AUTOMATED MCQ

(INFSCI2560: WEB TECHNOLOGIES AND STANDARDS)

A PROJECT BY GROUP11

A COURSE PROJECT BY:

- Prathamesh Marathe
- Sarthak Killedar
- Shanim Manzoor
- Vishal Karande

INTRODUCTION:

Automated MCQ is a concept inspired from a very popular app called QuizUp. In today's day and age, self-motivated education is becoming the norm, where people from all walks of life learn something from the comfort of their own home. With this in mind, we are beginning to see a major change in the education sector as well as a burgeoning requirement for websites where users could test themselves on various topics.

A major advantage of today's web is how everything is connected in a similar pattern to hubs/authorities in a network, this makes it easy to refer to other websites from one single platform if needed by using useful outbound links.

Another key benefit is the way that topics can be streamlined as well as users can compare themselves to peers to see how they rank and where they would need to improve to get the work done.

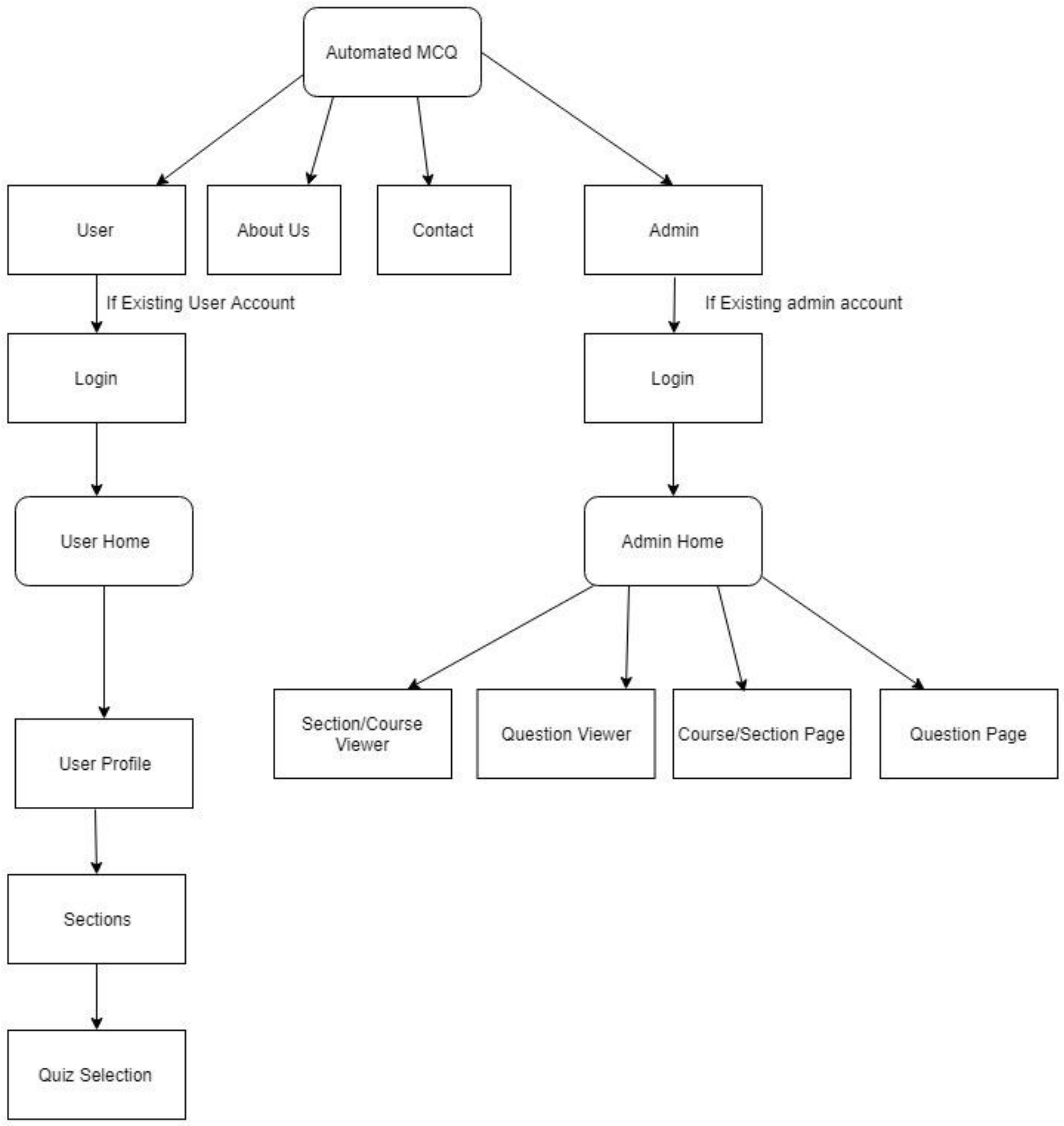
The timeline of the project would involve the following:

1. Build a template for the website indicating the user requirements as well as the basic wireframe for the user interface

2. Start building the backend for both user and admin taking into consideration how the flow of data between the frontend and

   backend could be best optimized
3. Build the frontend and start building iteratively for the different views that the user and admin would need.
4. Test the interface and server connections.
5. Test the implementation for the user and check for bugs and improve the simplicity of user interface wherever possible.

FIGURE 1: Automated MCQ : A flow Diagram



OBJECTIVE:

- The aim of the project was to build a website which would allow users to test their proficiency in various subjects in the form of quizzes.
- These quizzes would be randomly generated from a database where they would be assigned the difficulty level and the format of the questions would be of multiple-choice ones.
- Build a website with a clean interface allowing for users to have a simpler time using the website.
- The Admin will have the ability to create content based on various topics henceforth called courses and subtopics called sections
- These sections will have quizzes with 3 difficulty levels namely easy, moderate and hard
- The user will be able to choose the quiz depending on his familiarity, once he's done with the quiz, he will be able to submit his answers and receive his score on his user profile

TEAM MEMBER’S CONTRIBUTION:

PRATHAMESH MARATHE:

- Frontend Contribution: Admin side frontend using AngularJS
- Backend Contribution: User side backend using Express and server connection using Mongoose (for MongoDB)
- Attended project meetings to discuss flow of website
- Built the static pages such as Contact us,About etc.

SARTHAK KILLEDAR:

- Frontend Contribution: Admin side frontend using AngularJS, login and registration capabilities for both user and admin.
- Backend Contribution: User side backend using Express and server connection using Mongoose(for MongoDB)
- Attended project meetings to discuss flow of website.
- Started the repo on Azure for the project

SHANIM MANZOOR:

- Frontend Contribution: User side frontend (mainly the profile page) using AngularJS

- Backend Contribution: Admin side backend discussions and structure with Vishal
- Attended project meetings to discuss flow of website as well as was in charge of documentation of the project

VISHAL KARANDE:

- Frontend Contribution: User side frontend using AngularJS and ascertaining the interface flow from the homepage
- Backend Contribution: Admin Side backend built using Express and Mongoose for Server Connection
- Attended Project meetings to discuss flow of website
- Merging of User and Admin side mostly done by Vishal

TECHNOLOGY USED:

- Static Webpages: HTML, CSS(using Bootstrap)
- Frontend: AngularJS
- Backend: Express and Node
- Database: MongoDB
- Javascript for basic functions in controllers etc.

AngularJS:

AngularJS (also written as Angular.js) is a JavaScript-based open-source front-end web application framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications. The JavaScript components complement Apache Cordova, a framework used for developing cross-platform mobile apps. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–viewmodel (MVVM) architectures, along with components commonly used in rich Internet applications. (This flexibility has led to the acronym MVW, which stands for "model-view-whatever" and may also encompass model–view–presenter and model–view–adapter.) In 2014, the original AngularJS team began working on the Angular application platform.

