Silent Partner Software Capstone

Specification Document - WIP

Goals

The goal, as outlined by Silent Partner Software, is to create a conversational interface to assist users in the non-for-profit space better manage their donors. Conversational interfaces are valued as future tech for their extreme dedication to simplicity and the belief that a conversation is a faster mode of conveying work than clicking through menus.

Modules & Features

Core features for MVP

1. Creating / Managing donors with minimum input
    a. Define Management:
    Management includes
        i. Add
        ii. Edit
        iii. Sort & Categorize
2. Post donations and print receipts (a whole single process)
    a. Enter Donations into a batch
    b. Post Batch
    c. Issue Receipts
3. Get and print reports
    a. E.g. Donors in Montreal
    b. Donations between two days
    c. Donations to a specific account
4. Optional - Additional Tables of Features
    a. Relationships between donors
    b. Additional Donor Categorization options

Built to grow and scale

- Architecture must scale and must be ready to accept new fields/keywords/actions etc
- Multiple customers, potentially millions of donor records
- 100’s of millions of donation entries

UI/UX Elements

- Conversational interactions
    - Suggestion based vs keyword detection
    - Guess of what the intent was based on input
    - Response feedback
- Dynamic display of data
    - Different displays based on next steps
    - Forms, pre filled or empty
    - Lists of data structured based on content
- Simplistic
    - Self explanatory
    - Little to no direction
    - Concise explanations
- Easy to navigate if need be
    - Form info follows text entered
    - Possibility to return to previous pages
    - Web browser... kind of... view Tech talk for more information.

Use Cases/User Stories and Requirements

Basic Use Cases

- Enter and Modify Donors
- Enter Donations
- Issue Receipts
- Run Reports (Search for Donors, Donations, and certain specific details)

All forms should contain only the minimum fields required.

For this System
Each User has an Organization they are a part of. Each donor they enter belongs to that Organization. If the user is a Supervisor, they can see the entire list of donors and all of their details. If they are some other limited access user type (Telemarketer), they will only be able to see some donor details. This other user type will only be made for testing, for now. Eventually user types with different privileges will exist.

Adding and Modifying a Donor

1	As a user, I want to be able to create a new donor
Story Points	
Priority	
Description	The user is able request to create a new donor 1) If the user is allowed to, the system responds properly by providing them a form with the minimum details needed, the user completes the form and the new donor is created. 2) Otherwise, the system returns an error.

The **minimum** details required:

- Automatically generated Donor Number
- Category (big donor, company, potentially big donor)
- Type of Donor (Individual, Corporation, Charity, Other)
- Do they want a Receipt annually or every time they donate?
- E-mail, Name, Phone Number, …

2	As a user, I want to be able to modify my donors
Story Points	
Priority	
Description	The user is able request to modify a donor, 1) If the user is allowed to, the system responds properly and provides them with a form prefilled with the minimum details for a donor and an option to delete. The user makes their changes and the system updates this donor. 2) Otherwise, the system returns an error.

The system should also be able to return specific details about a donor to the user upon request.

3	As a user, I want to be able to request details about a donor
Story Points	
Priority	
Description	The user is able request specific or general information about a donor, 1) If the user is allowed to and the requested donor exists, the system responds properly and provides them with a view of the donor's general details or the specific information requested. 2) Otherwise, the system returns an error.

Examples:

Input: "What language does Sam speak?"
Output: "Sam's primary language is Italian"

Input: "What is Sam's receipt preference?"
Output: "Sam likes to get annual receipts."

Input: "What are the notes on Sam?"
Output: "Here are the notes on Sam [notes on Sam]"

From Sam: *Users often try to pull up details on people from the cloud right before meeting with them. Lets them know if they've been thanked yet, what they speak, how much they've donated in the past, when the last time they donated was, etc.*

Input: "Show me Sam"
Output: "Here are Sam's details" + some view with all of Sam's details in it, like on the console

Other things to be able to ask about Donors

Donor's largest lifetime donation
Donor's most recent donation
Giving for a Donor

Give me Sam's donations

//heavy domain logic requests
Give top 10 donors…
Show me all of the donors who gave me a gift in the last year

Show me all of the givings from Sam in July

Managing Donations

The **minimum** details required:

- The Donor Number of the donor selected
- Assume first office//Select your Office (if more than one office exists for this user)
- Date of donation
- Motivation Code (why they made this donation, ex: general / newsletter)
- Designation Account (your bank account) //from user, related to a lookup table, my not be necessary
- Amount
- Is it a Charitable Gift?

