# **טכנולוגיות אינטרנט מתקדמות - 61776 (WEB)**

הגשת פרויקט

**Group #15 B20 Speech-to-text for language practice**



|  |  |
| --- | --- |
| שם חבר.ת הצוות | תז |
| Kfir Amoyal | 205710411 |
| Israel Ohayun | 312334717 |
| Fatmeh Zoabi | 212279855 |
| Vladi Trinter | 321257412 |

**Project Summary: AlefBet - Hebrew Learning Platform**

AlefBet is an interactive Hebrew learning platform designed to enhance vocabulary acquisition and pronunciation skills. The platform offers a user-friendly experience with personalized learning features, engaging quizzes, and seamless progress tracking.

#### **Guest Users**

* View the homepage with an engaging **Hero Carousel** highlighting key platform features.
* Explore the platform’s functionalities, including word translation and pronunciation.
* Navigate through different sections using the **Navbar** and **Hamburger Menu** for responsive access.

#### **Registered Users**

* **Personalized Word List:** Save and organize Hebrew words and sentences with **MyWords**.
* **Translation & Pronunciation:** Translate words using Google Translate API and listen to Hebrew pronunciation.
* **Gamified Learning:** Engage in quizzes to test knowledge and reinforce learning.
* **Learning Progress Tracking:** Monitor word retention, study streaks, and practice sessions in the **Learning Statistics** section.
* **Profile Management:** Update personal details and learning preferences in **ProfileInformation**.
* **User Authentication:** Secure sign-in and sign-up functionalities powered by Firebase.

AlefBet combines intuitive UI/UX with **Firebase authentication**, **Next.js SSR support**, and a **dynamic translation API**, ensuring an engaging and effective language-learning experience.

## Requirements Summary:

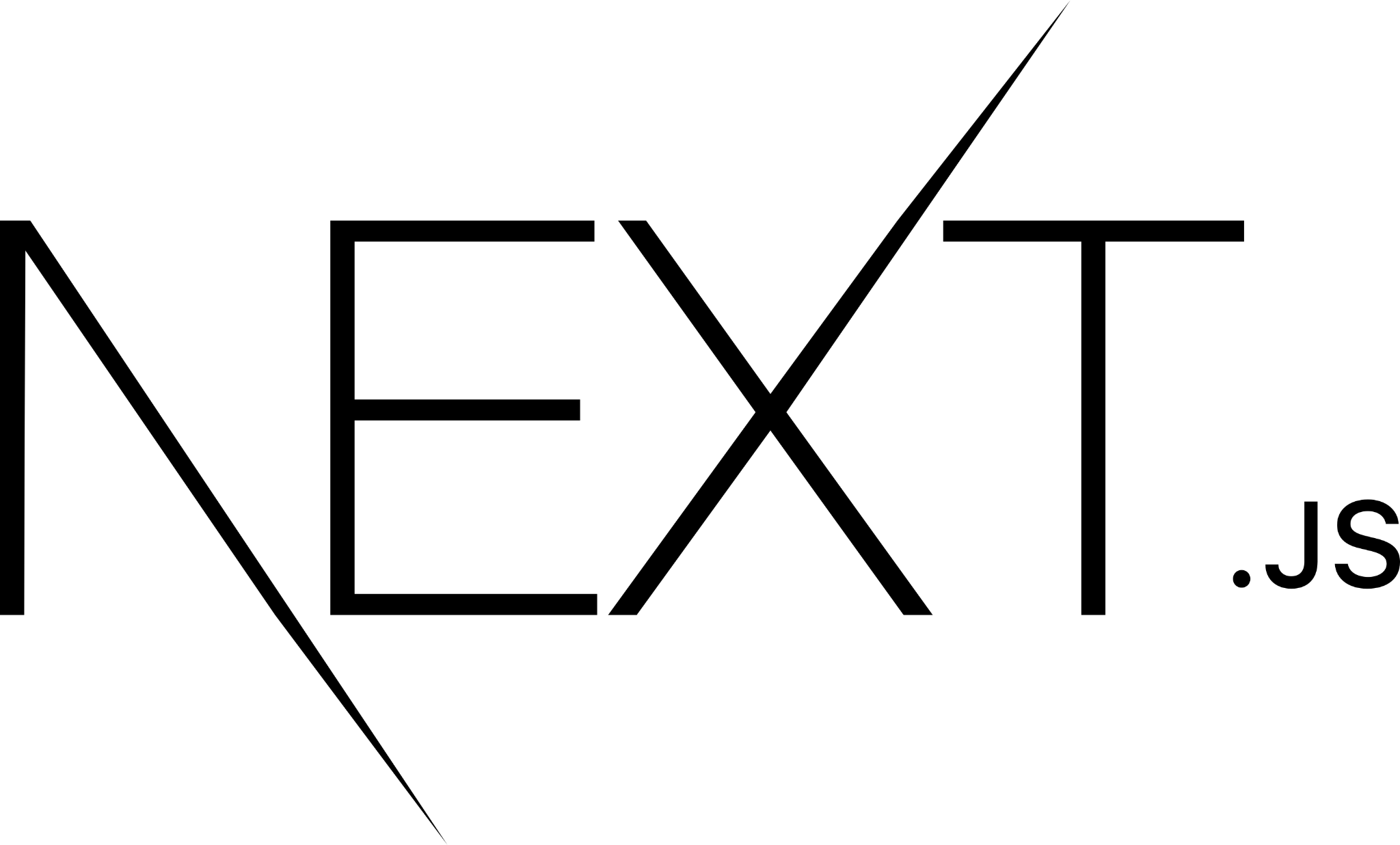
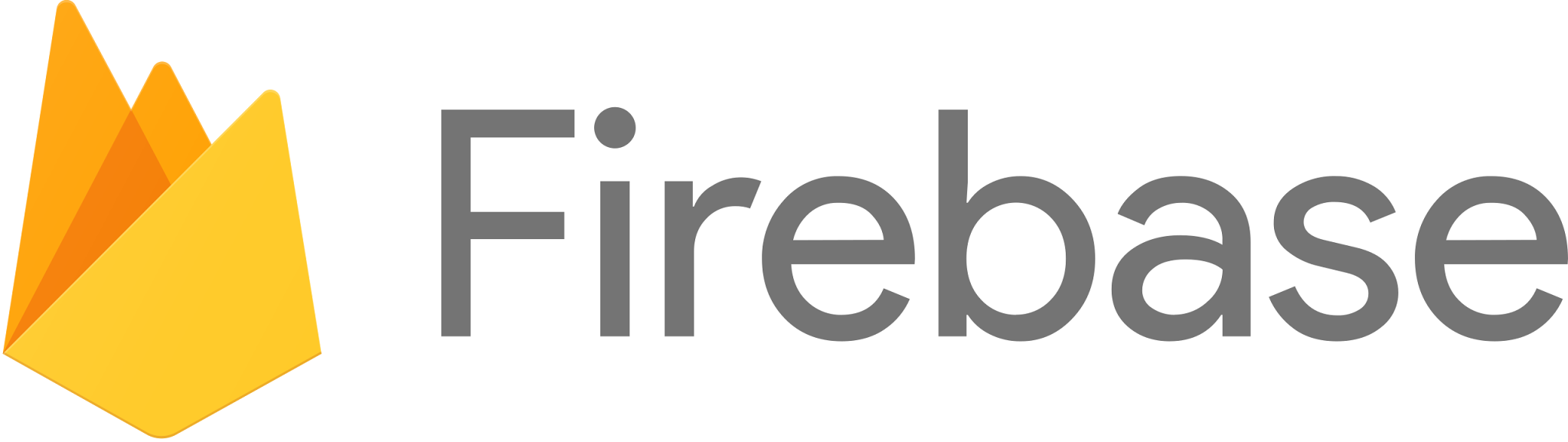
**Speech-to-text for language practice,Pronunciation scoring, Grammar correction,Vocabulary highlighting,Learning progress tracking**

The difficulties we have encountered and how we dealt with them:

[https://docs.google.com/presentation/d/1PwhJtREC4pXKo2vPygi\_eDHaQdJsxA5QG-T2EBei3d0/edit?usp](https://docs.google.com/presentation/d/1PwhJtREC4pXKo2vPygi_eDHaQdJsxA5QG-T2EBei3d0/edit?usp=sharing)

[=sharing](https://docs.google.com/presentation/d/1PwhJtREC4pXKo2vPygi_eDHaQdJsxA5QG-T2EBei3d0/edit?usp=sharing)

## **Main Technologies of Our Website:**

* Frontend - Next.js version 15 - directory app
* Backend - Next.js version 15 - directory app



* Database - Firebase - Firestore & Authentication
* Styling - Tailwind & MUI

### **Our GitHub repository:**

<https://github.com/kfiros94/WebDevelopment_G15>

GitHub Pages:

<https://github.com/kfiros94/WebDevelopment_G15/deployments/github-pages>

### **AlefBet website link:**

<https://web-development-g15.vercel.app/>

2. יש למנות מהנדס מערכת בכל צוות, אשר יהיה אחראי על הגדרת והקצאת המשימות בתרגיל זה.  
 נא לרשום את שם הסטודנט בתרגיל זה. על מהנדס המערכת לכתוב כיצד נעשתה חלוקת העבודה מול הצוות, מה היו המשימות של כל חבר צוות, האם היה ממשק בין חברי הצוות, והאם המשימות מולאו:  
**מהנדס המערכת שנבחר: פאטמה זועבי**

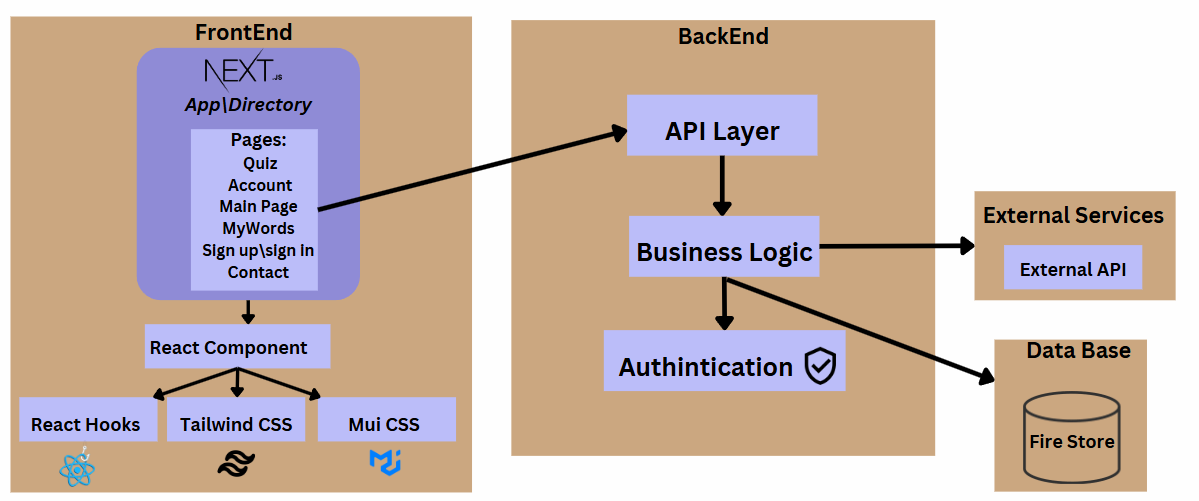
|  |  |  |
| --- | --- | --- |
| **שם חבר הצוות** | **משימות שהוקצו** | **משימות שהושלמו** |
| Kfir Amoyal, Fatmeh Zoabi | 1. מעבר על תקינות דרישות פונקציונליות ולא פונקציונליות. | **Done** |
| Israel Oyahon,  Fatmeh Zoabi | 2.בדיקה האם דיאגרמות ארכיטקטורה שלנו תקינות. | **Done** |
| Israel Oyahon,  Kfir Amoyal,  Vladi,  Fatmeh Zoabi | 3.בדיקה האם דיאגרמת ה-USE CASE שלנו תקינה | **Done** |
| Israel Oyahon,  Kfir Amoyal,  Vladi,  Fatmeh Zoabi | 4.נדרש לתקן את ה-USE CASE | **Done** |
| Israel Oyahon,  Kfir Amoyal, | 5.נדרש להוסיף דיאגרמה של היררכיית תיקיות וקבצים. | **Done** |
| Vladi,  Fatmeh Zoabi | 6**.יש לבנות תיק מתכנת לפי הדרישות הבאות:** 6.1 יש לתעד ולהסביר את כל הפונקציות המרכזיות בקוד שלכם | **Done** |
| Kfir Amoyal | 6.2 יש לפרט את כל ה -API שהשתמשתם בהם | **Done** |
| Kfir Amoyal | 6.3 לייצר קישור ל- DB (לתת הרשאות צפייה לנעמי או לברר מה לעשות) | **Done** |
| Israel Oyahon,  Kfir Amoyal | 6.4 הסבר על סביבות מיוחדות שהתקנתם | **Done** |
| Israel Oyahon,  Kfir Amoyal,  Vladi,  Fatmeh Zoabi | 6.5 שיתוף פרומפטים שהשתמשתם בהם בכלי AI לגבי שימוש בHOOKS | **Done** |
| Israel Oyahon | 7.**יש לבנות תיק מתכנת לפי הדרישות הבאות:**  7.1 צילומי מסך של האפליקציה | **Done** |
| Israel Oyahon | 7.2 הסבר שימוש על כל צילום מסך | **Done** |
| Israel Oyahon,  Kfir Amoyal,  Vladi,  Fatmeh Zoabi | 8.**סגירת קצוות בקוד של הפרויקט.** 8.1 תיקון רספונסיביות בכל הדפים. | **Done** |
| Kfir Amoyal | 8.2 סגירת הפיתוח של ה-QUIZ | **Done** |
| Israel Oyahon,  Kfir Amoyal,  Vladi,  Fatmeh Zoabi | 8.3 בדיקה + הוספת תיעוד בכל קבצי הפרויקט. | **Done** |

2. הציגו רשימת דרישות פונקציונליות ולא פונקציונליות (בנפרד, יש לסווג דרישות לא פונקציונליות לפי wikipedia NFR).

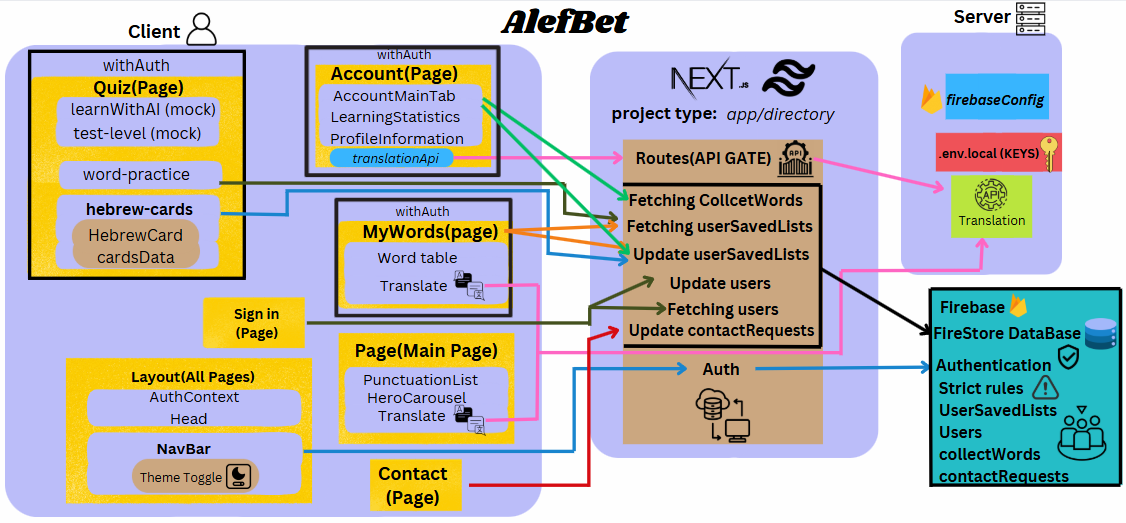
**Functional and non-functional requirements (according to Wikipedia (NFR)):**

