Levva

Título do documento		
/ tipo Documentação técnica	**/área responsável** Tecnologia	**/ data** 18/06/2025

Sumário

1. Introdução

Esta documentação técnica tem como objetivo fornecer um guia completo e detalhado que permita a um desenvolvedor experiente reconstruir a funcionalidade da API CROSS a partir do zero. Ela descreve a funcionalidade de cada endpoint, seus objetivos, fluxos e regras de negócio implementadas, os modelos de dados envolvidos (entidades, relacionamentos, estruturas) e exemplos de pseudocódigo ou trechos explicativos baseados na lógica inferida do sistema DataFlex original.

2. Visão Geral da Funcionalidade CROSS

O sistema CROSS, através desta API, gerencia informações cruciais para operações logísticas e comerciais, focando em dois domínios principais:

- Gestão de Taxas (TaxController): Permite a consulta, filtragem e gerenciamento de diversas taxas aplicáveis em diferentes etapas do processo logístico (origem, frete, destino). Isso inclui identificar a quais classes uma taxa pertence, listar todas as taxas, buscar taxas por nome ou classe, e obter taxas associadas a propostas específicas.
- Gestão de Cidades (CityController): Oferece funcionalidades para buscar e validar informações sobre cidades, países e rotas. Isso é essencial para operações de comércio internacional, permitindo a seleção de cidades de origem, destino, pontos de via, e a obtenção de descrições formatadas e informações de país.

A API é projetada para ser consumida por outras aplicações que necessitam dessas informações para processos como cotação, criação de propostas, e gerenciamento de dados mestres.

3. Validação de Sessão do Usuário

Muitos endpoints da API CROSS requerem validação da sessão do usuário para garantir a segurança e o controle de acesso. O processo geral é:

1. O endpoint recebe um parâmetro userSession (string).
2. O sistema consulta a tabela CSAG311 para verificar se o userSession fornecido existe e está ativo.

3.1. Pseudocódigo para Validação de Sessão:

```
FUNCTION ValidarSessao(userSessionToken AS STRING) RETURNS BOOLEAN

DECLARE sessaoEncontrada AS BOOLEAN = FALSE

// Conectar ao banco de dados

DB_CONNECT()

// Consultar tabela de sessões

QUERY_RESULT = DB_EXECUTE_SQL("SELECT 1 FROM CSAG311 WHERE UsuarioSessao = '" + userSessionToken + "' AND ATIVO = 'S'")
```

```
IF DB_HAS_ROWS(QUERY_RESULT) THEN

sessaoEncontrada = TRUE

END IF

DB_CLOSE()

RETURN sessaoEncontrada

END FUNCTION
```

3. Se a sessão for inválida, o endpoint geralmente retorna uma resposta de erro (ex: 401 Unauthorized) ou uma lista vazia, interrompendo o processamento adicional.
4. Se a sessão for válida, o processamento do endpoint continua.

3.2. Estrutura da tabela CSAG311 (Sessão de Usuário)

Campo	Tipo	Descrição	Chave	Relacionamentos
UsuarioSessao	VARCHAR(50)	Identificador único da sessão	PK	-
ID_USUARIO	VARCHAR(20)	Código do usuário associado à sessão	FK	`CSAG310.ID_USUARIO`
DT_INICIO	DATETIME	Data e hora de início da sessão	-	-

DT_ULTIMO_ACESSO	DATETIME	Data e hora do último acesso	-	-
IP_ORIGEM	VARCHAR(15)	Endereço IP de origem da sessão	-	-
ATIVO	CHAR(1)	Indicador se a sessão está ativa (“S” para Sim, “N” para Não)	-	-

4. Endpoints de Taxas (TaxController)

4.1. GET api/Tax/selected-classes

Descrição da Funcionalidade e Objetivo: Este endpoint tem como objetivo identificar e retornar as classes de taxa (Origem, Frete, Destino) que estão especificamente associadas a um determinado ID de taxa. Isso é útil para entender em quais etapas do processo logístico uma taxa particular é aplicável.

Método DataFlex Correspondente (Inferido): f_classesSelTaxa

Fluxos e Regras de Negócio:

1. Receber userSession e taxId como parâmetros.
2. Validar Sessão do Usuário: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.
3. Consultar a tabela HCGS3001 para encontrar o registro correspondente ao taxId fornecido.
4. Se o taxId não for encontrado em HCGS3001, retornar uma lista vazia.
5. Obter o valor do campo CLASSE_TAXA da tabela HCGS3001. Este campo contém uma string com os códigos das classes concatenados (ex: “OF”, “D”, “OFD”).
6. Separar os códigos de classe individuais da string CLASSE_TAXA.
7. Para cada código de classe obtido, consultar a tabela CCGS221 para obter a descrição correspondente da classe.
8. Retornar uma lista de objetos TaxClass, cada um contendo o código (ID) e a descrição (DS) da classe de taxa.

4.1.1. Modelos de Dados Envolvidos

- Requisição:
  - userSession (string, query param)
  - taxId (string, query param)
- Resposta: IEnumerable<TaxClass>

```
public class TaxClass
{
    public string? ID { get; set; }  // Código da classe de taxa (ex: 'O',
```

'F', 'D')

public string? DS { get; set; } // Descrição da classe de taxa (ex: 'Origem', 'Frete', 'Destino')

}

- Tabelas do Banco de Dados:

Tabela	**Campo**	**Tipo**	**Descrição**	**Chave**	**Relacionamentos**
HCGS3001 (Taxas)	ID_TAXA	VARCHAR(10)	Código único da taxa	PK	HCGS3006.ID_TAXA
HCGS3001 (Taxas)	NM_TAXA	VARCHAR(100)	Nome/descrição da taxa	-	-
HCGS3001 (Taxas)	CLASSE_TAXA	VARCHAR(10)	Classes associadas (concatenadas, ex: “OFD”)	-	Referência lógica a CCGS221.CLASSE
CCGS221 (Classes de Taxa)	CLASSE	CHAR(1)	Código da classe (“O”, “F”, “D”)	PK	-
CCGS221 (Classes de Taxa)	DESCRICAO	VARCHAR(50)	Descrição da classe	-	-

4.1.2. Pseudocódigo para Reconstrução

```
FUNCTION ValidarSessao(userSessionToken AS STRING) RETURNS BOOLEAN
  DECLARE sessaoEncontrada AS BOOLEAN = FALSE
```

```
  // Conectar ao banco de dados
  DB_CONNECT()

  // Consultar tabela de sessões
  QUERY_RESULT = DB_EXECUTE_SQL("SELECT 1 FROM CSAG311 WHERE
UsuarioSessao = '" + userSessionToken + "' AND ATIVO = 'S'")

  IF DB_HAS_ROWS(QUERY_RESULT) THEN
    sessaoEncontrada = TRUE
  END IF

  DB_CLOSE()

  RETURN sessaoEncontrada
END FUNCTION
```

4.2. GET api/Tax/classes

Descrição da Funcionalidade e Objetivo: Este endpoint retorna a lista padrão e fixa de todas as classes de taxa disponíveis no sistema (Origem, Frete, Destino). É usado para preencher opções de filtro ou seleção em interfaces de usuário.

Método DataFlex Correspondente (Inferido): f_classes

Fluxos e Regras de Negócio:

1. Este endpoint não requer validação de sessão.
2. Retorna uma lista predefinida (hardcoded) de classes de taxa.
   - Classe 'O': Descrição 'Origem'
   - Classe 'F': Descrição 'Frete'
   - Classe 'D': Descrição 'Destino'
3. A ordem de retorno é geralmente fixa (Origem, Frete, Destino).

4.2.1. Modelos de Dados Envolvidos

- **Requisição:** Nenhuma
- **Resposta:** IEnumerable<IdValueClassItem>

```
public class IdValueClassItem
{
    public string? Id { get; set; }// Código da classe ('O', 'F', 'D')
    public string? Value { get; set; }// Nome da classe ('Origem', 'Frete', 'Destino')
    public string? Classe { get; set; }// Código da classe (pode ser
```

redundante com Id)

}

- **Tabelas do Banco de Dados:** Nenhuma consulta direta é feita para este endpoint, pois os dados são fixos. A tabela CCGS221 serve como referência conceitual.

4.2.2. Pseudocódigo para Reconstrução

```
FUNCTION GetTaxClasses() RETURNS LIST OF IdValueClassItem
  DECLARE taxClassesList AS NEW LIST OF IdValueClassItem

  // Classe Origem
  origemClass AS NEW IdValueClassItem
  origemClass.Id = "O"
  origemClass.Value = "Origem"
  origemClass.Classe = "O"
  taxClassesList.ADD(origemClass)

  // Classe Frete
  freteClass AS NEW IdValueClassItem
  freteClass.Id = "F"
  freteClass.Value = "Frete"
  freteClass.Classe = "F"
  taxClassesList.ADD(freteClass)

  // Classe Destino
  destinoClass AS NEW IdValueClassItem
  destinoClass.Id = "D"
  destinoClass.Value = "Destino"
  destinoClass.Classe = "D"
  taxClassesList.ADD(destinoClass)
```

```
  RETURN                                                    taxClassesList
END FUNCTION
```

4.3. GET api/Tax/list

Descrição da Funcionalidade e Objetivo: Este endpoint tem como objetivo fornecer uma lista completa de todos os códigos e nomes de taxas cadastradas no sistema. É comumente utilizado para popular listas de seleção (combos) em interfaces de usuário, permitindo que o usuário escolha uma taxa específica.

Método DataFlex Correspondente (Inferido): f_lista_HCGS3001

Fluxos e Regras de Negócio:

1. Receber userSession como parâmetro.
2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.
3. Consultar a tabela HCGS3001 para obter todos os registros de taxas.
4. Apenas taxas ativas (onde o campo ATIVO é 'S', por exemplo) devem ser consideradas.
5. Para cada taxa ativa, extrair o ID_TAXA (código da taxa) e NM_TAXA (nome da taxa).

6. Os resultados devem ser ordenados alfabeticamente pelo NM_TAXA para facilitar a busca pelo usuário.
7. Retornar uma lista de objetos ComboItem, cada um contendo o código (ID) e o nome (DS) da taxa.

4.3.1. Modelos de Dados Envolvidos

- **Requisição:**
  - userSession (string, query param)
- **Resposta:** IEnumerable<ComboItem>

```
public class ComboItem
{
  public string? ID { get; set; }   // Código da taxa
  public string? DS { get; set; }   // Nome/descrição da taxa
}
```

- **Tabelas do Banco de Dados:**
  - HCGS3001 (Taxas) - *Estrutura detalhada anteriormente (Seção 4.1). Campos relevantes: ID_TAXA, NM_TAXA, ATIVO.*
  - CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

4.3.2. Pseudocódigo para Reconstrução

```
FUNCTION GetTaxList(userSessionToken AS STRING) RETURNS LIST OF
ComboItem
  DECLARE taxList AS NEW LIST OF ComboItem

  IF NOT ValidarSessao(userSessionToken) THEN
    RETURN taxList // Retorna lista vazia se sessão inválida
```

```
  END IF

  DB_CONNECT()

  // Buscar todas as taxas ativas, ordenadas pelo nome
  QUERY_TAXAS = DB_EXECUTE_SQL("SELECT ID_TAXA, NM_TAXA FROM HCGS3001
WHERE ATIVO = 'S' ORDER BY NM_TAXA")

  WHILE DB_HAS_NEXT_ROW(QUERY_TAXAS)
    taxCode AS STRING = DB_GET_FIELD(QUERY_TAXAS, "ID_TAXA")
    taxName AS STRING = DB_GET_FIELD(QUERY_TAXAS, "NM_TAXA")

    newItem AS NEW ComboItem
    newItem.ID = taxCode
    newItem.DS = taxName
    taxList.ADD(newItem)
  END WHILE

  DB_CLOSE()
  RETURN taxList
END FUNCTION
```

4.4. GET api/Tax/all-with-classes

Descrição da Funcionalidade e Objetivo: Este endpoint busca todas as taxas cadastradas e, para cada uma, retorna suas classes associadas (Origem, Frete, Destino) de forma concatenada. O objetivo é fornecer uma visão geral de todas as taxas e suas aplicabilidades, útil para relatórios ou configurações gerais do sistema.

