Science, Engineering & Education, 11, (1), 2026, 125-133

Smart Dictionary - Original Development of an Online Dictionary Powered by GPT-4o Mini

Nikolay Nikolaev[1], Georgi Hristov[1,2*]

[1]*Second English Language School „Thomas Jefferson", 26 Trayanova Vrata St., Sofia 1408, Strelbishte Residential Complex*
[2]*Sofia University "St. Kliment Ohridski", 15 Tsar Osvoboditel Blvd., Sofia 1504*

Received 18 September 2025, Accepted 20 January 2026
DOI: 10.59957/see.v11.i1.2026.17

ABSTRACT

This paper presents an original web application project called Smart Dictionary, designed to support vocabulary learning through interactive methods. The software combines functionalities of a personal dictionary, knowledge testing modules, flashcard systems, and automated file import/export. All dictionary features are enhanced by an integrated AI system. In addition, a modern user interface with intuitive navigation has been developed and proposed.

Keywords: smart dictionary, vocabulary learning, GPT-4o mini, flashcards, file import.

INTRODUCTION

The choice of topic was driven by the need to develop an effective and attractive tool for studying new foreign language vocabulary. The functionalities of the project are directly inspired by a survey conducted at the Second English Language School 'Thomas Jefferson' (Fig. 1).

Existing solutions are often complex, expensive, or inconvenient to use. The goal was to create a free, lightweight, and accessible application powered by artificial intelligence. The application allows for the creation of personalized dictionaries, the implementation of interactive learning methods using artificial intelligence. It features an intuitive interface, and support for multilingualism.

EXPERIMENTAL

The following Tech Stack is used to implement the web page:

Frontend: HTML5, CSS3, JavaScript (ES6+); Authentication: Google OAuth 2.0 [1]; Storage: LocalStorage, SessionStorage; Additional libraries: JWT Decode [2]; Artificial Intelligence: GPT-4o mini [3].

HTML5, CSS3 and JavaScript (ES6+) are responsible for the visualization of the web page and its main functions. User authentication occurs via Google OAuth 2.0 [1]. A secure and intuitive way to store data, log in and out of a profile, via Google Accounts. For data storage itself, LocalStorage and SessionStorage are used,

**Correspondence to: Georgi Hristov, Second English Language School „Thomas Jefferson", Trayanova Vrata St., Sofia 1408, Strelbishte Residential Complex, and Sofia University "St. Kliment Ohridski", 15 Tsar Osvoboditel Blvd., Sofia 1504, e-mail: georgi.hristov@2els.com*

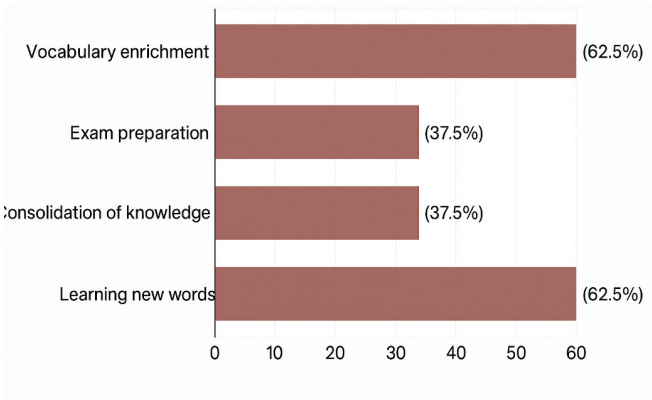

Fig. 1. Functionality survey.

which allow for the permanent preservation of sessions between different browser instances. JWT Decode is needed for Error Handling and Debugging (Fig. 2) [2].

Client architecture (SPA)

- Single Page Application (SPA).
- Fast navigation between screens without reloading.
- Modular structure.
- Logic encapsulation, avoiding global variables, code reuse.
- Advantages: organized code, easy testing, scalability.
- Component-based design.
- Adaptability through container queries (adapt according to the container, not the entire page).

