Wort Monster Web App Documentation

Contents

1 Overview

Wort Monster is an innovative web application designed to help users master complex German compound words. The application provides an interactive interface for building, managing, and learning compound words, along with their sub-words and translations. Integrated tools like Google OAuth for authentication, Supabase for database management, and browser-based Text-to-Speech (TTS) functionality make the learning process seamless and engaging.

2 Features

- **User Authentication:**
  - Secure login using Google OAuth, with user data stored and synchronized through Supabase.
  - Continuous authentication monitoring via Firebase to maintain session state.
- **Compound Word Management:**
  - Create compound words with sub-words, translations, and gender attributes.
  - Edit existing words or delete them as needed.
  - Fetch words dynamically from a centralized database for real-time updates.
  - Display words for interactive learning.
- **File Management:**
  - Upload JSON files with compound word data.
  - Create new JSON files with user-defined names.
  - Overwrite existing files with updated data while ensuring no duplicates.
- **Word Set Management:**
  - View, create, edit, and delete word sets.
  - Add new words to a set, dynamically link words, and synchronize sets with the database.
  - Search public word sets using advanced filters such as set name, words, or creator.
- **Search and Highlighting:**
  - Search public word sets with filters for ‘name‘, ‘word‘, and ‘creator‘.
  - Highlight sub-words dynamically in compound words during hover or interaction.
- **Dynamic Sub-Word Highlighting:**
  - Automatically highlights changes between sub-words and their original forms.
  - Visualizes added parts in green and removed parts in red strikethrough.

  - Offers learners contextual clarity about word transformations.

- **Modal for Sub-Word Details:**
  - Provides detailed views for individual sub-words.
  - Displays translations, original forms, and gender attributes.

- **Text-to-Speech Functionality:**
  - Pronounce compound words and sub-words using the browser's 'SpeechSynthesis' API.
  - Users can adjust playback speed dynamically to suit their learning pace.
  - Dynamic speech speed control via an interactive slider.

- **User Context Management:**
  - Use 'UserContext' for centralized user state management.
  - Integrates smoothly with the authentication system to propagate user information across components.

3 Setup Instructions

3.1 Clone the Repository

To set up the project locally, follow these steps:

1. Clone the repository:

```
git clone git@git.ucsc.edu:tvu8/wort-monster.git
cd wort-monster
```

3.2 Frontend Setup

1. Create a React app for the project:

```
npx create-react-app german-compound-words
cd german-compound-words
```

2. Install Tailwind CSS:

```
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

3. Configure Tailwind in `tailwind.config.js`:

```
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

4. Update the file `src/index.css`:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

5. Install Firebase:

```
npm install firebase
```

6. Set up a `.env` file and configure environment variables:

```
REACT_APP_FIREBASE_API_KEY=<your_api_key>
REACT_APP_FIREBASE_AUTH_DOMAIN=<your_auth_domain>
REACT_APP_FIREBASE_PROJECT_ID=<your_project_id>
REACT_APP_FIREBASE_STORAGE_BUCKET=<your_storage_bucket>
REACT_APP_FIREBASE_MESSAGING_SENDER_ID=<your_sender_id>
REACT_APP_FIREBASE_APP_ID=<your_app_id>
REACT_APP_FIREBASE_MEASUREMENT_ID=<your_measurement_id>
```

7. Start the frontend development server:

```
npm start
```

3.3 Backend Setup

1. Navigate to the backend directory:

```
cd backend
```

2. Install dependencies:

```
npm install
```

3. Configure Supabase in `.env`:

```
SUPABASE_URL=<your_supabase_url>
SUPABASE_ANON_KEY=<your_supabase_anon_key>
```

4. Start the backend server:

```
npm start
```

5. Access the server at `http://localhost:5001`.

3.4 Branching and Version Control

1. Ensure you are on the `main` branch:

```
git checkout main
```

2. Create a new branch:

```
git checkout -b <your_branch_here>
```

3. Make your changes and commit:

```
git add .
git commit -m "<your_message>"
```

4. Push your branch to the repository:

```
git push --set-upstream origin <your_branch_here>
```

4 API Reference

4.1 Word Routes

Get All Words

- **Endpoint**: `GET /getall`
- **Response**: Returns all compound words with non-null sub-words.

