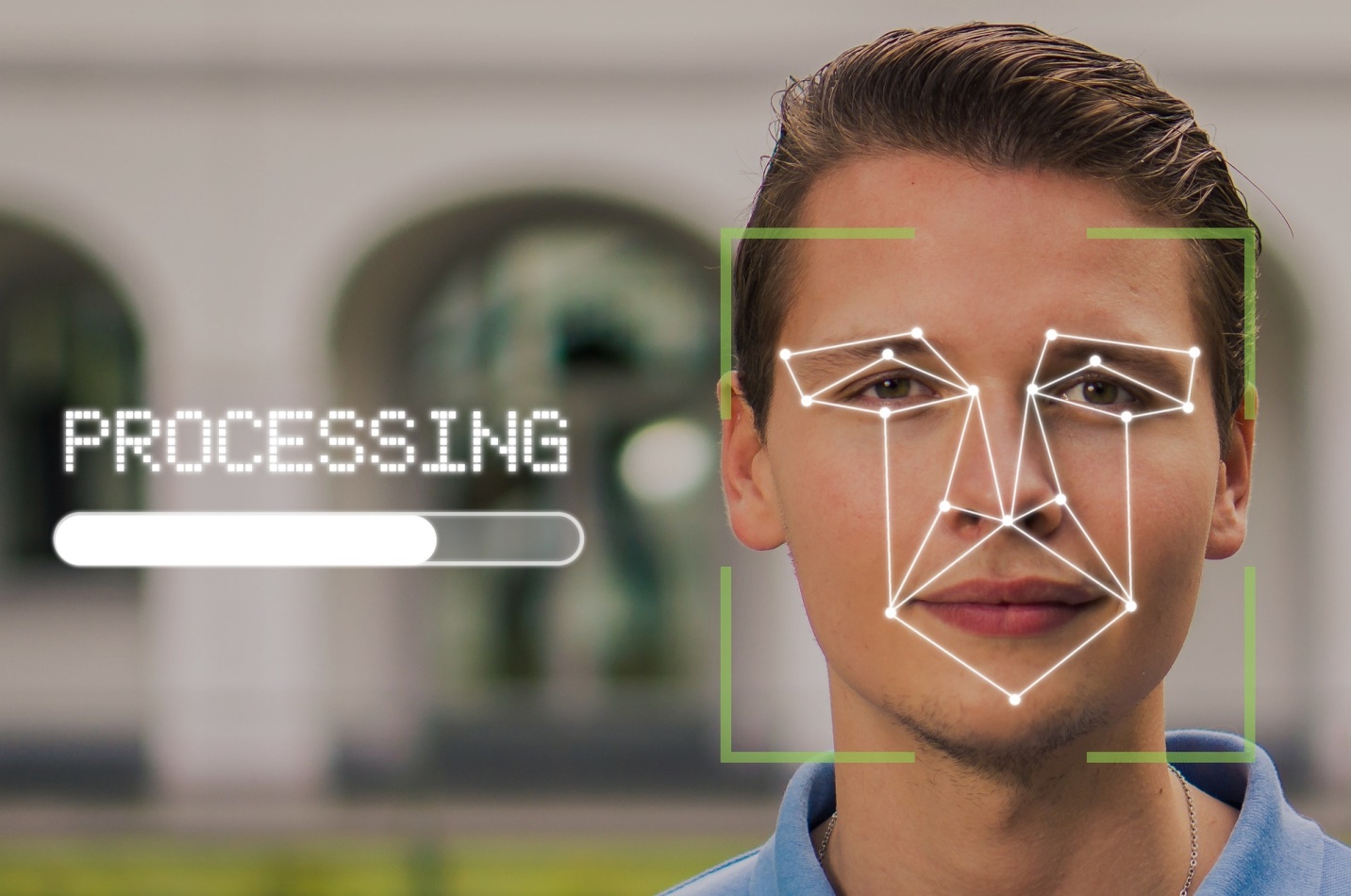
**Ultimate Guide To FACEIO**



Facial recognition technology called FACEIO is employed for online user identification. For increased security during online transactions and access to sensitive information, it can be linked into websites and apps to verify a person's identity using their distinctive face features. It can be used in addition to more established authentication techniques, such passwords, for increased security. It's crucial to consider any potential drawbacks, such as bias, errors, and privacy issues.

