iOS Applications
Based On Conference Navigator

Independent Study Report
INFSCI 2950
Fall 2013

Author: Jin Gan
Mi Lou
Di Ai

Supervisors: Prof. Peter Brusilovsky
PhD. Xidao Wen

Introduction

Conference Navigator is a personal conference scheduling tool with social linking and recommendation features. Users can control access to their information in Conference Navigator and have very good user experience in it.

This website made a great contribution on collecting the conferences together with every detail of the conferences, as well as arranging the calendar for every user to attend the conferences. Moreover, the authors' list showed all related authors and conferences under them. Users could make friends with those authors and take attention on their conferences. What's more, it would provide some recommendations to users according to their reading records and rating provided by users.

With the development of the website and the convenience users' require, mobile application became more practical for users. Therefore, we paid our effort on it.

Goals

The independent study is worked based on the old app run on iOS 3. During the summer semester, to accomplish several conference's app, the group first mission is to learn the programming method of iOS, the old app's logic, and find the differences between iOS 3 and iOS 6, and based on the iOS 3's version to make correction so that the application can run on iOS 6.

During the configuration, we found that the old code and third-party packages are not running well on iOS 6 because some detailed part is either right or wrong for the new complier. So, after the release of iOS 7, our group decided that all the codes and programs logic should be rewritten.

And in the fall semester, we learned the UI regulation of iOS 7 and new methods on iOS 7, and reprogramed the application based on the old version with some promotion, and try to document all the information we know, so that the application can go further and improvement can be easier to make.

Technical Details

A. Development Environment and Techniques

1. iOS 6, iOS 7
2. SQLite Manager on Firefox
3. Xcode 4, Xcode 5
4. NaviCat

B. Emulation Environment

Xcode iOS simulator and iPhone

C. Technical Detail

To check the data on the sever side, by connecting via SSH on any SQL workbench is available by given account. For editing the program, our group use Xcode. In the app, we used several third-party packages. (Details in Appendix)

D. Source Link

1. Conference Navigator official website:

   http://halley.exp.sis.pitt.edu/cn3/portalindex.php

2. EC-TEL 2013 iTunes page:

   https://itunes.apple.com/us/app/ec-tel-2013/id694446202?mt=8#!

General Description

In general, the application is divided into five sections, Conference (ConferenceViewController.m), Proceeding (ProceedingViewController.m), People (AuthorListViewController.m), My Program (MyProgramViewController.m), Recommendation (Recommendations.m).

At the Conference page it is the general schedule of the conference by day (by difference api the conference might be different). By typing into the selected day, (SessionListViewController.m), all the session will be divided by time period, and type into the selected session, there will be the content.

At the Proceeding page divided into three tags, to list all content into three kinds of ways: by author, by title and by type. And during the development, we were planned to build the search function, but due to we have to first solve the problem of local storage, we found we got no more time on fixing the search function, which I found that a good project management should be make next time.

People page is to list all the people's name in case users want to find the paper of specific person. Also, if you have log in, you can see your friends.

In the My Program pages, the app provide link to view the content saved in reading list and the content make as schedule. And because there was a recommendations link on it, we haven't removed it for future designing.

In the Recommendation pages, there are contents the system provided, and after typing into it, at the up-right corner, we set a rate button to rate the relevance.

A. Story Board

Initialize view of start is *UITabbarViewController*, nesting five *NavigationController*. Use the Navigation Controller to release multilevel views.

1. **Conference**

The Conference interface lists current Conferences, its controller is *ConferenceViewController*. By clicking the item in the Conference interface, it will get into the Session table. Conferences in the session list are ordered by the time zone. The view controller of the Session list is *SessionListViewController.* Click the item in the Session list would get into the session interface, which lists all related contents. Its view controller is *SessionDetailListViewController*. When clicking the item in the Content list, it gets into the Content detail. Its view controller is *ContentDetailViewController*. We could see all the detail in the Content interface. Moreover, we could see the list of all the authors. When clicking specified author, it goes into the author detail interface which view controller is *AuhtorDetailViewController*.

2. **Proceeding**

Proceeding interface lists all the Content data, it could be viewed by author, title or type. Its controller is *ProceedingViewController*. By clicking the item in the list, it goes into the interface is *ContentDetailView*, which is the same as by clicking the *SessionDetailListViewController* .

3. **People**

The people interface lists all the authors and friends made by users. By clicking the author in author list, it goes into the detail about the specified author. Its view controller is *AuthorDetailViewController*. At this list, it shows all the papers related to this author. It goes into the Content interface by clicking the paper under the author.

At the Friends interface, it would check the user whether logged in or not to make sure it would show the friend of the specified user. If it already logged in, it would show the friends directly, or it would goes to the Log-in interface. The view controller of Log-in interface is *LoginViewContorller*.

