

Globalização Dinâmica em Vue

Relatório Final do Projeto Integrador

Xavier Ribeiro Outeiro



Licenciatura em Engenharia Informática e Computação

Tutor na U.Porto: Carla Teixeira Lopes
Orientador na empresa/Proponente: Daniel Pereira Silva

28 de Junho de 2024

Conteúdo

1	Introdução	2
1.1	Enquadramento	2
1.2	Objetivos e resultados esperados	2
1.3	Estrutura do relatório	2
2	Metodologia utilizada e principais atividades desenvolvidas	3
2.1	Metodologia utilizada	3
2.2	Recursos utilizados	3
2.3	Atividades desenvolvidas	3
3	Desenvolvimento da solução	5
3.1	Requisitos Funcionais	5
3.2	Requisitos Não Funcionais	5
3.3	Tecnologias	6
3.4	Arquitetura da Aplicação	6
3.5	Base de dados	6
3.5.1	Estrutura Inicial	7
3.5.2	Estrutura Final	7
3.6	Arquitetura da API	9
3.7	Solução desenvolvida	10
3.7.1	Mudança de idioma	11
3.7.2	Atualização dinâmica de textos	15
3.8	Funcionalidades Adicionais	24
3.8.1	Erro de Fetch	24
3.8.2	Indicador de carregamento	25
3.9	Validação	25
4	Conclusões	26
4.1	Resultados alcançados	27
4.2	Lições aprendidas	27
4.3	Trabalho futuro	27

1 Introdução

1.1 Enquadramento

Este relatório tem como objetivo apresentar o desenvolvimento e respetivos resultados do projeto **Globalização Dinâmica em Vue**, realizado no âmbito da unidade curricular Projeto Integrador, do 3º ano da Licenciatura em Engenharia Informática e Computação, da Faculdade de Engenharia da Universidade do Porto. Este projeto de estágio individual, desenvolvido em ambiente empresarial, na empresa Konk Consulting¹, contando com a orientação, por parte da FEUP, da Prof.^a Carla Teixeira Lopes, e, por parte da Konk Consulting, do Eng.^o Daniel Pereira Silva.

Sendo a Konk Consulting uma empresa com experiência nos mercados globais, é conhecido e reconhecido pela empresa que, além da importância de chegar ao máximo de pessoas e culturas, a crescente expansão das empresas a nível global reflete a necessidade primordial de aplicações que suportem múltiplos idiomas e preferências regionais. E é nesta premissa que os objetivos deste projeto assentam.

Vue é uma framework de front-end amplamente utilizada e que tem um papel crucial na construção de interfaces dinâmicas e responsivas. Este projeto utiliza e explora as funcionalidades que o Vue oferece, de forma a atingir os objetivos propostos, detalhados na Subsecção 1.2.

1.2 Objetivos e resultados esperados

Este projeto tem em vista investigar como o Vue lida com a globalização, mais especificamente de que forma é que consegue apoiar a implementação de uma aplicação multilinguagem. Para além disso, este projeto tem, também, como objetivo implementar esta globalização de uma forma dinâmica, de modo a que os utilizadores, possam com facilidade editar e personalizar os textos apresentados na aplicação. Esta facilidade na perspetiva do utilizador terá que ser garantida a partir de uma interface simples e intuitiva. Do ponto de vista computacional, o objetivo é que as alterações, tanto de idioma, como as respetivas atualizações por parte do utilizador, não necessitem de ser recompiladas, ou seja, que estas alterações, de idioma e dos textos da aplicação, aconteçam dinamicamente. Outro dos objetivos deste projeto passa também por implementar as soluções na forma de uma aplicação completa, isto é, que seja criada uma base de dados e uma API REST de forma a que a aplicação consiga interagir com a base de dados.

1.3 Estrutura do relatório

O relatório é desenvolvido ao longo de 3 secções para além da introdução. Na segunda secção são descritas as atividades desenvolvidas ao longo do projeto

¹<https://www.konkconsulting.com/EN>

assim como a metodologia e recursos utilizados. A secção mais extensa do relatório, a terceira, é onde é descrito todo o desenvolvimento das soluções implementadas de forma a alcançar os objetivos definidos na primeira secção. É nesta secção que são, também, definidos os requisitos funcionais e não funcionais, são listadas as tecnologias utilizadas e é descrita a arquitetura da aplicação assim como as arquiteturas da base de dados e da API. Finalmente, na última secção são feitas algumas reflexões sobre lições aprendidas no decorrer do projeto e, também, em relação aos resultados alcançados e possível trabalho futuro no contexto deste projeto.

2 Metodologia utilizada e principais atividades desenvolvidas

Esta secção descreve a forma como este projeto foi realizado, isto é, qual foi a metodologia e os recursos utilizados, e também quais foram as atividades desenvolvidas ao longo do tempo.

2.1 Metodologia utilizada

Após uma consulta, a partir da documentação de Vue.js e da realização do tutorial *Getting Started*, foi iniciado o desenvolvimento das soluções para os objetivos propostos para o projeto. Este projeto foi realizado nos escritórios da Konk Consulting e foi utilizada uma metodologia que se baseou num desenvolvimento iterativo e individual com reuniões semanais de acompanhamento com o orientador por parte da Konk Consulting.

2.2 Recursos utilizados

Foram utilizados vários recursos, que permitiram e facilitaram o desenvolvimento deste projeto:

- Vuejs.org² para o estudo e exploração de Vue.
- Git para controlo de versões.
- Postman API³ para teste de endpoints da API.
- MongoDB Compass⁴ para visualização da base de dados.

2.3 Atividades desenvolvidas

As atividades desenvolvidas neste projeto distribuíram-se no tempo como demonstrado na Figura 1.

²<https://vuejs.org/>[5]

³<https://www.postman.com/>

⁴<https://www.mongodb.com/products/tools/compass>[1]

Na fase inicial do projeto, o foco principal foi o estudo e exploração do Vue. Foram também exploradas algumas possibilidades de implementação das soluções finais.

A partir desse ponto foi iniciado o desenvolvimento da aplicação final que se estendeu praticamente até ao fim da data final do projeto incluindo, além do desenvolvimento de todas as soluções implementadas, o desenvolvimento da base de dados, descrito na Subsecção 3.5, e o desenvolvimento da API REST, utilizada para estabelecer a comunicação entre a aplicação e a base de dados, descrito na Subsecção 3.5.

Paralelamente ao desenvolvimento da aplicação, a partir do segundo terço do projeto e até ao final, foi, também, escrito este relatório.

