Module Interface Specification for Software Engineering

Team 15, ASLingo
Andrew Kil
Cassidy Baldin
Edward Zhuang
Jeremy Langner
Stanley Chan

January 17, 2024

1 Revision History

Date	Version	Notes
Jan 16, 2024	1.0	Andrew, Stan, Edward; Finished back-end MIS breakdown
Jan 17, 2024	1.1	Jeremy, Cassidy; Finished front-end MIS breakdown, fixed some formatting

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at Software Requirements Specification

symbol	description
AC	Anticipated Change
ASL	Shorthand for American Sign Language. It is a form of sign language primarily used in the US and in parts of Canada.
ASLingo	The commercial name for the project.
CV	Refers to Computer Vision, the field of technology that invloves processing visual input ot achieve various means.
HSR	Shorthand for "Health and Safety Requirements", a subsection of Non-Functional Requirements.
FR	Shorthand for Functional Requirements.
LR	Shorthand for "Legal Requirements", a subsection of Non-Functional Requirements.
LFR	Shorthand for "Look and Feel Requirements", a subsection of Non-Functional Requirements.
MSR	Shorthand for "Maintainability and Support Requirements", a subsection of Non-Functional Requirements.
OER	Shorthand for "Operational and Environmental Requirements", a subsection of Non-Functional Requirements.
OpenCV	Refers to the Open Computer Vision Library library available for free to developers in order to develop Computer Vision applications.
M	Module
MG	Module Guide
PR	Shorthand for "Performance Requirements", a subsection of Non-Functional Requirements.
SR	Shorthand for "Security Requirements", a subsection of Non-Functional Requirements.
SRS	Software Requirements Specification
UC	Unlikely Change
UHR	Shorthand for "Usuability and Humanity Requirements", a subsection of Non-Functional Requirements.

Contents

3 Introduction

The following document details the Module Interface Specifications for our project ASLingo. Learning a new language can be an arduous task that only gets more challenging with age, as individuals may find it difficult to dedicate time and effort to it. American Sign Language (ASL) is particularly hard due to its visual and gestural nature, which is not found in other, verbal languages. The purpose of this project is to ease that challenge by providing an online, easy-to-access web platform for individuals to learn new signs and test their comprehension at their own pace in a fun, interactive manner. Focusing in on consistent effort and continuous feedback, ASLingo provides real-time guidance to ensure users stay on track to achieving their goals of learning ASL.

Complementary documents include the Software Requirements Specification and Module Guide. The full documentation and implementation can be found here: ASLingo Github Repo.

4 Notation

The structure of the MIS for modules comes from Hoffman And Strooper 1995, with the addition that template modules have been adapted from Ghezzi Et Al 2003. The mathematical notation comes from Chapter 3 of Hoffman And Strooper 1995. For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Software Engineering.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of Software Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Software Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	Video Input Module
Behaviour-Hiding Module	Hand Sign Recognition Module Controller Module Data Processing Module Machine Learning Module Landing Page Module Exercise Module Login/Sign Up Module
Software Decision Module	Hand Sign Verification Module Data Collection Module Testing and Verification Module Exercise Selection/History Module Account Management Module

Table 1: Module Hierarchy

6 MIS of Hand Sign Recognition Module

6.1 Module

HSR

6.2 Uses

Machine Learning Module, Video Input Module

6.3 Syntax

6.3.1 Exported Access Programs

Name	In	Out	Exceptions
determine_handsign	-	String	TIME_LIMIT_REACHED

6.4 Semantics

6.4.1 State Variables

- MAX_DECISION_FRAMES - Frames needed to determine when the user has settled on a hand sign
- TIMEOUT_LIMIT - Amount of time in seconds before the user automatically fails the question

6.4.2 Environment Variables

None

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

determine_handsign():

- output: The name of the determined handsign
- exception: exc := TIME_LIMIT_REACHED

6.4.5 Local Functions

process_features()

7 MIS of Hand Sign Verification Module

7.1 Module

HSV

7.2 Uses

Hand Sign Recognition Module, Controller

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
verify_handsign	-	Boolean	-

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

verify_handsign():

- output: True/False for if the expected handsign was made
- exception: exc := None

