Relatório de Projeto

Filipe Dominguez dos Santos

Engenharia Informática

nov | 2023

GUARDA
POLI
TÉCNICO

POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

Text-to-Speech (TTS) - *Cache* de ficheiros de áudio gerados

RELATÓRIO DE ESTÁGIO

PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA INFORMÁTICA

Filipe Dominguez dos Santos

Novembro / 2023

Escola Superior de Tecnologia e Gestão

Text-to-Speech (TTS) - *Cache* de ficheiros de áudio gerados

RELATÓRIO DE ESTÁGIO

PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA INFORMÁTICA

Professor Orientador: José Fonseca

Filipe Dominguez dos Santos

Novembro / 2023

Agradecimentos

Em primeiro lugar, quero agradecer aos meus pais, irmã e namorada que são o principal pilar do meu percurso académico. Quero agradecer a todos aqueles que fizeram parte de toda a minha formação, aos meus amigos e professores.

Quero ainda agradecer a toda a equipa da Altice Labs pela rapidez em me recrutar e pelo entusiasmo e apoio que me deram desde o princípio ao fim. Quero deixar um agradecimento especial ao Virgílio Cunha que, apesar de não ser o meu orientador dentro da empresa, deu-me um apoio enorme em qualquer dúvida que tivesse. E, não menos importante, ao Fernando Delfim pelas suas palavras e ajuda constante que me motivaram ao longo de todo o estágio.

Ao meu orientador de estágio, Professor José Carlos Fonseca, pela disponibilidade e conselhos dados ao longo da escrita deste relatório.

A todos, o meu sincero obrigado!

Ficha de Identificação

Aluno

Nome: Filipe Dominguez dos Santos

Número: 1702072

Licenciatura: Engenheira Informática

Estabelecimento de Ensino

Escola Superior de Tecnologia e Gestão (ESTG) – Instituto Politécnico da Guarda (IPG)

Morada: Av. Dr. Francisco Sá Carneiro 50, 6300-559 Guarda

Telefone: 271 220 100

Entidade Acolhedora do Projeto em Contexto de Estágio

Nome: Altice Labs

Morada: Rua Eng. José Ferreira Pinto Basto 3810-106 Aveiro

Contacto: 234 403 200

Duração do Projeto em Contexto de Estágio

Inicio: 17 de julho de 2023

Fim: 16 de outubro de 2023

Supervisor de Estágio (Entidade de Acolhimento)

Nome: Fernando Delfim

Docente Orientador de Estágio

Nome: José Carlos Fonseca

Grau Académico: Doutor

Resumo

A Altice Labs tem se focado na criação de uma nova geração de serviços de Interactive Voice Response (IVR), com o uso de inteligência artificial para permitir a adaptação em tempo real das interações com os clientes. Um elemento fundamental desses serviços é a funcionalidade de Text-to-Speech (TTS), que é amplamente usada para fornecer informações em resposta às solicitações dos clientes imitando uma interação humana.

O objetivo deste projeto é resolver os desafios associados ao uso de TTS em tempo real, em comparação com a utilização dos tradicionais anúncios pré-gravados. Os problemas residem no atraso percetível e nos custos associados à síntese em tempo real. Neste projeto foi desenvolvido um componente inovador que minimiza essas desvantagens, através da implementação de uma *cache* de arquivos de áudio previamente processados.

Com a utilização desta *cache*, os serviços continuam a solicitar a síntese de texto em tempo real, mas o que é efetivamente reproduzido é um anúncio que foi previamente gerado e guardado na *cache* como resultado de uma síntese anterior. Esta abordagem proporciona uma resposta mais rápida e eficiente, enquanto, simultaneamente, reduz os custos associados à síntese em tempo real.

No âmbito deste projeto foi implementada uma cache e conduzidas avaliações para analisar os benefícios desta abordagem, incluindo a redução de atrasos e custos operacionais. O trabalho foi desafiador devido à complexidade técnica envolvida na criação da *cache* de áudio em formato de *hash table* para os serviços de TTS já existentes. Além disso, foi necessário garantir que a solução fosse robusta e escalável para atender à procura crescente de interações de voz em tempo real. A conclusão bem-sucedida deste projeto representará um avanço significativo na capacidade da Altice Labs em fornecer serviços de IVR aprimorados e económicos à empresa.

Palavras-Chave: TTS, IVR, Hash Table, cache

Abstract

Altice Labs has focused on creating a new generation of Interactive Voice Response (IVR) services, using artificial intelligence to enable real-time adaptation of customer interactions. A key element of these services is TTS functionality, which is widely used to provide information in response to customer requests, imitating human interaction.

The goal of this project is to solve a challenge associated with using real-time TTS, compared to using pre-recorded announcements. The problems lie in the noticeable delay and costs associated with real-time synthesis, compared to the pre-recorded announcements. The project seeks to develop an innovative component that minimizes these disadvantages, creating a cache of audio files.

With the use of this cache, services continue to request real-time text synthesis, but what is actually played is an ad that was previously generated and stored in the cache as a result of a previous synthesis. This approach provides a faster and more efficient response, while reducing the associated costs of real-time synthesis.

As part of this project, a cache will be implemented and evaluations will be conducted to assess the benefits of this approach, including the reduction of delays and operational costs. The work is challenging due to the technical complexity involved in creating the audio cache in Hash Table format in existing TTS services. Additionally, it needs to ensure that the solution is robust and scalable to meet the growing demands of real-time voice interactions. The successful completion of this project will represent a significant advancement in Altice Labs ability to provide enhanced and cost-effective IVR services to the company.

Keywords: TTS, IVR, Hash Table, cache

Índice

Agradecimentos i

Ficha de Identificação iii

Resumo v

Abstract vii

Índice ix

Índice de Figuras xi

Índice de Tabelas xiii

Índice de Listagens xv

Lista de Siglas xvi

1 Introdução 1

1.1 Motivação e Enquadramento 1

1.2 Caraterização sumária da Altice Labs 2

1.3 Descrição do problema 2

1.4 Objetivos 3

1.5 Estrutura do Relatório 3

2 Estado da Arte 5

2.1 O que é *cache* 5

2.2 Algoritmos de pesquisa na *cache* 5

2.2.1 Pesquisa Linear 6

2.2.2 Hash Table 7

3 Metodologia 11

3.1 Metodologia *Waterfall* 11

3.2 Aplicação da metodologia no projeto 12

4 Análise de requisitos 15

4.1 Requisitos funcionais 15
4.2 Diagrama de casos de uso 15
4.3 Descrição dos casos de uso 16
4.4 Diagrama de Fluxo 18
5 Tecnologias 21
6 Implementação 23
6.1 Resolução de requisitos 23
6.2 Hash Table 26
6.2.1 SDBM 27
6.2.2 JDB2 27
6.2.3 Construção da *Hash Table* 29
7 Verificação e Validação 37
8 Conclusão 43
9 Bibliografia 44
10 Anexos 49

Índice de Figuras

Figura 1 Pesquisa Linear 7

Figura 2 Hash Table 8

Figura 3 Linked Lists 9

Figura 4 Representação Modelo Waterfall 11

Figura 5 Diagrama de Caso de Uso 16

Figura 6 Diagrama de Fluxo da Cache 19

Figura 7 Testes Colisão de Algoritmos 28

Figura 8 Diagrama de funcionamento 29

Figura 9 Inicialização do serviço 38

Figura 10 Pedidos de síntese 39

Figura 11 PhonerLite Pedido de Síntese 39

Figura 12 Primeiro Pedido de Síntese (Serviços da Google) 40

Figura 13 Terceiro Pedido de Síntese (Serviços da Google) 40

Figura 14 Quarto Pedido de Síntese (Encontrado em Cache) 40

Figura 15 Cache Persistente 41

Figura 16 Novo Pedido de Síntese 41

Índice de Tabelas

Tabela 1 Caso de uso: Ativação de Cache 17

Tabela 2 Caso de uso: Guardar todos os parâmetros da síntese 17

Tabela 3 Caso de uso: Defenir localização cache persistente 18

Tabela 4 Caso de uso: Definir tamanho da cache 18

Índice de Listagens

Listagem 1 Algoritmo SDBM 27

Listagem 2 Algoritmo JDB2 28

Listagem 3 Construtor Hash Table 30

Listagem 4 Definição de Parâmetros 31

Listagem 5 Cache Persistente 31

Listagem 6 Formatação ficheiro WAV 32

Listagem 7 Limpeza da Cache 33

Listagem 8 Construtor Linkedlist 33

Listagem 9 Construtor cacheitem 34

Listagem 10 Garantia MD5 35

Lista de Siglas

CSV	Comma-separated Values
HTML	Hypertext Markup Language
IP	Internet Protocol
IVR	Interactive Voice Response
JSON	JavaScript Object Notation
MD5	Message Digest Algorithm 5
PAAS	Platform-as-a-Service
RAM	Random Access Memory
SQL	Structured Query Language
STT	Speech To Text
TTS	Text To Speech
VOIP	Voice Over Internet Protocol
WAV	Waveform Audio File Format
WSL	Windows Subsystem for Linux

1 Introdução

Este relatório tem como objetivo descrever o desenvolvimento do projeto em contexto de estágio realizado na empresa Altice Labs no âmbito da unidade curricular de Projeto de Informática. O projeto consiste na implementação de uma *cache* nos serviços automáticos telefónicos que usam Text to Speech (TTS).

1.1 Motivação e Enquadramento

A Altice Labs possui um Media Server IP genérico em *cloud* no seu portefólio, oferecendo uma ampla variedade de recursos avançados de processamento de diferentes tipos de mídia, tais como reprodução e gravação de anúncios, reconhecimento de fala e TTS.

No início dos serviços telefónicos, eram pré-gravados ficheiros de áudio, também conhecidos como guias vocais pré-gravados. Estes ficheiros são sínteses de frases previamente gravadas e que de forma linear guiam o cliente ao longo da chamada. Esta solução, apesar de resolver o problema primário de resposta ao cliente, não permite uma comunicação assertiva e detalhada, uma vez que os guias vocais pré-gravados permanecem inalterados ao longo da chamada.

Os guias vocais pré-gravados têm sido substituídos pelo TTS, que é uma tecnologia que tem tido uma forte evolução e adoção nos últimos anos dentro da Altice Labs, estando cada vez mais presente tanto em pequenos como grandes projetos (Saber-e-Fazer, 2003). De facto, uma das áreas em que o TTS é mais utilizado é nos serviços automáticos telefónicos, pois facilita a atribuição de áudio mais específico, sem quaisquer erros, e atendendo às necessidades de comunicação de cada cliente.

No entanto, a tecnologia TTS tem algumas desvantagens face à utilização de guias vocais pré-gravados que precisam de ser mitigadas. É precisamente na mitigação dessas desvantagens que se insere o presente projeto em contexto de estágio.

Sendo o meu último ano de licenciatura era importante adquirir competências profissionais e experienciar o mundo do trabalho. A Altice Labs, uma empresa com vários anos de experiência, ofereceu-me essa oportunidade com a atribuição deste

projeto que me permitiu desenvolver e trabalhar numa tecnologia crescente e especialmente útil para a empresa.

1.2 Caraterização sumária da Altice Labs

A origem da Altice Labs remonta a 1950, cujo nome da empresa era GECA. Na altura foi criada uma equipa de engenharia para desenvolver internamente a mais avançada tecnologia de telecomunicações da época. As capacidades foram alargadas a um ecossistema dinâmico de parceiros, indústrias e universidades na Europa, com o objetivo de transformar conhecimento em valor na forma final de produtos e serviços inovadores (História Altice Labs, 2023).

Em 1972 o GECA passa a ser CET (Centro de Estudos de Telecomunicações) com uma finalidade adicional: escolas. Foi também necessário requalificar todos os técnicos e engenheiros para uma nova tecnologia: eletrónica.

Foi em 1999 que o CET passou a ser PT Inovação, e que mais tarde, em 2016 passa a ser Altice Labs, o centro de pesquisa, desenvolvimento e inovação para o grupo Altice.

Altice Labs foca-se no desenvolvimento de produtos e serviços inovadores para o mercado das telecomunicações e das tecnologias de informação. Promove a cooperação com as universidades e outros institutos nacionais e internacionais, assumindo-se como um verdadeiro agente da transferência do conhecimento para o mercado e para a indústria. A Altice Labs assume-se como motor e agente de transformação e inovação tecnológicas, visando melhorar e facilitar a vida das pessoas e das empresas.

1.3 Descrição do problema

O uso do TTS apresenta duas importantes desvantagens em comparação com a utilização de guias vocais pré-gravados: atrasos na resposta ao cliente e custos mais elevados. De facto, os anúncios pré-gravados são significativamente mais rápidos, já que não necessitam da utilização de serviços externos, como é o caso dos serviços de fala da Google, que tem de ser acedidos quando se usa TTS. Por sua vez, a utilização de serviços externos acarreta um custo extra por cada caractere do texto, algo que não está presente no caso dos anúncios pré-gravados.

1.4 Objetivos

O objetivo deste projeto consiste no desenvolvimento de uma solução que melhore os atrasos e os custos associados à utilização do TTS. Esta solução é baseada na criação de uma *cache* de ficheiros de áudio que permite ao TTS reduzir a necessidade de síntese de novos ficheiros de áudio, possibilitando a resolução simultânea dos dois inconvenientes do uso de TTS.

Os serviços automáticos telefónicos continuam a solicitar que determinado texto seja sintetizado em tempo real pelo TTS, mas com a introdução desta *cache* o que é efetivamente tocado é um anúncio que ficou guardado na *cache* na sequência de uma síntese anterior. A pesquisa deste ficheiro na *cache* será muito mais rápida do que a necessidade de fazer síntese de voz, e, simultaneamente de custos reduzidos, dado que, não são utilizados os serviços de fala da Google, que têm de ser contratualizados. De referir, que para a construção da *cache* será necessária uma procura de algoritmos de pesquisa na *cache* que atendam à rapidez de resposta esperada por um serviço automático telefónico e que também tenham em consideração os recursos presentes nas máquinas da Altice Labs em que o TTS já está implementado. Para além disso, será necessário fazer uma análise prévia do serviço de TTS já implementado na Altice Labs para garantir um desenvolvimento fluido e que não prejudique o código previamente desenvolvido.

1.5 Estrutura do Relatório

O presente relatório está estruturado em 8 capítulos, que descrevem as diferentes etapas do processo de desenvolvimento do projeto proposto. Os capítulos que se seguem a esta introdução são os seguintes:

- O capítulo 2 apresenta o estado de arte, nomeadamente, o estudo de diferentes abordagens para a implementação de uma *cache*;
- O capítulo 3 consiste na descrição da metodologia usada no projeto;
- No capítulo 4 é apresentada a análise de requisitos feita no projeto;
- No capítulo 5 são abordadas as tecnologias necessárias para o projeto;
- O capítulo 6 descreve a implementação e resolução dos requisitos propostos;

- No capítulo 7 é explicado como é feito o processo de compilação e execução do código feito, juntamente com a sua validação;
- O capítulo 8 consiste na conclusão do projeto.

2 Estado da Arte

O estado da arte define-se como o estudo da informação para a realização de um projeto de maneira a identificar qual o estado de conhecimento sobre o tema (Estado da Arte, 2003). Neste capítulo é feita uma descrição sobre o que é *cache*, a sua importância, as várias alternativas de construção e ainda uma análise objetiva dos vários tipos de algoritmos de pesquisa usados na mesma.

2.1 O que é *cache*

Cache, no contexto apresentado, é um componente de *software* que armazena dados para que futuros dados solicitados possam ser entregues rapidamente. Os dados em *cache* resultam de um processamento de dados anteriores ou de uma cópia de dados armazenados. Os dados armazenados, conhecidos como *cache* persistente, desempenham o papel de preservar os dados em disco. Já a *cache* em memória, opera na RAM, que é um tipo de memória mais rápida do que a de disco, mas é volátil, ou seja, é eliminada quando o sistema é reiniciado. A junção da *cache* persistente e da *cache* em memória garante que nenhum dado é perdido e ao mesmo tempo assegura rapidez na resposta. Para que a *cache* em memória esteja de acordo com a *cache* em disco, é executado o processo de cópia da *cache* em disco para memória sempre que o sistema é reiniciado. Para que as vantagens da *cache* sejam atendidas é necessário um bom algoritmo de construção e pesquisa na *cache* (SMITH, 1982).

2.2 Algoritmos de pesquisa na *cache*

Ao procurar por "*Sorting and Searching Algorithms*" na plataforma Google, foram encontradas alternativas que demonstram diferentes abordagens de pesquisa na *cache*. De seguida são descritas duas alternativas, sendo que uma é a melhor abordagem e a outra é uma má abordagem para o problema apresentado, demonstrando o quão importante é a escolha de um algoritmo de pesquisa na *cache*.

Para demonstrar a eficácia de cada algoritmo, será usada a métrica "complexidade de tempo", juntamente com o gráfico da relação entre o número de elementos e o tamanho da matriz.

