Datava 2: Hardware Control from the Internet

Guilherme Alves, Andrew Hupp, Julius Kaminski, Nicholas Karst

Date: June 12th, 2020

Table of Contents

1. Introduction

Datava Inc. is a software company specializing in business software for other companies. One of their clients is Purakal Cylinders, an Oregon-based pneumatic and hydraulic cylinder manufacturer. As the cylinders are manufactured they are marked with tags that hold information about the cylinder such as serial number, specifications, and a QR code. These tags are created using a laser marker (shown in Figure 1). The marker takes a template and then loads the required information into certain fields. While the marker comes with software to operate the hardware, it does not match up with how Purakal wishes to structure their workflow. Luckily, the interface implements an API that allows for a great deal of control over the hardware, such as allowing a user to set template parameters, request previews, and query the marker's status in several ways. With this API, Datava has been hired to build a custom interface for the laser marker that will match Purakal's wishes and allow them to create these tags easier and faster.

This custom interface would use the API created by the laser marker's manufacturer, as well as various database tables specifying the necessary information to simplify Purakal's process. To further optimize this process, job information would be loaded by scanning QR codes that are already in use throughout their manufacturing process. The process of scanning this QR code would query the relevant database tables for necessary information and display this data to the user. This project was then passed on to this field-session team for implementation. This implementation had the main goal of making the process of completing these jobs more seamless and easy enough for a first time user to operate.

Figure 1: TYKMA Minilase™ Laser Marking System

2. Specifications

Functional Specifications

The system loads the initial data of the templates and the marks from an interface that is Purakal specific. From there, the user is shown what cylinders are a part of a particular order and can choose to do all available marks or only certain marks. Then the system will take the selected jobs and will pass the information to the laser marker through the TCP/IP interface. At this point, the operator will only need to load the material to be marked and start the laser marker.

In order for this system to be flexible, the database holds some configuration regarding the marker. As implemented, the tables hold the marker's address and default request timeout, but other configuration information may also be added later if necessary. This configuration has the advantage of allowing a single marker to move within the network and allows multiple markers to be used if future expansion is deemed appropriate.

Non-functional Specifications

The user interface is designed to be simple to use and easy to understand. The application displays to the user the available jobs to mark in a table. This table displays the order in which marks will be made, the name of the template, and the parameters that will be used to mark the template. The user can choose to mark all or mark only the top row, which can be changed by dragging and dropping the desired row to the top. The top row will also be previewed to the right of the table so the user is informed of what the mark should look like. At any point, the user can stop the process after the current part is done or immediately using two buttons on the interface.

Since our application was built upon Datava's existing codebase we had to use the same languages and technologies as their other applications. For the client-side work, we used JavaScript and the framework Ext JS. This framework allowed our user interface to handle all the requests to the server-side of the project and facilitated extensible code. On the server-side, PHP was the programming language of choice, with the database used to store and load data being a Microsoft SQL server. For version control we use Subversion. Most of the coding standards being used are standard practice for each language.

3. Architecture

This system architecture was built largely in two parts: the user interface and the back-end. The following subsections describe the architecture of each of these parts in detail.

Front-End Architecture

To accomplish these tasks, the team used a version of Ext JS to create an interface that would be pleasing to the end-user and was also able to communicate effectively with the back-end. The goal of this interface is to display information to the user in an understandable and pleasing way, and receive input and send it along to the back-end. The user interface is largely a table that displays the template ID and the parameters for that template. The information of this table will be gathered by an outside system. For efficiency, this table has a drag and drop feature that allows users to change the order that parts are marked in. The UI is set up so that the specific template of the first item automatically appears in a Preview Panel to the right of the table with

any mutable fields set to the row's parameters. The interface features Ext JS buttons to allow the user to control the applications. Those elements use various event listeners to update its interface dynamically. When directed by the user, the interface would generate Ajax requests to direct the back-end to perform certain actions. These actions could simply update the interface or control the laser marker and what gets loaded into it. The functions would then carry out their respective actions until the system has been updated or the next user input is required. Importantly among these actions is the ability to choose which jobs to send to the laser marker. This is done by either choosing to mark the first job in the drag and drop grid or choosing to mark all available jobs. This lets the user be flexible in what they want to achieve and how.

