

Computer Science Revision Website - for GCSE Students  
Natalia Witkowska  
4235

[**Analysis**](#_gnnm73d9dmrp) **5**

[Overview](#_20kdxhr600rd) 5

[Client and the problem](#_o6ybncikgqyd) 5

[Investigation of the Project/Research](#_bz00dzqo7ho0) 5

[Questionnaire](#_3niiya624f05) 8

[Analysis of questionnaire](#_d54kmd6cupu) 10

[Potential Solutions](#_t6pqspiy2kqk) 11

[Objectives against Performance Criteria](#_b4resf27qc4i) 11

[Modelling the project](#_8p62bbogc3de) 13

[Constraints/Limitations](#_t6tcra5s91yc) 15

[**Documented Design**](#_j094hrh7r5s3) **16**

[Overall system design and Databases](#_275vtx1ffg1v) 16

[Creating Databases](#_i3sv34wzblg) 17

[Validating Data](#_3lbgklsglkhz) 19

[User Interface](#_910ekt6q93dq) 20

[Bootstrap](#_dvlk8ec9og3w) 20

[HTML](#_s71w392ggv5z) 20

[CSS](#_o89uvgc4w9jx) 20

[UI Design](#_z0djasmh5yvd) 21

[**Technical Solution**](#_nk0v7d9p026p) **22**

[main.py](#_ogsb4y3aciwq) 22

[layout.html](#_ryl6mvcvdeao) 29

[index.html](#_z6epkrepllyc) 30

[register.html](#_j5wqln71hync) 31

[home.html](#_jegq7q6nbggl) 32

[profile.html](#_m2lviw69qoi1) 32

[quiz.html](#_qnunsfcbyrvj) 33

[question.html](#_p4gqjcjigrst) 34

[flashcard.html](#_kriqmcgct01z) 35

[style.css](#_qenw2s3duq12) 36

[**Testing**](#_jz131v70yigh) **40**

[Test Tables](#_n6l9knfnejg0) 40

[**Evaluation**](#_dtuvoqpi1tm) **42**

[Achieving Objectives](#_gucsq9rm8fh7) 42

[Potential for Future Development](#_kds193wynzpd) 42

# 

# Analysis

## Overview

I am designing a Computer Science Revision website for GCSE Students.

This website will give the students access to tips about how to revise specific difficult content, allow them to take quizzes that quiz them on the content, and in turn, it should be used to help students with learning Computing definitions.

I aim to create a website that includes: dynamically changing content (i.e. charts of students' progress), a sorting database, cross-table SQL, a login system with admin/teacher login (including sessions), and quiz taking.

I will be researching similar quiz websites online that include a login option and websites that store data (eg overall marks on a quiz) in a database.

## Client and the problem

The problem is being investigated for a secondary school teacher teaching GCSE Computer Science to students. They test students using quizzes on paper or using online resources. Questions are searched for and compiled into a document which can be printed, along with a mark scheme. However, there are only a finite number of questions, with no way to add more or change the details of the existing ones and the teachers would also like a place where students could create their own quizzes for revision purposes. Therefore, they would like to have one single place online where they could check the progress of students, set quizzes and allow students and teachers to create their own.

## Investigation of the Project/Research

I will be basing my research on similar existing systems. This will include other quiz websites with logins (such as Quizlet). A very important part of the research stage is a questionnaire to gather data from students. This will allow me to understand what students want from the finished web application and how they intend to use it. It will also be helpful in seeing what students would want as they will be the majority of people using the website.

Investigating Quizlet

As a memorization tool, Quizlet lets registered users create sets of terms and definitions customized for their own needs. These sets of terms can then be studied under several study modes such as flash cards, and long term learning.

Flash Cards - This mode is similar to paper flash cards. Users are shown a "card" for each term, which they can flip over by clicking or using the arrow keys or space bar. The user has the option for the face of the card to be an image, a word, or both.

Long Term Learning - This is a mode that quizzes users on pre set or self-made questions and answers. Repetition of terms answered incorrectly increases in frequency and a dashboard shows learning progress over time.

| Pros | Cons |
| --- | --- |
| * It functions as a memorisation tool to assist students' learning. * It offers a variety of learning modes, including four study modes and two study games. * It allows students to share their works with others in Friends and Groups. | * Some flashcards made by other users might give misleading or wrong information. * It’s good for general knowledge (definitions of specific words/learning other languages) but not as good for curriculum specific knowledge which often combines understanding with memorisation and combines knowledge from many different areas. |

Overall, Quizlet is used as a revision website for students rather than a tool for teachers to use. Quizlet isn't intended to be an assessment tool, while it does offer a Teacher Subscription that is used to check student progress, I would like to make a free version that is specific to Computer Science teachers. The website I would be creating would both be a tool for teachers, and for students rather than just for students.

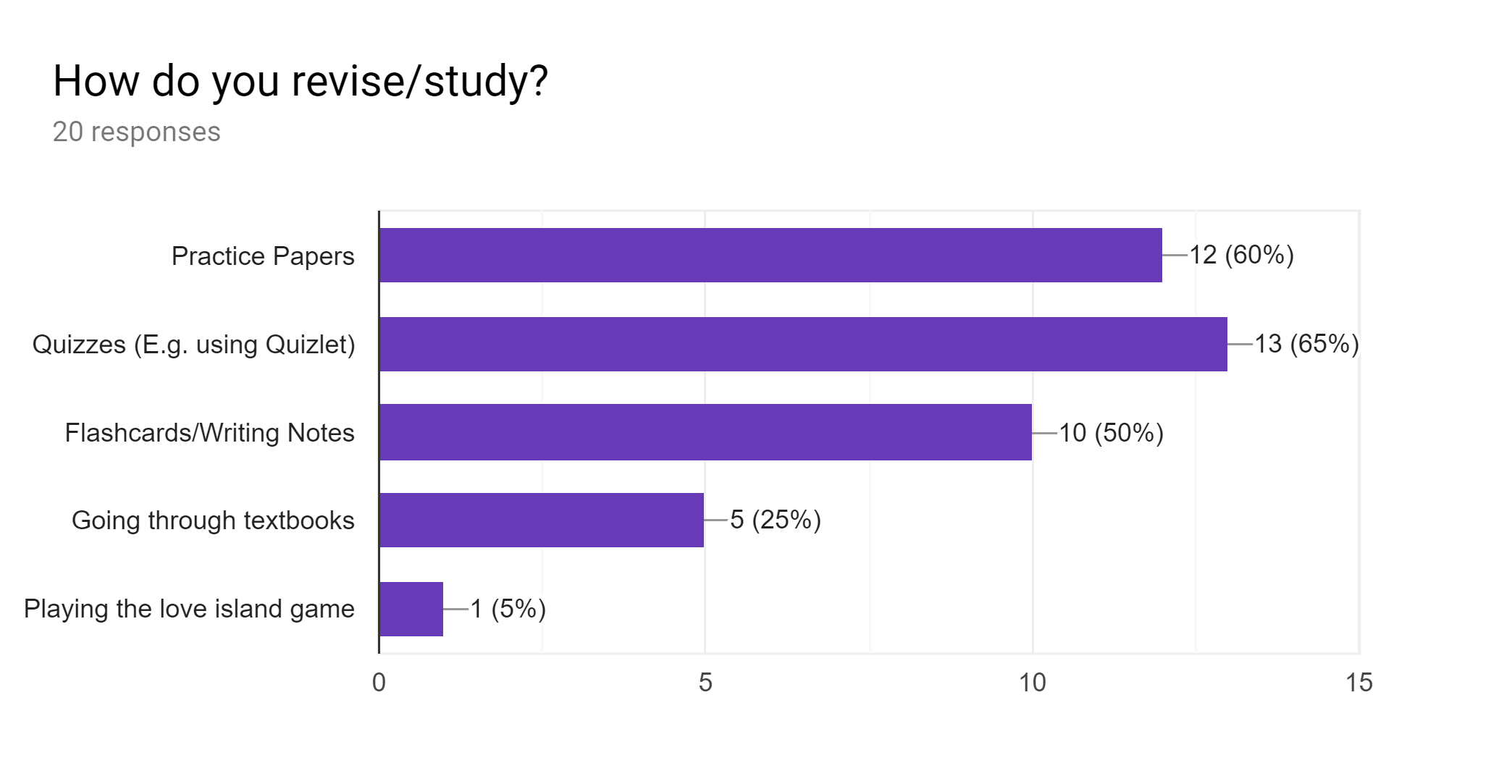
Investigating Kahoot!