Security

Authentication

- Google OAuth 2.0 for fast and secure login [1].
- Protection against XSS:
- Input data validation, output encoding, Content Security Policy [4].

Additional measures

- HTTP Security Headers, secure cookies (Secure, HttpOnly, SameSite), protection against CSRF attacks.

AI integration

- Model used:GPT-4o mini [3].

- *Functionalities*

- Generation of interactive tests with contextually similar but fundamentally different answers.
- Chat system for User-to-AI communication which enables the AI to perform different tasks inside the app.
- Text File Scraping (.TXT, .RTF, .DOCX, .DOC, .MD, and .HTML and .XML).

Integration with GPT-4o mini provides AI-powered tests, file import scraping, and an AI chat assistant [3]. A. AI Tests: The model generates contextually relevant multiple-choice questions from dictionary entries.

File Import: The platform automatically extracts and structures text content from uploaded

```
function handleCredentialResponse(response) {
    try {
        const data = jwt_decode(response.credential);
        console.log('Google Sign-in Data:', data);
        sessionStorage.setItem('user', JSON.stringify(data));
        // Remove the dictionary loading from here since it will be loaded on app.html
        window.location.href = 'app.html';
    } catch (error) {
        console.error('Sign-in Error:', error);
        showMessage('Грешка при влизане', 'red');
    }
}
```

Fig. 2. Error handling for the sign-in logic.

files. The AI scrapes and organizes the data so the frontend can process and import it seamlessly. The organized words, definitions and Parts of Speech get directly imported into the dictionary. and C. AI Chat Assistant: Users can interact with an intelligent assistant capable of explanations, launching tests, and adapting to prior context.

Development

Integration with GPT-4o mini: AI Tests, File Scraping, and AI Chat Assistant

Generating AI Tests

• GPT-4o mini uses advanced natural language processing (NLP) and machine learning algorithms to analyze the content of dictionary entries (words and definitions) [5].

• Through REST API requests to OpenAI, data from the existing dictionary is sent for processing, and the AI module generates questions that contain one correct answer and several (usually 3) randomly selected incorrect options.

• Configuration parameters for the difficulty level and question type can be specified as arguments in the API request, providing customized tests.

• The received response (formatted as JSON), containing the questions and the shuffled options, is used by the user interface for visual presentation with animated transitions and a progress indicator.

Scraping text files

The AI is used to automatically extract and structure text content from Text documents.

- The function takes a text file uploaded by the user containing a list of words.
- It reads the content of the file and prepares instructions for an AI to clean it up (Fig. 3).

The AI is asked to:

- Put each word in a neat format: word, definition, part of speech.
- Fill in any missing definitions or grammatical types.
- Translate parts of speech into English if needed.
- Keep the original language of the words and definitions.

The AI processes the text and returns a clean, organized list.

```
@app.post("/structure-words")
async def structure_words(file: UploadFile = File(...)):
    contents = await file.read()
    raw_text = contents.decode("utf-8")

    # Construct the prompt for the AI
    prompt = (
    "You will receive a list of unstructured or partially structured word entries, written in English or Bulgarian.\n"
    "Your task is to convert them into a clean CSV-like format with the structure:\n"
    "word,definition,part of speech\n\n"
    "Rules:\n"
    "- If the definition is missing, guess a reasonable one.\n"
    "- If the part of speech is missing, infer it.\n"
    "- If the part of speech is written in Bulgarian, translate it to English.\n"
    "- Keep the output clean: one word per line, no headers, no numbering.\n"
    "- Use lowercase for the word, capitalize the definition.\n"
    "- Preserve the original language of the word and definition (e.g., if the word is in Bulgarian, keep the definition in Bulgarian).\n"
    "- Always output the part of speech in English: noun, verb, adjective, or adverb.\n"
    "- Example (English): dog,A domesticated animal,noun\n"
    "- Example (Bulgarian): куче,Домашно животно,noun\n\n"
    "Here is the input:\n"
    + raw_text
)
```

Fig. 3. The prompt that the AI follows.

