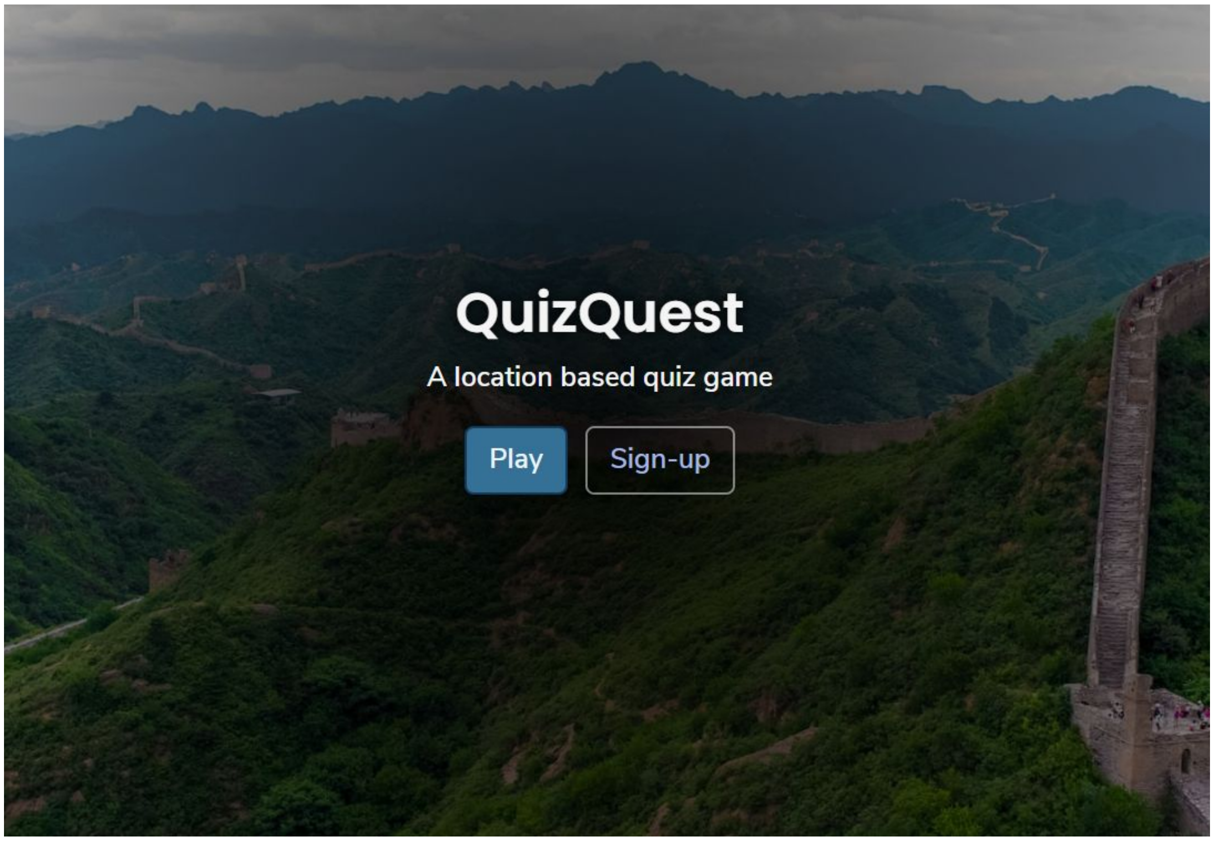
Score : 93/100
Some formatting issues has to corrected

QuizQuest



A location based browser game

Prepared by
Group 8
Jonathon Repta, Edward Liang, Jennifer Alonso, Amal Syed
for use in CS 440
at the
University of Illinois Chicago

November 2020

Table of Contents

List of Figures

List of Tables

I Project Description

1 Project Overview

QuizQuest is a free to play game that runs in the browser. *QuizQuest* combines traditional quiz gameplay with location based features that encourage users to explore locations both near and far. Furthermore, *QuizQuest* has unique community features which encourage users to compete for the highest score.

