Web based system for TRY application
Version 2.0
Krishna Rao
# Table of Contents

1. Introduction .......................................................................................................................... 4
2. Background .......................................................................................................................... 5
3. Analysis ................................................................................................................................ 5
4. Goals .................................................................................................................................... 6
5. Technology ............................................................................................................................ 7
   5.1. Modular Approach .......................................................................................................... 7
   5.2. Easy to debug .................................................................................................................. 8
   5.3. Management of the File System ..................................................................................... 8
   5.4. Support for executing shell scripts ............................................................................... 8
6. System .................................................................................................................................... 8
   6.1. Design ............................................................................................................................. 8
      6.1.1. Render Login Form ................................................................................................... 9
      6.1.2. Render Form for submitting assignments ............................................................... 9
      6.1.3. Display Output ......................................................................................................... 10
      6.1.4. User Authentication ................................................................................................. 10
      6.1.5. Accept assignment details and files ........................................................................ 10
      6.1.6. Store files and run scripts on files ........................................................................... 10
   6.2. Communication .................................................................................................................. 10
      6.2.1. Action Sequence Diagram ....................................................................................... 10
      6.2.2. Authentication ......................................................................................................... 13
      6.2.3. Activity diagram for running scripts ........................................................................ 14
   6.3. File System ....................................................................................................................... 16
      6.3.1. TRY Directory ........................................................................................................... 16
      6.3.2. Instructor Directory ................................................................................................ 16
      6.3.3. Course Directory ...................................................................................................... 16
6.3.4. Assignment Directory ................................................................. 17
6.3.5. Instructor-Files Directory ............................................................ 17
6.3.6. Submission Directory ................................................................. 18
6.3.7. Scratch Directory ......................................................................... 18
6.3.8. Log Directory .............................................................................. 19
6.3.9. submission-details file ................................................................. 19
6.3.10. TRY Scripts .................................................................................. 20
6.4. Environment variables ..................................................................... 23
  6.4.1. WEB_TRY_INSTRUCTOR_DIR ....................................................... 23
  6.4.2. WEB_TRY_STUDENT_ID ............................................................ 23
  6.4.3. WEB_TRY_ERROR ...................................................................... 23
  6.4.4. WEB_TRY_LATE ........................................................................ 23
6.5. Security ............................................................................................. 24
  6.5.1. SSL ............................................................................................. 24
  6.5.2. Authentication ............................................................................ 24
  6.5.3. File System Access ..................................................................... 24
7. System Execution .................................................................................. 25
8. Conclusions .......................................................................................... 31
9. Future Work .......................................................................................... 32
10. Acknowledgements ............................................................................. 32
11. References ............................................................................................ 33
1. Introduction

In the field of Computer Science, it has become imperative to judge a student’s knowledge by assigning him programming assignments. An assignment is given to the student, and he is expected to submit the completed assignment before the due date set by the instructor. The best way to grade this assignment is to run the project and check its output against various test cases. There are multiple ways of testing such assignments.

The instructor can accept submission from the students until a set due date. After the due date, he can run the program of each student manually and check its output. This method is very simple to implement and is one of the standard methods of submitting assignments. The assignments are saved in a Dropbox and there is usually an application to upload the files to a Dropbox. However, there are few limitations to this approach. The instructor must manually run the project for each student and check its output. This is time-consuming and the instructor may not have the required time/resource to dedicate to this task. Also, a student is given only a single chance to submit his assignment. He will not be able to correct his mistakes after getting the feedback from the instructor. His program should be good enough to pass all the test cases designed by the instructor.

There is a second approach which solves few restrictions mentioned above. The instructor can create the assignment and provide test cases to students. The student will develop the program and run the test cases against these files. Finally, he will submit the screenshot of the result along with the files. This approach minimizes the time taken by the instructor to grade the project as he can check the screenshot to grade it. Also, the student can modify his code till he gets the program to satisfy each test case. This is however not efficient, as it heavily depends on a student’s integrity to ensure correctness of the results. The student may copy the screenshot of another student and submit it as his own. Also, since the test cases are known to the student, it is easier to develop a program conforming to these test cases. In the real world, it is important to run your program against unknown test cases, as they are the source of most of the defects/bugs. Hence many instructors do not use this approach.

