MarkUP

Huixia(Maggie) Wang
A20324156

Background

MarkUp is one of subproject of Project Factor. Project Factor was initiated to build a suite of applications that would automatically generate text transcripts from IIT course videos using open-source software tools. Project Factor is divided by three parts:

- Project Convert
- Project Window Pane
- Project MarkUp

Introduction

For project Convert and project Window Pane, they convert video or audio(such as a speech) into text with timestamp using machine translator. Project MarkUp is responsible for handling those unstructured text. About project Convert and project Window Pane, please refer to other teammates' research paper. Now let's focus on my project - MarkUp.

MarkUp is an Android application, which allows a user to enter a drag and drop application and mark unstructured text via a simple GUI interface. The Goal of MarkUp:

- Convert unformatted text into formatted text
- Build user Content Management System
- Formatted text can be rendered into a DocBook or HTMLBook which allows for export to PDF, HTML and even ePub
- PDF and HTML formats can be printed too - making read -time book

Functionalities of MarkUp

Functionalities of MarkUp I have implemented is following:

- User authentication
- Account management
- Multi-user synchronous editing capabilities

- All functions performed using Android touch based interface targeting tablets or phone
- All user markup saved and referenced from a common database
- Single user interface which will add, remove and share functionality based in user type(readers, user)

Next, I will explain how I implement MarkUp project. Hope students who want to go on my work pick up my work easily.

Installation

Software Tools

There have two parts for software installation, backend and front end. Sails.js installation is prepare for backend. Android Studio installation is for front-end. I will explain them respectively.

Sails.js Version

0.12.0

Sails.js Installation

To install the latest stable release with the command-line tool:

```
sudo npm -g install sails
```

On Windows (or Mac OS with Homebrew), you don't need sudo:

```
npm -g install sails
```

You may see some npm WARN deprecated messages when installing Sails. These can be safely ignored. They don’t come from Sails itself, but rather from some packages that Sails relies on--packages which themselves rely on some older modules. The Sails team is committed to keeping the framework secure and stable, and sometimes that (counter-intuitively) requires using some older versions of packages. See this thread for more info about the warnings, and this one for a discussion about updating dependencies in general.

You can go to http://sailsjs.org/get-started see more details.

Javac -version Version

1.6.0_65

Android Studio Installation

Firstly, you go to http://developer.android.com/sdk/index.html to download Android Studio, then While the Android Studio download completes, verify which version of the JDK you have: open a command line and type javac -version. If the JDK is not available or the version is lower than 1.8, download the Java SE Development Kit 8.

To install Android Studio on Windows, proceed as follows:

- Launch the .exe file you downloaded.
- Follow the setup wizard to install Android Studio and any necessary SDK tools.

On some Windows systems, the launcher script does not find where the JDK is installed. If you encounter this problem, you need to set an environment variable indicating the correct location. Select Start menu > Computer > System Properties > Advanced System Properties. Then open Advanced tab > Environment Variables and add a new system variable JAVA_HOME that points to your JDK folder, for example C:\Program Files\Java\jdk1.8.0_77.

You can go to http://developer.android.com/sdk/installing/index.html see more android studio installation details.

Android Studio Configuration

This is my build.gradle file that includes configuration information, such as compileSdk, buildTools and dependencies information.

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.1"
    useLibrary 'org.apache.http.legacy'

    defaultConfig {
        applicationId "com.example.miaodonghan.markupproject_01"
        minSdkVersion 15
        targetSdkVersion 23
```