4	As a user, I want to be able to enter a new donation
Story Points	8
Priority	
Description	The user is able request to create a new donation, possibly including the donor's name in the request 1) If the user is allowed to, if the donor name was not a part of the request, the system first requests the donor responsible for the donation. If the selected donor is valid, the system responds properly by providing them a form with the minimum details needed, the user completes the form and the new donation is created. 2) Otherwise, the system returns an error.

Input: "Create a new donation"
Output: "Okay, who is this donation coming from?"
Input: "It's from Bradley"
Output: "Alright, now give me the other donation details" + donation form

Input: "Create a new donation from Bradley" || "Bradley made a new donation" || ...
Output: "Alright, now give me the other donation details" + donation form

Reports

These are not the reports found in the console. By reports, we mean special queries on the database. Each report may take different parameters and act within different contexts. This will be some heavy logic, maybe only a few should be built for the MVP.

Examples:

“What is Sam’s largest lifetime donation?”
“What is Sam’s most recent donation?”
“What is Sam’s giving?”
“What is Sam’s giving from October until December?”
“Give me all of Sam’s donations”
“Give me my top 10 donors”
“Show me all of the donors who gave me a gift in the last year”
“Show me all of the givings from Sam in July”
“What language does Sam speak?”
“Has Sam been thanked for his most recent donation?”
“What type of donor is Sam?”
“Show me the givings in December for all Corporation Donors”

Each report takes some parameters, most require starting dates.

5	As a user, I want to be able to request reports
Story Points	HUGE
Priority	
Description	The user is able request to generate some sort of report by asking a question or asking for a view 1) If the user is allowed to, and if the request is valid, the system processes and generates the report 2) Otherwise, the system returns an error.

This is where that huge Natural Language to SQL goes.

Receipts

There a couple of templates for receipts. Each Donor has a receipt preference which is set when they are entered, this preference includes how they want to receive the receipt (by mail or electronically and in which template, depending probably on what type of donor they are).

Receipts also need to include gifts somehow. Not really clear on the specifics of this right now, but the idea is that it's pretty much just a big function we have to implement to generate data to throw into some templates.

6	As a user, I want to be able to issue receipts, potentially including a donor's name
Story Points	
Priority	
Description	The user is able request to issue a receipt to a donor 1) If the user is allowed to, and if the donor's name was not specified, the system first asks for which donor a receipt should be issued. If the requested donor exists, then the system generates a receipt and returns it to the user. 2) Otherwise, the system returns an error.

Examples:

Input: "Make a receipt"
Output: "For which donor?"
Input: "For Sam"
Output: Sam's receipt based on his receipt preferences as a donor

Input: "Make a receipt for Sam"
Output: Sam's receipt based on his receipt preferences as a donor

Login

7	As a user, I want to be able to login with my credentials
Story Points	3

Priority	
Description	The user is able request to issue a receipt to a donor 1) The user logs into the website. 2) Otherwise, the system denies entry.

Tech Talk

The backend will be a simple **.NET/C# web** stack, using **AWS** for server hosting. The frontend website will be coded in **XXXX using YYY libraries and ZZZ frameworks**.

The Website

The frontend will serve to send Natural Language requests to the server via a simple text box *(think chat box)* and to display the results crunched by the server be it data, graphs, or forms for further, more-detailed input.

The Backend

TODO:

- How to make more specific SQL style commands? “Donors *over $200*”
- DB schemas
- Diagrams, etc

The backend will host **C# server-side** code on an **AWS server**. C# makes the most sense because of the team’s and SPS’s familiarity, the fact that C# is OO, and that C#/MVC/.NET has a very strong support community + documentation.

Dispatcher

The main module, or ***Dispatcher***, will be responsible for relaying NL requests to API.ai and back; executing custom SPS code when necessary; and displaying the results back to the frontend website. Its architecture is an important matter as it must be built to scale as keywords, actions, and code to execute expands.

It will implement the **API.ai** API to manage the chatbot functionality taken from the website’s user input. API.ai makes sense as the chatbot choice because it is the company that seems to take conversational UX most seriously in this space. Additionally, its pricing plans and industry support make it reasonable to assume they are aiming for longevity (which is what SPS hopes). Very good documentation, too.