A complexidade de tempo de um algoritmo quantifica a porção de tempo usada por um algoritmo para completar uma iteração em função do tamanho da entrada do problema. A complexidade de tempo de um algoritmo é frequentemente expressa usando a notação *big* O (*The Design and Analysis of Computer Algorithms*, 1974).

Existem vários tipos de complexidade de tempo, mas os usados neste projeto vão ser, tal como referido, o melhor caso O(1) e o que representa uma má abordagem O(n).

Um algoritmo tem tempo constante ou O(1) quando o tamanho do *input* não interfere na sua resposta, garantindo sempre o tempo que o *software* demora a completar apenas uma iteração independentemente.

Tem tempo linear ou O(n) quando o tamanho do *input* é diretamente proporcional ao tempo de resposta, ou seja, o *software* terá um tempo de resposta igual ao tamanho do *inputs* multiplicado pelo tempo que demora a completar uma iteração.

2.2.1 Pesquisa Linear

O termo de pesquisa linear expressa um tipo de pesquisa em listas, vetores ou matrizes de modo sequencial, ou seja, a comparação é feita de elemento a elemento, até ser encontrada a solução. A grande desvantagem da pesquisa linear é o tempo que demora, já que cresce proporcionalmente com o número de elementos (Knuth, 1998).

Na melhor das hipóteses, a busca é concluída na primeira tentativa, no entanto, na pior hipótese, o elemento a ser procurado encontra-se na última posição. Isto, numa lista de milhões de elementos, pode demorar minutos a ser encontrado, o que seria incomportável num serviço telefónico que tem de ter um tempo de resposta perto do imediato.

Como podemos observar na Figura 1 concluímos que a complexidade de tempo da pesquisa linear é igual a O(n) ou, como referido anteriormente, um algoritmo com tempo linear.

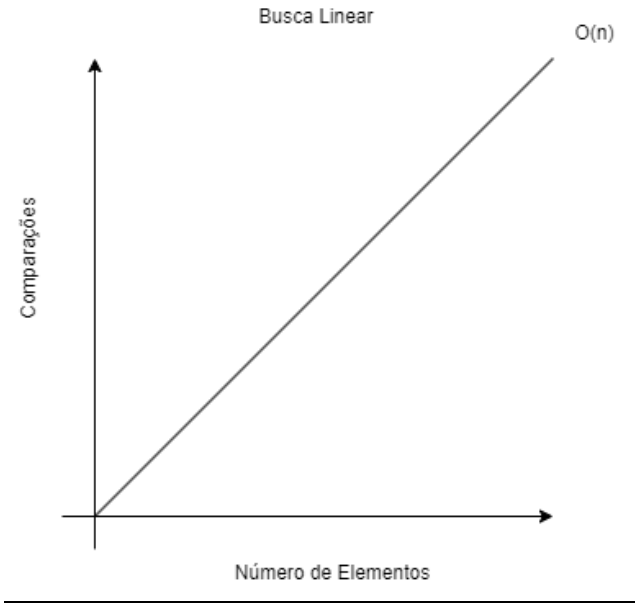

Figura 1 Pesquisa Linear

2.2.2 Hash Table

Hash Table é uma estrutura de dados que implementa uma matriz associativa. A matriz associativa, também designada por dicionário, é um tipo de dados que mapeia chaves para valores. O que diferencia um dicionário de uma *hash table* é a implementação de uma função *hash*. Isto é, numa *hash table*, a chave passa por uma função *hash* que calcula um índice. Depois de calculado esse índice não existe a necessidade de qualquer iteração, sendo que o índice indica os valores estão guardados (Mehlhorn & Sanders, 2008).

Como podemos observar na Figura 2 concluímos que a *hash table* tem uma complexidade de tempo igual a O(1) sendo direto o resultado, ou como referido anteriormente, o algoritmo tem um tempo constante para qualquer valor pesquisado.

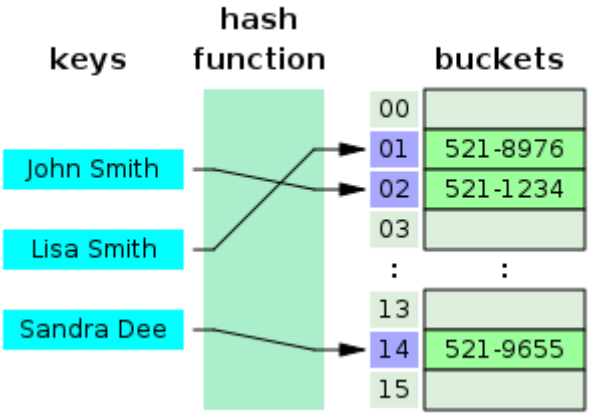

Figura 2 Hash Table

Adaptado de: https://en.wikipedia.org/wiki/Hash_table

Apesar da *hash table* ter uma resposta de tempo constante, há um problema com a sua implementação: as colisões de *hash*. Há uma colisão de *hash* quando valores distintos geram o mesmo *hash*, ou seja, sínteses de voz distintas gerarem o mesmo índice. Para prevenir a reutilização de um índice já com informação útil e eliminar esta desvantagem da *hash table*, serão implementadas *linked lists* para cada índice existente na *hash table*. A *linked list* é um tipo de dados ordenado de forma linear constituídos por dois campos: pelo item, responsável por guardar os parâmetros da síntese e pelo ponteiro que direciona para a próxima *linked list* que gerou o mesmo *hash*. Enquanto não existir colisão o ponteiro terá o valor nulo. Podemos observar na Figura 3 a estrutura da *hash table* com a implementação das *linked lists*. (Cormen, Leiserson, Rivest, & Stein)

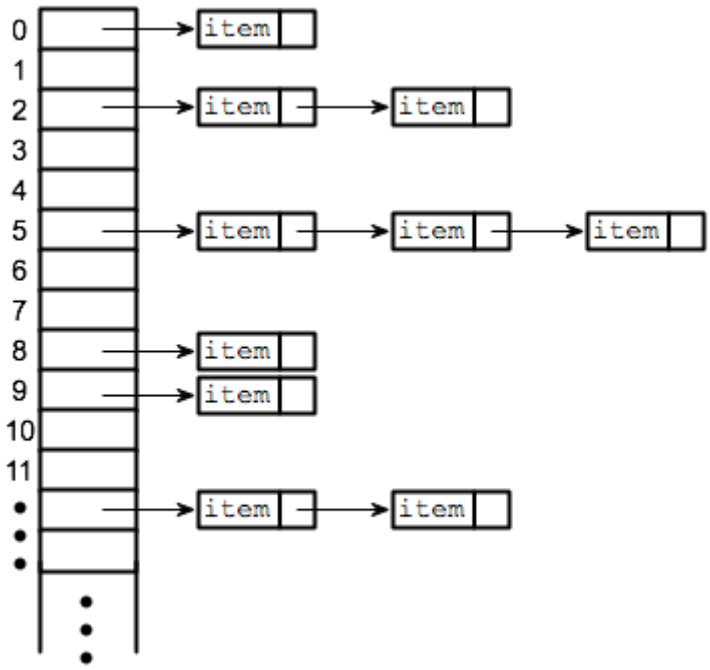

Figura 3 Linked Lists

Adaptado de: https://he-s3.s3.amazonaws.com/media/uploads/0e2c706.png

Esta implementação, usando *hash table* e *linked lists*, elimina a complexidade de tempo de O(1), uma vez que será necessário aplicar uma pesquisa linear em casos de colisão. No entanto, existem, no máximo, duas ou três colisões por índice o que faz desta solução a melhor para o contexto apresentado.

3 Metodologia

Neste capítulo é abordada e analisada a metodologia utilizada no projeto. Este tipo de metodologia é a utilizada pela Altice Labs em todos os seus projetos.

3.1 Metodologia *Waterfall*

O modelo *Waterfall* propõe uma abordagem sistemática, linear e sequencial no que toca ao desenvolvimento de software como podemos observar na Figura 4.

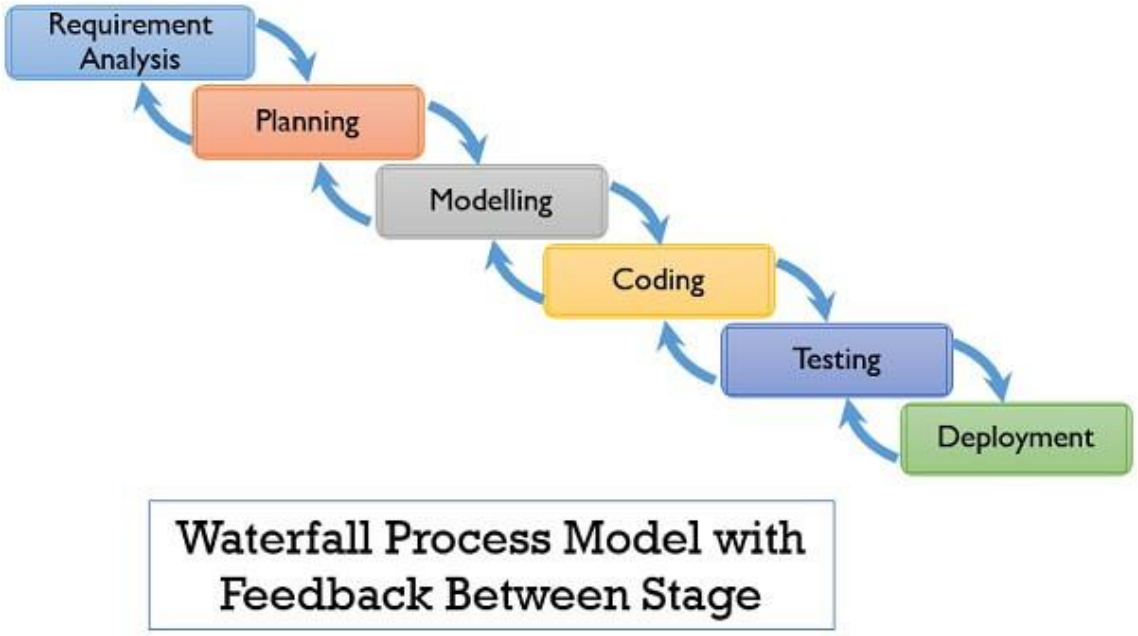

Figura 4 Representação Modelo Waterfall

Adaptado de: https://binaryterms.com/wp-content/uploads/2020/02/waterfall-process-model-with-feedback-between-stages.jpg

Waterfall é uma metodologia de gestão de projetos com uma abordagem muito simples que valoriza o planeamento sólido, fazendo-o uma única vez e de forma correta.

O gestor do projeto tende a ser o grande responsável pelo percurso a ser tomado. Em primeiro lugar, o trabalho é planeado de forma meticulosa e, de seguida, é executado aderindo aos requisitos para entrega do projeto num único ciclo (Adetokunbo A.A. Adenowo, 2013).

No modelo *Waterfall*, cada fase deve ser completada antes da próxima fase começar e não há sobreposição nas fases. Normalmente, numa abordagem *Waterfall*, o resultado de uma fase atua como entrada para a próxima fase.

Os requisitos (requerimentos, na **Erro! A origem da referência não foi encontrada.**) s ão definidos na íntegra antes de qualquer trabalho começar garantindo uma procura de soluções mais rápida e assertiva. Esta fase, que demorou, aproximadamente, uma semana consistiu não só na procura de soluções como na apresentação das mesmas.

De seguida, é feita a conceção (projeto, na **Erro! A origem da referência não foi e ncontrada.**), onde é elaborado um plano detalhado do projeto, tendo em conta os recursos e tempos de resposta. Permite transformar o desenho da solução em código na próxima fase com maior rapidez e facilidade. Esta etapa do planeamento durou cerca de 1 semana.

A fase da implementação (desenvolvimento, na **Erro! A origem da referência não foi e ncontrada.**) é a mais longa, prolongou-se durante 4 semanas, sendo apenas composta por desenvolvimento de código. Semanalmente, são apresentados os progressos, colocadas as dúvidas e dada uma estimativa para a finalização da fase.

Após a implementação são realizados os testes. Os testes concentram-se na lógica interna do *software* e nas funções a que deve atender. Estes testes são realizados para garantir que os requisitos do *software* foram cumpridos. Após estes testes, o *software* é entregue ao cliente.

Assim que o produto é entregue ao cliente, instalado e colocado em operação para que possa ser utilizado, podem ser detetados alguns erros que não foram encontrados ou que passaram despercebidos durante a fase de teste. Há também a possibilidade de o cliente querer fazer alterações no produto. Assim sendo, é necessário voltar à fase da implementação.

3.2 Aplicação da metodologia no projeto

Sendo este estágio de curta duração esta metodologia funcionou muito bem. Permitiu esclarecer e resolver de forma sequencial tudo o que era necessário em cada uma das fases. Todas as sextas-feiras eram feitas reuniões de modo a dar por concluída ou

estender a fase em que me encontrava no projeto. Para além destas reuniões, ao decorrer da semana sempre que era colocada uma questão, existia disponibilidade para esclarecer dúvidas ou pôr em causa decisões previamente tomadas.

4 Análise de requisitos

Antes de dar início a qualquer projeto é necessário definir os requisitos. Como foi usada a metodologia *waterfall*, todos os requisitos foram definidos previamente pela Altice Labs. Este capítulo apresenta esses requisitos, bem como, os diagramas para dar estrutura ao projeto.

4.1 Requisitos funcionais

Um requisito funcional define uma função do sistema, ou seja, é a representação das funcionalidades que o *software* desempenha (ROMAN, 1985).

Desta forma, foram apresentados os seguintes requisitos funcionais:

- Ativação do mecanismo de *cache* deve ser feito via configuração (desativo por *default*);
- A *cache* deve ser persistida em disco (sem qualquer dependência a *software* de terceiros);
- A informação da *cache* deve ser lida no arranque;
- A *cache* da síntese deve ser feita com base em todos os parâmetros da síntese já existentes;
- Permitir configurar o tamanho da *cache* (usar tamanho *default* caso não configurado);
- Permitir configurar localização dos artefactos (usar diretório *default* caso não configurado);
- Garantir que o ficheiro de áudio utilizado é o mesmo que foi guardado na primeira síntese;
- Implementar indicadores que permitam identificar se a *cache* foi usada.

4.2 Diagrama de casos de uso

O diagrama de casos de uso é a melhor forma de organizar os requisitos funcionais e de fazer o relacionamento entre os utilizadores do sistema e os casos a que têm acesso, como podemos observar na Figura 5.

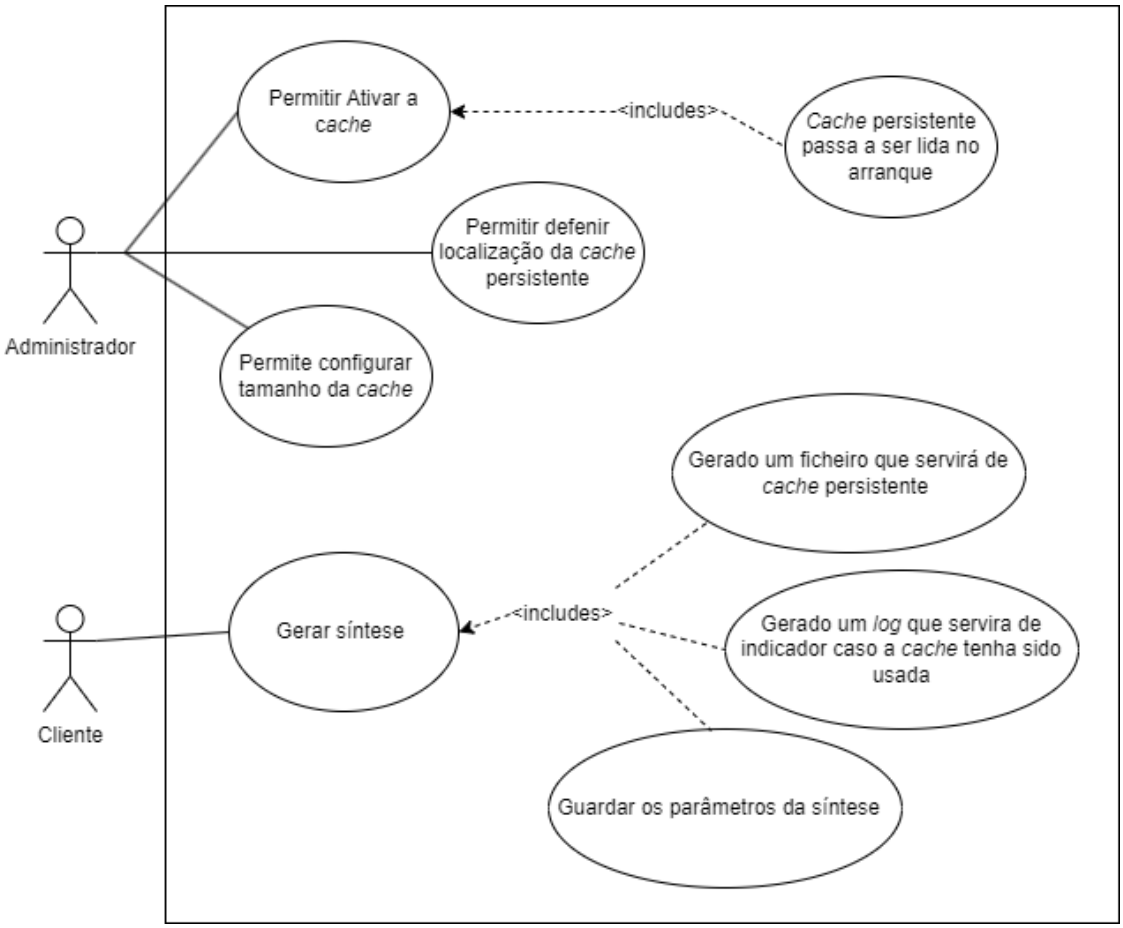

Figura 5 Diagrama de Caso de Uso

4.3 Descrição dos casos de uso

A descrição de um caso de uso serve para descrever pormenorizadamente todas as interações entre o ator e o sistema (Michel H. Fortuna, 2007).