```
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'us.feras.mdv:markdownview:1.1.0'
}
```

Hardware Tools

I use Nexus 7 API 22 (emulator in android studio). You could use any android platform version higher than android 4.0 API 15.

Other Tools - Postman

I always use Postman to debug backend, which takes less time than debug using android studio emulator or android device.

Postman installation

Postman now offers a Mac App. Unlike the Chrome app, the Mac app is packaged with add-ons that make request capturing and cookie handling seamless.

To install go to https://www.getpostman.com/apps, and click 'Get Mac App'.The download should take a few minutes, depending on your internet connection. Once you've downloaded the app, you can install and launch Postman like any other Mac app.

Run MarkUp application

After finishing installation, you could git pull MarkUp project form my github account. Before you run MarkUp , there have two thing to modify. Firstly, you open bash in your computer, then input ifconfig to find inet address. Secondly, you should find local.js via path MarkUP_WebSerivce/webservice/config, replace host by new inet address. Thirdly, open your android studio, then find Strings.xml via path ./app/res/values, you also need to replace http://old_inet_address:1337 by new inet address.

Architecture of MarkUp

Architecture of MarkUp is showed in figure 1:

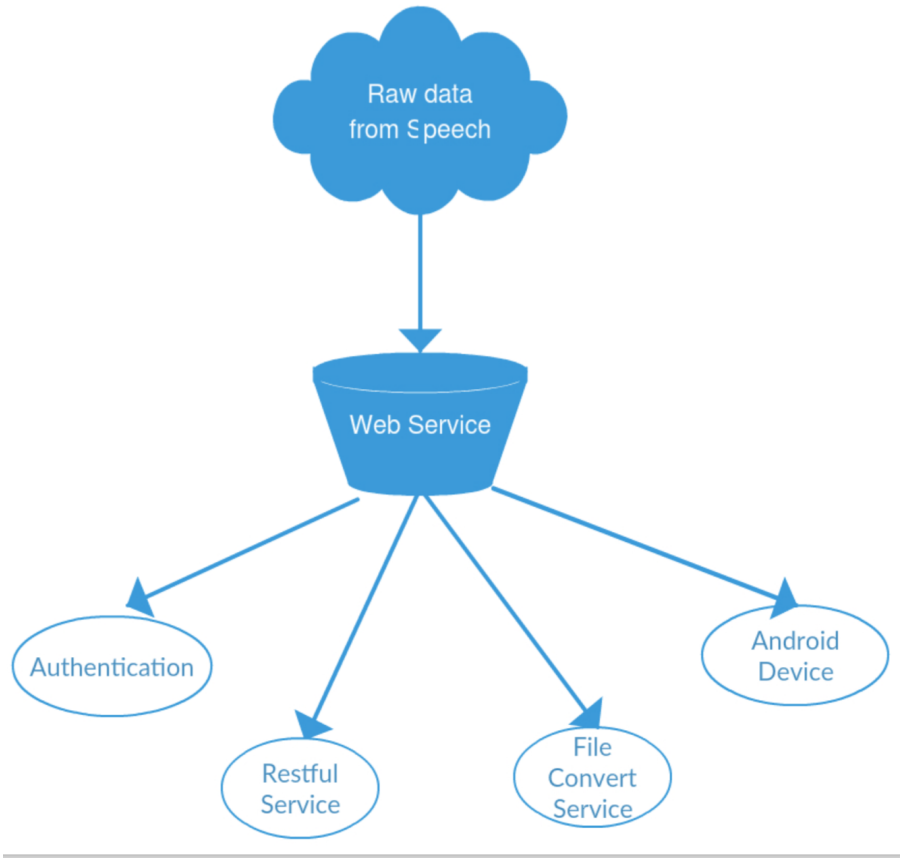

figure 1

Raw data from Speech in figure 1 is unstructured text that is converted from video or audio using machine translator. Web Service is server-side, which aims at storing data and handling logic relationships between models. MarkUp web service mains includes four services:

- Authentication

Authentication is a process in which the credentials provided are compared to those on file in a database of authorized users' information on a local operating system or within an authentication server. If the credentials match, the process is completed and the user is granted authorization for access.

- RESTful service REST stands for Representational State Transfer, which is an architecture style for networked hypermedia applications. It is primarily used to build Web service that are lightweight, maintainable and scalable. A service based on REST is called a RESTful service. A restful API is an application program interface that uses HTTP requests to GET,PUT,POST and DELETE data.
- File Convert Service File Convert Service is to convert markdown format to other document format, such as PDF, ePub, by using some libraries(pandoc).
- Android Device

Android Device is serve as an GUI interface. After server processed data, android device is used to render or display data in a great UI.

Flow chart of MarkUp

According to MarkUp Functionalities I implemented, I design a flow chart below so that you are able to understand MarkUp app clearly.

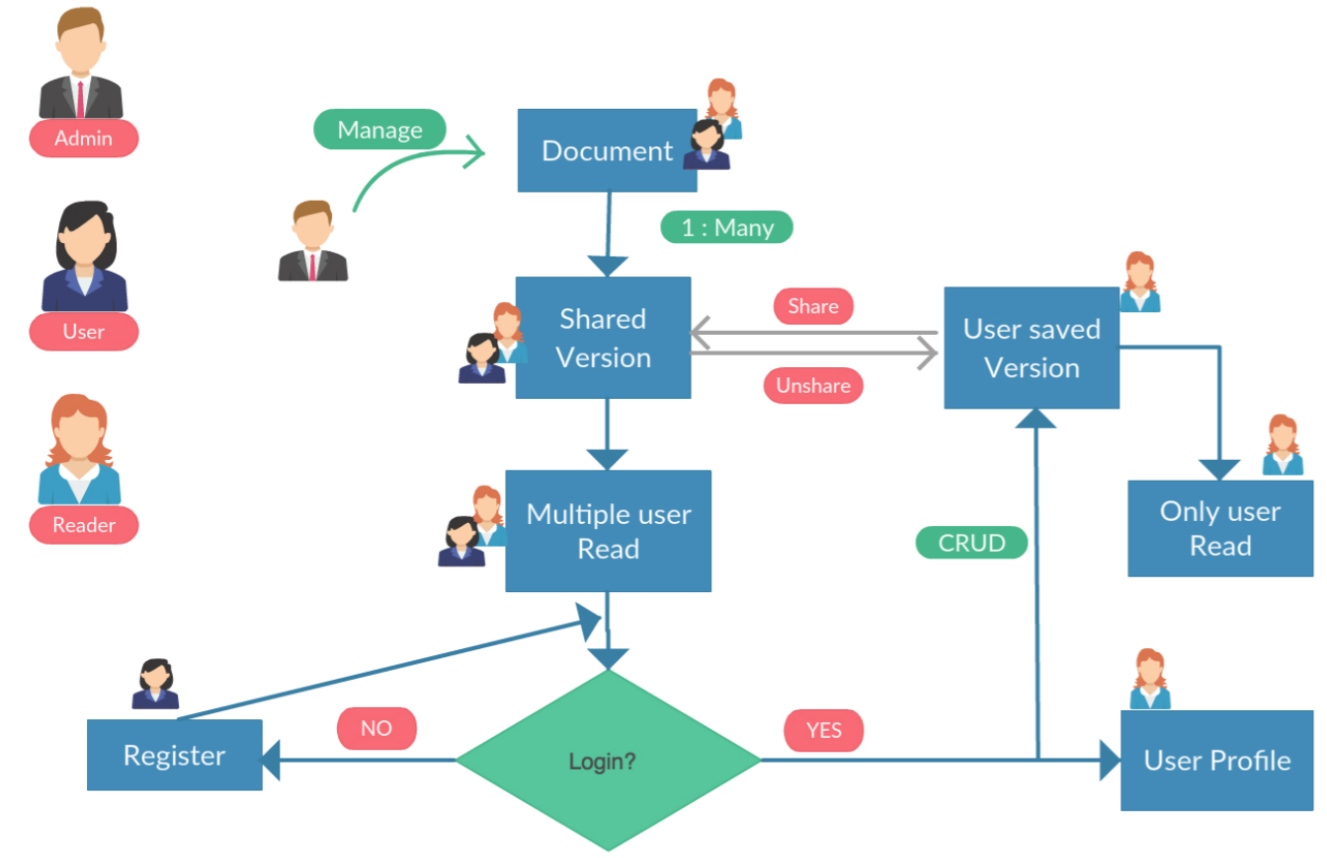

figure 2

Before introduce flow-chart above, let's assume that we are using IIT blackboard online video database now. Other subproject will convert each class speech into text, then those unstructured text translated by machine translator is put in database. Here, I defined every class text as a document. For users(student) who use MarkUp, he can edit document(unstructured text) to be user own modified document. So let's call those documents translated from machine translator as original version of document. Those documents modified by users are different versions of document. If one document is modified by multi-users, every user may create one or more versions of the document, so one document may have lots of versions.

As figure 2 shows, there have three user types, Admin, User and Reader.

Admin is responsible for manipulating database that stores unstructured/raw data. Let's use IIT blackboard online video as example again, after machine translator finish translating, admin has to check if there have new documents and then updated database by adding new translated documents into the database. Maybe, in the future, Factor project can add new document into database automatically without admin.

Things Admin can do:

- Admin can add new documents to database
- Admin has right to decide if new documents could be put in database
- Admin can update/delete documents in database

Things Users could do:

- User can access all documents and all shared versions in database
- User can edit any version they want to
- User can create new version by saving edit exist versions of document
- User can access his own documents and all shared versions in his profile.
- User can share/unshare their own versions in their profiles
- User can delete their own versions in their profiles

Shared versions are versions that users shared their own versions of documents so that all readers and other users can access their shared version of documents. If user doesn't share their own version of one document, all readers and other users(except himself) cannot access your unshared version of one document. One more thing, in model design, all versions of document state is unshared by default.

For readers, reader can access all documents, original version of documents and shared version of document. In addition, readers also can read and edit versions of documents. However, readers cannot save versions of documents they edit unless they register an account and logged in. Things readers could do:

- Readers can access all documents and all shared versions in database
- Reader can edit any versions of documents they want to, but cannot save versions they edit

Backend Design

After discussing MarkUp architecture and flow-chart, hope you can understand what I want to do. Now let's see how to implement those functionalities I mentioned above.

I chose sails.js as web framework, which is designed to emulate the familiar MVC pattern of frameworks like Ruby on Rails. About sails.js, you could access its website to learn how to use sails.js. Compared with usual MVC pattern, MVC pattern doesn't

include view in sails.js, therefore, android device/emulator play a role of view. In this section, I will explain models and controller respectively, explain android part in front-end section.

Model Design

According to explanation above,it is easy to know that, at least, there should have a model user and a model document in content management system because it is user to manipulate data(document). In addition, I also designed a version model. In order to describe model and their relationships clearly, I used E-R diagram to show attributes of models and relationship between different model.

An ER model is composed of entity types and specifies relationships that can exist between instances of those entity types. ER model is an abstract data model that defines a data or information structure that can be implemented in a database, typically a relational database. Rectangle represents entity, isometric square represents relation between models. Ellipse represents attributes of model.

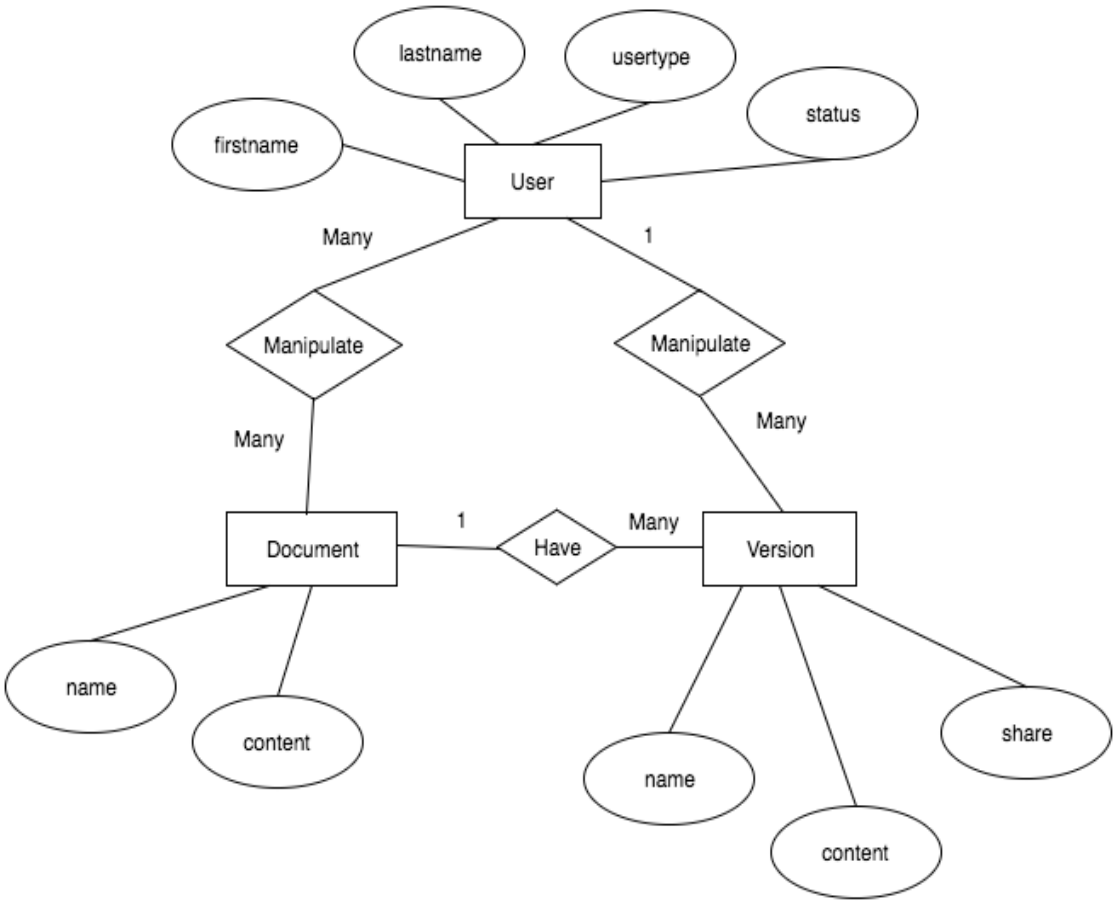

figure 3

As figure 3 shows, there have three models, document, version and user. Association(relationship) between models is following:

- Document : Version = 1: many
- User : Version = 1 : many
- User : Document = many : many

In addition to these three basic models, there is one more model, auth. Auth is designed to check if authorized user is logged. Auth E-R diagram is showed below.

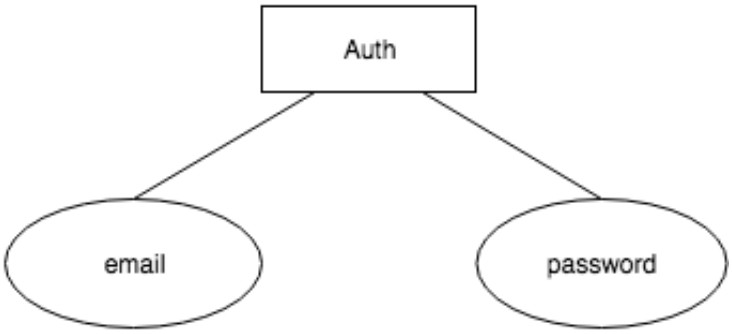

figure 4

If you want to know more details, you are able to find it via path MarkUP_WebSerivce/webservice/api/models.

Router Design

According to functionalities of MarkUp, I designed URLs that users can enter in the browser. These URLs map to some specific physical file mapped to a directory on the web server which we call the virtual directory. When user sends requests, controller in backend processes corresponding requests via URL. The figure 5 shows the logic clearly.

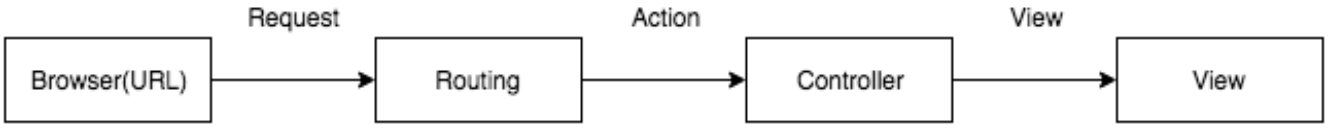

figure 5

I listed all URLs in MarkUp below:

Document:

```
'post   /api/doc':     'DocumentController.post',
'get    /api/doc':     'DocumentController.list',
'put    /api/doc/:docid': 'DocumentController.put',
'delete /api/doc/:docid': 'DocumentController.delete',
```

Version:

```
'get    /api/doc/:docid/version': 'VersionController.list',
'get    /api/doc/:docid/version/:vid': 'VersionController.get',
'post   /api/doc/:docid/version': 'VersionController.post',
'put    /api/doc/:docid/version/:vid': 'VersionController.put',
'delete /api/doc/:docid/version/:vid': 'VersionController.delete',
```

User Authentication:

```
'post   /api/auth/login': 'AuthController.login',
'post   /api/auth/logout': 'AuthController.logout',
'post   /api/auth/register': 'AuthController.register',
```

User_document:

```
'get    /api/:userid/docs':     'DocumentController.list_user',
```

User_version:

```
'get    /api/:userid/docs/:docid/version': 'VersionController.list_user',
'get    /api/:userid/docs/:docid/version/:vid': 'VersionController.get_use
'post   /api/:userid/docs/:docid/version': 'VersionController.post_user',
'delete /api/:userid/docs/:docid/version/:vid': 'VersionController.delete_
```

For routers above, I divided into five parts so that it is easy to let developers understand quickly. Now, let me explain URLs above, for example:

```
'get    /api/doc':     'DocumentController.list',
```

For the code above, when user enters ip:port/api/doc in browser, backend will find list function in DocumentController to do login operations.

