

To: Professor Pisano

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

Team: 24

Date: 2/17/2017

Subject: Second Deliverable Test Report



1. **Project Objective**

1.1 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. **Test Objective and Significance**

**2.1. Power supply**

In order to provide power to whole circuit and entire system, Anker AK-70SMI9300-S12P1NA battery and Samsung OEM standard replacement battery(3.8V/ 9.88Wh EB-B600BUBEAMZ) are used. This is significant because the toy must be able to work functionally at least 3 hours. So better power supply means longer usage life. The objective of this test is to determine which kind of battery are we going to use in the toy to meet the requirement--- work 3 hours continuously and provide safety voltage to the circuit in order not to burn the whole system.

**2.2 Connection between charging port, battery and entire circuit**

To build the overall system of the toy, we have to assemble the battery, the charging port and the circuit. TP4056 module is used to connect these three parts into one block. It has one Micro USB charging port and other two terminals for battery and circuit The objective of testing the connection between these three parts is to determine if the whole system (port to battery, battery to circuit, port to circuit) is working functionally or not. A good testing result means the entire system can work independently and successfully.

**2.3 Android Application**

The Android mobile application needs to scan for the wireless networks available and allow the user to select one and enter the credentials for the Trivia Throw Toy to connect to. The app must also require the user to create an account and login in order to use the app. The mobile application is significant to Trivio because it will better enhance the user’s experience, allowing for customize of the toy by picking fact categories and viewing the history of facts that have been played with the option of upvoting/downvoting it. The voting system is also significant in assisting the server in gathering data in what is considered a “fun” fact.

**2.4 iOS Application**

The iOS mobile application operates as a counterpoint to the Android mobile application to increase the size of the potential market for our product. The app must have identical functionality to the Android application so that the experiences for end users are identical. It must be able to send Wi-Fi credentials over to the Trivia Throw Toy to allow the toy to connect as an Internet-of-Things device. The app must also require the user to create an account and login in order to use the app. Users must be able to view a history of the fun facts they have listened to, as well as select and vote upon the fun fact to crowdsource the best facts for other users to hear.

**2.5 Wifi Module**

The ESP8266 is used to connect the toy the internet as well as act as an interface between the toy and the mobile app. The initiative behind using just a wifi module to carry out these two functions without a bluetooth module is to save cost and reduce energy and power consumptions. the wifi module can be an Access Point as well as a Station/Client at the same time. The Arduino is used as an interface between the toy and the wifi module as the commands from the mobile app are sent to the arduino, which then sends it over to the wifi module for execution.

**2.6 Server and Database**

All the trivia facts are stored on a central SQL Server database, this allows us to reduce memory consumption on the Arduino,as the alternative involves each Toy having its own version of Trivia facts. Having established the significance of a central database, the trailing paragraphs explore the importance of a web server. There are two issues with allowing the arduino to access this database by using SQL scripts :

* Difficult to protect against SQL injection attacks as once an attacked has the connection string to the database(can be read from arduino), there is nothing preventing him/her from modifying the database.
* It is inefficient to access the database directly, especially if data sets are large.

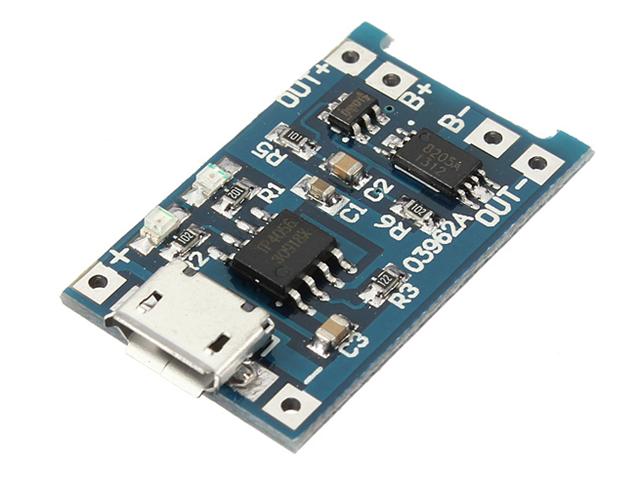
These issues are mitigated by creating a server that exposes a Web API. The server acts as an interface between the database and the arduino. It authorizes requests and allows us greater control over what information any given toy can obtain. Further it enables functionality on smartphones like creation of user accounts, trivia history and voting on trivia. The general objective revolves around testing the interaction between HTTP requests to the API and the corresponding impact on the database. Every API call should draw data from the database, whose contents can be viewed independently using database visualization tools.

1. **Equipment and Setup**

**3.1 Battery and charging module**

Both of Anker AK-70SMI9300-S12P1NA battery and Samsung OEM standard replacement battery(3.8V/ 9.88Wh EB-B600BUBEAMZ) are cell-phone lithium batteries. The size of Anker AK-70SMI9300 is 2.4x2x0.2 inches; weight: 2.6 ounces. Small and light enough to fit in the ball. The voltage is 3.8V; capacity is 2200mAh. The size of Samsung OEM standard replacement battery is 6 x 0.8 x 9 inches, slightly bigger than the first one but it is small enough to fit in the toy. The capacity is 2600mAh but it is lighter than the first one -- 0.8 ounces.

The TP4056 module (size:6 x 4 x 0.1 inches, weight: 0.3 ounces) has two terminals for battery (B+ and B-) which are connecting to the positive and negative terminals of the power source. And another two output terminals(OUT+ and OUT-) are responsible for transferring charges from battery to the whole circuit. User will use Micro USB port to charge the system when there is no charge in the battery.



**3.2 Android Mobile Application Implementation**

Android Studio is used to develop the app with the Visual Studio emulator to test it out for debugging purposes. The app development can be broken down into four main categories: registration/login, managing WiFi, picking a category, and viewing fact history.

**3.2.1 User Login and Registration**

The app startup activity *MainActivity* is implemented to check if the user has logged in before on the phone. The class *SaveSharedPreferences* is created to be responsible for saving the user session so that the user will continue to be logged in upon the next time the app starts up. The functions *saveUsername* and *checkUsername* is created to save the username to the phone if the user has logged in successfully and check to see if there is a username saved. If there is no username saved, *MainActivity* redirects the user to the signin page which also has the option to register for an account. A POST request is made to the server when the user enters in a valid email and password and clicks the sign in button. For user login, the POST request sends the user email and password in JSON format and will receive from the server, if successful, a JSON object containing the user’s ID. For user registration, the POST request sends the user’s first name, last name, email, and password. When the is successful, the app redirects the user, starting the intent *MainMenuActivity.*

**3.2.2 Managing WiFi Connection for Trivio Toy**

*CheckPhoneConnectionActivity* is significant because it provides instructions for allowing the user to connect the Trivio toy to a wireless network. It prompts the user to connect the phone to the WiFi the toy emits. It checks the correct connection to the toy by attempting the start the socket connection and if it receives the toy’s socket emitting message that confirms it is the Trivio toy. Once successful the next activity *ManageWifiActivity* scans for the wireless networks available using the *WifiManager* provided by Android. The user will be able to select a network from the populated list and then enter in the credentials and emits it through the socket connection to the toy. If the toy successfully connects, it will emit a confirmation message that it is connected and the app will display that it was successful.

**3.2.3 User Fact History Retrieval**

*FactHistoryActivity* sends a GET request to the server, giving it the user’s ID. If the GET request is successful, it receives a JSON object of the list of facts the user’s toy has played. It then populates the listview to display them using a custom listview *factItem* arrayadapter. When the fact is clicked, it redirects the user to the *ViewFactActivity*, passing the *factItem* to the intent. *ViewFactActivity*  displays the individual fact and allows the user to upvote/downvote it. Clicking on the upvote/downvote button sends a PUT request to the server on the user’s vote for the fact.

**3.2.4 Fact Category**

*CategoryActivity* sends a GET request to the server for the available categories. It receives the JSON object and parses it into a list which displays the categories on the activity. When a specific category is picked, it sends a PUT request to the server, updating what category the user picked.

**3.3 iOS Mobile Application Implementation**

XCode is used to develop the iOS mobile application in the Swift language. The app development can be broken down into four main categories: registration/login, managing WiFi, picking a category, and viewing fact history.

**3.3.1 User Login and 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 device in *LoginViewController,* 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 GET and POST 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.

**3.3.2 Managing WiFi Connection for Trivio Toy**

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.

**3.3.3 User Fact History Retrieval**

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 fact ID to the next view. This information is used to display the fact and make the PUT request for voting.

**3.4 Wifi Module Implementation**

The ESP8266, shown in Figure 3.4.1, is an Internet-Of-Things device that connects to the Arduino using UART.

