March 2019

ASSISTments Cross-Platform Mobile App

Adam Sullivan Goldsmith
Worcester Polytechnic Institute

Benjamin Conner Emrick
Worcester Polytechnic Institute

ASSISTments Cross-Platform Mobile Application

An Interactive Qualifying Project
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

by
Ben Emrick and Adam Goldsmith

Date:
March 1, 2019

Submitted to:

Professor Neil Heffernan
Worcester Polytechnic Institute

Abstract

This Interactive Qualifying Project (IQP) delivers the ASSISTments web-based system to a mobile context through the creation of a cross-platform mobile application available for both Android and iOS devices. This improved mobile app serves as a replacement to the existing native ASSISTments apps that currently exist in the respective application distribution services. This cross-platform mobile solution significantly consolidates the size of the app by providing a single code base that deploys to both Android and iOS. While written utilizing an intermediary framework, the app maintains truly native app components in order to maintain user interface (UI) familiarity and deliver a seamless experience. Furthermore, this project structure unifies the mobile experience across different operating systems. The improved ASSISTments app includes a feature set in parallel with the existing applications while fixing prominent issues and making notable enhancements to UI appearance and user experience (UX) work flows.

Acknowledgments

We would like the thank Professor Neil Heffernan for offering the opportunity to build a revised ASSISTments mobile application. As the advisor of this IQP, he offered insight and guidance into the design and functionality of the application. Professor Heffernan orchestrated a collaboration with Oak Middle School in Shrewsbury, Massachusetts, in order to conduct user testing to evaluate the effectiveness of the system.

Thank you to Anthony Botelho, a PhD Candidate at WPI, who worked supervised this project and worked very closely with us throughout the IQP to offer feedback and ensure that the application follow ASSISTments standards. Anthony performed numerous administrative tasks such as granting us access to existing source code, distribution platforms, and connecting us with valuable development resources.

Cristina Heffernan and Cindy Starks were crucial in defining the user experience and providing direction for the overall UI.

Thank you to Chris Donnelly and David Magid for their contributions consisting of software development advice related to web technologies, and best practices within the ASSISTments system.

Thank you to Courtney Mulcahy, a teacher at Oak Middle School, for allowing us to visit and conduct user testing with her students.

Authorship

The sole authorship of this paper and the associated ASSISTments mobile app belong to Adam Goldsmith and Benjamin Emrick. The development of this application was completed exclusively at Worcester Polytechnic Institute (WPI) for ASSISTments, a free public service of Worcester Polytechnic Institute created by Neil and Cristina Heffernan.

List of Figures

List of Tables

List of Listings

Contents

1 Introduction

ASSISTments is a free homework and student tutoring service offered by WPI, primarily targeted at elementary and middle school students. It is primarily accessed via a web interface that that aims to intelligently replicate the behavior of human tutors in assisting student comprehension. The ASSISTments Tutor utilizes immediate feedback on assessments and other features to promote timely and calculated student-teacher interactions. This information streamlines the decision making process for teachers interested in adjusting educational strategies in order to accelerate student learning. ASSISTments leverages data and advanced metrics to equip teachers with actionable intelligence about the classroom environment. This computerized approach to education allows teachers and administrators to perform highly granular analysis that increases the efficiency and accuracy of instructional methodologies and assessment techniques. ASSISTments continues to research how emerging technologies and computing advancements can benefit the modern academic system.

Due to the increasing prominence of mobile devices in a highly digital society, ASSISTments plans to increase the accessibility of their platform through the delivery of an app. Several iterations of mobile apps were written in previous years to allow for several "native" features that could not be implemented entirely in the web interface. Features that are well-suited for mobile devices include a drawing pad for work submission, auto-login with stored credentials, and "offline mode". Offline mode enables students to download assignments within the Tutor and complete them without network connectivity. Optimizing ASSISTments for the mobile domain increases the potential of this educational tool through the platform's ability to garner attention and engagement.

1.1 Existing Apps and Infrastructure

Prior to this project, there were two separate apps for each platform, with different codebases but similar features. Each app was written using native iOS and Android development stacks, respectively. However, the Android app had been de-listed from the app store due to updated policies, while the iOS app had not been updated for approximately a year. After correcting the violation of the Google Developer Program Policy, the Android version was reinstated on the Play Store. The previously active Android app possesses several shortcomings. The login functionality is broken and always displays a "Login failed" alert with valid credentials. Therefore, the only method to enter the app includes circumventing the proper login work flow by selecting the "Register" label while already logged in. This will grant access to within the app with the desired account. Furthermore, Offline Mode, which allows students to download assignments to their device, complete them offline and then submit them once they regain connectivity, is non-functional. Once Offline mode is invoked on Android the app enters a seemingly infinite hanging phase that persists throughout the session. This loading state, which prevents user activity, often extends beyond the immediate session and interferes with successive app launches. Offline Mode renders the entire ASSISTments app deadlocked.

Additionally, several of the features of the iOS app were non-functional. There are numerous UI inconsistencies across Apple devices and the app frequently encounters a state that forces unexpected crashes. The most unsatisfactory element of both the Android and

iOS app is the inability to save the user's login credentials. While both apps contain a UI element for remembering the account credentials on the login screen, this feature continuously fails. Due to this defect, users must enter their username and password every time the app launches, which significantly thwarts a positive user experience.

After completing a discovery of the existing ASSISTments apps, we identified multiple areas of improvement. Through a series of consultations with Professor Heffernan and Anthony Botelho, we collectively defined a direction for design and development. The foremost priority was to enhance the general performance of the app through minimizing crashes and unexpected conditions that renders the interface inoperative. This requires an emphasis on reliability and stability. Additionally, we aim to develop a more reactive system that behaves faster and reduces latency. A major benefit of a cross-platform app is the improvement in maintainability. A single codebase requires far fewer resources and knowledge to service and than independent native applications. These non-functional requirements guided the UX design process and directly impacted the implementation of presentation and backend logic.

1.2 Features

The designated feature set for the cross-platform app parallels that of the existing app. Adjustments to previous logic will be necessary in order to achieve a more performant and stable foundation. We will introduce complementary features to support the primary functionality. There were several high-level features that were required for the app to be useful:

- basic webview functionality
- login credential saving
- offline mode: There must be an interface for users to download a problem, access the tutor while offline, and submit assignments once they return online
- image upload: submit images from the device photo library on assignment questions with a show work option
- scratch pad: upload work from an in-app scratch pad on assignment questions with a show work option
- crash and custom event logging with notifications

1.3 Tools and Technology

There are a number of cross platform mobile frameworks available. These can generally be separated into three categories: JavaScript in a WebView (ie non-Native JavaScript), Native JavaScript, and other languages. The most popular options are shown in Table 1.

The considered cross-platform frameworks featured a variety of different base languages. Since JavaScript is used throughout the web app, we contend that consistency with existing technologies is preferable. Through a JavaScript framework, we are also able to utilize the robust set of non-browser dependent JavaScript libraries. Therefore, Xamarin and Flutter were ruled out because C# and Dart would be new languages to the project. Also, the

Framework	Native?	Language	Backing	JS Frameworks	Release date
Cordova	No	JavaScript	Apache		2009
PhoneGap	No	JavaScript	Adobe		2009
Ionic	No	JavaScript	Ionic		2013
React-Native	Yes	JavaScript	Facebook	React, Vue	May 2013
Nativescript	Yes	JavaScript	Progress Software	Angular, Vue	2014
Xamarin	Yes	C#	Microsoft	N/A	Feb 2013
Flutter	Yes	Dart	Google	N/A	Dec 2018

Table 1: Comparison of Cross-Platform Mobile Frameworks

pursuit of cross-platform development should not come at the expense of form and function. From the remaining options, React-Native and NativeScript are the only frameworks built on native Android and iOS ecosystems ("Xamarin vs React Native", 2018)[3]. This allows these frameworks to avoid using a WebView-based wrapper, which frameworks such as Cordova and its derivatives PhoneGap and Ionic require. Thus, Cordova, PhoneGap, and Ionic were ruled out due to the complexity of the ecosystem, and the fact that they merely recreate native behavior. While native rendering frameworks require more knowledge of iOS and Android systems, the ability to implement platform specific UIs with familiar visual components significantly increases user comfort and overall satisfaction of the app (Stoychev, 2017)[[1]. Native visual elements are also optimized for the mobile operating system. Through an expansive set of custom UI widgets, NativeScript aims to achieve full cross-platform code reuse. NativeScript projects share roughly 90% of the source code across platforms compared to 70% in React-Native projects ("Xamarin vs React Native", 2018)[3]. This, in turn, demands more effort to fashion independent visual layout's across unique platforms. However, unison is a priority in the ASSISTments cross-platform app. This deems a framework that advocates for platform-specific interface generation undesirable relative to a streamlined UI implementation. NativeScript is a rapidly growing framework with an expanding community and open-source presence. Despite its lack of maturity compared to React-Native, NativeScript maintains a robust collection of plugins. These additives greatly facilitate development and increase the capabilities of the app. There are several available plugins that are relevant to the ASSISTments feature set including *crashlytics* support, *media file picker* for image upload and *secure storage* for account credential security.

Lastly, we analyzed React-Native and NativeScript's support of front-end frameworks. Due to time constraints in a 7-week development period, we favored a lightweight framework. Vue.js is a progressive framework that offers a powerful engine for building complex systems. Vue.js supports single-file components, which reduces the division of code and facilitates a simple and clean syntax that will reduce the size and complexity of the source code. This is imperative for sustaining high readability and a serviceable application (Torkut, n.d)[2]. NativeScript includes a *nativescript-vue* plugin which serves as an intermediary between Nativescript components and the Vue.js virtual DOM. This support is unparalleled in the cross-platform mobile application community. Furthermore, this plugin permits NativeScript projects to utilize single-file Vue components, which are well-suited for the ASSISTments app. Ultimately, NativeScript's advanced support for Vue.js, compatible plugins, and truly

native ecosystem render it the most practical framework for development.

2 Methodology

Initially, development was planned to be done in a agile-like manner, with weekly sprints resulting in a testable demo app. However, due to somewhat optimistic expectations about the times to get a prototype functional and in the two app stores, there was not a testable beta for Android in the Google Play Store until the fourth week.

2.1 Version Control

Git was used for version control as it allowed us to use a distributed branching development strategy while still being able to commit back to the ASSISTments Subversion server using 'git-svn'. Due to this, we could develop features independently while maintaining functional code for further development. For example, this allowed us to keep the "work offline" feature in a separate branch until it was stable enough to merge, as it required some changes that broke other core functionality to function. A private GitHub repository was used to aid in synchronization of the code between developers.

2.2 Development Life Cycle

We structured the requirements into GitHub "issues" and used a GitHub Project for Kanban-style project tracking. Each of us would pick tasks to work on, ranked by perceived importance and difficulty. The tasks can be seen in Appendix E.

2.3 User Testing Procedure

User testing is a valuable component of the software development lifecycle. This process allows for developers to gauge user experience and better understand the performance and compatibility of the application in a realistic context. This procedure aims to uncover information that is vital to user acceptance and likeability that may otherwise be unapparent. It is critical to define a standardized testing procedure in order to and prevent variability and optimize the results.

User testing will be conducted through an in-person session including a test subject, facilitator and an observer. Both the observer and facilitator assume administrative roles as individuals with significant insight into the functionality and design of the featured application. If there is only one administrator present for the testing session, then they may serve as both the facilitator and observer. The test subject must possess limited knowledge of the applications capabilities relative to the administrators. Furthermore, the subject should be unaffiliated with the direct development and/or design of the software. The test will utilize the subject's preferred personal mobile device. If the subject does not possess a personal device then they will be provided with a suitable iPad Mini with the application installed. A beta version of the application must be downloaded to the subject's personal device via the respective application distribution service. Furthermore, existing ASSISTments users will

conduct the testing session with their personal account. The Teacher overseeing the class is responsible for assigning a sample assignment that will be upgraded for the students to complete. If the subject does not have a previously created account they will proceed with the ASSISTments test account. The ASSISTments test account assumes the Student role within the platform and will contain simple assignments for testing purposes.

It is the facilitator's duty to first present a brief introduction of the application. They will continuously prompt the test subject to complete desired tasks throughout the experience. Upon issuing the introductory description of the app and the testing environment, the facilitator should encourage the subject to think aloud through the experience. This requires the test subject to verbally announce their thought process underlying each decision and express their perception. In order to maintain the integrity and genuine nature of the test, the facilitator must not influence or coerce the subject in any manner. The facilitator must avoid providing explicit instructions on how to complete the assigned task. They must refrain from offering direction concerning how to navigate or operate any aspect of the application. They may not offer help or interfere with the test subjects instinctual usage of the application. The facilitator's role is to merely prompt action without offering any form of guidance. If the administrators of the test have not selected any specific tasks for the subject to complete, the facilitator may instead instruct the user to use the app as freely as possible.

The role of the observer is to collect information throughout the entire duration of the testing session. The observer is responsible for recording the sequence of clicks, movements, and interactions with the user interface. They must take detailed notes of the overall experience and any record any alarming tendencies or behaviors. They may record comments on the usability and general satisfaction of the application inferred from certain behaviors or audible feedback. They are not responsible for interacting with the test subject in any capacity. It is necessary that they are positioned such that both the device screen and user are visible. This is crucial for understanding in-app actions and allows for analysis of the user's expressions, which serve as subtle indicators of judgment.

Prior to the testing session, the administrators may select specific components, features, and/or workflows that they wish to collect information about. The granularity of such tasks may range from testing the effect of a layout orientation to the design of an entire in-app process. The pre-selected features of the ASSISTments mobile app that were evaluated during user tests include login, remember user credentials, web view navigation, offline assignment completion and submitting a bug report.

To initiate the test the observer will record the usage conditions including the mobile phone operating system, manufacturer and the user's ASSISTments role if the subject is an existing ASSISTments user. The facilitator will then describe the format of the testing session, offer a brief explanation of the platform for those unfamiliar. The facilitator will then inform the user that they are encouraged to communicate their thought process and internal reactions in a clear and honest manner throughout the testing. Once the user has agreed to the conditions and understands the format, they will be asked to complete a series of pre-determined tasks or requested to use the app freely. Meanwhile, the observer will record all meaningful interactions with the application and note tendencies, common behaviors, and subtle details that provide insight into the user experience. The subject will be instructed to complete the following tasks:

1. On first open of the app - login with your account credentials and select for the app to remember your username and password
2. On successful login and load of web view - force close the application
3. Re-open the app
4. Complete an assignment and explore/navigate the web view
5. Complete an assignment in offline mode
6. Upload an image for show work
7. Logout
8. Clear the username and password fields

After the completion of the assigned tasks and any other operations deemed necessary by the administrators, the subject will be requested to share immediate feedback, although this is not required. At the conclusion of the session, the subject will be presented with an opportunity to complete a survey. The survey is not mandated and may be completed outside of the testing environment at the subject's convenience. It is suggested that the survey is submitted within two (2) days of the testing session in order to preserve an accurate recollection of the event.

3 Implementation

3.1 Overall Structure

The project's structure, shown in Listing 1 All of the code for the app is found in the `app` directory. The entry point for the app is `app.js`, which contains Nativescript plugin registration and setup for Vue's Devtools. Most of the app is contained in Vue single file components (SFCs) in the `components` directory. There is also a `mixins` folder which contains code that is used across multiple SFCs. There is a constants file at `constants.js` that currently just contains a mapping of URLs. All of the code injected into the webview can be found at `local.js`. Most of the common style in the app can be found in `_app-common.scss`, and the style's colors can be set in `_app_variables.scss`. Each SFC can also contain its own style.

3.1.1 Single File Components

The use of single-file components was in line with the goal of a more concise code base with a simple logical structure. Vue offers the ability to couple related information in a single location in order to maximize development efficiency. Through coupling an entity's structure, style and behavior in a single location it becomes transportable across the system and contributes to a modular project structure. This is exemplified in the implementation of icons defined in `IconButton.vue` visible in Listing 2. The `<script>` supports the acceptance

```
./
├── .editorconfig
├── .gitignore
├── .prettierrc
├── README.md
├── app
│   ├── App_Resources
│   │   └── [omitted] ...
│   ├── _app-common.scss
│   ├── _app-variables.scss
│   ├── app.js
│   ├── app.scss
│   ├── attribution.txt
│   ├── components
│   │   ├── App.vue
│   │   ├── Attribution.vue
│   │   ├── CreateAccount.vue
│   │   ├── Home.vue
│   │   ├── IconButton.vue
│   │   ├── LoginScreen.vue
│   │   ├── ReportBug.vue
│   │   ├── ScratchPad.vue
│   │   ├── Settings.vue
│   │   └── WebViewWrapper.vue
│   ├── constants.js
│   ├── fonts
│   │   └── FontAwesome.ttf
│   ├── local.js
│   ├── mixins
│   │   └── networkConnected.js
│   └── package.json
├── firebase.nativescript.json
├── jsconfig.json
├── package-lock.json
├── package.json
├── patches
│   └── @nota
│       └── nativescript-webview-ext+5.0.3.patch
└── webpack.config.js
```

Listing 1: Project Structure

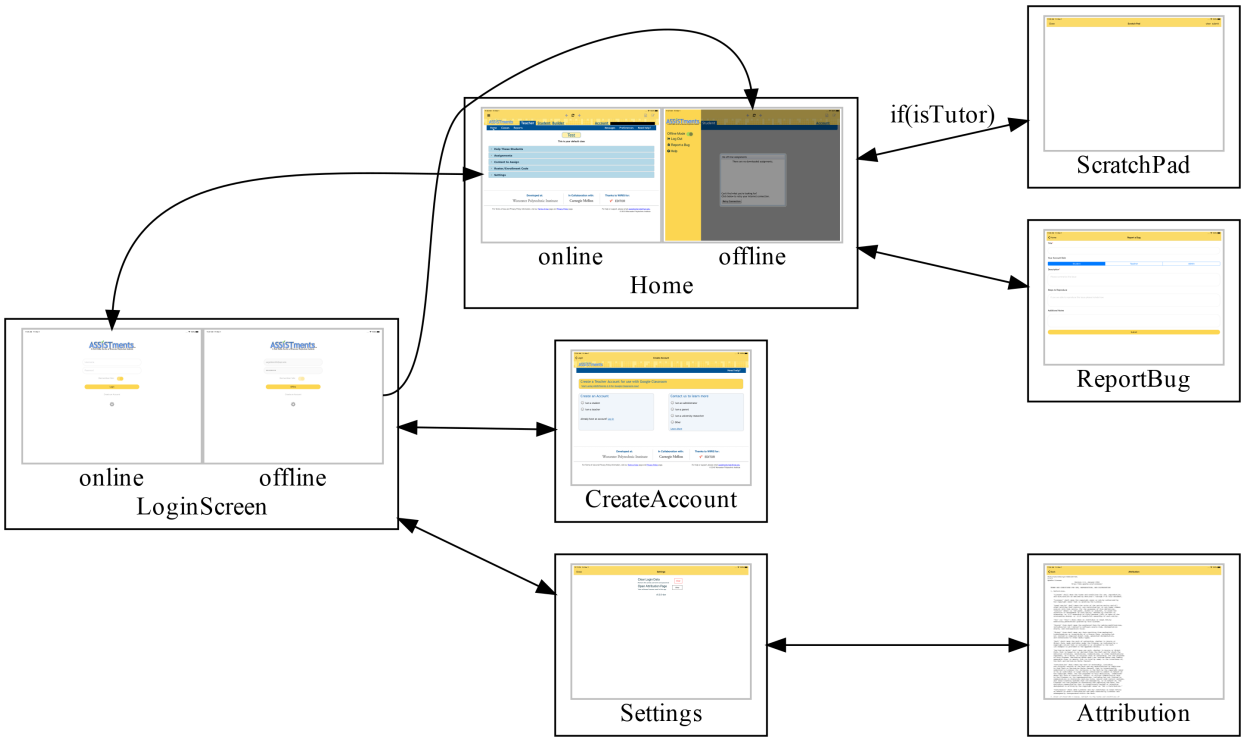


Figure 1: Navigation Flow

of data from an external source that impacts the presentation of the icon determined in `<template>`. The appearance is set in `<style>` which applies a default style that icons will inherit. Appearance can be overridden elsewhere in the app depending on design requirements.