7.4.5 Local Functions

None

8 MIS of Controller Module

8.1 Module

Controller

8.2 Uses

Exercise Module, Hand Sign Verification Module

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
send_requested_handsign	String	None	-
get_requested_handsign	None	String	-
send_passFail	Bool	None	-
get_passFail	None	Bool	-

8.4 Semantics

8.4.1 State Variables

- h - handsign variable to store intermediary data
- pass - Boolean to determine if the question was answered correctly

8.4.2 Environment Variables

None

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

send_requested_handsign():

- output: None

- exception: exc := None

get_requested_handsign():

- output: The expected handsign being asked by the front-end
- exception: exc := None

send_passFail():

- output: None
- exception: exc := None

get_passFail():

- output: The result of comparing the expected answer to what the back-end determined
- exception: exc := None

8.4.5 Local Functions

None

9 MIS of Data Collection Module

9.1 Module

DCM

9.2 Uses

None

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
read_training_set	training_imgs_path	-	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

None

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

read_training_set():

- transition: training.csv updated with raw training data
- output: none
- exception: exc := None

9.4.5 Local Functions

None

10 MIS of Data Processing Module

10.1 Module

DPM

10.2 Uses

Data Collection Module

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
process_training_data	training.csv	-	-

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

None

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

process_training_data():

- transition: training.csv updated with processed training data
- output: none
- exception: exc := None

10.4.5 Local Functions

None

11 MIS of Machine Learning Module

11.1 Module

MLM

11.2 Uses

Data Processing Module

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
SignLangModel	processed_frame_data	String	-

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

None

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

SignLangModel():

- transition: none
- output: Predicted hand sign for given processed frame data
- exception: none

11.4.5 Local Functions

train()

12 MIS of Testing and Verification Module

12.1 Module

Tester

12.2 Uses

Hand Sign Verification Module

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

None

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

None

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

None

12.4.5 Local Functions

testDataCollectionModule()
testDataProcessingModule()
testMachineLearningModule()
testVideoInputModule()
testHandSignRecognitionModule()

testHandSignVerificationModule()

13 MIS of Video Input Module

13.1 Module

Cam

13.2 Uses

None

13.3 Syntax

13.3.1 Exported Access Programs

Name	In	Out	Exceptions
get_frame_data	video input	NumPy ndarray	-

13.4 Semantics

13.4.1 State Variables

None

13.4.2 Environment Variables

None

13.4.3 Assumptions

None

13.4.4 Access Routine Semantics

get_frame_data():

- transition: raw video input is turned into an array of shape (height, width, channels)
- output: the frame data read through the video feed in terms of a NumPy array
- exception: exc := None

14 MIS of Landing Page Module

14.1 Module

Landing Page Module.

14.2 Uses

None

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
getAboutInfo	-	String	-
getInstructions	-	String	-

14.4 Semantics

14.4.1 State Variables

- aboutInfo: string
- instructionInfo: string

14.4.2 Assumptions

None

14.4.3 Access Routine Semantics

getAboutInfo():

- transition:
- output: aboutInfo
- exception:

getInstructions():

- transition:

- output: instructionInfo
- exception:

14.4.4 Local Functions

None

15 MIS of Exercise Module

15.1 Module

Exercise

15.2 Uses

Controller Module

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
getQuestions	-	string, video file	
getAnswers	-	string, CV file	-
getQuestionDifficulty	-	int	
getExerciseType	-	string	
getQuestionScores	-	int	
getExerciseTotalScore	-	int	

15.4 Semantics

15.4.1 State Variables

- questions : seq of string, seq of video files
- answers : seq of string, seq of CV files
- difficultyLevels : seq of $\mathbb{N}$, [1, 5]
- exerciseTypes : seq of string
- questionScores : seq of $\mathbb{Z}$
- exerciseTotalScore : $\mathbb{Z}$, initialized to 0 at the start of each exercise

15.4.2 Environment Variables

- userAnswer := string, CV file, depending on the type of exercise the user is given

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

getQuestions():

- output: questions := seq of string, seq of video files
- exception: None

getAnswers():

- output: answers := seq of string, seq of CV files
- exception: None

getQuestionDifficulty():

- output: difficultyLevels := seq of $\mathbb{N}$
- exception: None

getExerciseType():

- output: exerciseTypes := seq of string
- exception: None

getQuestionScores():

- output: questionScores := seq of $\mathbb{Z}$
- exception: None

getExerciseTotalScore():

