**CURE Technical Instruction Document**

**Written by: Joshua Brown**

**(1) Introduction**

The Cure Application was designed using Meteor Framework. The main system has 6 major subsystem components, each containing their own private functions and some are interconnected via a backend event handling system. The purpose of this document is to provide developers some simple, direct instructions for getting the Cure Application up and running in a development environment and for how basic design of the system to better understand how to work from the code implemented.

**(2) Setting Up Meteor to Run Cure**

**Required Tools**

The Cure App is built on Meteor but has many other major framework specific tools that are used. Upon first building (and subsequent builds) the application developers will need the following:

* Meteor (install via their instructions based on your computer OS)
  + Cordova is a core component, pre-built inside of Meteor which builds the iOS and Android apps
* Android Studio (with Android SDK)
* XCode v7 or higher (if on Mac)
* Plugins from Meteor and NPM (Described Below)

**Running Cure App**

Meteor can build as a web application by simply running in a terminal (or ‘cmd’ in windows):

***meteor run***

After which case, simply visiting url: *localhost:3000* in a web browser will display the application but key functionality will not work. In order for the application to run via Android or XCode there is some additional setup required.

**Android Use**

* Install *Android Studio* and run it
  + Doing so should automatically install Android SDK and emulation tools for Android
* Set up a phone emulator via the AVD Manager in Android Studio
  + This should be under **Tools**->**Android**->AVD Manager
  + NOTE: You need to install API version 26 or higher on the AVD Manager in order to run Cure. The app runs on lower version actual phones, but not on lower version emulated phones.
* Make sure to install at least version 8.1 of *Java*.
* Set up some (local or system) environment variables as follows:
  + Make a new variable called ANDROID\_HOME
    - Set the value to the path to the root of your Android SDK folder (probably inside of Android Studio folder somewhere unless you specified a different location during installation)
  + Under your currently existing PATH (or Path) variable add the following folder paths (exactly):
    - %ANDROID\_HOME%\tools
    - %ANDROID\_HOME%\platform-tools
  + Also add your Java SDK path to your PATH variable (ONLY if you didn’t add it automatically through the installer)
    - Usually: C:\Program Files\Java\jre-*version#here*\bin
    - NOTE: if you are running a Java 8 version number you will likely have to use *jdk* instead of *jre* in the path.

Having done the following, developers should be able to now add android to the meteor project and run it via an emulator or a real phone. Further instructions for this setup are located at:

<https://guide.meteor.com/mobile.html#installing-prerequisites-android>

To run via emulator developers need to travel (via cmd or terminal) to the source code folder containing a “.meteor” folder. The location should be under “/src/Cure” if the project was cloned/downloaded from GitHub. Once there, run the following commands:

***meteor add-platform android***

***meteor run android***

If the developer wants to run directly to an Android device they must plug the device into their PC via USB and have the device running developer mode with USB Debugging enabled under the developer mode menu. Afterwards simply run the following command:

***meteor run android-device***

This will directly install the app to the phone and launch afterwards.

**iOS Use**

For iOS one simply needs to install their preferred version of XCode (v7 or higher) from the App Store. Once installed the emulator can immediately be used after adding iOS as a platform.

In terminal on a MacOS computer/laptop navigate to the project source code directory, “/src/Cure”. Then run the following commands:

***meteor add-platform ios***

***meteor run ios-device***

From our experience simply running the “ios” argument doesn’t open the emulator from XCode, so it’s better to run “ios-device” which will open XCode and build the app. You will then be able to choose which device you want to run (ios simulator or real hardware) from the drop-down in the upper-left next to the name of your application. (This is to the right of the “Play” and “Stop” symbol buttons). For real hardware (iOS Phone) see below.

**Real Hardware for iOS**

The application cannot be run on a real device unless you are part of a development team (or an individual development account). Contact your team management/project owners to be added to their development account. Below is a link to all of the relevant instructions for becoming an Apple Developer and running an app on real hardware once your account is verified.

*Adding an Apple Developer Account to XCode Project*

<https://help.apple.com/xcode/mac/current/#/devaf282080a>

*Signing Workflow*

<https://help.apple.com/xcode/mac/current/#/dev60b6fbbc7>

Any further instructions needed may also be located at Meteor’s site at the following page:

<https://guide.meteor.com/mobile.html#running-on-ios>

**(3) Package Setup**

There are many smaller packages from Meteor and Node Package Manager (NPM) which have to be installed for basic functionality of subsystems. Below will be separated both Meteor (Cordova) packages that should be run and below that, NPM packages.