The IconButton component can now be used for templating in other components. The icons nested in the RadSideDrawer in Home.vue assume the same form so they can be efficiently implemented with individualized parameters as shown. Abstracting reusable portions of code enforces consistency, limits redundancy and yields an easy to manage adaptive structure.

The `IconButton` component can now be used for templating in other components. The icons nested in the `<RadSideDrawer>` in `Home.vue` assume the same form so they can be efficiently implemented with individualized parameters as shown Listing 3. Abstracting reusable portions of code enforces consistency, limits redundancy and yields an easy to manage adaptive structure.

3.1.2 Code Style

The project includes `.editorconfig` and `.prettierrc` files to define the style of code. This aided in merging code, as it reduced the possible differences in representation of the code,

```vue
<template>
  <Button
    v-on="$listeners"
    class="fa iconButton"
    v-bind="{ 'text.decode': text }"
    :isEnabled="!!isEnabled"
  />
</template>

<script>
  export default {
    props: { text: String, isEnabled: { type: Boolean, default: true } },
  };
</script>

<style scoped lang="scss">
  @import '../app-variables';
  .iconButton {
    font-size: 25%;
    horizontal-align: left;
    vertical-align: center;
    background-color: transparent;
    z-index: 0; // fix for android button shadow

    &[isEnabled='false'] {
      color: $disabled;
    }
  }
</style>
```

Listing 2: IconButton Definition [from `components/IconButton.vue`]

```
<portal to="sidedrawer" tag="ScrollView">
   <StackLayout>
     <StackLayout class="sidedrawer-list-item hstack">
       <Label class="label" text="Offline Mode" />
       <Switch :isEnabled="networkConnected === offline" v-model="offline" />
     </StackLayout>

     <IconButton
       class="sidedrawer-list-item"
       @tap="logout"
       text="&#xf08b; Log Out"
     />
     <IconButton
       class="sidedrawer-list-item"
       @tap="report"
       text="&#xf188; Report a Bug"
     />
     <IconButton
       class="sidedrawer-list-item"
       @tap="help"
       text="&#xf059; Help"
     />
   </StackLayout>
 </portal>
```

Listing 3: IconButton as a Template Component [from `components/Home.vue`]

3.2 Login and Cookie Syncing

The first version of the login code, shown in Listing 4 tried to login by injecting code into the login page, filling out the login form, and submitting it.

```
assistments_login(event) {
  let webview = event.object;

  // TODO: handle cases where analytics fails to load (ie adblocker)
  webview
    .loadUrl('https://www.assistments.org/account/login')
    .then(e => {
      if (e.url === 'https://www.assistments.org/account/login') {
        // try to login
        e.object.executeJavaScript(
          `login.value = ${JSON.stringify(this.username)};
             password.value = ${JSON.stringify(this.password)};
             remember_me.checked = true;
             document.querySelector('.login_window form').submit()`
        );
      }
    })
    .catch(err => console.error(err));
},
```

Listing 4: Login with Injected JS [from `components/Home.vue`]

That code is ugly and slow, as it requires loading and rendering the login page. Additionally, the only way it can detect a successful/failed login is by checking the next page navigated to in the webview, which requires keeping track of more state in `Home`, which was not ideal. This was therefore replaced with the code in Listing 5. This version is located in `LoginScreen`, leading to better code organization and readability, as well as more consistent user experience. This solution is also faster and allows much better detection of failure states (network error vs incorrect credentials, etc). Additionally, it does the login outside of the webview, which would allow us to interact with the webapp from the same session without injecting js, as the same session and auth cookies are set in both places. However, it does require syncing cookies into the webview, as they are separate on both platforms.

On Android this was relatively easy, and can be seen in Listing 6. This can be done right after first initialization of the webview, and works perfectly.

On iOS, however, there is a bug affecting cookie synchronization into a WKWebView. Generally speaking, the bug seems to cause the callbacks of the `WKWebsiteDataStore`'s cookie storage to either return early or never return after the data store is associated with a webview. This manifests itself in the current code as the inability to login more than once in a single session on iOS. The first implementation of a workaround for this bug was implemented in a patch to the `@nota/nativescript-webview-ext` package shown in Listing 7. This workaround simply sets all of the cookies in a new datastore before creation of the webview. However, this implementation used a non-persistent datastore, meaning that the solution

```
// ... lines reformatted for readability
let login_response;
try {
  login_response = await httpRequest({
    url: URLS.login,
    method: 'POST',
    dontFollowRedirects: true, // allow us to capture the redirect
    timeout: 2000,
    headers: { 'Content-Type': 'application/x-www-form-urlencoded;charset=UTF-8' },
    content: this.encodeFormBody({
      login: this.username,
      password: this.password,
      remember_me: 1,
      commit: 'Log+in',
    }),
  });
} catch (err) {
  this.logging_in = false;
  alert({ title: 'Login Failed!', message: err.toString(), okButtonText: 'OK' });
  return;
}

// a redirect = probably sucessful login
// TODO: check if this assumtion always holds true. We
// could also check cookies instead
if (login_response.statusCode !== 302) {
  this.logging_in = false;
  alert({ title: 'Login Failed!', message: 'Username or password incorrect',
   ↪  okButtonText: 'OK' });
  return;
}
// ... some code omitted
this.$navigateTo(Home, {
  clearHistory: true,
  props: { initialURL: login_response.headers.Location, username: this.username },
});
```

Listing 5: Login with Cookie Syncing [from `components/LoginScreen.vue`]

```
// allow 3rd party cookies
const cookieMgr = android.webkit.CookieManager.getInstance();
cookieMgr.setAcceptThirdPartyCookies(webview.android, true);

// manually sync cookies
const cookieHandler = java.net.CookieHandler.getDefault();
if (cookieHandler) {
  const cookies = cookieHandler.getCookieStore().getCookies();
  Array.from(cookies.toArray()).forEach(c =>
    cookieMgr.setCookie('www.assistments.org', String(c))
  );
  cookieMgr.flush();
}
```

Listing 6: Cookie Syncing on Android [from `components/WebViewWrapper.vue`]

was quite incompatible with offline mode, as offline mode needs to save to the app cache and localstorage to function.

The current implementation moves the code into the `attemptLogin` method of `LoginScreen`, and is shown in Listing 8. This implementation tries to force setting the cookies using JavaScript's `Promise` and `async/await` features to wait until all of the cookies are set, as `defaultDataStore` seems to be slower than `nonPersistentDataStore`, probably because it needs to save the data to the phones storage instead of just RAM. However, due to the bug the `setCookieCompletionHandler` function will not ever call its callback, causing the login to hang. This is currently "worked around" by popping up an alert on the second login, informing users that they must close the app to login again.

```
--- a/node_modules/@nota/nativescript-webview-ext/webview-ext.wkwebview.js
+++ b/node_modules/@nota/nativescript-webview-ext/webview-ext.wkwebview.js
@@ -162,6 +162,20 @@ var WKWebViewWrapper = (function () {
         configuration.setValueForKey(true, "allowUniversalAccessFromFileURLs");
         this.wkCustomUrlSchemeHandler = new CustomUrlSchemeHandler();
         configuration.setURLSchemeHandlerForURLScheme(this.wkCustomUrlSchemeHandler,
         owner.interceptScheme);
+
+        configuration.websiteDataStore = WKWebsiteDataStore.nonPersistentDataStore();
+        let cookies = NSHTTPCookieStorage.sharedHTTPCookieStorage.cookies;
+
+        function setCookie(cookie) {
+          return new Promise(resolve => {
+            configuration.websiteDataStore.httpCookieStore.setCookieCompletionHandler(
+              cookie,
+              resolve
+            );
+          });
+        }
+        Promise.all(Array.from(cookies).map(c => setCookie(c)));
+
         var webview = new WKWebView({
             frame: CGRectZero,
             configuration: configuration,
```

Listing 7: Cookie Syncing Code on iOS with patch [from `patches/@nota/nativescript-webview-ext+5.0.2.patch`]

```
// Sync iOS cookies must be done before webview is created to
// work around a bug introduced in 11.3
let cookieStore = WKWebsiteDataStore.defaultDataStore().httpCookieStore;
let cookies = NSHTTPCookieStorage.sharedHTTPCookieStorage.cookies;
await Promise.all(
  Array.from(cookies).map(
    c => new Promise(res => cookieStore.setCookieCompletionHandler(c, res))
  )
);
```

Listing 8: Cookie Syncing Code on iOS [from `components/LoginScreen.vue`]

3.2.1 Offline

Initially, the support for offline mode was just a button on the login screen and a toggle that would bring you to the offline assignments listing page (`https://www.assistments.org/assistments/student/index.html#offlineUserAssignmentList/`). After our first user test, we decided that it would be better for it to automatically detect the network state, and prompt the user to switch into offline mode. To this end, we wrote a mixin (Listing 10) that provided the network state, as the Nativescript `connectivity` module did not allow for multiple handlers. With this mixin in place, we could just react to changes in the networkConnected variable. On the login screen, losing network connection disables the login fields, and switches the "Login" button to an "Offline" button. On the Home screen this prompts the user to switch to offline mode immediately. If the user does not switch, they can then later switch with the toggle in the side drawer. The toggle is disabled when the states match (ie offline with no network connection or online with a network connection) as it otherwise seemed to only add confusion.

The other part of offline support is enabling localstorage and app cache support. On iOS, this is enabled by default, as long as the you are using the default datastore. On Android, this is done by setting `domStorage="true"` on the WebViewExt, and the code in Listing 9.

```
1 settings.setAppCacheEnabled(true);
2 settings.setAppCachePath(androidApp.context.getCacheDir().getAbsolutePath());
```

Listing 9: Android Application Cache Configuration [from `components/WebViewWrapper.vue`]

3.2.2 Login Saving

To save login information across app starts, we used the `nativescript-secure-storage` plugin, which provides an abstraction around the native secure storage methods on the two platforms. The code for this can be seen in Listing 12 and Listing 11

```
import * as connectivity from 'tns-core-modules/connectivity';

const callbacks = [];

connectivity.startMonitoring(connection => {
  for (let callback of callbacks) {
    callback(connection);
  }
});

export default {
  data() {
    return {
      networkConnected: false,
    };
  },

  created() {
    callbacks.push(this.checkConnected);
    this.checkConnected(connectivity.getConnectionType());
  },

  destroyed() {
    let index = callbacks.indexOf(this.checkConnected);
    if (index > -1) callbacks.splice(index, 1);
  },

  methods: {
    checkConnected(connection) {
      switch (connection) {
        case connectivity.connectionType.none:
          console.error('No connection');
          this.networkConnected = false;
          break;
        case connectivity.connectionType.mobile:
          console.error('Mobile connection');
          this.networkConnected = true;
          break;
        default:
          console.error('Non-mobile connection');
          this.networkConnected = true;
          break;
      }
    },
  },
};
```

Listing 10: Network Connectivity Mixin [from `mixins/networkConnected.js`]

```
// in created()
this.username = this.secureStorage.getSync({ key: 'username' }) || '';
this.password = this.secureStorage.getSync({ key: 'password' }) || '';

if (this.username && this.password) {
  this.remember = true;
  if (!this.dont_autologin && this.networkConnected) this.attemptLogin();
}
```

Listing 11: Getting Stored Credentials with SecureStorage [from `LoginScreen.vue`]

```
// after successful login
if (this.remember) {
  this.secureStorage.setSync({ key: 'username', value: this.username });
  this.secureStorage.setSync({ key: 'password', value: this.password });
}
```

Listing 12: Saving Login Information with SecureStorage [from `LoginScreen.vue`]

3.3 Webview Specifics

There are a number of things that any web browser will have, but must be implemented manually in a webview-based app.

3.3.1 JS Dialogs

First is the JavaScript dialogs: `alert()`, `confirm()`, and `prompt()`. These turned out to be relatively easy, but just another thing to implement. Additionally, Nativescript did not provide a wrapper, so this had to be done per platform. The iOS code is shown in Listing 14, and the Android code is shown at Listing 13.

3.3.2 Page Scaling

Another issue that is typically solved automatically is the scaling of the webpage. ASSISTments' website is very much designed for desktop use, and is not terribly reactive, with lots of absolutely positioned elements and other elements that flow poorly.

On Android, the WebView has a number of settings that were set, as seen in Listing 15, which set it to load zoomed out to fit all content, use a "wide" viewport instead of the website's viewport, and try to expand text a bit to improve readability.

On iOS, a similar setup is actually enabled by default, but was not working due to an issue with the WebViewExt plugin. The plugin, by default, sets a viewport on every page. This would normally be helpful, but in this case it made the site behave poorly. This was fixed by patching the package, as shown in Listing 16.

```
// ... lines reformatted for readability
let myWebChromeClient = android.webkit.WebChromeClient.extend({
  onJsAlert(view, url, message, result) {
    alert({title: 'Alert', message: message, okButtonText: 'OK'})
      .then(r => (r ? result.confirm() : result.cancel()));
    return true;
  },

  onJsConfirm(view, url, message, result) {
    confirm({title: 'Confirm', message: message,
             okButtonText: 'OK', cancelButtonText: 'Cancel'})
      .then(r => (r ? result.confirm() : result.cancel()));
    return true;
  },

  onJsPrompt(view, url, message, defaultValue, result) {
    prompt({message: message, okButtonText: 'OK',
            cancelButtonText: 'Cancel', defaultText: defaultValue})
      .then(r => (r.result ? result.confirm(r.text) : result.cancel()));
    return true;
  },
});
webview.android.setWebChromeClient(new myWebChromeClient());
```

Listing 13: Android JS Dialogs [from `components/WebViewWrapper.vue`]

3.3.3 Webview Navigation Buttons

The navigation buttons were originally placed in the sidedrawer, but moved into the action bar after our first user test, as the user did not find the sidedrawer. The functions for this were provided by Nativescript's WebView, so this was just a matter of binding buttons (Listing 17).

3.4 SideDrawer

A sidedrawer was implemented early on in the project due to a desire to consume less of the screen space with buttons than the previous app. The previous app had a bottom tab bar that switched between screens, but consumed rather a lot of space for something that was rarely used. To this end, we put much of the functionality there, as well as some aditional functions, into a sidedrawer that would only be present in in the Home screen. However, due to a somewhat odd structuring requirement, it had to be present at the top level of the app.

NativeScript's navigation is based around `Frame`s and `Page`s. A call to `vm.$navigateTo` switches the `Page` displayed in the current `Frame`. `Page`s also contain `ActionBar`s and set the statusbar's color. A `Frame` can only contain `Page`s.

The official sidedrawer component, `RadSideDrawer`, displays above its children, so the `Page` needs to be inside the sidedrawer. However, since `Frame`s can only be navigated to `Page`s, there would need to be a `Page` both around and inside the `RadSideDrawer`. This was done in an earlier implementation, but worked rather poorly due to the presence of two

```
// ... lines reformatted for readability
let myWKUIDelegate = NSObject.extend(
  {
    webViewRunJavaScriptAlertPanelWithMessageInitiatedByFrameCompletionHandler(
      webView, message, frame, completionHandler) {
      alert({title: 'Alert', message: message, okButtonText: 'OK', b})
        .then(() => completionHandler());
    },
    webViewRunJavaScriptConfirmPanelWithMessageInitiatedByFrameCompletionHandler(
      webView, message, frame, completionHandler) {
      confirm({title: 'Confirm', message: message,
               okButtonText: 'OK', cancelButtonText: 'Cancel'})
        .then(completionHandler);
    },
    webViewRunJavaScriptTextInputPanelWithPromptDefaultTextInitiatedByFrame
     ↪  CompletionHandler(
      webView, prompt, defaultText, frame, completionHandler) {
        prompt({message: prompt, okButtonText: 'OK',
                cancelButtonText: 'Cancel', defaultText: defaultText})
        .then(result => completionHandler(result.result ? result.text : null)
      );
    },
  },
  { protocols: [WKUIDelegate] }
);

webview.ios.UIDelegate = new myWKUIDelegate();
```

Listing 14: iOS JS Dialogs [from `components/WebViewWrapper.vue`]

```
// force loading page zoomed out
webview.android.setInitialScale(1);
const settings = webview.android.getSettings();
settings.setLoadWithOverviewMode(true);
settings.setUseWideViewPort(true);
settings.setLayoutAlgorithm(
  android.webkit.WebSettings.LayoutAlgorithm.TEXT_AUTOSIZING
);
// disable zoom controls
settings.setDisplayZoomControls(false);
```

Listing 15: Android Zoom Settings [from `components/WebViewWrapper.vue`]

```
--- a/node_modules/@nota/nativescript-webview-ext/webview-ext.wkwebview.js
+++ b/node_modules/@nota/nativescript-webview-ext/webview-ext.wkwebview.js
@@ -346,12 +360,7 @@ var WKWebViewWrapper = (function () {
     };
     WKWebViewWrapper.prototype.loadWKUserScripts = function (autoInjectJSBridge) {
         if (autoInjectJSBridge === void 0) { autoInjectJSBridge = this.autoInjectJSBridge; }
-        if (!this.wkUserScriptViewPortCode) {
-            this.wkUserScriptViewPortCode = this.makeWKUserScriptPromise(nativescript_webview_bridge_loader_1.metadataViewPort);
-        }
         this.wkUserContentController.removeAllUserScripts();
-        var wkUserScriptViewPortCode = this.wkUserScriptViewPortCode;
-        this.addUserScriptFromPromise(wkUserScriptViewPortCode);
         if (!autoInjectJSBridge) {
             return;
         }
```

Listing 16: Disabling viewport injection in WebViewExt on iOS with patch [from `patches/@nota/nativescript-webview-ext+5.0.2.patch`]

```
<IconButton
  @tap="webview.goBack()"
  :isEnabled="webview && webview.canGoBack"
  text="&#xf060;"
/>
<IconButton @tap="webview.reload()" text="&#xf021;" />
<IconButton
  @tap="webview.goForward()"
  :isEnabled="webview && webview.canGoForward"
  text="&#xf061;"
/>
```

Listing 17: Webview Navigation Buttons [from `components/Home.vue`]

`Page` elements. Since the top level `ActionBar` had to be disabled, it broke the coloring of the statusbar, as well as causing other bugs, especially on newer iphone models without a home button. Additionally, it meant that reusing the sidedrawer in other screens was impossible without reimplementing large parts of it or moving things into separate but very tightly coupled components.

