Improving the maintainability, reliability and the usability of an existing software system

Yang Liu

July 18, 2017

Abstract

This project redesigned and refactored the backend and frontend of ASSISTments mobile App. Naming conventions, design patterns and best programming practices were applied to improve the readability, flexibility and reliability of the system. Crashlytics was integrated for administrators and developers to monitor the crashes and usage statistics of the app. CocoaPods was also added to manage the dependencies for the front end. The app also got a nicer UI specially optimized for mobile devices.

Acknowledgement

I would give special thanks to Professor Heffernan for providing me the opportunity to work on ASSISTment project. Professor offered me lots of hints on understanding the database architecture, on fixing the bugs and on improving the performance of the SQL queries. My colleagues, Christopher Donnelly and David Magid helped me configure the developed environment, understand the SDK, fix the bug on the back end and deploy my new implementation to production server. I also want to thank Anthony Botelho, as my supervisor, coordinated the meetings and helped me to understand the core database structure. Lastly, I would like to thank Cristina Heffernan and Andrew Burnett for giving me suggestions on the mobile UI.

Authorship

I wrote all the sections and edited all of them myself.

Contents

List of Figures

1 Introduction

Shortly after improving Maine state students' math grades by 20%, ASSISTments became one of most popular intellectual tutoring systems on the Internet. The system now has at least 500 monthly active users from its mobile app and a lot more from the web. The research team regularly receives bug reports, feature requests and administrative permission questions from the users. The developers were not always able to respond to user feedbacks quickly enough due to lack of centralized database structure documentation and readable code.

The current version of the system is fairly complex and takes developers and researchers few weeks to learn the very basic structure. For example, there are *Sequence*, *Section*, *Assistment* and *Problem* tables in the database. Inexperienced developers would assume that each sequence has multiple sections, each section has multiple problems and each section contains a list of problems. However, this is not the case with our current database structure. Each *Sequence* actually contains *Section*s with recursive tree structure. And each leaf of the tree is a *Problem Section*. Each *Problem Section* contains only one *Assistement* and each *Assistment* has multiple sub *Problem*s. It cost me and some of my colleagues few weeks to partially understood the database structure. Well maintained documentation will significantly reduces the amount of time we spent on learning the fundamentals of our database. The learning curve would be even smoother for new developers if the team hosts an up to date wiki documenting the structure of the database and the installation of the SDK.

According to software development good practices [Fut] on Github, effective automated testing and crash monitoring strategies will improve the maintainability and stability of the software system. A few months ago, the app crashed when my supervisor was recording the video hint. The team has no idea towards the source of the crash. Developers assumed the cashes to be Apple's bug until I integrate Crashlytics into the app. Crashlytics not only monitors the crashes but also report which line of the code leads to the crash. This greatly helped me locate the bug and fix it with one line of code. Right after the fix, my supervisor found another crash inside the app. The bug was caused by part of the incorrect implementation of the SDK. Adding few proper test cases would help the developers to catch the problem and fix it a head of time.

Recently, the app encountered a serious data lost. The app was mistakenly hooked to the testing server, caused significant loss of student's test answers. Even though my supervisor and I fully tested the app by hand, none of us were able to catch the error, mainly because the testing server has exactly the same user interface as the production. Two weeks after receiving bug reports, the team was eventually able to locate the bug. When the developers were trying to push the updates to the users, they found the system did not have force update implementation at the time. The update was not fully adopted by users until few weeks later. In the new version, we added force update and backend diagnostic code to support possible disaster recovery in the future.

The goal of this project was to help developers identify and implement the best practices that would make ASSISTments maintainable, usable and reliable. Useful tools, techniques, architectures, frameworks and design patterns were introduced in order to give the developers and researchers complete control over the entire process and its outcomes. The strategies were intended to ease and simplify developers' life during their future maintenance. Emphasis was placed on applying modern architectures and designed patterns to the system making it maintainable and scalable. The project is also intended to help other software development and research teams through the example of ASSISTments by giving them the confidence and technical knowledge to improve their own systems.

2 Problem Assessment and Solutions

The problem of writing readable, flexible and reliable code for an existing software system is a significant concern of the developers working on big projects. Inappropriate design and practices can slow down the system, cause bugs, lose customers and even lead to complete rewrite of the application. We will explore the maintainability, reliability as well as the usability problems that face developers. We are also going to take look at my solutions after each problem.

2.1 Code Smell

The key advantage of software system compared with traditional manufacture is its ability to adapt to the possible or future changes in its requirements. Here is piece of sample source code in the mobile server:

```
// Java
/**
 * Create the problem set with video hint as customization
 * @param request the input request containing all the information needed
 * @return the database id of the newly created problem set
 * @throws NotEnoughInfoException if the post request does not have enough
 * information
 * @throws NotFoundException if the original ids (that are copying from)
 * not found
 */
@Override
public int createCustomizedProblemSetWithVideoHint(Map<String, Object> request)
    throws NotEnoughInfoException, NotFoundException{

    int oldSequenceId;
    String ast_id_list;
    int problemId;
    int similarProblemId;
    int contentCreatorId;
    String videoHint;
    boolean gives_answer;
    boolean generic_explanation;

    try{
        oldSequenceId = (int)request.get("sequence_id");
        ast_id_list = (String)request.get("ast_id_list");
        problemId = (int)request.get("id");
        similarProblemId = (int)request.get("similar_id");
        contentCreatorId = (int)request.get("content_creator_id");
        videoHint = (String)request.get("video_hint");
        gives_answer = (boolean)request.get("gives_answer");
        generic_explanation = (boolean)request.get("generic_explanation");
    }catch(NullPointerException e){
        throw new NotEnoughInfoException();
    }

    if(ast_id_list == null || videoHint == null){
        throw new NotEnoughInfoException();
    }
```

```
41          List<String> astStrIdList = Arrays.asList(ast_id_list.split(","));
            List<Integer> astIdList = new ArrayList<Integer>();
43          for(String s : astStrIdList) astIdList.add(Integer.valueOf(s));

45          int problemAstId = sequenceManager
                .findAssistmentIdByProblemId(problemId);
47          int similarProblemAstId = sequenceManager
            .findAssistmentIdByProblemId(similarProblemId);
49
            int postTestAstId = 0;
51          int similarPosttestAstId = 0;
            int problemTemplateId = sequenceManager
53              .findAssistmentInfoByAssistmentId(problemAstId).getParentId();
            int similarProblemTemplateId = sequenceManager
55              .findAssistmentInfoByAssistmentId(similarProblemAstId).getParentId();

57          boolean postTestFlag1 = true, postTestFlag2 = true;
            for(int l : astIdList){
59              int tempParentId = sequenceManager
                    .findAssistmentInfoByAssistmentId(l).getParentId();
61              if(postTestFlag1){
                    if(problemTemplateId == tempParentId){
63                      postTestAstId = l;
                        postTestFlag1 = false;
65                      continue;
                    }
67              }
                if(postTestFlag2){
69                  if(similarProblemTemplateId == tempParentId){
                        similarPosttestAstId = l;
71                      postTestFlag2 = false;
                    }
73              }
                if(!postTestFlag1 && !postTestFlag2)
75                  break;
            }
77
            if(postTestAstId == 0L || similarPosttestAstId == 0L)
79              throw new NotFoundException(NotFoundExceptionType.DATABASE_ID,
                "Need two post test items but there are not enough");
81
            astIdList.remove(new Integer(problemAstId));
83          //astIdList.remove(new Integer(similarProblemAstId));
            astIdList.remove(new Integer(postTestAstId));
85          //astIdList.remove(new Integer(similarPosttestAstId));

87          long seed = System.nanoTime();
            Collections.shuffle(astIdList, new Random(seed));
89
            // Create new problem set
91          int newSequenceId = deepCopySequence(oldSequenceId, contentCreatorId);

93          // Add video hint to both p1 and p1Morph and put them into problem sections
```

```
        int customizedAstId = deepCopyAssistmentWithVideoHint(problemAstId,
95          videoHint, contentCreatorId);
        int customizedSectionId = copyManager
97      .createProblemSectionByAstId(
            newSequenceId, customizedAstId);
99      //int customizedSimilarAstId = deepCopyAssistmentWithVideoHint(
            similarProblemAstId, videoHint, contentCreatorId);
101     //int customizedSimilarSectionId = copyManager
        // .createProblemSectionByAstId(newSequenceId,
103     //  customizedSimilarAstId);

105     // Put two post test items, p1 and p1Morph into problem sections
        int problemSetionId = copyManager.createProblemSectionByAstId(
107         newSequenceId, problemAstId);
        //int similarProblemSectionId = copyManager.createProblemSectionByAstId(
109     //  newSequenceId, similarProblemAstId);
        int postTestSectionId = copyManager.createProblemSectionByAstId(
111         newSequenceId, postTestAstId);
        //int similarPosTestSectionId = copyManager.createProblemSectionByAstId(
113     //  newSequenceId, similarPosttestAstId);

115     // Put two place holder problems into problem sections and remove them
        // from assistment list
117     int placeHolderSectionId1 = copyManager
            .createProblemSectionByAstId(newSequenceId, astIdList.remove(0));
119     int placeHolderSectionId2 = copyManager
            .createProblemSectionByAstId(newSequenceId, astIdList.remove(0));
121     int placeHolderSectionId3 = copyManager
            .createProblemSectionByAstId(newSequenceId, astIdList.remove(0));
123     int placeHolderSectionId4 = copyManager
            .createProblemSectionByAstId(newSequenceId, astIdList.remove(0));
125
        // Create the choose condition section with 8 different child sections
127     int chooseCondtionSectionId = createChooseConditionSectionWithFourProblems(
        problemSetionId, postTestSectionId, customizedSectionId,
129         placeHolderSectionId1,
            placeHolderSectionId2,
131         placeHolderSectionId3,
            placeHolderSectionId4,
133         newSequenceId);

135     // Create the skill builder section with all the rest of the original
        // problem set
137     List<Integer> skillBuildersectionIdList = new ArrayList<Integer>();
        for(int l : astIdList){
139         skillBuildersectionIdList
                .add(copyManager.createProblemSectionByAstId(newSequenceId, l));
141     }
        int skillBuilserSectionId = copyManager
143         .createSectionByChildSections(newSequenceId, skillBuildersectionIdList,
                Section.SectionType.MASTERY.getTypeName(), "--- {}");
145             //TODO move type string to enum class

147     // Create If then else head section with choose condition section
```

```
            // at first and mastery section at second
149         List<Integer> finalSectionIdList = new ArrayList<Integer>();
            finalSectionIdList.add(chooseCondtionSectionId);
151         finalSectionIdList.add(skillBuilserSectionId);
            int headSectionId = copyManager
153             .createSectionByChildSections(
                newSequenceId,
155             finalSectionIdList,
                Section.SectionType.IF_THEN_ELSE.getTypeName(),
157             "--- \nCorrectness: 99\nEmptyBranch: emptyThen\n");

159         // Link sequence with head section
            sequenceManager.linkSequenceWithHeadSection(newSequenceId,headSectionId);
161
            // Store url into user generated hints table
163     copyManager.createUserGeneratedHint(sequenceManager.findProblemByAssistmentId(
            customizedAstId).getId(),
165         newSequenceId,
            contentCreatorId,
167         decodeEmbedCode(videoHint),
            gives_answer,
169         generic_explanation);
            return newSequenceId;
171     }
```