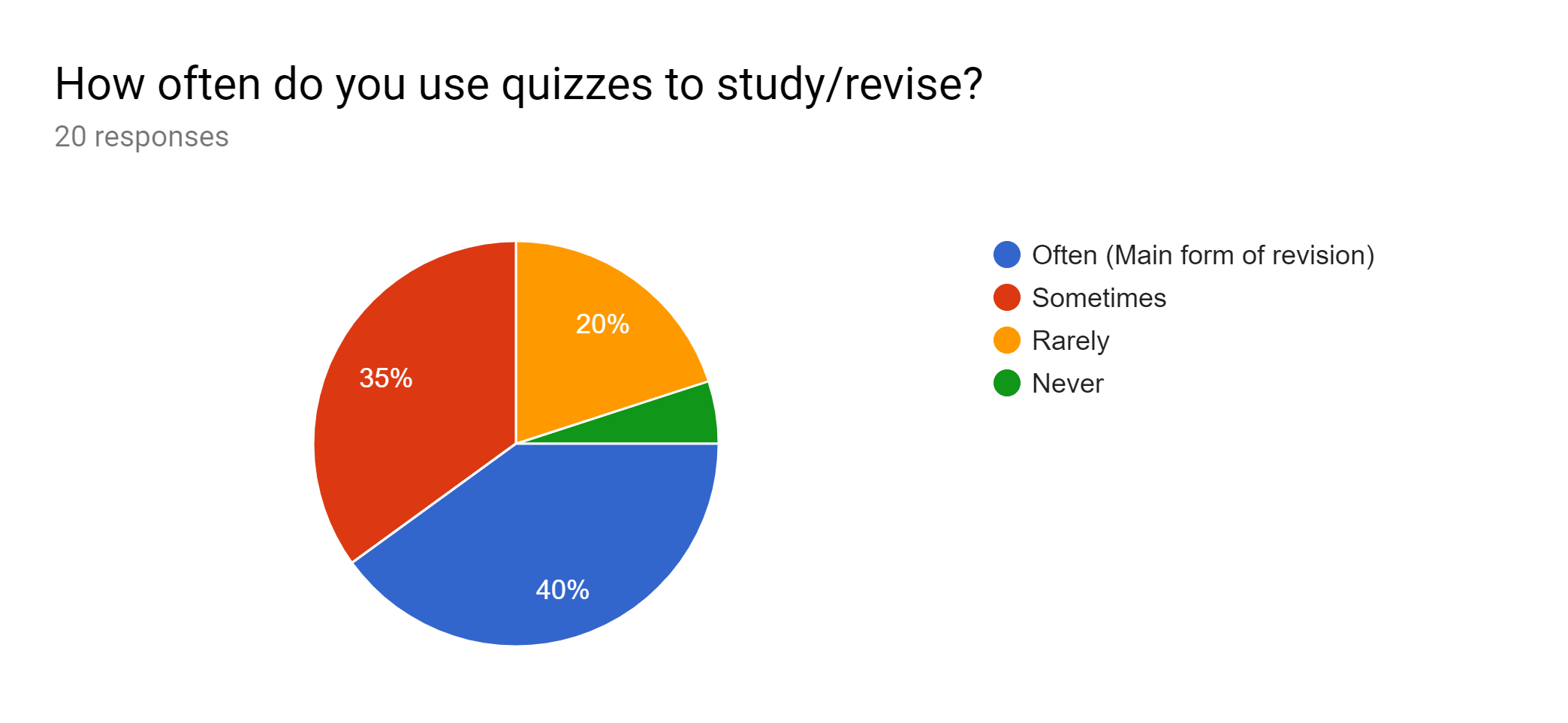
Kahoot! is a game-based learning platform, used as educational technology in schools and other educational institutions. Its learning games, "Kahoots", are multiple-choice quizzes that allow user generation and can be accessed via a web browser or the Kahoot app.

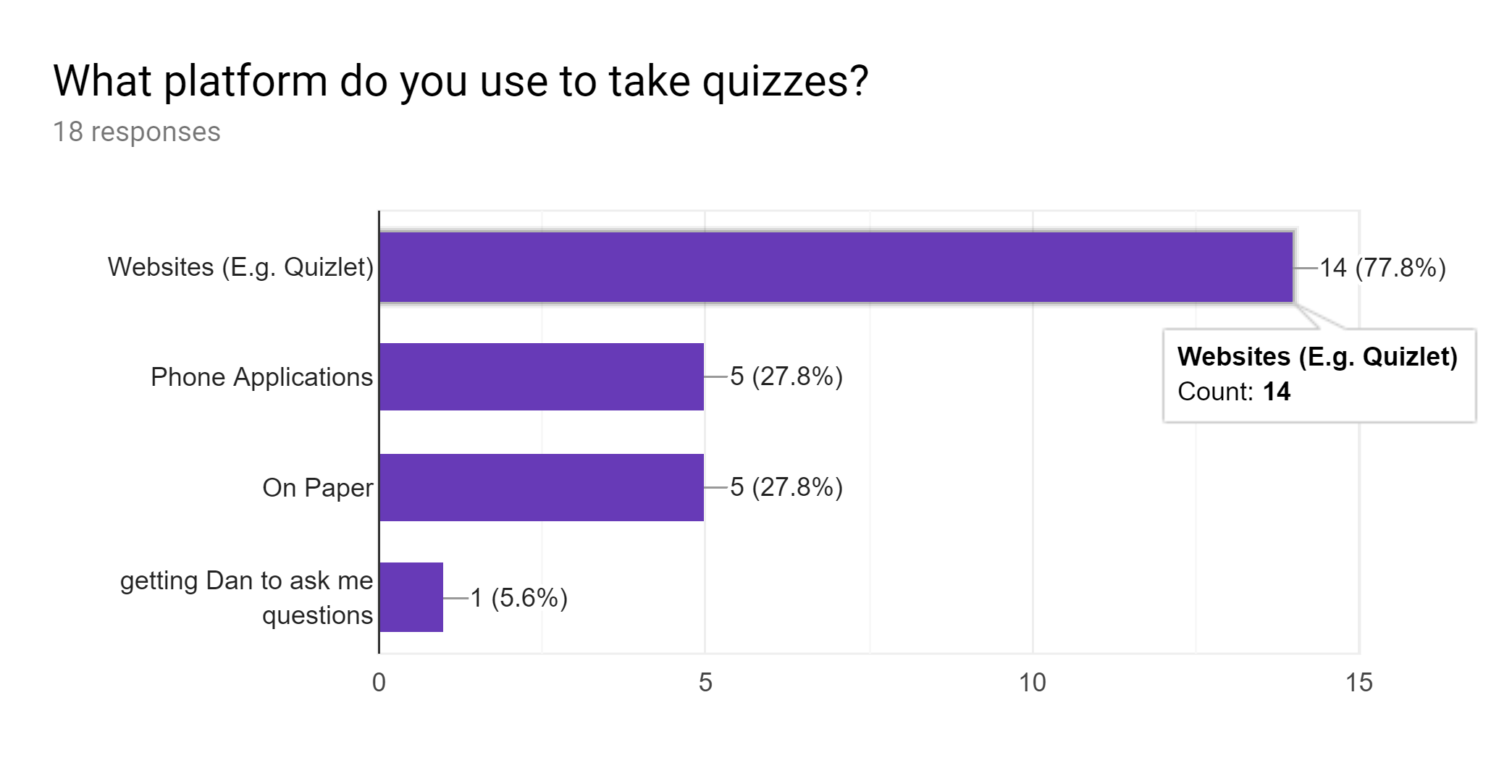
| Pros | Cons |
| --- | --- |
| * It functions as a memorisation tool to assist students' learning. * Student engagement with Kahoot is very high as it is very fast paced. * The Kahoot website is very easy to use for both teachers and students. | * The major disadvantage that is found with Kahoot is that it is hard to track student progress. In order to track progress the teacher would have to breakdown usernames and then record the number of answers each student got right in every Kahoot that was played. |