- **Nome:** Nome do caso de uso;
- **Ator:** Ator(es) que participam no caso de uso;
- **Descrição:** A descrição do caso de uso deve ser uma frase simples e que consiga descrever a funcionalidade do mesmo;
- **Pré-condição:** Condição que tem de ser verificada antes do caso de uso;
- **Caminho principal:** Caminho a seguir pelo ator e respetivas respostas do sistema durante o caso de uso;
- **Caminho alternativo:** Erros que podem acontecer durante o caso de uso;
- **Pós-condição:** Condição que acontece após o caso de uso.

Na Tabela 1, Tabela 2, Tabela 3 e Tabela 4 estão representados os casos de uso utilizados no projeto.

Nome	Ativação da c*ache.*
Ator	Administrador.
Descrição	O caso de uso refere-se à ativação da *cache.*
Pré-condição	Acesso ao código.
Caminho principal	1. O ator altera a variável de ambiente responsável pela ativação da *cache;* 2. O ator reinicia o sistema; 3. O sistema carrega a *cache* em memória com a *cache* persistente.
Caminho alternativo	1. A) Se a alteração da variável de ambiente for feita incorretamente estará presente uma mensagem de erro em *log* – "Erro ao configurar a cache"
Pós-condição	3. A) Ao carregar a *cache* persistente estará presente um *log* de todos os itens presentes na *hash table*.

Tabela 1 Caso de uso: Ativação de Cache

Nome	Gerar síntese.
Ator	Cliente.
Descrição	O caso de uso refere-se à geração de uma síntese.
Pré-condição	Acesso a um serviço telefónico.
Caminho principal	1. O ator gera um pedido de síntese com parâmetros válidos; 2. O sistema recebe o pedido de síntese; 3. O sistema reparte o pedido de síntese nos devidos parâmetros; 4. O sistema procura a síntese na cache; 5. O sistema responde com o áudio correspondente à síntese pedida.
Caminho alternativo	3. A) Caso os parâmetros sejam inválidos o sistema responde com erro – "Parâmetros de síntese incorretos"; 4. A) Caso a síntese não exista em *cache* é criado um ficheiro de *cache* persistente e adicionada a síntese com todos os parâmetros à *cache* em memória.
Pós-condição	5. A) Gerados *logs* a indicar que os serviços de fala da Google foram utilizados.

Tabela 2 Caso de uso: Guardar todos os parâmetros da síntese

Nome	Definir localização da *cache* persistente.
Ator	Administrador.
Descrição	O caso de uso refere-se à definição da localização da *cache* persistente.
Pré-condição	Acesso ao código.
Caminho principal	1. O ator altera a variável de ambiente responsável pela localização da *cache* persistente*;* 2. O ator reinicia o sistema.
Caminho alternativo	1. A) Se a alteração da variável de ambiente for feita incorretamente estará presente uma mensagem de erro em *log* - "Diretório inexistente".
Pós-condição	2. A) Ao reiniciar o sistema é escrito um *log* que indica que a *cache* persistente passa a ser lida e escrita na localização definida.

Tabela 3 Caso de uso: Defenir localização cache persistente

Nome	Definir tamanho da *cache*.
Ator	Administrador.
Descrição	O caso de uso refere-se à definição do tamanho da *cache.*
Pré-condição	Acesso ao código.
Caminho principal	1. O ator altera a variável de ambiente responsável pelo tamanho da *cache;* 2. O ator reinicia o sistema.
Caminho alternativo	1. A) Se a alteração da variável de ambiente for feita incorretamente estará presente uma mensagem de erro em *log* – *"*Tamanho incorreto ou não configurado*"*
Pós-condição	2. A) Ao reiniciar o sistema é escrito um *log* indicador do tamanho da *cache.*

Tabela 4 Caso de uso: Definir tamanho da cache

4.4 Diagrama de Fluxo

Um diagrama de fluxo de dados mapeia a sequência de informações e passos dentro de um processo ou sistema. Usa um conjunto de símbolos e formas padronizadas representando as diferentes etapas que são necessárias para que um processo seja realizado corretamente (O que é um diagrama de fluxo de dados?, 2023).

Neste contexto, foi feito o diagrama de fluxo de dados com o intuito de representar como funciona a implementação da *cache* mesmo quando não é utilizada, como podemos observar na Figura 6.

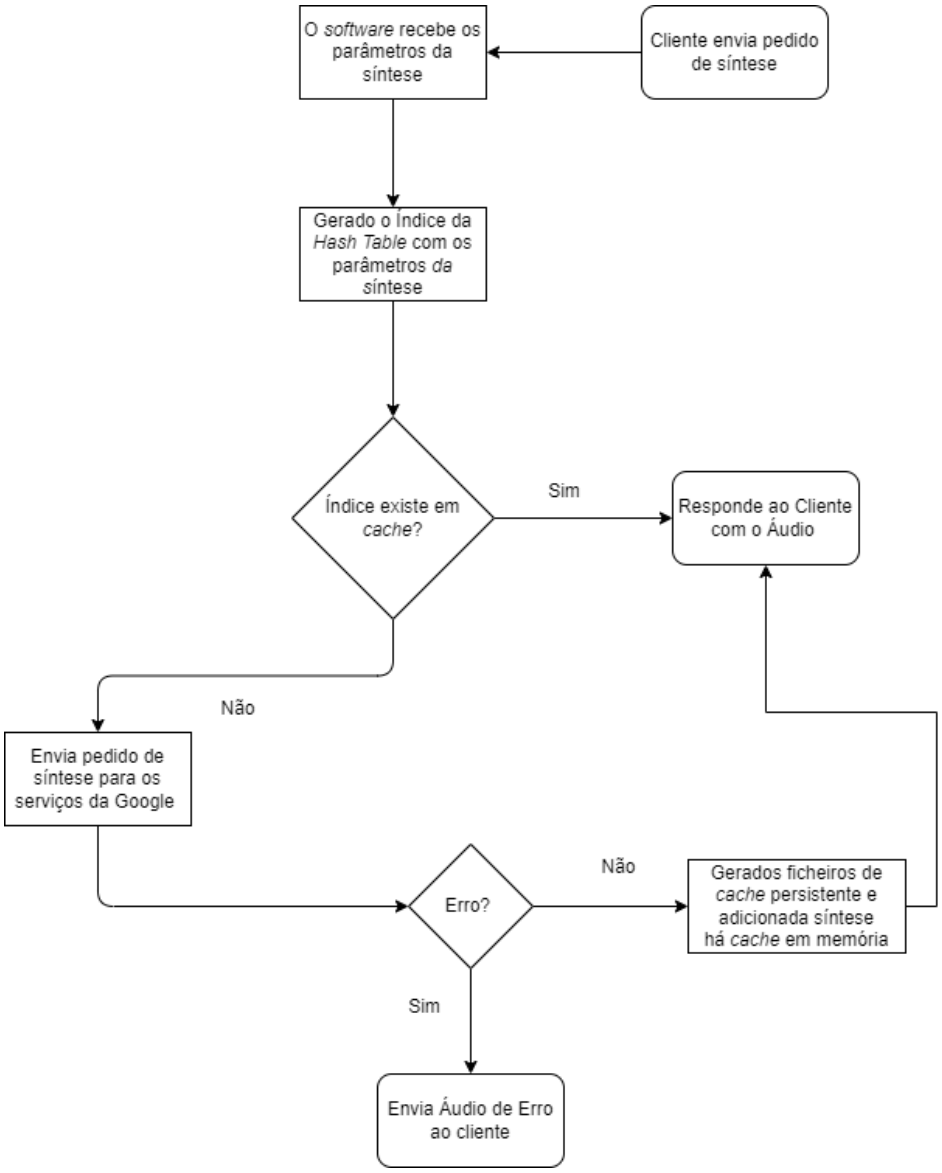

Figura 6 Diagrama de Fluxo da Cache

O processo de decisão é o seguinte: após o momento em que a síntese é pedida pelo cliente o sistema recebe e gera um índice com a informação pedida através da função *hash*. Caso esse índice não exista, o sistema envia o pedido de síntese aos serviços da Google e, de seguida, adiciona a resposta à *cache* em memória. Em conjunto, são criados os ficheiros que constituem a *cache* persistente, garantindo, assim, que quando o sistema é reiniciado nada é perdido. Caso o índice exista em *cache* de memória é enviado o áudio ao cliente.

5 Tecnologias

Para a realização do projeto foram utilizadas várias ferramentas e tecnologias, nomeadamente:

- ***PhonerLite***

PhonerLite é uma aplicação de softphone, ou seja, um programa que faz ligações VoIP permitindo realizar chamadas através da internet e que pode ser usada em sistemas operacionais Windows (What is Phoner?, 2023). Foi a ferramenta que permitiu a utilização dos serviços telefónicos automáticos durante o desenvolvimento do projeto, permitindo perceber pontos importantes, como atrasos ou falhas nos áudios enviados ao cliente.

- **Docker**

Docker é um conjunto de produtos de plataforma como serviço (PaaS) open source que tem como objetivo facilitar o desenvolvimento e execução de aplicações em ambientes isolados. Utiliza um modelo de containers que agrupa softwares, bibliotecas e arquivos de configuração. Todos os containers são executados por um único *kernel* do sistema operacional e, portanto, usam menos recursos do que as máquinas virtuais (Docker, 2023). Os serviços telefónicos automáticos foram replicados para um *container Docker*, dando total liberdade para modificar e implementar a cache no software previamente desenvolvido.

- ***Visual Studio Code***

O Visual Studio Code é um editor de código desenvolvido pela Microsoft que pode ser usado em Windows, Linux ou MacOS. Inclui uma compatibilidade com uma grande variedade de línguas de programação, nomeadamente C e C++ (Getting Started, 2023). Este software de edição de código também permitia, através de uma extensão, conectar aos servidores remotos da Altice Labs onde os serviços telefónicos automáticos estão alojados, facilitando a edição do código responsável pelos mesmos.

- **C**

A linguagem de programação C foi criada por Dennis Ritchie, em 1972. Atualmente, é uma das linguagens de programação mais comuns e usadas em diferentes arquiteturas. Esta linguagem tem a capacidade de compilar código de forma eficiente com um tempo de execução baixo. Para além disso apresenta um elevado nível de portabilidade, ou seja, pode ser executada em vários tipos de sistemas operacionais com poucas ou nenhumas modificações (History of C, 2000).

- **C++**

C++ , linguagem de programação criada por Bjarne Stroustrup em 1979, é uma linguagem de programação baseada na linguagem C, assim sendo, a base e estrutura de ambas as linguagens é muito semelhante (History of C++, 2023). O TTS, previamente desenvolvido pela Altice Labs, foi realizado utilizando as linguagens de programação C e C++, pelo que, estas foram as duas linguagens usadas no projeto apresentado neste relatório.

- **Ubuntu WSL (*Windows Subsystem for Linux*):**

Windows Subsystem for Linux, abreviado como WSL, é um módulo do sistemas operacionais Windows 10, 11 e *Server* 2019 que disponibilizar um ambiente Linux compatível no sistema da Microsoft. Desta forma, é possível executar programas específicos dos sistemas GNU/Linux dentro do próprio Windows sem a necessidade de emuladores ou do uso de máquinas virtuais (Ubuntu on WSL, 2023). Uma vez que o serviço de TTS era implementado num ambiente Linux, foi importante testar e compilar o código para a *cache* num ambiente idêntico.

6 Implementação

Neste capítulo é descrito o processo de implementação do projeto e respetivas etapas.

6.1 Resolução de requisitos

A resolução dos requisitos propostos foi feita de modo que cada requisito atenda a rapidez esperada de um serviço telefónico automático, garantido que nenhum tempo é perdido de forma desnecessária durante a utilização da cache. Para além disso foi tido em atenção o poder computacional em que o software inicial estava alojado.

Seguidamente é discutida a forma como os requisitos propostos pela Altice Labs foram abordados, tendo em vista a sua resolução.

- **Ativação do mecanismo de cache dever ser feito via configuração (desativo por *Default*);**

  Será criada uma variável de ambiente booleana designada: "*enable_cache*". Esta variável terá o valor desativa (*false*) por *default* e ativada quando a utilização da cache for pretendida.

- **Garantir que o ficheiro de áudio utilizado é o mesmo que foi guardado na primeira síntese.;**
  Para assegurar que não existe corrupção ou alteração dos ficheiros de áudio foi criado, para cada um deles um *checksum*[1] obtido a partir de um *hash*. Ao usar o *hash* garantimos que ficheiros iguais criam resultados iguais e em caso de eventual corrupção de algum ficheiro, é criado um hash diferente do original. O algoritmo de *hash* escolhido foi o MD5, já que é um hash muito rápido de calcular, é também um algoritmo em que é muito improvável gerar colisões, sendo a probabilidade de isso acontecer é de 1 em 2^{128}.

[1] Checksum é um hash usado para verificar a integridade de dados.

- **A *cache* da síntese deve ser feita com base em todos os parâmetros da síntese já existentes;**

  Serão guardados na cache todos os parâmetros já existentes no serviço de TTS, sendo estes os seguintes:

  - ***Name*:** Nome do ficheiro WAV, que é gerado automaticamente pelo serviço TTS;
  - ***Language*:** Representação da linguagem usando o código em formato BCP-47[2];
  - ***Voice-Name*:** Este campo é usado para definir a voz usada pelos serviços de voz da Google;
  - ***Voice-Gender*:** Campo é usado para definir o género da voz usada pelos serviços de voz da Google;
  - ***Content*:** Frase dada como *input* para que a síntese seja feita;

  Juntamente será adicionado um parâmetro relativamente à garantia de não existir corrupção ou alteração dos ficheiros de áudio:

  - ***Hash*:** Algoritmo MD5 usado para garantir que não há corrupção/alteração dos ficheiros de áudio.

- **A *cache* deve ser persistida em disco (sem qualquer dependência a *software* de terceiros);**

  Este ponto consiste em assegurar que todos os atributos são guardados num ficheiro em disco para ser usado como memória persistente, juntamente com o ficheiro WAV que será usado para guardar o áudio. Sem depender de *software* de terceiros, duas ótimas opções para *cache* persistente são: o uso de ficheiros CSV ou JSON. Sendo assim, serão demonstrados como cada formato seria usado no contexto apresentado.

  - **JSON**

```
WaveList = {
    "Wave-File":{
        "Name":"FileName",
        "Attributes":{
            "language":"",
```

[2] BCP-47 é um código ou etiqueta padronizada usada para identificar línguas humanas na Internet.

```
                "voice-name":"",
                "voice-gender":"",
                "content":"",
                "hash":MD5(wavefile)
            }
        }
    }
```

- **CSV**

```
"Name|language|voice-name|voice-gender|sampling-
rate|content|hash"
""|""|""|""|""|""|""
```

Ambas as opções são viáveis, no entanto, como deve ser cumprido o menor tempo de leitura possível, o formato de ficheiro CSV consegue ser mais rápido, uma vez que, o ficheiro JSON exige o uso de mais caracteres para a sua construção e também a utilização de bibliotecas como "nlohmann" (Lohmann, 2022) para poder ser interpretado. Com o formato CSV é possível repartir os atributos da síntese utilizando apenas as ferramentas presentes na linguagem C++ procurando pelo separador usado no ficheiro, de uma forma rápida e sem a necessidade de utilização de bibliotecas.

- **Permitir configurar tamanho e idade da *cache* (usar tamanho *default* caso não configurado);**

  Neste campo o tamanho da *hash table* vai ser dividido em dois componentes. Um será responsável por definir o número de *linked lists* (cache_size), o outro será responsável por definir o número total de itens que podem coexistir na *hash table* (cache_max_items):

  - ***Cache_Size*:** Número total de *Linked Lists* alocadas;
  - ***Cache_Max_Items*:** Número total de itens na *Hash Table*.