The code above is trying to generate a experiment to test out the effectiveness of a newly uploaded video hint. The code has few maintainability and flexibility issues. It is not modular. Modularity would help developers to reuse code in another part of the software. This speed up the implementation of other feature having similar business logic in the future.

Another problem of the code is that the method is too long. It's really hard to follow and understood the business logics when there is too much unrelated information around. The code parsing the request parameters should be extracted to a standalone helper method with concise and meaningful name following the conventions. This will greatly improve the readability of the code. When the maintainer is trying figure out the business logic of code, he only need to take look at the function name. If the he want to review the detailed implementation, he may then check out the chunk of code inside the method. It would also help when developers are making changes to the code, they only need to modify the method extracted, without worry about the rest of code. And fixing bug in this method will also fix the abnormal behaviors in other part of system referenced it.

There are too many inline comments in the code snippet. Comments should be used to clarify very confusing concept or logic in the code, helping maintainer to understood the complex algorithm underneath. Well written code should be self-evident and readable and ideally need no comment to explain its business logic. Readable code can help developers to understand the underneath business logics and adapt to requirement changes quickly. Ashod Nakashian, a software developer, explored the need of code readability in his online article [Nak]. He claimed that software developers spend majority of their time adding, removing and editing small pieces of documentation and code. Most the time they are making small changes on a relatively large code-base. They rarely write independent piece of code unless they are starting a project from scratch. Most of the existing projects were passed from the last generation of developers to the next. They write methods to implement new features and modify the exiting code to adopt to their current needs. All of those requires reading a large amount of code to understand the current implementation and decide the necessary changes to produce the expected outputs. It become especially changeling for developers to understand the confusing source code with imminent deadlines.

To above code is factored to improve code readability and reusability. Here is the refactored code:

```
/**
 * Create a control trail experiment to test effectiveness of a hint
 *
 * v:         MobileHint to test
 * p1:        MobileProblem customized by instructor
 * p0:        MobileProblem randomly selected from problem set
 * p1Morph:   MobileProblem similar to the customized problem
 * r:         MobileProblem randomly chosen to prevent cheating
 * p:         MobileProblem randomly chosen to prevent cheating
 *
 */
@Override
public MobileExperiment generateHintExperiment(Sequence sequence,
MobileProblem modifiedMobileProblem,
Function<MobileProblem, MobileProblem> addHintToProblem) {
    MobileExperiment mobileExperiment = new MobileExperiment();
    int newSequenceId = mobileProblemSetManager
        .makeNewCopy(sequence);
    mobileExperiment.setSequenceId(newSequenceId);
    List<MobileProblem> mobileProblems = mobileProblemSetManager
        .getProblems(sequence);
    List<MobileChooseCondition> mobileChooseConditions = makeChooseConditions(
    newSequenceId,
    mobileProblems,
    modifiedMobileProblem,
    addHintToProblem);
    for (MobileChooseCondition mobileChooseCondition : mobileChooseConditions)
        mobileExperiment.addChooseCondition(mobileChooseCondition);
    mobileSkillBuilderManager
        .makeChooseConditionSkillBuilder(
        newSequenceId,
        mobileChooseConditions,
        mobileProblems);
    return mobileExperiment;
}

@Override
public MobileExperiment generateVideoHintExperiment(
Sequence sequence,
MobileProblem modifiedMobileProblem,
MobileVideoHint videoHint) {
    return generateHintExperiment(
    sequence,
    modifiedMobileProblem,
    (MobileProblem problem) -> hintManager.addVideoHintTo(problem, videoHint));
}
```

2.2 Design Patterns

Design patterns can help developers to organize their code while solving repeating design issues in the software system. Alexander Shvets, experienced consultant on software development issues in banking and

healthcare, described design patterns in his online post[Shv]:

> Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

Unfortunately, the original front end code of the app was trying to avoid using patterns. The code even skipped Apple's default MVC pattern. There are some views and controllers out there. However, the code didn't have any business model designed to preserve the application data. It pulled data from the backend server and temporarily saved the attributes of into variables. When I was trying to access the data in views and controllers, it become exceptionally confusing and error prone. Here is piece of sample code form the original app front end.

```
// Objective-C
- (void) assignProblemWithCustomizaiton:(BOOL) isCustomized{

    //set up nsurlsession for assigning problem
    NSURLSessionConfiguration *sessionConfig =
        [NSURLSessionConfiguration defaultSessionConfiguration];

    [sessionConfig setHTTPAdditionalHeaders:@{@"Accept": @"application/json"}];

    NSURLSession *session =
        [NSURLSession sessionWithConfiguration:sessionConfig
                                                    delegate:nil
                                               delegateQueue:nil];
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:
    [NSURL URLWithString:ASSIGN_PROBLEMSET_URL]
        cachePolicy:NSURLRequestUseProtocolCachePolicy
        timeoutInterval:50.0];
    [request addValue:@"partner=\"mobileApp\""
        forHTTPHeaderField:@"assistments-auth"];
    [request addValue:@"application/json"
        forHTTPHeaderField:@"Content-Type"];
    [request
        setHTTPMethod:@"POST"];

    //build post request to assign problem
    AppDelegate *appDelegate = (AppDelegate *)
        [[UIApplication sharedApplication] delegate];

    NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
    [formatter setDateFormat:@"MM/dd/yyyy HH:mm a"];
    NSString *dueDateStr = @"";
    if(_dueDate != nil) dueDateStr = [formatter stringFromDate:_dueDate];
    NSString *releaseDateStr = @"";
    if(_releaseDate != nil) releaseDateStr =
        [formatter stringFromDate:_releaseDate];

    NSMutableDictionary *assignRequestDict =
        [[NSMutableDictionary alloc] initWithObjectsAndKeys:
```

```
                                                appDelegate.username,@"username",
                                                appDelegate.password,@"password",
                                                classIDForAssign, @"class_id",
                                                releaseDateStr, @"release_date",
                                                dueDateStr, @"due_date",
                                                nil];
    if(isCustomized) {

        [assignRequestDict setValue:_sequenceIDForAssign forKey:@"sequence_id"];

    }else {

        [assignRequestDict setValue:[NSNumber
            numberWithInteger:
            [self.mProbID integerValue]]
            forKey:@"sequence_id"];
    }

    NSLog(@"assign request dictionary:%@", assignRequestDict);

    NSError *error;
    NSData *postdata = [NSJSONSerialization
        dataWithJSONObject:assignRequestDict options:0 error:&error];
    //    if(error) {
    //
    //        NSError *jsonError = [NSError
    //          errorWithDomain:JSONParseErrorDomain code:0 userInfo:nil];
    //        [self showErrorAlert:jsonError];
    //        NSLog(@"json parse error:%@", error);
    //
    //        return;
    //    }
    //
    [request setHTTPBody:postdata];

    NSURLSessionDataTask *assignProblemSetTask =
        [session dataTaskWithRequest:request
            completionHandler:
            ^(NSData *data,NSURLResponse *response,NSError *error) {

        BOOL fail = NO;

        if(error) fail = YES;
        else if(data) {

            NSDictionary *res =
                [NSJSONSerialization
                    JSONObjectWithData:data options:kNilOptions error:&error];

            NSLog(@"assign problemset response:%@", res);

            if(error) {

                fail = YES;
```

```
94                  }
96                  fail |= [[res valueForKey:@"status"] isEqualToString:@"fail"];
98
              } else {
100
                  fail = YES;
102           }
104           dispatch_async(dispatch_get_main_queue(), ^{
106               if(fail) {
                      UIAlertView *assignAlert =
108                   [[UIAlertView alloc]initWithTitle:@"Fail"
                          message:@"Assign failed!" delegate:nil
110                       cancelButtonTitle:@"OK" otherButtonTitles: nil];
                      [assignAlert show];
112               } else {
                      //  UIAlertView *assignAlert =
114                   // [[UIAlertView alloc]initWithTitle:@"Congratulations!"
                      // message:@"Assigned Successfullly!" delegate:nil
116                   // cancelButtonTitle:@"OK" otherButtonTitles: nil];
                      // [assignAlert show];
118               }
              });
120       }];
122       [assignProblemSetTask resume];
124   }
      #pragma mark- Process Data
126   -(void)processData
      {
128       self.namesOfNextLayer = [self.Tree allKeys];
          NSArray * tep =
130       [self.namesOfNextLayer sortedArrayUsingSelector:@selector(compare:)];
          self.namesOfNextLayer = tep;
132       self.numOfThisLayer = self.namesOfNextLayer.count;
          if ([[self.Tree objectForKey:[self.namesOfNextLayer objectAtIndex:0]]
134           objectForKey:@"Title"] ||
           [[self.Tree objectForKey:[self.namesOfNextLayer objectAtIndex:0]]
136       objectForKey:@"Id"]) {
              self.isFinalLayer = true;
138           self.mProbTitle=[[self.Tree objectForKey:[self.namesOfNextLayer
              objectAtIndex:0]] objectForKey:@"Title"];
140           self.mProbID =[[self.Tree objectForKey:[self.namesOfNextLayer
              objectAtIndex:0]] objectForKey:@"Id"];
142       }else
          {
144           self.isFinalLayer = false;
          }
146
      }
```

```
148
    -(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
150 (NSIndexPath *)indexPath
    {
152
        NSDictionary* sender = [self.Tree objectForKey:
154     [self.namesOfNextLayer objectAtIndex:indexPath.row]];
        self.mProbID = [sender objectForKey:@"Id"];
156     self.mProbTitle= [sender objectForKey:@"Title"];
        self.nextTitle= [self.namesOfNextLayer objectAtIndex:indexPath.row];
158     if (self.isFinalLayer) {
            UIAlertView *wantAlert =
160         [[UIAlertView alloc] initWithTitle:@"Do you want to"
            message:@"customize this problem set?"
162         delegate:self
            cancelButtonTitle:@"Cancel"
164         otherButtonTitles:@"Customize and Assign",@"Assign",nil];
            [wantAlert setTag:5];
166         [wantAlert show];
            //Edit when the 2nd Api comes out
168     }else
        {
170         [self performSegueWithIdentifier:@"segueToSelf" sender:sender];
        }
172

174 }
```