Método DataFlex Correspondente (Inferido): buscarTaxas (ou similar que retorne todas as taxas com suas classes)

Fluxos e Regras de Negócio:

1. Receber userSession como parâmetro.
2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.
3. Consultar a tabela HCGS3001 para obter todos os registros de taxas.
4. Apenas taxas ativas (onde o campo ATIVO é 'S', por exemplo) devem ser consideradas.
5. Para cada taxa ativa, extrair o ID_TAXA (código da taxa), NM_TAXA (nome da taxa) e CLASSE_TAXA (string concatenada das classes, ex: "OFD").
6. Os resultados devem ser ordenados alfabeticamente pelo NM_TAXA.
7. Retornar uma lista de objetos IdValueClassItem, cada um contendo o código (Id), nome (Value) e a string de classes (Classe) da taxa.

4.4.1. Modelos de Dados Envolvidos

- **Requisição:**
  - userSession (string, query param)
- **Resposta:** IEnumerable<IdValueClassItem>

```
public class IdValueClassItem
{
  public string? Id { get; set; }      // Código da taxa
  public string? Value { get; set; }   // Nome da taxa
```

```
    public string? Classe { get; set; } // Classes associadas (concatenadas, ex: "OFD")
}
```

- **Tabelas do Banco de Dados:**
  - HCGS3001 (Taxas) - *Estrutura detalhada anteriormente (Seção 4.1). Campos relevantes: ID_TAXA, NM_TAXA, CLASSE_TAXA, ATIVO.*
  - CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

4.4.2. Pseudocódigo para Reconstrução

```
FUNCTION GetAllTaxesWithClasses(userSessionToken AS STRING) RETURNS LIST OF IdValueClassItem

DECLARE allTaxesWithClassesList AS NEW LIST OF IdValueClassItem

IF NOT ValidarSessao(userSessionToken) THEN

RETURN allTaxesWithClassesList // Retorna lista vazia se sessão inválida

END IF

DB_CONNECT()

// Buscar todas as taxas ativas com suas classes, ordenadas pelo nome

QUERY_TAXAS = DB_EXECUTE_SQL("SELECT ID_TAXA, NM_TAXA, CLASSE_TAXA FROM HCGS3001 WHERE ATIVO = 'S' ORDER BY NM_TAXA")

WHILE DB_HAS_NEXT_ROW(QUERY_TAXAS)

taxCode AS STRING = DB_GET_FIELD(QUERY_TAXAS, "ID_TAXA")

taxName AS STRING = DB_GET_FIELD(QUERY_TAXAS, "NM_TAXA")

taxClassesString AS STRING = DB_GET_FIELD(QUERY_TAXAS, "CLASSE_TAXA")
```

```
newItem AS NEW IdValueClassItem

newItem.Id = taxCode

newItem.Value = taxName

newItem.Classe = taxClassesString

allTaxesWithClassesList.ADD(newItem)

END WHILE

DB_CLOSE()

RETURN allTaxesWithClassesList

END FUNCTION
```

4.5. GET api/Tax/search/by-class

Descrição da Funcionalidade e Objetivo: Este endpoint permite buscar taxas filtrando opcionalmente por uma classe de taxa específica (Origem, Frete ou Destino). Se nenhuma classe for fornecida, ele pode retornar todas as taxas ou um subconjunto padrão. O objetivo é

permitir que os usuários encontrem rapidamente taxas aplicáveis a uma determinada etapa do processo logístico.

Método DataFlex Correspondente (Inferido): fSerchComboTaxes (ou similar que busca taxas com filtro de classe)

Fluxos e Regras de Negócio:

1. Receber userSession e, opcionalmente, classId como parâmetros.
2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.
3. Construir uma consulta à tabela HCGS3001.
4. Apenas taxas ativas (onde o campo ATIVO é 'S', por exemplo) devem ser consideradas.
5. Se classId for fornecido e não for vazio:
   - Filtrar os resultados para incluir apenas taxas onde o campo CLASSE_TAXA contenha o classId fornecido (ex: CLASSE_TAXA LIKE '%F%' se classId for 'F').
6. Se classId não for fornecido ou for vazio:
   - Retornar todas as taxas ativas (ou um comportamento padrão definido, como não retornar nada se o filtro é esperado).
7. Para cada taxa correspondente, extrair ID_TAXA (código), NM_TAXA (nome) e CLASSE_TAXA (string concatenada das classes).

8. Os resultados devem ser ordenados alfabeticamente pelo NM_TAXA.
9. Retornar uma lista de objetos IdValueClassItem.

4.5.1. Modelos de Dados Envolvidos

- **Requisição:**
  - userSession (string, query param, obrigatório)
  - classId (string, query param, opcional): Código da classe para filtrar (ex: 'O', 'F', 'D')
- **Resposta:** IEnumerable<IdValueClassItem>

```
public class IdValueClassItem
{
  public string? Id { get; set; }          // Código da taxa
  public string? Value { get; set; }       // Nome da taxa
  public string? Classe { get; set; } // Classes associadas (concatenadas)
}
```

- **Tabelas do Banco de Dados:**
  - HCGS3001 (Taxas) - *Estrutura detalhada anteriormente (Seção 4.1). Campos relevantes: ID_TAXA, NM_TAXA, CLASSE_TAXA, ATIVO.*
  - CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

4.5.2. Pseudocódigo para Reconstrução

```
FUNCTION SearchTaxesByClass(userSessionToken AS STRING, filterClassId AS STRING) RETURNS LIST OF IdValueClassItem

DECLARE filteredTaxesList AS NEW LIST OF IdValueClassItem

IF NOT ValidarSessao(userSessionToken) THEN

RETURN filteredTaxesList // Retorna lista vazia se sessão inválida

END IF

DB_CONNECT()

sqlQuery AS STRING = "SELECT ID_TAXA, NM_TAXA, CLASSE_TAXA FROM HCGS3001 WHERE ATIVO = 'S'"

IF filterClassId IS NOT NULL AND filterClassId IS NOT EMPTY THEN

// Adiciona filtro por classe se classId for fornecido

// A implementação exata do LIKE depende do SGBD (ex: CONCAT('%', filterClassId, '%') ou similar)

sqlQuery = sqlQuery + " AND CLASSE_TAXA LIKE '%" + filterClassId + "%'"

END IF

sqlQuery = sqlQuery + " ORDER BY NM_TAXA"

QUERY_TAXAS = DB_EXECUTE_SQL(sqlQuery)

WHILE DB_HAS_NEXT_ROW(QUERY_TAXAS)

taxCode AS STRING = DB_GET_FIELD(QUERY_TAXAS, "ID_TAXA")
```

```
taxName AS STRING = DB_GET_FIELD(QUERY_TAXAS, "NM_TAXA")

taxClassesString AS STRING = DB_GET_FIELD(QUERY_TAXAS, "CLASSE_TAXA")

newItem AS NEW IdValueClassItem

newItem.Id = taxCode

newItem.Value = taxName

newItem.Classe = taxClassesString

filteredTaxesList.ADD(newItem)

END WHILE

DB_CLOSE()

RETURN filteredTaxesList

END FUNCTION
```

4.6. GET api/Tax/search/by-name

Descrição da Funcionalidade e Objetivo: Este endpoint permite buscar taxas cujo nome comece com uma string específica fornecida. O objetivo é facilitar a localização rápida de taxas por nome, especialmente útil em interfaces de usuário com funcionalidade de busca incremental ou autocompletar.

Método DataFlex Correspondente (Inferido): buscarTaxasPorNome

Fluxos e Regras de Negócio:

1. Receber userSession e initial como parâmetros.
2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.

3. Construir uma consulta à tabela HCGS3001 para buscar taxas cujo nome (NM_TAXA) comece com a string initial fornecida.
4. A busca deve ser case-insensitive (não diferenciar maiúsculas de minúsculas).
5. Apenas taxas ativas (onde o campo ATIVO é 'S', por exemplo) devem ser consideradas.
6. Para cada taxa correspondente, extrair ID_TAXA (código) e NM_TAXA (nome).
7. Os resultados devem ser ordenados alfabeticamente pelo NM_TAXA.
8. Retornar uma lista de objetos TaxItem.

4.6.1. Modelos de Dados Envolvidos

- **Requisição:**
  - userSession (string, query param, obrigatório): Identificador da sessão do usuário
  - initial (string, query param, obrigatório): Prefixo do nome da taxa para busca
- **Resposta:** IEnumerable<TaxItem>

```
public class TaxItem
{
  public string? Id { get; set; }          // Código da taxa
  public string? Value { get; set; }       // Nome da taxa
  public string? Value2 { get; set; } // Descrição adicional (não
```

utilizado neste endpoint)

}

- **Tabelas do Banco de Dados:**
  - HCGS3001 (Taxas) - *Estrutura detalhada anteriormente (Seção 4.1). Campos relevantes: ID_TAXA, NM_TAXA, ATIVO.*
  - CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

Campo	**Tipo**	**Descrição**	**Chave**	**Relacionamentos**
ID_QUERY	VARCHAR(50)	Identificador único da query	PK	-
DESCRICAO	VARCHAR(200)	Descrição da finalidade da query	-	-
SQL_TEXT	TEXT	Texto completo da query SQL	-	-
... (demais campos)	...	...	...	...

