**التوثيق للمطورين المستقبلين**

* الملف يتألف من مجلدين رئيسيين ) لا تنسى فك ضغط الملفات(

1. المجلد Dist

يحتوي على مجلد الموقع الجاهز الذي يتم رفعه مباشرةً إلى الاستضافة.

1. المجلد التطويري (Th development)

يحتوي على مجلد التطوير للتعديلات المستقبلية.

كيفية تعديل الملفات التطويرية:

1. افتح الملف في أي بيئة تطويرية متكاملة (IDE).
2. قم بتشغيل npm run dev.
3. ابدأ التطوير.

**التقنيات المستخدمة في الموقع:**

1. HTML
2. CSS
3. JavaScript
4. Bootstrap 5.3
5. Vue 3 (CAPI)
6. Pinia
7. Vite
8. Oh Vue Icons للحصول على الرموز
9. Axios لعمل طلبات الويب
10. Vue Router
11. SWAL2

**تفاصيل الملفات**

1. **Index.html**

****

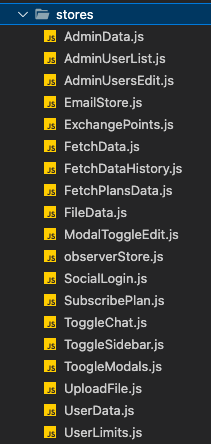
يحتوي هذا الملف على خطوط جوجل ورمز HTML الرئيسي

1. **.env**

****

Contain the back end api links that must be prefixed with (VITE) to work

1. **Stores folder**



Contain pinia stores for state management.

**Explain store job**

The store folder in this project contains Vuex-like store implementations using Pinia for state management.Here is a comprehensive and concise usage summary of the files inside the store

folders:

**observerStore.js:**

This store defines the useObserverStore store using Pinia.

It provides a function observeElement to watch an element's intersection with the viewport.

The observeElement function uses an IntersectionObserver to track the visibility of an element and update a reactive variable accordingly.

**FetchDataHistory.js:**

The useFetchHistory store is defined in this file.

It includes functions to fetch chat history data from the backend.

The getHistory function retrieves chat history data and processes it to update the store state.

It also handles mapping and processing of chat history data, including identifying MP3 files and other file types.

**AdminUserList.js:**

The useAdminUserList store is defined to manage user list-related data.

Functions like getUsersList, filterUsers, searchUsers, and paginateUsers are provided to fetch and filter user data from the backend.

Error handling using FireError function and managing loading state are also included in this store.

**AdminData.js:**

The useAdminData store manages admin-related data and file information.

Functions like getFileList, FilterFiles, editFileStatus, deleteFile, and getUsersList are implemented to handle file and user data.

It includes functions for updating file status, deleting files, and filtering files based on various criteria.

Error handling using FireError function and managing loading state are also part of this store.

admins.These stores encapsulate the state and provide functions to interact with and update the state based on backend data and user actions.

**FetchData.js:**

The useFetchData store is defined to manage data fetching and chat history.

It includes functions to handle user questions, responses, loading states, and sending requests to the backend.

Functions like pushQuestion, pushData, sendRequest, sendRequestVoice, sendRequestWithFile, deleteAllChat, and removeMessageFromCurrentData are implemented to manage chat interactions and data fetching.

It also handles sending requests with files, voice recordings, and managing chat history.

**AdminUsersEdit.js:**

The useEditUser store is defined to handle user editing functionality for admins.

It includes functions like EditUser to update user data, FireError for error handling, and managing loading state.

The store interacts with the backend to update user information and displays success or error messages using SweetAlert.

**UserData.js:**

This file likely contains a store for managing user data.

It may include functions to fetch and update user information, handle authentication, and store user-related data.

The store could manage user authentication status, user profile data, and interactions with the backend related to user data.

ToggleChat.js:

**ToggleChat**

store handle toggling chat-related functionalities.

It includes functions to toggle between chat history and current chat data, manage chat settings, and control chat-related UI elements.

**userData store**

it is file manages user data, including the user's name, email, ID, and points.

