**GROUP 17**

**MEASUREMENT AND INSTRUMENTATION**

**COMPUTER ENGINEERING 2**

**PROJECT REPORT**

**BUILDING A QUIZ SYSTEM.**

**TABLE OF CONTENT**

1. Concept of the project
2. Methodology and implementation
3. Architecture of the project
4. The server
5. The frontend
6. Block diagram
7. Sample working example
8. Results and conclusion
9. Future recommendations
10. Reference
11. Group members

**CONCEPT OF THE PROJECT.**

The project simply named quiz, is a web-based quiz system. Its sole purpose is to accurately assess individuals on a quiz. It does this by awarding marks according to the answer chosen and the time taken in answering. This allows us to rate two or more individuals with regards to how fast they were. The project aims to not be very complex to understand and hence implements for the most part, only required features.

**METHODOLOGY AND IMPLEMENTATION.**

The problem specified was one that could be achieved with many different technologies. However, we chose to build our solution around the web. This is because the web is readily accessible to everyone and offers seemingly great user experience with little effort. It is also understood by most other people and people will not need to download a whole app onto their computer or mobile phone just to answer one simple quiz. In the goal of not creating complex solutions, no javascript frontend framework (eg. Vue, angular, react) was used in the project.

There are a few api endpoints but these were not designed for single page applications.

The development was done on github with the repository <http://www.github.com/Joetib/quiz>

**ARCHITECTURE.**

The project is divided into two sections, the backend and a front end. Splitting the codebase this way has a couple of advantages and disadvantages. Here we ended up using about two general purpose languages to build a worthwhile solution.

THE BACKEND:

The backend is built on top of python and the Django framework. Python was chosen because of its simplicity and that the Django framework comes with batteries included reducing boiler plate code.

The backend manages user authentication and storage of quizzes. It also keeps the results of every solution attempted on a quiz.

However, if a person answers a quiz more than once, only the most recent results is stored in the database to reduce bloating the database.

Data collected during the creation of a quiz include

1. The title of the quiz
2. A short description of the quiz
3. The maximum time allocated to each question (in seconds)
4. The maximum marks allocated to each question
5. Pass mark for each question.
6. Questions
7. Each question must have a question text
8. Options that can be selected from.
9. At least one option must be denoted as being the correct answer
10. If there is already an answer marked as correct and another answer happens to be denoted as correct, the previously marked correct answer will be denoted as incorrect

Only authenticated users can answer and create quizzes.

On receival of these details, they are validated with the help of Django-rest-framework serializers. When they are of the required format, then they are saved to the database.

Sqlite3 was chosen as our database since it was small and didn’t require any difficult configuration. Also, the hosting plan we used for testing was a free tier offered by pythonanywhere.com and hence we could not configure any server-based database like postgresql, mysql, mongodb and the like.

THE FRONTEND:

The frontend, made with html, css (with bootstrap4) and javascript handles most of the work implementation. The frontend, relying heavily on plane javascript handles the collection of all required data for the creation and modification of quizzes.

All attempts to a quiz are also conducted by the frontend. As part of this, it checks the time it takes to answer a question. If the designated maximum time for each question reaches, it skips to the next question.

If an answer if chosen for a question it also calculates the marks based on the time it took to answer the question.

1. If the answer is wrong, then the marks allocated is zero.
2. If the answer is right, and the user answered within the first ten percent of the time allocated for each question, he/she is awarded the entire marks allocated to a question. Without this approach, we found that it will be practically impossible to have a person score all the marks
3. If the answer is right, and the user answered after the first ten percent of the time allocated, his score is calculated with the methodology below.

