Napkin Notes

Ehud Cohen and Daniel Kohanim

Mobile OS: iOS

Project Description

The Napkin Note app proposes to create a quick and easy location to place ideas and notes, with fast access to both text and images in one simple app. Take down notes in class or at the job or maybe snap a picture of something interesting or important, and jot down a note about the location. The simple and clean interface of Napkin Notes, with basic typed notes, drawing or photo inclusion, would allow for quick and convenient jotting down of important notes or ideas, without the hassle of selecting options or changing settings in most current market note taking applications.

Features:

- Text input
- In App photography
- On screen drawing
- Import Images from library
- Organized notes at start-up

Market Research

The app's design and content is designed towards students. In a situation where a student is in a classroom or has a project deadline coming up, the student will need an easy and quick way to save information and ideas. By providing a tool that will save text, images and drawings all on a phone, students will have an advantage in managing information.

The apple app store has many utility apps already in existence. This app intends to focus on college utility apps, and even further focus on the niche of data maintenance and note taking. Within this niche there are already competitors with varying features. Below is a chart comparing the Napkin Note app with other similar apps:

	Text	Import from library	Draw in notes	In app camera	Voice Recording	Price
Napkin Note	✓	✓	✓	✓		Free
Sketch Pad	✓		✓			$0.99
Simple Note	✓					Free
Snap-it	✓	✓		✓		$0.99
Evernote	✓	✓		✓	✓	Free

Planned Enhancements & Scale-Up

In order to stay comparative in a market that has a large size of funding and new entrants, we plan adding additional features to the Napkin Note app:

Future features

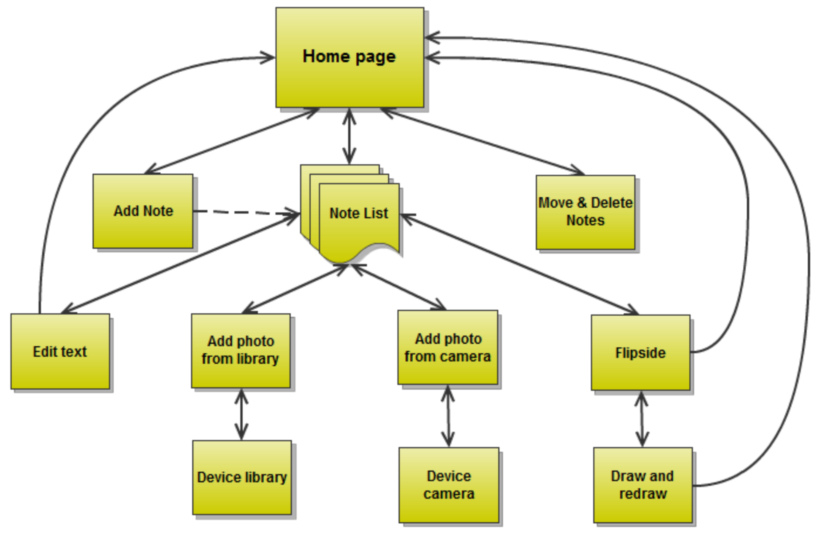
- Export notes via email
- Voice recording integration
- Note categorization
- Note reminders

More features

- GPS tagger of where notes and photos were taken
- Google Map feature
- Social media integration (Facebook and Twitter)
- Search functionality
- Multi-platform integration

In order to support the creation of new features an upgrade fee will be implemented. Were the basic app features will be free, but additional features will be priced. Through the additions of features such as social media sharing and multi-platform integration, we expect to be able to penetrate and build competitive barriers in the niche market of mobile college utilities.

