Software Requirements Specification Template

The following annotated template shall be used to complete the Software Requirements Specification (SRS). The instructor must approve any modifications to the overall structure of this document.

Template Usage:
Text contained within angle brackets (‘<’, ‘>’) shall be replaced by your project-specific information and/or details. For example, <Project Name> will be replaced with either ‘Smart Home’ or ‘Sensor Network’.

Italicized text is included to briefly annotate the purpose of each section within this template. This text should not appear in the final version of your submitted SRS.

This cover page is not a part of the final template and should be removed before your SRS is submitted.

Acknowledgements:
Sections of this document are based upon the IEEE Guide to Software Requirements Specification (ANSI/IEEE Std. 830-1984).

Static_Code_Checker

Software Requirements Specification

V1.0.0

Feb 1 2020

Matt Brown: Designer | Programmer
Will Fitzgerald: Commenter | DevOps | Programmer
Jake Gadaleta: Supreme Leader|DevOps|Programmer
Nick Hertzog: Programmer|Designer|Dev Ops

Revision History

Date	Description	Author	Comments
Feb 9 2020	V1.0.0	All	Created the initial document

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date

Table of Contents

List of Figures

List of Tables

1. Introduction

<The introduction to the Software Requirement Specification (SRS) document should provide an overview of the complete SRS document. While writing this document please remember that this document should contain all of the information needed by a software engineer to adequately design and implement the software product described by the requirements listed in this document.>

1.1 Purpose

This SRS document is meant to explain what this software does, and how it does it. This is mostly a document for our clientele, Teaching Professionals and Interviewers / Interviewees

1.2 Scope

This product runs a localized server using the Python package, Flask. Flask hosts a web server with the ECMAscript(JavaScript) files: BootStrap and JQuery, which is heavily used in our front end design. This product will be able to compile Java source code (.java files) and compare their outputs against that of a master file. We wish to take this time to stress that this is a localized project please do not put it on a server as it opens you up to potential security flaws. This can be used to speed up the general grading process or for an interview to constantly check code. This application works much like LeetCode or HackerRank to compare and contrast code, but is run locally which can a) speed up runtime and b) keep code private

1.3 Definitions, Acronyms, and Abbreviations

<This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.>

1.4 Overview

This document will continue to list out the functionality of this application along with explaining use cases and explain the depency choices that we have made. For more information on how this document is formatted please see the above table.

2. General Description

A system built to help in the process of grading either for teachers or in an interviewing process.

- Teacher Grading
    - Allows teachers to instantly grade a class
- Student
    - Allows students to test their code against inputs / outputs given by a teacher
- Interviewer Grading
    - Allows Interviewers to inspect and grade code
- Interviewee

- Allows interviewers to test their code against inputs / outputs given by a teacher

2.1 Product Perspective

This product stands on its own independent from any existing product. It implements functionality present in the Java Runtime Environment to compile code but that's all
<. "WE NEED A DIAGRAM" >
<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>

2.2 Product Functions

This product can:

- Compile Code and Compare the Outputs
    - Teachers and Interviewers can Compile a list of src files and compare their outputs against a master output to return grading information
    - Students can compile their local code and compare it next to a master output from a professor, this will also support multi test cases

- Grade
    - As mentioned above the user can grade a large selection of source files, this allows teachers to skip over source code that is completely fine and give students who require more time just that

2.3 Users and Characteristics

- Student
    - Allows students to run test given inputs against expected outputs.
- Teachers
    - Allows teachers to grade an entire classes homework at the same time
- Interviewer
    - Allows interviewees to test their code while obscuring the ability to see expected output and input
- Interviewer
    - Allows the interview to grade the interviewees while being able to test and change inputs

2.4 General Constraints

<This subsection of the SRS should provide a general description of any other items that will limit the designer/developer's options for designing/developing the system.>

Support for certain programming languages will be limited as some languages only like to work effectively on certain operating systems. Time will also play into limiting the potential of this software as this is like an 8 week project.

2.5 Assumptions and Dependencies

<This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption might be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.>

- There will be an OS present
- Users will be able to properly setup input and output files
- Users will be able to edit and manipulate code in a "linting"- free environment

2.6 Operating Environment

The best part of this project is that it uses python to run, this gives the ability to cross compile. This means that if the computer is running Python 3.2 or higher the application will also run. We recommend that you use Python 3.7.X as it is currently (2/5/20) stable.
<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.>

3. Specific Requirements

<This will be the largest and most important section of the SRS. The customer requirements are embodied within Section 2 (functions), but this section will give the D-requirements that are used to guide the project's software design, implementation, and testing.

