**Technical Solution**

Contents

[1 Introduction 2](#_Toc127885736)

[1.1 About this document 2](#_Toc127885737)

[1.1.1 Purpose & Scope of the docment 2](#_Toc127885738)

[2 Component Design 2](#_Toc127885739)

[2.1 Component Design Diagram 2](#_Toc127885740)

[2.1.1 Overall Workflow 2](#_Toc127885741)

[2.1.2 Low level Design 2](#_Toc127885742)

[3 Technology & Frameworks to be used 3](#_Toc127885743)

[4 Solution Approach 3](#_Toc127885744)

# Introduction

## About this document

### Purpose & Scope of the document

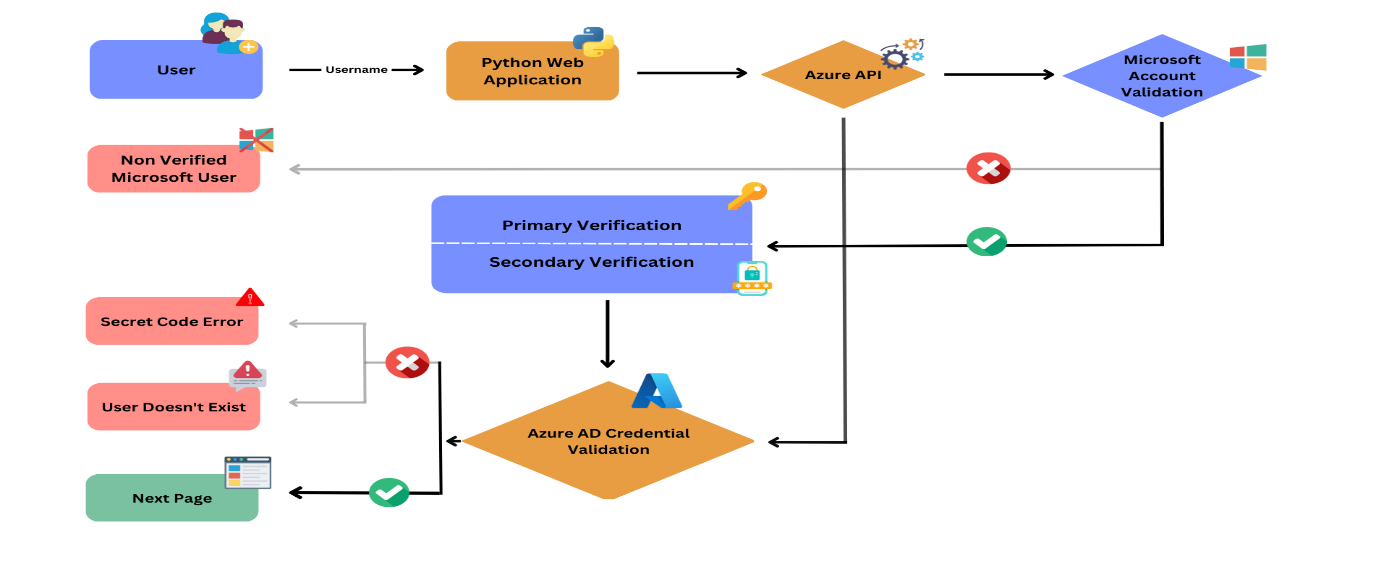
### The purpose of our problem statement, "Azure AD Authentication using Python", is that Azure AD can be used to enforce multi-factor authentication for user logins, adding an extra layer of security beyond passwords. This can prevent unauthorized access to sensitive data and applications, and using the Azure AD user, you can use any kind of service by using the same username and password. We used two-step authentication here, with the primary being a password and the second being either a mobile authentication app or Windows Hello. Passwords are easy to hack where as if the user has these kinds of Verification methodologies that are used to secure the Information.

