**Functional Testing Plan**

To: Professor Pisano

From: Jacob Dansey, Changshuo Fu, Christine Low, Urvashi Mohnani, Neil Sanghrajka

Team: 24

Date: 3/27/2017

Subject: Functional Test Plan



1. **Project Overview**

The Trivia Throw Toy will be a ball-shaped toy that speaks in a clear voice a trivia fact that has been intelligently pulled from Internet sources upon detecting it’s been thrown or shaken. The user will have the ability to hear facts by category and also up vote/down vote facts to enhance other user’s experience. A mobile application will be used as the interface for picking categories, managing Wi-Fi and voting. An accelerometer is used within the toy to detect motion. Upon detection, a fact is pulled from the proper categories cache and spoken aloud through a speaker. If the cache is near empty, a call to a cloud server is made to pull more facts in. The server handles the pulling in of “fun” facts from various Internet sources.

1. **Customer Requirements**

**Hardware requirements**

1. **Size and Shape:**
   1. The toy must withstand motion: Since the toy is triggered by throwing/shaking it, it must be designed in a manner that the components do not move from their original position.
   2. The toy must be comfortable to hold: The device is coated with stress ball material or a substitute.
   3. The toy must be designed for single hand use: The target size range of the toy is between the size of a baseball and football.
   4. The toy should have the capability to translate to different encasings, such as a stuffed animal, besides a ball, to appeal to different demographics.
2. **Speech**:
   1. The speech must be audible : Controls on the device will allow the volume to be changed from 30-80 decibels. The default will be 60 dB(volume of an average human conversation).
   2. The speech must be clear: An average speed of 140 words per minute must be targeted by the text to speech module on the device.
3. **Battery**
   1. Rechargeable: The batteries on the toy must support charging from any standard 120V and 60Hz AC electricity power outlet
   2. Battery life: The device must support at least 3 hours of continuous usage and indicate to the user when the battery life is under 20%. The toy must automatically shut down when battery <5%.
   3. The toy will be turned on with a switch to preserve battery life when it is not being used. Once the device is charged to >5% battery, it will be able to switch on to be fully functional.
4. **Motion activation**: The toy must be able to detect motion as it is the only way of triggering the toy to say a fact. It must also be able to differentiate between a throw/shake and simply being moved from one position to another.
5. **Connectivity**

The toy must be able to connect to WiFi as an Internet of Things (IoT) device. The wifi module must be able to emit a signal of its own so that the smartphone can connect to the wifi module initially to set up a connection between the toy and the app. Emitting a signal of its own will take away the need of a bluetooth module, reducing the number of modules and the overall cost of the product.

**Software requirements**

1. **Fact retrieval:** 
   1. Instantaneous: The time between the device being triggered and the speech module reading a fact must be less than ½ second.
   2. Minimal reliance on WiFi speed: While internet connectivity is essential to retrieve facts from the server, by caching facts locally on the toy at any given time fact retrieval can be made almost independent of the WiFi speed.
2. **Ability to work offline:** Even without a connection to the server the toy must be useful for at least a small duration of time. At any given time at least 50 facts must be stored locally on the device.
3. **Algorithmic requirements**
4. Trivia Parser: An algorithm must be created that is able to extract Trivia from Web pages.
5. REST API: Server code to handle HTTP requests to retriever trivia, create users, vote and get a user’s fact history.
6. **Mobile application:** An android/iOS application must help the user connect the toy to a particular WiFi network. Users can view their fun fact history and up/down vote facts for crowdsourced optimization.
7. **Non repetition of facts:** At no point should a user hear the same trivia again. If all the facts in the database are exhausted the user must be intimated.
8. **Database size:** The initial dataset must contain at least 1000 unique facts. Every time, the algorithm is run it must add new facts to the database, not exceeding 50,000.
9. **Status messages:** The toy must indicate to the user when certain situations occur. This may done using the text to speech module or LEDs. These situations are:
   1. No WiFi connectivity and facts from cache have been exhausted
   2. Battery critically low (<5%)
   3. Unable to pair with smartphone after searching for >2 minutes.
10. **Equipment and Setup**
    1. **“Toy”**

The schematic of the toy is shown in Figures 3.1.1 and 3.1.2 below.

