Software Quality

Deliverable 2.3

TTCW Voting System - Voting System

Fixed/Improved Analysis Design

We have the updated version of the prototype in " **./Voting_system_new_version** ".

https://github.com/fox-Y/CPTS_583-Software-Quallity

It contains:

1.Guest user registration system

2.Complete vote system

3.Administrator login system

4.Result Statistic system

Quality Report

Product Quality Metrics

(Based on TTCW Voting System - Deliverable 1-2)

Quality: Availability

Goal: System should always be available for the all end user to use (rather than centralized polling sites).

Metrics: System should be available to the user 99% of the time.

Validation results: The system is available 100% of the time as the database is not remotely accessible, so it must be saved and accessed locally. Accessing the database is hard coded into the system and is available if all files are downloaded properly.

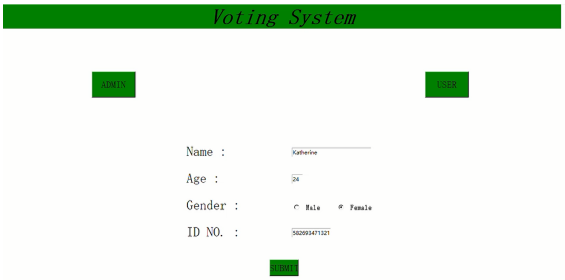
Quality: Reliability

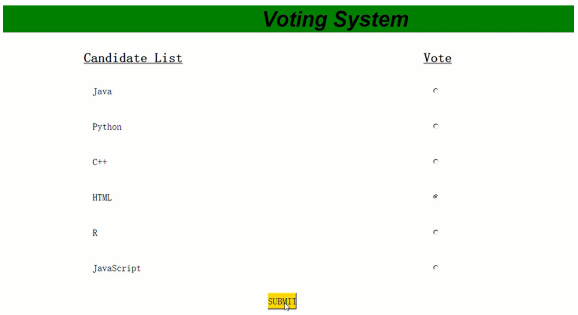
Goal: System should allow users to successfully cast their votes for any candidate.

Metrics: System should 100% correctly interpret user input and convert to a vote.

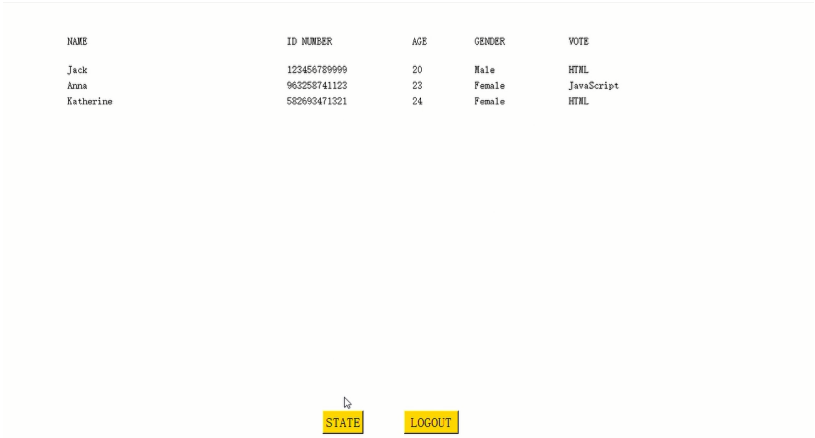
Validation results: The system could correctly show the user information and the result.

When Voting:





In Result:



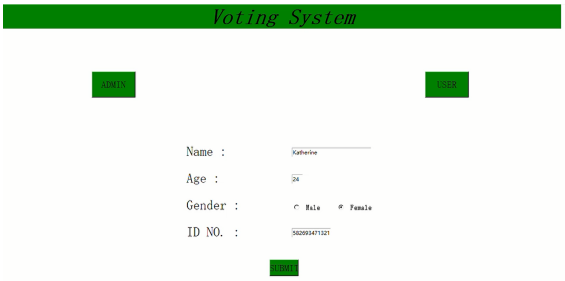
Quality: Robustness

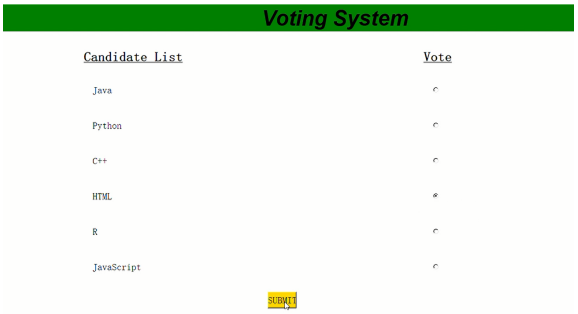
Goal: System should allow users to successfully cast their votes for any candidate.

Metrics: System should 100% correctly interpret user input and convert to a vote. Then show graphs of the political region and current voting statistics.

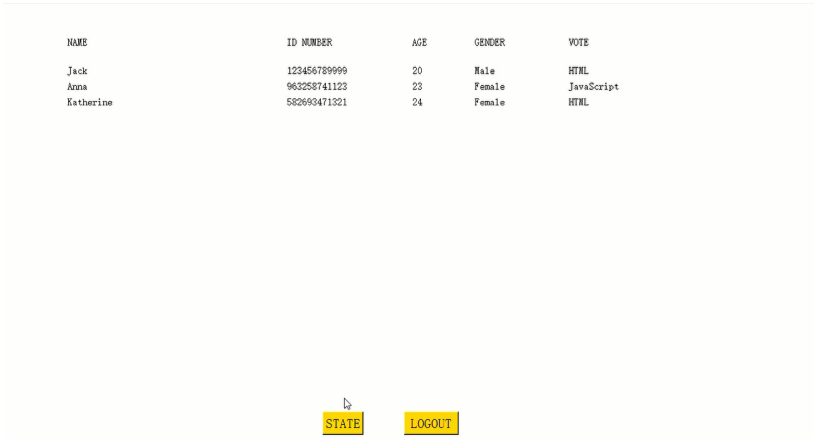
Validation results: The system could correctly show the user information and the result. However, the statistics and region information is not implemented.

When Voting:





In Result:



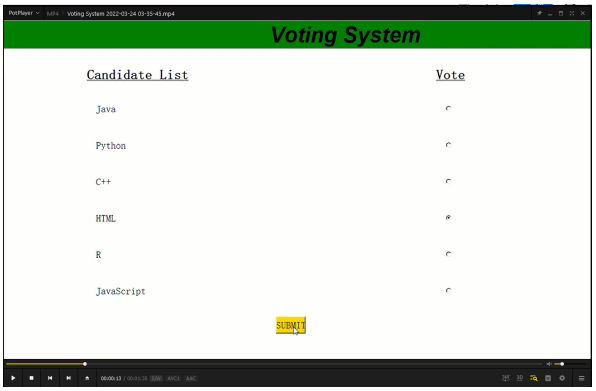
Quality: Learnability

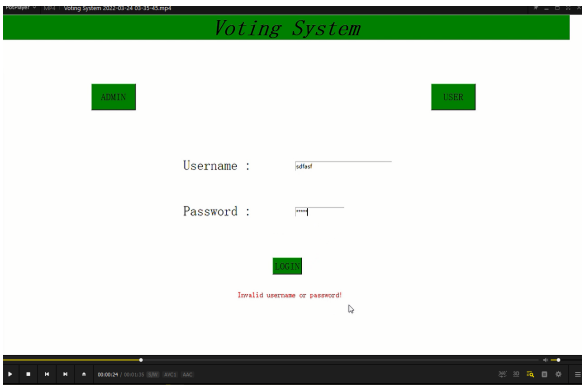
Goal: Users should be able to simple cast their vote.

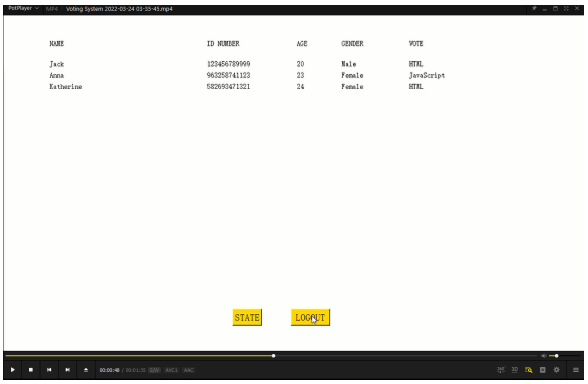
Metrics: A user should be able to cast a vote in a few minutes.

