Software Requirements Specification for ClickEase

October 25, 2018

Revisions

1.0	October 25, 2018	Initial draft
1.1	October 26, 2018	Stated which sections are not applicable
1.2	October 26, 2018	Rephrasing and editing

TABLE OF CONTENTS

1. Introduction - 2.5 pts

1.1 Purpose.

This document describes the software requirements for a university based clicker Web App. It is designed for the developer, maintainer, and designer of the Web App.

1.2 Scope.

The function of the Web App is to allow professors to pose questions to students immediately collect and view the responses of the entire class. The system's users would be university students and professors.

1.3 Overview.

The remainder of this document is organized as follows: There will be some definitions and terms in the next subsection. Section 2 contains any documents referenced in this SRS. Sections 3 identifies the specific functional requirements, the external interfaces, and performance requirements of the web-app. Section 4 contains qualifications of requirements. Section 5 contains traceability of requirements. Section 6 contains additional notes.

1.4 Definitions.

- MERN Stack - MongoDB Express.JS React.JS Node.JS
- Clicker - A multiple choice response tool tied to a student's ID

1.5 Referenced Documents

- https://keyholesoftware.com/company/creations/white-papers/why-node-js/
- https://www.mongodb.com/scale/when-to-use-nosql-database

2. System Overview

This section will provide a high level System Overview for the Application. The Web App's purpose is to phase out the university's current use of physical clickers within large lectures. The Web App will incorporate a student view as well as an instructor view. The instructor side of the Web App will allow the professors to set up classes. These classes can then allow them to invite students given a registration code, so they can properly connect to that class. The instructors side of the Web App also allows the instructor to set up and ask questions. Additionally, they can also see a student's progress throughout the semester.

The student side of the Web App, is not as robust. They sign in and are greeted with a selection page. The selection page allows them to select a class they are already registered for, or add a new class (using the registration code the professor would give).

Table 1 - Application Components

Component	**Description**
Web Client	The Web Client is the frontend of the Web App. It will allow a user (Professor/Student) to interact with the Web App. This includes creating new accounts, answering questions, seeing progress.
Backend Server/ Statistical Sources	The Backend Server of the Web App will provide the database information on students and professors. The backend server will also aggregate student information.

The Web App will have three primary actors:

Table 2 - Web App Actors

Actor	Description
User (Student/Professor)	The user can be either a student or professor. They will interact with the Web Client to perform specific actions that are separate for the two actors, as stated above.

Browser	Will host the Web App and provide input and output into the backend server.
Statistic Services	Backend API will keep track of how students are doing in their registered classes and present the information