*Meteor Packages:*

Frozeman’s persistant-minimongo

This package allows us to store any mongodb data

generated locally into the meteor project’s filesystem

so it can be recalled later.

Cordova Plugin Calendar

This plugin helps Cure communicate with the user’s

native calendar on either iOS or Android.

Cordova Plugin Call Number

This plugin allows Cure access to Android’s and iOS’

phone calling system (used for Hot Line)

*Installing Meteor Packages:*

Run the following script in cmd/terminal at source code:

***meteor add frozeman:persistant-minimongo2***

***meteor add cordova:cordova-plugin-calendar@5.1.4***

***meteor add cordova:cordova-plugin-call-number@1.0.1***

*NPM Packages:*

There are many packages from NPM so instead of listing them we will simply provide the command necessary to install them all. The command is as follows:

***meteor npm install --save @babel/runtime @fortawesome/fontawesome-free bootstrap jquery popper.js tiny-date-picker leaflet leaflet-geosearch*** ***moment***

(NOTE: this NPM command is all ONE LINE with spaces delimiting each plugin to install)

All plugins should be installed automatically. Just make sure spelling is accurate as incorrect spelling will cause freezing of NPM while it searches its entire database for the incorrect plugin.

Once all plugins are installed the application should run correctly.

**Note about Cordova Plugins**

All Cordova plugins are installed using the normal “meteor add” command. However, for cordova plugins the installation is a little special because Meteor does not directly install the cordova plugins once the developer adds them like with other plugins. Instead Cordova plugins are only properly installed once the application attempts to run on a mobile device (or mobile emulator). Meteor does this by reading the plugin added and calling Cordova’s plugin installation functions.

As a result there are specific criteria for adding Cordova plugins so that Meteor knows to send a plugin to Cordova during build time. The structure of adding a Cordova plugin is as follows:

***meteor add cordova:****<plugin-name-here>****@****<version-number-here>*

The bold text is always entered as-is. The bracketed text is subject to whatever plugin is being installed. The <plugin-name-here> is the name of the plugin (without spaces). The “@” symbol is required after the plugin name. The <version-number-here> is the specific version number (usually of format: x.x.x eg: 2.0.7). Given that all required criteria are met, the application should build without errors. Usually if errors are encountered that stem from Cordova it is because of an incorrect plugin addition or misuse of an installed Cordova plugin.

**(4) Code Structure**

The Cure application separates each subsystem into their own directories, which handle the visuals (HTML) AND background coding (Javascript) as necessary for that subsystem. Everything is generated starting from the main.html and Controller.js files. The main.html file calls the main page and Meteor loads all requisite html and Javascript pages up to three subfolders deep. These html pages are hidden from main.html by use of *templates* which are called as needed when new pages are accessed.

All Javascript files are loaded immediately by the Controller inside of Controller.js via instantiation of class objects which link to each specific subsystem’s Javascript code. Subsequently, whenever a subsystem needs to call on some local event (such as a button press or text entry) they can simply call it and it will be loaded via the correct Javascript code.

**Blaze Templating**

Meteor, by default, uses Blaze for templating HTML content and serving it when templates are called. This is used extensively throughout the Cure App in order to hold off displaying content until it is called by a user function. This could include pushing a button on the navigation bar, creating a text event and more.

On the main.html document we make use of dynamic templates to generate new pages to the main page. All basic headers and navigation bars are built on the main file. The center content is generated dynamically through a template located in main.html.

In future, when a developer wants to design a new subsystem for the application they would create a new HTML document. At the start of the new document immediately create a new template as such:

***<template name=”mynewmenu”>***

***...html code here...***

***</template>***

It should be noted that there is no need to create <DOCTYPE>, <html>, <head>, nor <body> elements. These are already available to the app always via the main html file. Anything that needs to be added to the <head> should be added there in main.html. Also remember that Meteor loads all Javascript files three subfolders deep from the root directory. So it may not be necessary to include a <script> tag for some Javascript file unless it’s too deep to be recognized by Meteor.

**Calling a Template**

Once a template has been created it can be linked to any page via a Javascript file which references that template. Simply create a new Javascript file where the new HTML document was created and create a new class. Ensure that you create a class with “export default class <class\_name>”. This is imperative so that the class can be seen by the Controller, in addition to including the .js file to Controller.js that will be discussed later.

Once the class is created create this.var\_name for each template function needed. An common example would be:

***this.templateEvents = Template.HotDial.events({***

***...code here...***

***});***

Notice how we reference Template.HotDial.events(). Every Blaze template has access to some default template functions. The criteria is

*Template (calling a template). <template\_name> (the name*

*that was given to that template).events({}) (the events*

*function where you will call local events such as*

*“click #\_id”)*.

