DMS Lite Release Summary

Team members

Name and Student id	GitHub id	Number of story points that member was an **author** on.
Dmitry Svoiski (26893570)	dsvoid	18
Jacob Desrochers (26919529)	Jabobstev	10
Dionysios Kefallinos (27019920)	LambHoot	10
Tyler Ramsay (26948065)	starvator	13
Philippe Miriello (27031246)	pmiri	15
Michael Bilinsky (26992358)	Netopya	15
Mathieu Beauchemin (26760953)	Disva (also *disva)	15

Project summary

A conversational interface to assist users in the nonprofit space better manage their donors and donations. It must be able to create and manage donors, post donations, generate and view reports, as well and produce receipts for donors. In addition to these core features the application must be built in a way that supports easy scalability and robust extensibility for future development.

A working copy of Release 1 can be accessed here with the following credentials:

U: q@q.com
P: P@ssword123

Velocity

Total: 4 stories, 18 points over 7 weeks
Iteration 1 (1 story, 3 points, 2 issues)

Login : 3 Story Points
Included various setup activities. Key steps included linking the database to the application, and forming the database schema, as well as establishing hosting.

Iteration 2 (1 story, 5 points, 3 issues)

View Donors: 5 Story Points
Integration of API.ai into the system. Added the view donor command to display a specific donor that is pulled from information in the database.

Iteration 3, Release 1 (2 stories, 10 points, 11 issues)

Create Donor: 5 Story Points
Implemented Create Donor command and taught API.ai to recognize such commands in natural language. Upon request to create a donor, a creation form is displayed allowing the user to input all necessary information to be persisted to the database.

Modify Donor: 5 Story Points
Implementation of the Modify Donor command allows for users to recall any donor by name, number, or email address in natural language to further edit the corresponding donor's details.

Current Velocity: 7 Story Points / Iteration

Plan up to next release

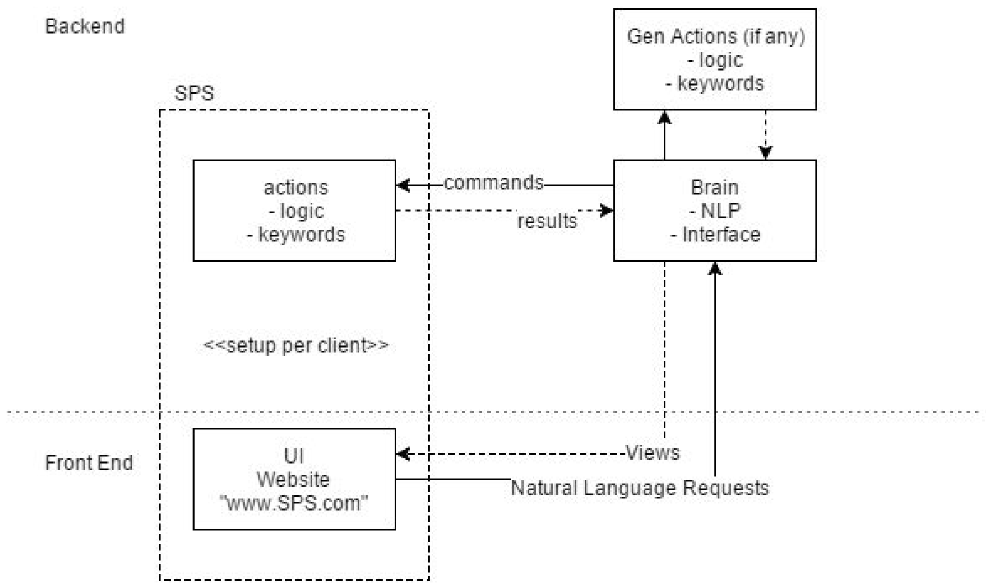
Total: 9 stories, 57 points, over 6 weeks
Iteration 4 (3 stories, 23 points)
Iteration 5 (4 stories, 21 points)
Iteration 6, Release 2 (2 stories, 13 points)

The next iterations seem to have much higher story point values which puts them outside of our velocity. However, this is due to our original iterations not work done with regards to the program's framework, architecture and planning. Now that these elements are already established, much more time can be spent on tasks that actually count towards story points. In terms of actual effort spent, we believe that these goals are feasible to us.