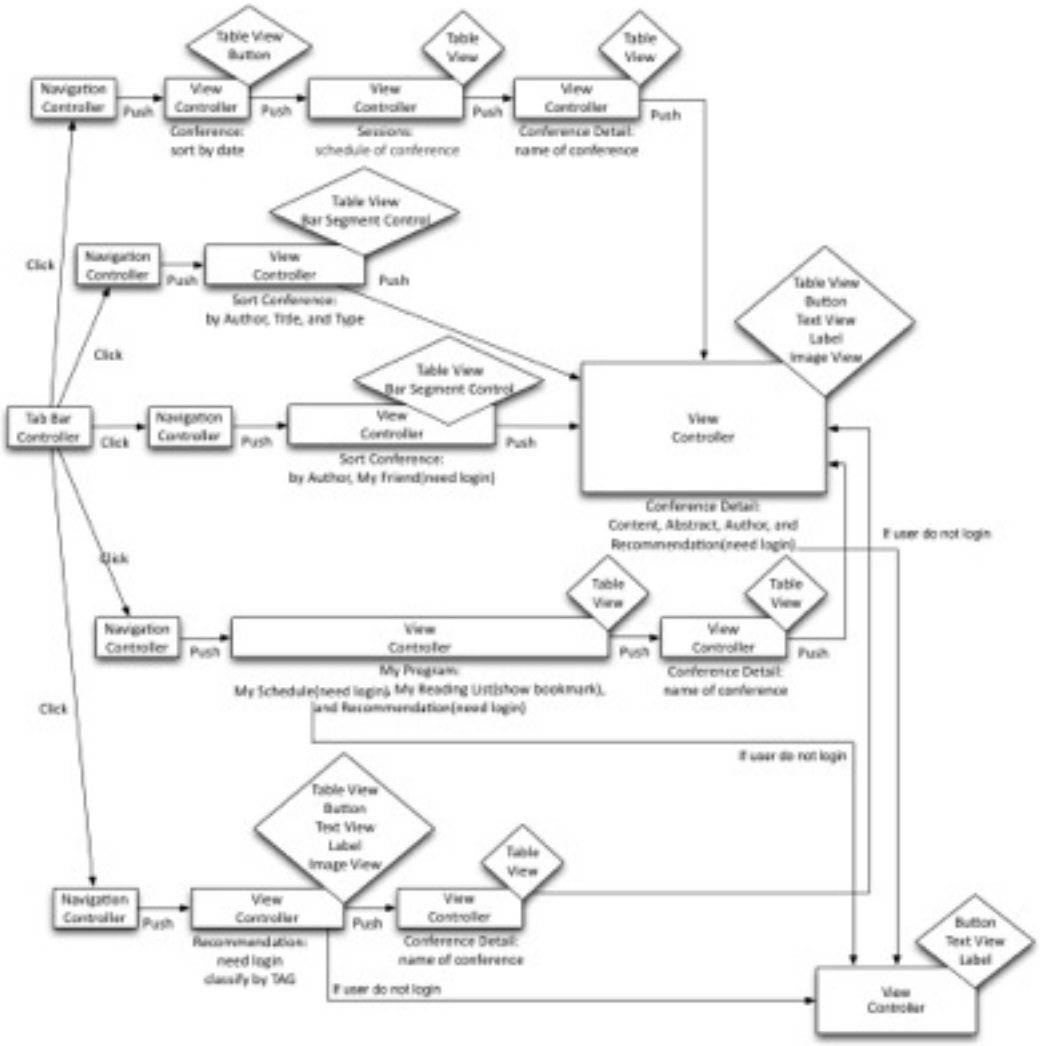
4. **My Program**

My Program interview's view controller is *LoginViewContorller*. It has three options, My Schedule, My Reading list and Recommendations. The view controller of My Schedule is *MyScheduleViewController*. The view controller of My Reading List is *MyReadingListViewCOntroller*. For these two above, user should login first. By clicking the item in them, it goes into the corresponding content. The Reccomendations shows the conference from the website. It also requires the user to login first.

5. **Recommendation**

(The same as the Recommendation in My Program)

B. **Framework**



The main five Controllers' code file structure based on framework (find in folder 'controller'):

1. ConferenceViewController –> SessionListViewController –> ContentDetailViewController –>
[ContentSummaryCell + AuthorDetailViewController](consist into the last page)

2. ProceedingViewController

3. AuthorListViewController

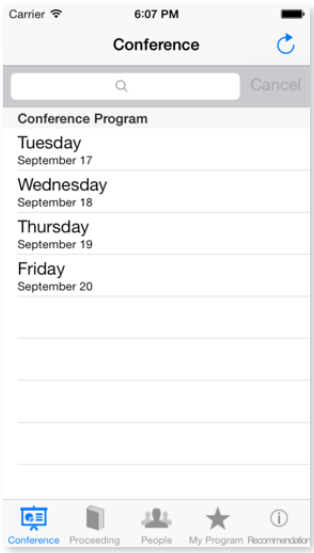
4. MyProgramViewController –> [LoginViewController + SignUpViewController] –>
[MyScheduleViewController + MyReadingListViewController + Recommendations](three options)

5. Recommendations

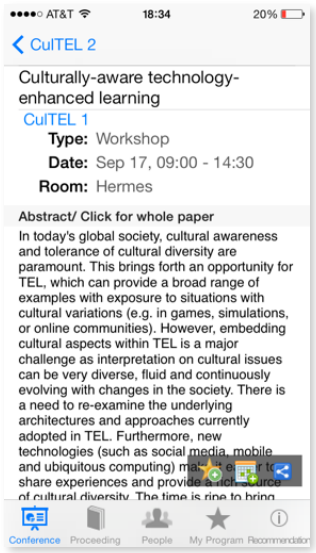
Use the Tab Bar controller to display the five parts of function, Conference, Proceeding, People, My Program and Recommendation.

Functions in Detail

1. Conference



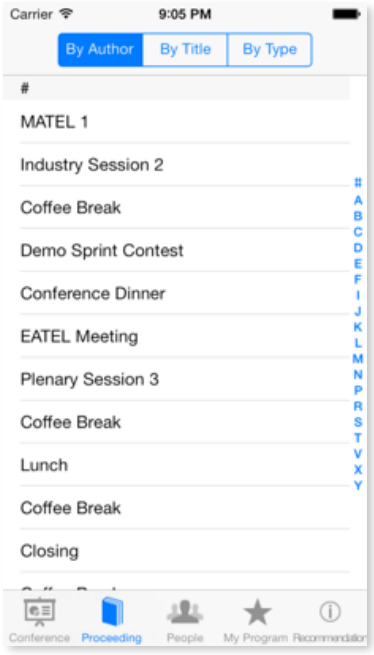
There is a searching bar on the top for searching all the items.



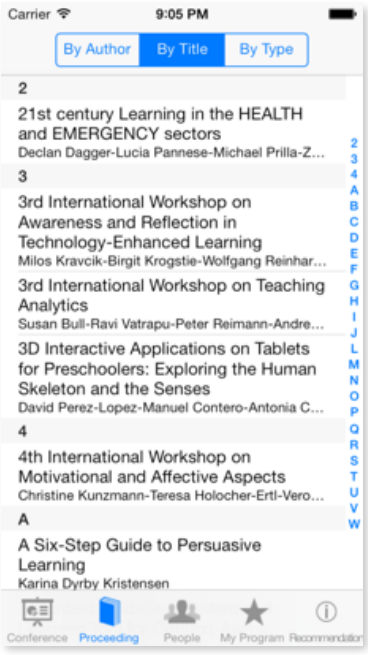
After click into the specified conference, there are three buttons showed up down the right corner. By clicking the star button, the conference would be saved to the “My Reading List” and would be

2. Proceeding

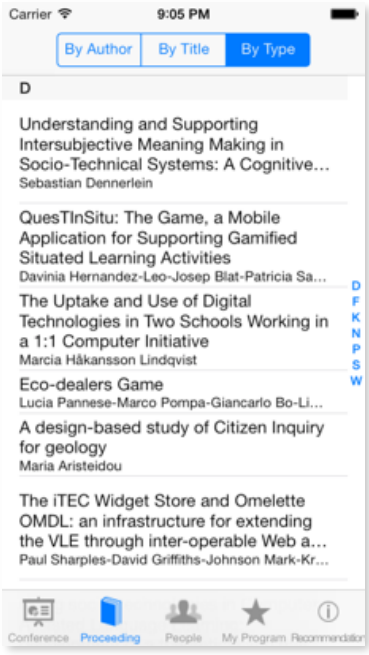
We list all the conferences by the following filter:



By Author

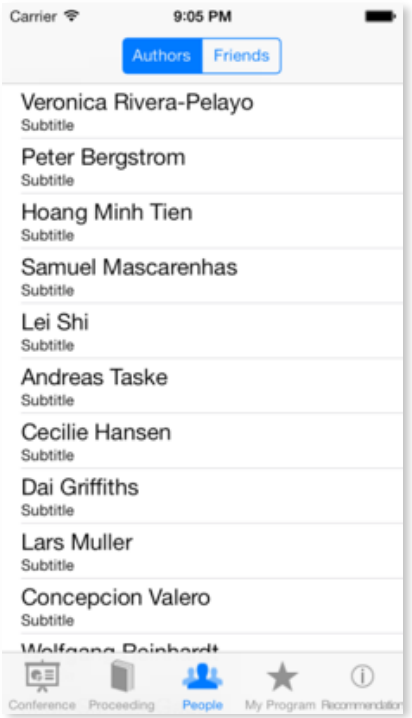


By Title

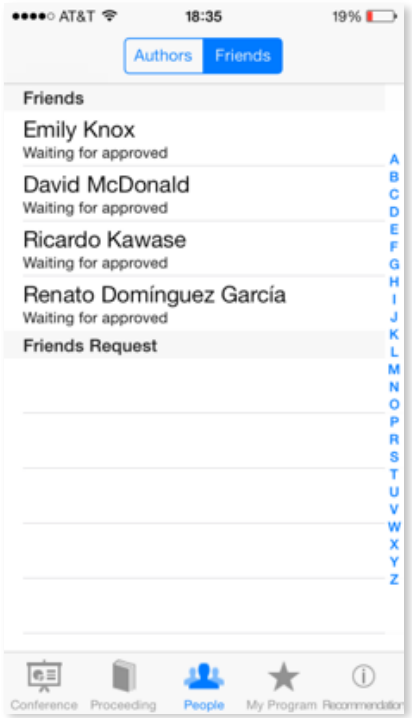


By Type

3. People

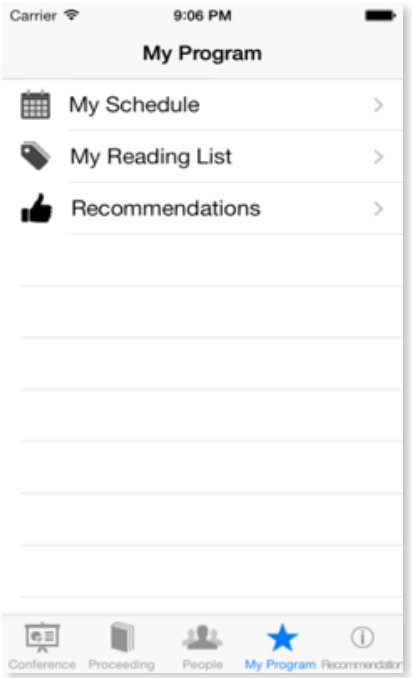


It includes all the authors related to the Conference Navigator.



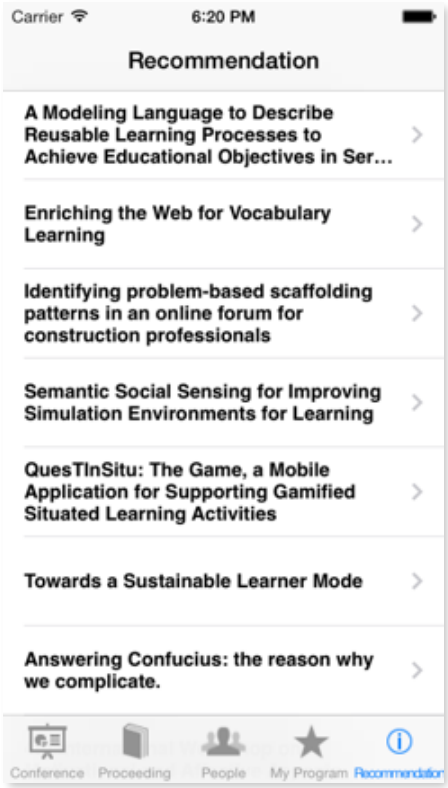
Users could make friend with the authors related to conferences after applying to be friends.

4. My Program

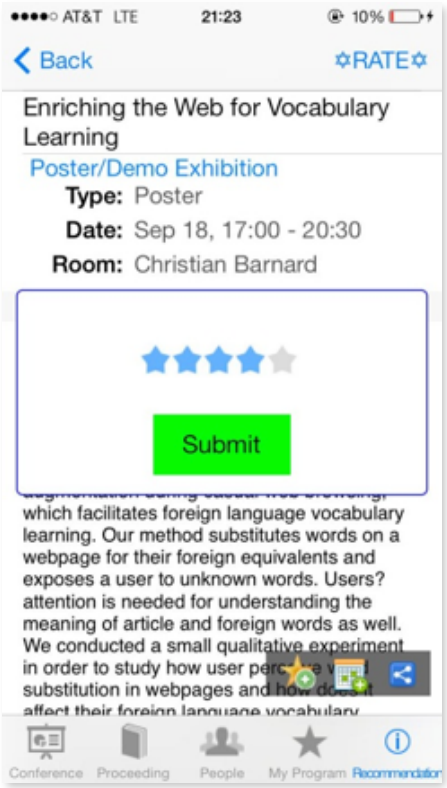


The conference scheduled by user who login would be save to “My Schedule”.
The conferences users liked would show on the “My Reading List”.
“Recommendations” lists those conferences return from the website.

5. Recommendation

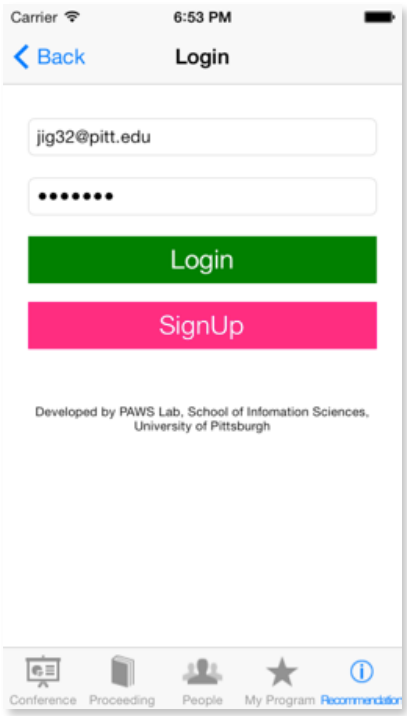


The same as the Recommendations in My Program.



Since we need to do the recommendation due to the rating records provided by the users. We add a rating button on the right top of the interface.