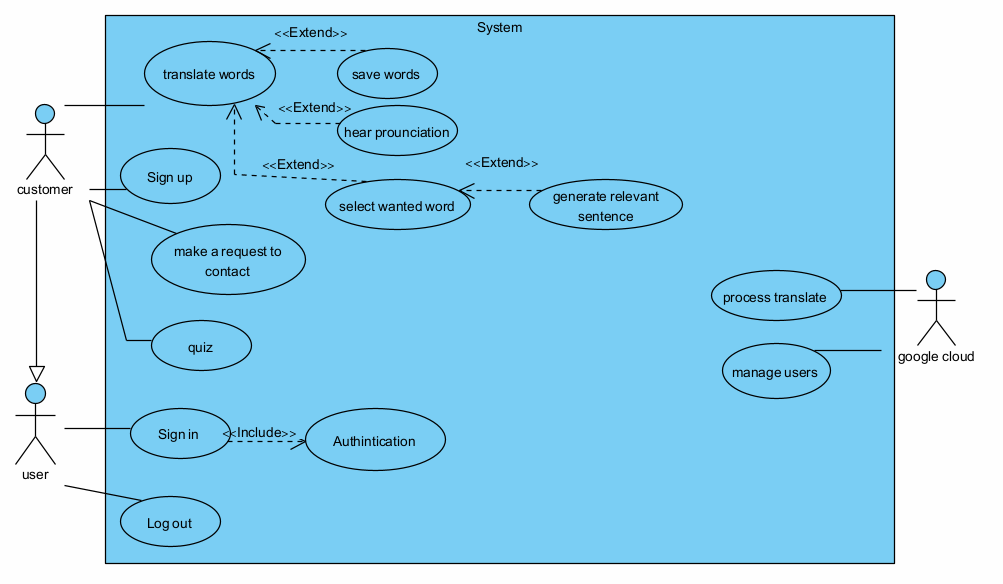
|  |  |
| --- | --- |
| FR | NFR |
| 1. The system will allow user input | 1.1 Usability - Input type will be text in English |
| 2. The system will allow input display | 2.1 Usability - Input display will be in English |
| 3. The system will allow website display switching | 3.1 Usability - Website display can be dark or light |
| 4. The system will allow website login | 4.1 Security - Login will be with private username and password |
|  | 4.2 Security - System administrators will independently add/remove users from the database |
| 5. The system will allow contact with system developers | 5.1 Interoperability -Messages will be sent directly to a dedicated email |
|  | 5.2 Data Integrity - Message must include: sender's name, reason for contact, email and/or mobile |
| 6. The system will allow automatic translation | 6.1 Usability - Translation will be from English to punctuated Hebrew |
| 7. The system will allow translation display | 7.1 Usability - Translation will be displayed in Hebrew to the user as text |
| 8. The system will allow text-to-speech | 8.1 Usability - The spoken text will be in Hebrew according to punctuation |
| 9. The system will allow word marking | 9.1 Usability - Words will be in punctuated Hebrew |
|  | 9.2 Usability - Marking will be for only one word in a sentence |
| 10. The system will display word list | 10.1 Usability - Word list will be according to the chosen word |
|  | 10.2 Usability - The list will contain the word with different punctuation |
| 11. The system will allow word selection | 11.1 Usability - The selected word is from the punctuated word list of (10) |
| 12. The system will perform sentence randomization | 12.1 Usability - Sentence randomization will be according to the selected punctuated word of (11) |
|  | 12.2 Usability - The randomised sentence will be in punctuated Hebrew |
|  | 12. Usability - The randomised sentence will be displayed to the user in a text box |
| 13. The system will allow creating new container | 13.1 Usability - The container will include:   * English text box * Translation button * Hebrew translation textbox * Audio button * Selected word list * Sentence randomization button * Save button |
| 14. The system will allow saving the card (container) | 14.1 Security - To save a card, the user must be logged in and registered in the system |
|  | 14.2 Usability - Saved cards will be displayed on another page called "My Cards" |
| 15. The system will allow displaying card list | 15.1 Usability - The main screen will display a temporary card list |
|  | 15.2 Usability - "My Cards" will display a list of saved cards |
| 16. The system will allow clearing the screen | 16.1 Usability - The screen will be cleared of the card list on the main screen |
| 17. The system will allow deleting cards | 17.1 Usability - Deleted cards will be from the "My Cards" list |
| 18. The system will be responsive to different screen sizes | 18.1 Compatibility - The system must function properly on mobile, tablet, and desktop devices |
|  | 18.2 Compatibility - The system must pass standard mobile responsiveness tests |

3. מבנה ואינטראקציה:  
 א. הציגו ארכיטקטורה מעודכנת של האתר (תרשים הכולל את האלמנטים המרכזיים)  
**(דיאגרמה ראשונה במבט על)**



**(דיאגרמה דפים וקומפוננטות והתקשורת ביניהם)**

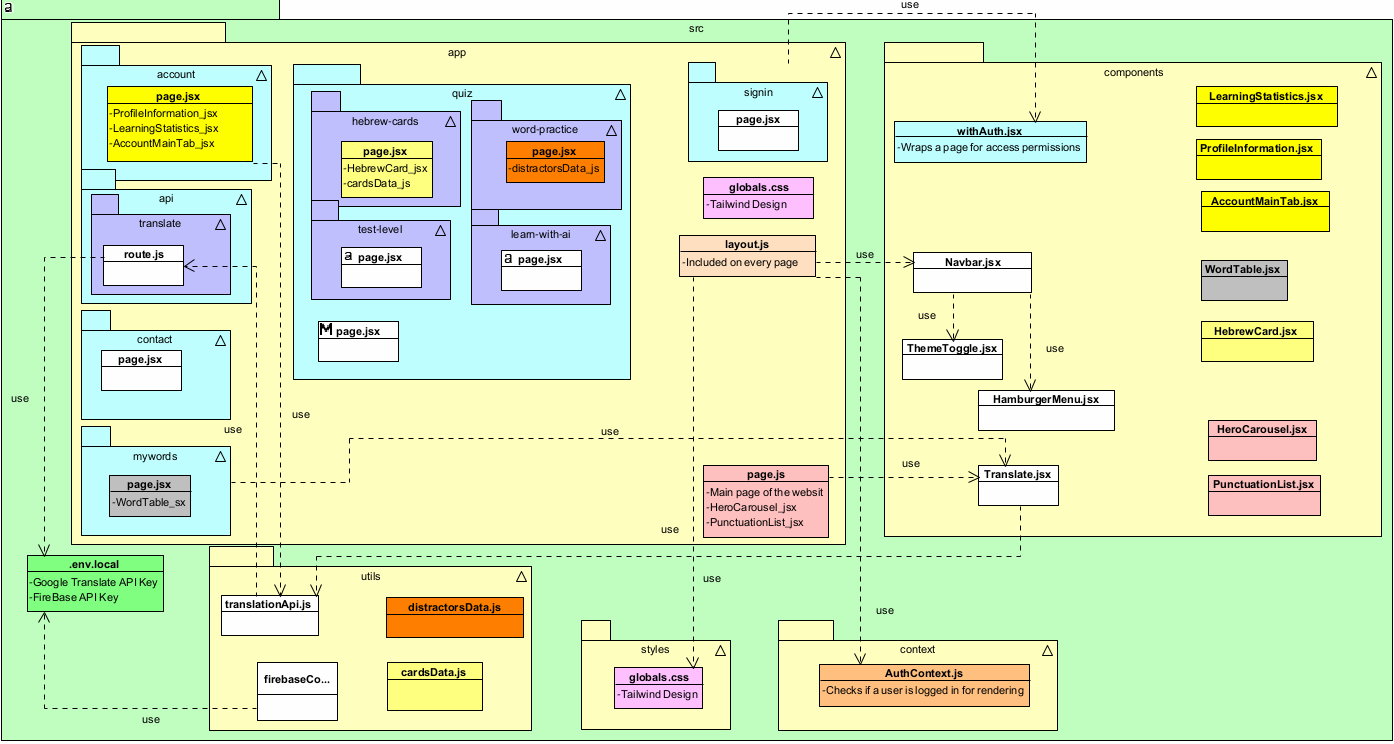


ב**.הציגו דיאגרמת use case המתארת את השימוש באתר:**

4. יש להציג מבנה סופי של האתר שלכם:

א. האתר מומש ב -Next גרסת app, וכן שימוש ב Tailwind:

**(דיאגרמה המתארת את התיקיות והקבצים השונים): יש חשיבות לצבעים**



ב. יש לפרט את פריטי המידע :  
**דיאגרמת מבנה DB:**



**פירוט על ה-API של Google Cloud Translate:** (השימוש תרגום מילים מאנגלית לעברית)

**Google Cloud Translate API** is a cloud-based service that enables real-time text translation between over 100 languages. It supports automatic language detection, batch translation, and custom models via AutoML.

**פירוט על ה-SpeechSynthesisUtterance API:**(השימוש מתרגם טקסט בעברית לקול בעברית)

**SpeechSynthesisUtterance API** is a web API that enables text-to-speech (TTS) functionality in browsers. It allows developers to convert text into spoken audio, control speech parameters (pitch, rate, volume), and select different voices. This API is part of the Web Speech API and works natively in modern browsers.

5. יש להגיש תיק למתכנת:

**א. יש לתעד ולהסביר את כל הפונקציות המרכזיות בקוד שלכם.**

|  |  |  |  |
| --- | --- | --- | --- |
| **page.js(Main page)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| punctuations (state variable) | Stores an array of punctuation objects received from the Translate component. | Initial value: []; updated via handleWordSelection callback. | An array of punctuation objects that is passed to the PunctuationList component for rendering. |
| isVisible  (state variable) | Controls the visibility (and thus the animation) of the Featured Content section. | Initial value: false; updated via the IntersectionObserver callback in the useEffect hook. | A boolean flag that, when true, triggers the CSS animation (e.g., slide-up/fade-in) for the section. |
| featuredRef  (ref hook) | Holds a reference to the Featured Content section’s DOM element, allowing the IntersectionObserver to monitor it. | Initial value: null; later set to the DOM element via the ref attribute. | A reference to the DOM element used to determine when the section is in view. |
| handleWordSelection  (function) | Called by the Translate component to update the punctuation list when new punctuated forms are found. | A parameter punctuatedForms (an array of punctuation objects). | Calls setPunctuations(punctuatedForms) to update the punctuations state. |
| clearPunctuationList (function) | Clears the punctuation list (useful when a user resets or clears input). | No input parameters. | Resets the punctuations state to an empty array ([]). |
| useEffect (hook for IntersectionObserver) | Sets up an IntersectionObserver on featuredRef so that when the Featured Content section becomes at least 10% visible, it triggers the animation. | Dependencies: [] (runs only once on mount); uses featuredRef.current to attach the observer. | When the element is at least 10% visible, setIsVisible(true) is called; also returns a cleanup function that unobserves the element on unmount. |
| HeroCarousel (component) | Renders the hero banner/carousel at the top of the page. | No props are passed. | Displays a carousel of hero images or content. |
| PunctuationList (component) | Displays a list of punctuation objects generated by the Translate component. | Props: punctuations (the punctuation array) and an onSelect callback (logs the selected word). | Renders a UI list of punctuation items and triggers the onSelect event when an item is clicked. |
| Translate (component) | Provides translation functionality by processing user input and determining punctuation forms. | Props: onWordSelection (to send back detected punctuation objects) and onClear (to clear the punctuation list). | Processes user input and invokes the appropriate callbacks to update or clear the punctuation list. |
| Home  (component) | The main page component that composes all the above elements, managing state and layout. | No external props; uses internal state and hooks. | Renders the complete home page layout including the HeroCarousel, PunctuationList, Translate, Featured Content, and Testimonial sections. |

|  |  |  |  |
| --- | --- | --- | --- |
| **layout.js(Contained on every page)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| metadata  (variable) | Contains the application’s metadata (title and description) used by the <Head> component for SEO and browser tab display. | None | An object: { title: "AlefBet", description: "Learn Hebrew with AlefBet" } |
| AuthProvider (component) | Wraps the entire application to provide authentication context, enabling any child component to access user/auth data. | Child components (via the children prop) | Provides an authentication context for the entire app |
| RootLayout (component) | The main layout component that structures the Next.js application with a consistent <html> structure, metadata, Navbar, main content, and footer. | children (the dynamic content of each page) | Renders the overall HTML layout including <html>, <Head>, <body>, <Navbar>, <main>, and <footer> elements |
| Head (component from next/head) | Manages document head elements such as the favicon, title, and meta description. | Elements: <link>, <title>, and <meta> tags | Injects the provided metadata into the document’s <head> section |
| Navbar (component) | Displays the navigation bar at the top of the page, offering consistent site navigation. | None (no explicit props passed) | Renders the navigation UI for the application |
| children (prop in RootLayout) | Represents the page-specific content that is passed into the layout. | Dynamic content from individual pages | Rendered within the <main> container, forming the primary content area |
| <main>  (HTML container) | Provides the main container for the page content with responsive layout and spacing. | Content via the children prop | Displays the main page content styled with Tailwind CSS utility classes |
| <footer>  (HTML element) | Renders a simple footer with copyright information at the bottom of the page. | Static text ("© 2024 AlefBet. All rights reserved.") | Displays the footer content on every page |

|  |  |  |  |
| --- | --- | --- | --- |
| **signin(page)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| SignInSignUp (component) | A combined sign-up/sign-in component that uses Firebase Authentication and Firestore to manage user accounts. Renders a responsive UI with a form (left) and an illustration (right). | Implicitly receives no external props; it manages its own state and uses internal functions. | A complete UI for user authentication with dynamic form fields, status messages, and navigation after sign in. |
| useRouter (hook) | Next.js hook used for client-side navigation. It allows redirection (e.g., after successful sign in). | No explicit input; called to obtain the router object. | A router object with methods like push for navigating between pages. |
| isSignUp  (state variable) | Boolean flag that determines if the component is in "Sign Up" mode or "Sign In" mode. | Initial value: false; toggled via user action (clicking the toggle text). | A boolean value used to conditionally render extra sign-up fields and adjust submission logic. |
| formData  (state variable) | Holds all input values from the form (email, password, username, firstName, lastName). | Initial value: an object with empty strings for each field; updated via the handleChange function. | An object representing the current form values, used in authentication requests. |
| message  (state variable) | Stores status messages (e.g., success notifications) to provide feedback to the user. | Initial value: ""; updated within handleSubmit upon successful operations. | A string displayed in the UI to inform the user about the operation’s result. |
| error  (state variable) | Holds error messages to provide feedback if something goes wrong during authentication. | Initial value: ""; updated in the catch block within handleSubmit. | A string displayed in the UI (typically styled in red) indicating error details. |
| handleChange (function) | Updates the formData state when any form input changes. | Receives an event object from an input’s onChange event (providing name and value). | An updated formData object with the modified field value. |
| handleSubmit (function) | Manages form submission for both sign-up and sign-in. For sign-up, it creates a user, saves additional data to Firestore, and signs out the user; for sign-in, it authenticates the user and navigates to the account page. | Receives the form submission event (e), plus uses formData and isSignUp for logic branching. | On sign-up: Creates a user, writes to Firestore, signs out, sets a success message, and toggles mode; on sign-in: Authenticates the user, sets a success message, and redirects to /account. In case of errors, sets an appropriate error message. |
| createUserWithEmailAndPassword (Firebase function) | Firebase function used to create a new user account using email and password. | auth instance, email, and password from formData. | A userCredential object containing the newly created user details upon successful account creation. |
| setDoc (Firestore function) | Saves additional user information in Firestore under a document path (users/{uid}). | A document reference (doc(db, "users", user.uid)) and an object containing all formData plus a default field (learningLevel: "Beginner"). | A promise that resolves when the user data is successfully written to Firestore. |
| signOut (Firebase function) | Signs out the current user from Firebase Authentication. | auth instance. | A promise that resolves upon successful sign-out, ensuring the user must sign in explicitly afterward. |
| signInWithEmailAndPassword (Firebase function) | Firebase function used to authenticate an existing user using email and password. | auth instance, email, and password from formData. | A userCredential object containing the authenticated user details upon successful sign in. |
| router.push('/account') (method call) | Navigates the user to the /account page after a successful sign-in. | A string path: "/account". | Triggers a client-side route change to the account page. |
| Toggle Link (UI element) | A clickable text element that toggles the component between sign-up and sign-in modes. | A click event from the user. | Inverts the isSignUp state, updating the UI to show either extra sign-up fields or the sign-in form. |
| Dynamic Form Fields (UI elements) | Renders different input fields based on the isSignUp state. Extra fields (username, firstName, lastName) appear only in sign-up mode, while email and password fields are always shown. | The current value of isSignUp and the formData values. | Conditional rendering of input fields, enabling users to provide the necessary information for sign-up or sign-in. |
| Status Messages (UI elements) | Displays success or error messages to give users feedback about their authentication attempt. | The message and error state values. | Rendered text elements in the UI (green for success, red for errors) informing the user of the outcome. |
| Right Section Illustration (UI elements) | Shows dynamic heading text, a descriptive paragraph, and an image that change based on the sign-up or sign-in mode. | The current isSignUp state. | A visually distinct section that displays either a "Join Us Today!" message with a sign-up illustration or a "Welcome!" message with a sign-in illustration. |

