# CSIS 659
# Service-Oriented Computing
## Final Project Write-Up

Hassam Solano-Morel

# Table of Contents

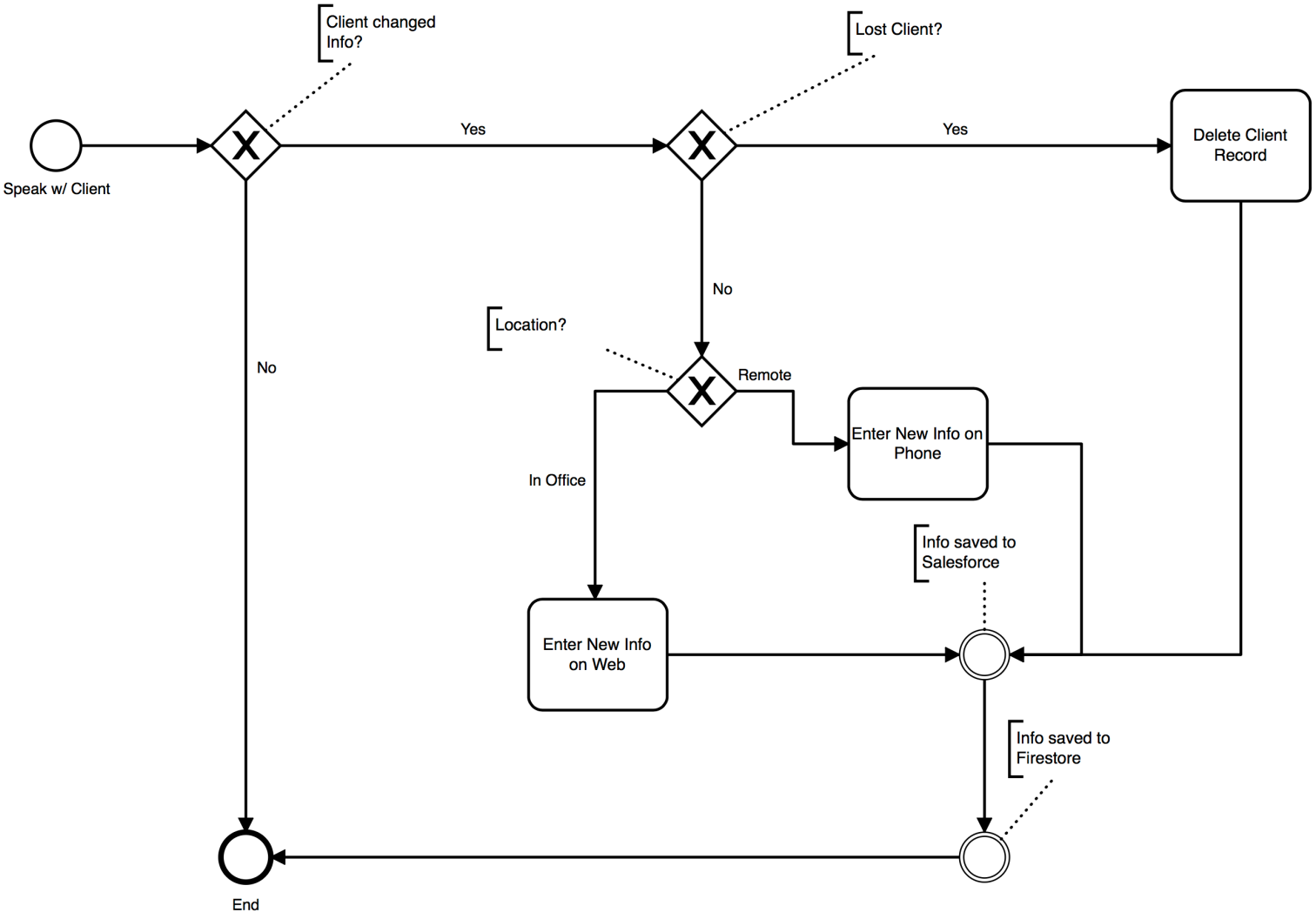


*Source code, screenshots, and demo videos can be viewed @:
https://github.com/hassamsolanomorel/CSIS659_Final

# Introduction

As an aspiring entrepreneur I believe that practice makes perfect. To this effect I have started my own software development shop named 'SolanoSoftwareSolutions' (SSS for short). One of the many administrative tasks that is essential to my business is keeping accurate records of my current clients. In order to achieve this goal, the project being presented herein describes a proof of concept for a cross-platform client list application called "SSSClients". The application leverages several web services along with native frameworks to maintain an accurate record of client information that is redundantly archived (in case of emergencies).

