

EXPLICAÇÃO LPROG

GRUPO 2DI2

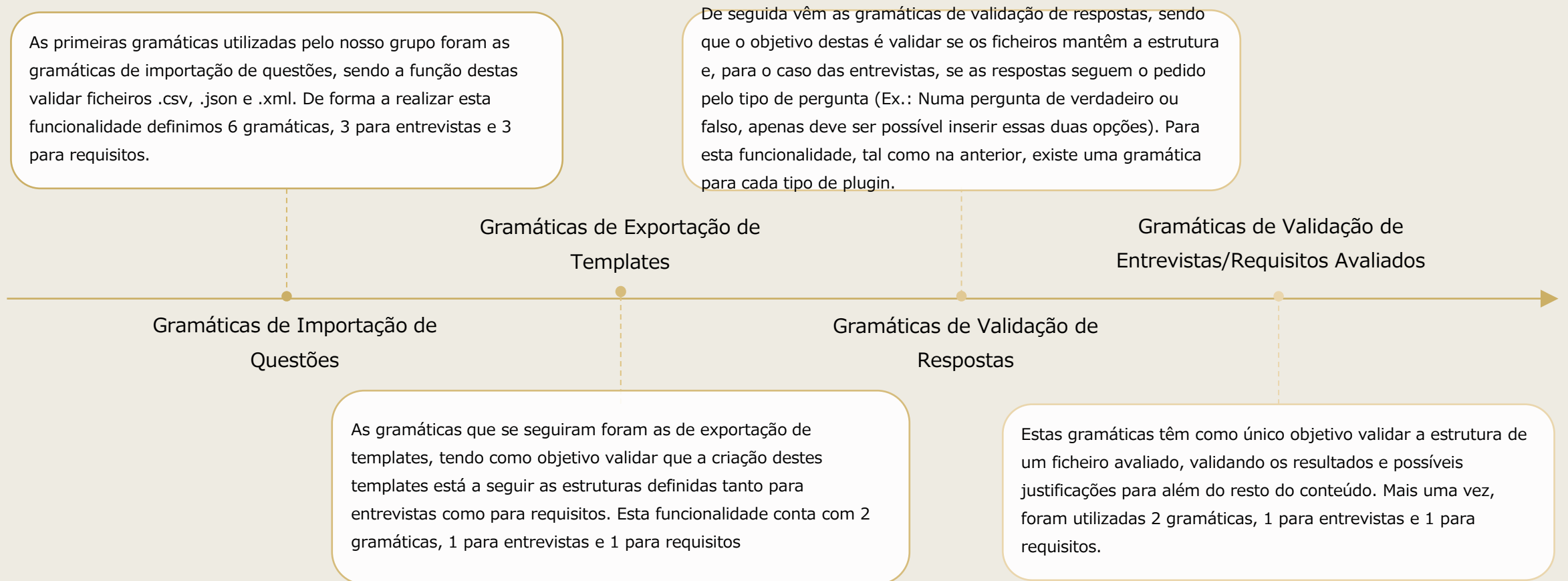
Temas Abordados:

- Uso das gramáticas no Projeto;
- Gramáticas Utilizadas;
- Uso de Listeners e de Visitors;
- Overview.

Introdução

Este documento tem como objetivo explicar sucintamente as gramáticas criadas e utilizadas por nós no projeto integrador de LAPR4. Também iremos apresentar as nossas razões e o nosso pensamento para explicar o uso de Listeners ou de Visitors nas diferentes funcionalidades.

Linha Cronológica das Gramáticas



Gramáticas de Importação de Questões

OBJETIVO E UTILIZAÇÃO DAS GRAMÁTICAS:

- Estas gramáticas foram criadas com o objetivo de através de ficheiros de extensão .csv, .json ou .xml verificar se estes seguem a estrutura correta definida entre a equipa e o cliente.
- No âmbito do Projeto Integrador estas gramáticas foram importantes para conseguirmos fazer a importação das questões para os plugins e podermos futuramente carregar os mesmos dinamicamente sempre que necessário.
- Para a parte da importação aproveitamos os Listeners de cada uma das gramáticas, pois, após alguma pesquisa, chegamos à conclusão de que estes são bastante úteis para importação e exportação de dados.
- Sendo assim, percorremos toda a Parse Tree e através dos Listeners obtivemos as informações necessárias para importar as questões para o sistema.
- Estas gramáticas foram concebidas para as User Stories: US-1008, US-1009 (Requisitos) e US-1011 (Entrevistas).



Gramática InterviewCSV.g4

ESTRUTURA DESEJADA:

"Tipo de Questão";"Cotação da Pergunta";"Tipo de Cotação";"O Corpo da Questão";"Uma Possível Resposta";"A cotação da resposta";"Outra Possível Resposta se necessário";"A cotação da resposta"

Requisitos:

- O tipo de questão deve ser um dos seguintes tipos:
 - Verdadeiro Ou Falso;
 - Resposta Curta;
 - Resposta com número inteiro;
 - Resposta com número decimal;
 - Resposta com Data;
 - Resposta com Tempo;
 - Resposta com Escala Numérica;
 - Escolha singular;
 - Escolha Múltipla;
- As cotações das perguntas devem ser entre 0 e 100 e o seu tipo deve ser percentagem, valores ou pontos;
- As cotações das possíveis respostas devem ser entre 0 e 100 e o seu tipo é predefinido como percentagem;



Gramática InterviewCSV.g4

PARSER RULES DA GRAMÁTICA:

```
questions: question+ EOF;
body:
    ''' (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ (
        ''' (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ ''' (
            (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+
        )?
    )? ''';
questionBody: body;
type:
    ''' (
        TRUE_FALSE_QUESTION
        | SHORT_TEXT_ANSWER_QUESTION
        | SINGLE_ANSWER_CHOICE_QUESTION
        | MULTIPLE_ANSWER_CHOICE_QUESTION
        | INTEGER_NUMBER_QUESTION
        | DECIMAL_NUMBER_QUESTION
        | DATE_QUESTION
        | TIME_QUESTION
        | NUMERIC_SCALE_QUESTION
    ) ''';
questionCotation: cotation;
answer: body;
answerCotation: cotation;
question:
    type ';' questionCotation ';' cotationType ';' questionBody (
        ';' answer ';' answerCotation
    )+ (NEWLINE)?;
cotation:
    ''' (TWO_DIGIT_NUMBER | FRACTIONAL_NUMBER | '100') ''';
cotationType: ''' ( '%' | 'POINTS' | 'VALUES') ''';
```



Gramática InterviewCSV.g4

LEXER RULES DA GRAMÁTICA:

```
TRUE_FALSE_QUESTION: 'True/False';
SHORT_TEXT_ANSWER_QUESTION: 'Short text answer';
SINGLE_ANSWER_CHOICE_QUESTION: 'Choice, with single answer';
MULTIPLE_ANSWER_CHOICE_QUESTION:
|   'Choice, with multiple answers';
INTEGER_NUMBER_QUESTION: 'Integer Number';
DECIMAL_NUMBER_QUESTION: 'Decimal Number';
DATE_QUESTION: 'Date';
TIME_QUESTION: 'Time';
NUMERIC_SCALE_QUESTION: 'Numeric Scale';
TWO_DIGIT_NUMBER: NUMBER NUMBER?;
NUMBER: [0-9];
LETTER: [a-zA-Z];
MEMBER: [.,;:/#+!/?@*)([\]] | '\]' | '|' | '-';
FRACTIONAL_NUMBER:
|   TWO_DIGIT_NUMBER ('.' | ',') TWO_DIGIT_NUMBER;
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;
NEWLINE: ('\r'? '\n')+;
WS: [ \t\n\r]+ -> skip;
```



Gramática InterviewJSON.g4

ESTRUTURA DESEJADA:

```
[
  {
    "type": "Tipo de Questão",
    "cotation": "Cotação da Questão",
    "cotationType": "Tipo de Cotação",
    "body": "Corpo da Questão",
    "possibleAnswers": [
      {
        "answer": "Resposta",
        "cotation": "Cotação da Resposta"
      }
    ]
  },
  {
    "type": "Tipo de Questão",
    "cotation": "Cotação da Questão",
    "cotationType": "Tipo de Cotação",
    "body": "Corpo da Questão",
    "possibleAnswers": [
      {
        "answer": "Resposta",
        "cotation": "Cotação da Resposta"
      },
      {
        "answer": "Outra Resposta se necessário",
        "cotation": "Cotação da Resposta"
      }
    ]
  }
]
```

Requisitos:

- O tipo de questão deve ser um dos seguintes tipos:
 - Verdadeiro Ou Falso;
 - Resposta Curta;
 - Resposta com número inteiro;
 - Resposta com número decimal;
 - Resposta com Data;
 - Resposta com Tempo;
 - Resposta com Escala Numérica;
 - Escolha singular;
 - Escolha Múltipla;
- As cotações das perguntas devem ser entre 0 e 100 e o seu tipo deve ser percentagem, valores ou pontos;
- As cotações das possíveis respostas devem ser entre 0 e 100 e o seu tipo é predefinido como percentagem;