The solution to this was to use a Vue plugin called Portal-Vue. This provides the ability to render parts of the DOM elsewhere from where they were defined. In this case, it allows the `RadSideDrawer` to be at the top level of the app in `App.vue`, as seen in Listing 18, while the things in the sidedrawer could be placed in `Home.vue`, as shown in Listing 19. Since these controls are now all in the same place as where they are used, bindings can be made directly, which greatly simplified the code.

```
<template>
  <RadSideDrawer ref="drawer" gesturesEnabled="false">
    <StackLayout ~drawerContent class="sidedrawer-left">
      <Image class="sidedrawer-header" src="res://logo" stretch="aspectFit" />
      <portal-target name="sidedrawer" tag="ScrollView" />
    </StackLayout>

    <Frame ~mainContent>
      <LoginScreen />
    </Frame>
  </RadSideDrawer>
</template>

<script>
import Vue from 'vue';
import LoginScreen from './LoginScreen.vue';

export default {
  components: { LoginScreen },
  mounted() {
    Vue.prototype.$drawer = this.$refs.drawer.nativeView;
  },
};
</script>
```

Listing 18: RadSideDrawer Implementation [from `App.vue`]

3.5 Firebase and Crashlytics

Crashlytics is a software development kit for crash reporting, statistical analysis and application event logging. Insight into the status and performance of an app at a highly granular level equips ASSISTments personnel with the necessary information to address deficiencies in a timely and effective manner. Furthermore, they are able to better understand user interaction and tendencies through metrics such as active users, audience engagement and custom key performance indicators. This information is critical in gauging the effectiveness

```
<portal to="sidedrawer" tag="ScrollView">
  <StackLayout>
    <StackLayout class="sidedrawer-list-item hstack">
      <Label class="label" text="Offline Mode" />
      <Switch :isEnabled="networkConnected === offline" v-model="offline" />
    </StackLayout>

    <IconButton
      class="sidedrawer-list-item"
      @tap="logout"
      text="&#xf08b; Log Out"
    />
    <IconButton
      class="sidedrawer-list-item"
      @tap="report"
      text="&#xf188; Report a Bug"
    />
    <IconButton
      class="sidedrawer-list-item"
      @tap="help"
      text="&#xf059; Help"
    />
  </StackLayout>
</portal>
```

Listing 19: SideDrawer Contents [from `Home.vue`]

of business and technological decisions and delivers meaningful knowledge about the existing strategy. A communicative system with accurate status reporting is crucial in building maintainable software that engenders a knowledgeable surrounding team.

In the previous iOS app, Crashlytics was integrated and managed through Fabric, a platform that houses a suite of tools designed to assist mobile app development teams. The Fabric dashboard for the previous iOS app is visible in Figure 2. The old Android app was not configured with Crashlytics or Fabric and thus possessed no external statistics reporting mechanism. This inconsistency was addressed during the project.

Figure 2: Fabric Dashboard

Initially, Crashlytics would be configured through Fabric for both the Android and iOS. However, at the time of implementation Google announced the eventual deprecation of Fabric. Instead, Google encourages adopting *Firebase*, an updated toolset for application metric tracking. The Firebase Spark plan is free for general commercial use and provides a more complete analytics package than the antiquated solution, Fabric. Therefore, the `nativescript-firebase` plugin was used within the cross-platform app to set up Crashlytics and custom event logging for both Android and iOS. Firebase integration also includes default monitoring and statistical generation that is present in the Firebase Console dashboard in Figure 3

Prior to configuring the `nativescript-firebase` plugin, a Firebase account was created with a map the platform-specific AppIDs. Next, Firebase generated downloadable configuration files to be imported to the project structure. Upon installing the plugin, a script was executed that prompted user input for the setup of the initial environment parameters. Finally, the dependency was imported in app.js and initialized for activity shown in Listing 20. This provided access to the Crashlytics and Analytics APIs.

While methods exposed in the Crashlytics API can be invoked to force crashes, there

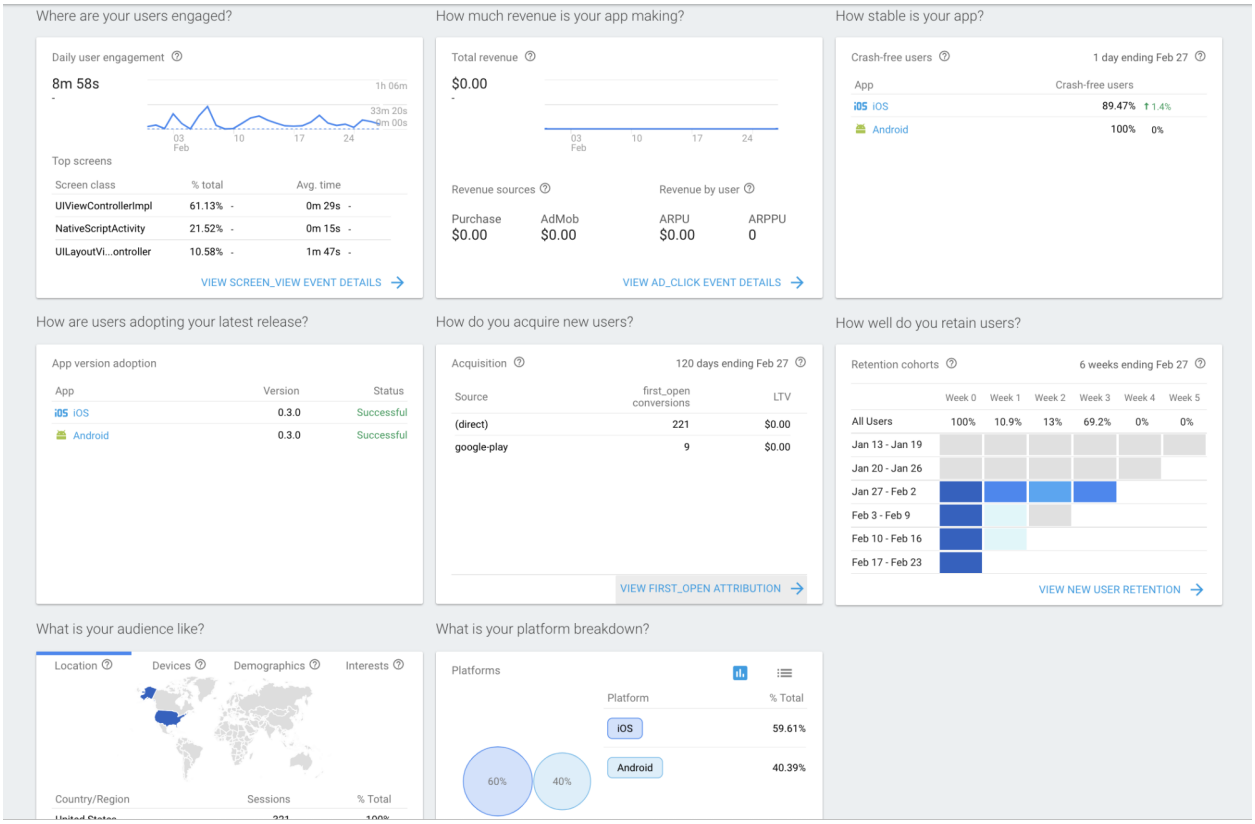


Figure 3: Firebase Dashboard

```
import * as firebase from 'nativescript-plugin-firebase';
firebase
  .init()
  .then(
    () => console.log('firebase.init done'),
    error => console.log('firebase.init error: ' + error));
```

Listing 20: Firebase Initialization

is no further installation needed to inform the system to begin monitoring both fatal and non-fatal app crashes. Firebase automatically collects crash information and reports data to the developer console. This communication is not real-time but instead updates every 3-4 hours. Figure 4 shows the Crashlytics dashboard within Firebase with several reported crashes. These crashes can be investigated further to uncover pertinent information such as the stack trace shown in the Figure 5 and device data in Figure 6.

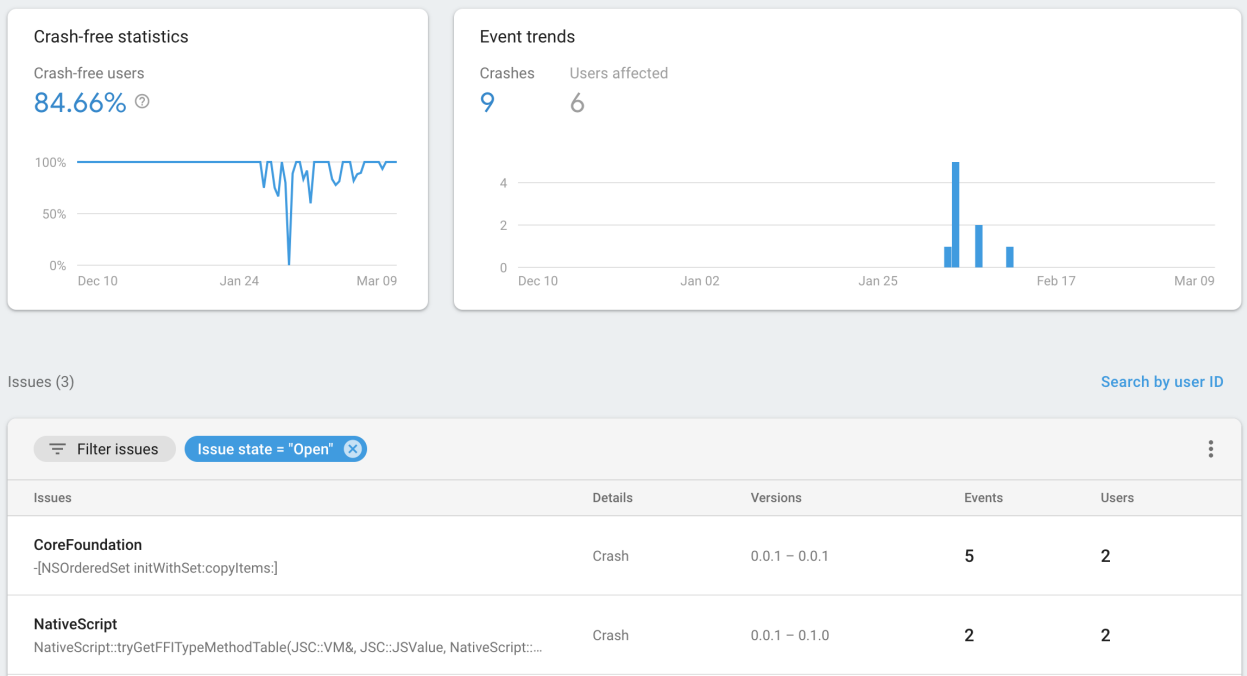


Figure 4: Firebase Crashlytics

In order to enhance user analytics, each screen within the app is recorded with a set screen name (ie. "Login Screen", "Report a Bug", etc.). Screen information is attached to several metrics involving activity, user engagement, and retention. Furthermore, Firebase automatically collects data about user events such as "screen_view," "session_start," and "app_open" where screen information is critical. An extensive list of events that are automatically tracked by the app is visible in Figure 7.

Coupling the generic user events monitored, Firebase supports customization of unique events that can be manually triggered throughout the app. The ASSISTments cross-platform app is configured to report when a user enters and launches an assignment within the Tutor. This is done examining the webview URL on page load. Every time a page loads the current URL is checked against the previous URL. If the current URL resembles an assignment in the tutor and the previous URL represents a page outside of the Tutor, then a *tutor_launch* event is emitted. A *tutor_exit* event is triggered in the opposite scenario. The corresponding logic is displayed in Listing 21 The event is logged with a call to `LogEvent()` passing the custom event key `tutor_launch` or `tutor_exit` depending on the state of the current and previous page.

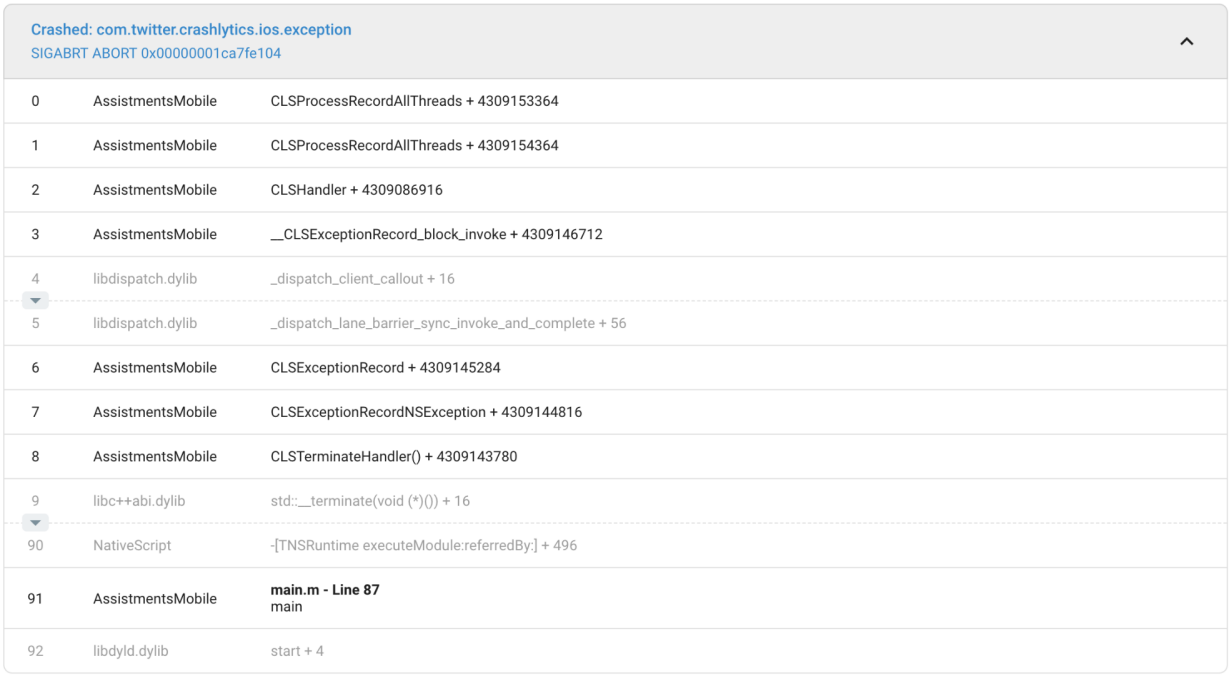


Figure 5: Stack Trace

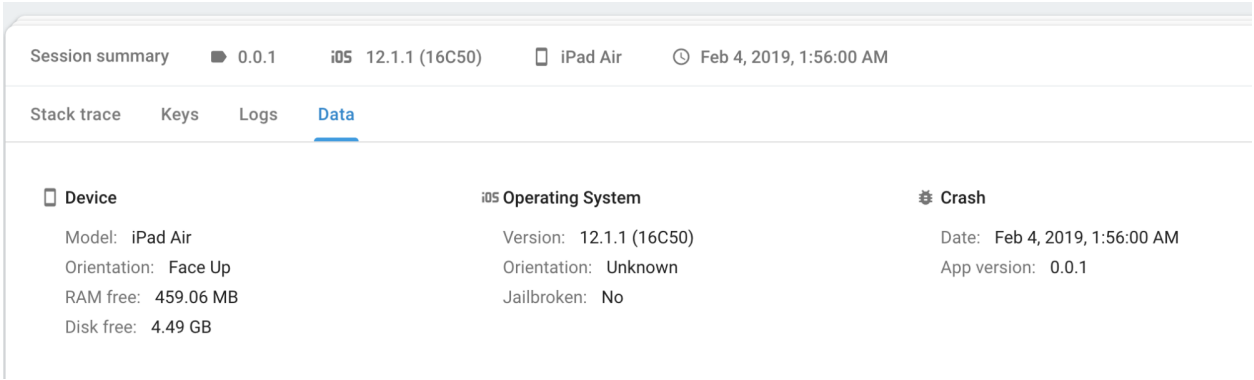


Figure 6: Firebase Device

```
// send event if we have navigated from a non-tutor page
// to the tutor or vice-versa
if (this.isTutor !== this.oldURL.startsWith(URLS.tutor)) {
  firebase.analytics.logEvent({
    key: this.isTutor ? 'tutor_launch' : 'tutor_exit',
  });
}

this.oldURL = this.webview.src;
```

Listing 21: Tutor Event [from `components/Home.vue`]