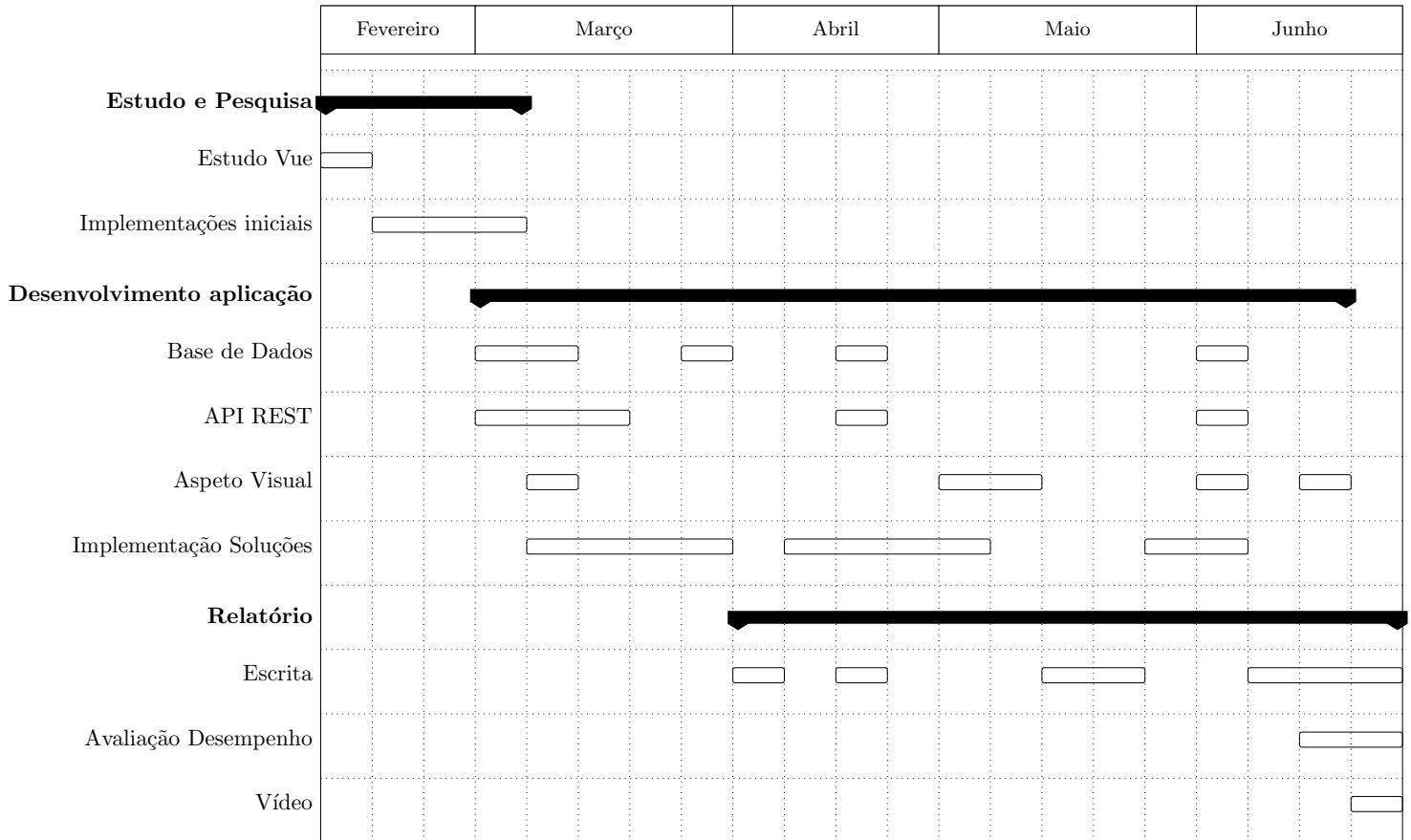


Figura 1: Diagrama de Gantt que demonstra a distribuição no tempo das atividades realizadas ao longo do projeto.

3 Desenvolvimento da solução

Esta secção cobre todos os pontos relativos ao desenvolvimento da solução, desde a identificação dos requisitos funcionais e não funcionais até à validação da solução desenvolvida, sob a forma de testes de desempenho. Mas principalmente, descreve detalhadamente todo o desenvolvimento das soluções implementadas e, também, o desenvolvimento da base de dados e da API, utilizada para a comunicação entre a aplicação e a base de dados.

3.1 Requisitos Funcionais

Inicialmente e de forma a cumprir os objetivos propostos para o projeto foram identificados os principais requisitos funcionais:

- Armazenamento das traduções em base de dados. Implementação descrita na Subsecção 3.5.
- Possibilidade de alteração do idioma da aplicação com base nos idiomas disponíveis. A descrição da solução implementada encontra-se na Subsecção 3.7.1.
- Possibilidade, por parte do utilizador, de edição, atualização e criação de textos da aplicação. Implementação encontra-se detalhada na Subsecção 3.7.2.

3.2 Requisitos Não Funcionais

Os requisitos não funcionais identificados como mais importantes foram:

- Eficiência - Espera-se que a aplicação não necessite de ser compilada para cada idioma e que nenhuma das suas funcionalidades requeira uma atualização da página.
- Desempenho - É expectável que a aplicação utilize o menor número de recursos computacionais possível e de forma eficiente, isto é, que não se façam pedidos ao servidor de dados que já estão em memória, ou pedidos ao servidor de dados que não são necessários.
- Usabilidade - Espera-se que a interface da aplicação torne a alteração e tradução de textos intuitiva e descomplicada, de forma a que os utilizadores não tenham que fazer nada mais para além da tarefa que tinham como objetivo realizar.

Em relação a restrições do projeto, além de ter de estar finalizado até dia 28 de junho de 2024, tem, também, de utilizar a framework Vue.

3.3 Tecnologias

Neste projeto foram utilizadas as seguintes tecnologias:

- Vue.js ⁵ - *Framework* de JavaScript para desenvolvimento *Frontend*. Esta *framework* enquadra-se bastante bem no contexto deste projeto devido a utilizar um sistema reativo de renderização.
- Node.js Express ⁶ - *Framework* de JavaScript utilizado na criação da API, descrita detalhadamente na Subsecção 3.6.
- MongoDB ⁷ - Base de dados NoSQL utilizada no desenvolvimento da aplicação, cuja descrição pode ser lida na Subsecção 3.5.2.
- i18n ⁸ - Biblioteca de Vue utilizada no desenvolvimento da solução referente ao objetivo da mudança dinâmica de idioma. Solução que se encontra apresentada na Subsecção 3.7.1.

3.4 Arquitetura da Aplicação

Com o objetivo de desenvolver e implementar as soluções face aos objetivos definidos no início do projeto, desenvolveu-se uma aplicação web de notícias como prova de conceito.

A aplicação é desenvolvida em Vue.js e realiza pedidos HTTP a um servidor que implementa uma API REST, desenvolvido em Node.js Express, servidor este que se comunica com a base de dados, em MongoDB, de forma a satisfazer os pedidos feitos pela aplicação conforme visível na Figura 2.

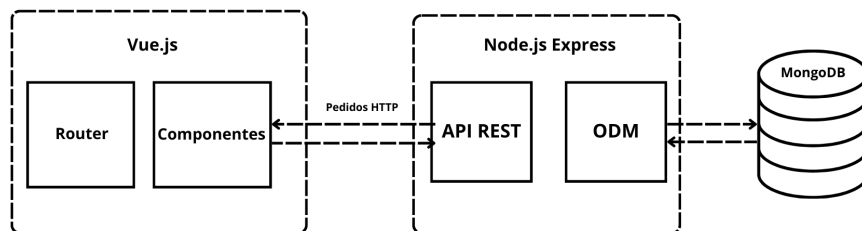


Figura 2: Desenho da Arquitetura da Aplicação.

3.5 Base de dados

A estrutura da base de dados ao longo do projeto foi sofrendo alterações, sendo que as dois principais estados da base de dados, o estado inicial e o estado final, são descritos nesta Subsecção.

⁵<https://vuejs.org/>[5]

⁶<https://expressjs.com/>[2]

⁷<https://www.mongodb.com/>[4]

⁸<https://www.i18next.com/>[3]

3.5.1 Estrutura Inicial

Inicialmente, de modo a testar possíveis implementações da mudança de idiomas e da alteração textos de forma dinâmica, ainda não existia realmente uma base de dados, foram utilizados ficheiro JSON com uma estrutura como demonstrada no Extrato de código 1.

```
1 { "Idiomas": [  
2   {  
3     "id": "1",  
4     "nome": "idioma1",  
5     "vocabulario": {  
6       "Identificador": "Traducao",  
7       ...  
8       "noticias" : {  
9         "Identificador": {  
10          "Identificador": "identificador",  
11          "Titulo": "titulo",  
12          "Subtitulo": "subtitulo",  
13          "Corpo": "corpo"  
14        },  
15        ...  
16      }  
17    }, ... ] }
```

Extrato de Código 1: Estrutura de dados utilizada numa fase inicial.

3.5.2 Estrutura Final

De acordo com os requisitos do projeto, foi necessário criar uma base de dados de forma a armazenar os dados dos idiomas e todo o vocabulário relativo a cada um destes idiomas, sendo necessário tomar uma decisão entre a utilização de uma base de dados relacional ou uma base de dados não relacional.

Como a estrutura de dados no Extrato de código 1 se revelou adequada e bastante compatível com a utilização do i18n, que traduz os textos de acordo com um par Chave - Tradução, em MongoDB, por diversas razões.

A primeira delas assenta no facto de facilitar o desenvolvimento da solução, em relação a opção de uma base de dados relacional, pois não é necessário realizar o mapeamento da estrutura em JSON para uma estrutura relacional.

Outra razão que levou a esta decisão foi que a agilidade de desenvolvimento que uma base de dados não relacional oferece, o se adequa, não só ao desenvolvimento das soluções mas também à dimensão e tempo de duração do projeto.