App Flowchart



User Manual

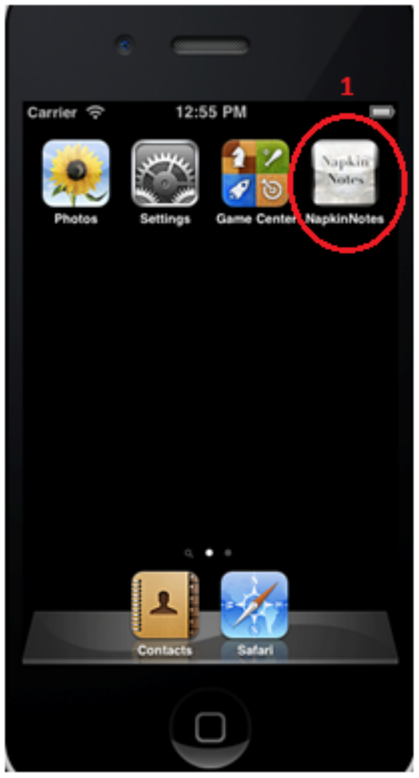
This document will serve as a guide for all users of the Napkin Notes application.

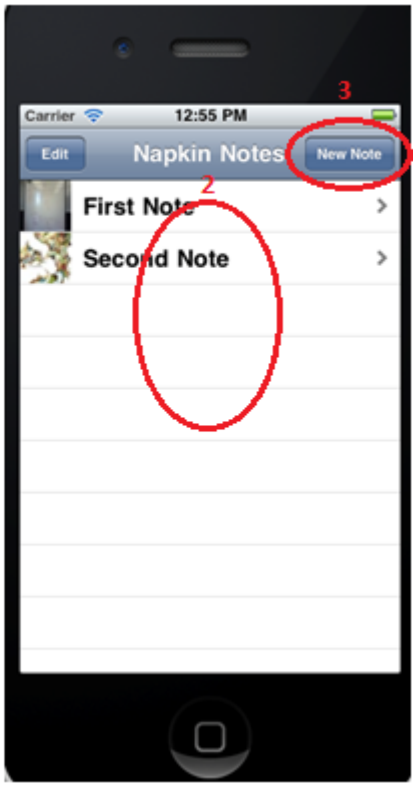
Table of contents

Compatibility

- This app works on 3rd generation iPod Touch (without the function of in app photography).
- 4th generation iPod Touch, iPhone 3, iPhone 4 and 4S

App Icon and Main Screen

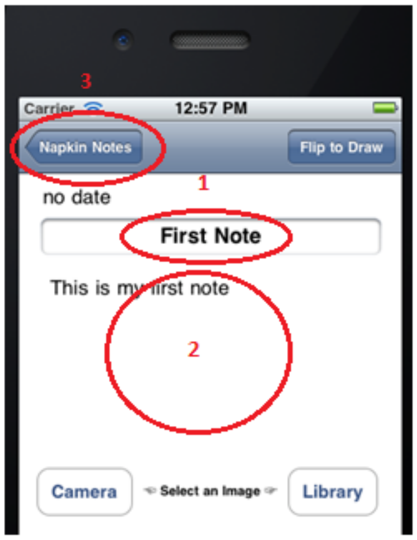


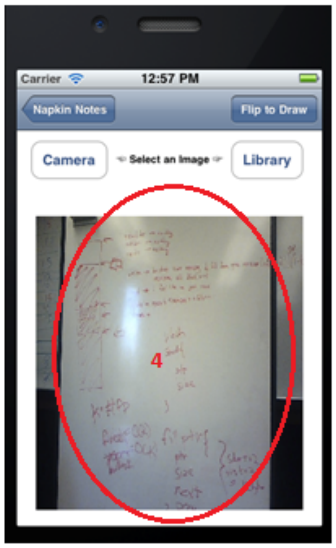


1. The Napkin Note app can be accessed once downloaded onto the device by clicking on its icon.

2. The main page of the app has a list of all the notes currently saved onto the app, which can be accessed by pressing one of the notes.
3. New notes can be created by clicking on the ‘New Note’ button.

