<table>
<thead>
<tr>
<th><strong>Authors:</strong></th>
<th>William Vuong, Michael Long</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client:</strong></td>
<td>Steven D. Sheetz</td>
</tr>
<tr>
<td><strong>Professor:</strong></td>
<td>Edward A. Fox</td>
</tr>
<tr>
<td><strong>Course:</strong></td>
<td>CS 4624 Spring 2015</td>
</tr>
<tr>
<td><strong>Version:</strong></td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Location:</strong></td>
<td>Blacksburg, VA, USA</td>
</tr>
</tbody>
</table>

## Project Documentation

For **Grickit – Movie Decision Making Tool**
Executive Summary

This is the final report for Grickit, the Movie Decision Making tool created by William Vuong and Michael Long with the guidance of Dr. Steven D. Sheetz, the client. This final report will consist of three main sections: the user’s manual, the developer’s manual, and our lessons learned.

The user’s manual will be a guide for a person who intends to utilize the Movie Grickit that was created and wants more information about the Grickit and its purpose. Included is a general overview as well as in depth look at the roles available for users of the Grickit system. We will explore how each user affects the Grickit system to ultimately calculate and display the intended results. Included are screenshots so that the reader can easily identify what features we are referencing in this final report.

The developer’s manual will be a guide for future developers to understand our design choices and the structure of our Grickit implementation. We start with a high level overview of the technical workflow. It will also feature extensive information about the RESTful web API created to store the Grickits and handle calculations. We will discuss the various technology choices utilized in creating this API. The accompanying live documentation will be discussed and shown to the developer example use cases. Validation of user entered data and regression testing were done to ensure a good user experience.

The developer’s manual also contains information about the front implementation. This section starts with the general design and thought process for creating the Movie Grickit. We will go in depth about what technologies were chosen and how they were utilized. Examples will be shown with code snippets about how the technology is used in the scope of this project. Interesting considerations taken with the creation of the front end have been noted and discussed. Included throughout the developer’s manual are screenshots to give a visual representation of the design choices.

The lessons learned section contains the issues was came across while developing this tool and our solutions for them. Our main issues arose when deploying on to server as required by our client. The hardware was older (Windows Server 2003) and therefore we had versioning issues with the technologies we were using and their minimum OS supported. Our database solution required a newer OS, so we pivoted to a different storage solution, that utilizes a similar noSQL structure but data is stored in a file. This is not the optimal solution but was appropriate given the circumstances. We also had issues with the website needing to be retrieved through HTTPS. Therefore any calls using HTTP had to routed through our HTTPS server.

Our choice in separation of responsibilities for the AHP algorithm and our presentation layer were due the goal of later producing a generalized Grickit that does not depend on a specific domain set. Our back end solution has no notion of the movie data in our Movie Grickit. We suggest that modifications are made on top of our solution instead of starting from scratch when expanding, as our back end will be a good base for when a generalized front end is created.

The Movie Grickit was an interesting project that had many considerations to take note of in both the back end handling of the AHP algorithm and data storage as well as the front end user experience and design choices. Considerations were kept in mind for the future generalized solution with the back end already ready to support a generalized solution. We were able to deploy on the intended hardware despite some minor obstacles. Overall the client was happy with the end result and will be moving forward with the project to expand its use cases.
# Figure Table

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Grickit Home Page</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>“Mission Impossible” Query</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>Final Alternative Selection Screen</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Generate Grickit Button</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>Link to current Grickit and Facebook/Twitter sharing.</td>
<td>6</td>
</tr>
<tr>
<td>1.6</td>
<td>Facebook Sharing page</td>
<td>6</td>
</tr>
<tr>
<td>1.7</td>
<td>Twitter sharing page</td>
<td>7</td>
</tr>
<tr>
<td>1.8</td>
<td>Top Results</td>
<td>7</td>
</tr>
<tr>
<td>1.9</td>
<td>All Results</td>
<td>8</td>
</tr>
<tr>
<td>2.0</td>
<td>Weighting pairs section</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>High-Level Grickit Workflow</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>A list of all of the implemented API Routes from the live documentation webpage</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>An example route from the Swagger UI</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Code Coverage output from npm test</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>App.js file</td>
<td>17</td>
</tr>
<tr>
<td>2.6</td>
<td>creator_view.html</td>
<td>18</td>
</tr>
<tr>
<td>2.7</td>
<td>weighting_view.html Results section</td>
<td>19</td>
</tr>
<tr>
<td>2.8</td>
<td>weighting_view.html pair weighting section</td>
<td>20</td>
</tr>
</tbody>
</table>
# Table of Contents