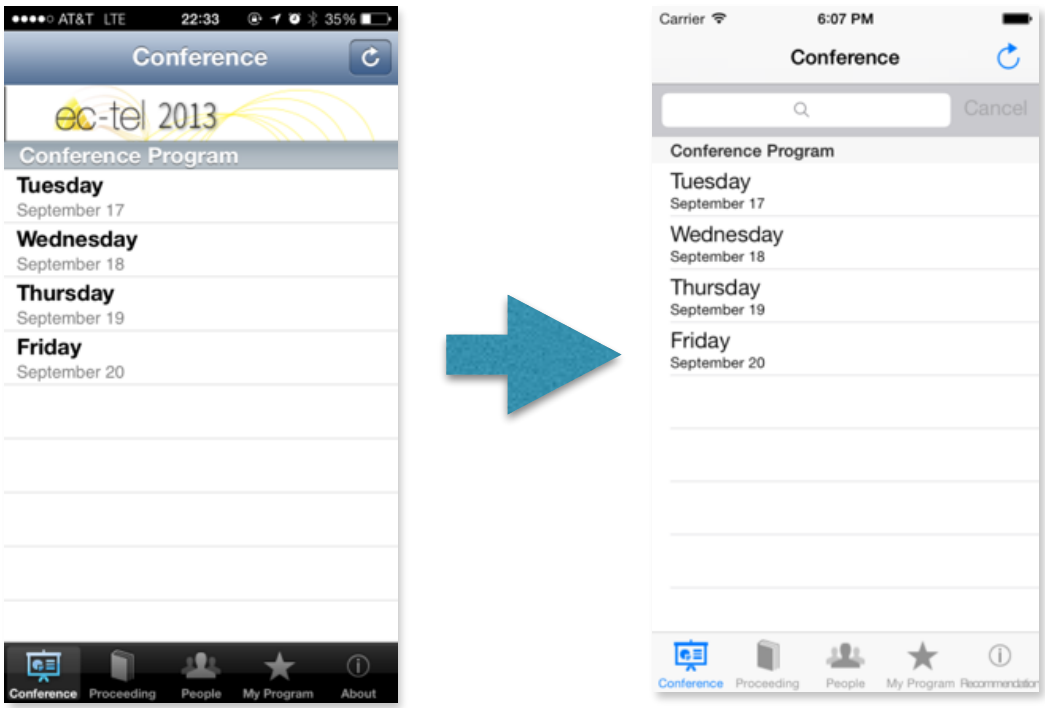
6. Log-in



It would show up when using the “My Schedule” and “My Reading Lists” under “My Program”.
Since every user have their own schedule, user need to login to save their own message.