4.6.2. Pseudocódigo para Reconstrução

```
// Função auxiliar para buscar query armazenada (se implementado assim)
FUNCTION GetStoredQuery(queryId AS STRING, parameters AS DICTIONARY)
RETURNS                                                          STRING
  //   Busca   a   query   armazenada   e   substitui   os   parâmetros
  //      Implementação      depende      do      sistema      específico
END FUNCTION
```

FUNCTION SearchTaxesByName(userSessionToken AS STRING, initialString AS STRING) RETURNS LIST OF TaxItem

DECLARE taxesByNameList AS NEW LIST OF TaxItem

IF NOT ValidarSessao(userSessionToken) THEN

RETURN taxesByNameList // Retorna lista vazia se sessão inválida

END IF

DB_CONNECT()

// Opção 1: Consulta direta

sqlQuery AS STRING = "SELECT ID_TAXA, NM_TAXA FROM HCGS3001 WHERE ATIVO = 'S' AND UPPER(NM_TAXA) LIKE UPPER('" + initialString + "%') ORDER BY NM_TAXA"

// Opção 2: Usar query armazenada (se implementado assim)

// sqlQuery = GetStoredQuery("WS_HCGS3001_BUSCAR_TAXAS", { "initial": initialString })

QUERY_TAXAS = DB_EXECUTE_SQL(sqlQuery)

WHILE DB_HAS_NEXT_ROW(QUERY_TAXAS)

```
taxCode AS STRING = DB_GET_FIELD(QUERY_TAXAS, "ID_TAXA")

taxName AS STRING = DB_GET_FIELD(QUERY_TAXAS, "NM_TAXA")

newItem AS NEW TaxItem

newItem.Id = taxCode

newItem.Value = taxName

newItem.Value2 = NULL // Não utilizado neste endpoint

taxesByNameList.ADD(newItem)

END WHILE

DB_CLOSE()

RETURN taxesByNameList

END FUNCTION

// Função auxiliar para buscar query armazenada (se implementado assim)

FUNCTION GetStoredQuery(queryId AS STRING, parameters AS DICTIONARY) RETURNS STRING

// Busca a query armazenada e substitui os parâmetros

// Implementação depende do sistema específico

END FUNCTION
```

4.7. GET api/Tax/search/by-class-filter

Descrição da Funcionalidade e Objetivo: Este endpoint é similar ao endpoint search/by-class, mas com uma diferença importante: ele não requer validação de sessão do usuário. Seu objetivo é permitir que componentes do sistema que não têm contexto de sessão (como processos em background ou jobs agendados) possam filtrar taxas por classe.

Método DataFlex Correspondente (Inferido): buscarTaxasPorClasse

Fluxos e Regras de Negócio:

1. Receber classFilter como parâmetro opcional.
2. **Não requer validação de sessão do usuário**.
3. Construir uma consulta à tabela HCGS3001.
4. Apenas taxas ativas (onde o campo ATIVO é 'S', por exemplo) devem ser consideradas.
5. Se classFilter for fornecido e não for vazio:
   - Filtrar os resultados para incluir apenas taxas onde o campo CLASSE_TAXA contenha o classFilter fornecido.
   - Pode utilizar uma query armazenada na tabela CSAG367_FVT com ID 'WS_HCGS3001_BUSCAR_TAXAS_CLASSE'.
6. Se classFilter não for fornecido ou for vazio:
   - Retornar todas as taxas ativas.
7. Para cada taxa correspondente, extrair ID_TAXA (código) e NM_TAXA (nome).

8. Os resultados devem ser ordenados alfabeticamente pelo NM_TAXA.
9. Retornar uma lista de objetos TaxItem.

4.7.1. Modelos de Dados Envolvidos

- **Requisição:**
  - classFilter (string, query param, opcional): Filtro de classe ('O', 'F', 'D')
- **Resposta:** IEnumerable<TaxItem>

```
public class TaxItem
{
  public string? Id { get; set; }          // Código da taxa
  public string? Value { get; set; }       // Nome da taxa
  public string? Value2 { get; set; } // Descrição adicional (não utilizado neste endpoint)
}
```

- **Tabelas do Banco de Dados:**
  - HCGS3001 (Taxas) - *Estrutura detalhada anteriormente (Seção 4.1). Campos relevantes: ID_TAXA, NM_TAXA, CLASSE_TAXA, ATIVO.*
  - CSAG367_FVT (Queries SQL Armazenadas) - *Estrutura detalhada na Seção 4.6.*

4.7.2. Pseudocódigo para Reconstrução

```
FUNCTION SearchTaxesByClassFilter(classFilter AS STRING) RETURNS LIST OF TaxItem

DECLARE taxesByClassList AS NEW LIST OF TaxItem

DB_CONNECT()

sqlQuery AS STRING

IF classFilter IS NULL OR classFilter IS EMPTY THEN

// Se não houver filtro de classe, retorna todas as taxas ativas

sqlQuery = "SELECT ID_TAXA, NM_TAXA FROM HCGS3001 WHERE ATIVO = 'S' ORDER BY NM_TAXA"

ELSE

// Se houver filtro de classe, filtra por classe

// Opção 1: Consulta direta

sqlQuery = "SELECT ID_TAXA, NM_TAXA FROM HCGS3001 WHERE ATIVO = 'S' AND CLASSE_TAXA LIKE '%" + classFilter + "%' ORDER BY NM_TAXA"

// Opção 2: Usar query armazenada (se implementado assim)

//  sqlQuery  =  GetStoredQuery("WS_HCGS3001_BUSCAR_TAXAS_CLASSE",  { "classFilter": classFilter })

END IF

QUERY_TAXAS = DB_EXECUTE_SQL(sqlQuery)

WHILE DB_HAS_NEXT_ROW(QUERY_TAXAS)

taxCode AS STRING = DB_GET_FIELD(QUERY_TAXAS, "ID_TAXA")

taxName AS STRING = DB_GET_FIELD(QUERY_TAXAS, "NM_TAXA")

newItem AS NEW TaxItem

newItem.Id = taxCode

newItem.Value = taxName

newItem.Value2 = NULL // Não utilizado neste endpoint

taxesByClassList.ADD(newItem)
```

```
END WHILE

DB_CLOSE()

RETURN taxesByClassList

END                                                    FUNCTION
```

4.8. GET api/Tax/for-proposal

Descrição da Funcionalidade e Objetivo: Este endpoint recupera as taxas associadas a uma proposta específica, opcionalmente filtradas por classe. O objetivo é permitir a visualização e edição das taxas incluídas em uma proposta comercial, facilitando a gestão de propostas.

Método DataFlex Correspondente (Inferido): taxas_tarifarioxproposta

Fluxos e Regras de Negócio:

1. Receber proposalId (obrigatório) e classFilter (opcional) como parâmetros.
2. Construir uma consulta que faça um JOIN entre as tabelas HCGS3001 (taxas) e HCGS3006 (taxas da proposta).
3. Filtrar os resultados para incluir apenas registros onde HCGS3006.ID_PROPOSTA seja igual ao proposalId fornecido.
4. Se classFilter for fornecido e não for vazio:
   - Adicionar um filtro adicional para incluir apenas taxas onde o campo CLASSE_TAXA contenha o classFilter fornecido.
5. Para cada registro correspondente, extrair:

- HCGS3001.ID_TAXA (código da taxa)
- HCGS3001.NM_TAXA (nome da taxa)
- HCGS3006.DESCRICAO (descrição específica da taxa na proposta)

6. Os resultados devem ser ordenados alfabeticamente pelo NM_TAXA.
7. Retornar uma lista de objetos TaxItem, onde Value2 contém a descrição específica da taxa na proposta.

4.8.1. Modelos de Dados Envolvidos

- **Requisição:**
  - proposalId (string, query param, obrigatório): ID da proposta
  - classFilter (string, query param, opcional): Filtro de classe ('O', 'F', 'D')
- **Resposta:** IEnumerable<TaxItem>

```
public class TaxItem
{
  public string? Id { get; set; }          // Código da taxa
  public string? Value { get; set; }       // Nome da taxa
  public string? Value2 { get; set; } // Descrição específica da taxa na proposta
}
```

- **Tabelas do Banco de Dados:**

- HCGS3001 (Taxas) - *Estrutura detalhada anteriormente (Seção 4.1). Campos relevantes: ID_TAXA, NM_TAXA, CLASSE_TAXA.*
- HCGS3006 (Taxas da Proposta) - Tabela que relaciona taxas a propostas.

Campo	**Tipo**	**Descrição**	**Chave**	**Relacionamentos**
ID_PROPOSTA	VARCHAR(10)	Código da proposta	PK, FK	HCGS3005.ID_PROPOSTA
ID_TAXA	VARCHAR(10)	Código da taxa	PK, FK	HCGS3001.ID_TAXA
DESCRICAO	VARCHAR(200)	Descrição específica da taxa na proposta	-	-
VALOR	DECIMAL(15,2)	Valor da taxa na proposta	-	-
MOEDA	CHAR(3)	Código da moeda do valor	FK	CSAG331.SIGLA
... (outros campos)	...	...	...	...

4.8.2. Pseudocódigo para Reconstrução

```
FUNCTION GetTaxesForProposal(proposalId AS STRING, classFilter AS STRING) RETURNS LIST OF TaxItem

DECLARE taxesForProposalList AS NEW LIST OF TaxItem

IF proposalId IS NULL OR proposalId IS EMPTY THEN

RETURN taxesForProposalList // Retorna lista vazia se proposalId não for fornecido

END IF

DB_CONNECT()

sqlQuery AS STRING = "SELECT t.ID_TAXA, t.NM_TAXA, p.DESCRICAO " +

"FROM HCGS3001 t " +

"JOIN HCGS3006 p ON t.ID_TAXA = p.ID_TAXA " +

"WHERE p.ID_PROPOSTA = '" + proposalId + "'"

IF classFilter IS NOT NULL AND classFilter IS NOT EMPTY THEN

// Adiciona filtro por classe se classFilter for fornecido

sqlQuery = sqlQuery + " AND t.CLASSE_TAXA LIKE '%" + classFilter + "%'"

END IF

sqlQuery = sqlQuery + " ORDER BY t.NM_TAXA"

QUERY_TAXAS = DB_EXECUTE_SQL(sqlQuery)

WHILE DB_HAS_NEXT_ROW(QUERY_TAXAS)

taxCode AS STRING = DB_GET_FIELD(QUERY_TAXAS, "ID_TAXA")

taxName AS STRING = DB_GET_FIELD(QUERY_TAXAS, "NM_TAXA")

taxDescription AS STRING = DB_GET_FIELD(QUERY_TAXAS, "DESCRICAO")
```

```
newItem AS NEW TaxItem

newItem.Id = taxCode

newItem.Value = taxName

newItem.Value2 = taxDescription

taxesForProposalList.ADD(newItem)

END WHILE

DB_CLOSE()

RETURN taxesForProposalList

END                                                    FUNCTION
```

4.9. GET api/Tax/search/by-class-and-name

Descrição da Funcionalidade e Objetivo: Este endpoint combina as funcionalidades dos endpoints search/by-class e search/by-name, permitindo buscar taxas filtrando simultaneamente por classe e por prefixo de nome. O objetivo é oferecer uma busca mais refinada e flexível, útil para interfaces de usuário com filtros combinados.

Método DataFlex Correspondente (Inferido): fResearchTaxsbyClass

Fluxos e Regras de Negócio:

1. Receber userSession, initial (opcional) e classId (opcional) como parâmetros.

2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.
3. Construir uma consulta à tabela HCGS3001.
4. Apenas taxas ativas (onde o campo ATIVO é 'S', por exemplo) devem ser consideradas.
5. Aplicar filtros condicionais:
    - Se initial for fornecido e não for vazio, filtrar para incluir apenas taxas onde NM_TAXA comece com initial.
    - Se classId for fornecido e não for vazio, filtrar para incluir apenas taxas onde CLASSE_TAXA contenha classId.
    - Se ambos forem fornecidos, aplicar os dois filtros (AND lógico).
    - Se nenhum for fornecido, retornar todas as taxas ativas.
6. Para cada taxa correspondente, extrair ID_TAXA (código), NM_TAXA (nome) e CLASSE_TAXA (string concatenada das classes).
7. Os resultados devem ser ordenados alfabeticamente pelo NM_TAXA.
8. Retornar uma lista de objetos IdValueClassItem.

