Tech Review

for

Boxeur - Case Designer

Prepared by

Evan Hopper-Moore

Oregon State University

CS 46X - Senior Software Engineer Project

Kirsten Winters

Scott Fairbanks

November 8th, 2019

Abstract

The purpose of this document is to outline some options for solutions for the web interface of Boxeur, a tool for students to create 3D enclosures. The options investigated include for the choice of styling language, architectural style, and web framework to be used. Along with a short analysis and comparison of options, recommendations for final choices of technologies are included to be considered by the Boxeur team.

CONTENTS

1 Introduction 3

2 Technologies 3

2.1 Styling Language . . . 3

2.1.1 Description . . . 3

2.1.2 Options . . . 3

2.1.3 CSS . . . 3

2.1.4 LESS . . . 3

2.1.5 SCSS . . . 4

2.1.6 Comparison . . . 4

2.1.7 Recommendation . . . 4

2.2 Architectural Style . . . 5

2.2.1 Description . . . 5

2.2.2 Options . . . 5

2.2.3 REST API . . . 5

2.2.4 SOAP . . . 5

2.2.5 Comparison . . . 5

2.2.6 Recommendation . . . 6

2.3 Framework Choice . . . 6

2.3.1 Description . . . 6

2.3.2 Options . . . 6

2.3.3 Django . . . 6

2.3.4 Node.js . . . 6

2.3.5 Comparison . . . 7

2.3.6 Recommendation . . . 7

References 8

1 Introduction

Boxeur is a web based tool for designing 3D enclosures for fabrication through 3D printing and laser cutting. The website's interface will be simplistic in design, only spanning a few pages for creation of cases, logging in, and recovering previously saved sessions. Because this project is not necessarily large in nature, I have placed value in finding lightweight solutions that still offer robust technology to support the required features.

2 Technologies

2.1 Styling Language

2.1.1 Description

CSS is the basis for almost all website styling and is a core element of web design. There are a few alternatives that offer extra features on top of CSS that could result in an easier process for our group. These extension languages could be worth using to keep our project concise, readable, and scalable.

2.1.2 Options

1) CSS
2) LESS
3) SCSS

2.1.3 CSS

Cascade Styling Sheets (CSS) is the basis for styling in web development and describes how HTML elements and content should look and behave [1]. CSS solved the issue of inline styling in HTML which was not scalabale as it was hard to manage on larger sites. All browsers support CSS as it is now the standard language for styling HTML. It's impossible to escape using CSS in modern day web development but there are some language extensions that add more features on top of and compile down to CSS.

2.1.4 LESS

Leaner Style Sheets (LESS) is a "backwards compatible language extension for CSS" [2]. LESS improves on CSS mainly by adding the capability for nesting. Nesting leads to cleaner and easy to read code as the more standard curly-brace blocks are made possible. LESS also adds the capability for variables, functions, mixins and importing other `.less` files. Variables make it easier to maintain a website, for example a color used for buttons across the site could be changed in one line without having to search and replace in multiple files. Mixins provide similar functionality but with chunks of code, making it faster to write code that instead would have to be copied in multiple places. Mixins could be especially useful with browser specific properties (ie `-moz-*` and `-webkit-*`). LESS files are compiled into CSS either at run time on the browser using JavaScript or using a Node.js command line tool. Depending on what framework we choose for the site we could use either method.

2.1.5 SCSS

Sassy Cascade Styling Sheets (SCSS), also known as Sass, is ”the most mature, stable, and powerful professional grade CSS extension language in the world” [5]. It is based on Ruby and compiles down to CSS, but unlike LESS it cannot be compiled in the browser. SCSS provides the same features of LESS such as variables, nesting, mixins, and functions while also supporting more functionality such as loops and control directives. Control directives in SCSS add functionality for the traditional statements `if/else`, `for`, `each`, and `while` which can also increase scalability and readability in our stylesheets.

2.1.6 Comparison

CSS is the basis for styling HTML across the web and is supported in all browsers. However, CSS can be improved upon by using CSS extension languages such as LESS and SCSS which compile down to CSS either in the browser or manually on the command line. The differences between LESS and SCSS are slight but it seems like SCSS is more popular, especially as it was just adopted as the base for Bootstrap 4 [4]. Both languages offer similar functionality such as nesting, variables, mixins, and functions but SCSS adds a few more features such as control directives. Even though LESS has less functionality than SCSS, it can be compiled using a JavaScript file included in the in HTML. This also means that when writing in LESS, every time the page is reloaded the most current styling is used as opposed to SCSS which could be out of date and could lead to confusion with team members.

2.1.7 Recommendation

CSS is certainly capable of doing anything we would need for our project but the added benefits of LESS or SCSS would make our project easier to read and write. SCSS boasts more features than LESS, however it requires the extra work of installing Ruby in our project. The language I recommend we use is LESS because it has less overhead than SCSS as it simply requires including a JavaScript script, but still supports the main features of nesting, mixins and more.