Gramática InterviewJSON.g4

PARSER RULES DA GRAMÁTICA:

```
questions: '[' question (',' question)+ ']' EOF;
body:
    ''' (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ (
        ''' (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ ''' (
            (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+
        )?
    )? ''';
questionBody: body;
type:
    ''' (
        TRUE_FALSE_QUESTION
        | SHORT_TEXT_ANSWER_QUESTION
        | SINGLE_ANSWER_CHOICE_QUESTION
        | MULTIPLE_ANSWER_CHOICE_QUESTION
        | INTEGER_NUMBER_QUESTION
        | DECIMAL_NUMBER_QUESTION
        | DATE_QUESTION
        | TIME_QUESTION
        | NUMERIC_SCALE_QUESTION
    ) ''';
questionCotation: cotation;
answer: body;
answerCotation: cotation;
question:
    '{ "type" ':' type ',' "cotation" ':' questionCotation ',' "cotationType" ':'
    cotationType ',' "body" ':' questionBody ',' "possibleAnswers" ':' '[' '{ "answer"
    ':' answer ',' "cotation" ':' answerCotation '}' (
    ',' '{ "answer" ':' answer ',' "cotation" ':' answerCotation '}'
    )* ']' '}'
cotation:
    ''' (TWO_DIGIT_NUMBER | FRACTIONAL_NUMBER | '100') ''';
cotationType: ''' ( '%' | 'POINTS' | 'VALUES' ) ''';
```



Gramática InterviewJSON.g4

LEXER RULES DA GRAMÁTICA:

```
TRUE_FALSE_QUESTION: 'True/False';
SHORT_TEXT_ANSWER_QUESTION: 'Short text answer';
SINGLE_ANSWER_CHOICE_QUESTION: 'Choice, with single answer';
MULTIPLE_ANSWER_CHOICE_QUESTION:
    'Choice, with multiple answers';
INTEGER_NUMBER_QUESTION: 'Integer Number';
DECIMAL_NUMBER_QUESTION: 'Decimal Number';
DATE_QUESTION: 'Date';
TIME_QUESTION: 'Time';
NUMERIC_SCALE_QUESTION: 'Numeric Scale';
TWO_DIGIT_NUMBER: NUMBER NUMBER?;
NUMBER: [0-9];
LETTER: [a-zA-Z];
MEMBER: [.,;:/#+!?*)([\]] | '\\' | '|' | '-';
FRACTIONAL_NUMBER:
    TWO_DIGIT_NUMBER ('.' | ',') TWO_DIGIT_NUMBER;
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;
WS: [ \t\n\r]+ -> skip;
```



Gramática InterviewXML.g4

ESTRUTURA DESEJADA:

```
<Questions>
  <Question>
    <Cotation>Cotação da Pergunta</Cotation>
    <CotationType>Tipo de Cotação</CotationType>
    <Type>Tipo de Questão</Type>
    <Body>Corpo da Questão</Body>
    <PossibleAnswersList>
      <PossibleAnswers>
        <Answer>Uma Possível Resposta</Answer>
        <Cotation>Cotação da Resposta</Cotation>
      </PossibleAnswers>
    </PossibleAnswersList>
  </Question>
  <Question>
    <Cotation> Cotação da Pergunta </Cotation>
    <CotationType> Tipo de Cotação </CotationType>
    <Type> Tipo de Questão </Type>
    <Body>Corpo da Questão</Body>
    <PossibleAnswersList>
      <PossibleAnswers>
        <Answer>Uma Possível Resposta</Answer>
        <Cotation>Cotação da Resposta</Cotation>
      </PossibleAnswers>
      <PossibleAnswers>
        <Answer>Outra Possível Resposta</Answer>
        <Cotation>Cotação da Resposta</Cotation>
      </PossibleAnswers>
    </PossibleAnswersList>
  </Question>
</Questions>
```

Requisitos:

- O tipo de questão deve ser um dos seguintes tipos:
 - Verdadeiro Ou Falso;
 - Resposta Curta;
 - Resposta com número inteiro;
 - Resposta com número decimal;
 - Resposta com Data;
 - Resposta com Tempo;
 - Resposta com Escala Numérica;
 - Escolha singular;
 - Escolha Múltipla;
- As cotações das perguntas devem ser entre 0 e 100 e o seu tipo deve ser percentagem, valores ou pontos;
- As cotações das possíveis respostas devem ser entre 0 e 100 e o seu tipo é predefinido como percentagem;



Gramática InterviewXML.g4

PARSER RULES DA GRAMÁTICA:

```
questions: '<Questions>' question+ '</Questions>' EOF;
text: (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+;
question:
    '<Question>' questionCotation cotationType type body possibleAnswersList '</Question>';
cotation:
    '<Cotation>' (TWO_DIGIT_NUMBER | FRACTIONAL_NUMBER | '100') '</Cotation>';
cotationType:
    '<CotationType>' ('%' | 'POINTS' | 'VALUES') '</CotationType>';
type:
    '<Type>' (
        TRUE_FALSE_QUESTION
        | SHORT_TEXT_ANSWER_QUESTION
        | SINGLE_ANSWER_CHOICE_QUESTION
        | MULTIPLE_ANSWER_CHOICE_QUESTION
        | INTEGER_NUMBER_QUESTION
        | DECIMAL_NUMBER_QUESTION
        | DATE_QUESTION
        | TIME_QUESTION
        | NUMERIC_SCALE_QUESTION
    ) '</Type>';
body:
    '<Body>' text (
        ('<' | '/' | '>' | '</') text (
            ('<' | '/' | '>' | '</') text
        )?
    )? '</Body>';
questionCotation: cotation;
answerCotation: cotation;
possibleAnswersList:
    '<PossibleAnswersList>' possibleAnswers* '</PossibleAnswersList>';
possibleAnswers:
    '<PossibleAnswers>' answer answerCotation '</PossibleAnswers>';
answer: '<Answer>' text '</Answer>';
```



Gramática InterviewXML.g4

LEXER RULES DA GRAMÁTICA:

```
TRUE_FALSE_QUESTION: 'True/False';
SHORT_TEXT_ANSWER_QUESTION: 'Short text answer';
SINGLE_ANSWER_CHOICE_QUESTION: 'Choice, with single answer';
MULTIPLE_ANSWER_CHOICE_QUESTION:
    'Choice, with multiple answers';
INTEGER_NUMBER_QUESTION: 'Integer Number';
DECIMAL_NUMBER_QUESTION: 'Decimal Number';
DATE_QUESTION: 'Date';
TIME_QUESTION: 'Time';
NUMERIC_SCALE_QUESTION: 'Numeric Scale';
TWO_DIGIT_NUMBER: NUMBER NUMBER?;
NUMBER: [0-9];
LETTER: [a-zA-Z];
MEMBER: [.,;:/#+!?@*]([\]] | '\]' | '|' | '-');
FRACTIONAL_NUMBER:
    TWO_DIGIT_NUMBER ('.' | ',') TWO_DIGIT_NUMBER;
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;
WS: [ \t\n\r]+ -> skip;
```



Gramática RequirementsCSV.g4

ESTRUTURA DESEJADA:

“Corpo da Questão”; “Uma possível resposta/Outra possível resposta caso necessário”; “Justificação caso não sejam atingidos os requisitos mínimos nesta questão”

Requisitos:

- N/A



Gramática RequirementsCSV.g4

PARSER RULES DA GRAMÁTICA:

```
questions: question+ EOF;
body:
    "" (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ (
        "" (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ "" (
            (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+
        )?
    )? "";
questionBody: body;
answer: body;
minimumRequirement: body;
question:
    questionBody ';' answer ';' minimumRequirement (NEWLINE)?;
```



Gramática RequirementsCSV.g4

LEXER RULES DA GRAMÁTICA:

```
TWO_DIGIT_NUMBER: NUMBER NUMBER?;  
NUMBER: [0-9];  
LETTER: [a-zA-Z];  
MEMBER: [.,;:/#+!?@*)([\\] | '\\'' | '|' | '-';  
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;  
NEWLINE: ('\\r'? '\\n')+;  
WS: [ \\t\\n\\r]+ -> skip;
```



Gramática RequirementsJSON.g4

ESTRUTURA DESEJADA:

```
[
  {
    "body": "Corpo da Questão",
    "possibleAnswers": [
      "Uma possível resposta",
      "Outra Possível resposta caso necessário",
    ],
    "minimumRequirement": "Corpo da Justificação"
  },
  {
    "body": "Corpo da Questão",
    "possibleAnswers": [
      "Uma possível resposta",
      "Outra Possível resposta caso necessário",
    ],
    "minimumRequirement": "Corpo da Justificação"
  },
]
```

Requisitos:

- N/A



Gramática RequirementsJSON.g4

PARSER RULES DA GRAMÁTICA:

```
questions: '[' question (',' question)+ ']' EOF;
body:
    ''' (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ (
        ''' (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ ''' (
            (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+
        )?
    )? ''';
questionBody: body;
answer: body;
minimumRequirement: body;
question:
    '{' '"body"' ':' questionBody ',' '"possibleAnswers"' ':' '[' answer (
        ',' answer
    )* ']' ',' '"minimumRequirement"' ':' minimumRequirement '}';
```



Gramática RequirementsJSON.g4

LEXER RULES DA GRAMÁTICA:

```
TWO_DIGIT_NUMBER: NUMBER NUMBER?;  
NUMBER: [0-9];  
LETTER: [a-zA-Z];  
MEMBER: [.,;:/#+!~@*) ( ] | '\ ' | '| ' | '- ';  
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;  
WS: [ \t\n\r]+ -> skip;
```



