WEB-BASED MODEL SLICING
FOR 3D PRINTERS

BY

MICHAEL U.B. MEDING

B.S., UNIVERSITY OF MASSACHUSETTS LOWELL (2015)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF MASSACHUSETTS LOWELL

Author . . . .

April 29, 2016

Certified by. . . . .

Fred G. Martin

Professor

Thesis Supervisor

Certified by. . . . .

Jeff Brown

Associate Professor

Thesis Reader

Accepted by . . . .

Jie Wang

Department Chair

Web-Based Model Slicing

for 3D Printers

by

Michael U.B. Meding

Abstract of a thesis submitted to the faculty of the
Department of Computer Science
in partial fulfillment of the requirements
for the degree of
Master of Science
University of Massachusetts Lowell
2016

Thesis Supervisor: Fred G. Martin
Title: Professor

Thesis Reader: Jeff Brown
Title: Associate Professor

Abstract

In the field of 3D printing, there is currently a large gap between hardware and software. A hobbyist machine can now be purchased for less than $500, but optimal software can be difficult to find, and the only free options available are Cura and Repetier Host. Cura may be run on the Windows, Mac, and Linux operating systems, while Repetier Host is intended only for Windows. This leaves a considerable software gap, which could be filled by a web-based application unbound to a particular system. The purpose of this research was to construct an open source, web-based slicing software to facilitate the use of 3D printers, making 3D printing more approachable for users who have limited experience with technology. Additionally, this application creates opportunities for STEM educators to teach students about 3D printing in a simple and practical way, thereby beginning to bridge the gap between hardware and software.

Acknowledgments

The work in this thesis was only possible because of all the helpful and guiding members of the UMass Lowell Computer Science community. Many people have helped me not just in research effort, but in introducing me to new thoughts and ideas that became foundational to this project.

This research would not have been possible without the helpful guidance of Professor Fred Martin who's passion for education has been a constant source of inspiration. I would also like to thank Kelsea Gildawie for all the late nights spent helping me put my thoughts on paper. It would not have been possible to finish this research without her help.

I also wish to acknowledge my parents Uwe and Sally, without whom getting a masters degree would not have been possible in the first place.

Contents

4 Client Side

5 Server Side

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Over the past few years, 3D printing, has become immensely popular due in part to its reduction in cost and its easy availability. An enthusiast can now purchase a do it yourself (DIY) 3D printer kit for less than $500 however, one major pitfall that remains is the software support which leaves much to be desired. Most of the big name companies that used to hold all of the patents for 3D printers have retained the software patents but not the hardware patents. This creates a gap in knowledge between building the printer and running it. This research has served as an exploration of what software existed in the open source for 3D slicing and using that software to build a web based 3D print slicer. This will, effectively "democratize" the world of 3D printing, much in the same way that Google has democratized the way we search the web. In an article written by Harvard Business review, they theorize that this rise in the popularity of 3D printing will spur an industrial revolution as manufacturing becomes more personalized and decentralized (D'Aveni, 2015).

To print something on a 3D printer, there is a multi-step process that can be daunting to many first time users. The first step in the process is to either create or download a model. Creating a model can be achieved with any standard 3D CAD software, such as AutoCAD or SolidWorks. Downloading pre-existing models from an online repository can be done from websites such as Thingiverse or YouMagine. Once a model file is obtained, it is time to "slice" the model. Slicing is the act of

taking this model file and splitting it up into many thin layers that the 3D printer can understand. This process can only be executed by a dedicated slicing software which can often be complicated to use and difficult to install. Once the file has been sliced, the resulting file is a G-code file, which is simply a set of movement instructions that the printer head must follow. This file is then loaded to an SD card or sent via a print server similar to the way that a normal 2D printer is networked. Once the G-code file has been loaded all, that is left is to hit print either manually using the printers interface for the SD card or by hitting print on the network interface for the file that was uploaded.
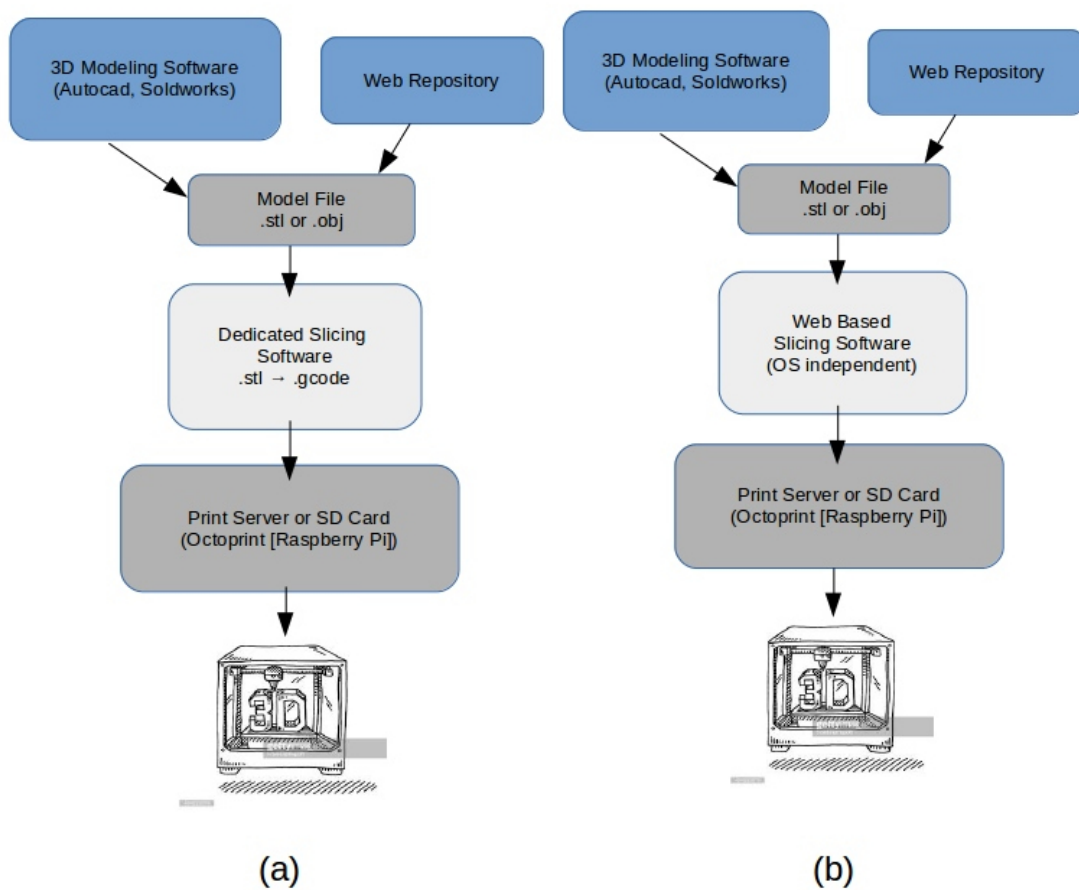
Figure 1-1: (a) High level view of the normal 3D printing process. (b) High level view of proposed new process using WebSlicer.

1.1 The RepRap Idea

The Replication Rapid-Prototyper Project (RepRap) is a movement with the goal of providing Open-source, diy 3D printers at low cost (Jones et al., 2011). RepRap printers are 3D printers with the additional ability to produce most of the parts necessary to assemble another identical printer. This idea also extends outside of hardware but to the software as well. Much of the software available for RepRap style printers are open source projects put together by the community. Unfortunately, community based software development is slow and leads to hardware that outpaces its software.

1.2 Sudden Growth of 3D Printing

In an article written by Forbes, the market trend for 3D printing was analyzed. They determined that 3D printing is becoming one of the fastest growing emerging markets, stating that "The worldwide 3D printing industry is now expected to grow from \$3.07 billion in revenue in 2013 to \$12.8 billion by 2018, and exceed \$21 billion in worldwide revenue by 2020" (Columbus, 2015). 3D printing may be reaching more people as a result of the availability of the RepRap designs. Additionally, building a 3D printer from a kit only requires basic hand tools and a moderate knowledge of electronics, which means that it has been opened up to a much broader audience.

1.3 Purpose of Research

The purpose of this research is to construct a web based slicer and simplify it for users who not well versed in 3D printing, so they are able to slice models and run their 3D printers. This will make 3D printing much more approachable for occasional users who are not prepared to spend thousands of dollars on a professional machine and expensive software. Additionally, this opens up opportunities for educators in STEM programs to teach students about 3D printing in a simple and practical way. Under normal circumstances, this would not be a feasible project for a year-long Masters

thesis; however, many of the technologies required to complete this project exist in varying states of completeness. Combining them into a cohesive application will be the subject of this study.

1.4   Research Objectives

This research encountered several milestones which had to be met for it to be considered complete. These milestones are listed in no particular order; however, several of them must be completed before allowing other milestones to be completed as detailed below. However, several of them must be completed before allowing other milestones to be completed as detailed below.

CuraEngine is an open-source slicing engine designed to take model files in .stl file format and convert them into G-code for 3D printing. For this project, wrapping this code and making it callable from the web is the heart of this application.

1.4.1   Design Intuitive Web Interface

This project requires both an intuitive and easy to way to slice a model file for 3D printing. This would be completed using Bootstrap and AngularJS because they are both scalable and flexible for almost any web design. These technologies also allow for small scale or even mobile use.

1.4.2   Run Beta Testing with Actual Users

No research would be complete without testing the software in question. Running a closed beta test of this software is a milestone that must be met as one of the major aspects of this application is its usability. The goals of this test were to see if WebSlicer is both easy to use and useful to those who decide to use it with their 3D printer.

1.5   Related Work

AstroPrint is an all-inclusive cloud operating system for 3D printers. It attempts to encompass the entire package of 3D printing to a dedicated Raspberry Pi, or similar computer system. It is also one of the only software platforms that currently exists that attempts to do web based slicing. Unfortunately, the AstroPrint software tries to accomplish too many tasks at once and has become somewhat like a Swiss army knife, being capable of many functions but only complete a few tasks well. Additionally, their cloud-based slicing software, while being reasonably fast, lacks any support for reviewing the sliced model. This review stage is critical for anyone who is printing something that will take more than a few hours to complete.

1.6   Thesis Map

- Chapter 2: Background and design choices.

  This chapter discusses many of the external resources used and why they are relevant to this research.

- Chapter 3: Software architecture review.

  Compiled here is a complete overview of the architecture of WebSlicer. It follows the user through the entire process of slicing while explaining briefly some of the details of what is really happening.

- Chapter 4: Client side in detail.

  Here is where all of the possible client interactions and application specific details are explained.

- Chapter 5: Server side in detail.

Details of exactly how a C++ executable is tied into a Java process are explained here as well as the exposed web API created for this research.

- Chapter 6: Discussion about usability testing and further improvements.

  Included in this chapter are the formal results from closed beta testing and a discussion of the printer which inspired this research.

Chapter 2

Libraries & Existing Code

This chapter discusses many of the external resources used and why they are relevant to this research. All of the resources discussed in this chapter are open source projects and libraries.

2.1 CuraEngine

CuraEngine is the server side of a larger application called Cura. Cura is an open source 3D print slicer designed by Ultimaker that is part of the software suite for their Ultimaker line of 3D printers (Kuipers, 2016). As it is the main portion of the server side of WebSlicer, Cura will be discussed extensively in latter portions of this paper.

The reason for choosing to use CuraEngine as the slicing engine is that it has the most clear separation between application and interface. Another option was to use Slicer, which is much older and has better documentation, but is unfortunately written on Windows and would require a large amount of extra effort to get working properly for my needs. CuraEngine is also platform agnostic, as it is written in C++ and uses one library called protobuf, which is its main interface library for the Cura application.

2.2 Client Side Libraries

When structuring a website for optimal layout and scalability, there are few better options than using the combination of Bootstrap and AngularJS. Bootstrap is a client side CSS library which includes most commonly used CSS options such as buttons and input fields and styles them all accordingly. AngularJS creates a client side environment to support higher level language constructs and features that are normally reserved for complex server side applications.

2.2.1 Bootstrap

Bootstrap is one of the most popular CSS and JS frameworks for developing responsive and mobile projects on the web (Mark Otto, 2016). Being responsive means that Bootstrap is capable of being dynamically displayed on screens of varying sizes. When the screen size changes, Bootstrap is capable of moving and resizing elements on the page without losing any content or overlapping items (Mango, 2013). In addition, it speeds up the process of developing web based applications, as it removes the need to write heavy amounts of CSS to make an application look and preform nicely. Bootstrap also includes many standard features which most web developers use every day, further simplifying the process of building a website.

2.2.2 AngularJS

AngularJS is a framework that allows for easy development of dynamic web pages (Brat Tech LLC, 2016). It contains many ways to logically separate code similar to popular object oriented programming languages like Java or C++. Freeman (2014) states, "the goal of AngularJS is to bring the tools and capabilities that have been available only for server-side development to the web client and, in doing so, make it easier to develop, test, and maintain rich and complex web applications" (p. 48). Therefore, the use of Bootstrap for the unified look, responsiveness, and mobile support combined with the power of AngularJS as a framework, make it the easy choice for a web based application such as WebSlicer.

2.3 JavaEE

JavaEE is a layer built on top of JavaSE the standard Java development environment (Gosling, 2016). This additional layer provides many important architectural interfaces, such as the ability to encapsulate code in EJB containers or web containers for deployment of large applications which may have many of such containers (Pilgrim, 2013). JavaEE also provides frameworks for working with RESTful web services which streamlines the creation of a useful application API.

There are many server side frameworks which have support for RESTful web services, but most lack real scalability like that provided in JavaEE. Furthermore, JavaEE provides the ability to easily distribute computing as more resources are needed. As this research may eventually scale to supporting many users, it was logical to choose a framework which allowed for this kind of growth. The use of JavaEE as a framework also provides the full platform cross compatibility which is standard in Java. Allowing for cross platform compatability would facilitate the process of linking many computers that are running on different operating systems together, further simplifying the prospect of scaling.

To run a JavaEE based application, it must run in a web capable application container. This container must be placed on an application server which exposes it to the web under some context. For WebSlicer, the application server WildFly was chosen for this task as it is well known for its reliability (Fleury, 2016).

2.4 OctoPrint

OctoPrint is a small server that connects a Raspberry Pi to a 3D printer and serves printer info and files to the printer remotely (Häußge, 2016). The Raspberry Pi serves a small webpage which allows the user to easily keep track of the temperature of the printer and the current print progress. This interface allows for a web based connection between your remote g-code files and the local printer (Horvath, 2014).

OctoPrint also contains its own API which makes for easy integration into other web applications.

The fact that OctoPrint is so easily integrated into other web based applications made it a logical choice for integration with WebSlicer. Integrating with OctoPrint would allow for one touch printing by allowing me to send g-code files directly to a connected OctoPrint server. The only interaction that is required of the user is to connect their OctoPrint server which is done by simply indicating the address at which the server can be reached at. Additionally, users are required to include an API key to allow for other applications to directly access the servers API.

2.5 G-Code Visualizer

When building the G-code visualizer, it seemed logical to find one which already existed and worked well. gCodeViewer is an open source project originally written by Nils Hitze (Hitze, 2015). This project was written in pure JavaScript and would be seemingly easy to integrate with WebSlicer.

Unfortunately, many of the libraries that were needed to make this code needed to be updated. Additionally, integrated into the existing AngularJS framework which was more challenging than anticipated. Thus, what ultimately remained from the existing open source project was very small.

Chapter 3

Methodology

This chapter discusses the architecture of WebSlicer. It also serves to justify the design choices that were made.

3.1 Research Design

This thesis is a mixture of both research and design implementation. The research portion of this project focused on linking an existing C++ application (CuraEngine) into a larger JavaEE based project. This research also included running a closed beta test of the software and logging the results.