Each requirement in this section should be:

- Correct
- Traceable (both forward and backward to prior/future artifacts)
- Unambiguous
- Verifiable (i.e., testable)
- Prioritized (with respect to importance and/or stability)
- Complete
- Consistent (with other requirements)
- Uniquely identifiable (usually via numbering like 3.4.5.6)

Attention should be paid to carefully organizing the requirements presented in this section so that they may be easily accessed and understood. Furthermore, this SRS is not the software design document, therefore one should avoid the tendency to over-constrain (and therefore design) the software project within this SRS.>

3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.2 Hardware Interfaces

3.1.3 Software Interfaces

3.1.4 Communications Interfaces

3.2 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

3.2.1 REQ-1: Upload input file

3.2.1.1 Description and Priority

This is a feature for users to use where they can upload a text file containing a series of inputs that will be used to check to see if the code is correct. This requirement is of high priority as it is key to the basic functionality of this software. Rating 9.

3.2.1.2 Stimulus/Response Sequences

<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

The user shall be able to click an upload input file button, navigate through their file explorer to locate the file and then upload the file to the software.

3.2.1.3 Functional Requirements

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute a use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use "TBD" as a placeholder to indicate when necessary information is not yet available.>

Requirement of uploading a .txt file

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

3.2.1.3.1 REQ-1: For the input file, the user needs to be able to navigate their systems file explorer to locate the file.

…….

3.2.2 REQ-2: Upload Output File

3.2.2.1 Description and Priority

This feature will be available for users to allow the upload of an output file that will be used alongside the dedicated input file. The output file will just be a formatted text file which will be used to check the output of the users compiled code. This feature is of high priority, rated at a 9.

3.2.2.2 Stimulus/Response Sequences

The user shall be able to click an upload output file button, navigate through their file explorer to locate the file and then upload the file to the software.

3.2.1.3 Functional Requirements

Requirement of uploading a .txt file

3.2.2.3.1 REQ-1: For the output file, the user needs to be able to navigate their systems file explorer to locate the file.

3.2.3 REQ-3 - Teacher View

3.2.3.1 Description and Priority

This feature will be utilized by the teacher to view student output. Also the students code will be linked creating the option to click their code and have it open with the teachers default code editor by the file extension.

3.2.3.2 Stimulus/Response Sequences

The user shall click on a button that will direct them to the teacher view part of the application

3.2.3.3 Functional Requirements

Laptop to access the application and then clicking the "Teacher View" button

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

3.2.1.3.3 REQ-1: User must have the application open to teacher view

3.2.4 REQ-4 - Student View

3.2.4.1 Description and Priority

This feature will be used to allow students to view their own code as well as their teacher's code as the example output. The feature will allow students to view any errors they have and compare to the example output.

3.2.4.2 Stimulus/Response Sequences

The user shall click on a button that will direct them to the student view part of the application

3.2.4.3 Functional Requirements

Laptop to access the application and then clicking the "Student View" button

3.2.1.3.4 REQ-1: User must have the application open and left click student

3.2.5 REQ-5 - Interviewer View

3.2.5.1 Description and Priority

This feature will allow Interviewer's to look at an interviewee's code and to compare that code to the example output provided by the interviewer.

3.2.5.2 Stimulus/Response Sequences

The user shall click on a button that will direct them to the student view part of the application.

3.2.5.3 Functional Requirements

Laptop to access the application and then clicking the "Interviewer View" button.

3.2.1.3.5 REQ-1: User must have the application open and left click the student

3.2.6 REQ-6 - Interviewee View

3.2.6.1 Description and Priority

This feature will be utilized by the teacher to view student output. Also the students code will be linked creating the option to click their code and have it open with the teachers default code editor by the file extension.

3.2.6.2 Stimulus/Response Sequences

The user shall click on a button that will direct them to the interviewee view part of the application.

3.2.6.3 Functional Requirements

Laptop to access the application and then clicking the "IntervieweeView" button.

3.2.1.3.6 REQ-1: User must have the application open and left click the interviewee

3.2.7 REQ-7 - Exporting CSV/PDF Files

3.2.7.1 Description and Priority

This feature will allow the teacher or interviewer to export a file containing the score of each submission. This is of high importance because it will allow the teacher to save time by seeing which students have perfect output or have messed up. Priority level is an 8.