Overall Architecture and Class diagram

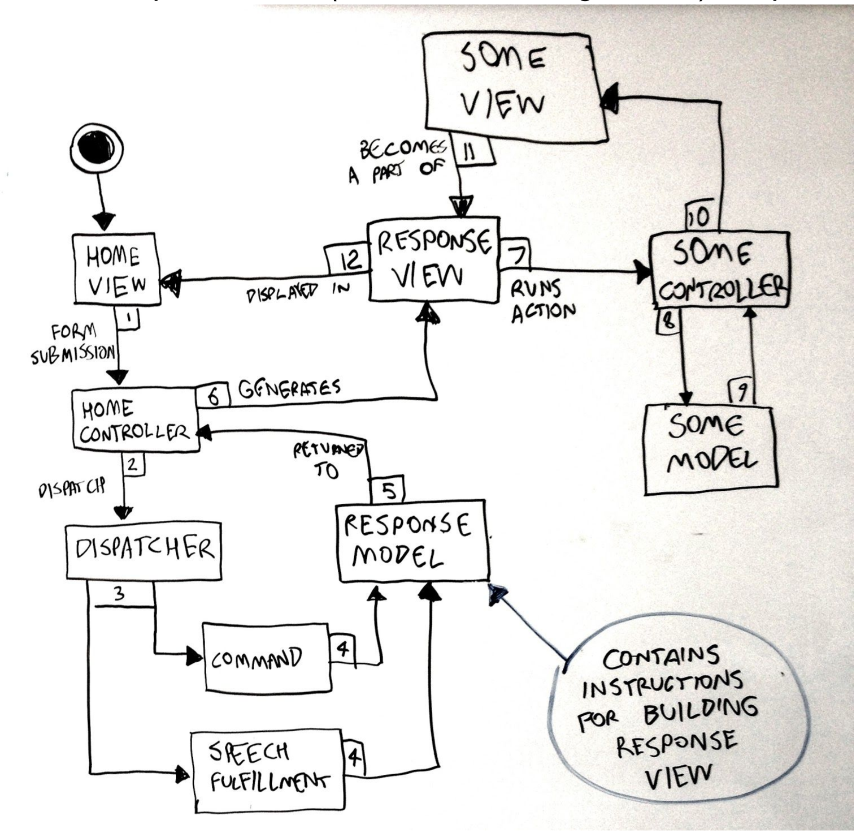
Earlier specifications found linked to Issue #1 on GitHub