Gramática RequirementsXML.g4

ESTRUTURA DESEJADA:

```
<Questions>
  <Question>
    <Body>Corpo da Questão</Body>
    <PossibleAnswersList>
      <PossibleAnswers>Uma possível resposta</PossibleAnswers>
      <PossibleAnswers>Outra possível resposta caso necessário</PossibleAnswers>
    </PossibleAnswersList>
    <MinimumRequirement>Corpo da Justificação</MinimumRequirement>
  </Question>
  <Question>
    <Body>Corpo da Questão</Body>
    <PossibleAnswersList>
      <PossibleAnswers>Uma possível resposta</PossibleAnswers>
      <PossibleAnswers>Outra possível resposta caso necessário</PossibleAnswers>
    </PossibleAnswersList>
    <MinimumRequirement>Corpo da Justificação</MinimumRequirement>
  </Question>
</Questions>
```

Requisitos:

- N/A



Gramática RequirementsXML.g4

PARSER RULES DA GRAMÁTICA:

```
questions: '<Questions>' question+ '</Questions>' EOF;
text: (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+;
question:
    '<Question>' body possibleAnswersList minimumRequirement '</Question>';
body:
    '<Body>' text (
        ('<' | '/' | '>' | '</') text (
            ('<' | '/' | '>' | '</') text
        )?
    )? '</Body>';
possibleAnswersList:
    '<PossibleAnswersList>' possibleAnswers* '</PossibleAnswersList>';
possibleAnswers:
    '<PossibleAnts>' text (
        ('<' | '/' | '>' | '</') text (
            ('<' | '/' | '>' | '</') text
        )?
    )? '</PossibleAnswers>';
minimumRequirement:
    '<MinimumRequirement>' text (
        ('<' | '/' | '>' | '</') text (
            ('<' | '/' | '>' | '</') text
        )?
    )? '</MinimumRequirement>';
```



Gramática RequirementsXML.g4

LEXER RULES DA GRAMÁTICA:

```
TWO_DIGIT_NUMBER: NUMBER NUMBER?;  
NUMBER: [0-9];  
LETTER: [a-zA-Z];  
MEMBER: [.,;:/#+!/?@*)([\\]] | '\\'' | '|' | '-';  
FRACTIONAL_NUMBER: TWO_DIGIT_NUMBER ('.' | ',') TWO_DIGIT_NUMBER;  
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;  
WS: [ \\t\\n\\r]+ -> skip;
```



Listeners Utilizados

Listener InterviewCSV

```
public class InterviewCsvListener extends InterviewCsvBaseListener {  
    private final List<InterviewQuestionDTO> questions = new ArrayList<>();  
    private InterviewQuestionDTO current;  
    private List<InterviewAnswer> answers;  
    private Answer answer;  
  
    @Override  
    public void enterQuestion(final InterviewCsvParser.QuestionContext ctx) {  
        answers = new ArrayList<>();  
        current = new InterviewQuestionDTO();  
    }  
  
    @Override  
    public void exitQuestion(final InterviewCsvParser.QuestionContext ctx) {  
        questions.add(current);  
    }  
  
    @Override  
    public void enterQuestionBody(final InterviewCsvParser.QuestionBodyContext ctx) {  
        current.setBody(extractValue(ctx));  
    }  
  
    @Override  
    public void enterQuestionCotation(final InterviewCsvParser.QuestionCotationContext ctx) {  
        current.setCotation(extractValue(ctx));  
    }  
  
    @Override  
    public void enterCotationType(final InterviewCsvParser.CotationTypeContext ctx) {  
        current.setCotationType(extractValue(ctx));  
    }  
}
```

25

Listener InterviewJSON

```
public class InterviewJsonListener extends InterviewJsonBaseListener {  
    private final List<InterviewQuestionDTO> questions = new ArrayList<>();  
    private InterviewQuestionDTO current;  
    private List<InterviewAnswer> answers;  
    private Answer answer;  
  
    @Override  
    public void enterQuestion(final InterviewJsonParser.QuestionContext ctx) {  
        answers = new ArrayList<>();  
        current = new InterviewQuestionDTO();  
    }  
  
    @Override  
    public void exitQuestion(final InterviewJsonParser.QuestionContext ctx) {  
        questions.add(current);  
    }  
  
    @Override  
    public void enterQuestionBody(final InterviewJsonParser.QuestionBodyContext ctx) {  
        current.setBody(extractValue(ctx));  
    }  
  
    @Override  
    public void enterQuestionCotation(final InterviewJsonParser.QuestionCotationContext ctx) {  
        current.setCotation(extractValue(ctx));  
    }  
  
    @Override  
    public void enterCotationType(final InterviewJsonParser.CotationTypeContext ctx) {  
        current.setCotationType(extractValue(ctx));  
    }  
}
```

26

Listener InterviewXML

```
public class InterviewXmlListener extends InterviewXmlBaseListener {  
    private final List<InterviewQuestionDTO> questions = new ArrayList<>();  
    private InterviewQuestionDTO current;  
    private List<InterviewAnswer> answers;  
    private Answer answer;  
  
    @Override  
    public void enterQuestion(final InterviewXmlParser.QuestionContext ctx) {  
        answers = new ArrayList<>();  
        current = new InterviewQuestionDTO();  
    }  
  
    @Override  
    public void exitQuestion(final InterviewXmlParser.QuestionContext ctx) {  
        questions.add(current);  
    }  
  
    @Override  
    public void enterQuestionBody(final InterviewXmlParser.BodyContext ctx) {  
        current.setBody(extractValue(ctx));  
    }  
  
    @Override  
    public void enterQuestionCotation(final InterviewXmlParser.QuestionCotationContext ctx) {  
        current.setCotation(extractValue(ctx));  
    }  
  
    @Override  
    public void enterCotationType(final InterviewXmlParser.CotationTypeContext ctx) {  
        current.setCotationType(extractValue(ctx));  
    }  
}
```

27

Listener RequirementsCSV

```
public class RequirementsCsvListener extends RequirementsCsvBaseListener {  
    private final List<RequirementQuestionDTO> questions = new ArrayList<>();  
    private RequirementQuestionDTO current;  
  
    @Override  
    public void enterQuestion(final RequirementsCsvParser.QuestionContext ctx) {  
        current = new RequirementQuestionDTO();  
    }  
  
    @Override  
    public void exitQuestion(final RequirementsCsvParser.QuestionContext ctx) {  
        questions.add(current);  
    }  
  
    @Override  
    public void enterQuestionBody(final RequirementsCsvParser.QuestionBodyContext ctx) {  
        current.setBody(extractValue(ctx));  
    }  
  
    @Override  
    public void enterAnswer(final RequirementsCsvParser.AnswerContext ctx) {  
        final String value = extractValue(ctx);  
        final String[] answer = value.split("<\\s*>");  
        final List<Answer> answers = new ArrayList<>();  
        for (String ans : answer) {  
            answers.add(Answer.valueOf(ans));  
        }  
        current.setPossibleAnswers(answers);  
    }  
}
```

28

Listener RequirementsJSON

```
public class RequirementsJsonListener extends RequirementsJsonBaseListener {  
    private final List<RequirementQuestionDTO> questions = new ArrayList<>();  
    private RequirementQuestionDTO current;  
    private List<Answer> answers = new ArrayList<>();  
  
    @Override  
    public void enterQuestion(final RequirementsJsonParser.QuestionContext ctx) {  
        current = new RequirementQuestionDTO();  
    }  
  
    @Override  
    public void exitQuestion(final RequirementsJsonParser.QuestionContext ctx) {  
        current.setPossibleAnswers(answers);  
        answers = new ArrayList<>();  
        questions.add(current);  
    }  
  
    @Override  
    public void enterQuestionBody(final RequirementsJsonParser.QuestionBodyContext ctx) {  
        current.setBody(extractValue(ctx));  
    }  
  
    @Override  
    public void enterAnswer(final RequirementsJsonParser.AnswerContext ctx) {  
        final String value = extractValue(ctx);  
        answers.add(Answer.valueOf(value));  
    }  
}
```

29

Listener RequirementsXML

```
public class RequirementsXmlListener extends RequirementsXmlBaseListener {  
    private final List<RequirementQuestionDTO> questions = new ArrayList<>();  
    private RequirementQuestionDTO current;  
  
    @Override  
    public void enterQuestion(final RequirementsXmlParser.QuestionContext ctx) {  
        current = new RequirementQuestionDTO();  
    }  
  
    @Override  
    public void exitQuestion(final RequirementsXmlParser.QuestionContext ctx) {  
        questions.add(current);  
    }  
  
    @Override  
    public void enterBody(final RequirementsXmlParser.BodyContext ctx) {  
        current.setBody(extractValue(ctx));  
    }  
  
    @Override  
    public void enterPossibleAnswersList(final RequirementsXmlParser.PossibleAnswersListContext ctx) {  
        List<Answer> answers = new ArrayList<>();  
        for (RequirementsXmlParser.PossibleAnswersContext answerContext : ctx.possibleAnswers()) {  
            String value = extractValue(answerContext);  
            answers.add(Answer.valueOf(value));  
        }  
        current.setPossibleAnswers(answers);  
    }  
}
```