- The function then sends this structured word list back to the user as plain text.
- If anything goes wrong, it returns an error message instead.

AI chat assistant

- The chat module is enriched with functionality based on AI, the which allows users to interact with an intelligent virtual assistant.
- When entering a request in the chat window, the message is sent to the OpenAI API, where algorithms for semantic analysis and contextual understanding are used.
- The response generated by GPT 4o mini includes explanations for questions and can perform actions within the application (for example, run tests,
- All communication is done over secure HTTPS connections using API keys and input validation, which ensures security and traceability.
- Additionally, the AI assistant can remember the context of previous chat messages, providing more personalized and adaptive help in real time.

Technical implementation and security measures

• API communication:

- The integration uses asynchronous HTTP requests (with fetch or Axios) to the OpenAI API, which send user data and receive responses in JSON format [6].

- All connections are encrypted via HTTPS, and access to the API is protected by unique API keys and restrictions on the scope of requests.

• Error logging and handling:

- Every interaction with the AI Model is logged for troubleshooting and performance analysis purposes.

- Errors returned by API requests are caught with appropriate handling mechanisms and notified to users via messages in the interface.

Conclusion on the GPT-4o mini integration

The integration with GPT-4o mini significantly contributes to expanding the functionality of the Smart Dictionary by:

- Automatic generation of personalized AI tests that adapt questions according to the content of the dictionary.

- Efficient extraction and structuring of data from text files, which facilitates rapid updating of the dictionary.

- Providing an interactive AI chat assistant that offers help, guidance and performs tasks in real time.

Overview of the components

HTML Structure (index.html & app.html)

Upon launching the application (index.html), the user is presented with a clean and minimalistic user interface. The main elements include:

- Navigation bar (navbar) with the logo and navigation buttons.

- "Hero" section, which presents the functionalities through attractive text, animated elements and icons.

- Authentication container, which also contains a Google Sign-In button and an alternative for guest access.

While app.html implements the main Fig.6 view-dictionary section logic of the dictionary, tests, flashcards and import. The page uses separate "view" sections, each of which is responsible for a specific functionality (for example, view-dictionary, view-flashcards, view-quiz, view-import) (Fig. 6). Using JavaScript, the content is switched according to the user's choice [7].

CSS Styles (style.css)

The styles are defined using CSS variables, which makes it easier to maintain the dark theme and color of the applications. The main features are:

- Gradient lines for backgrounds (for example - var(--bg-gradient-1) and var(--bg-gradient-2)).

- Smooth transitions and animations that improve the user experience.

- Adaptability to different devices through media queries (for example, templates for mobile devices).

- A separate section for flashcard styles, which get a modern look with highlighting shadows, rounded corners and animated transitions when turning the cards.

Specific classes such as .flashcard-modal, .flashcard-container and .flashcard-inner have been defined for flashcards, which are responsible for:

- Displaying a modal window with a blurred background layer.

- Centering the content and design with clean, modern borders and shadows.

- Detailed styles for the front and back of the flashcards - use linear gradients to create depth and visual interactivity.

JavaScript functionality (script.js)

- Authentication

- Authenticates the user via sessionStorage.

- Uses jwt-decode to decode Google Sign-In tokens. If successful, the user is redirected to app.html with a custom dictionary.

- Dictionary management

- Add words with validation for length, XSS protection and presence.

- Local storage: LocalStorage (authenticated users) or sessionStorage (guests).

- Dynamic visualization and ability to delete words (updateWordList).

- Test system

- When pressing the “Start test” button, the generator checks if there are at least 4 words in the dictionary (Fig. 4).

