays\_tutorial/27\_face\_recognition\_spoofing/code/face-attendance-system/Silent-Face-Anti-Spoofing/resources/anti\_spoof\_models'). It's better to make these paths configurable or, even better, pass them as parameters.

## 1.3.3. Global Variables:

Some variables, such as self.cap and self.db\_dir, are defined as attributes of the class. It might be more appropriate to encapsulate them within methods where they are used, reducing their scope.

## 1.3.4. Error Handling:

The code lacks proper error handling. For example, if there's an issue with the webcam or the face recognition model, the application may crash. Implementing proper error handling can make the application more robust.

## 1.3.5. Magic Numbers:

There are magic numbers (e.g., 20 in self.\_label.after(20, self.process\_webcam)) scattered throughout the code. Consider using named constants or class attributes for such values to improve code readability.

## 1.3.6. Comments:

While the code includes some comments, there could be more to explain complex logic, especially in the login and logout methods.

## 1.3.7. Singleton Pattern for Webcam:

The code creates a new webcam (cv2.VideoCapture) instance each time add\_webcam is called. It might be more efficient to use a singleton pattern to ensure only one instance of the webcam is created.

## 1.3.8. Unit Testing:

in the whole code, I don't see any tests, in software development exists a simple rule: First you write the test, and only then do you start coding.

## 1.3.9. File Handling:

The file handling code (with open(self.log\_path, 'a') as f) could benefit from a more modular approach, perhaps encapsulating the logging functionality in its own class.

# Comprehensive Design Pattern Exploration

## Overview of Design Patterns

### 2.1.1 Creational Design Pattern

#### Singleton

Singleton is a creational design pattern because it deals with the process of object creation. Specifically, it ensures that a class has only one instance and provides a global point of access to that instance. (SourceMaking.com, Singleton Design Pattern, 2023)

### 2.1.2. Behavioural Design Pattern

#### Template Method Pattern

The template method is a method in a superclass, usually an abstract superclass, and defines the skeleton of an operation in terms of several high-level steps. These steps are themselves implemented by additional helper methods in the same class as the template method. The helper methods may be either abstract methods, in which case subclasses are required to provide concrete implementations, or hook methods, which have empty bodies in the superclass. Subclasses can (but are not required to) customize the operation by overriding the hook methods. The intent of the template method is to define the overall structure of the operation while allowing subclasses to refine, or redefine, certain steps. (SourceMaking.com, Template Method Design Pattern, 2023)

### 2.1.3 Architectural Pattern

#### Model-View-Controller (MVC)

MVC is an architectural pattern that deals with the overall structure of a software application. It separates an application into three interconnected components: Model (handles data and business logic), View (displays the user interface), and Controller (manages user input and updates the Model and View accordingly). The MVC pattern provides a way to organize and manage the flow of data and user interactions in a systematic manner. (Point, 2023)

## The reasoning behind my selection.

### The use of Singleton

Using Singleton in the Database Manager class ensures that only one instance of the Database class is created, this means that there is only one connection to the database.

In my database, there's only one table. However, in the future, additional tables might be introduced. If updates to the database are made from various classes, it could become challenging to keep track of changes across different tables. Thanks to the Singleton pattern, I can be sure that, if any changes are made (CRUD processes), I will be working with the most recent data in my database.

### 2.2.2 The use of Template Method Pattern

The main class is Main App, which is a subclass of `tk.Tk` (Tkinter's main application window class). The Main App has two subclasses: Main Frame and Registration Frame. This hierarchy represents the template method pattern, where Main App provides the overall structure, and subclasses (Main Frame and Registration Frame) customize certain steps. Now the Main App class has a method called `show\_frame`. This method is like the template method. It takes a parameter (cont.), which represents the frame to be shown. The template method is responsible for which frames will be shown.

Main Frame and Registration Frame extend the functionality of the template method. They customize the behaviour of the application for different scenarios.

In Main Frame, the methods login, logout, and `show\_registration\_frame` are examples of hook methods. These methods have specific implementations in the superclass but can be overridden in the subclasses to customize behaviour.

This way the code organizes the application using the template method pattern, where the main class (Main App) defines the overall structure, and subclasses customize specific steps to achieve different functionalities (login, registration, etc.). The use of external classes and libraries enhances modularity and functionality.

### The use of Model-View-Controller (MVC)

#### Original Version:

* The application logic, UI elements, and database operations are all in the same class (`App`).
* No clear separation between the responsibilities of managing the UI, database, and user-related operations.
* Operations such as creating a new user and recognizing a user same as creating a new user, recognizing a user are scattered throughout the `App` class.
* Limited modularity makes it challenging to reuse code for database or user operations.
* Limited flexibility for adding new features or modifying existing ones.
* Limited error handling: exceptions are not explicitly handled.