Taking **T** to be the maximum time allocated to each question

And **M** to be the maximum marks allocated to each question

And **t** to be the time it took the user to choose an answer,

And **m** to be the marks that the user got for the question,

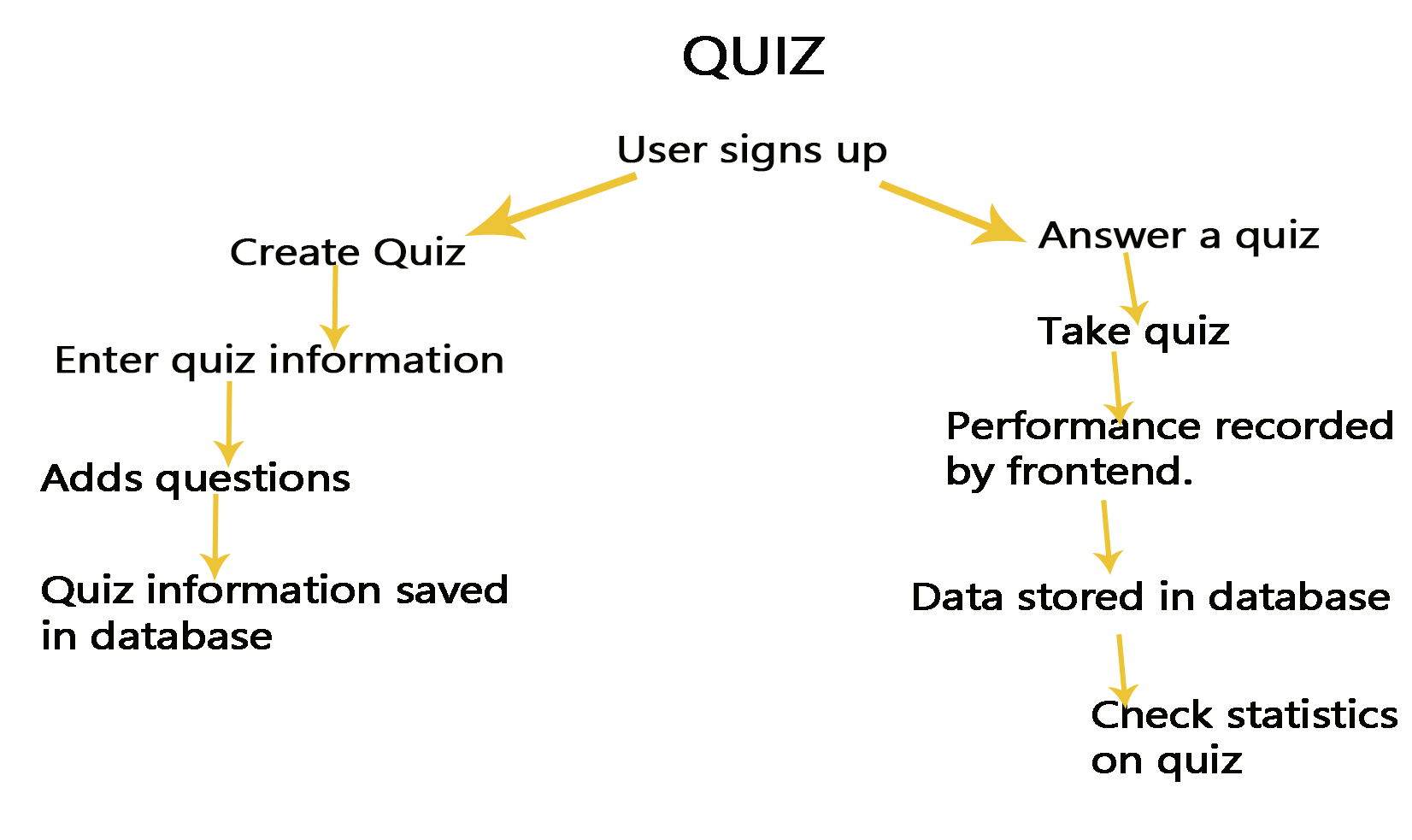
Hence, the minimum scores a person can get for answering the question right, is half the maximum score allocated. And the remaining half depends on how long it took to answer the question.

This data is then sent to the backend to be stored in the database, where validations are made on the data before being saved. Again, only authenticated users can take a quiz.

The assessing of the quiz is handled by the frontend for a few reasons.

1. This reduce having to work with websockets thus making the design simple.
2. This reduces load on the server as it does not need to do all calculations and keeping live pool of connections. Much of the heavy work is done by the frontend and hence, requests to the server are minimal making it great for scaling better even in its current state.

**BLOCK DIAGRAM.**

****

**SAMPLE WORKING**

A sample usage video is attached to the folder with the report. It is named ‘test\_video.mp4’.

It simply shows how the quiz system works by taking up a quiz on the deployed server at <https://www.komlearn.pythonanywhere.com> This site holds a few quizzes that can be quickly tested out to check its performance. The server is small and offers only a few megabytes of ram and storage, yet the website performs fairly well.

To run the code sample, steps are in the readme.md file.

**RESULTS AND CONLUSION**

Django and a little javascript was used to create a fast voting game project. The website is working as planned. All the sections are working perfectly and there are no faults. All you need is a stable network and you are good to go. The project determines the fastest person to tap the button and indicates it. There are some quizzes you can attempt so as you attempt it shows the fastest to get it correctly.

The site runs perfectly and it is able to perform the task assigned so the project was a success. And it is easily accessible because it is on the internet

**FUTURE RECOMMENDATIONS.**

The current build of the website is very useful and would do for its present needs but moving forward there could be more features, functionality or implementation that could be changed and/or added onto it for user friendliness.

First of all, a much robust database such as PostgreSQL and MySQL could be used. PostgreSQL helps administrators ensure data protection and integrity and build fault tolerant environments and help manage dataset. It is free and also extensible.

The improvement of User Interface and experience by adding some call to action buttons and animations to help the visual appeal. Also help in the easy navigation of the website as a whole.

An Exam feature could be added to the website as feature. This feature could allow for Tutors to create Exam format questions where a fixed set of questions is timed for results. This is unlike the quiz because it is a popup type. With this Exam feature Multiple Choice Questions that is randomly distributed to students in order to reduce cheating. The results of the Exam would be sent to the tutor to use for other things.

REFERENCE

1. The Django documentation : [www.djangoproject.com/documentation](http://www.djangoproject.com/documentation)
2. Django 3 by example book.
3. Javascript tutorial on [www.tutorialspoint.com](http://www.tutorialspoint.com)

**GROUP MEMBERS**

1. OTI BOATENG JOSEPH
2. AGBODJIVE ETORNA KOFI
3. BOAHENE KWADWO BAAFI
4. DORKENOO HOPE
5. ANANI CLINTON BAMDEN
6. BOTCHWAY SOLOMON