IOU

SHARE AND TRACK EXPENSES WITH YOUR FACEBOOK FRIENDS

Jose B. Gomes - 12500741

Sunday, 8th, March, 2015

word count: 4789

word count excluding headers and sample code: 4789

BSc Computing Project Report,
Birkbeck College,
University of London, 2015

Abstract

This is the abstract

Table of Contents

Abstract 2
Table of Contents 3
Chapter 1 5
  Introduction 5
Chapter 2 7
  Literature review and Context 7
  Splitwise 7
    What is it 7
    Where it falls short 7
    Where it exceeds 7
  I.O.U - I Owe You 8
    What is it 8
    Where it falls short 8
    Where it exceeds 8
  Still Waiting 8
    What is it 8
    Where it falls short 8
    Where it exceeds 8
  IOU - by INEBAS 8
    What is it 8
    Where it falls short 8
    Where it exceeds 9
  Conclusion 9
    The problem 9
    How does IOU approach it 9
Chapter 3 10
  Research/Development Method 10
    The functional requirements 10
    The backend technology choices 11
    The frontend technology choices 11
    The decisions 12
  The workflow 13
Chapter 4 14
  Data/Findings/Designs 14
    Information diagram 14
    Designs 14
Chapter 5 15
  Analisys/Evaluation/Testing 15

Chapter 6 16
Conclusion/Recommendations 16
Chapter 7 17
Review/Reflections 17
Chapter 8 18
References 18
Chapter 9 19
Bibliography 19
Chapter 10 20
Appendix - About this documentation 20
The dilemma 20
The solution 20
The workflow 21
Developing locally 22
Roadmap 25
Details 25

Chapter 1

Introduction

The idea for IOU came up from a need raised long before the current technology allowed its' existence.

Being a foreigner living in London for over a decade, means that I had to share my house expenses with my housemates. This is not constrained to myself and my housemates, most of my friends also share their houses or flats. In fact this problem is not even constrained to a foreigner living in a different country, most students and young citizens also find it much easier to share their houses, easing down not only their expenses, but making friends and having due company in the process.

IOU is a simple application, that allows users to share and track their expenses with their Facebook friends, in fact, IOU has been designed so that one is not constrained to only use it for such purposes, one may wish to use it in order to track his own expenses, or who owes who in the last Barbecue party or even a family using it to know what groceries to buy.

The concept aims to be as simple as it can get, users create or are added to lists of products, each user can only view and edit lists that he/she belongs to. Each list has two sublists, a list of users and a list of products. The list of products is then subdivided into two lists, products that are due to be bought and products that have been already bought.

The sum of all products bought in a list, divided by the number of users, is the ideal amount each list member should had spent, however, in practice a member will rarely be in this situation, IOU will then label members as either a debtor or a creditor, if one is a debtor, he/she should be aware that is his/her turn to buy the next round of products.

A list lifecycle can last as long as it's members find suitable, when it is time to close the list, they may review how much each members owe each other, exchange money (which happens without any interference of IOU), then close the list.

For example, a list of expenses between two long term friends may never be closed and a list between housemates may be closed once one or more of the housemates decides to live somewhere else.

By no means IOU guarantees that the members who overspent money in a list will get paid back, it assumes that users have a "Gentlemen's" agreement among each other and should simply be used to track who's turn to buy products is and somewhat act as a reminder as to what should be bought.

At any given time, a user can open any given list and view who owes him/her and who he/she

owes money to. This number should be optimised in terms of change, to avoid hassle in terms of money exchange, that is, if one is a creditor in a list, he/she should never need to pay anyone back, likewise, if one is a debtor, he/she should only pay back creditors their given percentage. This in turn should avoid users paying someone who will then use the money to pay someone else.

To make this all work, IOU is distributed as a mobile phone application and enforces users to login using their Facebook profiles. What makes IOU possible now is the fact that most of the population nowadays happen to own a Facebook account as well as a smartphone with access to the internet. This makes the application very convenient, enabling users to easily find their Friends and check what needs to be bought on the go.