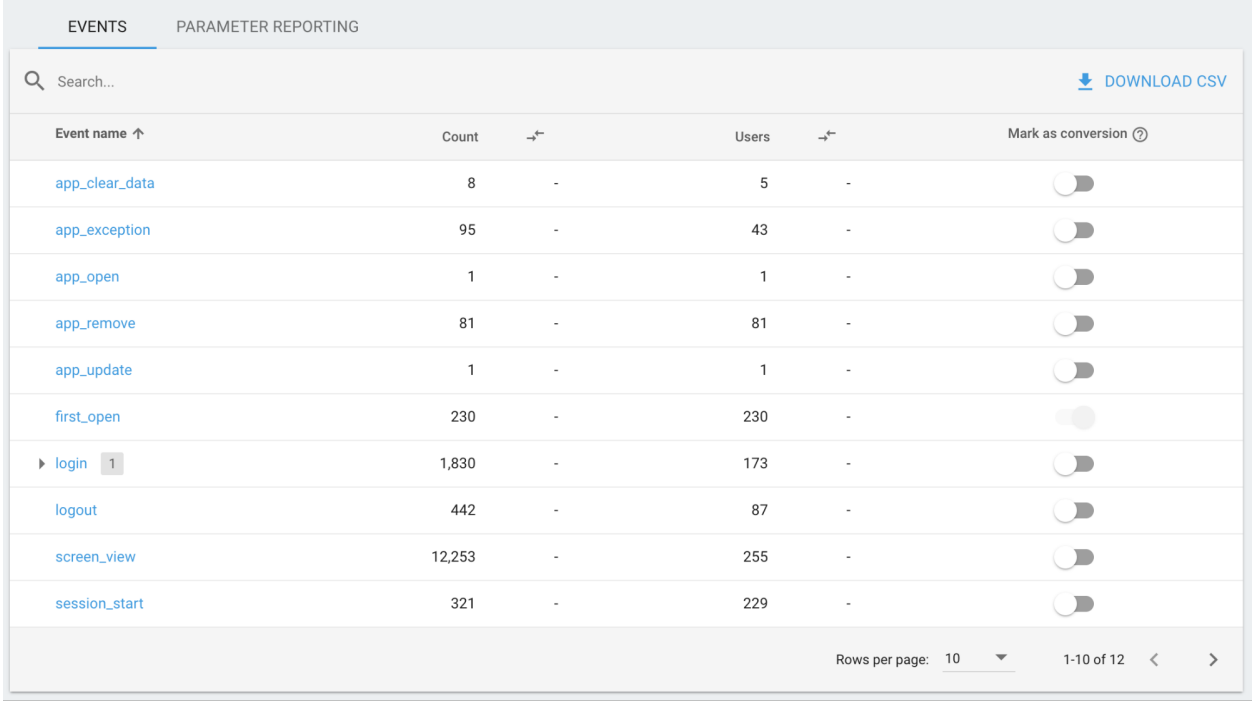


Figure 7: Firebase Events

4 Results and Discussion

4.1 Code size

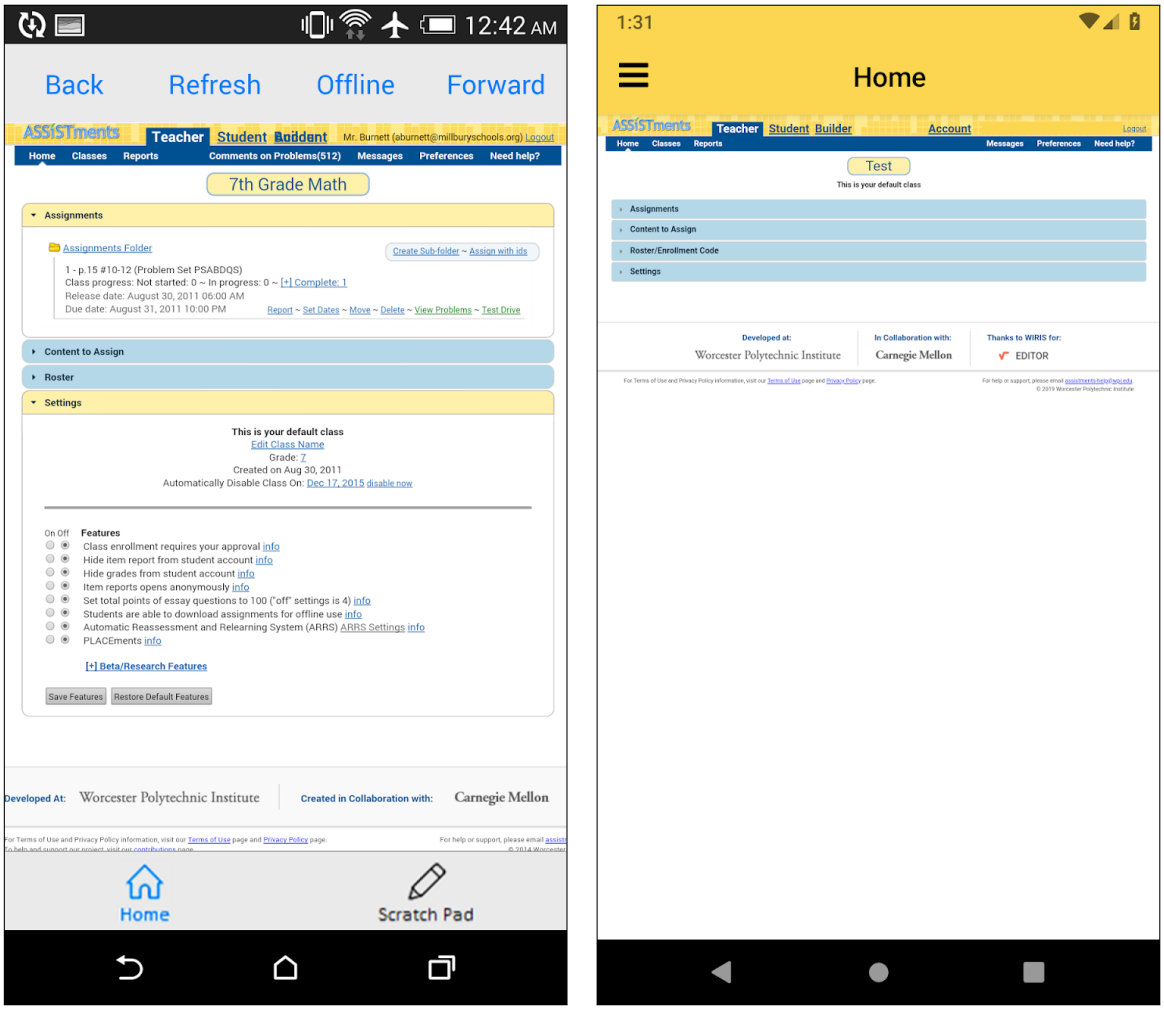
The unified app is uses significantly less code, with both old apps around 4K lines each compared to around 1K lines in the new unified app. This can be seen in Table 2, Table 3, Table 4, and Table 5. This means that it should be easier to read the code, as there is less of it and it is less spread out.

Language	files	blank	comment	code
Java	21	858	631	3836
XML	32	138	12	866
SUM:	53	996	643	4702

Table 2: LoC of Old Android App (`client/android/assistments/app/src`)

4.2 User Test Analysis

We conducted the first round of user testing with a tentative feature-complete app. At this stage, all of the desired high-level features were integrated into a single build. While we were aware of the potential shortcoming and instability, we opted to include unpolished features in order to receive feedback. The conditions of the test were structured in adherence to the user testing procedure. The environment included a single test subject who was a first-time user

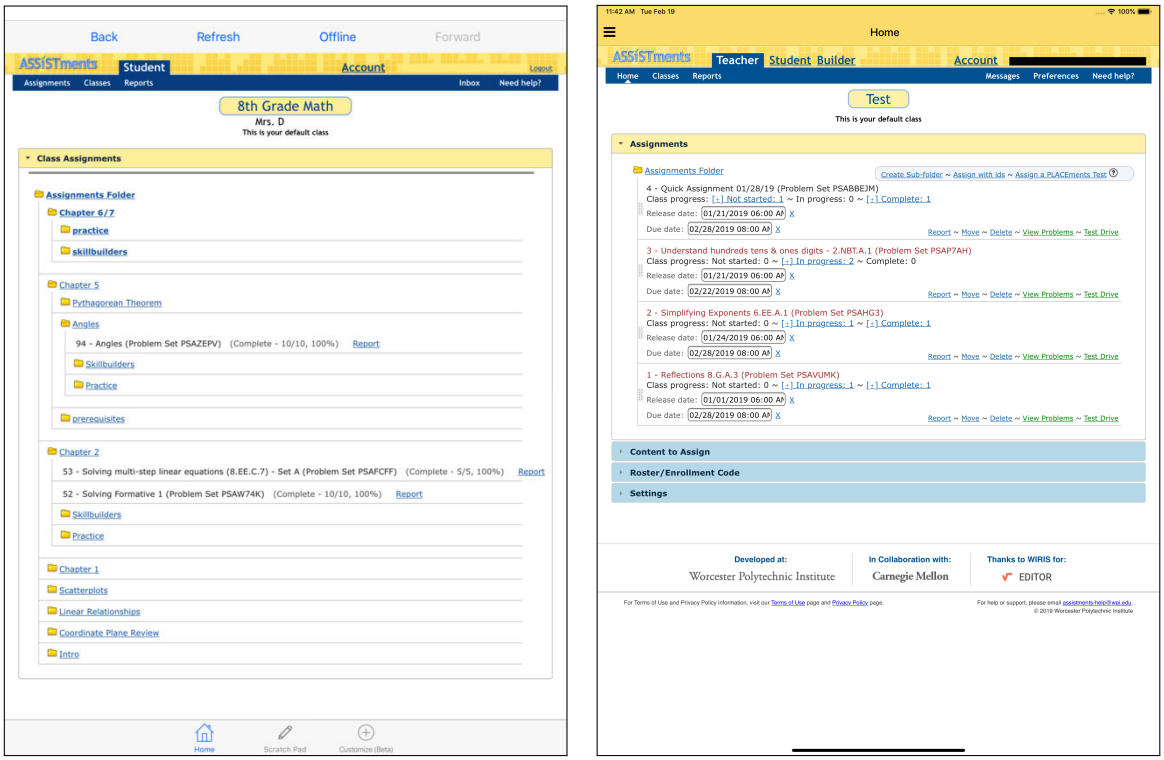


(a) Old App (b) New App

Figure 8: Comparison of Old and New Android Apps

Language	files	blank	comment	code
Objective C	21	1301	674	3109
Swift	30	274	317	1005
C/C++ Header	21	204	194	404
JSON	7	0	0	290
SUM:	79	1779	1185	4808

Table 3: LoC of Old iOS App (`client/iOS/ASSISTments/current/{Assistments,*.swift}`)



(a) Old App (b) New App

Figure 9: Comparison of Old and New iOS Apps

Language	files	blank	comment	code
Vuejs Component	10	136	27	1101
JSON	11	0	0	519
JavaScript	5	46	39	423
XML	8	14	6	94
Markdown	1	31	0	77
Sass	3	23	17	66
Gradle	2	18	8	59
SUM:	40	268	97	2339

Table 4: LoC of Unified App (all tracked files except `package-lock.json`)

Language	files	blank	comment	code
Vuejs Component	10	136	27	1101
JSON	8	0	0	435
JavaScript	4	27	11	175
XML	8	14	6	94
Sass	3	23	17	66
Gradle	2	18	8	59
SUM:	35	218	69	1930

Table 5: LoC of Unified App (just app folder)

and uninvolved in any aspect of the design or development of the product. Both a facilitator and observer were present and served as administrative figures. The subject tested the iOS version on an iPad and the Android version on a Google Pixel.

There were several issues uncovered through the perspective of a new user. Most notably, the existing Offline Mode workflow proved unintuitive and difficult to follow. The user located an Offline switch in the SideDrawer of the UI and immediately assumed to interact with this component when prompted to invoke Offline Mode. However, this feature is ineffective without the prior download of an assignment. The user struggled to determine how to download an assignment for offline access. Instead, the subject entered Offline Mode without any viewable assignments and continued to toggle between online and offline. The user closed the app and attempted to log in with immediate entrance into Offline Mode. Due to this inability, they suggested an option to enter Offline Mode from the login screen if the device is disconnected from a network. Completion of the Offline Mode process required significant interference and instruction from the administrators. This suggested that the workflow was not optimized for satisfactory user experience and proved too complex to navigate.

In consideration of the noted flaws with Offline Mode, we initiated a process to re-design the experience. We contended that the system should possess a higher level of intelligence and interact with the user through a series of prompts based on the app's state. Since Offline Mode is intended exclusively for use when the device lacks a network connection, we reasoned that monitoring and communicative device connectivity to the user is appropriate behavior. The updated Offline Mode experience includes UI confirm dialogs informing the user of a change in the device's connectivity state and prompting them to enter the mode that reflects the current state. Furthermore, the Offline Mode toggle in the SideDrawer is disabled if a a connectivity change has not been registered by the app. If the app is online and has not lost fallen off of the network, then the user will not be able to enter Offline Mode. It is only until the app enters a disconnected state that the user can accept to enter Offline Mode via the confirm dialog or toggling on the enable switch. Lastly, when the device is disconnected and the app is launched, the Login Screen will be altered to only allow for entrance into the app through Offline Mode. The standard login fields and button are disabled as login will fail to complete successfully in this state.

Another alarming concern with the current design is the seemingly unrelated nature of the native UI components of the app, such as the Action Bar and the webview. When the user was prompted to Logout they had immense difficulty identifying the correct UI element to invoke this action. The subject's immediate tendency was to search the containerized webview for a Logout element. The user neglected the hamburger icon in the top left corner of the Action Bar to open the SideDrawer where the Logout button existed. We considered possible re-designs to avoid this perception, yet several options involved flooding the UI with an excess of visible icons and buttons. In order to avoid an overwhelming visual experience, we concluded that the SideDrawer and its items should remain unchanged at the moment. If future tests corroborated this user's judgment and revealed similar concerns, then we would introduce changes to correct the apparently confusing SideDrawer layout. In order to combat this issue while maintaining a minimalistic UI layout, we replaced the title in the Action Bar of the "Home" Screen with the webview navigation icons from the SideDrawer. This is an effort to convey the association of the Action Bar and its native components with the

embedded ASSISTments webview.

Next, we observed the subject's frustration with multiple unresponsive icon and button taps. There appeared to be significant delays between action requests and fulfillment. Also, the active region surrounding the area of several icons was too small. Therefore, we increased the sensitivity of UI items with an associated tap event and incorporated an animation that signals a registered interaction.

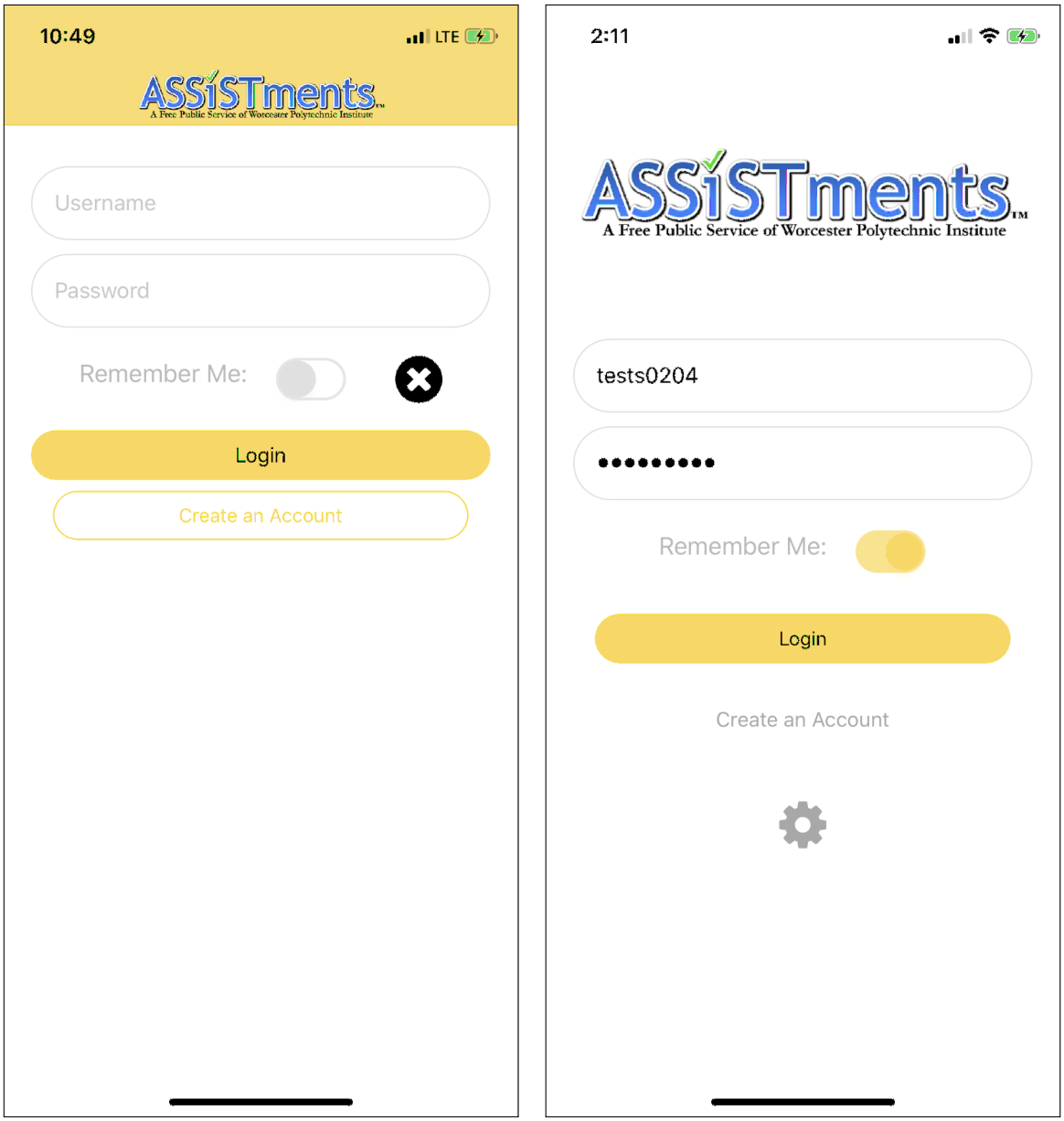
During the testing session, the subject was prompted the clear the username and password fields on the on the Login Screen. Despite the presence of a single icon to perform that task, the user failed to recognize its value and proceeded to manually erase the contents of each text field individually. Furthermore, the subject communicated the expectation of a small icon inside the text field position on the rightmost side to clear the respective text field contents. From this information, we redesigned the entire Login Screen UI seen in Figure 10 in order to add a icon that routes to a Settings Screen. Within the Settings page there is an option to clear the saved login credentials for the app. We figured that this feature should be farther removed from the Login Screen because it should only be invoked in the case of a shared device that should not always auto login with a single account. Since this will be a seldom-used feature its visibility is reduced in order to avoid confusion. Furthermore, the action button seen in Figure 11 on the Settings Screen features descriptive text and a confirmation dialog in order to explicitly communicate its purpose.

4.2.1 Oak Middle School

In another round of user testing, we visited an 8th grade class of existing ASSISTments users at Oak Middle School in Shrewsbury, Massachusetts. While we strived to adhere to user testing procedure, the environment stimulated disorganized evaluation. Rather than observing the usage of a single subject at a time, multiple user sessions had to be monitored simultaneously due to the volume of concurrent tests. Much of the feedback generated from this experience was received from in-class discussions and individual responses to the user survey. A combination of observations, active vocal feedback and general questions gave us actionable information from this interaction.

A common tendency among the students was to utilize Landscape Mode. Several subjects conveyed their displeasure with the minimized text in portrait orientation. They reacted to this visibility limitation by zooming in or continuing in landscape orientation. Due to the expected high frequency of Landscape Mode we decided to make the Action Bar dynamic and manipulate the device's status bar. With a quick swipe *up* gesture the Action Bar will hide giving rise to more available screen space for the webview to occupy. Furthermore, when the Action Bar is hidden the webview can extend into the status bar to further utilize available screen space. The Action Bar will return with a quick swipe *down* on the screen. During the test, an incidental trigger of this event was observed. A user encountered a state where the Action Bar disappeared unexpectedly and they were unaware of how to enforce its visibility. Therefore, we added a Toast when the Action Bar is hidden. The Toast will appear at the bottom of the screen for a short period of time with direction pertaining to unhiding the Action Bar.