Simplified Abstract Architecture

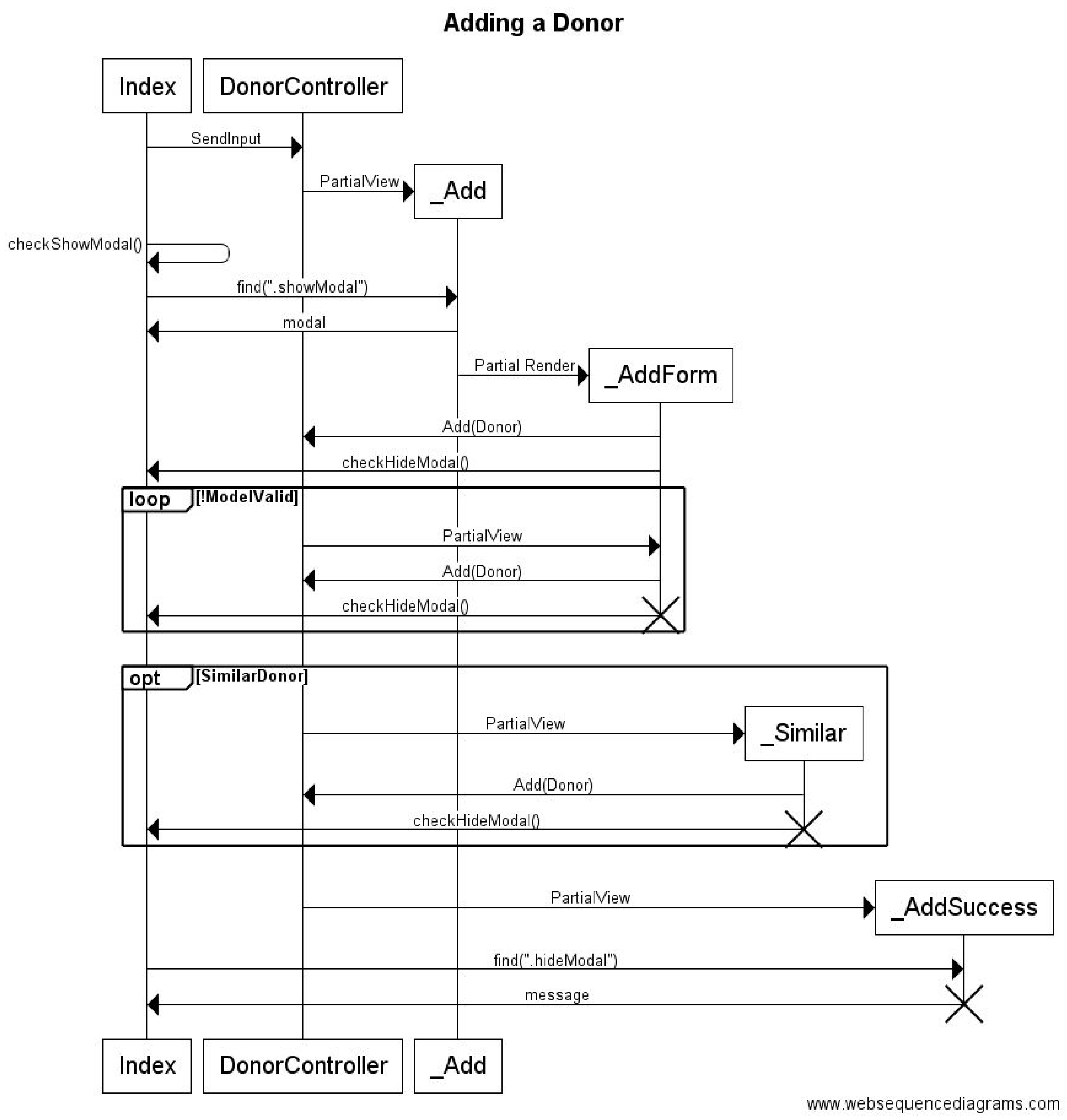


Process by which a user request generates the appropriate view(s)