**User’s Manual**

1. Introduction  
   1.1 Purpose  
   1.2 Overview  
2. Grickit Creation  
   2.1 Alternative Selection  
   2.2 Criteria Selection  
   2.3 Grickit Generation  
3. Grickit Sharing  
   3.1 Via Facebook and Twitter  
4. Grickit Weighting  
   4.1 Weighting Results  
   4.2 Weighting Criteria/Alternatives

**Developer’s Manual**

1. Introduction  
   1.1 Purpose  
   1.2 Overview of the Grickit Project  
   1.3 High-Level Workflow  
2. Back-end Implementation  
   2.1 REST API  
   2.2 Technology  
   2.3 Design Overview  
   2.4 AHP Implementation  
   2.5 Live Documentation  
   2.6 GET Routes  
   2.7 POST Route  
   2.8 PUT Routes  
   2.9 DELETE Routes  
   2.10 Joi Validation  
   2.11 Regression Testing  
   2.12 Back-end Deployment Changes  
3. Front-end Implementation  
   3.1 General Design  
   3.2 Technology Choices  
   3.3 Angular JS MVC

**Lessons Learned**

1. Introduction  
   1.1 Purpose  
2. Software Versions & Deployment  
   2.1 Hardware Issues  
3. Security  
   3.1 SSL Certificates

**Acknowledgements**

1. Client
1. Introduction

1.1 Purpose
The purpose of this document is to give a potential user the information necessary to effectively use the Grickit system that has been created for this project. This will entail the selection of criteria and alternatives, the sharing of a created Grickit, and the completion of the weighting of criteria and alternatives.

1.2 Overview
This document will follow a user as they go from Grickit creation to the completion of the process. The user moves through two distinct roles, starting as the “Master User” of a particular Grickit, where they will select the base criteria and alternatives that will be weighted by everyone that they invite. Then, they will transfer to the “Weighting User” role, and every other person invited using the social functionality will also assume this role. More detail on these roles can be found in the Front-End section of the Developer’s Manual.

2. Grickit Creation

2.1 Alternative Selection
Upon arrival on the Grickit homepage, the user will be greeted with the following screen.

![Grickit Home Page](image)

The user’s first action is to select a group of alternatives, of which there must be at least 2. To do this, they would search a term related to the movie they intend to selection. For example, if our intended movie is named “Mission Impossible”, we type that and hit the enter key or search button.
Fig 1.2: “Mission Impossible” Query

The user is presented with the above results list, and selects the trilogy of Mission Impossible movies. An example of the final view for alternative selection is shown below.

Fig 1.3: Final Alternative Selection Screen

2.2 Criteria Selection

The user can then select the criteria that will be weighed against for the Grickit they are created. This can be simply done by selecting the criteria that they wish to use until all those that are desired are highlighted, as shown below.

Fig 1.4: Criteria Selection
2.3 Grickit Generation

Finally, the user can click the Generate Grickit button, as shown below, and they will be redirected to the weighting page, where they can share the Grickit and complete the weighting themselves. Should the user want to reset the selected alternatives and criteria, the Reset button is also conveniently located here.

![Generate Grickit Button](image)

Fig 1.4: Generate Grickit Button

3. Grickit Sharing

3.1 Via Facebook and Twitter

Once the user has clicked the Generate Grickit button, the web application will redirect them to the weighting page where the user can share the Grickit to others via multiple social networks. Grickit has integrated support for Facebook and Twitter. See figure 2.1 below.

![Grickit Link](image)

Figure 1.5: Link to current Grickit and Facebook/Twitter sharing.

The user can choose to copy the link in the read only text box and share that via whatever medium they desire. Selecting the Facebook button will automatically utilize the current logged in Facebook user and prepare a status post that contains the Grickit link. The user can then choose to share to a subset of their Facebook friends, their timeline, or in a private message. See figure 2.2 below.

