مستندات پروژه مدیریت ترجمه

معرفی پروژه

این پروژه یک سیستم مدیریت ترجمه واژگان است که به کاربران امکان می‌دهد واژه‌ها و ترجمه‌های آنها را در زبان‌های مختلف مدیریت کنند. سیستم دارای دو بخش اصلی است:

1. پنل مدیریت (Dashboard): برای افزودن، ویرایش و سازماندهی واژه‌ها

2. نمایش عمومی (PublicView): برای مشاهده واژه‌های ترجمه شده

معماری پروژه

- فرانت‌اند: React.js با استفاده از:

- React Router برای مدیریت مسیرها

- Context API برای مدیریت state سراسری

- Framer Motion برای انیمیشن‌ها

- Tailwind CSS برای استایل‌دهی

- ذخیره‌سازی داده‌ها: localStorage مرورگر

اجزای اصلی

1. TranslationContext: مدیریت state و عملیات مربوط به ترجمه‌ها

2. Dashboard: رابط مدیریت واژه‌ها

3. PublicView: نمایش واژه‌ها برای کاربران عمومی

4. Layout: چارچوب کلی برنامه و مسیریابی

دلایل انتخاب ساختار و منطق پیاده‌سازی :

ساختار داده

از یک آرایه از اشیا برای ذخیره واژه‌ها استفاده شده که هر شیء شامل:

- `keyword`: واژه اصلی

- `translations`: آبجکتی شامل ترجمه‌ها به زبان‌های مختلف

دلایل انتخاب:

1. سادگی: ساختار خطی و ساده برای مدیریت واژه‌ها

2. انعطاف‌پذیری: امکان اضافه کردن زبان‌های جدید بدون تغییر ساختار

3. سازگاری با localStorage: ذخیره‌سازی آسان در localStorage

4. کارایی: دسترسی سریع به واژه‌ها با استفاده از متدهای آرایه

منطق پیاده‌سازی

1. مدیریت حالت (State Management):

- استفاده از Context API به جای Redux به دلیل:

- سادگی پیاده‌سازی برای پروژه‌های متوسط

- یکپارچه‌سازی بهتر با React - نیاز نداشتن به کتابخانه‌های اضافی

2. Drag & Drop:

- پیاده‌سازی دستی بدون استفاده از کتابخانه‌های خاص به دلایل:

  - کنترل کامل بر رفتار سیستم

  - سبک‌بودن و عدم وابستگی به کتابخانه‌های خارجی

  - سادگی نیازهای پروژه

3. مدیریت ترجمه‌ها:

- استفاده از localStorage به جای پایگاه داده به دلیل:

  - نیاز نداشتن به backend برای این نسخه

  - سادگی توسعه و تست

  - عملکرد سریع برای داده‌های محدود

4. رابط کاربری:

- استفاده از Tailwind CSS برای:

  - توسعه سریع و رسپانسیو

  - کاهش حجم فایل‌های CSS

  - سفارشی‌سازی آسان

مزایای معماری انتخاب شده::

1. مقیاس‌پذیری: امکان اضافه کردن قابلیت‌های جدید مانند:

- زبان‌های بیشتر

- گروه‌بندی واژه‌ها

2. کارایی: بهینه‌سازی‌های انجام شده شامل:

- بهینه‌سازی عملیات جستجو و فیلتر

- استفاده از lazy loading برای کامپوننت‌ها

3. تجربه توسعه:

- کدهای تمیز و سازمان‌یافته

- مستندسازی مناسب

- تست‌پذیری بالا

Translation Management Project Documentation

Project Overview

This project is a vocabulary translation management system that enables users to manage words and their translations in different languages.

The system consists of two main components:

1. Admin Panel (Dashboard): For adding, editing, and organizing vocabulary
2. Public View: For viewing translated terms

Technical Architecture :

Project Architecture

- Frontend: React.js using:
  - React Router for route management
  - Context API for global state management
  - Framer Motion for animations
  - Tailwind CSS for styling
- Data Storage: Browser localStorage

Core Components :
1. TranslationContext: Manages state and translation operations
2. Dashboard: Vocabulary management interface
3. PublicView: Public display of translated terms
4. Layout: Main application framework and routing

Rationale for Selected Architecture and Implementation Logic :

Data Structure

An array of objects is used to store vocabulary terms, where each object contains:

- `keyword`: The original word - `translations`: Object containing translations in different languages

Selection Justification:
1. Simplicity: Linear and straightforward structure for vocabulary management
2. Flexibility: Easy addition of new languages without structural changes
3. localStorage Compatibility: Simple storage in localStorage
4. Efficiency: Fast access to terms using array methods

Implementation Logic :

1. State Management:
   - Context API preferred over Redux because:
     - Simpler implementation for medium-sized projects
     - Better integration with React
     - No need for additional libraries
2. Drag & Drop:
   - Custom implementation without external libraries because:

- Full control over system behavior
- Lightweight with no external dependencies
- Matches project's simple requirements

3. Translation Management:
- localStorage used instead of database because:
  - No backend required for this version
  - Simpler development and testing
  - Fast performance for limited data

4. User Interface:
- Tailwind CSS selected for:
  - Rapid responsive development
  - Reduced CSS file size
  - Easy customization

Benefits of the Chosen Architecture

1. Scalability: Capability to add new features like:
- Additional languages
- Vocabulary grouping
- Change history

2. Performance: Implemented optimizations include:
- Search and filter operation optimization
- Lazy loading for components

3. Development Experience:
- Clean and organized code
- Proper documentation
- High testability