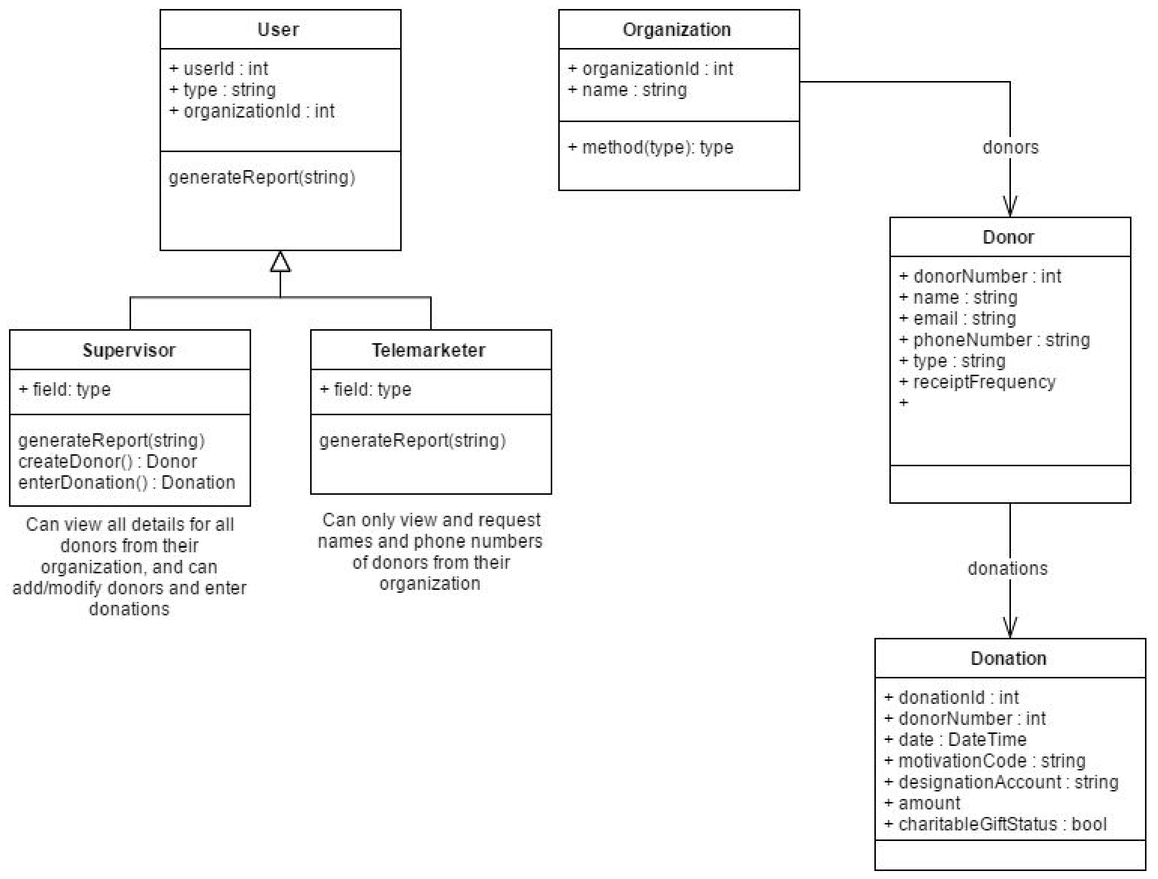
Refer to [Issue 18](#) for an explanation of (and the reasoning behind) the process below.