The API.ai C# SDK will be installed to the project via NuGet and handled in tandem from the Dashboard.

The server will implement a **Command Design Pattern** for the execution of SPS code to API.ai action mappings. From the returned JSON, the *action* keyword will be parsed and turned into a command from the codebase to be executed. *Parameters* will be sent as a simple Dictionary<string, Object> object.

Scaling and DB module

As this project is a very simple MVP of the basic functionality, building the system to scale in both usage and functionality is extremely important.

A module should be built to automatically poll certain important keywords to be recognized by the API.ai Entity recognizer. For example, some last names may be too uncommon for it to naturally be recognized by the Machine Learning and must be explicitly taught. As API.ai is still in beta for its more ambitious features, some programmatic compromises must be made on our end until they beef up certain features.

Entity - DB harmony
Done with CRON jobs or on clients connecting to the website, the Entity builder module should routinely poll from the DB to update and teach the AI new terms if any.

DB SCHEMA Tables:

- Donor,
- Donations,
- Batches (collection of donations)
- Organizations
- Receipts
- Users

View Builder

Most if not all commands executed from the SPS codebase will generate a view to be displayed on the website. The 3 parts are the View/Form, the DB entry and the Model class, as it would in an MVC stucture. Potentially using the Row Metadata taken from the database table, an engine can be create to automatically map objects to fields in an dynamically building form engine. That way, any following command created in the future would not require new forms to be built, saving a lot of time to the developers.

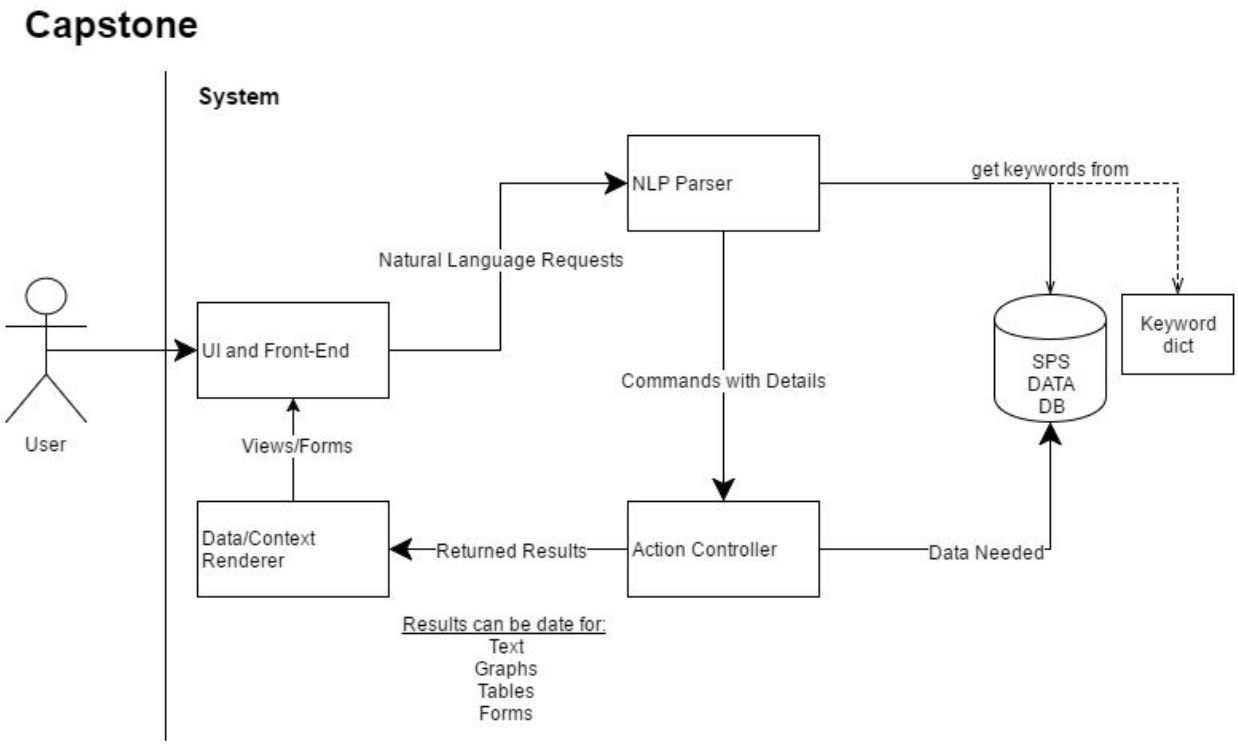
The Chatbot

Using **API.ai**, we will use their dashboard to build and strengthen the bot and its state-machine. Since machine **learning** is very important for bots like these, input examples need to be fed into the machine. This can be done programmatically, as well.