The third approach addresses the shortcomings of both the approaches. The instructor can set up an assignment and write scripts to build and run the project. Whenever the student submits the files for the assignment, the scripts provided by the instructor will be run against these files and the student will get immediate feedback from the application. The instructor can now check the logs created by the application to test and grade the student’s work. Thus, this minimizes the
time taken by the instructor in testing and grading the project. The test cases are hidden from the student, and he only gets the feedback regarding successful/unsuccessful execution. This approach is very useful as the student is given a chance to modify his code till he satisfies all the test cases. Since he cannot look at the test cases, he must change his code based on logical presumptions, eventually leading to better code.

Thus, the third approach not only helps the instructor in testing the project, but it also helps the student to become a better developer. This project is based on the third approach, and we will look at a web based approach to achieve the same goal.

2. Background

The first version of the TRY application was developed in 1989 by Kenneth Reek, an associate professor at Rochester Institute of Technology [1]. It was developed as a software package for the UNIX operating systems with the goal of testing student programs and was developed using C programming language [2]. The application was accessed through the terminal and a sample TRY command looked as follows:

```
'try instr_id proj_id files'
```

The student specified the instructor id and the project id of the assignment and the names of the files to be submitted. The application would fetch the mentioned files and the scripts provided by the instructor would be run on them in a secure environment. The output of the submission would be shown immediately to the student after running the scripts.

The application was configurable, but it had some disadvantages. Since it was developed for the UNIX operating system, it was not accessible directly from other operating systems. The student had to install special tools to remotely login to a Unix machine, transfer the files and then submit the assignment using the above command. Also, there was a small learning curve for the student, and the instructor had to take additional efforts in educating the student about the application.

3. Analysis

Due to the widespread use of the Internet and the rapid development of the web frameworks, it is becoming increasingly efficient to create a web application. A Web application is an application program which is hosted on the remote server and is delivered to the user through a browser over the internet. The architecture of the Web application is shown in Figure 1. The application program can be developed using any programming language and can be hosted on any operating system.
The host server is usually referred as a web server. One of the primary goals of the program is to serve the web page to the browser which can be rendered and displayed to the user. Since the user does not access the application directly and uses the browser to access it, the operating system of the web server and that of the user will not matter. This increases the accessibility of the application.

A lot of work has been done in the field of web security and constant efforts are made to make the interaction between the client and the server more secure. Many web technologies have been developed like OAuth Mechanism [5] and the SSL technology [4], which makes the connection very secure and difficult to hack. Since the application is not directly accessible to the user and the interaction between the user and the server is secure, the application program on the server can prevent malicious attacks.

4. Goals

The different goals of the application can be enumerated as follows:

i) Authenticate the student and only allow valid RIT students to access the application. Other users must not be able to access the application.

ii) Accept submission from the student and run the scripts provided by the instructor on the submitted files. The output of the execution and the instructor comments must be shown to the student.

iii) The execution of the scripts on the submitted files must be carried out in a sandbox, and the student must not be able to access the other directories in the TRY System. The other directories used in the system will be shown in the later section.

iv) Accept submission from the student only if the date of the submission is before the due date. The instructor must set the due date.
5. Technology

In this project, I built a web based application for the TRY system. The web server consists of the front-end and the back-end server. The front-end server represents the user interface of the application and the back-end server accepts the submission and runs scripts on them. I have used ReactJS [6] with Redux [7] for developing the client server and NodeJS [8] for developing the server.

ReactJS is a JavaScript library developed by Facebook and Instagram for creating user interfaces [6]. Its primary purpose is to handle the View part of the Model-View-Controller (MVC) model and render elements on the screen. The most time-consuming part of rendering the display is the browser DOM (Document Object Model) manipulation. Whenever the data changes and the elements update, the HTML DOM gets updated, and it takes time to render these changes on the screen. ReactJS makes use of a Virtual DOM to compare the previous DOM tree structure with the new one and handles the final DOM update efficiently. ReactJS also introduces the concept of components, and the user interface is designed using these components. Custom components can be built and used across the application which promotes reusing of the code. ReactJS thus optimizes the rendering on the screen and improves code efficiency by making use of the components.