Key Features of FACEIO

FACEIO may include the following features that could make it a viable tool for online authentication:

* Convenience: Users of FACEIO can authenticate themselves more quickly and efficiently than by entering a password or providing the answers to security questions by simply taking a picture of their face.
* Increased security: FACEIO can add an extra degree of protection for online transactions and access to sensitive information by employing facial recognition technology.
* Multi-factor authentication: The authentication procedure can be made more secure by using it in conjunction with other authentication techniques like passwords and security questions.
* Liveness detection: The ability of the technology to tell whether the camera's subject is a real person and not a photograph or video will prevent fraud.
* Mobile compatibility: A broad spectrum of consumers can use the technology because it can be used on mobile devices with front-facing cameras.
* Remote verification: For firms that must confirm the identification of clients who are not physically present, the technology can be utilized for remote verification.

FACEIO is more open and user-friendly because it doesn't require any specialist hardware or software on the user's device. Regardless of the server-side language or front-end framework, it can also be adapted to other websites and web-based apps. It still has drawbacks, though, like potential prejudice, errors, and privacy concerns, which should be taken into account before implementation.

Integrating FACEIO In Your Web Application Or Website

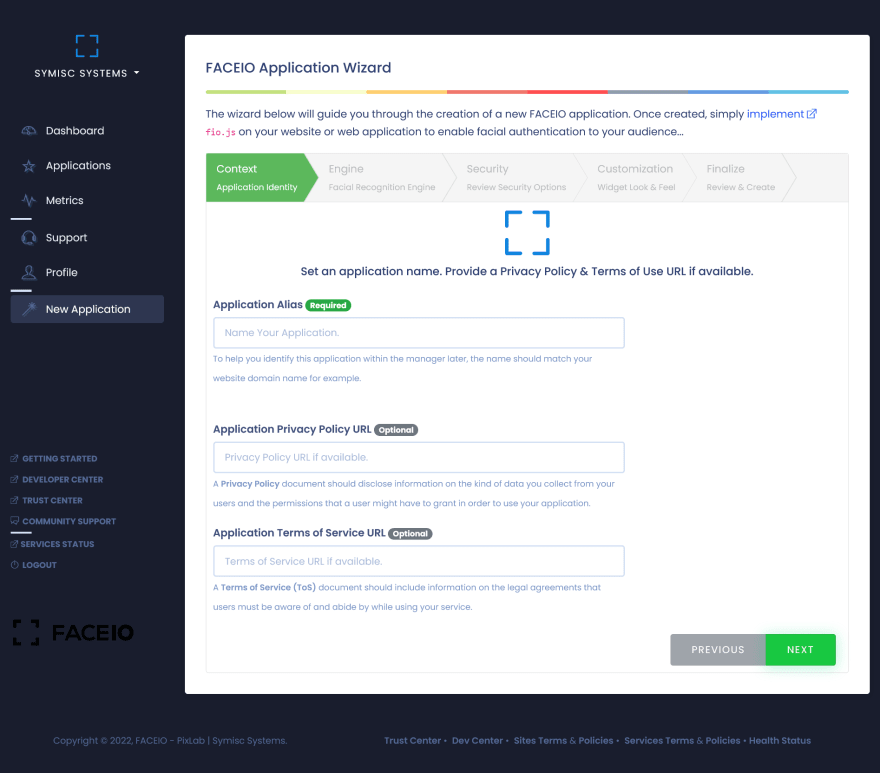
To add FACEIO to your web-app or website, you will need to integrate the technology into your existing codebase. In most cases, this entails adding a few lines of JavaScript code to your front-end code and creating an API in your server-side code.

The steps for including FACEIO in your website or web application are described in the following broad overview:

Step 1

[Sign up for a FACEIO account](https://console.faceio.net/login). You will need to create an account with a FACEIO provider to access and use the technology on your web-based application or website.

Use your email address to log in to the FACEIO Console initially. Following a series of automated steps, the Application Wizard produces your first application after connecting to your website or web application. The following steps usually involve entering the program's name, choosing an engine for facial recognition, selecting a cloud storage area, looking over security settings, and adjusting the Widget layout.