# Component Design

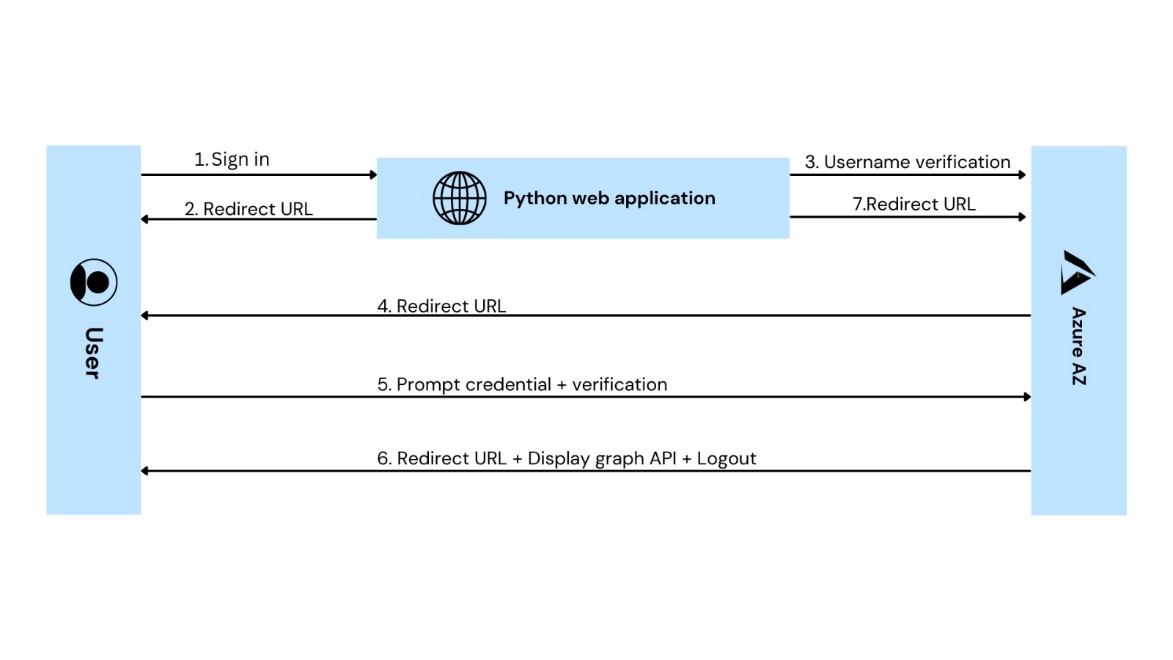
## Component Design Diagram

We designed the component with the verification as password as primary source, Mobile app authentication or Windows Hello as a secondary source. Where Azure AD supports almost seven different authentication methods, But our component will only use these from the authentication because in case of password less verification, user need an enabled or external sensor used to scan the Bio metrics. Most of the passwords are easy to hack for the hackers to secure the content and the information of the user, we go with password less verification. In password less verification we have biometrics, an authenticator app or Hello world are the methods. This component uses Windows hello and the Mobile app authentication for the Two-Step verification to avoid external devices and to make sure that all the users are comfortable with the verification steps.

**Overall Workflow:**



Here is the overall flow diagram for Azure AD Authentication, starts from user Point up to the verification of username. This diagram will give the clarity about the working flow and error occurrence like Non verified user, Secret code error, User doesn’t Exist. These are the three possible error Occurrence in this component.

**Sequence diagram:**

The above given sequence diagram will gives the insights about how the sub components are working and the interaction between the sub components.

