General Overview

I've been testing on a phone with KaiOS 2.5.1.1, which is running Firefox 48. Most web features supported after this version (such as multiline strings) most likely will not work. All application files should be packaged in a zip file with a manifest file describing the security level of the app and the specific permissions that this app requires, such as geolocation. More information on the manifest can be found following this link.

Views

The application itself is a simple website (HTML, CSS, and Javascript). A good example app can be found here. The entire app is built upon one HTML page which is split into multiple <div> tags with the "view" class. By default, the "view" class is hidden through CSS styling, but is given an additional "active" class to denote when to display any view. An active "view" is displayed as a grid with two columns by default and all the items added to it as its children will dynamically increase the rows in the grid as necessary.

Data Persistence

As a web app, there are two options for data persistence: local storage or indexedDB
I would recommend using local storage as it is simple and should be enough for storing small amounts of data (up to 5 MiB). Local storage stores things using key-value pairs, but as most Javascript objects can be serialised and deserialised into JSON, you can store more complex

information as well. If you need to use indexedDB for larger quantities of structured data, I would recommend reading this page from KaiOS regarding some of their recommendations.

Navigation, Navigation Items, and Navigation Key Handlers

To make the app simpler to traverse, some number keys are mapped to navigation as well. This can be modified by modifying the event listener for "keydown" event types in *app.js*.

Selectable objects in a view are given the attribute "nav-selectable". The STATE object keeps an ordered list of the navigable items of the current view and reloads a new list when a new view becomes active. For views that have multiple traversable objects, it is necessary for that view to have the attribute "nav-column-size" with the corresponding number of columns. This value is used to increment/decrement the selected item index when navigating up or down the grid. The app is implemented this way as initially I wanted to avoid having an assumed number of columns and it stuck.

The *focus()* method is called to change the selected item. It is important to note that normally unfocusable tags such as <div> require the attribute "tabIndex" to be set or you will not be able to *focus()* on them. When focused, the selected item also has the "nav-selected" attribute set to true. This is styled differently in CSS and provides more visual feedback to the selected item. I would recommend changing the current background and focus colours as these were chosen arbitrarily and may not be the best under bright sunlight.

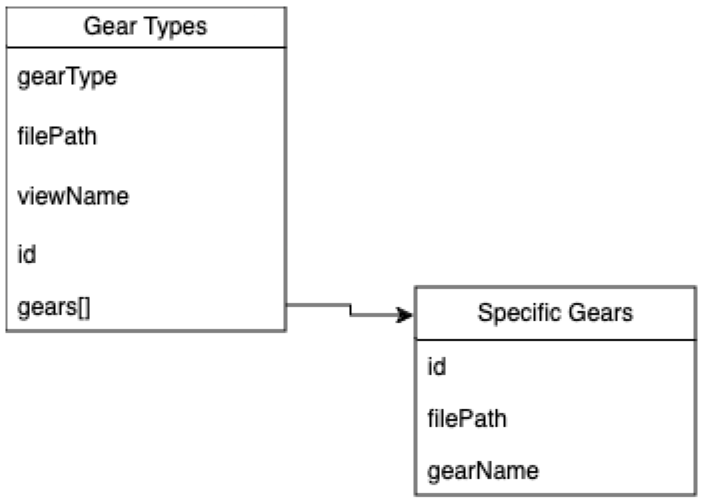
The behaviour when the enter or softleft keys are pressed is mostly unique between views. That is why each view is assigned its own *enterKeyHandler* and *softleftKeyHandler* method which details what should happen when the enter key is pressed. Note that these methods are not in-built JS ones but are assigned to them in the program.

Image Boxes and Configuration

Most of the selectable items in this app are currently a <div> tag with the CSS class "imageBox". These boxes then contain an image and a text underneath that image. These containers have an additional attribute "localID", which is the integer ID that identifies the thing this box represents in its local group (such as fishes, units, gears etc.).

The application is technically a web app, so it is impossible to perform file traversal to load the images at runtime. That is why images and subtitles for units, gear types, specific gears, and fishes are loaded based on the configuration file *config.js*.