- Random questions are generated, where one correct answer is selected for each word, the rest are supplemented with incorrect, similar words

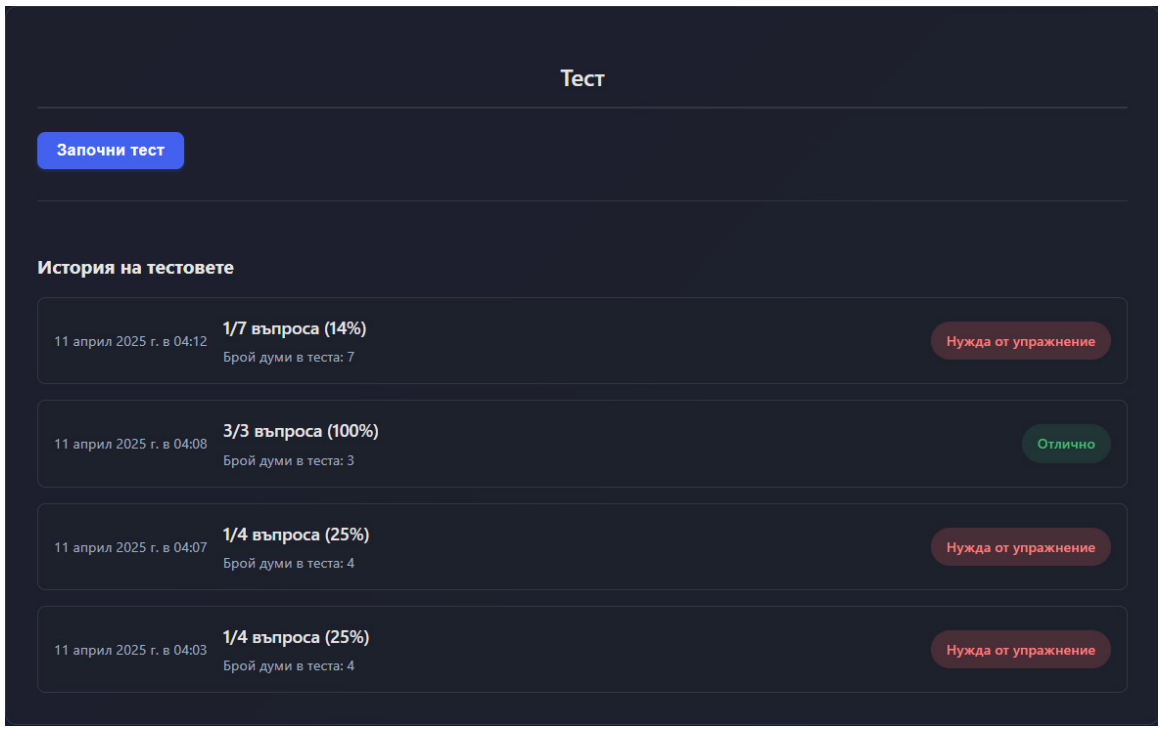

Fig. 4. Test section.

that could be used in the context, powered by AI

- The answers are shuffled and visualized with animated transitions, while monitoring the progress bar.

- After submitting the test, a score (in percentages) is visualized and animated signals for correct and incorrect answers are displayed (Fig. 4).

- Flash cards

- Modal dialog with scrambled words. Functionalities: card flip, navigation (previous/next), progress indicator.

- Animations with CSS transformations and JavaScript for state management.

- File import

- With the click of a button the user is able to import an unstructured list of words that automatically gets ran through AI and put into the dictionary

- Based on predefined options (paragraph splitting, ignoring short texts), the data is extracted and imported into the dictionary.

- This module makes it possible to automatically add multiple words from a single document.

- Chat functionality

- The chat interface allows communication with an AI assistant - application management via chat.

- The chat has a responsive design, uses a separate modal overlay solution and supports automatic adjustable height of the text area.

- Views management

- Dividing the interface into independent views (dictionary, test, flashcards, etc.). Transition between them via navigation buttons with updating CSS classes and saving the state in sessionStorage.

