SE 3XA3

REST Assured

November 10, 2017

Module Guide

Team 31

Dawson Myers	myersd1	400005616
Yang Liu	liuy136	400038517
Brandon Roberts	roberb1	400018117

Contents

List of Tables

List of Figures

1 Revision History

Date	Version	Notes
10-Nov-2017	0.0	Revision 0

Table 1: **Revision History**

2 Introduction

2.1 Context

The Module Guide (MG)s purpose is to decompose the system into modules to present its modular structures. The MGs creation is guided by principles and patterns, following the creation of the Software Requirements Specification, which specified functional and non-functional requirements to be implemented for the reconstruction of the Sails Live Chrome app, named the REST Assured Test Client. The MG serves in part to demonstrate how the system meets the functional and nonfunctional requirements specified in the SRS. The MG prepares the foundation for the MIS, which states the syntax of all exported routines (inputs, outputs, and exceptions), and semantics (state variables, environment variables, assumptions and special considerations, and access routine semantics) in more detail for the modules specified in the MG.

2.2 Audience

The MG is aimed to provide information for three potential groups of audience:

- Designers/developers: The MG provides module specifications and the traceback to requirements to ensure designers and developers can check for consistency and feasibility, and determine how the project satisfies the requirements specified.
- New project members: The MG provides guidance for any onboarding project member to understand the projects big picture and to be able to quickly locate desired relevant information.
- Maintainers: The MG provides guidance for future maintenance for any anticipated changes. By the outline of the module hierarchy and interpretation of the uses relationships, maintainers are able to infer risk factors and better assess aspects involved in potential changes to the system. Note it is assumed that maintainers hold the responsibility to update relevant sections after making changes.

2.3 Design Principles

Several design principles were used to guide the decomposition of the system into modules, these include information hiding and encapsulation, design for change, low coupling / high cohesion, and avoidance of the existence of cycles in the uses relation hierarchy. The principle of information hiding ensures that modules each holds a secret hidden from the system (for example, the file storage module hides I/O access). The principle of encapsulation ensures that the module interface remains the same (providing the same service to the system) even if the implementation details may change. Designing for change allows the design to be flexible and reusable. Low coupling / high cohesion ensures different modules are not interdependent on one another and that all components of each module are grouped together to contribute to a single well-defined goal of the module. Elimination of cycles in the uses relation ensures modules are not cyclically interdependent and thus do not cause errors or infinite loops.

2.4 Document Structure

The MG is organized with the following structure:

Section 3 Anticipated and Unlikely Changes to the systems implementation used for the traceability matrices.

Section 4 Module Hierarchy details all the modules and their hierarchy by secrets.

Section 5 Connection Between Requirements and Design details the relationship between the requirements to modules.

Section 6 Module Decomposition details the module name, secret, and service/responsibility. for each module.

Section 7 Traceability Matrices provide details on the connection from the requirements to the modules.

Section 8 Uses Hierarchy between Modules details the uses relations interrelationships between modules.

Decomposing a system into modules is a commonly accepted approach to developing. software. A module is a work assignment for a programmer or programming team. We advocate a decomposition based on the principle of information hiding. This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules laid out by Parnas, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.

- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed. The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software.

The rest of the document is organized as follows.

Section 3 lists the anticipated and unlikely changes of the software requirements.

Section 4 summarizes the module decomposition that was constructed according to the likely changes.

Section 5 specifies the connections between the software requirements and the modules.

Section 6 gives a detailed description of the modules.

Section **??** includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section

Section **??** the use relation between modules.

3 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 3.1, and unlikely changes are listed in Section 3.2.

3.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The format of the output data.

AC2: The format of the input parameters.

AC3: The default settings.

AC4: The implementation of the data structure that stores JSON profile.

AC5: The graphical user interface elements used to retrieve user input and their format.

3.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: Storage method of output data.

UC3: There will always be a source of input data external to the software.

4 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

...

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Module RestStubEvaluator Module Profile Module ProfileStore Module Jquery Module RestSuite Module
Software Decision Module	RestStub Module RestChain Structure Module
Module Name	Module Number
Input Module	M1
RestStubEvaluator Module	M2
Profile Module	M3
ProfileStore Module	M4
Jquery Module	M5
RestChain Module	M6
RestStub Module	M7
RestSuite Module	M8

Table 2: Module Hierarchy

5 Connection Between Requirements and Design

The design of the system aims to satisfy the requirements specified in the SRS. The decomposition of the system into component modules analyzes the connection between requirements and modules, which is detailed in the Traceability Matrices section below. To satisfy requirements, several design decisions were made with design principles in mind to guide the decomposition of the system into modules. The I/O class handles input and output of data storage and is used by other modules for this purpose to meet functional requirements, thus demonstrating the design principle of information hiding and encapsulation. The application class RestSuite connects all components in the system to ensure the program can be executed. Nonfunctional requirements related to appearance and usability are satisfied in the RestSuite.display module, since it handles user interaction and interface functions and capabilities. The programming language selected (JavaScript) enables cross-platform support of the application by using the Chrome JavaScript runtime, making it platform agnostic. Another design decision was to implement a drop down menu for the selection of the type of REST requests, which eliminates invalid input and contribute to an intuitive usability design. In addition, buttons are utilized to further simplify request submission for the user. These design decisions holistically contribute to ensure the meeting of the cultural requirements.

