(arc)

Mobile devices, especially tablets, have seen widespread use as consumption devices; they are primarily used and suited for viewing content. Reading textual content is one of the iPad's major use cases, and there exist many popular and well designed applications that present text-based content in a user friendly way. Currently, this category includes books, magazines, newspapers, and so on.

Viewing code is a subset of consuming textual content, with a niche market we feel has a great need for it.

Reading code is a common task for most developers, for any number of reasons: to become familiar with a new project, to better understand an existing system, or even just for self improvement. Our current text editors and IDEs make both editing and reviewing code extremely easy, though on many occasions we find ourselves on the go and without access to such tools. We propose a code reading application which allows users to review code as easily and effectively as the more full-fledged editors found on computers today.

(arc)[1] is a code reader.

[1] (define arc '(arc reads code))

Table Of Contents

Application Overview

In short, the application we plan to develop has one purpose: present code on the iPad in the manner we've come to expect from the (numerous) capable text editors that we usually use on a computer. The purpose of this application is to enable developers to review code (that they or others have written) while on the go: the iPad is a device well-suited for this, being a portable content consumption device with a reasonably-sized screen.

It should be emphasised that this application is solely for reading code: editing is not supported. We feel editing code is best done on a computer, and has already been (very capably) handled there. Reading code on the go is, as we explore in the next section, a problem that remains without a satisfactory solution.

Existing Applications

A number of text editing (and in particular, code writing) applications for the iPad exist, with varying feature sets. However, while the best of them are mostly feature complete, they leave much to be desired in the following areas.

- Cost
- Aesthetics
- Simplicity

Textastic (http://www.textasticapp.com/) is perhaps the best existing application for what we want to do. It looks nice, has most of the features you expect of a modern text editor, and is fairly simple to use.

However, it costs USD$8.99.

We believe that we can replicate a subset of the features in Textastic (minus the editing functionality) and even improve upon some of it. For instance, the DropBox integration, in our opinion, leaves much to be desired.

Instead of charging for use of our application, we intend to offer it free and charge for additional niceties.[2]

[2] hipster colorschemes, languages, and so on

Proposed Features

We intend to include the following features in our app.

Document Sources

- Email/Web
- Cloud Services

Users should be able to open text files received through email or found on the web, using the iOS "Open in..." feature. In addition, the app will connect to cloud services (in particular DropBox, SkyDrive, and Google Drive) to allow users to open files they already have stored there.

Code Styling

- Syntax Highlighting
- Word Wrap
- Colour Schemes
- Font Family
- Font Size
- Tab Width
- Whitespace Indication
- Line Numbers

Syntax highlighting is absolutely necessary for any application that deals with code, as it makes reading code much simpler; similarly, word wrapping helps in this area. Users should also be able to select different colour schemes, and change the font (family and size). Showing if whitespace is made up of tabs or spaces can be useful, while line numbers help users keep track of their position within the file (and easily reference it.)

Code Semantics

- Code Folding
- Substring Search
- Parenthesis Matching

Code folding (the ability to collapse certain blocks of statements) helps make large files easier to navigate by hiding sections of code from view: this is a feature missing from current text editors on the platform. Users should also be able to search for substrings both within the current document as well as across all documents within a directory.

User Experience

- Vim/Emacs-style Status Bar

The status bar shows the file type, encoding, the current line number, and other useful information.

Possible "Good To Have" Features

The following are some possible features that we will consider adding if time permits.

Extensible

- Textmate Bundles
- Vim/Emacs syntax packages

As far as possible, we'd try to be compatible with existing packages for Textmate, Sublime Text, Vim and Emacs.

Code Semantics

- Symbol lookup
- Fuzzy Search

Symbol lookup entails selecting a symbol (such as a variable) and being able to see where in the code it was defined. Fuzzy search would allow users to search for substrings and see similar (not necessarily exact) matches.

Document Sources

- GitHub Integration
- BitBucket Integration
- Opening Compressed (i.e. Zip) Files
- RSS Feeds

Being capable of accessing code in repositories stored on GitHub or BitBucket would certainly be useful for developers of projects. Opening a compressed file holding multiple code files would also be nice to have (as code commonly comes in compressed files, which are generally unsupported on iOS.)

Another possible addition would be the ability to subscribe to RSS Feeds containing code snippets.