30

Listener InterviewCSV

```
public class InterviewCsvListener extends InterviewCsvBaseListener {

    private final List<InterviewQuestionDTO> questions = new ArrayList<>();
    private InterviewQuestionDTO current;
    private List<InterviewAnswer> answers;
    private Answer answer;

    @Override
    public void enterQuestion(final InterviewCsvParser.QuestionContext ctx) {
        answers = new ArrayList<>();
        current = new InterviewQuestionDTO();
    }

    @Override
    public void exitQuestion(final InterviewCsvParser.QuestionContext ctx) {
        questions.add(current);
    }

    @Override
    public void enterQuestionBody(final InterviewCsvParser.QuestionBodyContext ctx) {
        current.setBody(extractValue(ctx));
    }

    @Override
    public void enterQuestionCotation(final InterviewCsvParser.QuestionCotationContext ctx) {
        current.setCotation(extractValue(ctx));
    }

    @Override
    public void enterCotationType(final InterviewCsvParser.CotationTypeContext ctx) {
        current.setCotationType(extractValue(ctx));
    }
}
```

```
@Override
public void enterType(final InterviewCsvParser.TypeContext ctx) {
    current.setType(extractValue(ctx));
}

@Override
public void enterAnswer(final InterviewCsvParser.AnswerContext ctx) {
    answer = Answer.valueOf(extractValue(ctx));
}

@Override
public void enterAnswerCotation(final InterviewCsvParser.AnswerCotationContext ctx) {
    answers.add(new InterviewAnswer(answer, Cotation.valueOf(extractValue(ctx))));
    current.setPossibleAnswers(answers);
}

private String extractValue(final ParserRuleContext ctx) {
    final Token startToken = ctx.getStart();
    final Token stopToken = ctx.getStop();
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());
    final String intervalText = ctx.start.getInputStream().getText(interval);
    return intervalText.substring(beginIndex:1, intervalText.length() - 1);
}

public List<InterviewQuestionDTO> questions() {
    return questions;
}
```

Listener InterviewJSON

```
public class InterviewJsonListener extends InterviewJsonBaseListener {

    private final List<InterviewQuestionDTO> questions = new ArrayList<>();
    private InterviewQuestionDTO current;
    private List<InterviewAnswer> answers;
    private Answer answer;

    @Override
    public void enterQuestion(final InterviewJsonParser.QuestionContext ctx) {
        answers = new ArrayList<>();
        current = new InterviewQuestionDTO();
    }

    @Override
    public void exitQuestion(final InterviewJsonParser.QuestionContext ctx) {
        current.setPossibleAnswers(answers);
        answers = new ArrayList<>();
        questions.add(current);
    }

    @Override
    public void enterQuestionBody(final InterviewJsonParser.QuestionBodyContext ctx) {
        current.setBody(extractValue(ctx));
    }

    @Override
    public void enterQuestionCotation(final InterviewJsonParser.QuestionCotationContext ctx) {
        current.setCotation(extractValue(ctx));
    }

    @Override
    public void enterCotationType(final InterviewJsonParser.CotationTypeContext ctx) {
        current.setCotationType(extractValue(ctx));
    }
}
```

```
@Override
public void enterType(final InterviewJsonParser.TypeContext ctx) {
    current.setType(extractValue(ctx));
}

@Override
public void enterAnswer(final InterviewJsonParser.AnswerContext ctx) {
    answer = Answer.valueOf(extractValue(ctx));
}

@Override
public void enterAnswerCotation(final InterviewJsonParser.AnswerCotationContext ctx) {
    answers.add(new InterviewAnswer(answer, Cotation.valueOf(extractValue(ctx))));
    current.setPossibleAnswers(answers);
}

private String extractValue(final ParserRuleContext ctx) {
    final Token startToken = ctx.getStart();
    final Token stopToken = ctx.getStop();
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());
    final String intervalText = ctx.start.getInputStream().getText(interval);
    return intervalText.substring(beginIndex:1, intervalText.length() - 1);
}

public List<InterviewQuestionDTO> questions() {
    return questions;
}
```

Listener InterviewXML

```
public class InterviewXmlListener extends InterviewXmlBaseListener {

    private final List<InterviewQuestionDTO> questions = new ArrayList<>();
    private InterviewQuestionDTO current;
    private List<InterviewAnswer> answers;
    private Answer answer;

    @Override
    public void enterQuestion(final InterviewXmlParser.QuestionContext ctx) {
        answers = new ArrayList<>();
        current = new InterviewQuestionDTO();
    }

    @Override
    public void exitQuestion(final InterviewXmlParser.QuestionContext ctx) {
        questions.add(current);
    }

    @Override
    public void enterBody(final InterviewXmlParser.BodyContext ctx) {
        current.setBody(extractValue(ctx));
    }

    @Override
    public void enterQuestionCotation(final InterviewXmlParser.QuestionCotationContext ctx) {
        current.setCotation(extractValue(ctx));
    }

    @Override
    public void enterCotationType(final InterviewXmlParser.CotationTypeContext ctx) {
        current.setCotationType(extractValue(ctx));
    }
}
```

```
@Override
public void enterType(final InterviewXmlParser.TypeContext ctx) {
    current.setType(extractValue(ctx));
}

@Override
public void enterAnswer(final InterviewXmlParser.AnswerContext ctx) {
    answer = Answer.valueOf(extractValue(ctx));
}

@Override
public void enterAnswerCotation(final InterviewXmlParser.AnswerCotationContext ctx) {
    answers.add(new InterviewAnswer(answer, Cotation.valueOf(extractValue(ctx))));
    current.setPossibleAnswers(answers);
}

private String extractValue(final ParserRuleContext ctx) {
    final Token startToken = ctx.getStart();
    final Token stopToken = ctx.getStop();
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());
    final String intervalText = ctx.start.getInputStream().getText(interval);
    return intervalText.replaceAll(regex:"<[^>]+>", replacement:"").trim();
}

public List<InterviewQuestionDTO> questions() {
    return questions;
}
```

Listener RequirementsCSV

```
public class RequirementsCsvListener extends RequirementsCsvBaseListener {

    private final List<RequirementsQuestionDTO> questions = new ArrayList<>();
    private RequirementsQuestionDTO current;

    @Override
    public void enterQuestion(final RequirementsCsvParser.QuestionContext ctx) {
        current = new RequirementsQuestionDTO();
    }

    @Override
    public void exitQuestion(final RequirementsCsvParser.QuestionContext ctx) {
        questions.add(current);
    }

    @Override
    public void enterQuestionBody(final RequirementsCsvParser.QuestionBodyContext ctx) {
        current.setBody(extractValue(ctx));
    }

    @Override
    public void enterAnswer(final RequirementsCsvParser.AnswerContext ctx) {
        final String value = extractValue(ctx);
        final String[] answer = value.split(regex:"/");
        final List<Answer> answers = new ArrayList<>();
        for (String ans : answer) {
            answers.add(Answer.valueOf(ans));
        }
        current.setPossibleAnswers(answers);
    }
}
```

```
@Override
public void enterMinimumRequirement(final RequirementsCsvParser.MinimumRequirementContext ctx) {
    current.setMinimumRequirement(extractValue(ctx));
}

private String extractValue(final ParserRuleContext ctx) {
    final Token startToken = ctx.getStart();
    final Token stopToken = ctx.getStop();
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());
    final String intervalText = ctx.start.getInputStream().getText(interval);
    return intervalText.substring(beginIndex:1, intervalText.length() - 1);
}

public List<RequirementsQuestionDTO> questions() {
    return questions;
}
```

Listener RequirementsJSON

```
public class RequirementsJsonListener extends RequirementsJsonBaseListener {

    private final List<RequirementsQuestionDTO> questions = new ArrayList<>();
    private RequirementsQuestionDTO current;
    private List<Answer> answers = new ArrayList<>();

    @Override
    public void enterQuestion(final RequirementsJsonParser.QuestionContext ctx) {
        current = new RequirementsQuestionDTO();
    }

    @Override
    public void exitQuestion(final RequirementsJsonParser.QuestionContext ctx) {
        current.setPossibleAnswers(answers);
        answers = new ArrayList<>();
        questions.add(current);
    }

    @Override
    public void enterQuestionBody(final RequirementsJsonParser.QuestionBodyContext ctx) {
        current.setBody(extractValue(ctx));
    }

    @Override
    public void enterAnswer(final RequirementsJsonParser.AnswerContext ctx) {
        final String value = extractValue(ctx);
        answers.add(Answer.valueOf(value));
    }
}
```

```
@Override
public void enterMinimumRequirement(final RequirementsJsonParser.MinimumRequirementContext ctx) {
    current.setMinimumRequirement(extractValue(ctx));
}

private String extractValue(final ParserRuleContext ctx) {
    final Token startToken = ctx.getStart();
    final Token stopToken = ctx.getStop();
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());
    final String intervalText = ctx.start.getInputStream().getText(interval);
    return intervalText.substring(beginIndex:1, intervalText.length() - 1);
}

public List<RequirementsQuestionDTO> questions() {
    return questions;
}
```

Listener RequirementsXML

```
public class RequirementsXmlListener extends RequirementsXmlBaseListener {

    private final List<RequirementsQuestionDTO> questions = new ArrayList<>();
    private RequirementsQuestionDTO current;

    @Override
    public void enterQuestion(final RequirementsXmlParser.QuestionContext ctx) {
        current = new RequirementsQuestionDTO();
    }

    @Override
    public void exitQuestion(final RequirementsXmlParser.QuestionContext ctx) {
        questions.add(current);
    }

    @Override
    public void enterBody(final RequirementsXmlParser.BodyContext ctx) {
        current.setBody(extractValue(ctx));
    }

    @Override
    public void enterPossibleAnswersList(final RequirementsXmlParser.PossibleAnswersListContext ctx) {
        List<Answer> answers = new ArrayList<>();
        for (RequirementsXmlParser.PossibleAnswersContext answerContext : ctx.possibleAnswers()) {
            String value = extractValue(answerContext);
            answers.add(Answer.valueOf(value));
        }
        current.setPossibleAnswers(answers);
    }
}
```

```
@Override
public void enterMinimumRequirement(final RequirementsXmlParser.MinimumRequirementContext ctx) {
    current.setMinimumRequirement(extractValue(ctx));
}

private String extractValue(final ParserRuleContext ctx) {
    final Token startToken = ctx.getStart();
    final Token stopToken = ctx.getStop();
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());
    final String intervalText = ctx.start.getInputStream().getText(interval);
    return intervalText.replaceAll(regex:"<[^>]+>", replacement:"").trim();
}

public List<RequirementsQuestionDTO> questions() {
    return questions;
}
```