Compare with old version

1. **Conference :**

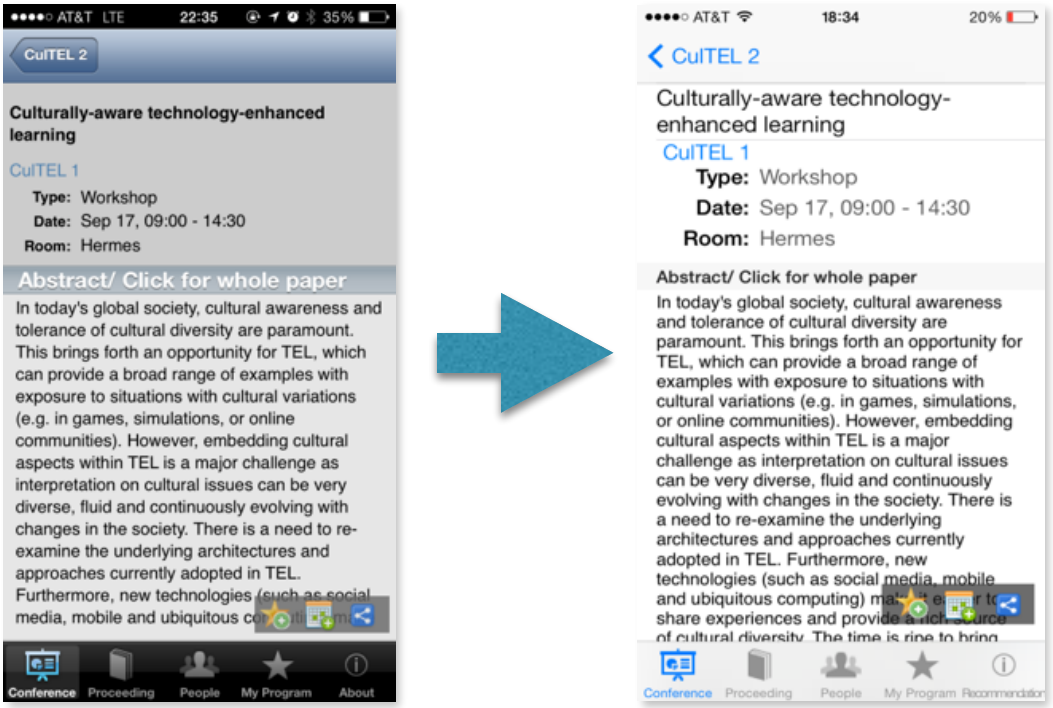


Remove the logo, add search bar

2. Conference detail - Session :

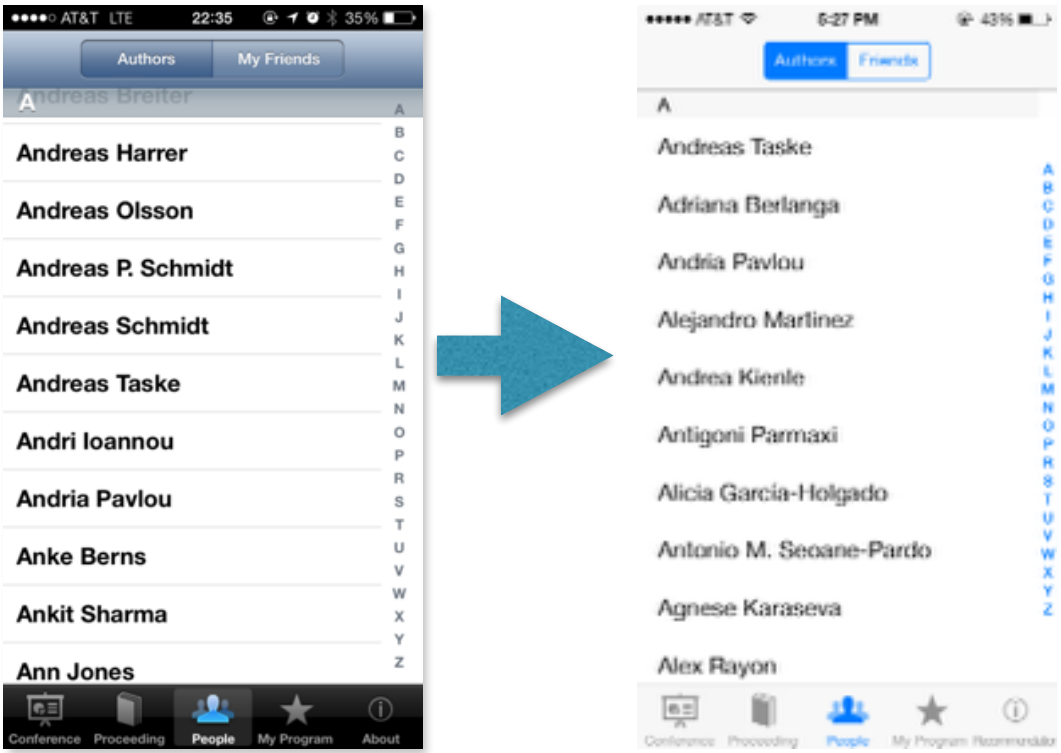
Adapt to iOS 7's style

3. Conference content detail :

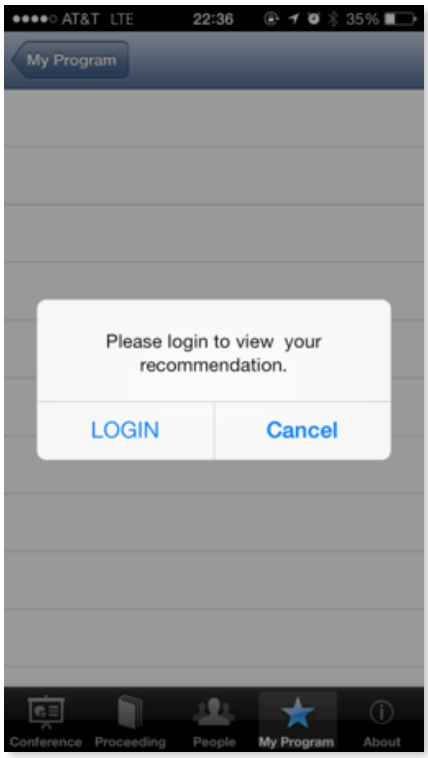


Fonts style or arrangement changed

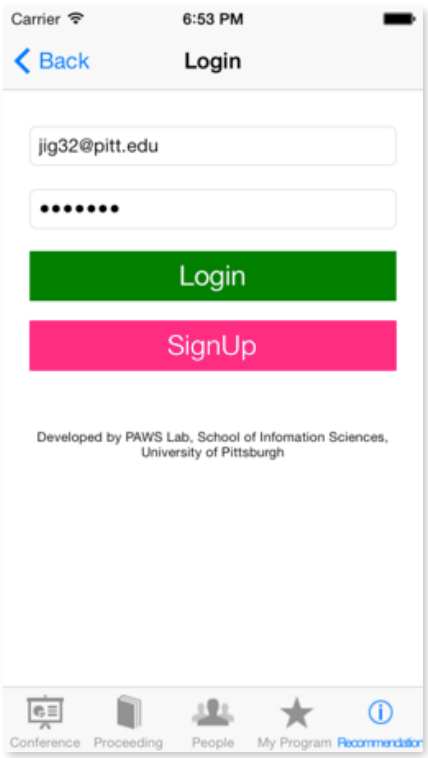
4. Author Tag:



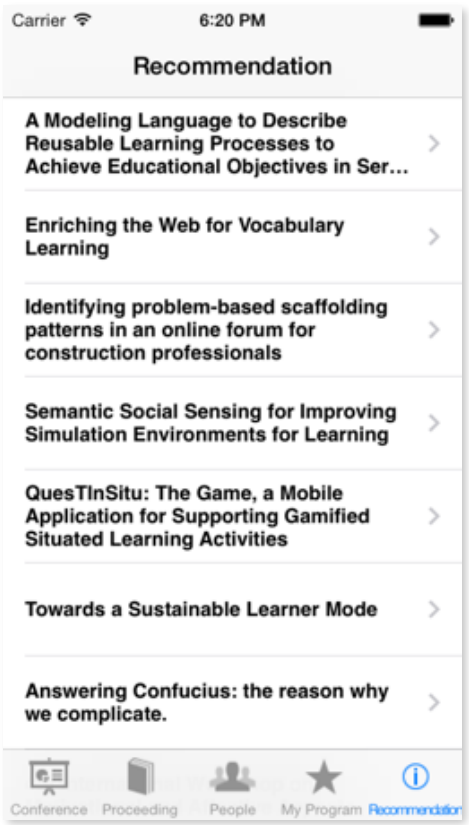
Adapt to iOS 7's style



5. Login **view :**



Re-design Login window changed



Canceled the about page and let the recommendation page get into tag bar

Conclusion and Future Work

A. **Limitation**

As our team don't have too much experience on building iOS app, our work is bases on the guide line of Apple's iOS development center, comparing to other similar application, the functions and UI might looks simple. But during the development we found that the importance of server side, by fetching different data from different tables. The work is harder to figure out the logic behind the application. We hope our work would update the foundation of the app for further development. And make it easier for beginner to adapt to.

About the functions of the app, we paid more attention on the recommendation session and successfully add rating function to it. But also, we failed in building multiple tags. Also, we added search function to the app to more easily found the content user might want to have. Though the search function is made, but due to the limited time in fall semester, we found that the key word of search must be the start of original title, which means the search function is kind of inflexible.

The interaction of the app is still need improvement. Like we have built rating function to input user's preference, but actually we have not think about the feedback from it. Also as Professor advised, feedback for Add To Reading List or Add to Schedule can be make. Also, by these improvements, the accessibility will be better. The other controversial function is about the Logout, we think unless we successfully create a multiple conferences application, the Logout function is not that helpful. The other possible interaction way is to add friend on the CN as we can do on the web site. Also, many of the operation should have feedback once user typed it.

B. **Future work**

1. Building a multiple conferences application. A multiple conference can be made so that we can concern more about the User Interaction design. But before that, we need to make sure to sever side is functioning well, and clear the code for fetching data, and local storage(which we have done during the independent study, but a second look should be make always).

2. Collecting API for possible function, though our group sometime think many functions, but we always struggling with the API to apply the functions. So make continuous communicate with the people building the sever is always needed and important.

3. Take more advantage of web-mobile technic to develop the app so that the application can be more interactive.

Appendix

A. Code Explanation

1. **Models**

Program includes 9 models, User, Session, Event, Content, Presentation, Author, RecommenItem, RecommendationItem and BookMarkItem.

2. **ViewController**

(Described in Story board.)

3. **CN3Config**

This file saved all the internet APIs' address setting information, for the convenience of modifying and quoting. It defines API's basic address.

```
#define BASE_URL @"http://halley.exp.sis.pitt.edu/cn3mobile/"
```

4. **DataManager**

It controls all the exchanges of data. It downloads the data from internet with parsing them and saves them to the local.