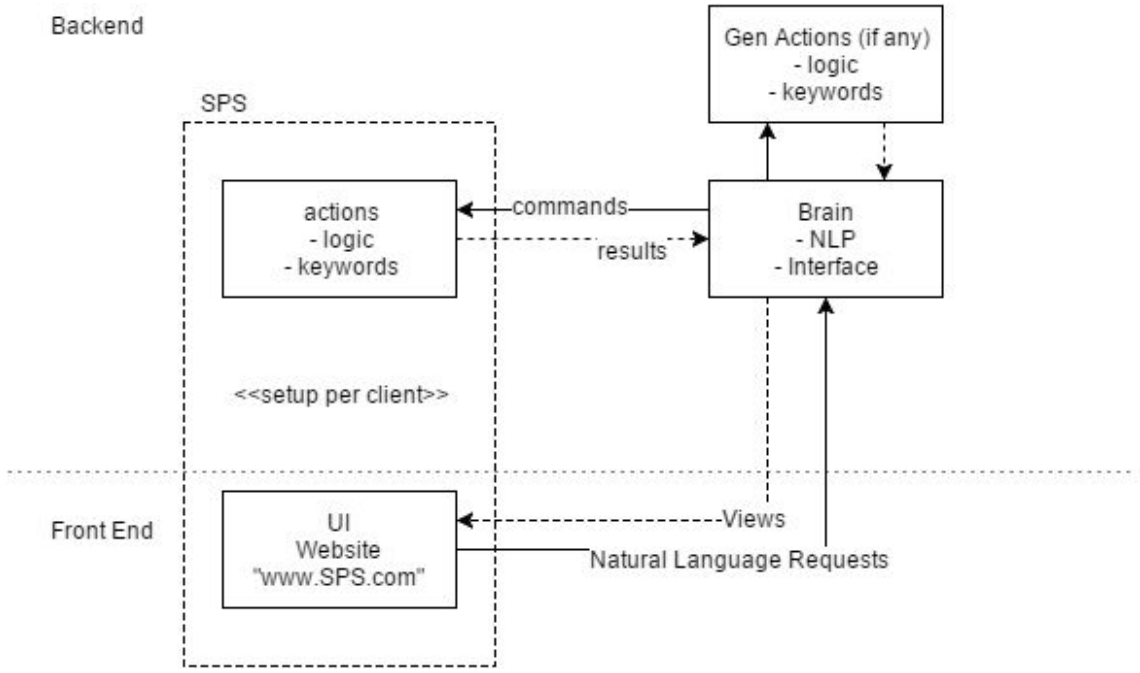
> **On chatbots**
> Chatbots consist of 3 elements: Natural Language Processing, Machine Learning, and sophisticated State Machine Navigation.
>
> *Natural Language processing* is the engine responsible for extracting keywords and "Intents" from a **set list of keywords and sentence structures**.
> *Machine Learning* improves this process by gradually building and growing those lists from positive and negative experiences.
> Finally, the *state machine navigation* wraps up these two engines and provides developers the ability to guide users through graphed "stories" entirely through Natural Language with programmer-friendly outputs in JSON format.
>
> **Ref: API.ai documentation - Key Concepts**
> **Agents** correspond to applications. Once you train and test an agent, you can integrate it with your app or device.
>
> **Entities** represent concepts that are often specific to a domain as a way of mapping natural language phrases to canonical phrases that capture their meaning.
>
> **Intents** represent a mapping between what a user says and what action should be taken by your software.
>
> **Actions** correspond to the steps your application will take when specific intents are triggered by user inputs. An action may have parameters for specifying detailed information about it.
>
> **Contexts** are strings that represent the current context of the user expression. This is useful for differentiating phrases which might be vague and have different meaning depending on what was spoken previously.