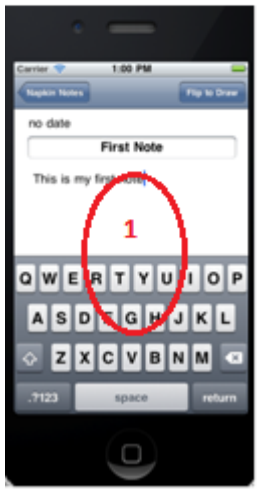
Note Screen and Navigating

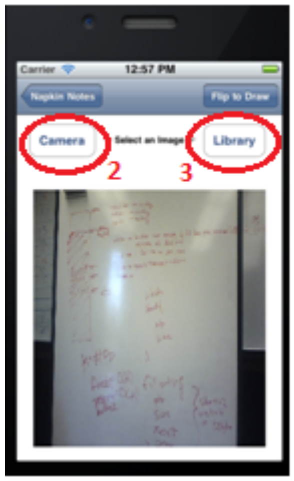


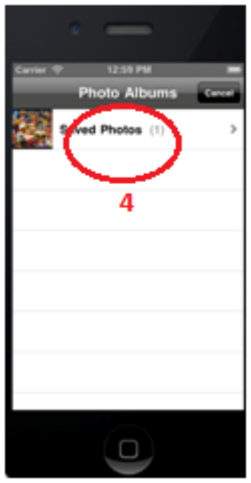


1. Title of the note
2. Text within the note
3. Click ‘Napkin Notes’ to save and to go back to main screen
4. Picture within the note (the picture will also be the thumb nail on the main screen)
    a. To get to this view you must scroll down the note

Inputting/editing Text and Pictures



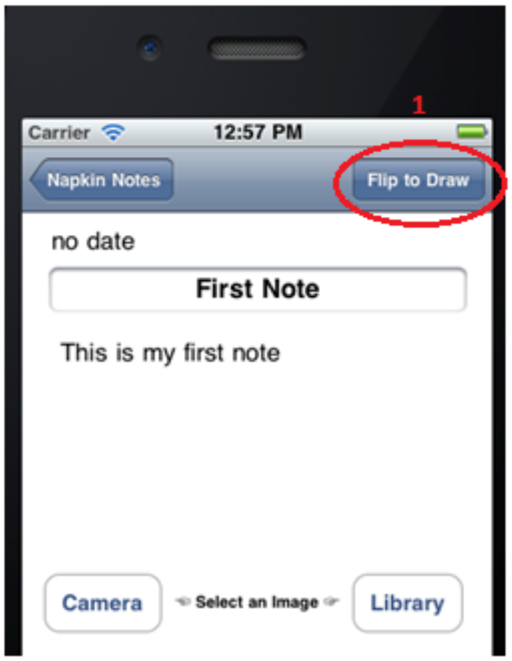


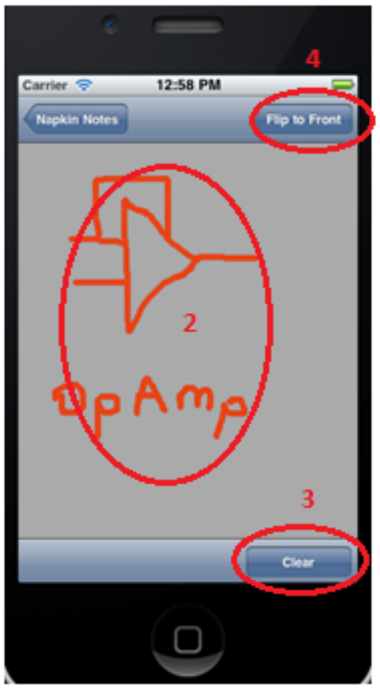


1. By clicking on the text area (as shown above) or on the title area a keyboard will appear for entering text

2. By clicking on the ‘Camera’ button you will be able to take a photo using the device’s built in camera. The picture taken will replace the current picture
3. By clicking on the ‘Library’ button you will open the library of photos already on your phone
4. Select from any of the photos in your library to replace the picture of the note you are currently in