Chapter 2

Literature review and Context

Although a good idea, IOU is not alone, in fact, there are several other smartphone applications that allow expense sharing and beyond, in this chapter I aim to rationalise each of the main alternatives and contrast where IOU exceeds and falls short.

The list of IOU phone apps comparison in this chapter is by no means exaustive, neither in number of features nor in the number of applications. It simply names the most popular applications available and analyses their main features.

Splitwise

What is it

A fully featured money spplitting application that allows users to track expenses and even caters for fairness common bill sharing problems such as, to which extent is a girlfriend a housemate and how rent should be split.

Where it falls short

Although it integrates with Google+, it has no integration with Facebook, making it a little harder to find your friends and rather difficult to use in different devices, having to remeber which email address and password was used when one first signed up for a service seems slightly out of fashion now.

For some, this may be a plus, but I find it has too many features, making it complicated to get started with, for example, when creating an item, the user has to stipulate how to split it, debts simplification is an option and it does not enforce the usage of groups.

It only tracks what users have spent, not what is due to be bought.

Where it exceeds

Other than the points made above, it is really hard to find something wrong with this application, it is available for all major platforms, including a web version, it is very complete, boasts a beautiful interface and is never wrong in when the subject comes to maths accuracy. Most of all, it is free.

If I were to choose an application other than IOU, Splitwise would be my first choice.

I.O.U - I Owe You

What is it

It is a simple application to keep track of IOU notes.

Where it falls short

It is only available for Android users and the interface is not particularly pleasant to look at. It does not integrate with social media of any kind operating more as a stand alone application.

Where it exceeds

Simplicity, it does what it says in the tin, tracks IOU notes.

Still Waiting

What is it

Yet another IOU reminder application.

Where it falls short

It does not integrate with Facebook and the app is not consistent through different platforms, the iOS version being much more superior both in terms of features and interface. The iOS version is not free of charge. Much like 'I.O.U - I Owe You', it is constrained to track IOU notes and not having a 'due to buy queue'.

Where it exceeds

It has a pleasant interface and a pragmatic approach towards grouping bills based on people. Has a good set of options and even allows adding pictures and geolocation to your bills.

IOU - by INEBAS

What is it

An application for tracking expenses, bills and IOU notes.

Where it falls short

It is only available in the App Store and is not free of charge.

Where it exceeds

It has a beautiful and pleasant to use interface. Tracks lending and borrowing money.

Conclusion

In short, if one is looking for an IOU application, the options are endless, but what sets me into still wwant to create yet another IOU application is the fact that although all applications mentioned above (even those not mentioned), may do their job quite well, they fall short in the most fundamental issue, they do not solve my problem.

The problem

All applications tested and tried during the research time for IOU, I found that apps consistenly ignore the workflow of the user and simply focus on a more general approach to bill sharing, if they were to narrow their scope to a smaller niche, then they would be able to become the de facto app for that group.

The main feature that sets IOU apart from the others, is that fact that it allows users to set a desire to buy something, for example, if I am throwing a party, I may set up items to be bought, on a list, as my friends buy them, they mark the items as bought along with how much they spent, or, if I finish the milk at home, I may add it to my housemates list, perhaps another housemate will notice it before when strollilng in the supermarket before going home. IOU is not only a maths tool, but also, a communication tool.

How does IOU approach it

The way I envision IOU usage is by being an app that is checked every time one goes to the supermarket or remebers something to be bought for the weekend party or drinks the last bit of milk from the fridge. Once it becomes a habit to use it you no longer need to make groceries lists or ask anyone what is missing for the party or for the house in general.

In addition, IOU's datafeed happens real time, being always connected via a web socket, this means users do not need to trigger a page refresh, when the data changes, your screen changes too.