```
'get    /api/doc/:docid/version/:vid': 'VersionController.get',
```

Compared to previous one, this router is more complicated. There have two more parameters, :docid and :vid, which are passed into get function in VersionController in order to do some logic operations.

Controller Design

Controller is responsible for processing incoming requests, handling input, saving data and sending a response to send back to the client. In sails.js, web server will normally map incoming URL requests directly to files on the server. In controller class, processing incoming requests will map corresponding RESTful API method to processing login operations.

If you want to see controller code in MarkUp, you can find corresponding code in path MarkUP_WebSerivce/webservice/api/controllers.

Authentication

There have many authentication methods, I chose WaterLock. The logic about how authentication works is showed in figure 5. Without authentication, after user sends request from browser, URL will map restful API method in corresponding controller, with authentication, incoming requests from user have to be checked by WaterLock before incoming requests arrive at controller.

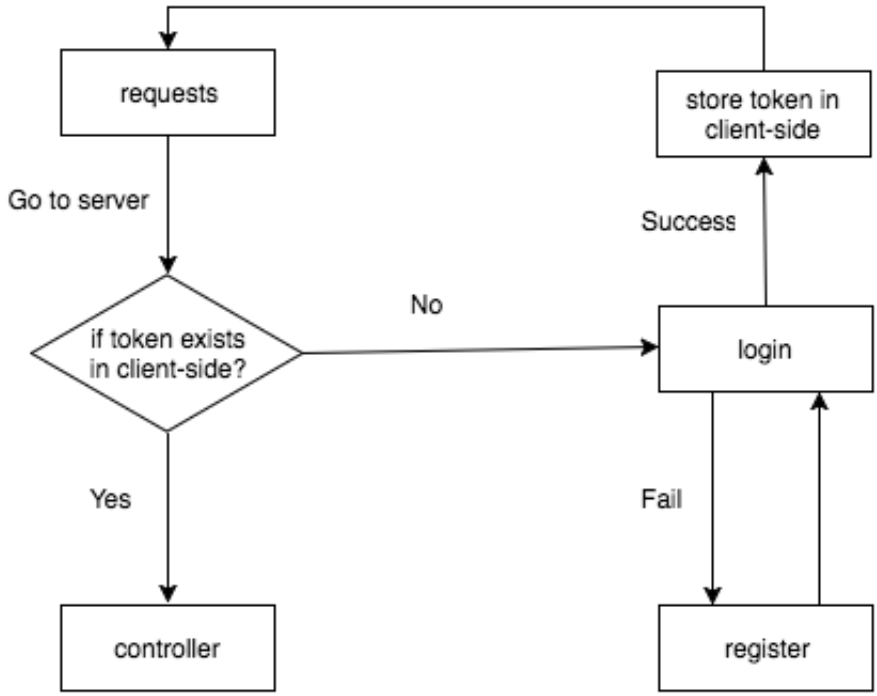

figure 6

If there have token in client-side, request will go to controller to execute corresponding operations, otherwise, user has to login first. If user logs in successfully, server will sent token to client-side, client-side will store token in browser or other user computer and then send request again to server, otherwise,user has to register and back to login. In client-side, it is different between usual authentication method and WaterLock, WaterLock puts token in header. When user sends request via browser, token information is sent to server.

About how to implement this in project, I will explain it with code. The code section below is part of authentication in backend, you could find it in MarkUP_WebSerivce/config/policies. This section means, if you want to access post_user, put_user, delete_user, get_user and list_user methods in VersionController, you have to do authentication. It is same with DocumentController part.

However, there are only part of methods in controllers to check if user logged in or not. Why I don't set all methods in controllers to do authentication? Reason is simple, for reader, they can access parts of information in MarkUp without logging in.

```
VersionController: {
  post_user: ['hasJsonWebToken'],
  put_user:['hasJsonWebToken'],
  delete_user: ['hasJsonWebToken'],
  get_user:['hasJsonWebToken'],
  list_user:['hasJsonWebToken'],
},