|  |  |  |  |
| --- | --- | --- | --- |
| **quiz(page)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| Quiz (component) | Main component that displays a list of image-based buttons for different quiz/learning paths. Wrapped with an authentication HOC to ensure only logged-in users can access it. Renders an animated headline and a responsive grid of clickable image buttons that navigate to quiz routes. | Implicitly receives no external props; utilizes internal variables, hooks, and functions. | A complete quiz UI with an animated headline and multiple image buttons that navigate the user to specific quiz pages upon click. |
| useRouter (hook) | Next.js hook for client-side routing. Provides navigation functionality (e.g., redirecting the user after a button click). | No explicit input. | A router object with methods such as push used for programmatic navigation. |
| buttons (variable) | An array of configuration objects for each quiz option. Each object specifies a background image URL, title, description, button width, and navigation path. | Statically defined data (no runtime input). | An array of button configuration objects used to dynamically render image buttons. |
| withAuth (Higher-Order Component) | An authentication wrapper that restricts access to the wrapped component, ensuring only authenticated users can view it. | Receives the Quiz component as its child. | A new component that enforces authentication before rendering the Quiz component. |
| ImageButton (styled component) | A styled version of MUI's ButtonBase that serves as the clickable image button. It manages dimensions, hover effects, and responsive behavior. | Receives styling props from the theme and inline style properties (e.g., width). | A styled button element that renders each quiz option with background imagery and hover transitions. |
| ImageSrc  (styled component) | A styled span that acts as the background image container for each button. It covers the entire button area. | Inline style for the background image URL. | A styled span element displaying the background image for a quiz option. |
| Image  (styled component) | A styled span that overlays the image, containing the title and description text centered over the background. | Theme-based inputs for typography and color styling. | A styled span element that renders the overlay text (title and description) on each button. |
| ImageBackdrop (styled component) | A styled span that provides a semi-transparent dark overlay over the background image to improve text readability, with a smooth opacity transition on hover. | Theme-based inputs for background color, opacity, and transition settings. | A styled span element that appears as a dark overlay on each image button, modifying opacity on hover. |
| ImageMarked (styled component) | A styled span that renders a small horizontal mark beneath the text on hover, serving as a decorative element. | Theme inputs for dimensions, color, and transition effects. | A styled span element that appears as a visual marker below the text during hover interactions. |
| StyledHeadline (styled component) | A styled MUI Typography component that displays an animated, gradient headline. It incorporates fade-in and scale-in keyframe animations. | Theme-based inputs for typography, gradient background, and inline keyframe definitions for animations. | A styled Typography element that renders an animated headline with gradient text, fade-in, and scale-in effects |

|  |  |  |  |
| --- | --- | --- | --- |
| **word-practice(page)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| WordPracticePage (Component) | A quiz-like component that fetches the user's saved words/sentences from Firestore, constructs multiple-choice questions with random distractors, and presents a quiz UI for practicing Hebrew. | No external props; relies on Firebase auth, Firestore, and a local distractors dataset. | Renders the complete quiz interface including question prompts, answer options, progress feedback, and end-of-quiz results. |
| useEffect (Hook) | On component mount, fetches the user's saved words from Firestore, formats them into quiz questions (with distractors), and updates state. | Triggered on mount (empty dependency array). | Updates the questions state with formatted quiz questions and sets loading to false once data is fetched. |
| handleAnswerClick (Function) | Checks the user's selected answer against the correct answer. If correct, advances to the next question (or ends the quiz if it was the last question); if incorrect, decreases the number of attempts and ends the quiz if no attempts remain. | Receives the user’s selected answer (a string) along with current question data from state. | Updates state variables: selectedAnswer, currentQuestionIndex, attemptsLeft, isQuizOver, and quizCompletedSuccessfully based on answer correctness. |
| handleRetry (Function) | Resets the quiz state to allow the user to try the quiz again. | No parameters. | Resets state variables (attemptsLeft, currentQuestionIndex, isQuizOver, selectedAnswer, and quizCompletedSuccessfully) to their initial values. |
| questions  (State Variable via useState) | Holds an array of formatted quiz question objects (each containing the English prompt, correct Hebrew answer, a shuffled array of options, and a flag indicating word vs. sentence). | Initial value: []; later updated by the useEffect hook. | An array of question objects for the quiz. |
| currentQuestionIndex (State Variable via useState) | Tracks which question in the quiz is currently being displayed. | Initial value: 0. | A number representing the current question's index. |
| selectedAnswer (State Variable via useState) | Stores the answer option that the user has selected for the current question. | Initial value: null. | A string representing the selected answer or null if none is selected. |
| attemptsLeft (State Variable via useState) | Keeps count of the remaining attempts for the current question. | Initial value: 3. | A numeric value indicating how many attempts remain. |
| isQuizOver (State Variable via useState) | Indicates whether the quiz has ended, either due to completion or because attempts ran out. | Initial value: false. | Boolean value (true when the quiz is finished). |
| quizCompletedSuccessfully (State Variable via useState) | Reflects whether the quiz was completed successfully (i.e., all questions answered correctly). | Initial value: false. | Boolean value; true if the quiz ended with success. |
| loading (State Variable via useState) | Indicates if the quiz data is still being fetched from Firestore. | Initial value: true. | Boolean value; becomes false once data has been loaded or an error occurs. |
| distractorsData (Variable) | Local dataset that provides arrays of distractor options for words and sentences used in constructing quiz questions. | Static data imported from a local file. | An object with arrays (e.g., distractorsData.words and distractorsData.sentences) used to populate multiple-choice options. |
| auth (Variable) | Firebase Authentication instance used to check if a user is logged in. | Imported from firebaseConfig. | Provides the current user's authentication details (e.g., via auth.currentUser). |
| db (Variable) | Firestore Database instance used to fetch the user's saved words/sentences. | Imported from firebaseConfig. | Used for Firestore operations such as retrieving document snapshots. |
| getDoc & doc (Firebase Functions) | Firestore functions used to create a document reference and fetch its data (the user's saved words/sentences). | doc is called with (db, "userSavedLists", user’s email); getDoc takes that document reference. | getDoc returns a document snapshot containing the saved words data (if it exists). |
| Loading UI (Box & Typography UI Elements) | Displays a loading message while quiz data is being fetched from Firestore. | Triggered when the loading state is true. | Renders a centered Box with a Typography element showing “Loading quiz...”. |
| Quiz Over UI (Box, Typography, Button, and Image UI Elements) | Shows the final screen when the quiz is over. Depending on the outcome, it displays either a success message (with an image) or a failure message, along with a "Retry Quiz" button. | Activated when isQuizOver is true; uses quizCompletedSuccessfully to choose between success or failure messages. | Renders a full-screen Box with styled Typography messages, an image (if the quiz was passed), and a Button that calls handleRetry when clicked. |
| Quiz Question UI (Box, Paper, Typography, LinearProgress, Button UI Elements) | Presents the current quiz question including the English prompt, a progress bar indicating quiz progress, a display of remaining attempts, and a list of answer options rendered as buttons. | Uses the current question object from questions[currentQuestionIndex], along with attemptsLeft and selectedAnswer state values. | Renders a Paper containing the question text, multiple-choice answer Buttons (with visual feedback based on selection), and a LinearProgress bar showing quiz progress. |

|  |  |  |  |
| --- | --- | --- | --- |
| **hebrew-cards(page)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| HebrewCardsPage (component) | Main component that displays Hebrew learning cards organized by categories. It allows the user to select a category from a dropdown and then shows the corresponding Hebrew cards in a grid. | No external props; uses internal state and imported data/components. | Renders a complete page with a header, a category dropdown, and a grid of HebrewCard components. |
| useState (hook) for selectedCategory | React hook used to manage the current category selection state. | Initial value: the first key from cardsData.categories (i.e., Object.keys(cardsData.categories)[0]). | A state tuple: selectedCategory (current category string) and setSelectedCategory (function to update it). |
| selectedCategory (state variable) | Holds the currently selected category that determines which set of Hebrew cards to display. | Initial value based on the first category key from cardsData.categories. | A string representing the current category; used to filter the cards displayed. |
| cardsData (variable) | Contains all card information and categorization data imported from a local file. | Static data imported from ../../../utils/cardsData. | An object with a categories property used to populate the dropdown and the cards grid. |
| HebrewCard (component) | A component that renders a single Hebrew learning card with its associated information. | Prop: a card object (including properties such as id and other card details). | Renders the UI for an individual Hebrew card. |
| Page Heading (UI element) | The header element that displays the title of the page ("Hebrew Cards Practice"). | Static text "Hebrew Cards Practice" and predefined Tailwind CSS classes for styling. | An <h1> element with prominent styling rendered at the top of the page. |
| Category Selector (UI element) | A dropdown (<select>) that allows the user to choose which category’s cards to display. | Uses selectedCategory as the current value and an onChange handler that updates this state. The options are generated by mapping over the keys of cardsData.categories. | Renders a styled dropdown that, when changed, updates the selectedCategory state to filter displayed cards. |
| Cards Grid (UI element) | A responsive grid layout that displays the Hebrew cards for the selected category. | Uses the selectedCategory state and the corresponding array from cardsData.categories to map and render each card via the HebrewCard component. | A grid of HebrewCard components rendered in a responsive layout according to the selected category. |

|  |  |  |  |
| --- | --- | --- | --- |
| **mywords(page)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| MyWords (component) | Manages the user's saved words and sentences. It fetches data from Firestore, enables adding new words or sentences (via translation and highlighting), and displays them in separate tables. | No external props; uses imported utilities (Firestore, Firebase auth, translation API) and child components. | Renders a full-page UI including translation input, action buttons, and tables for words and sentences; manages Firestore updates. |
| useState for words | Holds an array of word objects (each with English, Hebrew, pronunciation, and type "word") saved by the user. | Initial value: [] | Current list of saved word objects. |
| useState for sentences | Holds an array of sentence objects (each with English, Hebrew, pronunciation, and type "sentence") saved by the user. | Initial value: [] | Current list of saved sentence objects. |
| useState for highlightedWords | Tracks the Hebrew words that the user highlights during translation. | Initial value: [] | An array of highlighted Hebrew words selected by the user. |
| useState for translatedSentence | Stores the final translated Hebrew sentence received from the Translate component. | Initial value: null | A string representing the Hebrew translation of an input sentence. |
| useState for englishSentence | Stores the original English sentence input that is being translated. | Initial value: null | A string representing the original English sentence. |
| useState for userEmail | Keeps the user's email (from Firebase auth) for Firestore read/write operations. | Initial value: null | A string representing the current user's email address. |
| useEffect for fetching saved lists | On component mount, fetches the user's saved words/sentences from Firestore, formats them into word or sentence objects (using the transliterateHebrew function), and updates local state. | Runs once on mount (empty dependency array); uses auth.currentUser and Firestore's getDoc. | Updates the words and sentences state variables with the formatted data; if no document exists, initializes one with an empty array. |
| transliterateHebrew (function) | Converts Hebrew text to an approximate English phonetic representation using a character mapping. | A Hebrew text string. | A string where each Hebrew character is replaced by its corresponding English phonetic representation. |
| handleAddWords (function) | Processes highlighted words: translates each to English, creates word objects, updates the local words state, clears highlighted words, and updates the Firestore document. | Uses highlightedWords state and calls translateText for each word. | Updated words state with new word objects; Firestore document updated with new "English -> Hebrew" entries; cleared highlightedWords state. |
| handleAddSentence (function) | Saves a new sentence by combining the original English sentence and its translated Hebrew version. It creates a sentence object, updates local state, and writes the update to Firestore. | Uses translatedSentence and englishSentence state variables. | Updated sentences state with the new sentence object; Firestore document updated with new sentence entry; resets translation input state. |
| deleteWord (function) | Removes a word or sentence from the corresponding local state array and updates the Firestore document to reflect the deletion. | Receives an index and a type ("word" or "sentence"). | Updated words or sentences state (with the selected item removed); Firestore document updated with the revised list. |
| playAudio (function) | Uses the browser's SpeechSynthesis API to pronounce the given Hebrew text aloud. | A Hebrew text string. | Initiates speech synthesis to speak the text; alerts the user if the API is unsupported. |
| Translate (child UI component) | Handles translation input from the user. It provides callbacks when translation is complete, words are highlighted, or input is cleared. | Props: onTranslationComplete, onWordSelection, and onClear callbacks. | Calls the provided callbacks with the English input, translated Hebrew sentence, and an array of highlighted words. |
| WordTable (child UI component for Words) | Displays a table of saved word objects with interactive options (delete, play audio). | Props: words (the words state array), onDelete callback, and onPlayAudio callback. | Renders a table listing words with buttons to delete or play the word's audio. |
| WordTable (child UI component for Sentences) | Displays a table of saved sentence objects with interactive options (delete, play audio). | Props: words (actually the sentences state array), onDelete callback, and onPlayAudio callback. | Renders a table listing sentences with buttons to delete or play the sentence's audio. |
| Action Buttons (UI elements: Save Words & Save Sentence) | Two buttons that allow the user to save highlighted words or the translated sentence. Their appearance and enabled state depend on whether the necessary data is present. | For "Save Words": depends on highlightedWords.length; for "Save Sentence": depends on translatedSentence state. | On click, invoke handleAddWords or handleAddSentence, updating state and Firestore accordingly. |

|  |  |  |  |
| --- | --- | --- | --- |
| **contact(page)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| ContactUsPage (component) | A contact page that renders a two-column layout: one section with static contact details (phone, email, location, working hours) and another with a contact form. Also displays a responsive penguin image at the bottom. | Implicitly receives no external props; uses internal state and functions. | Renders the complete contact page UI with form inputs, status messages, and contact information. |
| useState for formData | Manages the form input values including first name, last name, mobile, email, and message. | Initial value: { firstName: "", lastName: "", mobile: "", email: "", message: "" } | An object representing current form input data and a setter (setFormData) to update it. |
| useState for status | Holds status messages (success or error) to inform the user after submitting the form. | Initial value: "" | A string representing the current status message and a setter (setStatus) to update it. |
| handleChange (function) | Updates the corresponding field in formData when the user types into an input or textarea. | An event object from the input’s onChange event (providing name and value). | Updates the formData state with the new value for the targeted field. |
| handleSubmit (function) | Triggered when the user submits the contact form; prevents default submission, saves the form data to Firestore with a timestamp, updates the status message, and resets the form. | The form submission event (e) and the current formData state. | On success: sets a success status message and resets formData; on failure: sets an error status message. |
| addDoc (Firebase function) | Saves a new document in Firestore’s contactRequests collection containing the form data and a timestamp. | A Firestore collection reference (from collection(db, "contactRequests")) and an object with formData plus timestamp: new Date(). | A promise that resolves when the document is successfully added to Firestore. |
| collection (Firestore function) | Generates a reference to the contactRequests collection in Firestore. | The Firestore instance (db) and the collection name "contactRequests". | A reference used by addDoc to add a new document. |
| Left Section – Contact Details (UI element) | Displays static contact information such as phone number, email, location, and working hours, each paired with a relevant icon. | Static data (contact details and icon image sources). | Renders a styled block with contact information for the user. |
| Right Section – Contact Form (UI element) | Presents the contact form with inputs for first name, last name, mobile, email, and message along with a submit button; displays status messages after submission. | Uses formData and status state values; input events trigger handleChange and form submission triggers handleSubmit. | Renders a functional contact form UI that lets users send a message and see feedback. |
| Penguin Image (UI element) | Displays a responsive penguin image at the bottom of the page as a decorative element. | Image source /pingwingSimple.png, alt text, and responsive styling classes. | Renders a styled image that adjusts responsively based on screen size. |

|  |  |  |  |
| --- | --- | --- | --- |
| **account(page)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| Account (component) | Main account page that renders user profile information, learning statistics, and account management tabs. It also manages state for a punctuation list and monitors the visibility of a featured section. | No external props; wrapped by the authentication HOC (withAuth) | Renders the account page UI with a grid layout for profile, statistics, and main tabs; manages internal state updates. |
| useState (punctuations) | State hook that stores an array of punctuation objects (e.g., words selected elsewhere). | Initial value: [] | Current punctuation list used by the component (updated via callbacks). |
| useState (isVisible) | State hook that tracks whether a "Featured Content" section is visible in the viewport. | Initial value: false | A boolean that becomes true when the observed element is at least 10% visible. |
| useRef (featuredRef) | Creates a mutable ref intended to reference the DOM element for the featured content section. | Initial value: null | A ref object to be attached to a DOM element; used by the IntersectionObserver (though not assigned in the current JSX). |
| handleWordSelection (function) | Callback function that updates the punctuations state when words are selected. | Parameter: punctuatedForms (an array of punctuation objects) | Updates the punctuations state with the provided array. |
| clearPunctuationList (function) | Resets the punctuation list state to an empty array. | No parameters. | Sets the punctuations state to []. |
| useEffect (IntersectionObserver) | Sets up an IntersectionObserver on the element referenced by featuredRef so that when at least 10% of the element is visible, isVisible is set to true. | Dependency: [featuredRef]; runs on component mount and when featuredRef changes | When the observed element is in view, updates isVisible to true; cleans up the observer on unmount. |
| ProfileInformation (component) | Displays the user's profile details such as name, picture, etc. | No props provided. | Renders profile information in the first section of the grid layout. |
| LearningStatistics (component) | Shows the user's learning progress and statistics. | No props provided. | Renders learning statistics alongside profile information within the grid layout. |
| AccountMainTab (component) | Provides the main account tab or navigation area for additional account-related features. | No props provided. | Renders the main account content in the lower part of the grid layout. |
| withAuth (HOC) | Higher-order component that wraps the Account component to enforce that only authenticated users can access it. | Receives the Account component as input. | Returns an enhanced component that restricts access to authenticated users. |