Gramáticas de Exportação de Templates

OBJETIVO E UTILIZAÇÃO DAS GRAMÁTICAS:

- Estas gramáticas foram criadas com o objetivo de validar que ficheiros de entrevistas ou de requisitos seguem a estrutura definida.
- No âmbito do Projeto Integrador estas gramáticas foram usadas de forma a conseguirmos validar que os templates gerados pelo programa estavam a ser gerados de forma correta e seguindo os requisitos do cliente.
- Para estas funcionalidades não foram utilizados nem Listeners nem Visitors e apenas percorremos as Parse Trees de forma a verificar se existiam ou não erros de sintaxe.
- Estas gramáticas foram concebida para as User Stories: US-1012 (Entrevistas) e US-2003 (Requisitos)



Gramática Interview.g4

ESTRUTURA DESEJADA:

TITLE: Software Engineer Interview

NAME:

EMAIL:

COTATION: 30%

QUESTION TYPE: Choice, with single answer

QUESTION: What is the capital of France?

[1] London

[2] Paris

[3] Berlin

ANSWER:

COTATION: 30%

QUESTION TYPE: True/False

QUESTION: Python is a statically typed language.

ANSWER:

COTATION: 40%

QUESTION TYPE: Short text answer

QUESTION: What is the name of the process to find bugs in software?

ANSWER:

COTATION: 30%

QUESTION TYPE: True/False

QUESTION: Python is a statically typed language.

ANSWER:

Requisitos:

- Para além dos requisitos já mencionados na parte das gramáticas de importação, nesta gramática é exigido que exista um campo para: o título do Template, o nome do candidato, email do mesmo e para as respostas do candidato para cada uma das perguntas.
- O corpo da questão pode conter opções dependendo do tipo e caso contenha o mesmo deve ter o identificador da opção e o texto da opção (Ex.: [1] Texto da Opção).



Gramática Interview.g4

PARSER RULES DA GRAMÁTICA:

```
start: 'TITLE:' text NEWLINE 'NAME:' NEWLINE 'EMAIL:' NEWLINE content+ EOF;
content:
    cotation cotationType NEWLINE 'QUESTION TYPE:' type 'ANSWER:' NEWLINE;
cotation:
    'COTATION:' (TWO_DIGIT_NUMBER | FRACTIONAL_NUMBER | '100');
cotationType: ( '%' | 'POINTS' | 'VALUES');
choice: option NEWLINE (option NEWLINE)+;
option: '[' (TWO_DIGIT_NUMBER | LETTER) ']' text;
text: (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ (('[' text? ']')+ text?);
type: (
    (
        TRUE_FALSE_QUESTION
        | SHORT_TEXT_ANSWER_QUESTION
        | INTEGER_NUMBER_QUESTION
        | DECIMAL_NUMBER_QUESTION
        | DATE_QUESTION
        | TIME_QUESTION
        | NUMERIC_SCALE_QUESTION
    ) NEWLINE 'QUESTION:' text NEWLINE
    | (
        (
            SINGLE_ANSWER_CHOICE_QUESTION
            | MULTIPLE_ANSWER_CHOICE_QUESTION
        ) NEWLINE 'QUESTION:' text NEWLINE choice
    )
);
```



Gramática Interview.g4

LEXER RULES DA GRAMÁTICA:

```
TRUE_FALSE_QUESTION: 'True/False';
SHORT_TEXT_ANSWER_QUESTION: 'Short text answer';
SINGLE_ANSWER_CHOICE_QUESTION: 'Choice, with single answer';
MULTIPLE_ANSWER_CHOICE_QUESTION:
    'Choice, with multiple answers';
INTEGER_NUMBER_QUESTION: 'Integer Number';
DECIMAL_NUMBER_QUESTION: 'Decimal Number';
DATE_QUESTION: 'Date';
TIME_QUESTION: 'Time';
NUMERIC_SCALE_QUESTION: 'Numeric Scale';
TWO_DIGIT_NUMBER: NUMBER NUMBER?;
NUMBER: [0-9];
LETTER: [a-zA-Z];
MEMBER: [.,;:/#+!?*)(] | '\\' | '|' | '-';
FRACTIONAL_NUMBER:
    TWO_DIGIT_NUMBER ('.' | ',') TWO_DIGIT_NUMBER;
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;
NEWLINE: ('\r'? '\n')+;
WS: [ \t\n\r]+ -> skip;
```



Gramática Requirements.g4

ESTRUTURA DESEJADA:

TITLE: Data Scientist

NAME:

EMAIL:

Enter the number of years of experience (integer)

ANSWER:

Select one Degree (none | bachelor | master | PhD)

ANSWER:

Select one or more Programming Languages (python| java | javascript | scala | swift | ruby | r | sql)

ANSWER:

Requisitos:

- Para além dos requisitos já mencionados na parte das gramáticas de importação, nesta gramática é exigido que exista um campo para: o título do Template, o nome do candidato, email do mesmo e para as respostas do candidato para cada um dos requisitos;
- As questões devem começar com um #;
- As opções devem ser mostradas entre parênteses e separadas por um separador “|”.



Gramática Requirements.g4

PARSER RULES DA GRAMÁTICA:

```
start: 'TITLE:' text NEWLINE 'NAME:' NEWLINE 'EMAIL:' NEWLINE content+ EOF;
```

```
text: (TEXT | LETTER | TWO_DIGIT_NUMBER | MEMBER)+ ('#' + text?)?;
```

```
content: '#' text NEWLINE 'ANSWER:' NEWLINE?;
```



Gramática Requirements.g4

LEXER RULES DA GRAMÁTICA:

```
TWO_DIGIT_NUMBER: NUMBER NUMBER?;  
NUMBER: [0-9];  
LETTER: [a-zA-Z];  
MEMBER: [.,;:/+!~*)([\\]] | '\\'' | '\\|' | '\\-';  
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;  
NEWLINE: ('\\r'? '\\n')+;  
WS: [ \\t\\r\\n]+ -> skip;
```



Gramáticas de Validação de Respostas

OBJETIVO E UTILIZAÇÃO DAS GRAMÁTICAS:

- Estas gramáticas foram criadas com o objetivo de validar as respostas contidas nos ficheiros preenchidos pelos candidatos.
- No âmbito do Projeto Integrador estas gramáticas foram usadas de forma a podermos validar o tipo de input esperado e o tipo de input recebido e também para verificar se o ficheiro não foi retornado com uma estrutura errada antes do ficheiro ser carregado para o sistema.
- Para estas funcionalidades foram utilizados Visitors de forma a podermos verificar se o email do ficheiro que está a ser carregado para o sistema e associado a um candidato corresponde ao email desse mesmo candidato.
- Foram também utilizados Visitors para avaliar as respostas dos candidatos, percorrendo estes a Parse Tree e coletando as respostas que depois irão ser usadas para calcular ou definir um resultado final.
- Estas gramáticas foram concebida para as User Stories: US-1017 (Entrevistas), US-2004 (Requisitos), US-1018 (Entrevistas) e US-1015 (Requisitos).



Gramática InterviewAnswers.g4

ESTRUTURA DESEJADA:

TITLE: Software Engineer Interview

NAME: Paulo

EMAIL: micro@email.com

COTATION: 20%

QUESTION TYPE: Choice, with single answer

QUESTION: What is the primary use of HTML?

[1] Styling web pages

[2] Structuring web pages

[3] Server-side computations

ANSWER: 2

COTATION: 20%

QUESTION TYPE: Choice, with multiple answers

QUESTION: Which of the following are sorting algorithms?

[1] Bubble Sort

[2] Quick Sort

[3] Binary Search

[4] Depth-First Search

ANSWER: 1 2

COTATION: 10%

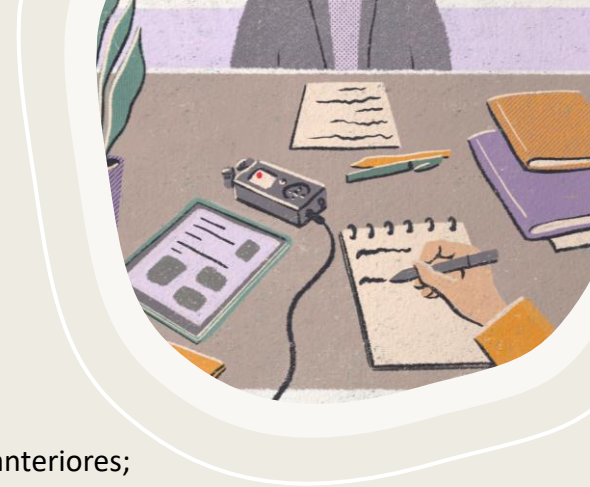
QUESTION TYPE: True/False

QUESTION: Is JavaScript a compiled language?

