Visualization: Mobile Application

Report

To start with, this Mobile Application named as **LarvaeLy** is simply a concept to see the fish larva (data points) as a marker on to the map layout. For this purpose, an elementary application is build on react native platform for only Android till this point. As a result the normal user can simply see all the fishes on the map with customized markers with its detailed information, instead of reading from **csv** file definitely increases the overhead for all users and kills the purpose of visualization.

In this app, there are few scenes with respect to its own specification.

- Home Scene
- Map Scene
- Map Clustered Scene
- Fish details Scene
- App Settings Scene

Home Scene:

In home scene, its simply a landing page or the start up of the application, where user will first see this scene after boot up screen. There are few information provided displayed to the anonymous user, which is just copied from the GitHub repository. Including this, Images, and a redirection button to navigate to the Map scene.

Map Scene:

In Map scene, it is basically the core scene or the heart of the application where a user can see all the fish data points collected, directly on to the Map with a custom Marker. Also, on clicking the data point we have a small fish card showing the details of the fish larva, which is to bring up specific fish larva card on selection, and displaying some information like: Observation ID, Trajectory ID, and Timestamp.

Map Clustered Scene:

It is again a Map view but with clustered data points. Here, the Markers of the closer data points will end up into a numbered clustered, showing the count of data points being pushed together into one single cluster. On zoom in the cluster with explode and evolving the map in to normal points. On zoom out, it will again join the data points into the numbered cluster.

Fish Details Scene:

So basically this scene is the higher level of information when clicking the info card of the fish larva, and navigating to specific fish card leads to details scene. This will show up number of information: Observation ID, Trajectory ID, MPA, Z - Value, **Land** with icon as positive or true for the presence of the land and icon as negative or false when the land value is 0 and finally, the Timestamp value.

App Settings Scene:

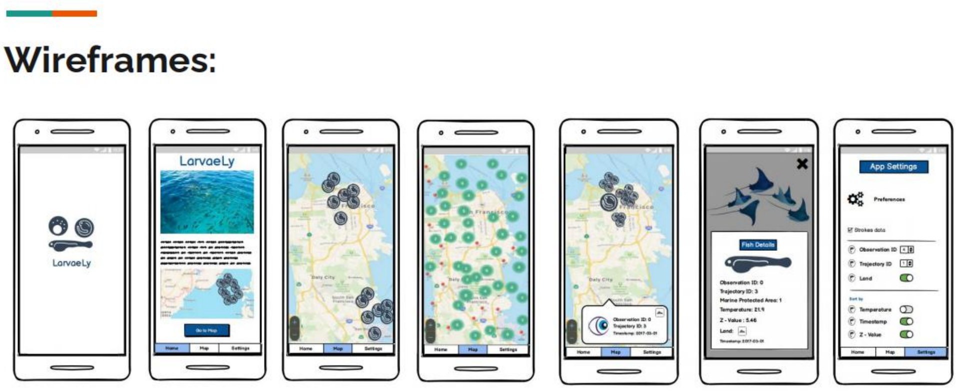
This scene is the most important scene where actual data toggling will be performed, here the user can change the filters according to its preferences. Moreover, options for filters are pretty straight-forward.

- **Enable/Disable Stokes data**: This will bring up only the stokes or no stokes data.
- **Search by specific Observation ID**: This will bring all the points with specific observation ID.
- **Search by specific Trajectory ID**: This will bring all the points with specific trajectory ID.
- **Switch buttons:** Enable/Disable land
- **Sort functions:** Temperature, Timestamp and Z - val.

In a nutshell, this small application is just to look over the data which is being fetched from the back-end (**database**) and visualizing them on the mobile application.

Project Insight:

First up is the design which was built on **Balsamiq Wireframes.**



As this is already available in the presentations and **.bmpr** file.

Code structure & Scaffolding:

This app is built on react native “open-source UI software framework created by Facebook”.

Firstly, setting up all the dependencies to make this application run on android platform.
https://reactnative.dev/docs/environment-setup

After this, setting up the emulator (the Android device) using AVD manager.

