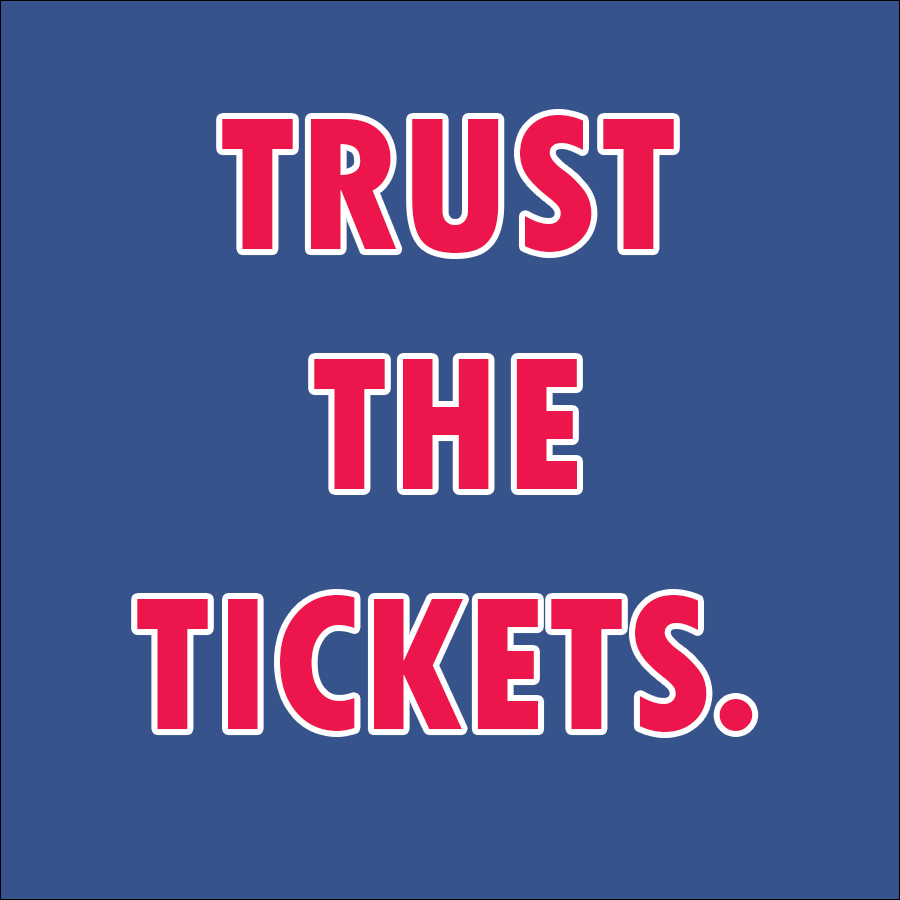
TrustTheTickets.com  
Detailed Design Document

****

Initial Draft Date: October 9th, 2017

Created by:

