SE 3XA3: Module Guide
The Resume Shotgun

Team 5, Proper Mars Tribe
Gavin Jameson, jamesong
Jeremy Langner, langnerj
Sam Gorman, gormans

March 17, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
March 15	0.1	Everyone contributed to MG

1 Introduction

The Resume Shotgun is a project originally made as a script to automatically scrape the site Glassdoor and apply to jobs found in your area in a certain field. In developing this project, the goal was to make it into a more user-friendly application; to use a structure more akin to MVC, and expand its functionality. This document describes the different modules used in the project, what their purpose is, and how they interact with each other. The SRS describes the requirements for the project as a whole to function, giving context for the existence of the modules described here. The MIS is generated from the source code using doxygen, and details how to interact with modules at a low level, which paired with this document, gives a conclusive high and low level overview of the modules.

1.1 Purpose

The Resume Shotgun is an application that allows users to efficiently search for and apply for jobs utilizing online job portals and websites. The project is implemented with Python and supported libraries to allow for automation and scraping of websites to allow the user to search for jobs based on keywords and locations. Upon successful searches, the user will confirm an application submission based on the results and submit all relevant info for the job.

1.2 Scope of the Module Guide

The Module Guide is a specific design specification developed by David Parnas to ensure key software engineering principles are met. The following software engineering principles will be at the forefront of the implementation; **information hiding, anticipation of change, separation of concerns, and modularization**. The Module Guide alters from the SRS and MIS since it condenses specific information necessary to the module interaction and behaviour within the system to identify and implement design principles listed above.

1.3 Document Structure

This document is is separated into the following sections:

1. Anticipated and Unlikely Changes.
2. Module Hierarchy.
3. Connection Between Requirements and Design.
4. Module Decomposition.
5. Traceability Matrix.
6. Use Hierarchy Between Modules.

Each section has corresponding subsections condensing the relevant information to allow for improved readability and reader comprehension.

2 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 **??**, and unlikely changes are listed in Section **??**.

2.1 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 version of Python used in the project (i.e. updating to most recent, stable release)

AC2: The UI design and layout, during selection of profile information and/or job confirmations.

AC3: The .pdf recognition ability to ensure proper matching with the user and jobs occur.

AC4: The complexity of reCAPTCHA handling and bypassing autonomous browsing checkpoints.

AC5: The amount of input data to search job boards for within a single use.

AC6: The profile data that is required to be set before the project will allow applications (i.e. resume, location).

AC7: The scraping methods for different websites/job boards.

AC8: The amount of different websites/job boards.

2.2 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: The style of the UI (i.e. text vs graphical)

UC2: The input method for the program to take in data.

UC3: The amount of saved profiles or ”users” on a single device.

3 Module Hierarchy

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

M1: User Interface Module

M2: Indeed Link Scraper Module

M3: Glassdoor Link Scraper Module

M4: Application Module

M5: User Profile Module

M6: Site Information Module

M7: Interface Messages Module

M8: PDF Interpretation Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	User Interface Module Application Module User Profile Module Interface Messages Module PDF Interpretation Module
Software Decision Module	Indeed Link Scraper Module Glassdoor Link Scraper Module Site Information Module

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table ??. This table ensures that the any stakeholders or developers wishing to change a FR have a specific reference to where it is within the source code to change if need be.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. 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.

5.1 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

5.2 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: Python

5.2.1 User Interface Module (M??)

Secrets: The method of getting information from the user.

Services: Gets correctly formatted inputs and displays values for the user profile.

Implemented By: menu.py

5.2.2 Application Module (M??)

Secrets: The method of applying to a job link.

Services: Provides connection point from user to a website with confirmation.

Implemented By: apply.py

5.2.3 User Profile Module (M??)

Secrets: The method of storing user information.

Services: Keeps track of and allows changing of user information, can save it to or load it from a non-volatile location.

Implemented By: userProfile.py

5.2.4 Interface Messages Module (M??)

Secrets: The method of displaying information to the user.

Services: Provides text for the visual of a UI, formats messages so that it appears uniform in a window.

Implemented By: menuMessages.py

5.2.5 PDF Interpretation Module (M??)

Secrets: The method of analysing a PDF file.

Services: Extracts information from a PDF that could be helpful in a job search (e.g. skills).

Implemented By: searchPDF.py

5.3 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: –

5.3.1 Indeed Link Scraper Module (M??)

Secrets: The method of getting links.

Services: Gets a list of links from Indeed.

Implemented By: get_links_indeed.py

5.3.2 Glassdoor Link Scraper Module (M??)

Secrets: The method of getting links.

Services: Gets a list of links from Glassdoor.

Implemented By: get_links_glassdoor.py

5.3.3 Site Information Module (M??)

Secrets: Available sites and information required to interact with them.

Services: Provides list of sites that can be scraped, connects a site to a module for scraping it.

Implemented By: sites.py

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements (as listed in the SRS) and between the modules and the anticipated changes (as listed in section ??).

Req.	**Modules**
FR1	M??, M??, M??
FR2	M??
FR3	M??
FR4	M??, M??, M??
FR5	M??, M??, M??
FR6	M??, M??
FR7	M??, M??
FR8	M??

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC??	All
AC??	M??
AC??	M??
AC??	M??, M??
AC??	M?? (using), M?? (storing)
AC??	M??
AC??	M??, M??
AC??	M??, additional module per new site

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of 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.

UsesHierarchy.jpg

Figure 1: Use hierarchy among modules