|  |  |  |  |
| --- | --- | --- | --- |
| **api \ translate(API)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| POST (API endpoint function) | Handles incoming POST requests to translate text using the Google Translate API. Expects a JSON payload containing text and an optional target language code. | HTTP Request with JSON body: { text, target } | A JSON response (using NextResponse.json) containing either the translated text or an error message. |
| NextResponse (API response helper) | Utility from Next.js used to construct HTTP responses with JSON payloads and specific HTTP status codes. | A JSON object and an optional status code (e.g., { translatedText: ... } or error message with { status: 400/500 }) | An HTTP response returned to the client. |
| req.json() (method on Request) | Parses the incoming request body as JSON to extract necessary fields (text and target). | The raw HTTP request body. | A JavaScript object with properties text and target. |
| Input Validation | Checks that the text field exists and is not empty. If validation fails, an error response is returned immediately. | The extracted text from the JSON payload. | If invalid, returns a JSON error response with status 400: { error: "Please provide a sentence to translate." } |
| Environment Variable: apiKey | Retrieves the Google Translate API key from environment variables to authenticate the API call. | process.env.GOOGLE\_TRANSLATE\_API\_KEY | An API key string used in the request URL. |
| fetch (HTTP call to Google Translate API) | Sends a POST request to the Google Translate API endpoint with the provided text and target language. | URL constructed with the API key; request body: JSON stringified object with `{ q: text, target: target |  |
| Error Handling | Checks whether the response from the Translate API is successful. If not, it parses the error details and throws an error, which is caught in the catch block. | Non-OK HTTP response and error JSON data from the API. | Throws an error that is caught and results in a JSON response with status 500 and a generic error message. |
| Data Parsing | Extracts the translated text from the successful API response JSON. | The parsed JSON from the Translate API response. | An object with the property translatedText (accessed as data?.data?.translations[0]?.translatedText). |
| Return Response | Returns the final JSON response to the client, either with the translation result or an error message. | The output from the data parsing or error handling process. | A NextResponse JSON object with either { translatedText: "..." } or an error object. |

|  |  |  |  |
| --- | --- | --- | --- |
| **AccountMainTab(components)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| punctuationMarks (variable) | An array of default punctuation marks used when no constraints apply. | None | Array of Hebrew punctuation marks (e.g. "ָ", "ַ", "ִ", ...) |
| punctuationConstraints (variable) | An object mapping specific Hebrew letters to their allowed punctuation marks (or none). | None | Object with keys as letters and values as arrays of allowed punctuation marks |
| addRandomPunctuation (function) | Appends a random punctuation mark to each character in a given Hebrew word, respecting letter-specific constraints. | A Hebrew word (string) | A new string with each character followed by a randomly chosen punctuation mark (or none) |
| AccountMainTab (component) | Renders a multi-container interface for a mini "sentence translator" workflow. Users can input a sentence, translate it, view and interact with translated words, and save their results. | No external props; uses imported APIs and Firestore utilities | A complete UI section that displays a list of container instances with input areas, translation results, word lists, and action buttons |
| useState for userEmail | Stores the currently logged-in user’s email for Firestore operations. | Initial value: null | A string representing the user's email (or null if not logged in) |
| useState for collectWords | Holds Firestore data from the collectWords collection; used for grammar/dotted word lookup. | Initial value: {} | An object mapping base words to their associated data (e.g. dotted forms and meanings) |
| useState for containers | Maintains an array of container objects representing individual sentence translator workflows. Each container includes an ID, input text, translated words, a word list (punctuatedWords), and a selected meaning. | Initial value: An array with one default container object | An array of container objects used to render separate translator instances |
| useEffect (auth listener) | On mount, checks for an authenticated user and listens for auth state changes to update userEmail. | None | Updates userEmail state based on Firebase auth (either sets user email or null) |
| useEffect (fetch collectWords) | Fetches all documents from the Firestore collectWords collection and stores them in state as an object for fast lookup. | None | Updates collectWords state with an object where keys are document IDs and values are their data |
| handleAddContainer (function) | Adds a new container instance to allow the user to start a fresh sentence translation workflow. | None | Appends a new container object (with default values) to the containers state |
| handleClearContainers (function) | Resets the interface by clearing all existing containers and leaving a single fresh container. | None | Sets containers state to an array with one default container object |
| handleRemoveContainer (function) | Removes a specific container (by ID) from the containers state. | A container’s unique ID | Updated containers state with the selected container removed |
| handleTranslate (function) | Uses the translateText API to translate the input sentence of a specific container. Splits the returned translation into individual words and updates the container’s translatedWords (while resetting word list and selection). | A container ID and its inputText (string) | Updates the selected container’s translatedWords with an array of word objects (each with an id and text) |
| handleWordClick (function) | When a translated word is clicked, cleans the word (removing punctuation) and looks it up in collectWords to update the container’s word list (punctuatedWords). | A container ID and the clicked word (string) | Updates the container’s punctuatedWords with dotted forms from collectWords or a fallback message if unavailable |
| handleWordListClick (function) | On clicking a word from the displayed word list, looks up the corresponding meaning from collectWords and sets it as the container’s selectedMeaning. | A container ID and the clicked dotted word (string) | Updates the container’s selectedMeaning with the associated meaning object (or sets it to null) |
| handlePlayPronunciation (function) | Combines the container’s translated words into a sentence and uses the browser’s SpeechSynthesis API to pronounce it in Hebrew. | A container ID | Initiates audio playback of the container’s translated sentence (or alerts if unavailable) |
| handleSave (function) | Saves the user’s input sentence and its translated result to Firestore under userSavedLists/{userEmail}. | A container ID (and its inputText and translated sentence) | Updates the Firestore document by appending a new "English -> Hebrew" entry; shows success or error alerts |
| handleInputChange (function) | Updates the inputText property of a specific container when the user types in the textarea. | A container ID and new text value | Updates the container’s inputText in the state |
| Clear All Button (UI element) | A button at the top-right of the component that clears all containers and resets the interface. | Click event | Triggers handleClearContainers to reset containers state |
| Main Wrapper (UI element) | A scrollable section that wraps all container instances, allowing multiple translator workflows to be displayed. | Receives containers state for mapping | Renders each container in a vertically spaced, scrollable layout |
| Container – Word List Area (UI element) | Displays the current container’s punctuatedWords as a list; each word is clickable to trigger handleWordListClick. | Uses the container’s punctuatedWords array | Renders a styled list of words (or fallback messages) for grammar lookup |
| Container – Main Area (UI element) | Includes a textarea for input, a Translate button, a display area for translated words (each clickable to trigger handleWordClick), and buttons for audio playback and removal. | Uses container properties: inputText and translatedWords | Renders the interactive translation interface for that container |
| Create New Sentence Button (UI element) | A button that, when clicked, calls handleAddContainer to add a new translator container. | Click event | Appends a new container object to the containers state |

|  |  |  |  |
| --- | --- | --- | --- |
| **HamburgerMenu(components)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| HamburgerMenu (component) | A responsive navigation drawer that slides in from the right. It displays navigation links and toggles sign in/up or sign out based on the user state. | Props: user (object or null), handleSignOut (function) | Renders a hamburger icon button and a Drawer containing menu items and authentication controls. |
| useState (hook for drawerOpen) | Manages whether the Drawer is open (true) or closed (false). | Initial value: false | A boolean state variable (drawerOpen) controlling the visibility of the Drawer. |
| toggleDrawer (function) | Returns an event handler that toggles the drawerOpen state based on a boolean input. It ignores keyboard events for 'Tab' and 'Shift'. | Input: A boolean (open) and an event object | Updates drawerOpen state to either true (open) or false (closed). |
| IconButton (UI element for hamburger icon) | A clickable button that displays the hamburger (menu) icon on small screens. | Click event (handled by toggleDrawer(true)) | Opens the Drawer when clicked. |
| MenuIcon (UI element) | MUI icon component representing the hamburger menu icon. | No additional input (used as child of IconButton) | Provides a visual hamburger icon with responsive color styling. |
| Drawer (UI element) | Material UI component that slides in from the right, acting as a navigation drawer. | Props: anchor="right", open={drawerOpen}, onClose={toggleDrawer(false)} | Renders a side-drawer that displays menu links and closes when clicking outside or pressing the close button. |
| List, ListItem, ListItemButton, ListItemText (UI elements) | Compose the navigational menu within the Drawer. Each clickable item (wrapped in a Next.js Link) navigates to a different route and closes the Drawer. | Input: Each ListItemButton receives a click event and a destination href (e.g., /mywords, /account, /contact, /quiz) | Renders a list of navigation links that close the Drawer upon clicking. |
| Sign In/Sign Out Control (UI element) | Conditionally renders a navigation item: if user exists, shows a "Sign Out" button that calls handleSignOut; otherwise, shows a "Sign In/Up" link. | If user is truthy: click event calls handleSignOut and closes the Drawer; if falsy: renders a Link to /signin | Displays a user icon alongside the appropriate text ("Sign Out" or "Sign In/Up") and triggers authentication actions on click. |
| CloseIcon (UI element) | MUI icon component representing a close ("X") icon used in the Drawer header. | No additional input (used as child of IconButton) | Provides a visual cue for closing the Drawer. |
| userIcon (variable/UI element) | An imported SVG image used as a user icon in the Sign In/Sign Out control. | No dynamic input; static asset imported from local files. | Displays a small user icon next to the authentication text in the Drawer. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Navbar(components)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| Navbar (component) | Top-level navigation bar that displays the site logo, greeting (if signed in), desktop navigation links, mobile hamburger menu, sign in/up or sign out options, and a theme toggle button. | No external props; relies on internal state, Firebase auth, Firestore data, and imported components | Renders a responsive navigation bar with different layouts for desktop and mobile, and updates based on user authentication state. |
| useState (user, setUser) | Stores the current authenticated user object (or null if not signed in). | Initial value: null | A state variable (user) that holds the Firebase user object once authenticated. |
| useState (firstName, setFirstName) | Holds the first name of the current user for display in the greeting message. | Initial value: "" (empty string) | A state variable (firstName) used to display a personalized greeting (e.g., "Hello, John!"). |
| useEffect (auth state listener) | Subscribes to Firebase auth state changes. When a user signs in, it fetches the user's Firestore document to retrieve the firstName field. | onAuthStateChanged(auth, callback) callback receives the current user object | Updates user and firstName state variables based on authentication status; unsubscribes on unmount. |
| handleSignOut (function) | Prompts the user for confirmation and, if confirmed, signs out via Firebase Auth and resets the user state. | Uses window.confirm and calls Firebase's signOut(auth) | Signs out the current user and sets user state to null. |
| Link with logo (UI element) | A clickable element linking to the home page that displays the AlefBet logo and site name. | Uses imported logo asset (its src) and a text label "AlefBet" | Renders a logo image and text that navigates to the home page ("/") when clicked. |
| Greeting Message (UI element) | Displays a personalized greeting ("Hello, {firstName}!") when a user is signed in. | Depends on user and firstName state | Renders a text span with a greeting message next to the logo. |
| Desktop Links (UI elements) | A set of navigation links (using Next.js Link and MUI Button) to pages such as My Words, Quiz, Account, and Contact Us. Shown on large screens (lg+). | Each Link receives an href (e.g., "/mywords", "/quiz") and button styling props | Renders a row of buttons that navigate to the respective pages when clicked. |
| Sign Out / Sign In/Up Button (UI element) | Conditionally renders either a Sign Out button (if user exists) or a Sign In/Up link (if not authenticated), each accompanied by a user icon. | Uses user state and handleSignOut for the sign-out action; otherwise, links to "/signin" | Renders the appropriate button or link that triggers authentication actions (sign-out or navigation to sign in/up). |
| ThemeToggle (UI element) | Renders a button or control that toggles between light and dark themes. | No additional input; imported from ./ThemeToggle | Displays a theme toggle control that adjusts the UI theme when interacted with. |
| Mobile Hamburger Menu (UI element) | A dynamically imported HamburgerMenu component that appears on screens smaller than lg; it provides mobile-friendly navigation and sign-out options. | Props passed: user and handleSignOut | Renders a responsive side-drawer navigation menu for mobile us |

|  |  |  |  |
| --- | --- | --- | --- |
| **ThemeToggle(components)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| ThemeToggle (component) | A functional React component that provides a toggle button to switch between "light" and "dark" themes. It reads the saved theme from localStorage (or falls back to the system preference) and applies the corresponding class to the document root. | No external props; relies on browser localStorage and window.matchMedia | Renders a button displaying an SVG icon (sun or moon) based on the current theme; clicking the button toggles the theme and updates localStorage and the document class. |
| useState (hook for theme) | Declares a state variable (theme) to store the current theme mode. | Initial value: "light" | Returns the current theme value and a setter function (setTheme) for updating the theme. |
| useEffect (initial theme setup) | Runs once on mount to check for a saved theme in localStorage or use the system's preferred color scheme. It then updates the theme state and toggles the "dark" class on the document's root accordingly. | No external input; reads from localStorage.getItem("theme") and window.matchMedia("(prefers-color-scheme: dark)") | Updates the theme state with the saved or default theme and applies or removes the "dark" class on document.documentElement. |
| toggleTheme (function) | Toggles the theme between "light" and "dark". It updates the state, modifies the document's class list, and persists the new theme in localStorage. | Uses the current value of theme from state | Sets a new theme value ("dark" if current is "light" and vice versa), updates localStorage, and toggles the "dark" class on the document. |
| Button (UI element) | A clickable button that triggers the toggleTheme function when pressed. It conditionally renders one of two SVG icons based on the current theme. | OnClick event (calls toggleTheme); relies on the current theme state | Returns a styled button element containing an SVG icon representing either the sun (if in dark mode) or the moon (if in light mode). |
| SVG Icons (UI elements) | Two inline SVG elements that visually represent the current theme: a sun icon (displayed when in dark mode, to indicate a switch to light mode) and a moon icon (displayed when in light mode, to indicate a switch to dark mode). | Depends on the theme state value | Renders the appropriate SVG icon within the button based on whether theme is "dark" or "light". |

|  |  |  |  |
| --- | --- | --- | --- |
| **Translate(components)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| Translate (component) | Provides a user interface to input an English sentence, translate it to Hebrew, display the translation, allow individual word selection (with callback support), and play the translation via speech synthesis. | Props: onTranslationComplete, onWordSelection, onClear; internal state values | Renders an input field, action buttons (Translate, Clear, Play Pronunciation), and a clickable display of the translated text |
| useState for englishInput | Holds the current English sentence entered by the user. | Initial value: "" (empty string) updated via input change events. | Current value of the user's English input. |
| useState for translation | Stores the Hebrew translation returned from the translation API. | Initial value: "" updated in the translation process. | A string containing the translated Hebrew text. |
| useState for loading | Tracks whether the translation process is in progress to disable buttons and display loading text. | Initial value: false; updated during the API call in handleTranslate. | A boolean indicating if the translation is ongoing. |
| useState for selectedWord | Stores the Hebrew word that the user has clicked on for highlighting. | Initial value: ""; updated via handleWordClick. | A string representing the currently selected word. |
| handleTranslate (function) | Validates the input, calls the translateText API (targeting Hebrew), updates the translation state, and triggers the parent callback with the English input and translated result. | Uses the current englishInput state; target language hard-coded to "he". | Updates translation with the Hebrew text and calls onTranslationComplete(englishInput, translatedText). |
| handleClear (function) | Resets the component’s state by clearing the input, translation, and selected word; also calls the parent’s onClear callback if provided. | Triggered by the Clear button’s click event. | Resets englishInput, translation, and selectedWord to empty values and calls onClear(). |
| handleWordClick (function) | Sets the clicked word as selected and (if provided) passes an array with a word object (including a dummy meaning) to the parent callback. | Receives a word (string) from the translated text display when a word is clicked. | Updates selectedWord and calls onWordSelection([{ word, meaning: \Meaning of ${word}` }])`. |
| handlePlayPronunciation (function) | Uses the browser’s SpeechSynthesis API to speak the translated Hebrew text aloud. | Uses the current translation state. | Initiates speech synthesis (or alerts if not supported) to pronounce the translated text. |
| Input Field (UI element) | A text input where users type an English sentence to be translated. | Value bound to englishInput; onChange events update this state. | Updates the englishInput state with user input. |
| Translate Button (UI element) | Triggers the translation process when clicked. Displays "Translating..." during the API call. | Click event; disabled when loading is true. | Calls handleTranslate to perform the translation. |
| Clear Button (UI element) | Clears the current input, translation, and selection when clicked. | Click event. | Calls handleClear to reset relevant state values. |
| Translated Text Display (UI element) | Shows the translated Hebrew text, split into individual words. Each word is clickable to trigger handleWordClick and is highlighted if selected. | Renders based on the translation state string. | Displays clickable words with visual highlighting for the selected word. |
| Play Pronunciation Button (UI element) | When clicked, plays the translated Hebrew text using speech synthesis. | Click event; visible only if a translation exists. | Calls handlePlayPronunciation to read the translation aloud. |