Action Diagram showing the Create Donor story



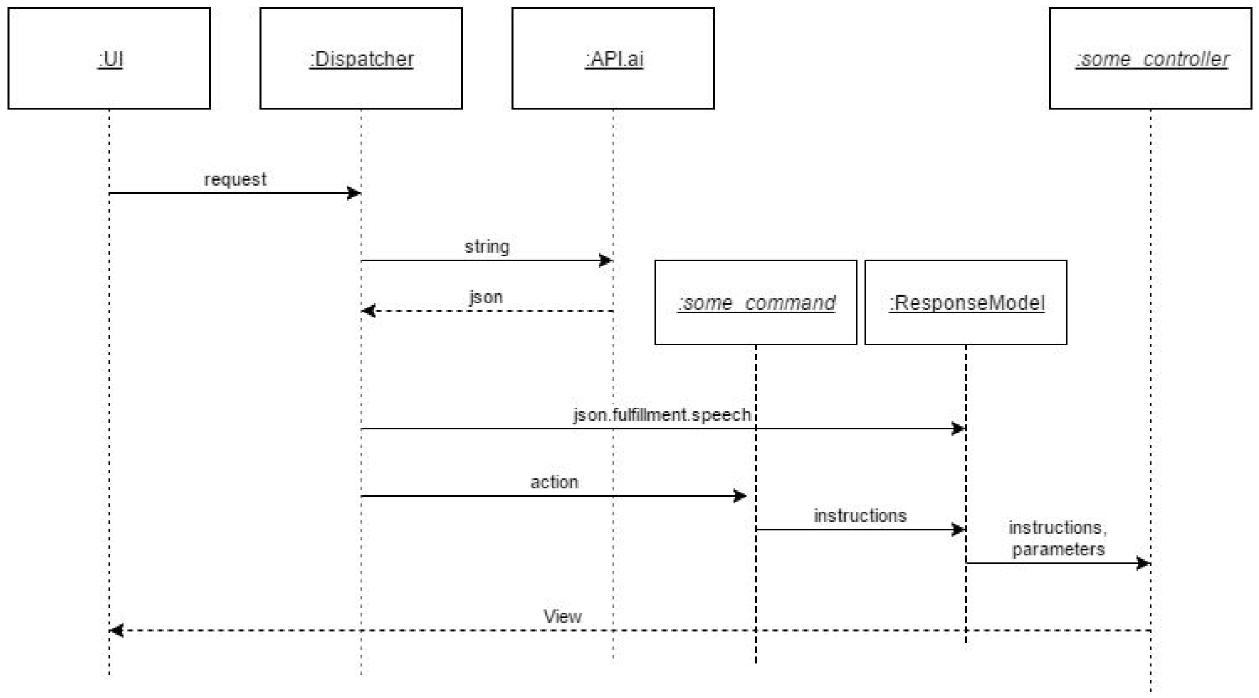
Class Diagram



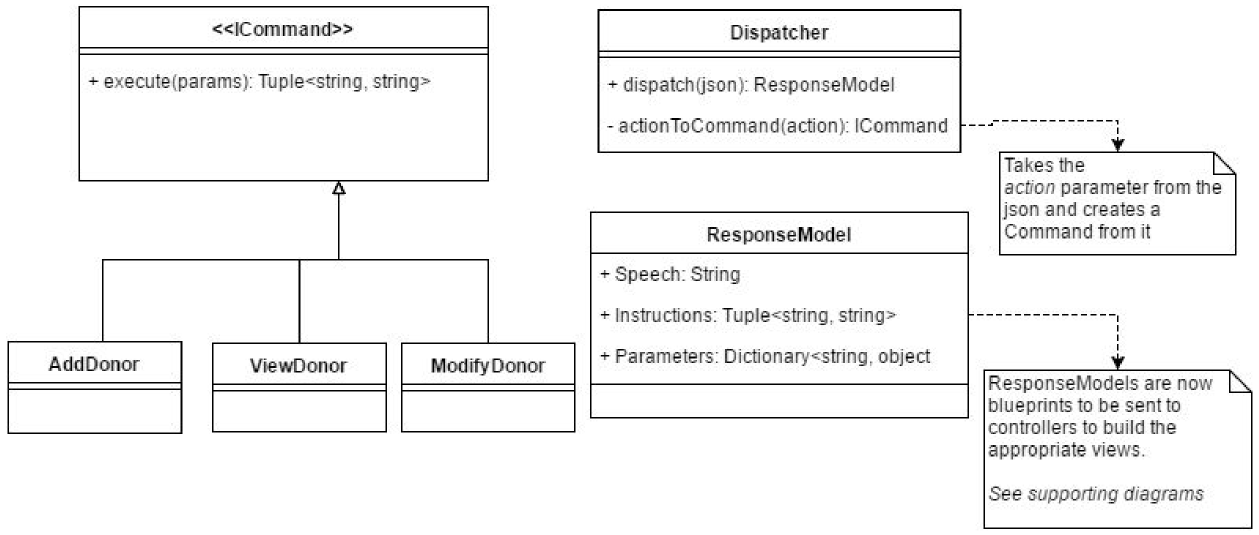
The Dispatcher

The dispatcher is the main component that handles natural languages requests, turns them into commands, and executes the appropriate command within the codebase.