# BPMN Model



# Web Service - (Apex REST API)

The backend hub for the SSSClients application is built on force.com using the Apex programming language. Apex is a full-featured derivative of Java with extras that allow for easy DML invocation. As a plus, because this version of Apex runs on Salesforce's lighting platform, it can directly manipulate sObjects maintained on the platform. This section describes the REST API that handles synchronization across the application.

**Class: SSSClientsManager**

The Apex class 'SSSClientsManager' contains all of the code needed to implement the backend REST API for the application.

**getAccounts( ):**

- HTTP Method: GET
- Purpose: Retrieves all SSSClient__c records.
- Return: All SSSClient__c record objects in a JSON array.

**newClient( ):**

- HTTP Method: POST
- Purpose: This method acts as an 'upsert' method.
  - It will first look for an existing id within the request body.
    - If one is found the corresponding SSSClient__c record will be updated with the provided changes (also contained in the request body).
    - If no id is found, the method will insert a new SSSClient__c record with the provided field information.
  - Once the record has been successfully upserted into the Salesforce database a POST request will be sent to the Firestore service with the record information
    - WORKAROUND NOTE: Apex does not allow for PATCH requests to be sent. However the Firestore API expects PATCH requests when one only wants to update field values. To ensure accurate syncing on Firestore, the method will first initiate a DELETE for any current Firestore record of the SSSClient__c object. It then sends a POST request with the updated information.
- Return: The method returns a JSON encoded version of the record that was upserted.

**deleteClient( ):**

- HTTP Method: DELETE
- Purpose: Deletes the SSSClient__c record with the provided id
  - The method first removes the record copy in Firestore and then deletes the record on Salesforce
- Requirements: The record id must be included as part of the request URI
  - i.e. …/SSSClients/{id}
- Return: A JSON encoded string with a status and message for the transaction

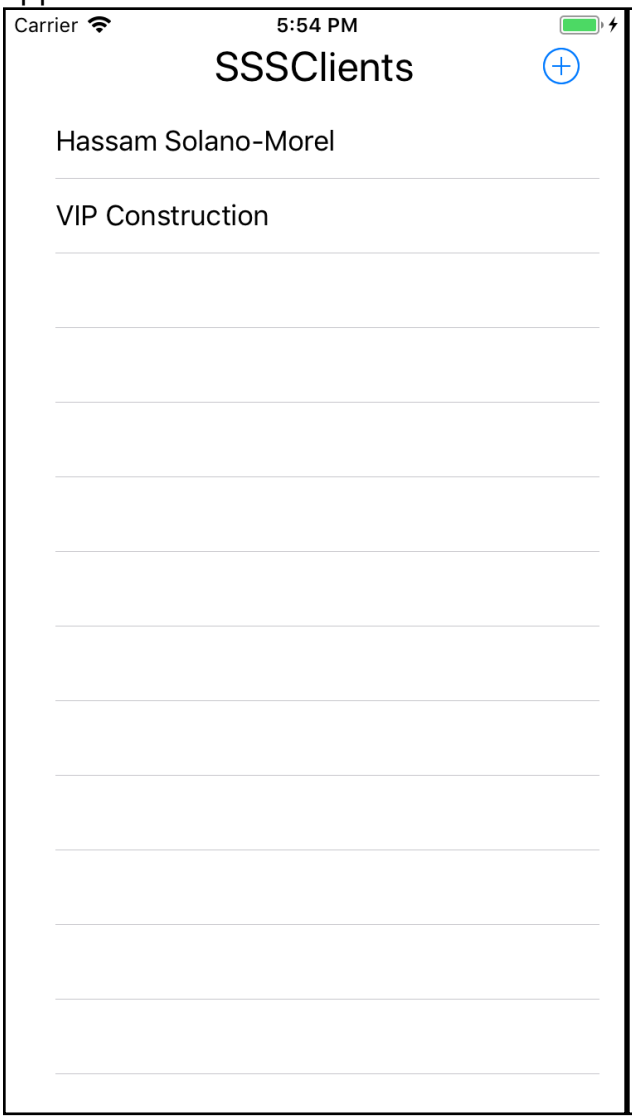
**syncFS(id , action) - ‘syncFireStore’:**
- Internal method
- Annotations: ‘@Future(callout=true)’
  - This annotation indicates that the method is runs asynchronously, after all DML operations have been completed. It also indicates that the method itself will make HTTP calls.
- Purpose: This method is tasked with making the required HTTP calls to the Firestore service as determined by ‘action’
  - The method acts as more of a dispatch to more fine-grain methods described below.