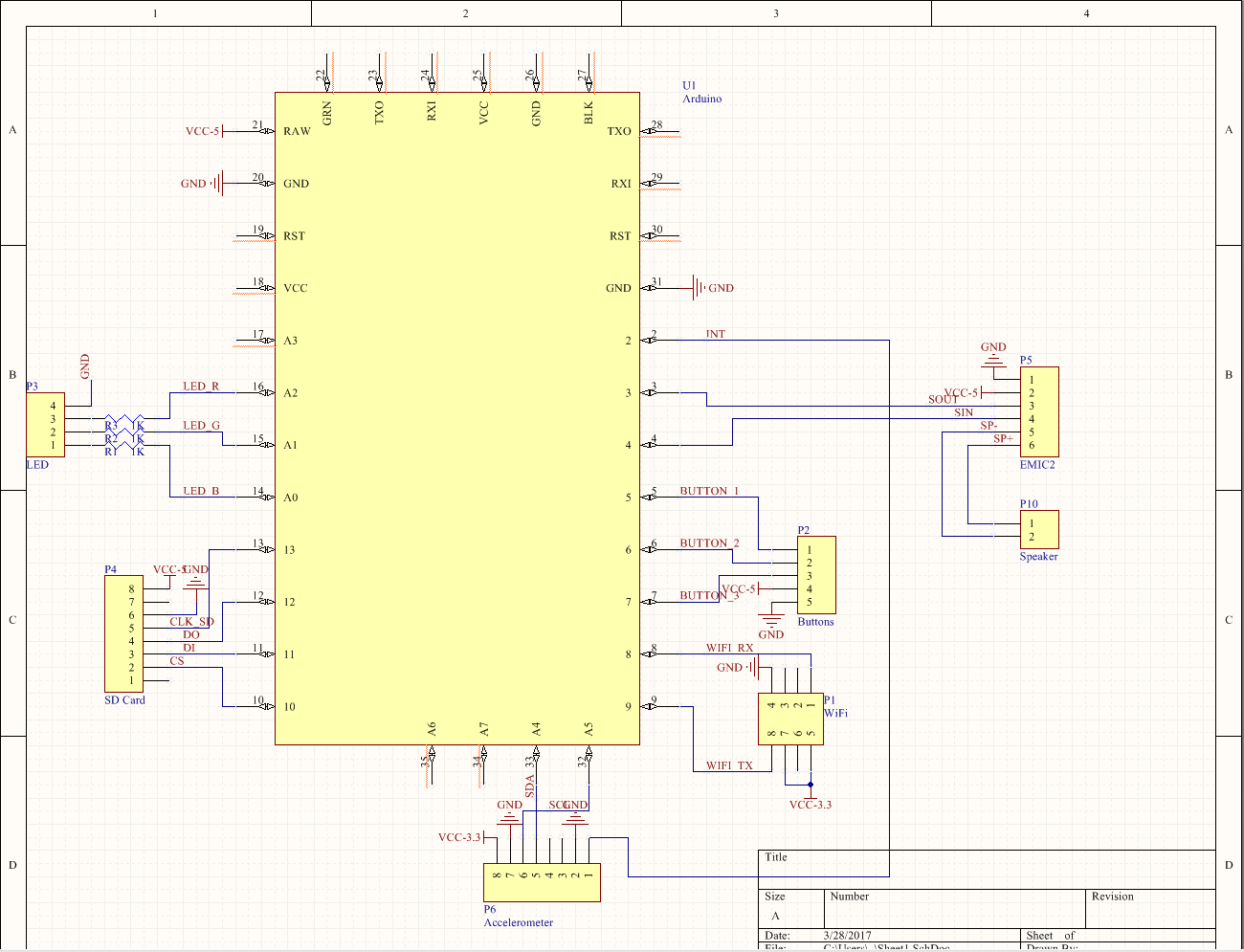


Figure 3.1: Part 1 of the overall schematic

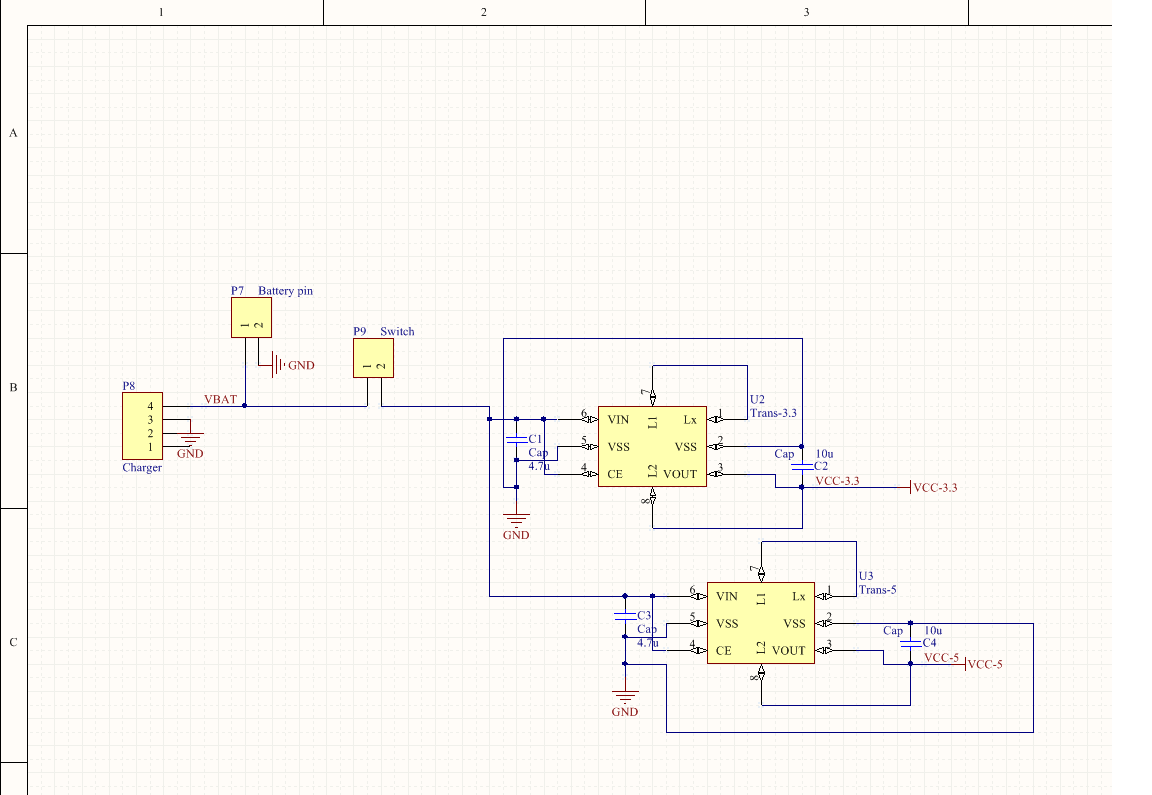


Figure 3.2: Part 2 of the overall schematic

* + 1. **WiFi**

The ESP8266 WiFi module is used with the Arduino for our purposes. The WiFi communicates with the Arduino over UART and requires 3.3V input power. The set up for the WiFi module is shown in Figure 3.1.1 below.

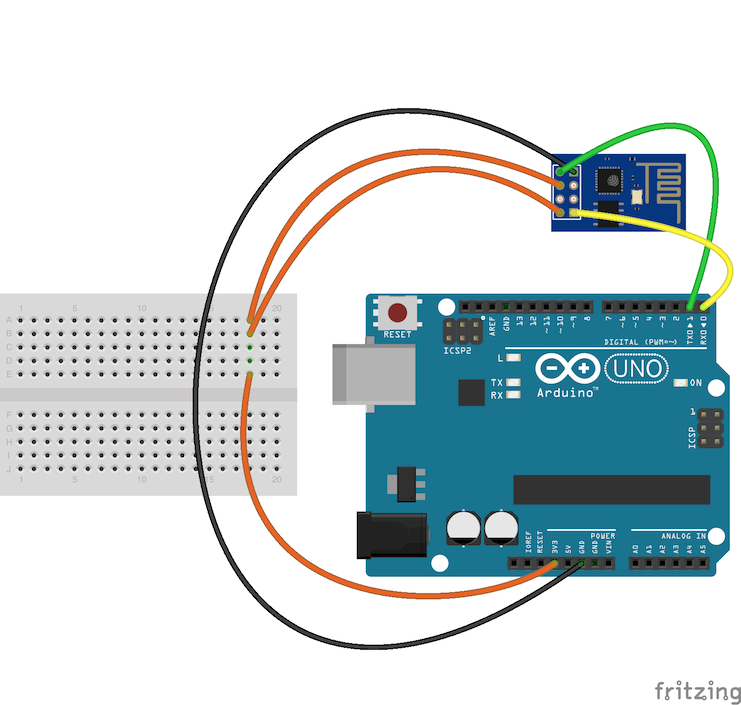


Figure 3.1.1: WiFi module connection with the Arduino

* + 1. **TTS and Speaker**

The TTS module used is called the EMIC2. It is the best string parser compatible with the arduino that offers many features as volume set, voice change etc. The emic2 connects to a speaker for the facts to be vocalized and the speaker used is a 3W and 8 ohms speaker, which gives us a better vocal clarity. The EMIC2 connects to the speaker from SP+ and SP-. The schematic of this connection is shown in Figure 3.1.2 below.

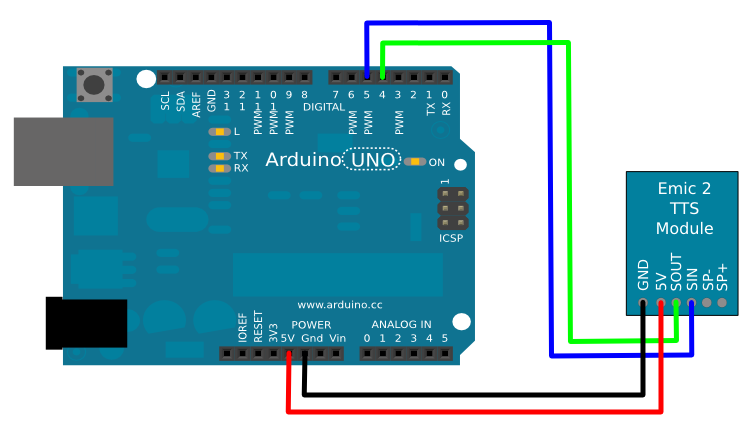


Figure 3.1.2: TTS module and Arduino connection

* + 1. **Accelerometer**

The Accelerometer used is the MPU-6050 with 3-axis detection feature. This feature is essential since the ball can be thrown in any direction by the user to trigger a fact. The accelerometer is set such that only a hard shake or throw triggers the fact vocalization to prevent it from starting up while walking with it.