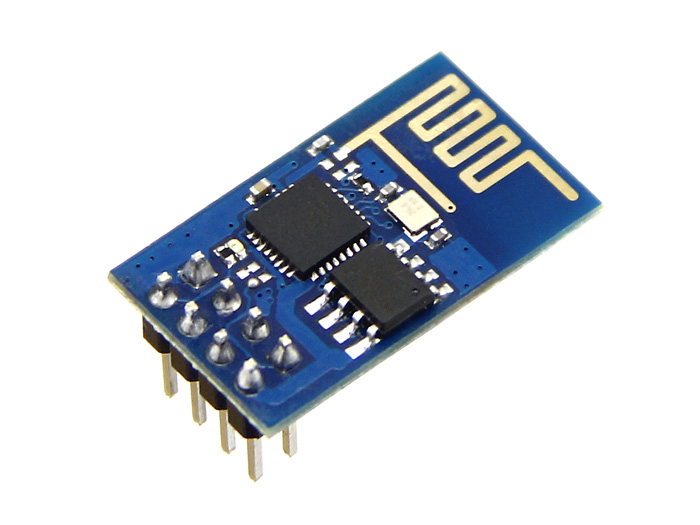


Figure 3.4.1 ESP8266, Wifi module

The Tx and Rx pins on the wifi module is connected to pins on the Arduino that have been defined as Rx and Tx using the SoftwareSerial library. The ESP8266 can handle a maximum voltage of 3.3V only, thus it’s powered by the 3.3V pin on the Arduino. The Arduino has a ESP8266 library that supports connection and command transfer between the two. The ESP8266 has to be set to a specific baud rate depending on the firmware and the ESP8266 library has a function called autoSetBaud(), which attempts to connect to the ESP over the different baud rates and sets it to the specific one that worked. The circuit is shown below in Figure 3.4.2.

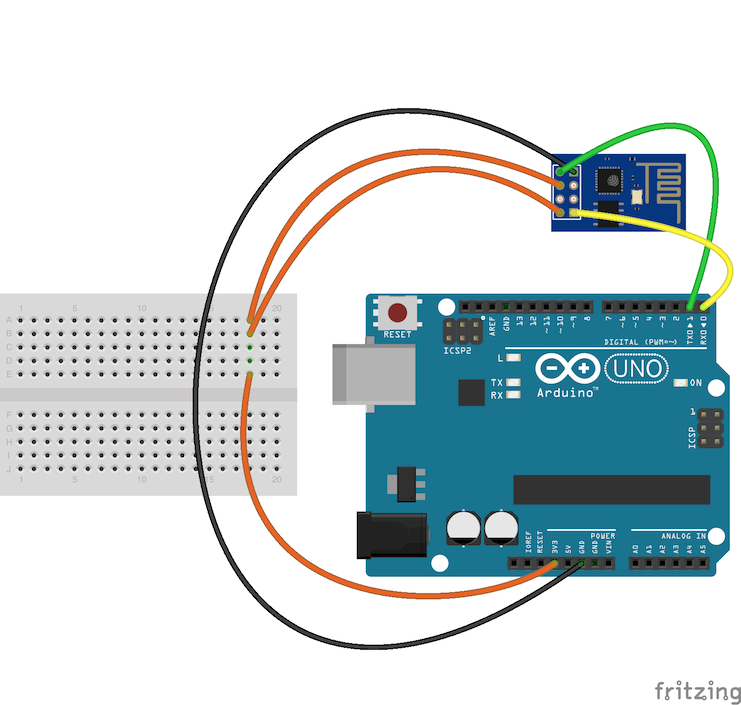


Figure 3.4.2 Connection between Arduino and ESP8266

The wifi module then needs to be told whether to be in Station/Client mode, Access Point(AP) mode or both. The Station/Client mode allows the wifi module to only connect to a wifi network. The AP mode allows the wifi module to act as a router. That is, devices can connect to this network and be on their own private LAN network, but will not be connected to the internet. The last mode enables the wifi module to be in both Station/Client mode as well as AP mode. Therefore, the wifi emits its own signal as well as being connected to the internet.

The wifi module is set to start up in AP mode with a TCP server running on port 333 so that a device can connect to it. Once a device connects to it, the wifi module sends a message to the device informing it that it has received the connection. When the app gets this response, it informs the user that the phone and ball are connected. The mobile app scans for available wifi networks and displays it to the user. The user then selects the wifi network and enters in the password, which the app sends to the TCP server running in the format “ssid:password”.

Once the Arduino receives this information, it breaks it and sends it to the wifi module telling it to connect to the specific wifi network. Once this is done, the wifi module is online and can then connect to the server to get the facts and TCP server is stopped. A reset button will be implemented that will allow the user to change the wifi network they want to connect to. Once the module connects to a wifi network, it will remember its credentials and automatically connect to it when detected.

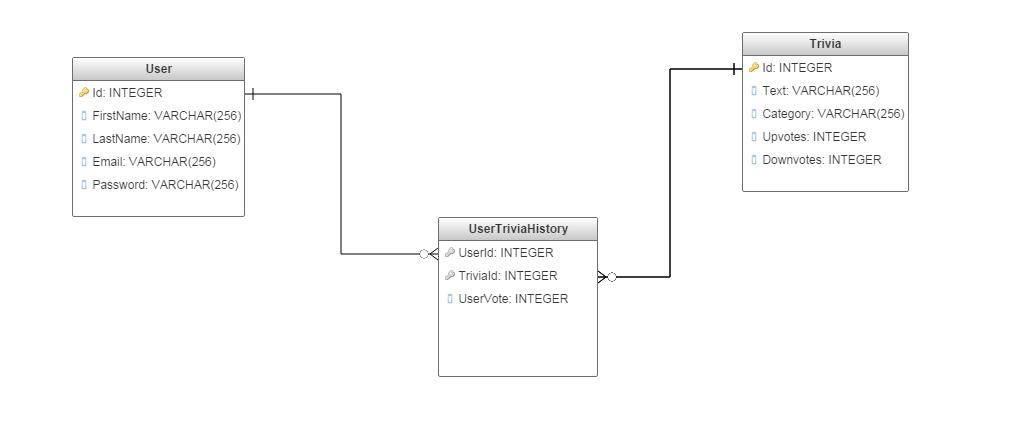
**3.4 Database and Server Implementation**

**3.4.1 Database**

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

* **User**: Information about all user acounts
* **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.

The diagram below shows the Database schema.



The database is created using ASP.NET Entity Framework, which is an Object Relation Mapping tool for databases. The benefit of this approach is it allows us to create, modify and access database using classes and properties instead of SQL statements. The example below shows how instead of using a SQL statement, all the Trivia in the database can be returned using Db.Trivia.ToList() method.

//Return List of all Trivia in Database  
 [HttpGet]  
 [Route("")]  
 public IEnumerable<Trivia> GetTrivia()  
 {  
 return Db.Trivia.ToList();  
   
 }

**3.4.2 Server Implementation**

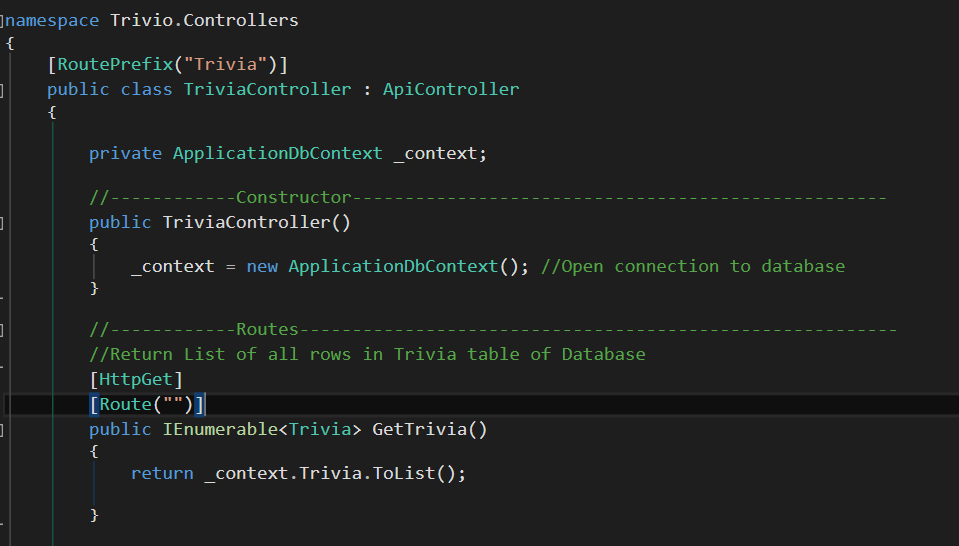
The server is implemented using ASP.NET framework. It performs multiple functions to access the database found primarily in TriviaController.cs and UserController.cs. The server exposes two broad categories of Web API that follow the REST convention. These are:

* 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 |

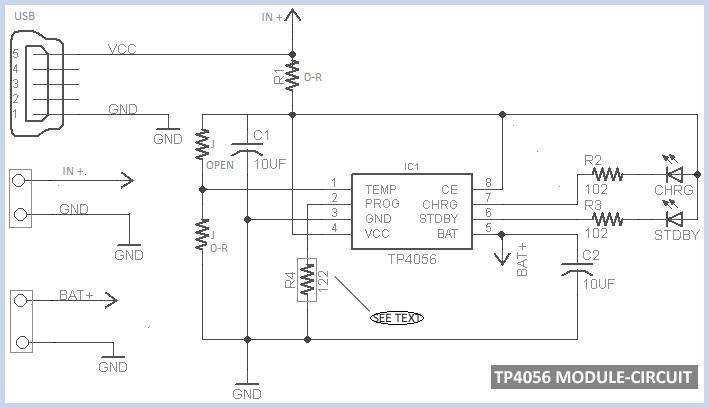


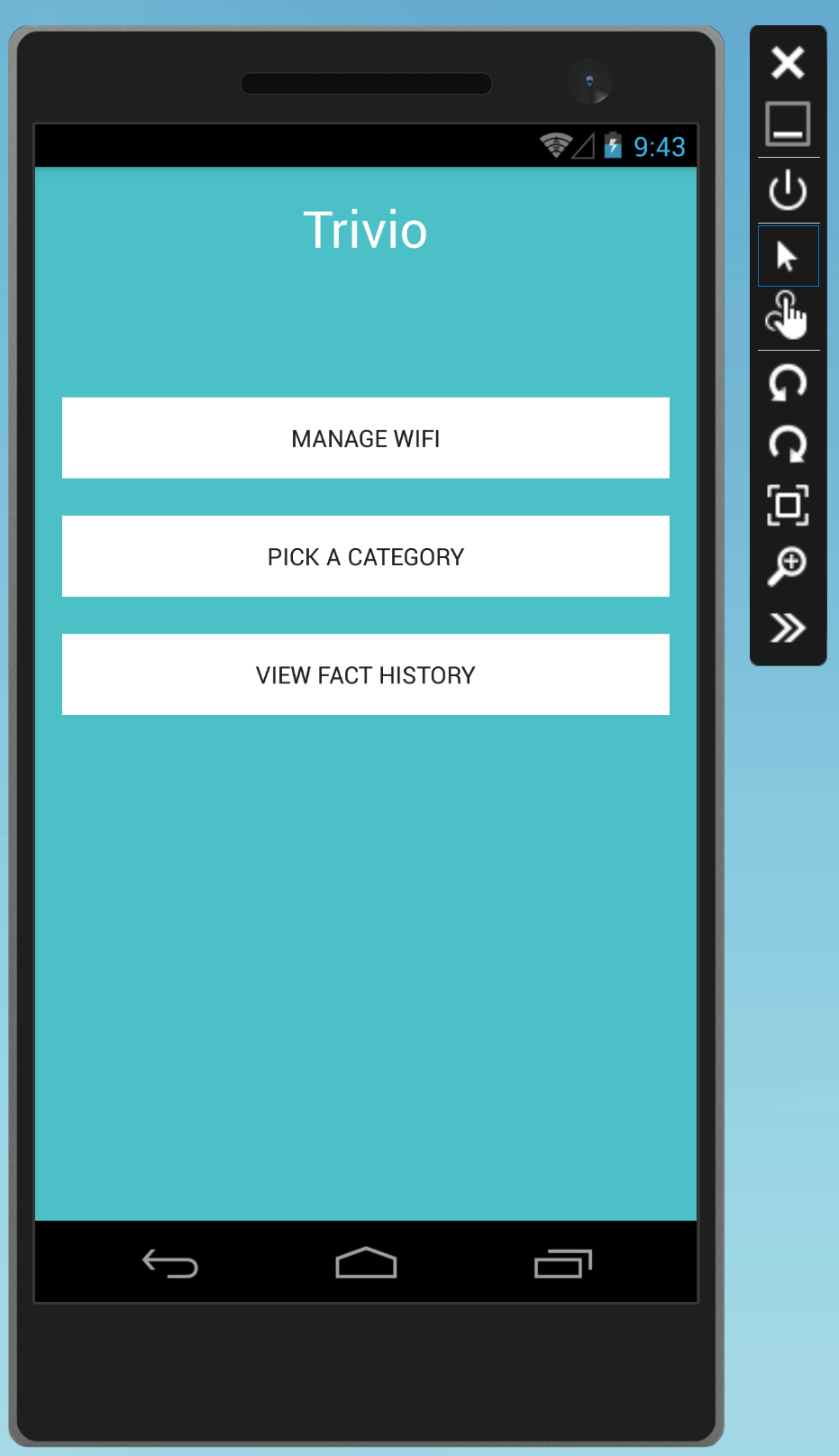
Implementation of the first endpoint GET:/Trivia is shown on the right. The code leverages entity framework to treat tables as objects and query them using built in methods instead of SQL.

1. **Measurements and Data**

**4.1 Battery and TP4056 module**

When fully charged, the voltage between Anker AK-70SMI9300-S12P1NA’s positive and negative terminals is 3.8V. Capacity is 2200mAh. The input voltage of the circuit is 3.7V; other modules within the circuit work functionally. When fully charged, the voltage between Samsung OEM standard replacement battery’s positive and negative terminals is 3.8V/9.88Wh. Capacity is 2600mAh. The input voltage of the circuit is 3.75V; other modules within the circuit work functionally

TP4056 module is used for single lithium or parallel lithium battery charging. The charging model is linear charging with Micro USB input interface. When battery is fully charged, the power will supplied by the battery. When there is no charge in the battery which means the toy need to be charged, user need to connect the Micro USB port to the power supply. The red LED on the module should lights up to indicate that the battery is charging. When the battery is fully charged the green LED should lights up that indicate the user to stop charging.