NodeJS is a JavaScript runtime built on Google’s V8 engine [8]. V8 engine compiles the JavaScript code into native machine code and executes it at high speeds. JavaScript language is an event based language which fires an event whenever any action is performed and these events are collectively known as an event loop. The event loop of NodeJS is a single thread that performs all the I/O operations like reading/writing to network connections or reading/writing to the file-system asynchronously [9]. Whenever the Node application executes an I/O operation, it sends an asynchronous task to the loop along with a callback function, and then proceeds to execute the rest of the program. After the execution of this asynchronous task, the event loop executes the callback function. This asynchronous handling of the tasks helps in building an efficient and scalable web applications capable of handling multiple simultaneous connections.

The advantages of using this framework and library are as follows:

5.1. Modular Approach

ReactJS is a lightweight library that has been built exclusively for rendering the components on the screen. It is very convenient to modularize the application and develop small modules. These
modules can then be integrated to run the entire application. Since this application is the first phase and will have additional features in the future, modularizing the application guarantees extensibility. Modules like Login, file system can be built which can be combined to create a single page web application. In the future, new functionality can be built by developing new modules and attaching to the existing system.

5.2. Easy to debug

Redux JS creates a state tree which captures the state of the variables at a given point of time. Since it is based on functional programming, this state is immutable and cannot be changed. Whenever an action is triggered, new state is created and pushed on top of the existing state. This makes debugging easier by analyzing the erroneous state. Since our application will be used for submitting the assignments correctness of the application is an important part. This can be ensured by using the tree based state of functional ReduxJS.

5.3. Management of the File System

One of the main modules in our system will save files submitted by the students in their respective directories. Since multiple students will use the system at a given point of time it is necessary to handle the file uploads concurrently so that the system can efficiently handle multiple save requests. NodeJS provides excellent support for asynchronous operations and is hence suitable for this application. When a student A submits the file, the server will start off a process for saving these files and until the files are uploaded, the server thread can process other files. Once the files are saved the callback function will be called to process the files submitted by A.

5.4. Support for executing shell scripts

This application will invoke many user-written shell scripts and process their output asynchronously. Hence it is important for the server to be able to execute the shell scripts and catch their output and execution status. NodeJS has many libraries built for this purpose and they can be used extensively in our application.

6. System

6.1. Design

This application is very similar to the original TRY application used at RIT. It can accept large number of files and there is no restriction on the type of the files that can be submitted. This product also provides new features like restricting the type of files that can be submitted or
ensuring the submission of certain files. These features will be described in detail in section 6.3.9. The web server shown in Figure 1 is divided into two servers: the client server representing the front end of the web application and the backend server to accept the submission and run scripts on the files. Henceforth in the report, the client server will be referred as the client and the backend server will be referred as the server. The different components used in the system are shown in Figure 2.

The components on the client server are as follows:

**6.1.1. Render Login Form**
This is responsible for rendering the login screen on the browser. The students use the login page to log into the system. They can log in using the RIT google account. The authentication mechanism will be explained in detail in section 5.2.

**6.1.2. Render Form for submitting assignments**
This is responsible for rendering the form needed for submitting assignments on the browser. Once the student accesses the application, he selects details like an instructor, course, and the assignment. This form also has a file upload module through which the students can submit the relevant files.
6.1.3. Display Output
After submitting the files, various scripts are run on them on the server side. The output of each script and the comments added by the instructor are shown to the student and sample output messages include *init phase completed successfully, build phase completed successfully, etc.*

The components on the server are as follows:

6.1.4. User Authentication
This is used to authenticate the student. A token is generated and is sent to the client server. This token is passed to the server by the client on each request and it will be verified to ensure the authenticity of the request.

6.1.5. Accept assignment details and files
Once the student logs into the system, he will see the list of instructors. On selecting the instructor, the server will send the appropriate course sections followed by the appropriate assignments. This block takes care of these dynamic manipulations.

6.1.6. Store files and run scripts on files
After selecting the details and submitting the files, the student clicks on the Submit button. This block saves the files into the appropriate directory and runs the scripts provided by the instructor against these files.

6.2. Communication
This section shows in detail the communication between the different components in the application like student, client, server, and Google server.