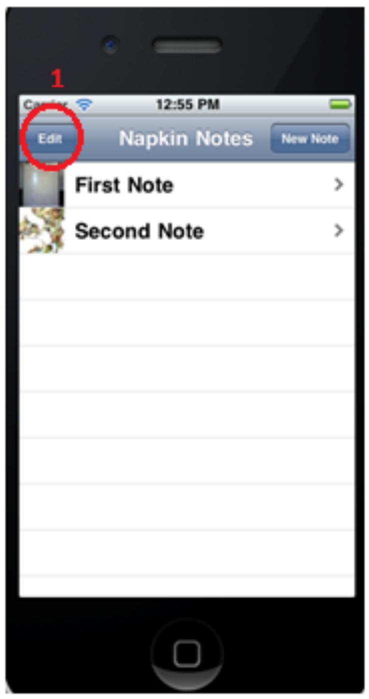
Drawing in note

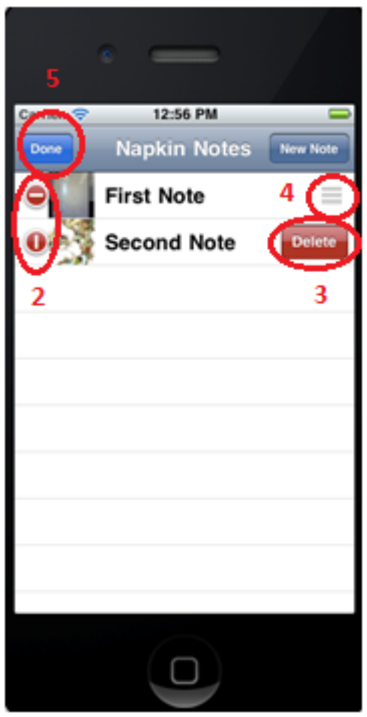




1. To make a drawing in the note, click on ‘Flip to Draw’ to flip to the backside of the note
2. Once on the backside, you can draw where you’d like
3. You can click the ‘Clear’ button to erase the entire drawing
4. By click on ‘Flip to Front’ you will be able to go back to the note screen

Deleting and ordering notes

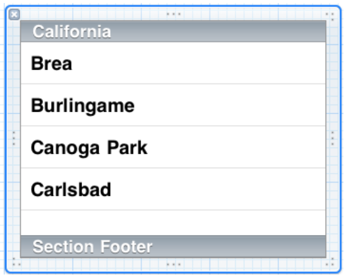




1. Click on the ‘Edit’ button to enable the delete and order features
2. By clicking on the ⊖ symbol next to the note you would like to delete, the ‘Delete’ button will appear (and the symbol will rotate to ⦶)
3. Click on the ‘Delete’ button to remove the selected note
4. By clicking and dragging the ≡ symbol you can move that note before or after any other note
5. Click the ‘Done’ button when editing is finished

Technical Specification and Design

The design is based around a Table View as the Root View Controller in a Navigation-based Application. The `UINavigationController` in the App Delegate files implements a specialized view controller that manages the navigation of hierarchical content.