Also, through usage observation, a bug in the Scratch Pad became apparent. A user uploaded their work as part of a problem and the image appeared in the TinyMCE text



(a) Old Login Screen (b) New Login Screen

Figure 10: Comparison of Login Screens

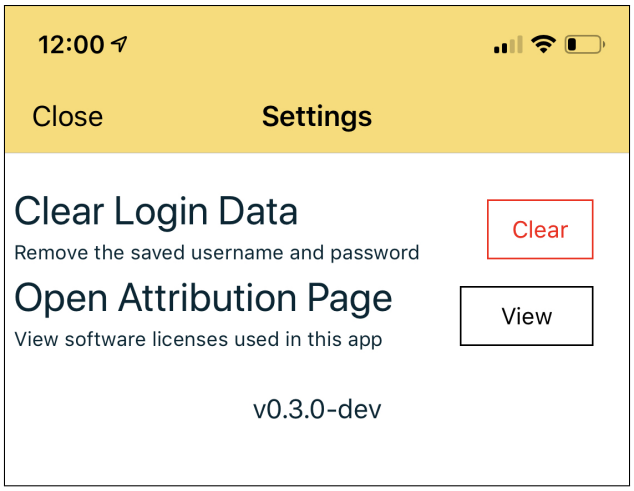
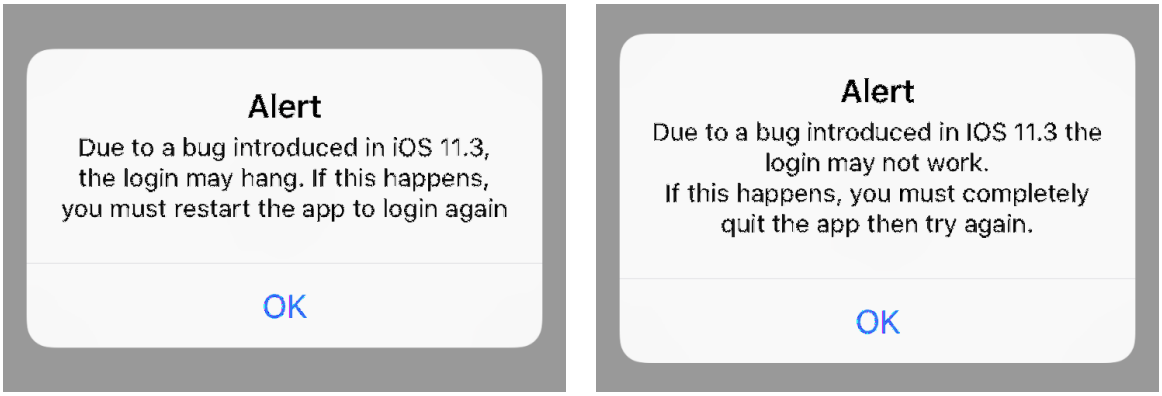


Figure 11: Settings

editor with a black background. We corrected the performance of this feature to only allow the upload of Scratch Pad work with a white canvas background. We fixed a bug that allowed the Scratch Pad to open when the icon was disabled on the UI. Lastly, we changed the wording on the alert dialog that pops up on Login during the same session as a Logout event. The user test revealed that the original phasing was too technical. This was evident through a student interpretation of the message as requiring because a student interpreted the directions as a necessity to uninstall the app. The wording was changed to better appeal to a young audience as shown in Figure 12.



(a) Old Login Screen (b) New Login Screen

Figure 12: Comparison of Login Screens

4.3 NativeScript

NativeScript served to be a powerful tool that was well-suited for the development criteria of the project. A primary goal of a cross-platform structure was to maximize code sharing across the Android and iOS platforms. The framework supported this endeavor with

natively implemented professional UI components. This allowed for template components such as `<ActionBar>` and `<RadSideDrawer>` to conform to the design practices of each mobile operating system. Therefore the app follows platform-specific design paradigms through a single codebase. Furthermore, NativeScript's engine provides flexible development options. The integration with Vue.js allowed for development with web technologies, which remains consistent with the the remainder of ASSISTments software. With JavaScript support on top of the Node.js runtime environment, we utilized the `npm` package manager to equip the app with relevant plugins. Numerous plugins were integrated to extend the functionality of the app and deliver a more performant system supported by the rapidly growing NativeScript community.

5 Conclusion

The conclusion of the development phase of the ASSISTments app resulted in build version 1.0.0 available on both the Play Store and App Store. There were several incremental builds (0.2.3 and 0.3.0) published to the respective distribution services throughout the duration of the project. These builds were utilized for internal testing and established the presence of the new app in the marketplace. This was helpful in navigating the review process mandated for distribution approval prior to the final release. Once the initial new app request was submitted, successive version pushes required significantly less overhead to become publicly listed.

Despite ultimately completing multiple rounds of publication, we failed to satisfy the intended goal of weekly app updates. We intended to operate under an agile framework and follow an iterative regimen with rigidly defined weekly priorities. However, we adopted a more flexible strategy due to frequent modification of expectations, functionality, and design. While high-level features remained constant throughout the project, incoming requests impacted the priority of tasks. The original development lifecycle was hindered by the inability to gain proper access and permission to the Apple Developer Program until week 4 of the project. This significantly delayed the app's entrance into the App Store in any viable capacity. Therefore, we were unable to properly assess the app through incremental user testing sessions. Since official user testing was only performed within the final two weeks of the project, some of the actionable knowledge from these experiences was not manifested in the final version of the app. Instead of relying on validated user concerns to guide incremental revisions, we utilized our best judgment to provide seemingly meaningful updates. Iterations of the app were governed largely by the overall feature list determined early in the term in combination with periodic feedback from informal internal testing.

Version 1.0.0 of the cross-platform ASSISTments mobile app includes implementation of all the features outlined in Section 1.2 in a functional capacity. The ASSISTments cross-platform app achieved feature parody with the previous mobile software. Additionally, the app corrected several existing bugs and removed defects that interrupted the user experience. Furthermore, several abnormal screen navigation routines were corrected. Overlaying the technical details of the system is a heightened sense of aesthetic appeal for the UI. Constructing a visually pleasing interface was a high priority. Effective design compliments the functionality of the system by communicating its purpose and invoking a sense of appre-

ciation. This translates to a more favorable application that users are likely to visit and frequently engage with.

The app in abundant with technological advances compared to the former ASSISTments mobile solution. The app now successfully remembers the user's account information and will perform an automatic login every time the app launches. Coupling this feature is an added option to clear the saved account data and reset the app's user recognition protocol. Numerous UI components are now located in a collapsable `RadSideDrawer`. This serves to declutter the interface and compartmentalize related items, which emphasizes the role of the embedded webview. Offline Mode, is optimized for the mobile domain through systemic recognition of network connectivity. The offline procedure is streamlined through more intelligent state management communication. This replaces the previous pressure to execute a complex sequence of decisions with an intuitive course of action. This is a major advancement in the latest version of the app because Offline Mode is better suited for the mobile context is than in the web platform. The support for image and scratch pad upload for show work is also preferred through the use of a mobile device. Also, subtle adjustments such as hiding the Action Bar, presenting a form for reporting a bug, allowing access to the user guide from within the app and analytics logging increase the power of ASSISTments. An exhaustive account of the completed work is marked as "Done" in Appendix E.

The ASSISTments cross-platform app was built to create value by producing a highly accessible learning assistant. The prevalence of digital consumption among the ASSISTments target demographic substantiates the need for a mobile presence. Through correcting poor aspects of the previous app and inserting positive additions, the mobile app now serves as a practical complement to the the web platform. The cross-platform nature of this app extends the scope of the project and significantly increases the reach of the technology. This effort coincides with the ASSISTments mission to be an instrumental tool in the educational community through the use of applied technology and research findings.

6 Future Work

This project laid the foundation for a cross-platform app, but there remains areas to improve. The app relies heavily on interacting with the site in a way that is not well defined, or likely to be stable. Some features require injection of JavaScript into the webview which interacts with DOM elements. These elements are also not well defined (with `class`es or `id`s), and so the injected JS relies on the DOM hierarchy to pick them. This is extremely fragile, as any change in how GWT creates this hierarchy will cause the selector to fail. In the future, it would be better to write this in a more defined, stable way so that components of the app are are entirely dependent on the web architecture.

6.1 Known Issues

Image and scratch pad submission doesn't work on free response, TinyMCE will break the viewport and render the in-app webview compressed.

Due to native handling of Action Bar UI, the icons assume an inconsistent layout across devices and screen sizes. While this does not impact performance or functionality the pre-

sentation it diminishes the presentation.

As explained in Section 3.2, you cannot login more than once per app launch on iOS, due to a bug in cookie syncing.

7 References

[1] Stoychev, E. (2017, December 22). NativeScript/NativeScript. Retrieved February 28, 2019, from https://github.com/NativeScript/NativeScript/wiki/Why-NativeScript?

[2] Torkut, D. (n.d.). Choosing Between Vue.js and ReactJS in 2019: What's Best for Your Project? Retrieved February 28, 2019, from https://www.codica.com/blog/react-vs-vue-2019/

[3] Xamarin vs React Native vs Ionic vs NativeScript: Cross-platform Mobile Frameworks Comparison. (2018, October 04). Retrieved February 28, 2019, from https://www.altexsoft.com/blog/engineering/xamarin-vs-react-native-vs-ionic-vs-nativescript-cross-platform-mobile-frameworks-comparison/

Appendices

A User Tests

A.1 User Interview 1 on 02/19

A.1.1 Round 1

OS Android

Version 8.1

Device Pixel 2

Account Used Test Student

1. Observational Notes:
    - Failed Login with Test Student account
        - `javax.net.ssl.SSLPeerUnverifiedException: hostname www.assistments.org not verified`
        - Turns out to be that he was on the guest wifi
2. Immediate Feedback
    - none

A.1.2 Round 2

OS iOS

Version

Device iPad Mini

Account Used Test Student

1. Observational Notes:
    - Screwed up autologin? Logged is asgoldsmith instead? How even?
        - Logged in with test → force close app → reopen autologged in as asgoldsmith → logout and logout screen displays test credentials... ???
    - Log out: tried account, couldn't find logout
        - Never found side drawer, thought it was like safari, where the bar is not part of the webapp
    - Tried again, and it logged in as right account after force close (first tried to log out in app, which obviously hung when he tried to log back in)

- Noticed scroll was buggy in submit work box
- Tried to use the built-in image submission for show image work, didn't notice the app one
- Progress sidedrawer a bit hard to find, might be just due to being unfamiliar with normal webapp
  - Tried to navigate to assignment listing when prompted to find progress
- Tried to login offline, then saw the offline button
  - Suggested offline button should be login offline (and corresponding login online)
  - Saw download button earlier
  - Offline broken again, whoops
  - Still can't logout offline
- Did see button for report bug earlier
  - Role selector not obvious (is this who I am or who I am reporting to?)
  - Did note that he can edit the bug email
- Thought the clear 'x' might be to prevent remember me, just cleared the text fields manually
- Scratchpad missing back button, can "submit" always, even when show work is not present
  - Thought it was weird that two things in top right are disconnected from rest of app
  - No feedback on adding an image
- "Home" title not representative of content of app
- Have to draw something to cancel scratchpad
- Never discovered the hide actionbar swipe
  - Tried to swipe on the actionbar itself when swipe
  - Felt like it should be much more responsive and require less of a flick gesture

(a) Pt 2 - with offline!

- Sidebar Switch seems intuitive ish
- Offline doesn't show correctness
- Tried to enter username password when going offffline from login screen
- Buttons sometimes not registering clicks, but show animations when pressed quickly
- Did not see submit button on assignments page
  - Brings you to the wrong place after submission anyway (back to offline screen), hard to get out
  - Should be highlighted or auto-pressed or something?

- Or prompt natively - Submit or Cancel
- Expected that once online it should auto submit

2. Immediate Feedback

   - Top bar too distinct, didn't think to look there for interacting with web app
   - Didn't know what the two top right buttons were

A.1.3 Round 3

OS Android

Version

Device LG G7

Account Used Test Student

Lots of whitespace

Rotate to landscape makes it much better, portrait requires some sort of zoom to make it work at all

Would be nice if text looped - responsive

Assignments =¿ report similar to report a bug, should be "view report" or something

Ipad was better

Computer still better than app

- Only reason to use is probably offline

A.2 Oak Middle School 2/25

A.2.1 Tasks

1. Login with personal account
2. Go to tutor and complete a few problems
   (a) Upload an image from device for submission
   (b) Upload a scratchpad image
   (c) Upload both on the same problem
3. Download the assignment and go offline then try to complete
4. Close the app and reopen
5. Logout then login to generate alert and observe response. Is the alert clear?

(a) Once back in → report a bug

Questions:

1. What did you think about offline mode?
2. Would you use the app to upload images?
3. If you use old app, would you use this one instead?
   (a) Would you use it instead of the website?

A.2.2 User Interview 1

OS iOS

Version

Device iPhone

Account Used

1. Observational Notes:
   - Did open the side drawer in tutor
   - Immediately found image for upload
     - Is it in?
2. Immediate Feedback
   - none

A.2.3 User Interview 2

OS iOS

Version

Device iPhone

Account Used own

1. Observational Notes:
   - Accidentally found the actionbar hiding, but didn't notice it?
   - Logged in fine
   - Started landscape, rotated to portrait when he couldn't see something
     - Zoom not working quite right, same as webapp
   - Crashed on submit work button

    - But worked after a restart
    - I biased a bit on image submission, because he did not know the normal behaviour
        * So he opened the box beforehand as well
        * Other student pointed it out
  - Knew where sidebar button was immediately upon prompting
  - Got an out of space error when downloading for offline
    - "97 pythagorean theorem finding leg..."
  - Did read error on re-login, but didn't restart until I asked
  - Tried to just uncheck the remember-me toggle when prompted to try to clear password
    - Didn't seem to notice the settings button

2. Immediate Feedback

   - none

A.2.4 Other students

- Scratchpad sketch has a black background when added to show-work
- Submit a bug page not submitting when no email set
  - School email can't send to non-domain emails
  - Probably needs to be a HTTP POST instead

B User Survey

Default Question Block

How did you test the app?

- ◯ In-person with an observer and instructions
- ◯ On my own time

What version of the ASSISTments mobile app did you test?

- ◯ iOS (Apple - iPhone or iPad)
- ◯ Android (Non-Apple devices)
- ◯ Both

Are you an existing ASSISTments user?

- ◯ Yes
- ◯ No, but I am familiar with ASSISTments
- ◯ No, I have never heard of ASSISTments

What is your role within ASSISTments?

- ☐ Student
- ☐ Teacher
- ☐ Admin
- ☐ Other

What is your affiliation with ASSISTments?

- ☐ I have used ASSISTments in an academic class as either a Student or Teacher
- ☐ I am a member of the ASSISTments Team at WPI
- ☐ Other:

Have you used the old version of the ASSISTments mobile app? If so, which platform?

- ☐ Yes, iOS
- ☐ Yes, Android
- ☐ No, I have never used the ASSISTments mobile app

Which platform does your personal mobile device use?

- ◯ iOS (Apple)
- ◯ Android (Samsung, LG, Google Pixel)
- ◯ I don't have a personal device
- ◯ Other:

What type of device are you most likely to use the ASSISTments mobile app on?

- ☐ Phone (iPhone, Samsung, LG, Google Pixel, etc.)
- ☐ Tablet (iPad, Galaxy Tablet, etc.)
- ☐ Google Chromebook
- ☐ Other:

What did you like MOST about the app?

What was the WORST part about the app?

Was there any part of the app that surprised you? How did it differ from your expectations?

Which task (or tasks) were EASIEST to complete?

Which task (or tasks) were most DIFFICULT to complete or did not function as expected?

How likely are you to...

	Extremely unlikely 1	2	3	4	5	6	Extremely likely 7
recommend the app to a friend							
use the app again							
use the app instead of the website							

How satisfied were you with the overall experience of the app?

	Extremely dissatisfied 0	1	2	4	5	6	Extremely satisfied 7
Satisfaction Level							

Are there any features that should be added or removed? Please explain.

What other feedback or suggestions do you have?

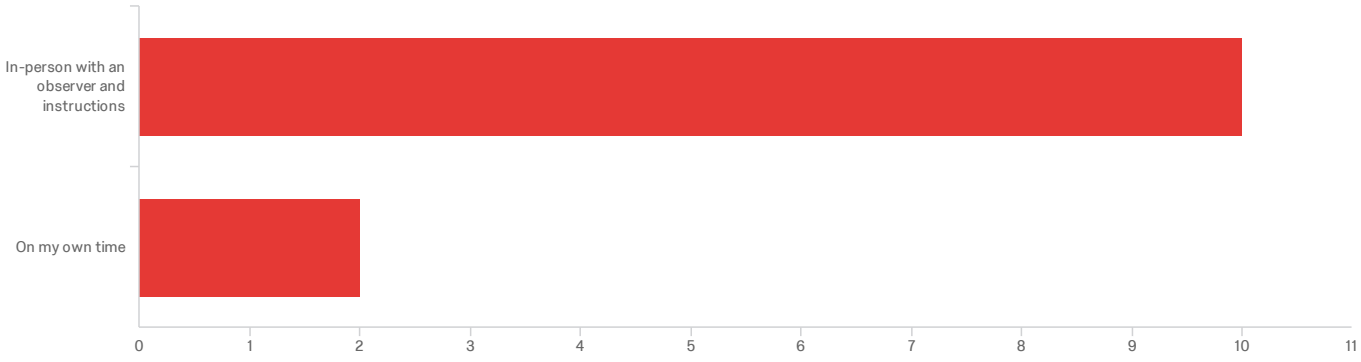
C User Survey Results

Default Report

ASSISTments Mobile App IQP

March 6, 2019 11:04 PM MST

Q.1 - How did you test the app?

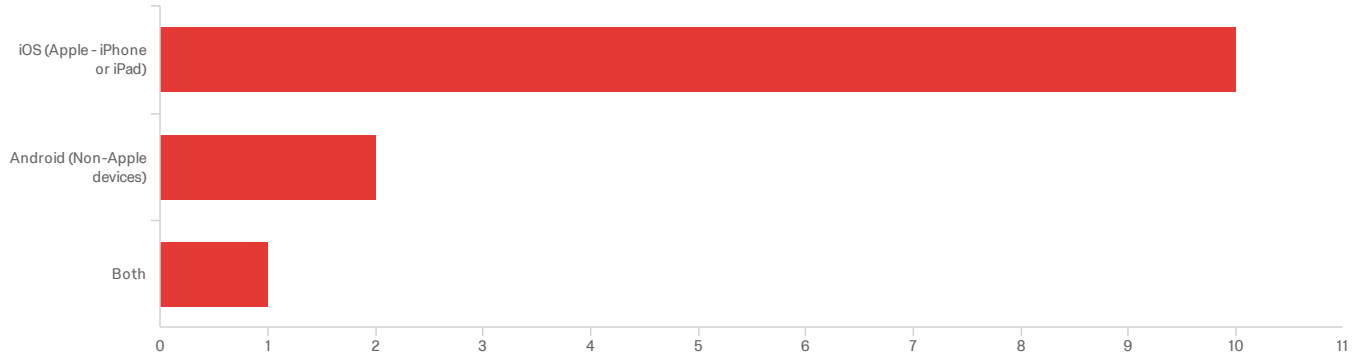


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	How did you test the app?	1.00	2.00	1.17	0.37	0.14	12

#	Field		Choice Count
1	In-person with an observer and instructions	83.33%	10
2	On my own time	16.67%	2
			12

Showing rows 1 - 3 of 3

Q1.1 - What version of the ASSISTments mobile app did you test?