#### Re-designed Version:

* The responsibilities are more clearly separated into different classes (`UserManager`, `DatabaseManager`, `User`).
* `UserManager` is responsible for user-related operations. It clearly defines methods like `login`, `logout`, and `recognize\_user`
* Database operations are encapsulated within one class. Methods like `get\_usr\_encod\_as\_arr`, `insert\_user`, `find\_user`, etc., are defined in `DatabaseManager`.
* Improved modularity, allowing for easy reuse of the `DatabaseManager` and `UserManager` classes in other parts of the application or different projects if needed. Additional features can be easily integrated into any class without affecting the entire application.
* Class encapsulating - each class has a specific responsibility, making it easier to understand and modify individual components without affecting the entire application. Such as the `User` class which represents the user entity.
* Improved error handling, class `LogManager` records every error into the log file which makes it easy to track what most common problems occur when the user is trying to register or just log in.

In summary, the newly redesigned version promotes modularity, separation of concerns, and code reusability, making it easier to maintain and extend in the future.

# Integrated Design and Architecture Pattern Application

## 3.1. UML diagram

On Figure 4 illustrated UML Diagram for entire application (Frolov, Design Pattern UML, 2023)

A diagram of a company

Description automatically generated with medium confidence

Figure 4 UML Diagram

## Code samples and Snippets

### 3.2.1. The application starts at the main class.

A screenshot of a computer program

Description automatically generated

Figure 5 Starting File

In the main class, I created an obj of the class App, then I calling method start(), this method creates an instance of the ‘MainApp’ class which is responsible for the main window of the GUI. For this I need to import MainApp from View directory, where I have FileManager.py file.

### 3.2.2. Main App Window

A screen shot of a computer program

Description automatically generated

Figure 6 Main App

From line 10 till line 15 I am importing classes which I will be using later in this File

The \_\_init\_\_ method initializes the main application. Then it calls the constructor of the parent class (tk. Tk) to set up the main window. Then I create a container that will contain different frames of the application. In my case Main Frame and Registration Frame. On line 28 I create an empty Dictionary (self. frames = { }) to store instances of the frames.

Using For loop I create instances of each frame.

It adds each frame instance to the self. frames dictionary for later access. In line 33 I configured it to be the same size as the container. In line 35 I display the main frame as the first frame to be displayed. Method show\_frame is responsible for displaying the requested frame.

Line 40 – 42 it checks if the frame is Registration frame, and if so then image will be updated to the last image from the video capture on the Main Frame, I actually had a huge problems with passing image from one frame to another, I found solution on Stack overflow website where one person provided me with the link to similar situation, and after a conversation we came with this idea.

### The switching between frames

#### Main Frame

A computer screen shot of a computer code

Description automatically generated

Figure 7 Switching to Registration Frame

This is a part of code from the Main Frame Class.

Line 66 – I create a button, and if user presses this button, it triggers show\_registration\_frame method.

Line 73 - creates a snapshot from webcam and stores it in snap variable.

Line 76 – Using controller, I specify which frame to display, and as second parameter I am passing a variable which holds a snapshot from Line 73

#### 

#### Registration Frame

A computer screen shot of a program code

Description automatically generated

Figure 8 Switching to Main Frame

Line 114 – I created a button ‘try again’ which uses a lambda function, in which using controller I specify what frame to display.

### Registration window

A screen shot of a computer program

Description automatically generated

Figure 9 Registration Frame Constructor

Line 107 – FileManager - Manages file-related operations, used to get the path for error logs.

Line 108 – LogManager - Manages logging, specifically error logs in this case.

Line 111 – UIManager - Manages the UI components, used to create labels, buttons, etc.

A computer screen shot of text

Description automatically generated

Figure 10 Registration Method

In same class I have registration methos which is responsible for new users’ registration.

And in this method, I created an instance of class User Manager.py

Line 142 – UserManager - Manages user-related operations, including registering new users.

Line 156 – Using obj. of User Manager class, I’m passing all users’ inputs which will be used in registration process.

### Home Screen

A computer screen shot of a program

Description automatically generated

Figure 11 Main Frame Constructor

Line 50 - controller - Responsible for managing frame transitions.

Line 51 - cam - handling webcam-related functionalities.

Line 52 - usr\_mngr - Responsible for managing user-related operations.

Line 53 - main\_window - Responsible for managing GUI components.

Line 54 - db\_mngr: An instance of the DatabaseManager class, for handling interactions with a database.

Line 59 - Webcam Initialization:

self.cam.start\_webcam ( self.camera\_label, 0 ) - Initiates the webcam using the start\_webcam method of the WebcamManager. It specifies the label (camera\_label) as the target for displaying the webcam feed and uses the camera index, in my case, it is 0.

### Using Singleton

A computer screen shot of a program

Description automatically generated

Figure 12 Webcam Manager Class

By having a single instance, you avoid unnecessary duplication of resources and potential conflicts. A singleton ensures that there is only one point of access to the Webcam Manager throughout the application. The \_\_new\_\_ and \_\_init\_\_ methods control the initialization process.

Line 11 - \_\_new\_\_ method ensures that only one instance is created,

Line 18 - \_\_init\_\_ method initializes the instance variables only if the instance is not already initialized. By having a single instance, you avoid unnecessary memory consumption that would occur if multiple instances of Webcam Manager were created.

A computer screen shot of a program

Description automatically generated

Figure 13 Database Manager Class

In the Database Manager, a singleton ensures that there is only one connection instance, preventing unnecessary duplication and potential resource conflicts. same as in the Webcam Manager class.