2 Project Domain

There isn't much that needs to be understood to evaluate QuizQuest. The online quiz format should be familiar to anyone who has played a similar online game. The only possible background one might need to understand the project is an understanding of geography and perhaps some knowledge of locations featured in the game so that one can fact check information on each quiz.

3 Relationship to Other Documents

For our implementation of QuizQuest, we strayed somewhat from the design specified in the documentation originally written by Tsz Lam, Tan Le, Anthony Nedumgottil, Weiheng Ruan. We based our project almost entirely upon the details laid out in Group 2's (Fall 2016) description summary document. We were unable to use more specific details from any other supplementary documents they provided due to the fact that we implemented our game using web technologies rather than mobile ones. Despite this, we nonetheless implemented nearly every feature documented in their description summary.

4 Naming Conventions and Definitions

QuizQuest has three user classes/ranks. A "moderator" user class, "user" user class, and "f2p" user class. The "f2p" user class refers to a class of user which does not have an account on the site and instead is "free-to-play". These users have the ability to view the leaderboard/user profiles, browse quizzes, and take quizzes. The "user" class refers to a class of users which has created an account. A "user" has access to every feature the "f2p" class does, plus they are able to submit quizzes and their information is stored within the quiz database. The moderator class is a class of users which has every privilege the "user" class does, plus they have the ability to approve/reject quizzes that have been submitted for approval. Furthermore, they also have the ability to download a copy of the database so that changes can be made manually.

QuizQuest should be written in "camel case" and contain no spaces between "quiz" and "quest".

4a Definitions of Key Terms

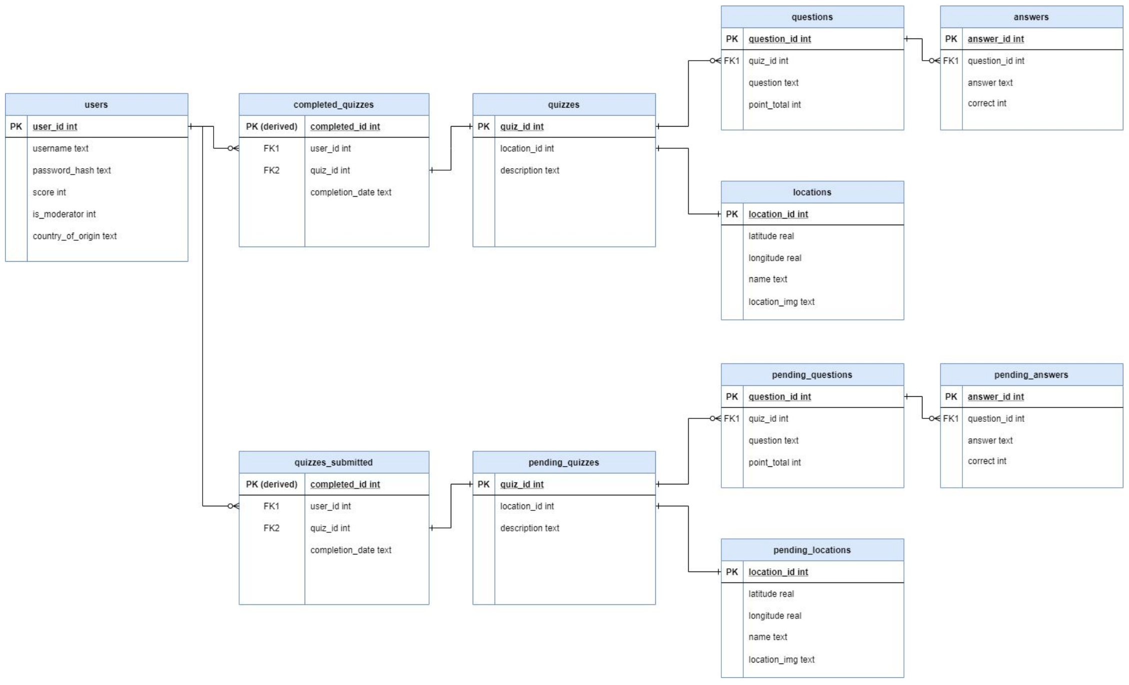
Location: A location in quizquest describes a specific point of interest that has a name, an image, and a coordinate pair associated with it. Each location may have multiple quizzes associated with it, but each quiz may only be associated with one location (one to many relationship).