3.2.7.2 Stimulus/Response Sequences

The user shall be able to click an export file button, navigate through their file explorer to select where the file should be saved.

3.2.7.3 Functional Requirements

Having run submissions to create a file.

3.2.1.3.7 REQ-1: For the Exporting CSV/PDF files, the user needs to be able to navigate their systems file explorer to locate where the file to be saved.

3.2.8 REQ-8 - Highlight Differences in Output

3.2.8.1 Description and Priority

This feature will be available for all users being accessed once the user submits their code. Its purpose is to let the user see the differences within the submission and given example output.

3.2.8.2 Stimulus/Response Sequences

The user shall submit their work and see a highlighted difference in the output if they are wrong.

3.2.8.3 Functional Requirements

Submit a working file, we will work out an output if the code does not compile.

3.2.1.3.8 REQ-1: The user must run the code through the application. There must be no differences in the code to have no highlighted text.

3.2.9 REQ-9 - Run Student/Interviewee view code

3.2.9.1 Description and Priority
This is a feature available for all users and can be accessed from each view. A user will be able to upload some code files which they can compile. Priority is high with a rating of 9.

3.2.9.2 Stimulus/Response Sequences
The user shall be able to click the upload button and be able to provide a code file through navigating their file explorer.

3.2.9.3 Functional Requirements
Allow for uploading of python, ruby, java, and Go files.

3.2.1.3.9 REQ-1: User must upload a file and left click the run code button

3.2.10 REQ-10 - Comparing code to given output within output file

3.2.10.1 Description and Priority
This is a feature available to all users that will compare the output generated from their code files and input files to the information within the output file provided. Priority is high with a rating of 9.

3.2.10.2 Stimulus/Response Sequences
The user shall be able to view a results screen/document that will tell them their score that they have received from testing their code.

3.2.10.3 Functional Requirements
Allow for output window or file (xlsx(like)/pdf)
…

…

3.3 Use Cases

3.3.1 Use Case #1

Use Case Name	Upload Input File
Reference	Section 3.2.1
Trigger	The reader presses the upload input file button
Precondition	The user has already selected their designated view
Basic Path	1. The user can select the upload input file button. 2. The user will then be prompted with the file explorer where they can navigate their computer to find the proper input file. 3. The user can then select their designated file and it will be uploaded into the software.
Alternative Paths	None
Postcondition	The selected file is uploaded to the software and it visually shows the user that the upload was successful.
Exception Paths	The user may exit the filer explorer and abandon the file upload.
Other	None

3.3.2 Use Case #2

Use Case Name	Upload Output File
Reference	Section 3.2.2
Trigger	The reader presses the upload outputfile button
Precondition	The user has already selected their designated view
Basic Path	1. The user can select the upload output file button. 2. The user will then be prompted with the file explorer where they can navigate their computer to find the proper output file. 3. The user can then select their designated file and it will be uploaded into the software.
Alternative Paths	None
Postcondition	The selected file is uploaded to the software and it visually shows the user that the upload was successful.
Exception Paths	The user may exit the filer explorer and abandon the file upload.
Other	None

3.3.3 Use Case #3

Use Case Name	Teacher View
Reference	Section 3.2.3
Trigger	The user selects the Teacher button
Precondition	The user has already launched the software and is on the opening screen.
Basic Path	1. The user will be on the homepage of the software 2. The user will need to left click the teacher button to select the teacher view
Alternative Paths	None
Postcondition	The user is navigated to the teacher view page and the page successfully loads
Exception Paths	None
Other	None

3.3.4 Use Case #4

Use Case Name	Student View
Reference	Section 3.2.4
Trigger	The user selects the Student button
Precondition	The user has already launched the software and is on the opening screen.
Basic Path	1. The user will be on the homepage of the software 2. The user will need to left click the teacher button to select the student view
Alternative Paths	None
Postcondition	The user is navigated to the student view page and the page successfully loads
Exception Paths	None

Other	None

3.3.5 Use Case #5

Use Case Name	Interviewer View
Reference	Section 3.2.5
Trigger	The user selects the interviewer button
Precondition	The user has already launched the software and is on the opening screen.
Basic Path	1. The user will be on the homepage of the software 2. The user will need to left click the interviewer button to select the student view
Alternative Paths	None
Postcondition	The user is navigated to the interviewee view page and the page successfully loads
Exception Paths	None
Other	None