DocumentController: {
  get_user:['hasJsonWebToken'],
  list_user:['hasJsonWebToken'],
},
```

Front-end Design

I mentioned in previous section, android device is viewed as view part in MVC framework, so android development means front-end development.

Layout and Activity

When you make an Android Application, the first thing you will do is create an activity.

These are where all the action happens, because they are where all action happens, because they are the screens that allow the user to interact with your app.

Another incredibly important part of building an Android application is creating a layout that the users of the application interact with.Android Studio offers an advanced layout editor that allows you to drag-and-drop widgets into your layout and preview your layout while editing the XML. Within the layout editor, you can switch between the Text view, where you edit the XML file as text, and the Design view. Just click the appropriate tab at the bottom of the window to display the desired editor. If you want to know more, you could go to http://developer.android.com/sdk/installing/studio-layout.html.

Here, I won't say a lot of details about how to create an android application. If you have taken ITMD555, it is easy to understand MarkUp.

AsyncTask

AsyncTask is an abstract class provided by Android which helps us to use UI thread properly. This class allows us to perform background operations and show its result on the UI thread without having to manipulate threads.

Android implements single thread model and whenever an application is launched, a thread is created. Assuming we are doing network operation on a button click in our application. On button click a request would be made to the server and response will be awaited. Due to single thread model of android, till the time response is awaited our screen is non-responsive. So we should avoid performing long running operations on the UI thread. This includes file and network access. To overcome this, we can create new thread and implement run method to perform this network call, so UI remains responsive.