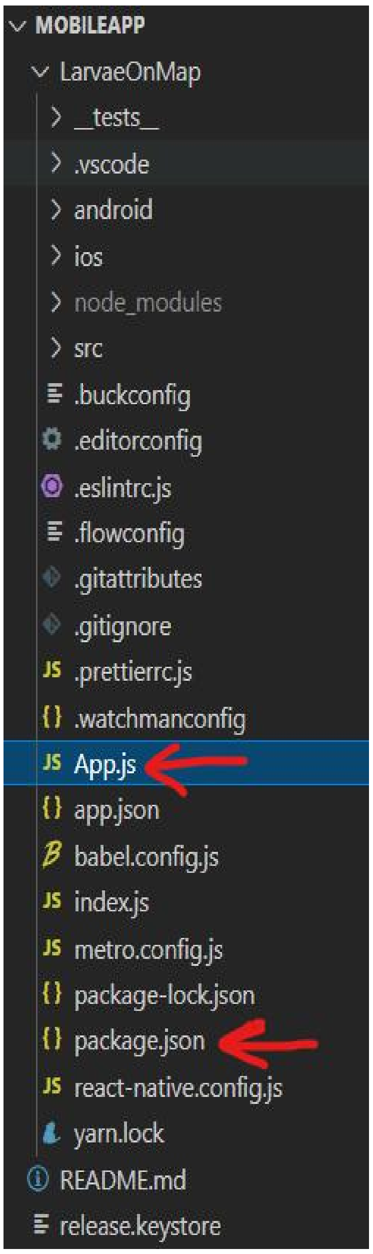
Next, starting up the new application using `react-native-cli.`

```
npx react-native init LarvaeOnMap
```

In the start the name was setup as **LarvaeOnMap** and later renamed to **LarvaeLy.**

App structure:

In general, to start with the `react-native-cli` we get this structure.



```
MOBILEAPP
  LarvaeOnMap
    > _tests_
    > .vscode
    > android
    > ios
    > node_modules
    > src
    .buckconfig
    .editorconfig
    .eslintrc.js
    .flowconfig
    .gitattributes
    .gitignore
    .prettierrc.js
    .watchmanconfig
    App.js
    app.json
    babel.config.js
    index.js
    metro.config.js
    package-lock.json
    package.json
    react-native.config.js
    yarn.lock
  README.md
  release.keystore
```

The most important of all is the **package.json** file where all the dependencies are installed into the **node_modules** folder.

Next, two native folders namely **android** and **ios** are there for all the Native code structure but here we are only concerned with the **Android**.

package.js

Lets quickly talk about the package file, in which it contains the most useful **npm** packages,
`react-native-screens`
`react-native-safe-area-context:` used specifically for navigation between the screens.
`react-native-paper:` For UI elements
`react-native-vector-icons:` For Icons,
`react-native-splash-screen:` For Splash screen or boot up screen.
`react-native-maps and react-native-maps-super-cluster:`
It is the most important packages on which we will talk shortly.

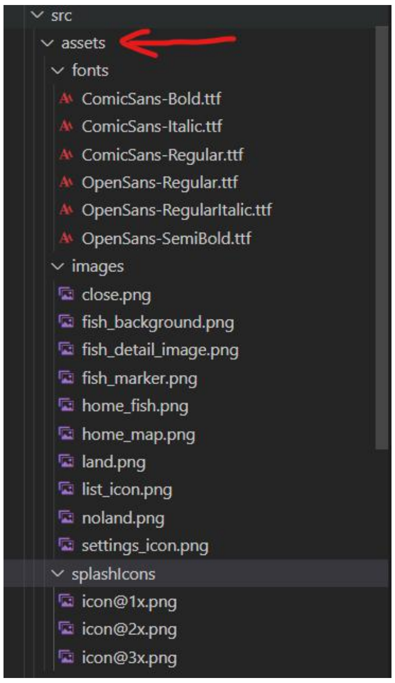
App.js

This includes the most initial screen of the whole application.
with the **SplashScreen** to get hidden when the app is loaded and returning the NavigationContainer as a parent tag, contains the **StatusBar** and **MainTabsNavigator**.

We will look into **MainTabsNavigator** later into ./src folder.
Next up inside **./src folder,** we first look at assets folder.

./src/assets includes:

In this we have all the fonts, images, and splash Icons, that was used to create the Android Splash Screen.