- Error handling and accessibility

- Errors are handled via status messages in `.status-container`.

- Support for aria-labels for forms and buttons, compliance with accessibility standards (a11y).

Technical understanding

Initialization

- When loading index.html, the user sees a clean presentation of application information and options for authentication or guest access (Fig. 6).

- When clicking on one of the buttons, event listeners are called, which handle navigation and

```
function generateQuestion() {
    const quizContainer = document.getElementById('quiz-container');
    const generateQuizButton = document.getElementById('generate-quiz');

    if (words.length < 4) {
        showMessage('Добавете поне 4 думи в речника, за да започнете тест!', 'red');
        return;
    }

    setLoading(generateQuizButton, true);

    try {
        // Create modal container if it doesn't exist
        let modalOverlay = document.querySelector('.quiz-modal-overlay');
        if (!modalOverlay) {
            modalOverlay = document.createElement('div');
            modalOverlay.className = 'quiz-modal-overlay';
            document.body.appendChild(modalOverlay);
        }

        // Prepare quiz data
        const quizQuestions = [];
        const usedWords = new Set();
        const totalQuestions = Math.min(10, words.length);
```

Fig. 5. Generating a template for the test.

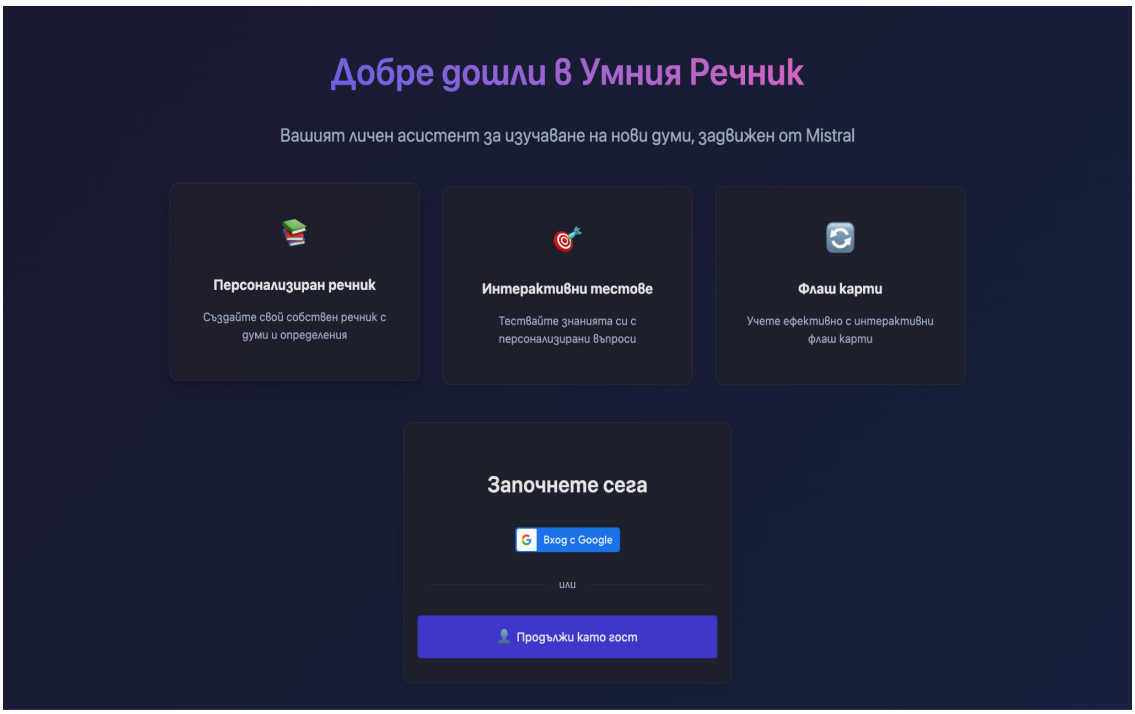

Fig. 6. Home page.

display of the corresponding view.

