Module Interface Specification for Plutos

Team #10, Plutos
Payton Chan
Eric Chen
Fondson Lu
Jason Tan
Angela Wang

January 17, 2025

1 Revision History

Date	Version	Notes
01/13/2024	0.1	Sections 2–5
...	...	...

2 Symbols, Abbreviations and Acronyms

Refer to Section 1.3 of the SRS Documentation for general abbreviations and acronyms. Additional abbreviations and acronyms are listed below.

Table 1: List of Abbreviations and Acronyms

symbol	**description**
M	Module
MG	Module Guide
MIS	Module Interface Specification
R	Requirement
SRS	Software Requirements Specification

Contents

3 Introduction

The following document details the Module Interface Specifications (MIS) for the Plutos project.

Complementary documents include the System Requirement Specifications (SRS) and Module Guide (MG). The full documentation and implementation for the project can be found in the Plutos repository.

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 Plutos.

Table 2: Data Types

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 Plutos 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, Plutos 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

An overview of the module decomposition can be found in Section 5 of the Module Guide document.

6 MIS of OCR Processing Module

6.1 Module

OCR Processing Module

6.2 Uses

This module uses image processing libraries and text parsing utilities. It interacts with the Input Format Module and Machine Learning Module to process data effectively.

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
processImage	Image file (binary)	Text data (structured)	FileError
validateImage	Image file (binary)	Boolean	FormatError

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

This module interacts with the file system to read image files and uses external OCR libraries or APIs to extract text data.

6.4.3 Assumptions

The input image is in a supported format (e.g., JPEG, PNG). The OCR library or API is available and correctly configured.

6.4.4 Access Routine Semantics

processImage():

- transition: Parses the image and converts it into structured text data.
- output: Structured text data extracted from the image.
- exception: Throws a FileError if the image cannot be read or an unsupported format is provided.

validateImage():

- transition: Validates the input image format and dimensions.
- output: Returns true if the image is valid; false otherwise.
- exception: Throws a FormatError if the image format is invalid.

6.4.5 Local Functions

localImageProcessing():

- This function applies pre-processing steps to the image, such as resizing, noise reduction, or thresholding, before OCR is applied.

localTextExtraction():

- This function uses an OCR library to extract raw text from the pre-processed image.

7 MIS of Machine Learning Module

7.1 Module

ML Module

7.2 Uses

OCR Processing Module (MIS of OCR Processing Module)

7.3 Syntax

7.3.1 Exported Constants

N/A

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
categorize	item: Item	category: string	InvalidInputError

7.4 Semantics

7.4.1 State Variables

N/A

7.4.2 Environment Variables

N/A

7.4.3 Assumptions

This specification assumes that the model has been trained and is ready to classify items based on the input data.

7.4.4 Access Routine Semantics

categorize(item: Item):

- transition: N/A
- output: string
- exception: InvalidInputError – thrown if the input is not an Item object

7.4.5 Local Functions

N/A

8 MIS of Budget Calculation Module

8.1 Module

Budget Calculation Module

8.2 Uses

Machine Learning Module (MIS of Machine Learning Module)

8.3 Syntax

8.3.1 Exported Constants

N/A

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
calculate_budget	history: List[Transaction] savings_goals: BudgetGoals	suggested_budget: BudgetGoals	InvalidInputError

8.4 Semantics

8.4.1 State Variables

N/A

8.4.2 Environment Variables

N/A

8.4.3 Assumptions

N/A

8.4.4 Access Routine Semantics

calculate_budget(history: List[Transaction], savings_goals: BudgetGoals):

- transition: N/A
- output: BudgetGoals

- exception: InvalidInputError – thrown if the input is not valid

8.4.5 Local Functions

N/A

9 MIS of Authentication Module

This module manages user authentication, utilizing Firebase Authentication to handle login and registration. It also interacts with React Native components for the user interface.

9.1 Module

Authentication Module

9.2 Uses

N/A

9.3 Syntax

9.3.1 Exported Constants

- **auth**: An instance of Firebase Authentication, initialized using the Firebase app.
- **db**: An instance of Firebase Firestore, initialized using the Firebase app.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
loginWithEmailPassword	email (string), password (string)	user object	InvalidEmail, MissingPassword, InvalidCredential, GeneralError
createUserWithEmailAndPassword	email (string), password (string)	user object	InvalidEmail, WeakPassword, GeneralError

9.4 Semantics

9.4.1 State Variables

The module maintains state for username, password, and credential error messages to support user interactions and error handling.