Methods in detail:

- Rating corresponding Content. The first parameter is the number of Content, the second one is the rate.

```
-(void)rateContentOfId:(NSNumber *)contentId forValue:(int
  )value{
    NSString* url=[NSString stringWithFormat:RATE_URL,self
      .loginuser.userid,contentId,@(value)];
    AFHTTPRequestOperationManager *manager = [
      AFHTTPRequestOperationManager manager];
    manager.responseSerializer = [
      AFHTTPResponseSerializer serializer];
    [manager GET:url parameters:nil success:^(
      AFHTTPRequestOperation *operation, id responseObject
      ) {
        NSString* str=[[NSString alloc] initWithBytes:[
          responseObject bytes] length:[responseObject
          length] encoding:NSASCIIStringEncoding];
        NSLog(@"%@",str);

    } failure:^(AFHTTPRequestOperation *operation, NSError
       *error) {
        NSLog(@"%@",error);
    }];
}
```

- Uploading the Content list recommended by users. After be downloaded successfully, it would call the oncoplete method automatically and send the list.

```
-(void)loadRecomendataion:(void (^)(NSArray *recomends))
  oncomplete{
    User* u=self.loginuser;
    NSString *url=[NSString stringWithFormat:
      RECOMENDATION_URL,u.userid,self.conferenceId];
    AFHTTPRequestOperationManager *manager = [
      AFHTTPRequestOperationManager manager];
    manager.responseSerializer = [
      AFHTTPResponseSerializer serializer];
    [manager GET:url parameters:nil success:^(
      AFHTTPRequestOperation *operation, id responseObject
      ) {
        NSString* str=[[NSString alloc] initWithBytes:[
          responseObject bytes] length:[responseObject
          length] encoding:NSASCIIStringEncoding];
        NSArray* recomens=[self
          parseUserRecomendataionItems:str];
        if (oncomplete) {
            oncomplete(recomens);
        }
    } failure:^(AFHTTPRequestOperation *operation, NSError
       *error) {
        if (oncomplete) {
            oncomplete(@[]);
        }
    }];
}
```

- Acquiring the Reading List of all users.

```
-(NSDictionary *)getMyReadingList{
    return self.allMyreadingList;
}
-(BOOL)isInReadingList:(NSNumber *)contentid{
    if (self.allMyreadingList[[NSString stringWithFormat:
      @"%@",contentid]]) {
        return YES;
    }else{
        return NO;
    }
}
```

- Acquiring the Friend list of all users.

```
-(void)loadMyfriend:(void (^)(NSArray *friends))oncomplete{
    NSMutableArray* friends=[NSMutableArray array];
    NSString* url=[NSString stringWithFormat:MY_FRIEND_URL,self.loginuser.userid
      ];
    AFHTTPRequestOperationManager *manager = [AFHTTPRequestOperationManager
      manager];
    manager.responseSerializer = [AFHTTPResponseSerializer serializer];
    [manager GET:url parameters:nil success:^(AFHTTPRequestOperation
      *operation, id responseObject) {
        NSString* str=[[NSString alloc] initWithBytes:[responseObject bytes]
          length:[responseObject length] encoding:NSASCIIStringEncoding];
        TBXML* xml=[TBXML tbxmlWithXMLString:str];
        if (xml) {
            TBXMLElement*Items=[TBXML childElementNamed:@"Items" parentElement:
              xml.rootXMLElement];
            if (Items) {
                TBXMLElement* Item=[TBXML childElementNamed:@"Item"
                  parentElement:Items];
                while (Item) {
                    NSMutableDictionary* f=[NSMutableDictionary dictionary];
                    f[@"name"]=[TBXML textForElement:[TBXML childElementNamed:@"
                      name" parentElement:Item]];
                    f[@"requestStatus"]=[TBXML textForElement:[TBXML
                      childElementNamed:@"requestStatus" parentElement:Item]];
                    f[@"userID"]=[TBXML textForElement:[TBXML childElementNamed:
                      @"userID" parentElement:Item]];
                    [friends addObject:f];
                    Item=[TBXML nextSiblingNamed:@"Item" searchFromElement:Item]
                      ;
                }
            }
        }
        if (oncomplete) {
            oncomplete(friends);
        }
    } failure:^(AFHTTPRequestOperation *operation, NSError *error) {
        NSLog(@"%@",error);
        if (oncomplete) {
            oncomplete(nil);
        }
    }];
}
```

- Judging a Content is in the Reading List or not.

```
-(BOOL)isInReadingList:(NSNumber *)contentid{
    if (self.allMyreadingList[[NSString stringWithFormat:
      @"%@",contentid]]) {
        return YES;
    }else{
        return NO;
    }
}
```

- Adding a Reading List item.

```
-(void)addMyReadingList:(NSNumber *)contentid{
    self.allMyreadingList[[NSString stringWithFormat:@"%@"
      ,contentid]]=contentid;
}
```

- Removing a Reading List item.

```
-(void)removeReadingList:(NSNumber *)contentid{
    [self.allMyreadingList removeObjectForKey:[NSString
      stringWithFormat:@"%@",contentid]];
```

- Adding bookmark. Put the item into “My Schedule” list. This would apply the data to API.

```
-(void)bookmarkContentid:(NSNumber *)cid onComplete:(void
  (^)(BOOL success))oncomplete

{
    NSString* url=[NSString stringWithFormat:BOOKMARK,self
      .loginuser.userid,cid];
    AFHTTPRequestOperationManager *manager = [
      AFHTTPRequestOperationManager manager];
    manager.responseSerializer = [
      AFHTTPResponseSerializer serializer];
    [manager GET:url parameters:nil success:^(
      AFHTTPRequestOperation *operation, id responseObject
      ) {
        NSString* str=[[NSString alloc] initWithBytes:[
          responseObject bytes] length:[responseObject
          length] encoding:NSASCIIStringEncoding];
        NSLog(@"%@",str);
        if (oncomplete) {
            oncomplete(YES);
        }
    } failure:^(AFHTTPRequestOperation *operation, NSError
       *error) {
        NSLog(@"%@",error);
        if (oncomplete) {
            oncomplete(NO);
        }
    }];
}
```

- Removing bookmark.

```
-(void)unbookmarkContentid:(NSNumber *)cid onComplete:(
    void (^)(BOOL success))oncomplete{
    NSString* url=[NSString stringWithFormat:UNBOOKMARK,
        self.loginuser.userid,cid];
    AFHTTPRequestOperationManager *manager = [
        AFHTTPRequestOperationManager manager];
    manager.responseSerializer = [
        AFHTTPResponseSerializer serializer];
    [manager GET:url parameters:nil success:^(
        AFHTTPRequestOperation *operation, id
        responseObject) {
        NSString* str=[[NSString alloc] initWithBytes:[
            responseObject bytes] length:[responseObject
            length] encoding:NSASCIIStringEncoding];
        NSLog(@"%@",str);
        if (oncomplete) {
            oncomplete(YES);
        }
    } failure:^(AFHTTPRequestOperation *operation, NSError
         *error) {
        NSLog(@"%@",error);
        if (oncomplete) {
            oncomplete(NO);
        }
    }];
}
```