3.2 Working Procedure

As shown in Figure 3-1, the application will have 3 major components that all need to work together in a cycle until the user decides that the output is what they desire.

3.2.1 Web Interface

The web interface includes a set of forms for collecting the user's settings for their printer and the settings as they relate to printing the model itself. This interface also includes a method so the user may retain the files they have uploaded for future use.

Web Based Slicer Detailed View
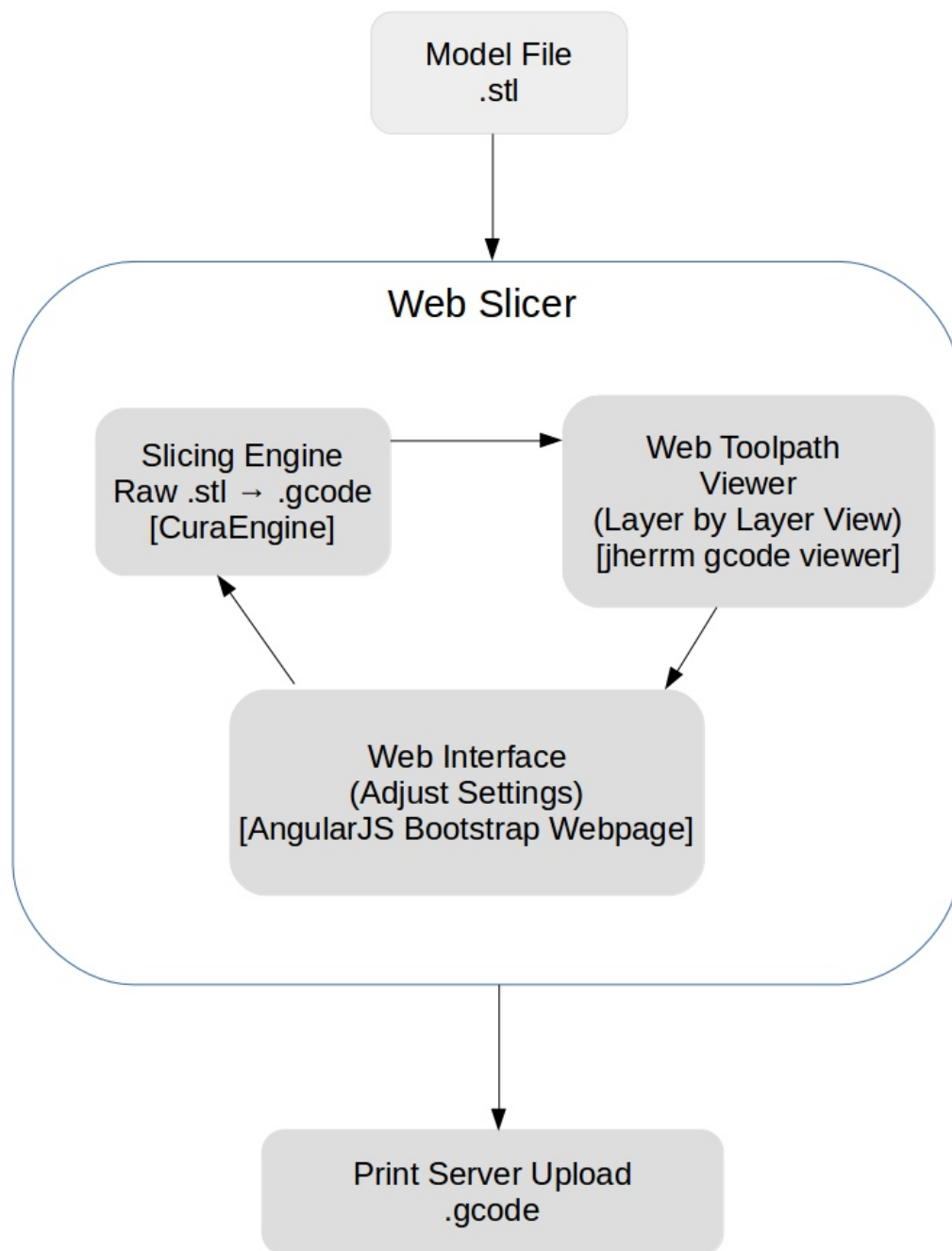
Figure 3-1: High level view of how WebSlicer functions and how users will interact with it

This does not include the actual slicing engine, which must be driven and accessed independently.

3.2.2 Slicing Engine

To comprehend the slicing engine it is vital to understand what G-code is. G-code consists of sequential machine instructions that command a 3D printer to move in a sequence of patters to build a three dimentional part (Thornton, 2012). Listing 3.1 shows a sample of G-code that results from slicing using CuraEngine. Comments in G-code are preceded by a semicolon and the first line denotes the type of G-code, we are only focusing on RepRap G-code in this case. Commands in G-code follow a standard pattern of a letter, either M or G, and a number which denotes the type of command. All subsequent items that are separated by spaces are arguments to the command.

Listing 3.1: A sample of G-code that results from slicing using CuraEngine.

```
1  {
2  FLAVOR:RepRap
3  M190 S110.000000
4  M104 S245.000000
5  M109 S245.000000
6  G21 ;metric values
7  G90 ;absolute positioning
8  M82 ;set extruder to absolute mode
9  M107 ;start with the fan off
10 G28 X0 Y0 ;move X/Y to min endstops
11 G28 Z0 ;move Z to min endstops
12 G1 Z15.0 F9000 ;move the platform down 15mm
13 G92 E0 ;zero the extruded length
14 G1 F200 E3 ;extrude 3mm of feed stock
15 G92 E0 ;zero the extruded length again
```

```
16  G1 F9000
17  ;Put printing message on LCD screen
18  M117 Printing...
19  ;Generated with Cura_SteamEngine master
20  ;LAYER_COUNT:158
21  ;LAYER:0
22  M107
23  G0 F7200.000000 X99.430 Y80.760 Z0.300
24  G0 X102.806 Y78.243
25  ;TYPE:SKIRT
26  G1 F1800.000 X103.337 Y78.033 E0.01074
27  G1 X103.441 Y77.996 E0.01282
28  G1 X103.842 Y77.875 E0.02070
29  G1 X104.142 Y77.800 E0.02651
30  }
```

The slicing engine rests at the core of this project as most of the computation time is spent preforming geometry calculations and converting them to G-code. This process of converting from model to G-code is known as slicing. An application which is encapsulated and preforms slicing is known as as a slicing engine. The engine which carries out the raw geometry calculations for WebSlicer is CuraEngine which is written in C++. Thus, this portion of the project required deploying the CuraEngine application on a remote server and creating a RESTful API to interface with it.

3.2.3   Web Tool Path Viewer

After configuring and generating the G-code representation for a 3D model, there must be a way to visually review how the slicing engine will divide the model. This is executed by loading the resulting G-code from the slicing engine into a web based visualizer. The user is then able to view each of the layers and the steps involved

in creating each one in detail. As displayed in Figure 3-1, this process of changing settings, slicing, and reviewing can occur an arbitrary number of times before the user decides they are satisfied with the result.

3.3 Review and Usability Testing

After preforming all steps of the working procedure, it was then necessary to test WebSlicer by running a small beta test. This beta test consisted of a small group of users with varying familiarity with 3D printing to test how simple WebSlicer is to use through a series of simple tasks. These simple tasks ranged from uploading a file to finding and adjusting the correct setting for a printer.

Chapter 4

Client Side

This chapter discusses the client side of WebSlicer in detail. It also includes some necessary background about AngularJS as it will be crucial to understanding latter sections of this chapter.

4.1 AngularJS

In order to understand the structure of WebSlicer a few things must be known about how AngularJS applications are structured and how AngularJS itself works.

4.1.1 Controllers & Data Binding

The controller structure in AngularJS that sits between the view and the JavaScript world acts like the glue between the two. From a controller the user is allowed access to "scope" variables, which are similar to normal JavaScript variables, but they have a special two-way data binding property. Two-way data binding is one of the most interesting features of AngularJS, as it gives the user real time updates as the user interacts with a view. In traditional JavaScript, a developer must identify key events that occur in the browser as the trigger for a sequence of events. AngularJS also has a sequence of events but no trigger is required as the sequence of events is triggered automatically. This action is known as a digest cycle. AngularJS has a watch function

for each variable attached to its scope. When any changes to this variable occur, they propagate that change to the functions that are associated with that variable (Freeman, 2014).

4.1.2   Factories & Services

JavaScript has many pitfalls, including the lack of any kind of larger design pattern support, such as Object Oriented Programming (OOP). The current trend with web based applications is to create single page applications with the same functionality as prior designs that have many pages. AngularJS is the answer to this with Factories and Services which mimic the design of a POJO (Plain Old Java Object) and Singleton objects in Java.

4.1.3   Directives

A directive is one of the most powerful structures in AngularJS. It allows the programmer the power to write their own HTML5 tag with its own parameters and rules. Directives also contain support for templates which are HTML snippets that replace the main tag when the code is loaded. Combining this with the use of the aforementioned design patterns creates a very powerful tool to create and organize dynamic content.

4.2   WebSlicer AngularJS Structure

This section discusses the entire client side structure of WebSlicer when it is broken down into its respective Controllers, Factories, and Directives.

4.2.1   app.js

The full graphical AngularJS structure of WebSlicer is shown in Figure 4-1. Flow through this diagram starts with the app.js node which represents the main controller of the application. This can be thought of as a main function in C++. The main

WebSlicer Angular Structure
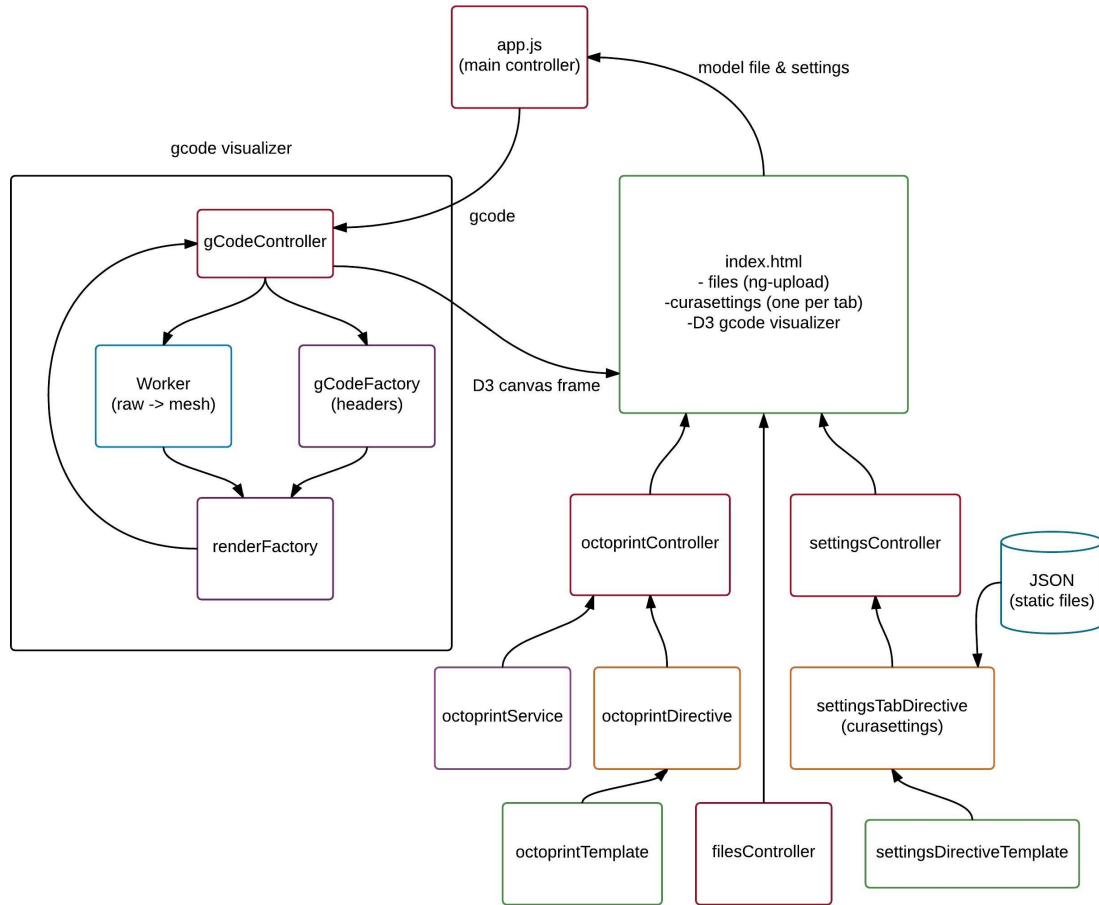
Figure 4-1: Full AngularJS structure breakdown

control variables are also inside of app.js which are similar to global variables. Variables in this controller are used to store the current settings, file pointer, and output G-code.

4.2.2 index.html

Figure 4-1 also shows that index.html is large hub and, as WebSlicer is a single page web application, this is the only static HTML file. Index.html has several other functions, such as bringing in all libraries and custom directives.

4.2.3 Settings

As shown in Figure 4-2, settings have a long path that they must travel before they are submitted to be used while slicing. This data flow starts with loading a static JavaScript Object Notation (JSON) file, which describes the settings in a pattern as shown in Listing 4.1.

Listing 4.1: A sample from a static settings file in JSON format.

```
1  {
2      "setting": "layer_height",
3      "default": 0.1,
4      "type": "float",
5      "category": "Quality",
6      "label": "Layer Height (mm)",
7      "description": "Layer height in millimeters. This is
            the most important setting to determine the
            quality of your print. Normal quality prints are
            0.1mm, high quality is 0.06mm. You can go up to 0
            .25mm."
8  }
```

Listing 4.2: An example of a ng-repeat looping construct in HTML5.

```
1  <table class="table">
2    <tr><td>Action</td><td>Done</td></tr>
3    <tr ng-repeat="item in todos">
4      <td>{{item.action}}</td>
5      <td>{{item.done}}</td>
6    </tr>
7  </table>
```

A directive called curasettings takes this static JSON file and divides it so an input field exists in the template for each setting object. AngularJS provides this

functionality through the use of an ng-repeat, which is written in a similar fashion to that of a for-loop in Python which is shown in Listing 4.2. Using a template for a directive in this case also means that the user can have control over many different fields, such as drop-downs and number-specific inputs. The curasettings directive also gave the logical separation of one static JSON file per tab of settings in the UI, which made it simple to find and modify the settings as needed.

Settings in WebSlicer have a complex method of being tracked after loading to the client. In most applications such as this, there would be an individual watch on each field or a submit button which would trigger grabbing all the fields. WebSlicer, however, uses a single object to track all of the settings for the application and uses a dynamic method of AngularJS to map from the input field to a field of a single object. Thus, when the settings are submitted, there is no more interpolation needed, as the settings object already has the current state as shown in the Figure 4-2 as "Pull Current Settings."

4.2.4   G-code Visualizer

The G-code visualizer for WebSlicer is written using a combination of D3 and a JavaScript web worker. From Figure 4-1, it can be seen that the controller for the visualizer is sent in a G-code file, which it splits up to two separate services. The controller completes an initial parse of the file, which places each one of the lines into its own entry in an array before exchanging it with gCodeFactory and Worker. Both of these files parse through this entire array, but do so at roughly the same time to expedite the process of visualizing. The worker takes the array of lines and ignores the header to just focus on converting the raw movement commands into D3 lines so that they may be rendered. The gCodeFactory takes the headers from the array and uses them to do analytics of the G-code file, such as total print time.

The final steps in the process of visualizing the G-code require splitting the G-code file into layers for rendering. This task which is made simple by following the given

tags in the G-code which signal a layer change. Once all of the cumbersome tasks of parsing and splitting up the G-code file are finished, it is simply a matter of returning layer requests with canvas frames. Each time a layer is requested, a progress for that layer is also sent. The controller then indexes to the layer height in the array and renders a frame with the number of lines that are described by the current progress. As array indexing is nearly instantaneous, the visualizer once parsed and loaded runs very quickly to display layers.