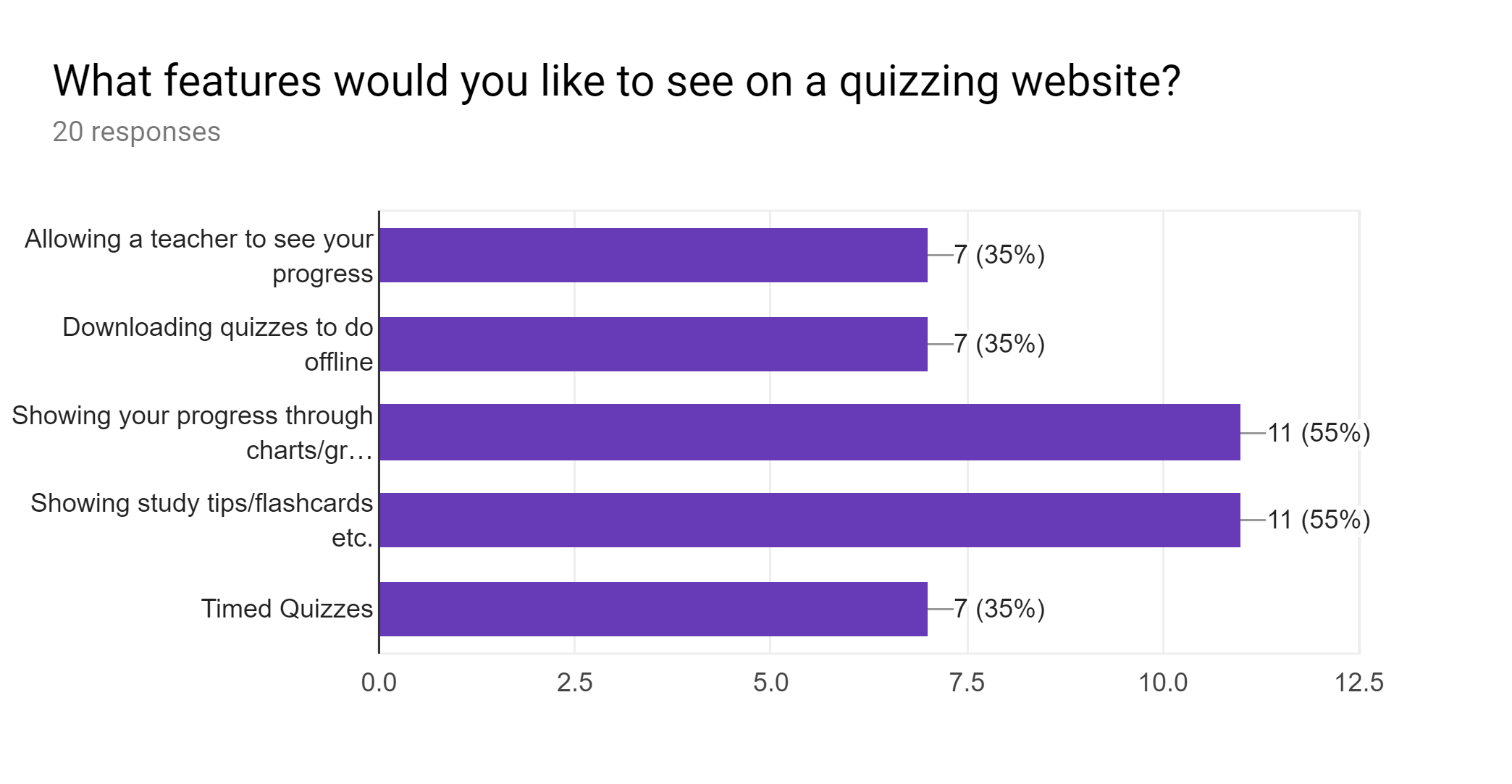
Overall, Kahoot is a great memorisation tool for students, however, it is more of an application to use during school time where all students play together, and is not very suitable for single player. This platform is great however it’s not really a revision tool for examinations, rather, it’s a way to engage students in fast paced memory recall. Usually the quizzes offer short answers and would not be suitable for learning things such as definitions of key terms in Computer Science.