9.4.2 Environment Variables

The module interacts with the Firebase Authentication API for user management and relies on Firebase configuration to communicate with the backend services.

9.4.3 Assumptions

It is assumed that the Firebase configuration is valid and correctly set up. Network connectivity is also assumed to be available for authentication operations.

9.4.4 Access Routine Semantics

loginWithEmailPassword():

- transition: Authenticates the user with the given email and password, and transitions the application to the "Overview" page if successful.
- output: Returns a user object containing user details upon successful login.
- exception:
  - **InvalidEmail:** The email address is not valid.
  - **MissingPassword:** No password was provided.
  - **InvalidCredential:** Email or password is incorrect.
  - **GeneralError:** A generic error occurred during login.

createUserWithEmailAndPassword():

- transition: Creates a new user account with the provided email and password.
- output: Returns a user object containing user details upon successful registration.
- exception:
  - **InvalidEmail:** The email address is not valid.
  - **WeakPassword:** The password provided is too weak.
  - **GeneralError:** A generic error occurred during registration.

9.4.5 Local Functions

validateCredentials():

- Checks the validity of the entered username and password before attempting login or registration.

handleErrors():

- Maps Firebase error codes to user-friendly error messages displayed on the UI.

10 MIS of Upload Interface Module

10.1 Module

[Short name for the module —SS]

10.2 Uses

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

10.4 Semantics

10.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

10.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

10.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

10.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

10.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

11 MIS of Results Display Module

11.1 Module

[Short name for the module —SS]

11.2 Uses

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

11.4 Semantics

11.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

11.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

11.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

11.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

11.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

12 MIS of Input Format Module

12.1 Module

[Short name for the module —SS]

12.2 Uses

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

12.4 Semantics

12.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

12.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

12.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

12.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

12.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

13 MIS of Output Generation Module

13.1 Module

[Short name for the module —SS]

13.2 Uses

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

13.4 Semantics

13.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

13.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

13.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

13.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

13.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

14 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   During the deliverable, the team was able to divide the work up well and the process went seamlessly for each member finishing up their corresponding section punctually. As well, any questions or concerns that the team had were brought up and discussed in an orderly manner in order to resolve any confusion.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   During the completion of the MG document, we were confused regarding the structure of the DAG diagram, and what the module hierarchy was supposed to look like. The issues surrounding the module hierarchy were resolved during our meeting with Lucas, and the remaining issues we had with the DAG diagram were discussed as a group. A major pain point was the structure of the DAG diagram and after a lengthy discussion, we reached out to Lucas with a couple of solutions to compare which meets the requirements of the section better.

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

   One of the main design decisions that we weren't sure about was what users want to see on their home page. After asking potential users, we found that a potential pain point (when it comes to budgetting) for many users was that they are unaware of how much they've spent over a certain interval, and how much is left in their budget due to the accumulation of small expenses. Furthermore, we asked users about potential features that they'd like to have included into the app and one of the most requested

features were spending metrics. The combination of these feedback points led to us displaying users spending metrics and habits on the home page.

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

   During the completion of the design doc, we expect that the MIS doc may undergo changes as we haven't fully built out all of the modules outlined in the MG/MIS document. We anticipate that there may be changed regarding the software archietecture. However, we may need to update our Hazard Analysis and SRS corresponding to what receipt types are expected after further testing.

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

   In order to make the project better, we were thinking of adding more functionality (i.e. a social aspect where you could split bills with others within the app) and adjusting the classification and parsing model to be able to identify any type of receipt. For example, currently, some receipts heavily abbrieviate their items on the receipt, making it difficult for the classification model to be able to identify the item and categorize it. With more resources, we could potentially integrate our system with common grocery store chains to train the model in order to identify those cryptic items on receipts.

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)

   One of the main design decisions that we stayed away from was the use of too many user inputs. Our solution is to make a more efficient experience for users, and as a result we decided to stay away from designs that would require many user input fields.