For more on API.ai*, refer to the* API.ai *documentation and the Usage Manual in* Help/Manuals > Using API.ai

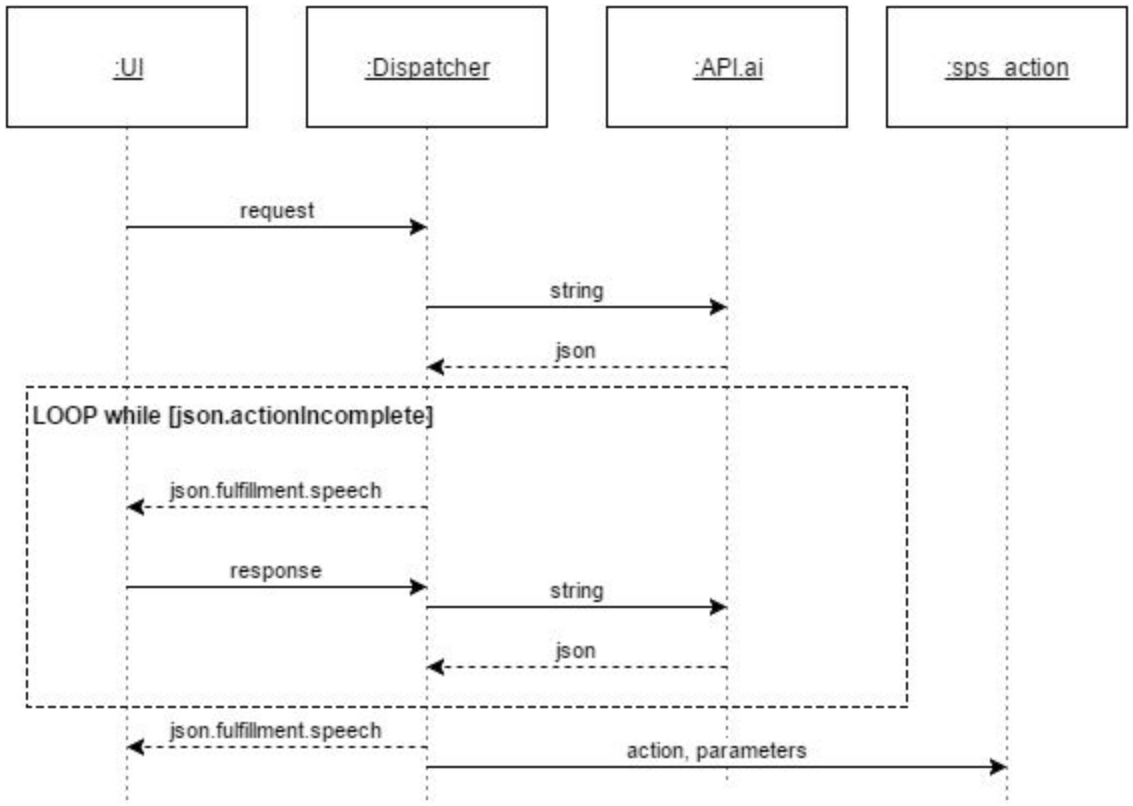
Diagrams and Data



Layered architecture



Conversation with API.ai to be dispatched as commands for the SPS code to execute



Donor Data:

- TITLE
- Last NAME
- First NAME
- DONORNO
- ADDR1
- ADDR2
- CITY
- PROV
- ZIP
- COUNTRY
- PHONE
- EMAIL
- DONor TYPE
- Anonymous