  Também será introduzida a idade da *cache* implementada como um novo atributo a cada item existente na *hash table*, designado: "*age*". Este atributo serve para indicar há quanto tempo a síntese existe na *cache* em segundos, e quando solicitado novamente atualizado para zero. Ao atualizar a idade para zero garantimos que, durante o processo de limpeza, os pedidos menos

requisitados, ou com idade maior sejam eliminados e os elementos que tem um elevado número de pedidos sejam salvaguardados.

- **Permitir configurar localização dos artefactos (usar diretório *default* caso não configurado);**

  Este requisito trata-se de uma variável de ambiente responsável por guardar o diretório da *cache* persistente, sendo este o "*cache_path*" com *default* no diretório "/tmp" existente nos sistemas UNIX/LINUX. O diretório temporário encontra-se como default para garantir que, caso a cache seja ativada por engano, o seu conteúdo é eliminado quando o sistema é reiniciado.

- **A informação da *cache* deve ser lida no arranque;**

  Este ponto consiste numa função responsável por colocar na cache em memória todos os ficheiros CSV presentes na cache persistente. Essa função terá o nome "*loadCache*" e será apenas executada quando o sistema é inicializado.

- **Implementar indicadores que permitam identificar se a *cache* foi usada.**

  Para finalizar, sempre que é feito um pedido de síntese, serão feitos logs na consola indicando se a *cache* foi usada ou não, o que vai permitir ao administrador perceber a utilidade do uso da *cache*. Depois de cada síntese serão feitos 2 *logs* na consola que representam o número de vezes que cada serviço foi utilizado, nomeadamente:

  - "KPI *Cache*": Número incremental relativamente ao uso de *cache*;
  - "KPI *Google Services*": Número incremental relativamente ao uso dos serviços da *Google*.

6.2 Hash Table

O primeiro processo da construção de uma *hash table* foca-se na procura do melhor método de *hash* para gerar o índice com o mínimo de colisões possível e com a maior rapidez de resposta. Depois de uma procura na plataforma *Google* por "*best hash function for hash table*", foram encontrados dois algoritmos com percentagens de colisão idênticas, o SDBM e JDB2. O que define um bom algoritmo de geração de *hash* não é apenas a sua capacidade de criar poucas colisões, mas também a facilidade e rapidez de o gerar. Seria possível usar algoritmos de *hash* com colisões muito inferiores,

mas a vantagem da rapidez de resposta da *hash table* seria drasticamente reduzida. Sendo assim, os algoritmos escolhidos são um bom compromisso entre tempo de resposta e colisões reduzidas.

6.2.1 SDBM

O SDBM foi criado para uma biblioteca de base de dados, com autor desconhecido. Apesar de ter mais de 30 anos de idade, é considerado um ótimo algoritmo de geração de *hash* (Hash Functions SDBM, 1990).

A Listagem 1 demonstra como o algoritmo SDBM foi implementado no projeto. O Algoritmo recebe como input o conteúdo da síntese e o número máximo de *linked lists* presentes na *hash table*. São de seguida percorridos todos os caracteres do conteúdo, e de forma sistemática calculado o hash final.

```
unsigned long HashTable::SDBMHash(char* str, int size){
    unsigned int hash = 0;
    unsigned int i = 0;
    unsigned int len = strlen(str);
    for (i = 0; i < len; i++) {
        hash = (str[i]) + (hash << 6) + (hash << 16) - hash;
    }
    return hash%size;
};
```

Listagem 1 Algoritmo SDBM

6.2.2 JDB2

Este algoritmo foi proposto por Dan Bernstein. O porquê de funcionar tão bem nunca foi propriamente explicada pelo autor do algoritmo, no entanto continua a ser dos melhores algoritmos de geração de índices atuais (Hash Functions JDB2, 1990).

A Listagem 2 demonstra como o algoritmo JDB2 foi implementado no projeto. O Algoritmo recebe como input o conteúdo da síntese e o número máximo de *linked lists* presentes na *hash table*. Apesar de serem algoritmos distintos, tanto o algoritmo SDBM e JDB2 funcionam de maneira idêntica, sendo a única distinção, a forma como são calculados.

```
unsigned long JDB2Hash(char *str, int size) {
        unsigned long hash = 5381;
        int c;
        while ((c = *str++))
            hash = ((hash << 5) + hash) + c;
        return hash%size;
}
```

Listagem 2 Algoritmo JDB2

Nos dois algoritmos é usado como parâmetro de entrada a "*Cache_Size"*, que no código é representado como *size*, que é responsável por definir o número máximo de *linked lists* presentes na *hash table*. Este valor garante que o índice gerado esteja dentro dos valores de índice existentes na *hash table*, ou seja, valores compreendidos entre 0 e "*Cache_Size"*.

Com a implementação de ambos os algoritmos, é necessário decidir qual usar como default. Para isso foram conduzidos testes, presentes na Figura 7, de modo a perceber qual deles gera o menor número de colisões.

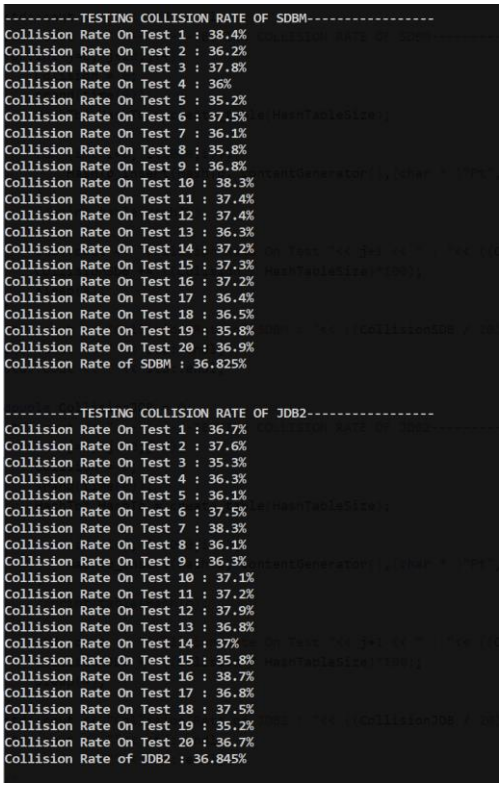

Figura 7 Testes Colisão de Algoritmos

Sendo a média de colisão dos dois algoritmos muito próxima, com o algoritmo SDBM a gerar colisões com média de 36.825% e o JDB2 36.845%, foi considerado o algoritmo com melhor desempenho durante os testes realizados – o algoritmo SDBM. Apesar dos testes realizados, teoricamente, os dois algoritmos: SDBM e JDB2 são idênticos e resultariam numa *hash table* rápida e funcional.

6.2.3 Construção da *Hash Table*

A *hash table* é constituída por 3 classes, para permitir que seja mais fácil interpretar o código e alterar em caso de erros. Cada classe é constituída por dois tipos de ficheiro, um *header* (.h) responsável pela declaração de variáveis, constantes e funções, e um ficheiro de código C++ (.cc) que desenvolve todas as funções declaradas no ficheiro *header*.

Na Figura 8 esta presente de forma resumida o funcionamento das 3 classes referidas. Ao longo da descrição das classes será usada a numeração presente na figura de modo a indicar onde cada função acontece.

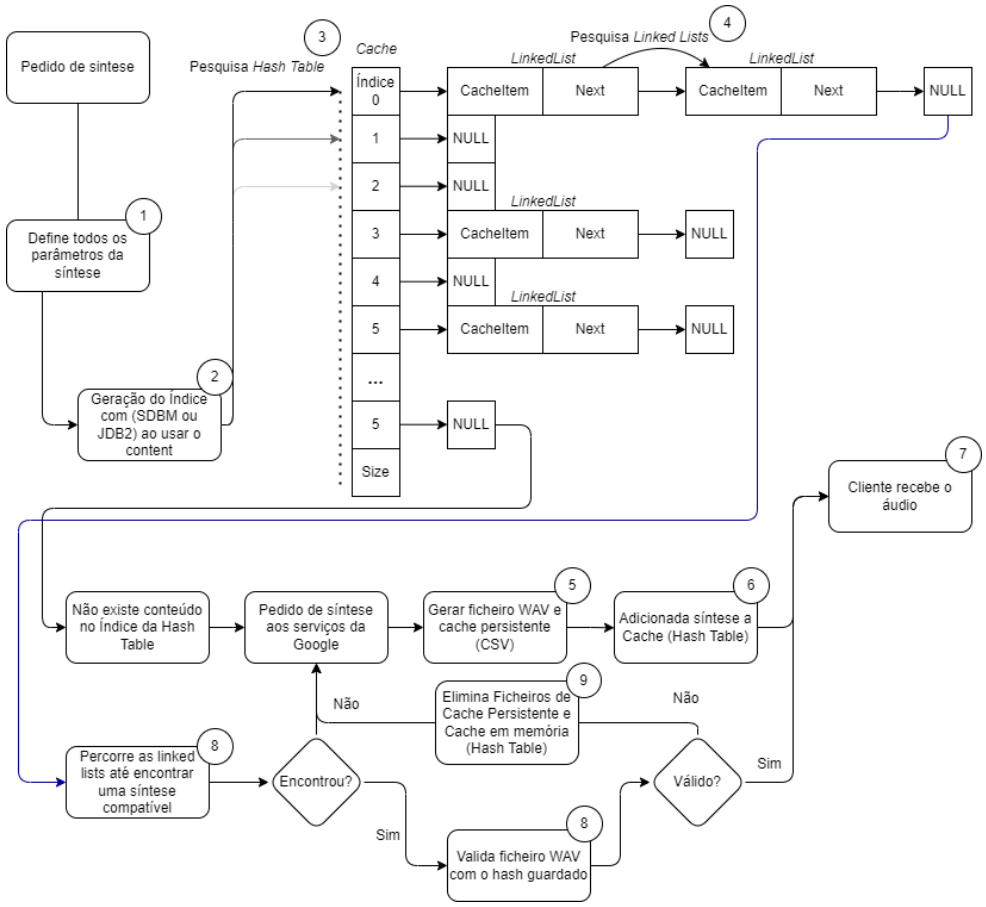

Figura 8 Diagrama de funcionamento

- ***Cache* (*cache.cc* e *cache.h*)**

A classe *cache* servirá como intermediário entre o TTS já existente e a *hash table*. Em simultâneo, como podemos observar na Listagem 3, também servirá para alocar toda a memória que possibilita o funcionamento da *hash table*.

```
HashTable::HashTable(int size, int max)
{
    this->size = size;
    this->count = 0;
    this->max = max;
    //Allocates one linked list for every index
    LinkedList** buckets = new LinkedList*[size];
    for (int i = 0; i < size; i++)
        buckets[i] = NULL;
    overflow_buckets = buckets;
}
```

Listagem 3 Construtor Hash Table

Esta função, também chamada de construtor, recebe o tamanho (*size*) da *hash table* e o número máximo de itens (*max*) que pode ter em simultâneo. De seguida, é alocada uma *linked list* para cada índice existente.

Uma vez que o TTS define sistematicamente os parâmetros da síntese, antes de enviar o pedido de síntese para a Google, será necessário construir uma função para definir na *hash table* cada parâmetro. Podemos observar na Listagem 4 as funções necessárias, e na Figura 8 balão número 1 onde acontece a definição dos parâmetros.

```
void HashTable::setSSML(string SSML){
    content = SSML.substr(SSML.find("\">")+2, SSML.find("<voice")-
SSML.find("\">")-2);
}
void HashTable::setLanguage(string lang){
    language = lang;
}
void HashTable::setVoiceName(string voicename){
    voiceName = voicename;
}
void HashTable::setVoiceGender(string voicegender){
    voiceGender = voicegender;
}
void HashTable::setContent(string cont){
    content = cont;
```

```
}
void HashTable::setSamplingRate(int samplingrate){
      HashTable::samplingRate = samplingrate;
}
```

Listagem 4 Definição de Parâmetros

Depois de enviado o pedido de síntese e recebido o áudio em formato WAV por parte dos serviços da Google, na Listagem 5 está presente a função responsável por criar o ficheiro CSV que constitui a cache persistente. Na Figura 8 podemos observar esta função no balão 5.

```
void HashTable::write_cache_file(string cacheName, string content, string
language, string voiceName, string voiceGender, int samplingRate, string
hash){
    try{
        string csv = cacheName + ".csv";
        string wavFile = cacheName + ".wav";
        ofstream myfile;
        myfile.open(string(file_path) + "/" + csv);
        myfile << "Name|language|voice-name|voice-gender|sampling-
rate|content|hash" << endl;
        myfile << wavFile + "|" + language + "|" + voiceName + "|" +
voiceGender +"|"  + to_string(samplingRate) + "|" + content + "|" + hash;
        myfile.close();
        wgss_xlog(L_INFO,"[CACHE] - File : %s, Written in : %s",
csv.c_str() , file_path);
    }catch(...){
        wgss_xlog(L_ERR,"[CACHE] - Error Writting in Csv File");
    }
}
```

Listagem 5 Cache Persistente

Como o serviço de TTS envia o áudio ao cliente em conjuntos de bits, um ponto importante da cache será também a formatação dos ficheiros de áudio WAV que guardam a síntese. Isto garante que o cliente receba o áudio da mesma forma que recebe se os serviços da Google fossem usados. A função na Listagem 6 demonstra a solução para o problema, juntamente podemos observar no balão 7 da Figura 8 a fase em que a formatação do áudio acontece.

```
  int HashTable::setAudio(char ***audio, int **size, char* wavefile){
      this->foundCacheItem = 0;
      char fileName[100];
      snprintf(fileName, sizeof(fileName),
  (std::string(file_path)+std::string("%s")).c_str(), wavefile);
      std::ifstream filecache(fileName, std::ifstream::binary);
```

```
        if(filecache.is_open()){
            // get length of file:
            filecache.seekg(0 , filecache.end);
            **size = (int)filecache.tellg();
            //**size = (int)filecache.tellg();
            filecache.seekg(0 , filecache.beg);
            //Set The content of the file in Audio
            char* cacheAudio = new char[**size];
            filecache.read(cacheAudio, **size);
            **audio = cacheAudio;
            delete[] cacheAudio;
            filecache.close();
            return 1;
        }
        return 0;
    }
```

Listagem 6 Formatação ficheiro WAV

Mais tarde, no projeto, sempre que a *cache* atingia o número máximo de itens que podem coexistir, sentia-se lentidão de resposta já que era feito um processo de limpeza pouco eficaz. Inicialmente, sempre que era pedida uma nova síntese, era feita uma procura linear da síntese mais antiga e era eliminada, causando lentidão na resposta ao cliente. Para resolver o processo de limpeza da *cache* quando esta atinge o limite sem prejudicar o tempo de resposta constantemente, foi definido que, quando a *cache* atingisse o máximo, uma percentagem da *cache* seria eliminada em simultâneo. Assim, garantimos que o processo de limpeza é feito em períodos e não constantemente. Podemos observar na Listagem 7 a função usada para a resolução do problema.

```
void HashTable::cleanUp(){
    double percent = 1/(double)cleanup_percent;
    int itemsToDelete = this->size / percent;
    int itemCounter = 0;
    int indexOlderItem = 0;
    int olderItem = 0;
    CacheItem* item = NULL;
    CacheItem* temp = NULL;
    LinkedList* head = NULL;
    while(itemCounter < itemsToDelete){
        for (int i = 0; i < size; i++){
            head = this->overflow_buckets[i];
            if(head != NULL){
                temp = head->getOlderItem();
                if(olderItem < time(NULL) - temp->getAge()){
                    olderItem = time(NULL) - temp->getAge();
                    indexOlderItem = i;
```

```
                    item = temp;
                }
            }
        }
        LinkedList* top = this->overflow_buckets[indexOlderItem];
        this->overflow_buckets[indexOlderItem] = top->removeItem(item-
>getContent(), item->getLanguage(), item->getVoiceName(), item-
>getVoiceGender(), file_path);
        itemCounter++;
        this->count--;
        olderItem = 0;
        wgss_xlog(L_DBG,"[CACHE] - Deleted Item in Index : %d",
indexOlderItem);
    }
};
```

Listagem 7 Limpeza da Cache

É também nesta classe que é feita a pesquisa na *hash table*, presente na da Figura 8 balão 4. Será também responsável por indicar à classe *linkedlist* que a adição ou remoção de um item tem de ser feita.

- ***Linkedlist* (*linkedlist.cc* e *linkedlist.h*)**

O construtor da classe *linkedlist*, presente na Listagem 8, consiste na alocação de memória que permita o seu funcionamento.

```
LinkedList::LinkedList(CacheItem* item){
     this->item = item;
     this->next = NULL;
}
```

Listagem 8 Construtor Linkedlist

Assim que um item for atribuído, a *linked list* presente no índice da *hash table* deixa de ser nula. Esta passa a ter um valor para o item e um valor nulo para o "*next*". Caso o índice já tenha um item atribuído, será alocado um espaço de memória com uma nova *linked list* e o valor que estava nulo em *next* guarda o espaço de memória para esta nova *linked list*, também conhecido de ponteiro, permitindo, assim, uma cadeia de itens sem que nenhum seja perdido para colisões de *hash*.