Finally, having Facebook integration, allows users to simply refer to IOU as an extension of an application they are already using. Making the eco system simple and familiar.

Chapter 3

Research/Development Method

Chapter two described the problem domain, this chapter explains the rationale behing the technological decisions taken in order to achieve the best results in the implementation of IOU.

The functional requirements

IOU is mostly an online application, user data gets saved in the server and devices being used to access the data should be ubiquitous, that is, accessible from both a native mobile applicatoin or from a web appliation running on a browser.

The application should only allow login and registration via Facebook and the presence of websockets is a big plus, so that users do not need to refresh their views in order to see the most up-to-date information, much like a chat room.

Users should be able to create an unlimited amount of lists, on which they can add both members and products. Each list will also have a name and an icon associated to them. Both of which are editable. A list can only be viewed, edited or closed by their members.

Once a product is added to a list, it should be treated as an item that is still due to be bought. Any user from any given list can 'buy' those products. Adding or editing the price and name of any product, both bought and due to be bought, can be done at any time. There should be a timestamp for each bought product that has been edited, staing the last time it was edited, any bought product that has been edited should display this timestamp.

Every list should also have a members view, where more members can be added and it should clearly state the breakdown of how much each member owes or is owed in relation to the user currently logged in.

The main list view should display the current user overall balance, that is, taking into account all lists he/she belongs to. Within the single list views, the overall balanced presented should be constrained to the current list.

The Login page, Terms and conditions of usage an Privacy Policy pages should be open to the public, however, all other pages should be private.

A left navigation menu should be present on all private pages, showing the picture and name of who is logged in as well as links to the home view, the terms and conditions view, the privacy policy view and a logout feature, that should clear all local storage data, logout the user and redirect to the login page.

A go home button should be present on all screens to aid navigation.

The main action of each page should be presented in a block button in the bottom of each page.

The backend technology choices

There is no doubt that to achieve it, data should be provided by means of a Web Service Restful API, where the endpoints expose data in a convinient JSON format to be consumed by either a native moblie application or from a web application running on a browser. However, the options on how to implent such structure are endless. Below is a list of the choices and the rationale behind them:

- PHP backend using Laravel:
    - Pros: This would've been the easiest solution, Laravel is a very flexible framework that provides powerful and yet simple to use features. It's routing system could easily cater for the nescessities of IOU, queues are easy to integrate and packages such as socialite can take care of the Facebook authentication.
    - Cons: Although there is a possibility to use other databases, MySQL is still the predominant option within the PHP world. Websockets are rather difficult to integrate. Even though HHVM can compile PHP code to C, it still lacks the extra speed boost nescessary for a real time app.
- Node.js:
    - Pros: Works really well with noSQL databases and websockets. Frameworks like Sails make it really easy to create and expose a Restful API and it performs quite well in terms of speed.
    - Cons: Not as easy as with PHP to find a suitable free server to start the work.
- Firebase:
    - Pros: No need to write an API, store your data on Firebase, set up your security rules and you are ready to go. Full support for websockets and Facebook integration. Fast speed from noSQL databases and the entry plan is free of charge.
    - Cons: If you want to process anything other than data, such as images and payment, you will still need a backend system other than Firebase. As your application expands plans become more expensive.

The frontend technology choices

For the frontend app, the choices were more limited, that is, in order to run a native application on a phone, one must develop constrained to the architecture he/she is writting if for. For example, Java is the native environment for Android apps and iOS developers have a choice between Swift

and Objective-C.

Another option was to create a hybrid phone application, that is, write it using common web application technologies such as HTML, CSS and JavaScript, then compile it using either PhoneGap or Apache Cordova to run in an in-app browser. As a side note, PhoneGap is a branch from the open source project Apache Cordova administered by Adobe.

The last option was to use Titanium AppCelerator, which sits somewhere in between a native and a hybrid application, Titanium exposes a JavaScript API and converts them to the native function calls to the target architecture on compiling time.