Meta _data for Donors

- Create Date
- LASTCHANGE
- TIMESTAMP

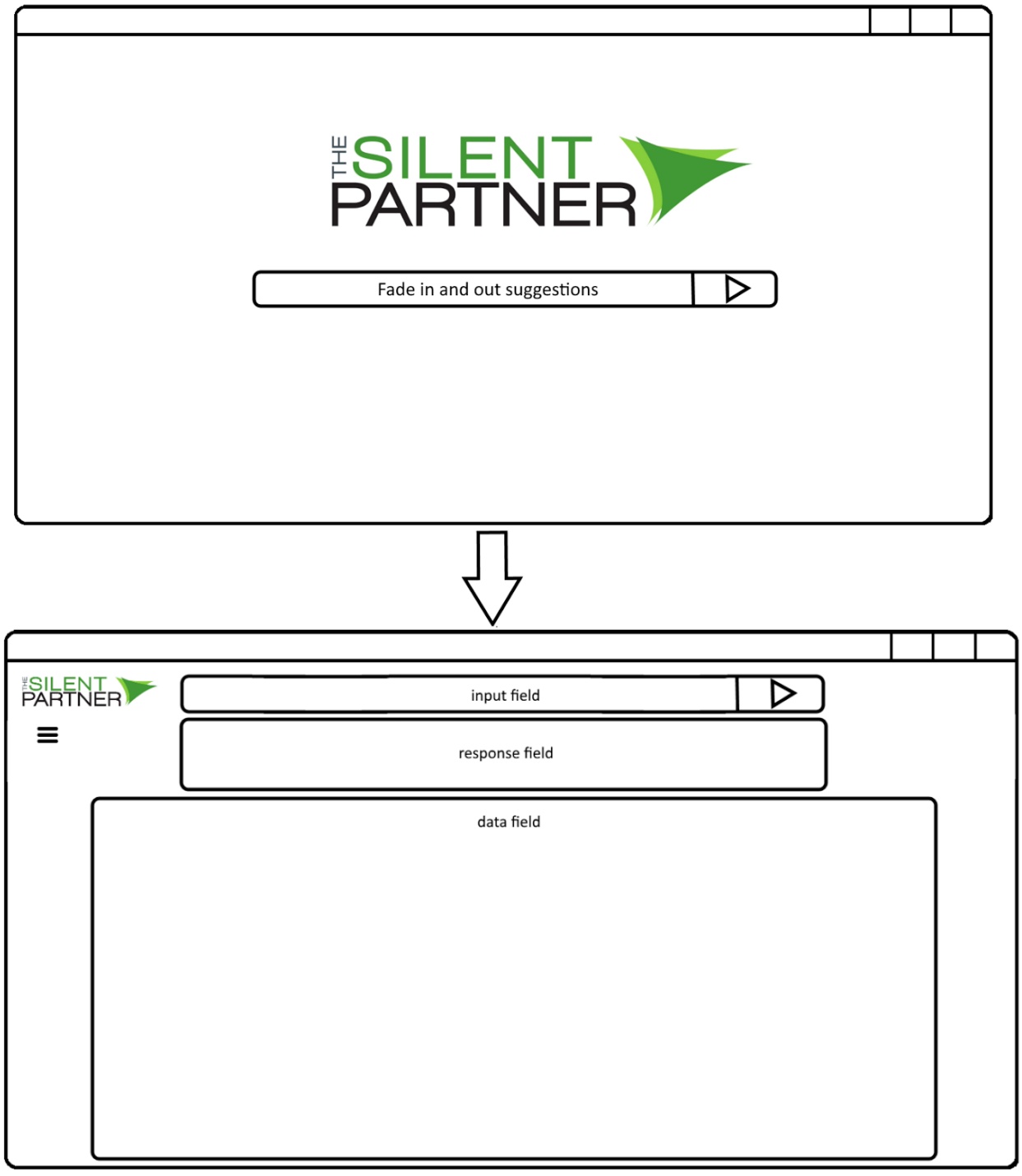
Data for Donations:

- DONORNO
- AMT
- ACCount
- MOTIVation Code
- Donations DATE
- Payment TYPE
- CURRENCY
- Donation Type
- Unique donation ID
- notes
- Batch Number
- anonymous