Esta decisão assentou, ainda, em haver já alguma experiência prévia com bases de dados relacionais e muito pouca experiência com bases de dados não

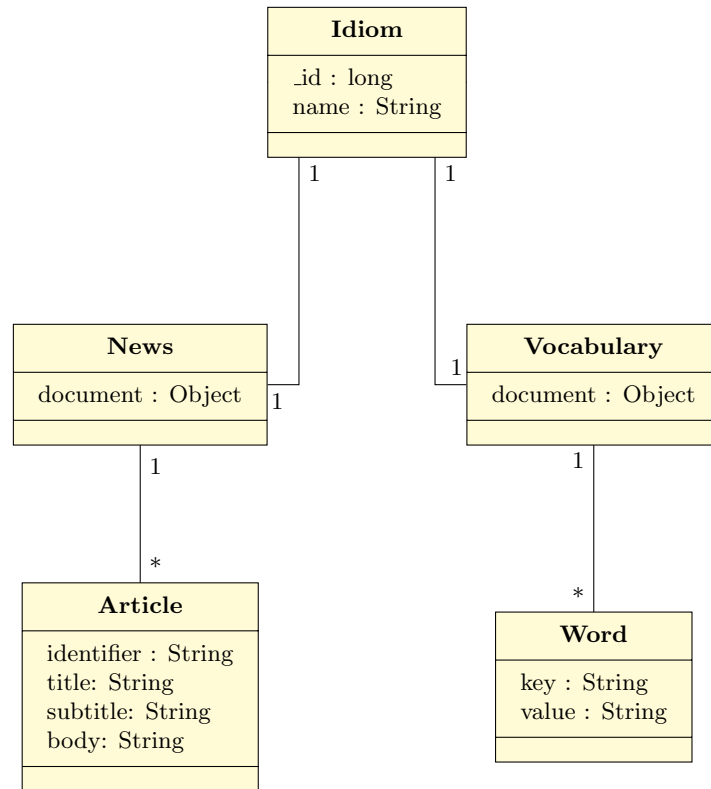


Figura 3: Diagrama relativo à estrutura final dos dados de cada idioma.

relacionais, considerou-se, então, que este projeto seria uma excelente oportunidade para explorar a utilização de bases de dados não relacionais.

Sendo que a estrutura dos dados de todos os conjuntos de idiomas é a mesma, esta estrutura pode ser demonstrada graficamente a partir da Figura 3.

Os nomes utilizados na Figura 3 são descritos nos pontos seguintes:

- *Idiom* - Representa um documento na base de dados e contém como atributos o nome reservado `_id` do documento e o nome do idioma.
- *Vocabulary* - Funciona como um agregador do conjunto de pares Chave-Tradução (*Word*).
- *News* - Agrupa um conjunto de artigos (*Article*), artigos estes que têm como atributos o identificador, o título, o subtítulo e o conteúdo do artigo.

3.6 Arquitetura da API

De forma a realizar a interação entre a aplicação e a base de dados criou-se uma API REST em Node.js Express, que permite realizar os pedidos que constam na Tabela 1.

Método e Descrição	Resposta
GET /Idioms Retorna um array de objetos correspondentes a todos os idiomas da base de dados.	<pre>[{"_id": "662bd33018e1e450ade1971c", "name": "en", "vocabulary": { ... }, "News": { ... } }, { ... }]</pre>
GET /Idioms/:name Retorna o objeto correspondente ao idioma {{name}}.	<pre>{"_id": "662bd33018e1e450ade1971c", "name": ":name", "vocabulary": { "Home": "Home", "About": "About", ... }, "News": { ... }}</pre>
GET /Idioms/:name/vocabulary Retorna o objecto correspondente ao vocabulário do idioma {{name}}.	<pre>{"vocabulary": { "Home": "Home", "About": "About", "about-title": "This is an about page", "...": "..."} }</pre>
GET /Idioms/get/names Retorna um array de objetos com um par {name : {{name}}}.	<pre>[{"name": "en"}, {"name": "..."}]</pre>
GET /Idioms/mainLanguage Retorna uma string correspondente ao node do idioma principal.	<pre>{"main_language": "en"}</pre>

Continua na próxima página

Método e Descrição	Resposta
POST /Idioms Cria um objeto na base de dados de acordo com o corpo do pedido.	<pre>{ "_id": "new_id", "name": "new_language", "vocabulary": { ... }, "News": { ... } }</pre>
PUT /Idioms/:name/vocabulary/ Atualiza o vocabulário do idioma {{name}} de acordo com a key e o value passado no corpo do pedido, se a key não existir, o par key:value é criado, se a key existir é atualizado o value correspondente.	<pre>{ "vocabulary": { "existing_key": "new_value", "new_key": "value" } }</pre>
PUT /mainLanguage/ Atualiza na base de dados o idioma principal.	<pre>{ "main_language": "new_main_language" }</pre>
DELETE /Idioms/:name Apaga o objeto do idioma pelo nome.	<pre>{ "message": "Idioma :name apagado com sucesso" }</pre>
DELETE /Idioms/:name/vocabulary/:key Apaga uma key do vocabulário de um idioma.	<pre>{ "message": "Key :key apagada do vocabulario de :name com sucesso" }</pre>

Tabela 1: Possíveis pedidos a realizar à API

3.7 Solução desenvolvida

As soluções desenvolvidas neste projeto são divididas em duas subsecções. A primeira descreve a implementação que procura alcançar o objetivo de ser possível a alteração do idioma da aplicação de uma forma dinâmica. A segunda detalha a implementação de várias soluções que pretendem ir de encontro com o objetivo do utilizador ter a possibilidade de editar e personalizar dinamicamente os textos apresentados na aplicação.

3.7.1 Mudança de idioma

Face ao desafio da aplicação suportar diversos idiomas, sem haver a necessidade de compilar o código uma vez para cada idioma, e também mantendo os princípios de uma *Single Page Application* (SPA), optou-se pela implementação de uma abordagem que consiste em, a partir de vários conjuntos de pares Chave - Valor, um conjunto para cada idioma, e sempre que a opção do idioma do website é alterada, todos os componentes textuais do website que sejam implementados com a formatação correta, isto é, que sejam escritos da forma: `{{ $t("Chave") }}` serão atualizados para esse idioma. É possível utilizar esta solução em todos os elementos HTML que possam ter conteúdo textual.

Para isso é necessário instanciar um objeto do tipo I18N, a partir da biblioteca i18n de Vue, e como se pode ver no Extrato de código 2, o objeto criado é exportado, podendo assim ser importado e utilizado em todas as páginas de Vue, sendo necessário nos ficheiros em que é implementada tradução dinâmica.

```
1  const i18n = createI18n ({
2    locale: "en",
3    fallbackLocale: "en",
4    legacy: false,
5    messages: loadLocaleMessages(),
6  });
7
8  export default i18n;
```

Extrato de Código 2: Função que instancia um objeto da biblioteca i18n.

Os atributos do objeto i18n são:

- locale: idioma atual, que pode ser alterado no menu dropdown de seleção de idioma que está presente no header da aplicação
- fallbackLocale: idioma de recurso, para quando o idioma atual não existe
- legacy - booleano referente à versão de Vue.js que estamos a utilizar
- messages - contém as *messages* do idioma atual

De forma a inicializar o atributo messages é realizada uma chamada à função *loadLocaleMessages()*, cuja versão inicial é demonstrada no Extrato de código 3, e a sua versão final demonstrada no Extrato de código 4.

Numa primeira fase do projeto, ao iniciar a aplicação, a partir de uma base de dados em MongoDB, eram carregados, via HTTP, todos os dados relativos aos idiomas, isto é, todos os documentos da coleção de idiomas. Sendo que estes dados eram depois armazenados num array de objetos *idioms* em que cada objeto correspondia a um idioma, como demonstrado no Extrato de código 3.

```

1 function loadLocaleMessages() {
2   let locales = [];
3   for (const idiom of idioms) {
4     locales.push({ [idiom.name]: {...idiom.vocabulary,
5                                   ...idiom.News} });
6   }
7   const messages = {};
8   locales.forEach((lang) => {
9     const key = Object.keys(lang);
10    messages[key] = lang[key];
11  });
12  return messages;
13 }

```

Extrato de Código 3: 1ª implementação da função `loadLocalMessages()`.

E a partir destes idiomas era construído o objeto *messages*, como demonstrado na função acima, ou seja, se, por exemplo, quiséssemos o vocabulário do idioma "en" poderíamos obtê-lo ao fazer `messages["en"]`. E, deste modo, a partir das keys das *messages* era possível obter todos os idiomas disponíveis, utilizando a propriedade do `i18n`, *availableLocales*. Esta propriedade era utilizada num menu dropdown no header da aplicação de forma a ser possível alterar o idioma atual, que é obtido desta forma: `let locale = useI18n()`.