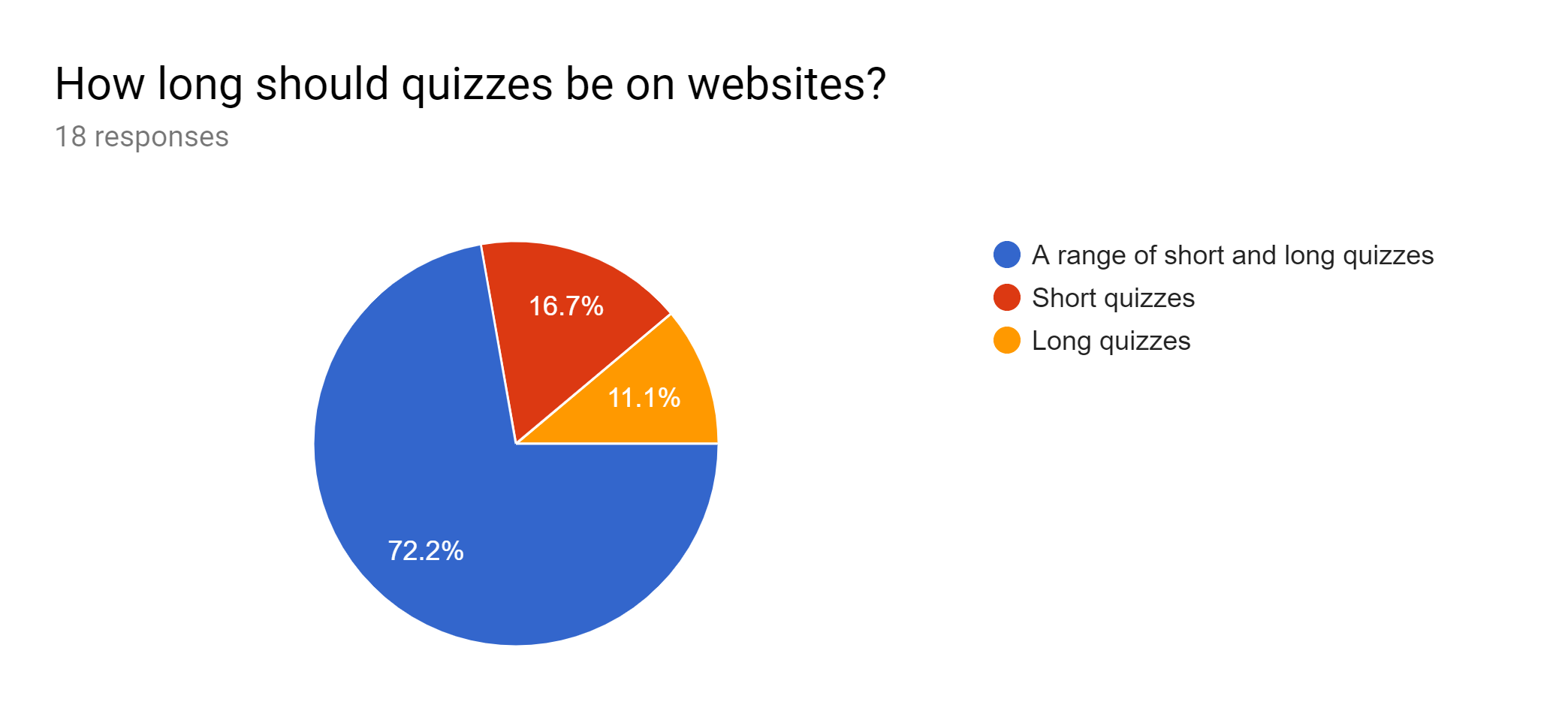
## Questionnaire

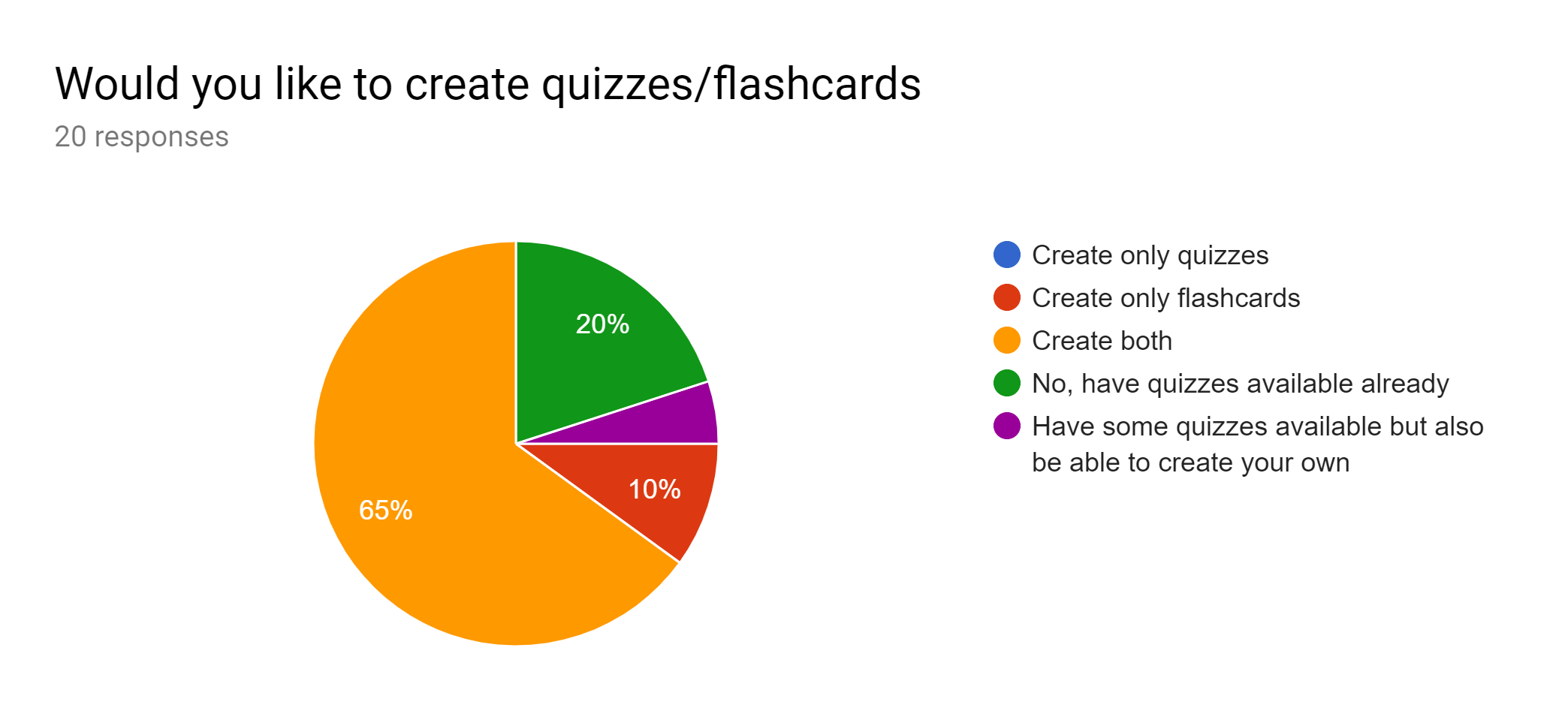












### Analysis of questionnaire

The questionnaire shown above shows the answers by 20 students about how they revise. From the results, a sizable portion of them do use quizzes to study/revise, and use websites as a platform to do these quizzes. A vast majority of them would like to see their progress being tracked and showing tips and creating flashcards, which suggests that a web based application is an effective way to do this as it allows to do this in one place.

## Potential Solutions

Taking into account the questionnaire completed by students and the clients needs there are a few solutions that could be created for the problem.

| **Solutions** | **Positives** | **Negatives** |
| --- | --- | --- |
| A quiz system using PHP and data is stored using sqlite. | PHP is easily connected with the database and makes the connection securely with databases. It has a built-in module that is used to connect to the database easily. | Security - Easy to hack as language is very simple and easy. SQL injection is very common on websites that use PHP. |
| A web-based system using Flask. Students would create logins and their scores would be stored in a database using MySQL. | Flask’s main advantage is that it is one of the most used Python web frameworks, which is why there are a lot of available tutorials and libraries for it. | Being a mini web serving framework, it doesn't provide enough scalability for larger projects. |

## 

## Objectives against Performance Criteria

I created my objectives based on what the students have said through the questionnaire and what the clients wanted. While students thought allowing teachers to see progress was unnecessary, I will still be implementing it as the clients (teachers) would want it. Implementing this feature also allows the website to differ from other similar applications (e.g. Quizlet). I did take students advice by adding the objective of flashcards to my project. This would be a great tool for revision for students which is something the client also wanted.

| **Objectives** | **Performance Criteria** |
| --- | --- |
| 1. Create a website that allows people to login/create an account | 1.1. Using unique usernames  1.2. Allowing a user to logout  1.3. Be able to register and then use registered account to login |
| 1. Maximise security | 2.1. Use hashing techniques to keep passwords protected  2.2. Allowing only the student to see their progress  2.3. Passwords checked to see if they include numbers and capital letters to minimise hackers using brute force attacks |
| 1. Showing Progress | 3.1. Showing data in tables  3.2. Showing data/progress in charts/graphs  3.3. Allowing teachers to see progress  3.4. Be able to view users' history of their quiz results. Storable by either date or percentage  3.5. I will use cross-table parameterised SQL statements with aggregate functions to retrieve data and show this data in either tables or charts. |
| 1. Creating/Editing Quizzes | 4.1. Allow users to create multiple choice quizzes |
| 1. Completing Quizzes | 5.1. Have a range of short and long quizzes  5.2. Timed and untimed quizzes  5.3. Allow users to access premade quizzes from the database |
| 1. Creating/Completing Flashcards | 6.1. Allow users to access premade flashcards from the database  6.2. Allow user to create their own flashcards  6.3. Allow users to print the flashcards in a flashcard format |

## Modelling the project

|  | This sequencing diagram was created to help me visualise how the login system will link my python code to the database I will be using.  It includes both registration and login and how they both are associated with the web browser and database.  It also shows how the user interacts with the website. |
| --- | --- |
|  | I created a data flow diagram to help conceptually think about how quizzing scores would be saved to a database. |
|  | I will be using werkzeug security, which is an extension that provides utility functions for hashing data within a python application.  This will be used to keep passwords safe in my user database. |

## 

## 

## 

## Constraints/Limitations

Software and Hardware constraints

There is only a limited amount of software available for me to design the new website and I would not be able to actually put the website online as I would need to buy a domain name etc.

Quiz constraints

Although there are question types in Computer Science that involve images or diagrams with labeling that feature in exam/assessment papers, I will be limited by time to create a certain amount of quizzes. I will only be creating text based quizzes due to these time limitations.