The schematic for the accelerometer with the Arduino is shown in Figure 3.1.3 below.

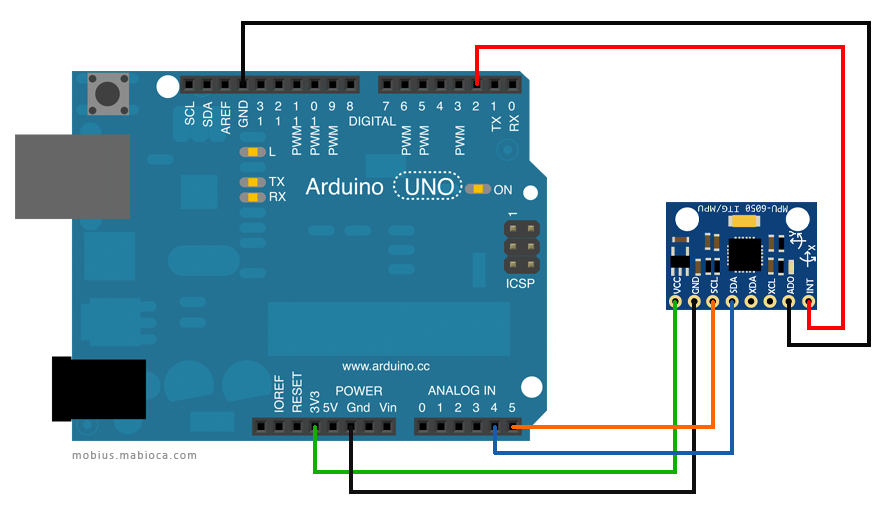
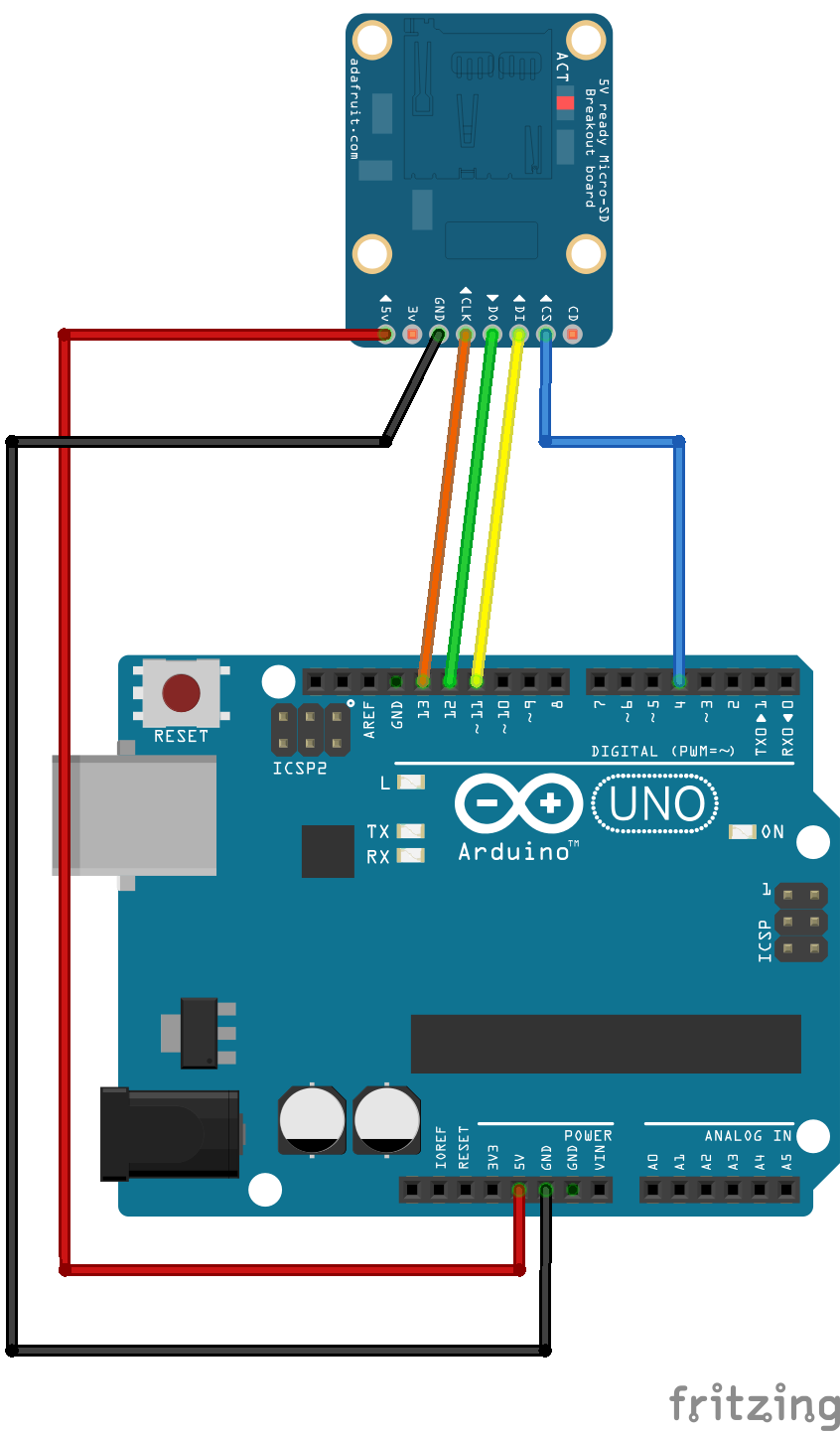


Figure 3.1.3: Schematic of accelerometer and Arduino

* + 1. **SD card**

The SD Card module is used to read and write the facts to the local storage on the toy. This is done so that the fact is vocalized within 0.5seconds of detecting a throw or shake. The toy connects to the server and downloads facts to ensure that it always has 50 facts in local storage. The external storage was necessary since the arduino does not have much EEPROM memory, only about 2kB. The schematic for this part is shown in Figure 3.1.4 below.