Quiz: A quiz describes a specific set of questions/answers that are associated with a given location. A quiz may be associated with only a single location. Each quiz may have multiple questions associated with it, but each question may only be associated with one quiz (one to many relationship).

4b UML and Other Notation Used in This Document

We are using an entity model relationship diagram to document the relationships between each table in our database. See 4c for a specific diagram describing said relationships.

4c Data Dictionary for Any Included Models



This diagram shows the entity model relationship for every table in our database. Due to the limitations of SQLite, booleans are represented in a C-style (1 represents true, 0 represents false). We have a mirror of the quiz/locations tables in order to isolate pending quizzes from approved ones.

Images have three distinct sizes. “Original”, “1280”, and “200”. Original refers to the full resolution images stored on Wikipedia's servers and are the only links stored in the *QuizQuest* database. The original image is stored so that other sizes can be generated on the fly. “1280” is the image size used for each distinct quiz page (ie, /quiz/1). “200” is the image size used for the thumbnails of each quiz on the explore page. All of the aforementioned images are stored as jpegs.

Figure 1- model for database

II Project Deliverables

Over the course of the semester, we have worked to complete a fully functioning full stack web application. By the release displayed during class (the final release), we had a project which contained a quiz GUI, a map GUI, a leaderboard, routes for user profiles, user account creation, a GUI for submitting your own quiz, a GUI for moderators to approve quizzes, and dedicated pages for error handling. Furthermore, we created plenty of content for the game, including 20 locations, 20 quizzes, 110 questions, and 450 answers. On top of this, we made our application completely mobile friendly and included accessibility features for the entire site. The site can be accessed and played via a version hosted on Heroku.

5 First Release

Our first release was on Friday October 4th and covered the first four weeks of development. For the first release, we completed the fundamentals of the game. This included the boilerplate for the site’s HTML/CSS, a functioning map GUI, necessary tables in the database, several quizzes for the game, and a functioning quiz GUI.

6 Second Release

Our second release was on Friday November 1st and covered the following four weeks of development. For the second release, we flushed out many features of the site. Between the first and second release, we handled account creation/log in (with properly encrypted passwords and secure session storage), created routes for each user profile, error handling so that specific , and we improved how quiz gameplay was implemented. Furthermore, we transitioned our site from a statically/client side rendered application to one that’s rendered server side.

7 Comparison with Original Project Design Document

In comparison with the design detailed by Group 2 (Fall 2016), our game runs within the browser, rather than natively on mobile. Furthermore, we use mapbox, leaflet.js, and openstreetmaps for map information instead of Google Maps. Because of these choices, our game does not require dedicated GPS hardware and instead relies upon IP information and the browsers Geolocation API to derive the users location.

III Testing

8 Items to be Tested

Database Helper Functions:

ID: 1

Description: These functions are used by the server to insert user-submitted quizzes into the database.

API endpoints:

ID: 2

Description: Test the API endpoints which allow the client and server to interact.

Sign In/Sign up Page Frontend:

ID: 3

Description: Users can sign up and sign in to the website properly by posting a form to the server.

Submit Quiz Frontend:

ID: 4

Description: Users can submit a quiz to the server so that it can be approved by a moderator.

9 Test Specifications

ID# 1.1 - Insert Empty Quiz

Description: Test the database helpers by attempting to insert an empty quiz.

Items covered by this test: Database helper functions

Requirements addressed by this test: This test addresses the data storage requirements.

Environmental needs: This test requires QuizQuestionAnswer.js to run.

Intercase Dependencies: N/A

Test Procedures: Run the test_db_function.js file.