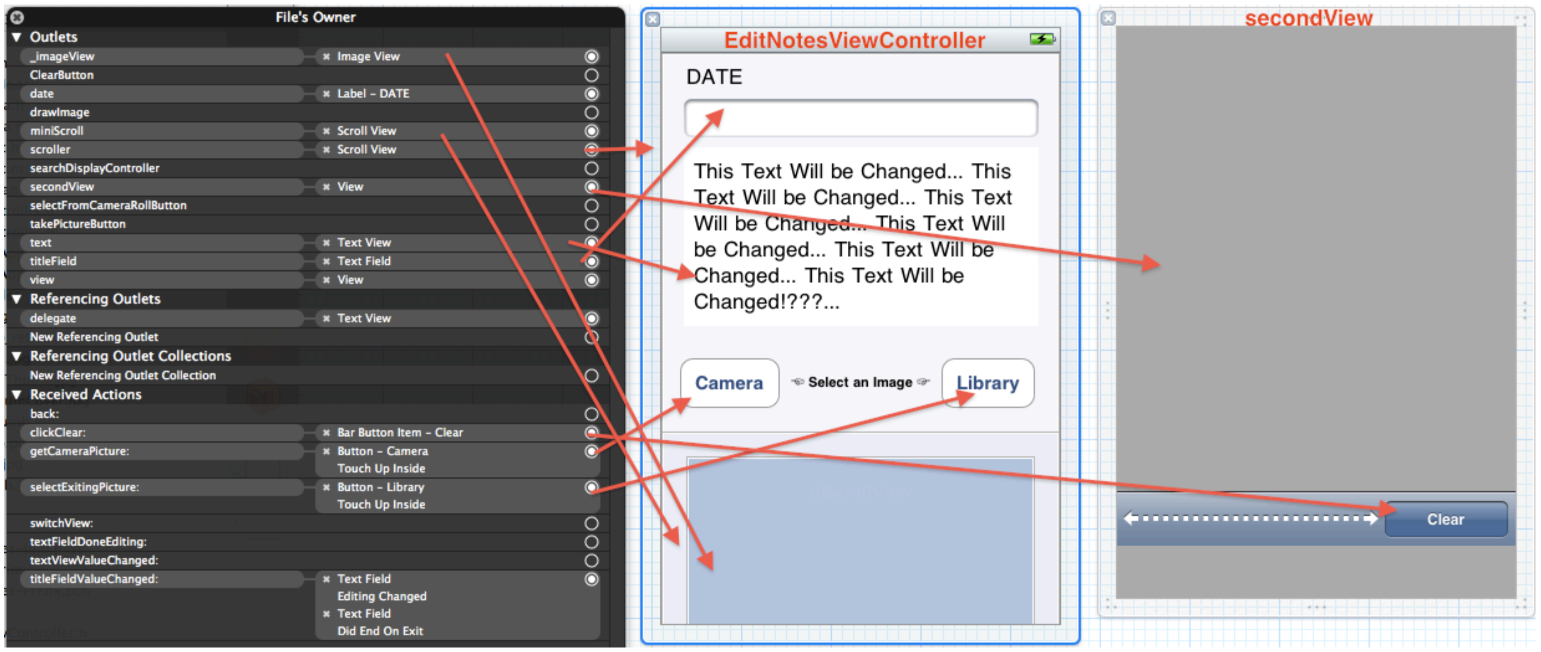


All notes created are stored as pointers in an `NSMutableArray` in the table. The generic Table View in Xcode is seen to the right, and since we chose to use a Navigation-based application to begin with, we start with a working, but empty, list as seen on an iPhone below.

At this point, making use of the Navigation Controller's bar button items, we allow the user to either edit the current rows (swap order of notes or delete notes) and also to add a new note (see User Manual for more detail on the layout of the main screen).

Once inside the main app, the core is built around a single large scroll view (labeled "scroller") in which all other views (text, image, etc.) is embedded. This enables the front page to be able to display the date, title, text and image used all in one view. Below you can see some of the buttons and views used for the Notes View Controller. A text field is used for the title, a scrolling text view for the user written text, a label for the date, two buttons for selecting images from the camera or library and a second scroll view with an image view embedded that allows for the user to pinch and zoom in and out of the image, and scroll to see the rest of the image.

As well, a second view is allocated that allows the user to draw via touch in the view's frame. To switch between these two views, we make use of the Navigation Controller's bar button items yet again to add and remove the second "subview" to and from the "superview".



File's Owner

▼ Outlets	
_imageView	Image View
ClearButton	
date	Label – DATE
drawImage	
miniScroll	Scroll View
scroller	Scroll View
searchDisplayController	
secondView	View
selectFromCameraRollButton	
takePictureButton	
text	Text View
titleField	Text Field
view	View
▼ Referencing Outlets	
delegate	Text View
New Referencing Outlet	
▼ Referencing Outlet Collections	
New Referencing Outlet Collection	
▼ Received Actions	
back:	
clickClear:	Bar Button Item – Clear
getCameraPicture:	Button – Camera Touch Up Inside
selectExitingPicture:	Button – Library Touch Up Inside
switchView:	
textFieldDoneEditing:	
textViewValueChanged:	
titleFieldValueChanged:	Text Field Editing Changed; Text Field Did End On Exit

EditNotesViewController
DATE
This Text Will be Changed... This Text Will be Changed... This Text Will be Changed... This Text Will be Changed... This Text Will be Changed... This Text Will be Changed!???...
Camera
Select an Image
Library
secondView
Clear

High Level Discussion of Code

The application is split up into 3 code blocks, each with multiple header and implementation files. The first code block involves files that store the information pertaining to each note, like the note's title, text, image etc. The second block involves files that form the Root View Controller and the App Delegate that run the entire application. The final block involves those files that handle editing the information in a note.