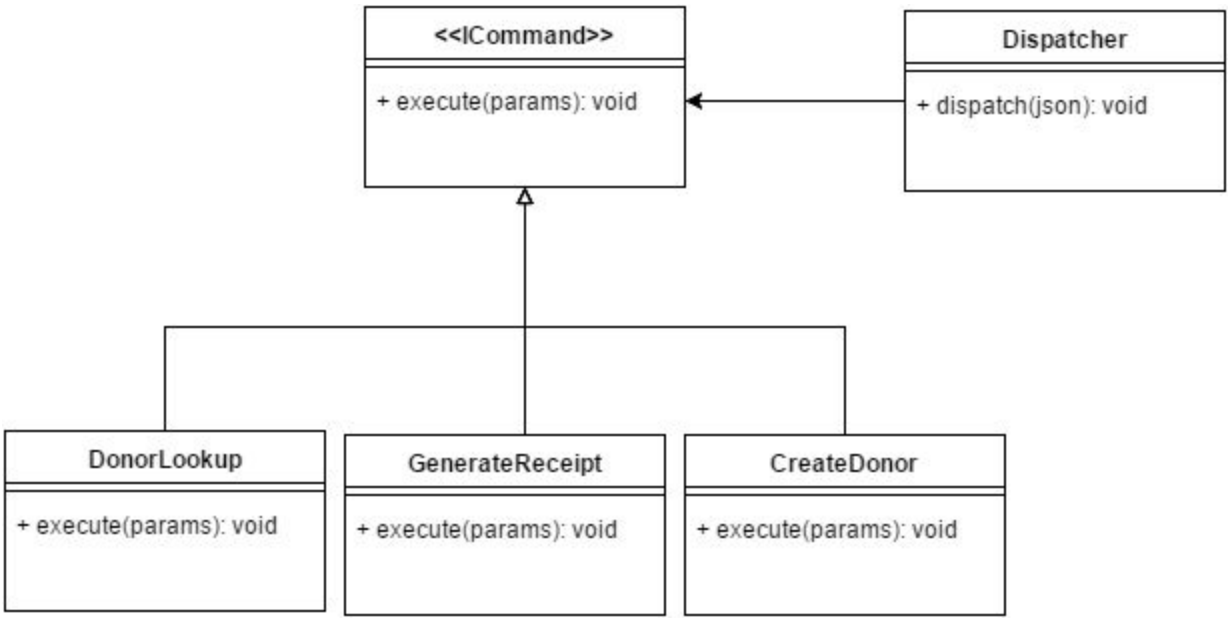
Meta_data for Donations

- USERNAME
- Post Date

UI Mockups


THE SILENT PARTNER
Fade in and out suggestions
THE SILENT PARTNER
input field
response field
data field

Dispatcher to SPS codebase (sample)

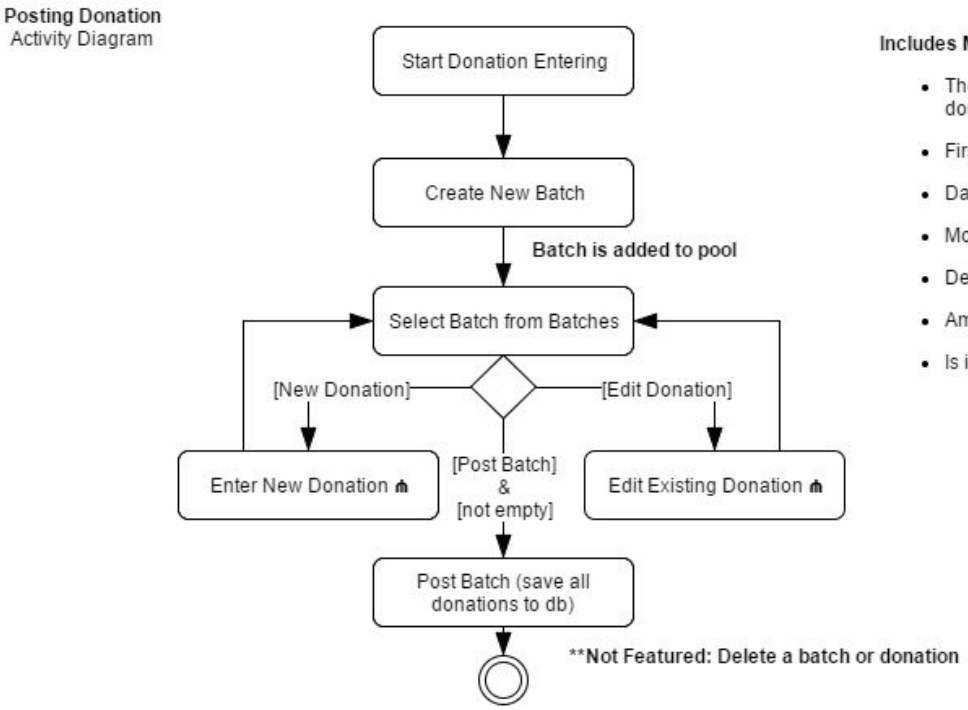


Code snippet example for Dispatcher.dispatch(json)