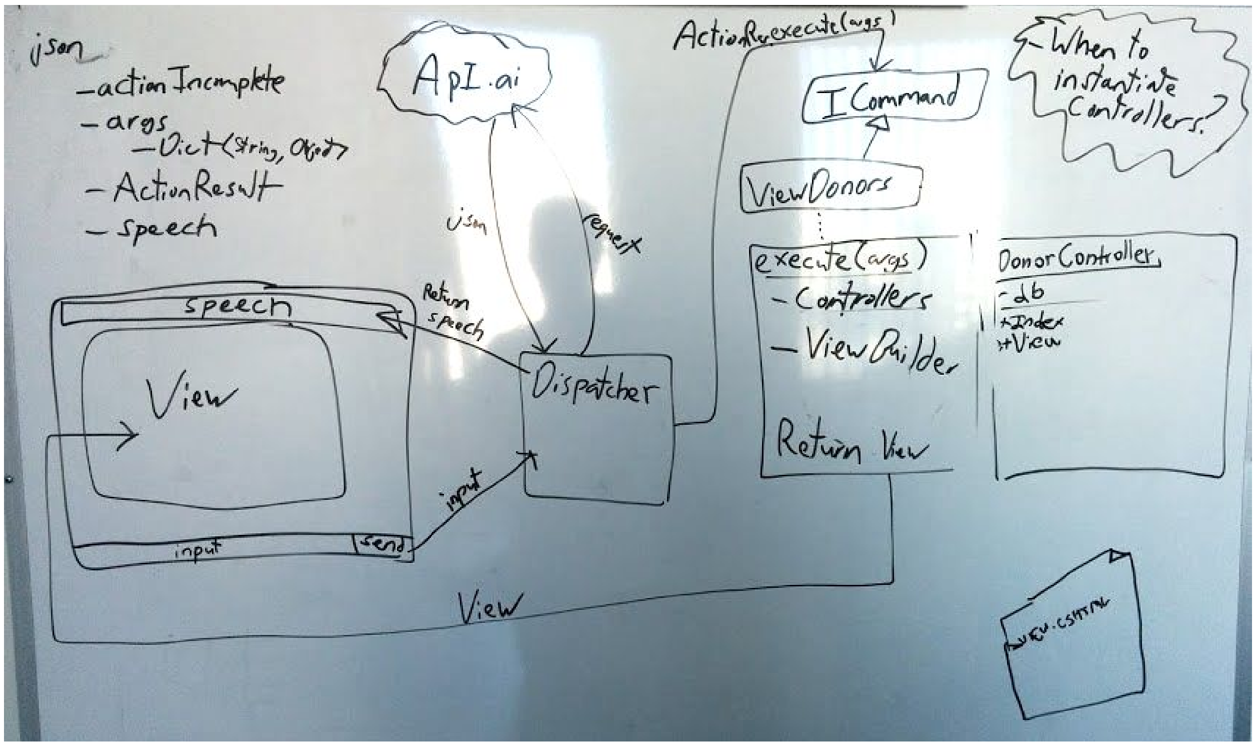
Sequence Diagram



Current Command Pattern Implementation

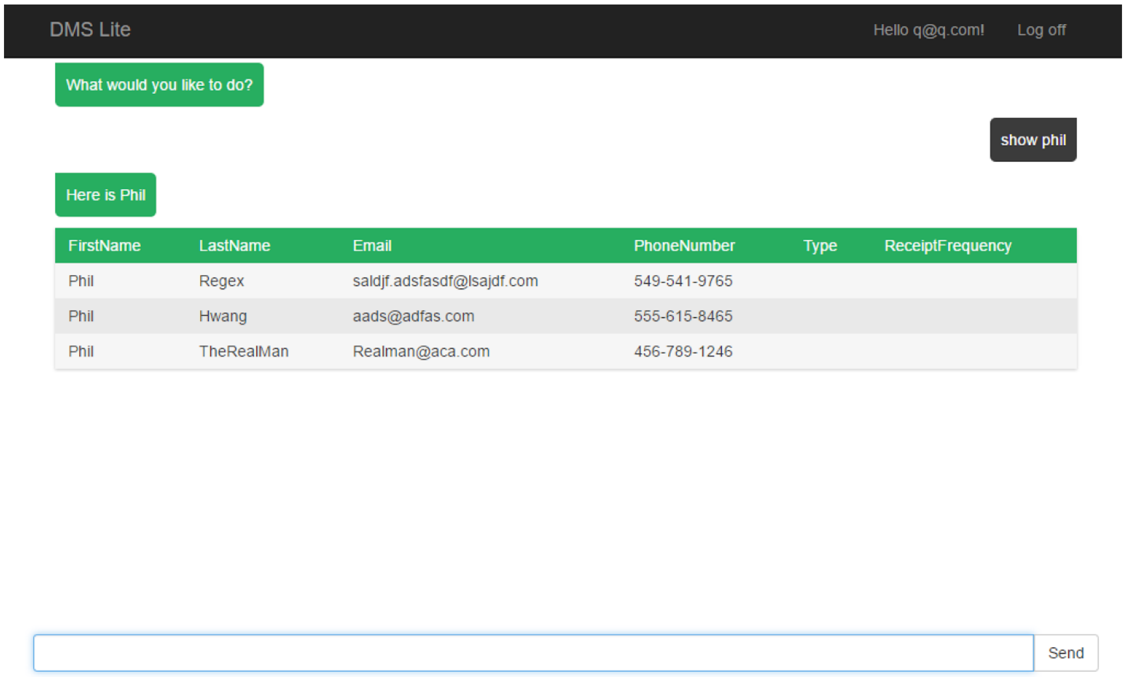

<<ICommand>>
+ execute(params): Tuple<string, string>
AddDonor
ViewDonor
ModifyDonor
Dispatcher
+ dispatch(json): ResponseModel
- actionToCommand(action): ICommand
Takes the *action* parameter from the json and creates a Command from it
ResponseModel
+ Speech: String
+ Instructions: Tuple<string, string>
+ Parameters: Dictionary<string, object
ResponseModels are now blueprints to be sent to controllers to build the appropriate views.