It contains getUserData(): A function to fetch user data from the backend.

Sends a request to the backend API endpoint specified in VITE\_USER\_DATA.

Uses VueCookies to get the JWT token for authentication.

Updates the state variables with the user's name, email, ID, and points from the response.

**userLimits store**

Its is file manages user limits and subscription information.Here is a summary of the functionality provided in the useUserLimits store:

it contains:

getUserLimits(): A function to fetch user limits from the backend.

Sends a request to the backend API endpoint specified in VITE\_GET\_USER\_LIMIT.

Updates the state variables with the limits for text messages, audio messages, and image media from the response.

deleteSubscription(): A function to handle deleting a user's subscription.

Displays a confirmation dialog using Swal to prompt the user to renew their subscription.

If the user confirms, it redirects to the prices page and reloads the page.

Sends a request to the backend to delete the user's subscription.

SocialLogin store

Contain functions to login with google , Facebook and twitter

**useUploadFile store**

its is file handles the functionality related to uploading files in the application.Here is a summary of the functionality provided in the useUploadFile store:

it contains:

submitData(): A function to submit file data for uploading.

Initiates the loading state.

Retrieves the JWT token from VueCookies for authentication.

Makes a POST request to the backend API endpoint specified in VITE\_FILE\_UPLOAD with the file data and metadata.

Shows a success message using Swal if the file upload is successful, allowing the user to return to the chat page.

Sets the errorUpload variable to true if an error occurs during the file upload.

Resets the loading state and error/success variables after a delay of 2 seconds.

**useToggleModals** store file

showExchangeModel: A reactive variable to control the visibility of the exchange model modal.

Functions:

toggleShowExchangeModal(): A function to toggle the visibility of the exchange model modal.

Inverts the value of showExchangeModel to show/hide the modal.

**useToggleSidebar store file**

it includes

ToggleSidebar(): A function to toggle the visibility of the sidebar of chat body

This store provides a way to toggle the visibility of the sidebar in the chat page.It allows users to show or hide the sidebar based on their interactions with the application.

**SubscribePlan store**

subscribe(plan, period): This function sends a POST request to subscribe to a plan with a specified renewal period. It uses axios to make the request, handles the success response by opening a new tab with a provided redirect URL, and shows an error message if there's an issue.

deleteSubscribe(): This function sends a POST request to delete the current subscription. It uses axios, and in case of an error, displays an error message.

getSubscribe(): This function retrieves the user's subscription details using a GET request. It checks if the user has an active subscription, sets a flag (active), and handles various scenarios like redirecting to a pricing page if no active subscription is found.

subscribeFree(): This function subscribes the user to a free plan. It sends a GET request and upon success, shows a success message and redirects the user to a chat page.

**ExchangePoints store**

exchangePoints(type): This function is used to exchange points for rewards of a specified type. It sends a POST request to a specified endpoint (import.meta.env.VITE\_EXCHANGE\_POINTS) with the reward type. Upon success, it displays a success message using Swal (SweetAlert2), and on error, it shows an error message. Finally, it refreshes user limits and user data using functions from the useUserLimits and useUserData stores respectively.

**FileData store**

Used to show file in chat page and contain file name variable

**EmailStore store**

sendOtp(email): This function is used to send an OTP (One-Time Password) to the provided email address. It stores the email in the session storage to be used later for verification. It makes a POST request to a specified endpoint (import.meta.env.VITE\_SEND\_OTP), and upon success, it redirects the user to the "verify-email" page. If there is an error, it displays an error message using Swal (SweetAlert2).

verfiyOtp(otpCode): This function is used to verify the OTP entered by the user. It retrieves the email from the session storage, checks if the email exists, and then makes a POST request to a specified endpoint (import.meta.env.VITE\_VERIFY\_OTP) with the email and OTP code. Upon successful verification, it redirects the user to the "login" page and displays a success message. If there is an error during verification, it displays an error message using Swal.

sendOtpAgain(): This function is used to resend the OTP to the previously stored email address. It retrieves the email from the session storage, checks if the email exists, and then makes a POST request to a specified endpoint (import.meta.env.VITE\_SEND\_OTP). Upon success, it displays a success message using Swal. If there is an error, it displays an error message.