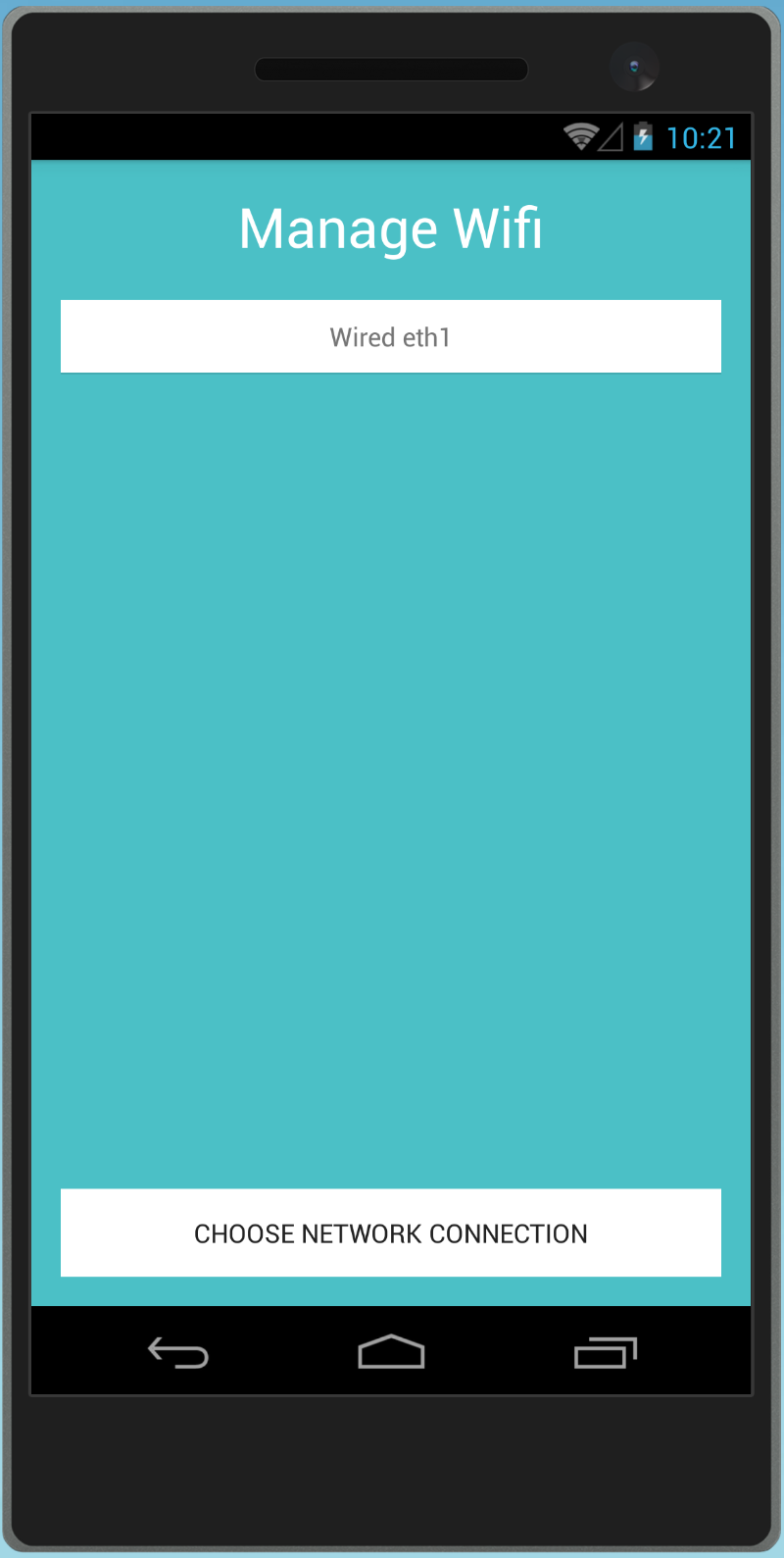
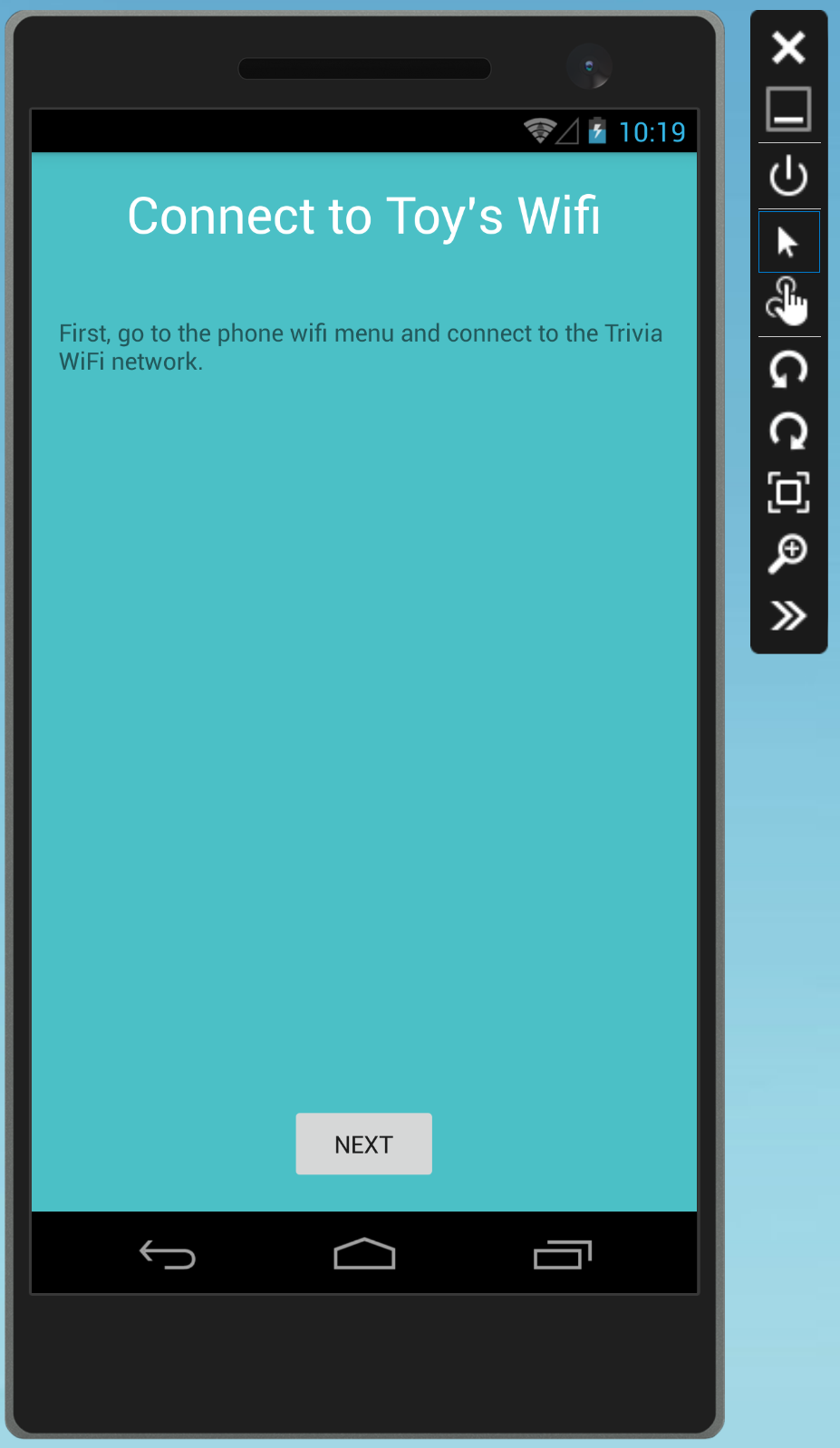
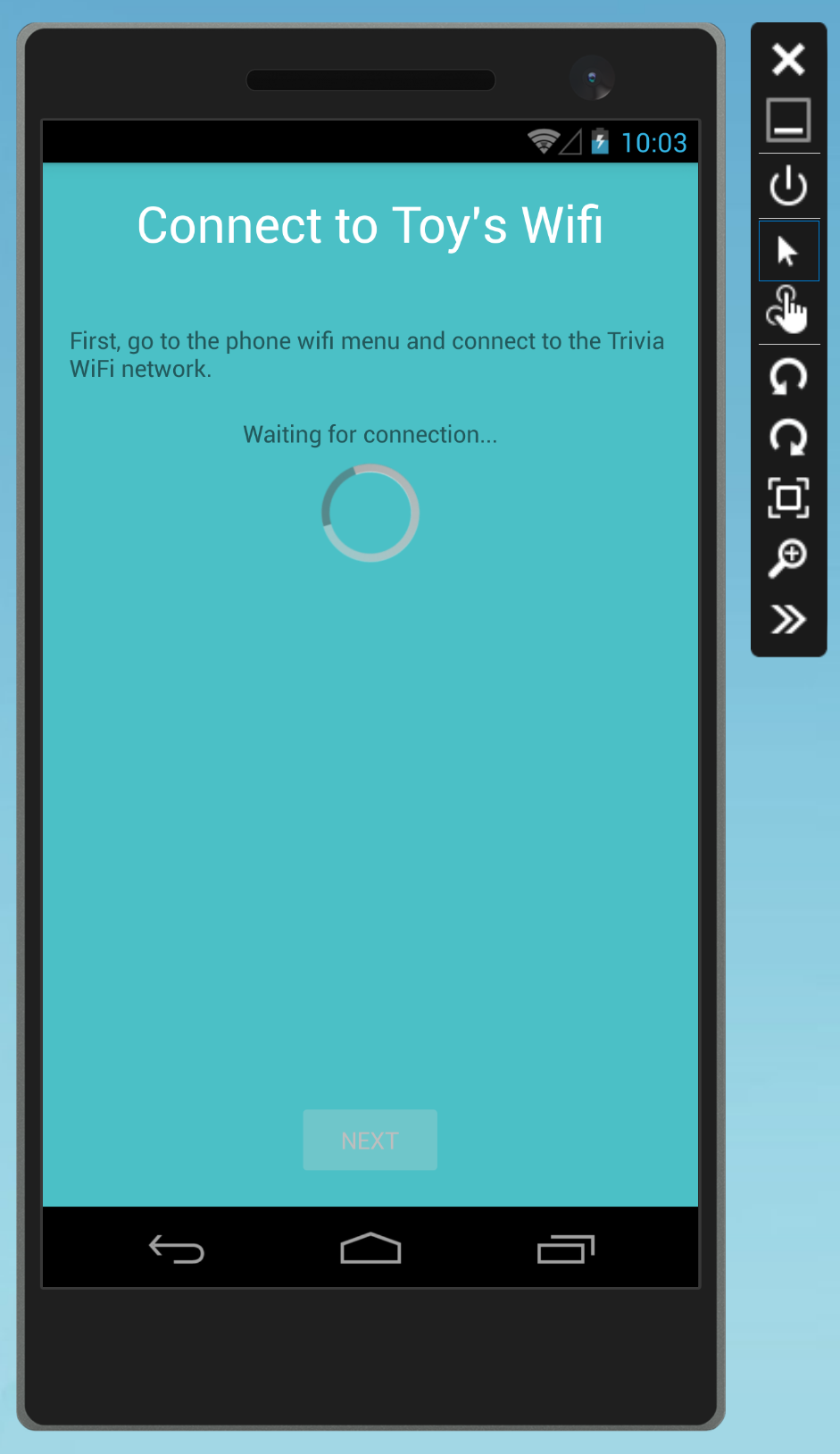


**4.2 Android App Results**

When the POST and GET request for the user logging in is successful, it will redirect the user to the Main menu page of the app.

