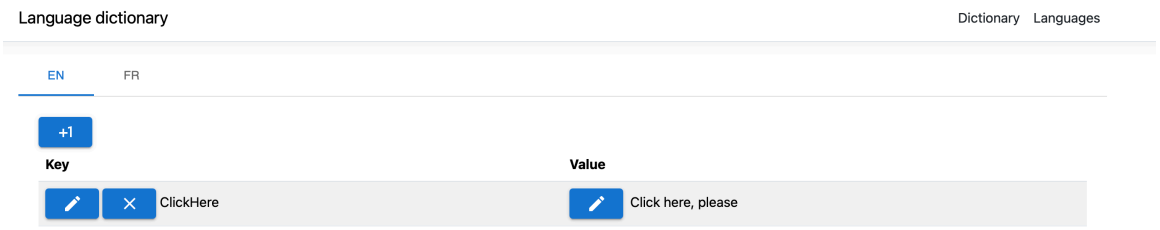
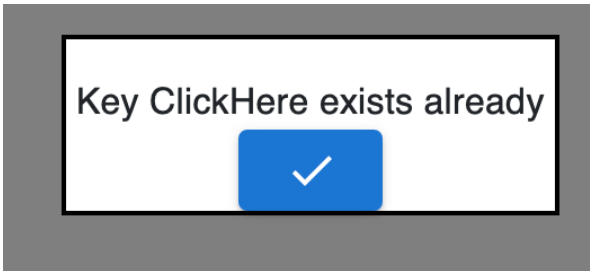
Overview of the Application

This application serves as a **localized dictionary manager** designed to facilitate translations for multilingual applications. It ensures that the data structure for localization remains consistent and error-free by enforcing a uniform set of keys across all supported languages. This approach addresses common pitfalls in localization workflows, such as missing keys or incorrect translations, especially when dealing with Excel files sent to and received from professional translators.



Being added by «+1» button, new key appears at both languages at once: En..

..and Fr



Attempt to add existing key causes error

Key Features

1. **Centralized Localization Management**:
    - Maintains a set of unique keys that are consistent across all languages.
    - Values for each key are the translations corresponding to different languages.
    - React router provides switching between Dictionaries and list of languages.

2. **Technology Stack**:
    - **Backend**: C# with ASP.NET Core (.NET 7), implemented as a Web API means we got M - Models and C - Controllers, but we do not have V - Views, there is React instead of that.
    - **Frontend**: React with Materialize for UI components (tabs, buttons and modal windows are Materialize).
    - **Database**: SQLite (for the demo purposes) with an ORM Entity Framework. Can be easily switched to PostgreSQL. There is one autogenerated migration.

3. **Data Models**:
    - **Keys**: Represents the unique set of keys shared across all languages.
    - **Languages**: Lists the supported languages.
    - **Values**: Stores the translations, linked to both keys and languages through foreign keys.

      Example of model (partly):

```
public class Values
{

[Key]
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
[Required(ErrorMessage = "{0} is required")]
public int RowId { get; set; }

[ForeignKey("KeyId")]
[Required(ErrorMessage = "{0} is required")]
public required Keys Key { get; set; }
```

4. **Development and Deployment**:
    - Development mode supports **SPA Proxy with live reload**.
    - Production mode compiles React into a static build without live reload.

5. **Validation and Consistency**:
    - Enforces key consistency to eliminate common localization errors (set of keys is the same for all languages, and they are unique).

- Future plans to include Excel import/export with built-in validation mechanisms for keys.

6. **Logger**:
   - It is Serilog with output to file and console.
7. **HTTP requests testing**
   - Swagger is provided;
   - Collection of Postman request is provided. Each response got a test checking the status code.
8. **Testing Frameworks**:
   - **Backend**: nUnit with mocked DbContext and logger.
   - **Frontend**: Jest for mocking HTTP requests and simulating user interactions like button clicks and input changes.
7. **REST API**:
   - Comprises nine endpoints leveraging LINQ for efficient querying.

     Example:

```
var row = _context.Values.First(
        row => row.Language.LanguageValue == args.LanguageValue
        &&
        row.Key.KeyValue == Regex.Escape(args.KeyValue));
row.Value = Regex.Escape(args.Value);
_context.SaveChanges();
_logger.LogInformation("Existed value updated successfully");
row.Value = args.Value;
var result = new JsonResult(row);
result.StatusCode = 200;
return result;
```

     Note: Values are saving to DB being escaped.
8. **React Component Structure**:
   - **App**: Main application wrapper.
   - **TabPages**: Manages tabs for each language.
   - **Dictionary**: Displays the dictionary for a specific language and renders rows for each key.
   - **RowDictionary**: Handles individual key-value pairs.

Example: Dictionary component render RowDictionary. Dict comes from HTTP response.

```
<tbody>
 {this.state.dict.map(row=>
    <RowDictionary
       language={this.props.language}
```

```
            key = {row.key.keyValue}
            keyName = {row.key.keyValue}
            value = {row.value}
            updateWholeDictionary = {this.populateDetails.bind(this)}
        />
    )}
    </tbody>
```

Example of handling request on client-side:

```
addRequest(arg) {
        fetch(REQUEST_URLS.AddKey, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({newKey: arg})
        }).then(response=>response.json().then(
            data=>({
                status: response.status,
                body:data
                })
            )).then(data=>{
                if (data.status==200) {
                    this.populateDetails();
                } else {
                    console.log(`got error ${data.status}`);
                    this.setState({
                        showModalMessage: true,
                        modalMessage: `${data.body}`
                    });
                }
            }).catch((error) => alert(`Response addKey returned ${error}`));
        }
```

Advantages of the Application

- **Robust Data Integrity**: Prevents issues like missing translations or inconsistent key sets that arise in manual workflows with Excel files.
- **Scalable Backend**: Built with .NET 7 and EF Core, making it easy to integrate new features or swap out the database.
- **User-Friendly Frontend**: React and Materialize offer a modern, responsive interface for managing translations.
- **Comprehensive Testing**: Ensures high-quality code with both unit tests and mocks for server and client components.

Limitations and Future Work

- **Current Limitations**:
    - MVP status means some advanced features, like Excel import/export, are not yet implemented.
    - Private repository restricts external contributions or peer reviews.

- **Planned Enhancements**:
    - **Excel Import/Export**: To streamline translator collaboration.
    - **Validation Mechanisms**: Automatic detection of missing or duplicate keys during imports.
    - **Scaling**: implementation of MemoryCache, lazy loading and pagination.

P.S. If you have any questions or you want to have look to the code, please feel free to contact me.