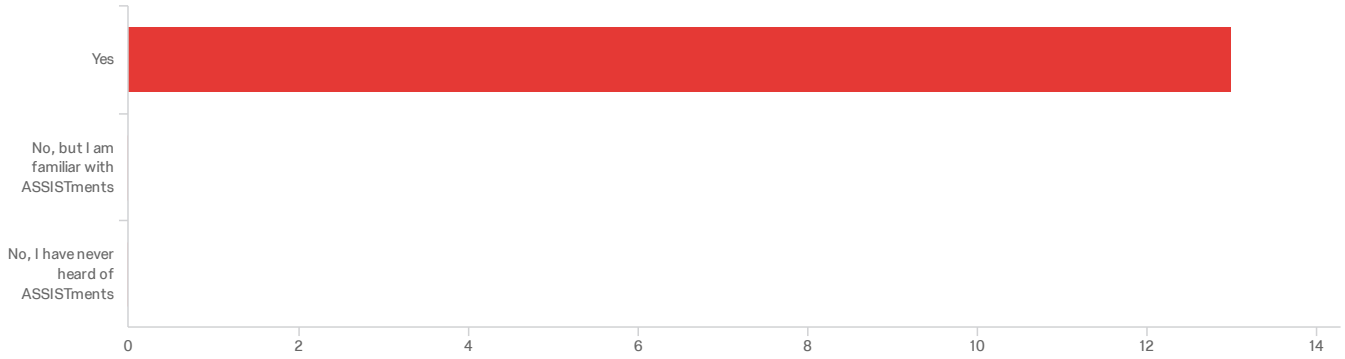


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	What version of the ASSISTments mobile app did you test?	1.00	3.00	1.31	0.61	0.37	13

#	Field		Choice Count
1	iOS (Apple - iPhone or iPad)	76.92%	10
2	Android (Non-Apple devices)	15.38%	2
3	Both	7.69%	1
			13

Showing rows 1 - 4 of 4

Q2 - Are you an existing ASSISTments user?

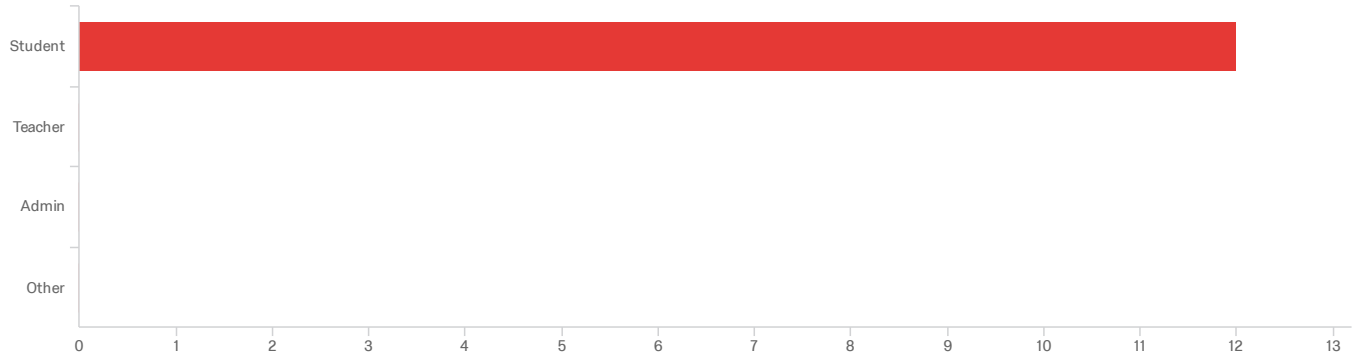


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Are you an existing ASSISTments user?	1.00	1.00	1.00	0.00	0.00	13

#	Field		Choice Count
1	Yes	100.00%	13
2	No, but I am familiar with ASSISTments	0.00%	0
3	No, I have never heard of ASSISTments	0.00%	0
			13

Showing rows 1 - 4 of 4

Q3 - What is your role within ASSISTments?



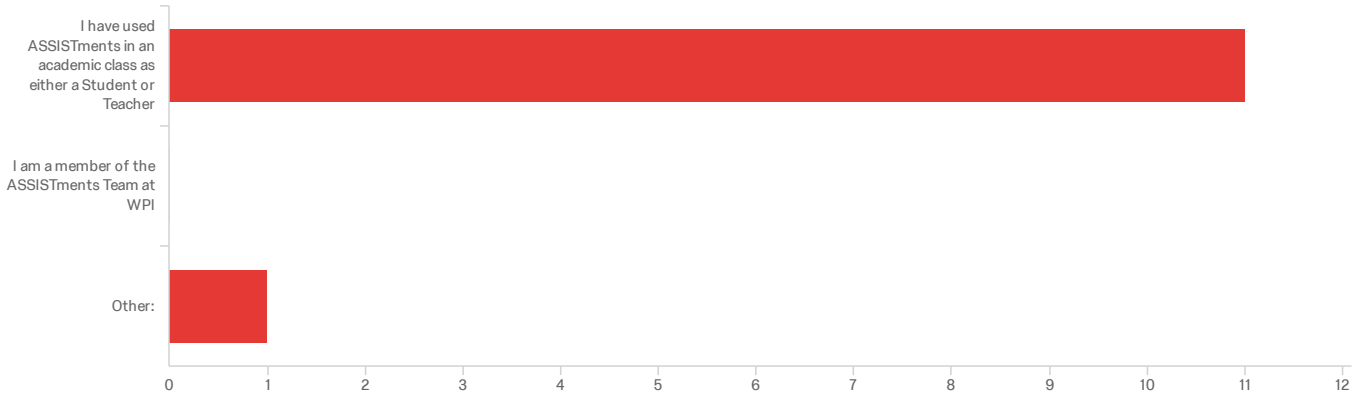
#	Field		Choice Count
1	Student	100.00%	12
2	Teacher	0.00%	0
3	Admin	0.00%	0
4	Other	0.00%	0
			12

Showing rows 1 - 5 of 5

Q3_4_TEXT - Other

Other

Q4 - What is your affiliation with ASSISTments?



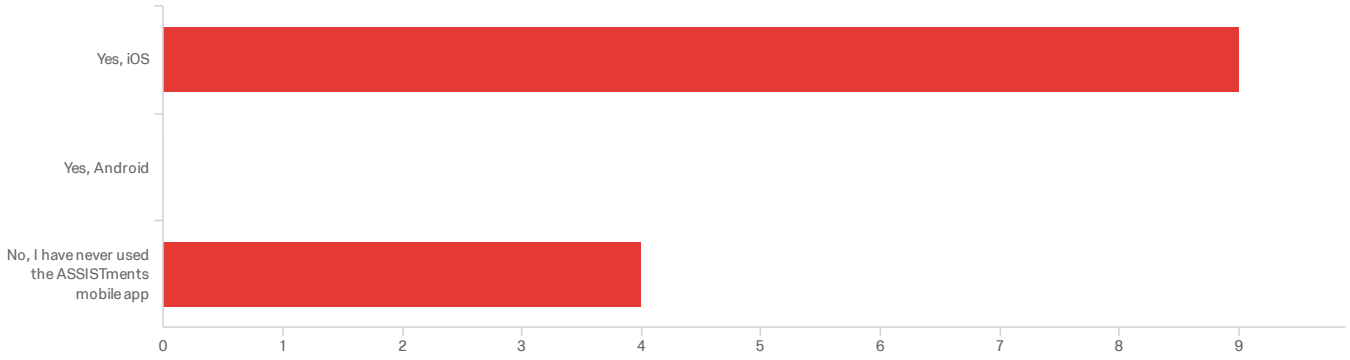
#	Field		Choice Count
1	I have used ASSISTments in an academic class as either a Student or Teacher	91.67%	11
2	I am a member of the ASSISTments Team at WPI	0.00%	0
3	Other:	8.33%	1
			12

Showing rows 1 - 4 of 4

Q4_3_TEXT - Other:

Other:
Son of Neil and Christina Heffernan

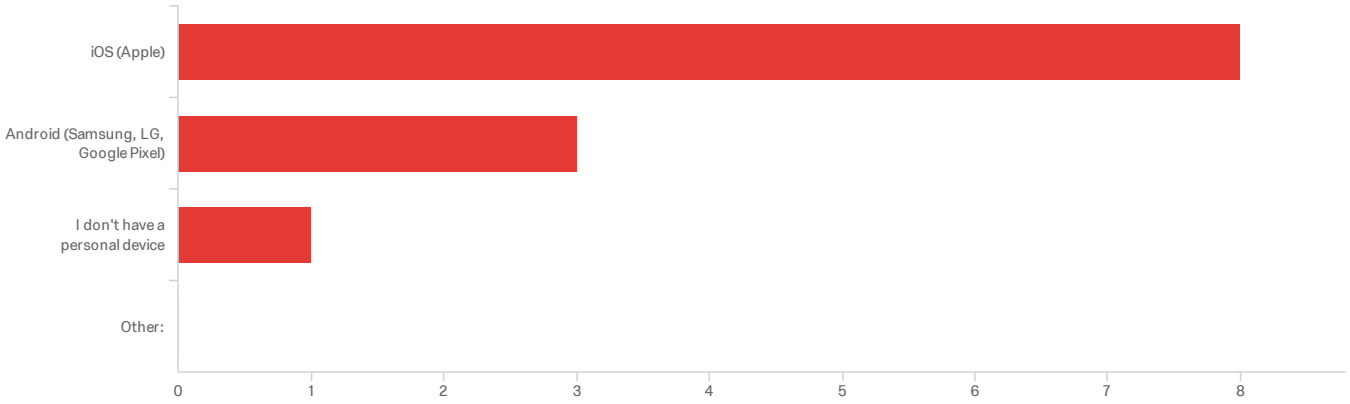
Q5 - Have you used the old version of the ASSISTments mobile app? If so, which platform?



#	Field		Choice Count
1	Yes, iOS	69.23%	9
2	Yes, Android	0.00%	0
3	No, I have never used the ASSISTments mobile app	30.77%	4
			13

Showing rows 1 - 4 of 4

Q6 - Which platform does your personal mobile device use?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Which platform does your personal mobile device use? - Selected Choice	1.00	3.00	1.42	0.64	0.41	12

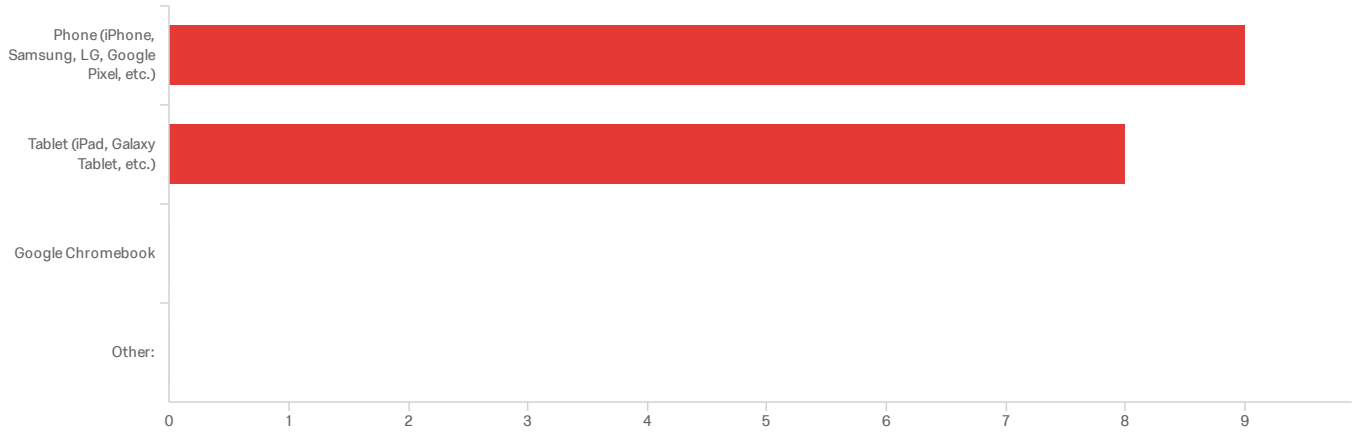
#	Field		Choice Count
1	iOS (Apple)	66.67%	8
2	Android (Samsung, LG, Google Pixel)	25.00%	3
3	I don't have a personal device	8.33%	1
4	Other:	0.00%	0
			12

Showing rows 1 - 5 of 5

Q6_4_TEXT - Other:

Other:

Q7 - What type of device are you most likely to use the ASSISTments mobile app on?



#	Field		Choice Count
1	Phone (iPhone, Samsung, LG, Google Pixel, etc.)	52.94%	9
2	Tablet (iPad, Galaxy Tablet, etc.)	47.06%	8
3	Google Chromebook	0.00%	0
4	Other:	0.00%	0
			17

Showing rows 1 - 5 of 5

Q7_4_TEXT - Other:

Other:

Q8 - What did you like MOST about the app?

What did you like MOST about the app?
It helps me remember older concepts with the reassessment tests.
Offline mode
It works fast
I liked how it was smooth to use
It’s easy and fast
I like to do math on it
The ability to do Assistments offline.
No WiFi setting
Offline
It was faster than the older one I was using.
Doing problems was very easy and worked without problem

Q9 - What was the WORST part about the app?

What was the WORST part about the app?
Some reassessment tests have way too many questions. When you get a question wrong multiple times, it doesn't tell you the answer
The math
Depending on the phone the questions are small
The font was pretty small. I was using the app on my phone, so the screen was a bit smaller. Therefore, the font was smaller
Having to sign in and reassessment tests
Sometimes it gets stuck zoomed in
It was confusing at first.
Glitchy
Small
Well since it's a phone obviously the screen is smaller.
The sketchbook was very buggy

Q10 - Was there any part of the app that surprised you? How did it differ from your expectations?

Was there any part of the app that surprised you? How did it differ from yo...
Just one time we had weird quiz questions that gave us a score at the end, but it never appeared again.
No.
No
I was surprised in a positive way at the major improvements the app had undergone. Before, the app was very slow. Now it is pretty smooth, it could be smoother, but I'm satisfied for now
It was like the websites with an app like outside with a sort of control panel. I thought it would be the website
Nope
The ability to do them offline. It exceeded them.
I didn't think you'd be able to do it without WiFi.
Nope
I thought it would still be slower but it was pretty fast.
The report button next to the assignments looks like a button to report the assigment for a bug or something. I would just change it to say "view report"

Q11 - Which task (or tasks) were EASIEST to complete?

Which task (or tasks) were EASIEST to complete?
The reassessment tests with only 1 or 2 questions are very easy to complete
Logging in
The work
It was easiest to do the multiple choice questions
The tests
Fractions
Checking the remember my password feature.
Just doing the Assistments
Zooming in
Getting back to all my assignments quicker.
Multiple choice was easiest

Q12 - Which task (or tasks) were most DIFFICULT to complete or did not function as expected?

Which task (or tasks) were most DIFFICULT to complete or did not function a...
The reassessment tests with over 10 questions are very hard because whenever you get a question wrong, you have to do at least 3 more questions for that subject because you get redemption' questions
Making it load
Nothing
There werent really any tasks which did not function as expected or were difficult to complete
Reassessment tests
The letter ones
Checking the downloadable assignments feature.
I didn't understand the download thing at first
Nothing
Zooming in to click on the right answer.
It was unintuitive to get to the logout button because the three lines looked like they were a separate thing outside of the site.

Q13 - How likely are you to...

#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	recommend the app to a friend	3.00	6.00	4.50	0.96	0.92	6
2	use the app again	5.00	7.00	5.50	0.76	0.58	6
3	use the app instead of the website	4.00	7.00	5.67	1.37	1.89	6

Q14 - How satisfied were you with the overall experience of the app?

#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Satisfaction Level	3.00	7.00	5.13	1.17	1.36	8

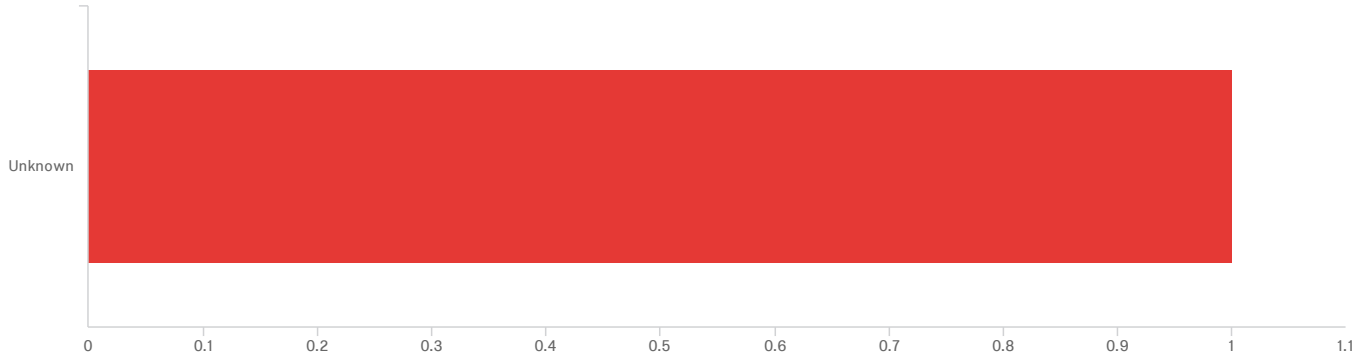
Q15 - Are there any features that should be added or removed? Please explain.

Are there any features that should be added or removed? Please explain.
When we get the problem wrong in a reassessment test, possibly provide an explanation on how to actually do the problem. I know there are the redemption tests for that, but I've spoken with others and we all agree that immediate instruction would be helpful
I think that the remediation or reassessment tests should be removed
How when there is reassessment test you sometimes have to do like 30 questions in one
No
Add in calculator
No
Make the finished assignments a different color than red or light red.
Make the site different on the app than the web version so it fills the screen more and is dynamic with the text and fills the screen.

Q16 - What other feedback or suggestions do you have?

What other feedback or suggestions do you have?
Probably make things a little less website looking (make it more compatible for a mobile app). Also, make the font bigger. I find myself having to
I don’t like how when you are doing a test if you get one wrong you have to restart
None
Make it so the screen can adjust to my phone because right now it’s very small and hard to read
None
Don’t give the same reassisments even when the student has mastered them.
The login and offline buttons when you log in would make more sense if they we're "login online"/"login" and "login offline" to me.