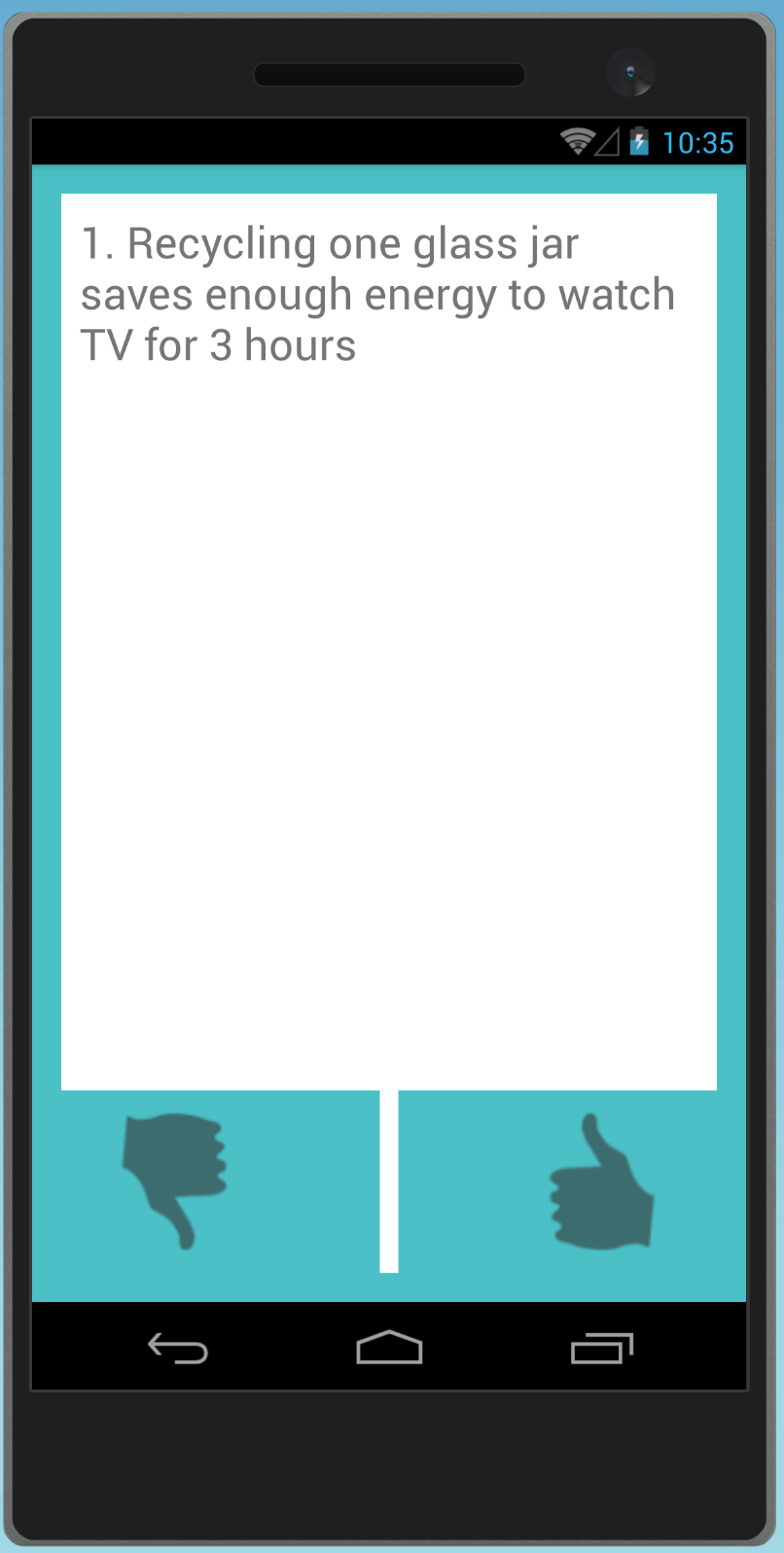
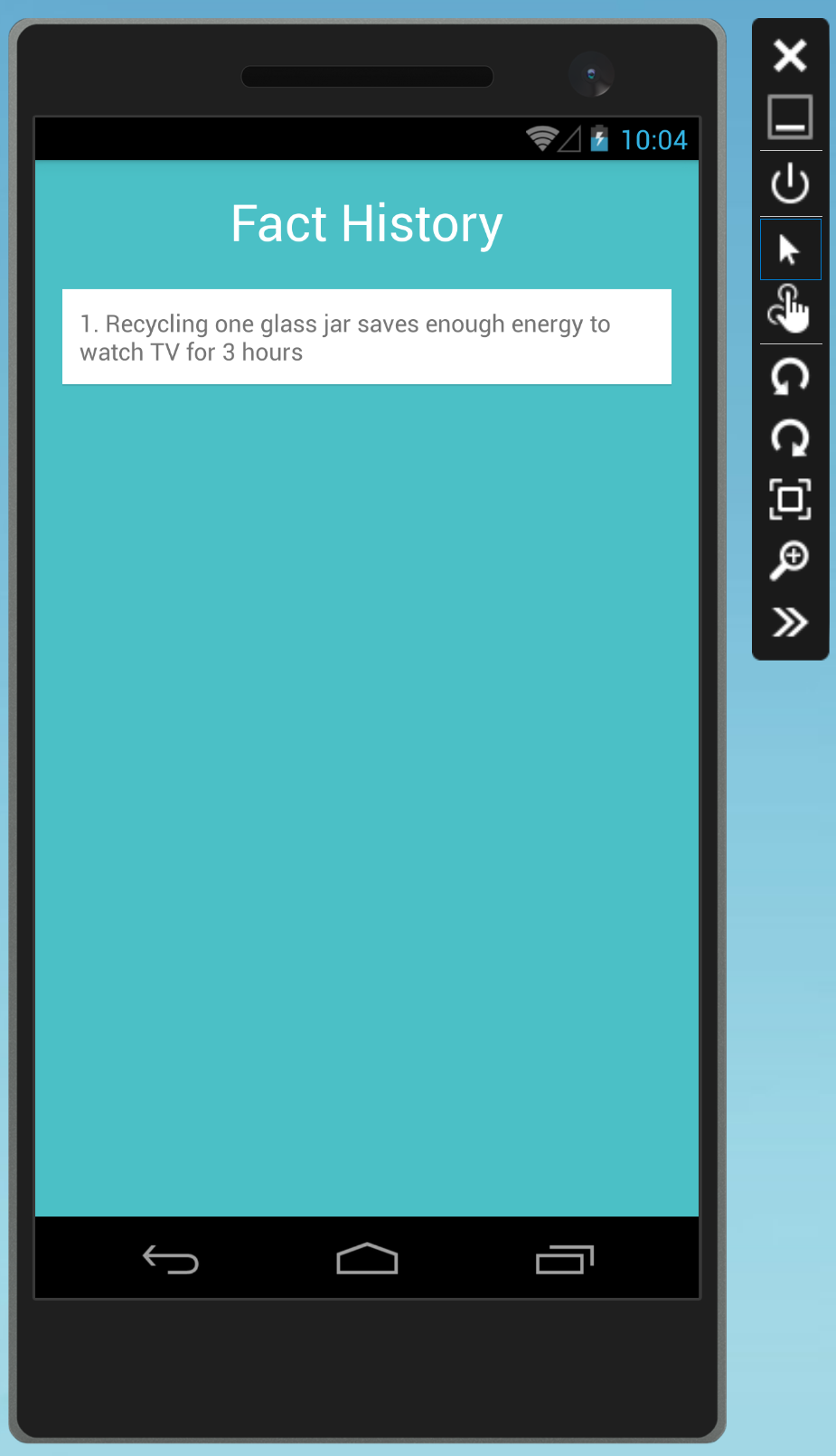
When the socket connection is successful, the loading progress will stop and allow the user to continue to the next page for selecting a wireless network. When the scan for wireless networks is successful, the page will display a list of networks to choose from as shown in figure 4.2.1.

Figure 4.2.1 Connecting the Trivio Toy to Wifi via the Android app



When the GET request for the user’s history is successful, the app will show a list of the facts the user has played on the toy, as shown in figure 4.2.2.

Figure 4.2.2 Displaying User Fact History and Voting



**4.3 iOS App Results**

A user is able to register through through the SignUpViewController with first name, last name, email address and password. Once registered the user is given a unique userID which is stored on the phone’s device. On the login screen when a unique userID is detected, the user is automatically logged in to the application. If there is no userID already detected, then the user logs on with their email and password and receives their userID. This ID is then stored to start the session and the user is logged in to the application.