6.2.1. Action Sequence Diagram
The sequence of the actions performed by the components is shown in Figure 3. The diagram recognizes three users viz. Student, Client, and the Server. The student performs all the actions in the browser and hence, the browser has not been shown as a separate user in the diagram.
The chain of actions start when the student accesses the application by entering the url on the browser. He is directed to the client which renders the Login page on the browser. To submit assignments, the student must be authenticated and hence he must click on the Login button to proceed. This application uses Google OAuth technology to authenticate the students.
sequence of actions performed during authentication will be explained in detail in the next section. Once the student is successfully authenticated, the token sent by the Google server is sent to the application server for verification. On successful verification, the server will generate a json web token and send to the client. The client will save the token in the local storage of the student’s browser. This json token has a validity of one hour and it is exchanged between client and server during each request. The server will process the request only if the token is valid and this prevents any unauthorized access.

After the server returns the json token, the client redirects the student to the web page designed for the submission. The student selects the instructor from the list of instructors populated in the html dropdown element. The client sends the instructor id to the server to fetch list of courses offered by the instructor. This list will include the courses setup by the instructor to be accessible via TRY system. The list of courses will be populated in the html dropdown element for the course. The student selects the course and the client sends this course-id to the server to fetch the list of assignments. The student selects the assignment and the client sends this assignment-id to the server. Each assignment can have few restrictions like the allowed file format and it can mandate submission of few files. The instructor must set these settings, which will be explained in the section 6.3.9. The server sends these settings, if any, to the client and they are displayed to the student. The student then uploads files using the file upload module. The client validates the files to check their compliance with the settings setup by the instructor. If the files are valid, the submit button is enabled and the student can submit the files for the selected assignment. If the files are not valid, an appropriate error message will be shown to the student.

The client sends these files to the server. The instructor sets up four scripts viz. init, build, run and cleanup. These scripts are executed against the submitted files and the output of each phase is logged. The activities carried in this stage will be explained in detail in section 6.2.3. The instructor can add comments in each phase and these comments, along with the output of each phase, is sent to the client. Finally, the client displays the message to the student. The sequence of actions concludes when the student clicks on the Logout button. The client deletes the token stored in the local storage of the browser of the student and he must log in again to access the application. Thus, the application is not accessible via the back and forward buttons in the browser after the student logs out. The student can also restart the submission process by clicking a link. This will retain the values of the instructor, course, and the assignment id and will reset other messages and files. He can start the submission process by uploading the files. The sequence of actions for the submission remains the same as explained earlier.
6.2.2. Authentication

This application uses OAuth 2.0 technology for authenticating the students. The sequence of actions executed for authentication can be shown in the following figure:

![Authentication Process Diagram](image)

Figure 4 Google OAuth

The student clicks on the Login button on the web page to log into the application. This application makes use of Google OAuth technology and hence student must log in using the google RIT account having domain g.rit.edu. When the student clicks on the Login button, the request is sent to Google’s OAuth server which redirects him to the sign-in dialog box. He enters his google RIT account credentials and the Google server validates his credentials. If the credentials are valid,
an id_token is generated and is sent to the client. The client now sends this id_token to the server. To verify the authenticity of the login request, the server verifies the id_token received by the client by sending it to the Google server. On successful verification, the server will generate a json web token using the HS256 algorithm [10] and send it to the client. This token will be added to the request header by the client and hence each request will contain the token. The server will send a response to the client only if the token is valid. In the case of a malformed or expired token, an error will be sent to the client and the student must log in again to access the application. This mechanism adds a layer of security and prevents the server from unauthorized access.

6.2.3. Activity diagram for running scripts

The activity diagram is as shown in Figure 5. When the student submits the files for the assignment, the client sends the files to the server. The server executes the scripts provided by the instructor against the submitted files. However, before running the scripts, the TRY system sets up the environment for the execution. Some environment variables are set by the TRY system which can be accessed by any script. Section 6.4 will explain the environment variables in detail. To prevent the student files from accessing the directories inside the TRY system, the files must be executed in a sandbox. Hence, the student's files and the instructor files are copied into a scratch directory.

After its execution, the scripts provided by the instructor are executed. This section will only list the sequence of the execution of the script. The individual script will be explained in detail in the later section. The first script is the init script. If it completes successfully, the build script will be called. If the build script completes successfully, the run script is executed. After the successful completion of the run script, the cleanup script will be called. Finally, the post-execution script will be executed. If any of the above-mentioned scripts fail, the post-execution script will be called.