4.9.1. Modelos de Dados Envolvidos

- **Requisição:**
    - userSession (string, query param, obrigatório): Identificador da sessão do usuário

- initial (string, query param, opcional): Prefixo do nome da taxa para busca
- classId (string, query param, opcional): ID da classe para filtrar ('O', 'F', 'D')

- **Resposta:** IEnumerable<IdValueClassItem>

```
public class IdValueClassItem
{
  public string? Id { get; set; }          // Código da taxa
  public string? Value { get; set; }       // Nome da taxa
  public string? Classe { get; set; } // Classes associadas (concatenadas)
}
```

- **Tabelas do Banco de Dados:**
  - HCGS3001 (Taxas) - *Estrutura detalhada anteriormente (Seção 4.1). Campos relevantes: ID_TAXA, NM_TAXA, CLASSE_TAXA, ATIVO.*
  - CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

4.9.2. Pseudocódigo para Reconstrução

```
FUNCTION SearchTaxesByClassAndName(userSessionToken AS STRING, initialString AS STRING, classId AS STRING) RETURNS LIST OF IdValueClassItem

DECLARE taxesList AS NEW LIST OF IdValueClassItem

IF NOT ValidarSessao(userSessionToken) THEN

RETURN taxesList // Retorna lista vazia se sessão inválida

END IF

DB_CONNECT()

sqlQuery AS STRING = "SELECT ID_TAXA, NM_TAXA, CLASSE_TAXA FROM HCGS3001 WHERE ATIVO = 'S'"

// Adiciona filtro por nome se initial for fornecido

IF initialString IS NOT NULL AND initialString IS NOT EMPTY THEN

sqlQuery = sqlQuery + " AND UPPER(NM_TAXA) LIKE UPPER('" + initialString + "%')"

END IF

// Adiciona filtro por classe se classId for fornecido

IF classId IS NOT NULL AND classId IS NOT EMPTY THEN

sqlQuery = sqlQuery + " AND CLASSE_TAXA LIKE '%" + classId + "%'"

END IF

sqlQuery = sqlQuery + " ORDER BY NM_TAXA"

QUERY_TAXAS = DB_EXECUTE_SQL(sqlQuery)

WHILE DB_HAS_NEXT_ROW(QUERY_TAXAS)

taxCode AS STRING = DB_GET_FIELD(QUERY_TAXAS, "ID_TAXA")

taxName AS STRING = DB_GET_FIELD(QUERY_TAXAS, "NM_TAXA")

taxClassesString AS STRING = DB_GET_FIELD(QUERY_TAXAS, "CLASSE_TAXA")
```

```
newItem AS NEW IdValueClassItem

newItem.Id = taxCode

newItem.Value = taxName

newItem.Classe = taxClassesString

taxesList.ADD(newItem)

END WHILE

DB_CLOSE()

RETURN taxesList

END                                                    FUNCTION
```

5. Endpoints de Cidades (CityController)

5.1. GET api/City/country

Descrição da Funcionalidade e Objetivo: Este endpoint tem como objetivo identificar e retornar o código do país ao qual uma cidade específica pertence. É útil para validações geográficas e para determinar regras específicas de país em operações logísticas ou comerciais.

Método DataFlex Correspondente (Inferido): f_bPaisCidade

Fluxos e Regras de Negócio:

1. Receber userSession e cityId como parâmetros.
2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar um erro apropriado.
3. Consultar a tabela CSAG325 para encontrar o registro correspondente ao cityId fornecido.
4. Se o cityId não for encontrado em CSAG325, retornar um erro 404 (Not Found).
5. Extrair o valor do campo PAIS da tabela CSAG325, que contém o código do país (ex: 'BR', 'US').
6. Retornar o código do país como uma string.

5.1.1. Modelos de Dados Envolvidos

- **Requisição:**
  - userSession (string, query param, obrigatório): Identificador da sessão do usuário
  - cityId (string, query param, obrigatório): ID da cidade
- **Resposta:** string (código do país, ex: 'BR')

- **Tabelas do Banco de Dados:**

Nome da Tabela	Campo	Tipo	Descrição	Chave	Relacionamentos
CSAG325 (Cidades)	ID_CIDADE	VARCHAR (10)	Código único da cidade	PK	HCGS3000.ORIGEM, HCGS3000.DESTINO, HCGS3000.VIA
CSAG325 (Cidades)	DESCRICAO	VARCHAR (100)	Nome da cidade	-	-
CSAG325 (Cidades)	UF	VARCHAR (5)	Código do estado/província	-	TsubdivisionCode.SUBDIVISION_CODE
CSAG325 (Cidades)	PAIS	CHAR(2)	Código do país	FK	CSAG329.SIGLA
CSAG325 (Cidades)	... (outros campos)	...	...	...	...

- CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

5.1.2. Pseudocódigo para Reconstrução

```
FUNCTION GetCountryForCity(userSessionToken AS STRING, cityIdentifier AS STRING) RETURNS STRING

IF NOT ValidarSessao(userSessionToken) THEN

THROW ERROR("Sessão inválida")

END IF

IF cityIdentifier IS NULL OR cityIdentifier IS EMPTY THEN

THROW ERROR("ID da cidade não fornecido")

END IF

DB_CONNECT()

QUERY_CIDADE = DB_EXECUTE_SQL("SELECT PAIS FROM CSAG325 WHERE ID_CIDADE = '" + cityIdentifier + "'")

IF NOT DB_HAS_ROWS(QUERY_CIDADE) THEN

DB_CLOSE()

THROW ERROR("Cidade não encontrada", 404) // Not Found

END IF

countryCode AS STRING = DB_GET_FIELD(QUERY_CIDADE, "PAIS")

DB_CLOSE()

RETURN countryCode

END FUNCTION
```

5.2. GET api/City/search/by-trade

Descrição da Funcionalidade e Objetivo: Este endpoint busca cidades com base em um prefixo de nome e, opcionalmente, por uma lista de países. A resposta é formatada para ser facilmente utilizável em contextos de operações de comércio (trade), como a seleção de cidades de origem ou destino em cotações.

Método DataFlex Correspondente (Inferido): buscarCidadesPorPaisesPorTrade

Fluxos e Regras de Negócio:

1. Receber userSession, cityNamePrefix (obrigatório) e countryIds (opcional, lista de IDs de países separados por vírgula) como parâmetros.
2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.
3. Construir uma consulta que faça um JOIN entre as tabelas CSAG325 (cidades) e CSAG329 (países).

4. Filtrar os resultados para incluir apenas cidades onde CSAG325.DESCRICAO comece com o cityNamePrefix fornecido (case-insensitive).
5. Se countryIds for fornecido e não for vazio, adicionar um filtro para incluir apenas cidades cujo CSAG325.PAIS esteja na lista de countryIds.
6. Apenas cidades ativas (onde CSAG325.ATIVO é 'S', por exemplo) devem ser consideradas.
7. Para cada cidade correspondente, formatar uma descrição no padrão "Cidade(UF) - País" (ex: "Sao Paulo(SP) - Brasil").
8. Extrair CSAG325.ID_CIDADE (código da cidade), a descrição formatada, CSAG325.PAIS (código do país) e CSAG325.UF (código do estado/província).
9. Os resultados devem ser ordenados alfabeticamente pela descrição da cidade (CSAG325.DESCRICAO).
10. Retornar uma lista de objetos CityCountryTradeItem.

5.2.1. Modelos de Dados Envolvidos

- **Requisição:**
  - userSession (string, query param, obrigatório)
  - cityNamePrefix (string, query param, obrigatório)
  - countryIds (string, query param, opcional): Lista de IDs de países separados por vírgula (ex: "BR,US,AR")
- **Resposta:** IEnumerable<CityCountryTradeItem>

```
public class CityCountryTradeItem
{
    public string? Id { get; set; }        // Código da cidade
    public string? Value { get; set; }     // Descrição formatada (Cidade(UF) - País)
    public string? Pais { get; set; }      // Código do país
    public string? Uf { get; set; }        // Código do estado/província
    public string? Iata { get; set; }      // Código IATA (pode não ser preenchido neste endpoint)
    public string? Port { get; set; }      // Informação de porto (pode não ser preenchido)
}
```

- **Tabelas do Banco de Dados:**
  - CSAG325 (Cidades) - *Estrutura detalhada anteriormente (Seção 5.1). Campos relevantes: ID_CIDADE, DESCRICAO, UF, PAIS, ATIVO.*
  - CSAG329 (Países)

Campo	Tipo	Descrição	Chave	Relacionamentos

SIGLA	CHAR(2)	Código do país (ex: 'BR', 'US')	PK	CSAG325.PAIS
DESCRICAO	VARCHAR(100)	Nome do país	-	-
... (outros campos)	...	...	...	...

- CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

5.2.2. Pseudocódigo para Reconstrução

```
FUNCTION SearchCitiesByTrade(userSessionToken AS STRING, cityNamePrefix AS STRING, countryIdsString AS STRING) RETURNS LIST OF CityCountryTradeItem

DECLARE citiesList AS NEW LIST OF CityCountryTradeItem

IF NOT ValidarSessao(userSessionToken) THEN

RETURN citiesList // Retorna lista vazia se sessão inválida

END IF

IF cityNamePrefix IS NULL OR cityNamePrefix IS EMPTY THEN

RETURN citiesList // Prefixo do nome da cidade é obrigatório

END IF

DB_CONNECT()

sqlQuery AS STRING = "SELECT C.ID_CIDADE, C.DESCRICAO AS NOME_CIDADE, C.UF, C.PAIS AS CODIGO_PAIS, P.DESCRICAO AS NOME_PAIS " +
```

```
"FROM CSAG325 C " +

"JOIN CSAG329 P ON C.PAIS = P.SIGLA " +

"WHERE C.ATIVO = 'S' AND UPPER(C.DESCRICAO) LIKE UPPER('" +
cityNamePrefix + "%')"

IF countryIdsString IS NOT NULL AND countryIdsString IS NOT EMPTY THEN

// Converte a string de IDs de país em uma lista para a cláusula IN

// Ex: "BR,US" se torna "'BR','US'"

formattedCountryIds AS STRING =
FormatCountryIdsForSQL(countryIdsString)

sqlQuery = sqlQuery + " AND C.PAIS IN (" + formattedCountryIds + ")"

END IF

sqlQuery = sqlQuery + " ORDER BY C.DESCRICAO"

QUERY_CIDADES = DB_EXECUTE_SQL(sqlQuery)

WHILE DB_HAS_NEXT_ROW(QUERY_CIDADES)

cityId AS STRING = DB_GET_FIELD(QUERY_CIDADES, "ID_CIDADE")

cityName AS STRING = DB_GET_FIELD(QUERY_CIDADES, "NOME_CIDADE")

cityUf AS STRING = DB_GET_FIELD(QUERY_CIDADES, "UF")

countryCode AS STRING = DB_GET_FIELD(QUERY_CIDADES, "CODIGO_PAIS")

countryName AS STRING = DB_GET_FIELD(QUERY_CIDADES, "NOME_PAIS")

formattedDescription AS STRING = cityName

IF cityUf IS NOT NULL AND cityUf IS NOT EMPTY THEN

formattedDescription = formattedDescription + "(" + cityUf + ")"

END IF

formattedDescription = formattedDescription + " - " + countryName

newItem AS NEW CityCountryTradeItem

newItem.Id = cityId
```

```
newItem.Value = formattedDescription

newItem.Pais = countryCode // Ou countryName, dependendo do requisito

newItem.Uf = cityUf

// newItem.Iata e newItem.Port podem ser nulos ou preenchidos se houver joins adicionais

citiesList.ADD(newItem)

END WHILE

DB_CLOSE()

RETURN citiesList

END FUNCTION

// Função auxiliar para formatar IDs de país para SQL IN clause
FUNCTION FormatCountryIdsForSQL(countryIds AS STRING) RETURNS STRING
  //         Ex:         "BR,US,AR"         ->         "'BR','US','AR'"
  //     Implementação     depende     da     linguagem     e     SGBD
END                                                              FUNCTION
```

5.3. GET api/City/search/via

Descrição da Funcionalidade e Objetivo: Este endpoint busca cidades que podem ser utilizadas como pontos de via ou rota em operações de

transporte. Uma característica distintiva é a inclusão de um item especial “DIRECT SERVICE” se o prefixo de busca sugerir isso, indicando uma rota direta sem paradas intermediárias.