Esta classe será, também, responsável pela procura em *linked lists*, presentes na Figura 8, balão 4 e 8. Pela adição e remoção de itens tanto em *cache* em memória e persistente, presentes na Figura 8, balão 6 e 9.

- ***cacheitem (cacheitem.cc e cacheitem.h)***

O construtor relativo a classe *cacheitem*, presente na Listagem 9, trata de alocar memória para guardar os parâmetros da síntese. É no construtor que é gerado o MD5 inicial com o ficheiro de áudio para que, em próximas sínteses, possa ser usado como garantia de não existir corrupção ou alteração do ficheiro de áudio.

```
CacheItem::CacheItem(char* keyi, char* languagei, char* voiceNamei,
char* voiceGenderi, int samplingRate, char* wavefilei, char* hashi){
    strcpy(key , keyi);
    strcpy(language , languagei);
    strcpy(voiceName , voiceNamei);
    strcpy(voiceGender , voiceGenderi);
    strcpy(wavefile , wavefilei);
    this->age = time(NULL);
    this->samplingRate = samplingRate;
    if(hashi == NULL){
        std::string md5Result = generateMD5(wavefilei);
        std::string nullString = "";
        if (md5Result.length()==0){
            strcpy(hash , nullString.c_str());
        }else{
            strcpy(hash , md5Result.c_str());
        }
    }else{
        strcpy(hash , hashi);
    }
}
```

Listagem 9 Construtor cacheitem

Será também nesta classe que estará alojada a função MD5, Listagem 10, responsável por receber o ficheiro de áudio WAV e gerar o *hash* utilizado na comparação de futuros pedidos de síntese, também presente na Figura 8, balão 8. Para que possa ser gerado o MD5 será usada uma biblioteca com a função MD5 já desenvolvida, a biblioteca OpenSSL (Mark Cox, 1999).

```
std::string generateMD5(char* wavefile) {
    //Open Wave File and Gets All data
    char fileName[FILE_NAME_SIZE];
    snprintf(fileName, sizeof(fileName), "%s%s", file_path_item,
wavefile);
    std::ifstream filecache(fileName, std::ifstream::binary);
    if(filecache.is_open()){
        // get length of file:
        filecache.seekg(0 , filecache.end);
```

```
        int size = filecache.tellg();
        filecache.seekg(0 , filecache.beg);
        //Set The content of the file in Audio
        char cacheAudio[size];
        filecache.read(cacheAudio, size);
        filecache.close();
        //Md5 Generating
        unsigned char digest[MD5_DIGEST_LENGTH+1];
        MD5(reinterpret_cast<const unsigned char*>(cacheAudio), size,
digest);

        std::ostringstream md5Stream;
        for (int i = 0; i < MD5_DIGEST_LENGTH; ++i) {
            md5Stream << std::hex << std::setw(2) << std::setfill('0')
<< static_cast<int>(digest[i]);
        }
        return md5Stream.str();
    }else{
        wgss_xlog(L_ERR,"[CACHE] - Error Trying to Open Wave File:
%s", wavefile);
        return "";
    }
    return "";
}
```

Listagem 10 Garantia MD5

7 Verificação e Validação

Na verificação e validação entra uma ferramenta fundamental neste projeto, os *containers Docker*. O *Docker* oferece portabilidade, consistência e escalabilidade para implantação de aplicações em diferentes ambientes. Os *containers Docker* são leves e isolados garantindo que nenhum erro prejudique um *software* já funcional. É, também, nos *containers* que são configuradas as variáveis de ambiente previamente referidas, que são variáveis definidas ao inicializar o container e utilizadas pelo software para a configuração do próprio. Uma imagem *Docker* trata-se de um conjunto de comandos necessários para inicializar um container.

Para começar, é necessário inicializar o *container Docker* em que o serviço telefónico automático já está implementado e funcional. Para isto, basta ir ao diretório onde a imagem *Docker* e os ficheiros de configuração estão presentes. Ao correr o seguinte comando, garantimos que o *container Docker* será gerado:

- docker compose up

De seguida compilamos o código desenvolvido para a implementação da cache, dentro do *container*:

- docker exec -it windless-mrcp /bin/bash -c “cd /build/src/wgss; scl enable devtoolset-8 ‘make’ ”

É neste passo que os erros no código são detetados. Qualquer erro relativamente ao desenvolvimento da *cache* será disposto na linha de comandos em formato de *logs*, caso exista.

Depois de compilado com sucesso copiamos o resultado da compilação para o *container* que aloja o serviço telefónico em que o TTS está presente, usando os seguintes comandos:

- docker cp /home/ptin_admin/windless-mrcp/trunk/src/wgss/lib/src/.libs/libwgss_api.so.0.0.0 unimrcp_tts_cache_windless-mrcp-wgss:/opt/ptin/windless-mrcp/lib64/

- docker cp /home/ptin_admin/windless-mrcp/trunk/src/wgss/src/.libs/libwgss_plugin.so.0.0.0 unimrcp_tts_cache_windless-mrcp-wgss:/opt/ptin/windless-mrcp/lib64/

O processo de compilação dá-se por concluído, consistindo apenas na compilação e copia da cache para o *software* já existente. Para podermos utilizar o serviço e visualizar o funcionamento do serviço TTS com a cache basta entrar dentro do *container*, com o comando:

- docker exec -it unimrcp_tts_cache_windless-mrcp-wgss bin/bash

Dentro do *container* corremos o serviço telefónico e, como demonstra a Figura 9, serão feitos todos os *logs* de inicialização do serviço de TTS e *cache*:

- ./entrypoint WGSS

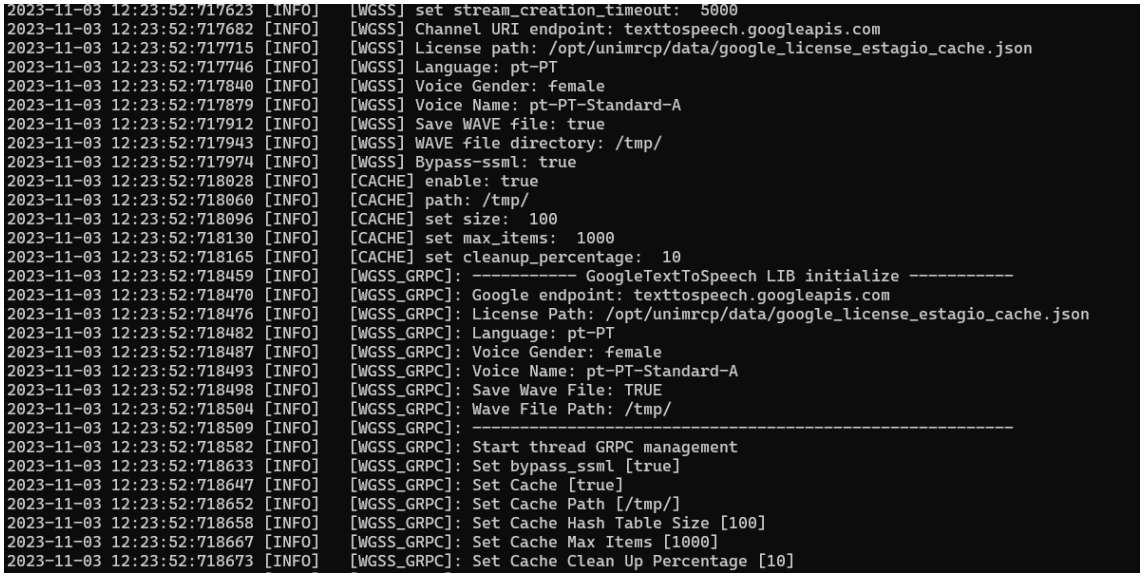

Figura 9 Inicialização do serviço

Com a inicialização do sistema serão feitos 5 *logs* relativos a *cache* (*enable*, *path*, *size*, *max_items*, *cleanup_percentage*). Com estes *logs* podemos verificar se todas as variáveis de ambiente são propriamente definidas. Seria nesta fase que qualquer erro seria visível em *logs*. Tanto erros de variáveis de ambiente como código que apesar de compilado com erros.

Com a *cache* implementada e o serviço TTS funcional, podemos gerar vários pedidos de síntese, presente na Figura 10, onde são definidas num ficheiro XML as sínteses a ser pedidas. O pedido de síntese tem o seguinte formato:

- <prompt xml:lang=”pt-PT”>Texto da síntese<voice name=”pt-PT-Standard-B”></voice></prompt>

```
[ptin_admin@wms-asterisk20-centos7 ~]$ cat /home/ptin_admin/unimrcp_tts_cache/conf/wms-vxml.service/just_gen_tts.xml
<?xml version="1.0" encoding="UTF-8"?>

<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">

<catch event="connection.disconnect">
                <log>Event:<value expr="_event"/> <value expr="_message"/> </log>
                <goto next="#exit"/>
</catch>

<form id="welcome">
    <block>
                <log>Play Bom</log>
                <!-- <prompt bargein="true"><audio src="/var/ptin/wms-vxml/vxml/say_something_after_beep.wav" /></prompt> -->
                <prompt xml:lang="pt-PT">Primeira<voice name="pt-PT-Standard-B"></voice></prompt>
                <prompt xml:lang="pt-PT">Segunda<voice name="pt-PT-Standard-B"></voice></prompt>
                <prompt xml:lang="pt-PT">Terceira<voice name="pt-PT-Standard-B"></voice></prompt>
                <prompt xml:lang="pt-PT">Segunda<voice name="pt-PT-Standard-B"></voice></prompt>
                <prompt xml:lang="pt-PT">Quarta<voice name="pt-PT-Standard-B"></voice></prompt>
                <prompt xml:lang="pt-PT">Quinta<voice name="pt-PT-Standard-B"></voice></prompt>
                <goto next="#exit"/>
    </block>
</form>



<form id="exit">
<block>
                        <log>Exit</log>
<exit/>
</block>
</form>
</vxml>
```

Figura 10 Pedidos de síntese

Depois de criado e guardado o pedido de síntese, recorremos ao *PhonerLite*. Os testes nesta aplicação são bastante simples sendo apenas necessário fazer uma chamada para o IP e o porto onde o serviço se encontra, como podemos observar na Figura 11.

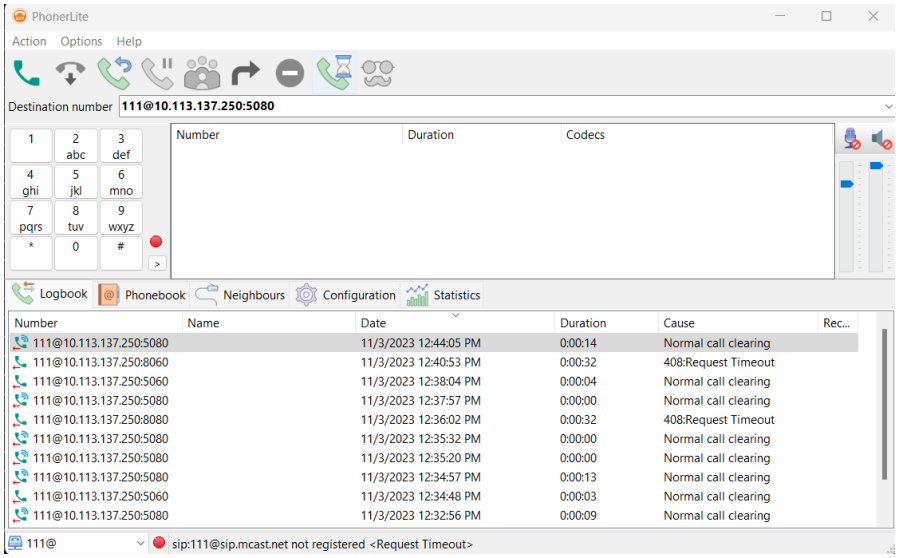

Figura 11 PhonerLite Pedido de Síntese

Com o pedido feito obtemos o resultado do serviço tanto em áudio que é recebido pelo *PhonerLite* como a visualização dos *logs* presentes na máquina. Nos *logs* estão presentes também todas as sínteses presentes na *cache*.

Na Figura 12 e Figura 13 como não existia sínteses em *cache* foram feitos os pedidos aos serviços da *Google* e gerados os ficheiros de *cache* persistente. Para perceber se a *cache* estava a correr corretamente foi duplicada uma síntese no pedido presente na Figura 10.

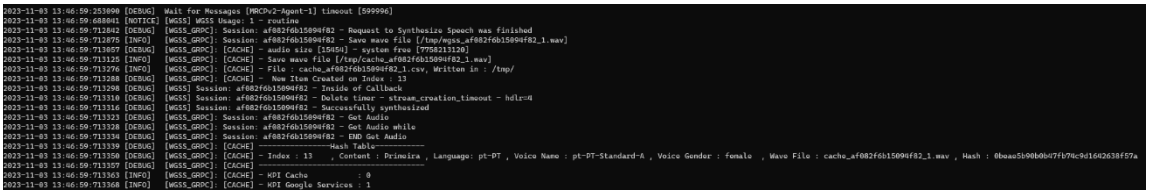

Figura 12 Primeiro Pedido de Síntese (Serviços da Google)

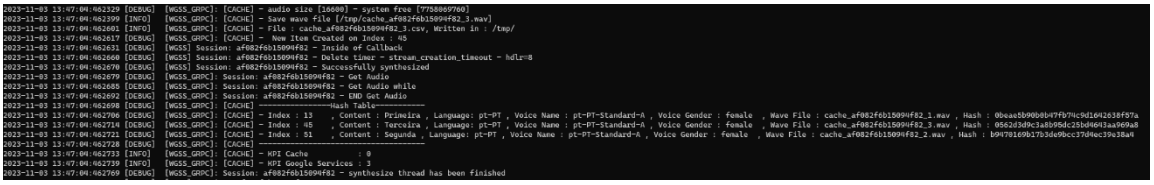

Figura 13 Terceiro Pedido de Síntese (Serviços da Google)

Na Figura 14 podemos perceber que a síntese duplicada é detetada e fica presente um *log* de que existe a síntese em *cache* e a mesma será usada.

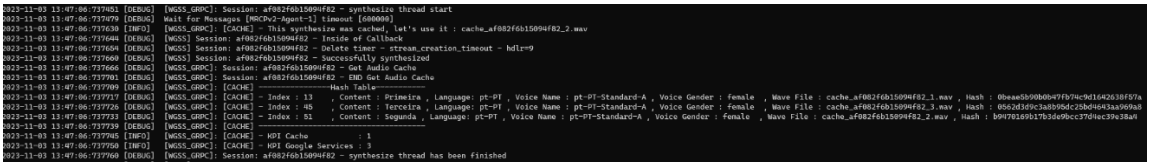

Figura 14 Quarto Pedido de Síntese (Encontrado em Cache)

Todos os ficheiros da *cache* persistente são corretamente gerados como podemos observar na Figura 15, em que estão presentes para cada ficheiro CSV um ficheiro WAV.

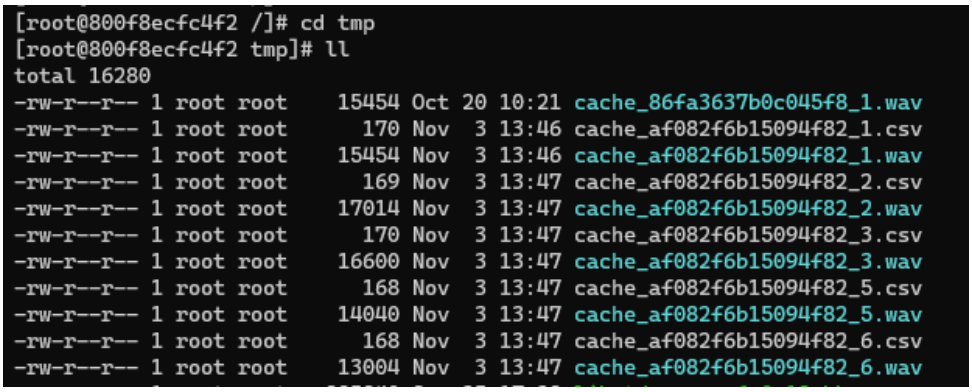

Figura 15 Cache Persistente

De seguida, para testar novamente a *cache* foram feitos novamente os seis pedidos demonstrados na Figura 10, com o objetivo de perceber se a *cache* estava a correr de forma correta. A intenção é que nenhum pedido seja feito aos serviços da *Google* e que todos sejam uma resposta da *cache*.