Input Specification: This test's input is a quiz object with an empty questions list.

Output Specifications: The output should be a single quiz with no questions inserted into the pending_quizzes table of the QuizQuest database. The test itself will detect if the quiz has been added or not and then output if it found the quiz in the database or not. It will then delete the quiz.

Pass/Fail Criteria: To pass the test, the quiz must be inserted into the database and then detected by the test. The test fails if the code it's calling throws an exception or it simply can't find the quiz in the database.

ID# 1.2 - Insert Quiz with Empty Question

Description: Test the database helpers by attempting to insert a quiz with questions, but whose questions have no answers.

Items covered by this test: Database helper functions

Requirements addressed by this test: This test addresses the data storage requirements.

Environmental needs: This test requires QuizQuestionAnswer.js to run.

Intercase Dependencies: The Insert Empty Quiz test must pass for this test to run.

Test Procedures: Run the test_db_function.js file.

Input Specification: This test's input is a quiz object with a single question in its question list. This question will have no associated answers.

Output Specifications: The output should be a single quiz inserted into the pending_quizzes table of the QuizQuest database, as well as a single question with no answers inserted into the pending_questions database. The test itself will detect if the quiz has been added or not and then output if it found the quiz in the database or not. It will do the same for the question. It will then delete both.

Pass/Fail Criteria: To pass the test, the quiz must be inserted into the database and then detected by the test. The test fails if the code it's calling throws an exception or it simply can't find the quiz in the database.

ID# 1.3 - Insert Quiz with Question and Answer

Description: Test the database helpers by attempting to insert a quiz with a single question which itself has a single answer.

Items covered by this test: Database helper functions

Requirements addressed by this test: This test addresses the data storage requirements.

Environmental needs: This test requires QuizQuestionAnswer.js to run.

Intercase Dependencies: The Insert Empty Quiz and Insert Quiz with Empty Question tests must pass for this test to run.

Test Procedures: Run the test_db_function.js file.

Input Specification: This test's input is a quiz object with a single question in its question list. This question will have a single associated answer..

Output Specifications: The output should be a single quiz inserted into the pending_quizzes table of the QuizQuest database, as well as a single question inserted into the pending_questions table and a single answer inserted into the pending_answers table. The test itself will detect if the quiz has been added or not and then output if it found the quiz in the database or not. It will do the same for the question and answer. It will then delete all of them.

Pass/Fail Criteria: To pass the test, the quiz must be inserted into the database and then detected by the test. The test fails if the code it's calling throws an exception or it simply can't find the quiz in the database.

<u>ID# 2.1 - "Location" api endpoints</u>

Description: Test answer api endpoints by requesting them with a combination of valid and invalid options.

Items covered by this test: /api/locations/* endpoints

Requirements addressed by this test: This test addresses the fetching of data for the rendering of the client side quiz GUI. These routes also cover the updating of session information.

Environmental needs: This test requires the server to be running locally

Intercase Dependencies: None

Test Procedures: Use Python's "requests" library with the PyTest framework to programmatically make requests to the answers endpoints

Input Specification: A combination of requests with valid quiz ids, invalid quiz ids, and invalid parameters

Output Specifications: The output should consist of a JSON file containing all relevant information as well as a matching status code

Pass/Fail Criteria: Items pass if the data retrieved matches the csv files used for insertion. Items fail if the data retrive does not match.

ID# 2.2 - “Quiz” api endpoints

Description: Test quiz api endpoints by requesting them with a combination of valid and invalid options.

Items covered by this test: /api/quizzes/* endpoints

Requirements addressed by this test: This test addresses the fetching of data for the rendering of the client side quiz GUI.

Environmental needs: This test requires the server to be running locally

Intercase Dependencies: None

Test Procedures: Use Python's "requests" library with the PyTest framework to programmatically make requests to the quiz endpoints

Input Specification: A combination of requests with valid quiz ids, invalid quiz ids, and invalid parameters

Output Specifications: The output should consist of a JSON file containing all relevant information as well as a matching status code

Pass/Fail Criteria: Items pass if the data retrieved matches the csv files used for insertion. Items fail if the data retrive does not match.