User Experience

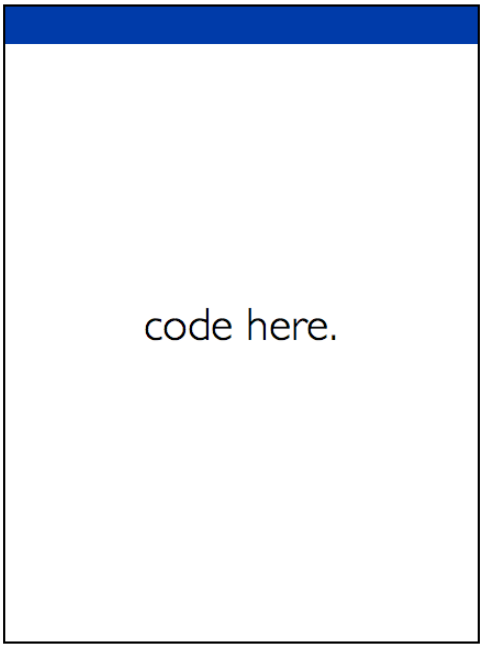
- Tabbed Views

Similar to the tabs offered in modern web browsers, tabbed views would allow users to quickly switch between several documents.

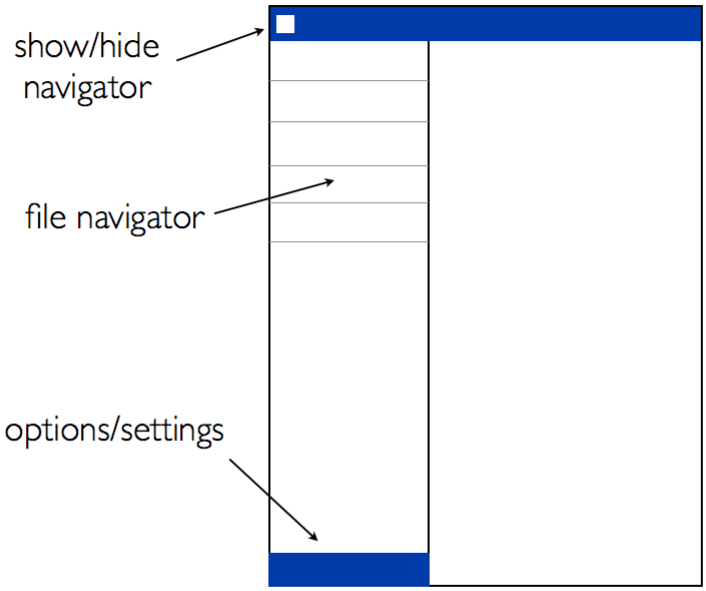
Project Schedule

Date	Milestone
17th March 2013	Receive Specifications
24th March 2013	Preliminary Design Document
7th April 2013	Progress Report 2
21st April 2013	Final Report
24th April 2013	Project Showcase