ANSWER: False

Requisitos:

- Os mesmos requisitos herdados das gramáticas anteriores;
- Os tipos de questão aceitam os seguintes inputs de resposta:
 - Verdadeiro Ou Falso -> "True" ou "False";
 - Resposta Curta -> Qualquer tipo de texto;
 - Resposta com número inteiro -> Números Inteiros;
 - Resposta com número decimal -> Números Decimais;
 - Resposta com Data -> Datas com o formato dd/mm/aaaa;
 - Resposta com Tempo -> Tempos com o formatos hh:mm(:ss)*;
 - Resposta com Escala Numérica -> Números no formato n ou n-n;
 - Escolha singular -> Um caractere ou um número;
 - Escolha Múltipla -> Uma sequência de caracteres ou de números sendo cada número ou caractere separado por um espaço;
- O email deve ter o formato texto@texto.



*opcional

Gramática InterviewAnswers.g4

PARSER RULES DA GRAMÁTICA:

```
start:
    'TITLE:' text NEWLINE 'NAME:' text NEWLINE 'EMAIL:' email NEWLINE content+ EOF;
content:
    cotation cotationType NEWLINE 'QUESTION TYPE:' type NEWLINE?;
cotation:
    'COTATION:' (TWO_DIGIT_NUMBER | FRACTIONAL_NUMBER | '100');
cotationType: ( '%' | 'POINTS' | 'VALUES');
choice: option NEWLINE (option NEWLINE)+;
text: (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ (
    ('[' text? ']')+ text?
)?;
option: '[' (TWO_DIGIT_NUMBER | LETTER) ']' text;
email: TEXT '@' TEXT;
question: 'QUESTION:' text;
type: (
    (
        true_false
        | short_text_answer
        | integer_number
        | decimal_number
        | date
        | time
        | numeric_scale
        | single_answer_choice
        | multiple_answer_choice
    )
);
```



Gramática InterviewAnswers.g4

PARSER RULES DA GRAMÁTICA (CONT.):

```
true_false:
    TRUE_FALSE_QUESTION NEWLINE question NEWLINE true_false_answer;
short_text_answer:
    SHORT_TEXT_ANSWER_QUESTION NEWLINE question NEWLINE text_answer;
single_answer_choice:
    SINGLE_ANSWER_CHOICE_QUESTION NEWLINE question NEWLINE choice single_answer_choice_answer;
multiple_answer_choice:
    MULTIPLE_ANSWER_CHOICE_QUESTION NEWLINE question NEWLINE choice multiple_answer_choice_answer;
integer_number:
    INTEGER_NUMBER_QUESTION NEWLINE question NEWLINE integer_answer;
decimal_number:
    DECIMAL_NUMBER_QUESTION NEWLINE question NEWLINE decimal_answer;
date: DATE_QUESTION NEWLINE question NEWLINE date_answer;
time: TIME_QUESTION NEWLINE question NEWLINE time_answer;
numeric_scale:
    NUMERIC_SCALE_QUESTION NEWLINE question NEWLINE numeric_scale_answer;
true_false_answer:
    'ANSWER:' (
        'True'
        | 'False'
        | 'TRUE'
        | 'FALSE'
        | 'true'
        | 'false'
    );
text_answer: 'ANSWER:' text;
integer_answer: 'ANSWER:' TWO_DIGIT_NUMBER;
decimal_answer: 'ANSWER:' FRACTIONAL_NUMBER;
date_answer: 'ANSWER:' DATE;
time_answer: 'ANSWER:' TIME;
numeric_scale_answer: 'ANSWER:' NUMERIC_SCALE;
single_answer_choice_answer:
    'ANSWER:' (TWO_DIGIT_NUMBER | LETTER);
multiple_answer_choice_answer:
    'ANSWER:' (TWO_DIGIT_NUMBER | LETTER) (
        TWO_DIGIT_NUMBER
        | LETTER
    )+;
```



Gramática InterviewAnswers.g4

LEXER RULES DA GRAMÁTICA:

```
TRUE_FALSE_QUESTION: 'True/False';
SHORT_TEXT_ANSWER_QUESTION: 'Short text answer';
SINGLE_ANSWER_CHOICE_QUESTION: 'Choice, with single answer';
MULTIPLE_ANSWER_CHOICE_QUESTION:
    'Choice, with multiple answers';
INTEGER_NUMBER_QUESTION: 'Integer Number';
DECIMAL_NUMBER_QUESTION: 'Decimal Number';
DATE_QUESTION: 'Date';
TIME_QUESTION: 'Time';
NUMERIC_SCALE_QUESTION: 'Numeric Scale';

TWO_DIGIT_NUMBER: NUMBER NUMBER?;
NUMBER: [0-9];
LETTER: [a-zA-Z];
MEMBER: [.,;:/#π+!~*>() | '\ ' | '|' | '-'];
FRACTIONAL_NUMBER:
    TWO_DIGIT_NUMBER ('.' | ',') TWO_DIGIT_NUMBER;
NUMERIC_SCALE: TWO_DIGIT_NUMBER ('-' TWO_DIGIT_NUMBER)?;
DATE: ('0' [1-9] | '1' [0-9] | '2' [0-9] | '3' [0-1]) '/' (
    '0' [1-9]
    | '1' [0-2]
) '/' TWO_DIGIT_NUMBER+;
TIME: ('0' [0-9] | '1' [0-9] | '2' [0-3]) ':' (
    '0' [0-9]
    | '1' [0-9]
    | '2' [0-9]
    | '3' [0-9]
    | '4' [0-9]
    | '5' [0-9]
) (
    ':' (
        '0' [0-9]
        | '1' [0-9]
        | '2' [0-9]
        | '3' [0-9]
        | '4' [0-9]
        | '5' [0-9]
    )
)?;
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;
NEWLINE: ('\r'? '\n')+;
WS: [ \t\n\r]+ -> skip;
```



Gramática RequirementsAnswers.g4

ESTRUTURA DESEJADA:

TITLE: Software Engineer Requirements

NAME: Paulo

EMAIL: micro@email.com

How long have you been involved in web development? (1 | 2 | 3 | 4 | 5+ years)

ANSWER: 5+ years

Can you list the programming languages you've utilized in your projects? (Ruby | PHP | TypeScript | Go)

ANSWER: RubyPlus

Have you ever developed a mobile app? (Yes | No)

ANSWER: Yes

Requisitos:

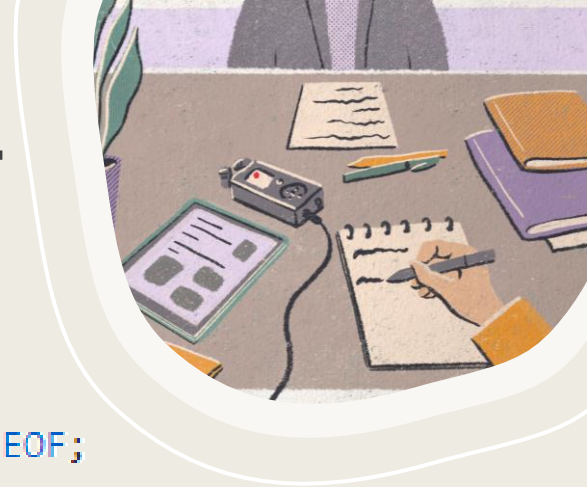
- Os mesmos requisitos herdados das gramáticas anteriores;
- O email deve ter o formato texto@texto.



Gramática RequirementsAnswers.g4

PARSER RULES DA GRAMÁTICA:

```
start:
| 'TITLE:' text NEWLINE 'NAME:' text NEWLINE 'EMAIL:' email NEWLINE content+ EOF;
text: (TEXT | LETTER | TWO_DIGIT_NUMBER | MEMBER)+ ('#' + text?)*;
email: TEXT '@' TEXT;
answer: text;
question: text;
content: '#' question NEWLINE 'ANSWER:' answer NEWLINE*;
```



Gramática RequirementsAnswers.g4

LEXER RULES DA GRAMÁTICA:

```
TWO_DIGIT_NUMBER: NUMBER NUMBER?;  
NUMBER: [0-9];  
LETTER: [a-zA-Z];  
MEMBER: [.,;:/#+!?*)([\\] | '\\'' | '|' | '-';  
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;  
NEWLINE: ('\r'? '\n')+;  
WS: [ \t\n\r]+ -> skip;
```



Visitors Utilizados

Visitor Interview

```
public class InterviewAnswersVisitor extends InterviewAnswersBaseVisitor<String> {  
  
    private String email;  
  
    @Override  
    public String visitStart(final InterviewAnswersParser.StartContext ctx) {  
        visitEmail(ctx.email());  
        return email;  
    }  
  
    @Override  
    public String visitEmail(final InterviewAnswersParser.EmailContext ctx) {  
        email = ctx.getText();  
        return email;  
    }  
}
```

47

Visitor Requirements

```
public class RequirementsAnswersVisitor extends RequirementsAnswersBaseVisitor<String> {  
  
    private String email;  
  
    @Override  
    public String visitStart(final RequirementsAnswersParser.StartContext ctx) {  
        visitEmail(ctx.email());  
        return email;  
    }  
  
    @Override  
    public String visitEmail(final RequirementsAnswersParser.EmailContext ctx) {  
        email = ctx.getText();  
        return email;  
    }  
}
```