Time constraints

The system needs to be completed by May 2020.

| **Entity Relationship Diagram** | This ERD shows how some of the tables in my database will be linked together through the use of primary and foreign keys.  This database is in 3rd normal form. |
| --- | --- |
| **Hierarchy Chart** | The program will have 4 main sections. These will be Quizzes, Flashcards, Progress, and Accounts.  Using a navigation bar and a drop down menu in the actual code will allow me to add “subsections” to these main sections shown in this diagram. |
| **Model View Controller Diagram** | I researched about the model view controller approach to websites and applications and created a diagram to help me with visualising it.  <https://blog.codinghorror.com/understanding-model-view-controller/>  This website helped me understand the concept more. |

## Creating Databases

| * **CREATE** **DATABASE** IF NOT EXISTS 'pythonlogin' **DEFAULT** **CHARACTER** **SET** utf8 **COLLATE** utf8\_general\_ci; * USE 'pythonlogin'; * **CREATE** **TABLE** IF NOT EXISTS 'accounts'( * 'id' **int**(11) NOT NULL AUTO\_INCREMENT, * 'username' **varchar**(50) NOT NULL, * 'password' **varchar**(255) NOT NULL, * 'email' **varchar**(100) NOT NULL, * **PRIMARY** **KEY** ('id') * ) | Throughout my project I will require many different SQL statements. This will include:   * Creating tables * Extracting Data * Inserting new records * Deleting existing data   Through MySQL and MySQL Workbench I am able to do this efficiently. I was able to create a local server and make the database called ‘pythonlogin’. |
| --- | --- |
| * **CREATE TABLE IF NOT EXISTS 'questions1' (** * **'qid' int NOT NULL AUTO\_INCREMENT,** * **'QuestionText' varchar(100) NOT NULL,** * **'Topic' varchar(100) NOT NULL,** * **PRIMARY KEY ('qid')** * **)** |  |
|  | There is a limitation in sqlite which means that using a composite key combined with  any form of auto-increment is not allowed.  MySQL does not have this problem and auto-increment is allowed as seen in these screenshots. |
| * **CREATE** **TABLE** IF NOT EXISTS 'answers' ( * 'aid' **int**(11) NOT NULL AUTO\_INCREMENT, * 'qid' **int**(11) NOT NULL, * 'AnswerText' **varchar**(100) NOT NULL, * 'Correct' **varchar**(100) NOT NULL, * **PRIMARY** **KEY** ('aid'), * CONTRAINT 'fk\_qid' **FOREIGN** **KEY**('qid') * **REFERENCES** questions1('qid') * ) | Here, I created my ‘answers’ table. It includes the answer id, question id (a foreign key), the answer text, and if the answer text is correct. This allows for multiple choice questions to be viewed. |
| * **CREATE** **TABLE** IF NOT EXISTS 'results' ( * 'resultid' **int**(11) NOT NULL AUTO\_INCREMENT, * 'id' **int**(11) NOT NULL, * 'qid' **int**(11) NOT NULL, * 'QuizResult' **varchar**(100) NOT NULL, * **PRIMARY** **KEY** ('resultid'), * **FOREIGN** **KEY**('qid') **REFERENCES** questions1('qid') * **FOREIGN** **KEY**('id') **REFERENCES** accounts('id') * ) | Here, I created my ‘results’ table where it keeps the user’s results for the quiz. It includes Result id, id (a foreign key), Question id (another foreign key), and Quiz Result. |

## 

## Accounts table

|  |
| --- |
| This is where the login details of the students are kept. This includes: the username, hashed password, and email of the student. Each user has a unique id assigned to their username which will be used as a foreign key in the results table. |

## 

## Questions table

|  |
| --- |
| This is where the questions for the quizzes are kept. This includes the Question Text and Topic the question is in. It also includes a unique question id |

## 

## Answers table

|  |
| --- |
| This is where the answers for the quizzes are kept. |

## 

## Results table

|  |
| --- |
| This is where the results of the user’s quizzes are kept. |

## 

## Validating Data

The following table contains all the data that could be entered into the system by the user, and

how I plan to validate it to ensure that all fields are correctly inputted and stored. The

erroneous data for most fields is a blank field, so in cases where the validation check is only

allowing the user to select the correct fields from a list or directory, there is no way to test this.

| **Data** | **Table in DB** | **Data Type** | **Validation** | **If Valid** | **Erroneous** |
| --- | --- | --- | --- | --- | --- |
| Username | accounts | string | Users may not have the same username as another | Username does not exist - allows user to register | Tells user that username is already taken |
| Email | accounts | string | Users enter a valid email | Email exists - allows user to register | Tells user to enter a valid email |
| Subject Name | questions | string | Allows the user to select from only existing files. | The file will exist | A name which doesn’t exist |

## 

## 

## User Interface

### Bootstrap

I will be using Bootstrap. I went on the website <https://themes.getbootstrap.com/>to get inspiration from existing themes that exist. At first I will use the basic theme to keep the website design basic while I design more of the “back-end” parts of the website (e.g. how the webpage will connect to databases etc.)

### HTML

Almost all the browsers around the globe are supported by HTML therefore, I will be using it to create my website.HTML is very easy to edit as there is no need to have a special interface or platform to edit it.

### CSS

CSS allows for more control over the presentation of web pages, so I will be using it together with HTML. I will be able to organise my code to make it easier for screen readers to follow, making for a more accessible web page.

### 

## 

## UI Design

|  | Design drawn on paper. This is what I wanted the login screen to look like and based the design off this. |
| --- | --- |
|  | This is the home page design I drew on paper. I also based my actual design off of this. |

# 

|  | This was how my website looked the first time I ran it. It included a homepage, login page, and sign up page. I used the basic bootstrap template. |
| --- | --- |
|  | This is the login page I created. You get sent to it when you click on the “Login” button on the homepage. |
|  | This is how the sign up page looked when I first ran it. You get sent to it when you click on the “Login” button on the homepage. |
|  | I have changed how the website homepage looks using icons from a website called [Font Awesome](https://fontawesome.com/icons?d=gallery). |
|  | This is the login page that is accessed when you first go onto the website. Buttons are used between logging in and registering a new user. |

# Technical Solution

### main.py

1. **from** flask **import** Flask, render\_template, request, redirect, url\_for, session, flash, jsonify, json, current\_app, send\_from\_directory
2. **from** flask\_mysqldb **import** MySQL
3. **import** MySQLdb.cursors
4. **import** re
5. **import** os
6. #from flask\_bcrypt import Bcrypt
7. **import** bcrypt

10. app = Flask(\_\_name\_\_)
11. #bcrypt = Bcrypt(app)
13. # secret key
14. app.secret\_key = os.urandom(24)
16. # Here are the Database connection details below
17. app.config['MYSQL\_HOST'] = 'localhost'
18. app.config['MYSQL\_USER'] = 'root'
19. app.config['MYSQL\_PASSWORD'] = 'root'
20. app.config['MYSQL\_DB'] = 'pythonlogin'
22. # Intialising MySQL
23. mysql = MySQL(app)
25. #quiz class
26. **class** DatabaseHandler(object):
27. #Database interface. All DB calls are made in this class.
29. **def** \_\_init\_\_(self,dbcursor):
30. self.\_cursor = dbcursor
32. **def** get\_question(self,topic):
33. #Return question text
34. self.\_cursor.execute("SELECT QuestionText FROM questions1 WHERE Topic=?",(topic,))
35. **return** self.\_cursor.fetchone()[0]
37. **def** get\_question\_count(self):
38. #Return total number of questions in DB
39. self.\_cursor.execute("SELECT count(\*) FROM questions1");
40. **return** self.\_cursor.fetchone()[0]
42. **def** get\_answers(self,questionid):
43. #Return all possible answers for a given question
44. self.\_cursor.execute("SELECT answertext,correct FROM answers WHERE questionid=?",(questionid,))
45. **return** [{'answertext':answertext, 'correct':correct} **for** answertext,correct **in** self.\_cursor.fetchall()];
47. **def** get\_score(self,userid):
48. #Return score for a given player id
49. self.\_cursor.execute("SELECT result FROM results WHERE id=?",(userid,))
50. **return** self.\_cursor.fetchone()[0]
52. **def** set\_score(self,userid,QuizResult):
53. #Set a player's score
54. self.\_cursor.execute("UPDATE results SET QuizResult = ? WHERE id = ?",(QuizResult,userid))
55. self.\_cursor.connection.commit()