New Metrics: A new user should be able to complete voting in 5 minutes.

Validation results: New users have learned to use the software very quickly because the interface of the software is very simple and clear. If there is a problem with the use, the user can also watch the demo video and learn to use it in few minutes. According to our testing video, it needs about 50 seconds to use all functions of the voting system. So, for new users, it is enough to learn how to use.







Quality: Usability

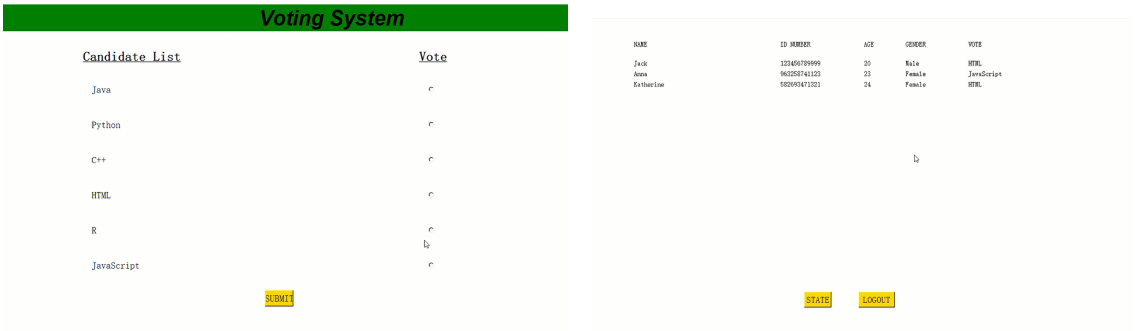
Goal: All users should be able to cast votes. Assuming system works properly.

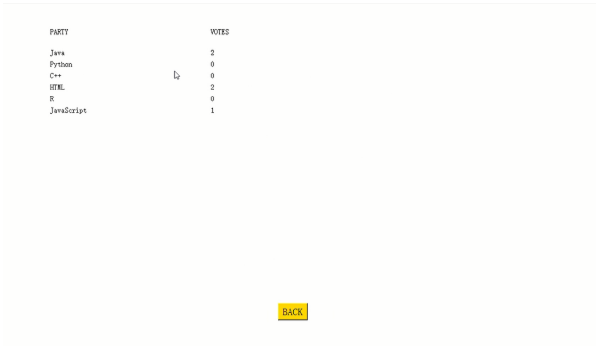
Metrics: Have a user-friendly interface.

Validation results: Because the system does not have complex functions, User Interface is clear. We use contrasting colors on the Graphic User Interface(GUI), such as black font and white backboard. It could make users use the system easily.

Users could type words and then click the button to choose / go to other interface

Disadvantages: Voting and Result pages are too clear and need more details.





If there are lots of users and candidates, it would be hard to look and analyze the system. It would be better to have the Filter function and Chart functions for statistics and analyzing.