??.

6 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

6.1 Hardware Hiding Modules

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

Created By: Brandon Roberts

Created On: October 13, 2017

Last Modified: November 30, 2017

6.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module ~~and the software decision module~~. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

Created By: Brandon Roberts

Created On: October 13, 2017

Last Modified: November 30, 2017

6.2.1 JsonParser Module

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: RestAssured

Created By: Brandon Roberts

Created On: October 13, 2017

Last Modified: November 20, 2017

6.2.2 Input Module

Secrets: Input

Services: Allows the user to specify the test conditions at which the user wishes to run tests with, the order the user wishes to run tests, and what type of tests will be performed.

Implemented By: javascript

6.2.3 RestStubEvaluator Module

Secrets: Test of request.

Services: Tests the success of request based on user-defined parameters.

Implemented By: javascript

Created By: Brandon Roberts

Created On: November 10, 2017

Last Modified: November 30, 2017

6.2.4 Profile Module

Secrets: Profile Information.

Services: Keeps track of user specific settings.

Implemented By: javascript

Created By: Brandon Roberts

Created On: October 13, 2017

Last Modified: November 10, 2017

6.2.5 ProfileStore Module

Secrets: Data Processing

Services: Processes user input and request and stores it for user and application analysis.

Implemented By: javascript

Created By: Brandon Roberts

Created On: October 13, 2017

Last Modified: November 10, 2017

6.2.6 Jquery Module

Secrets: Network connection

Services: Performs network operations such as retrieval of remote information and interaction between computers.

Implemented By: javascript

Created By: Brandon Roberts

Created On: October 13, 2017

Last Modified: October 13, 2017

6.2.7 RestSuite Module

Secrets: Visual Text

Services: Creates multiple instances of RestStubs and outputs the response data to the screen.

Implemented By: javascript

Created By: Brandon Roberts

Created On: October 13, 2017

Last Modified: November 30, 2017

6.2.8 RestStub Module

Secrets: RestStub

Services: Structures tests uniformly for the application to analyse.

Implemented By: javascript

Created By: Brandon Roberts

Created On: October 13, 2017

Last Modified: November 10, 2017

6.2.9 RestChain Module

Secrets: RestChain

Services: Offers a intuitive way for the user to order tests.

Implemented By: javascript

Created By: Brandon Roberts

Created On: October 13, 2017

Last Modified: November 10, 2017

6.3 Software Decision Module

~~[Secrets:] The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.~~

~~[Services:] Includes data structure and algorithms used in the system that do not provide direct interaction with the user.~~

Implemented By: –

6.3.1 Etc.

7 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
FR1	M1, M5, M8
FR2	M1, M2, M4, M5, M8
FR3	M1, M3, M4, M6, M7, M8
NFR1	M1, M3, M4, M6, M7, M8
NFR2	M1, M8
NFR3	M1, M3, M6, M8
NFR4	M1, M3, M6, M8
NFR5	M1, M8
NFR6	M1, M4, M5, M8
NFR7	M1, M4
NFR8	M1, M4
NFR9	M1, M3, M6, M8
NFR10	M1, M3, M6, M8

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC1	M8
AC2	M1
AC3	M1, M3, M4, M6, M8
AC4	M1, M3, M4, M8
AC5	M1, M3, M4, M6, M8

Table 4: Trace Between Anticipated Changes and Modules

8 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas said if two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure **??** illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