I refactored the code with MVVM (Model View View-Model) pattern to separate the view, the interaction logic and the application data into different components. I also put complex and reusable business logics into services. I utilized reactive programming (combination of observable pattern and functional programming) to dynamically bind view model to the view. When the data model changes, they will automatically update the view. Here is the sample code from the new app:

```
1   // Swift
    // SelectProblemSetViewController.swift
3   // Render table view for problem sets in certified folder
    extension SelectProblemSetViewController: UITableViewDelegate {
5       func tableView(_ tableView: UITableView, didSelectRowAt
        indexPath: IndexPath) {
            let folderContentViewModel = items[indexPath.row]
            var viewController: UIViewController!
9           if folderContentViewModel.isFolder {
                let selectProblemSetViewModel = SelectProblemSetViewModel(
11              folderContentViewModel: folderContentViewModel)
                viewController =
13                  SelectProblemSetViewController(
                    selectProblemSetViewModel:selectProblemSetViewModel)
15          } else {
                StateService
17                  .shared
                    .problemSetIdSubject
19                  .onNext(folderContentViewModel.id)
                let problemDetailViewModel = ProblemDetailViewModel()
```

```
                viewController =
                    ProblemDetailViewController(problemDetailViewModel:
                        problemDetailViewModel)
            }
            present(viewController, animated: true, completion: nil)
        }
    }

    // SelectProblemSetViewModel.swift
    // Prepare view model with data from core business model
    class SelectProblemSetViewModel {
        var title: BehaviorSubject<String> =
            BehaviorSubject<String>(value: "")
        var placeHolder: BehaviorSubject<String> =
            BehaviorSubject<String>(value: "")
        var id: BehaviorSubject<Int?> =
            BehaviorSubject<Int?>(value: nil)
        var items: BehaviorSubject<[FolderContentViewModel]> =
            BehaviorSubject<[FolderContentViewModel]>(value: [])
        var disposeBag = DisposeBag()

        init() {
            title.onNext("ASSISTments Certified")
            id.onNext(ASSISTmentsMobileAPI.ASSISTmentsCertifiedFolderId)
            placeHolder.onNext("Problem Set Name / Folder Name")

            let folderService = FolderItemService()
            folderService.folderItemsSubject.map { folderItems ->
                [FolderContentViewModel] in
                return folderItems.map({ folderItem ->
                    FolderContentViewModel in
                        FolderContentViewModel(
                        id: folderItem.id,
                        name: folderItem.name,
                        isFolder: folderItem.isFolder)
                })
                }.subscribe(onNext: items.onNext)
                .addDisposableTo(disposeBag)

            folderService
                .getItemInCertifiedFolderActionSubject
                .onNext(Void())
        }

        init(folderContentViewModel: FolderContentViewModel) {
            title.onNext(folderContentViewModel.name)
            id.onNext(folderContentViewModel.id)
            placeHolder.onNext("Problem Set Name / Folder Name")

            let folderService = FolderItemService()
            folderService.folderItemsSubject.map { folderItems ->
                [FolderContentViewModel] in
                    return folderItems.map({ folderItem ->
                        FolderContentViewModel in
```

```
                            FolderContentViewModel(
                                id: folderItem.id,
                                name: folderItem.name,
                                isFolder: folderItem.isFolder)
                    })
                }.subscribe(onNext: items.onNext)
                .addDisposableTo(disposeBag)
            folderService
                .folderIdSubject
                .onNext(folderContentViewModel.id)
        }
    }

    // FolderItemService.swift
    // Query data from the back end and convert them to business model

    class FolderItemService {
        var folderItemsSubject = PublishSubject<[FolderItem]>()
        var folderIdSubject = PublishSubject<Int>()
        var getItemInCertifiedFolderActionSubject = PublishSubject<Void>()
        var disposeBag = DisposeBag()

        init() {
            folderIdSubject
                .flatMapLatest({ folderId in
                    EnvService
                        .env.assistmentsMobileProvider
                        .request(.getItemsInFolder(folderId: folderId))
                })
                .filter({ response in
                    guard let httpResponse =
                    response.response as? HTTPURLResponse else { return false }
                    return httpResponse.statusCode < 400
                })
                .mapJSON()
                .map { json -> [FolderItem] in
                    let jsonArray = json as! [[String: Any]]
                    guard let items =
                        Mapper<FolderItem>()
                            .mapArray(JSONArray: jsonArray) else { return [] }
                    return items.sorted(by: { (firstItem, secondItem) -> Bool in
                        return firstItem.name < secondItem.name })
                }.subscribe(onNext: folderItemsSubject.onNext)
                .addDisposableTo(disposeBag)
            getItemInCertifiedFolderActionSubject.subscribe(onNext: { action in
                self.folderIdSubject
                    .onNext(ASSISTmentsMobileAPI.ASSISTmentsCertifiedFolderId)
            }).addDisposableTo(disposeBag)
        }
    }
```

I further applied strategy pattern to support internationalization for the new app.

```
    // Swift
```

```swift
    // SignInView.swift
    internal func addUsernameTextField(top: CGFloat) -> CGFloat {
        usernameTextField = UITextField()
        usernameTextField.autocorrectionType = .no
        usernameTextField.autocapitalizationType = .none
        usernameTextField.placeholder = I18nService
            .shared
            .language
            .emailOrUsername
        let topCorners =
            UIRectCorner(rawValue:
            (UIRectCorner.topLeft.rawValue | UIRectCorner.topRight.rawValue))
        return addTextSignInTextField(
        top: top,
        textField: usernameTextField,
        corners: topCorners)
    }

    internal func addPasswordTextField(top: CGFloat) -> CGFloat {
        passwordTextField = UITextField()
        passwordTextField.placeholder = I18nService.shared.language.password
        passwordTextField.isSecureTextEntry = true
        let bottomCorners =
        UIRectCorner(rawValue: (UIRectCorner.bottomLeft.rawValue |
            UIRectCorner.bottomRight.rawValue))
        return addTextSignInTextField(
            top: top,
            textField: passwordTextField,
            corners: bottomCorners)
    }

    // I18nService.swift
    class I18nService {
        static let shared = I18nService()
        var language: Language {
            let locale = NSLocale.autoupdatingCurrent
            let code = locale.languageCode!

            switch code {
                case "en":
                    return English()
                case "zh":
                    return Chinese()
                default:
                return English()
            }
        }
    }

    // Language.swift
    protocol Language {
        var signIn: String { get }
        var emailOrUsername: String { get }
        var password: String { get }
```

```
56          var assignments: String { get }
            var settings: String { get }
58          var scratch: String { get }
            var report: String { get }
60          var searchAssignment: String { get }
            var assignProblemSet: String { get }
62          var customizeProblem: String { get }
            var cancel: String { get }
64          var classes: String { get }
        }
66
        // English.swift
68      class English: Language {
            var signIn: String {
70              return "Sign In"
            }
72          var emailOrUsername: String {
                return "Email or username"
74          }
            var password: String {
76              return "Password"
            }
78          var assignments: String {
                return "Assignments"
80          }
            var settings: String {
82              return "Settings"
            }
84          var scratch: String {
                return "Scratch"
86          }
            var report: String {
88              return "Report"
            }
90          var searchAssignment: String {
                return "Title / Content"
92          }
            var assignProblemSet: String {
94              return "Assign Problem Set"
            }
96          var customizeProblem: String {
                return "Customize Problem"
98          }
            var cancel: String {
100             return "Cancel"
            }
102         var classes: String {
                return "Classes"
104         }
        }
```

2.3 Project Structure

Production software system often has lots of files, including source code, configurations, media assets and others. Most the time developers have to frequently navigate between them and make some changes. It's would be really challenging for developers to locate the correct file without well organized folders structure and meaningful filenames. The original ASSISTments mobile app has thousands of files directly located under project the root folder (Figure 1), which took me lots of efforts to find the target file to modify.

In the new app, I created groups and folders for classes and assets based on their functionalities (Figure 2). The *views* folder contains the classes handling the presentation, the *controllers* folder for the interaction handlers, the *models* folder for business models and the *services* for core business logics. Each folder contains a lot less classes than the original app. Now its easier for developer to find the class and resource they are looking for.