Esta implementação não satisfaz o requisito que propõe que a aplicação utilize o menor número de recursos computacionais possível e de forma eficiente, isto é, que não se façam pedidos ao servidor de dados que já estão em memória, ou pedidos ao servidor de dados que não são necessários.

De forma a satisfazer este requisito, ao inicializar a aplicação é verificado se existe em armazenamento local uma variável que contém a string que identifica o idioma preferencial do utilizador. Se não existir, é carregado, via HTTP, a a partir da base de dados, a string que identifica o idioma principal da aplicação, guardando-a em armazenamento local, sendo que essa string passará a ser o idioma preferencial do utilizador. Sempre que o utilizador alterar o idioma em que a aplicação se encontra, será alterada, em armazenamento local, a variável referente ao idioma preferencial do utilizador para o idioma agora selecionado. As leituras e alterações feitas a variáveis em armazenamento local são feitas utilizando os métodos `localStorage.getItem` e `localStorage.setItem`.

A partir da string correspondente ao idioma preferencial do utilizador, são carregados, também via HTTP, todos os dados relativos ao idioma cujo nome equivale à string do idioma preferencial, estes dados são o nome, o vocabulário e as notícias. O Extrato de código 4 faz corresponder o nome desse idioma ao seu vocabulário e notícias.

```

1 function loadLocaleMessages() {
2   return { [idiom.name]: { ...idiom.vocabulary, ...
      idiom.News } };
3 }

```

Extrato de Código 4: Implementação final da função `loadLocaleMessages()`.

Assim sendo, inicialmente só é carregado o idioma definido como preferencial. Esta mudança levanta um problema relativo à integração com a solução inicialmente desenvolvida. Problema este que consiste no facto da propriedade do `i18n` *availableLocales* passar a conter só o idioma preferencial, e, por isso, o menu dropdown deixar de ter disponíveis todos os outros idiomas que estão presentes na base de dados.

Por isso, ao iniciar a aplicação são carregados, via HTTP, todos os nomes correspondentes aos idiomas disponíveis. Sendo que o *availableLocales* contém apenas todas as chaves da propriedade *messages* do `i18n`, o menu dropdown passou a utilizar os nomes dos idiomas carregados na inicialização da aplicação que foram guardados numa variável, *idiom_names*.

Assim, sempre que o idioma é alterado, é chamada a função *getNewMessages* com o nome do idioma escolhido como argumento, que realiza um pedido HTTP ao servidor, retornando o vocabulário e as notícias do idioma escolhido. Sendo que depois o vocabulário e as notícias retornadas por esta função são adicionadas à propriedade do `i18n` *messages* utilizando o método *setLocaleMessage*, também do `i18n`, que recebe como argumentos o nome do idioma a adicionar e o conjunto de *messages* desse idioma, que neste caso são o vocabulário e as notícias do idioma escolhido.

```

1 localStorage.setItem('preferred_language', newLocale)
2   ;
3 if (!i18n.global.availableLocales.includes(locale.
4   value) && i18nFunctions.idioms == null) {
5   let new_locale_messages = await getNewMessages(
      locale.value);
6   i18n.global.setLocaleMessage(locale.value,
      new_locale_messages);
7 }

```

Extrato de Código 5: Função que obtém dados relativos a um idioma.

Como podemos ver no Extrato de código 5, este procedimento só se realiza caso o idioma não tenha sido já previamente carregado. Isto é garantido pela condição que verifica se o nome desse idioma não está incluído no array *availableLocales*, e também, se o array que contém todos os idiomas não existe ainda. Este último prende-se ao facto de algumas páginas da aplicação, como a página da tabela de alteração de traduções e a página onde se pode exportar os dados dos idiomas, necessitarem de todos os idiomas e não só do idioma selecionado num determinado momento.

O fluxo computacional descrito pode ser observado na Figura 4.

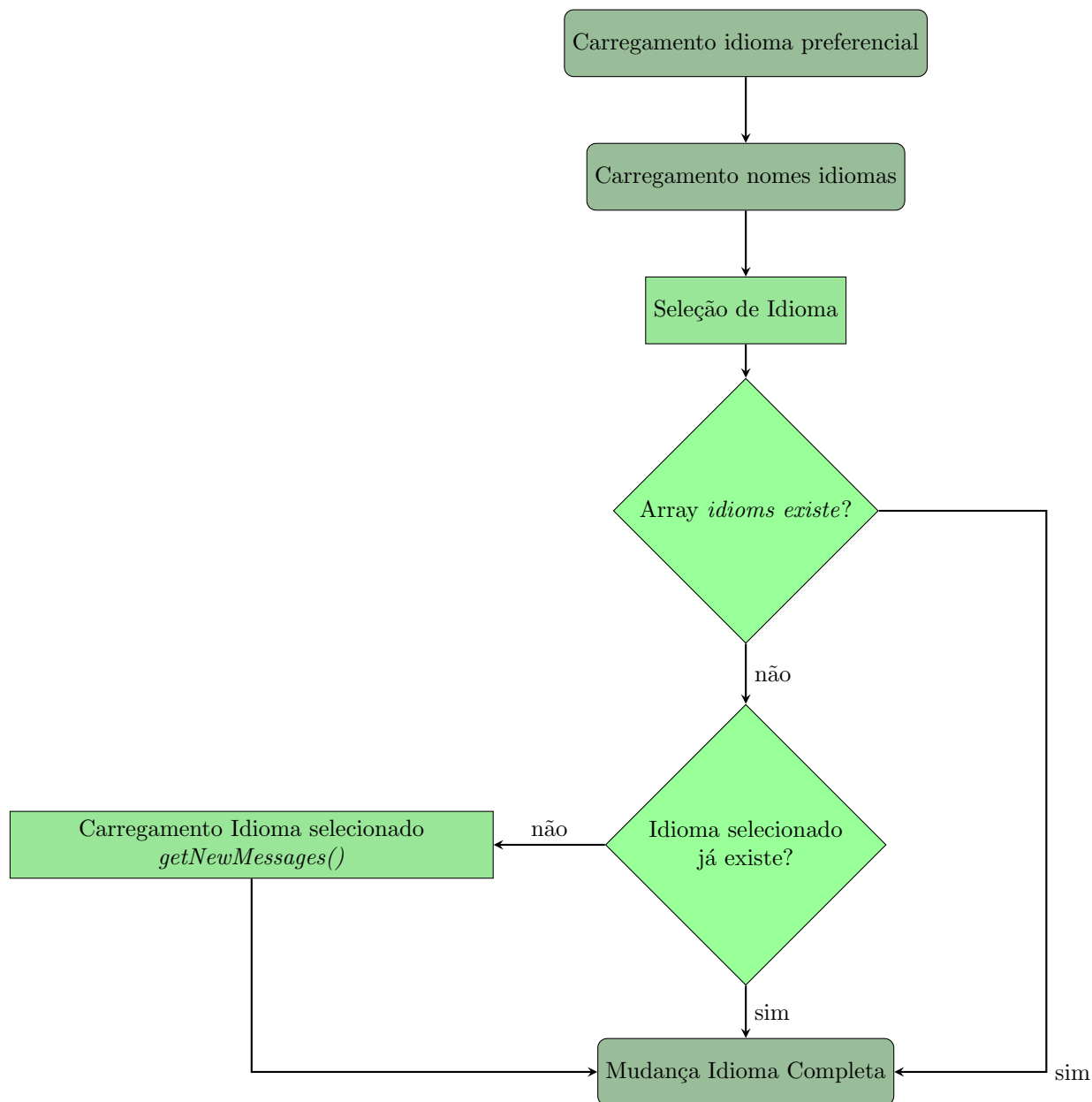


Figura 4: Fluxograma relativo ao processo de carregamento de idiomas.

Desta forma fica garantido que não são feitos pedidos ao servidor de dados que já estão em memória, nem são feitos pedidos ao servidor de dados que não

sejam necessários. Cumprindo assim o requisito proposto relativo ao desempenho da aplicação, definido na Subsecção 3.2.

Sempre que existe alguma atualização no componente da aplicação, utilizando o método *onUpdate* de Vue, é atualizada a variável *currentMessages* que contém o vocabulário e as notícias do idioma selecionado.