Figure 3.1.4: Schematic of SD card and Arduino

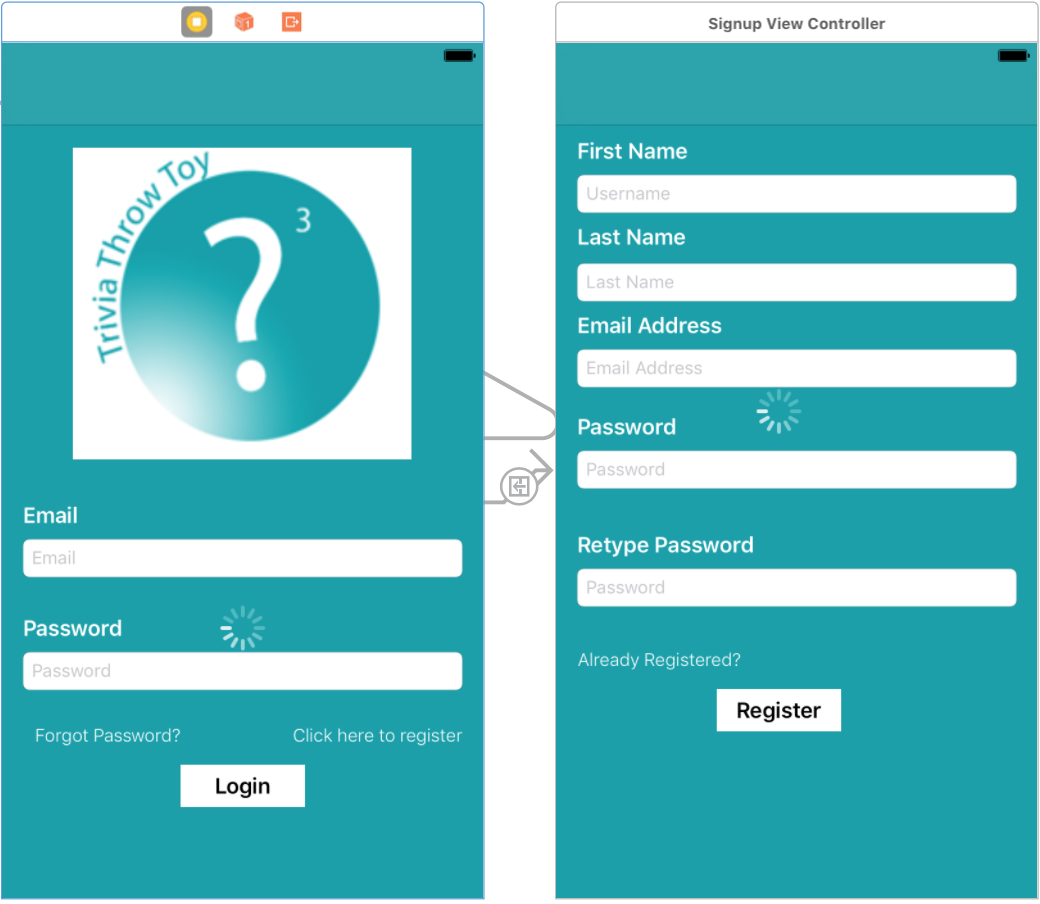
* + 1. **LEDs and Buttons**

There are three buttons in the circuit. One button is volume up, the other is volume down, and the third is to reset the WiFi. There is a switch that creates the connection between the battery and circuit to power on and off the circuit. There is one RGB LED that changes colour based on the status of the toy. For example, red is for low battery, blue is updating from server etc.

* 1. **Smartphone Application**
     1. **Login/Registration**

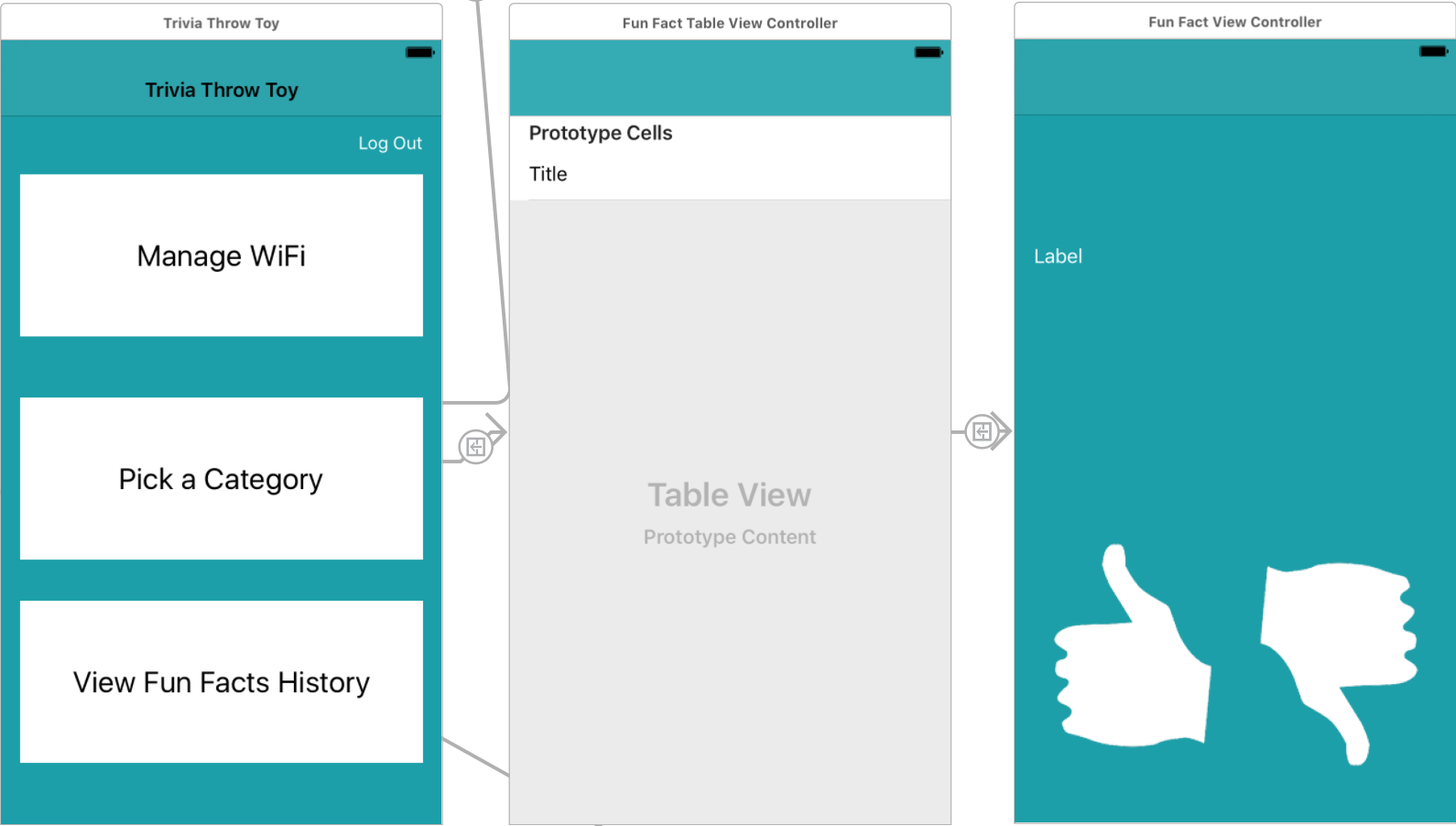
When a user registers with the application, they input their first name, last name, email and password as unique credentials in the *SignupViewController*. This information is then sent as an asynchronous POST request to the endpoint /Users/Register with a unique userID returned in a successful response JSON. The unique userID is stored on the device in the user defaults location with the value “*userID*”, and a session is initiated. When a unique userID is detected on the login screen, the user is automatically logged in to the application.

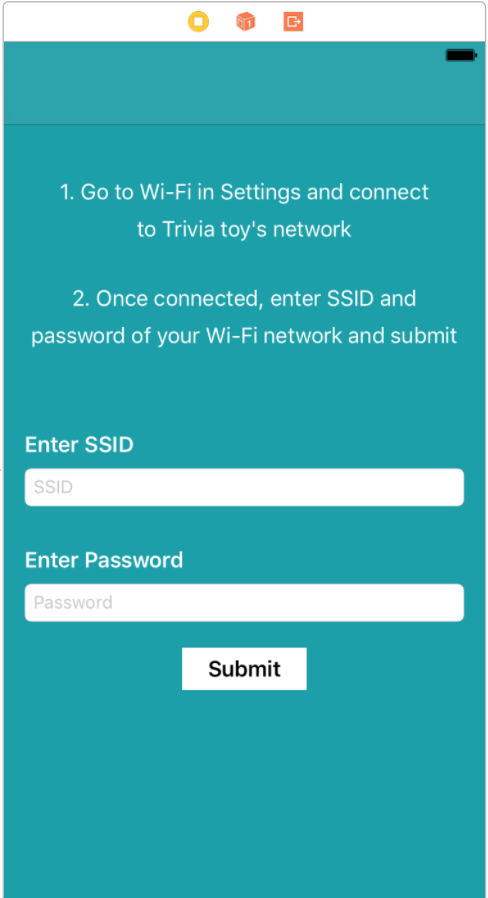
If their is no current session going, the user must login to the application with their email and password. The email and password are sent as as asynchronous POST request to the endpoint /Users/Login with their uniqueID returned in a successful response JSON in the *LoginViewController*. This ID is then stored back into the user defaults “*userID*” value to begin a new session and the user is entered into the application. Within the application the unique ID is used in HTTP requests to identify the proper fact history and category selection. When the logout button is pressed, the user defaults for “*userID*” are cleared to end the session and the user is brought to the main login screen.