ID# 2.3 - “Answer” api endpoints

Description: Test answer api endpoints by requesting them with a combination of valid and invalid options.

Items covered by this test: /api/answers/* endpoints

Requirements addressed by this test: This test addresses the fetching of data for the rendering of the client side quiz GUI. These routes also cover the updating of session information.

Environmental needs: This test requires the server to be running locally and for a recent version of Python3 to be installed

Intercase Dependencies: None

Test Procedures: Use Python's "requests" library with the PyTest framework to programmatically make requests to the answers endpoints

Input Specification: A combination of requests with valid quiz ids, invalid quiz ids, and invalid parameters

Output Specifications: The output should consist of a JSON file containing all relevant information as well as a matching status code

Pass/Fail Criteria: Items pass if the data retrieved matches the csv files used for insertion. Items fail if the data retrive does not match.

ID# 3 - Sign In/Sign up Frontend

Description: Test whether an account is successfully created and users are able to sign in.

Items covered by this test: Front End Sign-In and Sign-Up page

Requirements addressed by this test: This test addresses the frontend services for the Sign-Up and Sign-In page

Environmental needs: Selenium

Intercase Dependencies: Backend services should be functioning properly and storing user info in the server database

Test Procedures: Input Specification: Run the SignIn_SignUp.py file

Input Specification: test does not require any inputs

Output Specifications: For this test to be successful the py file should be able to locate the <a> tag with class name "btn btn-outline-primary nav-link ml-2 pl-2 pr-2" and logout. This indicates the file success in signing up, signing in and logging out.

Pass/Fail Criteria: An exception will be thrown. This probably happens when certain html elements can not be located by the file.

ID# 4 - Submit Quiz Frontend

Description: Test whether a quiz is successfully submitted

Items covered by this test: Front End submit quiz page

Requirements addressed by this test: This test addresses the frontend services for the submit quiz page

Environmental needs: selenium

Intercase Dependencies: backend services should be functional so that the quiz information can be stored in the database for unapproved quizzes

Test Procedures: run the submit_quiz.py

Input Specification: test does not require any inputs

Output Specifications: The test will not generate any exceptions and be able to approve the quiz in the approve quiz page just for admins. This indicates the success in submitting a quiz from the submit quiz page. The resulting dummy test will be viewable in the map somewhere.

Pass/Fail Criteria: An exception will be thrown. THis probably happens when certain html elements can not be located by the file.

10 Test Results

ID# - 1.1 Insert Empty Quiz

Date(s) of Execution: 11/27/2020..

Staff conducting tests: Amal Syed

Expected Results: The test driver to output, "Passed all tests."

Actual Results: The test driver outputted, "Passed all tests."

Test Status: Pass.

ID# - 1.2 Insert Quiz with Empty Question

Date(s) of Execution: 11/27/2020..

Staff conducting tests: Amal Syed

Expected Results: The test driver to output, "Passed all tests."

Actual Results: The test driver outputted, "Passed all tests."

Test Status: Pass.

ID# - 1.3 Insert Quiz with Question and Answer

Date(s) of Execution: 11/27/2020..

Staff conducting tests: Amal Syed

Expected Results: The test driver to output, "Passed all tests."

Actual Results: The test driver outputted, "Passed all tests."

Test Status: Pass.

ID# - 2.1 Fetch information from "locations" endpoints

Date(s) of Execution: 11/28/2020

Staff conducting tests: Jonathon Repta

Expected Results: For PyTest to output, “Passed all tests.”

Actual Results: PyTest outputted, “Passed all tests.”

Test Status: Pass

ID# - 2.2 Fetch information from "quizzes" endpoints

Date(s) of Execution: 11/28/2020

Staff conducting tests: Jonathon Repta

Expected Results: For PyTest to output, “Passed all tests.”

Actual Results: PyTest outputted, “Passed all tests.”