**updateFS( ):**
- Internal method
- Purpose: This method acts as the upsert method for records maintained in Firestore.

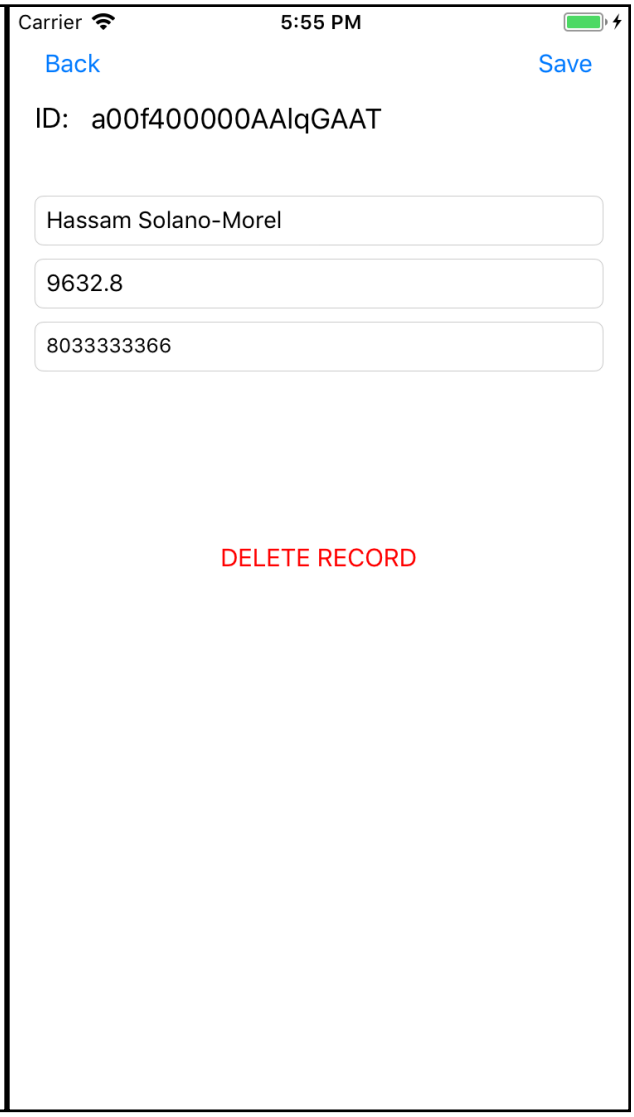
**deleteFS( ):**
- Internal method
- Purpose: Deletes the specified record from Firestore.

# Mobile App - (iOS)

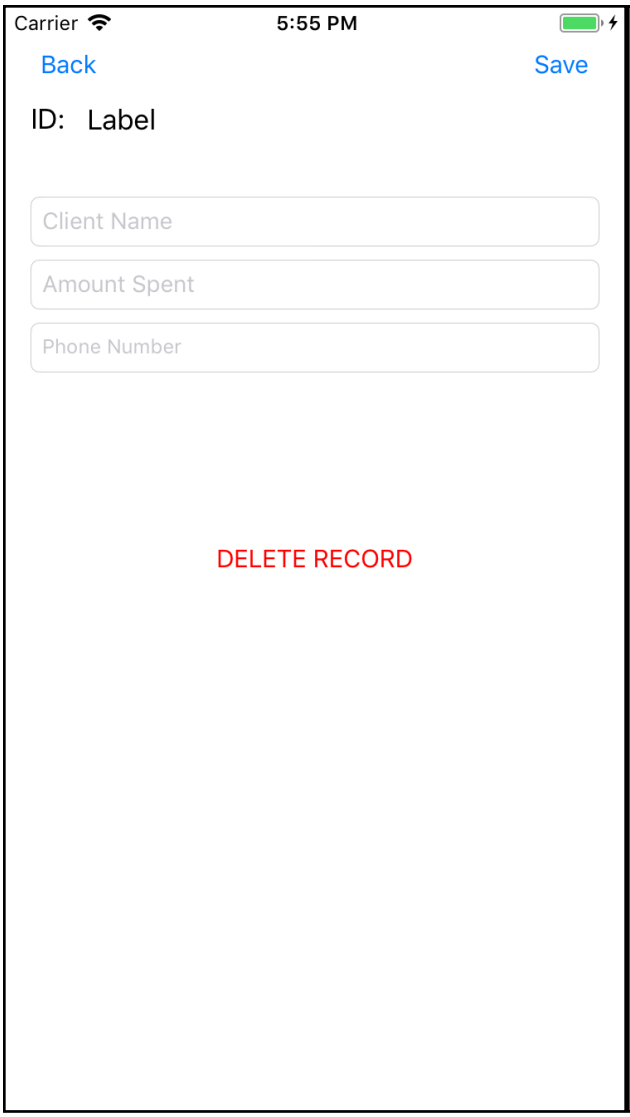
The SSSClient mobile app was built as a native iOS application. The application takes advantage of native frameworks ‘Alamofire’ for HTTP operations and ‘SwiftyJSON’ for JSON consumption. The application is a simple two-view solution that allows for both displaying and modification of data. Below are screenshots that highlight the application features. A video has also been included that demos the application in action.