![Facebook Sharing](image)

Figure 1.6: Facebook Sharing page

Sharing with Twitter is a similar situation. The user is brought to a page with a prefilled message allowing them to tweet out the link to their followers and the general public. See figure 2.3 below.
4. **Grickit Weighting**

4.1 **Weighting Results**

When creating a new Grickit there will be no results displayed to the user, but if the user is viewing a Grickit that was shared to them, results could already exist based on previous users weighting. If results do exist the top three will be shown to the user with the best matched result in the middle. See figure 3.1 below.

**Top Results**

![Top Results](image)

*Figure 1.8: Top Results*
Also there is a link to show all results below the top three, which is useful for when the user has selected a large amount of alternatives to compare to each other. This section includes the top three results as well, with a column showing the match percentage. See figure 3.2 below.

Figure 1.9: All Results

<table>
<thead>
<tr>
<th>Poster Year</th>
<th>Title</th>
<th>Abridged Cast</th>
<th>Critic</th>
<th>Audience</th>
<th>Runtime</th>
<th>Rating</th>
<th>Match Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>Terminator Genisys</td>
<td>Emilia Clarke, Jason Clarke, and others</td>
<td>-1%</td>
<td>98%</td>
<td>min</td>
<td>Unrated</td>
<td>50%</td>
</tr>
<tr>
<td>2009</td>
<td>Terminator Salvation</td>
<td>Christian Bale, Sam Worthington, and others</td>
<td>33%</td>
<td>54%</td>
<td>115 min</td>
<td>PG-13</td>
<td>9%</td>
</tr>
<tr>
<td>1991</td>
<td>Terminator 2: Judgment Day</td>
<td>Arnold Schwarzenegger, Linda Hamilton, and others</td>
<td>92%</td>
<td>94%</td>
<td>150 min</td>
<td>R</td>
<td>16%</td>
</tr>
<tr>
<td>1984</td>
<td>The Terminator</td>
<td>Arnold Schwarzenegger, Linda Hamilton, and others</td>
<td>100%</td>
<td>88%</td>
<td>108 min</td>
<td>R</td>
<td>17%</td>
</tr>
<tr>
<td>2003</td>
<td>Terminator 3 - Rise of the Machines</td>
<td>Arnold Schwarzenegger, Nick Stahl, and others</td>
<td>70%</td>
<td>47%</td>
<td>109 min</td>
<td>R</td>
<td>6%</td>
</tr>
</tbody>
</table>

4.2 Weighting Criteria/Alternatives

Below the results section is where the user can weight the alternatives and criteria. A tab interface is used to separate the two types to be weighted. On the left we have a table of contents that show the current pair being weighted and the others as well. A checkmark will denote a pair that as been weighted already. The right shows the actual pair to be weighted with a slider where the user can indicate their preference as strong or neutral. Included are buttons to go to the next and previous pair. There is also an auto advance checkbox to allow the next pair to be shown as soon as the current pair is weighted. See figure 3.3 below.
1. Introduction

1.1 Purpose
The purpose of this document is to give detailed instructions to future developers whose goal is to either extend this project or to branch off into new versions of the Grickit system. This includes extensive details of how to leverage the back-end implementation in order to create new Grickit clients.

1.2 Overview of the Grickit Project
The Grickit project as a whole is larger than the piece of software created from this particular project. The idea of a Grickit is a simple piece of technology that anyone can use in order to make a group decision based on a set of well-defined options. Notice that there is no restriction put on the type of decision that is able to be made with a Grickit. The goal of this project is to develop a fully functioning Grickit for one type of decision, that being the decision of what movie to choose as a group. The accompanying goal is to provide a clear path from the software created in this project to the full generalized Grickit software. This will be accomplished through the explanation of the generalized back-end solution, our example UI, as well as each of their sub-sections.

1.3 High-Level Workflow
The following is a very high level workflow of how a Grickit system may function. Each step below will be outlined in greater detail in the rest of the document.

![High-Level Grickit Workflow Diagram](image)

2. **Back-end Implementation**

2.1 **REST API**

Based upon the reader’s familiarity with REST APIs, they may need to refer to some of the excellent online resources on the subject before proceeding with this documentation. One recommended document would be the “Pearson Higher-Education RESTful Best Practices”.

2.2 **Technology**

The technologies chosen for the API implementation are listed below:

- Node.js²
  - Hapi.js Server Framework³
  - Joi.js Input Validation
  - Lab.js Testing Framework
- MongoDB⁴ (Please see the Deployment Changes section for further information)

Node.js provides extremely flexible development, as well as extremely fast executing and connection handling.