See supporting diagrams

Full View of Communication between modules



Hi-Fidelity Mockup of Application

Earlier Mockups can be found on the GitHub linked to Issue #2


DMS Lite

Hello q@q.com! Log off

What would you like to do?

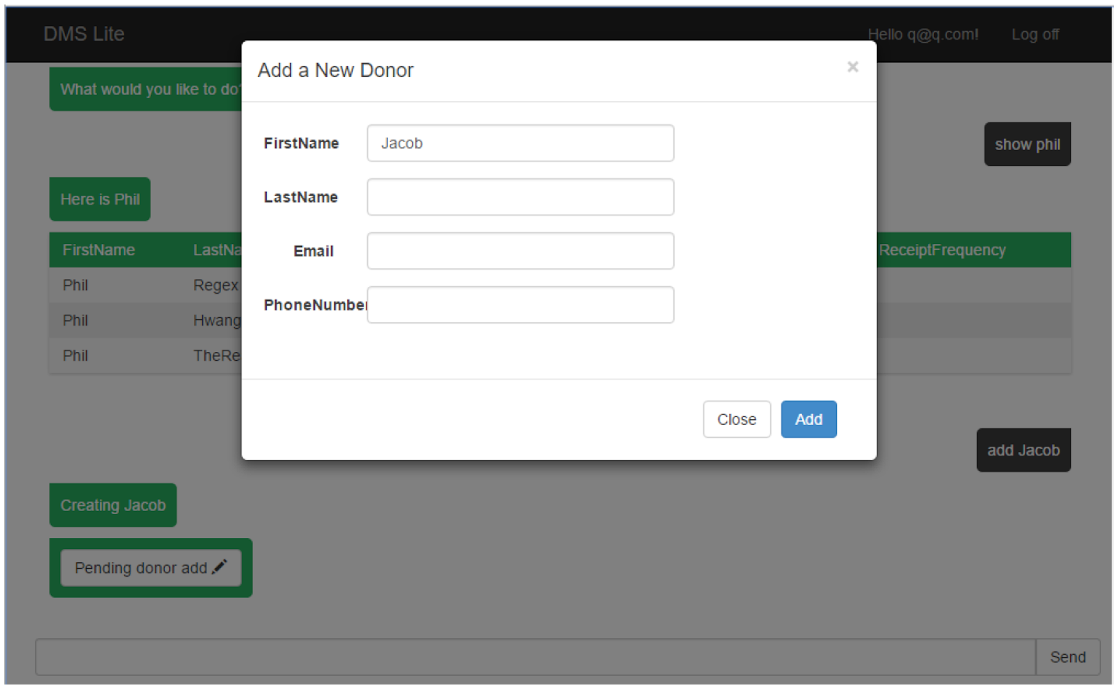
show phil

Here is Phil

FirstName	LastName	Email	PhoneNumber	Type	ReceiptFrequency
Phil	Regex	saldjf.adsfasdf@lsajdf.com	549-541-9765		
Phil	Hwang	aads@adfas.com	555-615-8465		
Phil	TheRealMan	Realman@aca.com	456-789-1246		

Send

Requesting a donor in DMSlite.