É a partir deste objeto, *currentMessages*, que podemos utilizar o método *\$t*, método da biblioteca do *i18n*, com a Chave do par que queremos utilizar, escrita no código na forma: `{{ $t("Chave") }}` e irá ser mostrado na página a Tradução correspondente à Chave, no idioma atual. O método *\$t* utiliza o par Chave - Tradução das *currentMessages* do locale atual, e por isso, sempre que houver uma mudança de idioma, todos os componentes que utilizem este método serão atualizados automaticamente, sem necessidade de realizar nenhum pedido ao servidor. Caso não exista nenhuma Tradução correspondente à Chave, a string apresentada será a que estiver a ser utilizada como Chave dentro do método *\$t*.

Além de importar a instância do *i18n* criada é também necessário configurar a nossa aplicação de modo a que seja possível a utilização do *i18n*, e isto foi feito no ficheiro *main.ts*, fazendo *app.use(i18n).mount('#app')*.

3.7.2 Atualização dinâmica de textos

Em relação à atualização dinâmica de textos foram implementadas várias soluções, de forma a explorar as vantagens e desvantagens das várias possibilidades, mas também de forma a que a aplicação se enquadrasse com as possíveis necessidades de qualquer potencial utilizador.

Todas as soluções utilizam a mesma função, *changeWord* para atualizar um componente textual.

```
1 function changeWord(  
2   key: string, //Chave a alterar  
3   word: string, //Nova Tradução  
4   locale: string, //Idioma no qual a tradução esta a  
      ser alterada  
5   currentMessages: CurrentMessages, //Vocabulário do  
      idioma atual  
6   currentMessages_locale: string, //Idioma atual  
7   idioms: any, //Todos os idiomas  
8   is_News: boolean //Booleano que verifica se o  
      texto a alterar e uma notícia  
9 )
```

Extrato de Código 6: Função que realiza a Mudança de Tradução de um par Chave-Tradução.

A função demonstrada no Extrato de código 6 verifica primeiro se o texto a alterar é ou não uma notícia de forma a definir corretamente o url do pedido a realizar, ou seja, se é *locale/news* ou *locale/vocabulary*, e depois realiza um pedido PUT ao servidor, que contém, no corpo do pedido, a Chave a alterar e a nova Tradução. Pedido este que irá criar na base de dados, no idioma correspondente ao *locale* no url, um par Chave-Tradução, caso a Chave não exista ainda. Caso exista, irá ser feita uma atualização da Tradução correspondente à Chave a alterar. Esta atualização é realizada a nível da base de dados, mas também localmente, onde são atualizadas as *currentMessages*, estrutura que contém todo o vocabulário do idioma atual. Esta atualização é feita, também, na estrutura que contém todos os idiomas, *idioms*.

Em relação à possibilidade do texto a alterar ser uma notícia, a única diferença relativa a outro texto é que a Chave de uma notícia é uma Chave composta, por exemplo, se quisermos alterar a tradução do Título da notícia com o identificador N1, a Chave desta notícia seria *N1.Title*.

Isto leva-nos a ter que diferenciar ligeiramente a implementação da alteração de textos simples e de notícias, ou textos com Chave composta. Esta diferença é notória, não só no url dos pedidos ao servidores, mas também na forma como a Chave a ser enviada tem que ser construída, devido a, no caso das notícias, de forma a construirmos a Chave na forma: *N1.Title*, precisarmos de percorrer o Objeto do idioma em profundidade. E isso é feito utilizando a função demonstrada no Extrato de código 7, em que se itera pelos segmentos divididos por um '.' da Chave a alterar até se encontrar o último segmento, onde se atribui o valor da nova Tradução.

```
1  let keys = key.includes('.') ? key.split('.') : []
2  let current = currentMessages
3  for (let i = 0; i < keys.length; i++) {
4    if (i === keys.length - 1) {
5      current[keys[i]] = i === keys.length - 1 ?
6        word : {}
7    }
8    else {
9      current = current[keys[i]];
10  }
```

Extrato de Código 7: Procedimento realiza a alteração da Tradução em Chaves Compostas.

Solução nº1 - Edição no local do elemento (Pop-up)

A primeira solução baseia-se na possibilidade de editar cada elemento textual no local onde ele aparece na aplicação. Isto é, quando é passado o cursor por cima de qualquer elemento, cuja a edição seja possível, é criado e mostrado um

botão de edição junto ao elemento que, se clicado, mostra um formulário num pop-up, onde é possível alterar a Tradução do elemento selecionado.

Para isso, no ficheiro principal Vue da aplicação, normalmente *App.vue*, é necessário utilizar o Extrato de código 9, que é executado na inicialização da aplicação, *onMounted*, que é também necessário em todas os ficheiros de páginas da aplicação em que seja utilizada esta solução, e sempre que existe alguma atualização, principalmente sempre que o idioma é alterado e/ou sempre que a página mostrada pelo *<RouterView>* é alterada, *onUpdated*. É também necessário que os elementos editáveis tenham duas classes, uma com o nome da Chave, o mesmo que é utilizado dentro do método *\$t* e outra com o idioma que esta atualmente selecionado, ou seja, com a estrutura demonstrada no Extrato de código 8.

```
1 <elemento class="Chave" :class="idioma selecionado">{{
    $t("Chave") }}</elemento>
```

Extrato de Código 8: Estrutura necessária para os elementos serem editáveis.

```
1   onMounted(() => {
2     utils.populateEditableElements(locale.value,
3       currentMessages);
4   });
5   onUpdated(() => {
6     currentMessages = i18n.global.getLocaleMessage(
7       locale.value);
8     utils.populateEditableElements(locale.value,
9       currentMessages);
10  });
```

Extrato de Código 9: Procedimento que realiza o povoamento Elementos editáveis.

Estes métodos chamam a função *populateEditableElements*, que recebe como argumentos, o idioma atual e todo o vocabulário do idioma atual.

A função inicialmente seleciona todos os elementos que contêm como classe o idioma atual, recebido como argumento, itera sobre eles e adiciona, a cada um, um *eventListener* que quando se passa por cima do elemento executa a função *addEditButton* que cria um botão de edição junto ao elemento. Botão esse que contém, também, um *eventListener* que faz com que quando é clicado chame a função *openPopUp*.

Esta função recebe como argumentos, além do idioma atual e do vocabulário do idioma atual, o elemento textual a ser editado e as classes desse elemento. Desta forma é possível obter a Chave a alterar, que é a classe que não é o idioma atual. A função *openPopUp* constrói o pop-up, sendo que o formulário que fica no centro do ecrã, e tudo à sua volta fica mais escuro, como se pode ver na Figura 5

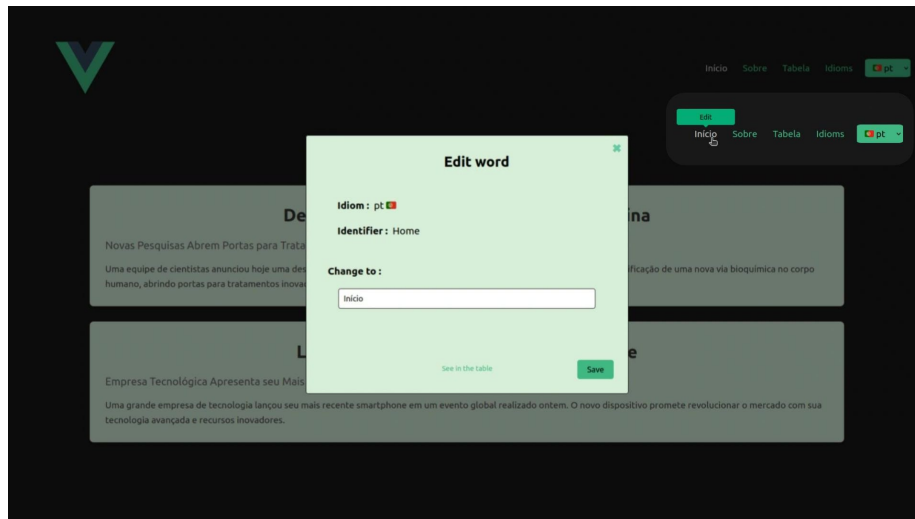


Figura 5: *ScreenShot* do Popup relativo à edição de Traduções.

O formulário contém:

- Informação sobre a Chave que se está a alterar e o idioma atual.
- Uma caixa de texto preenchida com o valor atual da Tradução do elemento selecionado.
- Botão para fechar o pop-up, sem guardar qualquer alteração. É também possível fechar o pop-up ao clicar na tecla ESC.
- Botão para guardar qualquer alteração realizada.
- Link para a Tabela onde é possível alterar todos os pares Chave-Tradução.