48

Visitor Evaluate Interview

```
public class EvaluateInterviewAnswersVisitor extends InterviewAnswersBaseVisitor<Map<QuestionBody, Answer>> {  
  
    private Map<QuestionBody, Answer> answers = new HashMap<>();  
    private String question;  
  
    @Override  
    public Map<QuestionBody, Answer> visitStart(final InterviewAnswersParser.StartContext ctx) {  
        visitChildren(ctx);  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitContent(final InterviewAnswersParser.ContentContext ctx) {  
        visitChildren(ctx);  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitText(final InterviewAnswersParser.TextContext ctx) {  
        question = extractQuestion(ctx);  
        return answers;  
    }  
}
```

49

Visitor Evaluate Requirements

```
public class EvaluateRequirementsAnswersVisitor extends RequirementsAnswersBaseVisitor<Map<QuestionBody, Answer>> {  
    private Map<QuestionBody, Answer> answers = new HashMap<>();  
    private String question;  
  
    @Override  
    public Map<QuestionBody, Answer> visitStart(final RequirementsAnswersParser.StartContext ctx) {  
        visitChildren(ctx);  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitContent(final RequirementsAnswersParser.ContentContext ctx) {  
        visitChildren(ctx);  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitQuestion(final RequirementsAnswersParser.QuestionContext ctx) {  
        visitText(ctx.text());  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitText(final RequirementsAnswersParser.TextContext ctx) {  
        question = extractValue(ctx);  
        return answers;  
    }  
}
```

```
@Override  
public Map<QuestionBody, Answer> visitAnswer(final RequirementsAnswersParser.AnswerContext ctx) {  
    final String answer = extractValue(ctx);  
    question = question.trim();  
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));  
    return answers;  
}
```

```
private String extractValue(final ParserRuleContext ctx) {  
    final Token startToken = ctx.getStart();  
    final Token stopToken = ctx.getStop();  
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());  
    final String intervalText = ctx.start.getInputStream().getText(interval);  
    return intervalText;  
}
```

55

Visitor Interview

```
public class InterviewAnswersVisitor extends InterviewAnswersBaseVisitor<String> {  
  
    private String email;  
  
    @Override  
    public String visitStart(final InterviewAnswersParser.StartContext ctx) {  
        visitEmail(ctx.email());  
        return email;  
    }  
  
    @Override  
    public String visitEmail(final InterviewAnswersParser.EmailContext ctx) {  
        email = ctx.getText();  
        return email;  
    }  
}
```


Visitor Requirements

```
public class RequirementsAnswersVisitor extends RequirementsAnswersBaseVisitor<String> {  
  
    private String email;  
  
    @Override  
    public String visitStart(final RequirementsAnswersParser.StartContext ctx) {  
        visitEmail(ctx.email());  
        return email;  
    }  
  
    @Override  
    public String visitEmail(final RequirementsAnswersParser.EmailContext ctx) {  
        email = ctx.getText();  
        return email;  
    }  
}
```

Visitor Evaluate Interview

```
public class EvaluateInterviewAnswersVisitor extends InterviewAnswersBaseVisitor<Map<QuestionBody, Answer>> {  
  
    private Map<QuestionBody, Answer> answers = new HashMap<>();  
    private String question;  
  
    @Override  
    public Map<QuestionBody, Answer> visitStart(final InterviewAnswersParser.StartContext ctx) {  
        visitChildren(ctx);  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitContent(final InterviewAnswersParser.ContentContext ctx) {  
        visitChildren(ctx);  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitText(final InterviewAnswersParser.TextContext ctx) {  
        question = extractQuestion(ctx);  
        return answers;  
    }  
}
```

Visitor Evaluate Interview (Cont.)

```
@Override
public Map<QuestionBody, Answer> visitType(final InterviewAnswersParser.TypeContext ctx) {
    if (ctx.true_false() != null) {
        visitTrue_false(ctx.true_false());
    } else if (ctx.short_text_answer() != null) {
        visitShort_text_answer(ctx.short_text_answer());
    } else if (ctx.integer_number() != null) {
        visitInteger_number(ctx.integer_number());
    } else if (ctx.decimal_number() != null) {
        visitDecimal_number(ctx.decimal_number());
    } else if (ctx.date() != null) {
        visitDate(ctx.date());
    } else if (ctx.time() != null) {
        visitTime(ctx.time());
    } else if (ctx.numeric_scale() != null) {
        visitNumeric_scale(ctx.numeric_scale());
    } else if (ctx.single_answer_choice() != null) {
        visitSingle_answer_choice(ctx.single_answer_choice());
    } else if (ctx.multiple_answer_choice() != null) {
        visitMultiple_answer_choice(ctx.multiple_answer_choice());
    }
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitTrue_false(final InterviewAnswersParser.True_falseContext ctx) {
    visitQuestion(ctx.question());
    visitTrue_false_answer(ctx.true_false_answer());
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitShort_text_answer(final InterviewAnswersParser.Short_text_answerContext ctx) {
    visitQuestion(ctx.question());
    visitText_answer(ctx.text_answer());
    return answers;
}
```

Visitor Evaluate Interview (Cont.)

```
@Override
public Map<QuestionBody, Answer> visitInteger_number(final InterviewAnswersParser.Integer_numberContext ctx) {
    visitQuestion(ctx.question());
    visitInteger_answer(ctx.integer_answer());
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitDecimal_number(final InterviewAnswersParser.Decimal_numberContext ctx) {
    visitQuestion(ctx.question());
    visitDecimal_answer(ctx.decimal_answer());
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitDate(final InterviewAnswersParser.DateContext ctx) {
    visitQuestion(ctx.question());
    visitDate_answer(ctx.date_answer());
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitTime(final InterviewAnswersParser.TimeContext ctx) {
    visitQuestion(ctx.question());
    visitTime_answer(ctx.time_answer());
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitNumeric_scale(final InterviewAnswersParser.Numeric_scaleContext ctx) {
    visitQuestion(ctx.question());
    visitNumeric_scale_answer(ctx.numeric_scale_answer());
    return answers;
}
```

Visitor Evaluate Interview (Cont.)

```
@Override
public Map<QuestionBody, Answer> visitSingle_answer_choice(
    final InterviewAnswersParser.Single_answer_choiceContext ctx) {
    visitQuestion(ctx.question());
    visitChoice(ctx.choice());
    visitSingle_answer_choice_answer(ctx.single_answer_choice_answer());
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitMultiple_answer_choice(
    final InterviewAnswersParser.Multiple_answer_choiceContext ctx) {
    visitQuestion(ctx.question());
    visitChoice(ctx.choice());
    visitMultiple_answer_choice_answer(ctx.multiple_answer_choice_answer());
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitQuestion(final InterviewAnswersParser.QuestionContext ctx) {
    visitText(ctx.text());
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitTrue_false_answer(final InterviewAnswersParser.True_false_answerContext ctx) {
    final String answer = extractAnswer(ctx);
    question = question.trim();
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitText_answer(final InterviewAnswersParser.Text_answerContext ctx) {
    final String answer = extractAnswer(ctx);
    question = question.trim();
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));
    return answers;
}
```

Visitor Evaluate Interview (Cont.)

```
@Override
public Map<QuestionBody, Answer> visitSingle_answer_choice_answer(
    final InterviewAnswersParser.Single_answer_choice_answerContext ctx) {
    final String answer = extractAnswer(ctx);
    question = question.trim();
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitMultiple_answer_choice_answer(
    final InterviewAnswersParser.Multiple_answer_choice_answerContext ctx) {
    final String answer = extractAnswer(ctx);
    question = question.trim();
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitInteger_answer(final InterviewAnswersParser.Integer_answerContext ctx) {
    final String answer = extractAnswer(ctx);
    question = question.trim();
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitDecimal_answer(final InterviewAnswersParser.Decimal_answerContext ctx) {
    final String answer = extractAnswer(ctx);
    question = question.trim();
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitDate_answer(final InterviewAnswersParser.Date_answerContext ctx) {
    final String answer = extractAnswer(ctx);
    question = question.trim();
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));
    return answers;
}
```

Visitor Evaluate Interview (Cont.)

```
@Override
public Map<QuestionBody, Answer> visitTime_answer(final InterviewAnswersParser.Time_answerContext ctx) {
    final String answer = extractAnswer(ctx);
    question = question.trim();
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitNumeric_scale_answer(
    final InterviewAnswersParser.Numeric_scale_answerContext ctx) {
    final String answer = extractAnswer(ctx);
    question = question.trim();
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));
    return answers;
}

@Override
public Map<QuestionBody, Answer> visitChoice(final InterviewAnswersParser.ChoiceContext ctx) {
    if (ctx.getText() != null) {
        question += " " + extractQuestion(ctx).replace(target:"\n", replacement:" ").replace(target:"\r", replacement:"");
    }

    return answers;
}

private String extractQuestion(final ParserRuleContext ctx) {
    final Token startToken = ctx.getStart();
    final Token stopToken = ctx.getStop();
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());
    final String intervalText = ctx.start.getInputStream().getText(interval);
    return intervalText;
}

private String extractAnswer(final ParserRuleContext ctx) {
    final Token startToken = ctx.getStart();
    final Token stopToken = ctx.getStop();
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());
    String intervalText = ctx.start.getInputStream().getText(interval);
    intervalText = intervalText.replaceFirst(regex:"ANSWER: ", replacement:"");
    return intervalText;
}
```