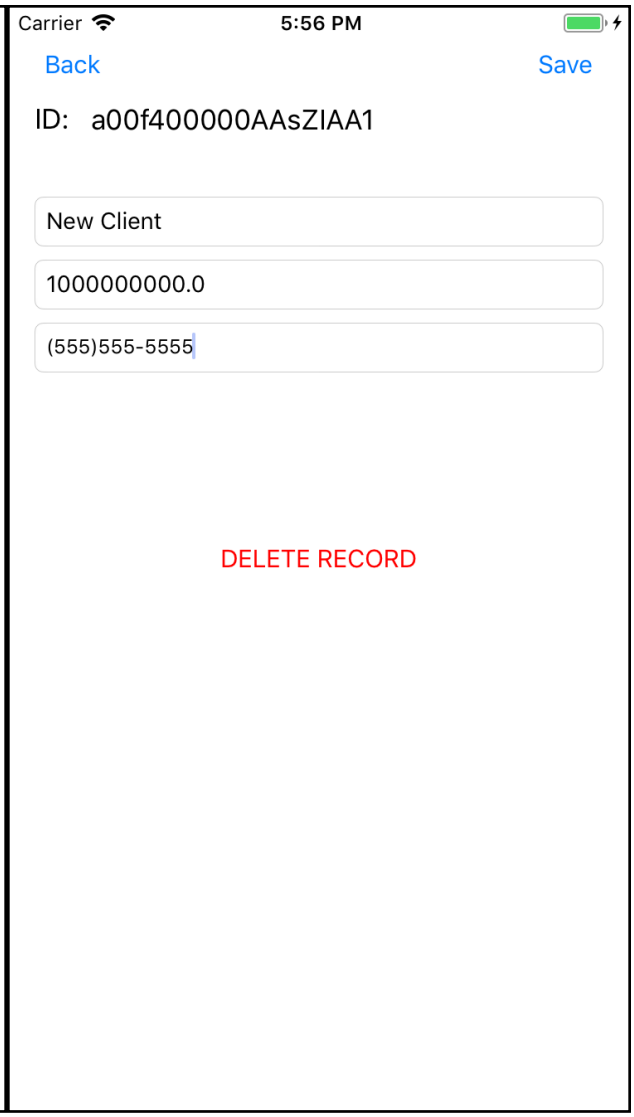

**View1: List of records**



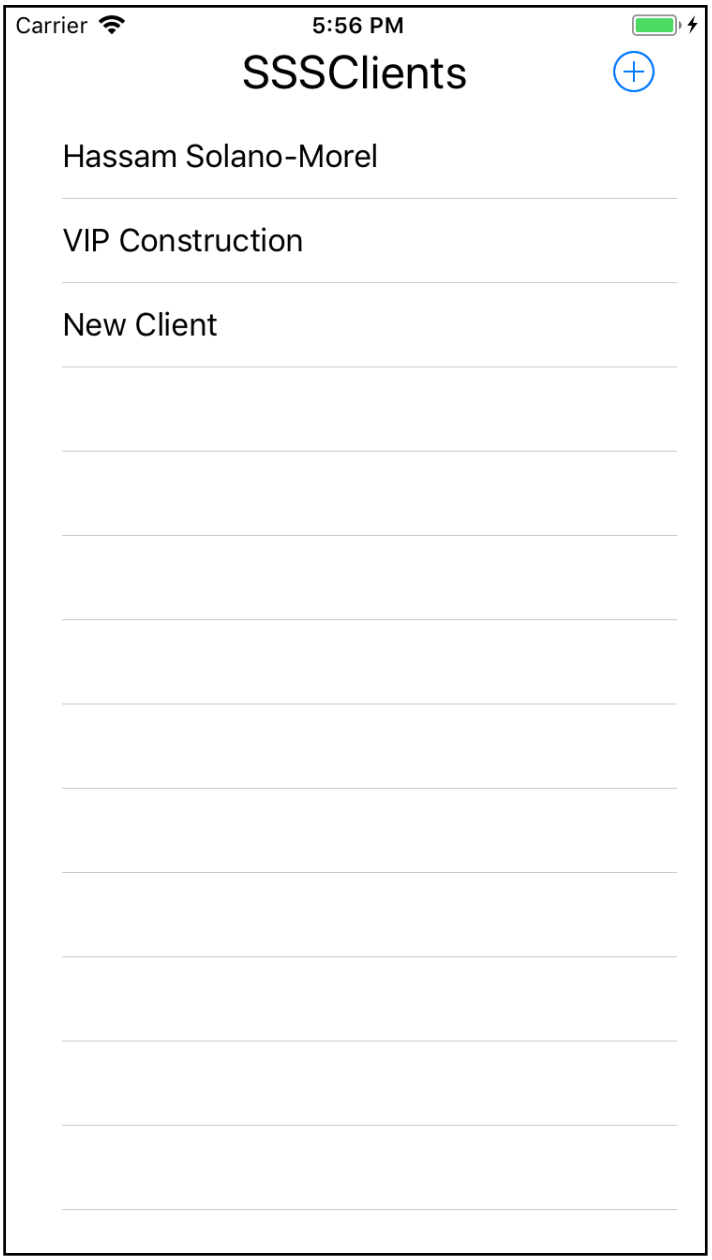

**View2 : Edit a record**




**View3 : Creating a new record**




**View4 : New record info saved**

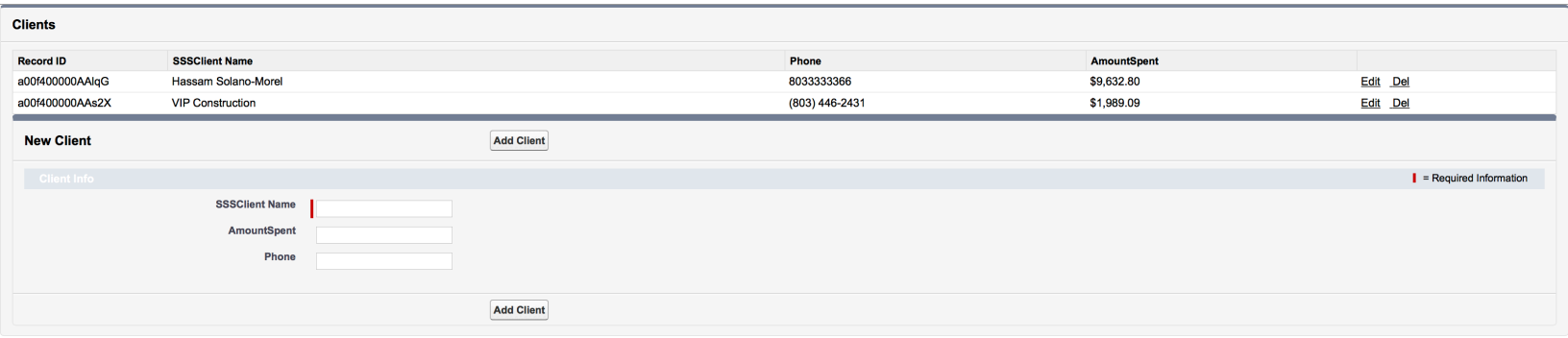



**View5 : List after new record**

# SAAS Service - (Salesforce)

The Force platform provided many of the services needed to accomplish many of the web-based requirements for this project. Firstly Salesforce (which is built on the Force platform) maintained the custom sObject used to represent clients (SSSClient__c). Secondly I was able to create a simple VisualForce webpage that displays records, allows for editing/deleting of existing record, and inserting of new records.