4.3   Key Challenges

Discussed in this section were the most notable challenges that had to be overcome when developing the client side of WebSlicer.

4.3.1   Visualizer Integration

The visualizer starter code for this was originally written by Nils Hitze as an open source project which had many other features (Hitze, 2015). This code, however, required a lot of work to integrate properly with the rest of the application. AngularJS, despite its many features does not mix well with other projects. Ultimately, the only code that remained from the original was the JavaScript web worker and some of the parser code.

4.3.2   Interpolating Settings

A method was required to be designed for the application to handle a lot of input fields. A simpler method would have been to create a series of fields, each with their own variable, and submit methods and triggers; however, if settings ever change format it would require refactoring many files. Thus, spending the extra time to design an intelligent method of handing large amounts of input data seemed logical. This, however, took much more time and effort than anticipated. At one point, this even required contacting the original developers of CuraEngine discuss the meaning of

some of their settings. Documentation for many of the settings was dissatisfactory and in many instances was non existent which further slowed down development.

4.4 Other Planned Integrations

During the development process often many items do not make it into the final product for various reasons. Included in this section are some items which were cut from WebSlicer before beta testing.

4.4.1 OctoPrint

A feature that was removed at a late state in the process of building this application was an integration with OctoPrint. OctoPrint has a exceptional API that allows for external applications to integrate easily, making it an ideal choice for this application. This integration was to allow a user who was running an OctoPrint server to be able to send files directly to their server. This would eliminate the need to download the G-code from WebSlicer only to be uploaded to the print server seconds later. It was decided at the last moment that this feature need not be in the minimum viable product and that time was best allocated to finishing more crucial features of the application.

4.4.2 Thingiverse & YouMagine

Another planned integration was the ability to import from a web based repository such as Thingiverse or YouMagine. These repositories are public sites where users can upload their 3D designs so others can 3D print them. Thingiverse in particular has a effective API for accessing models from their site, which would make it an easy integration for a web based slicing software; however, this feature was terminated early on, as it would have required too much unnecessary development time to finish.

4.5   Issues & Known Bugs

As mentioned in prior sections, WebSlicer was designed with a minimum viable product in mind. Developing a working 3D print slicer for the web was the primary task and all other features needed to support this or extend this functionality. For this reason, there is no login or user database, which would normally be the first item to be developed for an application such as this. There is also no way to view any of the models in 3D which, for most users, makes the software significantly harder to use.

Settings Data Flow
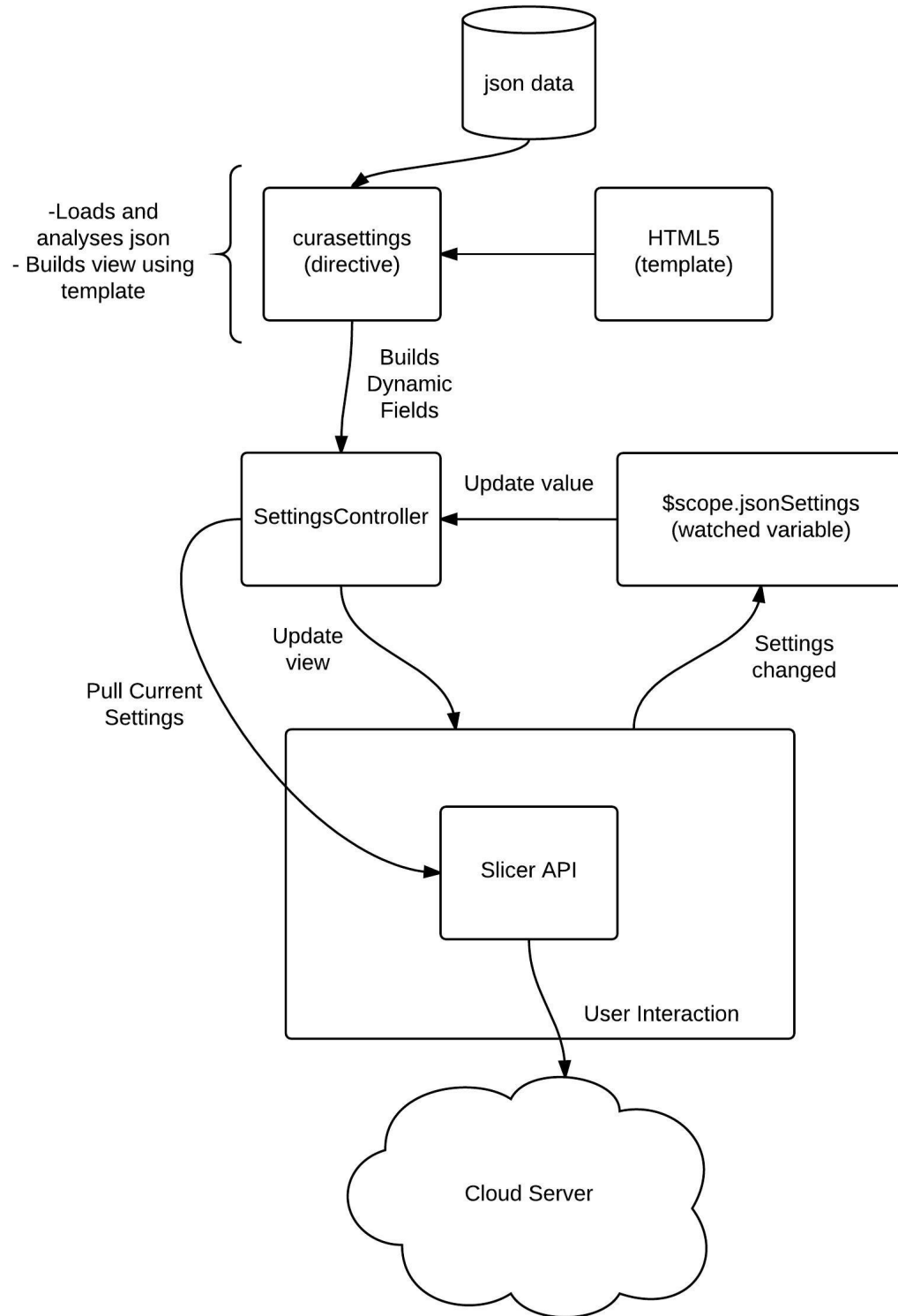
Figure 4-2: The flow of data through the application from beginning to end of client side user interaction

Chapter 5

Server Side

This chapter discusses the server side of WebSlicer in detail. Included with this is some necessary background about JavaEE and some of its functions.

5.1 JavaEE Structure

The server side of WebSlicer was written in JavaEE, the structure for which is shown in Figure 5-1. JavaEE was the optimal choice for this application, as it allowed for the easiest deployment and was also the easiest to scale (Pilgrim, 2013). Additionally, JavaEE has a excellent code packaging mechanism for web and non web based applications alike. The web container which is in use for this application exposes a RESTful API on a privately hosted server.

To further simplify the development process, Maven was also used. Maven is a build tool for Java and has support for deploying complex applications such as those in JavaEE (Vincent Massol, 2005). Thus, when a build was completed, it was automatically deployed and ready for testing.

WebSlicer JavaEE Structure
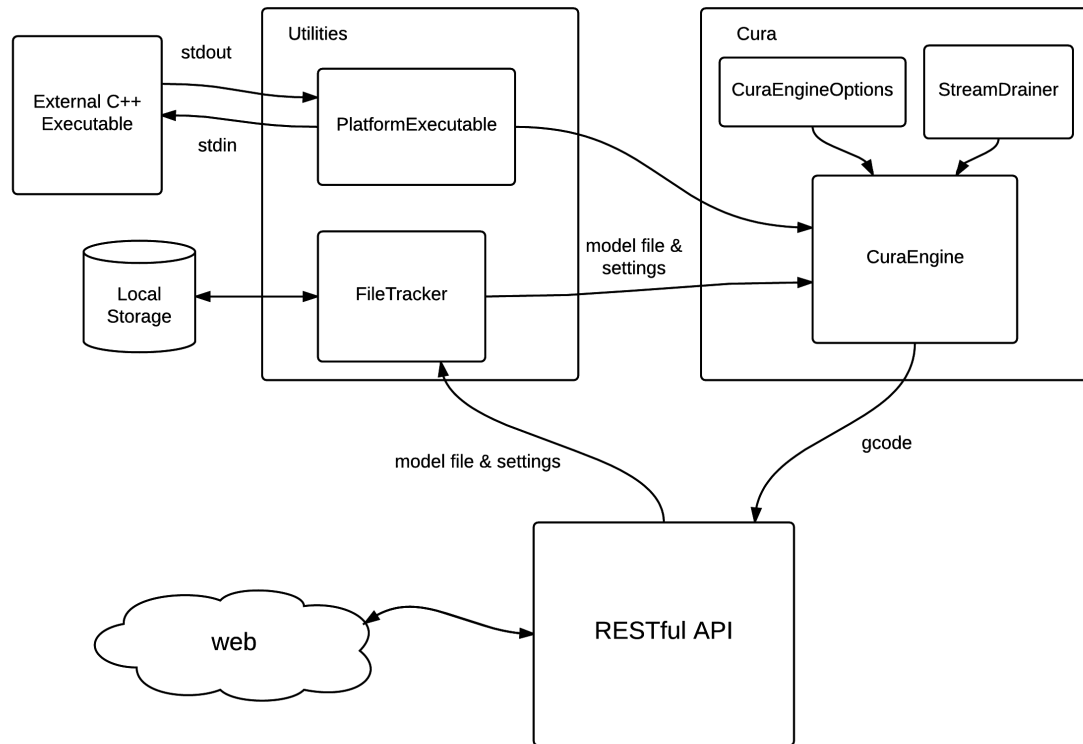
Figure 5-1: The structure of the server side of WebSlicer

5.2   ProcessBuilder

At the core of the server side application is an executable called CuraEngine. It is the main executable which is compiled from the open source slicing platform Cura which is written in C++. This presented a problem, as all of the server side code in this application is written in Java. ProcessBuilder was the solution to this problem as it is capable of redirecting the input and output streams of a local executable process into the Java server application. CuraEngine from Figure 5-1 uses a ProcessBuilder and the PlatformExecutable to create a runnable Java method that is capable of executing like a C++ executable. CuraOptions feeds the CuraEngine class with all of the parameters that it needs from the API. It gathers the path to the appropriate settings file and includes all of the parameters needed to run the CuraEngine executable.

Listing 5.1: An example of running CuraEngine C++ executable directly from the command line.

```
1 CuraEngine slice -v -j {settings.json} -g -e -o {output.
      gcode} -l {model-file.stl}
```

An example of running the CuraEngine C++ executable from the command line is shown in Listing 5.1. When the ProcessBuilder class of WebSlicer receives a slice command from the API it gathers the arguments listed in brackets and sends them to PlatformExecutable. PlatformExecutable then spawns a native process and pipes its input and output streams into the respective Java streams. In the meantime, StreamDrainer spawns a new thread and waits for the output stream that was created by PlatformExecutable. StreamDrainer's task is to take the unneeded output from stdout and pipe it into a log file for debugging.

After CuraEngine has finished slicing the current file and PlatformExecutable has returned the REST API, which has been waiting, it unblocks and starts reading the output gcode file. This file is then packaged and sent back to the client as the response of the "/slice/{clientId}/{modelId}" command as shown in Table 5.1.

5.3   REST API

The goal of the client is to satisfy all the requirements needed to run the "/slice" command from Table 5.1. A typical call pattern to the API in begins with creating a user if one does not already exist. Creating a new user requires calling "/setupClient" and recording the returned unique user identifier. Using the recorded unique identifier the client must then upload a model file and settings file to the corresponding "/importStl" and "/importSettings" functions. This returns unique identifier strings for all settings and model files that the client uploads to the server. Finally, to run a slice, the client takes all acquired unique identifiers and places them into the call to "/slice" as shown in Table 5.1. Upon successful completion of all aforementioned calls the user will be returned a G-code file in raw textual form.

Type	Address	Description
GET	/ping	A simple ping endpoint used for testing.
POST	/setupClient	Sets aside all needed files for a new client and return its unique ID.
POST	/importStl/{clientId}	Takes a MIME type file stream and imports the file to the clientId specified in the URL. It also returns a unique identifier for the file.
POST	/importSettings/{clientId}	Similar to importStl this endpoint takes a settings JSON file and imports it to the specified clientId
POST	/slice/{clientId}/{modelId}	This is the main slice function of the API. It combines all of the parameters specified by the calls before and returns a gcode file to the user.
POST	/testSlice	A test endpoint that requires no parameters and simply returns some arbitrary gcode to the user.
GET	/getFiles/{clientId}	Returns all the model file names and their tracking ID's that are associated with a clientId.

Table 5.1: Documentation of all exposed endpoints of my RESTful API

5.4 Key Challenges

5.4.1 ProcessBuilder Deadlock

One considerable bug encountered while developing this project was a thread deadlock issue. The server side code uses Java's ProcessBuilder, which builds a system native call to an executable and then pipes the input and output into the corresponding pipes of Java's stdio as shown in Figure 5-2. This is suitable for small platform executables with limited I/O, but can become problematic when complex native calls such as CuraEngine are used.

ProcessBuilder executes its normal writes to stdout and the drainer pipes them into a file; however, the drainer must wait for a file pointer using the fp.available() function. This is a non-blocking function that only estimates the buffer size that it has for the file. The check for file pointer availability was determining whether

WebSlicer Deadlock
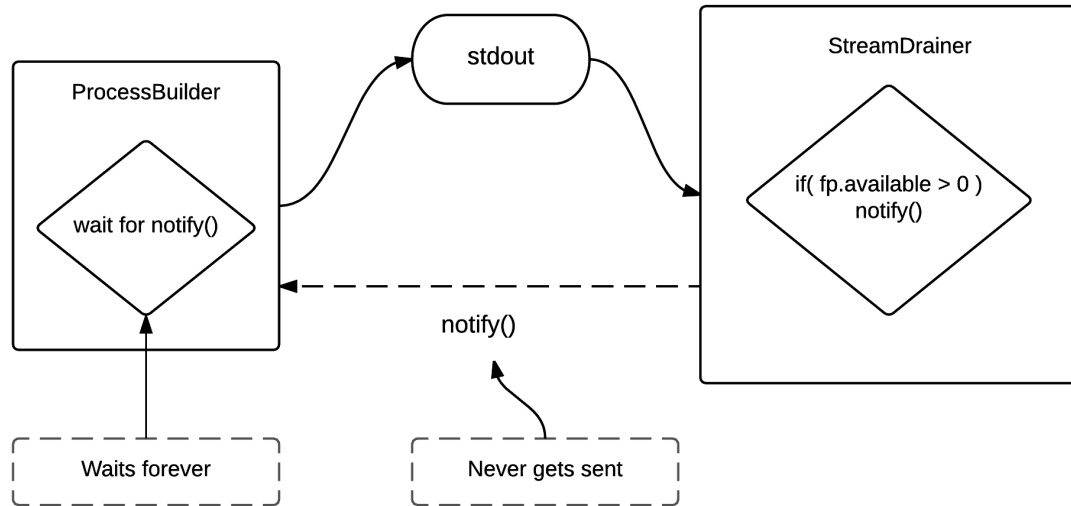
Figure 5-2: Diagram of a deadlock issue that took weeks to resolve

this function returned something greater than 0 as an estimate before notifying the ProcessBuilder that it was ready; however, the buffer size would often start as zero before allocation and, as this check was not part of a loop, it would stay stuck forever as the notify was missed.