Integrate the fio.js faceIO Javascript library on your website after creating your application and initialize it with your app's Public ID. By doing this, your website can employ facial recognition technology to verify users. Consider the privacy and security issues, and test the technology frequently to ensure it is functioning as intended.

Integrate the FACEIO JavaScript library into your front-end code. This typically involves adding a few lines of code to your HTML and JavaScript files. The JavaScript library allows you to access facial recognition technology and capture images of the User's face.

Step 2

Put the following code in the HTML page(s) you intend to use to launch fio.js before the closing /body> element.



This code first constructs the "faceio-modal" div element and then adds the "fio.js" FACEIO Javascript library from the FACEIO CDN (Content Delivery Network). The facial recognition widget or modal will be displayed on the page in the div element with the id "faceio-modal."

It's crucial to start the fio.js library with the app's public ID as well as any additional settings you wish to set, such as the language and other customization choices and the modal's design.

Step3:

To initialize fio.js, you must create a new faceIO() object and pass in your app's Public ID as an argument. This allows the fio.js library to connect to your specific application and use its facial recognition capabilities. The code should look something like this:



It's also important to remember that you may further personalize the facial recognition experience on your website by passing more options to the faceIO() objects, such as the modal's layout, language, and other customization choices.

The Public ID can be found in your application's dashboard in the settings or general section. This particular application-specific identification is necessary for the fio.js library to connect to your application and access its face recognition features.

Step 4

The enroll() and authenticate() methods are used to invoke the facial recognition widget and start the enrollment or authentication process, respectively. enroll() is used to register new users, capture their facial features, and save them in the cloud storage associated with your application. authenticate() is used to verify a user's identity by comparing their facial features with the ones previously enrolled in the system.

To invoke the widget, you can call these methods on the instantiated faceIO object like so:



Understanding Of Enroll() And Authenticate()

Enroll

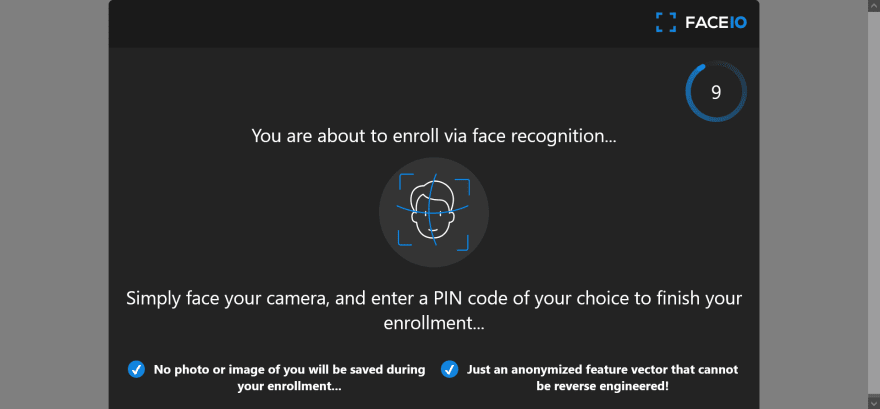
To enroll a new user in your application using the faceIO object, you can call the enroll() method. This method will bring about the FACEIO Widget and ask for the User's consent to access the webcam, extract and index the User's facial features for future authentication. It then takes in optional parameters such as locale for the interaction language, permissionTimeout for the time to wait for camera access permission, and a payload of arbitrary data such as an email address, name, or ID.

The enroll() method returns a promise that, when fulfilled, returns a userInfo object with a Facial ID unique to the enrolled User. If the User denies camera access permission or rejects the terms of use, the promise will be rejected with an error code. Additionally, suppose you have set up a [webhook handler](https://faceio.net/webhooks). In that case, the enroll() method will make a POST request to the configured URL after each successful enrollment, allowing for real-time updates and synchronization with your application backend.



In this example, we first instantiate the faceIO object. Then we create a function handleEnrollment that will take care of the enrollment process. Inside this function, we call the enroll() method and pass in the payload data, which is an arbitrary object that can contain any information about the User. The enroll() method returns a promise that resolves with a userData object which contains the facial ID and the payload data that was passed in.