Hapi³, Joi, and Lab are all part of a single set of frameworks developed by WalMart Labs, and we encourage future developers to explore their GitHub and external documentation.

MongoDB is the preferred noSQL implementation for the database, but as described in the Deployment Changes section, it was not possible for this to be used during this initial Grickit Project.
2.3 Design Overview
The Grickit server implementation consists of a collection of GET, POST, PUT, and DELETE routes that allow for the full Grickit functionality to be realized. Each route will be explained in detail in this document in order to provide full understanding of the system. Below is a simple overview of the available routes and a very brief description of each.

Grickit API Documentation
All public routes for the Movies Grickit API

<table>
<thead>
<tr>
<th>Method</th>
<th>Route</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/api/grickit</td>
<td>Gets the stored information for all grickits</td>
</tr>
<tr>
<td>POST</td>
<td>/api/grickit</td>
<td>Generates and returns a new unique ID for the Grickit, as well as stores the basic data</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/grickit</td>
<td>Deletes all of the current grickits from the database, this method is included for developer use</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/grickit/calcAltWeight</td>
<td>Calculate the alternative weights, and the resulting averages.</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/grickit/calcCritWeight</td>
<td>Calculate the criteria weights, and the resulting averages.</td>
</tr>
<tr>
<td>PUT</td>
<td>/api/grickit/calcFinalValues</td>
<td>Calculate the final values for each alternative.</td>
</tr>
<tr>
<td>GET</td>
<td>/api/grickit/[id]</td>
<td>Gets all data for a specific Grickit</td>
</tr>
<tr>
<td>DELETE</td>
<td>/api/grickit/[id]</td>
<td>Delete a particular grickit based on the supplied id, this method is included for developer use</td>
</tr>
</tbody>
</table>

Fig 2.2: A list of all of the implemented API Routes from the live documentation webpage

2.4 AHP Implementation
Prior to making any changes to the AHP calculation, a developer should ensure that they fully understand how the AHP calculation functions. Please refer to one of the many online resources that are available for more information on AHP and the given code for details of its implementation.

2.5 Live Documentation
Before going into each route, let’s outline the documentation that is available for the API. The developers of the Grickit project have provided a live documentation tool called Swagger\(^5\), which creates a webpage on the server’s domain which allows for live interaction with the different routes. This webpage is reached by going to http://{host}:{port}/documentation. For all of the routes that are detailed below, please see the Swagger\(^5\) documentation section for each as an accompaniment to the information given. An example of the page can be seen above in Figure 1.2, and an expanded route example is placed below.
2.6 GET Routes

2.6.1 `/api/grickit`
Returns an array that contains all of the database information for each of the currently active Grickits.

2.6.2 `/api/grickit/{id}`
Returns a JSON object containing all of the database information for a single Grickit, based on the ID provided.

2.7 POST Route

2.7.1 `/api/grickit`
Creates a new Grickit from the given criteria and alternative IDs. The payload should be formatted as follows:

```json
{
  "criteria": [
    0,
    1
  ],
  "alternatives": [
    123,
    321
  ]
}
```

Where the numbers 0 and 1 are the client’s IDs for the criteria in this particular Grickit. The numbers 123 and 321 are the client’s IDs for the alternatives that have been selected for this Grickit.

Fig 2.3: An example route from the Swagger UI.
The secondary function of this route is to create all of the AHP pairings for the criteria and alternatives, which are stored with the data object and retrieved via the GET on ID.

This call returns the new unique ID generated for the Grickit.

2.8 PUT Routes

2.8.1 /api/grickit/calcCritWeight
Calculates the criteria averages array for a Grickit, given all of the criteria weights, and stores it in the database.

```
{
    "grickitID": 0,
    "data": [
        {
            "criteriaIDs": [0, 1],
            "weight": 9
        }
    ]
}
```

The grickitID field is the unique ID returned by the POST, and the numbers 0 and 1 under criteriaIDs are the client’s IDs for the criteria in this particular Grickit that are being weighed against each other. The weight of 9 indicated the criteria 0 is strongly more favorable than criteria 1. A negative weight would be used to indicate that the second criteria is preferred.

This call returns the criteria averages array for the grickit.