Despite having to choose which is frontend technology is best suited for IOU, one more element had to be considered. That is, the addition of a framework that would allow faster development and provide amenites such as high level abstractions for the platform and testing facilities. This is specially important, since the app has to run on more than one platform, including web.

At this point, the idea of using either a native application solution or Titanium, were begining to fade, native apps would be constrained to one architecture only and therefore, have to be re-developed for every other platform I would want to have it deployed to. Titanium, solves that problem, but falls short if you want to deploy your app to the web. Not to mention a rather cryptic API.

That's when I came across Ionic, Ionic is a frontend SDK for developing hybrid mobile apps. Framewoks such as Lungo and Topcoat, do cover part of what Ionic does, that is, they provide consistent widgets that can be used interchangeably throughout different platforms. However, Ionic goes the extra mile, not only providing frontend widgets, but also exposing and API that wraps Apache Cordova and a tight integration with Angular.js, making it the ultimate production tool when the subject comes to developing hybrid phone applications.

The decisions

The final choice was to stick to Ionic for the frontend and Firebase for the backend.

The main rationale behind this decision was how both technologies combine and complement each other. Firebase provides a driver that integrates with Angular.js, converting it into a true 3-way data binding tool.

Angular.js is a tool backed up by Google which aims to structure JavaScript apps and impulse the development of applications that otherwise would be impossible to develop or to the very least take too long.

Besides, Angular's Eco System boasts a vibrant community, integration with Node.js workflow tools such as yeoman, bower and grunt are a breeze, E2E and Unit testing tools options are vast and Directives trully extend the vocabulary of HTML.

It still falls short in a sense that the application is not native, that is, some of the transition effects may not be as smooth as those provided natively and despite Firebase being a really interesting backend solution, in future, if the app needs to consume more powerfull features, such as image resize and integration with a payment gateway, Firebase will not sufice.

Nevertheless, the combination of those tools have been proven to be right for IOU, future improvements will dictate what other technologies will be used.

The diagram below represents the final decision in the application set up in a more visual manner:

..... diagram here .....

The workflow

Now that the technologies have been chosen, an efficient workflow must be devised in order to maximize production and generate good quality builds.

Chapter 4

Data/Findings/Designs

the project outcome. This might be data collected and tabulated or the design of a program, or whatever outcome was obtained.

Links for the application and the application home page and web version and github

Information diagram

Designs

Chapter 5

Analisys/Evaluation/Testing

assessing or testing the project outcome. If the project is of type 3 or 4 then any computer code shold be tested using range inputs.

Look into all angular js testing frameworks, Karma, Protractor and Jasmine.

Chapter 6

Conclusion/Recommendations

as a result of the project. The project does not need to have a positive conclusion. For example, it might prove that some system was not successful. You should indicate to what extent your objectives have been achieved.

- add different currencies
- add a users guide/help page
- create an explanation video
- allow ordering of items (lists and products)
- compile the app for Blackberry and windows phone
- allow archiving lists
- create invite friends feature
- allow removing items from list
- allow removing user from list if he/she did not buy anything yet
- send welcome email when user registers
- have a loading icon when pages are being loaded
- create a splash screen
- improve overall look an feel
- get an approval from Facebook to view a complete list of all friends for any user

Chapter 7

Review/Reflections

reflect about usage of noSQL and how hard it was to make a transition between the relational world to Firebase.

Chapter 8

References

Chapter 9

Bibliography

Chapter 10

Appendix - About this documentation

The dilemma

At the time of this writing, I faced a dilemma, which word processor is the best processor for writing this documentation.

The most obvious choices were:

- Microsoft Word
- Apple Pages
- Google Docs

Surely any of the choices above would be suitable, however, in my humble opinion, such options can be somewhat 'evil' when it comes to document formatting, turning most documentation written in one platform unreadable in the other or to the very least almost certainly not looking as intended.