Q8 - Topics



#	Field		Choice Count
1	Unknown	100.00%	1

Showing rows 1 - 1 of 1

End of Report

D User Guide

General Usage of the Mobile App for iOS and Android

Table of Contents

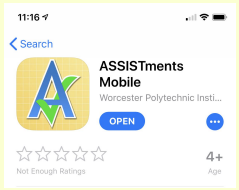
Use this slide to navigate to desired sections of the the user guide

Installation

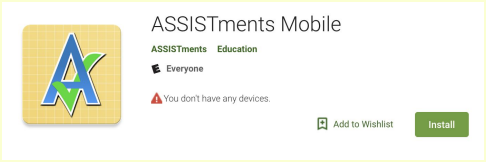
NOTE: These instructions primarily use iOS screenshots. The functionality and user interface on Android is identical. If there are variations between the versions, then screenshot from both platforms will be provide for demonstration

1. Install the app via the App Store (iOS) or Play Store (Android)

2. Ensure that you are downloading the version named “ASSISTments Mobile”

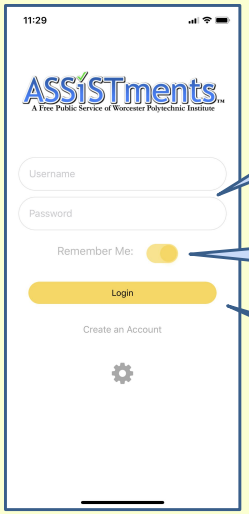


iOS



Android

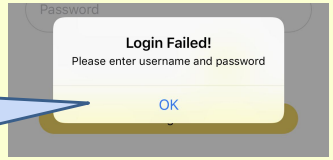
Login



1. Enter your username and password in the proper text fields

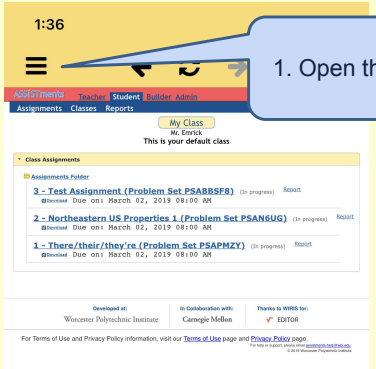
2. Turn the **Remember Me** toggle on (the display will be yellow) if you wish for the app to auto-login every time the app opens. This will save account credentials so that they do not need to be entered again

3. Click the **Login** button once you have completed entering your credentials

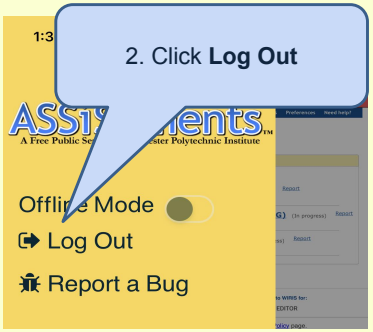


4. You will receive an alert if the username and/or password are incorrect. Tap **OK** and try again

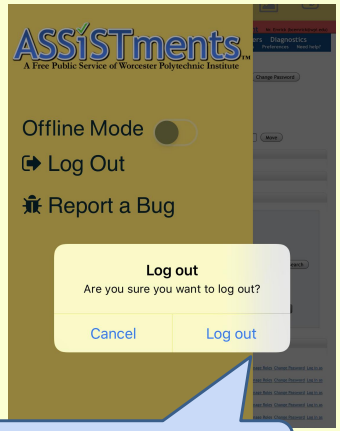
Logout



1. Open the **Side Drawer**



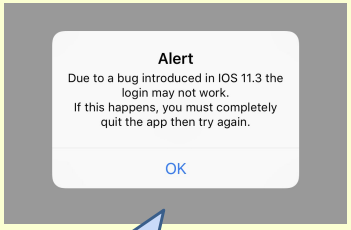
2. Click **Log Out**



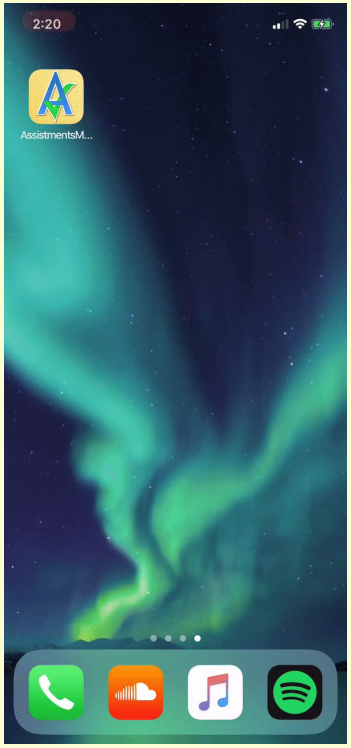
3. Confirm **Log Out** on dialog

More

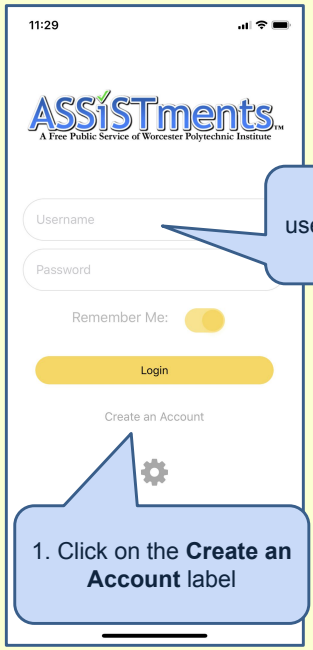
Logout iOS Bug



If you attempt to Login immediately after Log Out this alert will appear and you will not be able to enter the app. Click **OK** and follow the instructions. This video demonstrates how to quit the app on iOS

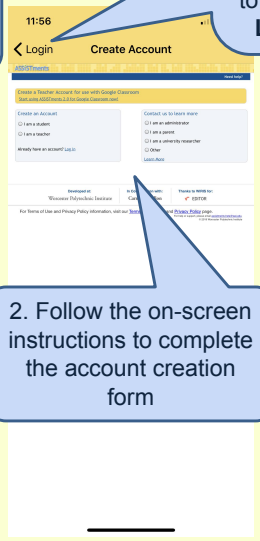


Create an Account



1. Click on the **Create an Account** label

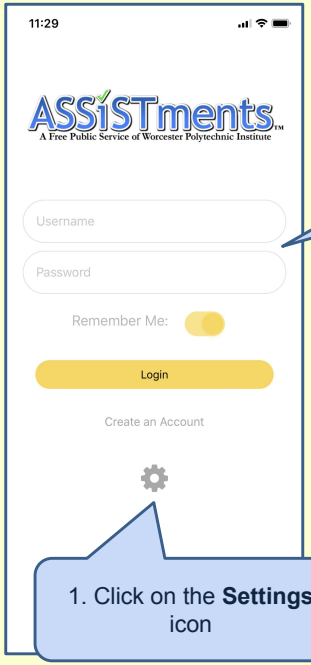
4.Enter your new username and password to login



2. Follow the on-screen instructions to complete the account creation form

3. Once you have finished the form and received your username and password. Click the **Back** button to navigate to the **Login Screen**

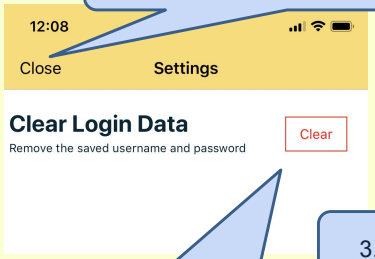
Clear Saved Login Data

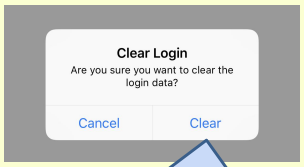


1. Click on the **Settings** icon

5. The username and password field will now be empty

4. Click **Close** to return to the Login Screen





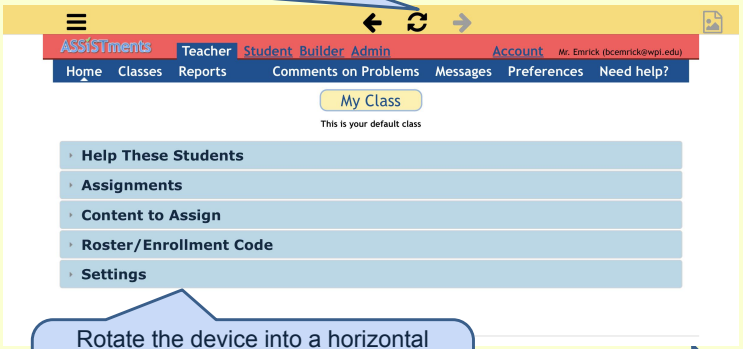
3. Click **Clear** on the dialog

2. Click the **Clear** button. This will erase the saved login credentials and will require input of a username and password to login again

Layout Overview

Quickly swipe **upward** on the screen to hide the Action Bar. *Quickly* swipe **downward** to display the hidden Action Bar

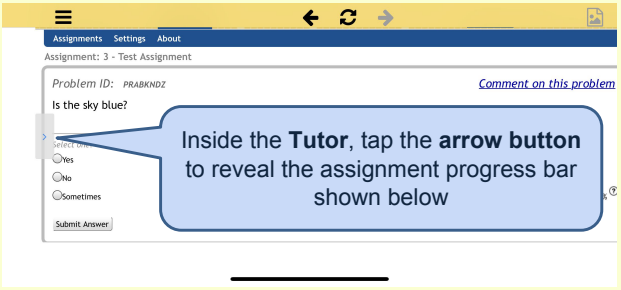
Use the **icons** in the center of the Action Bar to navigate through pages in the webview. The **left-facing arrow** will load the previous page. The **refresh icon** will reload the current page and the **right-facing arrow** will undo a backwards navigation



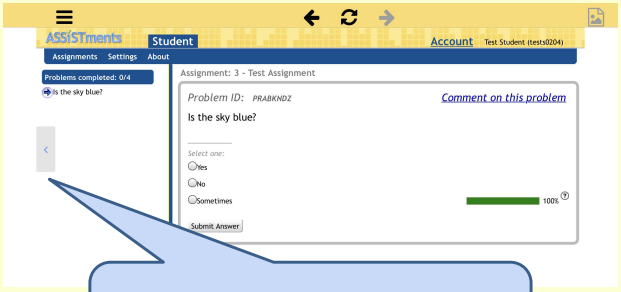
Rotate the device into a horizontal orientation to activate **Landscape Mode.** This will make text more legible, especially on smaller devices

More

Layout Overview



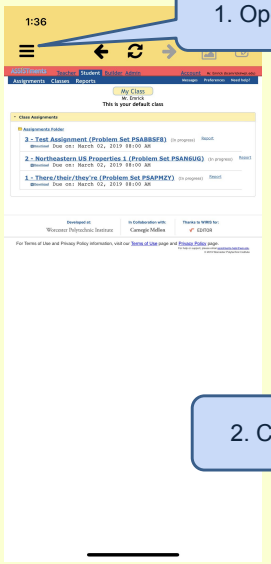
Inside the **Tutor**, tap the **arrow button** to reveal the assignment progress bar shown below



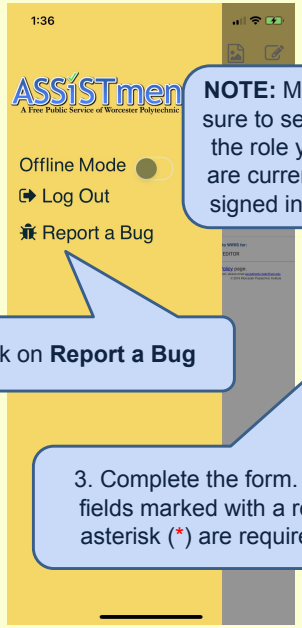
If open, tap again to **close** the assignment progress bar

TIP: Pinch and drag inside the webview to zoom IN and OUT!

Report a Bug

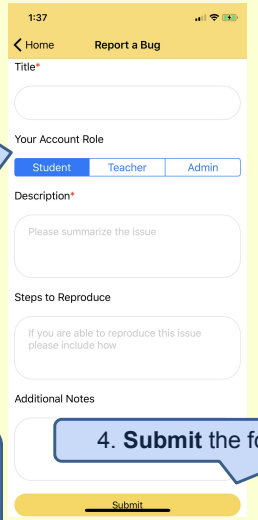


1. Open the **Side Drawer**



NOTE: Make sure to select the role you are currently signed in as

2. Click on **Report a Bug**



3. Complete the form. All fields marked with a red asterisk (*) are required

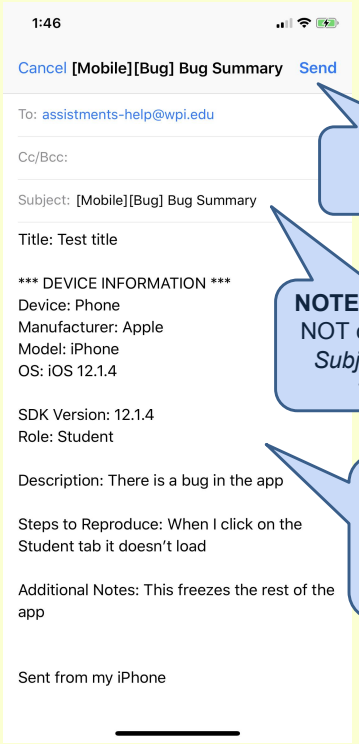
4. **Submit** the form

Report a Bug **iOS**

1. An email template will appear prefilled with the information submitted in the form

NOTE: On iOS the email can only be sent from within the Apple Mail app shown on right. If the Mail app is not configured you will not be able to submit the form on the **Report a Bug** screen. Also, if you cannot send an email to an address outside of the school district (to @wpi.edu) please contact someone who can

Mail App



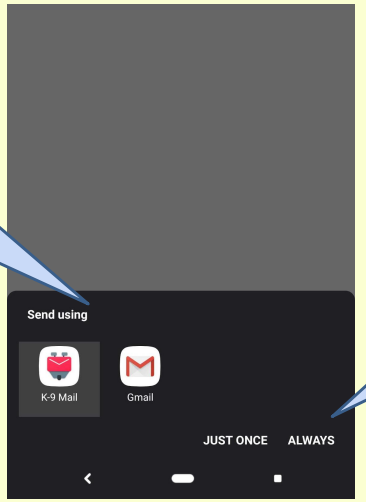
3. Click **Send**

NOTE: Please do NOT change the *Subject* or *To:* fields

2. **Edit** the input if necessary. Please do not delete important information

Report a Bug **Android**

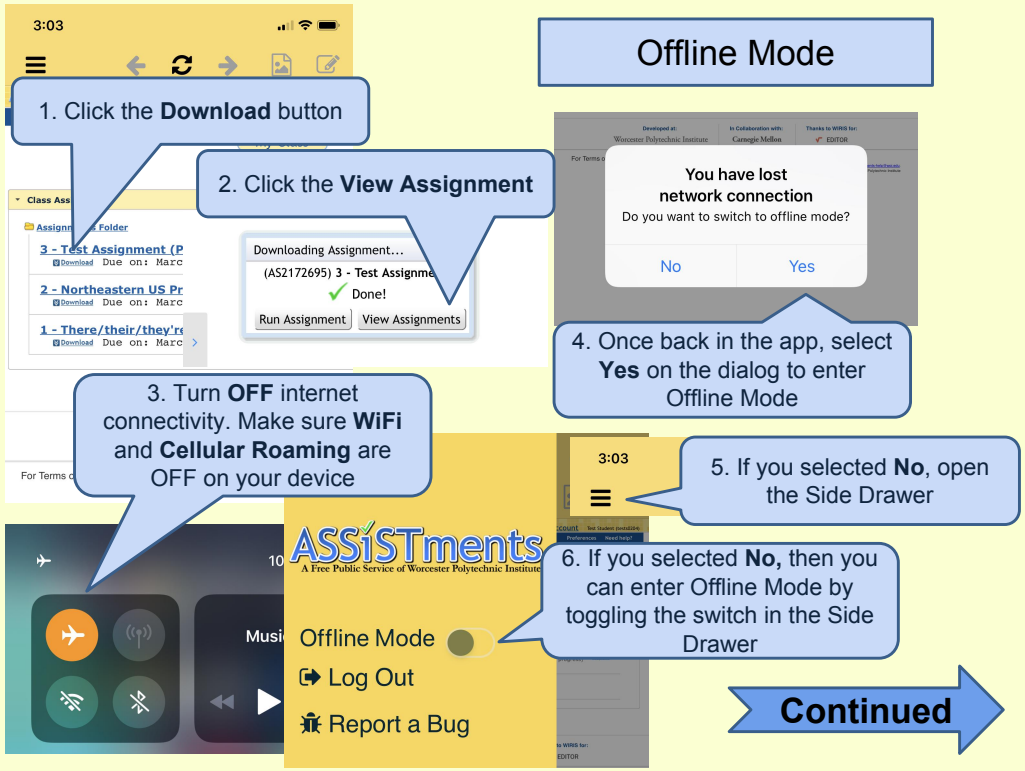
1. After **Submit** you may be prompted to select an email application



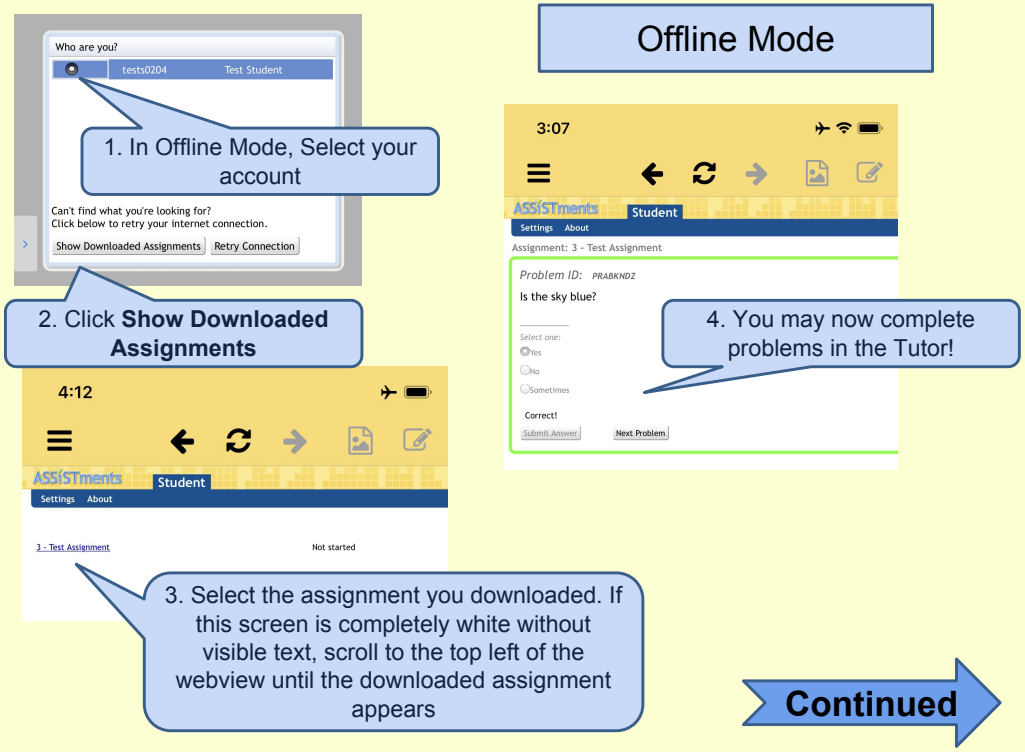
2. Select your preferred email application and choose **ALWAYS.** ASSISTments will send emails using the selected application on your device