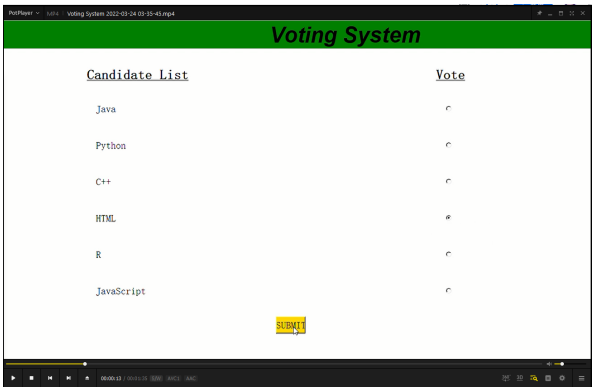
Quality: Efficiency

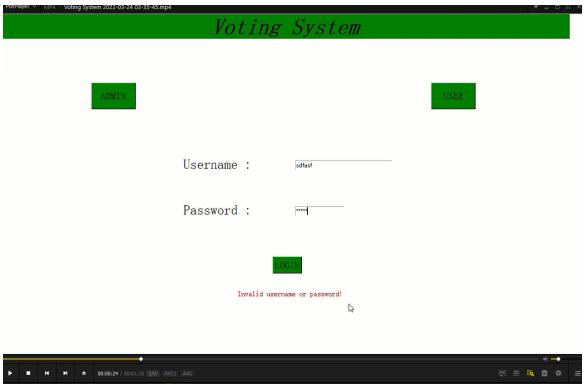
Goal: System should give feedback to users when voting and registering to vote.

Metrics: Should return all candidates info in less than 1 sec.

Validation results: After the system has finished entering the information and sending it out, such as log in, log out and submit the voting option, The delay time of the system response is always less than 0.5 seconds.

Most of the time in testing video is spent on typing and thinking about what to do.





Quality: Security

Goal: Secure login for administrators.

Metrics: Hash passwords when creating administrator accounts and save it in the database. Login includes 2 Factor Authentication.

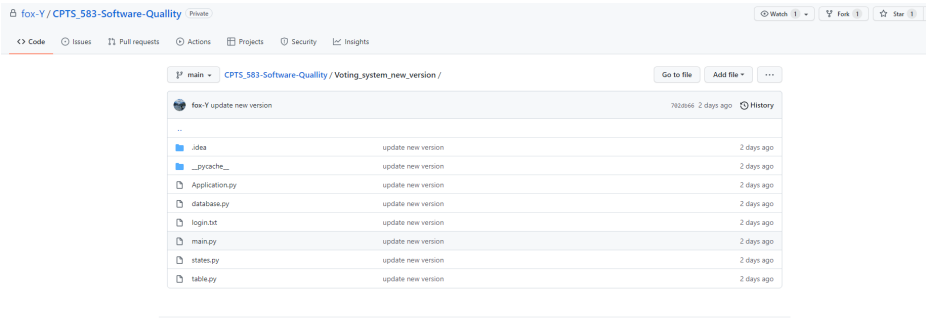
Validation results: The GUI does not have the ability to create an administrator so the database is currently filled with non-hashed passwords. The hash function Argon2 has been implemented and connected to the GUI, and is ready to be used when the database is updated. 2FA has been changed to captcha, instead of email OTP code, and is also fully implemented. Passwords are allowed to be longer than 6 characters and must include an uppercase letter and a number. Further functionality will include special characters and captcha response from GUI.

Quality: Portability

Goal: Service should be on a microservice architecture so individual components of the system can be upgraded individually.

Metrics: Individual services can be updated and switched without affecting other services

Validation results: Different services of the systems have their own subfunction, and they are classified into different py files. Then, we could easily find and fix the target part.



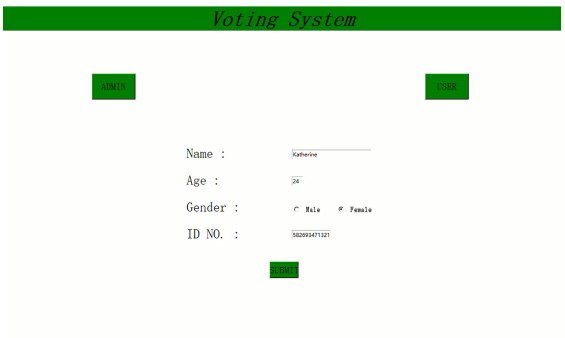
Process Quality Metrics

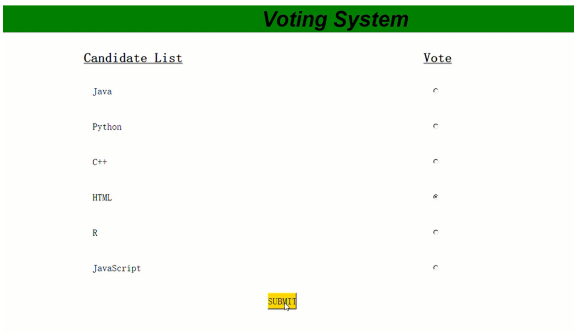
Quality: Maintainability

Goal: The system should be easy to maintain.