- **Note Information Storage:**

NapkinNotesData.h & NapkinNotesData.m (Data)

In these files, the values for the title, text and date are established as `NSString` text strings. We initialize the strings with a function call that creates and returns a new item using the specified properties, with our properties being the title, text and date as such:

```
  - (id)initWithTitle:(NSString*)title engText:(NSString*)engText
dateText:(NSString*)dateText;
```

NapkinNotesDoc.h & NapkinNotesDoc.m (Doc)

In these files, which inherit from the Data files, we again initialize our values for the image, the image thumbnail (used as the icon in the table view) and the drawing that is saved in the second view. Again we do so using:

```
  - (id)initWithTitle:(NSString*)title engText:(NSString*)engText
  thumbImage:(UIImage *)thumbImage fullImage:(UIImage *)fullImage
  drawing:(UIImage *)drawing dateText:(NSString*)dateText;
```

Having set up an object of the **Data** class in **Doc.h** and then synthesized that object:

```
NapkinNotesData *_data;

@synthesize data = _data;
```

We then allocate and pass the title, text and date from the **Data** files to the **Doc** as part of the `initWithTitle` function call:

```
  _data = [[NapkinNotesData alloc] initWithTitle:title
engText:engText dateText:dateText];
```

At this point, anytime we add or change any field in the app, the appropriate fields engText, dateText, title, drawing, fullImage and thumbImage should be changed to reflect that by accessing the **Doc** and **Data** classes.

- <u>**App Delegate and Root View Controller**</u>

NapkinNotesAppDelegate.h & NapkinNotesAppDelegate.m (Delegate)

These files establish the setup for the entire application, setting up the main window and Navigation Controller:

```
@interface NapkinNotesAppDelegate : NSObject
<UIApplicationDelegate> {
UIWindow *window;
UINavigationController *navigationController;
}
```

Once the application is launched, we can setup the initial `NSMutableArray` of the notes, and make the window visible to the user. All this is done inside the function:

```
  - (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
```

RootViewController.h & RootViewController.m (Root)

These files establish the root table view and the rows that will fill said table. We begin by setting the interface in the **Root.h** with an `NSMutableArray` for the storage of the notes in the table view and an instance of the notes editing view that will handle the actual contents for the notes themselves:

```
@interface RootViewController : UITableViewController {
NSMutableArray *_notes;
EditNotesViewController *_editBugViewController;
}
```

In the **Root.m** we set up what will appear when the root view loads. Aside from establishing the custom appearance for the table cells with the title and thumbImage taken from the **Doc** for all current notes stored, in `-(void)viewDidLoad`, we set the title of the Navigation bar to be the name of our app, and as well set up the button scheme, using the default “edit” built in to the Navigation bar, and a custom right bar button that when clicked will call a function to open a new note.

If the user selects the “edit” button, he enables two functions to be called. One checks the editing style and if the user selects delete, it deletes and removes from the list the note selected:

```
[_notes removeObjectAtIndex:indexPath.row];
[tableView deleteRowsAtIndexPaths:[NSArray
arrayWithObject:indexPath]
```

We also allow the user to move a row, by setting:

```
- (BOOL)tableView:(UITableView *)tableView canMoveRowAtIndexPath:
(NSIndexPath *)indexPath {
return YES;
}
```