- Download all items from “My Schedule”

```
-(void)loadAllMySchedule:(void (^)(NSDictionary* schdules)
    ) oncomplete{
    NSString* url=[NSString stringWithFormat:
        USER_BOOKMARK,self.loginuser.userid,self.
        conferenceId];
    AFHTTPRequestOperationManager *manager = [
        AFHTTPRequestOperationManager manager];
    manager.responseSerializer = [
        AFHTTPResponseSerializer serializer];
    [manager GET:url parameters:nil success:^(
        AFHTTPRequestOperation *operation, id
        responseObject) {
        NSLog(@"download all bookmark complete.");
        NSString* str=[[NSString alloc] initWithBytes:[
            responseObject bytes] length:[responseObject
            length] encoding:NSASCIIStringEncoding];
        self.myRemarks=[NSMutableDictionary dictionary];
        for (BookMarkItem* b in [self
            parseAllMySchedulesWithXMLStr:str]) {
            NSString* key=[NSString stringWithFormat:@"%@"
                ,b.contentId];
            self.myRemarks[key]=b;
        }
        if (oncomplete) {
            oncomplete(self.myRemarks);
        }
    } failure:^(AFHTTPRequestOperation *operation, NSError
        *error) {
        NSLog(@"%@",error);
        if (oncomplete) {
            oncomplete(nil);
        }
    }];
}
```

- Acquiring the users’ information who had logged in.

```
-(User *)loginuser {
    NSUserDefaults* defaults=[NSUserDefaults
        standardUserDefaults];
    if ([defaults objectForKey:@"loginuser"]!=nil) {
        NSString* userstr=[defaults valueForKey:@"
            loginuser"];
        NSDictionary* userdict=[NSJSONSerialization
            JSONObjectWithData:[userstr dataUsingEncoding:
            NSUTF8StringEncoding] options:kNilOptions
            error:nil];
        if (userdict) {
            self.theloginuser=[User fromdict:userdict];
        }
    }
    return self.theloginuser;
}
```

- Setting users' information who is logging in.

```
-(void)setLoginuser:(User *)aloginuser{
    self.theloginuser=aloginuser;
    NSUserDefaults* defaults=[NSUserDefaults
        standardUserDefaults];
    NSData* userdata=[NSJSONSerialization
        dataWithJSONObject:[aloginuser asDict] options:
        NSJSONWritingPrettyPrinted error:nil];
    NSString* userstr=[[NSString alloc] initWithData:
        userdata encoding:NSUTF8StringEncoding];
    [defaults setValue:userstr forKey:@"loginuser"];
    [defaults synchronize];
}
```

- The supplementary method for changing the Conent's authors into NSString type.

```
-(NSString *)authorsTextForContentId:(NSNumber *)contentid
    {
    NSMutableString* authornamesstr=[NSMutableString
        string];
    Content *c=[self contentOfId:contentid];
    NSArray* aps=c.authors;
    for (int i=0;i<[aps count];i++) {
        AuthorPresenter *ap=[aps objectAtIndex:i];
        [authornamesstr appendString:ap.name];
        if (i<[aps count]-1) {
            [authornamesstr appendString:@"-"];
        }
    }
    return authornamesstr;
}
```

- Acquiring all the Content in Preceedingviewcontroller.

```
-(NSArray *)preceedingContents{
    NSMutableArray* contents=[NSMutableArray array];
    for (NSString* key in self.allcontents) {
        Content* c=self.allcontents[key];
        [contents addObject:c];
    }
    return contents;
```

- Re-download all the data, which would be called by clicking “Refresh” button.

```
-(void)downloadData:(void(^)(BOOL))complete{
    [self.networkoperationqueue addOperationWithBlock:^{
        __block BOOL downloadcomplete=NO;
        [self loadAllMySchedule:^(NSDictionary *schdules) {
            downloadcomplete=YES;
        }];
        while (!downloadcomplete) {
            [[NSRunLoop mainRunLoop] runUntilDate:[NSDate distantFuture]];
        }
    }];
    [self.networkoperationqueue addOperationWithBlock:^{
        __block BOOL downloadcomplete=NO;
        [self downLoadContents:^(BOOL success) {
            downloadcomplete=YES;
            if (!success) {
                [self.networkoperationqueue cancelAllOperations];
                if (complete) {
                    complete(NO);
                }
            }
        }];
        while (!downloadcomplete) {
            [[NSRunLoop mainRunLoop] runUntilDate:[NSDate distantFuture]];
        }
    }];
    [self.networkoperationqueue addOperationWithBlock:^{
        __block BOOL downloadcomplete=NO;
        [self downloadEvents:^(BOOL success) {

            downloadcomplete=YES;
            if (!downloadcomplete) {
                [self.networkoperationqueue cancelAllOperations];
                if (complete) {
                    complete(NO);
                }
            }
        }];
        while (!downloadcomplete) {
            [[NSRunLoop mainRunLoop] runUntilDate:[NSDate distantFuture]];
        }
    }];
    [self.networkoperationqueue addOperationWithBlock:^{
        dispatch_async(dispatch_get_main_queue(), ^{
            if (complete) {
                complete(YES);
            }
        });
    }];
}
```

- Acquiring authors’ information in detail according to their ID.

```
-(Author *)authorWithId:(NSNumber *)aid{
    NSString* key=[NSString stringWithFormat:@"%@",aid];
    return self.allauthors[key];
```

- Acquiring all authors.

```
-(NSArray *)getAllAuthors{
    NSMutableArray *allauthors=[NSMutableArray array];
    [allauthors addObjectsFromArray:[self.allauthors
        allValues]];
    return allauthors;
}
```

- Acquiring all Content.

```
-(NSArray *)getAllContent{
    return [self.allcontents allValues];
}
```

- Acquiring current event.

```
-(Event *)getEvent{
    Event* event=[Event new];
    [event setEventid:self.conferenceId];
    [event setShortTitle:@"EC-TEL 2013"];
    [event setTitle:@"EC-TEL 2013"];
    [event setAbstraction:@"The European Conference on
        Technology Enhanced Learning (EC-TEL) is a unique
        opportunity for researchers, practitioners, and
        policy makers to address current challenges and
        advances in the field. "];
    NSDateFormatter *dateFormatter = [[NSDateFormatter
        alloc] init];
    [dateFormatter setDateFormat:@"MM-dd-yyyy"];

    [event setStartdate:[dateFormatter dateFromString:@"09
        -17-2013"]];
    [event setEnddate:[dateFormatter dateFromString:@"09-
        20-2013"]];
    [event setLocation:@"Paphos (Cyprus)"];
    return event;
}
```