DMS Lite

Hello q@q.com! Log off

What would you like to do

show phil

Here is Phil

Add a New Donor

FirstName: Jacob

LastName:

Email:

PhoneNumber:

Close Add

add Jacob

Creating Jacob

Pending donor add

Send

Adding a new Donor displays a creation form.

Show us the layers in your system and your domain classes. You can also include individual class diagrams in your stories on GitHub

Infrastructure

API.ai

API.ai is the sdk used to simplify the natural language processing element of the DMS lite program. Users enter commands into the input field and it is sent to API.ai be interpreted and turned into a command.

Wit.ai was considered initially but was rejected due to its lack of proper financing plan and its acquisition by Facebook, which raised concerns regarding its stability for our stakeholder. Another considered alternative was Mycroft's Adapt parser. It was rejected due to its infancy and lack of community support.

Entity Framework

https://msdn.microsoft.com/en-us/data/ef.aspx

Entity Framework is a Microsoft-supported object-relational mapper which eases development of programs with relational databases. It is capable of creating MySQL tables based on data classes developed in Visual Studio, and vice versa: connecting to a MySQL database lets Entity Framework generate data classes with their relationships intact.

Entity Framework was chosen due to the team's existing experience with it: the most likely alternative, NHibernate, would have required more time to learn.

MVC

https://www.asp.net/mvc

The model-view-controller web development pattern is commonplace and well-documented in ASP.NET and C# projects. Models represent data from the project's database as objects. Controllers contain functions necessary for the manipulation and display of those models. Views handle the display of output generated by controllers.

This framework and pattern were selected based on the stakeholder's requirements. The stakeholder ultimately wants his team of developers to be able to work on and maintain the system, and this is their most familiar environment.

Name Conventions

Our code follows the following C# coding and naming convention standard. In addition, we use a custom domain-specific naming convention.

Code: 5 key files

DMSLite/Controllers/HomeController.cs
Send input to Dispatcher, run the command the Dispatcher returns in a _Response view. (stub)

DMSLite/Commands/Dispatcher.cs
Get input from HomeController, process it with the API.ai engine, send a command for the HomeController to follow.

DMSLite/Views/Home/_Response.cshtml
Display the output of a command followed by the HomeController along with a Natural Language response from the API.ai engine. (stub)

DMSLite/Controllers/DonorsController.cs
Run CRUD operations for donors.

DMSLite/Views/Home/Index.cshtml
Displays the user interface.

Testing and Continuous Integration

All unit tests related to this release are found in DonorsControllerTest.cs. This test file includes all unit tests related to Donors and tests the functionality of the DonorController's methods. The 5 tests below can all be found in this file.

List the **5** most important tests

Test File path with clickable GitHub link	What is it testing (1 line description)
TestViewSpecificDonor	Test that requesting a specific valid donor returns that valid donor.
TestViewInvalidDonorAndParameter	Tests that searching for a donor with invalid criteria returns an error.
TestAddNewValidDonor	Tests that creating a valid new donor
TestAddDuplicateDonor	Tests that creating a duplicate donor displays a warning to the user before addition
TestModifyDonor	Tests that modifying a donor will not accept invalid input

Communication with Api.ai is currently untested: the parameters it returns after sending input are checked manually throughout development to ensure correct input. These tests require a thorough study of how Api.ai's servers process intent in a message. Unit tests for Api.ai would therefore consume some time to check for validity, which was spent testing the program’s MVC and validation capabilities instead.