But since Android follows single thread model and Android UI toolkit is not thread safe, so if there is a need to make some change to the UI based on the result of the operations performed, then this approach may lead some issues. So the android framework has given a very good patten which is enveloped into AsyncTask.

In MarkUp, there is a need to communicate with the server and get data from it, so I used AsyncTask to perform many times long running operations on the UI thread, otherwise, user experience will be very bad. For every AsyncTask in MarkUp app, it will map route I designed.

AsyncTask has four steps:

- doInBackground: Code performing long running operations goes in this method. when onclick method is executed in click of button, it calls execute method which accepts parameters and automatically calls doInBackground method with the parameters passed.
- onPostExecute: This method is called after doInBackground method completes processing. Result form doInBackground is passed to this method.
- onPreExecute: This method is called before doInBackground method is called.
- onProgressUpdate: This method is invoke by calling publish progress anytime from doInBackground call this method.

SharedPreference

Actually, android provides several options for user to save persistent application data.

- SharedPreference - store private primitive data in key-value pairs
- Internal Storage - store private data on the device memory
- External Storage - store public data on the shared external storage
- SQLite Database - store structured data in a private database
- Network Connection - Store data on the web with your own network server

Considered this application specific needs, I choose sharedPreference because MarkUp only needs to save small collection of key-values. For more details, please access to http://www.compiletimeerror.com/2015/02/android-shared-preferences-example-and-tutorial.html#.Vy0QpKMrJnY

MarkDown libraries

The important part in this app is converting unstructured text into formatted text. I used MarkDownView library to implement this functionality.This library is very well and easy to learn, let's see what's MarkDownView. MarkdownView (Markdown For Android) is an Android library that helps you display Markdown text or files (local/remote) as formatted HTML, and style the output using CSS. The MarkdownView itself extends Android webview and adds the necessary logic to parse Markdown (using MarkdownJ) and display the output HTML on the view.