Método DataFlex Correspondente (Inferido): buscarCidadesVia

Fluxos e Regras de Negócio:

1. Receber userSession, cityNamePrefix (obrigatório) e countryIds (opcional) como parâmetros.
2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.
3. Inicializar a lista de resultados.
4. **Regra Especial “DIRECT SERVICE”**: Se cityNamePrefix contiver “DIRE” (case-insensitive), adicionar um item especial à lista de resultados:
   - Id: “0”
   - Value: “DIRECT SERVICE”
   - Outros campos (Pais, Uf, Iata, Port) podem ser nulos ou vazios.
5. Construir uma consulta que faça um JOIN entre as tabelas CSAG325 (cidades) e CSAG329 (países).
6. Filtrar os resultados para incluir apenas cidades onde CSAG325.DESCRICAO comece com o cityNamePrefix fornecido (case-insensitive).

7. Se countryIds for fornecido e não for vazio, adicionar um filtro para incluir apenas cidades cujo CSAG325.PAIS esteja na lista de countryIds.
8. Apenas cidades ativas (onde CSAG325.ATIVO é 'S', por exemplo) devem ser consideradas.
9. Para cada cidade correspondente, formatar uma descrição no padrão "Cidade(UF) - País".
10. Extrair CSAG325.ID_CIDADE, a descrição formatada, CSAG325.PAIS e CSAG325.UF.
11. Adicionar os itens à lista de resultados.
12. Os resultados (após o item "DIRECT SERVICE", se presente) devem ser ordenados alfabeticamente pela descrição da cidade (CSAG325.DESCRICAO).
13. Retornar a lista de objetos CityCountryTradeItem.

5.3.1. Modelos de Dados Envolvidos

- **Requisição:**
  - userSession (string, query param, obrigatório)
  - cityNamePrefix (string, query param, obrigatório)
  - countryIds (string, query param, opcional): Lista de IDs de países separados por vírgula
- **Resposta:** IEnumerable<CityCountryTradeItem>

```
public class CityCountryTradeItem
{
    public string? Id { get; set; }    // Código da cidade ou "0" para
```

```
    DIRECT                                                    SERVICE
    public string? Value { get; set; } // Descrição formatada ou
"DIRECT                                                    SERVICE"
    public string? Pais { get; set; } // Código do país
    public string? Uf { get; set; } // Código do estado/província
    public string? Iata { get; set; } // Código IATA
    public string? Port { get; set; } // Informação de porto
}
```

- **Tabelas do Banco de Dados:**
  - CSAG325 (Cidades) - *Estrutura detalhada anteriormente (Seção 5.1).*
  - CSAG329 (Países) - *Estrutura detalhada anteriormente (Seção 5.2).*
  - CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

5.3.2. Pseudocódigo para Reconstrução

```
FUNCTION SearchCitiesVia(userSessionToken AS STRING, cityNamePrefix AS
STRING, countryIdsString AS STRING) RETURNS LIST OF CityCountryTradeItem

DECLARE citiesList AS NEW LIST OF CityCountryTradeItem

IF NOT ValidarSessao(userSessionToken) THEN

RETURN citiesList // Retorna lista vazia se sessão inválida

END IF
```

```
IF cityNamePrefix IS NULL OR cityNamePrefix IS EMPTY THEN

RETURN citiesList // Prefixo do nome da cidade é obrigatório

END IF

// Adicionar "DIRECT SERVICE" caso relevante

IF UPPER(cityNamePrefix).CONTAINS("DIRE") THEN

citiesList.ADD(New CityCountryTradeItem(ID = "0", Value = "DIRECT SERVICE"))

END IF

DB_CONNECT()

sqlQuery = "SELECT C.ID_CIDADE, C.DESCRICAO, C.UF, C.PAIS, P.DESCRICAO AS NOME_PAIS " +

"FROM CSAG325 C " +

"JOIN CSAG329 P ON C.PAIS = P.SIGLA " +

"WHERE C.ATIVO = 'S' AND UPPER(C.DESCRICAO) LIKE UPPER('" + cityNamePrefix + "%')"

// Adicionar filtro de países, se aplicável

IF countryIdsString IS NOT NULL AND countryIdsString IS NOT EMPTY THEN

sqlQuery += " AND C.PAIS IN (" + FormatCountryIdsForSQL(countryIdsString) + ")"

END IF

sqlQuery += " ORDER BY C.DESCRICAO"

QUERY_CIDADES = DB_EXECUTE_SQL(sqlQuery)

WHILE DB_HAS_NEXT_ROW(QUERY_CIDADES)

cityId = DB_GET_FIELD(QUERY_CIDADES, "ID_CIDADE")

cityName = DB_GET_FIELD(QUERY_CIDADES, "DESCRICAO")

cityUf = DB_GET_FIELD(QUERY_CIDADES, "UF")

countryCode = DB_GET_FIELD(QUERY_CIDADES, "PAIS")
```

```
countryName = DB_GET_FIELD(QUERY_CIDADES, "NOME_PAIS")

formattedDescription = cityName + IF cityUf IS NOT NULL AND cityUf IS
NOT EMPTY THEN " (" + cityUf + ")" ELSE "" END + " - " + countryName

citiesList.ADD(New   CityCountryTradeItem(ID   =   cityId,   Value   =
formattedDescription, Pais = countryCode, Uf = cityUf))

END WHILE

DB_CLOSE()

RETURN citiesList

END                                                              FUNCTION
```

5.4. GET api/City/description/{cityCode}

Descrição da Funcionalidade e Objetivo: Este endpoint retorna a descrição formatada de uma cidade específica no padrão "Cidade(UF) - País". O objetivo é padronizar a exibição de informações de cidades em diferentes partes do sistema, garantindo consistência na apresentação.

Método DataFlex Correspondente (Inferido): cidadeDescricaoPais

Fluxos e Regras de Negócio:

1. Receber cityCode como parâmetro de rota.
2. **Não requer validação de sessão do usuário**, pois é uma informação de referência não sensível.
3. Consultar a tabela CSAG325 para encontrar o registro correspondente ao cityCode fornecido.
4. Fazer um JOIN com a tabela CSAG329 para obter o nome do país.

5. Se o cityCode não for encontrado em CSAG325, retornar um erro 404 (Not Found).
6. Formatar a descrição no padrão “Cidade(UF) - País” (ex: “Sao Paulo(SP) - Brasil”).
7. Retornar a descrição formatada como uma string.

5.4.1. Modelos de Dados Envolvidos

- **Requisição:**
  - cityCode (string, path param, obrigatório): Código da cidade
- **Resposta:** string (descrição formatada)
- **Tabelas do Banco de Dados:**
  - CSAG325 (Cidades) - *Estrutura detalhada anteriormente (Seção 5.1). Campos relevantes: ID_CIDADE, DESCRICAO, UF, PAIS.*
  - CSAG329 (Países) - *Estrutura detalhada anteriormente (Seção 5.2). Campos relevantes: SIGLA, DESCRICAO.*

5.4.2. Pseudocódigo para Reconstrução

```
FUNCTION GetCityDescription(cityCode AS STRING) RETURNS STRING

IF IS_NULL_OR_EMPTY(cityCode) THEN

THROW ERROR("Código da cidade não fornecido")

END IF

DB_CONNECT()

sqlQuery AS STRING = "

SELECT

C.DESCRICAO AS NOME_CIDADE,

C.UF,

P.DESCRICAO AS NOME_PAIS

FROM CSAG325 C

JOIN CSAG329 P

ON C.PAIS = P.SIGLA

WHERE C.ID_CIDADE = '" + cityCode + "'"

QUERY_CIDADE = DB_EXECUTE_SQL(sqlQuery)

IF NOT DB_HAS_ROWS(QUERY_CIDADE) THEN

DB_CLOSE()

THROW ERROR("Cidade não encontrada", 404) // Not Found

END IF

cityName AS STRING = DB_GET_FIELD(QUERY_CIDADE, "NOME_CIDADE")

cityUf AS STRING = DB_GET_FIELD(QUERY_CIDADE, "UF")

countryName AS STRING = DB_GET_FIELD(QUERY_CIDADE, "NOME_PAIS")

formattedDescription AS STRING = cityName

IF NOT IS_NULL_OR_EMPTY(cityUf) THEN
```

```
formattedDescription = formattedDescription + "(" + cityUf + ")"
END IF
formattedDescription = formattedDescription + " - " + countryName
DB_CLOSE()
RETURN formattedDescription
END                                                    FUNCTION
```

5.5. GET api/City/proposal/via-points

Descrição da Funcionalidade e Objetivo: Este endpoint identifica cidades que podem servir como pontos intermediários (via) em um tarifário entre origem e destino específicos. O objetivo é permitir a seleção de rotas existentes ao criar propostas comerciais, considerando o modo de transporte e o tipo de cliente.

Método DataFlex Correspondente (Inferido): f_viaPoints

Fluxos e Regras de Negócio:

1. Receber pol (origem), pod (destino), mode (modal) e clientId (cliente) como parâmetros obrigatórios.
2. Determinar o tipo de cliente ('CL'/'CD') com base no clientId fornecido, consultando a tabela CSAG340.
3. Selecionar a tabela de tarifários apropriada com base no modal:
    - Se mode for 'AIR' (aéreo), usar a tabela HCGS3000_AIRFR8.
    - Para outros modais, usar a tabela HCGS3000.
4. Buscar tarifários que correspondam à origem (pol), destino (pod) e tipo de cliente determinado.
5. Apenas tarifários válidos na data atual devem ser considerados (dentro do período de validade).
6. Para cada tarifário válido, extrair o ponto de via:
    - Para o modal aéreo, o campo relevante é ID_CIDADE.
    - Para outros modais, o campo relevante é VIA.
7. Para cada ponto de via encontrado, buscar informações detalhadas na tabela CSAG325 e formatar a descrição no padrão "Cidade(UF) - País".
8. Os resultados devem ser ordenados alfabeticamente pelo nome da cidade.
9. Retornar uma lista de objetos IdValueItem.