Visitor Evaluate Requirements

```
public class EvaluateRequirementsAnswersVisitor extends RequirementsAnswersBaseVisitor<Map<QuestionBody, Answer>> {  
  
    private Map<QuestionBody, Answer> answers = new HashMap<>();  
    private String question;  
  
    @Override  
    public Map<QuestionBody, Answer> visitStart(final RequirementsAnswersParser.StartContext ctx) {  
        visitChildren(ctx);  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitContent(final RequirementsAnswersParser.ContentContext ctx) {  
        visitChildren(ctx);  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitQuestion(final RequirementsAnswersParser.QuestionContext ctx) {  
        visitText(ctx.text());  
        return answers;  
    }  
  
    @Override  
    public Map<QuestionBody, Answer> visitText(final RequirementsAnswersParser.TextContext ctx) {  
        question = extractValue(ctx);  
        return answers;  
    }  
}
```

```
@Override  
public Map<QuestionBody, Answer> visitAnswer(final RequirementsAnswersParser.AnswerContext ctx) {  
    final String answer = extractValue(ctx);  
    question = question.trim();  
    answers.put(QuestionBody.valueOf(question), Answer.valueOf(answer));  
    return answers;  
}  
  
private String extractValue(final ParserRuleContext ctx) {  
    final Token startToken = ctx.getStart();  
    final Token stopToken = ctx.getStop();  
    final Interval interval = new Interval(startToken.getStartIndex(), stopToken.getStopIndex());  
    final String intervalText = ctx.start.getInputStream().getText(interval);  
    return intervalText;  
}
```

Gramáticas de Validação de Entrevistas/Requisitos Avaliados

OBJETIVO E UTILIZAÇÃO DAS GRAMÁTICAS:

- Estas gramáticas foram criadas com o objetivo de validar os ficheiros avaliados com notas e resultados finais.
- No âmbito do Projeto Integrador estas gramáticas foram aplicadas de forma a garantir que os ficheiros gerados depois da avaliação e análise automática feita pelo sistema estão a ser criados sem erros de sintaxe.
- Para estas funcionalidades não foram utilizados nem Listeners nem Visitors e apenas percorremos as Parse Trees de forma a verificar se existiam ou não erros de sintaxe.
- Estas gramáticas foram concebida para as User Stories: US-1018 (Entrevistas) e US-1015 (Requisitos).



Gramática EvaluateInterviewAnswers.g4

ESTRUTURA DESEJADA:

TITLE: Software Engineer Interview

NAME: Paulo

EMAIL: micro@email.com

FINAL GRADE: 100%

COTATION: 20%

QUESTION TYPE: Choice, with single answer

QUESTION: What is the primary use of HTML?

[1] Styling web pages

[2] Structuring web pages

[3] Server-side computations

ANSWER: 2

GRADE: 100%

COTATION: 20%

QUESTION TYPE: Choice, with multiple answers

QUESTION: Which of the following are sorting algorithms?

[1] Bubble Sort

[2] Quick Sort

[3] Binary Search

[4] Depth-First Search

ANSWER: 1 2

GRADE: 100%

COTATION: 10%

QUESTION TYPE: True/False

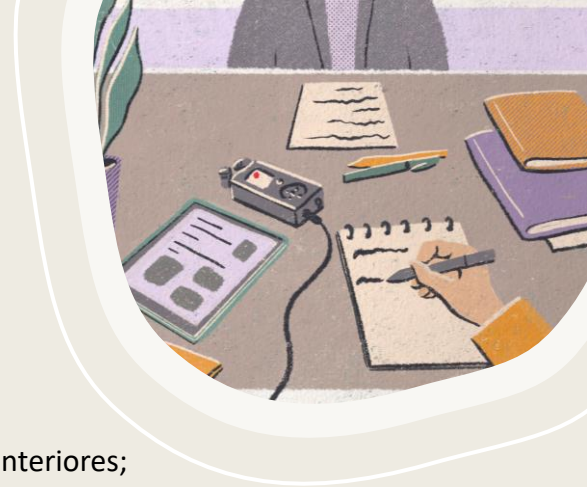
QUESTION: Is JavaScript a compiled language?

ANSWER: False

GRADE: 100%

Requisitos:

- Os mesmos requisitos herdados das gramáticas anteriores;
- As notas devem ser entre 0 e 100 e o seu tipo deve ser percentagem, valores ou pontos;



Gramática EvaluateInterviewAnswers.g4

PARSER RULES DA GRAMÁTICA:

```
start:
    'TITLE:' text NEWLINE 'NAME:' text NEWLINE 'EMAIL:' email NEWLINE 'FINAL GRADE:' cotation
    NEWLINE content+ EOF;
content:
    'COTATION:' cotation NEWLINE 'QUESTION TYPE:' type 'ANSWER:' answer? NEWLINE 'GRADE:' cotation
    NEWLINE?;
cotationType: ( '%' | 'POINTS' | 'VALUES' );
cotation:
    (TWO_DIGIT_NUMBER | FRACTIONAL_NUMBER | '100') cotationType;
choice: option NEWLINE (option NEWLINE)+;
text: (TEXT | TWO_DIGIT_NUMBER | LETTER | MEMBER)+ (
    '[' text? '']')+ text?
    );
answer: text;
option: '[' (TWO_DIGIT_NUMBER | LETTER) ']' text;
email: TEXT '@' TEXT;
type: (
    (
        TRUE_FALSE_QUESTION
        | SHORT_TEXT_ANSWER_QUESTION
        | INTEGER_NUMBER_QUESTION
        | DECIMAL_NUMBER_QUESTION
        | DATE_QUESTION
        | TIME_QUESTION
        | NUMERIC_SCALE_QUESTION
    ) NEWLINE 'QUESTION:' text NEWLINE
    | (
        (
            SINGLE_ANSWER_CHOICE_QUESTION
            | MULTIPLE_ANSWER_CHOICE_QUESTION
        ) NEWLINE 'QUESTION:' text NEWLINE choice
    );
);
```



Gramática EvaluateInterviewAnswers.g4

LEXER RULES DA GRAMÁTICA:

```
TRUE_FALSE_QUESTION: 'True/False';
SHORT_TEXT_ANSWER_QUESTION: 'Short text answer';
SINGLE_ANSWER_CHOICE_QUESTION: 'Choice, with single answer';
MULTIPLE_ANSWER_CHOICE_QUESTION:
    'Choice, with multiple answers';
INTEGER_NUMBER_QUESTION: 'Integer Number';
DECIMAL_NUMBER_QUESTION: 'Decimal Number';
DATE_QUESTION: 'Date';
TIME_QUESTION: 'Time';
NUMERIC_SCALE_QUESTION: 'Numeric Scale';

TWO_DIGIT_NUMBER: NUMBER NUMBER?;
NUMBER: [0-9];
LETTER: [a-zA-Z];
MEMBER: [.,;:/#+!?!* ) ( | '\ ' | '|' | '-' ];
FRACTIONAL_NUMBER:
    TWO_DIGIT_NUMBER ('.' | ',') TWO_DIGIT_NUMBER;
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;
NEWLINE: ('\r'? '\n')+;
WS: [ \t\n\r ]+ -> skip;
```



Gramática EvaluateRequirementsAnswers.g4

ESTRUTURA DESEJADA:

TITLE: Software Engineer Requirements

NAME: Paulo

EMAIL: micro@email.com

RESULT: REJECTED

How long have you been involved in web development? (1 | 2 | 3 | 4 | 5+ years)

ANSWER: 5+ years

REQUIREMENT RESULT: MET

Can you list the programming languages you've utilized in your projects? (Ruby | PHP | TypeScript | Go)

ANSWER: RubyPlus

REQUIREMENT RESULT: NOT MET

JUSTIFICATION: Knowledge of multiple programming languages shows versatility and adaptability.

Have you ever developed a mobile app? (Yes | No)

ANSWER: Yes

REQUIREMENT RESULT: MET

Requisitos:

- Os mesmos requisitos herdados das gramáticas anteriores;
- O resultado final deve ser: "APPROVER" ou "REJECTED";
- O resultado de cada requisito deve ser: "MET" ou "NOT MET";
- No caso do resultado de cada requisito ser "NOT MET" deve ser adicionada uma justificação.



Gramática EvaluateRequirementsAnswers.g4

PARSER RULES DA GRAMÁTICA:

```
start:
    'TITLE:' text NEWLINE 'NAME:' text NEWLINE 'EMAIL:' email NEWLINE 'RESULT:' DECISION NEWLINE
    content+ EOF;
text: (TEXT | LETTER | TWO_DIGIT_NUMBER | MEMBER)+ ('#' + text?);
email: TEXT '@' TEXT;
answer: text;
question: text;
result:
    'REQUIREMENT RESULT:' (
        'MET'
        | 'NOT MET' NEWLINE justification
    );
justification: text;
content:
    '#' question NEWLINE 'ANSWER:' answer NEWLINE result NEWLINE?;
```



Gramática EvaluateRequirementsAnswers.g4

LEXER RULES DA GRAMÁTICA:

```
DECISION: 'APPROVED' | 'REJECTED';
TWO_DIGIT_NUMBER: NUMBER NUMBER?;
NUMBER: [0-9];
LETTER: [a-zA-Z];
MEMBER: [.,;:/#+!?*)([\\] | '\\' | '|' | '-';
TEXT: (LETTER | TWO_DIGIT_NUMBER | MEMBER)+;
NEWLINE: ('\r'? '\n')+;
WS: [ \t\n\r]+ -> skip;
```





Overview

- **Pontos positivos:**

- Vasta variedade de gramáticas;
- Utilização tanto de Visitors como de Listeners.

- **Pontos menos positivos:**

- Organização das regras existentes nas diferentes gramáticas.