Once template events are instantiated we should include the javascript file so it can be seen by Meteor. Open the Controller.js file. Now, at the top include the statement:

***import <Class\_Name> from “./path/to/class.js”;***

The <Class\_Name> is the actual class name inside of the Javascript file being imported. After ‘from’ is the path (as a String) to the .js file.

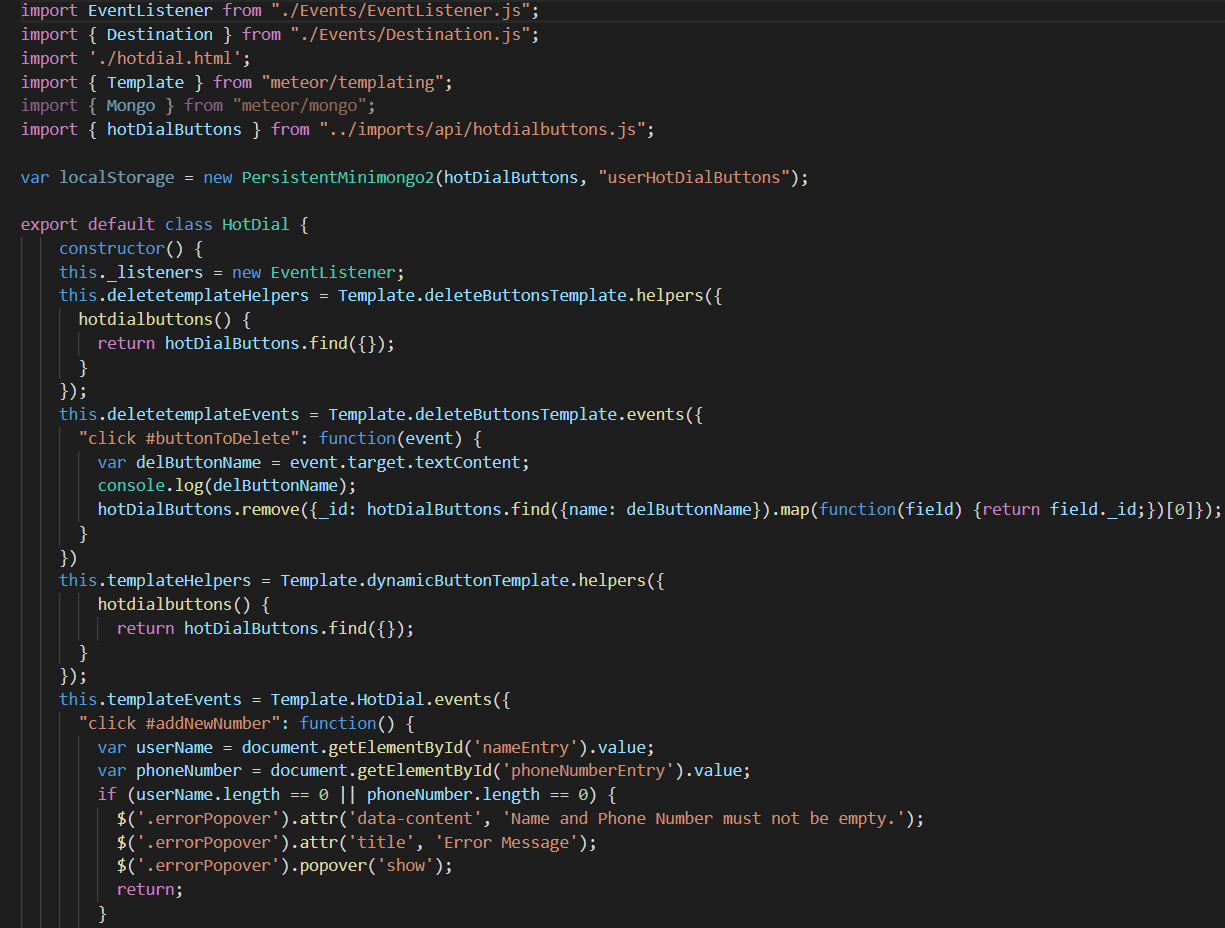
Once this is included in the Controller, you must add a proper instance of the class as an object. Do so by simply adding under the “constructor()”:

***this.var\_name = new <Class\_Name>();***

Now that the class is instantiated, the Javascript will be immediately called upon launching of the application. Below in *Figure 1* is a sample of created class which has some functionality built into it.