Fonts contains:
ComicSans - Bold, Italic and Regular and **OpenSans** Bold, Italic and Regular.

Images:
All the images are generally taken from google searched images and randomly edited on paint. (NO adobe **photoshop** tool used).

SplashIcons:
It has the three dimension images for the splash screen @1x (200 x 200) @2x (400 x 400) @3x (600 x 600) resolution.

./src/colors includes:

In this we have all the colors with its **HEX** code that are mainly used through out the application.

Next up we will jump into the **./src/scenes folder ,** in which we will walk through our discussed **screens.**

./src/scenes

In this first we will talk about the Tabs folder, which contains **MainTabsNavigator** on which we discussed previously as well. So, from App.js to this file, this contains all the information/components of all the scene which are being called through out this App. For instance; Home scene/ Map scene/ Fish details/ Settings scene.

This file is mainly acting as a routing file, in which I have used the `material-bottom-tabs` and then showing each of the tab individually: (Home / Map / Settings).

Notice that, inside the **Tab.Screen** of Map, there's another Stack Navigator: containing the Map scene and Fish Details scene.

```
import MaterialCommunityIcons from 'react-native-vector-icons/MaterialCommunityIcons';
import Ionicons from 'react-native-vector-icons/Ionicons';
```

Also, there's **tabBarIcon** as an option prop for the Tab.Screen and `react-native-vector-icons` used to display the icons from **MaterialCommunityIcons** and **Ionicons**.

Note: This is really straight-forward but it took me a lot of time to setup the navigation and combine them as tabs view application.

`./src/scenes/Home`

Now, Finally we move to our first initial scene; it contains basically simple JSX (JavaScript and XML) with the **ScrollView** from react-native, also the **Image** and **TouchableOpacity** used for the Map image and **Go to Map** button.
In between, you can also find some Text wrapped inside View tag, to show some information, from `import { Button, Text } from 'react-native-paper'`.

In addition, **HomeStyles.js** is there to bring some styles, alignment, justifications, fonts usage and so on. **Note**: the layout is build on **Flex boxes**.
Nothing fancy, but indeed time taking to building this complete scene.

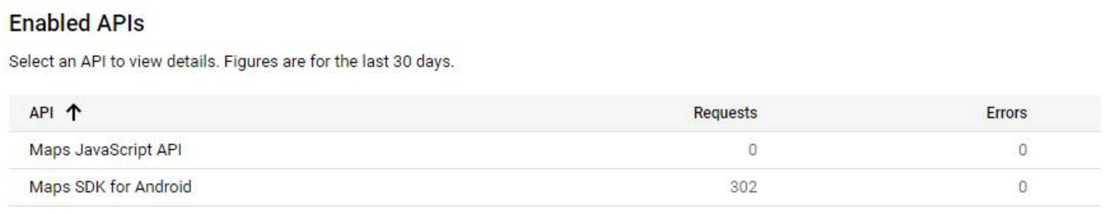
`./src/scenes/Map`

Next, we are now on Map scene, either from the bottom tab navigator or from the clickable button from the home scene.

To start with the Map scene, the most important part is the `react-native-maps` package, about which I talked in package.json part.

https://github.com/react-native-maps/react-native-maps

In initial commits; this scene was created to show the MapView with the provider as **google** and for that a project in Google Cloud Platform (GCP) is created to get the **API Key**, with the usage of **Maps SDK for Android** and **Maps JavaScript API** were enabled.


Enabled APIs

Select an API to view details. Figures are for the last 30 days.

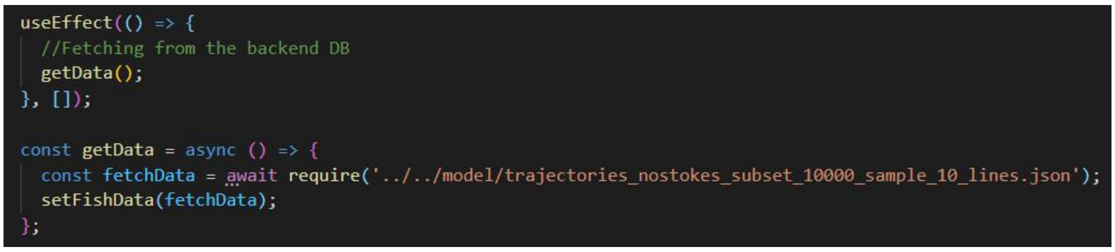
API ↑	Requests	Errors
Maps JavaScript API	0	0
Maps SDK for Android	302	0