This problem was solved by using the correct blocking file pointer available check. Occasionally, the buffer size was larger than 0 and the application ran suitably but, with some models, it would consistently fail, as the buffer had not been allocated yet. This solution is seemingly obvious yet it took many days to find and correct because the application did not fail consistently.

5.4.2   FileTracker Revamp

The first iteration of FileTracker was crude and not well planned out. It tracked two hash maps: one for model files and the other for settings files with no mind for the client who required access to those files. This worked for testing, but experienced

many pitfalls including the inability to reuse files that already existed. As soon as the client closed their session, those files were lost, which is a major inefficiency.

The fdmprinter.json file within the unique client folder is symbolically linked to the fdmprinter.json file within the common folder. The CuraEngine executable requires that all of the settings files rest within the same directory when performing a slice as shown in Listing 5.2. Unfortunately, this leaves the potential result of having this file copied for many clients. Thus, symbolically linking the file to the rescue.

The output.gcode and settings.json files are dynamically overwritten for every iteration so their existence here is only for the sake of running CuraEngine through file arguments. The user has no access to these files and is only able to obtain their content through the web interface, which parses in and out of files.

Listing 5.2: WebSlicer's underlying file structure supported by FileTracker.

```
1  webslicer/
2  - b1a2a69e-5893-4d7c-aa1f-d639fa3b4ed1/
3    - fdmprinter.json -> /tmp/webslicer/common/fdmprinter.
       json
4    - models/
5       - balanced_die_version_2.stl
6       - raldrich_planetary.stl
7    - output.gcode
8    - settings.json
9  - common/
10   - fdmprinter.json
11   - presets/
12      - prusa_i3.json
13      - ultimaker2.json
```

5.5   Future Improvements

Currently, FileTracker does not take advantage of the presets within the common/presets/ folder as described by Listing 5.2. These files contain the default settings for the corresponding printer, which are the ultimaker2 and a basic configuration of a prusa i3 variant. Optimally, the user would select from one of these starting presets

and then modify and save their own. This would allow users an optimal starting point while lowering the amount of starting knowledge and increasing the usability of WebSlicer.

This new file structure also allowed for an easy client index. In the future, the unique folder ID will become the client's identification number, which will be attached to their login. Additionally, simplifying the login process with Google's OAuth 2.0 system was also planned.

5.6   Issues & Known Bugs

Currently there is no way for the server to import existing user files into its structure. Thus, when the server is restarted for any reason, the supporting file structure with all user files is lost. Resolving this is just a matter of writing an initial import function that indexes all of the existing files. It was removed from the initial version due to time constraints.

Chapter 6

Discussion

This chapter summarizes the experience of running usability testing of WebSlicer and the construction of a RepRap 3D printer that was used during software trials.

6.1   Usability Testing

Good design goes through many design iterations before it is considered production ready. Between iterations feedback is given through usability testing. Unfortunately, this research had a short time line thus, only one cycle of usability testing was possible.

6.1.1   Task Description

WebSlicer was evaluated by giving research participants a series of tasks to preform in a controlled environment. This environment consisted of a lab computer with a small collection of model files and a web browser. The tasks that were given to participants are formalized in Table 6.1. Participants were also asked to fill out a questionnaire when they had completed the study. This questionnaire asked participants to rate the level of difficulty of each task from 1 to 5 and to include comments of what they thought of each task. Results of the average difficulty reported by users is shown in Table 6.1. As an incentive to participate, participants were given a small 3D printed figure of their choosing.

Task #	Task Name	Average Difficulty	Description
1	Create a new user	4.66 out of 5	Participants were simply asked to click a button which generated a unique ID for them.
2	Upload your file	5 out of 5	Uploading a model file of the participants choosing from a given USB drive.
3	Adjust slicer settings	4.33 out of 5	Participants were presented with a table of settings and their value. They were instructed to find these settings and change them while ignoring all other settings.
4	Analyze and download G-code	4.66 out of 5	This task simply required the user to toggle the visualizer to see the output of their actions in prior tasks.

Table 6.1: The tasks given to participants of the usability study. The participants were asked to complete the tasks in numerical order.

6.1.2 Results

The expected result of usability testing was to expose some minor flaws of the user interface (UI) design; however, the results showed that more than minor changes would be required before the next iteration of WebSlicer. G-code output in textual form exposed to the user was found to be a bug in certain cases during usability testing. The output when slicing certain models was so large that the browser crashed when trying to load the text to the screen. This bug could be resolved by filtering the amount of text displayed at one time to the user.

Another flaw that was exposed during testing was the functionality of the G-code visualizer. Most participants found that both the controls, and explanation of what it was meant to accomplish was lacking. Concluded from this is that more textual explanations and user guidance was required in further iterations of WebSlicer.

6.2   Printer Build

6.3   Future of WebSlicer

The goal of future versions of WebSlicer are to lay the groundwork for a larger manufacturing software tool. This manufacturing software will link together many 3D printers and allow for unified control from a web based hub. Unified control such as this would allow for users to preform small scale manufacturing which would fill a void that currently exists in the manufacture of plastic parts.

References

Brat Tech LLC, G. e. a. (2016). *AngularJS*. Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043.

Columbus, L. (2015). 2015 roundup of 3d printing market forecasts and estimates. *Forbes*.

D'Aveni, R. (2015). The 3-d printing revolution. *Harvard Business Review*.

Fleury, M. (2016). *Wildfly Application Server*. Red Hat.

Freeman, A. (2014). *Pro AngularJS*. Apress.

Gosling, J. (2016). *Java Enterprise Edition*. Sun Microsystems, Oracle.

Hitze, N. (2015). gcodeviewer. https://github.com/hudbrog/gCodeViewer.

Horvath, J. (2014). *Mastering 3D Printing*. Apress.

Häußge, G. (2016). *OctoPrint*. Impressum, Goethestraße 5 63179 Obertshausen Deutschland.

Jones, R., Haufe, P., Sells, E., Iravani, P., Olliver, V., Palmer, C., and Bowyer, A. (2011). Reprap – the replicating rapid prototyper. *Robotica*, 29:177–191.

Kuipers, T. (2016). *CuraEngine*. Ultimaker.

Mango, A. (2013). *Mobile First Bootstrap*. Packt Publishing.

Mark Otto, J. T. (2016). *Bootstrap*. Twitter.

Pilgrim, P. A. (2013). *Java EE 7 Developer Handbook*. Packt Publishing.

Thornton, J. (2012). Linuxcnc g-code. http://gnipsel.com/linuxcnc/g-code/gen01.html.

Vincent Massol, T. O. (2005). *Maven: A Developer's Notebook*. O'Reilly Media.

Appendix A

IRB Compliance Documents

Included in this appendix are all the IRB compliance documents that were requited for preforming usability testing.

A.1   IRB Exempt Form

The following pages are the IRB Exempt Form which contains the offical summery of the research to be undertaken.

IRB <u>EXEMPT</u> STATUS APPLICATION

IRB No.: 16-061-MAR-EXM   Rev. No./Date: 2/4-11-16

Submit all documents to <u>IRB@uml.edu</u>   Date Submitted to IRB: 3/29/16

(This form is ONLY for minimal risk research where no identifiers are collected.)

A. GENERAL INFORMATION

Project Title: WebSlicer Usability Testing	
PI: Fred Martin	Email:fredm@cs.uml.edu
Department: Computer Science	Work Address (Bldg and No.): Olsen Hall 306
Phone: 978-934-1964	Emergency Phone: 978-934-2705
Co-PI(s):	Co-PI(s) Contact Info:
Student Researcher: Michael Meding	Student Researcher Contact info: 214-334-1905

1. **Sponsor Information- Check One** (Double left click on each check box to access tool.)
(x) Not funded.
( ) Internal funding. Type:
( ) Government/Federal funding. List agency name:
( ) Subcontract. List organization name and include contact name, telephone no., and address:
( ) Other. List organization/company name and include contact name, telephone no., and
address:

The proposal to the funding source noted has also been submitted to the IRB: ( )Yes   ( )No

2. Project Personnel: Include the PI and all personnel who may interact with subjects or access
identifiable human subject data. Training certification should be submitted with the application.

Name and Title(Check one)	Email Address	Training Completed
Fred Martin (x)Faculty ( )Staff ( )Student	fredm@cs.uml.edu	(x) CITI Basic, Date: 06/27/2014 ( ) NIH, Date:
Michael Meding ( )Faculty ( )Staff (x)Student	mike@mikemeding.com	(x) CITI Basic, Date: 03/01/2016 ( ) NIH, Date:
( )Faculty ( )Staff ( )Student		( ) CITI Basic, Date: ( ) NIH, Date:
( )Faculty ( )Staff ( )Student		( ) CITI Basic, Date: ( ) NIH, Date:
( )Faculty ( )Staff ( )Student		( ) CITI Basic, Date: ( ) NIH, Date:
( )Faculty ( )Staff ( )Student		( ) CITI Basic, Date: ( ) NIH, Date:

41e16d377dc500ffbda8e27ba2e36b55

		( ) CITI Basic, Date:
( )Faculty ( )Staff ( )Student		( ) NIH, Date:

Additional personnel or other information:

a. Is this a student research project?   (x)Yes ( ) No
   If yes, (x) Graduate or ( ) Undergraduate, please specify below:
   ( ) Dissertation (x) Thesis ( ) Directed Study ( ) Class Project ( ) Other:

B. SCREENING QUESTIONS

   1. Conflict of Interest Disclosure:
   a.   Do you or any family members have a financial interest in this research activity (such as an equity position or outside consulting arrangement with the company whose drug, procedure, device or product is used or tested in this study)?   ( )Yes (x) No
   If **yes**, explain the nature of the relationship and the conflict(s):

   b.   Do other faculty or staff involved with this research have a financial interest in this research activity?
   ( )Yes (x) No
   If **yes**, indicate the nature of the relationship and the conflict(s):

   c. To your knowledge, does the University have a financial interest in the company whose drug, procedure, device or product is used or tested in this study (such as patent rights, equity)?   ( )Yes (x)No
   If **yes**, indicate the nature of the relationship and the conflict(s):

   2. Will the research expose participants to discomfort or distress beyond that normally encountered in daily life?
   ( )Yes (x)No

   3. Could disclosure of participants' responses outside the research reasonably place them at risk of criminal or civil liability or be damaging to their financial standing, employability, or reputation?   ( ) Yes (x)No

   4. Does any part of the research require deception or incomplete disclosure of information to participants?
   ( )Yes (x)No

   5. Will prisoners (or their data and/or specimens) be participants in the research?   ( )Yes (x)No

   6. For research proposed under categories 1-5, is the research subject to FDA regulations?   ( )Yes (x5)No

   Note: a **YES** for **questions 2-6** above indicates your research does NOT meet exempt criteria. Submit an Application for Expedited or Full Review.

C. EXEMPT CATEGORY CLAIMED (check all that apply):
( ) **1.** Research conducted in established or commonly accepted educational settings, involving normal educational practices, such as research on regular and special education instructional strategies, or research

41e16d377dc500ffbda8e27ba2e36b55

on the effectiveness of or the comparison among instructional techniques, curricula, or classroom management methods.

If you checked this category, to do the research under this exempt category, it must be conducted in commonly accepted educational settings and not deviate from normal educational practices.
Is this true? ( )Yes ( )No   If no, please submit an application for expedited or full review.

(X) 2. Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures, or observation of public behavior, **unless**: Information obtained is recorded in such a manner that human subjects can be identified, directly or through identifiers linked to the subjects; and any disclosure of the human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation.

If you checked this category, the activity may **not** involve any interactions of the researcher with children, if they are participants. Does it involve interactions with children?   ( )Yes (X)No
If yes, submit an application for expedited or full review.

( ) 3. Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures, or observation of public behavior that is not exempt under paragraph (2) of this section, if: the human subjects are elected or appointed public officials, or candidates for public office; or federal statute(s) require(s) without exception that the confidentiality of the personally identifiable information will be maintained throughout the research and thereafter.

( ) **4.** Research involving the collection or study of existing data, documents, records, pathological specimens, or diagnostic specimens, if these sources are publicly available or if the information is recorded by the investigator in such a manner that subjects cannot be identified, directly or through identifiers linked to the subjects.

For research under this category, will any of the data, documents, records, or biological specimens be collected or created **after** the date of this application?   ( )Yes ( )No
If yes, submit an application for expedited or full review.

For research under this category, will any of the information obtained from private sources of data, documents, records, or biological specimens be recorded by the investigator in such a manner that subjects could be identified directly or through identifiers linked to the subjects? ( )Yes ( )No
If yes, submit an application for expedited or full review.

( ) **5**. Research and demonstration projects conducted by or subject to the approval of department or agency heads, and which are designed to study, evaluate, or otherwise examine: public benefit or service programs; procedures for obtaining benefits or services under those programs; possible changes in or alternatives to those programs or procedures; or possible changes in methods or levels of payment for benefits or services under those programs.

( ) **6**. Taste and food quality evaluation and consumer acceptance studies, if wholesome foods without additives are consumed or if a food is consumed that contains a food ingredient at or below the level and for a use found to be safe, or agricultural chemical or environmental contaminant at or below the level found to be safe, by the Food and Drug Administration or approved by the Environmental Protection Agency or the Food Safety and Inspection Service of the U.S. Department of Agriculture.

D. **RESEARCH ACTIVITIES-Check all that apply:**

( )Internet or email data collection	(x)Observation of participants
( )Existing data, publicly available	( )Record review
( )Existing data, NOT publicly available	( )Research using existing specimens
( )Focus groups	(x)Surveys or questionnaires
( )Audio recordings	( )Interviews
( )Other:	

41e16d377dc500ffbda8e27ba2e36b55

E. **RESEARCH SUMMARY**

1.   Describe the research purpose and objectives: In this research study, we are evaluating a web-based 3D print slicer. Participants will choose a small model or figure from a set of pre selected models and follow the steps provided in the "Task Instructions" document to take that model from design to printable file. The objective of this study is to gain insight into the usability of this software so that it may be evaluated and improved as part of the thesis study.

2.   Describe the research methods: Participants will be recruited via email. Upon response from recruitment email participants will choose a time to join us in our lab Olsen 306, From there, participants will be asked to review the consent form (which will be provided as a web link) and then indicate that they have read the form and consent to the study in an online survey form. If they indicate they have not read the consent form, or decline consent, the session will end there. Sessons will last between 20 to 30 minutes depending on the amount of prior knowledge that the participant has of 3D printing and its surrounding technologies. Prior knowledge is not required to participate in this study. Upon consent, participants will be asked to complete a pre-activity survey on a computer in our lab Olsen 306. Then participants will be provided a printed version of the Task Instructions, and asked to use the web-based software to perform the set of tasks described therein. While each participant is completing the tasks, the researcher will record brief observations on a observations form (attached). After the tasks are complete, participants will be asked to take at post-study survey (which is provided as a continuation of the pre-survey). During this time, they can watch the 3D printer output a model. At the conclusion of the study, participants will receive a previously-printed model as a token of appreciation for their participation. The researcher will match surveys and observation notes by numbering the first participant "1", the second "2", and so on, on the observation form. No personally identifiable data will be collected during this study.

3. Describe the participant population: Students and faculty at UMass Lowell whom have little to no knowledge of 3D printing or its surrounding technologies.

4. Recruitment Information
a. Describe how the participants will be recruited:
Students will be recruited via email from the Computer Science email message board. The recruitment email will be sent from the PI's email and responses will be sent to the student researcher's email as included in the instructions in the email.

b. Indicate the anticipated number of participants: 6 to 8
c. Will any participants be under 18 years of age?   ( )Yes (x)No

If yes, justify and describe how you will meet the exemption requirements:

5. Estimate the duration of the study: 1 Month

6. Will all of the research activities be conducted at UMass Lowell?   (x)Yes ( )No

a. If no, list the site/collaborator(s):
b. A letter(s) has been submitted to the IRB from the collaborator to document how they intend to   support the research.   ( ) Yes   ( )No

7. Describe how participants will provide consent: Participants will be asked to read a consent form and then asked to click yes or no on the survey page.