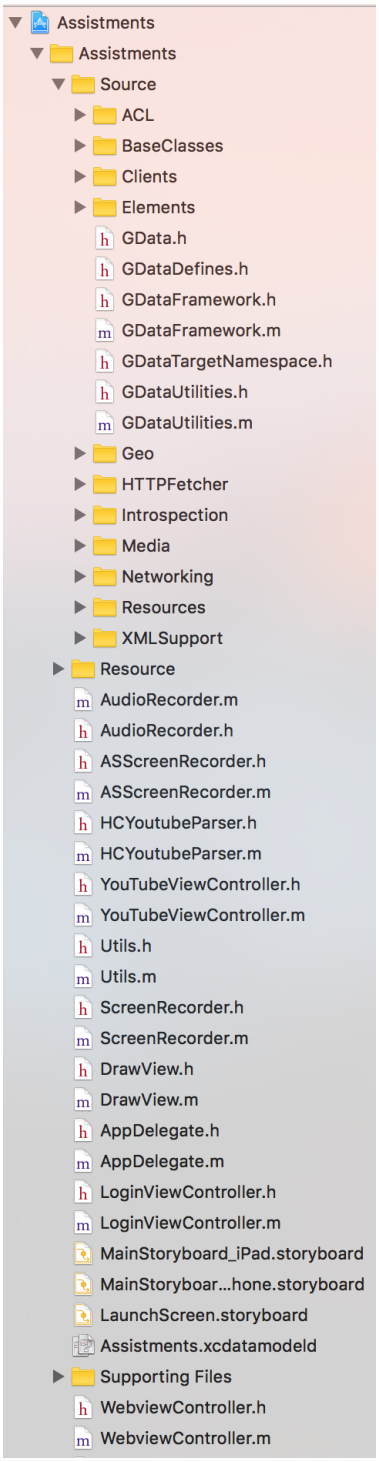

Figure 1: Old project structure

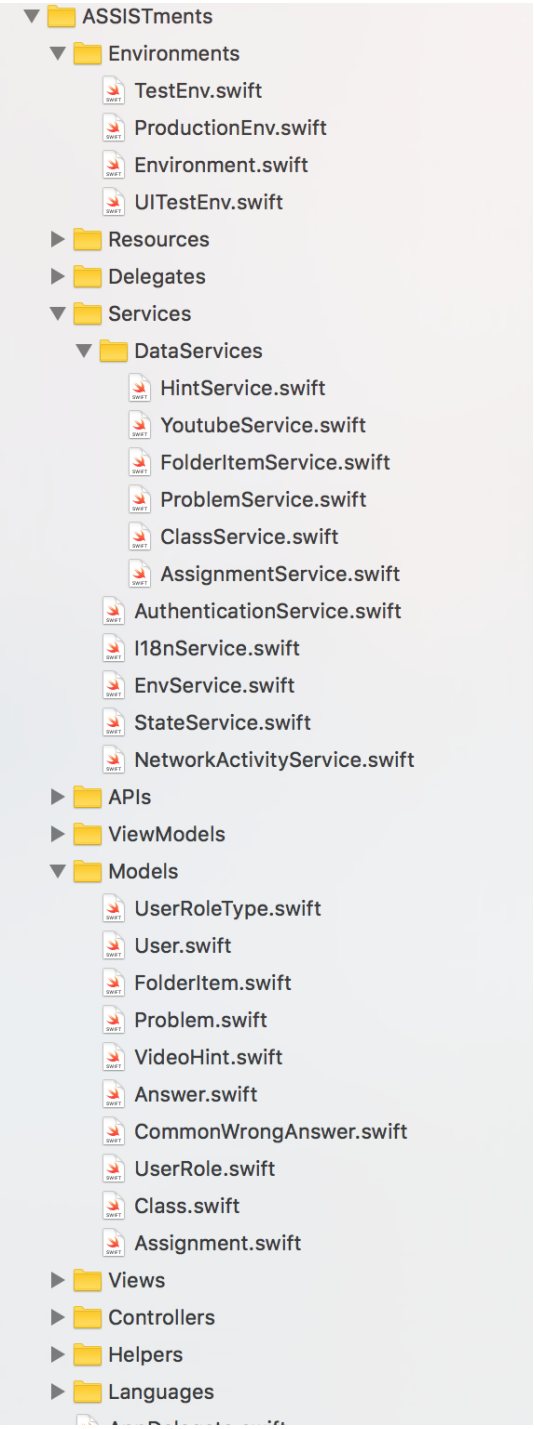

Figure 2: New project structure

2.4 Dependency Management

The original app directly copied the source code of the open source libraries into the project, witch not only distract developers from the real project source code but also made it hard to do source control. When updating the depended libraries or frameworks, developers have to manual edit the source code.

The new app unitized Cocoapod to manage external libraries, scaling the project elegantly. Cocoapod use Podfile to declare the denpendencies. Here is a sample Podfile:

```
# Podfile
# Uncomment the next line to define a global platform for your project
#platform :ios, '9.0'

# Comment the next line if you're not using Swift and don't want to use dynamic
# frameworks
use_frameworks!

target 'ASSISTments' do

    # Pods for ASSISTments

    # Use Fabric as developer tool manager
    pod 'Fabric'
    # Use Crashlytics for Crash Logging
    pod 'Crashlytics'
    # Use Moya for network operation
    pod 'Moya/RxSwift'
    # Use RxSwift and RxCocoa for reactive programming
    pod 'RxSwift',    '~> 3.0'
    pod 'RxCocoa',    '~> 3.0'
    # Use ObjectMapper for mapping json to swift model
    pod 'ObjectMapper', '~> 2.2'
end

target 'ASSISTmentsTests' do
    pod 'Moya/RxSwift'
    pod 'ObjectMapper', '~> 2.2'
end
```

After running *pod install* in the terminal, Cocoapod added a sub project containing all the needed libraries into the workspace of the app (Figure 3).

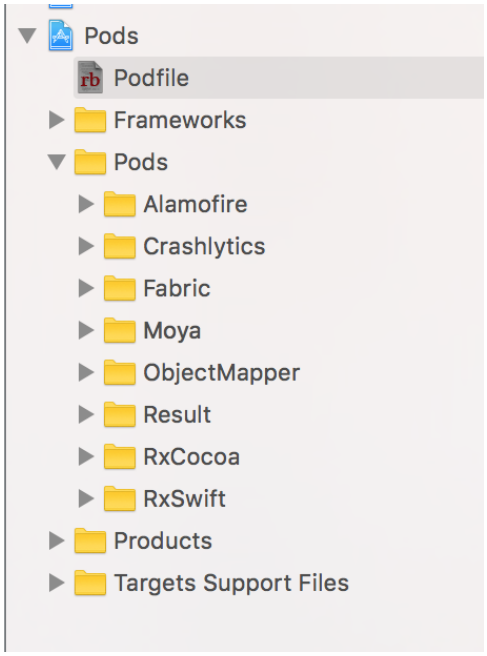

Figure 3: Generated sub project of libraries

Developers can now change the version of any dependencies in Podfile and run *pod install* again to update them conveniently.

2.5 Unit Testing

It's almost inevitable for developers to write buggy code. Buggy code can lose customers or even corrupt important data in the database. Unit testing and Test-driven Developer are great choices to the possible bugs. However, the original system include few ineffective integration tests instead of unit tests. Unit tests can validate the implementations without connecting to the database. The production data based integration tests are slow and may break when the data changes.

To reduce possible bug and improve the reliability of the system, I implemented a in memory testing database and bunch of mock service for the new app. And I also wrote unit tests to validate the core business logic at the back end with data fixture for each test case. Here are the mock services and the in-memory database (Figure 4):