With this, the **API_KEY** was added in the **AndroidManifest.xml** file and a number of configurations more.
This can also be found in: *commit/15e47255e46f8e49f13f63b07719d36bfea603fc*

Only to bring up the normal map on to the scene took a lot of time.
After this, Map scene default configurations were added, for example; **initialRegion** as latitude and longitude values, latitudeDelta and longitudeDelta also including the styles file and Map type is set to "**standard**".

To show up the, Zoom control buttons one trick was used to set the Margin **onMapReady** callback function, so as to fix the style of the Map and bring in the zoom in and out buttons on to the Map.

Moving further to the important file is from the **./src/model** which brings in the sample data points. This is indeed fetched using react-hooks inside the useEffect code block as shown below:



```
useEffect(() => {
  //Fetching from the backend DB
  getData();
}, []);

const getData = async () => {
  const fetchData = await require('../../model/trajectories_nostokes_subset_10000_sample_10_lines.json');
  setFishData(fetchData);
};
```

Here, it is clearly visible the **fetchData** is getting the data from the json file stored in the model folder, which will later change into the API call and data from the backend.

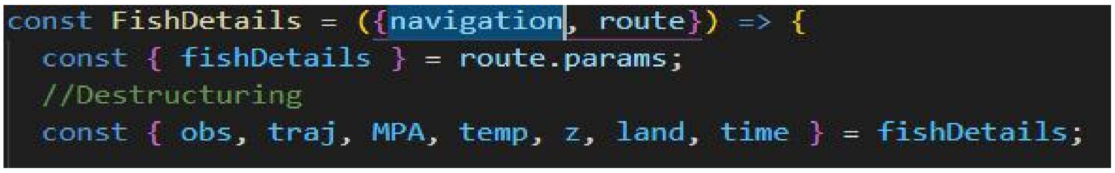
Finally stored in the state variable **fishData** and then inside the MapView, fishData is mapped to show up all the data points as Markers.

Now comes the interesting part to change the Markers into the customized fish Larvae icon named as **fish_marker.png** found in **./assets/images** folder.

Next, we have the **Callout** component from the `react-native-maps` displaying or acting as a tool-tip, indeed this also consumed a number of hours to set over the marker points correctly, and styling it with correct **fontSize** and **fontFamily**. This is the fish card showing the details of the fish larva, which is to bring up specific fish larva card on selection, and displaying some information like: Observation ID, Trajectory ID, and Timestamp.

./src/scenes/FishDetails

On clicking the fish card, navigating to specific fish card leads to fish details scene.
In this file, fishDetails from the prop is destructured and broken into multiple fields.



```
const FishDetails = ({navigation, route}) => {
  const { fishDetails } = route.params;
  //Destructuring
  const { obs, traj, MPA, temp, z, land, time } = fishDetails;
```

Also, note that navigation is passed from the previous map scene, for the cross button on top-left to close and pop out the scene back to its old map scene. This is wrapped into the **TouchableOpacity** with its **onPress** method. Moreover, to keep this look more aesthetic, **fish_background.png** found in **./assets/images** folder also added.

With the **infoContainer** and **boxContainer**; information was show as exactly displayed in the design with noticing the FishDetailsStyle.js which took so much of efforts to completely write the styles using flex boxes layout.

. Splash Screen

Now its time to discuss the Splash screen creation:
Following the guide from **Spencer Carli** of his medium article,

The *commit/5ce34dfad0b51eec5171bc0d007fafff316959a42* **and** *commit/55a98774db7ece8d84c2dbdb4453def691f07efb* shows all the configurations and set up in to the native android code namely: **MainActivity.java, AndroidManifest.xml** and the creation of **SplashActivity.java, background_splash.xml, launch_screen.xml, colors.xml and styles.xml** in the res folder of android native files.