5.5.1. Modelos de Dados Envolvidos

- **Requisição:**
    - pol (string, query param, obrigatório): Código da cidade de origem (Port of Loading)

- pod (string, query param, obrigatório): Código da cidade de destino (Port of Discharge)
- mode (string, query param, obrigatório): Código do modal de transporte
- clientId (string, query param, obrigatório): Código do cliente

- **Resposta:** IEnumerable<IdValueItem>

```
public class IdValueItem
{
  public string? Id { get; set; }      // Código da cidade
  public string? Value { get; set; }   // Descrição formatada (Cidade(UF) - País)
}
```

- **Tabelas do Banco de Dados:**
  - CSAG325 (Cidades) - *Estrutura detalhada anteriormente (Seção 5.1).*
  - CSAG329 (Países) - *Estrutura detalhada anteriormente (Seção 5.2).*

- CSAG340 (Clientes)

Campo	Tipo	Descrição	Chave	Relacionamentos
ID_CLIENTE	VARCHAR(10)	Código único do cliente	PK	-
NOME	VARCHAR(200)	Nome/razão social do cliente	-	-
TIPO	CHAR(2)	Tipo de cliente ('CL'/'CD')	-	-
... (outros campos)	...	...	...	...

- HCGS3000 (Tarifários)

Campo	Tipo	Descrição	Chave	Relacionamentos
ID_TARIFA	VARCHAR(10)	Código único do tarifário	PK	-
ORIGEM	VARCHAR(10)	Cidade de origem	FK	CSAG325.ID_CIDADE

DESTINO	VARCHAR(10)	Cidade de destino	FK	CSAG325.ID_CIDADE
VIA	VARCHAR(10)	Cidade de via/rota	FK	CSAG325.ID_CIDADE
MODAL	VARCHAR(5)	Modo de transporte	FK	CSAG337.SIGLA
TIPO_CLIENTE	CHAR(2)	Tipo de cliente (‘CL’/‘CD’)	-	-
DT_VALIDADE_INICIO	DATE	Data de início da validade	-	-
DT_VALIDADE_FIM	DATE	Data de fim da validade	-	-
… (outros campos)	…	…	…	…

- HCGS3000_AIRFR8 (Tarifários Aéreos)

Campo	**Tipo**	**Descrição**	**Chave**	**Relacionamentos**

ID_TARIFA	VARCHAR(10)	Código único do tarifário	PK	-
ORIGEM	VARCHAR(10)	Cidade de origem	FK	CSAG325.ID_CIDADE
DESTINO	VARCHAR(10)	Cidade de destino	FK	CSAG325.ID_CIDADE
ID_CIDADE	VARCHAR(10)	Cidade de via/escala	FK	CSAG325.ID_CIDADE
TIPO_CLIENTE	CHAR(2)	Tipo de cliente ('CL'/'CD')	-	-
DT_VALIDADE_INICIO	DATE	Data de início da validade	-	-
DT_VALIDADE_FIM	DATE	Data de fim da validade	-	-
... (outros campos)	...	...	...	...

5.5.2. Pseudocódigo para Reconstrução

```
FUNCTION GetProposalViaPoints(pol AS STRING, pod AS STRING, mode AS
STRING, clientId AS STRING) RETURNS LIST OF IdValueItem
  DECLARE viaPointsList AS NEW LIST OF IdValueItem

  IF pol IS NULL OR pol IS EMPTY OR pod IS NULL OR pod IS EMPTY OR
mode IS NULL OR mode IS EMPTY OR clientId IS NULL OR clientId IS EMPTY
```

```
THEN
    RETURN viaPointsList // Retorna lista vazia se algum parâmetro
obrigatório não for fornecido
  END IF

  DB_CONNECT()

  // Determinar o tipo de cliente
  QUERY_CLIENTE = DB_EXECUTE_SQL("SELECT TIPO FROM CSAG340 WHERE
ID_CLIENTE = '" + clientId + "'")

  IF NOT DB_HAS_ROWS(QUERY_CLIENTE) THEN
    DB_CLOSE()
    RETURN viaPointsList // Cliente não encontrado
  END IF

  clientType AS STRING = DB_GET_FIELD(QUERY_CLIENTE, "TIPO")
  currentDate AS DATE = GET_CURRENT_DATE()

  // Buscar pontos de via com base no modal
  sqlQuery AS STRING

  IF UPPER(mode) = "AIR" THEN
    // Modal aéreo - usar tabela HCGS3000_AIRFR8
    sqlQuery = "SELECT DISTINCT t.ID_CIDADE AS VIA_CIDADE " +
               "FROM HCGS3000_AIRFR8 t " +
               "WHERE t.ORIGEM = '" + pol + "' " +
               "AND t.DESTINO = '" + pod + "' " +
               "AND t.TIPO_CLIENTE = '" + clientType + "' " +
               "AND t.DT_VALIDADE_INICIO <= '" +
FORMAT_DATE(currentDate) + "' " +
               "AND t.DT_VALIDADE_FIM >= '" + FORMAT_DATE(currentDate)
+ "' " +
               "AND t.ATIVO = 'S'"
```

```
  ELSE
    // Outros modais - usar tabela HCGS3000
    sqlQuery = "SELECT DISTINCT t.VIA AS VIA_CIDADE " +
               "FROM HCGS3000 t " +
               "WHERE t.ORIGEM = '" + pol + "' " +
               "AND t.DESTINO = '" + pod + "' " +
               "AND t.MODAL = '" + mode + "' " +
               "AND t.TIPO_CLIENTE = '" + clientType + "' " +
               "AND t.DT_VALIDADE_INICIO <= '" +
FORMAT_DATE(currentDate) + "' " +
               "AND t.DT_VALIDADE_FIM >= '" + FORMAT_DATE(currentDate)
+ "' " +
               "AND t.ATIVO = 'S'"
  END IF

  QUERY_VIA_POINTS = DB_EXECUTE_SQL(sqlQuery)

  // Processar cada ponto de via encontrado
  WHILE DB_HAS_NEXT_ROW(QUERY_VIA_POINTS)
    viaCityId AS STRING = DB_GET_FIELD(QUERY_VIA_POINTS, "VIA_CIDADE")

    // Buscar detalhes da cidade
    QUERY_CIDADE = DB_EXECUTE_SQL("SELECT C.DESCRICAO AS NOME_CIDADE,
C.UF, P.DESCRICAO AS NOME_PAIS " +
                                  "FROM CSAG325 C " +
                                  "JOIN CSAG329 P ON C.PAIS = P.SIGLA "
+
                                  "WHERE C.ID_CIDADE = '" + viaCityId +
"'")

    IF DB_HAS_ROWS(QUERY_CIDADE) THEN
      cityName AS STRING = DB_GET_FIELD(QUERY_CIDADE, "NOME_CIDADE")
      cityUf AS STRING = DB_GET_FIELD(QUERY_CIDADE, "UF")
      countryName AS STRING = DB_GET_FIELD(QUERY_CIDADE, "NOME_PAIS")
```

```
      formattedDescription AS STRING = cityName
      IF cityUf IS NOT NULL AND cityUf IS NOT EMPTY THEN
        formattedDescription = formattedDescription + "(" + cityUf +
")"
      END IF
      formattedDescription = formattedDescription + " - " +
countryName

      newItem AS NEW IdValueItem
      newItem.Id = viaCityId
      newItem.Value = formattedDescription

      viaPointsList.ADD(newItem)
    END IF
  END WHILE

  // Ordenar por nome da cidade (já formatado em Value)
  SORT(viaPointsList, BY "Value")

  DB_CLOSE()
  RETURN viaPointsList
END FUNCTION

// Função auxiliar para formatar data para SQL
FUNCTION FORMAT_DATE(date AS DATE) RETURNS STRING
  // Implementação depende do formato esperado pelo SGBD
END FUNCTION
```

5.6. GET api/City/proposal/origin-cities

Descrição da Funcionalidade e Objetivo: Este endpoint localiza cidades que podem servir como origem em propostas, com base em

diversos critérios como prefixo de nome, país e modo de transporte. O objetivo é facilitar a seleção de cidades de origem ao criar propostas comerciais, considerando o contexto específico do cliente e do modo de transporte.

Método DataFlex Correspondente (Inferido): buscarCidadesOrigemProposta

Fluxos e Regras de Negócio:

1. Receber userSession, cityNamePrefix (obrigatório), countryId (opcional), mode (obrigatório) e clientId (obrigatório) como parâmetros.
2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.
3. Determinar o tipo de cliente ('CL'/'CD') com base no clientId fornecido, consultando a tabela CSAG340.
4. Construir uma consulta à tabela CSAG325 para buscar cidades que correspondam a um dos seguintes critérios:
   - Nome da cidade (CSAG325.DESCRICAO) começa com cityNamePrefix.
   - Para o modo aéreo, código IATA (HSAG325_AIR.IATA) começa com cityNamePrefix.
   - Para o modo aéreo, nome do aeroporto (HSAG325_AIR.NOME) começa com cityNamePrefix.
5. Se countryId for fornecido e não for vazio, adicionar um filtro para incluir apenas cidades cujo CSAG325.PAIS seja igual ao countryId.

6. Apenas cidades ativas devem ser consideradas.
7. Para cada cidade correspondente:
    - Formatar uma descrição no padrão “Cidade(UF) - País”.
    - Para o modo aéreo, incluir informações de aeroporto da tabela HSAG325_AIR.
    - Verificar se a cidade possui tarifário associado para o modo e cliente especificados.
8. Os resultados devem ser ordenados alfabeticamente pelo nome da cidade.
9. Retornar uma lista de objetos CityCountryTradeItem.

5.6.1. Modelos de Dados Envolvidos

- **Requisição:**
    - userSession (string, query param, obrigatório)
    - cityNamePrefix (string, query param, obrigatório)
    - countryId (string, query param, opcional): Código do país para filtrar
    - mode (string, query param, obrigatório): Código do modal de transporte
    - clientId (string, query param, obrigatório): Código do cliente
- **Resposta:** IEnumerable<CityCountryTradeItem>

```
public class CityCountryTradeItem
{
    public string? Id { get; set; }    // Código da cidade
```

```
    public string? Value { get; set; } // Descrição formatada (Cidade(UF) - País)
    public string? Pais { get; set; } // Código do país
    public string? Uf { get; set; } // Código do estado/província
    public string? Iata { get; set; } // Código IATA (para modo aéreo)
    public string? Port { get; set; } // Informação de porto (pode não ser preenchido)
}
```

- **Tabelas do Banco de Dados:**
  - CSAG325 (Cidades) - *Estrutura detalhada anteriormente (Seção 5.1).*
  - CSAG329 (Países) - *Estrutura detalhada anteriormente (Seção 5.2).*
  - CSAG340 (Clientes) - *Estrutura detalhada anteriormente (Seção 5.5).*
  - HSAG325_AIR (Aeroportos)

Campo	Tipo	Descrição	Chave	Relacionamentos
ID_CIDADE	VARCHAR(10)	Código da cidade	PK, FK	CSAG325.ID_CIDADE
IATA	CHAR(3)	Código IATA do aeroporto	-	-

NOME	VARCHAR(100)	Nome do aeroporto	-	-
... (outros campos)	...	...	...	...