3.3.6 Use Case #6

Use Case Name	Interviewee View
Reference	Section 3.2.6
Trigger	The user selects the interviewee button
Precondition	The user has already launched the software and is on the opening screen.
Basic Path	1. The user will be on the homepage of the software 2. The user will need to left click the interviewee button to select the student view
Alternative Paths	None
Postcondition	The user is navigated to the interviewee view page and the page successfully loads
Exception Paths	None
Other	None

3.3.7 Use Case #7

Use Case Name	Exporting CSV/PDF Files
Reference	Section 3.2.7
Trigger	A button present on the teacher | interviewer page
Precondition	The code has been compiled and catalogued by the user
Basic Path	1. Load code 2. Run code 3. Left Click the Button 4. Select the Save Path (handled by browser)
Alternative Paths	None
Postcondition	Storage size is acceptable
Exception Paths	none

Other	None

3.3.8 Use Case #8

Use Case Name	Highlight Differences in Output
Reference	Section 3.2.8
Trigger	Student Runs code
Precondition	code is valid
Basic Path	1. User loads file 2. User runs file
Alternative Paths	None
Postcondition	None
Exception Paths	None
Other	None

3.3.9 Use Case #9

Use Case Name	Run Student/Interviewee view code
Reference	Section 3.2.9
Trigger	Run Button
Precondition	Code is valid
Basic Path	1. User load code 2. User runs code
Alternative Paths	None
Postcondition	None
Exception Paths	None
Other	None

3.3.10 Use Case #10

Use Case Name	Comparing code to given output within output file
Reference	Section 3.2.10
Trigger	Code is run
Precondition	Code is valid
Basic Path	1. Code is run 2. Comparison is run 3. Pretty Printed Comparison is generated and displayed
Alternative Paths	None
Postcondition	None
Exception Paths	None
Other	None

…

3.4 Non-Functional Requirements

Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transaction shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc.).

3.5.1 Performance

We expect in the early stages that this piece of software will run slowly, once a working model is in place a lot of work can be done with Flask to speed up, at the end of the day we would like it to work like a locally run browser.

3.5.2 Reliability

This application will be solid. There aside from leaning on other aspects out of our control (ie. browser errors, errors in users code) this application will run fine.

3.5.3 Availability

Because this application runs on PyPi we can quickly install it on any machine that runs python, this also handles any and all dependencies.

3.5.4 Security

<Example: The server on which the Online Journal resides will have its own security to prevent unauthorized *write/delete* access. There is no restriction on *read* access. The use of email by an Author or Reviewer is on the client systems and thus is external to the system. The PC on which the Article Manager resides will have its own security. Only the Editor will have physical access to the machine and the program on it. There is no special protection built into this system other than to provide the editor with *write* access to the Online Journal to publish an article.>
Because every is run locally and users should only be running their code or the code of their students / interviewees, we hope that there is a level of trust.

3.5.5 Maintainability

The team uses github to sync all code this allows us to track and revert changes that break the application, there is *hopefully* going to be a git action that allows us to know if the code is good.

3.5.6 Portability I really wish jake could spell because right. this is the 5th time i've seen it incorrect

*I've
Because the application is easy to install it will make the software accessible to all accounts