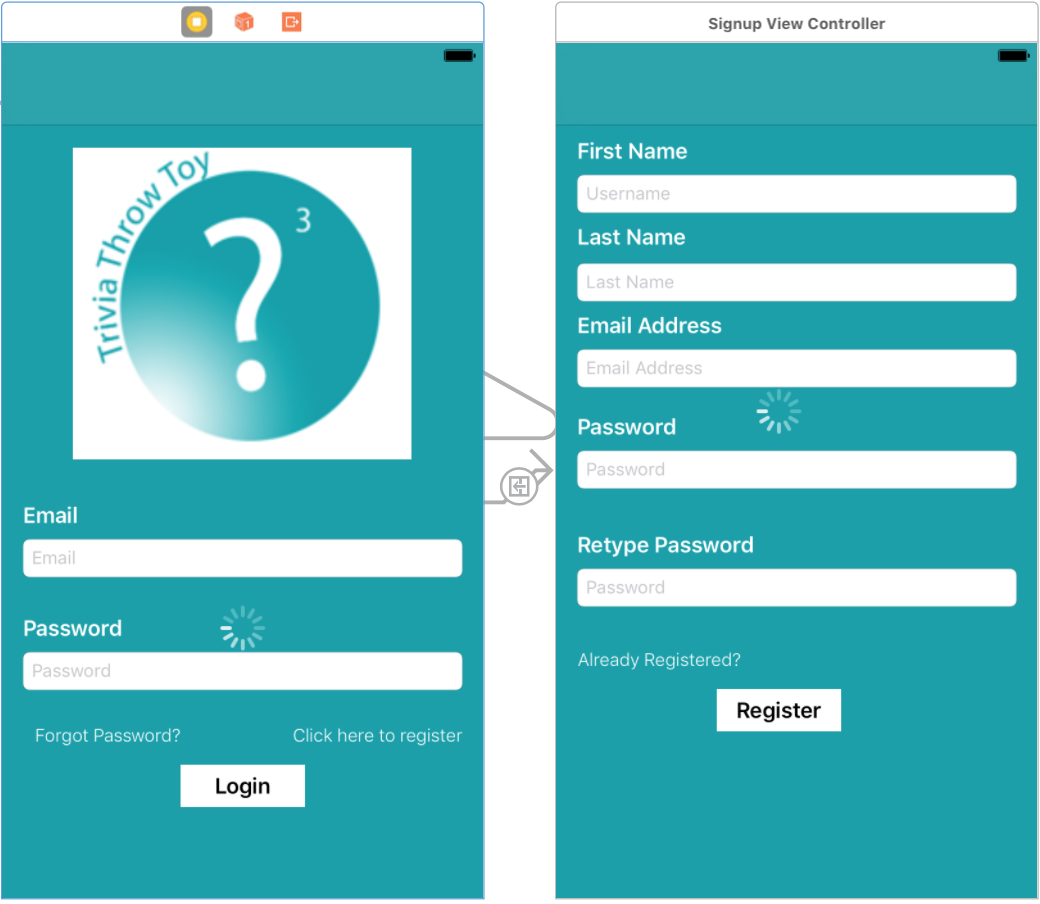


Figure 4.3.1 Login and Registration Screen

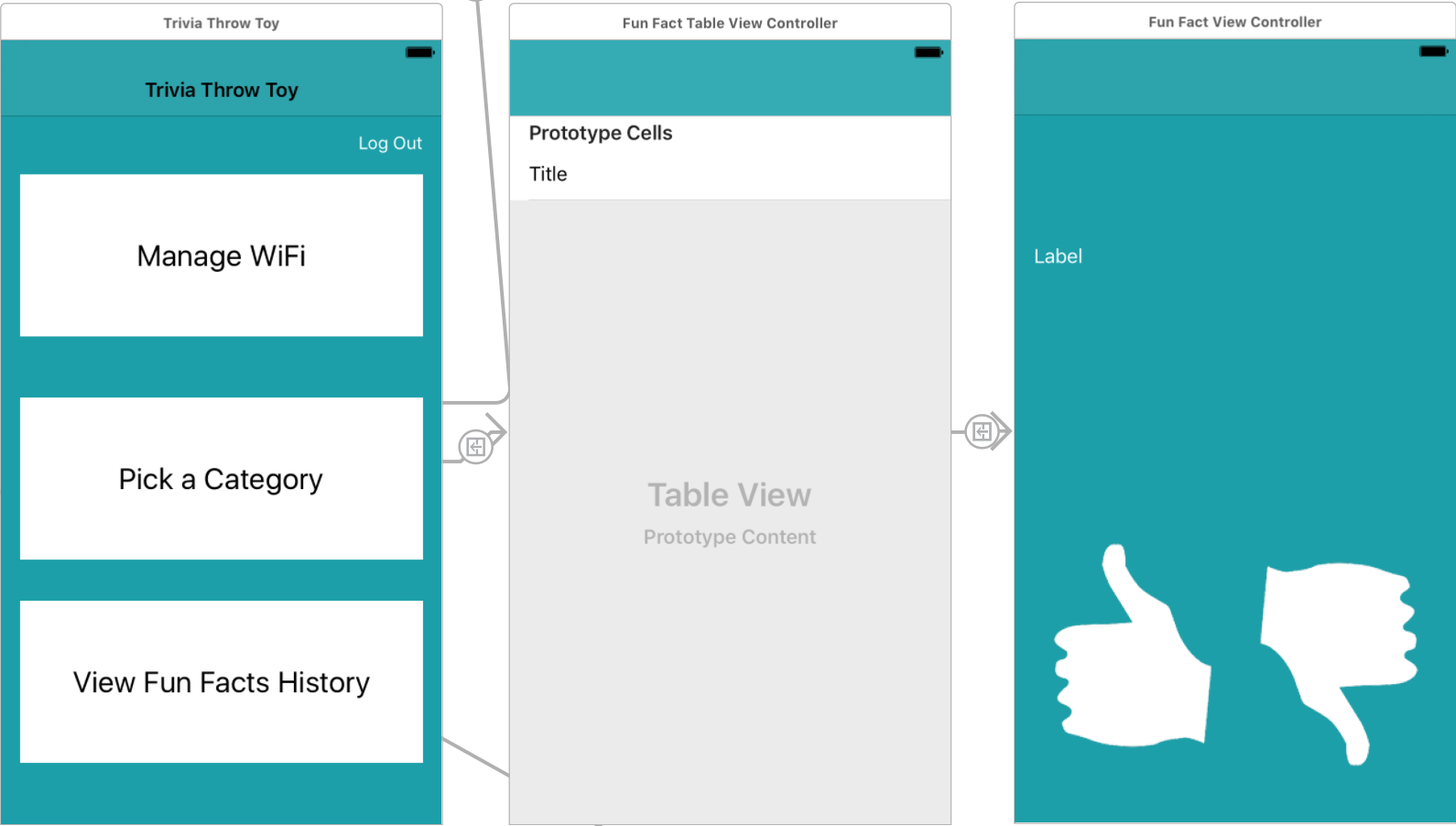
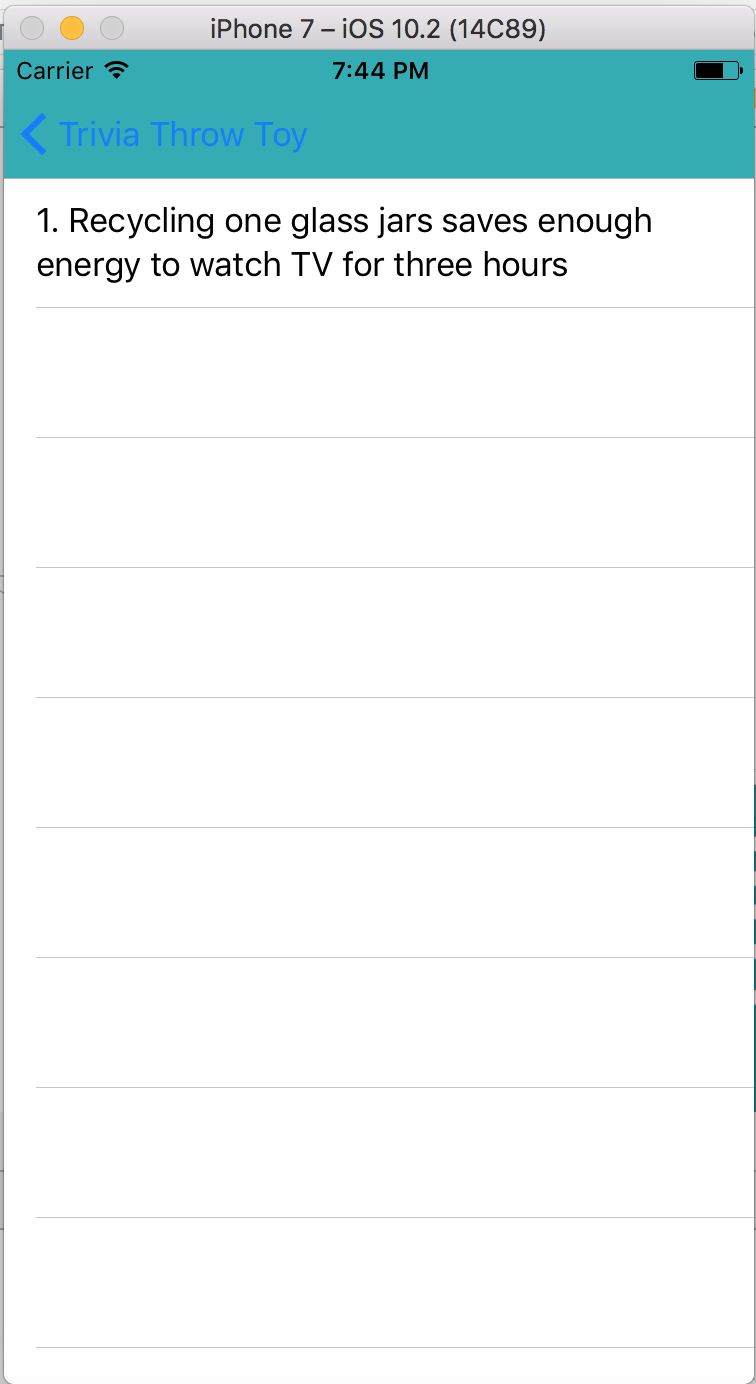
Once logged into the application the user has the option to manage the wifi, pick a category, or view their fun fact history.