Create Word

- **Endpoint**: `POST /create`
- **Request Body**:

```
{
  "compoundWord": "Straßenbahn",
  "translation": "Tram",
  "subWords": [
    {
      "word": "Straßen",
      "original": "Straße",
      "stays": false,
      "translation": "Street",
      "gender": "die"
    },
    {
      "word": "bahn",
      "stays": true,
      "translation": "Rail",
      "gender": "die"
    }
  ]
}
```

- **Response**: Returns the newly created word object with a unique ID.

Update Word

- **Endpoint**: `PUT /:id`
- **Request Body**:

```
{
  "compoundWord": "Straßenbahn",
  "translation": "Tram"
}
```

- **Response**: Updated word object.

Delete Word

- **Endpoint**: `DELETE /:id`
- **Response**: Confirmation of deletion.

5 Utility Functions

User Context

- **useUser():** Hook to access the ‘UserContext‘ for managing and retrieving user data.
- **UserProvider:** Wraps components with a ‘UserContext.Provider‘, allowing access to the ‘user‘ and ‘setUser‘ methods for managing user authentication state.

Authentication Flow

- **onAuthStateChanged():** Firebase method to monitor and update the user’s authentication state.
- **validateUser():** Validates and synchronizes the user context with the backend (‘userCheck‘).

6 Database Schema

6.1 Dictionary Table

Column	Type	Description
id	UUID	Unique identifier for the word.
word	String	Compound word.
sub_words	String	Sub-words in CSV format.
definition	String	Translation of the compound word.
gender	Int	Gender of the word (`0=der, 1=die, 2=das`).
pos	Int	Part of speech (future use).

6.2 Users Table

Column	Type	Description
uid	String	Firebase user ID.
name	String	User's name.
email	String	User's email.
picture	String	Profile picture URL.

7 Database Functions

Word Functions

- **deleteWord(wordId):** Deletes a word from the ‘dictionary‘ table.
- **getAllWords():** Fetches all words from the ‘dictionary‘ table.
- **getCompoundWords(word_ids):** Fetches compound words by their IDs.
- **getWord(word_id):** Fetches a single word by ID.
- **insertWord(...):** Inserts a new word into the ‘dictionary‘ table.
- **upsertWord(...):** Inserts or updates a word, ensuring uniqueness.

Word Set Functions

- **getAllSets():** Fetches all word sets from the ‘word_sets‘ table.
- **getSetById(set_id):** Fetches a word set by ID.
- **insertWordSet(name, user_id):** Inserts a new word set.
- **addWordToSet(...):** Adds a word to a set dynamically.

8 Known Issues

- **Text-to-Speech Compatibility:** May not work in some browsers without clearing cache.
- **Token Expiry:** Firebase tokens must be refreshed periodically.
- **Sub-Word Edge Cases:** Suffix logic might not handle certain combinations correctly.
- **Browser Limitations:** Speech synthesis may vary across different browser implementations.

9 Product Backlog

This section outlines potential future features and improvements for the Wort Monster web application.

9.1 Feature Enhancements

- **Cross-Set Word Saving:** Implement a button functionality that allows users to save words to other word sets they own, enhancing word organization and learning flexibility.
- **Deployment Strategy:** Develop a comprehensive deployment plan to make the application accessible to users, including considerations for:
  - Cloud hosting options
  - Continuous integration and deployment (CI/CD) pipeline
  - Performance optimization
  - Domain configuration

9.2 Learning Activities

- **Sub-Word Reconstruction Activity:** Create an interactive exercise where students:
  - Are presented with separated sub-words
  - Must reconstruct the full compound word
  - Receive immediate feedback on their composition
- **Compound Word Generation:** Develop an activity where:
  - Students are given sub-words
  - They must write down the complete compound word
  - The system validates their input against the correct compound word
- **Comprehensive Testing Module:** Design a robust testing framework including:

  - Unit tests for individual components
  - Integration tests for complex interactions
  - User experience (UX) testing
  - Performance and accessibility testing

9.3 Technical Improvements

- Enhance browser compatibility for Text-to-Speech functionality
- Implement more robust sub-word parsing and transformation logic
- Develop advanced caching strategies for improved performance