2.8.2 /api/grickit/calcAltWeight
Calculates the alternative averages array for a Grickit, given all of the alternative weights for each criteria, and stores it in the database.

```
{
    "grickitID": 0,
    "data": [
        {
            "criteriaID": 0,
            "alternativeIDs": [0, 5],
            "weight": 5
        }
    ]
}
```
The grickitID field is the unique ID returned by the POST, and the numbers under the criteriaID field indicate which criteria the alternatives are being weighed on. The alternativeIDs field indicates the pair of alternatives being weighed. Finally, the weight indicates which alternative is preferred for the criteria stated. A negative weight would be used to indicate that the second alternative is preferred.

This call returns the alternative averages array for the grickit.

### 2.8.3 /api/calcFinalValues

Uses the stored criteria averages and alternative average (which are present for every separate user that has submitted these), in order to calculate the final output values for each of the alternatives, also based on how many users have input data.

```javascript
{
    "grickitID": 0
}
```

The grickitID field is the unique ID returned by the POST.

This call returns an array of objects that indicate each alternative id and its corresponding final value, which can then be interpreted as a percentage.

### 2.9 DELETE Routes

#### 2.9.1 /api/grickit

Deletes all of the active Grickits stored in the database.

Returns true on success, false on an error.

#### 2.9.2 /api/grickit/{id}

Deletes the Grickit whose unique ID matches the id field in the query.

Returns true on success, false on error.
### 2.10 Joi Validation
The Joi package is used for validating input to the routes, and will return to the user that their input is in an incorrect format before the request hits any of the code written for the project.

### 2.11 Regression Testing
There is a regression testing suite written with the Lab package which can be run with the command “npm test”. This test suite is currently not 100% code coverage, but is greater than 90%, and should be used as a guide, but not an absolutely measure of accuracy should any changes be made to the server implementation in the future. An example output from the testing suite is shown below.

```
> GrickitServer@0.0.0 test /home/michael/grickit/server
> lab -t 95

...........
10 tests complete
Test duration: 6590 ms
The following leaks were detected: data
Coverage: 91.86% (65/799)
server.js missing coverage on line(s): 177, 178
lib/helpers.js missing coverage on line(s): 5, 6, 9, 13, 14, 15, 16, 18, 23, 24, 25, 26, 27, 28, 31
lib/routes/createGrickit.js missing coverage on line(s): 85, 89, 92
lib/routes/grickitAlternatives.js missing coverage on line(s): 23, 39, 40, 41, 42, 43, 44, 75, 76, 80, 81, 89, 146, 147
lib/routes/grickitCriteria.js missing coverage on line(s): 30, 31, 32, 50, 51, 52, 56, 58, 59, 64, 111, 112
lib/routes/grickitData.js missing coverage on line(s): 38, 42, 43, 44
lib/routes/grickitDelete.js missing coverage on line(s): 16, 20, 64, 65, 66
lib/routes/grickitFinalCalc.js missing coverage on line(s): 18, 19, 22, 23, 28, 32, 49, 74, 99, 101
Code coverage below threshold: 91.86 < 95
```

*Fig 2.4: Code Coverage output from npm test*

### 2.12 Back-end Deployment Changes
During deployment, it became clear that the development that had been completed was not tuned to the hardware that the software would be deployed on. The deployment hardware was a server running a 32-bit version of Windows Server 2003.

#### 2.12.1 Database
The deployment system is not compatible with any currently available versions of MongoDB\(^4\), and therefore the developers chose to go with an extremely simplified noSQL database solution, using the locallyDB\(^6\) package. The developers would like to note that for larger scale hosting environments, they greater endorse the use of the much more robust and safe MongoDB\(^4\) package.

#### 2.12.2 Security
Due to the server being on the Virginia Tech network, it must be accessed over https using a very
specific SSL certificate that is available on the system. For more information regarding the HTTP implementation of the server, code histories can be found on the project’s GitHub.

The other change that needed to be made was due to an issue on the front-end. Due to no longer being able to make HTTP calls, and only HTTPS, we were forced to move the Rotten Tomatoes API calls to the server. This goes against our design patterns of creating a fully general solution. But, we believe that on any future implementation of a Grickit, this would not be necessary, so we are still considering our API the general solution. For more information regarding this, please see the Lessons Learned section of this report.