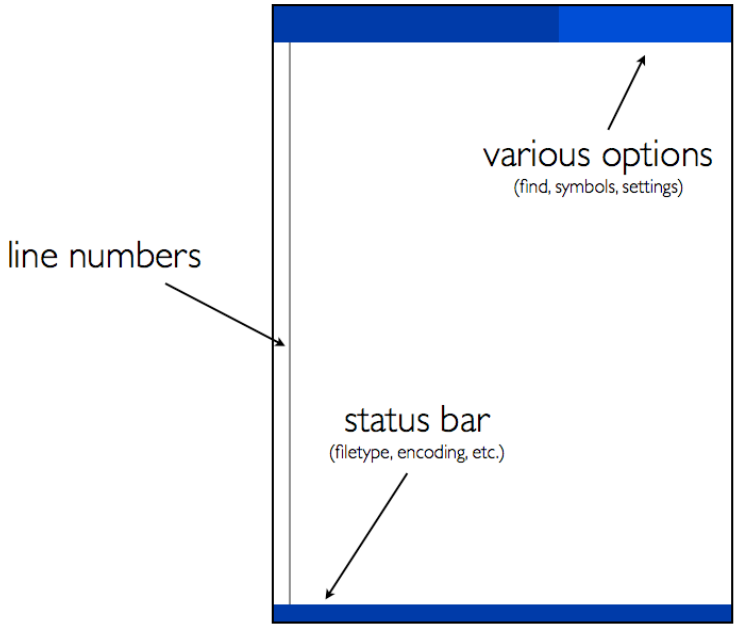
UI Mock-Ups



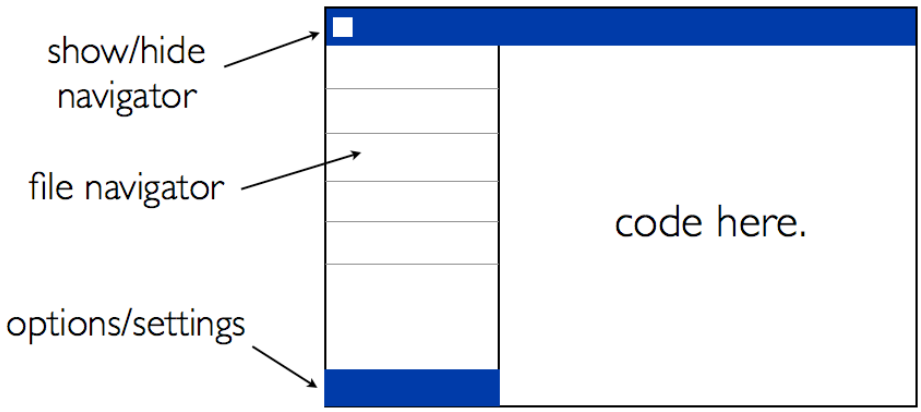
Portrait View



File Navigator View



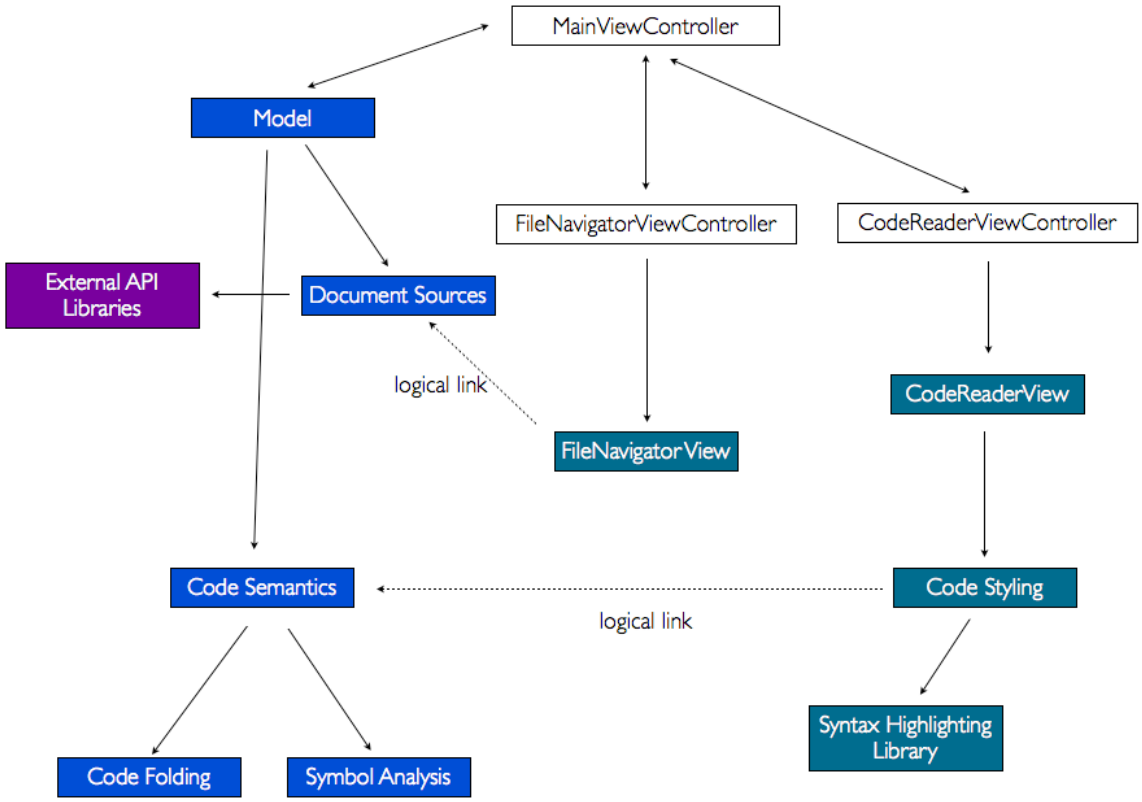
Code View



Landscape View

High-Level Design

Architecture Overview



Model

The Model handles both the file storage system and the business logic of the application.

The following are the possible sub-components for this component:

- Document Sources
- Code Semantics
- Symbol Analysis
- Code Folding

View

The View consists of the File Navigator View and the Code Reader View.

The following are the possible sub-components for this component:

- `FileNavigatorView`

- `CodeReaderView`

Controller

The Controller handles updates to and from the View and Model components.

The following are the possible sub-components for this component:

- `FileNavigatorViewController`
- `CodeReaderViewController`
- `FileSystemController`
- `CodeSemanticsController`

Individual Contribution and Roles

The roles below are mainly a guideline as to how we'll be splitting the work initially. We have every intention to change our work allocation as requirements and/or other issues arise during the project.

Omer

Omer is addicted to Red Bull and Lispy goodness.

He is responsible for the Model of the application; specifically, handling code semantics. He'll be writing code to understand code.

Michael

Michael likes coffee and occasionally travels. He codes a little.

He'll be in charge of overall product design and will be writing some of the View code.

Jerome

Jerome needs a better bio.

He is in charge of handling integration with DropBox, SkyDrive, and Google Drive.

Benedict

Benedict hates coffee, loves tea.

He'll be writing the Controller code and Substring Search.