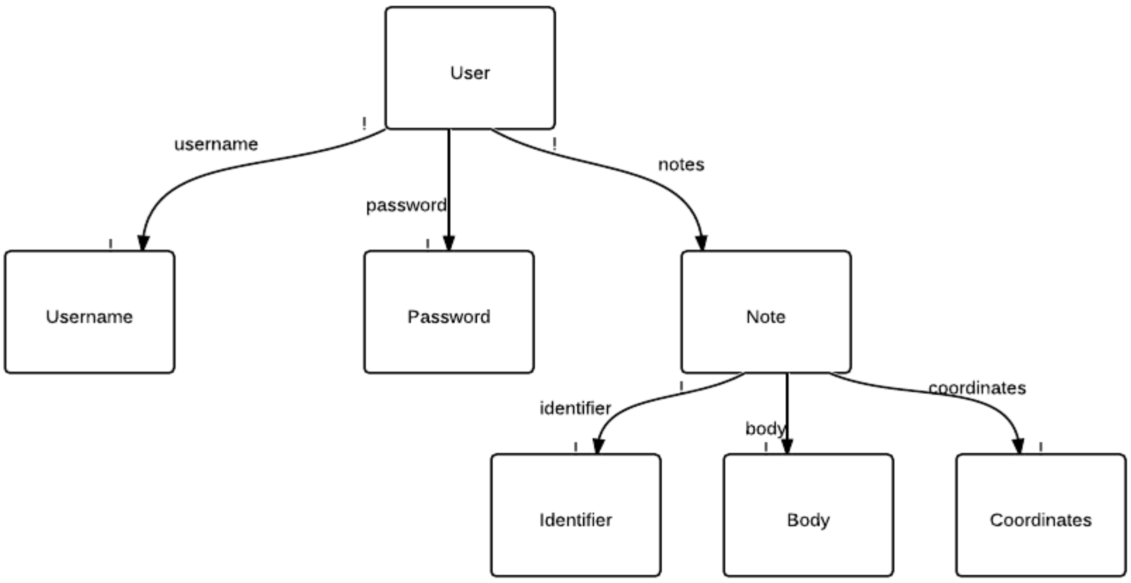
Rahul Rajagopalan
6.170

Network Stickies Object Models

Data Model for Storage



Ambiguities Resolved

The exact structure of the notes was not specified. I decided to model them after actual sticky notes, which are typically not formally organized into sections. As such, all text in the notes is stored in a single body string field, to give users as much freedom as possible. I also added coordinates fields, to keep track of where the notes are placed (allowing users to maintain organization) and identifiers so notes can be uniquely classified (it is possible that two notes could have the same body and be placed in the same location), avoiding collisions in the shelve file.

Complexities Ignored

The implementation details as to how the notes were stored are omitted from this object model. The model only shows the structure of the data. The notes were stored using Python's shelve module. It indexed UserData objects, containing hashed passwords and Note objects (serialized via JSON), using usernames as the keys.

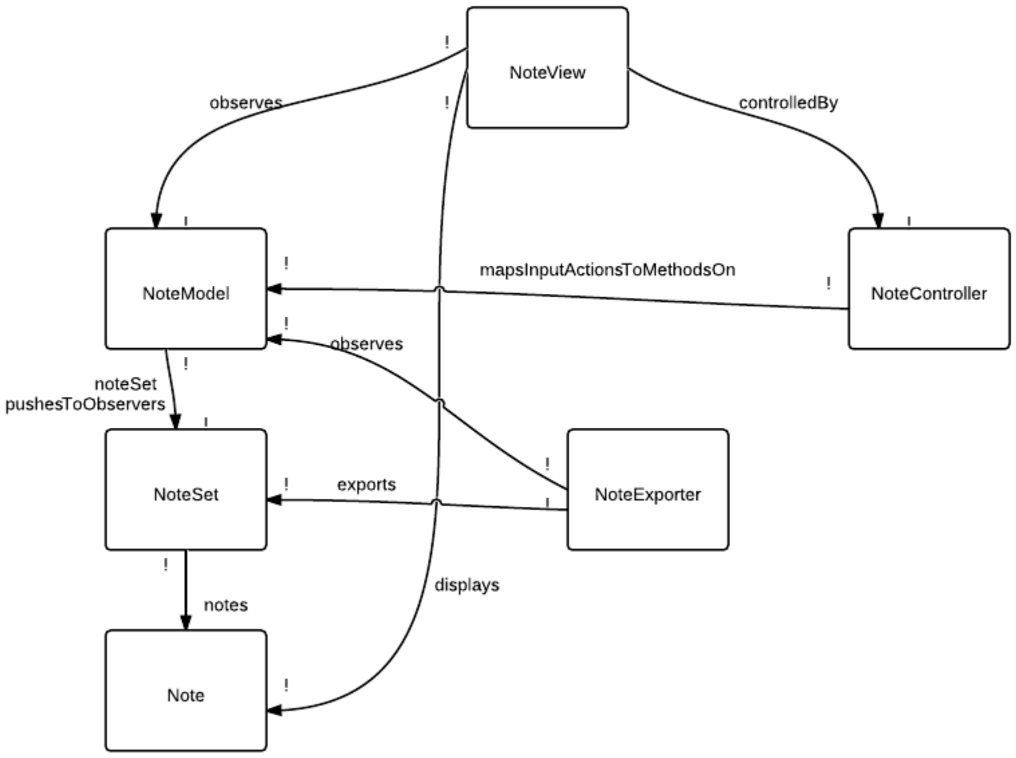
Designations

User is the top-level object representing a key-value pair in the shelve file. The keys are usernames and the values are UserData objects.

Coordinates are JavaScript objects containing information about the coordinates of a note. Their structure is an implementation detail (they are structured the same way jQuery structures coordinates when returning output for calls to .offset()).

Identifiers are unique string IDs that notes have, to keep track of multiple notes even if they have the same body text and coordinates.

JavaScript Object Graph



Ambiguities Resolved

It was not specified which entity should manage the notes while the user is editing them. I have done most of the object modeling and processing client-side, periodically using the server to save and load data via serialized NoteSet objects.

Complexities Ignored

This object model does not include the NoteMvcFactory functions. Their job is to build the graph as above, connecting all objects with their dependencies. If it was possible to draw an object model itself as a set in an object model, we could draw a one-to-many relation from NoteMvcFactory to the above graph, labeling the relation with "creates."

Designations

A Model-View-Controller architecture was used as the overall pattern.

NoteSets are immutable collection objects that contain all the Notes that a user has. Notes are also immutable.

A NoteModel contains a reference to the most recent NoteSet. It implements an Observable interface. It notifies its observers upon change by sending them a reference to the most recent NoteSet.

There is one NoteView for every note. It displays the Note on the screen, and observes NoteModel for changes. It sends input events to a NoteController object, which translates the inputs to calls on NoteModel.

NoteExporters also observe NoteModels. When the model changes, they serialize the NoteSet as a JSON string, and POST that string to the server to be saved. Currently the application only creates one NoteExporter, but more could easily be created if necessary for some reason.