|  |  |  |  |
| --- | --- | --- | --- |
| **withAuth(components)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| withAuth (Higher-Order Component) | Wraps a given component to enforce authentication. It listens for Firebase auth state changes and, if no user is authenticated, redirects to the sign-in page; otherwise, it renders the wrapped component. | A component to be wrapped (WrappedComponent) and any additional props passed to the returned component. | Returns a new component that either shows a loading message, redirects to /signin, or renders the wrapped component with its props. |
| useRouter (hook) | Next.js hook that provides access to client-side navigation methods, such as redirecting the user. | No explicit input; invoked inside the returned component. | A router object with methods like push, used to navigate (e.g., redirecting to /signin). |
| useState (hook for loading) | Manages a boolean state (loading) indicating whether the authentication check is still in progress. | Initial value: true (indicating loading until auth status is determined). | A state variable (loading) that is set to false once a user is authenticated; remains true during the auth check. |
| useEffect (auth state subscription) | Sets up a subscription to Firebase authentication state changes using onAuthStateChanged. If no user is found, it redirects to /signin; if a user is present, it updates the loading state to allow rendering of the wrapped component. | Dependency: [router]; executed on mount; uses auth instance and a callback function. | Executes the callback on auth state changes, triggering router.push("/signin") when no user exists or setting loading to false when a user is found. Returns an unsubscribe function for cleanup. |
| onAuthStateChanged (Firebase function) | Listens for changes in the user's authentication state. Calls a provided callback with the current user object (or null if not signed in). | Input: The auth instance and a callback function that receives a user object. | Invokes the callback with the current user status; returns an unsubscribe function to cancel the listener on unmount. |
| Loading Message (UI element) | A simple UI element displayed while the authentication state is being determined. | Rendered when loading is true. | A <div> element with a "Loading..." message centered on the screen. |
| Render Wrapped Component (UI element) | Once the user is authenticated (i.e., loading is false), the higher-order component renders the component passed as WrappedComponent with all its props. | Provided props forwarded to WrappedComponent after the auth check passes. | The output of <WrappedComponent {...props} />, allowing the wrapped component to render normally for authenticated users. |

|  |  |  |  |
| --- | --- | --- | --- |
| **AuthContext(components)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| AuthContext (variable) | A React context created with createContext to hold and share the authentication state (user and loading) across the application. | None (initialized without a default value) | A context object that can be provided to and consumed by descendant components. |
| useState (hook for user) | Manages the state for the currently authenticated user. Initially set to null until a user logs in or out. | Initial value: null | The current user object (or null if no user is authenticated) along with its setter (setUser). |
| useState (hook for loading) | Manages a boolean state that indicates whether the authentication check is still in progress. | Initial value: true | A boolean (loading) that is true while the auth state is being determined and false afterwards. |
| useEffect (in AuthProvider) | Subscribes to Firebase authentication state changes using onAuthStateChanged. It updates the user state with the current user (or null) and sets loading to false once the initial auth state is resolved. | Uses the auth instance and a callback that receives currentUser from Firebase. | Triggers updates to user and loading state; returns an unsubscribe function to clean up the listener on unmount. |
| AuthContext.Provider (JSX element) | Wraps its children with the AuthContext, providing the current values of user and loading so that they can be accessed anywhere in the component tree. | Props: value={{ user, loading }} and any children components. | Renders the children with access to the authentication context values. |
| useAuth (custom hook) | A helper function that wraps useContext(AuthContext) to allow easy consumption of the authentication context within any component. | None (internally uses AuthContext) | Returns an object containing the current authentication state, e.g., { user, loading }. |

|  |  |  |  |
| --- | --- | --- | --- |
| **firebaseConfig(Function)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| firebaseConfig (variable) | An object holding the Firebase configuration parameters sourced from environment variables. | Environment variables (process.env.NEXT\_PUBLIC\_FIREBASE\_API\_KEY, etc.) | A configuration object with keys such as apiKey, authDomain, projectId, etc. |
| app (variable) | The initialized Firebase application instance created by calling initializeApp with the configuration object. | The firebaseConfig object. | A Firebase app instance used to initialize Firebase services. |
| db (variable) | The Firestore database instance initialized by calling getFirestore with the Firebase app instance. | The Firebase app instance (app). | A Firestore database instance used for data storage and retrieval. |
| auth (variable) | The Firebase Authentication instance initialized by calling getAuth with the Firebase app instance. | The Firebase app instance (app). | A Firebase Authentication instance used to manage user authentication. |

|  |  |  |  |
| --- | --- | --- | --- |
| **translationApi(Function)** | | | |
| Component \ Function \ Hooks \ Variable | Description | Input | Output |
| translateText (async function) | An asynchronous function that translates a given text to a specified target language by sending a POST request to the /api/translate endpoint. It validates the input, handles API responses and errors, and returns the translated text or an appropriate error message. | text (string): The sentence to be translated.  target (string, default "he"): The target language code (e.g., "he" for Hebrew). | A Promise that resolves to a string containing the translated text, or an error message if the translation fails or if the input is invalid. |
| Input Validation | Checks if the provided text is non-empty after trimming. If the input is empty, it immediately returns a prompt for the user to enter a sentence. | The text input (string) | If empty after trimming, returns: "Please enter a sentence to translate." |
| Fetch Request | Constructs and sends a POST request to the /api/translate endpoint with a JSON payload containing the text and target language. | JSON payload: { text, target } with headers { "Content-Type": "application/json" } | A Response object from the API call |
| Response Handling | Checks whether the API response is OK. If not, it extracts error details and throws an error. If the response is successful, it parses the JSON to extract the translated text. | The Response object from the fetch call | On success: returns the value of data.translatedText (or "No translation available." if missing) |
| Error Handling | Catches any errors during the fetch process, logs them to the console, and returns a generic error message to inform the user. | An error thrown during the fetch or response handling process | Returns: "Failed to fetch translation. Please try again later." |

|  |  |  |
| --- | --- | --- |
| **.env.local(KEY)** | | |
| Component \ Function \ Hooks \ Variable | Description | Value |
| GOOGLE\_TRANSLATE\_API\_KEY | API key for the Google Translate API | Confidential Subscription with credit card Payment |
| NEXT\_PUBLIC\_FIREBASE\_API\_KEY | Firebase API key used for client-side Firebase initialization | Confidential Subscription with credit card Payment |
| NEXT\_PUBLIC\_FIREBASE\_AUTH\_DOMAIN | Firebase Authentication domain | Confidential Subscription with credit card Payment |
| NEXT\_PUBLIC\_FIREBASE\_PROJECT\_ID | Firebase Project ID | Confidential Subscription with credit card Payment |
| NEXT\_PUBLIC\_FIREBASE\_STORAGE\_BUCKET | Firebase Storage Bucket | Confidential Subscription with credit card Payment |
| NEXT\_PUBLIC\_FIREBASE\_MESSAGING\_SENDER\_ID | Firebase Messaging Sender ID | Confidential Subscription with credit card Payment |
| NEXT\_PUBLIC\_FIREBASE\_APP\_ID | Firebase App ID | Confidential Subscription with credit card Payment |
| NEXT\_PUBLIC\_FIREBASE\_MEASUREMENT\_ID | Firebase Measurement ID | Confidential Subscription with credit card Payment |

**ב. יש לפרט את כל ה -API שהשתמשתם בהם:**

**פירוט על ה-API של Google Cloud Translate:** (השימוש תרגום מילים מאנגלית לעברית)

**פירוט על ה-SpeechSynthesisUtterance API:**(השימוש מתרגם טקסט בעברית לקול בעברית)

**קישור ל- DB: ( נדרש הרשאות גישה למייל מאושר בלבד)**

[**https://console.firebase.google.com/u/0/project/web-g15/overview**](https://console.firebase.google.com/u/0/project/web-g15/overview)

## **1️⃣ התקנת Next.js**

תחילה, אם אין לך **Node.js**, תוריד ותתקין [מכאן](https://nodejs.org/).

לאחר מכן, צור פרויקט Next.js חדש עם הפקודה:

sh

npx create-next-app@latest my-hebrew-learning-app

cd my-hebrew-learning-app

או אם אתה משתמש ב- yarn:

sh

yarn create next-app my-hebrew-learning-app

cd my-hebrew-learning-app

💡 **במידה ואתה ממיר פרויקט קיים מ-React ל-Next.js, שים לב להשתמש בארכיטקטורת src/app.**

## **2️⃣ התקנת Tailwind CSS**

Tailwind הוא ה- CSS Framework שבו השתמשת בפרויקט.

sh

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p

לאחר מכן, פתח את **tailwind.config.js** ועדכן אותו כך:

js

/\*\* @type {import('tailwindcss').Config} \*/

module.exports = {

content: ["./src/\*\*/\*.{js,ts,jsx,tsx}"],

theme: {

extend: {},

},

plugins: [],

};

לבסוף, הוסף את Tailwind לקובץ ה- CSS הראשי שלך (למשל, globals.css):

css

@tailwind base;

@tailwind components;

@tailwind utilities;

## **3️⃣ התקנת Firebase**

כדי להשתמש ב- Firebase לאימות משתמשים וניהול מסד נתונים:

sh

npm install firebase

🔹 יצירת קובץ firebaseConfig.js בתיקיית src/utils/ (ראה דוגמאות שלך) עם פרטי החיבור מה- Firebase Console.

## **4️⃣ התקנת Material UI (MUI)**

sh

npm install @mui/material @emotion/react @emotion/styled

npm install @mui/icons-material

📌 ספרייה זו מספקת כפתורים מוכנים, טפסים, ועוד.

## **5️⃣ התקנת Swiper.js (לקרוסלה)**

sh

npm install swiper

ולא לשכוח להוסיף import "swiper/css"; ברכיבים שמשתמשים בו.

## **6️⃣ התקנת תמיכה ב- Google Translate API**

משתמשים ב- **fetch** כדי לקרוא ל- API, אך אם רוצים להשתמש בספרייה:

sh

npm install google-translate-api

במקרה שלך, אתה משתמש ב- **Next.js API Routes**, אז רק צריך לוודא שה- **GOOGLE\_TRANSLATE\_API\_KEY** מוגדר בקובץ .env.local.

## **7️⃣ התקנת Icons ב- Next.js**

בפרויקט שלך השתמשת גם ב- **react-icons** וגם ב- **@mui/icons-material**. אם תרצה עוד אייקונים:

sh

npm install react-icons

# 

# User Manual

**Introduction**

Welcome to AlefBet, a platform for learning Hebrew. The site allows users to learn, practice, and improve their Hebrew skills through word translation, personalized word lists, quizzes, and more.



## **Navigating the Website**

The website includes several main pages:

1. **Home Page** - Displays information about the site, promotional banners, and quick access to translation.
2. **Sign In/Sign Up Page** - Allows users to register a new account or log in to access full features.
3. **User Account** - Manage profile, learning statistics, and word history.
4. **Personal Word List** - Save translated words for future practice.
5. **Contact Page** - Reach out to the AlefBet support team. Click [here](https://web-development-g15.vercel.app/contact) for quick access.
6. **Quizzes & Tests** - Check your Hebrew knowledge with interactive quizzes.

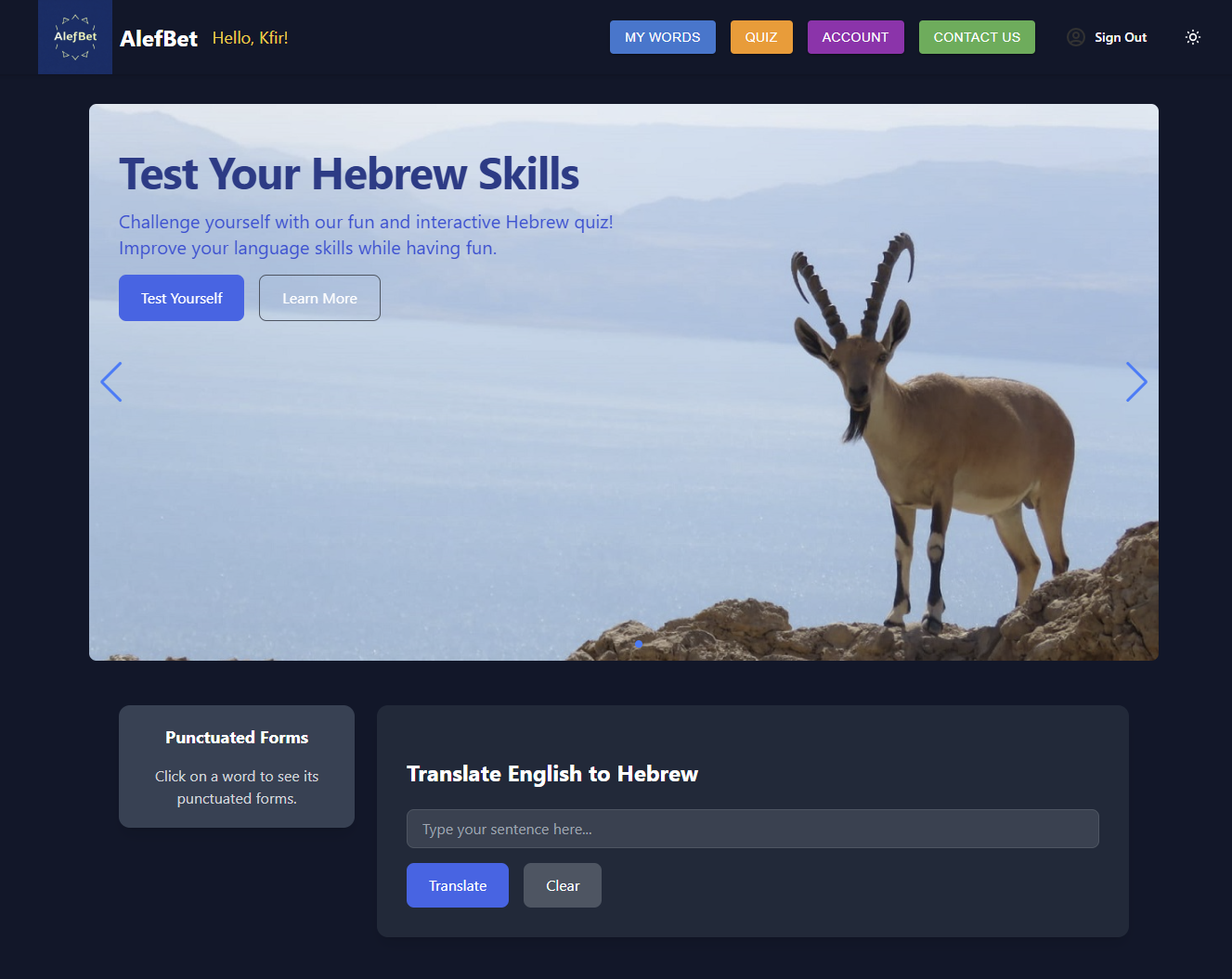
* Notice that Some pages are only accessible to registered users. See the explanation below for more details.