- If the user is authenticated, sessionStorage contains his data and the application automatically loads the dictionary from local storage.

Flow when adding and managing the dictionary

- The form for adding words contains input for the words and text area for the definitions.

- The validateInput function performs the primary validation - it checks whether data has been entered, whether its length corresponds to the specified constants, and special characters are removed through a simple definition of sanitize.

- After validation, the words are added to the global array "words", which is then saved in LocalStorage (or by a specific key for authenticated users). - updateWordList dynamically updates the DOM - creates new li elements for each word with the corresponding styles and the ability to delete via a button (Fig. 7).

Flash cards

- At startup, the "words" array is shuffled (via the shuffleArray function) to ensure randomness in the order (Fig. 8).

- A modal window (flashcard modal) is opened, which contains a container with a header, progress indicator, flash card content and navigation buttons.

- JavaScript event listeners handle clicks, and when the card is clicked, it is "flipped" by adding/removing the .flipped class.

- Navigation buttons allow switching to the next or previous card

RESULTS AND DISCUSSION

The Smart Dictionary has been developed as a prototype informed by informal conversations with students and teachers, which provided valuable insights into desired functionalities and

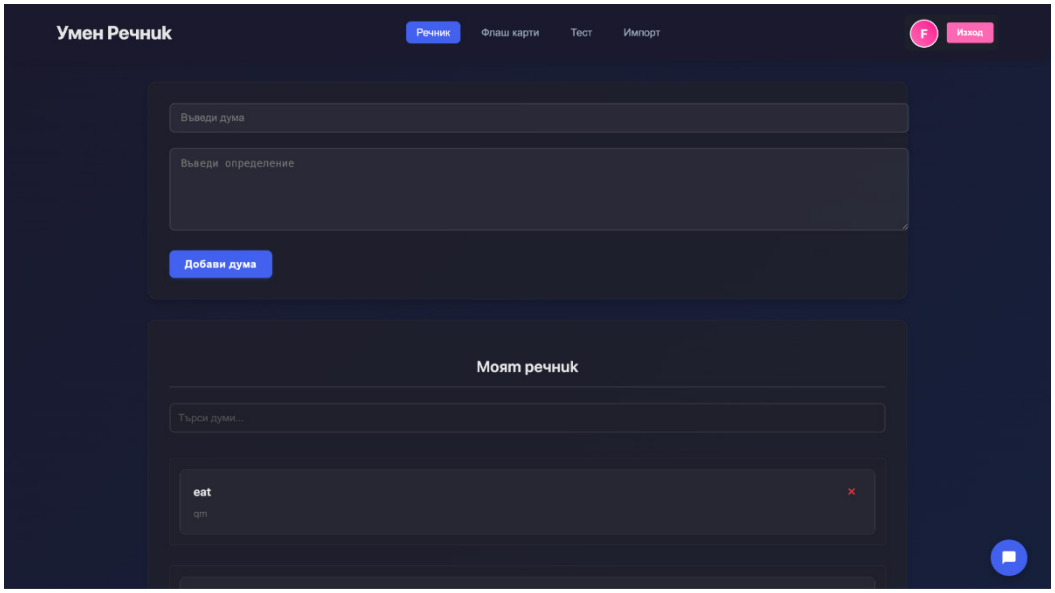

Fig. 7. Dictionary home page.