- HCGS3000 e HCGS3000_AIRFR8 - *Estruturas detalhadas anteriormente (Seção 5.5).*
- CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

5.6.2. Pseudocódigo para Reconstrução

```
FUNCTION GetProposalOriginCities(userSessionToken AS STRING,
cityNamePrefix AS STRING, countryId AS STRING, mode AS STRING,
clientId AS STRING) RETURNS LIST OF CityCountryTradeItem
  DECLARE originCitiesList AS NEW LIST OF CityCountryTradeItem

  IF NOT ValidarSessao(userSessionToken) THEN
    RETURN originCitiesList // Retorna lista vazia se sessão inválida
  END IF

  IF cityNamePrefix IS NULL OR cityNamePrefix IS EMPTY OR mode IS NULL
OR mode IS EMPTY OR clientId IS NULL OR clientId IS EMPTY THEN
    RETURN originCitiesList // Retorna lista vazia se parâmetros
obrigatórios não forem fornecidos
  END IF

  DB_CONNECT()

  // Determinar o tipo de cliente
```

```
  QUERY_CLIENTE = DB_EXECUTE_SQL("SELECT TIPO FROM CSAG340 WHERE
ID_CLIENTE = '" + clientId + "'")

  IF NOT DB_HAS_ROWS(QUERY_CLIENTE) THEN
    DB_CLOSE()
    RETURN originCitiesList // Cliente não encontrado
  END IF

  clientType AS STRING = DB_GET_FIELD(QUERY_CLIENTE, "TIPO")

  // Construir a consulta base
  sqlQuery AS STRING = "SELECT C.ID_CIDADE, C.DESCRICAO AS
NOME_CIDADE, C.UF, C.PAIS AS CODIGO_PAIS, P.DESCRICAO AS NOME_PAIS "

  // Para modo aéreo, incluir informações de aeroporto
  IF UPPER(mode) = "AIR" THEN
    sqlQuery = sqlQuery + ", A.IATA, A.NOME AS NOME_AEROPORTO "
  END IF

  sqlQuery = sqlQuery + "FROM CSAG325 C " +
                        "JOIN CSAG329 P ON C.PAIS = P.SIGLA "

  // Para modo aéreo, fazer join com tabela de aeroportos
  IF UPPER(mode) = "AIR" THEN
    sqlQuery = sqlQuery + "LEFT JOIN HSAG325_AIR A ON C.ID_CIDADE =
A.ID_CIDADE "
  END IF

  sqlQuery = sqlQuery + "WHERE C.ATIVO = 'S' "

  // Filtro por prefixo de nome
  IF UPPER(mode) = "AIR" THEN
    // Para modo aéreo, buscar por nome da cidade, código IATA ou nome
```

```
do aeroporto
    sqlQuery = sqlQuery + "AND (UPPER(C.DESCRICAO) LIKE UPPER('" +
cityNamePrefix + "%') " +
                          "OR UPPER(A.IATA) LIKE UPPER('" +
cityNamePrefix + "%') " +
                          "OR UPPER(A.NOME) LIKE UPPER('" +
cityNamePrefix + "%')) "
  ELSE
    // Para outros modos, buscar apenas por nome da cidade
    sqlQuery = sqlQuery + "AND UPPER(C.DESCRICAO) LIKE UPPER('" +
cityNamePrefix + "%') "
  END IF

  // Filtro por país
  IF countryId IS NOT NULL AND countryId IS NOT EMPTY THEN
    sqlQuery = sqlQuery + "AND C.PAIS = '" + countryId + "' "
  END IF

  sqlQuery = sqlQuery + "ORDER BY C.DESCRICAO"

  QUERY_CIDADES = DB_EXECUTE_SQL(sqlQuery)

  WHILE DB_HAS_NEXT_ROW(QUERY_CIDADES)
    cityId AS STRING = DB_GET_FIELD(QUERY_CIDADES, "ID_CIDADE")
    cityName AS STRING = DB_GET_FIELD(QUERY_CIDADES, "NOME_CIDADE")
    cityUf AS STRING = DB_GET_FIELD(QUERY_CIDADES, "UF")
    countryCode AS STRING = DB_GET_FIELD(QUERY_CIDADES, "CODIGO_PAIS")
    countryName AS STRING = DB_GET_FIELD(QUERY_CIDADES, "NOME_PAIS")

    formattedDescription AS STRING = cityName
    IF cityUf IS NOT NULL AND cityUf IS NOT EMPTY THEN
      formattedDescription = formattedDescription + "(" + cityUf + ")"
    END IF
    formattedDescription = formattedDescription + " - " + countryName
```

```
    newItem AS NEW CityCountryTradeItem
    newItem.Id = cityId
    newItem.Value = formattedDescription
    newItem.Pais = countryCode
    newItem.Uf = cityUf

    // Para modo aéreo, incluir código IATA
    IF UPPER(mode) = "AIR" THEN
      newItem.Iata = DB_GET_FIELD(QUERY_CIDADES, "IATA")
    END IF

    // Verificar se a cidade possui tarifário associado
    hasTariff AS BOOLEAN = CheckCityHasTariff(cityId, mode,
clientType)

    // Adicionar à lista apenas se tiver tarifário ou se estiver
buscando todas as cidades
    originCitiesList.ADD(newItem)
  END WHILE

  DB_CLOSE()
  RETURN originCitiesList
END FUNCTION

// Função auxiliar para verificar se uma cidade possui tarifário como
origem
FUNCTION CheckCityHasTariff(cityId AS STRING, mode AS STRING,
clientType AS STRING) RETURNS BOOLEAN
  currentDate AS DATE = GET_CURRENT_DATE()

  IF UPPER(mode) = "AIR" THEN
    QUERY_TARIFF = DB_EXECUTE_SQL("SELECT 1 FROM HCGS3000_AIRFR8 " +
```

```
                                "WHERE ORIGEM = '" + cityId + "' " +
                                "AND TIPO_CLIENTE = '" + clientType +
"' " +
                                "AND DT_VALIDADE_INICIO <= '" +
FORMAT_DATE(currentDate) + "' " +
                                "AND DT_VALIDADE_FIM >= '" +
FORMAT_DATE(currentDate) + "' " +
                                "AND ATIVO = 'S' " +
                                "LIMIT 1")
  ELSE
    QUERY_TARIFF = DB_EXECUTE_SQL("SELECT 1 FROM HCGS3000 " +
                                "WHERE ORIGEM = '" + cityId + "' " +
                                "AND MODAL = '" + mode + "' " +
                                "AND TIPO_CLIENTE = '" + clientType +
"' " +
                                "AND DT_VALIDADE_INICIO <= '" +
FORMAT_DATE(currentDate) + "' " +
                                "AND DT_VALIDADE_FIM >= '" +
FORMAT_DATE(currentDate) + "' " +
                                "AND ATIVO = 'S' " +
                                "LIMIT 1")
  END IF

  RETURN DB_HAS_ROWS(QUERY_TARIFF)
END FUNCTION
```

5.7. GET api/City/proposal/destination-cities

Descrição da Funcionalidade e Objetivo: Este endpoint localiza cidades que podem servir como destino em propostas, com base em diversos critérios, incluindo a origem já selecionada. O objetivo é facilitar a seleção de cidades de destino ao criar propostas comerciais,

considerando o contexto específico da origem, cliente e modo de transporte.

Método DataFlex Correspondente (Inferido): buscarCidadesDestinoProposta

Fluxos e Regras de Negócio:

1. Receber userSession, cityNamePrefix (obrigatório), countryId (opcional), mode (obrigatório), clientId (obrigatório) e pol (origem, obrigatório) como parâmetros.
2. **Validar Sessão do Usuário**: Executar a lógica de validação de sessão. Se inválida, retornar uma lista vazia.
3. Determinar o tipo de cliente ('CL'/'CD') com base no clientId fornecido, consultando a tabela CSAG340.
4. Construir uma consulta à tabela CSAG325 para buscar cidades que correspondam a um dos seguintes critérios:
   - Nome da cidade (CSAG325.DESCRICAO) começa com cityNamePrefix.
   - Para o modo aéreo, código IATA (HSAG325_AIR.IATA) começa com cityNamePrefix.
   - Para o modo aéreo, nome do aeroporto (HSAG325_AIR.NOME) começa com cityNamePrefix.
5. Se countryId for fornecido e não for vazio, adicionar um filtro para incluir apenas cidades cujo CSAG325.PAIS seja igual ao countryId.
6. Apenas cidades ativas devem ser consideradas.
7. Para cada cidade correspondente:

- Formatar uma descrição no padrão “Cidade(UF) - País”.
- Para o modo aéreo, incluir informações de aeroporto da tabela HSAG325_AIR.
- Verificar se a cidade é um destino válido para a origem (pol) especificada, considerando o modo e tipo de cliente.

8. Os resultados devem ser ordenados alfabeticamente pelo nome da cidade.
9. Retornar uma lista de objetos CityCountryTradeItem.

5.7.1. Modelos de Dados Envolvidos

- **Requisição:**
  - userSession (string, query param, obrigatório)
  - cityNamePrefix (string, query param, obrigatório)
  - countryId (string, query param, opcional): Código do país para filtrar
  - mode (string, query param, obrigatório): Código do modal de transporte
  - clientId (string, query param, obrigatório): Código do cliente
  - pol (string, query param, obrigatório): Código da cidade de origem (Port of Loading)
- **Resposta:** IEnumerable<CityCountryTradeItem>

```
public class CityCountryTradeItem
{
    public string? Id { get; set; }    // Código da cidade
```

```
    public string? Value { get; set; } // Descrição formatada (Cidade(UF) - País)
    public string? Pais { get; set; } // Código do país
    public string? Uf { get; set; } // Código do estado/província
    public string? Iata { get; set; } // Código IATA (para modo aéreo)
    public string? Port { get; set; } // Informação de porto (pode não ser preenchido)
}
```

- **Tabelas do Banco de Dados:**
  - CSAG325 (Cidades) - *Estrutura detalhada anteriormente (Seção 5.1).*
  - CSAG329 (Países) - *Estrutura detalhada anteriormente (Seção 5.2).*
  - CSAG340 (Clientes) - *Estrutura detalhada anteriormente (Seção 5.5).*
  - HSAG325_AIR (Aeroportos) - *Estrutura detalhada anteriormente (Seção 5.6).*
  - HCGS3000 e HCGS3000_AIRFR8 - *Estruturas detalhadas anteriormente (Seção 5.5).*
  - CSAG311 (Sessões de Usuário) - *Estrutura detalhada na Seção 3.*