Os testes corresponderam aos resultados pretendidos, sendo que todos foram usados pela *cache*. Podemos observar na Figura 16 que nas 12 sínteses pedidas temos os *logs* indicadores de uso de *cache* e dos serviços da *Google*, com 5 pedidos para a *Google* e 7 para a *cache*.

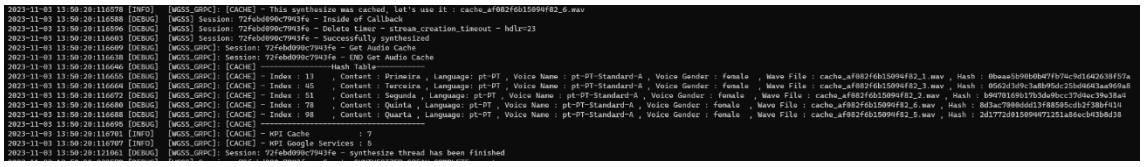

Figura 16 Novo Pedido de Síntese

Estes passos demonstram que a *cache* está em perfeito funcionamento, independentemente do número de pedidos que seja feito. Depois disto, foram feitas inúmeras sínteses no período restante de estágio de 1 mês para garantir que nenhum erro aconteceria. Não foi registado nenhum erro, resultando numa implementação com sucesso.

8 Conclusão

No decorrer do desenvolvimento da *cache*, a utilização de uma *Hash Table* tornava-se cada vez mais evidente de que seria a melhor abordagem. Esta garante tempos de resposta que nenhum outro método consegue.

Durante o processo de implementação da *cache*, um dos focos era desenvolver código que pudesse ser implementado em qualquer projeto e, com isto, acredito que a *cache* não se limita ao serviço de TTS.

No projeto a fase de testes de carga não foi feita, no entanto, a *cache* está a ser utilizada diariamente pelos colaborados da Altice Labs. Até à data do final do estágio a implementação da *cache* estava a funcionar na perfeição.

Sendo que o meu estágio tinha a duração de três meses e o projeto proposto foi terminado em cerca de dois meses, o restante tempo do estágio constituiu no estudo e implementação de um serviço contrário ao TTS, ou seja, Speech to Text (STT) a usar os serviços da Microsoft. Este trabalho serviu como um desafio extra ao meu estágio e também foi bom para esperar por resultados ou erros da *cache* implementada.

Faltando os testes de carga não existe garantias se o serviço com *cache* tem tempos de resposta maiores ao cliente mesmo que na teoria eles não existam. No entanto, o grande objetivo do projeto era reduzir os custos dos serviços de fala da *Google* e sabemos com certeza e, sem necessidade de testes de carga, que a utilização da *cache* é sem dúvida uma mais-valia para o projeto de TTS previamente desenvolvido.

9 Bibliografia

Adetokunbo A.A. Adenowo, B. A. (Julho de 2013). Software Engineering Methodologies. *A Review of the Waterfall Model and Object-Oriented Approach*, p. 8. Obtido de https://www.researchgate.net/profile/Adetokunbo_Adenowo/publication/344194737_Software_Engineering_Methodologies_A_Review_of_the_Waterfall_Model_and_Object-_Oriented_Approach/links/5f5a803292851c07895d2ce8/Software-Engineering-Methodologies-A-Review-of-th

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (s.d.). Introduction to Algorithms. *10.2 Linked lists*. Obtido de https://dl.ebooksworld.ir/books/Introduction.to.Algorithms.4th.Leiserson.Stein.Rivest.Cormen.MIT.Press.9780262046305.EBooksWorld.ir.pdf

Docker. (2023). Obtido de Docker: https://www.docker.com/resources/what-container/

Estado da Arte. (23 de Outubro de 2003). Obtido de Repositorium: https://repositorium.sdum.uminho.pt/bitstream/1822/899/3/C-Parte%20II%20-%20Estado%20da%20Arte%20-%20completo.pdf

Getting Started. (2023). Obtido de Visual Studio Code: https://code.visualstudio.com/docs/?dv=win&build=insiders

Hash Functions JDB2. (1990). Obtido de Yorku: http://www.cse.yorku.ca/~oz/hash.html

Hash Functions SDBM. (1990). Obtido de Yorku: http://www.cse.yorku.ca/~oz/hash.html

História Altice Labs. (2023). Obtido de AlticeLabs: https://www.alticelabs.com/history/

History of C. (2000). Obtido de LivingInternet: https://www.livinginternet.com/i/iw_unix_c.htm

History of C++. (2023). Obtido de unstop: https://unstop.com/blog/history-of-cpp

Knuth, D. (1998). Sorting and Searching. The Art of Computer Programming. Vol. 3 (2nd ed.). p. 792. Obtido de Wikipedia: https://seriouscomputerist.atariverse.com/media/pdf/book/Art%20of%20Computer%20Programming%20-%20Volume%203%20(Sorting%20&%20Searching).pdf

Lohmann, N. (12 de 11 de 2022). *Json*. Obtido de Github: https://github.com/nlohmann/json

Mark Cox, R. E. (1999). *https://www.openssl.org/*. Obtido de OpenSSL: https://www.openssl.org/

Mehlhorn, K., & Sanders, P. (2008). Hash Tables and Associative Arrays. Obtido de https://people.mpi-inf.mpg.de/~mehlhorn/ftp/Toolbox/HashTables.pdf

Michel H. Fortuna, C. M. (2007). Um Modelo Integrado de Requisitos com Casos de Uso. Obtido de https://d1wqtxts1xzle7.cloudfront.net/50455548/Um_Modelo_Integrado_de_Requisitos_com_Ca20161121-28973-1tm42wf-libre.pdf?1479733645=&response-content-disposition=inline%3B+filename%3DUm_Modelo_Integrado_de_Requisitos_com_Ca.pdf&Expires=1700506198&Signature

O que é um diagrama de fluxo de dados? (2023). Obtido de LucidChart: https://www.lucidchart.com/pages/pt/o-que-e-um-diagrama-de-fluxo-de-dados

ROMAN, G.-C. (1985). *A taxonomy of current issues in requirements engineering.*

Saber-e-Fazer. (2003). Obtido de AlticeLabs: https://www.alticelabs.com/wp-content/uploads/2021/07/Saber-e-Fazer_2003.pdf

SMITH, A. J. (Setembro de 1982). Cache Memories. p. 58. Obtido de TechTarget: https://dl.acm.org/doi/pdf/10.1145/356887.356892

The Design and Analysis of Computer Algorithms. (11 de Janeiro de 1974). *The Design and Analysis of Computer Algorithms*, p. 479. Obtido de Great Learning:

https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/The%20Design%20and%20Analysis%20of%20Computer%20Algorithms%20%5BAho,%20Hopcroft%20&%20Ullman%201974-01-11%5D.pdf

Ubuntu on WSL. (2023). Obtido de Ubuntu: https://ubuntu.com/wsl

What is Phoner? (2023). Obtido de Phoner: https://phoner.de/index_en.htm

10 Anexos

Cache.h

```
#ifndef CACHE_H
#define CACHE_H

#include "linkedlist.h"
#include "cacheitem.h"
#include <iostream>
#include <cstring>

using namespace std;

#define FILE_PATH "/opt/ptin/windless-mrcp/cache/"
#define CLEANUP_PERCENTAGE 10
static char file_path[256] = FILE_PATH;
static int cleanup_percent = CLEANUP_PERCENTAGE;

class CacheItem;
class LinkedList;

class HashTable
{
    LinkedList** overflow_buckets;
    int size;
    int count;
    int max;

    int CacheUsed = 0;
    int GoogleService = 0;

    //Metodo de encriptação
    unsigned long SDBMHash(char* str, int size);

    void insertItem(string cacheName, string content, string language,
string voiceName, string voiceGender, int samplingRate, bool
LoadCacheFiles, string hash);

    string language;
    string voiceName;
    string voiceGender;
    string content;
    int samplingRate;

    public:
```

```
    void setLanguage(string lang);
    void setVoiceName(string voicename);
    void setVoiceGender(string voicegender);
    void setContent(string content);
    void setSamplingRate(int samplingrate);
    void setSSML(string SML);

    char cacheWaveFile[256];
    bool foundCacheItem = 0;

    HashTable(int size, int max);
    ~HashTable();

    void loadCache();

    void write_cache_file(string cacheName, string content, string
language, string voiceName, string voiceGender, int samplingRate, string
hash);

    int search();
    void print_table();

    int getCount();

    int getCacheUses();
    int getGoogleUses();

    void cleanUp();

    int addItem(string fileName, string audio, int audioSize);
    int setAudio(char ***audio, int **size);
};

void setCachePath(char* path);
void setCleanUpPercentage(int cache_cleanup_percentage);

#endif
```

Cache.cc

```
#include "../inc/cache.h"
#include "../inc/apiwgss_logger.h"

#include <iostream>
#include <cstring>
#include <experimental/filesystem>
#include <fstream>
#include <mutex>

std::mutex g_num_mutex;

//----------------------------------//
//10% Limpeza                       //
//Processo de Limpeza               //
//Lançar Thread Para Limpeza Cache  //
//mutex --Procurar                  //
//----------------------------------//
//Memoria                           //
//                                  //
//----------------------------------//

void HashTable::setSSML(std::string SSML){
	strcpy(HashTable::content , (char*)SSML.substr(SSML.find("\">")+2,
SSML.find("<voice")-SSML.find("\">")-2).c_str());
	//HashTable::content = SSML;
}

void HashTable::setLanguage(char* lang){
	strcpy(HashTable::language, lang);
}

void HashTable::setVoiceName(char* voicename){
	strcpy(HashTable::voiceName , voicename);
}

void HashTable::setVoiceGender(char* voicegender){
	strcpy(HashTable::voiceGender , voicegender);
}

void HashTable::setContent(char* cont){
	strcpy(HashTable::content , cont);
}

void HashTable::setSamplingRate(int samplingrate){
	HashTable::samplingRate = samplingrate;
}
```

```
//Basicamente um ARRAY de LinkedLists, uma linkedlist para cada posição -
> Feito
//Construtor
HashTable::HashTable(int size, int max)
{
    this->size = size;
    this->count = 0;
    this->max = max;

    LinkedList** buckets = new LinkedList*[size];
    for (int i = 0; i < size; i++)
        buckets[i] = NULL;
    overflow_buckets = buckets;
}

//----------------------Hash Function ----------------------
unsigned long HashTable::SDBMHash(char* str, int size){
    unsigned int hash = 0;
    unsigned int i = 0;
    unsigned int len = strlen(str);
    for (i = 0; i < len; i++) {
        hash = (str[i]) + (hash << 6) + (hash << 16) - hash;
    }
    return hash%size;
};

//Load Cache
void HashTable::loadCache(){
      for (auto& i :
std::experimental::filesystem::directory_iterator(file_path)){
            std::string pathToCSV = i.path();
            if (strcmp(pathToCSV.substr((int)pathToCSV.length()-
3).c_str(), "csv") == 0){
                std::ifstream CSVFile(pathToCSV);
                std::string Attri;
                //Read and Discard the First Line
                std::getline(CSVFile, Attri);
                //Read and Stores the Second Line
                std::getline(CSVFile, Attri);
                //Counter to use on Switch
                int Count = 0;
                //Attributes to ADD
                char wavefile[FILE_NAME_SIZE]; //Count = 0
                char language[6]; //Count = 1
                char voiceName[32]; //Count = 2
                char voiceGender[10]; //Count = 3
                int samplingRate; //Count = 4
                char key[1024];  //Count = 5
```

```
                char hash[33]; //Count = 6
                //Split String
                char *ptr;
                ptr = std::strtok((char*)Attri.c_str(), "|");
                //strcpy(wavefile, ptr);
                snprintf(wavefile, sizeof(wavefile), "%s", ptr);
                while( ptr != NULL ) {
                    switch (Count){
                        case 1:
                            snprintf(language, sizeof(language), "%s",
ptr);
                        case 2:
                            snprintf(voiceName, sizeof(voiceName), "%s",
ptr);
                        case 3:
                            snprintf(voiceGender, sizeof(voiceGender),
"%s", ptr);
                        case 4:
                            sscanf(ptr, "%d", &samplingRate);
                        case 5:
                            snprintf(key, sizeof(key), "%s", ptr);
                        case 6:
                            snprintf(hash, sizeof(hash), "%s", ptr);
                    }
                    ptr = strtok(NULL, "|");
                    Count++;
                }
                //Closes CSVFile
                CSVFile.close();
                //Validate Wave File
                if(strcmp(hash, generateMD5(wavefile).c_str()) != 0 ){
                    wgss_xlog(L_ERR,"[CACHE] - Found Corruption in Wave
File : %s", wavefile);

                    std::string CsvFile = file_path
+std::string(wavefile).substr(0,std::string(wavefile).length()-4)+".csv";
                    //CSV File
                    if (std::remove(CsvFile.c_str())==0){
                        wgss_xlog(L_INFO,"[CACHE] - File : %s Deleted Due
To Corruption",
(std::string(wavefile).substr(0,std::string(wavefile).length()-
4)+".csv").c_str());
                    }else{
                        wgss_xlog(L_ERR,"[CACHE] - Error Deleting
Corrupted File : %s",
 (std::string(wavefile).substr(0,std::string(wavefile).length()-
4)+".csv").c_str());
                    }
                    //Wave File
```

```
                    if
(std::remove((file_path+std::string(wavefile)).c_str())==0){
                        wgss_xlog(L_INFO,"[CACHE] - File : %s Deleted Due
To Corruption", wavefile);
                    }else{
                        wgss_xlog(L_ERR,"[CACHE] - Error Deleting
Corrupted File : %s", wavefile);
                    }
                }else{
                    //Inserts Items and Tells the insert function that
its loading the cache

insertItem(key,language,voiceName,voiceGender,wavefile,samplingRate,
NULL, 1, hash);
                }
            }
      }
      this->print_table();
}

//Write in Cache for presistent Memory
void HashTable::write_cache_file(char* key, char* language, char*
voiceName, char* voiceGender, char* wavefile, int samplingRate , char*
sessionID, char* hash){
      try{
            std::ofstream myfile;
            myfile.open(file_path
+std::string("/")+std::string(sessionID)+std::string(".csv"));
            myfile << "Name|language|voice-name|voice-gender|sampling-
rate|content|hash" << std::endl;
            myfile << wavefile + std::string("|") + language + "|" +
voiceName + "|" + voiceGender +"|"  + std::to_string(samplingRate) +"|"+
key+"|"+ hash;
            myfile.close();
            wgss_xlog(L_INFO,"[CACHE] - File : %s, Written in : %s",
(std::string(sessionID)+std::string(".csv")).c_str() ,file_path);
      }catch(...){
            wgss_xlog(L_ERR,"[CACHE] - Error Writting in Csv File");
      }
}

void HashTable::cleanUp(){
    double percent = 1/(double)cleanup_percent;
    int itemsToDelete = this->size / percent;
    int itemCounter = 0;
    int indexOlderItem = 0;
    int olderItem = 0;
    CacheItem* item = NULL;
    CacheItem* temp = NULL;
```

```
    LinkedList* head = NULL;
    while(itemCounter < itemsToDelete){
        for (int i = 0; i < size; i++){
            head = this->overflow_buckets[i];
            if(head != NULL){
                temp = head->getOlderItem();
                if(olderItem < time(NULL) - temp->getAge()){
                    olderItem = time(NULL) - temp->getAge();
                    indexOlderItem = i;
                    item = temp;
                }
            }
        }
        LinkedList* top = this->overflow_buckets[indexOlderItem];
        this->overflow_buckets[indexOlderItem] = top->remove(item-
>getKey(), item->getLanguage(), item->getVoiceName(), item-
>getVoiceGender(), file_path);
        itemCounter++;
        this->count--;
        olderItem = 0;
        wgss_xlog(L_DBG,"[CACHE] - Deleted Item in Index : %d",
indexOlderItem);
    }
};


//LinkedList Insert Done
void HashTable::insertItem(char* key, char* language, char* voiceName,
char* voiceGender, char* wavefile, int samplingRate, char* sessionID,
bool LoadCacheFiles, char* hash){
    if (this->count >= this->max){
        wgss_xlog(L_ERR,"[CACHE] - Hash Table is Full, Proceding to Clean
Up : %s", wavefile);
        g_num_mutex.lock();
        this->cleanUp();
        g_num_mutex.unlock();
    }
    //Creates ITEM
    CacheItem* item = new CacheItem(key, language, voiceName,
voiceGender, samplingRate , wavefile, hash);
    // Hash
    int index = SDBMHash(key, this->size);
    //LinkedList
    LinkedList** head = this->overflow_buckets;

    if(!LoadCacheFiles){
        write_cache_file(key, language, voiceName, voiceGender, wavefile,
samplingRate, sessionID, item->getHash());
    }
```

```
    if (*(head+index) == NULL){
        *(head+index) = new LinkedList(item);
        wgss_xlog(L_DBG,"[CACHE] -  New Item Created on Index : %d",
index);
    }else{
        (*(head+index))->append(item, index);
        wgss_xlog(L_DBG,"[CACHE] -  New Item Created With Collision on
Index : %d", index);
    }
    this->count++;
};

void HashTable::insert(char* sessionID){
    g_num_mutex.lock();
    insertItem(this->content, this->language, this->voiceName, this-
>voiceGender, this->cacheWaveFile , this->samplingRate, sessionID, 0,
NULL);
    this->GoogleService++;
    this->foundCacheItem = 0;
    g_num_mutex.unlock();
}

int HashTable::search()
{
    if (this->content == NULL) return -1;

    int index = SDBMHash(this->content, this->size);
    LinkedList *head = this->overflow_buckets[index];

    if(head == NULL) return -1;

    char* name;
    FileStatus resultStatus= head->search(this->content,this->language ,
this->voiceName, this->voiceGender, &name, file_path);

    if (resultStatus == FILE_CORRUPTED){
        wgss_xlog(L_ERR,"[CACHE] - Found Corruption in Wave File");
        g_num_mutex.lock();
        //REVER , head = head->remove a dar erro
        this->overflow_buckets[index] = head->remove(this->content, this-
>language , this->voiceName, this->voiceGender, file_path);
        g_num_mutex.unlock();
        this->count--;
        return -1;
    }

    if(resultStatus == FILE_FOUND){
```

```
        snprintf(this->cacheWaveFile, sizeof(this->cacheWaveFile), "%s",
name);
        this->CacheUsed++;
        this->foundCacheItem = 1;
        return 0;
    }
    return -1;
}

void HashTable::print_table()
{
    wgss_xlog(L_DBG,"[CACHE] ----------------Hash Table-----------");
    for (int i = 0; i < this->size; i++)
    {
        LinkedList *head = this->overflow_buckets[i];
        if(head != NULL){
            head->printList(i);
        }
    }
    wgss_xlog(L_DBG,"[CACHE] ------------------------------------");
}

int HashTable::setAudio(char ***audio, int **size, char* wavefile){
    this->foundCacheItem = 0;
    char fileName[100];
    snprintf(fileName, sizeof(fileName),
(std::string(file_path)+std::string("%s")).c_str(), wavefile);
    std::ifstream filecache(fileName, std::ifstream::binary);
    if(filecache.is_open()){
        // get length of file:
        filecache.seekg(0 , filecache.end);
        **size = (int)filecache.tellg();
        //**size = (int)filecache.tellg();
        filecache.seekg(0 , filecache.beg);
        //Set The content of the file in Audio
        char* cacheAudio = new char[**size];
        filecache.read(cacheAudio, **size);
        **audio = cacheAudio;
        delete[] cacheAudio;
        filecache.close();
        return 1;
    }
    return 0;
}

int HashTable::getCount(){
    return this->count;
}
```

```
int HashTable::getCacheUses(){
    return this->CacheUsed;
}

int HashTable::getGoogleUses(){
    return this->GoogleService;
}

void setCachePath(char* path){
    snprintf(file_path, sizeof(file_path), "%s", path);
    wgss_xlog(L_INFO,"[CACHE] - Cache Path Changed to : ", file_path);
    setCachePathItem(path);
}

void setCleanUpPercentage(int cache_cleanup_percentage){
    cleanup_percent = cache_cleanup_percentage;
}
```