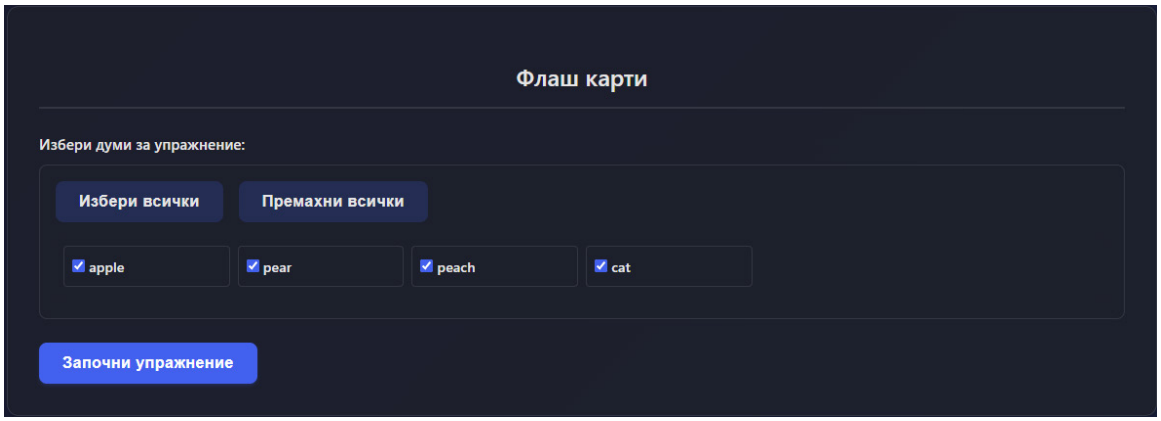

Fig. 8. Flashcards page.

ease of use. At this stage, the application has not yet been formally deployed in classroom settings. Instead, it will be offered to teachers and students in upcoming pilot activities in order to evaluate its effectiveness, usability, and pedagogical value.

Preliminary feedback suggests strong potential for integration into language learning, particularly through features such as AI-generated quizzes, interactive flashcards, and automated file import. However, systematic evaluation is still required to confirm its impact on vocabulary retention and learner motivation. The forthcoming collaboration with educators and learners will therefore play a crucial role in refining the tool and guiding its future development.

CONCLUSIONS

The Smart Dictionary project illustrates the potential of combining artificial intelligence with interactive learning methodologies to support vocabulary acquisition. By integrating features such as AI-generated quizzes, flashcards, file import, and chat-based assistance, the prototype

demonstrates both technical feasibility and pedagogical promise.

Although the system has not yet been formally introduced into classroom practice, its design was informed by informal discussions with teachers and students, ensuring that the functionalities reflect real educational needs. The forthcoming pilot implementation will provide valuable evidence on its effectiveness in authentic learning environments.

The implemented software product combines the following functionalities: adding, editing, and deleting words and their definitions. The learning process is supported by the implementation of artificial intelligence, which is used to generate tests with contextually relevant answers and a flash card system. Additional functionality is provided for importing/exporting dictionaries from Google Docs documents, which creates rich opportunities for the practical use of the software by students.

The technical implementation is distinguished by its modular organization, which ensures easy maintenance and the possibility of future expansion. The integration of Google OAuth 2.0 provides reliable authentication, and the application of modern approaches to error handling and input data validation ensures stable application performance.

In addition, in order to create a pleasant user experience, the dictionary features a modern user interface with adaptive design and smooth animations.

REFERENCES

1. Using OAuth 2.0 to Access Google APIs, retrieved September 09 2025 from https://developers.google.com/identity/protocols/oauth2
2. D. DeGroff, Online JWT Decoder, retrieved September 09 2025 from https://fusionauth.io/dev-tools/jwt-decoder.
3. Welcome to GPT-4o mini Documentation, retrieved September 09 2025 from https://cookbook.openai.com/examples/gpt4o/introduction_to_gpt4o
4. Content Security Policy Reference, retrieved September 09 2025 from https://content-security-policy.com/.
5. What are container queries?, retrieved September 09 2025 from https://devblondie.com/container-queries/
6. Resources for Developers by Developers, retrieved September 09 2025 from https://developer.mozilla.org/en-US/.
7. HTML Living Standard, retrieved September 09 2025 from https://html.spec.whatwg.org/multipage/webstorage.html.