App Technical Documentation

Mobile Development

This is a resource aimed at helping understand the codebase and making the addition of features easier.

Launch and Screens

Under the */mobile folder* there are two documents called index.ios.js and index.android.js. These are the two files that are run upon launching the app, for iOS and Android respectively. These two files are designed to be as simple as possible. They simply render the navigator with the initial route, which in this case is the login screen.

The code for the main screens can be found in the */mobile/Screens* folder. Each of these files extends the React Native Component class and thus implements the *render* method. To add a new screen, add another javascript file to this directory that extends the React Native Component class. Additionally, each of these files should export the Component class itself.

Navigation

Navigation is handled using React Native's built in Navigator class. To clearly see the current routes of Navigation reference the *Navigation.js* file under */mobile/helpers/*. This file contains the render function that the main navigator that the index files use to start up the app. To add a new screen to navigation a case needs to be added to this render function. To properly due this add an *if* case that renders the proper screen given a *routeId*. This *routeId* is provided when a new path is pushed onto the Navigator. Please reference this file for all the necessary *routeId*'s.

Navigating from one Component to Another

In the *Navigation.js* render function, we pass the navigator to each component that is rendered through the props of the class. Thus, in any of the component classes the Navigator can be accessed by `this.props.navigator`.

Minimally, to navigate from one component to another the following call has to be made, `this.props.navigator.push({id: 'PageName'});`, where PageName is the corresponding *routeId* from the render function. Of course this "push" can be replaced with an valid React Native navigation call per this documentation.

Currently, necessary information passing between screens is done through the navigator. When a new screen is pushed onto the Navigator additional information can be passed in the JSON of the navigator *push* or *replace*. This can then be accessed directly in the *render* function in the *Navigation.js* file and then passed to the given Component through its props.

Bottom Tab Bar

The tab bar on the bottom of the screen that switches between the Discover and StorkFront pages handles navigation slightly differently. This class can be referenced in the */mobile/Components/BottomTabBar.js* file. Rather than use navigation to switch between the Discover and StorkFront pages the BottomTabBar class renders the different screens manually (not through a Navigator call) based on what tab is selected. This is best seen in the *renderContent* function.

To render the Discover or StorkFront screen call the Navigator with the *routeId* mainView. This will cause the Navigator to show the Bottom Tab Bar, which by default will render the Discover page. To get the StorkFront screen to be shown on this navigation call pass 'storkFront' with the key 'route.screen' in the JSON of the Navigation call.

To add a new tab to the tab bar add the functionality in the `renderTabs()` function, which renders the tabs themselves, and in the `renderContent()` function, which renders the rest of the screen based on the state of the selected tab.

Animation

To animate the navigation the JSON key of 'sceneConfig' needs to be filled out with a value corresponding to a React Native scene configuration. An example where such animation is added are the `_register()` and `_toAboutPage()` functions from the *LoginScreen.js* file.

Push vs Pop vs Replace

The difference between pushing, popping, and replacing can best be described in the React Native documentation. However, it is important to note that there are several places when a push or replace is used when a pop seems more intuitive. The reason for this is to force a call to the ComponentDidMount function of the component being navigated to. Often this ComponentDidMount function includes code for refreshing the screen (grabbing new information from the server). Thus, there is no necessity for these pushes to be replaced with pops if more robust logic is added to the components to refresh with new information when desired.

Property Files, Styling, and Configuration

Properties File

The */mobile/resources/Properties.js* contains several properties. This includes information regarding the ASYNC storage keys and Google API Keys (used for geolocation).

Styling

All CSS styling is located in */mobile/Styles/Layouts.js*. This file is organized with headers that generally group styles by screen.

Config

All server URL configurations can be found in */mobile/Config/Server.js*.

Other resources

Other resources, such as images, can be found in the */mobile/resources/* folder. The current logo being used is the storkdLogo.png.

Test Suite

The test suit for the mobile application can be found under */mobile/**test**/*. The testing framework being used is Jest.

This test suit can be run with the command "npm test". Running the command "npm test -- -u" will overwrite the screenshots from the last test.

Keyboard Avoiding

There are several places in the code base where we shift the view so the user can see the input they are putting into a text field. For example, in the Login Screen we shift the screen up when the user enters their username and password. We are currently doing this using React Native's built in KeyboardAvoidingView.

Geolocation

Basic geolocation is achieved using React Native's built in geolocation API. Using this we are able to get the current location as a coordinate pair. An example of geolocation can be found in the `_getLocation(callback)` method in *NewPost.js*.