Back-End Architecture

The backend of the system handles all requests from the front-end. One such request is if the front end requests a preview, the back-end will load the correct template to the laser marker, set its parameters, and request the preview. The resulting image will then be passed back to the front-end, where it can be displayed to the user.

Interaction with the laser marker is done through a TCP/IP interface. This interface was designed to accept GET parameters passed to specific files at its IP address. To facilitate using this interface, the back-end also provides an API wrapper, allowing requests to be forwarded to any marker made by the same manufacturer. This API wrapper uses the *Templates*, *Parameters*, and *Job Info* tables to validate that all necessary data is present and that the provided data is safe to use (I.E. a user is not attempting to inject code to the server). All of the requests to the hardware are stored in their own control class that could be reused for any projects that require interaction with the laser marker. The back-end also handles all interactions with Datava's database. Though the feature was not able to be implemented in time, the back-end would do tasks such as logging what marks took place, at what time, and by whom. For this purpose, several tables were created to hold all necessary information.

These tables were arranged as displayed in the figure below. Each box represents a separate table, where its name is in bold, and each column is the text below it. The *Templates* table is required to hold any information about templates that could be used for the laser marker. Each template has a foreign key to the *Parameters* table, where its parameters are stored. These two tables allow multiple templates with different parameters to both work in the system. A *Job Info* table holds the currently selected jobs that need to be marked. This table has a foreign key referencing the templates table to match jobs with their required template. The *Log* table holds information about what jobs were done and when to let Purakal better keep track of their daily work. This table has a foreign key to the "Job Info" table for the job the log entry is referencing. Lastly, we have a *Lasers* table that is responsible for holding each laser marker's name and address within the network. Figure 2 shows a model of the relevant tables in the database.