Figure 4.3.2 Main Screen, FunFactTableView and FunFactViewController

By clicking on fun fact history, a table view comes up which is populated by previously heard fun facts. Clicking on a fact segues to the *FunFactViewController* where the user can up/down vote the fact in the server as shown in Figure 4.3.3.

Figure 4.3.3 Displaying fun fact history table view and view controller for voting

**4.4 Wifi Module**

Once the wifi module is in AP state, it has an IP address and when devices connect to it, it assigns those devices IP addresses within the network. The devices can send data and receive data from the wifi module as it acts as a TCP server listening on port 333.

The user is only able to access the wifi networks once it connects with the wifi module and receives an OK signal. The app then shows the user avalable wifi networks that he/she can join to. Once the wifi module receives the credentials of the wifi network the user wants to connect the toy to, the TCP server is stopped and no more communication occurs between the module and toy. Figure 4.4.1 below shows a screen of the Serial Monitor of the Arduino printing out the IP of the module, IP of device and statuses of the server and data transfer.

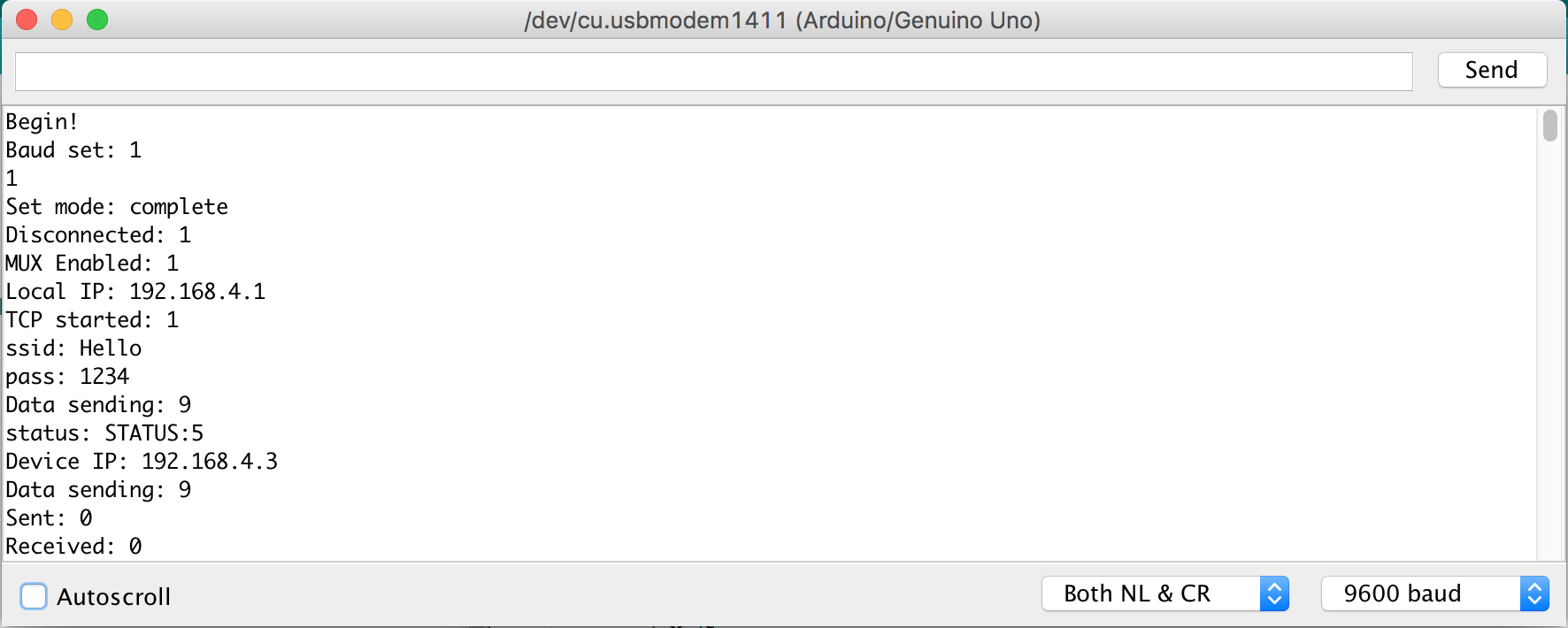


Figure 4.4.21 Arduino Serial Monitor with results from wifi module

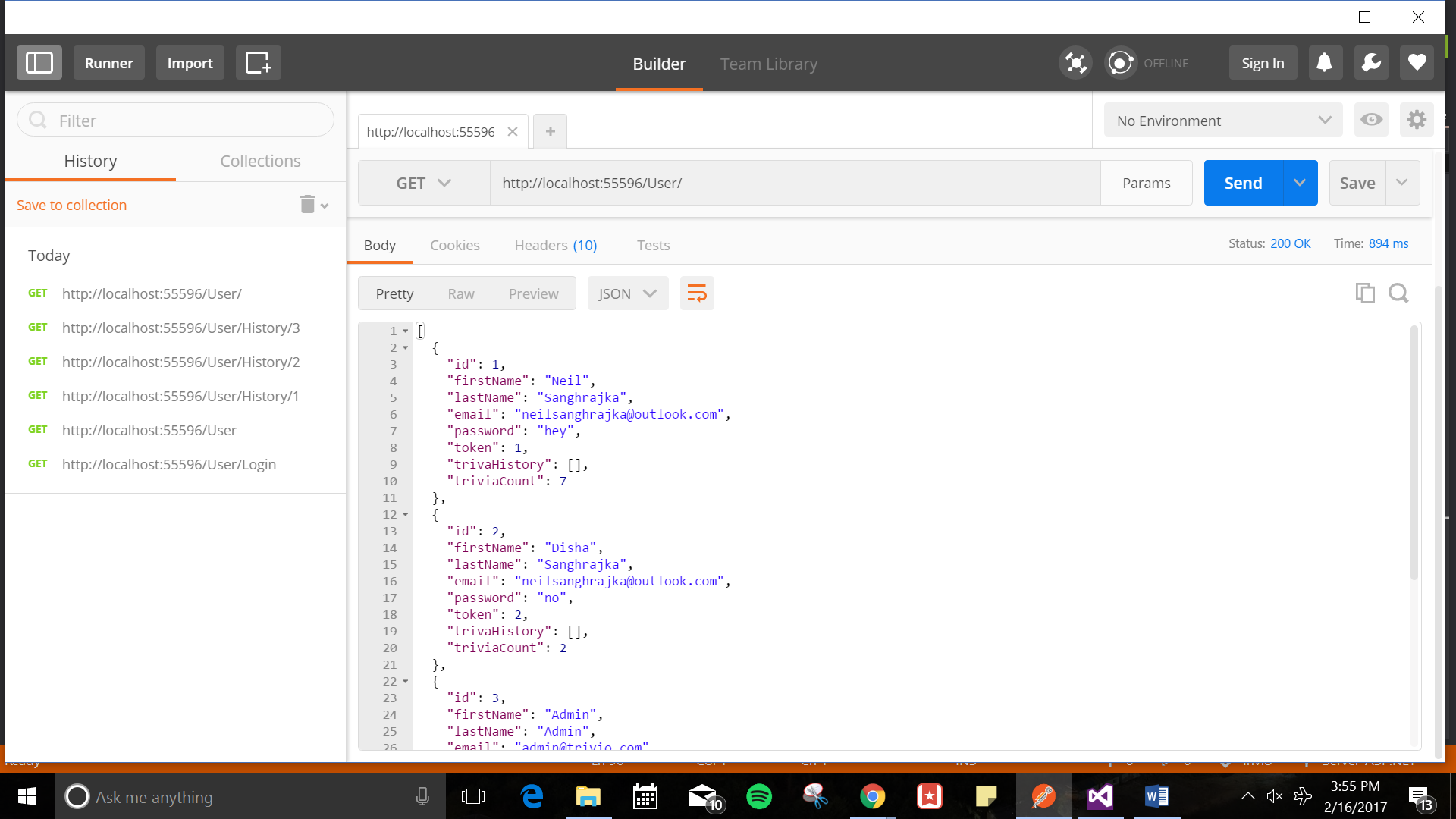
**4.5 Server and Database**

To test the API and database 3 two primary tools were used.

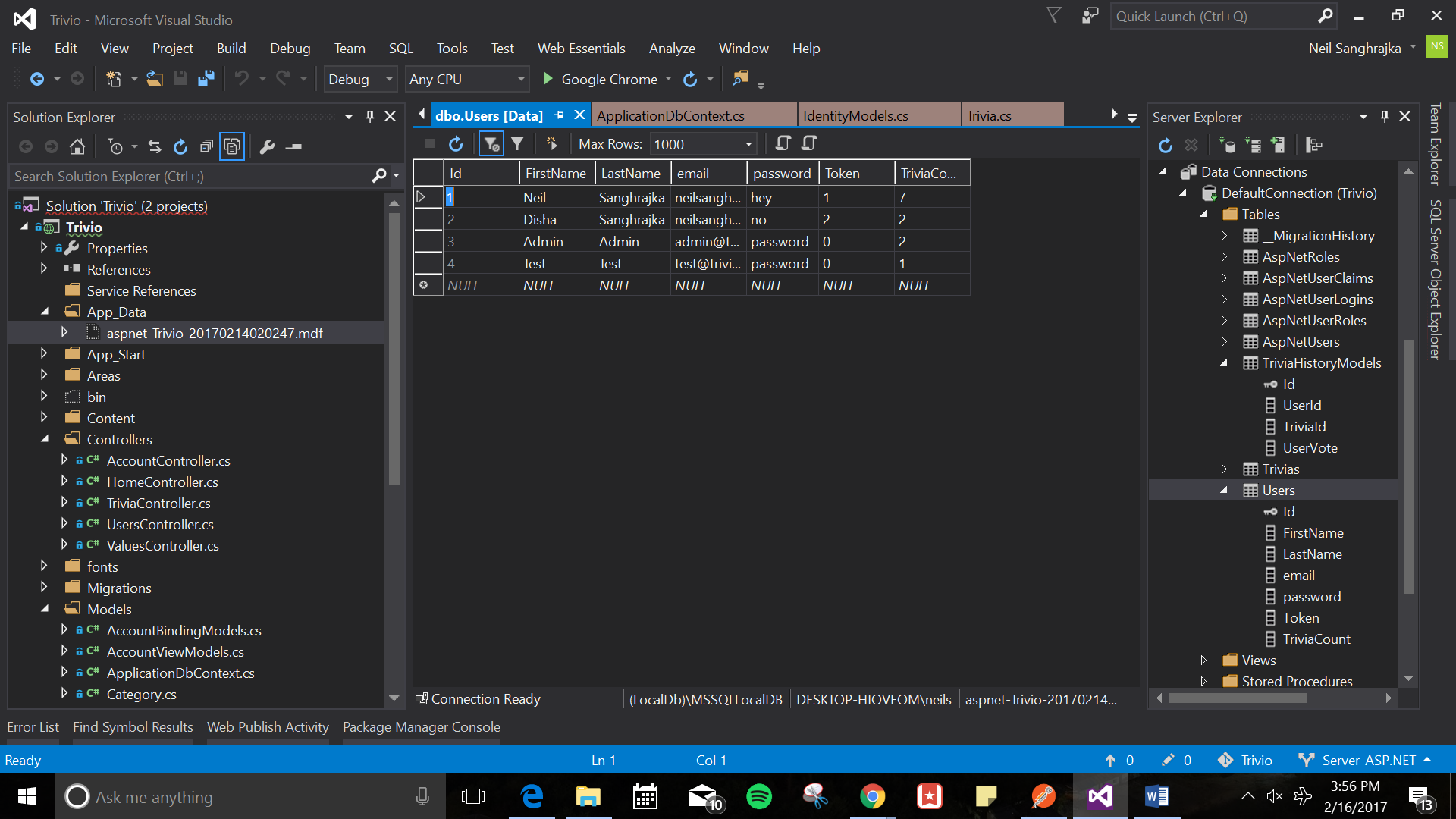
1. **Postman**: This Chrome Web Application is used to make HTTP requests and view the response from these requests. By testing our server in this manner, and not directly from the smartphone app or arduino we are able to independently test the server code. If we get an error using Postman, we can safely conclude that error is in the server code and not any other component of the app.
2. **Visual Studio Server Explorer**: This extension inside Visual Studio allows us to view and modify local database tables . Every API call mentioned above, either makes changes to the database or extracts data from a database. By making HTTP requests and then checking the impact on the database using a separate independent tool, we can ensure that the server code functions properly.
3. **Azure SQL console**: The SQL database is hosted on Microsoft Azure web services. Using the SQL console we can make direct queries to our remote database. This serves a similar function to Visual Studio Server Explorer, however we can now test online databases.

The testing procedure varied for different endpoints, however overall it can be summarized as:

1. Make HTTP request to the endpoints with correct data (Use POSTMAN)
2. Check if data synchronizes with the database.
3. Make HTTP request to endpoint with invalid data.(User POSTMAN)
4. Ensure that data does not persist.(Use Visual studio to visualize tables)

The results for every HTTP request were exactly as expected. Here is an example of a succesful test case for the endpoint GET:/Users/