This is extremely delicate work just to display this boot screen with the help of the package: `react-native-splash-screen`

Moreover, re-sizing the splash icons on specific resolutions: @1x (200 x 200) @2x (400 x 400) @3x (600 x 600) resolution and dropping them in to the ./res/mipmap specified folders.

. App icon

Next up we have this small fish larva app icon, it is the logo which is with the name **LarvaeLy** on the home menu screen of our smart phones.

To build this, https://appicon.co/ and https://easyappicon.com/ these two links were used to get the output icon files and placed in the resource folder of android native files. *commit/2e49dec5ce3c8517cd40ae08eec35fbd7f2a829d* can show all of these. These can be found in **./res/mipmap-hdpi** to **./res/mipmap-xxxhdpi** with **ic_launcher.png** and **ic_launcher_round.png** named for the icons.

. react-native. config

To build up the fonts which is clearly as same as the **FontFamily** used all over, this is bit of a tricky and time consuming task.

This file is specifically for the fonts setup in to react-native platform.
After pasting the .ttf files of ComicSans and OpenSans added in the assets folder, `react-native link` is executed to compile all the fonts natively into the Android ./res folder.

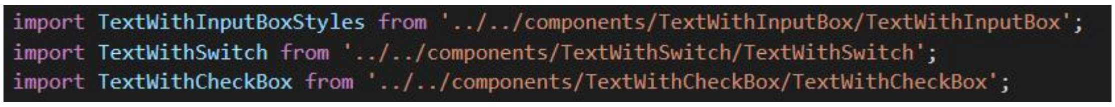
https://mehrankhandev.medium.com/ultimate-guide-to-use-custom-fonts-in-react-native-77fcdf859cf4

Next, coming back to the scenes directory, so far we have already discussed most of the scenes yet developed and now we will discuss the settings scene.

./src/scenes/Settings

As already discussed this part provides extreme usability to its user for as to changing or modifying the filters as based on their preferences.

Settings scene contains some of the **.src/components** namely TextWithInputBoxStyles, TextWithSwitch, and TextWithCheckBox.



```
import TextWithInputBoxStyles from '../../components/TextWithInputBox/TextWithInputBox';
import TextWithSwitch from '../../components/TextWithSwitch/TextWithSwitch';
import TextWithCheckBox from '../../components/TextWithCheckBox/TextWithCheckBox';
```

These are the components created just for the refactoring and to increase more re-usability, thus decreasing the file length.

This file so far in the *commit/9048aef0b1e3911588ca4f8fff8c623f60938af1* shows simple UI with fixed values and toggle buttons (**Switches**) to be used for the later purpose. Mainly considering the elements Image tag and Text tag wrapped inside the View as parent to display the headlines, Simply the layout with styles file.

The **bodyContainer** wraps the core components TextWithInputBoxStyles, TextWithSwitch, and TextWithCheckBox as having the **textLabel** and **val** as prop to low level or child components.

These three components on a higher view, uses `react-native-paper {TextInput, Switch, CheckBox}.` As a result this took incredibly large amount of time and efforts to developed solely for the purpose of UI for now, as this is precised and too detailed in design.

But, in consideration to link this with the back-end API (All the endpoints are missing) due to which this scene cannot serve its original purpose. Thus there is no natural fetching of analyzed and optimized data from the other team. This really disappoints the app on this stage without having the relation with the original or large scale data, just because of the absence of endpoints from the back-end team.

As now, this scene is only rational if in future might be connected with the API but for now its only a static UI elements. Also, note that the sort functions would be useless to perform as because on the map the sorting list wont reflect the View. Well, the presence of Land or No land fish points has its meaning but, it is possible to conflict with the filter of observation ID and trajectory ID.

I simply kept them here for now but can be improved or modified later on demand.