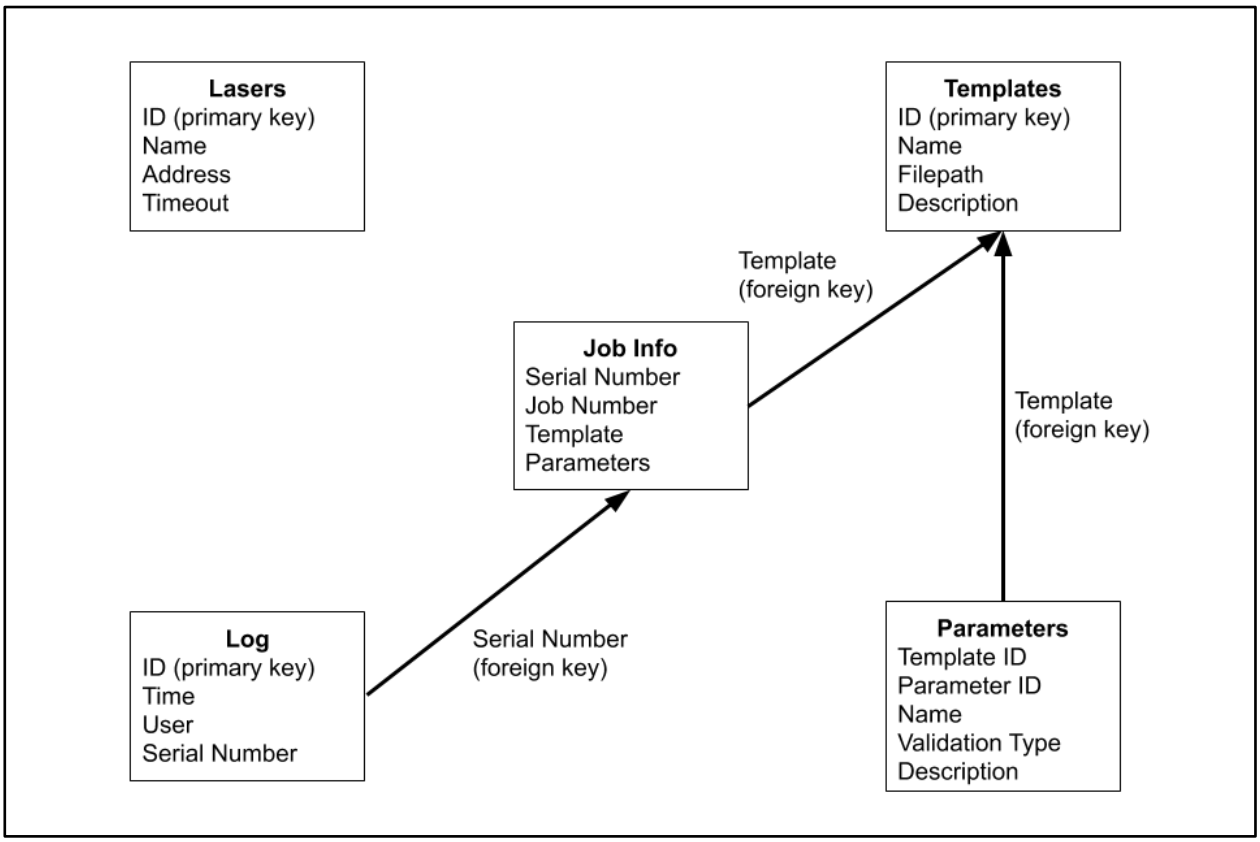

Figure 2: Database Model

4. Technical Design

Interacting with Hardware

The focus of this project was a laser marker manufactured by TykmaLaser. This laser marker and its software allow parameters to be set using a TCP/IP interface to load parameters and query for the marker's status. This TCP/IP interface is actually built as an HTTP server, such that specific 'files' direct the marker to perform certain actions, and additional parameters are passed as GET variables (The variables passed to a web server after the question mark '?' in a URL). Unfortunately, using these directly is both difficult and possibly unsafe. Therefore, an interface was built in PHP to manage communication with the marker.

This layer of abstraction was built to take in parameters from an Ajax request and route that information to the correct API function. This API is a thin wrapper on all of the marker's functions, allowing them to be called from PHP functions. On each request received from the user, a new control object is created. This control object contains the rest of the necessary information to generate a call to the laser marker (IP address and timeout). After the creation of that object, cURL is used to call the corresponding action on the laser marker and retrieve its response. Each call may generate a different type of response. For example, some calls request

the status of the machine, while others set a template to be used, and others retrieve the parameters that are currently set. All of these responses are then packaged up in JSON objects, either containing an error or the result of a query and passed back to the client. Overall, the system is designed as shown in Figure 3.