[https://en.wikipedia.org/wiki/AngularJS]

Express:

Express is the most popular Node web framework, and is the underlying library for a number of other popular Node web frameworks. It provides mechanisms to:

- Write handlers for requests with different HTTP verbs at different URL paths (routes).
- Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
- Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
- Add additional request processing "middleware" at any point within the request handling pipeline.

  [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction]

Node.JS:

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. npm is a Node.js package of open source library which is largest in the world.

It’s an asynchronous event driven JavaScript runtime, which is designed to build scalable network applications.It can handle many concurrent connections at a time, where when connection request are made concurrently for each connection a callback is fired .If there is no task to be performed Node will go to sleep.

Node.js connection handling mechanism is super efficient than our existing classical thread based model Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node are free from worries of dead-locking the process, since there are no locks. Almost no function in Node directly performs I/O, so the process never blocks. Because nothing blocks, scalable systems are very reasonable to develop in Node.

[https://codeburst.io/getting-started-with-node-js-a-beginners-guide-b03e25bca71b]

MongoDB:

MongoDB is a document-oriented database, not a relational one. Non relational means it does not store data in tables but in the form of JSON document. The primary reason for moving away from the relational model is to make scaling out easier. Here in MongoDB row is replaced with document and table is replaced with collection. It can be thought of as a group of documents.

Document is the basic unit of data for MongoDB, roughly equivalent to a row. It is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values maybe documents, arrays, and arrays of documents.

[https://www.codeproject.com/Articles/828392/Basics-of-MongoDB]

mongoDB®

Dependencies:

- "angular-animate": "^1.7.5",
- "angular-chart.js": "^1.1.1",
- "angular-cookies": "^1.7.5",
- "angular-route": "^1.7.5",
- "bootstrap": "3.3.7",
- "jquery": "^3.3.1"

Developer Dependencies:

- "@uirouter/angularjs": "^1.0.20",
- "angular": "^1.7.5",
- "angular-formly": "^8.4.1",

- "angular-formly-templates-bootstrap": "^6.5.1",
- "angular-ui-bootstrap": "^2.5.6",
- "api-check": "^7.5.5",
- "async": "^2.6.1",
- "body-parser": "^1.18.3",
- "cors": "^2.8.4",
- "crypto": "^1.0.1",
- "express": "^4.16.4",
- "express-jwt": "^5.3.1",
- "jsonwebtoken": "^8.3.0",
- "mongoose": "^5.3.8",
- "mongoose-unique-validator": "^2.0.2",
- "nodemon": "^1.18.7",
- "passport": "^0.4.0",
- "passport-local": "^1.0.0",
- "path": "^0.12.7",
- "yarn": "^1.12.1"

SETUP:

- The computer should have MongoDB installed
- The computer should have an IDE called **Webstorm** installed
- Once the source code is extracted, run **npm install** .This will install all the dependencies from the package.json file
- The server.js file will specify the Port (3000 usually) .
- To start the project use **npm run start** command.

- Type out Localhost:3000 in the web browser, preferably google chrome to view the website.

CHALLENGES:

TEAM:

- Communication issues at times due to modularity in terms of roles for team members

FRONTEND:

- Lot of the functionality originally planned such as leaderboards and moderation control had to be removed due to increasing complexity in design as well as time constraints
- Push notifications as well as RSS feeds had to be removed from the final template for the website.
- Lot of testing and trials were needed to get the synchronization between the frontend and backend

BACKEND:

- Some of the models were a bit complicated to suit the requirement and this made some of the coding in the fronted a lot tougher and time consuming

CONCLUSION:

- The project was excellent in allowing for an understanding of the basics of MEAN stack implementation of web application
- The project allowed for the understanding of the ins and outs of a web development project and how time intensive it can get as well as the importance of design of user interface flow beforehand to make the development easier.

FUTURE IMPROVEMENTS:

- Implementing **User Scoreboards**
- Blogs and learning resources
- User profile page improvements, More Random Quizzes