5.7.2 Pseudocódigo para Reconstrução

```
FUNCTION GetProposalDestinationCities(userSessionToken AS STRING,
cityNamePrefix AS STRING, countryId AS STRING, mode AS STRING,
clientId AS STRING, pol AS STRING) RETURNS LIST OF
CityCountryTradeItem
  DECLARE destinationCitiesList AS NEW LIST OF CityCountryTradeItem


  IF NOT ValidarSessao(userSessionToken) THEN
    RETURN destinationCitiesList // Retorna lista vazia se sessão
inválida
  END IF


  IF cityNamePrefix IS NULL OR cityNamePrefix IS EMPTY OR mode IS NULL
OR mode IS EMPTY OR clientId IS NULL OR clientId IS EMPTY OR pol IS
NULL OR pol IS EMPTY THEN
    RETURN destinationCitiesList // Retorna lista vazia se parâmetros
obrigatórios não forem fornecidos
  END IF


  DB_CONNECT()


  // Determinar o tipo de cliente
  QUERY_CLIENTE = DB_EXECUTE_SQL("SELECT TIPO FROM CSAG340 WHERE
ID_CLIENTE = '" + clientId + "'")


  IF NOT DB_HAS_ROWS(QUERY_CLIENTE) THEN
    DB_CLOSE()
    RETURN destinationCitiesList // Cliente não encontrado
  END IF


  clientType AS STRING = DB_GET_FIELD(QUERY_CLIENTE, "TIPO")


  // Construir a consulta base
  sqlQuery AS STRING = "SELECT C.ID_CIDADE, C.DESCRICAO AS
```

```
NOME_CIDADE, C.UF, C.PAIS AS CODIGO_PAIS, P.DESCRICAO AS NOME_PAIS "

  // Para modo aéreo, incluir informações de aeroporto
  IF UPPER(mode) = "AIR" THEN
    sqlQuery = sqlQuery + ", A.IATA, A.NOME AS NOME_AEROPORTO "
  END IF

  sqlQuery = sqlQuery + "FROM CSAG325 C " +
                        "JOIN CSAG329 P ON C.PAIS = P.SIGLA "

  // Para modo aéreo, fazer join com tabela de aeroportos
  IF UPPER(mode) = "AIR" THEN
    sqlQuery = sqlQuery + "LEFT JOIN HSAG325_AIR A ON C.ID_CIDADE =
A.ID_CIDADE "
  END IF

  sqlQuery = sqlQuery + "WHERE C.ATIVO = 'S' "

  // Filtro por prefixo de nome
  IF UPPER(mode) = "AIR" THEN
    // Para modo aéreo, buscar por nome da cidade, código IATA ou nome
do aeroporto
    sqlQuery = sqlQuery + "AND (UPPER(C.DESCRICAO) LIKE UPPER('" +
cityNamePrefix + "%') " +
                          "OR UPPER(A.IATA) LIKE UPPER('" +
cityNamePrefix + "%') " +
                          "OR UPPER(A.NOME) LIKE UPPER('" +
cityNamePrefix + "%')) "
  ELSE
    // Para outros modos, buscar apenas por nome da cidade
    sqlQuery = sqlQuery + "AND UPPER(C.DESCRICAO) LIKE UPPER('" +
cityNamePrefix + "%') "
  END IF
```

```
// Filtro por país
IF countryId IS NOT NULL AND countryId IS NOT EMPTY THEN
  sqlQuery = sqlQuery + "AND C.PAIS = '" + countryId + "' "
END IF

sqlQuery = sqlQuery + "ORDER BY C.DESCRICAO"

QUERY_CIDADES = DB_EXECUTE_SQL(sqlQuery)

WHILE DB_HAS_NEXT_ROW(QUERY_CIDADES)
  cityId AS STRING = DB_GET_FIELD(QUERY_CIDADES, "ID_CIDADE")
  cityName AS STRING = DB_GET_FIELD(QUERY_CIDADES, "NOME_CIDADE")
  cityUf AS STRING = DB_GET_FIELD(QUERY_CIDADES, "UF")
  countryCode AS STRING = DB_GET_FIELD(QUERY_CIDADES, "CODIGO_PAIS")
  countryName AS STRING = DB_GET_FIELD(QUERY_CIDADES, "NOME_PAIS")

  formattedDescription AS STRING = cityName
  IF cityUf IS NOT NULL AND cityUf IS NOT EMPTY THEN
    formattedDescription = formattedDescription + "(" + cityUf + ")"
  END IF
  formattedDescription = formattedDescription + " - " + countryName

  newItem AS NEW CityCountryTradeItem
  newItem.Id = cityId
  newItem.Value = formattedDescription
  newItem.Pais = countryCode
  newItem.Uf = cityUf

  // Para modo aéreo, incluir código IATA
  IF UPPER(mode) = "AIR" THEN
    newItem.Iata = DB_GET_FIELD(QUERY_CIDADES, "IATA")
  END IF
```

```
    // Verificar se a cidade é um destino válido para a origem
especificada
    isValidDestination AS BOOLEAN = CheckValidDestination(pol, cityId,
mode, clientType)

    // Adicionar à lista apenas se for um destino válido ou se estiver
buscando todas as cidades
    IF isValidDestination THEN
      destinationCitiesList.ADD(newItem)
    END IF
  END WHILE

  DB_CLOSE()
  RETURN destinationCitiesList
END FUNCTION

// Função auxiliar para verificar se uma cidade é um destino válido
para a origem especificada
FUNCTION CheckValidDestination(originCityId AS STRING,
destinationCityId AS STRING, mode AS STRING, clientType AS STRING)
RETURNS BOOLEAN
  currentDate AS DATE = GET_CURRENT_DATE()

  IF UPPER(mode) = "AIR" THEN
    QUERY_TARIFF = DB_EXECUTE_SQL("SELECT 1 FROM HCGS3000_AIRFR8 " +
                                  "WHERE ORIGEM = '" + originCityId +
"' " +
                                  "AND DESTINO = '" + destinationCityId
+ "' " +
                                  "AND TIPO_CLIENTE = '" + clientType +
"' " +
                                  "AND DT_VALIDADE_INICIO <= '" +
FORMAT_DATE(currentDate) + "' " +
                                  "AND DT_VALIDADE_FIM >= '" +
```

```
FORMAT_DATE(currentDate) + "' " +
                                    "AND ATIVO = 'S' " +
                                    "LIMIT 1")
  ELSE
    QUERY_TARIFF = DB_EXECUTE_SQL("SELECT 1 FROM HCGS3000 " +
                                    "WHERE ORIGEM = '" + originCityId +
"' " +
                                    "AND DESTINO = '" + destinationCityId
+ "' " +
                                    "AND MODAL = '" + mode + "' " +
                                    "AND TIPO_CLIENTE = '" + clientType +
"' " +
                                    "AND DT_VALIDADE_INICIO <= '" +
FORMAT_DATE(currentDate) + "' " +
                                    "AND DT_VALIDADE_FIM >= '" +
FORMAT_DATE(currentDate) + "' " +
                                    "AND ATIVO = 'S' " +
                                    "LIMIT 1")
  END IF

  RETURN DB_HAS_ROWS(QUERY_TARIFF)
END FUNCTION
```

6. Fluxos de Integração Principais

6.1. Fluxo de Criação de Proposta

Este fluxo representa a sequência típica de chamadas à API para criar uma proposta comercial:

```
FUNCTION CreateProposalFlow()
 // 1. Autenticação (não detalhada nesta documentação)
 userSessionToken AS STRING = AuthenticateUser(username,
password)
```

```
// 2. Seleção de Cliente (UI)
selectedClientId AS STRING = UI_SelectClient()

// 3. Seleção de Modo de Transporte (UI)
selectedMode AS STRING = UI_SelectTransportMode()

// 4. Buscar e Selecionar Cidade de Origem
originCityNamePrefix AS STRING = UI_GetOriginCitySearchPrefix()
originCities AS LIST = GetProposalOriginCities(userSessionToken,
originCityNamePrefix, NULL, selectedMode, selectedClientId)
selectedOriginCity AS STRING = UI_SelectOriginCity(originCities)

// 5. Buscar e Selecionar Cidade de Destino
destCityNamePrefix AS STRING = UI_GetDestinationCitySearchPrefix()
destinationCities AS LIST =
GetProposalDestinationCities(userSessionToken, destCityNamePrefix,
NULL, selectedMode, selectedClientId, selectedOriginCity)
selectedDestinationCity AS STRING =
UI_SelectDestinationCity(destinationCities)

// 6. Buscar e Selecionar Ponto Via (opcional)
viaPoints AS LIST = GetProposalViaPoints(selectedOriginCity,
selectedDestinationCity, selectedMode, selectedClientId)
selectedViaPoint AS STRING = UI_SelectViaPoint(viaPoints) // Pode ser
"0" para "DIRECT SERVICE"
```

```
  // 7. Buscar e Selecionar Taxas para cada classe
  taxClasses AS LIST = GetTaxClasses() // Retorna 'O', 'F', 'D'
  selectedTaxes AS LIST = NEW LIST

  FOR EACH taxClass IN taxClasses
    taxesForClass AS LIST = SearchTaxesByClass(userSessionToken,
taxClass)
    selectedTaxesForClass AS LIST = UI_SelectTaxes(taxesForClass,
taxClass)
    selectedTaxes.ADD_ALL(selectedTaxesForClass)
  END FOR

  // 8. Criar Proposta (não detalhado nesta documentação)
  proposalId AS STRING = CreateProposal(userSessionToken,
selectedClientId, selectedMode, selectedOriginCity,
selectedDestinationCity, selectedViaPoint, selectedTaxes)

  RETURN proposalId
END FUNCTION
```

6.2. Fluxo de Consulta de Taxas

Este fluxo representa a sequência típica de chamadas à API para consultar taxas:

```
FUNCTION TaxQueryFlow()
  // 1. Autenticação (não detalhada nesta documentação)
```

```
  userSessionToken AS STRING = AuthenticateUser(username, password)

  // 2. Obter Classes de Taxa
  taxClasses AS LIST = GetTaxClasses()

  // 3. Selecionar Classe para Filtrar (UI)
  selectedClass AS STRING = UI_SelectTaxClass(taxClasses)

  // 4. Buscar Taxas por Classe
  taxesByClass AS LIST = SearchTaxesByClass(userSessionToken,
selectedClass)

  // 5. Selecionar Taxa Específica (UI)
  selectedTaxId AS STRING = UI_SelectTax(taxesByClass)

  // 6. Obter Classes Associadas à Taxa Selecionada
  selectedTaxClasses AS LIST = GetSelectedTaxClasses(userSessionToken,
selectedTaxId)

  // 7. Exibir Detalhes da Taxa (UI)
  UI_DisplayTaxDetails(selectedTaxId, selectedTaxClasses)

  // 8. Opcionalmente, buscar propostas que usam esta taxa
  // (não detalhado nesta documentação)
END FUNCTION
```

6.3. Fluxo de Validação Geográfica

Este fluxo representa a sequência típica de chamadas à API para validação geográfica:

```
FUNCTION GeographicValidationFlow()
  // 1. Autenticação (não detalhada nesta documentação)
  userSessionToken AS STRING = AuthenticateUser(username, password)
```

```
  // 2. Buscar Cidade por Nome
  cityNamePrefix AS STRING = UI_GetCitySearchPrefix()
  cities AS LIST = SearchCitiesByTrade(userSessionToken,
cityNamePrefix, NULL)

  // 3. Selecionar Cidade (UI)
  selectedCityId AS STRING = UI_SelectCity(cities)

  // 4. Validar País da Cidade
  countryCode AS STRING = GetCountryForCity(userSessionToken,
selectedCityId)

  // 5. Obter Descrição Formatada da Cidade
  formattedDescription AS STRING = GetCityDescription(selectedCityId)

  // 6. Aplicar Regras Específicas por País
  IF countryCode = "BR" THEN
    // Aplicar regras específicas para Brasil
  ELSE IF countryCode = "US" THEN
    // Aplicar regras específicas para Estados Unidos
  END IF

  // 7. Exibir Resultado da Validação (UI)
  UI_DisplayValidationResult(selectedCityId, formattedDescription,
countryCode)
END FUNCTION
```