41e16d377dc500ffbda8e27ba2e36b55

8. Does research involve the use of publicly available or currently existing data?　　( )Yes (x)No

    a. If yes, list source of the data or specimens:

    b. Indicate whether the data is currently de-identified or how it will be de-identified:

9. Describe any potential risk to participants from participating in the research: There is no more than minimal risk to participants since no identifying information will be collected. Participants who experience discomfort from using a computer will be advised in the consent form to refrain from particpating.

10. Indicate how you intend to minimize any risks to participants: Participants will be informed that they may leave the study at any time should they feel uncomfortable and no identifying information will be collected.

11. Describe procedures to protect participants' privacy and confidentiality: Personally identifiable data will not be collected.

12. Describe the potential benefits from the research: Design improvements of the software being researched.

13. Check all of the supporting materials submitted with this application:

    (x) Questionnaires, surveys

    ( ) Standard Research Tools (published testing materials, etc.)

    (x) Recruitment Materials

    (x) Consent Documents

    (x) Other, list: Observation rubric to be used by researcher while participant completes tasks; Task Instructions document.

F. PRINCIPAL INVESTIGATOR ASSURANCE AND SIGNATURE

(X) I understand that, as the PI, I am ultimately responsibility for the protection of the rights and welfare of human participants and the ethical conduct of research under this protocol. I agree to conduct the study in accordance with the approved protocol and ensure that all personnel involved in the research will do the same.

(X) I agree to follow the UMass Lowell IRB Policies and Procedures.

(X) I certify that the information provided in this application is complete and correct, and believe that my project qualifies as Exempt from the Federal Regulations.

(X) I agree to personally conduct or supervise the described investigation(s).

(X) I agree to maintain copies of all questionnaires, survey instruments, interview questions, data collection instruments, and information sheets for human participants for three years following termination of the project,

(X) I understand all investigators associated with this research must renew their human participant research training every 3 years.

(X) I understand it is my responsibility to resubmit an application to the IRB if I need to make any changes that alter the exempt status determination and approval.

41e16d377dc500ffbda8e27ba2e36b55

(X) I understand this project will be closed by the IRB one year from the date of approval and records will be retained in the IRB office for 3 years after that date.

Project Title: WebSlicer Usability Testing

The signature page only may be submitted as a scanned document, faxed to x6012, or sent by intercampus mail to the IRB Administrator at Wannalancit, 2nd Floor. The entire application should be emailed as a word document to IRB@uml.edu

SIGNATURE: **Note: Students are not eligible to sign this page.**

PI Signature: /s/ Fred G. Martin	Date:3/29/16
Printed Name of PI: Fred G. Martin	

OR (X) Check here if submitted electronically from the PI's email account.

41e16d377dc500ffbda8e27ba2e36b55

A.2   IRB Consent Form

The document that follows is the consent form which was distributed to participants before beginning the usability study.

WEBSLICER USABILITY TESTING
INFORMED CONSENT

INTRODUCTION

You are invited to join a research study under the direction of Fred Martin and conducted by Michael Meding, "Webslicer Usability Testing." This study will involve the process of bringing a 3D model from digital design to physical world using our new slicing software. Please take whatever time you need to discuss the study with your family and friends, or anyone else you wish to. The decision to join, or not to join, is up to you.

In testing WebSlicer, we will be evaluating a web 3D print slicer. A print slicer is a piece of software which takes 3D models and divides them into virtual layers in preparation for being 3D printed. As a participant your role will be to choose a small model or figure from a set of pre-selected models. From there the participant will take this model through all of the steps necessary to prep it for being printed on a supplied 3D printer.

From this focused study I hope to gain insight into some of the strengths and weaknesses of my software design so that I may improve its final design.

WHAT IS INVOLVED IN THE STUDY?

If you decide to participate you will be asked to sit at a computer in one of our labs and answer questions from a pre-study survey. The questions will be about your general knowledge of 3D printing and should take no more than 5 minutes.

Participants will then be asked to perform a set of tasks to the best of their ability using our software. The tasks are short and simple, such as inputting given information into text fields or uploading a file.

Finally, participants will be asked to take a short post-study survey which asks them to rank the difficulty of given tasks from the prior exercise. Participants will also be able to add their own notes about their experience using our new software.

This research will take you about 30 min from start to finish.

You can stop participating at any time.

RISKS

There are no anticipated risks to participating in this study. However, if you experience any discomfort with using a computer then it is not recommended that you participate in this study.

BENEFITS TO TAKING PART IN THE STUDY?

There are no direct benefits to you however, participating in this research you may develop a better understanding of 3D printing and its surrounding technologies.

CONFIDENTIALITY

We will not be collecting any personally identifiable data during this study.

The results from this study will be published as a part of the student researcher's master's thesis and will only be shown in the form of what has been gained by the insight that you as a participant tell me is wrong with my software.

INCENTIVES

The only incentive in this study is the ability to take home a 3D printed model of your choice at the end of the study.

YOUR RIGHTS AS A RESEARCH PARTICIPANT

Participation in this study is voluntary. You have the right not to participate at all or to leave the study at any time. Deciding not to participate or choosing to leave the study will not result in any penalty or loss of benefits to which you are entitled, and it will not harm your relationship with WebSlicer and any University of Massachusetts Lowell staff.

CONTACTS FOR QUESTIONS OR PROBLEMS?

Call Fred Martin at 978-934-1964 or email at fredm@cs.uml.edu if you have questions about the study, any problems, unexpected physical or psychological discomforts, any injuries, or think that something unusual or unexpected is happening.

A.3   WebSlicer Recruitment Email

A recruitment email was needed to find a sufficient number of participants to preform usability testing.

Subject Line: 3D print slicer software prototype study

Professor Fred Martin and I, Michael Meding are looking for a small group of students to participate in our study, "Webslicer Usability Testing."

You will use our new software called WebSlicer. WebSlicer is a 3D print slicer designed to run entirely on the web. Its purpose is to take a model from a 3D modeling software such as AutoCAD and "slice" it into layers such that a 3D printer can understand how to build it.

Your participation in this study will take about 30 minutes upon arrival you will be asked to give your consent for the study. The study will take place in our lab (Olsen 306) and will be in the form of an online survey given on one of our lab computers. As a participant you are free to stop the study at any time with no consequence. As a token of our appreciation you will be able to go home with a 3D printed model of your choice.

If you choose to participate please send an email to Michael_Meding@student.uml.edu with the subject "WebSlicer Study". Please include your time availabilities and we will try our best to schedule you a time to meet with us.

Requirements:
- The ability to comfortably use a lab computer
- Some knowledge of the process of 3D printing is helpful but not required to participate

Please reply to Mike Meding, mikeymeding@gmail.com, if interested.

Thanks for your help!

A.4   WebSlicer Survey

The document that follows is the survey that was given to participants of the study. It consists of a few short questions and open ended comments.

WebSlicer Pre-Study Survey

Consent and questions while testing WebSlicer

* Required

1. **Have you read and signed the Consent Form and hereby give consent to participate in this study? ***
   Mark only one oval.

   ( ) Yes

   ( ) No   *Stop filling out this form.*

Pre-Study Question 1

2. **Have you ever used a 3D printer before?**
   Mark only one oval.

   ( ) Yes

   ( ) No   *Skip to question 5.*

Pre-Study Question 2

3. **Do you know what a slicer is or what slicing software does?**
   Mark only one oval.

   ( ) Yes

   ( ) No   *Skip to question 5.*

Pre-Study Question 3

4. **What slicing software have you used or know about?**
   Check all that apply.

   [ ] Cura

   [ ] Repetier Host

   [ ] KISSlicer

   [ ] Other: ....

Post-Study Survey

5. **How would you rate the difficulty of Task 1 (Create a new user)?** *
   Mark only one oval.

	1	2	3	4	5	
difficult	◯	◯	◯	◯	◯	easy

6. **Comments about Task 1**

   ....

7. **How would you rate the difficulty of Task 2 (Upload a file)?** *
   Mark only one oval.

	1	2	3	4	5	
difficult	◯	◯	◯	◯	◯	easy

8. **Comments about Task 2**

   ....

9. **How would you rate the difficulty of Task 3 (Adjust slicer settings)?** *
   Mark only one oval.

	1	2	3	4	5	
difficult	◯	◯	◯	◯	◯	easy

10. **Comments about Task 3**

    ....

11. **How would you rate the difficulty of Task 4 (Analyze and download Gcode)?** *
    Mark only one oval.

	1	2	3	4	5	
difficult	◯	◯	◯	◯	◯	easy

12. **Comments about Task 4**

    ....

13. **What is one thing that you would change
    about this software?**

    ....

14. **Did you run into any problems while preforming the tasks given?**

....

15. **Was anything difficult to use or hard to find when you needed it?**

....

16. **Additional comments**

....

....

....

....

....

Powered by

Google Forms

A.5   WebSlicer Tasks

Participants in the usability study were given a series of tasks as described by the document that follows.

WebSlicer

181 Bacon St
Natick, MA 01760
(214) 334-1905

Task Instructions

March 03, 2016

Overview

As a participant your goal today is to follow all the steps involved in taking a model from the virtual world into the physical world with the power of 3D printing. Starting by choosing from a small subset of models which I have pre-selected, you will download this and load it into the slicing software which I have developed. The models which you will be using for this study have been pre printed and you will be able to take one home at the end of the study.

Choose Your Model

3
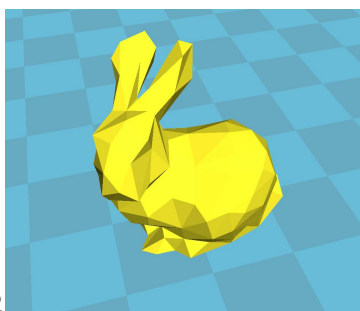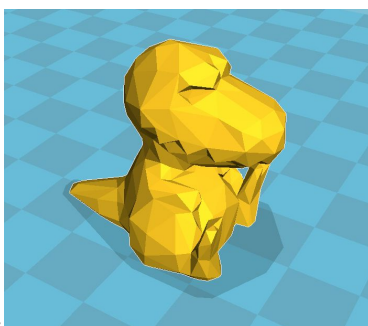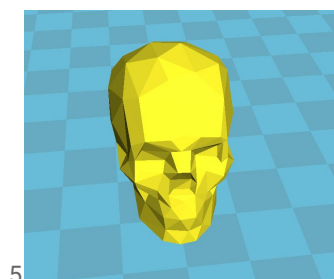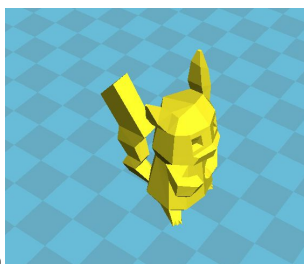
Task 1: Create a new user

User Goal: Get a new login ID.

This task is pretty simple. All you need to do is click the orange button labeled "generate new id." If you wish to return to your work later please copy this number as it is your reference number to your file.

Login

Client ID

8336d14c-b556-44db-b48f-73a4dc69e5cc

generate new id

4

Task 2: Upload your file

User Goal: Take your chosen model file and upload it to WebSlicer.

In this task you simply need to take the file which you have chosen and upload it using the upload button under slicer controls. The file will be provided to you on a USB stick. You upload to Webslicer in the same way that you would by selecting a file in Word.
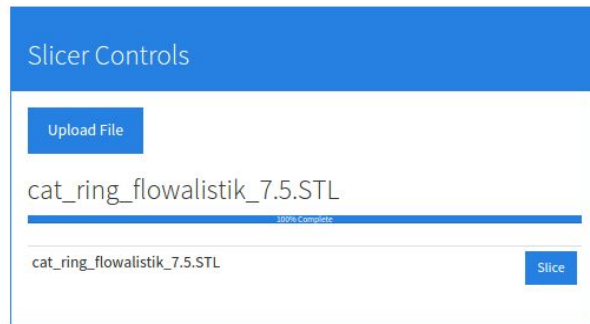
Your output should look something similar to the picture above.

5

Task 3: Adjust slicer settings

User Goal: Adjust settings so that the model will print correctly.
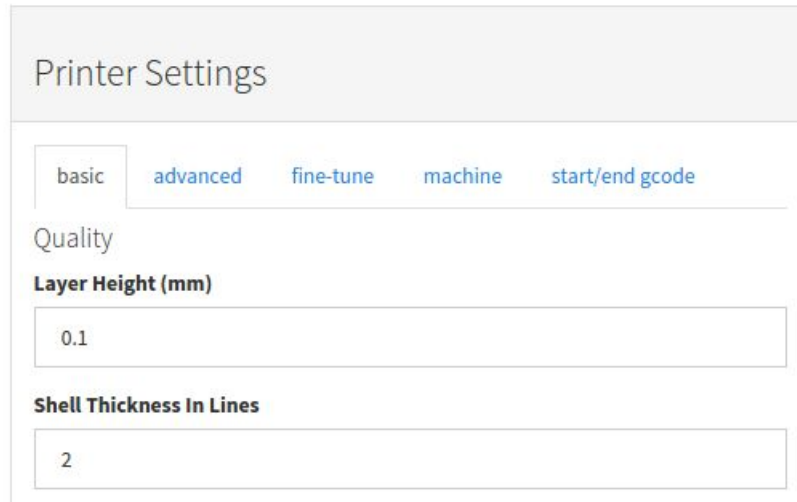
Basic Settings

The only settings that you will need to adjust will be under the basic tab. The settings that you need to find and change are as follows,

- Layer Height: 0.2
- Infill: 80%
- Filament Diameter: 1.75
- Shell Thickness in Lines: 2
- Printing Temperature: 245
- Bed Temperature: 110
- Print Speed: 40

The default values for the rest of the settings should do fine for your model.

6
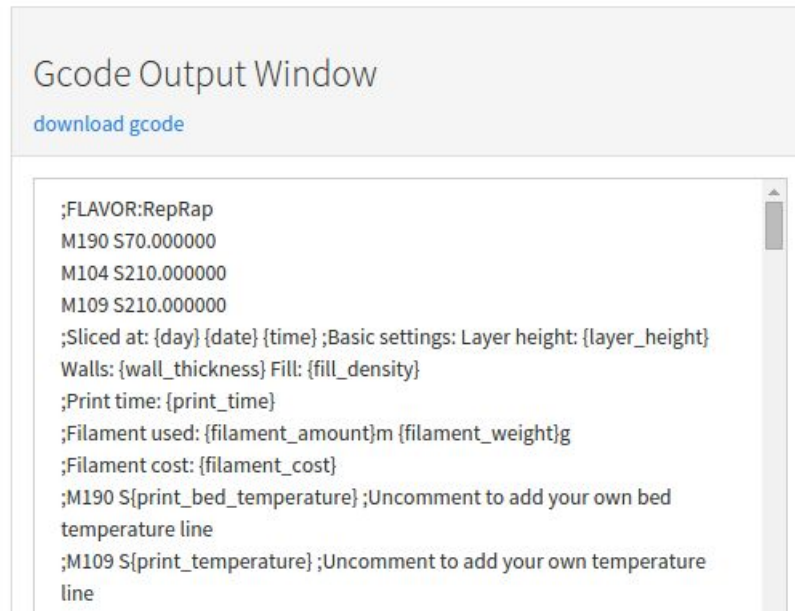
Your window will look different from the one above. The picture is simply as a reference.

Task 4: Analyze and download Gcode

User Goal: Use visualizer to see the resultant Gcode and download it for printing.

