Dizplai Poll Web Application

Development Document

This development document will follow the implementation of the Technical Test proposed by Dizplai as part of the application process for their Graduate Software role.

By: Brad Jay Millington

Contents

[1. Understanding the Brief 3](#_Toc194348235)

[1.1 Web Pages 3](#_Toc194348236)

[1.2 Server 3](#_Toc194348237)

[2. Deciding on IDE, Languages, and Packages 3](#_Toc194348238)

[3. Implementing the API 3](#_Toc194348239)

[4. Designing the HTML pages 3](#_Toc194348240)

[5. Reviewing the brief questions 3](#_Toc194348241)

[6. Future Work 3](#_Toc194348242)

# Understanding the Brief

For this project, the brief provided clear requirements with examples and recommendations to aid understanding. The main core is a web application which allows a user to place their vote on a displayed poll, by picking out of a range of pre-selected options. They should also then be redirected to a page displaying the current votes by percentage for each option. The scope of the work can be broken down into two main sections.

## Web Pages

Two main web pages will be developed for the front-end user interface, separating the two different functions of the site. Both of these pages should be responsive, and function appropriately on a range of different device screen sizes.

### Voting Page

The voting page will provide the user with the polls question, as well as the range of two to five options available for the user to choose between, with the user receiving a visual confirmation of which choice they have selected before they submit their answer. It is important that the user is only able to select one option before they submit, if a second option is selected the first must be removed.

### Confirmation Page

After the user has selected their option on the voting page, they should be redirected to a confirmation or results page, which will give the results of the poll by percentage of votes for each choice, this should be truncated to have no decimal places. For this, the options should also be arranged from most to least popular from the top of the page to the bottom, arranged as a leaderboard column.

## Server

A server will be needed to process requests from the web pages, this will also hold the resources for the polls and their votes, and an example poll format has been provided. The server will expose a RESTful API and requires the ability to “GET” a poll, “POST” a vote to a poll, and “GET” votes for the poll by their id. It is also important that it is able to provide an accurate response for a range of different error scenarios.

# Implementing the Server

The server and API were implemented using the recommended node.js Express technology recommended in the brief. I chose this mainly as I have experience working with it before back in my first year of uni, where I found the routing features mostly intuitive and easy to use, especially when backed up with detailed documentation and countless other third-party online resources. I also learnt that express.js is used by many of the applications that I use today, such as Netflix, Uber, and LinkedIn, meaning it is a major technology which I would like to have more experience in for applying and working within the computing industry.

To test my implementation of the server I used the Postman API platform to send and receive requests using JSON within the body for the payload and responses. This allowed me to troubleshoot just the server code without also having the consider if the code I had written for the web pages were producing the erroneous results. I used this to test the implementation of the error handling within the API, such as when a user attempts to vote on a poll which doesn’t exist within the server. Images of this testing can be seen in “Appendix A”.

# Designing the HTML pages

I had decided very early on that the separated web pages which allow voting and viewing results were going to be designed in a very similar way and make use of the same CSS code. This decision was mainly down to the time constraints I had through this project due to other professional responsibilities. One feature I wanted to ensure was provided on my webpages is the ability to produce a correctly formatted page no matter the number of options provided on the poll. This was implemented using ‘for each’ loops which iterated through each option in the JSON response provided by the API, and generated the relevant HTML code for each using the ‘create Element’ function.

I used the Google Chrome ‘developer tools’ feature in order to test my web pages responsiveness to different screen requirements, this allowed me to fine tune my flexbox settings to ensure that all content was formatted and visible within the screen area on any type of device. Images of this can be seen in “Appendix B”.

# Security Problems

There are a range of security problems which could be faced if this project were deployed its current state, these are down to the implementation of the RESTful API which lacks security measures.

## Authentication

Currently, if the server was hosted online, anyone that knew the address would be able to send and receive requests to the server, such as I did in my testing with the postman application. This means that anyone at all would have the ability to access the poll results using ‘GET’ requests, or even ‘POST’ their own answers. This could lead to any important data being collected being made public, as well as allowing random users manipulate the results of any given poll by sending repeat requests.   
This could be fixed with the proper authentication methods, such as OAuth, which uses access tokens to determine if access to data should be permitted.

## Rate Limiting

Should the server receive a large number of requests, it may cause it to slow down or become completely unusable, resulting in the site no longer processing any requests. This could be done by either a very large and unexpected increase in site traffic, or by a single user planning to manipulate poll results or carrying out a deliberate Distributed Denial of Service (DDoS) attack.

A single user or IP address should be limited to the number of requests which they are able to send to the server, preventing them being able to affect the performance of the site. In this project this could be achieved using the Express middleware package ‘express-rate-limit’.

# Future Work

There is a lot of work that I would still like to put into this project, These plans include:

* Adding another webpage which allows the user to select from a list of poll questions to participate in.
* Preventing a user from being able to participate in the same poll more than once.
* Store the polls and results within a MySQL database allowing persistence in.
* Adding functionality to the events screen which gives a visual representation of each options percentage of current votes.
* Adding fallback page for if contact to the server fails which asks the user to try again soon.

# Appendix A

A screenshot of a computer

AI-generated content may be incorrect.

Figure A valid vote being sent to the server, and receiving a correct 200 OK response

A screenshot of a computer

AI-generated content may be incorrect.

Figure An invalid vote being sent to the server, and receiving a 404 not found response with custom error message

A screenshot of a phone

AI-generated content may be incorrect.

Figure The layout of the index webpage when viewed on a small viewport such as iPhone SE

A screenshot of a computer

AI-generated content may be incorrect.

Figure The layout of the results webpage when viewed on a mobile device with rotation functionality.