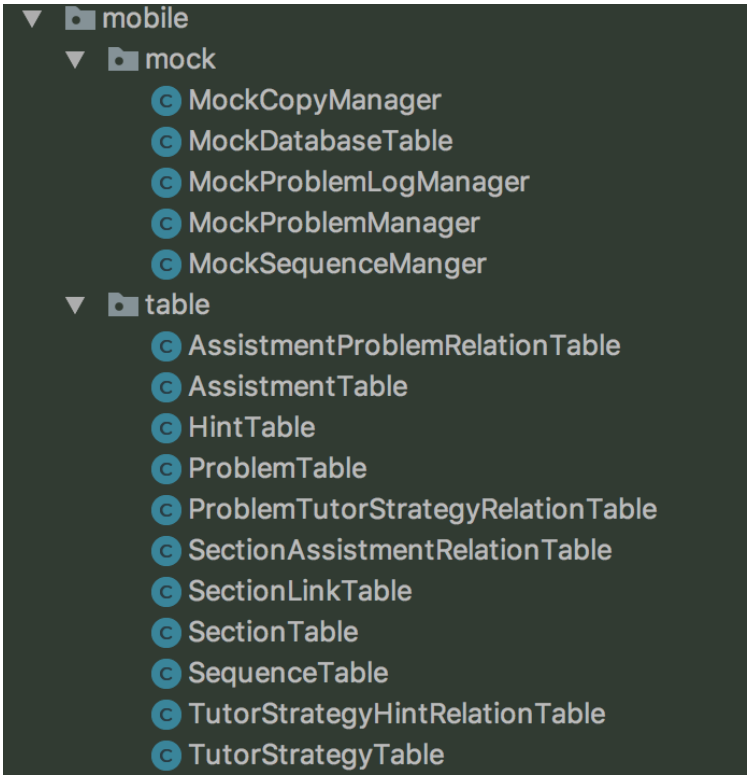

Figure 4: Mock services and in-memory database

Here is the sample context file for the unit tests:

```
// MobileProblemSetManagerTest-context.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

      <bean id="mock-sequence-manager"
        class="org.assistments.mobile.mock.MockSequenceManger"/>
      <bean id="mock-mobileProblem-log-manager"
        class="org.assistments.mobile.mock.MockProblemLogManager"/>
      <bean id="mobile-mobileProblem-set-manager"
        class="org.assistments.mobile.managers.impl.MobileProblemSetManagerImpl"/>
      <bean id="mobile-mobileProblem-manager"
        class="org.assistments.mobile.managers.impl.MobileProblemManagerImpl"/>
      <bean id="mobileProblem-manager"
        class="org.assistments.mobile.mock.MockProblemManager"/>
      <bean id="answer-manager"
        class="org.assistments.mobile.managers.impl.MobileAnswerManagerImpl"/>
      <bean id="mock-copy-manager"
        class="org.assistments.mobile.mock.MockCopyManager"/>
      <bean id="hint-manager"
        class="org.assistments.mobile.managers.impl.MobileHintManagerImpl"/>
      <bean id="tutor-strategy-manager"
        class="org.assistments.mobile.managers.impl
```

```
            .MobileTutorStrategyManagerImpl"/>
        <bean id="assistment-manager"
          class="org.assistments.mobile.managers.impl.MobileAssistmentManagerImpl"/>
        <bean id="sequence-table"
          class="org.assistments.mobile.table.SequenceTable"/>
        <bean id="section-table"
          class="org.assistments.mobile.table.SectionTable"/>
        <bean id="mobileProblem-table"
          class="org.assistments.mobile.table.ProblemTable"/>
        <bean id="tutor-strategy-table"
          class="org.assistments.mobile.table.TutorStrategyTable"/>
        <bean id="mobileProblem-tutor-strategy-relation-table"
          class="org.assistments.mobile.table.ProblemTutorStrategyRelationTable"/>
        <bean id="tutor-strategy-hint-relation-table"
          class="org.assistments.mobile.table.TutorStrategyHintRelationTable"/>
        <bean id="section-assistment-relation-table"
          class="org.assistments.mobile.table.SectionAssistmentRelationTable"/>
        <bean id="hint-table"
          class="org.assistments.mobile.table.HintTable"/>
        <bean id="assistment-mobileProblem-relation-table"
          class="org.assistments.mobile.table.AssistmentProblemRelationTable"/>
        <bean id="section-link-table"
          class="org.assistments.mobile.table.SectionLinkTable"/>
        <bean id="assistment-table"
          class="org.assistments.mobile.table.AssistmentTable"/>
        <bean id="mobile-section-manager"
          class="org.assistments.mobile.managers.impl.MobileSectionManagerImpl"/>
    </beans>
```

Here is a sample test case for problem set manager:

```
    // Java
    @Test
    public void getProblemsInSequenceTest() throws Exception {
        final Instant createdAt = Instant.now();
        final Instant updatedAt = Instant.now();

        final int SEQUENCE_ID = 384407;
        Section section = new Section();
        section.setType(Section.SectionType.PROBLEM.getTypeName());
        int headSectionId =  sectionTable.addRow(section);

        // Initialize database
        Sequence sequence = new Sequence();
        sequence.setHeadSectionId(headSectionId);
        sequence.setId(SEQUENCE_ID);
        sequence.setCreatedAt(createdAt);
        sequence.setUpdatedAt(updatedAt);
        sequenceTable.addRow(SEQUENCE_ID, sequence);

        final Integer[] DEFAULT_ASSISTMENT_IDS = {0, 1};
        sectionAssistmentRelationTable.addRow(headSectionId,
                                              DEFAULT_ASSISTMENT_IDS[0]);
        sectionAssistmentRelationTable.addRow(headSectionId,
```

```
                                                    DEFAULT_ASSISTMENT_IDS[1]);
25
        assistmentProblemRelationTable.addRow(DEFAULT_ASSISTMENT_IDS[0],0);
27      assistmentProblemRelationTable.addRow(DEFAULT_ASSISTMENT_IDS[1],1);

29      ProblemRow row1 = new ProblemRow(), row2 = new ProblemRow();
        row1.setRubyProblemType(FILL_IN);
31      row1.setId(0);

33      row2.setRubyProblemType(ALGEBRA);
        row2.setId(1);
35
        problemTable.addRow(0, row1);
37      problemTable.addRow(1, row2);

39      List<MobileProblem> mobileProblems = mobileProblemSetManager
            .getProblems(sequence);
41
        assertEquals(0, mobileProblems.get(0).getId());
43      assertEquals("Exact Match (case sensitive)",
            mobileProblems.get(0).getProblemType());
45
        assertEquals(1, mobileProblems.get(1).getId());
47      assertEquals("Algebraic Expression",
            mobileProblems.get(1).getProblemType());
49  }
```

2.6 Debugging

People asked on stack exchange whether they can always write "bug free" code [Ven]. Unfortunately, the answer is NO. Since the code may not always be perfect, effective debugging tools may help developers diagnostic the possible sources of the bugs and fix them efficiently. When I was working with the old app, it occasionally crashed due to network issue. I utilized Charles as HTTPS proxy to record the HTTPS request and corresponding response, and then eventually figured out the id of buggy sequences and fixed the incorrect implementation of the SDK. Here are screenshots of the proxy software:

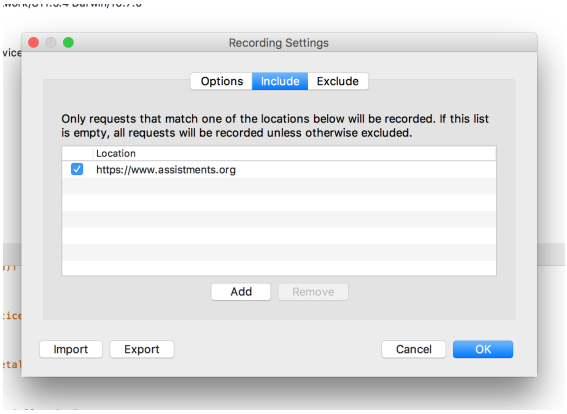

Figure 5: Setup host and path to record

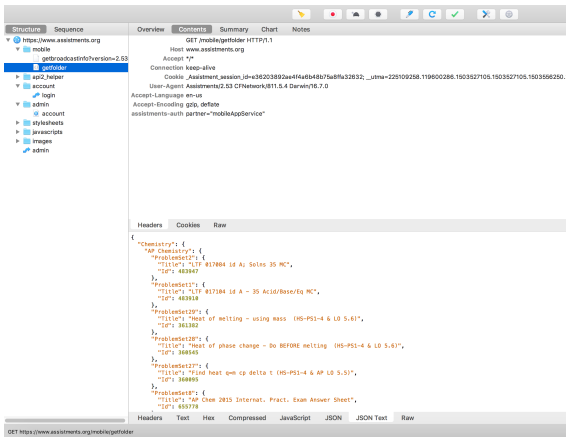

Figure 6: Show request and response JSON

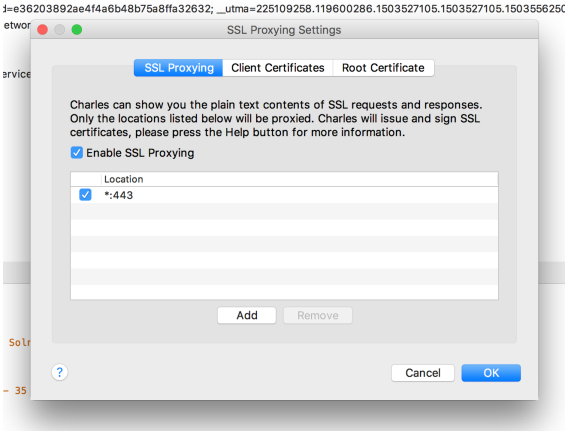

Figure 7: Enable HTTPS proxy

2.7 Crash Reporting and Statistics

When my supervisor and user were using the app on their devices, they found some random crashes. However, I, as developer, was trying to fix the bug but had no idea which lines of code cause the crash since I was not able to attach the debugger to users' devices. To ease the crashing reporting process, I integrated Crashlytics into app. This library will record the line number of code causing the problem and email me every time the app crash on users' devices. Then I would be notified, be able to locate the crash and fix the bug as soon as possible. Here are the email notification from Crashlytics (Figure 8) and the control panel of Crashlytics (Figure 9).