60. # http://localhost:5000/ - goes to login page if not logged into a session
61. @app.route('/')
62. **def** indexpage():
63. **if** 'loggedin' **in** session:
64. **return** redirect(url\_for('home'))
65. **else**:
66. **return** redirect(url\_for('login'))


70. # http://localhost:5000/login - this will be the login page, we need to use both GET and POST requests
71. @app.route('/login', methods=['GET', 'POST'])
72. **def** login():
73. # Output message if something goes wrong...
74. msg = ''
75. # Check if "username" and "password" POST requests exist (user submitted form)
76. **if** request.method == 'POST' **and** 'username' **in** request.form **and** 'password' **in** request.form:
77. # Create variables for easy access
78. username = request.form['username']
79. password1 = request.form['password']
80. # Check if account exists using MySQL
81. cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
82. cursor.execute('SELECT \* FROM accounts WHERE username = %s', [username])
83. account = cursor.fetchone()
84. # Fetch one record and return result
85. passwordcorrect = False
86. **if** account:
87. **if** bcrypt.checkpw(password1.encode(), (account['password']).encode()) == True:
88. passwordcorrect = True
89. **else**:
90. # Account doesnt exist or username incorrect
91. msg = 'Incorrect username!'
92. **if** account **and** passwordcorrect == True:
93. **if** account:# Create session data, we can access this data in other routes
94. session['loggedin'] = True
95. session['id'] = account['id']
96. session['username'] = account['username']
97. # Redirect to home page
98. **return** redirect(url\_for('home'))
99. **else**:
100. # Account doesnt exist or password incorrect
101. msg = 'Incorrect password!'
102. **else**:
103. # Account doesnt exist or password incorrect
104. msg = 'Incorrect password!'
105. # Show the login form with message (if any)
106. **return** render\_template('index.html', msg=msg)



111. # http://localhost:5000/home - this will be the home page, only accessible for loggedin users
112. @app.route('/home')
113. **def** home():
114. # Check if user is loggedin
115. **if** 'loggedin' **in** session:
116. # User is loggedin show them the home page
117. **return** render\_template('home.html', username=session['username'])
118. # User is not loggedin redirect to login page
119. **return** redirect(url\_for('login'))


123. # http://localhost:5000/logout - this will be the logout page
124. @app.route('/logout')
125. **def** logout():
126. # Remove session data, this will log the user out
127. session.pop('loggedin', None)
128. session.pop('id', None)
129. session.pop('username', None)
130. # Redirect to login page
131. **return** redirect(url\_for('login'))



136. # http://localhost:5000/register - this will be the registration page, need to use both GET and POST requests
137. @app.route('/register', methods=['GET', 'POST'])
138. **def** register():
139. # Output message if something goes wrong...
140. msg = ''
141. # Check if "username", "password" and "email" POST requests exist (user submitted form)
142. **if** request.method == 'POST' **and** 'username' **in** request.form **and** 'password' **in** request.form **and** 'email' **in** request.form:
143. # Create variables for easy access
144. username = request.form['username']
145. unhashed\_password = request.form['password']
146. password = bcrypt.hashpw(unhashed\_password.encode(), bcrypt.gensalt())
147. email = request.form['email']
148. # Check if account exists using MySQL
149. cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
150. cursor.execute('SELECT \* FROM accounts WHERE username = %s', [username])
151. account = cursor.fetchone()
152. # If account exists show error and validation checks
153. **if** account:
154. msg = 'Account already exists!'
155. **elif** **not** re.match(r'[^@]+@[^@]+\.[^@]+', email):
156. msg = 'Invalid email address!'
157. **elif** **not** re.match(r'[A-Za-z0-9]+', username):
158. msg = 'Username must contain only characters and numbers!'
159. **elif** **not** username **or** **not** password **or** **not** email:
160. msg = 'Please fill out the form!'
161. **else**:
162. # Account doesnt exists and the form data is valid, now insert new account into accounts table
163. cursor.execute('INSERT INTO accounts VALUES (NULL, %s, %s, %s)', (username, password, email))
164. mysql.connection.commit()
165. msg = 'You have successfully registered!'
166. **elif** request.method == 'POST':
167. # Form is empty... (no POST data)
168. msg = 'Please fill out the form!'
169. # Show registration form with message (if any)
170. **return** render\_template('register.html', msg=msg)

173. # http://localhost:5000/profile - this will be the profile page, only accessible for loggedin users
174. @app.route('/profile')
175. **def** profile():
176. # Check if user is loggedin
177. **if** 'loggedin' **in** session:
178. #All the account info needed for the user so it can displayed on the profile page
179. cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
180. cursor.execute('SELECT \* FROM accounts WHERE id = %s', [session['id']])
181. account = cursor.fetchone()
182. # Show the profile page with account info
183. **return** render\_template('profile.html', account=account)
184. # User is not loggedin redirect to login page
185. **return** redirect(url\_for('login'))

188. # http://localhost:5000/quiztopic
189. @app.route('/quiztopic', methods=['GET', 'POST'])
190. **def** quiztopic():
191. # Check if user is loggedin
192. **if** 'loggedin' **in** session:
193. #Getting question info so each topic can be displayed to the user
194. cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
195. cursor.execute('SELECT \* FROM questions1')
196. questions = cursor.fetchall()
197. topics = set()
198. **for** row **in** questions:
199. topics.add(row['Topic'])
200. select = request.form.get('Topics')
201. **if** select:
202. #When topic chosen, takes user to the topic quiz
203. **return** redirect(url\_for('quiz', select=select))
204. #Question topics are shown on this page using a drop down list
205. **return** render\_template('quiz.html', len = len(questions), topics=topics)
206. # User is not loggedin redirect to login page
207. **return** redirect(url\_for('login'))

210. # http://localhost:5000/quiz/<select>
211. @app.route('/quiz/<select>', methods=['GET', 'POST'])
212. **def** quiz(select):
213. # Check if user is loggedin
214. **if** 'loggedin' **in** session:
215. #selects questions that are from the topic chosen in prior page
216. cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
217. cursor.execute('SELECT \* FROM questions1 WHERE Topic = %s', [select])
218. questiontext = cursor.fetchall()
219. questions = []
220. qid = []
221. count = 0
222. answers = []
223. #show question text and the answers to the question as it is multiple choice
224. **for** row **in** questiontext:
225. questions.append(row['QuestionText'])
226. qid.append(row['qid'])
228. **for** i **in** qid:
229. cursor1 = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
230. cursor1.execute('SELECT \* FROM answers WHERE qid = %s', [qid[count]])
231. count =+1
232. answertext = cursor1.fetchall()
234. **for** row1 **in** answertext:
235. answers.append(row1['AnswerText'])
236. # user logged in so shows quiz page
237. **return** render\_template('question.html', username=session['username'], select=select, questions=questions, answers=answers)
238. # User is not loggedin redirect to login page
239. **return** redirect(url\_for('login'))







248. # http://localhost:5000/flashcard
249. @app.route('/flashcard')
250. **def** flashcard():
251. # Check if user is loggedin
252. **if** 'loggedin' **in** session:
254. cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
255. cursor.execute('SELECT \* FROM accounts WHERE id = %s', [session['id']])
256. account = cursor.fetchone()
257. #cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
258. #topic = request.form['topic']
259. #cursor.execute('SELECT \* FROM questions WHERE topic = %s', [topic])
260. #account = cursor.fetchone()
261. **return** render\_template('flashcard.html', account=account)
262. # User is not loggedin redirect to login page
263. **return** redirect(url\_for('login'))