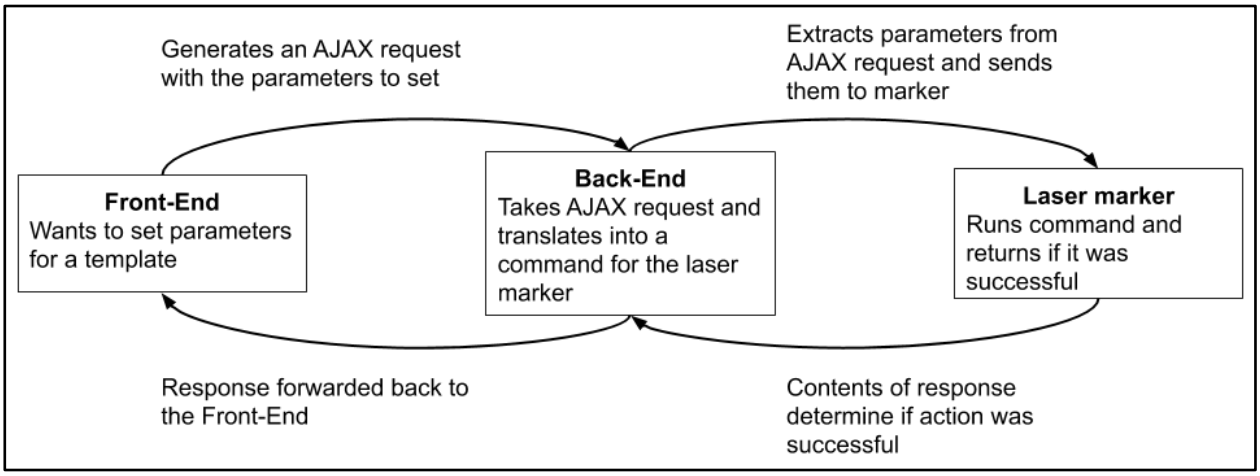

Figure 3: Information Flow Diagram

UI Design with Ext JS

The front-end also proved to have technical challenges with designing the UI. Datava's existing documentation and applications used a specific version of Ext JS for User Interfaces. Keeping with the client requests and consistency, we had to make sure that all the design and technical features in the front-end followed the Ext JS classes and formats. The Ext JS version documentation was very helpful in describing different features and how we can implement them. The trouble lied mainly in inexperience with JavaScript. We had a design plan to follow the Ext JS documentation but had to change most of the plan because of the way JavaScript works and handles code.

These design changes we had to make were to ensure robust communication with the back-end. Waiting for returns from the back-end was a significant issue with this communication. The UI had to be loaded in a way that didn't rely on the initial result value of Ajax requests, as Ajax requests are asynchronous. However, the requests could load data in the background by using callbacks. An example of this would be the Preview Panel. Since we wanted the image shown to change depending on which part was first in the completion order, an Ajax request and response had to be made for the Base64 encoded image. In our original plan, we made the code so that when the table refreshes after a change, the image would load in the same function. We learned that designing it this way would cause delays because of the way Ajax requests are loaded in JavaScript. The code we ended up using loaded the images through a callback, allowing the image to appear whenever the result came back. This allowed the interface, shown in Figure 4, to refresh and load while working within the asynchronous model of Ajax requests.