### Low level Design

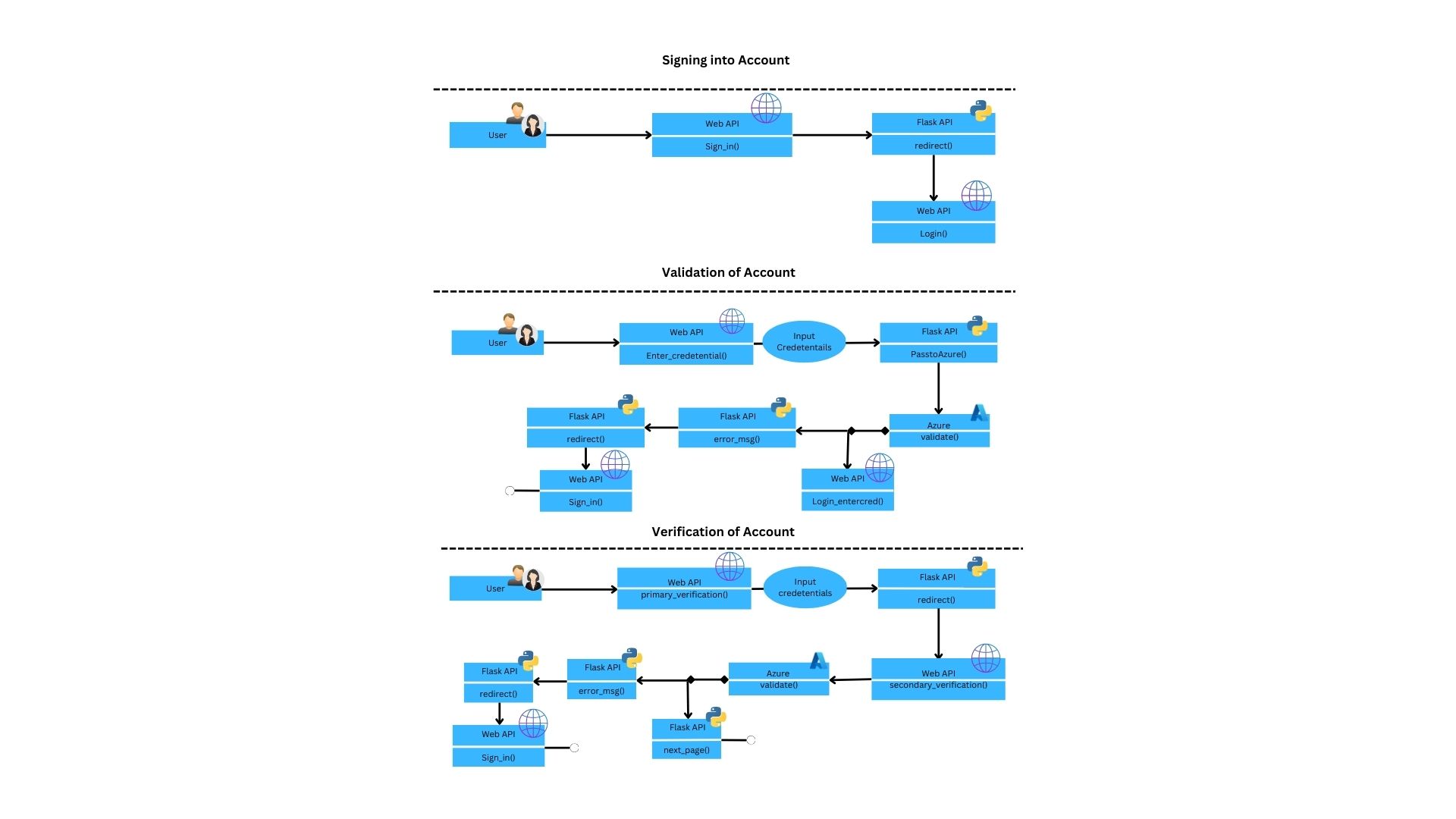
Initially, design of the software starts with the Python web application. When the user enters their username, Web application will accept it and send it to the Azure API Integration, which needs to make sure that the given username is verifies by Microsoft. It only accepts the Microsoft Mail ID. If the user has their own Microsoft Mail, then the component will move to the next stage, which is credential checking. We have a client secret key that will connect our Azure AD with the Python web application. If the given username matches our database, flow will move to the next page.

It consists of Three flows,

**The Initial Flow** will be for validation of the Microsoft Username, **Second Flow** will check for the credential and the client’s secret key. If both are correct then the **Final Step** will be navigated to the next page.

Whereas in the case of not matching username, at that time component will give the error message,

1. If the client secret key is correct but the username is not valid, then the error message will be a sign of trouble.
2. There may be a possibility of providing a not-valid key, and that time component will raise an error message called a Client Key Error.



# Technology & Frameworks to be used

This component designed using the technologies like

1. Flask 2.2.3
2. Azure AD
3. Authenticator Mobile APP

# Solution Approach

The solution for the given problem statement, based on its functionalities and responsibilities fragmented into the following sub-systems: UI, Python web application, Azure.

1. The UI part of the system is made to interact with the user, guides the user in entering credentials and displaying messages to correct the entries.
2. In this process the software pattern used is Singleton Pattern. In this pattern, a single instance of an authentication class is created and shared across the API to handle all authentication requests. This ensures that only one instance of the authentication class exists, and makes it easier to manage authentication logic across the entire application. All the requests are verified using azure active directory under one class that is designed for authentication.
3. As mentioned earlier the UI part of the system is responsible for interactions with the user. The UI passes the information or credentials provided by the user to the flask framework for validation and authentication. The diagnostic message in case of errors is passed to the user through the UI part.
4. The credentials passed to the application are username, password or pin or code. The username is passed in email ID format. Any other format will display an error message. The passwords is in a “free-form” format. The pin and code are strictly a series of numbers. Any characters or alphabets is not allowed.
5. The information given to the UI is passed on to the Flask framework working on the backend. The UI calls the Flask to validate the information. In some cases, when the information is of unsupported format the flask framework throws the error message without even passing it to the Azure AD to check.
6. In case of information in a compatible format, the framework passes the information to Azure AD. Suppose the username is not a Microsoft username or the user is not listed in the directory with permission to access the page or site, a diagnostic message is thrown. The same applies to the scenario where the client information given to the API is incorrect. When all the information entered by the user is correct, Azure approves the flask to the user to the subsequent page or site.
7. During Testing we create a web app service in Azure with Flask framework facilitating the UI with a specific client ID and client secret. The client secret and ID are entered in the app configuration module of the API. Bear in mind that while the credentials are validated and verified, so are the client ID and secret. Assume the client information is incorrect, Azure passes a diagnostic message to the framework which is flaunted in the UI.