Software Detailed Design Report

Team MGN

October 31, 2017

Breakdown of Individual Contributions

Joseph Olin

- Responsible for communication with American Printing House for the Blind
- Responsible for report proofing and submissions
- Will work on the server side of the app
- Will work on the client side of the app
- Will work on integration testing the app

Ben Pister

- Responsible for maintenance of the project website
- Will work on unit testing the server side of the app

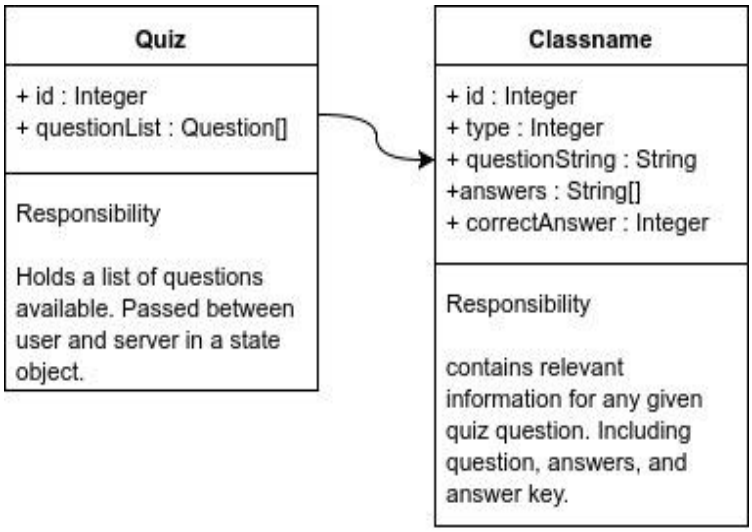
Michael Roark

- Responsible for compiling the reports prior to proofing
- Responsible for general documentation of the project
- Will work on the server side of the app
- Will work on the client side of the app