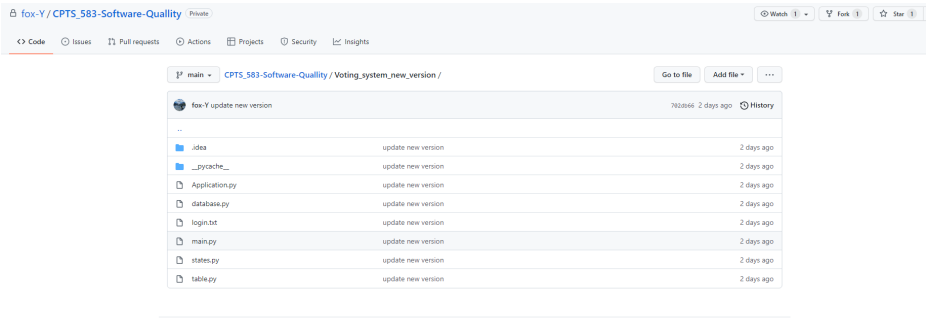
Metrics: The backend API server will follow the MVC design patterns. Code needs to be documented well and follow design patterns.

Validation results: The Model View Controller (MVC) design pattern specifies that an application consists of a data model, presentation information, and control information. For Model, there are the py files for different functions, such as setting GUI and connecting to the MySQL database. For View, the interface or page would change immediately through the data fetched from the Model. For Controller, we designed typing blocks and buttons in the form of GUI for making users easy to use the Voting system. Hence, each part of MVC are actually created and could work well as the full program.





Different services of the systems have their own subfunction, and they are classified into different py files.



Quality: Testability

Goal: System should be easy to add additional tests.

Metrics: Achieve 90% code coverage and 95% branch coverage.

Validation results: The system is not easily testable as all code was rewritten into one file. This made it difficult to test each individual module/core functions besides the GUI as we can not directly access those systems. This method also makes mutation coding more difficult as the whole system is involved rather than the singular module (unit testing). Additional code outside of the GUI was written to test the individual modules, but is not being accessed by the main GUI procedure.

Planned future tests:

1. *Modular Testing*
2. *Code Coverage Tests*
3. *Mutation Testing*

Quality Goals

Product Quality	Quality Goal	Quality Metric	Verification and validation results
Availability	System should always be available for the all end user to use (rather than centralized polling sites).	System should be available to the user 99% of the time.	There are detailed graphs and analysis above.
Reliability	System should allow users to successfully cast their votes for any candidate.	System should 100% correctly interpret user input and convert to a vote.	There are detailed graphs and analysis above.
Robustness	System should allow users to successfully cast their votes for any candidate.	System should 100% correctly interpret user input and convert to a vote.	There are detailed graphs and analysis above.
Learnability	Users should be able to simple cast their vote.	A user should be able to cast a vote in a few minutes.	There are detailed graphs and analysis above.
Usability	All users should be able to cast votes. Assuming system works properly.	Have a user-friendly interface.	There are detailed graphs and analysis above.
Efficiency	System should give feedback to users when voting and registering to vote.	Should return all candidates info in less than 1 sec.	There are detailed graphs and analysis above.
Security	System use only tally votes from verified users.	Enforce a location rule that only allows votes to be casted if location requirement is met.	There are detailed graphs and analysis above.
Portability	Service should be on a microservice	Individual services can be updated and	There are detailed graphs and analysis

	architecture so individual components of the system can be upgraded individually.	switched without affecting other services.	above.

Product Quality	Quality Goal	Quality Metric	Verification and validation results
Maintainability	The system should be easy to maintain.	The backend API server will follow the MVC design patterns. Code needs to be documented well and follow design patterns.	There are detailed graphs and analysis above.
Testability	System should be easy to add additional tests.	Achieve 90% code coverage and 95% branch coverage.	There are detailed graphs and analysis above.