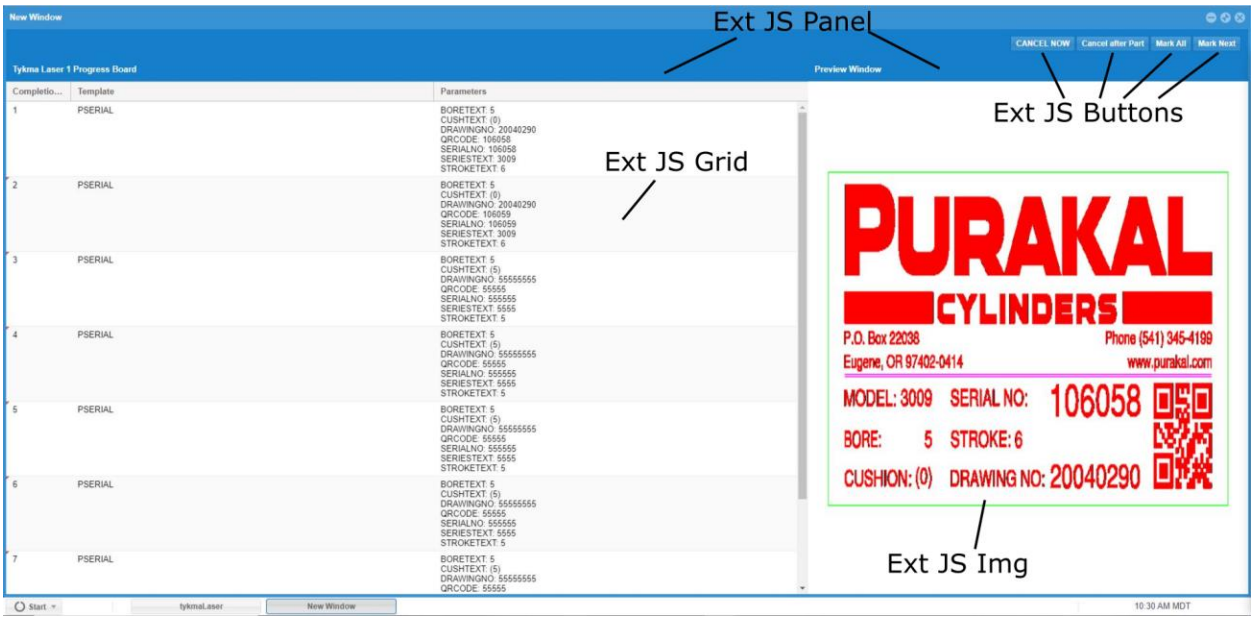

Figure 4: User Interface

5. Quality Assurance

Code Reviews

Throughout our development process, we used informal code reviews with our main contact from Datava to ensure we were writing higher quality code. These reviews helped us to better understand security implications, standard practices, and the expected structure of our project based on an existing codebase. As such, code reviews often highlighted necessary revisions to code architecture and prompted refactoring of the system.

Our main reason for doing code reviews was to ensure that our code meets quality expectations for applications built within Datava's system. However, code reviews also allowed Datava to guide development so that it reflects their latest decisions regarding their codebase. This prevented the unintentional use of legacy code practices used by older applications, allowing them to move the codebase in the desired direction.

In addition, having our code reviewed by someone more familiar with Datava's codebase and web development as a whole allowed us to more easily debug and prevent potential issues.

User Acceptance Testing

Throughout our time working with Datava, we have had weekly meetings with Purakal Cylinders, the end-user of our application. The main goal of these meetings was to learn more about their workflow and discuss the direction our development was going. This allowed us to get initial feedback from the end-user about what they do and don't want from the application and gave us a chance to discuss design decisions that had not previously come up. In addition,

we demonstrated our product to Purakal so that they could verify the application matches their product vision.

6. Results

Unimplemented Features and Future Work

Due to time constraints, some features were dropped from the project but are still planned to be implemented in the future by Datava. Firstly, usage information is not currently being logged to the database. The tables are set up to allow logging, but the backend is not able to handle it yet. Secondly, the QR scanning functionality was dropped to focus on polishing the interactions between our application and the marker. However, adding QR code integration should be relatively easy with the existing project structure.

Summary of Testing

Our team primarily used Google Chrome, Safari, and Firefox to access our web application. Since our application is based inside Datava's existing system, much of testing that aspect is out of our hands. Communication with the server, and by extension the laser marker, was tested using debugging tools in both Google Chrome and Visual Studio Code. The client-side UI elements were also tested using Google Chrome's debugging tools.

Results of Usability Tests

For our Usability Tests, we had meetings with both clients to get a list of wants for the project and put together a list of TODOs based on those wants. This included items like an interactable table to custom load parts, a preview of the template that will be marked, buttons to mark the next part or all parts, and buttons to cancel the process for safety and accuracy of the parts. However, in order for all of these UI functions to work, the back-end needed to be set up in a way that allowed for direct communication with the laser marker. This includes proper connections and code that the laser can understand and implement.

The conclusion of this test proved that our product would be a much-appreciated efficiency item for our client. While the client was not able to test the application before this report due to VPN issues, the main goals that were agreed upon were met. Our other client was able to test the functionalities of the back-end as he was able to see that the product not only communicates with the front-end and laser but is also robust.

Lessons Learned

1. It is not necessary to understand an entire codebase to add to it. This project was developed within Datava's existing codebase, but we didn't need to know how the entire system worked to create a new application within it.
2. Behavior and assumptions are not always well-documented. Multiple sections of the TCP/IP interface documentation contained incorrect information, which is why testing is important.

3. Always have a good understanding of available libraries before writing code. Knowing about helpful functions other people have written prevents you from reinventing the wheel and speeds up development.
4. Frameworks used by the client can sometimes be more difficult to learn than the languages they are written for. For example, Ext JS was much more complicated than JavaScript.