* + 1. **Fun Fact History and Voting**

The fact history of a user is displayed in a tableview in the *FunFactTableViewController.* Upon entering the table view controller, the userID of the user is pulled from the default preferences on the device. An asynchronous GET request is then sent to the endpoint User/{userID}/history and a JSON response is returned with the user’s fact history. Once the JSON is returned, the tableview has to reload the displayed data due to the nature of asynchronous calls. The JSON is then parsed and displayed in the proper cell of the table view in reverse chronological order. This is accomplished through the use of the function *tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell*. The index path of a cell is related to its location in the tableview and the return value is the cell of that specific index with the information it is displaying. Once a fact cell is selected from the tableview,the current view is segued to the *FunFactViewController* that displays the fact information as well as the ability to up vote/down vote said fact. Prior to this segue, the *FunFactTableViewController* must pass along the fun fact string and unique ID to the next view.





* + 1. **WiFi Management**

Managing the Wi-Fi connection is handled by the *WiFiViewController*. Upon entering the view, the user is prompted to go to the settings of the phone and connect to the wi-fi network being emitted by the toy. Doing this allows the phone application and the toy to communicate through a socket connection. It checks the connection by attempting to start the socket connection and then receives the toy’s socket emitting message that confirms it is the Trivio toy. After connecting to the proper network, the submit button is enabled and the user is prompted enter the network credentials of their Wi-Fi network. If the toy successfully connects, it will emit a confirmation message that it is connected and the app will display that it was successful.

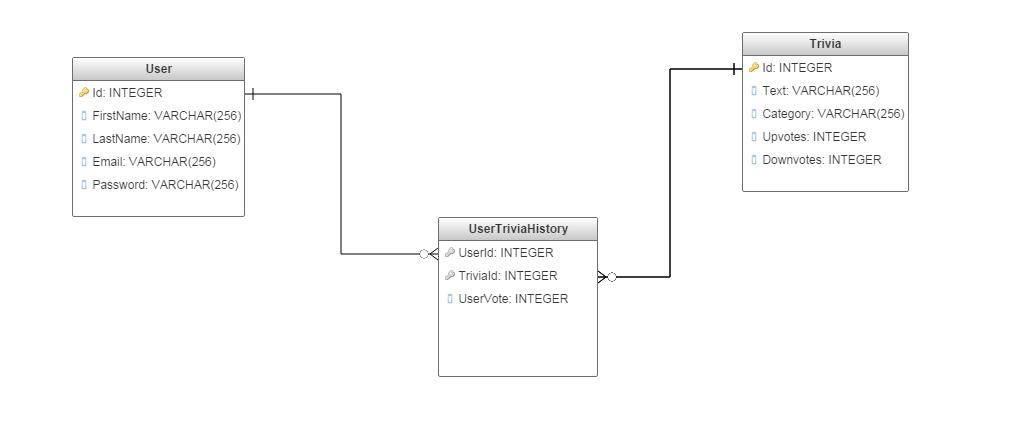
* 1. **Server**
     1. **Database Setup**

The SQL database is hosted on Microsoft Azure and contains 3 tables:

* **User**: Information about all user accounts
* **Trivia**: The collection of trivias, and the number of upvotes/downvotes on that trivia.
* **UserTriviaHistory**: This is a many-many relation table, created by joining the Id from Trivia and User table. It allows us to track the trivia history for a particular user.

To connect to the database using ADO.NET the following connection string is used: *Server=tcp:triviotoyserver.database.windows.net,1433;Initial Catalog=TrivioDb;Persist Security Info=False;User ID={your\_username};Password={your\_password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;*

The diagram below shows the Database schema.



* + 1. **Server Setup**

The server is implemented using ASP.NET framework. It performs multiple functions to access the database found primarily in TriviaController.cs and UserController.cs. The server exposes two broad categories of Web API that follow the REST convention. These are:

* User: These endpoints deal with tasks leading to management of user accounts. This API is primarily used by the smartphone application.
* Trivia: These endpoints deal with modifying and accessing the trivia stored in the database.

Detailed API documentation: <https://app.swaggerhub.com/api/neils95/trivio-api/1.0.0>

This table below provides a summary of API endpoints and expected outcome:

|  |  |  |  |
| --- | --- | --- | --- |
| **Endpoint** | **Description** | **Parameters** | **Return values** |
| GET:  Trivia/ | Returns JSON collection of all facts stored in database. | None | Status code : 200   * JSON list of Trivia   Status Code: 400 |
| GET:  Trivia/{UserId} | Returns a single Trivia string for user account linked to {UserId} | {UserId:int} in Path | Status code : 200   * Trivia string   Status Code: 400   * Error string |
| GET:  Users/ | Returns JSON collection of all Users stored in database | None | Status code : 200   * JSON list of users   Status Code: 400 |
| POST:  Users/Register | Creates new user account with unique userid. | JSON in request body {password,username,name} | Status code : 200  - Created user object(JSON)  Status Code: 400   * Preexisting account |
| POST: Users/Login | Log in existing user. | JSON in request body {password,username} | Status code : 200   * User object(JSON)   Status Code: 400   * Invalid Login detail |
| GET: User/History/{UserId} | Return array of TriviaId of a particular user’s Trivia history | {UserId:int} in Path | Status code:200   * Array of TriviaId (int)   Status code:400   * Invalid UserId |