3. Front-end Implementation

3.1 General Design
The general design of the front-end implementation for Grickit was to create a minimal, simple, mobile responsive web application that allows users a good experience on a wide variety of devices. As mentioned earlier we created two clear roles for a Grickit user: ‘Master’ user and ‘Weighting’ user. The master user’s role is to create the Grickit by choosing the alternatives and criteria that will be utilized in the AHP algorithm. The weighting user’s role will to be weight each pair of criteria and alternatives specifying their preference for a specific criteria/alternative. We therefore have two pages in web application, one corresponding to the ‘master’ user and one for the ‘weighting user’. We will explore in depth the design choices and technologies utilized to create these pages.

3.2 Technology Choices
Our main technology choices include HTML5, CSS3, and JavaScript. We utilized the following frameworks Bootstrap 3, AngularJS, Angular UI Bootstrap, and Angular Material in order to aid us to create this web application. Bootstrap 3 is a mobile first framework with the design and functionality of many modern websites, giving you stylized forms and HTML5 elements to help as achieve the look and feel desired. AngularJS is a JavaScript framework to enable us to use Model-View-Controller architecture when dealing with our data and the views. AngularJS is an extremely popular tool and is general purpose and fit our needs very well. Angular UI Bootstrap and Angular Material are frameworks providing design/stylized elements that allow for easy integration into the AngularJS framework.

3.3 Angular JS MVC
The general separation of responsibilities with AngularJS leads to html files (view) and JavaScript files (controllers). We have two html files creator_view.html and weighting_view.html for each user role in Grickit. These files represent the page showed to the user for the Grickit creation and weight steps. Also included is a folder called ‘partial’ that contains the code for the navigation bar that is displayed on both the master and weighting page. This navbar_partil.html file is included into the creator_view.html and weighting_view.html file with AngularJS.

3.3.1 Controllers
We have three JavaScript files utilized by AngularJS: app.js, creator.js, weighting.js. App.js defines our angular application and includes the Angular UI Bootstrap and Angular Material frameworks. It is also here that we define our routes for accessing the two web pages. Routes not specifically described in app.js will redirect the user to the creator (home) page. When we define the routes we
also define which controllers will be handling the binding of the data to each respective view. See figure 2.4 below.

```javascript
var grickitApp = angular.module('grickit', ['ui.bootstrap', 'ngMaterial', 'ngAnimate', 'ngRoute']);
grickitApp.config(['$routeProvider', function($routeProvider) {
$routeProvider.
  when('/creator', {templateUrl: './views/creator_view.html', controller: 'CreatorController'}),
  when('/weighting/:grickitId', {templateUrl: './views/weighting_view.html', controller: 'WeightsController'}),
  when('/weighting', {templateUrl: './views/weighting_view.html', controller: 'WeightsController'}),
  otherwise({redirectTo: '/creator'})
});
```

Figure 2.5 App.js file

The creator.js file mainly contains the controller used by the creator_view.html page. The controller contains the various methods that are called based on user interaction with the website, which mainly include interaction with the movie search bar, adding/removing movies, and selecting the criteria. The functions used and their logic is fairly straightforward. We utilize arrays and JavaScript objects to keep track of the user’s selections. Once the user clicks the generate Grickit button, our server is hit and the Grickit is created. Also included in this file is an error modal controller, which is shown to the user if they try to create a Grickit with less than two movies or two criteria selected. Angular has a notion of directives, which are how you utilize code to modify the HTML page. The controller’s job is just to prepare data. It is not supposed to manipulate the web page styling and layout directly. Creator.js includes three custom directives: showHowItWorks, focusOn, searchResultsMonitor. showHowItWorks is used toggle the sizing of the panels on the creator page in order to give space for the education aspect of the Grickit system. FocusOn and searchResultsMonitor are used to know if the user’s last active element was in the movie search results, otherwise the search results will be closed when the user clicks out of it.

The weight.js file mainly contains the controller used by the weighting_view.html page. Its responsibilities are to load the Grickit data, and prepare each alternative/criteria pair as the user weights them. It is also has methods to set/display results and confirm if alternative/criteria weighting has completed so that the data can be sent to our server to calculate the new results. The controller includes many help functions in order to create the necessary text to display to the user based on each criteria/alternative pair. Notable functionality includes the weightSlider directive which is used to auto-advance to the next pair (if the auto advance checkbox is checked) after the user has weighted the current one.