Further materials on Blaze Templating are available here:

<http://blazejs.org/guide/introduction.html>



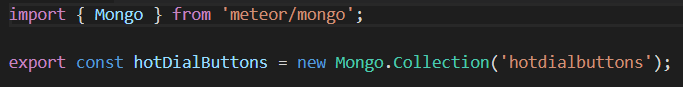
**Figure 1**: Here is a sample from the HotDial.js (Hot Line) showing class

creation

**Working with MongoDB**

In case a newly created subsystem needs to use MongoDB (the built-in database technology for Meteor), we’ve integrated a plugin which manages MongoDB locally. In this way, a subsystem’s database information does not need access to an online server to access or modify contents of a minimongo database table. This is important as user information should remain private and only accessible to the user. The only exception to this rule is the services run by the app that are not user generated (such as Leaflet Maps).

To begin using a MongoDB table, first navigate to “/Cure/imports/api” folder. Here create a new Javascript file which will only hold the newly created MongoDB Collection (table). Below in *Figure 2* is an example showing creation of a .js file which will hold the table.



*Figure 2: hotdialbuttons.js, which only contains the Mongo Collection*

It is necessary to create the Mongo Collection here because both Server-side and Client-side can read it. Although we are not using server-side connections most of the time, Mongo is designed to at least attempt to read it, and if it cannot at least see the server-side code (located in “/Cure/server/”) then it will fail to create or read a collection.

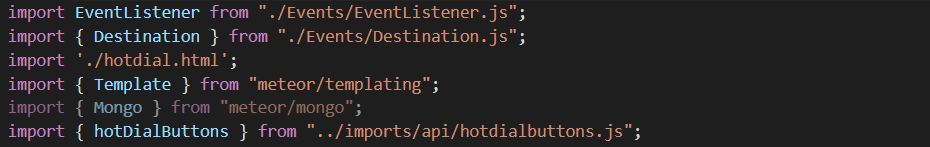
Now navigate back to the source code for your class (the Javascript file). You will need to import this mongodb as a Static Class (using brackets around class name). An example is below in *Figure 3* showing importing of “{ hotDialButtons }” as a Static Class.

From this point, **above** the creation of the class you need to create a new variable (whichever name you want) that is set equal to the PersistentMiniMongo object. The exact setup for this is as follows:

***var localStorage = new***

***PersistentMiniMongo2(hotDialButtons, “userButtons”);***

The first argument of the PersistentMiniMongo2 object is the actual MongoDB you wish to store/load locally (yes, it does both based on whether it’s found or not). The second argument is the name tag for the locally stored MongoDB. This name is used to create a new local storage instance and to search for an already existing instance of that name. So this one command will both load an existing instance of this local MongoDB or it will create a new one if it doesn’t exist.



*Figure 3: hotdial.js - import for MongoDB object hotDialButtons*

Now you’ve created a local storage point in the app to store the all MongoDB information so that no server is required. You can use the MongoDB object to add, remove, etc. just like usual, but can save it without need of a server running to keep track of the contents.

**(5) File Structure**

The file structure of Cure’s source code follows the Meteor application standard whilst taking a few liberties for internal files. The file structure is as follows from the highest level files down to inner sub-directories:

* .meteor
  + local
* client
  + Calendar
  + dist
  + Education
  + Events
  + Goaltracker
  + HotDial
  + Map
  + Settings
* imports
  + api
  + ui
* node\_modules
* public
  + fonts
  + graphics
  + images
* resourcesAdd
  + local
* server
* mobile-config.js

**.meteor**

This folder contains many rudimentary files which are needed to load plugins from meteor and, more importantly, Cordova. Most files stored here are only there as reference points for when ***meteor run*** is called so that Meteor knows what plugins to install and what settings to enforce on the application. In most cases you should never directly modify these files, let Meteor do it when you run commands through the command line tools.

**local**

The local folder contained inside of ***.meteor/*** is the folder that is automatically generated for you application whether being run as a web app or as a phone app. This contains all necessary components, plugins, and even Cordova itself is installed at build time here. You will never need to directly edit these files unless you are trying to modify icons for the Android and/or iOS builds. In this case, we recommend copying the folders from ***resourcesAdd/*** and following the instructional text file there.

**client**

The ***client/*** folder contains the bulk of the application. Anything inside of this folder will be served only on the client-side of the application. That means that any MongoDB calls will fail if run from code inserted inside of this folder. This is because at runtime Meteor reads anything inside the ***client/*** folder as ONLY local, client-side data, inaccessible to server-side (which MongoDB needs). This would remain true for any data or functions which might need some kind of emulated (or real) server.

