Universidade do Minho

Mestrado Integrado de Engenharia Informática
**Laboratórios de Informática III**

# UMYELP - PROJETO C

(MiEI-LI3 20/21)

GRUPO Nº25:


- Joana Maia Teixeira Alves (A93290)

- Maria Eugénia Bessa Cunha (A93264)

- Vicente Gonçalves Moreira (A93296)


Data de Entrega:

02-05-2021

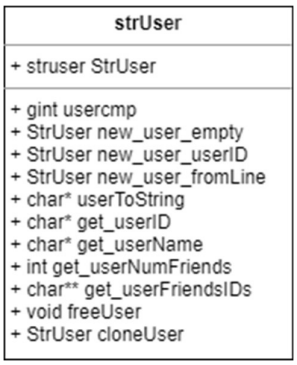
# ÍNDICE

# DESCRIÇÃO DAS APIs

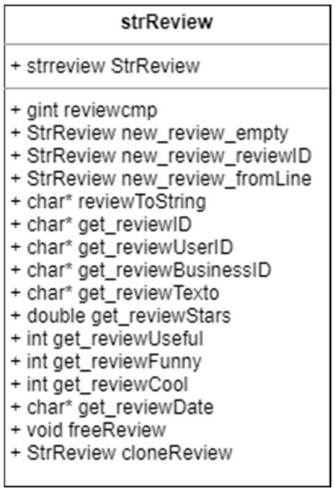
## FUNCAUX.H

Este módulo contém uma função auxiliar "fgetsdinamico" que permite a leitura de linhas de ficheiros de qualquer dimensão. Para além desta, contém a API para a estrutura "StringL". Esta é uma estrutura simples que armazena uma lista de *strings* alocando memória dinamicamente quando atinge a capacidade total.

## STRUSER.H - STRBUSINESS.H - STRREVIEW.H




```
/**
 * Estrutura de dados de Users
 */
typedef struct struser* StrUser;
```

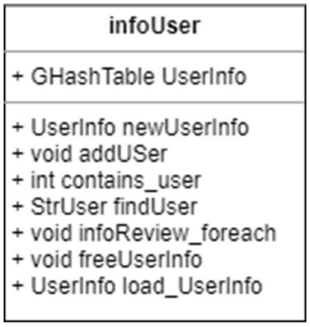



```
/**
 * Estrutura de dados de Review
 */
typedef struct strreview* StrReview;
```

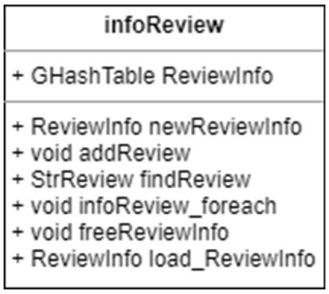

```
/**
 * Estrutura de dados de Business
 */
typedef struct strbusiness* StrBusiness;
```

APIs que manipulam as estruturas de dados *strBusiness*, *strUser* e *strReview*. Estas estruturas servem para armazenar a informação sobre um negócio/usuário/review, respetivamente. Há três opções de inicialização da estrutura: por omissão, recebendo um ID e/ou a partir de uma string formatada usando delimitadores. Estas usam principalmente *strings* como forma de armazenar a informação.

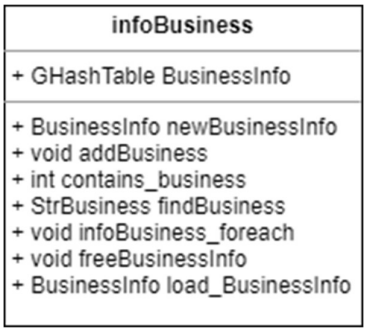
## INFOUSER.H - INFOBUSINESS.H - INFOREVIEW.H




```
/**
 * HashTable Completa de Utilizadores
 */
typedef GHashTable* UserInfo;
```




```
/**
 * HashTable Completa de Reviews
 */
typedef GHashTable* ReviewInfo;
```




```
/**
 * HashTable Completa de Negócios
 */
typedef GHashTable* BusinessInfo;
```

APIs que manipulam as estruturas de dados *BusinessInfo, UserInfo* e *ReviewInfo*. Estas estruturas servem para listar a informação sobre todos os negócios/usuários/reviews presentes num ficheiro.