Having a single instance promotes consistency in accessing and managing the database throughout the application. When I initialize the object of the class Database Manager in the constructor I create a connection with database, If I mayn’t use Singleton then every time, I create an instance of this class there also a new connection with a database will also be created. In other words, a singleton pattern allows maintaining a global state for the database connection. The \_\_new\_\_ and \_\_init\_\_ methods control the initialization process.

Line 9 - The \_\_new\_\_ method ensures only one instance is created,

Line 16 - the \_\_init\_\_ method initializes the instance variables only if the instance is not already initialized. This helps manage the initialization process effectively.

### Using of Model View Controller (MVC)

#### Old Version

A screenshot of a computer program

Description automatically generated

Figure 14 Old System Structure

As you can see the whole application is bee written and stored on 2 Python files,

#### A New Version

A screenshot of a computer program

Description automatically generated

Figure 15 A New System Structure

The Old system combines user interface (UI) elements, webcam processing, facial recognition, and business logic into a single class (App). I used MVC, which allowed me to separate these problems into different components, making the code more modular and easier to maintain. After separating the model (data and business logic), view (user interface), and controller (processing user input and updating the model), I made the components of the code more reusable. And now if something needs to be changed, I do not need to go through the file and search for specific lines of code, it does not ensure me if I make a change in one line of code the whole app won't crash. And don't forget about testing: I think this would be a nightmare. Besides this, if in the future this application is modified and new features need to be added, then the clear separation of concerns provided by MVC can make it easier to scale and add new features without breaking existing functionality.

# Presentation/Demonstration

## 4.1. Screen Record

[A screenshot of a computer

Description automatically generated](file:///C:\Users\tamta\OneDrive\Documents\MTU\Screencast%20from%2026-11-23%2020_41_41.webm)

Figure 16 Video Presentation

## Video Source

<https://drive.google.com/file/d/1R6tIk0bEAxllPnyW4zqhn2TTi2v7h6LF/view?usp=drive_link>

(Frolov, DesignPatterns\_Student\_Signin\_App, 2023)

# Testing, Validation, and Code Quality

In contemporary programming languages, it's common to find built-in support for xUnit-style test automation libraries. Additionally, numerous languages, such as Python and Java, offer a variety of xUnit-style frameworks. For example, Java extends its support beyond JUnit to include TestNG. Similarly, Python boasts multiple choices, including nose, pytest, and Nose2, alongside the built-in unittest framework. (Pajankar, 23 Feb 2017). And using this unittest I made several tests to make sure everything is working as a clocks.

## 5.1 Database Test

A screen shot of a computer program

Description automatically generated

## File Manager Test

A screenshot of a computer program

Description automatically generated

Figure 17 File mngr. test

A screenshot of a computer program

Description automatically generated

Figure 18 Web mngr. testing

## 5.2. Qodana

A screenshot of a computer

Description automatically generated

Figure 19 Qodana Result

# Conclusions

Design patterns - Singleton, MVC, and Template Method - make software smarter and easier to use. Singleton ensures that we have one simple way to work with certain types of objects, this ensures that no other objects of that type are created, especially in a situation where we need to create a connection between the database and the application, MVC helps keep our code organized and easy to understand. with its help, the same code can be used to create other applications, and it also makes it easier to add any new feature without disturbing the entire eco-system of the application. As for the Template Method, it allows us to change the appearance of the application in a dynamic way, with each view in my in this case, the framework can be endowed with its own features. In the future when I will be creating (working)an application from scratch, and when I'll be in the design phase and will definitely use those patterns, thanks to this assignment I clearly understand how this work, and in which situation I should apply them.

# References

ageitgey. (2018, Apr 2). *face\_recognition*. Retrieved from GitHub: https://github.com/ageitgey/face\_recognition

Frolov, K. (2023, 11 26). *Design Pattern UML.* Retrieved from Google Drive: https://drive.google.com/file/d/1BUPRhKGAZdsbriKAFfLyq3PqhLkkR7uK/view?usp=sharing

Frolov, K. (2023, 12 03). *DesignPatterns\_Student\_Signin\_App*. Retrieved from GitHub: https://github.com/ConstantineFrol/DesignPatterns\_Student\_Signin\_App/tree/working

Frolov, K. (Director). (2023). *Presentation* [Motion Picture].

https://github.com/computervisioneng/face-attendance-system. (2023, August 12). *face-attendance-system*. Retrieved from GitHub: https://github.com/computervisioneng/face-attendance-system

Pajankar, A. (23 Feb 2017). *Python Unit Test Automation.* Apress.

Point, T. (2023). *Model View Controller Pattern*. Retrieved from https://www.tutorialspoint.com/: https://www.tutorialspoint.com/python\_design\_patterns/python\_design\_patterns\_model\_view\_controller.htm

SourceMaking.com. (2023). *Singleton Design Pattern*. Retrieved from https://sourcemaking.com/: https://sourcemaking.com/design\_patterns/singleton

SourceMaking.com. (2023). *Template Method Design Pattern*. Retrieved from https://sourcemaking.com/.