The output of every script and the comments added by the instructor, if any, are logged into a file. Finally, all the output messages are sent to the client. These messages will be displayed to the student.
Figure 5 Activity diagram for running scripts
6.3. File System

The TRY system consists of multiple directories and each directory is responsible for holding few relevant files and subdirectories. The various directories and files used in the system are shown in the following figure:

![Diagram of File System]

**Figure 6 File System**

6.3.1. TRY Directory

This is the root directory of the application. It has a single directory for every instructor and a directory for storing the different scripts used by the TRY application. These try scripts will be covered in detail in the later section.

6.3.2. Instructor Directory

This is the root directory for each instructor. It has a subdirectory for each individual course and a special subdirectory called Scripts that will contain all the common scripts used by the instructor.

6.3.3. Course Directory

This directory is unique for each course taught by the instructor. It contains a subdirectory for each assignment.
6.3.4. Assignment Directory

This directory is unique for each assignment. It contains all the scripts and the files provided by the instructor and stores the files submitted by each student. It consists of four subdirectories viz. Log-Directory, Scratch-Directory, Instructor-files directory, and Submission directory. Each subdirectory has a special purpose which will be explained in the following sections. This directory also has a file called submission-details. It is used to provide details like due date and mandatory files for the submission. The format of this file will be explained in section 5.3.9.

6.3.5. Instructor-Files Directory

This directory contains all the files provided by the instructor for that assignment. It stores all the scripts provided by the instructor for each phase and the supporting files needed to run the assignment. It has two subdirectories called Files and Scripts to store the above-mentioned files.

6.3.5.1. Files

This directory contains the files that are needed to build and run the assignment successfully. It can contain the file carrying the main function or the supplemental functions needed to run the test cases. It can also store the file containing the test cases.

6.3.5.2. Scripts

This directory contains the scripts corresponding to each phase. It can also store any supporting scripts as deemed necessary by the instructor. The following files correspond to each phase of the execution of the application:

6.3.5.2.1. init

This is the first script to be executed after copying the files into the scratch directory. It can be used to do various things like set the custom environment variables or check a submission before building the project. If the script throws any error or sends out an error message through standard error (stderr), the build script will not be executed. The post-execution script will be invoked and the error message will be displayed to the student.

This script is mandatory. However, it can be left blank if there are no actions to be executed before running the build script.

6.3.5.2.2. build

This script is used to build the project by compiling the submitted files. If there are compilation errors in the project, the failure and the compilation error would be logged. When the application encounters any error in the build phase, or receives an error message through the standard error, the execution of run phase is suspended and the run script will not be invoked.
The post-execution script will be invoked and the error message will be displayed to the student.

This script is mandatory.

6.3.5.2.3. run
This file is used to run the project for that assignment. The instructor can provide tests which will be used to test the project. As mentioned earlier, the errors in this phase are also logged and displayed to the student. The output of each test case can also be shown to the student depending on the run script written by the instructor. This phase can receive error messages through stderr and the execution of the cleanup script will not be blocked. The instructor can use stderr to provide error message for individual test cases. However, there should be no error with a non-zero error status.

This script is mandatory.

6.3.5.2.4. cleanup
This is the last instructor provided script to be executed on the project files. It can have the functionality of cleanup of some variables or other post-run functionalities as deemed suitable by the instructor. This script is mandatory. However, it can be left blank if there are no actions to be executed after executing the run script.

6.3.6. Submission directory
This directory has a subdirectory for each student which is named after their student id. When the student submits the files for submission, a temporary directory called temp is created to hold the files. After running the scripts on these files, the files will either be discarded or saved based on the successful/unsuccessful execution of the scripts. In either case, the temp directory is deleted after running the scripts. Each student subdirectory contains the files submitted by the student and a special subdirectory called Previous-Submission. This subdirectory stores the files submitted during the previous submission. Before saving the files submitted in the current submission, the application copies all the files and stores them in Previous-Submission and then proceeds to save the current files.

6.3.7. Scratch Directory
This serves as a temporary work directory wherein the scripts are run on the submitted files during each phase of the application. After the student submits the assignment files, before executing the scripts on them, the application creates a directory for that student in the scratch directory. Then the files provided by the instructor and the files submitted by the student are copied into the
student scratch directory. The utility functions and the different try scripts are also copied. After the execution of all the phases, the student scratch directory is deleted and the files submitted by the student are saved in the submission directory depending on the status of the execution of the scripts.