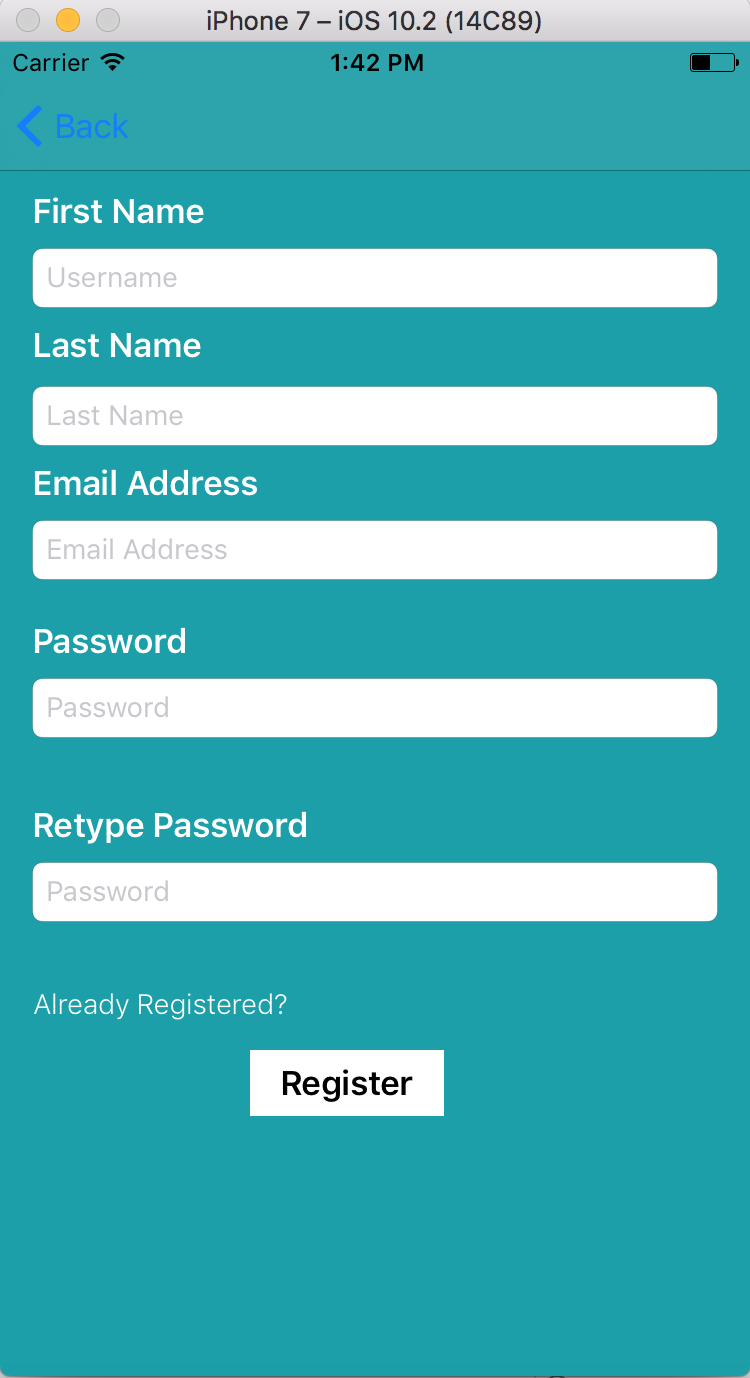
1. **Deliverable Overview**

Our deliverable is a functional display of the different components of the toy and how they work together. Our smartphone application for both Android and iOS demonstrate user’s ability to signup, login, view fun fact history and send wifi credentials to the toy. The toy is able to clearly detect an acceleration and speak out fun facts from the memory on the toy. When needed the toy is also able to call the server to pull more fun facts into the cache. The server has over 5,000 fun facts stored in its database. Through an API the server is able to handle every request for data and keep both the phone and toy in sync with each other. LED’s and button are used as IO devices for the user to interact with the toy and understand issues.

1. **Test cases** 
   1. **Create new user using Smartphone app**
      1. **Significance**

Creating new users is necessary for keeping track of the fact history as well as voting on facts for crowd sourced optimization. User IDs are required for most of the API calls and ensure that the the user never receives the same fact twice.

* + 1. **Procedure**

To sign up a new user, a user will go to the signup page from the main login page of the application. On the signup page, the users enters their name, email address and password. The password is verified by retyping the password and the email is verified to be an actual email. If the email address is already associated with an account an error message will be shown on the application. After the user inputs their information and presses sign up, a POST request is made to the API end point /User/Register with their name, address and password in the body. This stores the user information in the server. The server responds with a JSON that includes the unique user ID

* + 1. **Successful result:**

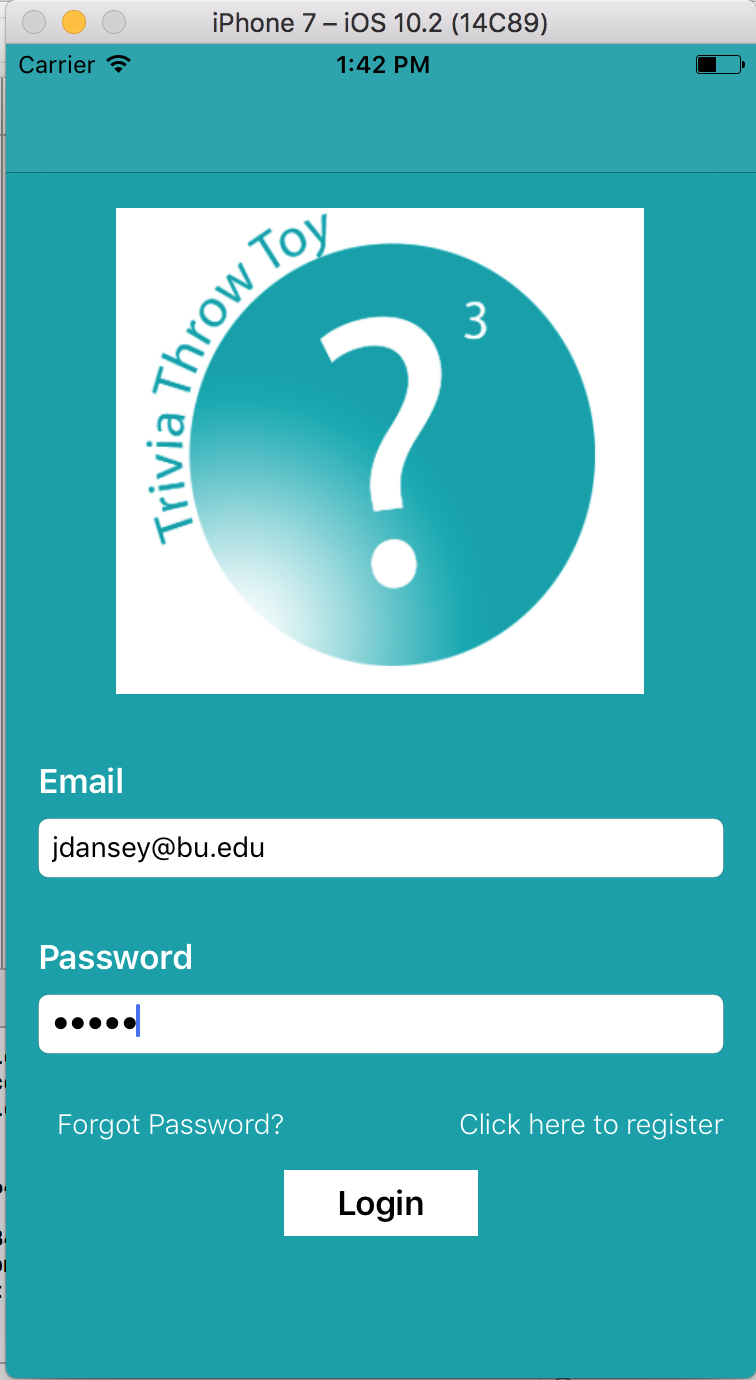
Users are able to successfully sign up for the application and get a unique user ID from the server. This ID is used for future requests to the server in order to keep information synced between the toy and the application

Screen Shot 2017-03-28 at 1.48.02 PM.png

* 1. **Login existing user using Smartphone app**
     1. **Significance**

Being able to login as an existing user allows for a full mobile and toy user experience. Logging in to the application allows the user to view all of the fun facts they have heard, vote on the facts, and send their userID to the phone via the wifi management.

* + 1. **Procedure**

When a user first opens the application they are taken to the login screen. On the login screen they are prompted to enter their email address and password. After inputting the information and pressing login, the application validates the entries and sends a POST request to User/Login with the email and password in the body of the request. If the request is invalid an appropriate error message is returned and shown on the application. If the request is valid, the user’s unique ID is returned and saved on the phone’s device to begin a session. This way when the user exits the application, the app can check the phone and automatically log the user in without them needing to input their credentials again. A user can reset the credentials via the logout button.