**FetchPlansData store**

fetchPlanData(): This function fetches data for basic, premium, and enterprise plans. It makes a GET request to a specified endpoint (import.meta.env.VITE\_GET\_PLANS\_DATA) to retrieve plan information from the backend. Upon a successful response, it assigns the received data to the basicPlan, premiumPlan, and enterprisePlan variables. If there is an error during the request, it displays an error message using Swal (SweetAlert2).

1. **Views folder**

التي تحتوي على كافة صفحات الموقع الرئيسية

A screenshot of a computer

Description automatically generated

1. R**outer folder**

A screen shot of a computer

Description automatically generated

يحتوي المجلد `index.js` الذي يستخدم إصدار 4 من Vue Router (`router-4`) للتحكم في عناوين URL في صفحة الويب.

***الملاحظات في index.js***

***1. المسارات (Routes)***

تحديد المسارات لكل صفحة في التطبيق وربطها بالمكونات المناسبة.

كل مسار يحتوي على path (المسار الذي يظهر في عنوان URL) و component (المكون المرتبط بالمسار.

.

***2. الحراسة (Guards)***

يُستخدم beforeEach لتحديد condition قبل دخول المسارات.

الحراسة تُستخدم لفحص شروط معينة قبل السماح بالانتقال إلى المسار المطلوب.

***3. حراسة التوثيق (Authentication Guards)***

يُستخدم meta: { requireAuth: true } لتحديد المسارات التي تتطلب مصادقة المستخدم قبل الوصول إليها.

يتم التحقق من وجود توكن مصادقة (JWT) قبل السماح بالوصول إلى المسارات المطلوبة، وإلا يتم توجيه المستخدم إلى صفحة تسجيل الدخول.

***4. حراسة التوثيق للمشرف (Admin Authentication Guards)***

يُستخدم meta: { requireAdminAuth: true } لتحديد المسارات التي تتطلب مصادقة المشرف قبل الوصول إليها.

يتم التحقق من وجود توكن مصادقة (JWT) وتأكيد صلاحيات المشرف قبل السماح بالوصول إلى المسارات المطلوبة.

**5. توجيه المستخدم بعد تسجيل الدخول (Redirect after Login)**

يُستخدم meta: { checkLgoin: true } لتحديد المسارات التي يتم توجيه المستخدم إليها بعد تسجيل الدخول.

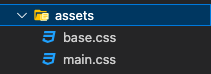
في حالة وجود توكن مصادقة (JWT) ومحاولة الوصول إلى صفحة تسجيل الدخول، يتم توجيه المستخدم إلى الصفحة الرئيسية.

***6. المسار الافتراضي (Default Path)***

يُحدد المسار NotFound للمسارات التي لا يتم العثور عليها، مثل الصفحات غير الموجودة.

يتم توجيه المستخدم إلى صفحة الخطأ 404 في حالة عدم وجود المسار المطلوب.

1. **assets folder**



1. base.css يحتوي على الأساسيات والأنماط الأساسية للموقع.
2. main.css يحتوي على الكود الرئيسي للتصميم مع إعادة تعيين (reset) الأساسية.
3. **Public folder**

A screenshot of a computer

Description automatically generated

Images -

هذا المجلد يحتوي على ملفات الصور المختلفة المستخدمة في جميع أنحاء الموقع، مثل:

ملفات صور أخرى تستخدم للرسوم التوضيحية، الخلفيات، أو أقسام معينة.

1. **Components folder**

A screenshot of a computer

Description automatically generated

يحتوي كل مجلد على مكون صفحته على سبيل المثال، يحتوي مجلد صفحة الدردشة على مكونات الدردشة وما إلى ذلك ….

**ملحوظات**

1. يتم تخزين رمز المستخدم (token) في ملفات تعريف الارتباط باسم jwt
2. إذا كان المستخدم هو المسؤول، فسيتم تخزين access إلى true في cookies