Anthony Orio

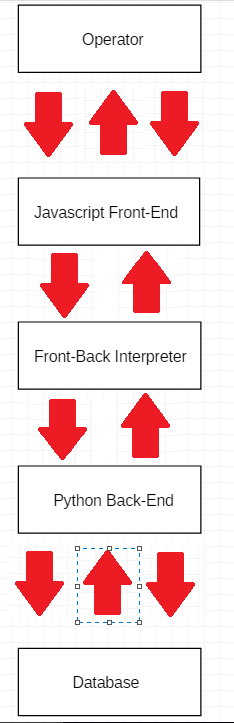
Christopher McKane

Curtis Baillie

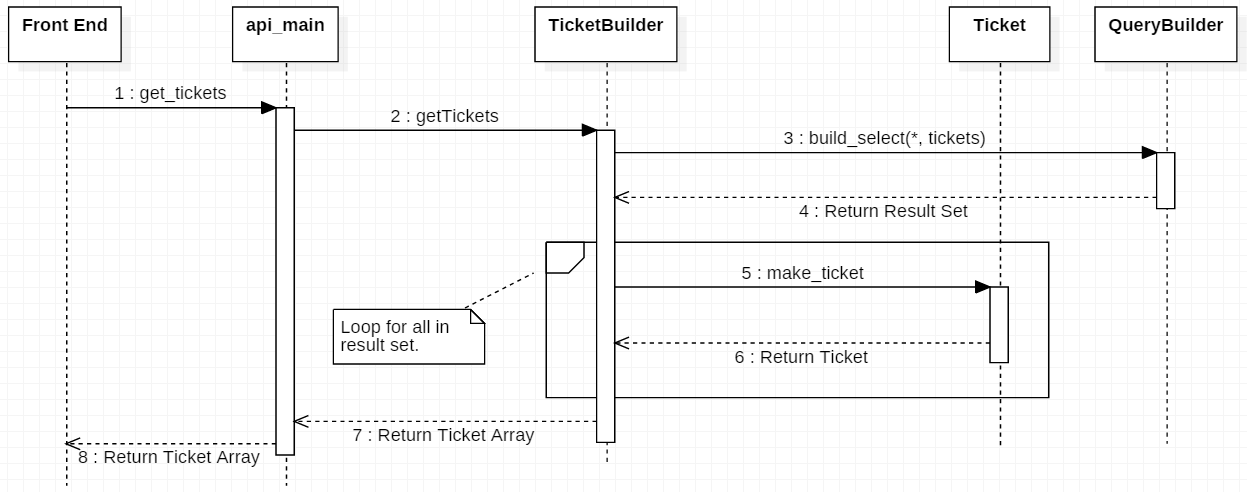
Derek Gaffney  
Jon D’Alonzo

Thomas Harker

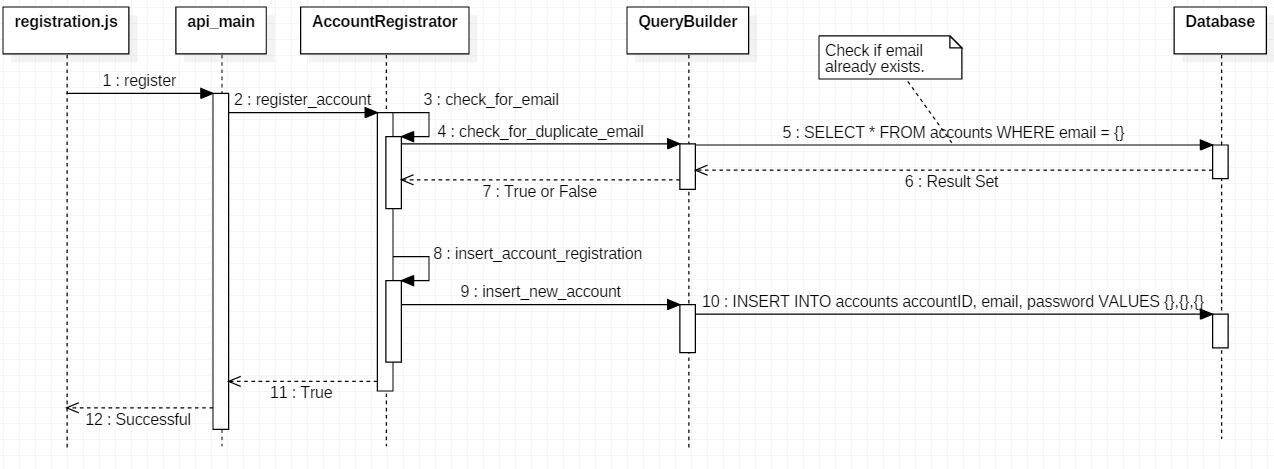
<https://github.com/JonDalonzo/Senior-Project>



Here lies the general flow of communication throughout the website. The operator will interact with the **Javascript Front-End**, which consists of many methods for each part of website. The three arrows demonstrate that the **Javascript Front-End** can communicate with **the Front-Back Interpreter** in many different ways. This includes methods for transactions, displaying tickets and registering accounts on the website. The **Front-Back Interpreter**, which consists of both Javascript and Python code. The Front-Back Interpreter is the central interface that will be used when calling methods within the Python code. The calls will be passed to Python in this interface and the python will return its data to the front end through this interface upon calls that make requests for data. The single IN and OUT arrows show how communication components can only occur through one path. The **Python Back-End** is responsible with handling all data related activities, such as communicating with the Database and performing operations that the user requests through the **Javascript Front-End**. The three arrows show how the **Python Back-End** can communicate with the **Database** through more than one path, as in many Python methods can access or update the **Database**.



The sequence diagram above shows the flow of operation from each component when retrieving a set of tickets to display in the front end. The Javascript Front-End will access methods in api\_main, a part of the Front-Back Interpreter, which will get the tickets from the TicketBuilder. The TicketBuilder will query for the tickets based on the search filters specified by the Operator on the front end. The result will be fetched from the database by the QueryBuilder, who is responsible for accessing and retrieving data from the database. The result set is returned to TicketBuilder, where a loop will make a Ticket object from each row of data returned in the Result Set. The Tickets will be packaged into an array and returned back up to the front end where it will be display to the operator.



The sequence diagram above depicts the operator action of registering an account successfully on the website. The registration Javascript file will call the register method in api\_main. The api\_main class will then call the register\_account method in AccountRegistrator, where it will check for duplicate emails. To check for duplicates the AccountRegistrator attempts to SELECT the email inputted by the operator from the database through the QueryBuilder. If this is successful, then the email exists otherwise the email does not exist yet and execution can continue. The valid email is then inserted into the database with information. Successful insertion of the new email is replied back to the operator.

Use Cases:

seller use cases

1. seller uploads tickets

upload one ticket

upload multiple tickets

2. seller modifies group

remove one or more tickets

add one or more tickets

update price of group

3. Seller creates account

4. Seller modifies account

5. Seller deactivates account

6. Seller views past transactions

buyer use cases

1. Searching functions

buyer searches for a specific game

buyer searches for games a specific team is playing in

buyer searches for best value (price)

buyer wants to browse

2. Buying functions

buyer buys subset of tickets in a group

buyer buys all tickets in a group

3. Buyer creates account

4. Buyer modifies account

5. Buyer deactivates account

6. Buyer views past transactions

General Use Case

Handling box seats vs. section seats

System use cases