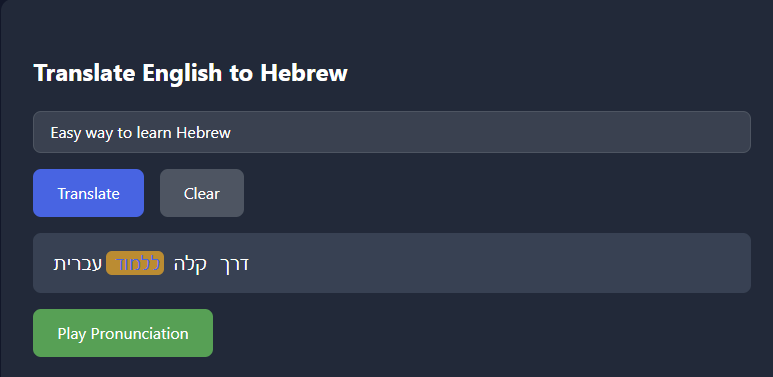
## 

## **Using the Website as a Guest (Not Logged In)**

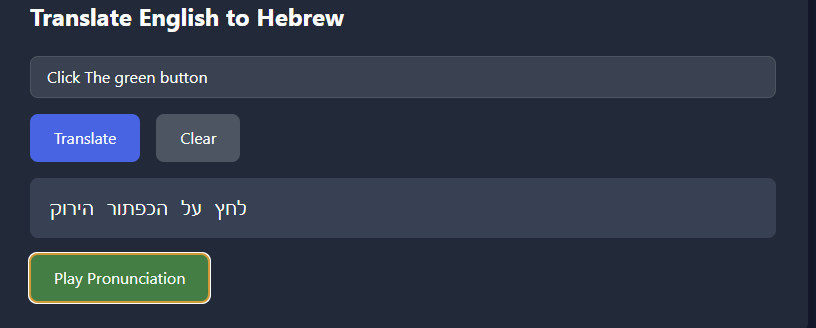
1. **View the Home Page** - Learn about the site, view promotional banners, and explore the available services.



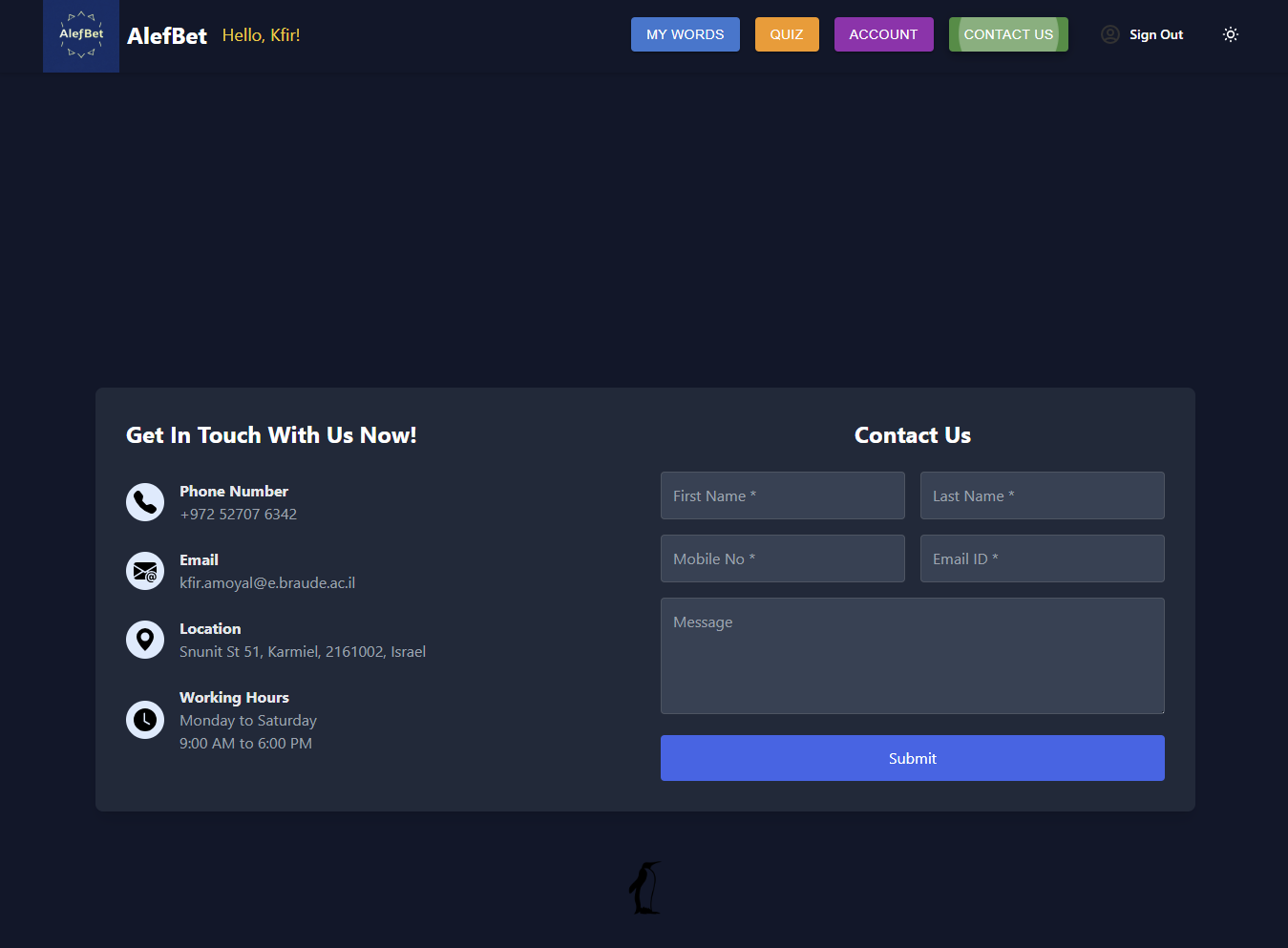
1. **Use Basic Translation** - Enter English text and translate it into Hebrew, but saving translations is not possible.



1. **Play Pronunciation** - After translating an English word/sentence you can play the audio of the Hebrew pronunciation. Click the green ‘Play Prononciation’ button.



1. **Access the Contact Page** - Send messages to the AlefBet team.

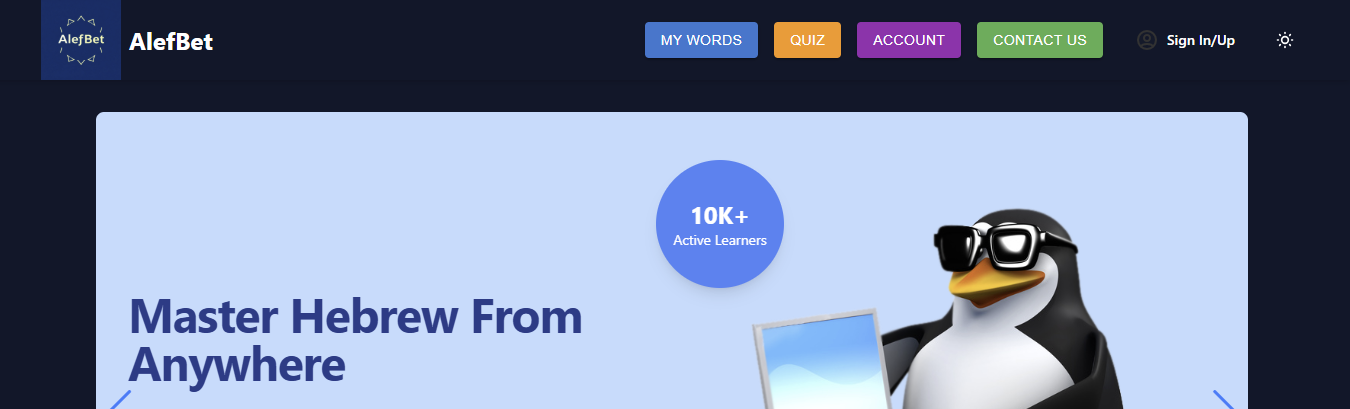


1. **Sign Up for an Account** - New users can register to gain access to full features.

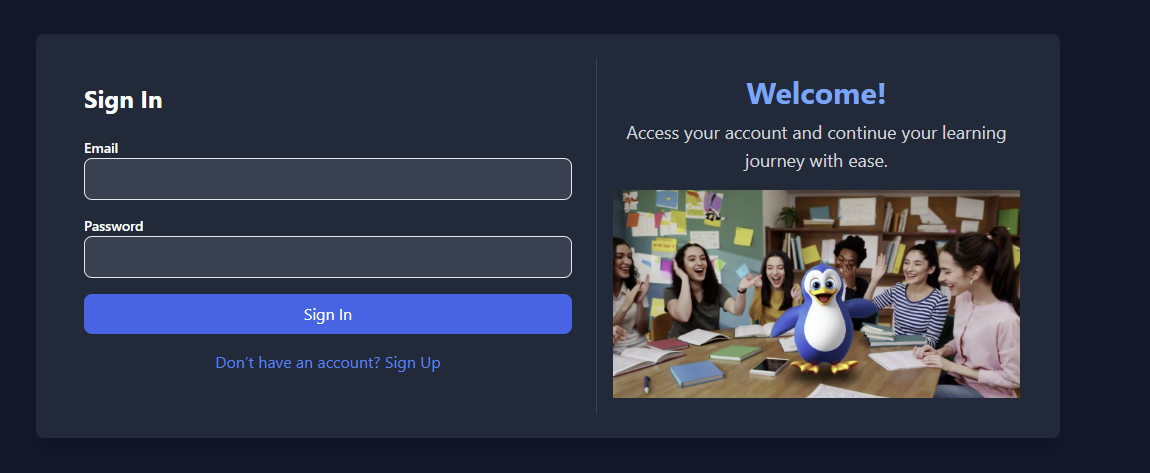


### **How to Register as a New User**

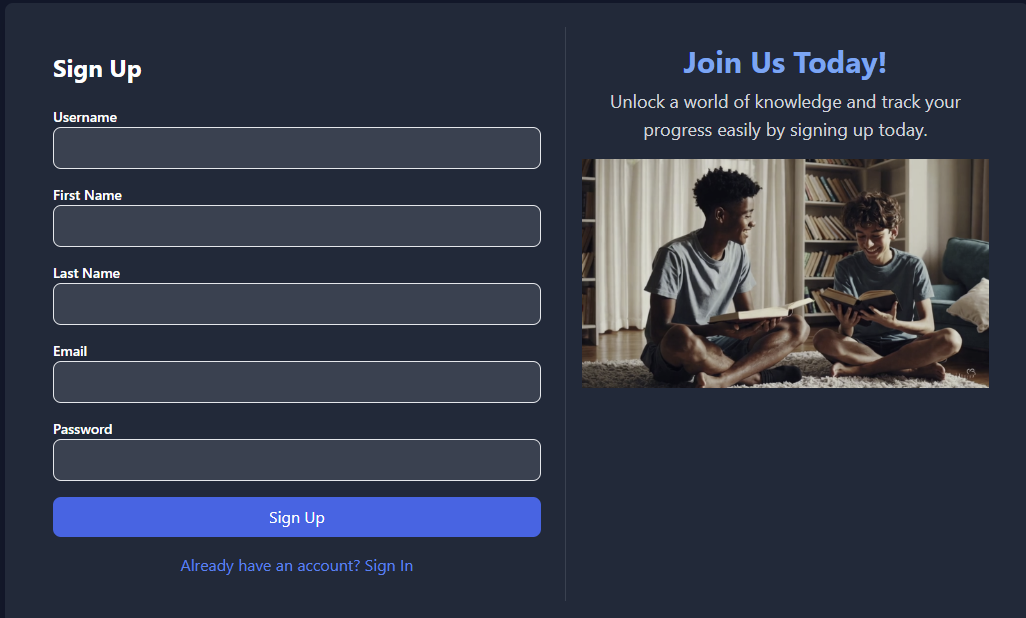
1. Navigate to the **Sign In/Sign Up** page from the main menu.



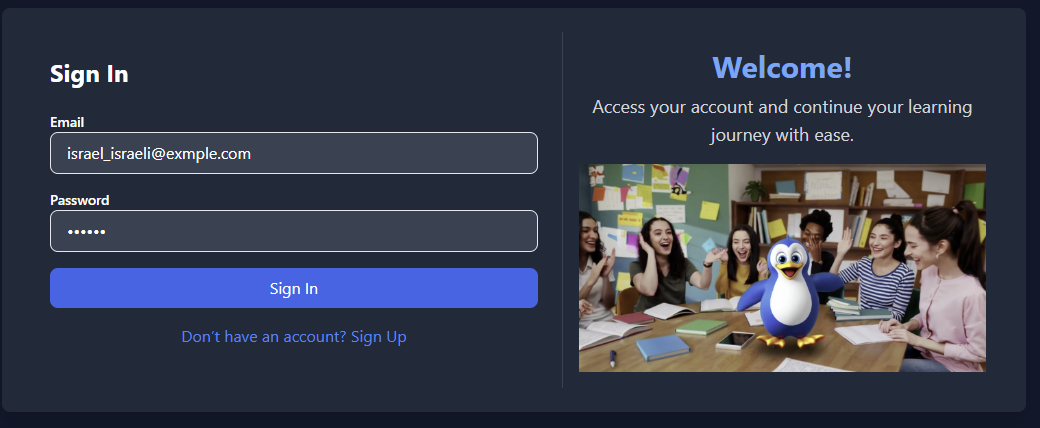
1. If you are not registered click “Don’t have an account? Sign Up”



1. Fill out the registration form, including first name, last name, email, and password. Notice that the password needs to be 6 or more characters.



1. After registering, the user is redirected to the login page. Fill in your username and password. Then click “Sign In”and start your Hebrew journey.

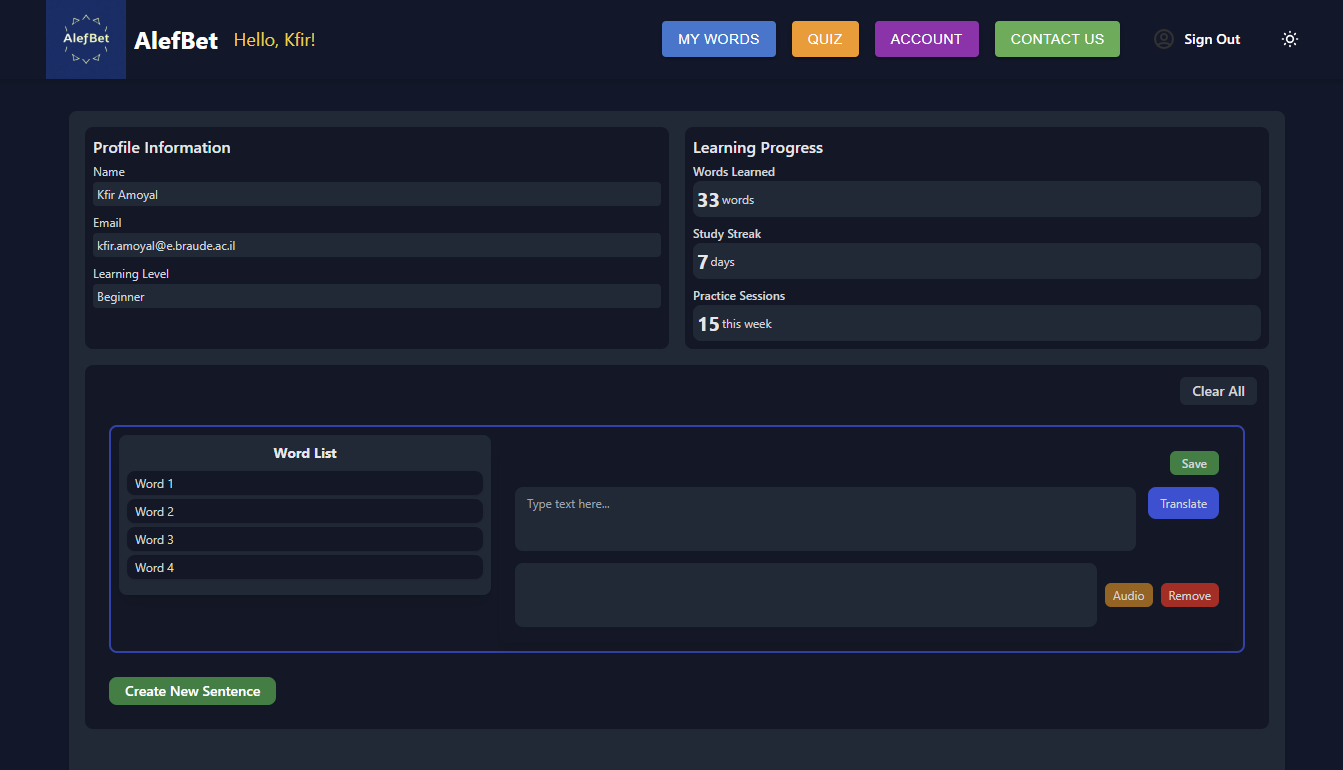


## **Using the Website as a Registered User (Logged In)**

Registered and logged-in users gain access to all features:

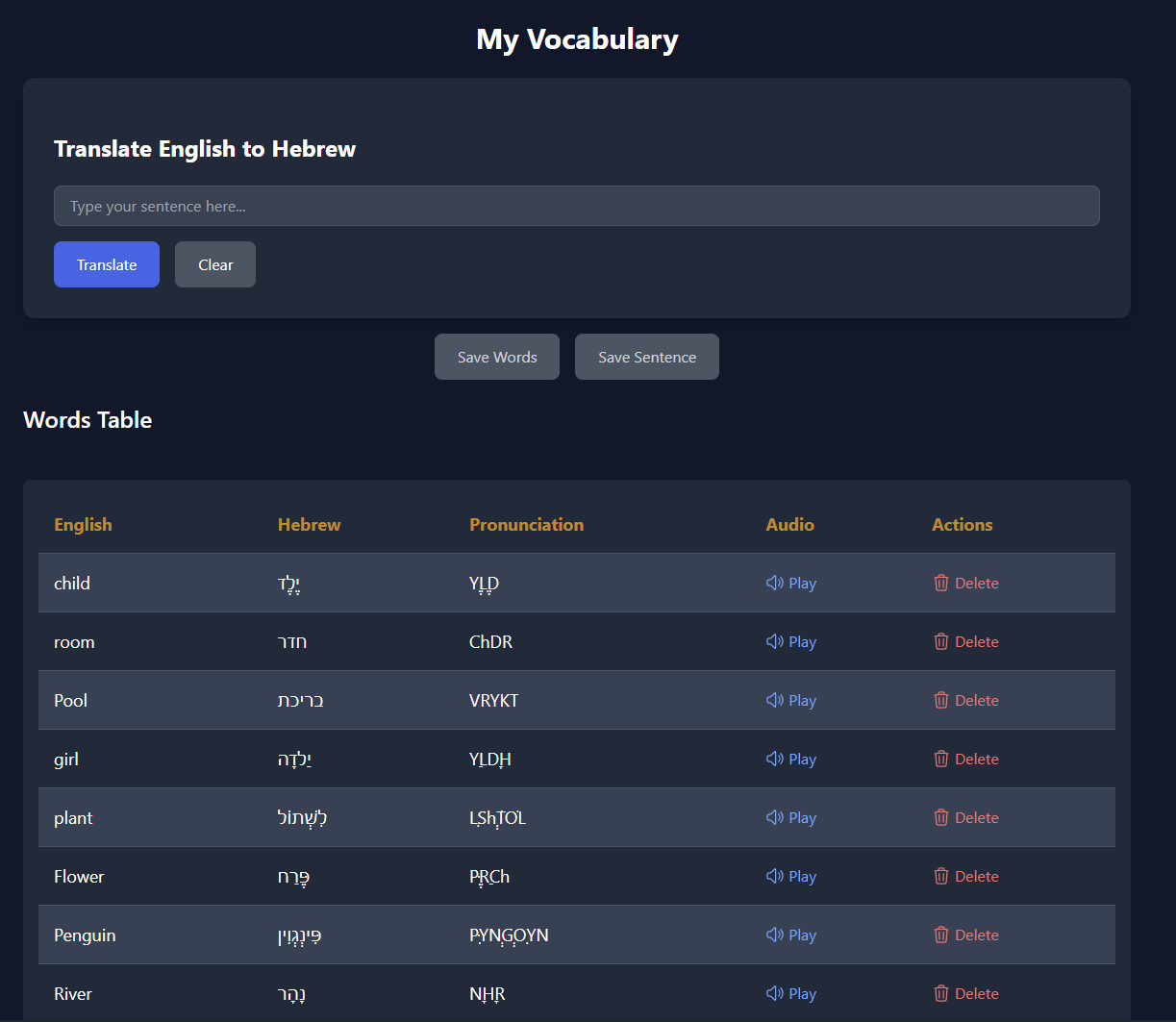
1. **Access User Account** - Manage personal details, update learning levels, and track learning progress.

[**Page Guide**](#_paanbxy55hkp)



1. **Save Words to a Personal Word List** - Any translated word or sentence can be saved for future practice.

[My Words Page Guide](#_99t4nncueihy)



1. **Take Hebrew Quizzes** - Test and deepen your knowledge through interactive quizzes.

[Quizzes Page Guide](#_fwlq7yudb2w9)



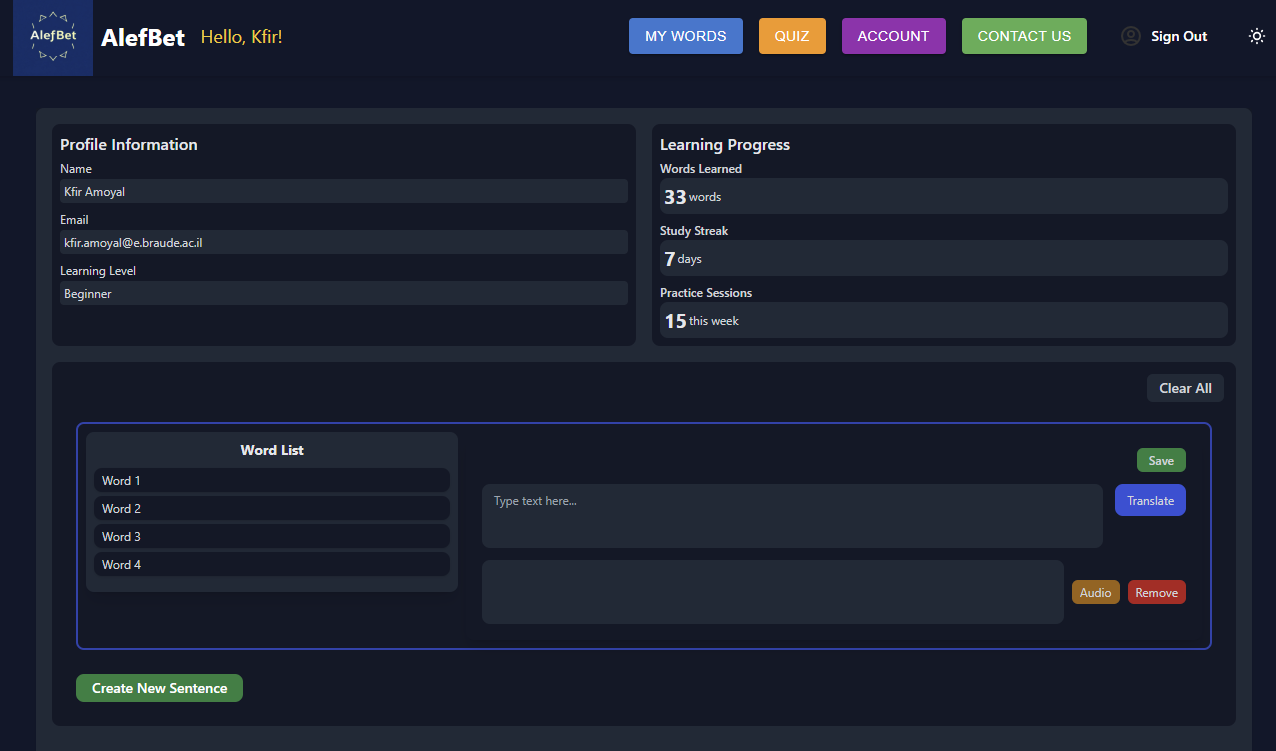
## **Pages Guide**

## Account Page.

After logging in, all features of the website will be available to you.

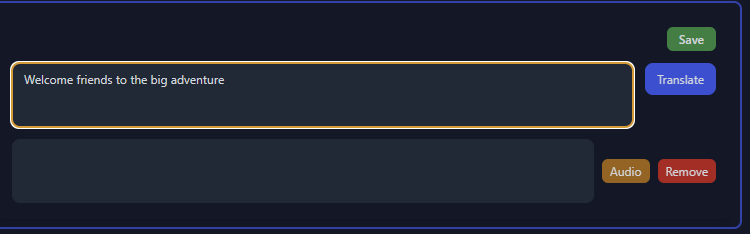
You will be redirected to the **Account** page.

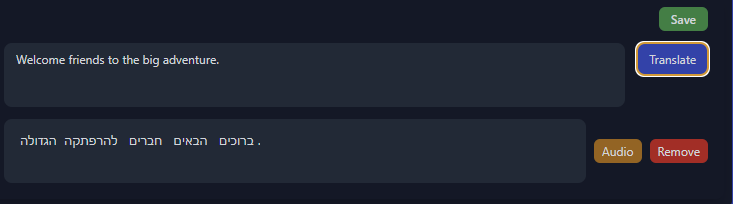
This is your personal area where you can track the number of words you have learned, your personal details, and your learning progress.



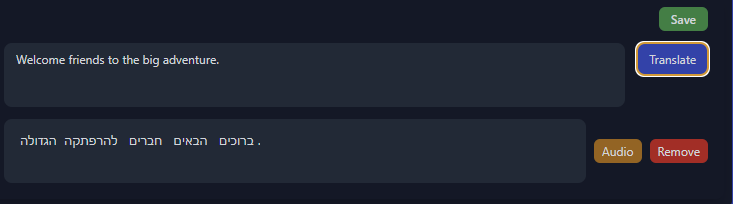
On this page, you can easily translate words and sentences from English to Hebrew.

Type an English sentence, then click “Translate” button.

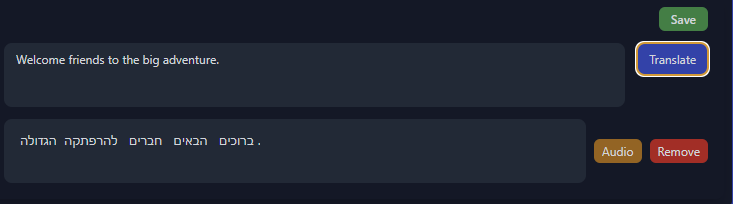




Now you can play the pronunciation by clicking “Audio” button

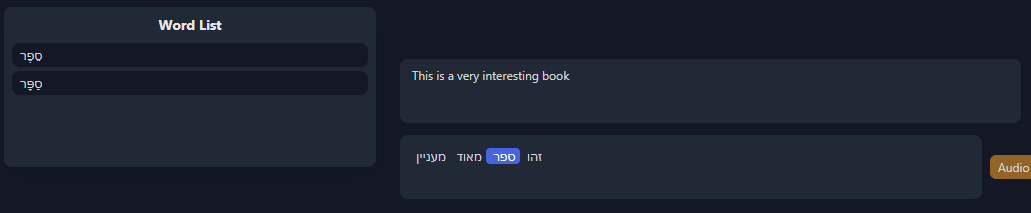


You can also save the sentence to the “My Words” page by clicking the “Save” button.

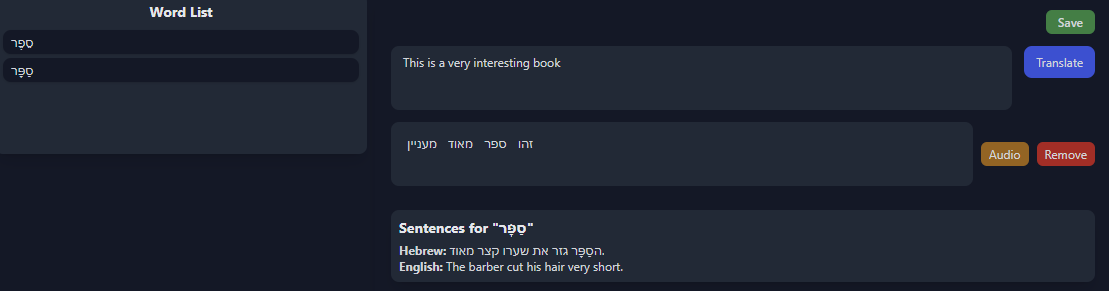


Some words have different meanings depending on their punctuation.

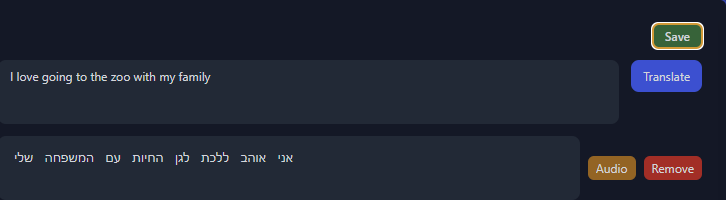
You can click words and see the different variations of this word:



Then you can click on the word from the ‘Word List’ and see an example that includes this word.

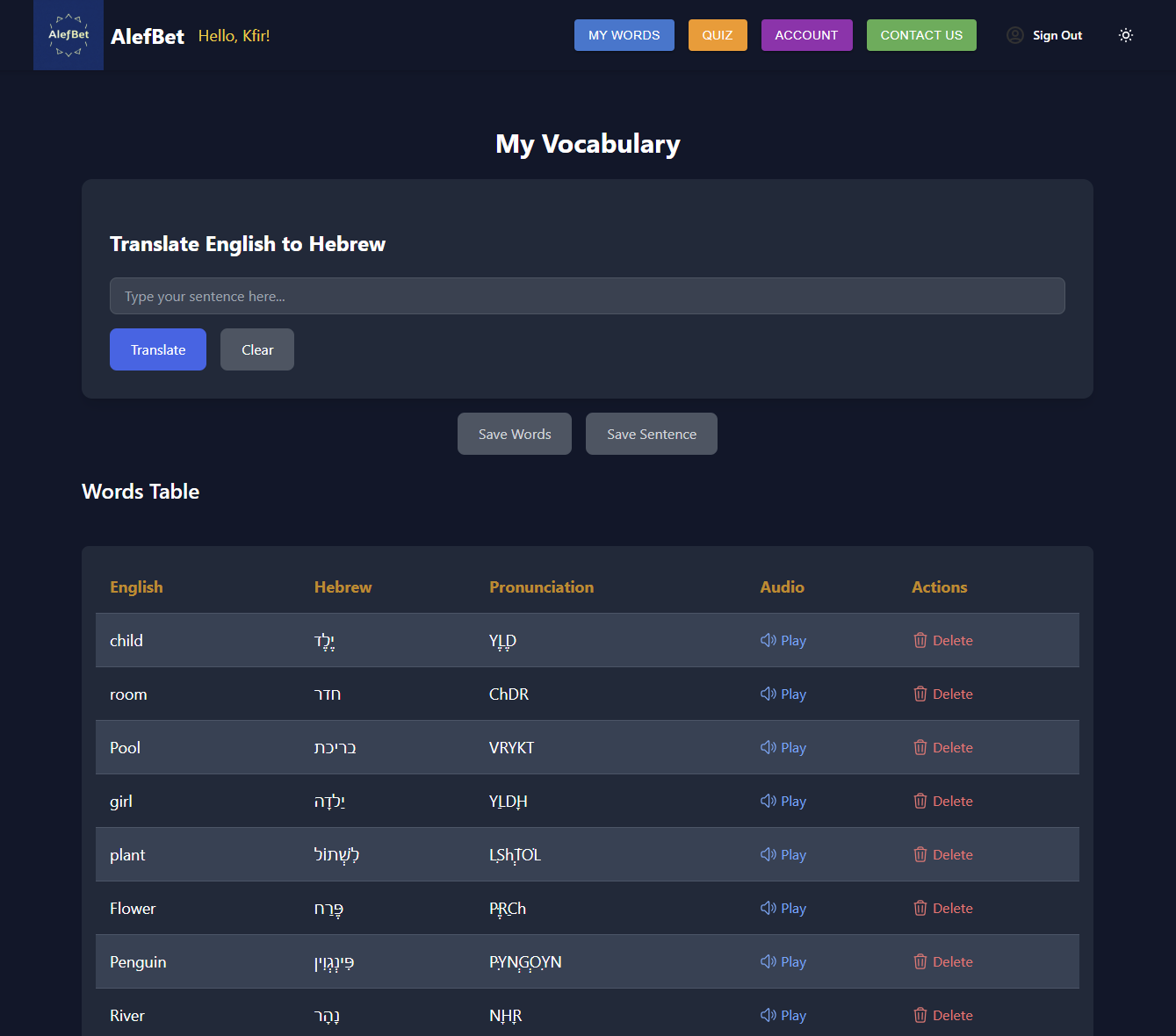


You can save any translated word or sentence to the “My Words” page by clicking the ‘Save’ button.



## My Words Page

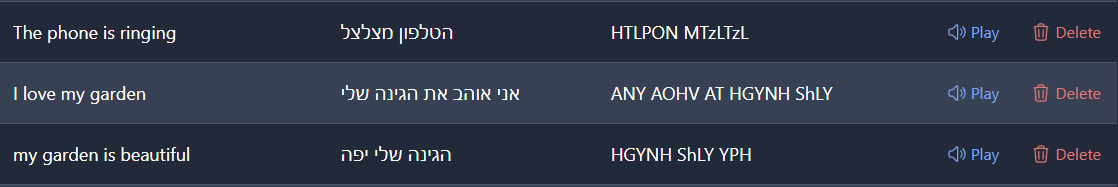
On the **My Words** page, you can track and review all the words saved for your account on the website.