Geocoding

To translate geolocation data into location information such as locality, address, or zipcode, we use a third party geocoder library. The coding is done app-side when a new post is uploaded, in *NewPost.js*. The library will attempt to use a native geocoder service if one is available. If there is none, it will default to using the Google Maps API. This requires a Google Maps API key: `Geocoder.fallbackToGoogle(GOOGLE_API_KEY);`. The `GOOGLE_API_KEY` constant can be found in */mobile/resouces/Properties.js*. A key can be acquired here: https://developers.google.com/maps/documentation/geocoding/get-api-key.

Dependencies

We used a variety of 3rd party libraries in our app. All of these dependencies are listed in the "package.json" file under the mobile folder. These dependencies can be added with the Node Package Manager (npm) by running the command “npm install”

DEV dependencies

The dev dependencies listed in the packages.json file are Jest and Babel, both of which are needed to run the test suit.

Note: All currently added dependencies are MIT licensed.

Performance Impacts

With additional app features and a larger user base it might be necessary to make changes to improve the performance of the app. Below we have included some easy fixes that we believe will boost performance.

Caching

We are currently caching the Discover posts in the Discover screen in asynchronous storage. Currently, the Discover screen is caching up to 20 posts when it makes a fetch call to the server. This can be seen in the fetchData method in the limit parameter of the fetch URL. This can be increased and decreased as desired.

Additionally, we are caching parts of the “user” object upon login, such as username and profile picture. We do such to avoid make a network call anytime we desire information about the user.

We believe any place where caching can be done to limit server calls will help boost the performance of the app. However, the tradeoffs of using more asynchronous phone storage needs to be considered.

Pushing in Navigation

There are several instances in the app where we push screens on to the Navigator in order to use the “pop” functionality and to take advantage of animations (which is not the case when using “replace”). We believe there may be instances where pushing may cause the navigation stack to get out of hand, so given time, this issue should be addressed. To learn more about how Navigation is handled and how to best address this problem please reference the navigation section of the Dev.md file in this folder.

Additionally, one suggestion that could improve app performance on the navigation front is to limit gestures (such as swiping back). These gestures are by default built into React Native navigation, but can be overridden.

Native modules

For the sake of quick development we used third party modules that allowed us to build functionality without writing native modules. Such an example of where we use a wrapper for functionality is the Gifted Chat- a module used to help handle in chat messaging. To improve performance more native modules can be written (Swift modules for iOS and Java for android). Native modules can be written for:

1. In app messaging
2. Camera roll access
3. Facebook integration

Bugs

This is a document describing some of the bugs we know exist and that arose during user testing, but that we did not have the time to fix.

- Access to camera roll - One iOS user reported that he was not able to upload a photo from his camera roll, but was able to take a picture from his phone and use it for the app. We suspect this is a permission problem and that he possibly pressed “Do not allow” when prompted to allow Storkd to access his camera roll. As a bug fix the user should be asked more than one time to grant permission access, to prevent a case where they accidentally denied the permission request. This would prevent them from digging through the iOS settings to grant permission.
- Keyboard - We are currently using React Native’s current default keyboard. Unfortunately for multiline text fields, such as in the New Post screen, this keyboard does not have a “Done” button, which prevents the user from exiting the keyboard.
- Swiping - In several points in the app there are places when the user can swipe left and right to navigate through the app in a less than desirable fashion. This is linked to our use of React Native’s navigator. Some of the transitions in the React Native navigator have built in gestures, the push has a built in “swipe right to go back” gesture for example. To prevent undesirable swiping these gestures need to be limited. One example of where this is done is the _toMainView() method in the LoginScreen.js file. Please reference this instance and apply such a fix to other transition cases.
- Exiting New Post Screen - When you press the Story text field in the New Post screen there is no intuitive way to exit the Keyboard (please reference the “Keyboard” bug above). Additionally, the screen is shifted upward to avoid the Keyboard. Thus, there is no way to access the “back” button. This creates a bug in which the user can not cancel the making of a New Post once they press the Story text field (unless they do unintuitive navigation where they click other text fields to exit the keyboard then press back).
- Modal Picker - In the New Post screen there is an option to select “Type.” This is a Modal Picker (3rd party extension of Component(overtop a text field with animation). Clicking the middle of the type textfield will open the Modal as desired. However, if the user clicks to the left of the text box, where there is an icon, they will be able to input any text they desire.
- Modal Picker - On certain Android devices, the modal picker is unable to be selected when used with a disabled text-input.
- Photo scaling - Make changes such that the photo sizing is more consistent. Sometimes the photos are stretched, other times scaled. Right now square photos look best.
- Discover Post Info Screen - When the user taps the asking price on the Discover Post “more information screen” the keyboard appears.
- New Post - If there is an error message stating that there is no geocoder available for the platform, check to see that there is a valid Google API key.