3. Once the email draft is open follow the instructions on the previous slide

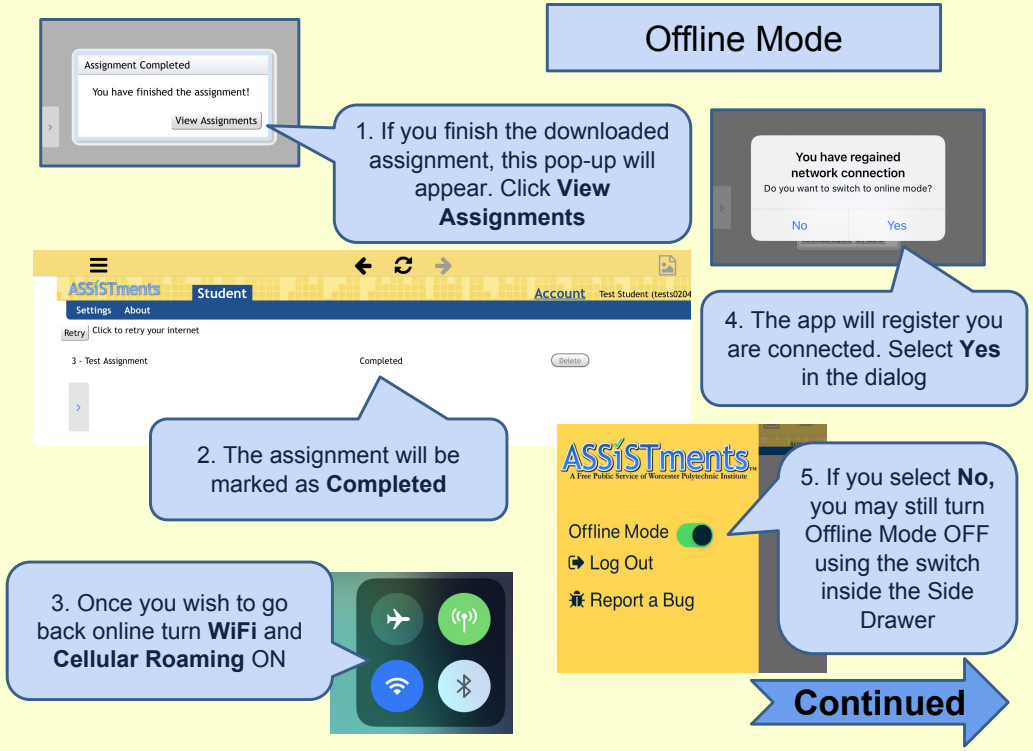
Offline Mode



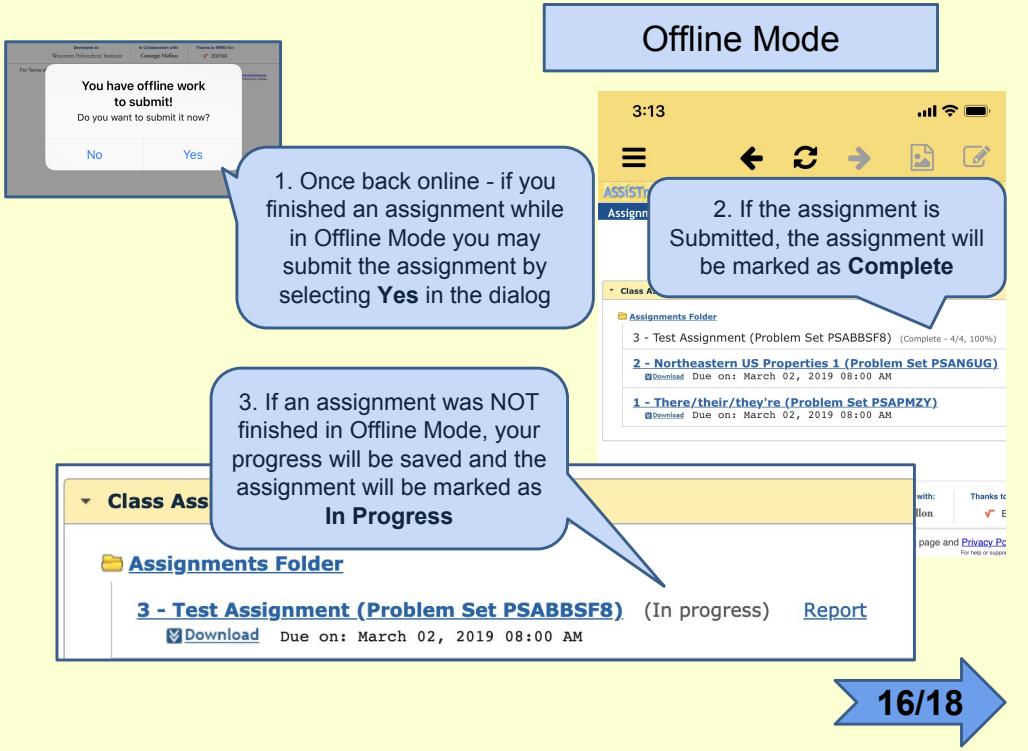
Offline Mode



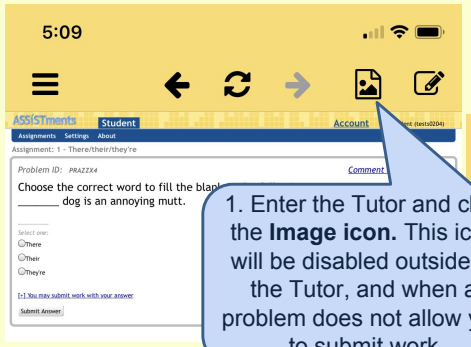
Offline Mode



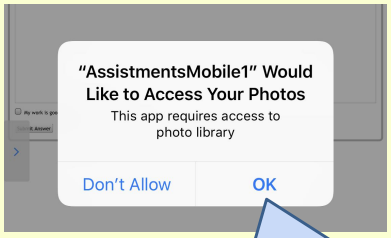
Offline Mode



Show Work - Image Submission

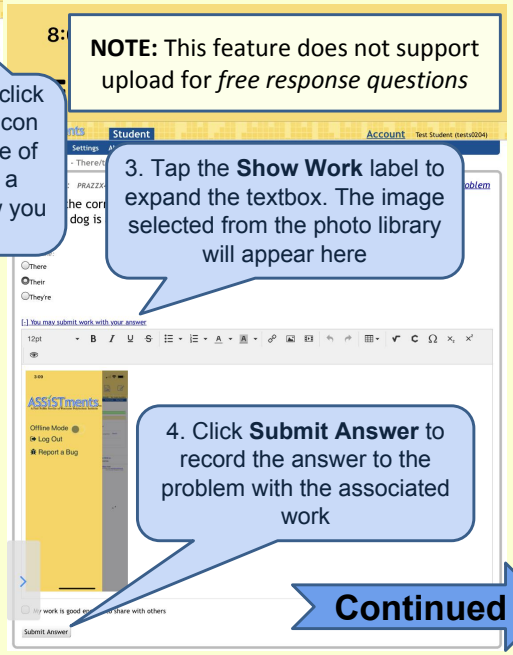


1. Enter the Tutor and click the **Image icon.** This icon will be disabled outside of the Tutor, and when a problem does not allow you to submit work



2. The first time this feature is launched ASSISTments will request photo library access. Select **OK**

NOTE: This feature does not support upload for *free response questions*

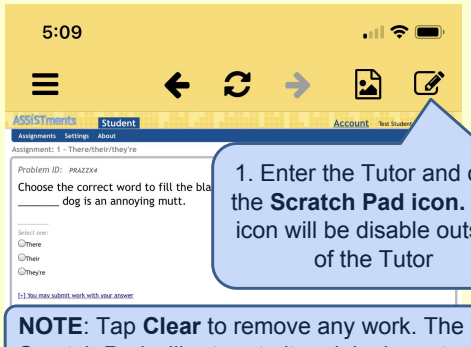


3. Tap the **Show Work** label to expand the textbox. The image selected from the photo library will appear here

4. Click **Submit Answer** to record the answer to the problem with the associated work

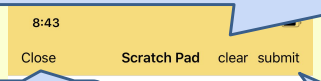
Continued

Show Work - Scratch Pad Submission



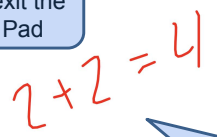
1. Enter the Tutor and click the **Scratch Pad icon.** This icon will be disable outside of the Tutor

NOTE: Tap **Clear** to remove any work. The Scratch Pad will return to its original empty state

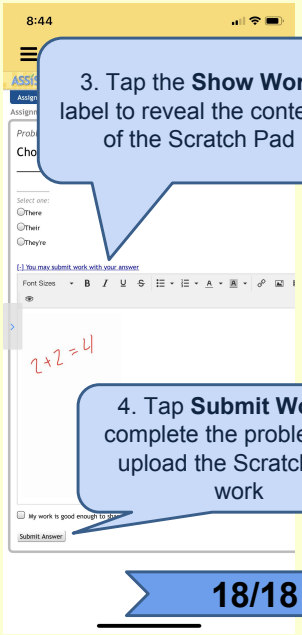


NOTE: Tap **Close** to exit the Scratch Pad

3. Tap **Submit** when finished



2. Draw in the Scratch Pad



3. Tap the **Show Work** label to reveal the contents of the Scratch Pad

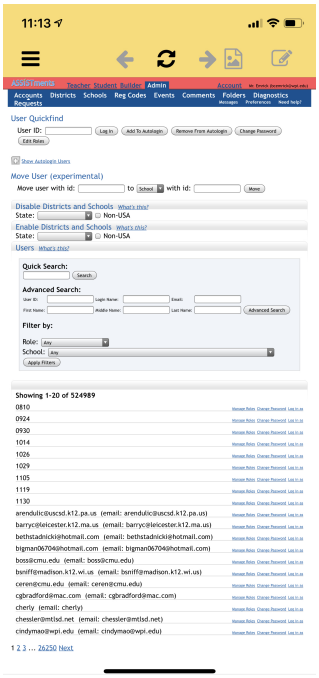
4. Tap **Submit Work** to complete the problem and upload the Scratch Pad work

18/18

E GitHub Project Task Management

E.1 Wontfix

- icons in the Action Bar are not aligned/spaced correctly [bug] [iOS]



  Action Bar is not optimized for iPhone X

- Scratch Pad work will not save after navigation away from the page [bug]
- Handle URL (open with Native App) [enhancement]

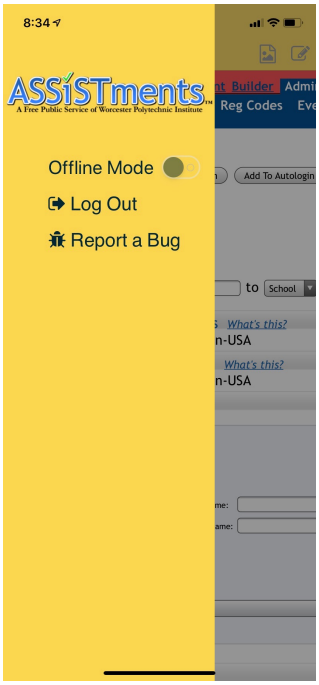
  From the Google classroom app, but might only matter for 2.0.

  References:

  - https://github.com/hypery2k/nativescript-urlhandler
  - https://developer.android.com/training/app-links/
  - https://developer.apple.com/library/archive/documentation/General/Conceptual/AppSearch/

  looks like it requires a file on the server for iOS support

- Side Drawer items have increased left-margin [bug] [iOS]



Not sure how to reproduce issue. I think it occurs on the first ever open of the app after switching portrait –¿ landscape –¿ portrait

- Support Assistments 2.0 [enhancement]

  Looks like this should mostly just be implementing #16.

- Detect offline capability of student and disable offline mode [enhancement]

  Either detect classes or check if there is downloaded content

- Enforce lack of network in offline mode [enhancement]

  ie force loading from cache only

- Fix aspect ratio of splash screen for iPhone X or later [bug] [iOS]

  Narrow whitespace at the bottom of the screen

- CreateAccount did not redirect to login on sucess [bug]

- Allow for image submission to free response questions [enhancement]

  Maybe optional?

- screen capture/video recording and uploading as show work [enhancement]

- Cannot navigate to any page from Admin [Android] [bug]

  Seems to be specific to Anthony's phone/account so far. Characterized by "batched" loading, where requests will appear to silently fail, then all happen at once.

  Unable to reproduce bug...

E.2 TODO

E.3 In Progress

E.4 Needs Testing

E.5 Done

- Remove old iOS app for sale and unpublish old Android from the Play Store [distribution]
- Update preview images in the Play Store [Android] [distribution]

  Should match Apple - or both should be changed if necessary
- Update existing Offline Mode Google Slide to link to new guide [distribution]
- Change name - currently AssistmentsMobile1 [distribution] [iOS]

  change to ASSISTments
- Link to help docs [enhancement]

  Probably in sidedrawer?
- Action Bar distorted and hides scratchpad button in Landscape orientation [bug] [iOS]

  Cannot see scratchpad icon. iPhone XS
- Generate PPTX for app help [distribution]

  probably just display some html docs that explain how to use any non-obvious features, or os-specific/other app based features like screen recording.
- Allow WebView to lay underneath status bar when Action Bar is hidden [enhancement]
- Open source licences page [enhancement]

  We should have a page in the app that describes the open-source libs we use, if their licenses require that.
- Change email for report a bug [distribution]

  assistments-help@wpi.edu
- Action Bar is hard to show after hiding it [enhancement]

  maybe a toast when it is hidden?
- Remember me switched ON as default [enhancement]
- Change wording on Login alert dialog [question]

  "Due to a bug introduced in IOS 11.3 the login may not work. If this happens, you must completely quit the app then try again."

- Scratchpad has black background when submitted [bug] [iOS]
- Scratchpad is not actually disabled [bug]
- Different app icons for Android and iOS [question]

  iOS with graph and Android no graph - should make both with graph
- Use modern Android icons [Android] [enhancement]

  the two part things, don't remember what they are called
- Can we make an entire PPTX for help [distribution] [question]

  Instead of just offline demo - a full app demo
- Change Role selection on report a bug page. Should auto select role based on logged in user [enhancement]

  Not autoselecting. Instead, add a more descriptive label
- Cannot login more than once per app launch [bug] [iOS]

  Something something cookie syncing Temp fix/workaround by displaying an alert dialog on hanging login - Can't force an app close...
- Styling/branding stuff [enhancement]
  - x Launch screen
  - x icons
  - x login screen banner
  - x colors
- Auto-submit offline work when connection re-established [enhancement]
- ScratchPad for work submission [enhancement]

  Maybe just call out to notability[1]
- Image Work Submission [enhancement]

  Students should be able to take/upload/draw images as their "show work"

  Implementation
  1. detect presence of "[-] You may submit work with your answer" on page
  2. provide button and picker to select image
     - or scratchpad (the mechanics for this are a bit different)
  3. upload image to server
  4. embed image into tinyMCE editor
  5. allow submission of problem as normal Maybe add button inside the page, instead of menu bar.

[1] <https://www.gingerlabs.com/>

- Offline Mode [enhancement]

  User can access certain features while offline

- Replace "Home" text in Action Bar with navigation icons for webview [enhancement]

- Redesign Login page to remove Action Bar and add a settings icon [enhancement]

- Create a Settings page that allows for clearing of login data [enhancement]

- Get a test account for automated testing [question]

- When in offline, a click on "logout" will never terminate/fail, making further requests fail too [bug]

  including an attempt to go back online

- CreateAccount does not redirect on logout, which you get when already logged in [bug]

- Register button on the Login Screen [enhancement]

  Ask about this Should add, but be called "Create an account"

- Add a "submit a bug/give feedback" button [enhancement]

  Should just prompt an email to somewhere. send an email to an address that doesn't exist yet, filled from a form that we check for completion

- Add button to clear saved username/pw [enhancement]

- Banner image too large on login screen on landscape devices [Android] [bug]

- Auto-hide the ActionBar on scroll to save space [enhancement]

  might be a bit tricky to get the scroll event from the webview. maybe we can get it from higher up?

- Handle JS alert, confirm, and prompt dialogs [enhancement]

  probably involves setting a WKUIDelegate and WebChromeClient.

- add Crashalytics or some other crash logging/notifications [enhancement]

- Add side Drawer [enhancement]

  almost certainly use RadSideDrawer[2]

  probably overkill implementation in Vue[3]

  Honestly not exactly sure what exactly we would put in it, maybe tabs (like the older app) would be better?

[2] <https://docs.telerik.com/devtools/nativescript-ui/Controls/NativeScript/SideDrawer/getting-started>

[3] <https://medium.com/@jamie_45704/nativescript-vue-creating-a-global-side-drawer-fd1c76683722>

- Collapse Problem Bar [enhancement]

  We should provide a button to collapse the list of problems sidebar, like the current iOS app.

- Add buttons for webview navigation (forward, back, refresh)
- Login Saving
- Basic Webview

F README

This is a cross-platform app client for ASSISTments[4].

G Building/Development

This app is written in NativeScript[5], specifically NativeScript Vue[6]. This allows it to share a common code base for both iOS and Android.

To get set up for building the app, follow the NativeScript setup instructions[7]. Then, run `npm install` in the top level directory of the project (which should be the same place this README can be found) to install the app's dependencies.

Then you can run the appropriate command to build/run/debug the app on a given platform, for example `tns run android --bundle` to run on android. Note that `--bundle` is quite important, as without it `tns` will not build the JS code.

You can make the app connect to Vue Devtools for debugging purposes with `--env.vueDevIP=130.215.`

G.1 Building for Distribution

Text in brackets (<>) should be replaced by the appropriate values.

G.1.1 Setting the version

The easiest way to change the version is to use NativeScript Sidekick, but if you want to do it manually, here's what you need to change (keeping the style of the existing values:

```
app/App_Resources/Android/AndroidManifest.xml:
    android:versionCode, android:versionName
app/App_Resources/iOS/Info.plist:
    CFBundleShortVersionString, CFBundleVersion
package.json: version
package-lock.json: version
```

[4] <https://www.assistments.org/>
[5] <https://www.nativescript.org/>
[6] <https://nativescript-vue.org/>
[7] <https://nativescript-vue.org/en/docs/getting-started/installation/>

To apply this, make sure you either remove the `platforms` directory, or re-run `tns prepare ios` (or `android`) to copy this new information into the build directory.

G.1.2 Build/Publish for iOS

You will need to acquire the Distribution Certificate and add it to your keychain. It will probably be a `.p12` file, which ideally someone else in ASSISTments has access to. Alternatively, anyone with an account that is an 'App Manager' in the WPI Developer group can revoke the old certificate, add a new one, and add it to the provision. You will also said provision, which you should just be able to download from Apple's certificates page[8].

```
tns build ios --release --for-device --bundle --env.production --env.uglify
 ↪  --provision <provisioning profile id or name>
tns publish ios <appleid> --ipa
 ↪  platforms/ios/build/Release-iphoneos/AssistmentsMobile.ipa
```

This should upload the package to App Store Connect. It will then process for a while, and you can go through the normal process to create a new version and submit it for review.

G.1.3 Build/Publish for Android

You will just need the keystore file and its passwords. Ideally someone else in ASSISTments has these, but alternatively they can probably be found in the previous Android app's files.

```
tns build android --aab --bundle --release --env.uglify --env.production --clean \
    --key-store-path <keystore file> --key-store-password <password> \
    --key-store-alias ASSISTments --key-store-alias-password <password>
```

Then upload the created `.aab` file to Google Play Console. You can use the `--copy-to <path>` argument to place the `.aab` in a more convenient location.

H Code structure

All of the code for the app is found in the `app` directory.

The entry point for the app is `app.js`. Most of the app is contained in Vue single file components (SFCs) in the `components` directory. There is also a `mixins` folder which contains code that is used across multiple SFCs.

There is a constants file at `constants.js` that currently just contains a mapping of URLs.

All of the code injected into the webview can be found at `local.js`.

Most of the common style in the app can be found in `_app-common.scss`, and the style's colors can be set in `_app_variables.scss`. Each SFC can also contain it's own style.

[8] <https://developer.apple.com/account/ios/certificate/>

I Authors

This app was originally written by Adam Goldsmith and Ben Emrick at WPI as an IQP project in C term of 2019.