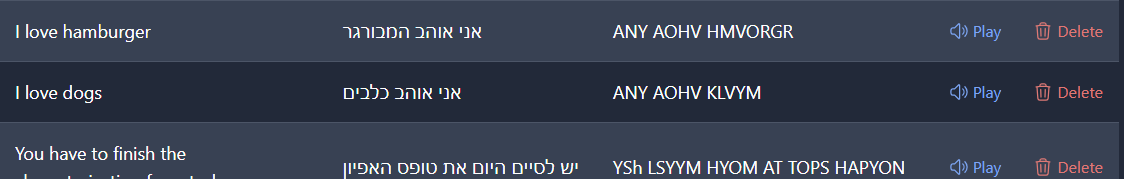
The saved words are divided into two tables: one for individual words and another for full sentences.



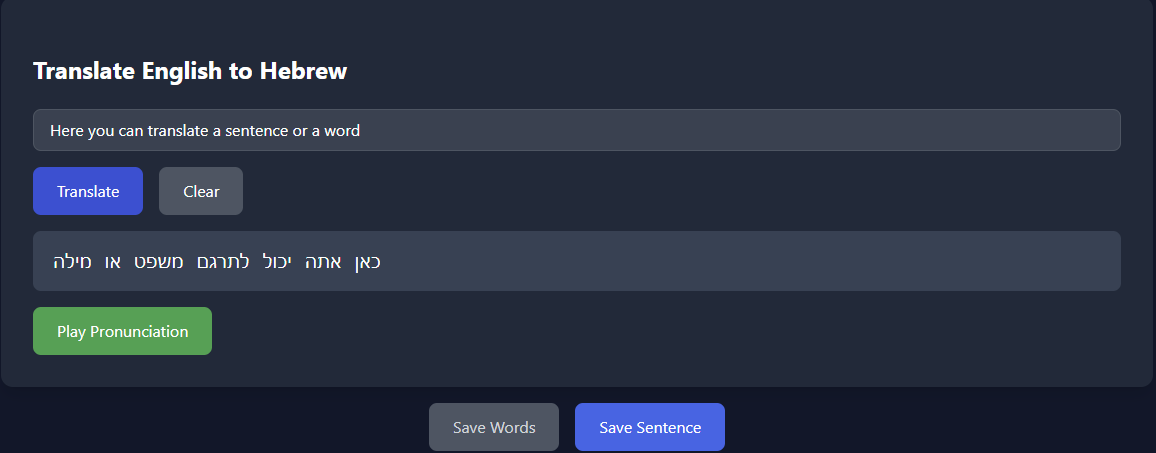
For each saved word or sentence, you can click the **'Play'** button to hear its pronunciation.



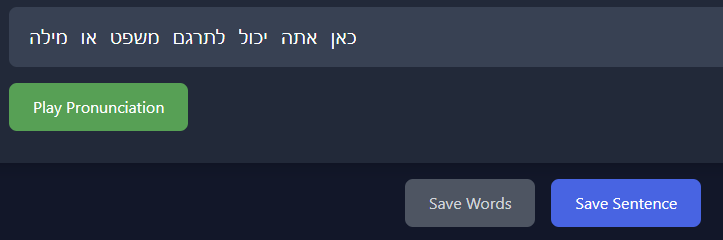
Additionally, you can remove a word or sentence from your saved words by clicking the **'Delete'** button.



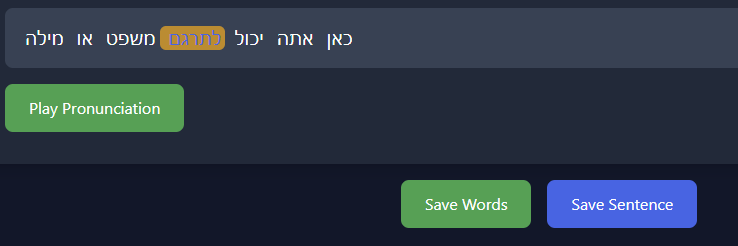
Moreover, you can translate and save words/sentences directly from this page using the translation section.



You can save the whole sentence by clicking the ‘Save Sentence’ button.

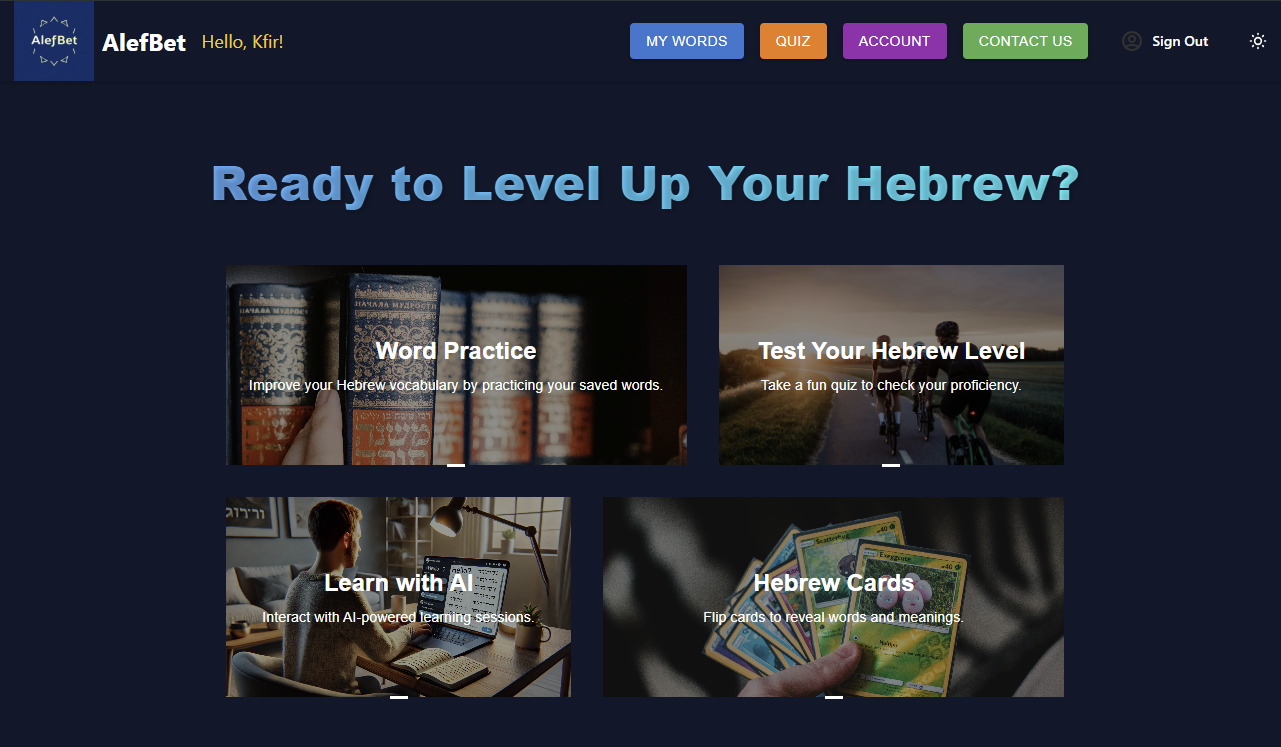


You can also choose one words from the sentence by clicking it and then click ‘Save Words’ button.



## Quiz Page

## **Quiz Page - Where the Fun Happens!**

The **Quiz** page is where you can enjoy interactive and engaging learning activities. There are multiple quiz features available for learners:

### 

### 

### 

### 

### 

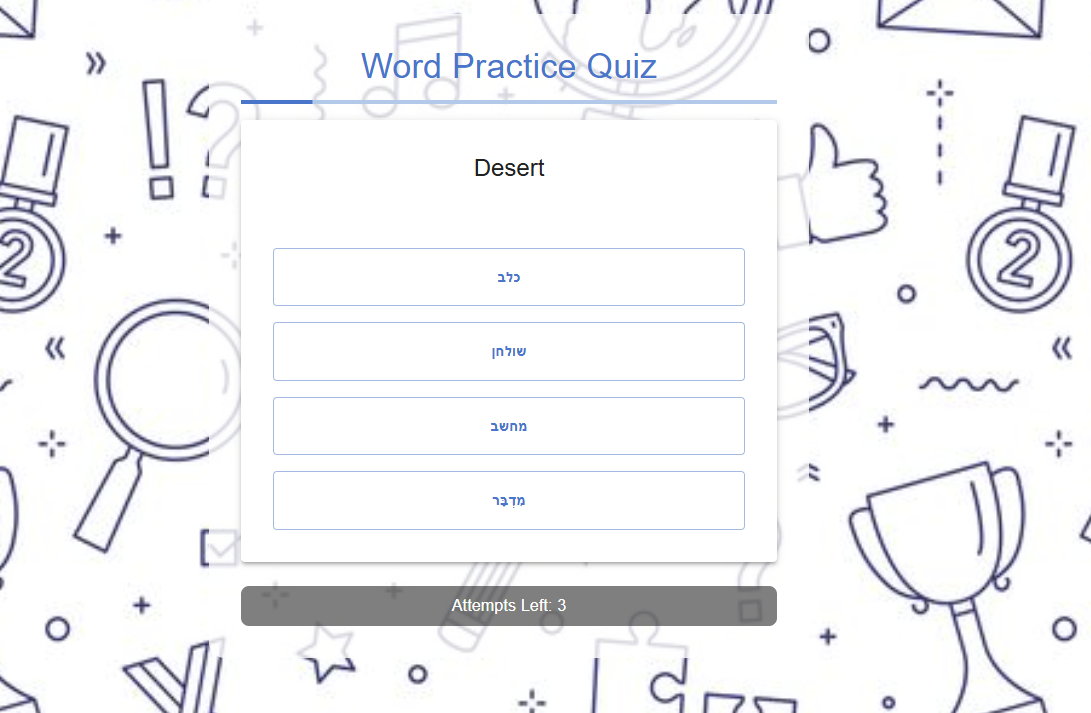
### 

### 

### 

### **Feature 1: Word Practice**

* This section tests you on the words you have learned from your saved words list.
* You will need to correctly answer 15 questions using words from your saved list.
* You have 3 attempts; if you get 3 answers wrong, you will be eliminated, but you can always restart.
* This is your opportunity to challenge your knowledge and reinforce your vocabulary in a fun and educational way.



### **Feature 2: Test Your Hebrew Level (Coming Soon)**

* This quiz is designed to determine your Hebrew proficiency level.
* You will answer various questions with different difficulty levels to assess your knowledge.



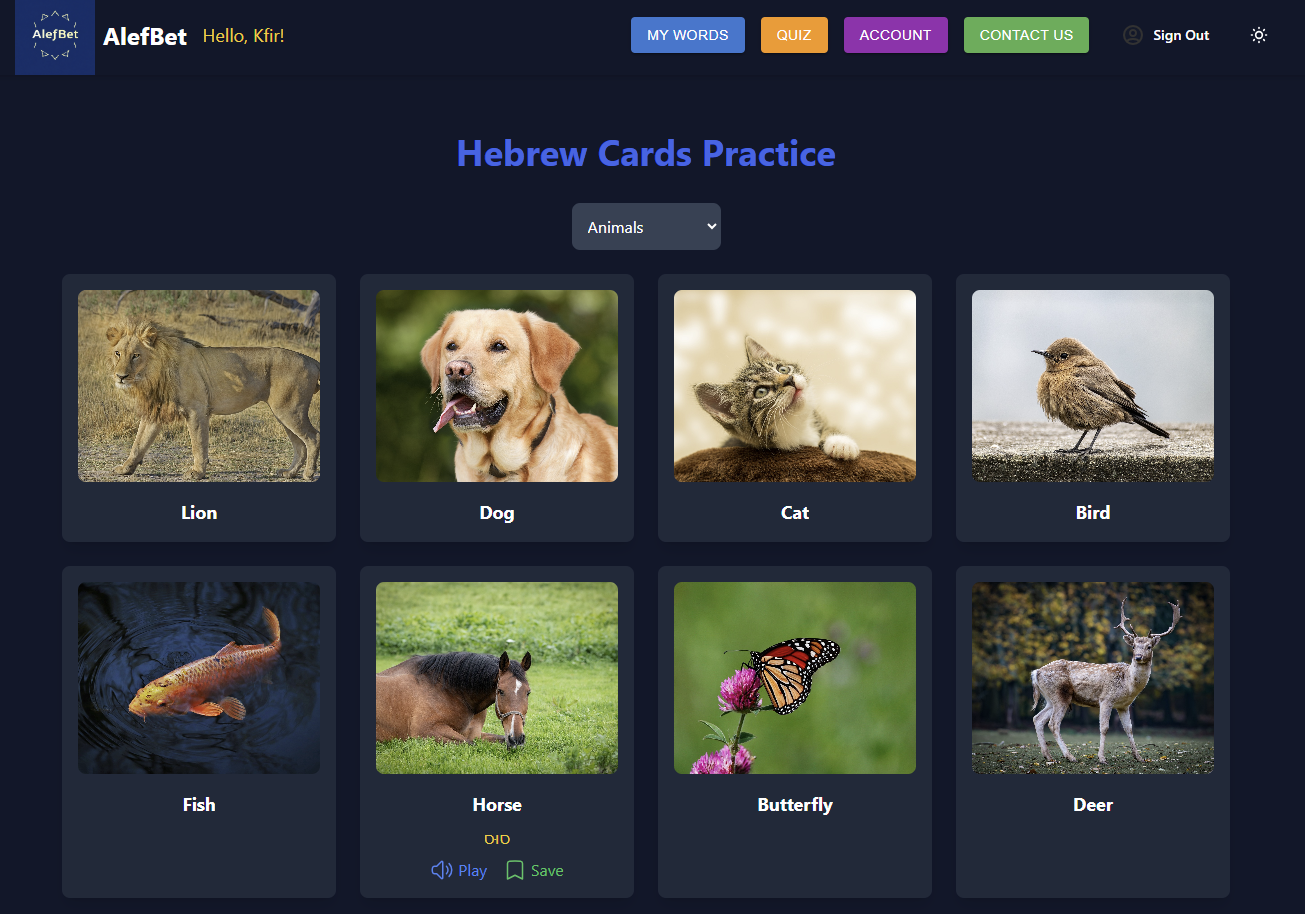
### **Feature 3: Learn with AI (Coming Soon)**

* This feature includes an AI-powered chatbot designed to help you learn Hebrew.
* You can engage in conversations about vocabulary, grammar, synonyms, and more.
* The best way to learn a language is by speaking it, and this tool will assist you in practicing Hebrew naturally.



### **Feature 4: Hebrew Cards**

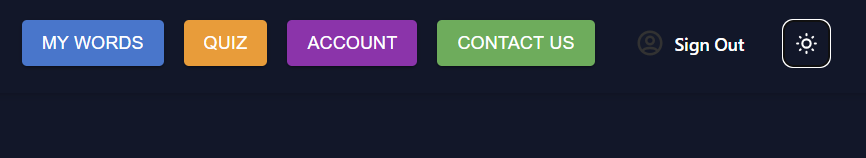
* In this section, you can memorize words from various categories in an interactive way.
* Flashcards with images will be displayed, and by clicking on an image, you will see the translation and pronunciation of the word.
* You can also save words from this section directly to your **My Words** list.
* Browse through different categories by clicking on the navigation buttons.





### **Dark Mode & Light Mode Toggle**

AlefBet offers a dark mode and light mode option for a comfortable learning experience. You can easily switch between the two modes by clicking the **theme toggle** button located in the navigation bar. This feature allows you to personalize the site's appearance according to your preference, making reading and navigation more convenient.

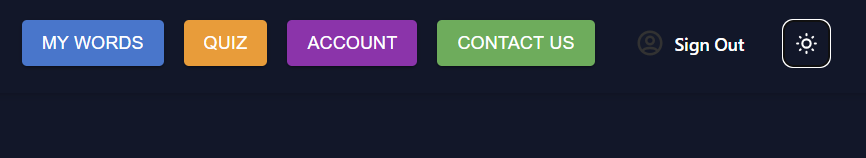


### **Sign Out & Data Preservation**

Users can log out of their accounts at any time using the **Sign Out** option available in the user menu. All your saved words, learning progress, and personal settings are securely stored, so you can resume learning exactly where you left off whenever you log back in.

The website includes several main pages:

* **Sign In/Sign Up Page** - Allows users to register a new account or log in to access full features.



## **Conclusion**

We hope this guide helps you make the most of **AlefBet** and enhances your Hebrew learning experience. With features like real-time translation, personalized vocabulary lists, interactive quizzes, and AI-powered learning, mastering Hebrew has never been more engaging and accessible.

Remember, consistency is key! Keep practicing, take advantage of the quizzes, and don’t hesitate to explore new words and sentences. Whether you are just starting or looking to refine your skills, **AlefBet** is here to support your journey.

If you have any questions, feedback, or need assistance, feel free to reach out through our [**Contact Page**](https://web-development-g15.vercel.app/contact).

Happy learning, and enjoy your journey with AlefBet! 🚀📚