O link do formulário direciona o utilizador para a linha da tabela onde se encontra a Chave que se está a alterar e o idioma atual. Isto é possível porque na construção da tabela, todas linhas, elementos `<tr>`, têm como classes a sua Chave e idioma, assim na construção do link isso é utilizado de forma a obter o elemento correspondente, e em seguida utiliza-se o método `scrollIntoView`, que faz com a página se posicione de forma a que o elemento fique no centro do ecrã. Este procedimento é demonstrado no Extrato de código 10.

```

1  const elementToScrollTo = document.querySelector('tr
    . ' + key + '.' + locale)
2  elementToScrollTo.scrollIntoView({ behavior: 'smooth',
    block: 'center' });

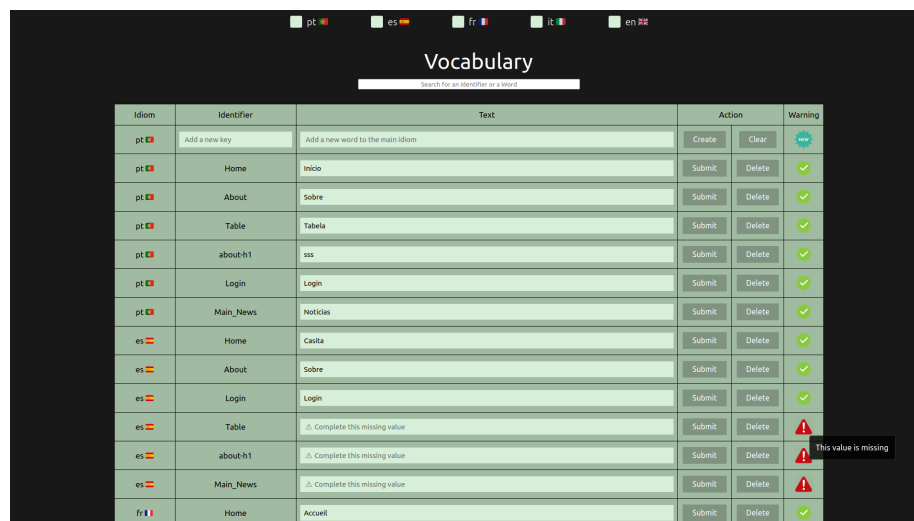
```

Extrato de Código 10: Procedimento que posiciona a página de acordo com o elemento escolhido.

Se alguma alteração for realizada ao valor escrito na caixa de texto, ao clicar no botão para guardar as alterações é executada a função `changeWord()`, demonstrada no Extrato de código 6.

Solução Nº2 - Tabela com todos os pares Chave-Tradução

A segunda solução implementada consiste em mostrar, numa tabela, Figura 6, todos os pares Chave-Tradução, de todos os idiomas, existindo a possibilidade de editar cada Tradução na linha da tabela correspondente ao par Chave-Tradução. É também possível criar e apagar palavras, filtrar as palavras que queremos ver na tabela por idioma e pesquisar, tanto por Chave como por Tradução. Além disto é mostrado um ícone no final de cada linha da tabela que indica se esse par está a ser utilizado ou não, ou se é um par em que a Chave não tem ainda uma Tradução respetiva.



Idiom	Identifier	Text	Action	Warning
pt	Add a new key	Add a new word to the main idiom	Create Clear	
pt	Home	Início	Submit Delete	✓
pt	About	Sobre	Submit Delete	✓
pt	Table	Tabela	Submit Delete	✓
pt	about-h1	sss	Submit Delete	✓
pt	Login	Login	Submit Delete	✓
pt	Main_News	Noticias	Submit Delete	✓
es	Home	Caixa	Submit Delete	✓
es	About	Sobre	Submit Delete	✓
es	Login	Login	Submit Delete	✓
es	Table	⚠ Complete this missing value	Submit Delete	⚠ This value is missing
es	about-h1	⚠ Complete this missing value	Submit Delete	⚠ This value is missing
es	Main_News	⚠ Complete this missing value	Submit Delete	⚠ This value is missing
fr	Home	Accueil	Submit Delete	✓

Figura 6: *Screenshot* da tabela com todos os pares Chave-Tradução

Relativamente a elementos textuais com Chave complexa, como é o caso das notícias, a implementação não foi tão extensiva como para os elementos textuais com Chave simples, limitando-se apenas à alteração dos textos, visto que a implementação nos elementos textuais de chave simples provou-se suficiente como prova de conceito, sendo que uma implementação em elementos textuais de Chave Complexa seria redundante.

Para construir e mostrar a tabela com todos os pares Chave-Tradução existentes é utilizada a diretiva de Vue, *v-for*, que itera sobre uma estrutura de dados construída de forma a implementar a filtragem de idiomas selecionados. Esta estrutura, *filteredIdioms*, é o resultado de uma chamada à função *getFilteredIdioms()* que retorna todos os idiomas, contendo o nome do idioma e todo o

seu vocabulário, cujo o nome está presente no array de idiomas selecionados.

```
1 const filteredIdioms = computed(() =>
  getFilteredIdioms());
```

Extrato de Código 11: Computação da variável `filteredIdioms`.

No Extrato de código 11, utilizado o método *computed*, de Vue, que torna o Objeto *filteredIdioms* reativo e qualquer alteração nos filtros leva a que os idiomas filtrados sejam recalculados novamente e que a tabela seja visualmente alterada quase instantaneamente e sem necessidade de recarregar a página.

Da mesma forma é construído Objeto *vocabulary*, utilizando o método *computed*, como pode ser observado no Extrato de código 12.

```
1 const vocabulary = computed(() => {
2   const idiomNames = idioms.map((
3     idiom: { name: any; }) => idiom.name);
4
5   return idiomNames.map((name: string) =>
6     getVocabulary(name));
7 });
```

Extrato de Código 12: Computação da variável `vocabulary`.

A diferença é que, neste caso, se faz corresponder o nome do idioma ao seu vocabulário, que é obtido por uma chamada à função, *getVocabulary()*, que recebe o nome do idioma como argumento. Esta função implementa a pesquisa por Chave ou por Tradução, ao retornar o objeto que a função *filterSearch()* retorna. Por sua vez, a função *filterSearch()* recebe como argumento, o vocabulário do idioma, equivalente ao que se encontra na base de dados, vindo do Objeto *idioms* que é obtido pelo pedido HTTP que é realizado ao iniciar a aplicação, e também o valor do elemento *<input>* relativo à pesquisa por Chave ou Tradução. Este valor está vinculado à variável que é passada na função, e vice-versa, utilizando a diretiva de Vue, *v-model*. Tendo isto em conta, a função *filterSearch()* limita-se a retornar uma cópia do vocabulário, passado como argumento da função, mas remove todos os pares que não contêm o valor de pesquisa tanto na Chave como na Tradução. Caso o valor de pesquisa esteja vazio, nenhum par é removido. Esta verificação não é sensível à diferença entre letras maiúsculas e minúsculas.

De modo a ter todos os pares Chave-Tradução, de todos os idiomas, na tabela, itera-se sobre cada idioma contido no Objeto *vocabulary*, e, cada linha da tabela, é composta por:

- Nome e bandeira do idioma.
- Chave.
- Input com o valor atual da Tradução, ou aviso em forma de placeholder, em caso de não existir ainda Tradução.

- Botão para confirmar a alteração do par Chave-Tradução,
- Botão para apagar o par Chave-Tradução.
- Ícone relativo ao estado de utilização atual do par Chave-Tradução.

Ao clicar no botão para apagar o par Chave-Tradução, é chamada a função *deleteElement()*, que, além de realizar um pedido HTTP de DELETE, de forma a apagar o par pretendido da base de dados, apaga-o, também, do Objeto *currentMessages*, relativo ao vocabulário do idioma selecionado atualmente.

O ícone varia de acordo com o estado de utilização atual do par Chave-Tradução, para isso é preciso definir o que significa um par Chave-Tradução estar, ou não, a ser utilizado. O significado de estar a ser utilizado é dependente do idioma principal, sendo que, para todos os pares de Chave-Tradução em falta, é adicionada uma linha na tabela, com a Chave correta e com a Tradução vazia, dado que um par estar em falta num determinado idioma significa que esse par de Chave-Tradução existe no idioma principal mas não existe nesse determinado idioma. Todos os pares que existem num certo idioma e que não existam no idioma principal são considerados como não estando a ser utilizados. Por esta última definição, se um par estiver a ser utilizado, se não existir Tradução para a Chave respetiva, considera-se que o par está em falta. De outra forma considera-se um par corretamente utilizado. Visualmente, para cada um destes estados, é demonstrado um ícone de forma a informar o utilizador.