When you have completed task 4 you have to press the slice button next to the model you wish to slice. The slice button should change into a spinning cog to indicate that it is working on slicing your model in the cloud. When it completes you should see some output that looks something like the following image,

7

Gcode Output Window

download gcode

```
;FLAVOR:RepRap
M190 S70.000000
M104 S210.000000
M109 S210.000000
;Sliced at: {day} {date} {time} ;Basic settings: Layer height: {layer_height}
Walls: {wall_thickness} Fill: {fill_density}
;Print time: {print_time}
;Filament used: {filament_amount}m {filament_weight}g
;Filament cost: {filament_cost}
;M190 S{print_bed_temperature} ;Uncomment to add your own bed
temperature line
;M109 S{print_temperature} ;Uncomment to add your own temperature
line
```

You should also see a visualizer window like the one in the next picture. This is where you can step through all of the layers of your model and see exactly what it will do while it is printing.

When you are satisfied with this output you can press the "download gcode" text as it appears in the photo above. You will then get your Gcode file ready to be printed.

8

Gcode Visualizer

layerNumber 2   **progress** 50
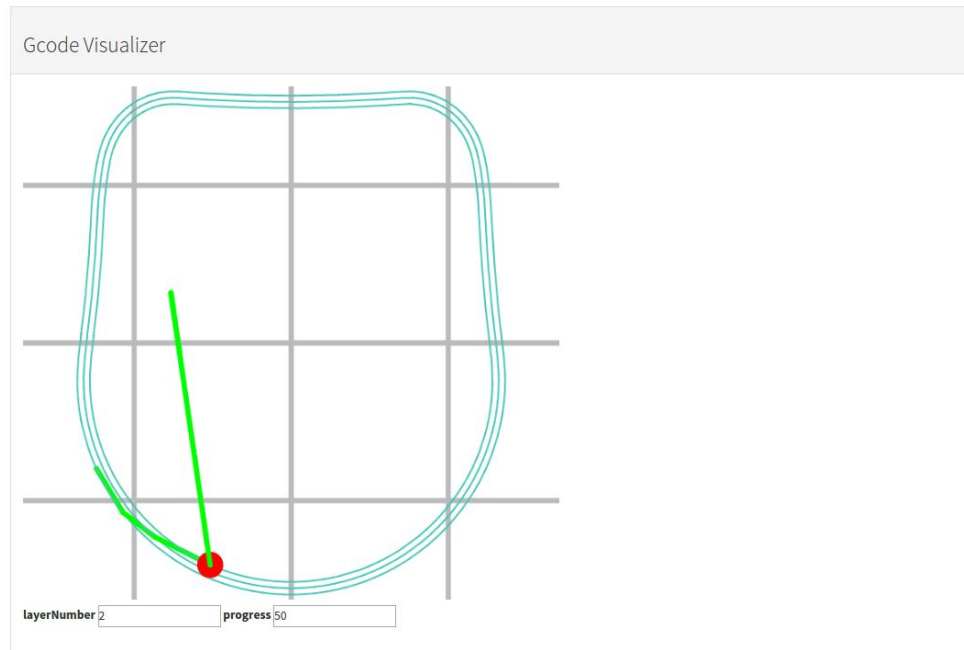
And that's it! You have successfully sliced your first model using WebSlicer. All that is left to do is grab the person conducting the research and tell them that you are done and you can watch as your model will get printed for you.

Thank you for your participation.

Appendix B

Server Side Code Appendix

Included in this appendix is the full source code written to run WebSlicer on a Wildfly application server. The section names are the file locations as they exist from the top level directory.

B.1   config/ApplicationConfig.java

Listing B.1: A required file in JavaEE for declaring the default application path when deployed.

```
 1 package com.partbuzz.slicer.config;
 2
 3
 4 import javax.ws.rs.ApplicationPath;
 5 import javax.ws.rs.core.Application;
 6
 7 /**
 8  * Created by mike on 1/9/16.
 9  */
10 @ApplicationPath("")
11 public class ApplicationConfig extends Application {
12 }
```

B.2   config/CORSFilter

Listing B.2: As CuraEngine allows cross origin responses a filter is requred to allow access by default in WildFly.

```
 1 package com.partbuzz.slicer.config;
```

```
2
3  import java.io.IOException;
4
5  import javax.ws.rs.container.ContainerRequestContext;
6  import javax.ws.rs.container.ContainerResponseContext;
7  import javax.ws.rs.container.ContainerResponseFilter;
8  import javax.ws.rs.ext.Provider;
9
10 /**
11  * @author mike
12  */
13 @Provider
14 public class CORSFilter implements ContainerResponseFilter {
15
16     @Override
17     public void filter(final ContainerRequestContext
          requestContext,
18                        final ContainerResponseContext cres) throws
                             IOException {
19         cres.getHeaders().add("Access-Control-Allow-Origin", "*");
20         cres.getHeaders().add("Access-Control-Allow-Headers", "
              origin, content-type, accept, authorization");
21         cres.getHeaders().add("Access-Control-Allow-Credentials",
              "true");
22         cres.getHeaders().add("Access-Control-Allow-Methods", "GET
              , POST, PUT, DELETE, OPTIONS, HEAD");
23         cres.getHeaders().add("Access-Control-Max-Age", "1209600")
              ;
24     }
25
26 }
```

B.3   cura/CuraEngine.java

Listing B.3: CuraEngine.java is the main code that preforms slicing throgh the CuraEngine C++ executable.

```
1  /*
2   * Copyright (c) 2016 Michael Meding -- All Rights Reserved.
3   */
4  package com.partbuzz.slicer.cura;
5
6  import com.partbuzz.slicer.util.CuraEngineException;
7  import com.partbuzz.slicer.util.PlatformExecutable;
8
9  import java.io.IOException;
10 import java.io.InputStream;
11 import java.util.ArrayList;
12 import java.util.List;
13 import java.util.logging.Logger;
14
15 /**
16  * Invoke the CURA slicing engine.
17  * <p>
18  * SAMPLE BASH
19  * ../../CuraEngine-master/build/CuraEngine slice -v -j ultimaker2
```

```
        . json −g −e −o " output / test . gcode " −l   .. / .. / models /
        ControlPanel . stl
20   *
21   * @author  mike
22   */
23  public class CuraEngine extends PlatformExecutable {
24
25      private static final Logger log = Logger . getLogger ( CuraEngine .
            class . getName () ) ;
26      private static final String CURAENGINE_PROG = "/usr/local/bin/
            CuraEngine ";
27      private final CuraEngineOptions options ;
28
29      public CuraEngine () {
30          this . options = new CuraEngineOptions () ;
31      }
32
33      public CuraEngineOptions options () {
34          return options ;
35      }
36
37      public String output = "";
38
39      /**
40       * Setup a stream drainer .
41       *
42       * @param fp the input stream
43       * @return the stream drainer
44       * @throws java . io . IOException
45       */
46      protected static StreamDrainer setupStreamDrain ( InputStream fp
            ) throws IOException {
47        final StreamDrainer drainer = new StreamDrainer ( fp );
48        // wait until we are draining
49        synchronized ( drainer ) {
50
51            log . info (" starting thread ");
52            Thread t = new Thread ( drainer );
53            t . setDaemon ( true );
54            t . start () ;
55
56            while (! drainer . hasStarted () ) {
57                try {
58                    log . info (" waiting for drainer thread ");
59                    drainer . wait (500);
60                    log . info (" drainer wait expired ");
61                } catch ( InterruptedException ex ) {
62                    Thread . interrupted () ;
63                    throw new IOException (" Unable to setup stream
                        drainer ");
64                }
65            }
66
67            log . info (" drainer started ");
68        }
69
70        return drainer ;
71      }
72
73      /**
```

```
74          *  Run the CURA engine.
75          *
76          *  @throws CuraEngineException
77          */
78         public void execute() throws IOException {
79             checkPlatformExecutable(CURAENGINE_PROG);
80
81             List<String> arguments = new ArrayList<>();
82             arguments.add(CURAENGINE_PROG); // add base executable
83             arguments.addAll(options.getOptions()); // add all options
                   in the correct order
84
85             for (String arg : arguments) {
86                 log.info("cura argument: " + arg);
87             }
88
89             // Process builder options for command line execution
90             ProcessBuilder pb = new ProcessBuilder();
91 //      pb.directory(basePath);
92             pb.redirectErrorStream(true);
93             pb.command(arguments);
94
95             // Create new thread and start execution
96             Process p = pb.start();
97             try {
98
99                 // Setup drains for both output and error streams
100                StreamDrainer stdout = setupStreamDrain(p.
                       getInputStream());
101 //             StreamDrainer stderr = setupStreamDrain(p.
       getErrorStream());
102
103                int status = p.waitFor(); // wait for slice to finish
104
105                if (status == 0) {
106
107                    // gather everything that is sent back from the
                           command.
108                    StringBuilder sb = new StringBuilder();
109                    InputStream fp = p.getInputStream();
110                    byte[] buffer = new byte[512];
111                    while (fp.available() > 0) {
112                        int n = fp.read(buffer);
113                        sb.append(new String(buffer, 0, n));
114                    }
115                    output = sb.toString();
116                    log.info(output); // just log the output for now
117                    log.info("STDOUT:" + stdout.getText());
118                    return;
119                } else {
120                    throw new CuraEngineException("unable to parse
                           gcode file");
121                }
122
123 //                if (options.haveIgnoreErrors()) {
124 //                    log.warning(sb.toString());
125 //                } else {
126 //                    throw new CuraEngineException(sb.toString());
127 //                }
```

```
128
129
130            } catch (InterruptedException e) {
131                throw new IOException(e);
132            } finally {
133                cleanupResources(p);
134            }
135        }
136
137        /**
138         * After execute is called the standard out of the call is
                    concatenated into output.
139         *
140         * @return
141         */
142        public String getOutput() {
143            return output;
144        }
145 }
```

B.4   cura/CuraEngineOptions.java

Listing B.4: All arguments to the CuraEngine C++ executeable must be compiled
before it can be executed. This code is the compiler for that argument list.

```
1 package com.partbuzz.slicer.cura;
2
3 import com.partbuzz.slicer.util.CuraEngineException;
4
5 import java.util.ArrayList;
6 import java.util.List;
7
8 /**
9  * Created by mike on 3/30/16.
10  */
11
12 public class CuraEngineOptions {
13
14     private String settingsFileName;
15     private String outputFileName;
16     private String modelFileName;
17     private boolean verbose = false;
18     private boolean p_option = false;
19     private boolean g_option = false;
20     private boolean e_option = false;
21     private boolean ignoreErrors = false;
22
23     /**
24      * Set the verbose mode.
25      *
26      * @return the options
27      */
28     public CuraEngineOptions verbose() {
29         this.verbose = true;
30         return this;
31     }
```

```java
32
33      /**
34       * Set the settings file. This is a JSON formatted file which
                includes all needed information for slicing the file.
35       *
36       * @param filename the settings filename
37       * @return the options
38       */
39      public CuraEngineOptions settingsFilename(String filename) {
40          this.settingsFileName = filename;
41          return this;
42      }
43
44      /**
45       * Set the output filename. The resulting gcode file name/
                location.
46       *
47       * @param filename the output filename
48       * @return the options
49       */
50      public CuraEngineOptions outputFilename(String filename) {
51          this.outputFileName = filename;
52          return this;
53      }
54
55      /**
56       * Set the model filename. This is the model to be sliced.
57       *
58       * @param filename
59       * @return options
60       */
61      public CuraEngineOptions modelFilename(String filename) {
62          this.modelFileName = filename;
63          return this;
64      }
65
66      /**
67       * This option is similar to verbose but instead of just
                logging the output to the screen it logs the output to
                the CuraEngine log files.
68       *
69       * @return the options
70       */
71      public CuraEngineOptions logProgress() {
72          this.p_option = true;
73          return this;
74      }
75
76      /**
77       * Switch setting focus to the current mesh group only.
78       * Used for one-at-a-time printing.
79       *
80       * @return the options
81       */
82      public CuraEngineOptions currentGroupOnly() {
83          g_option = true;
84          return this;
85      }
86
```

```
87      /**
88       * Adds a new extruder train for multi extrusion. For every
            time -e is included another extruder is added as on
            option to the slicer.
89       *
90       * @return the options
91       */
92      public CuraEngineOptions extruderTrainOption() {
93          e_option = true;
94          return this;
95      }
96
97      /**
98       * For debugging.
99       *
100      * @return the options
101      */
102     public CuraEngineOptions ignoreErrors() {
103         this.ignoreErrors = true;
104         return this;
105     }
106
107     public boolean haveIgnoreErrors() {
108         return ignoreErrors;
109     }
110
111     /*
112      * Get the options in the right sequence. To build a correct
            CuraEngine command line executable.
113      */
114     public List<String> getOptions() throws CuraEngineException {
115         List<String> list = new ArrayList<>();
116         list.add("slice"); // the main parameter of the CuraEngine
                executable.
117         if (verbose) {
118             list.add("-v");
119         }
120         if (p_option) {
121             list.add("-p");
122         }
123
124         if (settingsFileName == null) {
125             throw new CuraEngineException("no setting file defined
                    ");
126         } else {
127             list.add("-j");
128             list.add(settingsFileName);
129         }
130         if (g_option) {
131             list.add("-g");
132         }
133         if (e_option) {
134             list.add("-e");
135         }
136         if (outputFileName == null) {
137             throw new CuraEngineException("no output file defined"
                    );
138         } else {
139             list.add("-o");
140             list.add(outputFileName);
```

```
141            }
142            if (modelFileName == null) {
143                throw new CuraEngineException("no model file defined")
                      ;
144            } else {
145                list.add("-l");
146                list.add(modelFileName);
147            }
148
149            return list;
150        }
151 }
```

B.5 cura/StreamDrainer.java

Listing B.5: For ProcessBuilder to run correctly its output stream must be read from.