In general, converting the document to a PDF format remediates this problem, however, I am now left with the fact that my content is tightly coupled to the editor that created the documentation. Modification is difficult, for example, someone else may wish to edit the document, or perhaps if at some point in time in the near future I decide to present my documentation as an HTML document on the web, or perhaps as a deck of slides or even publish it in a book format, I may have to revise the entire text looking for any 'evil' formatting issues that was not visible in the former format.

Finally, as a writer, when writing documentation, I should concentrate on writing the documentation, and not about auto generated formatting issues that may arise and drag productivity in a typical writing session. The writer should only worry about the semantics and the content of his/her writing, formatting should be done separately, perhaps not even by the writer himself, or better, simply choose a new format from predefined options written by a talented designer.

LaTeX is a really good option to move away from the formatting problems mentioned above, and beyond, however, it does trap the writer with a little clutter to tinker with in terms of settings and so on. What I am trying to say is, once the document is finalised, it no longer consists of the content and the content only, but also carries several formatting tags. This is one hand demonstrates how powerful LaTeX can be, but in the other, may confuse and distract the writer.

The solution

Since this course is about computer science, I set off to find a solution that would allow me and

any other developer to run away from the masses and write simple interchangeable documentation with ease. Writing a solution that could potentially be further enhanced to the point a non programmer could also benefit from.

The requirements were:

- Write content and content only, without distractions
- Formatting should be written separately and be interchangeable/themeable
    - This also means that if some time in future I want to publish my content to a different media, I should be able to do so without too much effort
- It should not be coupled to any specific text editor
- It should have an automated building and deployment solution

The answer was always there, Markdown. Markdown is a really simple and easy to use markup language, it is the de facto standard for readme files in software development and widely used in blogs throughout the web.

A short introduction to Markdown syntax can be found in the link below:

http://daringfireball.net/projects/Markdown/syntax

Although Markdown may not be as powerful as LaTeX, the community around it is immense and really keen in providing further enhancements to it, the following link lists many projects and plugins that address some of those short-comes:

https://github.com/cben/mathdown/wiki/math-in-Markdown

The community also provides an excellent open source Markdown editor called Mou, although Mac specific, all other major operating systems have free alternatives to Mou:

http://25.io/mou/

Finally, there is an open source project which is now maintained by Jakob Truelsen and Ashish Kulkarni, called wkhtmltopdf, this headless command line tool, allows any HTML document to be converted to PDF, and since Markdown is easily convertible to HTML, we now have all the tools we need in order to create our documentation.

http://wkhtmltopdf.org/

The workflow

To make this work, we will need to automate every step of the process, so that we can only write Markdown, then compile/deploy our work with only one command and in the process, if we wish to do so, add some personalised styles to our document.

The application that holds the documentation has been scaffolded using Yeoman, more specifically, using a generator called generator-jekyllrb.

https://github.com/robwierzbowski/generator-jekyllrb

Although Yeoman is an NPM package, backed by Node.js, it combines three simple but powerful Ruby Gems:

- Jekyll: A static blog generator created by the GitHub Team. It provides an easy to use templating language called Liquid as well as giving the means to transform Markdown files into HTML, finally compiling the entire application into deployable static websites.
- Redcarpet: A Markdown to HTML converter.
- Compass: A CSS pre-processor extension of SASS.

Moreover, Jekyll makes it really easy to deploy and host your application directly on Github free of charge.

This Yeoman generator also leverages the power of Grunt and Bower into the workflow.

Most of the Grunt tasks used for this documentation have also been used for the main IOU application, however, a few notable differences are mentioned below:

- grunt-build-control: Allows deployment to Github via Git.
- grunt-jekyll: integrates Jekyll with Node.js.
- grunt-wkhtmltopdf: Compiles HTML files to PDF.