### layout.html

This is the layout for all the pages on the website - it will include buttons on the top showing the homepage, quiz page, and logout

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<title>**{% block title %}{% endblock %}**</title>**
6. **<link** rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}"**>**
7. **<link** rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.1/css/all.css"**>**
8. **</head>**
9. **<body** class="loggedin"**>**
10. **<nav** class="navtop"**>**
11. **<div>**
12. **<h1>**CS Quiz Website**</h1>**
13. **<a** href="{{ url\_for('home') }}"**><i** class="fas fa-home"**></i>**Home**</a>**
14. **<a** href="{{ url\_for('quiztopic') }}"**><i** class="fas fa-question-circle"**></i>**Quizzes**</a>**
15. **<a** href="{{ url\_for('flashcard') }}"**><i** class="fas fa-id-card-alt"**></i>**Flashcards**</a>**
16. **<a** href="{{ url\_for('profile') }}"**><i** class="fas fa-user-circle"**></i>**Profile**</a>**
17. **<a** href="{{ url\_for('logout') }}"**><i** class="fas fa-sign-out-alt"**></i>**Logout**</a>**
18. **</div>**
19. **</nav>**
20. **<div** class="content"**>**
21. {% block content %}{% endblock %}
22. **</div>**
23. **</body>**
24. **</html>**

### 

### index.html

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<title>**Login**</title>**
6. **<link** rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}"**>**
7. **<link** rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.1/css/all.css"**>**
8. **</head>**
9. **<body>**
10. **<div** class="login"**>**
11. **<h1>**Login**</h1>**
12. **<div** class="links"**>**
13. **<a** href="{{ url\_for('login') }}" class="active"**>**Login**</a>**
14. **<a** href="{{ url\_for('register') }}"**>**Register**</a>**
15. **</div>**
16. **<form** action="{{ url\_for('login') }}" method="post"**>**
17. **<label** for="username"**>**
18. **<i** class="fas fa-user"**></i>**
19. **</label>**
20. **<input** type="text" name="username" placeholder="Username" id="username" required**>**
21. **<label** for="password"**>**
22. **<i** class="fas fa-lock"**></i>**
23. **</label>**
24. **<input** type="password" name="password" placeholder="Password" id="password" required**>**
25. **<div** class="msg"**>**{{ msg }}**</div>**
26. **<input** type="submit" value="Login"**>**
27. **</form>**
28. **</div>**
29. **</body>**
30. **</html>**

### 

### register.html

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<title>**Register**</title>**
6. **<link** rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}"**>**
7. **<link** rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.1/css/all.css"**>**
8. **</head>**
9. **<body>**
10. **<div** class="register"**>**
11. **<h1>**Register**</h1>**
12. **<div** class="links"**>**
13. **<a** href="{{ url\_for('login') }}"**>**Login**</a>**
14. **<a** href="{{ url\_for('register') }}" class="active"**>**Register**</a>**
15. **</div>**
16. **<form** action="{{ url\_for('register') }}" method="post" autocomplete="off"**>**
17. **<label** for="username"**>**
18. **<i** class="fas fa-user"**></i>**
19. **</label>**
20. **<input** type="text" name="username" placeholder="Username" id="username" required**>**
21. **<label** for="password"**>**
22. **<i** class="fas fa-lock"**></i>**
23. **</label>**
24. **<input** type="password" name="password" placeholder="Password" id="password" required**>**
25. **<label** for="email"**>**
26. **<i** class="fas fa-envelope"**></i>**
27. **</label>**
28. **<input** type="email" name="email" placeholder="Email" id="email" required**>**
29. **<div** class="msg"**>**{{ msg }}**</div>**
30. **<input** type="submit" value="Register"**>**
31. **</form>**
32. **</div>**
33. **</body>**
34. **</html>**

### home.html

1. {% extends 'layout.html' %}
3. {% block title %}Home{% endblock %}
5. {% block content %}
6. **<h2>**Home Page**</h2>**
7. **<p>**Welcome back, {{ username }}!**</p>**
8. {% endblock %}

### 

### profile.html

1. {% extends 'layout.html' %}
3. {% block title %}Profile{% endblock %}
5. {% block content %}
6. **<h2>**Profile Page**</h2>**

9. **<div>**
10. **<p>**Your account details are below:**</p>**
11. **<table>**
12. **<tr>**
13. **<td>**Username:**</td>**
14. **<td>**{{ account['username'] }}**</td>**
15. **</tr>**
16. **<tr>**
17. **<td>**Email:**</td>**
18. **<td>**{{ account['email'] }}**</td>**
19. **</tr>**
20. **</table>**
21. **</div>**
23. {% endblock %}

### 

### quiz.html

1. <!DOCTYPE html**>**
2. {% extends 'layout.html' %}
4. {% block title %}Quizzes{% endblock %}
6. {% block content %}
8. **<div** id="quiz" class="col-xs-offset-3 col-xs-6"**>**
9. **<h2>**Quizzes**</h2>**

12. **<h3>**
13. **<label** for="Topics"**>**Choose a topic:**</label>**
14. **</h3>**
15. **<form** class="form-inline" method="POST" action="{{ url\_for('quiztopic') }}"**>**
16. **<div** class="form-group"**>**
17. **<div** class="input-group"**>**
18. **<select** name="Topics" class="selectpicker form-control"**>**
19. {% for i in topics %}
20. **<option** value="{{i}}"**>**{{i}}**</option>**
21. {% endfor %}
22. **</select>**
23. **</div>**
24. **<p></p>**
25. **<button** type="submit" class="btn btn-default"**>**Go**</button>**
26. **</div>**
27. **</form>**
28. **</div>**
30. {% endblock %}

### 

### question.html

### {% extends "layout.html" %}

### 

### {% block head %}

### **<style>**

### #exit {

### /\*align-items: right;\*/

### /\*color: black;\*/

### /\*background-color: lightskyblue;\*/

### }

### #question {

### border: solid 1px black;

### }

### 

### .button {

### background-color: powderblue;

### }

### **</style>**

### {% endblock %}

### {% block content %}

### **<div** id="question" class="col-xs-offset-3 col-xs-6"**>**

### 

### **<h1>**Question:**<h1>**

### **<h4>**

### {%for i in questions%}

### **<li>**{{i}}**</li>**

### {% endfor %}

### **</h4><br>**

### **<form** action='/quiz/{{ select }}' method='POST'**>**

### {%for i in answers%}

### **<button** type="submit" name="useranswer" value={{ i }}

### class="btn btn-info btn-lg btn-block" style="width:100%"**>**{{ i }}**</button><br>**

### {% endfor %}**<br>**

### **</form>**

### 

### **</div>**

### 

### {% endblock %}

### 

### {% block js %}

### **<script>**

### $("#dash").hide();

### **</script>**

### {% endblock %}

### flashcard.html

1. <!DOCTYPE html**>**
2. {% extends 'layout.html' %}
4. {% block title %}Flashcards{% endblock %}
6. {% block content %}
7. **<html>**
8. **<h2>**Profile Page**</h2>**
10. **<head>**
11. **<style>**
12. table {
13. font-family: arial, sans-serif;
14. border-collapse: collapse;
15. width: 100%;
16. }
18. td, th {
19. border: 1px solid #dddddd;
20. text-align: left;
21. padding: 8px;
22. }
24. tr:nth-child(even) {
25. background-color: #dddddd;
26. }
27. **</style>**
28. **</head>**
30. **<body>**
31. **<table>**
32. **<tr>**
33. **<th>**Flashcards**</th>**
34. **</tr>**
36. {%for i in topics%}
37. **<tr>**
38. **<td>**{{i}}**</td>**
39. **</tr>**
40. {%endfor%}
41. **</table>**
43. {% endblock %}