Syntax I implemented are below:

- header
- italic and bold
- indent
- list

Specific syntax introduction could be accessed in https://daringfireball.net/projects/markdown/syntax, or refer to figure 17.

Achievement

Home Page

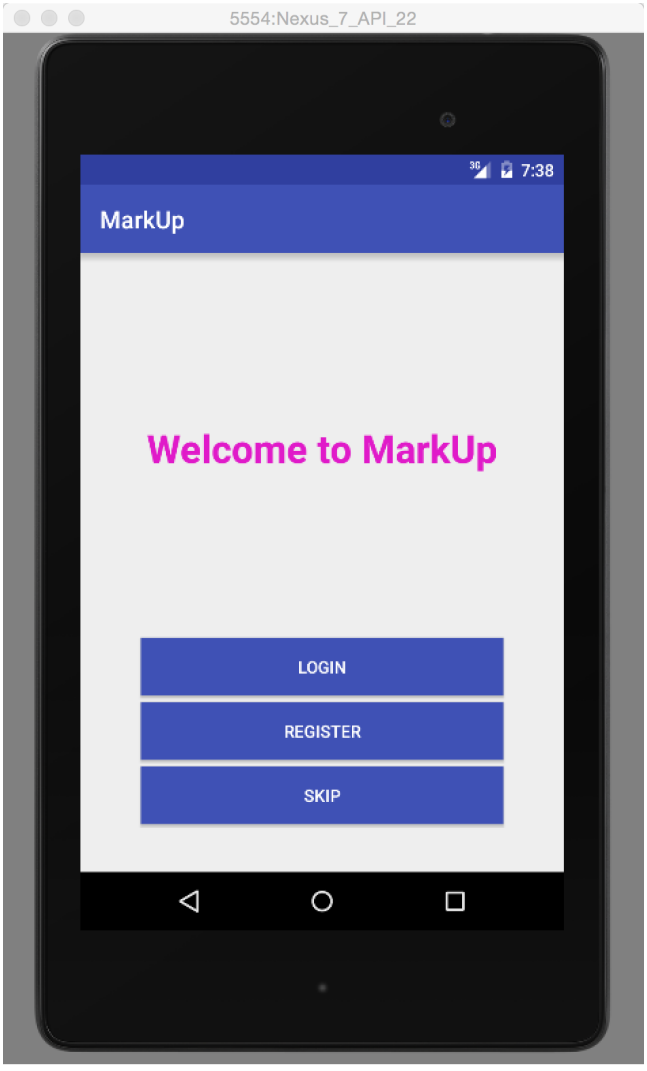

figure 7

Any user type can access home page.

Document List

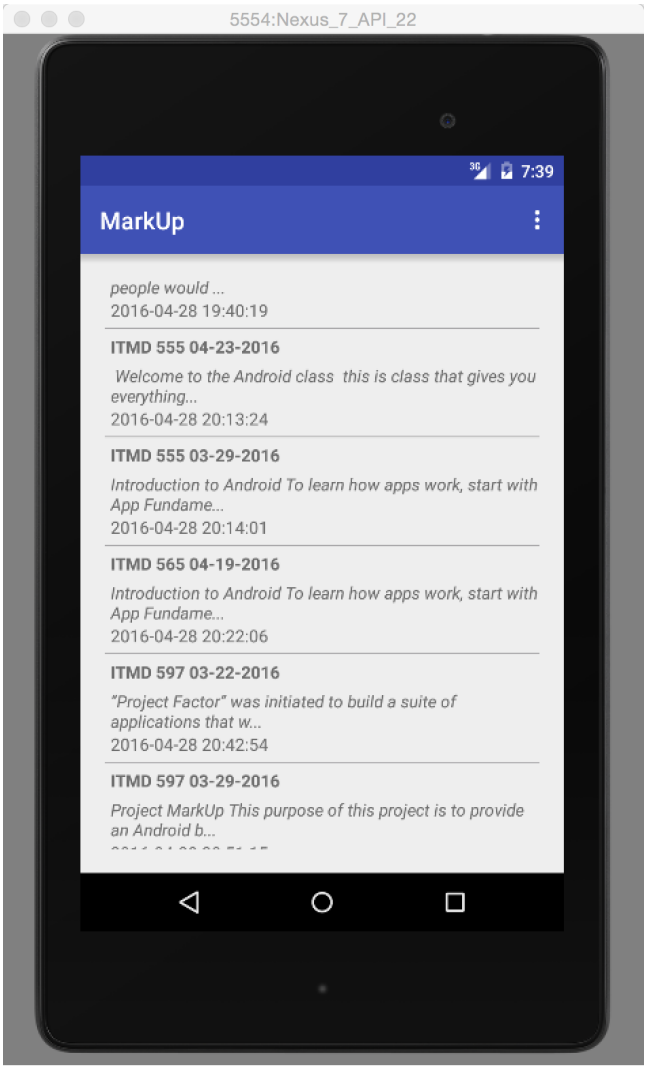

figure 8

Document list is shared for every one. After you click skip button in Home page, you can arrive at this page.

Version List for every document

figure 9

Version list is shared for every one. After you click one item in document list, you can arrive at this page.

Register

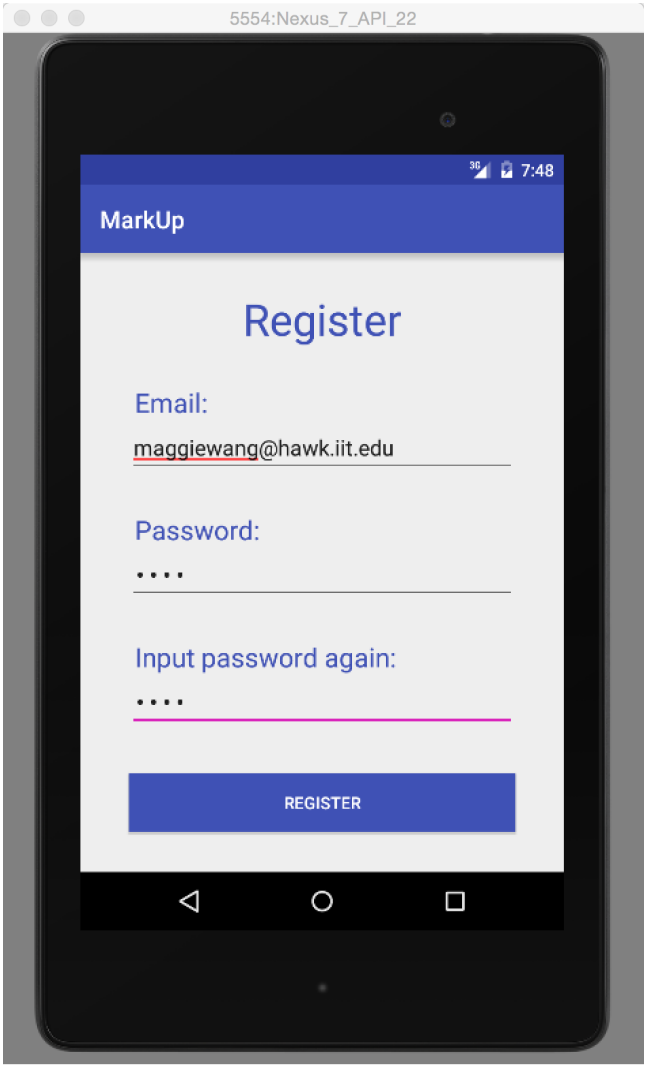

figure 10

After you click register button in Home page, you will go to this login page.

Login

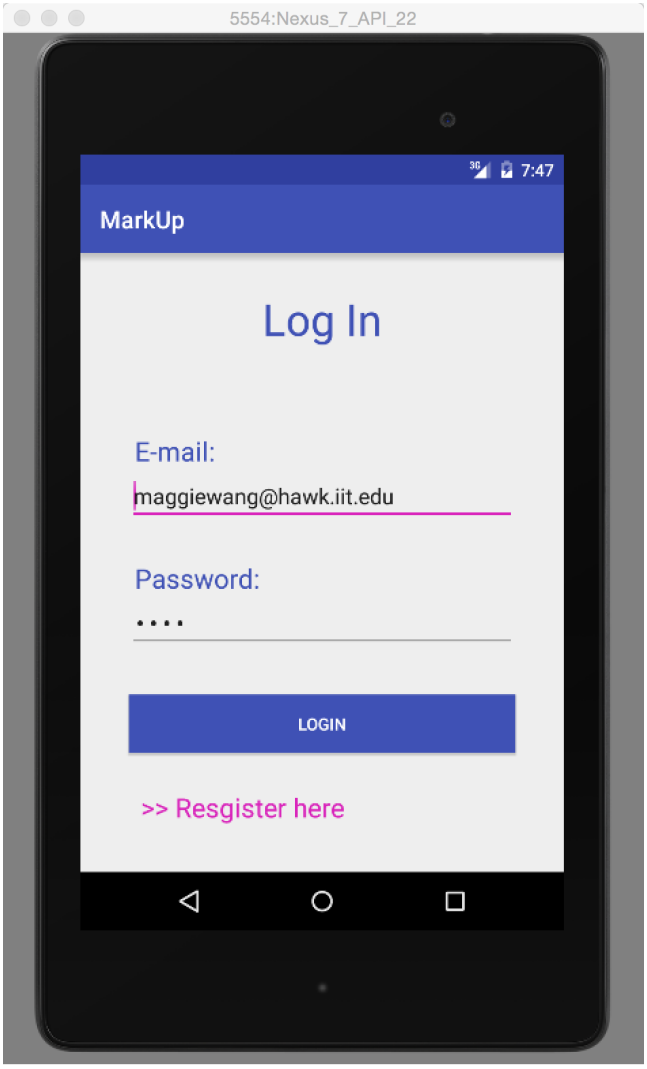

figure 11

After you click login button in Home page, you will go to this login page.

Dropdown List

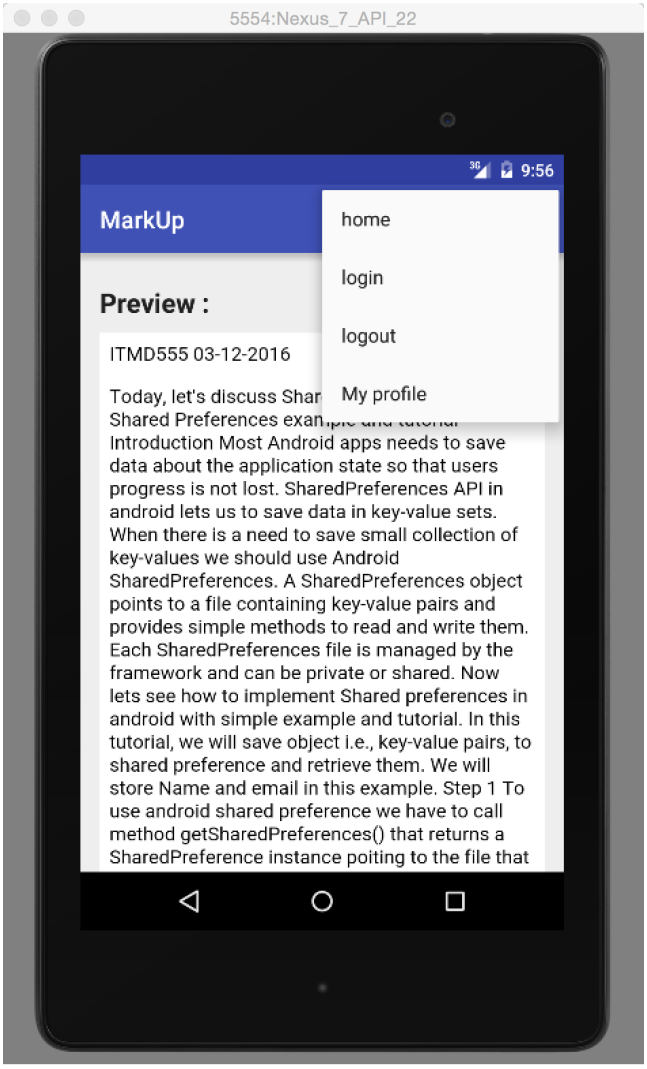

figure 12

User is able to access his own profile and intent to other pages.

Documents List in user profile

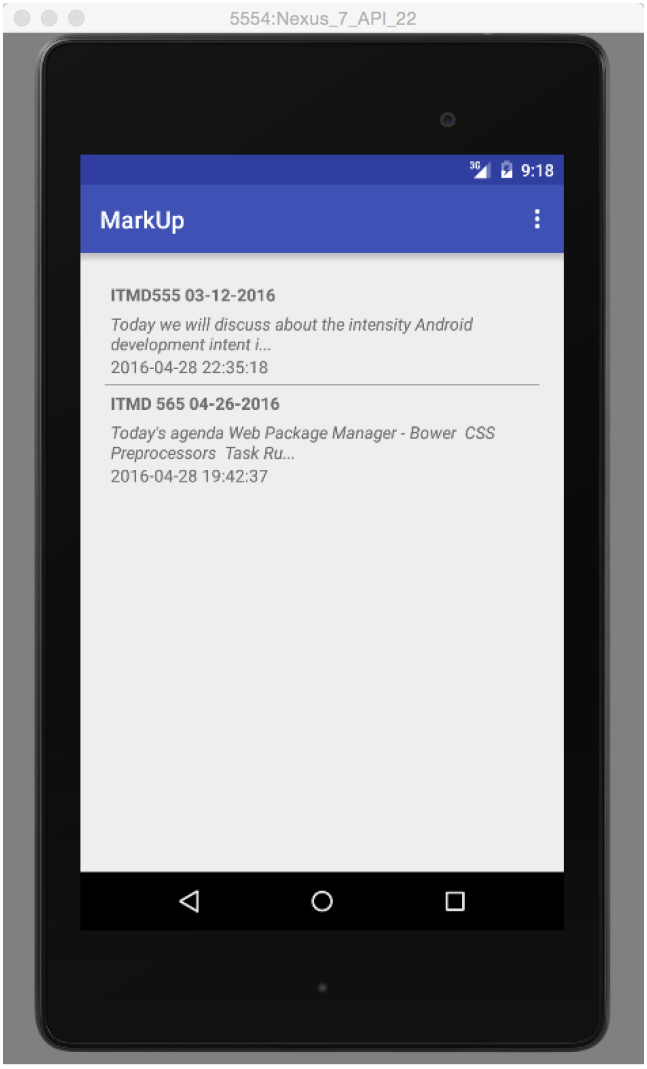

figure 13

Document list in user profile is only accessed by user themselves and only displays documents that user saved before. After you click my profile button in dropdown list showed in figure 12, you can arrive at this page.

Version List in user profile

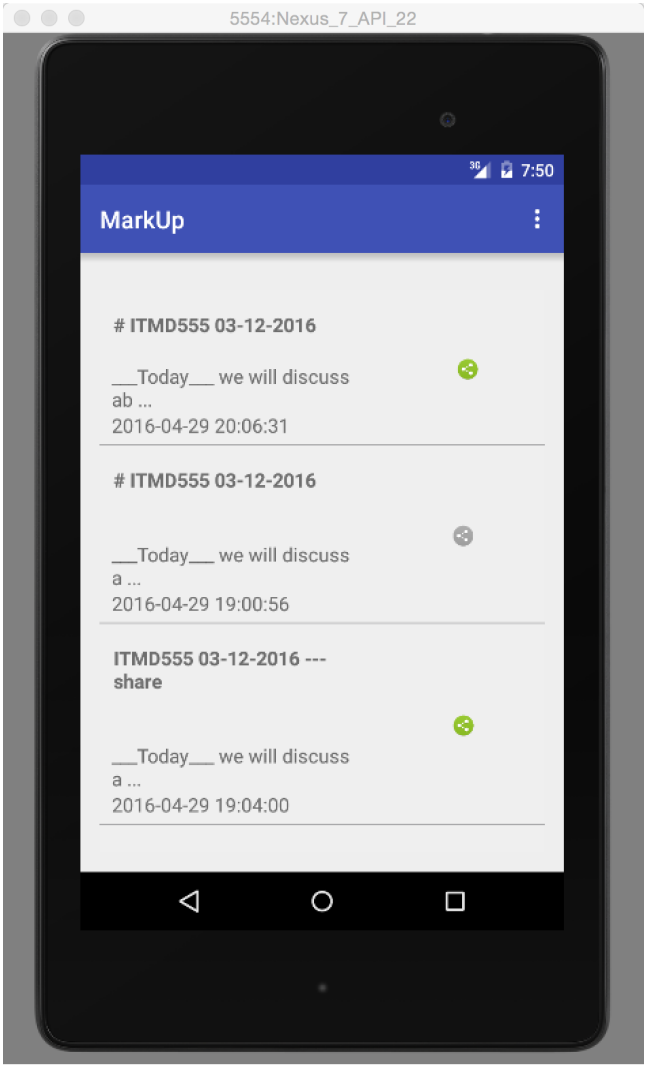

figure 14

Version list in user profile is only accessed by user themselves. After you click one item in document list, you can arrive at this page.

User authority

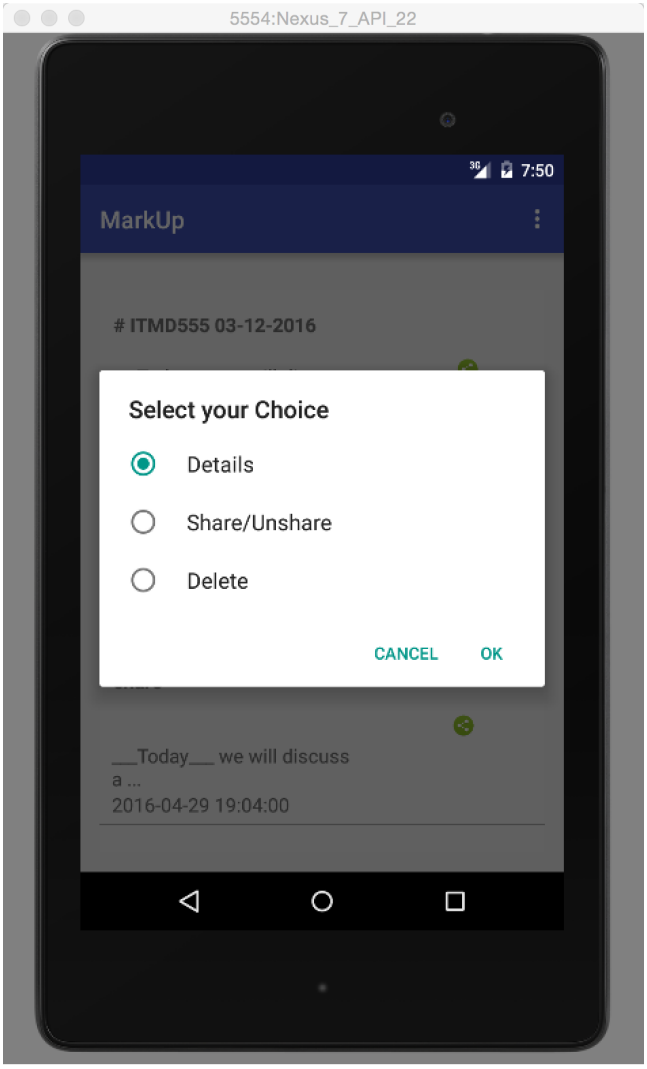

figure 15

User can chose any options showed in figure 15. Clicking Details can jump to MarkDown editor page showed in figure 16. Clicking share/unshare can change version state, if version is at shared state currently, version state will become unshared state after clicking, green icon will become grey, if version is at unshared state currently, version state will become shared state after clicking, grey icon will become green. Clicking delete is able to deleter this version.