* + 1. **Successful Result**

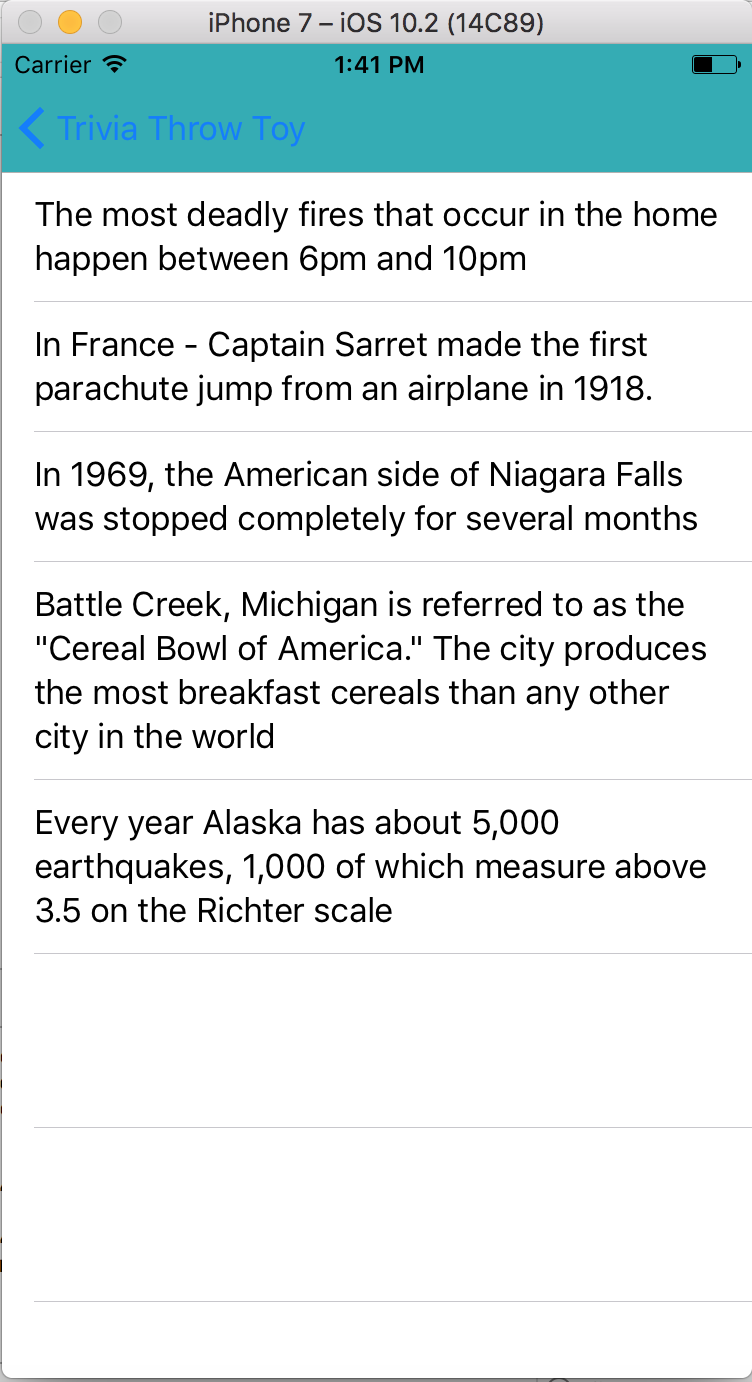
A user is successfully able to login in to the application with the right credentials. The unique userID is sent back from the server and stored on the device to allow for immediate logins. The user can also log out of the application to reset the session.

* 1. **Retrieve User Fact History**
     1. **Significance**

The fact history functionality of the application is an integral part of the toy experience. If a user was unable to hear a fact on the toy, they can simply check the application to identify what the fact was. The fact history is also the interface that allows users to vote on the facts. If they view their history and particularly like or dislike a fact they can vote accordingly.

* + 1. **Procedure**

After a user is logged in to the application, they have the option to view their fun fact history. The fun fact history displays a list of all the previously heard fun facts from the toy. When a user segues to this page, a GET request is made to the endpoint User/History/{id}, with id being the user’s unique id number. A successful response holds a JSON of all previously heard trivia facts with their unique fact IDs and other information such as the number of votes. This response is then parsed into a table for viewing in chronological order. Clicking on a specific fact segues the user to another view where they can vote on the fact.

* + 1. **Successful Result**

After a user successfully logs in, they can click on the fun fact history button and see a list of all facts that they have heard from the toy. They can then click on individual facts to be taken to a different screen for voting.

* 1. **Vote on Trivia**
     1. **Significance**

Allowing users to vote on trivia helps solve some of the pitfalls of random fact scraping from the Internet. Since there is no one moderating every single fact, crowdsourcing the overall opinion of each fact allows the database to optimize the experience over time.

* + 1. **Procedure**

After a user has selected a fact from the history table, the fact information needs to be passed along to the next view that uses the information. By knowing the cell that was clicked, the fun fact text and uniqueID get passed along to the next view controller which displays the fact. The unique fact ID is used for the up/down vote buttons. When clicking a button, a PUT request is made to the endpoint /Trivia/Vote/{id} where the id is the unique fact ID. The body of the PUT request includes the userVote (1 or -1 depending on up or down vote), user ID and fact ID. This information is used to keep track of who has voted on each fact to prevent duplicate votes.

* + 1. **Successful Result**

A user is able to successfully vote on each fact after selecting it from the history table. The fact records the user vote on the fact and prevents the user from voting multiple times. The user is alerted that they have voted and are the brought back to the fun fact history table.

* 1. **Transfer WiFi credentials between toy and smartphone app**
     1. **Significance**

Every IoT device needs a way of receiving WiFi credentials in order to access the Internet. However, the problem oftentimes is that the device often needs access to the Internet in order to receive the necessary data to access the Internet. Without the ability for the wifi to be passed along to the toy, the toy will be unable to function with internet capability and pull new facts in.

* + 1. **Procedure**

To connect the toy to the internet, the WiFi credentials have to be transferred from the mobile app to the toy. The overall idea behind this is to set up a TCP connection between the two and send data over on a port. The phone app has the TCP server setup and running on port 8080. The WiFi module acts as an Access Point, in the sense that it emits its own WiFi signal for the toy to connect to. Once the toy connects to the WiFi module, they are on the same local network and packets can be transferred back and forth between them.

The WiFi module can create a TCP connection with the mobile app, since the WiFi module knows the IP’s of the devices connected to it. Once this connection is established, the mobile app sends an “ok” to the WiFi module telling it that it has acknowledged it connection request. Once the WiFi module receives this acknowledgement, it sends back an “ok” to the phone app telling it that it has received its acknowledgement so a connection should be started. This is known as a 3 way handshake.

Once this 3 way handshake is successful and a connection is established, the phone app unlocks the “next” button for the user. Once this button is clicked, the app is redirected to a page that scans for all available WiFi networks and displays it to the user. The user can then select their desired WiFi network and type in the password. Once the user clicks the “connect” button, this data is sent over to the toy in the format “userid:wifi\_ssid:wifi\_pass.”