This code spawns a separate thread to read and log this output stream.

```java
 1 package com.partbuzz.slicer.cura;
 2
 3 import java.io.IOException;
 4 import java.io.InputStream;
 5 import java.util.logging.Level;
 6 import java.util.logging.Logger;
 7
 8 /**
 9  * Drain an input stream into a buffer.
10  */
11 class StreamDrainer implements Runnable {
12
13     private static final Logger log = Logger.getLogger(
           StreamDrainer.class.getName());
14     private final InputStream fp;
15     private final StringBuilder sb;
16     private boolean started;
17
18     public StreamDrainer(InputStream fp) {
19         this.fp = fp;
20         this.sb = new StringBuilder();
21         this.started = false;
22     }
23
24     public String getText() {
25         return sb.toString();
26     }
27
28     boolean hasStarted() {
29         return started;
30     }
31
32     @Override
33     public void run() {
34
35         // tell the invoker that we are draining
36         log.info("starting drainer thread");
37         synchronized (this) {
38             log.info("drainer thread  sync");
```

```
39
40                  started = true;
41                  notifyAll();
42
43                  byte[] buffer = new byte[512];
44                  try {
45                      boolean reading = true;
46                      int n;
47                      do {
48                          n = fp.read(buffer);
49                          if (n > 0) {
50                              sb.append(new String(buffer, 0, n));
51                          }
52                      } while (n >= 0);
53                  } catch (IOException ex) {
54                      Logger.getLogger("PlatformExecutor").log(Level.
                            SEVERE, null, ex);
55                  }
56                  log.info("done with the drainer");
57              }
58          }
59 }
```

B.6   rest/SlicerAPI.java

Listing B.6: The Wildfly application server hosts a public RESTful API which is where requests for slicing and data processing get sent to their corresponding methods.

```java
1  package com.partbuzz.slicer.rest;
2
3  import com.partbuzz.slicer.cura.CuraEngine;
4  import com.partbuzz.slicer.util.CuraEngineException;
5  import com.partbuzz.slicer.util.FileTracker;
6  import os.io.FileHelper;
7  import os.util.ExceptionHelper;
8  import os.util.StringUtils;
9  import os.util.json.DefaultJSONFactory;
10 import os.util.json.JSONException;
11 import os.util.json.JSONObject;
12
13 import javax.mail.internet.MimeBodyPart;
14 import javax.mail.internet.MimeMultipart;
15 import javax.mail.util.ByteArrayDataSource;
16 import javax.servlet.http.HttpServletRequest;
17 import javax.ws.rs.*;
18 import javax.ws.rs.core.Context;
19 import javax.ws.rs.core.MediaType;
20 import javax.ws.rs.core.Response;
21 import java.io.*;
22 import java.util.HashMap;
23 import java.util.logging.Level;
24 import java.util.logging.Logger;
25
26 /**
27  * Created by mike on 1/9/16.
28  */
29 @Path("slicer")
```

```
30 public class SlicerAPI {
31     private static final Logger log = Logger.getLogger(SlicerAPI.
          class.getName());
32     private final DefaultJSONFactory JSONFactory =
          DefaultJSONFactory.getInstance();
33
34     /**
35      * Simply returns 200 OK if called.
36      * <p>
37      *
38      * @param req
39      * @return
40      */
41     @GET
42     @Path("ping")
43     @Produces({MediaType.TEXT_PLAIN})
44     public Response ping(@Context HttpServletRequest req
45     ) {
46         log.info("ping called");
47         return Response.ok("PONG", MediaType.TEXT_PLAIN).build();
48     }
49
50     /**
51      * Add a new client to our database and file structure.
52      * This will set aside all of the files needed for a new
          client and link them correctly
53      *
54      * @param req
55      * @return The new uuid for that client so that we may track
          files
56      */
57     @POST
58     @Path("setupClient")
59     public Response setupClient(@Context HttpServletRequest req) {
60         try {
61             String uuid = FileTracker.setupNewClient();
62             JSONObject payload = new JSONObject();
63             payload.put("clientId", uuid);
64
65             return Response.ok().entity(payload).build();
66         } catch (CuraEngineException e) {
67             e.printStackTrace();
68             return Response.serverError().build();
69         }
70     }
71
72
73     /**
74      * Import an STL model file from the client
75      *
76      * @param req is the http request
77      * @return the names of the uploaded files or an error message
78      */
79     @POST
80     @Path("importStl/{clientId}")
81     public Response importStl(@Context HttpServletRequest req,
82                               @PathParam("clientId") String
                                    clientId) {
83         try {
84
```

```
85              ByteArrayDataSource bads = new ByteArrayDataSource(req
                   .getInputStream(), req.getContentType());
86              MimeMultipart mmp = new MimeMultipart(bads);
87
88              // Read the raw file
89              String filename = "";
90              for (int i = 0; i < mmp.getCount(); i++) {
91                  MimeBodyPart mbp = (MimeBodyPart) mmp.getBodyPart(
                       i);
92
93                  byte[] buffer = new byte[mbp.getSize()];
94                  mbp.getInputStream().read(buffer);
95
96                  // calculate a filename
97 //                  String md5Hex = StringUtils.encodeHexString(
       StringUtils.md5(buffer));
98 //                    filename = StringUtils.encode(md5Hex + "-" + mbp
       .getFileName());
99                  filename = StringUtils.encode(mbp.getFileName());
100
101                  // Save the file to disk
102                  File file = new File(FileTracker.getModelPathById(
                       clientId) + FileTracker.delimiter + filename);
103                  FileOutputStream fp = new FileOutputStream(file);
104                  fp.write(buffer);
105                  fp.close();
106
107              }
108
109              // register our new model file
110              String fileId = FileTracker.registerModelFile(clientId
                   , filename);
111
112              // build response message
113              JSONObject response = new JSONObject();
114              response.put("fileId", fileId);
115
116              return Response.ok().entity(response.toString()).build
                   ();
117          } catch (Exception e) {
118              return Response.ok().entity(ExceptionHelper.
                   getStackTrace(e)).build();
119          }
120      }
121
122      /**
123       * Import a settings file from the client
124       *
125       * @param req is the http request
126       * @return the names of the uploaded files or an error message
127       */
128      @POST
129      @Path("importSettings/{clientId}")
130      public Response importSettings(@Context HttpServletRequest req
           ,
131                                      @PathParam("clientId") String
                                          clientId) {
132          log.log(Level.INFO, "importing settings ");
133          StringBuilder sb = new StringBuilder();
```

```java
        try {
            String line;

            // parse input stream into a string
            try (BufferedReader br = new BufferedReader(new
                InputStreamReader(req.getInputStream()))) {
                while ((line = br.readLine()) != null) {
                    sb.append(line);
                }
            }
            String data = sb.toString(); // finish parse

            // Save to file
            String settingsPath = FileTracker.getSettingsFullPath(
                clientId);

            // write/overwrite file
            try (PrintWriter out = new PrintWriter(settingsPath))
                {
                out.print(data);
            }

            return Response.ok().build();
        } catch (IOException e) {
            e.printStackTrace();
            return Response.serverError().entity("File write error
                ").build();
        }
    }

    /**
     * Slice is the main function of this API. Given the
         parameters below it will return a formatted .gcode file
     * to the user.
     * <p>
     * The parameters required are a .stl file ID and a .json
         settings file ID.
     * These parameters are given when the above API methods are
         called. It is up to the client to track these ID's
     * <p>
     * SAMPLE JSON FORMAT
     * {"modelId":"UUID","settingsId":"UUID"}
     *
     * @param req is the http request
     * @return the names of the uploaded files or an error message
     */
    @POST
    @Path("slice/{clientId}/{modelId}")
    public Response slice(@Context HttpServletRequest req,
                          @PathParam("clientId") String clientId,
                          @PathParam("modelId") String modelId
                          ) {
        try {
            // SAMPLE COMMAND RESULT
            // ../../CuraEngine-master/build/CuraEngine slice -v -
                j ultimaker2.json -g -e -o "output/test.gcode" -l
                ../../models/ControlPanel.stl

            // create new CuraEngine platform executable
```

```
182                 CuraEngine ce = new CuraEngine();
183                 ce.options() // add options. order is irrelevant.
184                         .verbose()
185                         .currentGroupOnly()
186                         .extruderTrainOption()
187                         .logProgress()
188                         .settingsFilename(FileTracker.
                                getSettingsFullPath(clientId))
189                         .outputFilename(FileTracker.getOutputFilePath(
                                clientId))
190                         .modelFilename(FileTracker.getModelFullPath(
                                clientId, modelId));
191                 ce.execute();
192
193                 // build response from output file and return it.
194                 JSONObject response = new JSONObject();
195                 byte[] raw = FileHelper.readContent(new File(
                        FileTracker.getOutputFilePath(clientId)));
196                 String gcode = new String(raw);
197                 response.put("gcode", gcode);
198
199                 return Response.ok().entity(response.toString()).build
                        ();
200             } catch (IOException e) {
201                 return Response.serverError().entity(e.getMessage()).
                        build();
202             }
203         }
204
205     /**
206      * This function is to run a full test slice with a sample
             model and return its gcode.
207      * Eliminates the need for a UI to run backend tests.
208      * @param req is the http request
209      * @return the names of the uploaded files or an error message
210      */
211     @POST
212     @Path("testSlice")
213     public Response testSlice(@Context HttpServletRequest req) {
214         try {
215                 // create new CuraEngine platform executable
216                 CuraEngine ce = new CuraEngine();
217                 ce.options() // add options. order is irrelevant.
218                         .verbose()
219                         .currentGroupOnly()
220                         .extruderTrainOption()
221                         .logProgress()
222                         .settingsFilename("/tmp/webslicer/test/
                                prusa_i3.json")
223                         .outputFilename("/tmp/webslicer/test/output.
                                gcode")
224                         .modelFilename("/tmp/webslicer/test/testModel.
                                stl");
225                 ce.execute();
226
227                 // build response from output file and return it.
228                 JSONObject response = new JSONObject();
229                 byte[] raw = FileHelper.readContent(new File("/tmp/
                        webslicer/test/output.gcode"));
230                 String gcode = new String(raw);
```

```
231                    response.put("gcode", gcode);
232
233                    return Response.ok().entity(response.toString()).build
                          ();
234            } catch (IOException e) {
235                    return Response.serverError().entity(e.getMessage()).
                          build();
236            }
237        }
238
239        /**
240         * Get all of the model file names and tracking ids associated
                  with a client uuid.
241         *
242         * @param req
243         * @param clientId
244         * @return
245         */
246        @GET
247        @Path("getFiles/{clientId}")
248        @Produces("application/json")
249        public Response getFiles(@Context HttpServletRequest req,
250                                 @PathParam("clientId") String
                                       clientId) {
251            HashMap filesMap = FileTracker.getAllModelFiles(clientId);
252            return Response.ok().entity(filesMap).build();
253        }
254
255        /**
256         * A util function to parse an input JSON stream and parses to
                  a JSONObject
257         *
258         * @param req
259         * @return
260         */
261        private JSONObject parseJSONStream(HttpServletRequest req) {
262            StringBuilder sb = new StringBuilder();
263            try {
264                String line;
265
266                // parse input stream into a string
267                try (BufferedReader br = new BufferedReader(new
                      InputStreamReader(req.getInputStream()))) {
268                    while ((line = br.readLine()) != null) {
269                        sb.append(line);
270                    }
271                }
272                String data = sb.toString();
273
274                log.info(data);
275
276                return JSONFactory.jsonObject(data);
277
278            } catch (IOException | JSONException e) {
279                log.severe(e.getMessage());
280                throw new JSONException("Stream parse error.");
281            }
282
283        }
284 }
```

B.7   util/CuraEngineException.java

Listing B.7: This exception class gets thrown when any issue arises while CuraEngine is preforming a slice.

```java
/*
 * Copyright (c) 2016 Michael Meding -- All Rights Reserved.
 */
package com.partbuzz.slicer.util;

import java.io.IOException;

/**
 * A CURA engine exception.
 *
 * @author mike
 */
public class CuraEngineException extends IOException {

  public CuraEngineException() {
  }

  public CuraEngineException(String message) {
    super(message);
  }

  public CuraEngineException(String message, Throwable cause) {
    super(message, cause);
  }

  public CuraEngineException(Throwable cause) {
    super(cause);
  }

}
```

B.8   util/FileTracker.java

Listing B.8: FileTracker tracks all of the files uploaded to the server to a series of hash maps. It also issues unique identifiers for each file that gets uploaded so that they may be tracked and accessed simply.

```java
package com.partbuzz.slicer.util;

import os.io.FileHelper;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Map;
```

```java
import java.util.UUID;

/**
 * For tracking the associated files with a particular client id.
 * FILE STRUCTURE
 * /main---/(clientUUID)---output.gcode
 * -settings.json
 * -/models---(uuid-filename.stl)
 *  -(...stl)
 *  -...
 * <p>
 * -/common---fdmprinter.json
 * -/presets--ultimaker.json
 * -prusai3.json
 *  -...
 * <p>
 * Structure dictates file limit and will throw exceptions if # of
 *     files goes above limit.
 * <p>
 * Created by mike on 1/14/16.
 */
public class FileTracker {
    //TODO: This really needs to come from a properties file (JNDI
            path to properties file)
    public static String delimiter = File.separator;
    public static String basePath = delimiter + "tmp" + delimiter
        + "webslicer";
    public static String common = basePath + delimiter + "common";
    public static String modelDir = "models";
    public static String settingsFileName = "settings.json";
    public static String fdmprinterFile = "fdmprinter.json";
    public static String outputFile = "output.gcode";
    private static Map<String, HashMap<String, String>>
        clientFilesMap = new HashMap<>();

    /**
     * Same as above only for the model file.
     *
     * @param clientId The id of the client in question
     * @param fileId   The tracked filename which must be managed
         by client
     * @return
     */
    public static String getModelFileName(String clientId, String
        fileId) {
        return clientFilesMap.get(clientId).get(fileId);
    }

    /**
     * Get the full path to a model file
     *
     * @param clientId
     * @param fileId
     * @return
     */
    public static String getModelFullPath(String clientId, String
        fileId) {
        return basePath + delimiter + clientId + delimiter +
            modelDir + delimiter + getModelFileName(clientId,
```

```
                    fileId);
63      }
64
65      /**
66       * Get the path to the model file directory
67       *
68       * @return
69       */
70      public static String getModelPathById(String clientId) {
71          return basePath + delimiter + clientId + delimiter +
                modelDir;
72      }
73
74      /**
75       * Return the entire hashmap of model files associated with a
              client
76       *
77       * @param clientId
78       * @return
79       */
80      public static HashMap<String, String> getAllModelFiles(String
            clientId) {
81          return clientFilesMap.get(clientId);
82      }
83
84      public static String getOutputFilePath(String clientId) {
85          return basePath + delimiter + clientId + delimiter +
                outputFile;
86      }
87
88      /**
89       * Register a new model file with our registry
90       *
91       * @param fileName
92       * @return
93       */
94      public static String registerModelFile(String clientId, String
            fileName) {
95          String uuid = UUID.randomUUID().toString(); // generate
                new tracking uuid
96          clientFilesMap.get(clientId).put(uuid, fileName); // store
                 this filename using that new tracking uuid for that
                 client
97          return uuid;
98      }
99
100     /**
101      * Gets the fully qualified path to the settings file for a
              specified client
102      *
103      * @param clientId
104      * @return
105      */
106     public static String getSettingsFullPath(String clientId) {
107         return basePath + delimiter + clientId + delimiter +
                settingsFileName;
108     }
109
110     /**
```

```
111       * Remove all files associated with a client and remove its
              hashmap tracking in our server
112       *
113       * @param clientId
114       */
115      public static void removeClient(String clientId) {
116          try {
117              FileHelper.delete(new File(basePath + delimiter +
                     clientId));
118              clientFilesMap.remove(clientId);
119          } catch (IOException e) {
120              e.printStackTrace();
121          }
122      }
123
124      /**
125       * Completely remove model file from our disk
126       *
127       * @param clientId
128       * @param id
129       */
130      public static void removeModelFile(String clientId, String id)
              {
131          try {
132              FileHelper.delete(new File(getModelFullPath(clientId,
                     id)));
133              clientFilesMap.get(clientId).remove(id);
134          } catch (IOException e) {
135              e.printStackTrace();
136          }
137      }
138
139
140      /**
141       * Register and setup a new client with our registry.
142       *
143       * @return the new clients UUID.
144       */
145      public static String setupNewClient() throws
             CuraEngineException {
146
147          // generate new client uuid
148          String uuid = UUID.randomUUID().toString();
149
150          // Create empty file as clients home directory
151          if (!new File(basePath + delimiter + uuid + delimiter +
                 modelDir).mkdirs()) {
152              throw new CuraEngineException("Could not create file
                     path correctly");
153          }
154
155          // Symbolically link fdmprinter.json from common to this
                 clients home directory
156          try {
157              Path targetPath = Paths.get(basePath + delimiter +
                     uuid + delimiter + fdmprinterFile);
158              Path sourceFile = Paths.get(common + delimiter +
                     fdmprinterFile);
159              Files.createSymbolicLink(targetPath, sourceFile);
160          } catch (IOException e) {
```

```
161            e.printStackTrace();
162        }
163
164        // Give our new client a file tracking hashmap
165        clientFilesMap.put(uuid, new HashMap<String, String>());
               // initalize client with new empty hashmap
166
167        return uuid;
168    }
169
170 }
```

B.9 util/PlatformExecutable.java

Listing B.9: The PlatformExecutable class verifies if the local C++ executable is available and sets up the input and output pipes for CuraEngine if it is.

```
1 /*
2  * Copyright (c) 2016 Michael Meding -- All Right Reserved.
3  */
4 package com.partbuzz.slicer.util;
5
6 import java.io.Closeable;
7 import java.io.File;
8 import java.io.IOException;
9 import java.io.InputStream;
10
11 /**
12  * See if an executable is present and runnable.
13  *
14  * @author mike
15  */
16 public abstract class PlatformExecutable {
17
18    /**
19     * Check if a particular executable exists
20     *
21     * @param path is the path
22     */
23    protected static void checkPlatformExecutable(String path) {
24      File file = new File(path);
25      if (!(file.exists() && file.canExecute())) {
26        throw new UnsupportedOperationException(path + " is not
              supported on this platform");
27      }
28    }
29
30    /**
31     * See if a port is in the valid range 0-65535
32     *
33     * @param port the port number
34     */
35    protected static void checkPortNumberLegal(int port) {
36      if (port < 0 || port > 65535) {
37        throw new IllegalArgumentException("TCP/IP Port " + port + "
              illegal");
```

```java
38          }
39      }
40
41      /**
42       * Gather all the output a command may have sent back. Used if a
                system
43       * command did not finish successfully.
44       *
45       * @param p is the process
46       * @return the string
47       */
48      protected static String gatherCommandOutput(Process p) throws
            IOException {
49          StringBuilder sb = new StringBuilder();
50          InputStream fp = p.getInputStream();
51          byte[] buffer = new byte[512];
52          while (fp.available() > 0) {
53              int n = fp.read(buffer);
54              sb.append(new String(buffer, 0, n));
55          }
56
57          return sb.toString();
58      }
59
60      /**
61       * Cleanup the external process resources
62       *
63       * @param p is the process
64       */
65      protected static void cleanupResources(Process p) {
66          if (p == null) {
67              return;
68          }
69
70          close(p.getOutputStream());
71          close(p.getInputStream());
72          close(p.getErrorStream());
73          p.destroy();
74      }
75
76      private static void close(Closeable c) {
77          if (c != null) {
78              try {
79                  c.close();
80              } catch (Throwable t) {
81                  // ignore
82              }
83          }
84      }
85  }
```

Appendix C

Client Side Code Appendix

Included in this appendix is the full source code for the client side WebSlicer application.