- Check whether there is an update request.

```
-(void)checkUpdate:(void (^)(NSDate* time))oncomplete{
    NSString* url=[NSString stringWithFormat:CHECK_EVENT_UPDATE,
        self.conferenceId];
    AFHTTPRequestOperationManager *manager = [
        AFHTTPRequestOperationManager manager];
    manager.responseSerializer = [AFHTTPResponseSerializer
        serializer];
    [manager GET:url parameters:nil success:^(
        AFHTTPRequestOperation *operation, id responseObject) {
        NSString* str=[[NSString alloc] initWithBytes:[
            responseObject bytes] length:[responseObject length]
            encoding:NSASCIIStringEncoding];
        NSDate* updatetme=nil;
        TBXML* xml=[TBXML tbxmlWithXMLString:str];
        if (xml) {
            TBXMLElement* timestampElement=[TBXML
                childElementNamed:@"timestamp" parentElement:xml.
                rootXMLElement];
            if (timestampElement) {
                NSString* timestr=[TBXML textForElement:
                    timestampElement];
                NSLog(@"%@",timestr);
                NSDateFormatter* df=[[NSDateFormatter alloc] init]
                    ;
                [df setDateFormat:@"yyyy-MM-dd HH:mm:ss.SSS"];
                updatetme=[df dateFromString:timestr];
            }
        }

        if (oncomplete) {
            oncomplete(updatetme);
        }
    } failure:^(AFHTTPRequestOperation *operation, NSError *error)
        {
        NSLog(@"%@",error);
        if (oncomplete) {
            oncomplete(nil);
        }
    }];
}
```

- Acquiring all Presentations according to SessionId.

```
-(NSArray *)presentationsOfSessionId:(NSNumber *)sessionid
    {
    NSMutableArray *ps=[NSMutableArray array];
    for (Presentation* p in self.allPresentations) {
        if ([p.sessionId intValue]==[sessionid intValue])
            {
            [ps addObject:p];
        }
    }
    return ps;
}
```

- Judging item has been added to “My Schedule” or not.

```
-(BOOL)isRemarked:(NSNumber *)contentid{
    NSString* key=[NSString stringWithFormat:@"%@"
        ,contentid];
    if (self.myRemarks[key]) {
        return YES;
    }
    return NO;
}
```

- Setting item to “My Schedule”. (Add or delete)

```
-(void)setRemark:(NSNumber *)contentid remarked:(BOOL)
    remarked{
    NSString* key=[NSString stringWithFormat:@"%@"
        ,contentid];
    if (remarked) {
        BookMarkItem* item=[BookMarkItem new];
        item.contentId=contentid;
        self.myRemarks[key]=item;
    }else{
        [self.myRemarks removeObjectForKey:key];
    }
}
```

- Acquiring information of Content according to ContentID

```
-(Content *)contentOfId:(NSNumber *)cid{
    NSString* key=[NSString stringWithFormat:@"%@",cid];
    return self.allcontents[key];
}
```

- Acquiring all Content by authors' ID.

```
-(NSArray *)contentsByAuthorId:(NSNumber *)aid{
    NSMutableArray* contents=[NSMutableArray array];
    for (NSString* key in self.allcontents) {
        Content* c=self.allcontents[key];
        if ([self isContent:c byAuthorId:aid]) {
            [contents addObject:c];
        }
    }
    return contents;
}
```

- Acquiring Presentation by ContentId.

```
-(Presentation *)presentationOfContentId:(NSNumber *)cid{
    Presentation* findpresentation=nil;
    int cid_i=[cid intValue];
    for (int i=0; i<[self.allPresentations count]; i++) {
        Presentation* p=self.allPresentations[i];
        if ([p.contentId intValue]==cid_i) {
            findpresentation=p;
            break;
        }
    }
    return findpresentation;
}
```

- Acquiring Presentation by presentationId.

```
-(Presentation *)presentationOfPid:(NSNumber *)pid{
    Presentation* findpresentation=nil;
    int pid_i=[pid intValue];
    for (int i=0; i<[self.allPresentations count]; i++) {
        Presentation* p=self.allPresentations[i];
        if ([p.pID intValue]==pid_i) {
            findpresentation=p;
            break;
        }
    }
    return findpresentation;
}
```

- Acquiring Session by SessionID.

```
-(Session *)sessionOfSessionId:(NSNumber *)sid{
    Session* findsession=nil;
    int sid_i=[sid intValue];
    for (int i=0; i<[self.allsessions count]; i++) {
        Session* s=self.allsessions[i];
        if (sid_i==[s.sID intValue]) {
            findsession=s;
            break;
        }
    }
    return findsession;
}
```

- Acquiring Session according to corresponding date.

```
-(NSArray *)getSessionForDate:(NSDate *)date{
    NSMutableArray* sessions=[NSMutableArray array];
    NSDateFormatter* fm=[[NSDateFormatter alloc] init];
    [fm setDateFormat:@"MM-dd"];
    NSString* ds=[fm stringFromDate:date];
    for (Session* s in self.allsessions) {
        NSString* st=[fm stringFromDate:s.date];
        if ([st isEqualToString:ds]) {
            [sessions addObject:s];
        }
    }
    return sessions;
}
```

- Acquiring information of recommendation by current Content.

```
-(void)loadRecomendsOfContentId:(NSNumber *)cid
    oncomplete:(void(^)(NSArray *recomends))complete{
    AFHTTPRequestOperationManager *manager = [
        AFHTTPRequestOperationManager manager];
    manager.responseSerializer = [
        AFHTTPResponseSerializer serializer];
    NSString* contentsurl=[NSString stringWithFormat:
        RECOMEND_URL,cid,self.conferenceId,@(5)];
    [manager GET:contentsurl parameters:nil success:^(
        AFHTTPRequestOperation *operation, id
        responseObject) {
        NSString* str=[[NSString alloc] initWithBytes:[
            responseObject bytes] length:[responseObject
            length] encoding:NSASCIIStringEncoding];
        NSArray* recomens=[self parseRecomendsXmlStr:str];
        if (complete) {
            complete(recomens);
        }
    } failure:^(AFHTTPRequestOperation *operation, NSError
         *error) {
        if (complete) {
            complete(@[]);
        }
    }];
}
```

B. third-party packages

1. **RatingView**

   Use to show the rating stars interface.

   Download address: http://code4app.com/ios/Star-Rating-View/4f67f90b6803fadb44000002

2. **Progress**
   This is the circle of showing the rate of progress when starting up or updating.
   Download address: http://code4app.com/ios/MRProgress/527a5de56803fa071d000000

3. **TBXML**
   It's a resolver library using in iOS. It provides simple port for using it conveniently and the method of acquiring XML node and attribute value. It also could load XML file by file path, URL, XML file content, contents' string, etc.
   Eg, we acquire the information of session by using the textFortElement method in TBXML.