We log the userData object for debugging purposes and save the userData object to our backend. Finally, we notify the User that the enrollment was successful. We also handle any errors that may occur during the enrollment process. In this example, we call the handleEnrollment function when the User clicks the "Enroll" button



Authenticate

The authenticate() method recognizes and authenticates previously enrolled users on an application. It requests access to the browser's webcam and starts the facial recognition process, which only requires one frame and takes less than 100 milliseconds. Depending on the app's security configuration, the User may need to confirm their PIN code for the authenticate() method to succeed. The method also takes optional parameters for camera access permission timeout and interaction language.

The authenticate() method returns a promise with a userData object upon successful identification, which includes the payload data linked to the User during enrollment and a Facial ID unique identifier. The promise is rejected with an error code if camera access is denied.



In this example, calling the authenticate() method will bring about the FACEIO widget, which will request access to the User's webcam or frontal camera. Once the User grants access, the facial recognition process will begin. If the User is successfully identified, the promise returned by authenticate() will be resolved with a userData object containing information about the User. The security will be rejected with an error code if an error occurs.

You can also pass optional parameters to the authenticate() method, such as a permissionTimeout to specify the number of seconds to wait for the User to grant camera access or a locale to set the language for the interaction.

General Boiler Plate



Two buttons are present on the page: "Enroll New User" and "Authenticate User," which, when clicked, respectively, invoke the corresponding JavaScript functions "enrollNewUser()" and "authenticateUser()."

Using the "faceio.enroll()" method, the "enrollNewUser()" function initiates facial enrollment by sending in an options object that includes various parameters like "locale" and "payload." The "payload" option is used to link arbitrary data with the User, such as their email address or a unique user ID, while the "locale" field is used to establish the User's preferred language. The function logs the User's unique face ID, enrollment date, gender, and approximate age to the console and displays an alert message to the User if the enrollment procedure is successful. The function invokes the "handleError()" method, which is not specified in this code sample if an error occurs during the enrolling process.

Using the "faceio.authenticate()" method, the "authenticateUser()" function initiates facial authentication by sending in an options object that contains the "locale" parameter. The function logs a message to the console informing the User that they have been recognized, along with the User's facial ID and any related payload data, if the authentication process is successful. The function executes the "handleError()" function if an error occurs during the authentication procedure.

It's crucial to remember that the "handleError()" function is not declared in the code snippet; as a result, it must be added for the code to work correctly. The actual app public id generated by the service should also be used in place of faceIO("app-public-id").

Reasons Why fio.Js Require A Running HTTP Server To Load.

JavaScript is a client-side language, which means that it executes in the browser rather than on the server, making it one of the primary causes of this. The server requests the HTML, CSS, and JavaScript files required to render a web page whenever the browser loads a web page. The browser subsequently executes the JavaScript code, which can send more requests to the server. Fio.js must send queries to the FaceIO server to retrieve data or carry out operations using the [FaceIO REST API.](https://faceio.net/rest-api)

JavaScript code cannot, however, make requests to servers other than the one that provided the initial page because of the security measures built into the browser. The "same-origin policy" refers to this practice. The requests made by fio.js must be proxied through a server on the same origin as the web page to get around this restriction.

Fio.js employs the XMLHttpRequest (XHR) API, a browser API for sending HTTP queries, which is another reason it needs an active HTTP server. The XHR API must be loaded from a web server since it can only be used when JavaScript code is running within the context of a web page.

Maintaining Your Application

You may administer your FaceIO application using the Application Manager, a web-based interface. It offers a user-friendly interface for setting up and controlling your application's resources, including users, groups, and permissions.

When utilizing the Application Manager to manage your FaceIO application, you should keep the following in mind:

* **User Management:** In your application, you may create, edit, and delete users, add them to various groups and give them different permissions.
* **Group Management:** In your application, groups can be created, modified, and deleted, with users assigned and permissions set.
* **Permission management:** Users and groups can be given permission to access, write, and execute various resources in your application.
* **Resource management:** Resources in your application, such as database tables and files, can be created, edited, and deleted.
* **Monitoring analytics:** The Application Manager offers monitoring and analytics features like real-time performance metrics and logs to understand how your application is used.
* **Security:** The Application Manager offers robust security features like multi-factor authentication and encryption to safeguard your application and its data.
* **Integration:** using the Application Manager, you can connect your app to other services, such as social media sites and outside databases,
* **Scalability:** You may quickly scale your application using the Application Manager as necessary to handle rising user counts and additional traffic.