LinkedList.h

```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H

#include "cache.h"
#include "cacheitem.h"

#include <cstring>
#include <iostream>


using namespace std;

enum FileStatus{
    FILE_FOUND = 0,
    FILE_NOT_FOUND = 1,
    FILE_CORRUPTED = 2
};

class CacheItem;
class HashTable;

class LinkedList
{
    //Garantir Segurança no uso do objeto
    LinkedList *next;
    CacheItem *item; // -> fechada a CacheItem's ,existem maneiras de
tornar agnostico
```

```
    //EX: void *item //
    public:

    LinkedList(CacheItem* item);
    ~LinkedList(){
        return;
    };

    void append(CacheItem* item, int index);
    LinkedList* removeItem(string content, string language, string
voiceName, string voiceGender, char* file_path);

    void printList(int index);
    FileStatus search(string content, string language, string voiceName,
string voiceGender, char** fileName, char* file_path);

    CacheItem* getOlderItem();
    void removeFiles(string nameFile, char* file_path);
};

#endif
```

LinkedList.cc

```
#include "../inc/linkedlist.h"
#include "../inc/apiwgss_logger.h"

#include <iostream>
#include <cstring>
#include <stdio.h>


LinkedList::LinkedList(CacheItem* item){
        this->item = item;
        this->next = NULL;
}

void LinkedList::append(CacheItem* item, int index){
    // Inserts the item into the LinkedList.
    LinkedList* node = this;

    while (node->next != NULL)
    {
        node = node->next;
    }

    LinkedList* newnode = new LinkedList(item);
```

```
    node->next = newnode;
};


CacheItem* LinkedList::getOlderItem(){
    LinkedList* head = this;
    CacheItem* item = NULL;
    int olderItem = 0;
    while(head != NULL){
        if(olderItem < head->item->getAge()){
            item = head->item;
        }
        head = head->next;
    }
    return item;
}

//
void LinkedList::removeFiles(char* nameFile, char* file_path){
    std::string waveFile = std::string(nameFile);
    std::string CsvFile = file_path+waveFile.substr(0,waveFile.length()-
4)+".csv";
    //CSV File
    if (std::remove(CsvFile.c_str())==0){
        wgss_xlog(L_INFO,"[CACHE] - File : %s Deleted",
(waveFile.substr(0,waveFile.length()-4)+".csv").c_str());
    }else{
        wgss_xlog(L_ERR,"[CACHE] - Error Deleting File : %s",
(waveFile.substr(0,waveFile.length()-4)+".csv").c_str());
    }
    //Wave File
    if (std::remove((file_path+std::string(nameFile)).c_str())==0){
        wgss_xlog(L_INFO,"[CACHE] - File : %s Deleted", nameFile);
    }else{
        wgss_xlog(L_ERR,"[CACHE] - Error Deleting File : %s", nameFile);
    }
}

LinkedList* LinkedList::remove(char* key, char* language, char*
voiceName, char* voiceGender, char* file_path){
    LinkedList *prev = this;
    LinkedList *curr = this;
    LinkedList *temp = NULL;

    while(curr != NULL){
        if ((strcmp(curr->item->getKey(), key) == 0)&&
            (strcmp(curr->item->getLanguage(), language) == 0)&&
            (strcmp(curr->item->getVoiceName(), voiceName) == 0)&&
            (strcmp(curr->item->getVoiceGender(), voiceGender) == 0)){
```

```
            if(curr == this){
                if (curr->next != NULL){
                    temp = curr->next;
                    this->removeFiles(curr->item->getWaveFile(),
file_path);
                    delete(curr->item);
                    delete(curr);
                    return temp;
                }else{
                    this->removeFiles(curr->item->getWaveFile(),
file_path);
                    delete(curr->item);
                    delete(curr);
                    return NULL;
                }
            }else if(curr->next == NULL){
                this->removeFiles(curr->item->getWaveFile(), file_path);
                prev->next = NULL;
            }else{
                this->removeFiles(curr->item->getWaveFile(), file_path);
                prev->next = curr->next;
            }
            delete(curr->item);
            delete(curr);
            return this;
        }
        prev = curr;
        curr = curr->next;
    }
    return this;
 }


void LinkedList::printList(int index){
    int Count = 1;
    LinkedList *head = this;
    if (head != NULL){
        wgss_xlog(L_DBG,"[CACHE] - Index : %d    , Key : %s , Language:
%s , Voice Name : %s , Voice Gender : %s  , Wave File : %s , Hash : %s",
index, head->item->getKey(), head->item->getLanguage(), head->item-
>getVoiceName(), head->item->getVoiceGender(), head->item->getWaveFile(),
head->item->getHash());
        while (head != NULL){
            if(head->next != NULL){
                head = head->next;
                wgss_xlog(L_DBG,"[CACHE] -  Collision Num : %d    , Key :
%s , Language: %s , Voice Name : %s , Voice Gender : %s  , Wave File : %s
, Hash : %s", Count, head->item->getKey(), head->item->getLanguage(),
```

```
head->item->getVoiceName(), head->item->getVoiceGender(), head->item-
>getWaveFile(), head->item->getHash());
                Count = Count + 1;
            }else{
                head = head->next;
            }
        }
    }
}

FileStatus LinkedList::search(char* key, char* language, char* voiceName,
char* voiceGender, char** fileName, char* file_path){
    LinkedList* head = this;
    LinkedList* top = this;
    *fileName = NULL;

    if(head == NULL) return FILE_NOT_FOUND;

    while (head != NULL){
        //Validates every Attribute
        if (strcmp(head->item->getKey(), key) == 0 &&
            strcmp(head->item->getLanguage(), language) == 0 &&
            strcmp(head->item->getVoiceName(), voiceName) == 0 &&
            strcmp(head->item->getVoiceGender(), voiceGender) == 0 ){

            //Validate Wave File
            if(strcmp(head->item->getHash(), head->item-
>validateWaveFile()) != 0 ){
                return FILE_CORRUPTED;
            }
            //At this point every Attribute is valid and its returned the
WaveFile
            *fileName = head->item->getWaveFile();
            head->item->setAge();
            return FILE_FOUND;
        }else{
            head = head->next;
        }
    }

    return FILE_NOT_FOUND;
}
```

CacheItem.h

```
#ifndef CACHEITEM_H
#define CACHEITEM_H

#include "cache.h"
#include "linkedlist.h"
#include <iostream>
#include <ctime>

using namespace std;

#define FILE_NAME_SIZE      256

#define FILE_PATH_ITEM "/opt/ptin/windless-mrcp/cache/"
static char file_path_item[FILE_NAME_SIZE] = FILE_PATH_ITEM;

class LinkedList;
class HashTable;

class CacheItem{
    string content;
    string language;
    string voiceName;
    string voiceGender;
    int samplingRate;
    string wavefile;
    string hash;
    int age;

    public:

    CacheItem(string cachename, string content, string language, string
voiceName, string voiceGender, int samplingRate, string hash);
    ~CacheItem(){
        return;
    };

    string getContent();
    string getLanguage();
    string getVoiceName();
    string getVoiceGender();
    int getSamplingRate();
    string getWaveFile();
    string getHash();
    string validateWaveFile();

    void setAge();
```

```
    int getAge();
};

void setCachePathItem(char* path);
string generateMD5(string wavefile);

#endif
```

CacheItem.cc

```
#include "../inc/cacheitem.h"
#include "../inc/apiwgss_logger.h"

#include <iostream>
#include <cstring>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <openssl/md5.h>
#include <ctime>


std::string generateMD5(char* wavefile) {
    //Open Wave File and Gets All data
    char fileName[FILE_NAME_SIZE];
    snprintf(fileName, sizeof(fileName), "%s%s", file_path_item,
wavefile);
    std::ifstream filecache(fileName, std::ifstream::binary);
    if(filecache.is_open()){
        // get length of file:
        filecache.seekg(0 , filecache.end);
        int size = filecache.tellg();
        filecache.seekg(0 , filecache.beg);
        //Set The content of the file in Audio
        char cacheAudio[size];
        filecache.read(cacheAudio, size);
        filecache.close();
        //Md5 Generating
        unsigned char digest[MD5_DIGEST_LENGTH+1];
        MD5(reinterpret_cast<const unsigned char*>(cacheAudio), size,
digest);

        std::ostringstream md5Stream;
        for (int i = 0; i < MD5_DIGEST_LENGTH; ++i) {
            md5Stream << std::hex << std::setw(2) << std::setfill('0') <<
static_cast<int>(digest[i]);
        }
```

```
        return md5Stream.str();
    }else{
        wgss_xlog(L_ERR,"[CACHE] - Error Trying to Open Wave File: %s",
wavefile);
        return "";
    }
    return "";
}

CacheItem::CacheItem(char* keyi, char* languagei, char* voiceNamei, char*
voiceGenderi, int samplingRate, char* wavefilei, char* hashi){
    strcpy(key , keyi);
    strcpy(language , languagei);
    strcpy(voiceName , voiceNamei);
    strcpy(voiceGender , voiceGenderi);
    strcpy(wavefile , wavefilei);
    this->age = time(NULL);
    this->samplingRate = samplingRate;
    if(hashi == NULL){
        std::string md5Result = generateMD5(wavefilei);
        std::string nullString = "";
        if (md5Result.length()==0){
            strcpy(hash , nullString.c_str());
        }else{
            strcpy(hash , md5Result.c_str());
        }
    }else{
        strcpy(hash , hashi);
    }
}

char* CacheItem::getKey(){
    return this->key;
}

char* CacheItem::getLanguage(){
    return this->language;
}

char* CacheItem::getVoiceName(){
    return this->voiceName;
}

char* CacheItem::getVoiceGender(){
    return this->voiceGender;
}

char* CacheItem::getWaveFile(){
    return this->wavefile;
```

```
}

int CacheItem::getSamplingRate(){
    return this->samplingRate;
}

char* CacheItem::getHash(){
    return this->hash;
}

char* CacheItem::validateWaveFile(){
    return (char*)generateMD5(this->wavefile).c_str();
}

//Clean UP
int CacheItem::getAge(){
    return this->age;
}

void CacheItem::setAge(){
    this->age = time(NULL);
}

void setCachePathItem(char* path){
    snprintf(file_path_item, sizeof(file_path_item), "%s", path);
}
```