Test Status: Pass

ID# - 2.3 Fetch information from "answers" endpoints

Date(s) of Execution: 11/28/2020

Staff conducting tests: Jonathon Repta

Expected Results: For PyTest to output, “Passed all tests.”

Actual Results: PyTest outputted, “Passed all tests.”

Test Status: Pass

ID# - 3 Sign In/Sign up Frontend

Date(s) of Execution: 11/27/2020

Staff conducting tests: Jennifer Alonso

Expected Results: A py file generating a new account that can sign in through the server

Actual Results: The py file generated a loggable new account

Test Status: Pass

ID# - 4 Submit Quiz Frontend

Date(s) of Execution: 11/27/2020

Staff conducting tests: Edward Liang

Expected Results: A new dummy quiz generated and approved by the py file

Actual Results: The newly generated and approved quiz is now playable on the map

Test Status: Pass

11 Regression Testing

No regression testing was done.

IV Inspection

12 Items to be Inspected

- Database Helper Functions
- API endpoints
- Sign In/Sign up Page Frontend
- Submit Quiz Frontend

13 Inspection Procedures

Code Quality	Yes	No	comments
Does the site use valid css?	yes		N/A
Does the site use valid html?	yes		N/A
Does the site use a correct doctype?	yes		N/A
Does the site have javascript errors?	yes		N/A

Does the site have any broken links?	yes		N/A
Is the code well structured?	yes		N/A
Does the site use unnecessary classes or ids?	yes		N/A
Was the site able to properly store Quizzes into the database?	yes		N/A
Was the site able to store new users into the database?	yes		N/A
Was the site able to retrieve user info from the database?	yes		N/A

Table 1- Checklist for inspection procedures

14 Inspection Results

Quiz Submission GUI: When using the quiz submission GUI, the option to remove entire questions was given to the user. This functionality had a noticeable UX flaw. When the user removed a question, it did not warn them they were about to remove a question, nor did it provide a way for users to undo said state change. This could result in accidental question deletion and frustration for the user. Inspected: 11/28/2020, by Jennifer Alonso/Edward Liang

Database helper functions: Insertion into the “pending” tables of the database is working correctly. Furthermore, insertion of data from the “pending” tables into the “public” tables works correctly. However, through code inspection, it was discovered that when a quiz is approved by moderation, it does not get fully deleted from the database. Instead, only the location is removed from the pending table. This means that the GUI does not display the pending quiz information anymore (as the query relies on missing information), however, there is residual data still stored within the database. Inspected: 11/28/2020, by Amal Syed

API endpoints: Everything is working as intended, however, code inspection revealed that there were some legacy routes still found within the codebase from a time when the site was written in a client-side rendered style. Code inspection revealed that some code cleanup could be performed. Inspected: 11/28/2020, by Jonathon Repta

Sign up/sign in: No issues were found during code inspection. The only possible avenue for improvement could be rate limiting account creation/using recaptcha to protect against bots/DDOS attacks. Inspected: 11/28/2020, by Jennifer Alonso/Edward Liang

V Recommendations and Conclusions

The **quiz submission GUI** passed it's code inspection as the GUI is functional. However, concerns about the UX of the GUI were discovered through inspection so a possible avenue for improvement was discovered.

Database helper functions passed the inspection and were functional. However, issues related to quiz insertion (quiz deletion) were discovered during the code inspection. Although these issues are by no means site breaking, left unchecked the pending tables of the database will be polluted with "pending" information that should've been deleted.

The **API endpoints** passed their code inspection, however, inspection revealed that some routes went unused. Future improvements could be made to the codebase through the removal of these unused routes.

The **sign up/sign in** displayed no obvious errors from code inspection but provided insight into ways it could be polished further.

Overall, no major errors were discovered, however, many slight improvements were revealed during code inspection.