2.2 Architectural Style

2.2.1 Description

Throughout the implementation of our project we will have to deal with sending data back to the server side of our application, for example when implementing a log-in feature or fetching case data for restoring sessions. There are a few different popular methods to design our API and how we retrieve and send data to the server.

2.2.2 Options

1) REST API
2) SOAP

2.2.3 REST API

The Representational State Transfer(REST) API is an architectural style that defines how data is sent and the interaction between the server-side and the client-side [6]. A RESTful design for an API is based on using Uniform Resource Identifiers (URI) and the JSON data type to efficiently transfer data. JSON is a highly browser compatible data type which can be interpreted easily by JavaScript. REST APIs focus on simplicity by using the standard HTML protocols and keeping the interaction stateless, meaning all of the information required to get or send data is included in the request. REST API defines other principles such as having chacheable data and a hierarchical structure which can increase speed and security, respectively.

2.2.4 SOAP

Simple Object Access Protocol (SOAP) is a standards-based Web services access protocol which relies on XML to transfer messages. Developed by Microsoft, it fixed a lot of problems that developers had transferring data on the internet in its early days and became fairly standard in API designs. The SOAP envelope is a definition for how XML data is organized. The envelope contains headers, which can contain information such as credentials or definitions of the data type in the message, and a body block which contains the actual message. Using this structure, SOAP also provides built in security features through its standard called WS-Security, ideal for enterprise level usage.

2.2.5 Comparison

Because of SOAPs design structure, naturally it requires more overhead to support the XML envelope's headers and security protocols while the REST API values a slimmer design. This translates to using more bandwidth when using SOAP. REST API's architecture also supports caching and is widely known for excellent performance and scalability.

2.2.6 Recommendation

My recommendation between the two approaches for sending and receiving data is the RESTful API structure. Our project most likely won't need the extra security provided by SOAP, so it seems the lightweight choice of REST is right for us. We can use the guidelines in the REST API description on their website to keep our architecture clean and efficient while minimizing bandwidth usage.

2.3 Framework Choice

2.3.1 Description

Web Frameworks are APIs or software designed to support the deployment and creation of web services. For our project we need a framework that can support rendering dynamic content from databases for planned functionality such as restoring sessions, pulling user created case models, and publishing new models for sharing.

2.3.2 Options

1) Django
2) Node.js

2.3.3 Django

Django is a "high-level Python Web framework that encourages rapid development and clean, pragmatic design" [3]. Django follows a design pattern called Model Template View (MTV) that acts as a guideline for clean design. The models act as the data access layer and handle validating, interacting with, and relating data. The template layer, also referred to as the presentation layer, handles what is displayed on the page. The view layer is also called the business layer as it accesses models and populates templates, acting as the connection between the two layers. The MTV layout guides projects into a clean design with well separated layers. This high level design can also limit us in a way because we have less control over how we design our application.

2.3.4 Node.js

Node.js is a "JavaScript runtime [for building] scalable network applications" [7]. Node.js functions similar to a web server and lets developers define exacly how files are served, making it as lightweight as the developers define it to be. Node.js uses an event-driven model and focuses on controlling processes through callbacks in JavaScript. To handle connections, Node uses events to spawn new processes for connections and follows the defined instructions for serving files, making a simple one-file server possible.

2.3.5 Comparison

Django has a fair amount of overhead and setup required compared to Node.js, however, it does automatically support a MTV design pattern. In Node.js we would have to set up our own database access design and populating templates with content could be a hassle. Node.js, however, is integrated closely with JavaScript which makes accessing and changing HTML elements easy.

2.3.6 Recommendation

Because Node.js uses lower level design compared to Django, we are given the choice of just how robust the server will be. There is very little overhead involved in starting a project in Node.js and for Boxeur, which only has a handful of pages, I believe the correct framework choice is Node.js.

References

[1] w3schools.com. ”CSS Introduction.” W3Schools. https://www.w3schools.com/css/css_intro.asp

[2] The Core Less Team. ”Overview.” Getting Started - Less.js, http://lesscss.org/.

[3] Django Software Foundation. ”The Web Framework for Perfectionists with Deadlines.” Django, https://www.djangoproject.com/.

[4] Otto, Mark, and Jacob Thornton. ”Introduction.” Bootstrap, https://getbootstrap.com/docs/4.0/getting-started/introduction/.

[5] Catlin, Hampton, et al. “CSS with Superpowers.” Sass, https://sass-lang.com/.

[6] Fielding, Roy Thomas. ”REST API Tutorial.” Learn REST, https://restfulapi.net/.

[7] Node.js Foundation. “About Node.js.” Node.js, https://nodejs.org/en/about/.