The bulk of data inside the client-side are the visuals and backend coding related to the visual elements. The visual elements are coded in HTML and stored in subfolders depending on the subsystem for which they are a part. Like the visual side the backend code is stored with its respective subsystem, and is coded in Javascript. The few CSS files contained within are merely for styling of the HTML pages. In most instances the CSS files aren’t used, but instead Bootstrap classes are used for styling. Mostly the CSS files are used for tweaking existing Bootstrap classes or making fine-tunings.

This document will not directly go over the contents of each of the subfolders of ***client/***. This is because each folder is relatively self explanatory, as they mention subsystems of Cure for the most part. The exceptions are the ***dist/*** and ***Events/*** folders.

**dist**

This folder contains key information used by the Leaflet maps import. The Map subsystem uses Leaflet to deliver map services to Cure. This component is unfinished, but the files contained within exist to generate content on the map.

**Events**

This folder contains purely Javascript implemented event classes. These events are used by the Controller and some of the subsystems’ main Javascript files in order to communicate and pass data to one another. This was necessary as data passing from one Javascript file to another isn’t possible with typical Template events or local events (such as button click events) unless we shared many global values across the entire application. To avoid this, an event system was constructed which asks that any message passing subsystem should implement the ***handleMessage()*** method/function so that they can receive messages.

**imports**

Here lives any data that might be shared between both the client and the server. When Meteor runs it specifically checks here only whenever files here are called from either server or client side. It also offers the benefit that if code needs to be run locally (client) but needs MongoDB (server-side) it can be created here. In this way, we created all MongoDB objects here and export them out for the client code to call as needed.

The subfolders don’t contain much, so won’t be mentioned in detail. The ***api/*** folder contains all MongoDB creation instances. The ***ui/*** folder contains Goaltracker visual and backend coding data. In theory, this folder could be used for the entire application.

**node\_modules**

This folder may not be included whenever the Cure app is first cloned from Git. However, this folder contains the entirety of all NPM modules. Whenever a new NPM module is added from the Meteor commands they will be generated here. If one of the files needs to use the services from here they will have to have an ***import*** statement (if Javascript) or a ***<link>*** tag (if HTML) which directly references the path to the needed files from the modules stored in ***node\_modules/***.

Upon the first time Cure is pulled from Git that developer will likely have to use ***meteor add npm <module\_1> ... <module\_n>*** in order to add the required modules. If Meteor cannot find a module it will give an error, indicating as much and most of the time will provide the command statement that’s needed to run to allow compilation.

**public**

The ***public/*** folder is used by Meteor to read in images, backgrounds, fonts, and other related public information used by Cure. We store images and icons here. Some are used in app, and some are used later during compilation. Those used later in compilation (such as icons) are here only for storage purposes and must actually be copied to another location (specified later). The subfolders are self explanatory so this document will not go into further detail. Just know that if background images or fonts need to be added, they should be added to the respective subfolder.

**resourcesAdded**

This folder is not used by Meteor or Cure at all. This folder merely stores instructions and files corresponding to icon data so that whenever Cure builds an icon can be seen on Android and iOS.

**server**

Anything that needs to be run specifically on the server alone is stored here. Cure, as it is currently built, only uses a *main.js* file which runs the server. This is done so that MongoDB will accept creation and queries. Otherwise the main reason to put files here are if they need to run specifically on the server and should only be accessed by the client via a direct query or call to the server first.

**mobile-config.js**

This is not a folder, but rather an important Javascript file which is run by Meteor to do initial setup for Cure before building the mobile app through Cordova. Here we use the ***App*** keyword class to call innate functions for setting up effects, icons, app id, etc. for both Android and iOS. Some of the basic commands are as follows:

* **App.info**
  + Expresses key information about the application whenever it is launched on the app store.
* **App.icons**
  + Different key-value pairs for Android and iOS sized icons. This includes the values as paths to where the icons are located (***public/*** folder used here).
* **App.launchScreens**
  + This sets up the splash screen that shows as the app is being opened and during the initial loading period on mobile phones.
* **App.configurePlugin**
  + Sets specific plugins preferences. This is typically for plugins which integrate with some feature native to the phone or possibly available on the phone. This may include private keys, Oauthentication keys, etc. for logging into services or even possibly tokens for access to services such as Google accounts.

**(6) Closing**

Due to the nature of the project and our team, we will not be directly available for contact regarding future development of this application. This document was intended to explain decisions, application, and coding file structures to better understand how Cure is designed. Any additional questions one can be found at one of the following sites:

<https://guide.meteor.com/>

<https://www.npmjs.com/> (For finding NPM modules)

The public repository for all of Cure’s documentation and source code is at the following Github link:

<https://github.com/stevieclean/UNCG-CSE-Senior_Proj>