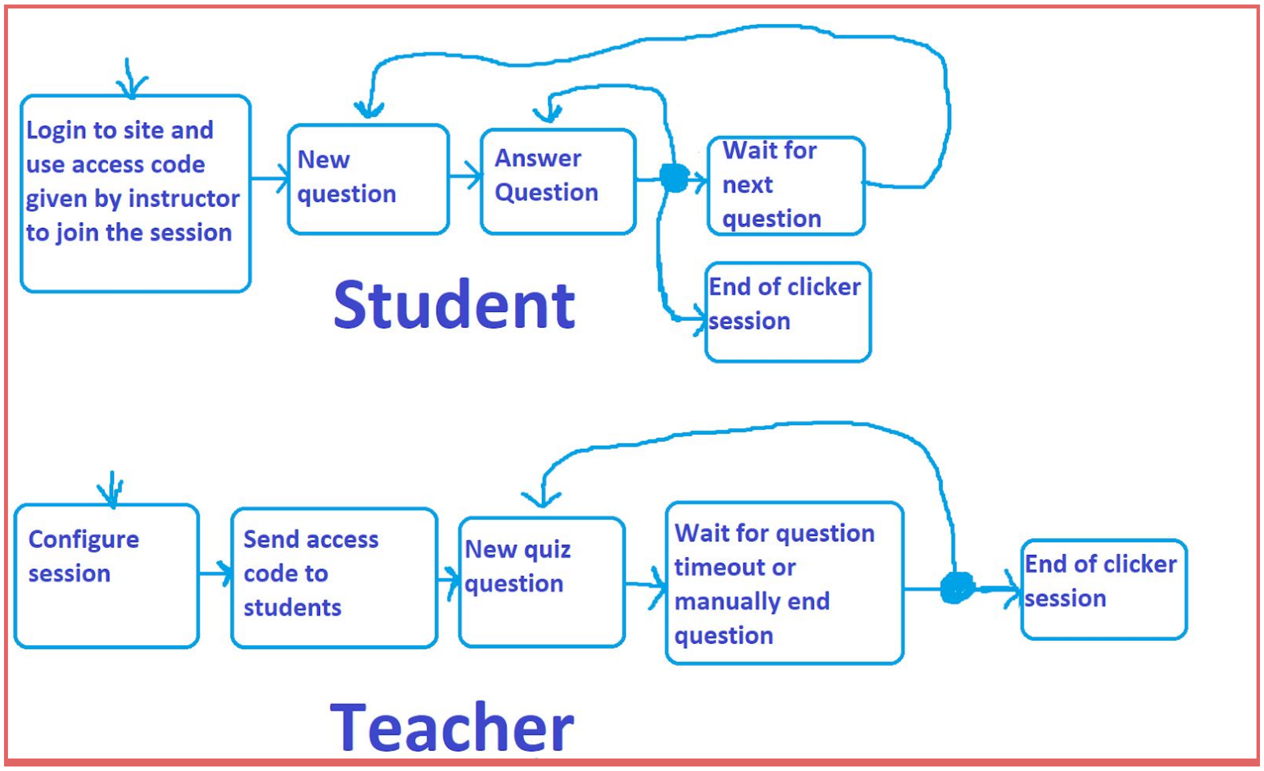
3. Requirements

3.1 Required states and modes.

During the startup of the Web App, users are greeted with a login screen. From here, they can choose to login to student mode or professor mode. The professor mode allows users to create a new course, send registration codes to students, create questions, and view student responses and statistics. The student mode allows students to add classes to their course list and answer questions during a quiz.

Students can be on two states during a quiz. If there is an active question that the student has not yet answered then they are in the answering state. If they have already answered the active question and are waiting for the next question then they are in the waiting state. There is some gray area as to which state the student is in if the session is configured to allow multiple submissions, and they have submitted an answer but there are still attempts remaining. For simplicity we will say that the student is in the waiting state only if they have submitted their final response for that question, and are in the answering state otherwise

Happy Paths


Login to site and use access code given by instructor to join the session
New question
Answer Question
Wait for next question
End of clicker session
Student
Configure session
Send access code to students
New quiz question
Wait for question timeout or manually end question
End of clicker session
Teacher

3.2 CSCI capability requirements.

No capability requirements have been identified. This section will be updated in future reisions.

3.3 CSCI external interface requirements.

There are no external interface requirements requirements.

3.4 CSCI internal interface requirements.

The application will consist of a client-server structure utilizing a MERN stack (MongoDB, Express.js, React.js, Node.js). The server's role is to track users' response statistics for an active quiz instance. The client's role is to send the user's selection for a particular question to the server. Further details will be provided in the design documentation.

3.5 CSCI internal data requirements.

Data shall be abstracted in the design document in such a way that there is passable data and compromising data. Passable data is data that can be passed to the user without compromising the quiz integrity (i.e. a question can be passable data but a quiz answer cannot be passable as it would provide a way for someone to cheat). Compromising data is data that could potentially compromise the integrity of the quiz (such as the answer to a question). This data should only be accessed by server side processes.

Entity	**Description**
User	The user object stores information pertaining to individual users. The subtypes are student and professor.
Student	The student object is a type of user user. Students may join sessions and submit answers.
professor	The professor object is a type of user. professors may create sessions and view qui results.
Course	The course object stores a list of users enrolled and history of past sessions.
Quiz	The quiz object is a container for a series of questions, the corresponding multiple choice answers, and the correct answer
Session	The session object contains a quiz and all records for users who have been invited to participate in that quiz.
Record	Stores an individual users quiz results for a particular quiz template.

3.6 Adaptation requirements.

There are no additional adaptation requirements.

3.7 Safety requirements.

There are no additional safety requirements.

3.8 Security and privacy requirements.

The application shall have login system to associate users with their profiles on the server. It is important that student profiles and professor profiles are kept separate, as students should not have the admin access that professors should. This separation will ensure that only professors can create quizzes and view the student reports. Certain consideration will need to be given as to what data is passed to users. We must make sure than information such as the correct answer or future questions are not sent to the students' clients.

Additionally, the database must be secured and encrypted to prevent unauthorized access to private information.

3.9 CSCI environment requirements.

The application shall not require any special configuration on the client side, only a modern web browser with internet access. The server will only require an average modern web server setup with no necessary special conditions for a small number of active users. For larger user bases it will be beneficial to scale the web server for the larger workload.

3.10 Computer resource requirements.

3.10.1 Computer hardware requirements.

The clicker webapp shall require a server that has at least one Network Interface Card with internet access in order to operate. One additional, separate, server is required in order to store database backups. Neither the main nor backup servers shall require any particular specialized hardware setup.