### 3.3.2 Views

As mentioned before we have two main views the master and weighting view represented by the creator_view.html and weighting_view.html files. The creator_view.html utilizes Bootstrap’s grid
system to separate the parts of the page.

Figure 2.6: creator_view.html

An input field with two buttons attached to it via Bootstrap’s button groups is the search bar. Bootstrap’s panels are used for containing the selected movies table and the criteria to be selected. The search results utilizes absolute position to be displayed right underneath the search bar and over top existing elements of the page. AngularJS’s ng-repeat directive is utilized to create rows of movie data based on the array retrieved from hitting the Rotten Tomatoes API. The table of movies selected is also
using a ng-repeat to show the list of movies that were selected.

The weighting_view.html is more complicated utilizing a larger variety of Bootstrap\textsuperscript{10} components. At the top of the page is a similar read-only input field with two buttons for sharing the Grickit. Following that are two Bootstrap\textsuperscript{10} panels that contain the results for the Grickit and the pairs to be weighted. The results panel uses an ng-style directive, which allows us to set the height of the element based on angular variables. We can therefore correspond the elements height to its match percentage. The show results button opens a Bootstrap\textsuperscript{10} collapse element, which moves the elements below it to display all the results to the user. The all results list similarly uses a ng-repeat. We use two separate arrays to store the top three results vs all the results.

Our weighting panel consists of two tabs one for criteria weighting and one for alternative weighting, both share the same interfaces but different data sets. In the panel we have a table of contents on our left and the pair on our right. The table of contents features highlighting to denote the currently selected pair as well as a checkmark next to it if it has been weighted already. The right section of the weighting panel has our pairs and a slider indicating strong/neutral preference. There is no default value; this is done with the weightSlider directive, hiding the default value if our weight has not been set. Below the slider are a previous and next button and an auto-advance checkbox. The previous and next button use the angular methods to lead the previous/next weighting pair. The auto-advance checkbox enables the weightSlider directive to load the next weighting pair.

\begin{center}
\textbf{Top Results}
\end{center}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2_7.png}
\caption{weighting_view.html Results section}
\end{figure}
Lessons Learned

1. Introduction

1.1 Purpose
The purpose of this section of the document is to discuss some of the lessons that the developers of the project learned based on issues that arose during the development phase, and our solutions to them. We hope that these will provide insight to future developers and allow them to avoid the same mistakes.

2. Software Versions & Deployment

2.1 Hardware Issues
Very often, everything does not go as planned during the deployment phase of projects of this scale. Our deployment hardware, a Windows Server 2003 box, did not support the version of the MongoDB software that we had used throughout testing. Due to hardware/software upgrades not being an option, we had to redesign our database plan.

We went with the simplest solution possible, albeit not as robust as MongoDB. We use a simple package called locallyDB, which simply stores the Grickit information in a file as JSON objects. We understand that this solution is not optimal, but due to the time and hardware restraints it was a necessary change.
3. Security

3.1 SSL Certificates
Due to the stringent restrictions imposed on Virginia Tech servers, there are some very precise guidelines that must be followed by web servers in order to pass the security audits that are constantly done on all servers that are a part of the network.

In order to comply with these guidelines, we made our web server only available over https, and used the same SSL certificate that was in use by the other web server running on the hardware, albeit on a different port. These were fundamental changes to our server object within Hapi.js, and following a round of testing we were satisfied that they were successful.

On the front-end, we had to make changes to all of the HTTP calls, due to the restriction that all calls must be HTTPS. Rotten Tomatoes API, which provides all of our alternative data for the particular Grickit implementation, does not have HTTPS versions of its API calls. Therefore, we created an API route on our server that makes the Rotten Tomatoes API calls and returns that information to the client. This change broke our idea of the server being the generalized Grickit solution. But, any future Grickit implementations will only use the general routes, and not this one that is specific to the Movie Decision Grickit, so we are still considering our API the general Grickit solution.

Acknowledgements

1. Client
The Grickit idea and this particular project comes from our client, Dr. Steven D. Sheetz. Dr. Sheetz also provided the deployment hardware and maintains the codebase and associated accounts. His contact information is below.

Steven D Sheetz
Email: sheetz@vt.edu
Phone: 540-231-6096
Fax: 540-231-2511
Office: Pamplin 3080
References