6.3.8. Log Directory

This directory has a log file for each student and it is named after their student-id. The sample log record is shown in Figure 5. The log record starts with the record of the username and the time of recording. The time is recorded in the following format: mm/dd/yyyy @ hh:mm:ss. The successful execution of each phase or error, if any, is also logged by the application. Finally, the comments provided by the instructor are also logged.

Log recorded for ksr9048 at 11/7/2016 @ 17:21:25
Comment added by instructor – In Init Script
Init phase completed successfully
Build phase completed successfully
Hello from child!
Hello, World
Run phase completed successfully
Comment added by instructor - Good job!!
Cleanup phase completed successfully
******************************************************************************

Figure 7 Sample Log file

6.3.9. submission-details file

This file contains the assignment specific settings. Each setting should be specified on a new line. A sample file is shown in figure 8.

This file can be used to restrict the type of files submitted by the student by mentioning the valid extensions. Multiple extensions can be separated by commas. For instance, .txt, .java allows the students to submit only .txt or .java files. This should be mentioned by naming the line as ‘Restricted files:’. It can also be used to ensure the submission of certain mandatory files. For
instance, *hello.java, readme.txt* specifies that students must submit these files for the submission. This should be mentioned by naming the line as *‘Mandatory files:’.*

The third feature of this file allows the instructor to specify the due date for the assignment. It needs to be specified in the mm/dd/yyyy hh:mm:ss format and should be mentioned by naming the line as *‘Due Date:’.* If the date of the submission is later than this due date, the submitted files will be stored in a subdirectory called LATE inside the student’s submission directory, to indicate late submission. This file can also be used to specify the late due date. This data should also be specified in the mm/dd/yyyy hh:mm:ss format and should be mentioned by naming the line as *‘Late Due Date:’.* If the assignment is submitted after this late due date, the submission will be discarded and no scripts will be run on it. The student will get the appropriate message indicating the non-acceptance of the submission.

The third setting i.e. Due Date is mandatory. All the other settings are optional. If the Due date is missing from the file, the error message will be displayed to the student. It will not be possible to submit the assignment till the error is fixed.

```
<table>
<thead>
<tr>
<th>Restricted files:</th>
<th>.txt, .java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory Files:</td>
<td>Child.java</td>
</tr>
<tr>
<td>Due date:</td>
<td>10/30/2016 23:59:59</td>
</tr>
<tr>
<td>Late Due date:</td>
<td>10/31/2016 23:59:59</td>
</tr>
</tbody>
</table>
```

*Figure 8 Sample submission-details file*

### 6.3.10. TRY Scripts

In addition to the instructor provided scripts, there are some scripts provided by the TRY application for facilitating the execution of all the scripts and saving the files submitted by the student. The try scripts used by the system are as follows:

#### 6.3.10.1. copy-script

This is the first script to be executed after the student submits the assignment files to the system. It creates a scratch directory for the student named after his username, inside the Scratch directory of the assignment. Once the student scratch directory is created, the files submitted by the student are copied into it. The files and scripts provided by the instructor are also copied into it. The scripts mentioned in the following sections are also copied into this student scratch directory. Thus, the environment is setup for running the following scripts.
6.3.10.2. Utility functions
There are some utility functions which can be used by the instructor in his scripts for providing feedback to the students. The use of echo is not encouraged in this application. It can be used by the instructor for debugging his scripts but it should not be present in the live scripts. If the application encounters an echo message, a warning message will be added to the output. The different utility functions used in the application are:

6.3.10.2.1. echoErrorMessage
The instructor can use this function to provide any error messages from the script. For instance, if the instructor wants to raise an error if certain submission criteria is not met, he can raise an error and use this function to give an error message to the student. The following example shows its usage:

```bash
echoErrorMessage "Hello.java file has not been submitted"
```

*Figure 9 Use of echoErrorMessage() function*

6.3.10.2.2. comment
The instructor can use this function to provide feedback to the students. As mentioned earlier, the use of echo function is not encouraged in this application and this utility function can be used to provide any feedback to the student. The following example shows its usage:

```bash
comment "Good job! Start preparing for next assignment!"
```