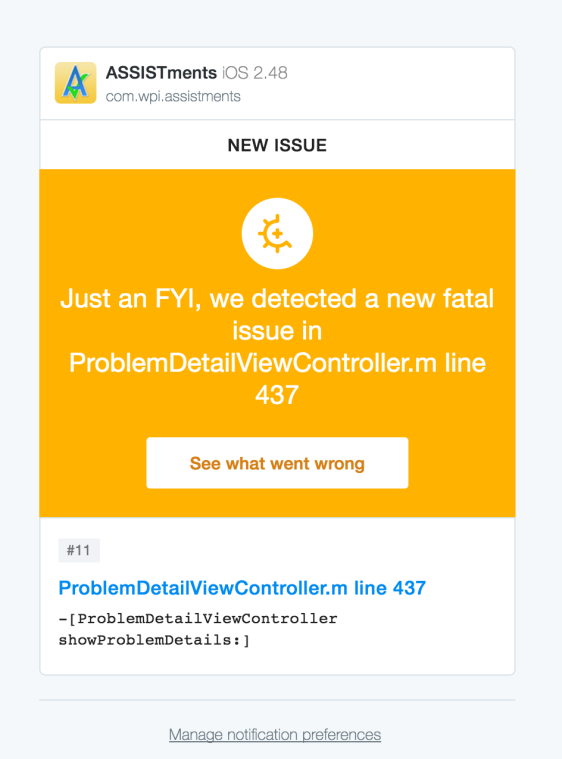

Figure 8: Crash notification email

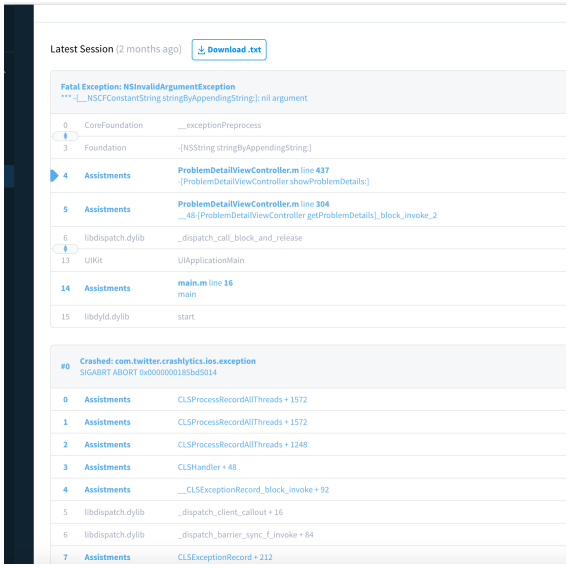

Figure 9: Control panel

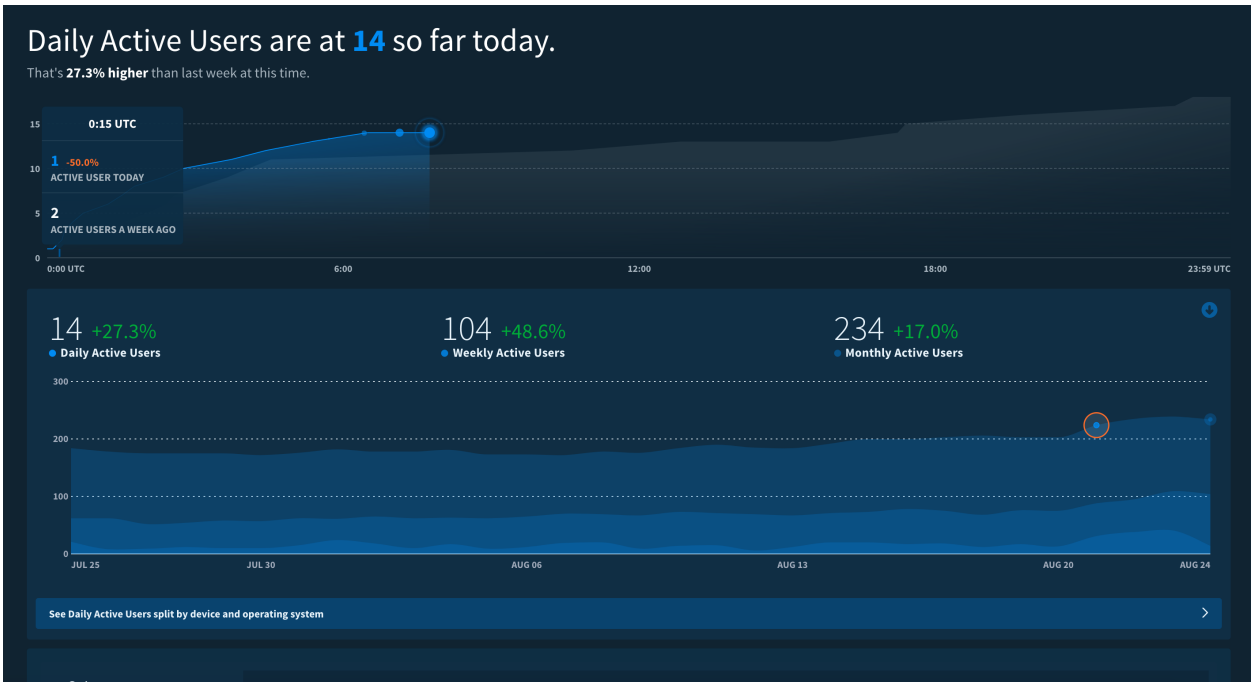

Figure 10: Active user statistics

Crashlytics also collected usage data for project manager and developers, letting the team monitor the number of active users and adoption rate of each version of the app (Figure 10).

2.8 User Experience

The original mobile app embedded a web view directly as the main interface (Figure 11). The web page is for desktop computers with no responsive layout or optimization for mobile. It's difficult for users to interact with it due to having very small font and buttons on the page. And it take 10 seconds to load the main interface, impatient users may not want to wait for that long to be able to use the app. To improve the usability and performance of the App, I redesigned the user interface (Figure 12) with Sketch and implemented part of them with delightful animations in swift. Now, it looks nicer, is much easier to interactive with and takes only milliseconds to load.