3.10.2 Computer hardware resource utilization requirements.

The clicker webapp shall not be more resource intensive than an average modern web server. The clicker shall not require any particular amount of resource allocation beyond the normal operating conditions of the server and network it is located on.

3.10.3 Computer software requirements.

The clicker webapp shall require a non-particular web server that is JavaScript capable and shall be accessible by any modern web browser. JavaScript and multiple JavaScript frameworks are required to be installed on the server in order for the webapp to operate. These include: Node.JS, React.JS, & Express.JS. A MongoDB database is required to store data including user authentication data, student responses, questionnaire templates, and other misc data that may need to be stored long term. The webapp server shall not require a particular operating system and shall be able to run on any system where its framework dependencies can.

3.10.4 Computer communications requirements.

No messaging software such as ActiveMQ or Amazon SNS/SQS shall be required as communications shall be handled by the Node.JS REST API. Each class using the clicker shall be given a daily UUID for the activities of that day. Students must be enrolled in a course, aware of the UUID, and be able to answer questions within a professor-specified timeframe in order for their responses to be considered. Therefore the webapp shall not require any use of GPS or other forms of geolocation for verification of its users. The webapp shall be accessible to anyone with an internet connection and a modern web browser and shall not require any tailored communications software or network configurations in order to be used by a client.

3.11 Testability and Usability.

The testing and Usability of the ClickEase will be done in three phases. The first phases will consist of testing the ability of the Professor/Instructor side of the Web App. This initial phase will see use creating sample/test professor users to test the user registration aspect. This is will allow us to test the Database portion for that user group. After achieving a proper registration page, phase 1 will shift focus on the professor-user being able to create classes and set a proper registration code to allow students to register for that specific class. This type of user should be able to create as many classes as they like. However, limitations on this specific feature might be set in-order to combat server side issues. Phase 1 shifts into the final phase of testing of creating questions and setting time limits of those questions. This type of testing will be rather forced testing. Questions will be manually entered in the DB and time settings will be handled by the front end (This feature and implementation is subject to change).

The second phase for testing will proceed into the second user group. The student side of ClickEase will be tested on whether a student is successfully able to join a class. If there information has a layer of separation as to not have their grades be viewable to the user. The hardest of the tests on the student side will be testing the server load for incoming clicker* responses. As the server is pinged with the continuous input of answers for a specific question, the server/API should only keep track of the last possible submitted answer within the allocated time. The next phase of testing will test the statistics ability of the site to relay information to the student about current form/grade in the class based on the clicker questions.

The final phase of testing is the incorporate both phase 1 and phase 2 together. This would allow users to creates accounts, post questions, and answer questions. This phase shall be the easier of the three since most of the in-depth testing of the features will be dealt with in the first two phases.

After all features laid out above are test and work accordingly, the focus will shift from testing to usability. This will consist of UI designs being changed until the Web App uses a intuitive design and implementation. While it is important that features work, it is more important that users find the features easy to use and easy to find. This will require surveys to see what students would like to see design wise. What color schemes and layout options they find to be the most engaging and will not impede on their concentration within a specific lecture.

3.12 Design and implementation constraints.

The webapp will require a MongoDB database and must follow the standard of NoSQL database flow. This in conjunction with main communications being done via REST endpoint leads to the standard of all Data Transfer Objects being serialized as JSON for network transmission and database storage. The web server shall follow the strict separation of backend and frontend by following the patterns of Node.JS and React.JS and keeping the two separate, only communicating via API calls.

As stated previously the webserver does not have any particular base hardware constraints but, as with any server, if traffic to the server increases beyond the normal case of the network and infrastructure it is deployed on then the infrastructure may need to be scaled as well. This fully dependent on the organization hosting the server software and the server load can be remedied easily by increasing hardware or by deploying multiple server instances with a network load balancer.

4. Qualification provisions

Software will be qualified through formal validation tests of the SRS level requirements. The Qualification Methods applied to the software shall include test, analysis, and demonstration.

4.1 Demonstration

Demonstration will be done by setting up a test session and showing the GUI for professors and students. It will also show the output of student statistics. This should show that the application is easy to navigate.

4.2 Test

Tests will be performed to ensure that the software behaves as expected under specific circumstances. Testing will include checking that the client's submission is received by the server and correctly recorded, that students can not access the professor role, registration process, and database security. We will use automated unit and integration tests as well as manual testing.

4.3 Analysis

Analysis will be done on the students' scores to make sure that they are stored correctly, and that the data is easily accessible. It will also be important to analyze the variation in time it takes for packets to go from client to server, then for the server to process the data. This analysis will assist in calibration and in understanding how performance requirements will scale to increased workloads.

5. Requirements traceability - 5 pts

TBD

6. Notes.

A. Appendixes.