Todas as estruturas foram implementadas utilizando como tipo de dados a *GHashTable* proveniente da biblioteca *glib* aconselhada pelos docentes. Decidimos escolher este tipo de dados dada a eficiência no que toca a inserção e procura de elementos.

Aplicamos métodos de inicialização, *free*, inserção, procura, *foreach* e de *load* a partir de um ficheiro, todas elas recorrendo a funções já implementadas na biblioteca *glib*.

## STATS.H

Módulo auxiliar ao "sgr". Este contém o número total de negócios, usuários e *reviews* lidos, também como duas estruturas designadas "Index", sendo uma especializada para listar os *users* e as suas respetivas *reviews* e a outra lista *businesses* e as suas respetivas *reviews*.

Estas estruturas são compostas por uma lista *GSequence* que armazena estruturas do tipo "DadToSons". Esta estrutura auxiliar genérica armazena uma *string* principal designada por *father* e um conjunto de *strings* "filhos".


## SGR.H

Módulo principal (Sistema de Gestão de Recomendações) do MODELO.

Este é responsável por toda a manipulação dos dados e pela execução das *queries*. Para além da assinatura requerida, adicionamos sete funções de controlo.

## QUERYARG.H

Módulo auxiliar ao "sgr". Foi criado devido à função pré-definida do *glib* para percorrer *GHashTable* (g_hash_table_foreach), dado que esta, para além da função a ser executada aos valores, apenas aceita um argumento como informação extra, encontramos então a necessidade de encapsular essas informações numa estrutura. (ex.: inteiros, *strings*, TABLE).

## TABLE.H

Módulo genérico para armazenamento de informação.

Para esta estrutura, decidimos que as *strings* seriam a forma mais genérica para guardar informação. Usando como auxílio as estruturas "StringL", estas guardam a informação ao longo de uma linha, utilizando depois uma lista destas estruturas para representar as várias linhas.

Também registamos um dado "infoLine" que serve para guardar informação sobre o conteúdo da TABLE.

## PAGINACAO.H

Módulo pertencente ao VISUALIZADOR, responsável pela apresentação do tipo de dados TABLE. Esta ajusta dinamicamente o tamanho das colunas, de modo a manter o alinhamento do conteúdo. Caso ultrapasse o limite definido das colunas, encurta o conteúdo apresentado.

## VISUALIZADOR.H

Módulo responsável pela apresentação de mensagens do programa ao utilizador. Guarda os menus de entrada, saída, ajuda, execução, entre outros. Permite também a apresentação de mensagens customizadas.

## VARIÁVEIS.H

Módulo auxiliar do CONTROLADOR. Este é responsável pela inserção/remoção/controlo das variáveis do programa em execução. Esta usa o tipo de dados *GHashTable,* sendo a *key* o nome da variável e o *value* a TABLE correspondente.