3.5 Design Constraints

We are only controlled by the limits of Html & Css & Javascript

3.6 Logical Database Requirements

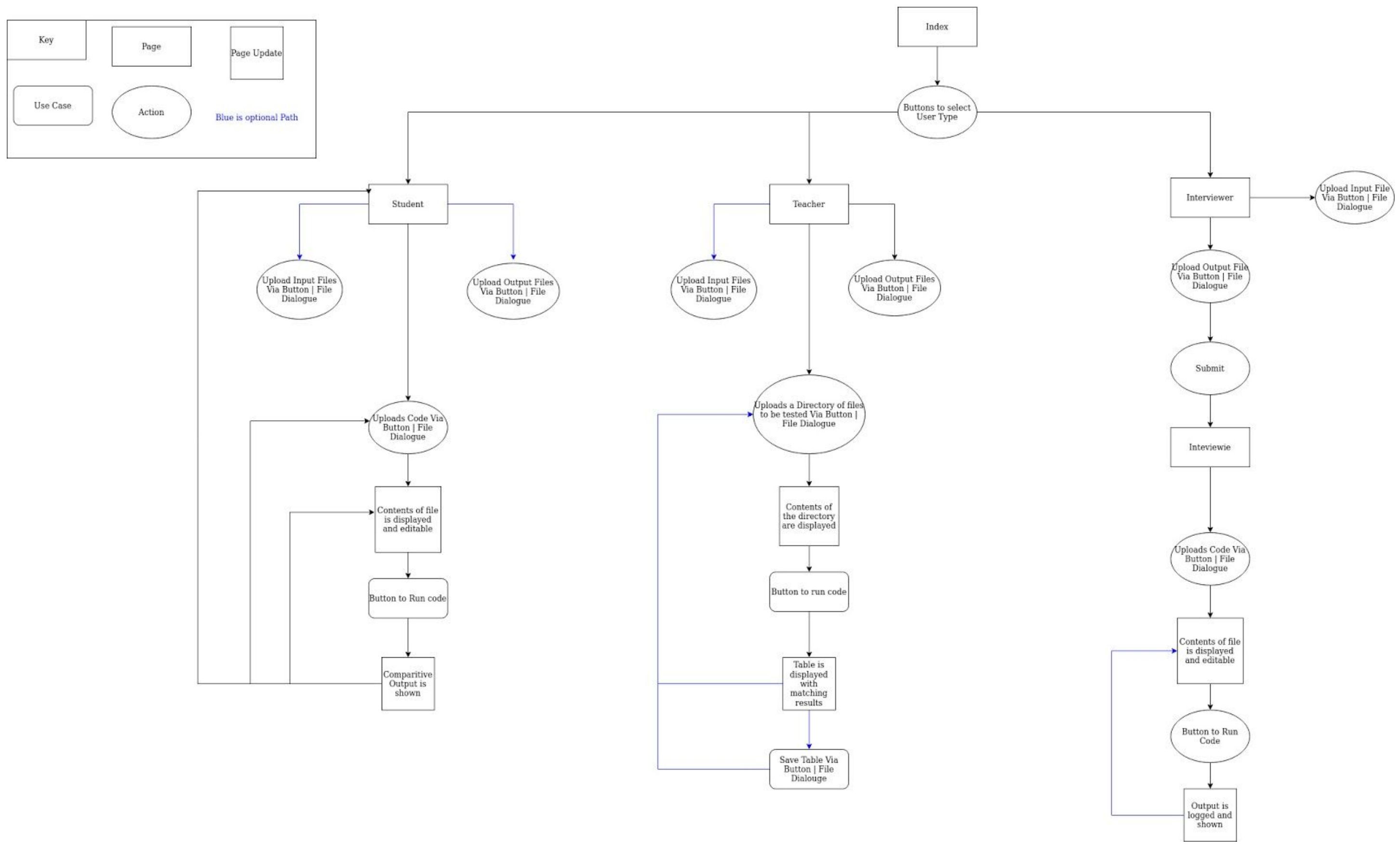
No Database will be needed for this project, at least within the current scope.

4. Analysis Models

<List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable the SRS's requirements.>

3.1 Sequence Diagrams

- All use cases are covered in the general graph as seen below


Key
Page
Page Update
Use Case
Action
Blue is optional Path
Index
Buttons to select User Type
Student
Teacher
Interviewer
Upload Input File Via Button | File Dialogue
Upload Input Files Via Button | File Dialogue
Upload Output Files Via Button | File Dialogue
Upload Input Files Via Button | File Dialogue
Upload Output Files Via Button | File Dialogue
Upload Output File Via Button | File Dialogue
Submit
Inteviewie
Uploads Code Via Button | File Dialogue
Contents of file is displayed and editable
Button to Run code
Comparitive Output is shown
Uploads a Directory of files to be tested Via Button | File Dialogue
Contents of the directory are displayed
Button to run code
Table is displayed with matching results
Save Table Via Button | File Dialouge
Uploads Code Via Button | File Dialogue
Contents of file is displayed and editable
Button to Run Code
Output is logged and shown

5. Change Management Process

This document will be changed by one (1) members of the group and then uploaded back to Github with a commit tag of "Updated the SRS <Date>" and update the table in Section 1 of this document

References

A. Appendices

<Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements. Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.>

A.1 Appendix 1

A.2 Appendix 2