Note: Not included: On_Result and On_Error event handlers can be added to simplify the two possible scenarios when interacting with the API. Also, execute will accept the parameters from the returned JSON.

```
// Get Client's Project Name
string project = System.Reflection.Assembly.GetExecutingAssembly().GetName().Name;

// Taken from API.ai returned JSON object
string action = "HelloCommand";

//Join strings to create classLocation
string classLocation = project + "." + action;

//Find the class as a Type
Type commandType = Type.GetType(classLocation);

//Check if null, if null return message to the UI
if (commandType == null) return;

//Cast and execute the command
ICommand command = Activator.CreateInstance(commandType) as ICommand;

try
{
    command.execute();
}
catch (Exception e)
{
    // Send Error message to the UI
    return;
}
```

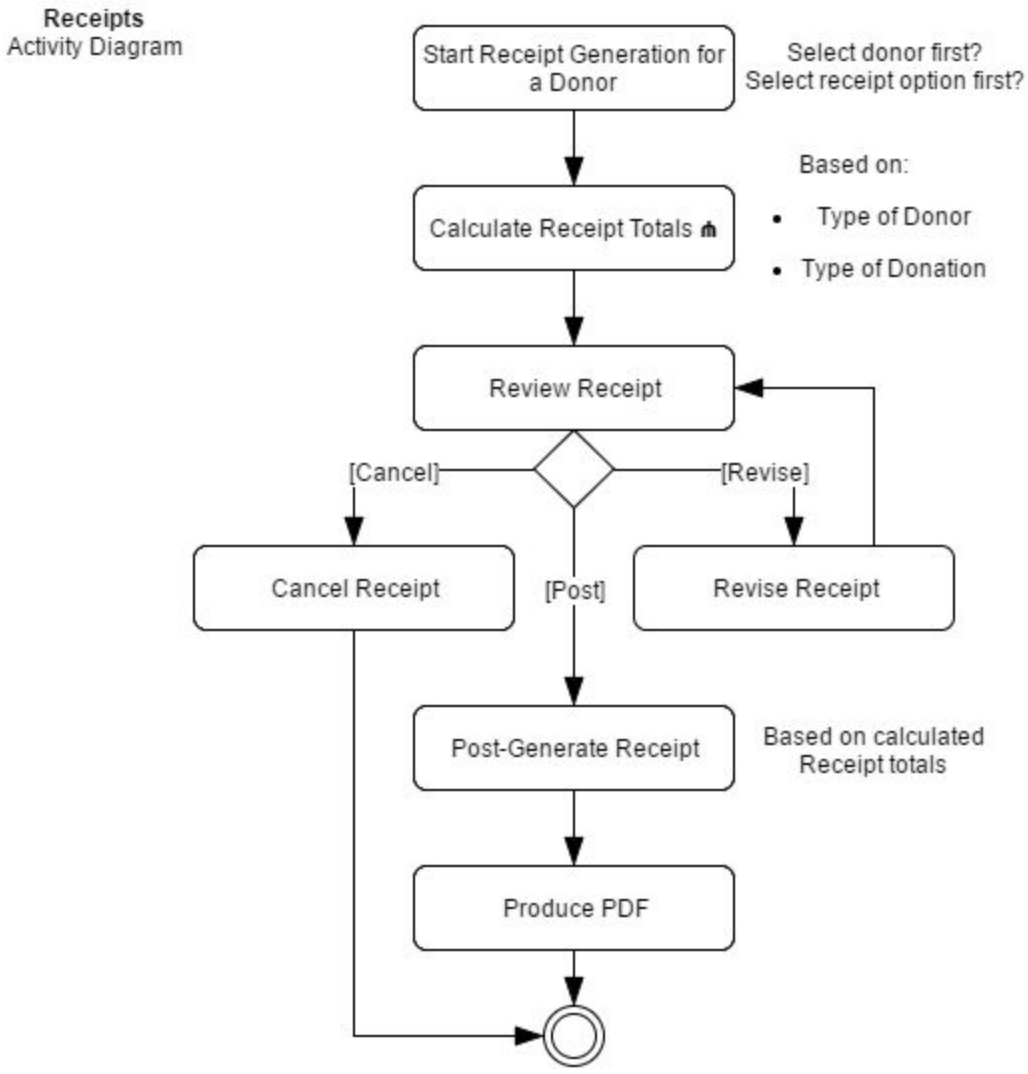
Posting a Donation



Includes Minimum Fields:

- The Donor Number of the donor selected
- First Office
- Date of donation
- Motivation Code
- Designation Account
- Amount
- Is it a Charitable Gift?

Generating Receipts



Generating Reports