## INTERPRETADOR.H

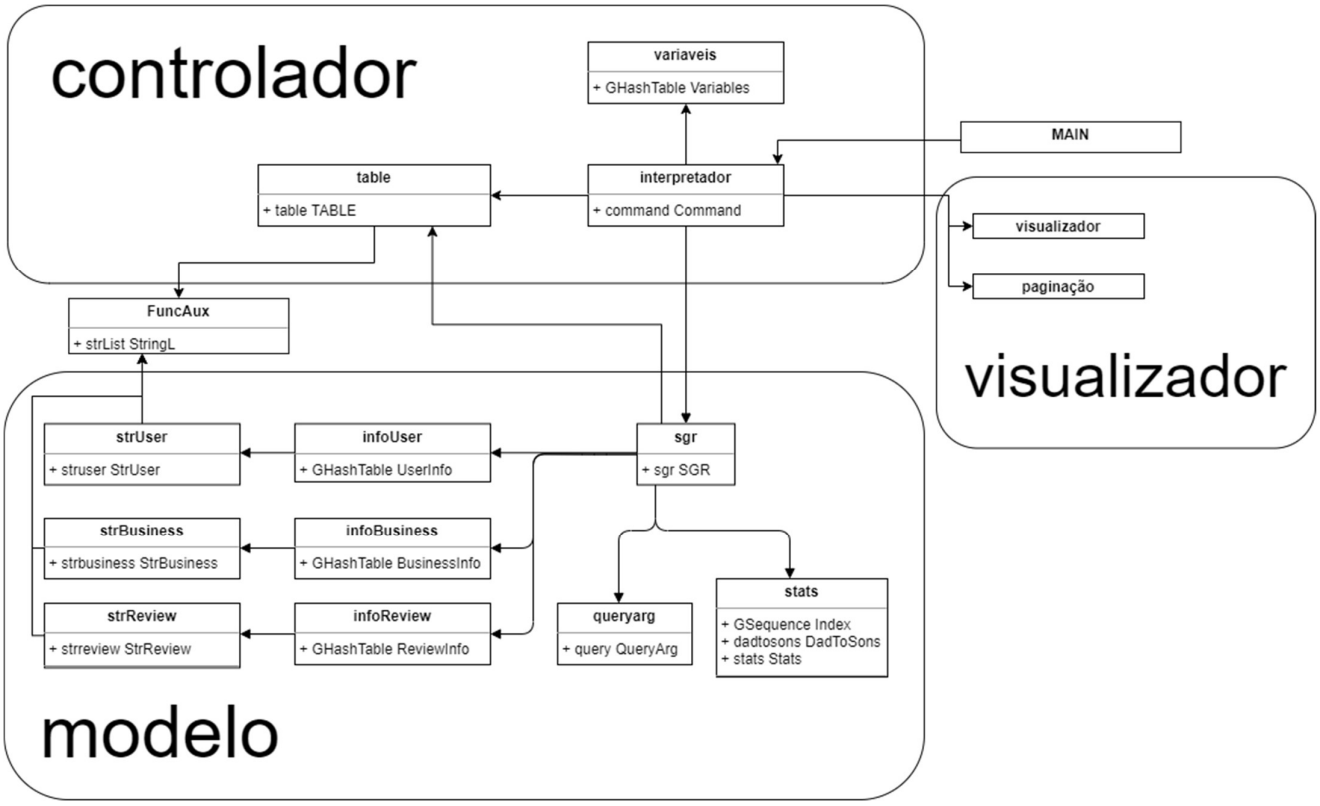
Módulo principal do CONTROLADOR.

É responsável pela descodificação dos *inputs* do utilizador, utilizando várias funções.

O trajeto de um *input* é o seguinte:

- Comando é escrito na *run*;
- Este é dividido, interpretado e validado pela *interpreta_linha*, com a ajuda das funções *interpreta_função*, *quantos_args* e *verifica_args*;
- Depois de validado segue para a função *executa_função*. Se o *input* corresponder a uma *query*, a execução desta é redirecionada para a função *executa_query*.

# ARQUITETURA FINAL DA APLICAÇÃO




# COMPLEXIDADE E OTIMIZAÇÕES DAS ESTRUTURAS

No início deste projeto, ao analisar o catálogo de estruturas do *glib*, decidimos que para listar as nossas estruturas de *users/negócios/reviews* iríamos utilizar o *data type* "g_ptr_array" (array de *pointers*), pois continha funções de *sort* que iriam ser úteis e a possibilidade de procura binária com uma simples função extra não nativa ao *glib*.

No entanto, esta simples função não correspondeu às expectativas. Assim, decidimos mudar o *data type* para *GSequence,* pois, para além de indicar que era útil para *scalable lists*, continha todas as funções que necessitávamos.

Depois de implementado, ao testarmos a sua performance na inserção de dados ordenados, os resultados foram desanimadores dado que este demorava cerca de 30 segundos. Consequentemente, mudamos o *data type* novamente, escolhendo desta vez a *GHashTable*. Esta escolha provou ser importante para a performance geral do programa.

O tipo de dados TABLE sofreu duas alterações ao longo do projeto. Este inicialmente era constituído apenas por uma matriz de *strings* (*char****), mas dado a complexidade de declaração e manipulação deste tipo de dados, alteramos para uma lista simples de *strings*, sendo os conteúdos das colunas divididas pelo delimitador ';'. Esta abordagem mostrou ser simples, mas ineficiente na manipulação de elementos individuais pois éramos obrigados a ler uma linha inteira. Entretanto, foi desenvolvido a estrutura "StringL" que permitia a manipulação simples de *strings* numa lista, de modo que decidimos implementá-la no tipo TABLE, ficando esta definida como uma lista de "StringL".