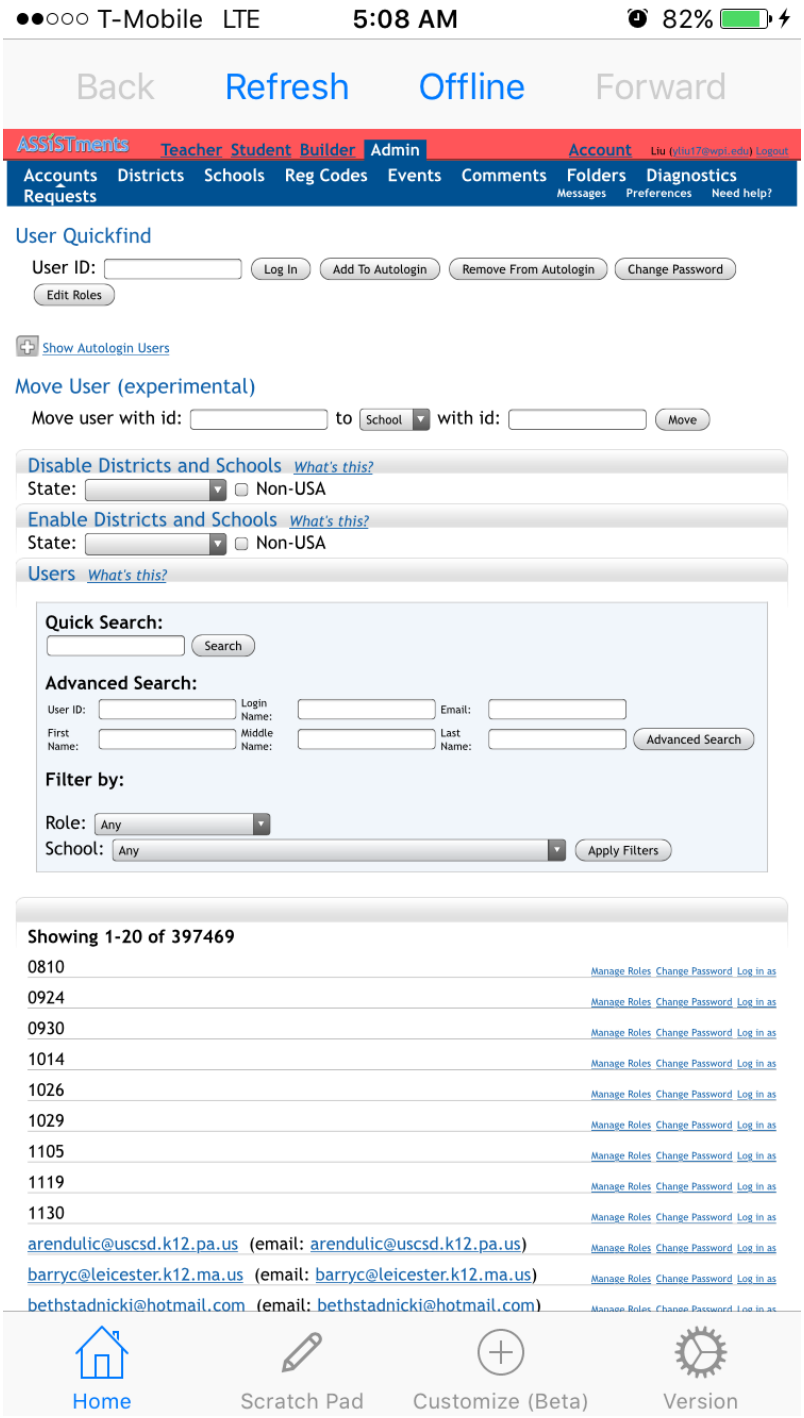

Figure 11: Original UI with Web View

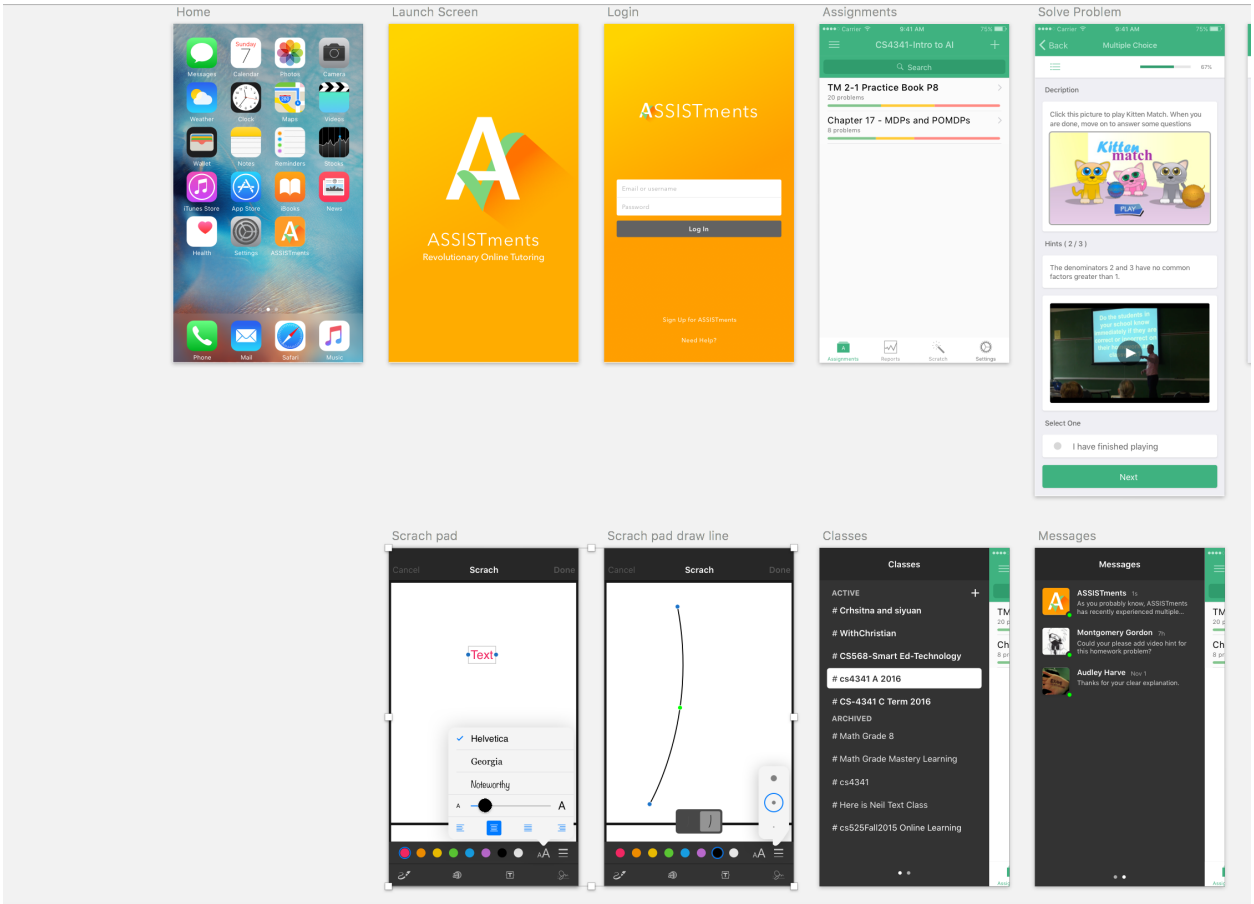

Figure 12: Redesigned UI

3 Conclusions

The APIs for the backend were successfully redesigned following RESTful API standards. The back end was fully refactored with object oriented design patterns. It was also tested with both thoughtful unit tests and specially design testing framework. So did the frontend. Both the backend and frontend code are now much readable, maintainable, reliable and enjoyable to work with. The UI of the App was fully redesigned specially for mobile devices and partially implemented. Now the app has better user experience and is much more enjoyable to use.

4 Future Works

Even though I already dedicated tons of hours to the app, there are still lot of things left to finished. The backend still has a little bit code left to clean up and has few malfunctioning test cases left to fix.

It is worth to mention than I didn't copy all the tutoring strategies of the original problem set when generating the experiment. New developers may have to finish this part of the implementation.

I was switching the front end code from Objective-C to Swift and implementing the Core Data service. They may also have to finish the service to fix the broken sign in screen and other feature relay on Core Data.

The *Remind me tomorrow* and *Never show up* buttons for soft update alert are not done yet even though the backend is already implemented. I also started the recording scratch pad drawing feature, which may also has to be finished in the future.

Lastly, I started a new app purely in Swift for the next generation of ASSISTments. Hopefully, it will be online in the future.

5 References

[Fut] Futurice. *iOS Good Practices*. URL: https://github.com/futurice/ios-good-practices.

[Nak] Ashod Nakashian. *Why Code Readability Matters*. URL: http://blog.ashodnakashian.com/2011/03/code-readability/.

[Shv] Alexander Shvets. *Design Patterns*. URL: https://sourcemaking.com/design_patterns.

[Ven] Joan Venge. *Is it possible to reach absolute zero bug state for large scale software?* URL: https://softwareengineering.stackexchange.com/questions/195571/is-it-possible-to-reach-absolute-zero-bug-state-for-large-scale-software.