Unstructured Text

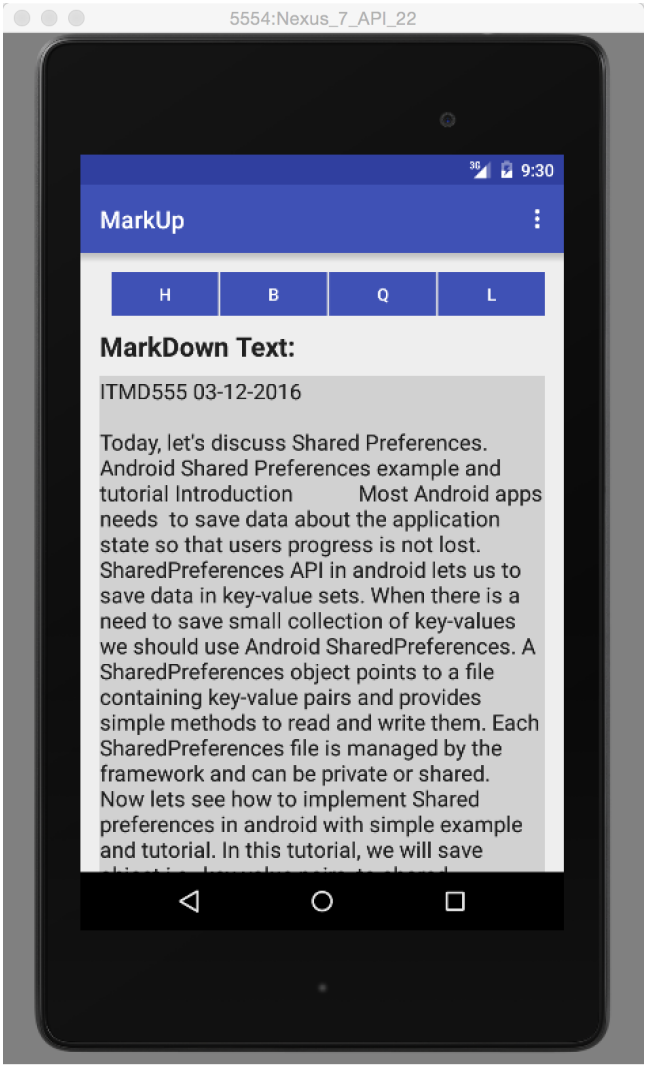

figure 16

This page is MarkDown editor. In this page, user is able to drag and drop buttons into text so that user can edit unstructured text.

MarkDown Preview without editing and MarkDown Preview with editing

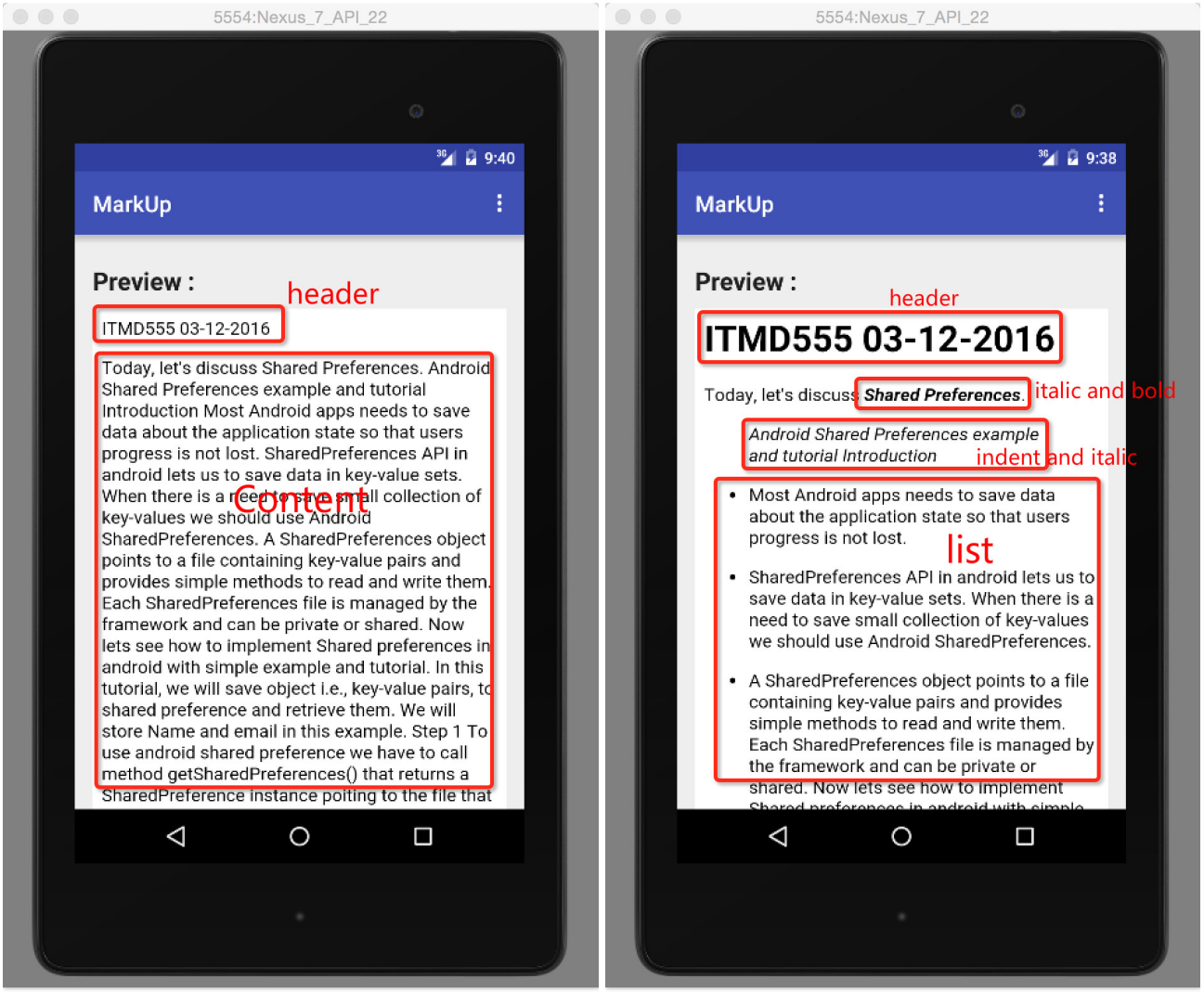

figure 17

Compared with two screenshots in figure 17, four syntaxes in MarkUp is easy to understand.

Work in the future

- Add multi-user type to management raw data, such as Administrator and owner
- IIT based user authentication
- User metrics tracking and reporting
- Export PDF and ePub document format
- Do better UI
- Integrating MarkUp with other components together

Conclusion

During developing MarkUp application, I met various problems. Most of problems I met were not so difficult to solve, like override format of MarkDown syntax, authentication problem and android AsyncTask utilization. I solved them via google online and ask professor. The most impressive problem I met is multi-user synchronous editing capabilities. This problem took me a few days to come up with solution. Actually, I didn't implement that multi-user could edit same document at the same time, but I implemented this functionality in user view. What I did is that let user save the document they want to edit to their own profile. For one document multi-user is editing, it seems that multi-user edit same one, however, they are editing different documents that have same content.

I implemented most of requirements and functionalities of this project. By developing this application, this project not only practiced my technical skills in Android development, but also improved my independent study skills. Hope there have other students who are interested in this project and are able to continue my work. Finally, I am looking forward to integrating with other components of Factor project together, and let Factor project become a truly product so that it can be published in market.