Also, if the user selects the right bar button item to create a new note, we set the date to the current one using `NSDate` and create a new note:

```
NapkinNotesDoc *newDoc = [[[NapkinNotesDoc alloc]
initWithTitle:@"Title" engText:@"Text:" thumbImage:nil
fullImage:nil drawing:nil dateText:str1] autorelease];

 [_notes addObject:newDoc];
```

Lastly of course if the user simply selects a note, we display that note and its information using:

```
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
```

and we add the view for the user to begin on the new note like so:

```
[self.navigationController
pushViewController:_editNotesViewController animated:YES];
```

- **<u>Edit Notes View</u>**

EditNotesViewController.h & EditNotesViewController.m (Edit)

These files form the major portion of the application, as they take care of all the note information, whether adding text, a title, image, or drawing. As described in the Technical Specifications and Design section, each part is tied to a UI class, whether TextField, TextView, ScrollView, Label etc.

Once the Notes View is loaded, either by selecting a note or choosing to make a new one from the Root View, we implement all the parts of the function `- (void)viewDidLoad` which include placing a "flip" button on the Navigation bar to flip to the second view which calls a function `-(IBAction)switchView:(id)sender` that will add the second view by adding it as a subview to the current view. This function also changes the "flip" button to a "back" button, that when pressed removes this second subview and returns to the front side.

Also inside the viewDidLoad is the establishment of the size and zoom scale for the scroll view which houses the imageView that contains the full image associated with that note.

As well, on load, the function `-(void)viewWillAppear:(BOOL)animated` sets the values for all the fields in the note according to what is in the **Data** and **Doc** files.

If the user flips to the second view, he has the option to flip back or draw on the second view's screen. The drawing is handled in three steps, corresponding to three functions: `touchesBegan, touchesMoved` and `touchesEnded`. In touchesBegan, it first checks if the user touched anything, then records the last point touched. In touchesMoved, the app recognizes that a finger has stayed on the screen and is moving. Among the other lines of code that record the touches, the main set that keeps the context and draws the current line is:

```
CGContextBeginPath(UIGraphicsGetCurrentContext());
CGContextMoveToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
CGContextAddLineToPoint(UIGraphicsGetCurrentContext(),
currentPoint.x, currentPoint.y);
```

Also in this function we make sure to record the drawing in the **Doc's** memory, doing so by saving it as a graphic image from the current context:

```
UIImage *drawing = UIGraphicsGetImageFromCurrentImageContext();
_notesDoc.drawing = drawing;
```

When the user lifts his finger, we repeat the context keep of the line, but only keep the lastPoint.x and lastPoint.y. Again we also save this image to the **Doc** so that every time the user wants to return to this specific note's drawing, the drawing is saved and intact as was left. Also, if the user wants to erase the drawing, he can click the clear button which sets the image to "nul".

For selecting images via the iPhone’s camera or the Library, we used the UIImagePickerController class. We reserve memory and instantiate an object called picker which will be used to grab the image, and then later select the source type as either the camera or the library in both functions respectively depending on which is wanted:

```
      self.picker = [[UIImagePickerController alloc] init];
_picker.sourceType = UIImagePickerControllerSourceTypeCamera;
_picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
```

We can then set the full image and the thumbnail image inside:

```
     - (void)imagePickerController:(UIImagePickerController *)picker
     didFinishPickingMediaWithInfo:(NSDictionary *)info  {

    [self dismissModalViewControllerAnimated:YES];

    UIImage *fullImage =
(UIImage *) [info objectForKey: UIImagePickerControllerOriginalImage];
    UIImage *thumbImage =
[fullImage imageByScalingAndCroppingForSize:CGSizeMake(44, 44)];

     _notesDoc.fullImage = fullImage;
     _notesDoc.thumbImage = thumbImage;
     _imageView.image = fullImage;
}
```

The call to `imageByScalingAndCroppingForSize` calls a function of a helper class called UIImageExtras we found online. The class simply scales and crops the image by a scale factor passed to it to resize to a target size (in our case, we want the thumbnail image to be 44 x 44) and makes use of CGRect, which is a structure that contains the location and dimensions of a rectangle, to make a thumbnail image for the table view.