VI Project Issues

15 Open Issues

Currently, most of our quizzes are quite easy. Typically a user would take the quiz and choose one of the four multiple choice answers. Upon choosing a wrong answer the website will present the correct answer before moving onto the next questions. This makes it very easy for anyone to get a max number of points and be on the leaderboard if they write all the answers for all the quizzes and retake it. Furthermore, our quizzes are not particularly long. Each quiz contains exactly 5 questions. Although this limit was placed on the team so that we could create a wide variety of quizzes, quizzes would need to be expanded to keep user interest in a business environment.

16 Waiting Room

There are several ways that *QuizQuest* could be expanded in the future. Most obviously, our game could fulfill the requirements of the original document and have a native mobile application alongside it. This could be done using React Native or a similar web-to-mobile application framework.

Secondly, *QuizQuest* could be improved by implementing a suite of community oriented features that take advantage of what makes the application unique. As it is implemented currently, *QuizQuest* is a fairly standard online quiz game and lacks certain features that would make it stand out in the market. In order to improve the game and make it stand out, *QuizQuest* should lean into its location based elements and utilize them to get users more involved. Essentially, *QuizQuest* should shift its focus to utilize similar features to those present in Ingress/Pokemon Go.

17 Ideas for Solutions

To increase leaderboard integrity and counter cheating by adding more questions with multiple answers and preventing any users from retaking the quiz X number of times per day.

To port *QuizQuest* to be a native mobile app someday. Re implement the site using a front end framework such as React.js so that we can easily port the application using React Native

The lack of community features might be off putting for new players. To address this, we can try adding some new features such as adding people as friends and creating friend groups, etc.

18 Project Retrospective

Overall, the development of *QuizQuest* went well. We met feature deadlines in a timely manner, developed a large suite of features, and put out a polished, professional looking product that was available for anyone to play by the presentation deadline. However, in retrospect, there are a few things that we could've focused on to improve our project. From the project's onset, we should have determined whether we wanted to use server side rendering or client side rendering for dynamic content (or both). Because we originally did not put much thought into how the site would be rendered, it led to some routes going unused and us rewriting components of our site from client side rendering to a server side rendering. Furthermore, it would have been a better use of our time to spend less time writing content for the application and instead focus on improving certain aspects of the applications UX. Finally, our application should have been tested using some sort of bottleneck on our internet/server speed. This would've given us a clearer picture of how a user might interact with the site on a non-local copy of the application.

VII Glossary

Node.js: An event driven single threaded runtime environment that executes JavaScript code outside the web browser. Allows us to use Javascript on both the front and backend.

Express.js: A flexible backend web application framework which allows for rapid prototyping.

SQLite: A library that implements a lightweight relational database and allows for databases to be stored as a flat file.

Bootstrap: CSS frontend framework for developing responsive websites

Location: A location in quizquest describes a specific point of interest that has a name, an image, and a coordinate pair associated with it. Each location may have multiple quizzes associated with it, but each quiz may only be associated with one location (one to many relationship).

Quiz: A quiz describes a specific set of questions/answers that are associated with a given location. A quiz may be associated with only a single location. Each quiz may have multiple questions associated with it, but each question may only be associated with one quiz (one to many relationship).

Front end: The part of a website that the user interacts with (GUI)

Back end: The part of the website where the user do not get to see (ie: server, database)

Database: structured data that is easily accessed, managed, updated, and typically stored electronically in a computer system

API (application programming interface): how applications interact and share data with one another

Selenium: an open source automation tool typically used for testing, botting.

Mapbox: A map tile provider similar to Google Maps

Leaflet.js: An open source JavaScript library used to build web mapping applications, compatible with Mapbox.

Server side rendering: Traditional form of rendering found on virtually all sites. Dynamic content is rendered by the server and served to the user as a static html file.

Client side rendering: Newer form of rendering found in many modern sites. Utilizes JavaScript running in the users browser to render content dynamically.

React Native: An open-source mobile application framework for converting web applications to native mobile applications.

VIII References / Bibliography

Original Creators of *QuizQuest*:

Group 2 -- Fall 2016:

Tsz Lam, Tan Le, Anthony Nedumgottil, and Weiheng Ruan

IX Index

No index entries found