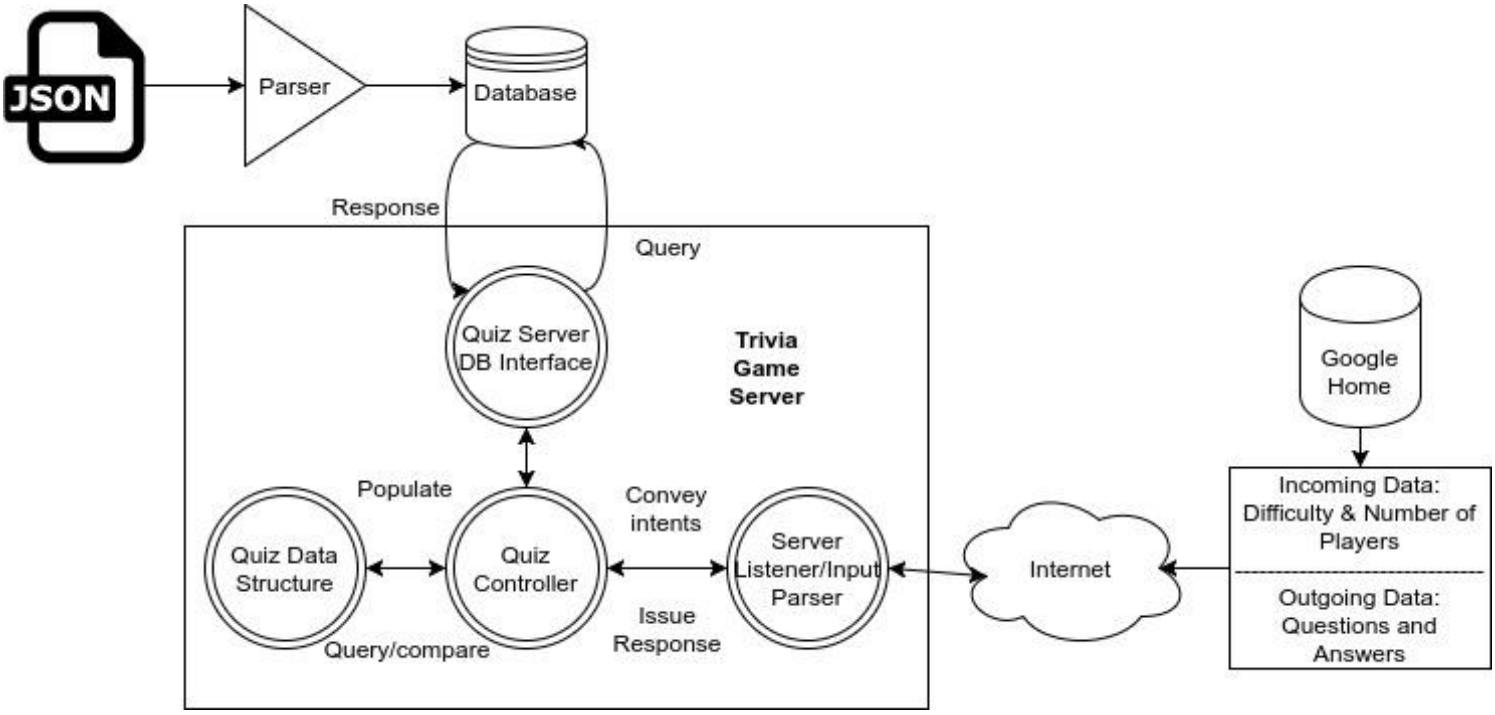
Data Structures

The two central data object we'll have are the Quizzes and Questions. The Questions will be held within the quiz object in an array. These objects will be populated with data from a JSON file.

Time permitting, the data will be placed in a remote database rather than a JSON file.



Architecture



Interface Design

Since this app is designed with blind and visually impaired people in mind, no user interface is required for this system. The entire system will function using audio input and output. Any UI that Google Assistant might display to the user would simply display the text representation of the audio output given to the user.

The code interface provided to the client side of the application is an HTTPS URL. This URL provides the entry point to the server application that continuously runs. The call to this URL is caught by the exported handler connected to the URL. In our application, the handler signature will look something like
exports.triviaHandler = functions.https.onRequest((req, res) => {}

This handler will operate as the command interpreter for the server app, and is denoted in the Architecture section as “Server Listener/Input Parser”.

The code interface provided to the server side of the application by the data access module may take multiple forms. However, as of right now it is likely this interface will be a web service request to a Node data server, which will serve a set of trivia data questions for each unique game session. Time permitting, this interface will be changed to queries on a remote database, which will return a set of trivia game questions. This interface is denoted in the Architecture section as “Quiz Server DB Interface”.

Procedural Design

loadQuiz
Retrieves a set of trivia quiz Question objects for the segssion via the data access interface.

triviaHandler
The exported HTTPS function that receives the request from the client side. This function contains an anonymous function which acts as the command interpreter of the server app by dispatching the user to the appropriate function based on the state they're in.

welcomeMessage
Function that builds and returns the audio message welcoming the user to the trivia game, and asking them for either difficulty or number of players, depending on if difficulty was mentioned on startup.

getNumberOfPlayers
Returns the message String which asks the user for the number of players.

getDifficulty
Returns the message String which asks the user for the difficulty.

buildQuestionString
Returns a trivia question String using a Question object.

retrieveQuestion
Gets a question object from the set of trivia questions that is maintained within the conversation session.

getAnswerMessageString
Returns a message String congratulating or consoling the user depending on if the provided answer was correct compared to the provided Question object.

isGameOver
Determines if the game is over, based on if the number of players in the game times the max number of questions per player equals the total number of trivia questions asked.

Products, Platforms, and Languages Used

The client of the system will run on Google Home and Google Home Mini devices. It will be written in JSON, following Google's client-side specification for Google Actions client-side syntax.

The server app will be written in Javascript. It is composed of a Node.js module, which will rely heavily on Google's Actions SDK framework for Google Assistant apps for maintaining the conversation with the user.

The data access module will likely be a NoSQL database stored on a remote server. Depending on if APH wants the database on their own server or not, this database may be kept on one of APH's servers or it may be setup remotely via Firebase Realtime Database.