There are three pieces of information that is necessary for all categories: an integer "id" that is unique within each category, a "filePath" denoting where the image file is located, and "[type]Name" that is the subtitle/hint text underneath the image file. Gears have an additional layer due to their hierarchical implementation and require more information. For each gear type, it requires an additional "viewName" field which is a unique HTML id assigned to the view of this gear type and a “gears” array that contains the specific gears under this gear type. So all the gears in this array will be loaded to the newly created view for this gear type.



To change the image for the done icon, change the *DONE_BUTTON_FILE_PATH* field in the config.js file.

Mapping

The map functionality is implemented using the LeaftletJS library, which has been downloaded and included with the app in the *lib* directory. This library requires the mapping tiles to be already downloaded as part of the app in the OSMdroid format. These tiles are memory intensive so it would be best to limit the bounds of the map (using *setMaxBounds()*) to a known area and only download and ship the relevant tiles. This option was chosen as other mapping APIs (such as Google Maps) require an active network connection, though if an active network connection is available, it should be relatively easy to switch out to use the Google Maps API as that is just loading the relevant JS file. To download tiles, you can follow this guide on StackOverflow (or get them from Open Street Maps directly, but this may require further processing). The default keyboard controls have been disabled to remap controls. The map object is stored in the STATE.mymap field and must be called for all interactions related to the map.

Technical Details

State

“STATE” is a global object for the app to track application state. This state does not persist if the app is closed. The STATE object contains the “currentRecord” field, which tracks the information currently being input by the user. Right now, this information is not persisted. The record is cleared every time a new record is started, so any manipulations or persistence will need to happen before then.

Geolocation Details

The geolocation permission must be added to the manifest in order for the geolocation api to be used. In order to keep geolocation running in the background, you need to request a “gps” wakeLock. This prevents the phone from going into deep sleep and causes significant battery drain (on full charge, about 10% per hour). After requesting the wakeLock, you can then assign a handler function by using the *Geolocation.watchPosition()* method that will fire every time the position of the device changes. However, upon testing, this method seems to fire about every second, and there does not seem to be a way of slowing it down other than to simply set a cooldown timer and ignore it firing during the cooldown timer.

The fact that the KaiOS phone might be used purely for data collection may actually be useful. The phone does not immediately shut down an app or prevent it from running in the background when exiting it. However, if another app is opened and requires a lot of resources (such as google maps), the background app will be stopped, including the *watchPosition()* handler.

Sideloading Applications

To load an application onto the phone, you may follow the tutorial by KaiOS. You will need Android Debug Bridge (ADB) installed and a way to build the app on the phone. For newer devices that are no longer able to run the old versions of Firefox, you can try using WaterFox Classic’s WebIDE (Tools > Web Developer > WebIDE). This version is insecure as it lacks security patches.

Debug mode must be enabled on the phone, a detailed guide on how can be found here. Try the browser debug first as it is simplest.

Helper Functions

changeViewTo(viewName)

Provided with the HTML ID of the new view, this function will take the necessary steps towards making the new view visible. These steps are:

1. Remove the active class from the previous view,
2. Add the active class to the newly selected view,
3. Load the new list of navigable items for the new view into the *STATE.naviItems* field
4. Set the focus to the first element in the of the new list of navigable items
5. Set *STATE.activeColumnSize* to the correct value by parsing the “nav-column-size” attribute
6. Change the softkey bar text

addNewImageBox(gridContainerID, localID, filePath, subtitle)

This function adds a new navigable image box child to a view. The parameters are: the HTML ID of the view you want to add to, the local integer ID of this new box, the file path to the image, and a string that is the subtitle to the image. This function will create a new child node and append it to the specified view's children nodes.

addStaticImageBox(gridContainerID, id, filePath, name)

This function adds a non-navigable image box child to a view. This function is similar to *addNewImageBox()* except the id provided will be the HTML id of this new image box rather than the localID.

addDoneButton(gridContainerID)

This function adds a navigable "done" image box. It takes the HTML id of the view to add the done button to as a parameter. The "localId" of "done" image boxes are always set to -1. It loads the image from *DONE_BUTTON_FILE_PATH* from the config file.

App Limitations

Currently, the app does not provide enough feedback when a bad operation happens, such as when no gear is selected on registration or if no fish is chosen when registering a catch.