- transition: exerciseTotalScore variable is updated after the user answers the question given to them in the exercise. This is updated in the Exercise Selection/History Module to keep track of the users total score for each question.
- output: exerciseTotalScore := seq of $\mathbb{Z}$
- exception: None

15.4.5 Local Functions

updateTotalScore: exerciseTotalScore $\rightarrow$ exerciseTotalScore + givenScore
givenScore: type int ($\mathbb{Z}$), [0, 10]

16 MIS of Exercise Selection/History Module

16.1 Module

Exercise Selection

16.2 Uses

Exercise Module, Account Management Module

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
createQuestionList	-	string, video file	-
createAnswerList	string, video file	string, CV file	-
getUserLevel	-	int	-
updateExerciseQuestionHistory	string, video file	string, video file	InvalidEntry
updateExerciseAnswerHistory	string, CV file	string, CV file	InvalidEntry
updateExerciseScoreHistory	int	int	InvalidEntry

16.4 Semantics

16.4.1 State Variables

- questionList := seq of string, seq of video files
- answerList := seq of string, seq of CV files
- exerciseQuestionHistory := seq of string, seq of video files
- exerciseAnswerHistory := seq of string, seq of CV files
- exerciseScoreHistory := seq of $\mathbb{Z}$

16.4.2 Environment Variables

None

16.4.3 Assumptions

None

16.4.4 Access Routine Semantics

createQuestionList():

- output: questionList := seq of string, seq of video files
- exception: None

createAnswerList():

- output: answerList := seq of string, seq of CV files
- exception: None

getUserLevel():

- output: type int $\mathbb{Z}$
- exception: None

updateExerciseQuestionHistory():

- output: exerciseQuestionHistory := seq of string, seq of video files
- exception: None

updateExerciseAnswerHistory():

- output:exerciseAnswerHistory := seq of string, seq of CV files
- exception: None

updateExerciseScoreHistory():

- output: exerciseScoreHistory := seq of $\mathbb{Z}$
- exception: None

16.4.5 Local Functions

None

17 MIS of Authentication Module

17.1 Module

Application Authentication Module.

17.2 Uses

None

17.3 Syntax

17.3.1 Exported Constants

None

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
logIn	[string, string]	bool	InvalidEmail, IncorrectPassword, UserDoesNotExist
register	[string, string]	-	InvalidRegistration

17.4 Semantics

17.4.1 State Variables

- emailInput: string
- passwordInput: string

17.4.2 Database Environment Variables

- *email*: string
- *password*: string

17.4.3 Assumptions

Using standard alphanumeric Unicode recognized keyboard.

17.4.4 Access Routine Semantics

login(string, string):

- transition:
- output: (validEmail(emailInput) $\wedge$ validPassword(passwordInput)
  $\Rightarrow$ checkIfUserExists(emailInput)
  $\Rightarrow$ checkPassword(checkPassword)
- exception:

  $\neg$validEmail(emailInput) $\Rightarrow$ InvalidEmail

  $\neg$checkIfUserExists(emailInput) $\Rightarrow$ UserDoesNotExist

  $\neg$checkPassword(checkPassword) $\Rightarrow$ IncorrectPassword

17.4.5 Local Functions

- validEmail(string): emailInput = *email*
- validPassword(string): length(emailInput) $\geq 8 \wedge \exists$ {A,B,C...Z} $\wedge \exists$ {0,1,2,..9}
- checkIfUserExists(string): emailInput = *email*
- checkPassword(string): passwordInput = *password*

18 MIS of Account Management Module

18.1 Module

Application Account Module.

18.2 Uses

None

18.3 Syntax

18.3.1 Exported Constants

None

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
updateLevel	int, int	-	-
getLevel	-	int	-

18.4 Semantics

18.4.1 State Variables

- difficulty: int
- score: int
- level: int

18.4.2 Database Environment Variables

- *level*: int

18.4.3 Assumptions

None

18.4.4 Access Routine Semantics

updateLevel(difficulty, score):

- transition: *level* = updateScoreDifficulty(difficulty, level)

- output:
- exception:

getLevel():

- transition:
- output: *level*
- exception:

18.4.5 Local Functions

- updateScoreDifficulty(difficulty, score): return calculated weight for score for given difficulty