1. Return value of HTTP request as seen in Postman

2. The corresponding table in database as seen using visual studio.

Similarly, all other test cases also passed. However, the only test case that failed was successful deployment of the server code to Azure. The local server works perfectly, however due to deployment issues the goal of having a live server was not met for this deliverable.

1. **Conclusion**

**5.1 Power supply and charging the battery**

With fully charged, the battery works at least 3 hours continuously. Also the battery can be charged normally when it is connected to the power output. In conclusion, the battery -- port-- circuit system works normally when there is no outside power supply connected. When the battery is fully charged, the system will stop charging the battery to prevent the overcharge. Also, the LEDs on the TP4056 module are able to indicate users the charging status.

**5.2 Android Application**

With the android application fully tested on the emulator, the apk was downloaded onto a physical Android phone to test it out. The phone was able to allow the user to register, login, and choose through the categories. When testing it out with the toy’s WiFi module, the application was able to open up a socket connection with the toy when connected to its WiFi. The application was also able to scan for the wireless networks available once the socket connection has been made and allowed the user to pick a network and send over the credentials through the socket. A confirmation page allowed the user to know that the toy was able to establish a successful connection with the network.

**5.3 iOS Application**

The iOS application is thoroughly tested via the XCode emulator. It is able to make the proper GET and POST calls for logging in and registering. The user session is able to be saved on the phone’s device so users do not need to log in again. Once logged in, the user is able to view fun fact history through a GET request with the userID and can also up/down vote a selected fact using the unique factID.

**5.4 Wifi Module**

The wifi module successfully operates in the mode that supports bot AP and Station/Client functionality. The TCP server is created to listen on port 333 and the mobile app is able connects the ESP8266 network. Once connected, the wifi module sends an OK signal to the mobile app, and the mobile app allows the user to select a wifi network to connect the toy to. Once the wifi module receives this data and successfully connects to the internet, the TCP server is stopped.

**5.5 Server and Database**

The database is succesfully hosted on Microsoft Azure web services. All the required tables are created and querying the database worked perfectly. All the requisite API endpoints also meet the specifications and are able to modify the database correctly as shown in the demonstration. The server was deployed locally for this deliverable as there were deployment issues when hosted on Azure. This needs to be fixed in the immediate future.