### style.css

1. \* {
2. box-sizing: border-box;
3. **font-family**: -apple-system, BlinkMacSystemFont, "segoe ui", roboto, oxygen, ubuntu, cantarell, "fira sans", "droid sans", "helvetica neue", Arial, sans-serif;
4. **font-size**: 16px;
5. -webkit-font-smoothing: antialiased;
6. -moz-osx-font-smoothing: grayscale;
7. }
8. body {
9. **background-color**: #435165;
10. **margin**: 0;
11. }
12. .login, .register {
13. **width**: 400px;
14. **background-color**: #ffffff;
15. box-shadow: 0 0 9px 0 rgba(0, 0, 0, 0.3);
16. **margin**: 100px auto;
17. }
18. .login h1, .register h1 {
19. **text-align**: center;
20. **color**: #5b6574;
21. **font-size**: 24px;
22. **padding**: 20px 0 20px 0;
23. **border-bottom**: 1px solid #dee0e4;
24. }
25. .login .links, .register .links {
26. **display**: flex;
27. **padding**: 0 15px;
28. }
29. .login .links a, .register .links a {
30. **color**: #adb2ba;
31. **text-decoration**: none;
32. **display**: inline-flex;
33. **padding**: 0 10px 10px 10px;
34. **font-weight**: bold;
35. }
36. .login .links a:hover, .register .links a:hover {
37. **color**: #9da3ac;
38. }
39. .login .links a.active, .register .links a.active {
40. **border-bottom**: 3px solid #3274d6;
41. **color**: #3274d6;
42. }
43. .login form, .register form {
44. **display**: flex;
45. flex-wrap: wrap;
46. justify-**content**: center;
47. **padding-top**: 20px;
48. }
49. .login form label, .register form label {
50. **display**: flex;
51. justify-**content**: center;
52. align-items: center;
53. **width**: 50px;
54. **height**: 50px;
55. **background-color**: #3274d6;
56. **color**: #ffffff;
57. }
58. .login form input[type="password"], .login form input[type="text"], .login form input[type="email"], .register form input[type="password"], .register form input[type="text"], .register form input[type="email"] {
59. **width**: 310px;
60. **height**: 50px;
61. **border**: 1px solid #dee0e4;
62. **margin-bottom**: 20px;
63. **padding**: 0 15px;
64. }
65. .login form input[type="submit"], .register form input[type="submit"] {
66. **width**: 100%;
67. **padding**: 15px;
68. **margin-top**: 20px;
69. **background-color**: #3274d6;
70. **border**: 0;
71. **cursor**: pointer;
72. **font-weight**: bold;
73. **color**: #ffffff;
74. transition: background-color 0.2s;
75. }
76. .login form input[type="submit"]:hover, .register form input[type="submit"]:hover {
77. **background-color**: #2868c7;
78. transition: background-color 0.2s;
79. }
80. .navtop {
81. **background-color**: #2f3947;
82. **height**: 60px;
83. **width**: 100%;
84. **border**: 0;
85. }
86. .navtop div {
87. **display**: flex;
88. **margin**: 0 auto;
89. **width**: 1000px;
90. **height**: 100%;
91. }
92. .navtop div h1, .navtop div a {
93. **display**: inline-flex;
94. align-items: center;
95. }
96. .navtop div h1 {
97. flex: 1;
98. **font-size**: 24px;
99. **padding**: 0;
100. **margin**: 0;
101. **color**: #eaebed;
102. **font-weight**: normal;
103. }
104. .navtop div a {
105. **padding**: 0 20px;
106. **text-decoration**: none;
107. **color**: #c1c4c8;
108. **font-weight**: bold;
109. }
110. .navtop div a i {
111. **padding**: 2px 8px 0 0;
112. }
113. .navtop div a:hover {
114. **color**: #eaebed;
115. }
116. body.loggedin {
117. **background-color**: #f3f4f7;
118. }
119. .content {
120. **width**: 1000px;
121. **margin**: 0 auto;
122. }
123. .content h2 {
124. **margin**: 0;
125. **padding**: 25px 0;
126. **font-size**: 22px;
127. **border-bottom**: 1px solid #e0e0e3;
128. **color**: #4a536e;
129. }
130. .content > p, .content > div {
131. box-shadow: 0 0 5px 0 rgba(0, 0, 0, 0.1);
132. **margin**: 25px 0;
133. **padding**: 25px;
134. **background-color**: #fff;
135. }
136. .content > p table td, .content > div table td {
137. **padding**: 5px;
138. }
139. .content > p table td:first-child, .content > div table td:first-child {
140. **font-weight**: bold;
141. **color**: #4a536e;
142. **padding-right**: 15px;
143. }
144. .content > div p {
145. **padding**: 5px;
146. **margin**: 0 0 10px 0;
147. }

# 

# 

# Testing

I need to begin testing by making sure that every button on every screen works. I also need to ensure that you can enter simple text into each field without crashing the software. I will be using test tables and user data testing to make sure the program is working successfully,

## Test Tables

Email

| **Test Number** | **Test Type** | **Test Data** | **Expected Result** | **Actual Result** | **Corrections** |
| --- | --- | --- | --- | --- | --- |
| 1 | Valid | Email = test@test.com | Should be valid so send to home page | Valid - states that user has “successfully registered” | No corrections needed |
| 2 | Valid | Email = test@test.co.uk | Should be valid so send to home page | Valid - states that user has “successfully registered” | No corrections needed |
| 3 | Invalid | Email = @test.a | Should be invalid so tells user to try again | Invalid - tells user “test.a is incomplete” | No corrections needed |
| 4 | Invalid | Email = 3.test@abc | Should be invalid so tells user to try again | Invalid - tells the user “invalid email address!” | No corrections needed |
| 5 | Null Value | Email = nothing entered | Should tell user to fill out the field | Tells user to fill out the field | No corrections needed |

| **Test** | **Test Result Screen Captures** |
| --- | --- |
| Email = test@test.com |  |
| Email = test@test.co.uk |  |
| Email = @test.a |  |
| Email = 3.test@abc |  |
| Email = nothing entered |  |

# 

Username

| **Test Number** | **Test Type** | **Test Data** | **Expected Result** | **Actual Result** | **Corrections** |
| --- | --- | --- | --- | --- | --- |
| 1 | Valid | Username = hi | Should be valid so send to home page | Valid - sends to home page | No corrections needed |
| 2 | Valid | Username = testing1234567890123456 | Should be valid so send to home page | Valid - sends to home page | No corrections needed |
| 3 | Invalid | Username = . | Should be invalid so tells user to try again | Invalid - states “Username must contain only characters and numbers!” | No corrections needed |
| 4 | Null Value | Username = nothing entered | Should tell user to fill out the field | Tells user to fill out the field | No corrections needed |

# 

| **Test** | **Test Result Screen Captures** |
| --- | --- |
| Username = hi |  |
| Username = testing1234567890123456 |  |
| Username = . |  |
| Username = nothing entered |  |

## 

## User Data Testing

| **Test** | **Test Result Screen Captures** |
| --- | --- |
| When logged in as a student it only shows the specific students data |  |
| When picking a topic from the quizzes, only questions from the topic are shown |  |
| User logs out successfully when log out button clicked |  |

# 

# 

# Evaluation

## Achieving Objectives

| **Objectives** | **Achieved?** |
| --- | --- |
| Create a website that allows people to login/create an account | Yes - Users are able to register with a unique username, users must input a username, password and email to reister. Users are then able to login with their username and password. |
| Maximise security | Yes - passwords are hashed using a hashing algorithm through bcrypt. It uses a randomly generated SALT to fully maximise security. |
| Showing Progress | Partially - Users are able to check their account details as of now. Will add dynamically changing graphs later to track progress. |
| Creating/Editing Quizzes | Not yet |
| Completing Quizzes | Partially - Users are able to pick a quiz they want to do according to topic. Currently two topics are available. |
| Creating/Completing Flashcards | Not yet |

## 

## Potential for Future Development

If I had more time to complete the project, I would create the flashcards and allow others to create new ones for others to use.