*Figure 10 Use of comment() function*

6.3.10.2.3. saveFile
The instructor can create new files from his scripts. For instance, he can create a log file and record certain aspects of the program. He can use this function to store the file in the student’s submission directory. This function creates the file in a temporary directory inside the student’s scratch directory. The post-execution script then moves this file into the student’s submission directory. The following example shows its usage:

```bash
saveFile log_file
```

*Figure 11 Use of saveFile() function*

6.3.10.3. try-init
This is the script executed by the TRY application for running the init phase. It sets up the utility function in the current shell and invokes the init script provided by the instructor. Since this script
sets up the utility functions in the shell, the instructor can use the functions directly in his script without sourcing the file. The try-init script is written as follows:

```bash
. ./utility-functions.sh
. ./init
```

*Figure 12 try-init script*

### 6.3.10.4. try-build

This is the script executed by the TRY application for running the build phase. It sets up the utility function in the current shell and invokes the build script provided by the instructor. The try-build script is written as follows:

```bash
. ./utility-functions.sh
. ./build
```

*Figure 13 try-build script*

### 6.3.10.5. try-run

This is the script executed by the TRY application for executing the run phase. It sets up the utility function in the current shell and invokes the run script provided by the instructor. The try-run script is written as follows:

```bash
. ./utility-functions.sh
. ./run
```

*Figure 14 try-run script*

### 6.3.10.6. try-cleanup

This is the script executed by the TRY application for executing the cleanup phase. It sets up the utility function in the current shell and invokes the cleanup script provided by the instructor. The try-cleanup script is written as follows:

```bash
. ./utility-functions.sh
. ./cleanup
```

*Figure 15 try-cleanup script*
6.3.10.7. post-execution

This is the last script to be executed on the files. After running all the four phases, this script is executed. If the application encounters any error in init, build or run phase, or any error message in init or build phase, the submission is discarded and no files are saved. If there is no error, the files are saved in the student’s submission directory. Before saving the files in the student’s directory, if there are previously saved files, they will be saved in a new subdirectory called Previous-Submission. Finally, the temp directory in the student’s submission directory is deleted and the student’s scratch directory is also deleted.

6.4. Environment variables

This application supports environment variables set by the application as well as the custom variables created by the instructor. Before executing the copy-script, the application sets few environment variables which can be used in the instructor scripts. The application sets the following environment variables:

6.4.1. WEB_TRY_INSTRUCTOR_DIR

Apart from the scripts created for individual assignments, the instructor can also use few common scripts created by him and placed in the Instructor’s directory. The instructor can use this variable to refer such common scripts placed in his directory. The current value of this variable is set as: WEB_TRY_INSTRUCTOR_DIR = ‘../../../../’ and this value is the relative path from the scratch directory.

6.4.2. WEB_TRY_STUDENT_ID

This variable stores the username of the current student.

6.4.3. WEB_TRY_ERROR

This variable is initialized to false. If the application encounters any error in init, build or run phase, this variable is set to true. It can be used in the cleanup script by the instructor to take some actions on encountering an error. The post-execution script also uses this value to save or discard the submitted files.

6.4.4. WEB_TRY_LATE

This variable is initialized to false. If the date of submission of the assignment is later than the due date specified by the instructor, this variable is set to true.
6.5. Security

This application has been securely designed and uses modern web security features.

6.5.1. SSL

This application has been designed using SSL or the Secure Sockets Layer technology. The server uses an SSL certificate and when the student accesses the application on the browser, the server sends this certificate to the browser along with the server’s public key. Rest of the communication is encrypted by using the session key and hence it provides security over the network.

6.5.2. Authentication

This application uses OAuth 2.0 technology for authenticating the students. As mentioned in section 6.2., the application adds a double layer of security by generating and verifying two tokens. Also, the client sends the token to the server for each request and this token must be valid for the server to process the request. This token is currently valid for an hour. If the token is malformed or expired, an error is sent to the client and the student must log in again to access the application. This ensures security for the students even if they use public machine and forget to logout from the application.

6.5.3. File System Access

The TRY Directory is the root directory of the application. It consists of the individual subdirectory for each instructor. A new security group is created to access this root directory. Only the web service and the instructor accounts are added to this security group and consequently, only they can access the file system. The server will block any other unauthorized user. This group policy ensures authorized access and prevents an attack or snooping by any other user.
7. System Execution

The execution steps of the application can be shown as follows:

i) The student enters the URL of the application in the browser.