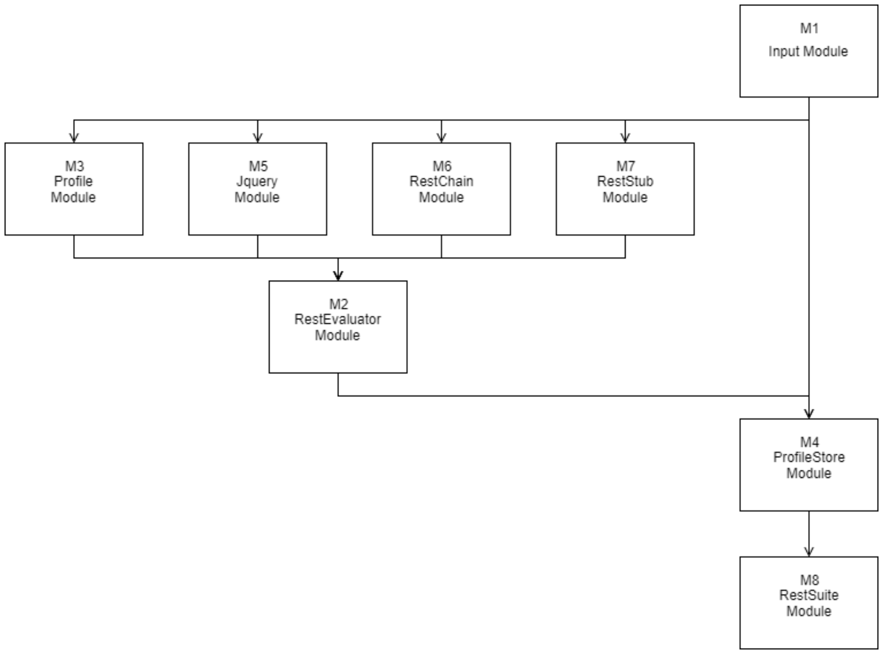
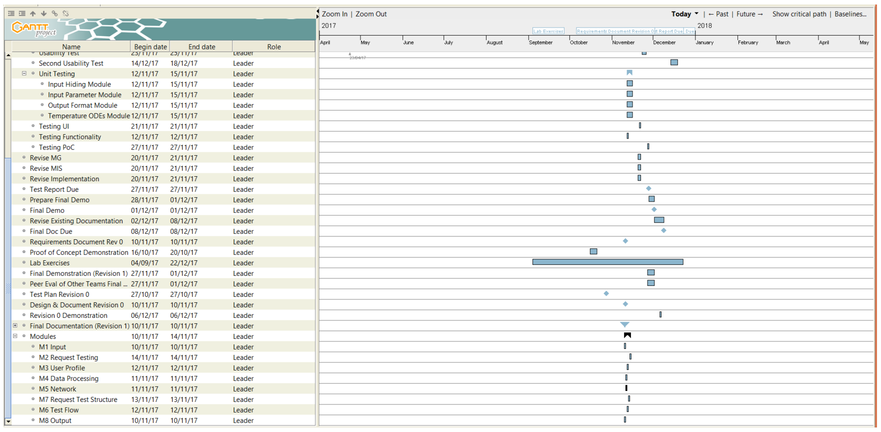


Figure 1: Use hierarchy among modules

9 Gantt



Name	Begin date	End date	Role
Usability Test	23/11/17	23/11/17	Leader
Second Usability Test	14/12/17	18/12/17	Leader
Unit Testing	12/11/17	15/11/17	Leader
Input Hiding Module	12/11/17	15/11/17	Leader
Input Parameter Module	12/11/17	15/11/17	Leader
Output Format Module	12/11/17	15/11/17	Leader
Temperature ODEs Module	12/11/17	15/11/17	Leader
Testing UI	21/11/17	21/11/17	Leader
Testing Functionality	12/11/17	12/11/17	Leader
Testing PoC	27/11/17	27/11/17	Leader
Revise MG	20/11/17	21/11/17	Leader
Revise MIS	20/11/17	21/11/17	Leader
Revise Implementation	20/11/17	21/11/17	Leader
Test Report Due	27/11/17	27/11/17	Leader
Prepare Final Demo	28/11/17	01/12/17	Leader
Final Demo	01/12/17	01/12/17	Leader
Revise Existing Documentation	02/12/17	08/12/17	Leader
Final Doc Due	08/12/17	08/12/17	Leader
Requirements Document Rev 0	10/11/17	10/11/17	Leader
Proof of Concept Demonstration	16/10/17	20/10/17	Leader
Lab Exercises	04/09/17	22/12/17	Leader
Final Demonstration (Revision 1)	27/11/17	01/12/17	Leader
Peer Eval of Other Teams Final ...	27/11/17	01/12/17	Leader
Test Plan Revision 0	27/10/17	27/10/17	Leader
Design & Document Revision 0	10/11/17	10/11/17	Leader
Revision 0 Demonstration	06/12/17	06/12/17	Leader
Final Documentation (Revision 1)	10/11/17	10/11/17	Leader
Modules	10/11/17	14/11/17	Leader
M1 Input	10/11/17	10/11/17	Leader
M2 Request Testing	14/11/17	14/11/17	Leader
M3 User Profile	12/11/17	12/11/17	Leader
M4 Data Processing	11/11/17	11/11/17	Leader
M5 Network	11/11/17	11/11/17	Leader
M7 Request Test Structure	13/11/17	13/11/17	Leader
M6 Test Flow	12/11/17	12/11/17	Leader
M8 Output	10/11/17	10/11/17	Leader

See Gantt Chart at the following url: Team 31 Gant Project