C.1   app.js

Listing C.1: Main app code where the angular module WebSlicer is defined. This is similar to a main function in C++.

```
1  /**
2   * Created by mike on 9/25/15.
3   */
4  (function () {
5      var app = angular.module("WebSlicer", ["ui.router", "ui.
           bootstrap", "ngAnimate", "ngFileUpload"]);
6
7      app.controller("Main", ["$http", "$scope", "$window", "$
           rootScope", function ($http, $scope, $window, $rootScope)
           {
8
9          // enable all bootstrap tooltips
10         //$(function () {
11         //    $('[data-toggle="tooltip"]').tooltip()
12         //});
13
14         // the main servers base url for making REST calls
15         $rootScope.baseUrl = "http://localhost:8080/WebSlicer/
               slicer";
16         $rootScope.clientId = "";
17         $rootScope.busy = false;
18
19         // environment vars
20         //$rootScope.baseUrl;
21         $scope.title = "Web Slicer";
```

```
22          $scope.displayOctoprintData = false;
23          //$scope.clientId = "";
24          $scope.modelFiles = {};
25          $scope.gcode = {"gcode": ""};
26
27          // make gcode available for download
28          var blob = new Blob([$scope.gcode.gcode], {type: 'text/
                plain'});
29          $scope.gcodeDownload = (window.URL || window.webkitURL).
                createObjectURL(blob);
30
31          /**
32           * Generate a new client id and layout all of the files
                 for this new client on the server
33           */
34          $scope.generateClientId = function () {
35              $http({
36                  method: 'POST',
37                  url: $rootScope.baseUrl + "/setupClient"
38              }).then(function successCallback(response) {
39                  console.log(response);
40                  $rootScope.clientId = response.data.clientId;
41              }, function errorCallback(response) {
42                  console.error(response);
43              });
44          };
45
46          //DOWNLOAD GCODE FUNCTION
47          $scope.downloadGcode = function (gcode) {
48              var blob = new Blob([gcode], {type: "application/json;
                    charset=utf-8;"});
49              var downloadLink = angular.element('<a></a>');
50              //var downloadLink = angular.element(elem.
                    querySelector(".downloadGcode"));
51              downloadLink.attr('href', window.URL.createObjectURL(
                    blob));
52              downloadLink.attr('download', 'output.gcode');
53              downloadLink[0].click();
54
55              //$window.open("data:application/json;charset=utf-8,"
                    + encodeURIComponent(gcode));
56          };
57
58
59          /**
60           * The main slice function. This will only operate if the
                 client has both uploaded a settings file and a model
                 file.
61           * Both of these files will be given an id that the client
                 must track in order to slice a file properly.
62           *
63           * @param modelId
64           */
65          $rootScope.slice = function (modelId) {
66              if (modelId) {
67                  $rootScope.busy = true;
68
69                  // run import settings to update settings on
                        server BEFORE SLICING
```

```
70                      $rootScope.importSettings()
71
72                          .then(function successCallback(response) {
73                              console.log(response);
74
75                                  // SEND SLICE REQUEST!
76                                  $http({
77                                      method: 'POST',
78                                      url: $rootScope.baseUrl + "/slice/" +
                                            $rootScope.clientId + "/" +
                                            modelId
79                                  }).then(function successCallback(response)
                                        {
80                                      // capture our gcode as a response.
81                                      console.log(response);
82                                      $scope.gcode = response.data;
83                                      $rootScope.busy = false;
84
85                                      // render our gcode in the 2D viewer
86                                      $rootScope.render(response.data.gcode)
                                            ;
87
88                                  }, function errorCallback(response) {
89                                      console.error(response);
90                                      $rootScope.busy = false;
91                                  });
92
93                          }, function errorCallback(response) {
94                              console.error(response);
95                              console.error("import settings before
                                    slice");
96                          });
97
98              } else {
99                  console.error("model or settings id missing");
100             }
101         };
102     }]);
103
104     app.config(["$stateProvider", "$urlRouterProvider", "$
            controllerProvider", "$compileProvider", function
            appConfig($stateProvider, $urlRouterProvider, $
            controllerProvider, $compileProvider) {
105
106         // default route
107         $urlRouterProvider.otherwise("home");
108
109         // states
110         $stateProvider
111             .state("home", {
112                 url: "/home",
113                 templateUrl: "public/index.html"
114             });
115
116         // controller config
117         $controllerProvider.allowGlobals();
118
119         // post compile config
120         $compileProvider.aHrefSanitizationWhitelist(/^\s*(https?|
                ftp|mailto|tel|file|blob):/);
```

```
121        }]);
122
123        app.directive('fileModel', ['$parse', function ($parse) {
124            return {
125                restrict: 'A',
126                link: function (scope, element, attrs) {
127                    var model = $parse(attrs.fileModel);
128                    var modelSetter = model.assign;
129
130                    element.bind('change', function () {
131                        scope.$apply(function () {
132                            modelSetter(scope, element[0].files[0]);
133                        });
134                    });
135                }
136            };
137        }]);
138 })();
```

C.2   index.html

Listing C.2: As this is a single page web application this is the only HTML file. It contains all of the libraries needed to run all subsequent sections.

```html
1  <!DOCTYPE html>
2  <html ng-app="WebSlicer">
3  <head lang="en">
4      <meta charset="UTF-8">
5      <title>Web Slicer</title>
6
7      <link rel="icon" type="image/png" href="img/logo22-plain.png"/>
8
9      <!--libraries-->
10     <script src="bower_components/jquery/dist/jquery.js"></script>
11     <script src="bower_components/bootstrap/dist/js/bootstrap.js"></
           script>
12     <script src="bower_components/angular/angular.js"></script>
13     <script src="bower_components/angular-animate/angular-animate.js"
           ></script>
14     <script src="bower_components/angular-ui-router/release/angular-
           ui-router.js"></script>
15     <script src="bower_components/angular-bootstrap/ui-bootstrap.js">
           </script>
16     <script src="bower_components/angular-bootstrap/ui-bootstrap-tpls
           .js"></script>
17     <script src="bower_components/ng-file-upload/ng-file-upload.js"><
           /script>
18
19     <link rel="stylesheet" href="bower_components/font-awesome/css/
           font-awesome.css">
20
21     <!--css-->
22     <!--<link rel="stylesheet" href="bower_components/bootstrap/dist/
           css/bootstrap.css">-->
23     <link rel="stylesheet" href="bower_components/bootstrap/dist/css/
           bootstrap-cosmo.css">
```

```html
24      <link rel="stylesheet" href="css/custom.css"/>
25
26      <!-- MAIN CONTROLLER AND CONFIG -->
27      <script src="app.js"></script>
28
29      <!--FILE CONTROLS-->
30      <script src="model/files/filesController.js"></script>
31      <script src="model/files/filesDirective.js"></script>
32
33      <!--SETTINGS-->
34      <script src="model/profile/settingsController.js"></script>
35      <script src="model/profile/settingsTabDirective.js"></script>
36      <script src="model/profile/settingsBuilderService.js"></script>
37
38      <!--OCTOPRINT-->
39      <script src="model/octoprint/octoprintService.js"></script>
40      <script src="model/octoprint/octoprintDirective.js"></script>
41      <script src="model/octoprint/octoprintController.js"></script>
42
43      <!--VISUALIZER-->
44      <script src="viewer/lib/zlib.min.js"></script>
45      <script src="viewer/gcodeFactory.js"></script>
46      <script src="viewer/renderFactory.js"></script>
47      <script src="viewer/viewer.js"></script>
48
49  </head>
50  <body ng-controller="Main">
51
52  <div class="container" ng-controller="SettingsController">
53  <div class="row">
54      <div class="col-lg-6 col-lg-offset-3">
55          <h1 class="title"><img src="img/logo22.svg" width="150px">{{
                title}}</h1>
56      </div>
57  </div>
58  <div class="row">
59  <div class="col-lg-6 col-lg-offset-3">
60      <div class="panel panel-default">
61          <div class="panel-heading">
62              <h3>Login</h3>
63          </div>
64          <div class="panel-body">
65          <form>
66          <div class="form-group">
67          <label for="clientId">Client ID</label>
68          <input class="form-control" type="text" id="clientId" ng-
                model="$root.clientId">
69          <button class="btn btn-warning" ng-click="generateClientId()"
                >generate new id</button>
70          </div>
71          </form>
72          </div>
73      </div>
74  </div>
75  </div>
76
77  <div class="row">
78
```

```
79  <!--CONTROLS-->
80  <div class="col-lg-6">
81
82  <!--<filecontrols></filecontrols>-->
83  <div ng-controller="FilesController">
84
85  <div class="panel panel-primary">
86      <div class="panel-heading">
87          <h3>Slicer Controls</h3>
88      </div>
89      <div class="panel-body">
90          <ul class="nav nav-pills nav-stacked">
91
92              <!--file chooser-->
93              <li role="presentation">
94
95                  <button class="btn btn-primary" type="file" ngf-
                        select="uploadFile($file,$invalidFiles)"
96                          ngf-max-height="1000" ngf-max-size="2MB">
97                      Upload File
98                  </button>
99
100                 <div ng-show="f.progress >= 0">
101                 <h3>{{f.name}} {{errFile.name}} {{errFile.$error}} {{
                        errFile.$errorParam}}</h3>
102                 <div class="progress">
103                 <div class="progress-bar" role="progressbar" aria-
                        valuenow="{{f.progress}}"
104                     aria-valuemin="0"
105                     aria-valuemax="100" style="width: {{f.progress
                        }}%;">
106                     <!--<div class="progress-bar" role="progressbar"
                        aria-valuenow="{{f.progress}}" aria-valuemin="
                        0"-->
107                     <!--aria-valuemax="100">-->
108                     {{f.progress}}% Complete
109                 </div>
110                 </div>
111                 </div>
112
113              </li>
114
115              <!--current file list-->
116              <li role="presentation" ng-if="$root.clientId">
117                  <p>{{modelFileId}}</p>
118
119                  <table class="table table-hover table-condensed">
120                      <tbody>
121
122                      <tr ng-repeat="item in modelFiles">
123                      <td>{{item.value}}
124                      <button ng-if="!$root.busy" class="btn btn-
                        primary btn-sm pull-right"
125                              ng-click="$root.slice(item.key)">
126                          Slice
127                      </button>
128                      <i ng-if="$root.busy" class="fa fa-3x fa-cog fa-
                        spin pull-right"></i>
129
```

```
130
131                                </td>
132                              </tr>
133
134                          </tbody>
135                      </table>
136
137              </li>
138
139              <!--slice button-->
140              <li role="presentation">
141                  <!--<button class="btn btn-primary" ng-click="
                         getFileList()">Refresh file list</button>-->
142              </li>
143
144          </ul>
145
146      </div>
147  </div>
148
149
150      </div>
151  </div>
152
153  <!--<div class="col-lg-6">-->
154  <!--<octoprint></octoprint>-->
155  <!--</div>-->
156
157  </div>
158
159  <div class="row" ng-controller="SettingsController">
160
161  <!--SETTINGS PANEL-->
162  <div class="col-lg-6">
163
164      <div class="panel panel-default">
165          <div class="panel-heading">
166              <h3>Printer Settings</h3>
167          </div>
168          <div class="panel-body">
169
170              <uib-tabset>
171              <uib-tab heading="basic">
172                  <curasettings src="model/profile/json/basic.json"></
                         curasettings>
173              </uib-tab>
174              <uib-tab heading="advanced">
175                  <curasettings src="model/profile/json/advanced.json">
                         </curasettings>
176              </uib-tab>
177              <uib-tab heading="fine-tune">
178                  <curasettings src="model/profile/json/fine-tune.json"
                         ></curasettings>
179              </uib-tab>
180              <uib-tab heading="machine">
181                  <curasettings src="model/profile/json/machine.json"><
                         /curasettings>
182              </uib-tab>
```

```html
183          <uib-tab heading="start/end gcode">
184              <curasettings src="model/profile/json/start-end-gcode
                      .json"></curasettings>
185          </uib-tab>
186          </uib-tabset>
187
188      </div>
189    </div>
190
191 </div>
192
193 <!--GCODE OUTPUT PANEL-->
194 <div class="col-lg-6" ng-if="gcode.gcode">
195      <div class="panel panel-success">
196          <div class="panel-heading">
197              <h3>Gcode Output Window</h3>
198              <a ng-href="{{gcodeUrl}}" ng-click="downloadGcode(gcode.
                      gcode)">download gcode</a>
199          </div>
200          <div class="panel-body">
201              <button class="btn btn-sm" ng-click="showGcode = !
                      showGcode" data-toggle="tooltip" data-placement="
                      bottom" title="WARNING! This may freeze browser!">
                      press to view gcode</button>
202              <textarea ng-if="showGcode" cols="30" rows="50" ng-model=
                      "gcode.gcode"
203                      class="form-control"></textarea>
204
205          </div>
206      </div>
207 </div>
208 </div>
209
210 <!--2D GCODE VISUALIZER-->
211 <div class="row" ng-controller="gcodeController" ng-show="gcode.gcode
      ">
212 <div class="panel panel-success">
213      <div class="panel-heading">
214          <h3>Gcode Visualizer</h3>
215      </div>
216      <div class="panel-body">
217          <canvas id="canvas" width="650" height="620"></canvas>
218          <form>
219              <div class="form-group">
220                  <label for="layerNumber">layerNumber</label>
221                  <input type="number" id="layerNumber" ng-model="
                          layerNumber">
222                  <label for="progress">progress</label>
223                  <input type="number" id="progress" ng-model="progress
                          ">
224              </div>
225          </form>
226      </div>
227 </div>
228 </div>
229
230 <div class="footer row">
231      <p class="text-center">&copy; Michael U.B. Meding 2016</p>
```

```
232  </div>
233
234  </div>
235  </div>
236  </body>
237  </html>
```