![Login screen](image)

*Figure 16 Login screen*
ii) On clicking the Login button, the student is redirected to the Google sign in screen. The student can access the application only by using the Google RIT credentials. Personal Google accounts cannot be used to authenticate the student.

Figure 17 Sign in screen
iii) On successful authentication, the student is redirected to the screen for submitting the assignment. Initially, the submit button is disabled and it will be enabled only if he selects valid instructor, course, and assignment. If any file related settings are specified, then he must select valid files to proceed with submission.

![Submission screen](image)

Figure 18 Submission screen
iv) The student selects the instructor, course, and the assignment. If the instructor has specified any settings for the assignment, it will be displayed to the student. The student clicks on *Click to Upload Files* button to open the file upload module and select files.

![Figure 19 File Upload screen](image-url)
v) After selecting the files, the client validates the selected files. If the validation fails, the error message is shown to the student. If there is no error, the web page displays the files selected by the user and the submit button is enabled.

![Selected files confirmation screen](image)

*Figure 20 Selected files confirmation screen*
vi) The student click on the Submit button to submit the files for the assignment. The client sends the files to the server. The server runs the scripts provided by the instructor against these files. The output of each phase and the instructor comments, if any, are recorded. Finally, the entire message is displayed to the student.

The student can click on Sign out to close the application or he can restart the submission process by clicking the link. The application will reset all the messages and the files and will retain the instructor, course, and the assignment.

![Image of submission process]

**Figure 21 Output of the submission**
8. Conclusions

The web based approach for the TRY application is very promising and in addition to the ease of using it, it also provides the security needed to run such sensitive applications. As seen in the previous sections, the application is very simple to use for an instructor as well as a student. Since the student accesses the application over a browser and the user interface is self-explanatory, there will not be a learning curve for the student. The following goals were achieved in the application:

i) Authenticate the student and only allow valid RIT students to access the application. Other users must not be able to access the application.

   The application allows access only if the student logs in using the google-RIT account. If the personal google account is used, the authentication fails and hence, he will not be allowed to access the application. Furthermore, the server processes the requests only if the tokens are valid. Any malicious user with an URL of the application will not be able to send a request to the server as he will not have the token signed by the server. Thus, the use of SSL combined with OAuth and json web token ensures security and privacy of the student and the application.

ii) Accept submission from the student and run the scripts provided by the instructor on the submitted files. The output of the execution and the instructor comments must be shown to the student.

   After the student logs into the system using a valid account, he selects the instructor, course, and the assignment and uploads files necessary for the submission. As seen in section 7, the scripts provided by the instructor are run against the files and the output of each phase and the instructor comments are shown to the student.

iii) The execution of the scripts on the submitted files must be carried out in a sandbox and the student must not be able to access the other directories in the TRY System.

   This was not accomplished in the current application. Refer section 9 for more details.

iv) Accept submission from the student only if the date of the submission is before the due date. The instructor must set the due date.

   Using the submission-details file, the instructor can set the due date for the assignment. If the student submits the assignment after the due date and no late submission is allowed, the submission is rejected and the student is shown appropriate message. If a late submission is
allowed, then the files are stored in a special directory called LATE inside the student’s submission directory.

9. Future Work

The future work of the application can be listed as follows:

i) Execute the scripts inside a sandbox. It can be achieved by running the web service as a root user. If the web service runs as a root user, it can change the root of the process by executing the chroot command. The web service can copy the files inside the scratch directory, change the root of the process and thus restrict the process from accessing any file above the Scratch directory.

ii) Allow instructors to use their personal account directory to save the scripts and the submitted files.

iii) Use the RIT credentials or the CS account credentials to authenticate the student. The current mechanism relies on the Google account provided by RIT. Hence, the current system would fail if the Google service is discontinued in the future.

iv) Deploy the application on the RIT web server or the CS department web server. The current server did not have the necessary packages and permissions for the installation and hence the application was deployed on a private cloud server.

v) Develop a module to facilitate the setup of an assignment. This will help the instructors to set up the assignment by automating the creation of required directories and files. This module will have an interface for the instructors to upload the necessary files and scripts. The system will create all the directories for the assignment and place the uploaded files in their respective directories.

10. Acknowledgements

I would like to thank Dr. James Heliotis for guiding me throughout the project and giving valuable insights.
11. References