Outras pequenas alterações incluem o abandono do *data type GArray*, utilizado na altura para o armazenamento dos *IDs* dos amigos do *user* e das categorias dos negócios, passando a utilizar o tipo "StringL". A *utility GDateTime* foi também abandonada a favor de uma *string* simples dado o comportamento errático na alocação e libertação de memória.

# ESTRATÉGIAS E OTIMIZAÇÕES DAS QUERIES

- **QUERY 1 –** Nesta *query* recorremos à leitura dos ficheiros com a ajuda do "fgetsdinamico". Monta as várias estruturas, validando-as, e de seguida insere estas nas listas correspondentes (*infoUser, infoBusiness, infoReview*). Depois de montadas as listas principais procedemos à construção dos "Index" auxiliares.
- **QUERY 2 –** Percorre a lista de negócios e recolhe (insere na TABLE) os que contêm a letra inicial pedida (*case insensitive*).
- **QUERY 3 –** Efetua uma procura do negócio na lista respetiva, devolvendo a sua informação.
- **QUERY 4 –** Utilizando o "Index" *users->reviews*, começamos por procurar o respetivo *user* nesse Index. De seguida, para cada *review* desse utilizador, encontramos o negócio correspondente (primeiro encontrar a review na lista, ler o *ID* do negócio e procurá-lo) e recolhemos a sua informação.
- **QUERY 5 –** Utilizando o "Index" *businesses->reviews*, para cada negócio do Index, começamos por verificar se a sua cidade corresponde à pedida. Caso haja correspondência, utiliza a função *calcula_stars* (que a partir dos *reviews* do negócio faz a sua média) e verifica se é maior ou igual ao valor pedido. Se ambas as condições se verificarem, recolhe as informações necessárias.
- **QUERY 6 –** Sendo uma expansão da *query* 8, esta armazena numa *GSequence* os negócios de uma cidade ordenados de forma decrescente das estrelas e vai inserindo numa *GHashTable* estas *GSequences* identificadas pela sua cidade. Por fim, depois de todas as cidades inseridas, apenas são recolhidos os primeiros **n** negócios de cada cidade.
  EXTRA: dado a performance lenta desta *query*, tivemos a ideia de criar um novo Index *city->business*, no entanto, não tivemos tempo de implementar.
- **QUERY 7 –** Utilizando o "Index" *users->reviews*, percorre os vários *users* descartando de imediato os que não têm mais do que uma *review*. Caso contrário, analisa a primeira *review* e obtém o estado do negócio (através do *ID* do negócio da *review*), percorrendo de seguida as restantes *reviews* obtendo os estados através do mesmo método, procurando diferenças entre estes e, se existirem, recolhe a informação do user.

- **QUERY 8 –** Utilizando o "Index" *businesses->reviews,* percorre os vários negócios e caso contenham a categoria desejada calcula as estrelas do mesmo e insere numa estrutura temporária (*busID* e *stars*) colocando-a de seguida numa *GSequence*. Depois de vistos todos os negócios, a *GSequence* é ordenada de forma decrescente de estrelas. Por fim, apenas são recolhidos os primeiros **n** negócios.
- **QUERY 9 –** Percorre a lista de *reviews* e testa se o mesmo contém a palavra pedida no seu texto. Se a palavra for encontrada, recolhe a informação.

# RESULTADO DOS TESTES REALIZADOS

| TESTES | TEMPO MÉDIO |
|---|---|
| LOAD INICIAL | 10,66 s |
| QUERY 1 - FREE & LOAD | 14.59 s |
| QUERY 2 | 56,74 ms |
| QUERY 3 | 129,2 micros |
| QUERY 4 | 5,89 ms |
| QUERY 5 | 334,36 ms |
| QUERY 6 | 8,86 s |
| QUERY 7 | 2,1 s |
| QUERY 8 | 293,57 ms |
| QUERY 9 | 587,10 ms |