The user ID is sent over so that the toy knows which user database to access for information like fact history, upvote, downvote. Once the toy receives this string, it breaks it apart so that the WiFi module can use the “ssid” and “pass” to connect to the internet. If the connection to the internet is successful, the toy sends a “yes” to the phone app telling it that WiFi connected successfully. The user is notified on the phone app and the TCP connection is closed. In the case that the connection is unsuccessful due to incorrect credentials, or router problems, the toy sends a “no” to the the phone app to notify the user and the TCP connection is closed. If the user wants to redo this process, he/she has to press the WiFi Setup button on the toy, and a new TCP connection will be started.

* + 1. **Successful Result**

The phone app successfully starts a TCP server and the WiFi module successfully creates a TCP connection with it. The acknowledgment messages are transferred between the two successfully and the credentials entered by the user is successfully received by the toy. The toy is then able to break the string received and try connecting the toy to the internet. On a success, a “yes” message is successfully sent and on a failure, a “no” message is sent. These messages are received by the phone app and used to notify the user the status of the WiFi connection.

* 1. **Connect to WiFi/Pull in more data**
     1. **Significance**

The toy must be able to connect and reset the WiFi in order to make the proper connection to the server. This connection is needed in order to pull new facts in as well as tell the server which facts have been listened to. Without the ability to connect to the Internet, the toy will eventually run out of the facts that are stored on the toy. Our project must be able to pull in a nearly endless supply of new facts for continuous enjoyment.

* + 1. **Procedure**

The toy is connected to the ESP8266 WiFi module which has the ability to act as server or client over the Internet. A client is established using the Arduino’s built-in WiFiESP library. To connect to the WiFI, the user should press the wifi connect button. Once done, in the smartphone app look for WiFi, and select. On selection, the credentials are transferred to the toy which establishes its own connection. This should result in an led color change. Once connected, when the toy is shaken it downloads facts from the server and it will speak it out loud.

* + 1. **Successful Result**

On successful connection to WiFI the toy led color should change. Also when the toy is shaken, if it can make an HTTP request, it will download trivia from the server and say it out loud. This is also visible in the arduino serial monitor.

* 1. **Power on/Power off**
     1. **Significance**

The toy must be able to power on and off in order to operate and conserve battery life. If the toy is unable to power on then it will not function properly as an IoT device. If the toy cannot power off it will be a waste of power and eventually die on it’s own.

* + 1. **Procedure**

A switch on the toy controls the power state of the toy along with an associated LED. When the LED is on, the toy obviously has power. When the switch position is changed the toy shuts down and the LED turns off. The switch is connected in series with the Battery and Charging Module, which enable the battery to charger regardless of whether the switch is open or closed.

* + 1. **Successful Result**

The arduino receives power and all the modules are initialized. The LED lights up as well.

* 1. **Charging the toy**

**5.8.1 Significance**

All hardware within trivia throw toy require lots of charge. In order to make sure every part in the toy works successfully (to detect the motion, the accelerometer must always be on; a high quality speaker is required to ensure outputted sounds are audible and articulate. to connect the toy to WIFI, the WIFI module should always be on) so charging the toy is needed and extremely important.

**5.8.2 procedure**

Trivia Throw Toy use high quality rechargeable battery which can support the toy work at least 3 hours continuously. When the battery is low, the system will automatically indicate the user to charge the toy . To charge the toy, just simply connect the trivia throw toy to the power supply with the wire which has Micro USB port . When user is charging the toy, LED on the port will indicate user the charging states: red LED means the battery is still charging. When the battery is fully charged, the green LED will light up to indicate the user that charging is complete.

**5.8.3 Successful Result**

After charging the battery, every part in the Trivia throw toy can work functionally and whole system can work continuously at least 3 hours. When battery is low, the LED indicator can indicate user to charge the toy. When the battery is fully charged, green LED will light up to indicate user the toy is charged and stop charging.

* 1. **Speak unique Trivia within 1/2s of shake/throw**
     1. **Significance**

The main attraction of the Trivia Throw Toy is the ability to throw a fun fact to a friend. This requires the device to be able to detect a throw/shake and almost immediately begin speaking the fact out loud. A fact must be spoken in a timely enough manner so that users are not waiting for a considerable amount of time.

* + 1. **Procedure**

Every loop iteration of the Arduino, a function is called to check the acceleration of the ball using an accelerometer on the toy. The acceleration is high enough that a clear throw or shake is needed to cause detection, not just a slight movement. After detecting an acceleration, a call to retrieve a fact from the cache is made. The fact is then retrieved from the proper file on the memory card. This fact is the sent to the text-to-speech module and converted from a string to an audible sentence spoken over the speaker.

* + 1. **Successful Result**

The toy successfully detects the acceleration of a throw or shake. The next fact from the cache is then read from memory and fed into the text to speech module. The speaker then audibly verbalized the fact within half a second of the detected shake.

* 1. **Volume button**
     1. **Significance**

The volume button is essential to allow for different styles of play and comfort sound levels. When playing outdoors, users will want more volume in order to compensate for the larger playing space and difficulty hearing. The opposite goes for indoors where users may be sitting next to each other on the couch and do not need the toy to be as loud.

* + 1. **Procedure**

There are two buttons on the toy that control the volume (up and down). By pressing the buttons, the toy makes a call to the TTS module which has built volume control in the library. The function increment or decrement the volume respectively.

* + 1. **Successful Result**

When the volume up button is pressed, the volume is successfully increased by 1 and the when the volume down button is pressed, the volume is successfully decreased by 1.

* 1. **LED status indicators**
     1. **Significance**

The LED status indicators provide outputs for the user to know what is going on with the system. The LED lights indicate connectivity to WiFi, low battery life and retrieval of a fact. These external outputs allow the user to understand issues with the toy and what actions need to take place in order for the toy to operate correctly.

* + 1. **Procedure**

The different colours that correspond to the different statuses are shown in the table below.

|  |  |
| --- | --- |
| **Colour** | **Status** |
| Red | Battery is <= 15% |
| Green | Connected to the Internet (WiFi) |
| Purple | Not connected to the Internet (WiFi) |
| Blue | Updating from the server |
| White | In WiFi setup mode |

* + 1. **Successful Result**

The LED successfully changes colour based on the status of the toy at that time.

1. **Conclusion**

In our functional testing we were able to successfully demo the following:

1. Smartphone application:
   1. Register new User accounts
   2. Login Existing user accounts
   3. View fact history for a user
   4. Transmit data to the arduino using sockets
   5. User Upvote/Downvote on Trivia
2. Arduino
   1. Emit WiFi signal to receive data from Smartphone.
   2. Connect to WiFi and make HTTP requests to the server
   3. LED status indicators.
   4. Successfully read/write facts from/to SD card.
   5. Detect acceleration
   6. Speak text using text to speech module
3. Battery
   1. Charge battery using micro usb cable
   2. Display battery power percentage
4. Casing and Circuitry
   1. 3D printed spherical case to enclose the toy.
   2. Purchased outer foam material.
   3. Printed PCB boards, designed using altium.

Issues/Pending Work:

1. PCB: We placed the order for the PCB early on and received it last week, however a mixup between two pins on the board made it impossible for use it for this deliverable.
2. Arduino Memory: All our components work together individually, however the arduino uno has 32K memory, and despite using an SD card to store facts this flash memory is not enough. An alternative solution is using the Arduino Mega which has considerably more space, however it does not have a mini version (Would make it easier to connect to the PCB). Consequently, when all the code is combined,we run in to various memory issues.
3. Outer Casing: We 3D printed the spherical outer case for the toy in EPIC on 3/21. However we were not able to receive it back on time.