apiwgss_grpc.cc

```
#include <iostream>

#include <chrono>
#include <fstream>
#include <iostream>
#include <iterator>
#include <string>
#include <thread>
#include <locale>
#include <cstdint>
#include <ctime>

#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <stdio.h>
```

```
#include <libxml/parser.h>
#include <experimental/filesystem>
#include <sys/stat.h>

#include <unistd.h>//usleep

#include "../inc/apiwgss_grpc.h"
#include "../inc/apiwgss_mngt.h"
#include "../inc/apiwgss_logger.h"
#include "../inc/ssml_parser.h"

//Cache Implementation
#include "../inc/cache.h"

//#define TO_CPP(a) (reinterpret_cast<GoogleTextToSpeech*>(a))
//#define TO_C(a)   (reinterpret_cast<googletexttospeech*>(a))

static char cfg_channel_uri[64] = GOOGLE_SERVICE_URI;
static char cfg_license_path[256] = GOOGLE_LICENSE_PATH;

static char default_language[6] = DEFAULT_LANGUAGE;
static char default_voice_name[32] = DEFAULT_VOICE_NAME;
static char  default_gender[16] = DEFAULT_VOICE_GENDER;

static bool save_wave_file = SAVE_WAVE_FILE;
static char wave_file_path[256] = WAVE_FILE_PATH;


static int default_sample_rate = 8000;
static AudioEncoding default_encoding = AudioEncoding::LINEAR16;

static char cfg_proxy[64];
//Result callback
static result_cb_t result_cb;

//Bypass SSML
static int bypass_ssml = BYPASS_SSML;

thread apiwgss_mngt_th;

//Cache
static bool enable_cache = CACHE_ENABLE;
static int cache_size = CACHE_HASHTABLE_SIZE;
static int cache_max_item = CACHE_MAX_ITEMS;
static char cache_path[256] = CACHE_PATH;
static int cache_cleanup_percentage = CACHE_CLEANUP_PERCENTAGE;

HashTable* Cache;// = new HashTable(cache_size, cache_max_item);
```

```
void apiwgss_grpc_initialize(char *endpoint, char *license, char
*language, char *gender, char *voice_name, int save_file, char
*file_path)
{
    wgss_xlog(L_INFO,"----------- GoogleTextToSpeech LIB initialize -----
------");
    if(endpoint!=NULL) snprintf(cfg_channel_uri, sizeof(cfg_channel_uri),
"%s", endpoint);
    wgss_xlog(L_INFO,"Google endpoint: %s", cfg_channel_uri);
    if(license!=NULL) snprintf(cfg_license_path,
sizeof(cfg_license_path), "%s", license);
    wgss_xlog(L_INFO,"License Path: %s", cfg_license_path);
    if(language!=NULL) snprintf(default_language,
sizeof(default_language), "%s", language);
    wgss_xlog(L_INFO,"Language: %s", default_language);
    if(gender!=NULL) snprintf(default_gender, sizeof(default_gender),
"%s", gender);
    wgss_xlog(L_INFO,"Voice Gender: %s", default_gender);;
    if(voice_name!=NULL) snprintf(default_voice_name,
sizeof(default_voice_name), "%s", voice_name);
    wgss_xlog(L_INFO,"Voice Name: %s", default_voice_name);
    // Wave File
    if(save_file==0 || save_file==1)  save_wave_file=save_file;
    wgss_xlog(L_INFO,"Save Wave File: %s",
save_wave_file?"TRUE":"FALSE");
    if(file_path!=NULL) snprintf(wave_file_path, sizeof(wave_file_path),
"%s", file_path);
    wgss_xlog(L_INFO,"Wave File Path: %s", wave_file_path);
    wgss_xlog(L_INFO,"-------------------------------------------------
------");

    /*
     * this initialize the library and check potential ABI mismatches
     * between the version it was compiled for and the actual shared
     * library used.
     */
    LIBXML_TEST_VERSION

    wgss_xlog(L_INFO,"Start thread GRPC management");
    apiwgss_mngt_th = thread(apiwgss_mngt_thread);
}

GoogleTextToSpeech::GoogleTextToSpeech(char *session_id, int size) {
    sessionID.assign(session_id, size);

    if(strlen(cfg_proxy) > 0)
    {
        setenv("grpc_proxy", cfg_proxy, 1);
        setenv("http_proxy", cfg_proxy, 1);
```

```
        setenv("https_proxy", cfg_proxy, 1);
    }

    synthesize_pending=false;
    synthesize_stopped=false;

    // Create a Speech Stub connected to the speech service.
    std::ifstream stream(cfg_license_path);
    if (!stream.is_open()) {
        wgss_xlog(L_ERR,"Session: %s - Cannot open license file: %s",
sessionID.c_str(), cfg_license_path);
    }
    std::string json_key(std::istreambuf_iterator<char>{stream}, {});
    auto call_credentials =
grpc::ServiceAccountJWTAccessCredentials(json_key, 6000);
    auto channel_credentials =
grpc::SslCredentials(grpc::SslCredentialsOptions());
    auto credentials =
grpc::CompositeChannelCredentials(channel_credentials, call_credentials);

    if(credentials)
    {
        wgss_xlog(L_DBG,"Session: %s - Have Credentials",
sessionID.c_str());
        wgss_xlog(L_INFO,"Session: %s - Create gRPC Channel",
sessionID.c_str());
        channel = grpc::CreateChannel(cfg_channel_uri, credentials);
        wgss_xlog(L_INFO,"Session: %s - Create gRPC Speech Stub",
sessionID.c_str());
        textToSpeech = TextToSpeech::NewStub(channel);

        request=new SynthesizeSpeechRequest();
        input = request->mutable_input();
        voice = request->mutable_voice();
        audio_config = request->mutable_audio_config();

        // Set default values
        voice->set_language_code(default_language);
        setGender(default_gender);
        voice->set_name(default_voice_name);

        audio_config->set_audio_encoding(default_encoding);
        audio_config->set_sample_rate_hertz(default_sample_rate);

        // runThread_open_synthesize_speech();
    }
    else
    {
```

```
        wgss_xlog(L_DBG,"Session: %s - Don't have Credentials",
sessionID.c_str());
    }
}

GoogleTextToSpeech::~GoogleTextToSpeech()
{
    finish_all_threads();

    delete request;
    request = NULL;
}


grpc_connectivity_state GoogleTextToSpeech::connectivity_state(){
    if(Cache->foundCacheItem){
        return GRPC_CHANNEL_READY;
    }

    if(channel){
        state = channel->GetState (0);
        /* switch(state){
            case GRPC_CHANNEL_IDLE:
                wgss_xlog(L_DBG,"Session: %s - Channel State - IDLE",
sessionID.c_str());
                break;//return false;
            case GRPC_CHANNEL_CONNECING:
                wgss_xlog(L_DBG,"Session: %s - Channel State -
CONNECING", sessionID.c_str());
                break;//return false;
            case GRPC_CHANNEL_READY:
                wgss_xlog(L_DBG,"Session: %s - Channel State - READY",
sessionID.c_str());
                wgss_xlog(L_DBG,"Session: %s - Connected with Google
Speech", sessionID.c_str(), state);
                break;//return true;
            case GRPC_CHANNEL_TRANSIENT_FAILURE:
                wgss_xlog(L_DBG,"Session: %s - Channel State -
TRANSIENT_FAILURE", sessionID.c_str());
                break;//return false;
            case GRPC_CHANNEL_SHUTDOWN:
                wgss_xlog(L_DBG,"Session: %s - Channel State -
TRANSIENT_SHUTDOWN", sessionID.c_str());
                break;//return false;
            default:
                wgss_xlog(L_DBG,"Session: %s - Channel State - UNKNOWN",
sessionID.c_str());
                break;//return false;
        } */
```

```
    }
    return state;

}

/*void GoogleTextToSpeech::create_context()
{
    if(context) delete context;
    context = new grpc::ClientContext();
}*/

void GoogleTextToSpeech::synthesize()
{
    grpc::ClientContext context;
    wgss_xlog(L_INFO, "Session: %s - Request to Synthesize Speech",
sessionID.c_str());

    if(enable_cache){
        if (Cache->search() != -1){
            wgss_xlog(L_INFO, "[CACHE] - Found Content in Cache While
Trying to Synthesize %s", Cache->cacheWaveFile);
            (*result_cb)(sessionID.c_str(), GRPC_SYNTH_OK);
            return;
        }else{
            synthesize_num++;
            synthesize_pending=true;
            status = textToSpeech->SynthesizeSpeech(&context, *request,
&response);
            synthesize_pending=false;
            wgss_xlog(L_DBG, "Session: %s - Request to Synthesize Speech
was finished", sessionID.c_str());

            if(status.error_code() == 0){
                if(synthesize_stopped)
                {
                    wgss_xlog(L_WARN, "Session: %s - Synthesize has been
canceled", sessionID.c_str());
                    return;
                }

                //Check Disk Space
                const std::experimental::filesystem::space_info si =
std::experimental::filesystem::space(file_path);
                if(response.audio_content().length() >
static_cast<std::intmax_t>(si.free)){
                    wgss_xlog(L_DBG, "[CACHE] - Enough Disk To add Cache
File");
                    string fileName = "cache_" +  sessionID + "_" +
to_string(synthesize_num);
```

```
                    wgss_xlog(L_INFO, "[CACHE] - Save wave file [%s%s]",
sessionID.c_str(), cache_path, (fileName+std::string(".wav")).c_str());

                    ofstream output(wave_file_path +
(fileName+std::string(".wav")));
                    output << response.audio_content();

                    //Cache test Implementation Dentro de if para
validação
                    snprintf(Cache->cacheWaveFile, sizeof(Cache-
>cacheWaveFile), "%s", ((char*)(fileName+std::string(".wav")).c_str()));
                    Cache->insert((char*)(fileName).c_str());

                    (*result_cb)(sessionID.c_str(), GRPC_SYNTH_OK);
                }else{
                    wgss_xlog(L_ERR, "[CACHE] - No Disk Space Available
to Add Cache Items");
                }
            }
            else{
                wgss_xlog(L_ERR, "Session: %s - Status: %d %s",
sessionID.c_str(), status.error_code(), status.error_message().c_str());
                (*result_cb)(sessionID.c_str(), GRPC_SYNTH_NOK);
            }
        }
    }else{
        synthesize_pending=true;
        status = textToSpeech->SynthesizeSpeech(&context, *request,
&response);
        synthesize_pending=false;
        wgss_xlog(L_DBG, "Session: %s - Request to Synthesize Speech was
finished", sessionID.c_str());

        if(status.error_code() == 0){
            if(synthesize_stopped)
            {
                wgss_xlog(L_WARN, "Session: %s - Synthesize has been
canceled", sessionID.c_str());
                return;
            }
            if (save_wave_file){
                string fileName = "wgss_" +  sessionID + "_" +
to_string(synthesize_num) +".wav";
                wgss_xlog(L_INFO, "Session: %s - Save wave file [%s%s]",
sessionID.c_str(), wave_file_path, fileName.c_str());

                ofstream output(wave_file_path + fileName);
                output << response.audio_content();
            }
```

```
            (*result_cb)(sessionID.c_str(), GRPC_SYNTH_OK);
        }else{
            wgss_xlog(L_ERR, "Session: %s - Status: %d %s",
sessionID.c_str(), status.error_code(), status.error_message().c_str());
            (*result_cb)(sessionID.c_str(), GRPC_SYNTH_NOK);
        }
    }

}

static void synthesize_speech_thread(GoogleTextToSpeech *gss)
{
    wgss_xlog(L_DBG,"Session: %s - synthesize thread start", gss-
>get_sessionId().c_str());
    gss->synthesize();
    wgss_xlog(L_DBG,"Session: %s - synthesize thread has been finished",
gss->get_sessionId().c_str());
}

void GoogleTextToSpeech::synthesizeSpeech(){
    Cache->foundCacheItem = 0;
    if(synthesize_pending)
    {
        wgss_xlog(L_WARN,"Session: %s - synthesize request is pending",
sessionID.c_str());
        synthesize_stopped=true;
        return;
    }
    synthesize_stopped=false;
    finish_all_threads();
    mtx.lock();
    if(!thread_synthesize_speech_ptr){
        thread_synthesize_speech = thread(synthesize_speech_thread,
this);
        thread_synthesize_speech_ptr = &thread_synthesize_speech;
    }
    mtx.unlock();


    // runThread_open_synthesize_speech();
    // Create new 'context' for each synthesize request
/*  grpc::ClientContext context;

    status = textToSpeech->SynthesizeSpeech(&context, *request,
&response);
    wgss_xlog(L_DBG, "Session: %s - Status: %d %s", sessionID.c_str(),
status.error_code(), status.error_message().c_str());
*/
```

```
    // if (status.error_code() != 0){
    //  grpc::ClientContext context;
    //  voice->set_language_code("en-US");
    //  input->set_text(status.error_message());
    //  textToSpeech->SynthesizeSpeech(&context, request, &response);
    // }

/*  if (save_wave_file){

////////////////////////////////////////////////////////////////
        ////////////// Save audio in file for testing purposes
///////////////
        string fileName = "tts_output_" +  sessionID + ".wav";
        ofstream output(wave_file_path + fileName);
        output << response.audio_content();

////////////////////////////////////////////////////////////////
    }
*/
}

void GoogleTextToSpeech::synthesizeStop(){
    wgss_xlog(L_INFO, "Session: %s - Stop synthesize session",
sessionID.c_str());
    synthesize_stopped=true;
}

void GoogleTextToSpeech::finish_all_threads()
{
    wgss_xlog(L_DBG,"Session: %s - Checking for threads",
sessionID.c_str());
    mtx.lock();
    if(thread_synthesize_speech_ptr)
    {
        mtx.unlock();
        wgss_xlog(L_DBG,"Session: %s - Waiting synthesize thread finish",
sessionID.c_str());
        if(thread_synthesize_speech_ptr->joinable())
            thread_synthesize_speech_ptr->join();
        mtx.lock();
        thread_synthesize_speech_ptr = NULL;
        wgss_xlog(L_DBG,"Session: %s - synthesize thread finished",
sessionID.c_str());
    }

    mtx.unlock();
}

void GoogleTextToSpeech::listVoiceNames(string lang){
```

```
    voicesRequest.set_language_code(lang);
    textToSpeech->ListVoices(&voicesContext, voicesRequest,
&voicesResponse);
    auto voices = voicesResponse.voices();

    for (int i = 0; i < voicesResponse.voices_size(); i++) {
        cout << "voice " << i << ": " << voices[i].name() << endl;
    }
}

void GoogleTextToSpeech::setText(string text){
    input->clear_ssml();
    input->set_text(text);
    //Cache
    Cache->setContent((char*)text.c_str());
}

void GoogleTextToSpeech::setSSML(string ssml){
    char lang[6];
    input->clear_text();

    if(!bypass_ssml){
        wgss_xlog(L_DBG,"Session: %s - SSML Parser - Get attribute lang",
sessionID.c_str());
        if( !xmlGetAttributeFromElement(ssml.c_str(), "speak", "lang",
lang, sizeof(lang)) )
        {
            wgss_xlog(L_DBG,"Session: %s - lang attribute found in SSML",
sessionID.c_str());
            wgss_xlog(L_INFO, "Session: %s - Set language [%s]",
sessionID.c_str(), lang);
            setLanguage(lang);
        }else{
            wgss_xlog(L_DBG,"Session: %s - No lang attribute found in
SSML");
        }
    }

    input->set_ssml(ssml);
    //Cache
    Cache->setSSML(ssml);
}

void GoogleTextToSpeech::setLanguage(string lang){
    voice->set_language_code(lang);
    //cache
    Cache->setLanguage((char*)lang.c_str());
}
```

```
void GoogleTextToSpeech::setVoiceName(string voiceName){
    voice->set_name(voiceName);
    //cache
    Cache->setVoiceName((char*)voiceName.c_str());
}

void GoogleTextToSpeech::setGender(string gender){
    voice->clear_name();
    if (gender == "male")
        voice->set_ssml_gender(SsmlVoiceGender::MALE);
    else if (gender == "female")
        voice->set_ssml_gender(SsmlVoiceGender::FEMALE);
    else
        voice-
>set_ssml_gender(SsmlVoiceGender::SSML_VOICE_GENDER_UNSPECIFIED);
    //Cache
    Cache->setVoiceGender((char*)gender.c_str());
}

void GoogleTextToSpeech::setAudioEncoding(string encoding){
    if (encoding == "LPCM")
        audio_config->set_audio_encoding(AudioEncoding::LINEAR16);
    else if (encoding == "MP3")
        audio_config->set_audio_encoding(AudioEncoding::MP3);
    else if (encoding == "OGG_OPUS")
        audio_config->set_audio_encoding(AudioEncoding::OGG_OPUS);
    else
        audio_config->set_audio_encoding(default_encoding);
}

void GoogleTextToSpeech::setSampleRate(int sampleRate){
    int supportedSamplingRates[4] = {8000, 16000, 32000, 48000};
    for (int i = 0; i < 4; i++){
        if (sampleRate == supportedSamplingRates[i])
            audio_config->set_sample_rate_hertz(sampleRate);
        else
            audio_config->set_sample_rate_hertz(default_sample_rate);
    }

    //Cache
    Cache->setSamplingRate(sampleRate);
}

void GoogleTextToSpeech::getAudio(char **audio, int *size){
    //TODO review
    if (Cache->foundCacheItem && enable_cache){
        wgss_xlog(L_DBG, "Session: %s - Get Audio Cache",
sessionID.c_str());
        if(Cache->setAudio(&audio, &size, Cache->cacheWaveFile)){
```

```
            wgss_xlog(L_DBG, "Session: %s - END Get Audio Cache",
sessionID.c_str());
        }else{
            wgss_xlog(L_ERR, "Session: %s - Cant Open Cache File",
sessionID.c_str());
        }
    }else{
        wgss_xlog(L_DBG, "Session: %s - Get Audio", sessionID.c_str());
        while((int)response.audio_content().length() == 0);
        wgss_xlog(L_DBG, "Session: %s - Get Audio while",
sessionID.c_str());
        *size = (int)response.audio_content().length();
        audioContent = response.audio_content();
        *audio = (char *)audioContent.c_str();
        wgss_xlog(L_DBG, "Session: %s - END Get Audio",
sessionID.c_str());
    }

    //Cache KPI
    if(enable_cache){
        Cache->print_table();
        wgss_xlog(L_INFO, "[CACHE] - KPI Cache           : %d", Cache-
>getCacheUses());
        wgss_xlog(L_INFO, "[CACHE] - KPI Google Services : %d", Cache-
>getGoogleUses());
    }
}


void apiwgss_grpc_proxy(const char* proxy){ snprintf(cfg_proxy,
sizeof(cfg_proxy), "http://%s", proxy);}

void apiwgss_grpc_register_result_callback(result_cb_t cb) {
    wgss_xlog(L_INFO,"Result callback registered");
    result_cb = cb;
}

void apiwgss_grpc_bypass_ssml(int bypass){
    wgss_xlog(L_INFO,"Set bypass_ssml [%s]", bypass?"true":"false");
    bypass_ssml=bypass;
}

void apiwgss_grpc_cache(int enable, char* path, int size, int max_items,
int cleanup_percentage){
    struct stat buffer;
    if(stat(path, &buffer) == 0){
        wgss_xlog(L_INFO,"Set Cache [%s]", enable?"true":"false");
        wgss_xlog(L_INFO,"Set Cache Hash Table Size [%s]", size);
        wgss_xlog(L_INFO,"Set Cache Max Items [%s]", max_items);
```

```
        wgss_xlog(L_INFO,"Set Cache Clean Up Percentage [%s]",
cleanup_percentage);
        enable_cache = enable;
        cache_size = size;
        cache_max_item = max_items;
        cache_cleanup_percentage = cleanup_percentage;
        snprintf(cache_path, sizeof(cache_path), "%s", path);
    }else{
        wgss_xlog(L_ERR,"Path to Cache is Invalid");
        enable_cache = 0;
    }
    if(enable){
        Cache = new HashTable(cache_size, cache_max_item);
        setCachePath(cache_path);
        setCleanUpPercentage(cache_cleanup_percentage);
        //Load Hash Table
        Cache->loadCache();
        Cache->setLanguage(default_language);
        Cache->setVoiceName(default_voice_name);
        Cache->setVoiceGender(default_gender);
    }
}
```