Documentation

Design:

This project implements a system to capture voting data from an HTML form, validate those data with JavaScript regex and use PHP to allow interaction between these data and a MySQL database. CSS was used to implement some of the required form validation user interface functionality.

How it works:

The user navigates to the HTML form. There, they are prompted to fill in the fields relevant to collecting voting data. After filling out the form, the user clicks the "Submit" button.

This calls the JavaScript validation functionality. The data are checked to ensure that:

- there are no empty fields
- all integer fields in fact contain integers and not other data types
- the polling station field contains exclusively alphanumeric characters
- the total votes field value is equal to the sum of those for candidate 1, candidate 2 and the rejected votes

If at least one of the above criteria is not met, the form does not get submitted. Instead, the field(s) with the error is/are highlighted in red, and the user is able to edit what they entered to fix the error. The user may click submit again, and if any errors persist, this process continues until the data are all valid.

When all fields are valid and the user clicks "Submit", the JavaScript validation is passed, so the HTML form calls PHP code to communicate with the MySQL database. First, the form sends the data to the PHP code using the "POST" method. The PHP code validates that the POST method is used, checks if the form contains data, and stores each form field in a variable. The PHP code then runs the same form validation procedures as in JavaScript to ensure that all data are valid. If valid, the PHP code then saves the form data in the StationVotes table of the MySQL database. Next, the code retrieves all records in the StationVotes table and prints them to the screen as a table.

In summary, the system receives and validates one record of user data, saves it to a database table, then prints out all records of that database table—including the record that the user just entered.

Design tradeoffs:

The system is much more robust than it is user-friendly. The HTML code and CSS in part 1 was not system-critical and thus was not prioritized to the extent that the form-database interaction was. Thus, although the system is bland and unattractive, it is not known to crash with any input whatsoever. This ensures that only valid data enters the database.

Possible improvements and extensions:

The user interface could be significantly improved by adding improved colouring, styling and layout to the form. Additionally, useful error messages may be added to the form validation process.

Test cases:

1	Enter integer in integer field such as “Votes for Candidate 1”, e.g. 3	Input accepted upon pressing “submit”.
2	Enter integer in integer field such as “Votes for Candidate 1”, e.g. “Hello”	Input not accepted and field turns red upon pressing “submit”.
3	Enter alphanumeric string in Polling Station field, e.g. “61A”	Input accepted upon pressing “submit”.
4	Enter alphanumeric string in Polling Station field, e.g. “61.A!”	Input not accepted and field turns red upon pressing “submit”.
5	Fill out all fields with valid data.	Input accepted upon pressing “submit”.
6	Leave at least one field blank.	Input not accepted and empty field turns red upon pressing “submit”.
7	Ensure that value entered for total votes is equal to the sum of the votes for candidates 1 and 2 and the rejected votes. e.g. Candidate 1: 3, Candidate 2: 5, Rejected: 1, Total: 9	Input accepted upon pressing “submit”.
8	Ensure that value entered for total votes is NOT equal to the sum of the votes for candidates 1 and 2 and the rejected votes. e.g. Candidate 1: 3, Candidate 2: 5, Rejected: 1, Total: 10	Input not accepted and each of the last 4 fields turns red upon pressing “submit”.

N.B. All integer fields were tested with tests 1 and 2.