O idioma principal pode ser alterado, para isso existe um menu de seleção que contém todos os idiomas disponíveis. Ao selecionar um novo idioma principal é executada uma função que altera a variável *main_language*, atualiza o atributo *fallbackLocale* do *i18n*, e realiza um pedido HTTP PUT para atualizar o idioma principal na base de dados.

De maneira a conseguir ter todos os valores dos elementos de input preenchidos com os valores atuais e de forma a utilizar os valores do input aquando da existência de uma alteração, foi necessário criar um Objeto reativo de Vue, *currentInput* e vinculá-lo ao valores dos elementos de input, utilizando a diretiva de Vue, *v-model*. À partida, é necessário povoar o objeto com os valores atuais de todos os pares Chave-Tradução, isto é feito com uma chamada à função *populateCurrentInput*, descrita no Extrato de Código 13.

```

1 for (let i = 0; i < idioms.length; i++) {
2   let vocabulary = getVocabulary(idioms[i].name);
3   for (let key in vocabulary) {
4     let finalKey = idioms[i].name + "." + key;
5     currentInput[finalKey] = vocabulary[key];
6   }
7 }

```

Extrato de Código 13: Função que preenche todos os inputs da tabela.

Cada valor dos elementos de input pode agora ser vinculado a cada valor correspondente deste objeto em cada linha da tabela utilizando a diretiva de Vue, *v-model*, como pode ser visto no Extrato de código 14.

```
1 v-model="currentInput[item.name + '.' + key]"
```

Extrato de Código 14: Utilização da diretiva de Vue, *v-model*.

Por fim, ao clicar no botão para confirmar a alteração do par Chave-Tradução, é executada a função *changeWord()*, demonstrada no Extrato de código 6.

Esta solução permite também a criação de palavras de pares Chave-Tradução no idioma principal. Isto é feito na primeira linha da tabela, e recebe dois valores em dois elementos de input diferentes, um referente à Chave e outro relativo à Tradução. Clicando no botão com a funcionalidade de criar o par, é executada a função *changeWord()*, apresentado no Extrato de código 6, devido ao facto de esta função ter um caso para lidar com um par que ainda não esteja presente na base de dados.

Sempre que o idioma principal é alterado a tabela é atualizada. A informação sobre a utilização de cada par Chave-Tradução é recalculada relativamente ao novo idioma principal e a tabela é reordenada para o idioma principal aparecer primeiro de forma a que a primeira linha, referente à criação de um par Chave-Tradução novo, não fique separada do idioma correspondente.

Solução Nº3 - Importar e Exportar Idiomas

Esta solução resume-se a providenciar uma alternativa para uma possível alteração de elementos textuais em maior escala. Para isso implentou-se a importação e exportação de ficheiros em formato JSON que contenham os dados relativos a um idioma, isto é nome, vocabulário e notícias.

É possível importar um ficheiro JSON, que tenha uma estrutura correta, isto é, que tenha, pelo menos, um atributo que seja uma string com duas letras, com o nome *name*. De forma a que o vocabulário do idioma seja corretamente armazenado é necessário que o ficheiro tenha um atributo, que seja um objeto, com o nome *vocabulary*. Se o ficheiro estiver corretamente estruturado e o idioma ainda não existir, é realizado um pedido HTTP POST para o url */Idioms*, que cria um documento na base de dados com nome, *name*, e vocabulário, *vocabulary*.

Em relação à exportação de idiomas em formato JSON, é feita uma chamada à função *exportJson()*, descrita no Extrato de código 15, que recebe como argumento um objeto correspondente ao idioma que se pretende exportar.

```

1 function exportJson(idiom) {
2   // converte o idioma numa String JSON
3   const json = JSON.stringify(idiom);
4
5   // cria um objeto semelhante a um ficheiro
6   const blob = new Blob([json], { type: 'application/
      json' });
7
8   // cria um objeto que referencia o objeto criado
9   const url = URL.createObjectURL(blob);
10
11  // cria um link de forma a que o evento .click()
      funcione
12  const a = document.createElement('a');
13  a.href = url;
14  a.download = idiom.name + '.json';
15  document.body.appendChild(a);
16
17  // começa o download do ficheiro
18  a.click();
19
20  //remove o link que ja nao e necessario
21  document.body.removeChild(a);
22  URL.revokeObjectURL(url);}

```

Extrato de Código 15: Função que exporta um ficheiro JSON relativo a um idioma.

De forma a implementar visualmente esta solução foi necessário criar uma pequena tabela que, para cada idioma mostra um botão que permite exportar o idioma. Adicionou-se também um botão e a respetiva funcionalidade de apagar um idioma por completo. Para isso é realizado um pedido HTTP DELETE que apaga o idioma contido no url *Idioms/"nome_do_idioma_a_apagar"*.

Nesta solução, e, devido às alterações realizadas de forma a garantir o cumprimento de requisitos, sempre que se cria ou elimina um idioma é necessário adicionar ou remover, respetivamente, o nome do idioma do array *idiom_names*.

Solução Nº4 - Formulário para adicionar novos idiomas

A última solução implementada passa por uma alternativa a importar um ficheiro JSON, para criar um novo idioma, e consiste num formulário onde se pode adicionar a quantidade de pares Chave-Tradução que se desejar.

O formulário espera receber o nome do idioma, que é verificado para ver se já existe e se tem exatamente 2 letras, e pelo menos um par Chave-Tradução. Mas é possível adicionar novos pares à medida que for necessário, de forma a isto ser conseguido implementou-se uma função em JavaScript que cria um novo

par de elementos de inputs um com a classe *key* e outro com a classe *value*, que recebe a Chave e a Tradução respectivamente, sendo que cada par tem a classe *pair*. Com todos estes novos pares de elementos de input é criado, também, um botão que permite remover os mesmos.

Para criar o idioma, existe no formulário um botão que quando clicado, executa a função *addIdiom()* que a partir do nome preenchido no formulário, e de todos os pares que tenham a classe *pair*, iterando sobre cada um deles de forma a construir o vocabulário, como é demonstrado no Extrato de código 16

```
1 pairs.forEach((pair) => {  
2     const key = pair.querySelector('.key').value;  
3     const value = pair.querySelector('.value').value;  
4     vocabulary[key] = value;  
5 }));
```

Extrato de Código 16: Procedimento que itera sobre todos os pares Chave-Tradução do formulário.

A partir do nome e do vocabulário, é construído um objeto referente ao novo idioma, que é passado como corpo do pedido HTTP POST para o url */Idioms*, de forma a criar o novo idioma.

Em semelhança à solução anterior, sempre que um idioma novo é criado, é adicionado também ao array *idiom_names*.

3.8 Funcionalidades Adicionais

Além das implementações desenvolvidas com vista a alcançar os principais objetivos do projeto, foram também implementadas algumas funcionalidades que pretendem colmatar possíveis falhas de rede.

3.8.1 Erro de Fetch

Considerando a possibilidade de existir uma falha no carregamento do idioma preferencial ou no carregamento de todos os idiomas, no caso das páginas que necessitam dos dados de todos os idiomas, foi implementada uma funcionalidade que indica ao utilizador que existiu um erro no carregamento. Isto é feito através de uma verificação da existência de dados na variável *currentMessages* e, no caso em que são necessários todos os idiomas, na variável *idioms*, ou seja se estas variáveis estiverem vazias, irá ser atribuído o valor de verdade à variável *errorOnFetch*, como pode ser observado no Extrato de código 17.

```
1 let errorOnFetch = Object.keys(currentMessages).length  
   === 0;  
2 //idioms em vez de currentMessages no caso de paginas  
   que necessitam de todos os idiomas
```

Extrato de Código 17: Procedimento que verifica a existência de algum erro no carregamento de idiomas.

É utilizada a diretiva de Vue *v-if* com o valor da variável *errorOnFetch* para mostrar num ecrã um aviso relativo a uma falha no carregamento dos idiomas, caso a variável *errorOnFetch* seja verdadeira ou nada é mostrado, caso contrário.

3.8.2 Indicador de carregamento

Tendo em conta a possibilidade da rede ser lenta e, por isso mesmo, o tempo de resposta aos pedidos realizados ao servidor ser maior, foi implementado um indicador de carregamento que é mostrado no ecrã enquanto ainda não estão disponíveis localmente os dados a serem mostrados ao utilizador.

