Glossary App

Project Overview

This project is a web-based glossary application designed to help clients manage and organize individual glossary terms and their corresponding definitions. Backend implementation of **SQLite database** ensures that it has a persistent data storage and retrieval functionality, and that it successfully talks to the **web API** with **ASP.NET Core**, written in **C#**. The Frontend written in **JavaScript (JS)** provides a simple, clean, user-friendly interface for adding, searching, editing, deleting, and viewing an alphabetically sorted list of the glossary terms and definitions.

Features & Functionality

- **Main Menu:** Users are presented with a main menu that allows them to navigate easily to different sections of the application, including adding, searching, editing and deleting terms, ultimately viewing the complete glossary.
- **Adding Terms:** Users can add new glossary terms and their definitions using a form. Input validation ensures that both the term and definition fields are not empty and contain only alphabetic characters. Duplicate terms are also checked before adding.
- **Searching for Terms:** Users can search for specific terms within the glossary. If a term is found, its definition is displayed, alongside with the functionality to Edit and/or Delete term. If not found, an alert is shown.
- **Editing Terms:** Users can edit existing glossary terms and their definitions. When editing, the current term and definition are populated in the edit form fields. Input validation ensures that the updated term is unique, not empty, and doesn't contain any numbers/special characters.
- **Deleting Terms:** Users can delete unwanted, existing glossary terms. Upon deletion, a success message is shown, and the glossary table is updated to reflect the changes.
- **Viewing Glossary:** Users can print and visualize the complete glossary dictionary in alphabetical order, showing each unique term, with its corresponding definition.

Main Files and Components Created

1. GlossaryApp\GlossaryWebApi\Program.cs: this serves as the entry point for the ASP.NET Core web API application. Written in C#, it configures the web host and sets up various components required for the application to run, including middleware, services, and controllers.
2. GlossaryApp\GlossaryWebApi\Controllers\GlossaryItemsController.cs: is our one and only controller class (like we mentioned above) that is responsible for handling HTTP requests related to glossary items. Also written in C#, it defines various action methods to perform CRUD (Create, Read, Update, Delete) operations on glossary items, such as adding new items (terms, definitions), retrieving all items, updating existing items, and deleting items.
3. GlossaryApp\GlossaryWebApi\Models\GlossaryContext.cs: this is where our backend database context for interacting with the SQLite database lives. Written in C#, the file contains the Entity Framework DbContext class, which represents the database context

for interacting with the SQLite database. It also defines the glossary items table schema and provides us methods for querying and manipulating the glossary data.

4. GlossaryApp\GlossaryWebApi\Models\GlossaryItem.cs: this is a definition of the GlossaryItem class representing glossary items. This C# file defines the GlossaryItem class, which represents a single glossary item entity. It therefore contains the properties for the term and definition of the glossary item.

5. GlossaryApp\GlossaryWebApi\Models\GlossaryItemDTO.cs: extremely like the file just mentioned above, this C# file defines the Data Transfer Object (DTO) for a glossary item. It represents a simplified version of a glossary item entity, typically used for transferring data between the client and server or between different layers of an application, without passing the property for 'Secret.'

6. GlossaryApp\GlossaryWebApi\wwwroot\css\site.css: simply our CSS styles for the web page layout, styles and appearance. It's a file that defines things including font styles, colours, button styles, form layouts, table styles, etc.

7. GlossaryApp\GlossaryWebApi\wwwroot\css\js\site.js: this is where the client-side functionality runs, handling user interactions. Written in JavaScript the file contains all the client-side functionality of the application. It will handle things like fetching data from the backend API, processing user input, displaying data on the webpage, and as well as managing user interactions such as adding, searching, editing, and deleting glossary terms.

8. GlossaryApp\GlossaryWebApi\wwwroot\index.html: lastly, we have the main entry point of the application. This file written in HTML contains the structure of the web page and includes sections for the main menu, adding terms, searching for terms, editing terms, and displaying the glossary.

Solution Thought/Action Process

1. **Planning and Brainstorming:** Defined all project requirements and features needed by reading over the solution scope. Brainstormed and drew up some plans for application logic.

2. **Created Console Application:** I managed to create a fully functional console version of this application in the initial stages of the project. Reason being was mainly to get a grasp of the scope and requirements of the solution, alongside wanting to make all the different functionalities work together.

NOTE: If you would like to see this console application too, let me know and I'd be more than happy to share it as well.

3. **Backend Development:** Created a web API with ASP.NET Core.

4. **Frontend Development:** Set up HTML and CSS structures, as well as implementing JavaScript functionality.

NOTE: Chose to run all together on the single server port, mainly for simplicity of project demonstration. Do understand that in future or larger scale, can opt to separate the backend project files from the frontend, and run both simultaneously but on 2 different calls. Examples of larger scale projects may include wanting to use the same backend code, but having the frontend code for Web, Mobile, maybe even Console etc. However, like mentioned in the the case of this demo simply ran app.UseDefaultFiles(); for enabling **index.html**, and app.UseStaticFiles(); for enabling **index.html** and **site.css** within current paths.

5. **Backend Development:** Implemented persistent data storage using SQLite database.
6. **Integration:** Connected frontend and backend components via API endpoints.
7. **Testing:** Tested thoroughly for bugs and usability issues throughout the whole process, that's why I don't really like putting several steps on this one. In some ways, the main test for the whole application was at the end, before submission, but like I mentioned, everything I coded was tested throughout the process to continue to enhance the solution.
8. **Deployment:** Deployed the application for public access and review.

How to Use

1. Download the ZIP folder and extract it on your local machine.
2. Once extracted, open up Visual Studio Code (VSC), and follow these steps:
    1. File → Open Folder → find the location of the extracted folder: GlossaryApp (select folder)
    2. Terminal → New Terminal → cd GlossaryWebApi
    3. CTRL + F5 to run and build the application
        - If this shortcut doesn't work, you might be prompted at the top search bar, simply choose the C# option, and then the one that creates a HTTPS run command. Or likewise you can simply run "dotnet run" and CTRL + click over the link provided. Note this link will most-likely be a HTTP link, not HTTPS. Then similarly CTRL + C to shut down application.
    4. A new https://localhost:{port} tab should open up and you should be able to interact completely with the application. The Debug Console on VSC will also be open and scripting live console updates on the processes you complete. Similarly, you can choose to open the Inspect element on Google Chrome to see console logs there too, but I added functionality so you wouldn't have to, any alert will have a message on the application itself.
    5. To close and stop the application, SHIFT + F5, or you can simply use the little toolbar that VSC opens up when the project is running and debugging, and click the red stop icon too. They do the same job.
    6. You can then go back to your terminal, as the Debug Console would have stopped, and as per the instructions: "hit any key to terminate" which will end up clearing the terminal and returning you to the original file path.
3. Use the main menu to navigate between different sections.
4. Add, search, edit, delete, and print glossary terms as needed.

Any questions about anything, comments or ideas for future enhancements, let me know, your feedback is truly appreciated! 😊