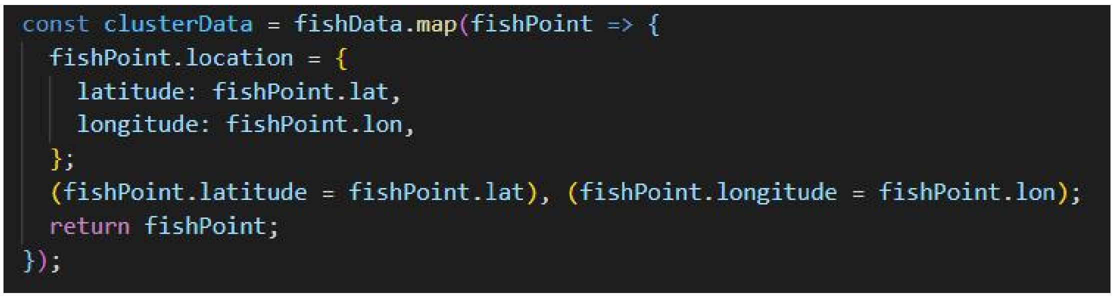
./src/scenes/Map -> ClusteredMapView

Now, I would like to continue on Map scene and discuss the very important part of this which is **ClusteredMapView** from the package `react-native-maps-super-cluster.`

https://github.com/novalabio/react-native-maps-super-cluster/

```
import ClusteredMapView from 'react-native-maps-super-cluster';
```

ClusteredMapView contains more or less similar behavior as from **MapView** but it has more aspects to discover. About the props, firstly, onMapReady, mapType and zoom controls options are the same as discussed in the Map scene, but we have the **data** which is actually mapping over the fishData and is passed in the form of **clusterData**; which is first transformed into the **fishPoint.location** object, as shown;

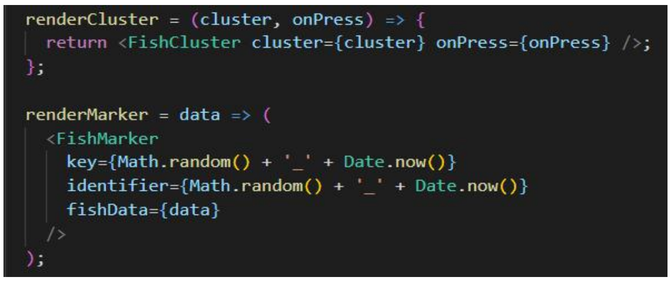


```
const clusterData = fishData.map(fishPoint => {
  fishPoint.location = {
    latitude: fishPoint.lat,
    longitude: fishPoint.lon,
  };
  (fishPoint.latitude = fishPoint.lat), (fishPoint.longitude = fishPoint.lon);
  return fishPoint;
});
```

About the **initialRegion**, it is hard coded value in the **const** object same as for the Map scene. **Line 14**

Next comes the two most important functions **renderCluster** and **renderMarker.**

Here, the **FishCluster** and **FishMarker** are not predefined or already built components but I have created them inside the **./src/components**.



```
renderCluster = (cluster, onPress) => {
  return <FishCluster cluster={cluster} onPress={onPress} />;
};

renderMarker = data => (
  <FishMarker
    key={Math.random() + '_' + Date.now()}
    identifier={Math.random() + '_' + Date.now()}
    fishData={data}
  />
);
```

FishCluster: This is the **cluster** which is displayed when the two or more markers, fish points come closer to each other. In this component, I used the **Marker** from `react-native-maps` and the cluster is destructured into **count**, **coordinate** and **cId** which is the cluster **id**. Inside the Marker tag, coordinate is where the cluster should be located, with that the cId as **key** and finally the **count** which is the number of points grouped together visible on to the Map and wrapped inside the Text tag.

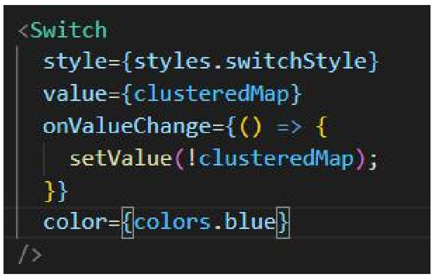
Notice in **FishClusterStyles.js** we have three important styling **classes** named as **clusterBox, counterBox and countStyle** to make this cluster look more aesthetic and closer to design, but not exactly like the design. This is very simple on overall look but really very time consuming.

FishMarker: It is basically the customize marker which is present or visible on to the Map. Inside the **./src/components** I have defined this Fish Marker explicitly with the customized fish Larvae icon named as **fish_marker.png** found in **./assets/images** folder. This is very straight forward.