Ao inicializar a aplicação, antes mesmo de ser mostrada a página principal da aplicação, é mostrado um ecrã em que apenas aparece um indicador de carregamento, enquanto todos os dados necessários para a página principal são carregados, neste caso, todos os pares Chave-Tradução relativos à linguagem principal. Este ecrã de carregamento é construído no ficheiro *index.html* sendo que se baseia num elemento *<div>* com um *id = "loading"*.

Criou-se também um componente Vue que funciona como uma camada intermédia, entre o componente principal *App.Vue* e os componentes mostradas de acordo com a página atual, com o objetivo de detetar se as respostas dos pedidos ao servidor já foram recebidas.

Relativamente ao caso em que o indicador de carregamento é mostrado no inicializar da aplicação, e sendo que o componente intermédio só é inicializado depois já existirem todos os dados necessários para mostrar ao utilizador a página principal, no momento em que o componente é inicializado, utilizando o método de Vue *onMounted()*, o elemento com *id="loading"* é removido.

Por isso, a necessidade da criação do componente intermédia prende-se com o facto da existência de outras situações em que é possível que seja necessário mostrar o indicador de carregamento, que são a mudança de idioma e respetivo carregamento dos pares Chave-Tradução relativos a esse idioma ou o carregamento dos pares Chave-Tradução de todos os idiomas. É neste componente que é detetado se já foi retornada a resposta do pedido ao servidor e enquanto não for é mostrado o indicador de carregamento utilizando a diretiva de Vue, *v-if*.

3.9 Validação

Foram realizados testes de desempenho de forma a perceber como é que a aplicação se comportava face a ter que efetuar o carregamento de um idioma com um número elevado de strings e como é que a aplicação lidava com este carregamento utilizando redes mais lentas.

Estes testes consistiram na medição do tempo de carregamento, ao inciar a aplicação, da sua página inicial, quando é feito o carregamento do idioma preferencial. Foi variado o número de notícias, com um tamanho constante em todos os casos, contidas no idioma preferencial de forma a variar o número de strings carregado, e a velocidade da rede sendo que foram realizados 3 ensaios

para cada par destas variáveis. De forma a variar a velocidade utilizaram-se as ferramentas de rede do Firefox⁹.

Os resultados dos testes realizados, demonstrados na Figura 7, mostram que para redes rápidas, acima de 4.0 Mbps, o tempo de carregamento é praticamente independente do número de strings carregadas. Para redes mais lentas o tempo de carregamento já é mais significativo, sendo que até às 200 notícias carregadas, o número de strings continua a não influenciar o tempo de carregamento. A partir das 1000 notícias carregadas o tempo de carregamento aumenta significativamente para redes mais lentas, verificando-se, também, durante a realização dos testes, que a renderização da página principal apresentava algumas quebras.

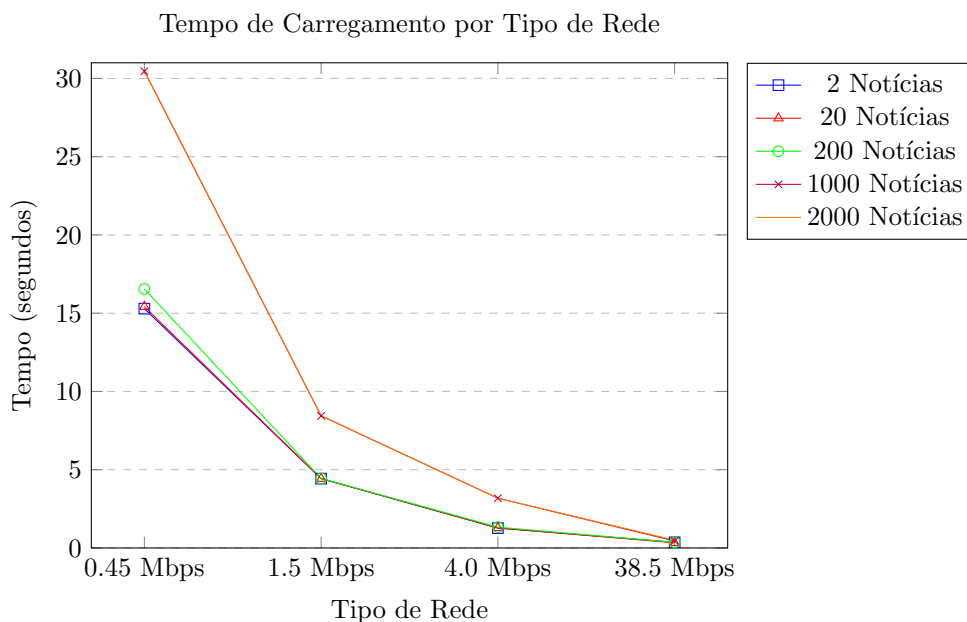


Figura 7: Tempo de carregamento da página principal da aplicação em diferentes tipos de redes para diferentes números de strings do idioma preferencial.

4 Conclusões

Esta última secção serve como reflexão sobre o trabalho realizado, enunciando também algumas ideias sobre trabalho futuro no contexto deste projeto.

⁹https://firefox-source-docs.mozilla.org/devtools-user/network_monitor/throttling/index.html

4.1 Resultados alcançados

Todos os objetivos propostos para este projeto foram alcançados com sucesso, sendo que foi implementada uma solução que permite a mudança de idioma sem existir a necessidade da aplicação ser recompilada, solução esta que se encontra descrita na Subsecção 3.7.1. Foram, também, implementadas quatro soluções diferentes que cumprem o objetivo de permitir ao utilizador editar todos os textos da aplicação, em qualquer idioma, soluções que são detalhadas na Subsecção 3.7.2. Além disto, e de forma a ter uma aplicação completa, que funciona como prova de conceito das soluções implementadas, foi criada uma base de dados e foi desenvolvida uma API REST que realiza a conexão entre a aplicação e a base de dados.

4.2 Lições aprendidas

Este projeto mostrou que é possível o desenvolvimento de uma aplicação em Vue em que alteração de idioma ocorra de forma dinâmica e também que, além de ser possível, existem várias possibilidades de implementar a edição dinâmica de traduções.

A realização deste projeto proporcionou, também, a possibilidade de trabalhar num contexto empresarial, o que foi uma nova e enriquecedora experiência.

4.3 Trabalho futuro

Ao longo do projeto surgiram ideias que não se materealizaram, devido a não serem consideradas uma prioridade e, também, face às restrições temporais deste projeto. A existir continuidade neste projeto, as ideias a implementar seriam:

- Autenticação - Apenas utilizadores com as permissões corretas poderiam ter acesso às páginas relativas à criação e edição de pares Chave-Tradução e por sua vez, só esses utilizadores poderiam criar/editar pares Chave-Tradução.
- Edição de palavras em formato JSON na própria aplicação - Solução adicional para resolver o problema proposto relativo a alteração dinâmica de textos e que consistiria na possibilidade do utilizador editar os pares Chave-Tradução de um idioma num editor de texto JSON embutido na aplicação. Eliminando assim a necessidade do utilizador exportar e/ou importar um idioma de forma a realizar alterações.
- Sugestão de tradução gerada por inteligência artificial - Funcionalidade que sugeriria uma possível Tradução para a Chave do par Chave-Tradução, permitindo ao utilizador decidir se a sugestão ia, ou não, de encontro com a tradução que o utilizador procurava.
- Suporte para tradução de mensagens interpoladas, ou seja, permitir que as strings utilizadas na aplicação contenham variáveis. Isto levantaria

alguns problemas com resolução possível, como lidar com a pluralização de palavras e com o formatação de números.

- Desenvolvimento de uma aplicação que extraísse as strings contidas de uma aplicação web, de forma a ser possível a utilização de algumas das soluções implementadas neste projeto a qualquer aplicação web sem necessidade de adaptar a aplicação web.

Referências

- [1] MongoDB Compass. Mongodb compass documentation. <https://www.mongodb.com/products/tools/compass>, 2024.
- [2] Express.js. Express documentation. <https://expressjs.com/>, 2024.
- [3] i18nNext. i18n documentations. <https://www.i18next.com/>, 2024.
- [4] MongoDB. Mongodb documentation. <https://www.mongodb.com/>, 2024.
- [5] Vue.js. Vue documentation. <https://vuejs.org/>, 2024.