Developing locally

If you wish to use this workflow for other projects or simply try out what has been created so far, please follow the following instructions and make sure all dependencies are met and available on your path:

Dependencies:

- Git
- Ruby and Ruby Gems
  - SASS
  - Compass
  - Jekyll
- Node.js and NPM
  - Grunt
  - Bower
- wkhtmltopdf

Git clone this repository and `cd` into it:

```
$ git clone https://github.com/jbonigomesbbk/jbonigomesbbk.github.io && cd
$_
```

Install the NPM packages:

```
$ npm install
```

Install the Bower dependencies:

```
$ bower install
```

Serve the app:

```
$ grunt serve
```

Other than `serve`, you may wish to run:

```
$ grunt serve:dist
```

The latter will first build the application then serve the optimised code, the former, may be a better option when debugging as files will not be minified and changes will be automatically refresh in the browser using 'browser sync'.

Once displayed in the brower, you may wish to explore the documentation and view the PDF generated version too, link found right below the left navigation bar.

The directory structure will look as follow:

- .jekyll
- .sass-cache
- .tmp
- app
- dist
- node_modules

And the root directory hold the following files:

- .bowerrc
- .csslintrc
- .editorconfig
- .gitattributes
- .gitignore

- .jshintrc
- _config.build.yml
- _config.yml
- bower.json
- Gemfile
- Gruntfile.js
- package.json

The top most directories, prefixed with a dot '.' only hold temporary files required by Jekyll to serve the application locally.

All files in the root directory are common configuration files for Git, Jekyll and Node.js.

The node_modules directory holds all the Grunt tasks.

The app directory is where we, developers will actually work.

The dist directory holds the last build from our application, that is, the result of our code. For every build, Grunt will delete this directory and re-create it based on our changes. The dist directory holds the optimised code that is ready for production.

The app directory tree structure looks like this:

- _bower components
- _includes
- _layouts
- _posts
- _scss
- docs
- fonts
- img
- js
- pdf

The top most directory `_bower_components` holds the Bower dependencies. All the directories prefixed with an underscore '_' will not be coppied to the dist directory during the build process.

- _includes: holds the cover page and the xsl file used to generate the table of contents in the PDF.
- _layouts: hold the layout for the main web page, the layout for the documentation page and the layout for the PDF file.
- _posts: holds the actual Markdown that makes up this documentation, they have been separated by chapters.

- _scss: the SASS styles for the app and the print.scss, used by the PDF file.
- docs: holds the documentation landing page.
- fonts: the web fonts used in this project.
- img: the images.
- js: the JavaScript.
- pdf: a placeholder HTML file containing all chapters and extending the PDF layout, as well as the actual generated PDF file.

During your workflow, you may notice that the PDF file looks well formated when running `grunt serve:dist` however, this statement does not hold true when running `grunt serve`, this is because Jekyll compiles the temporary CSS file in a different location when running on debug mode, to avoid too much overhead, when testing the PDF's layout you may use this short-hand:

```
$ grunt build
```

This will build the latest changes and save them under the dist directory, including a well formatted PDF file. You may now open the PDF directly:

```
$ open dist/pdf/index.pdf
```

If you wish to deploy this app to GitHub, you will need to update the remote option in the buildcontrol task, found in the Gruntfile.js, swap it with a valid GitHub Page repository path, more details about GitHub Pages, Jekyll and BuildControl can be found in the link below:

- https://pages.github.com/
- http://jekyllrb.com/
- https://github.com/robwierzbowski/grunt-build-control

Once set up, you may deploy your code using:

```
$ grunt deploy
```

Roadmap

- Keep improving this workflow and gathering feedback from others on how to improve it.
- Create a more generic version to use as a starting point for future documentations, perhaps a Yeoman generator.
- Fix the issue with no styles in the PDF's generated with `grunt serve`.
- Create new themes for the Markdown's, so far it is using the generic GitHub flavour styles.

Details

The source code for this documentation can be found at:

https://github.com/jbonigomesbbk/jbonigomesbbk.github.io

The latest build of this documentation can be found here:

http://jbonigomesbbk.github.io/docs