Now, here comes the problem, with the ClusteredMapView, I have spent so much of time to let the markers stop blinking when its **focused** or clicked on the Map.
Also, when the Marker is clicked the tooltip **(Callout)** is visible only for short time and then waved off because of the constant resetting of the state values.

This issue is still **open** *https://github.com/novalabio/react-native-maps-super-cluster/issues/63* and unsolvable on my point. After spending so much time, it was a good idea to separate the Clustered Map with the Normal Map.

For this purpose, I have introduced a toggle button **(Switch)** on the top left corner, just to enable and disable the MapViews. This **clusteredMap** is a state-variable and set inside the switch from `react-native-paper`. **onValueChange** is a callback which sets the value of clusteredMap.

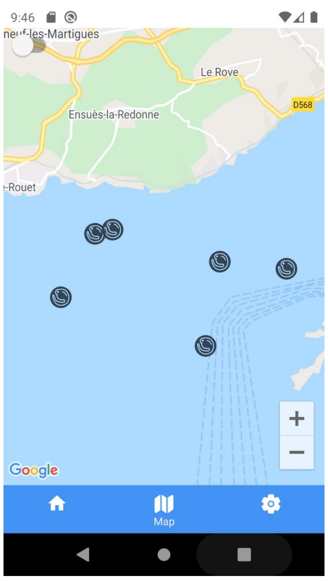


```
<Switch
  style={styles.switchStyle}
  value={clusteredMap}
  onValueChange={() => {
    setValue(!clusteredMap);
  }}
  color={colors.blue}
/>
```

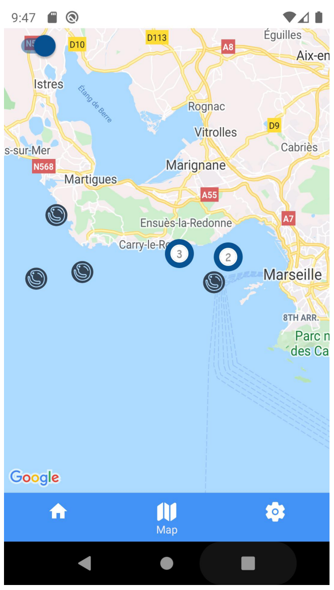
Using this technique, now, in the Map Scene, **conditional rendering** is performed whenever to show the **ClusteredMapView** or **NormalMapView**.

Note: When using the ClusteredMapView, it is **not possible** to go to Fish details scene, as there is no way to enable the tooltip or the FishCard visible. For this, to navigate to the fish-details scene, FishCard component has been made only available inside the NormalMapView. It is indeed exact same MapView which we have seen before in the Map scene discussion. All the fishData has been passed from the Map.js to the child NormalMapView.js which is inside **./src/components** and also the navigation prop so as to reach to the fish details from FishCard.

NormalMapView



ClusteredMapView



In this way, it is possible to jump back and forth between a **NormalMapView** and **ClusteredMapView.**

Now, at this point the **Map.js** is completely modified and refactored as observed in this *commit/f05838073aa2d9802339cb5ffa290a14208e42f9* which was recently pushed to the **S-mobileApp branch.**

Note from **Line 26 - Line 39** it is commented and marked as **old code for fetching data**, this is because the **fetchData** was mocked from the **./src/model** but when integrating the ClusteredMapView, fetching takes a bit of time where the fishData remained **undefined**, resulting in the crash of the application.

To resolve this problem, **fishData** is directly attained from the file, and not using the **async** and **await** technique, so as to have it ready before the view is completely loaded.

Conclusive Notes:

There is still some room for more improvements and refactoring but that is just for the sake of code cleanliness.

Furthermore, some aspects of the App is still missing which is indeed important and necessary but due to the failure in construction of endpoints from the backend team and it is not really possible to show all the data points directly from the raw files into the application, because ultimately if loading the 10000 data points on the map view results in the crash of the application, but overall really a nice experience in learning and working with this type of project.

Overall, so far so good, this is finally it for the **visualization front-end** part mainly focusing the **Mobile Application** - **LarvaeLy.** With only more target on designing and in creation of the UI elements but with functional MapView and ClusteredMapView features will small enhancements summed app in the Android application.

Thank you

Report by:
Shafiq Shams (stu219910)
September 21, 2021