Records are displayed in a simple table with corresponding edit and delete button for each record. Below this table a simple form is presented to the user. This form allows for insertion of new records into the system. Below is an annotated screenshot of the webpage:

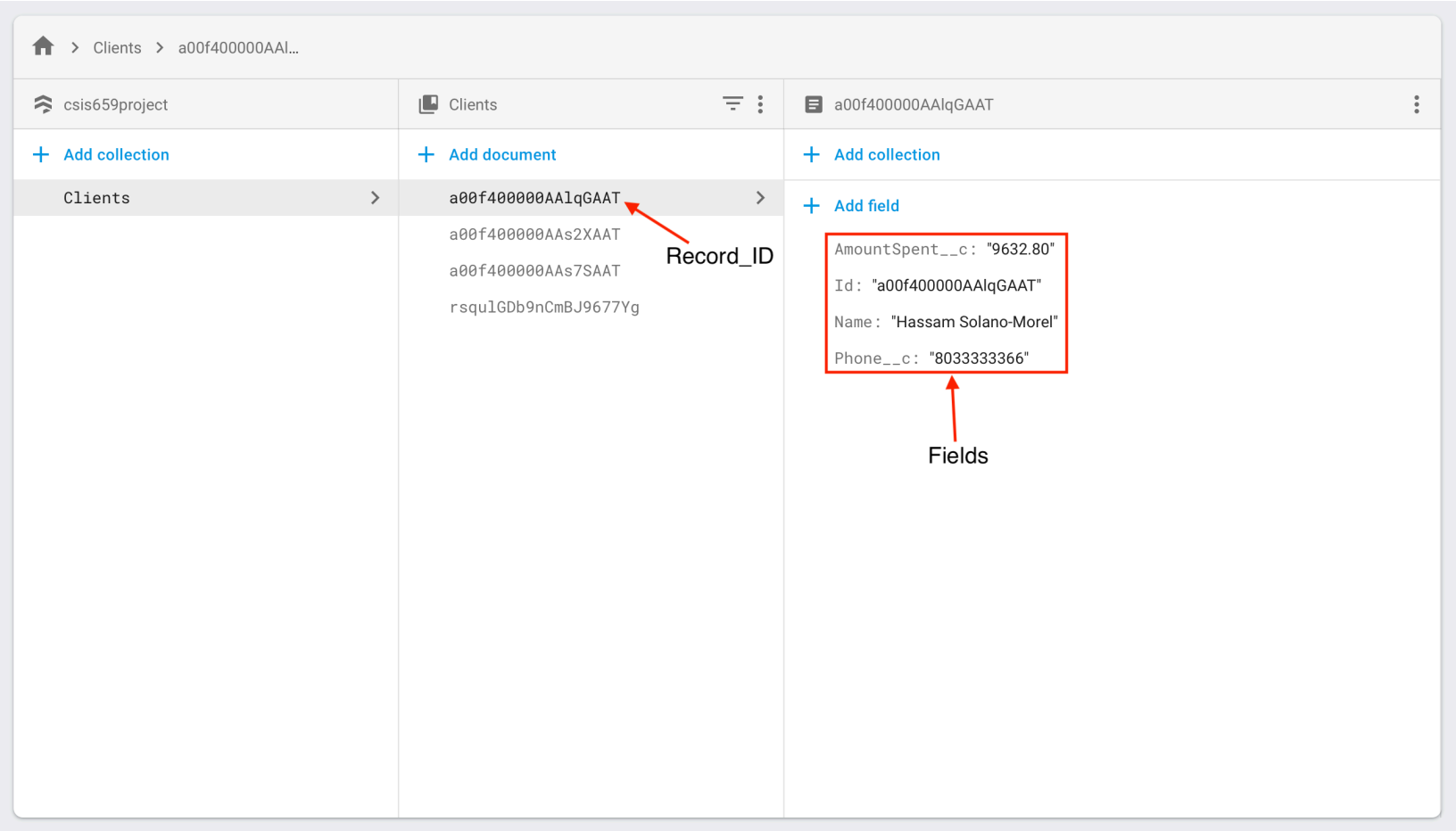


# Document Oriented Database - (Google Firestore)

For this project Firestore was used to satisfy the document oriented database requirement. In particular the service is used as a redundant backup to the original database maintained through Salesforce. The services keeps data as a series of “Collections” which encompass one to many “Documents”. Each Document can in turn contain several Collections and Fields. For the purpose of this project the database was structured as follows:

|root
———|Clients
———————|{RECORD_ID}
———————————|Name:String
———————————|AmountSpent__c:Double
———————————|Phone:String

Below is a screenshot of the database:




**Firestore Document Oriented Database**