

Our Lady of Bethlehem Childcare Enrollment System Technical Documentation

April 2017

Version 2.0

Document Revisions

|  |  |  |
| --- | --- | --- |
| **Date** | **Version Number** | **Document Changes** |
| Jan 30, 2016 | 1.0 | Initial Draft |
| Feb 6, 2016 | 1.1 | Added collection details |
| April 24, 2017 | 2.0 | 2017 Project Improvements – Forecast, etc. |
|  |  |  |

## Technical Documentation

Code exists in github repository: https://github.com/niesmo/childcare

Code for 2017 project updates exists in github repository: https://github.com/ericpurvis/Daycare17

## Folder Structure

Every folder that was added contains more or less 3 main directories.

### Client

This directory will contain everything that will be sent to the client side. This could be all the templates, all the stylings, and all the client side javascript that help render the page.

Within client, there could be more hierarchy of directories, but they serve as organizers and for the developers to find things easier.

### Lib

In the lib directory, you will find all the files that are going to be available in both the client and server side of the application. This includes all the collection definitions (we used aldeed:collection2) and the routing file (we used iron:router).

### Server

In this directory, there are all the files that is only available to the client side of the application. This is where the logic of the application is buried. In this directory, there usually only exist two files. method.js which is where the backend logic for that section of the application reside and publications.js which is where the server knows what data from the MongoDB database to send to the MiniMongo on the client side. If no information is specified in the publications.js, the client side will have no data to access.

### 

### 

### .meteor

Contains all information regarding versions/config for meteor

### config/Server/pretty-email.config.js

configures the email nicely for when an application is sent out.

### cron/server

Cron job setup for creating action items based on different criterias

### errors/client

Small MiniMongo for keeping track of errors on the page and dynamically showing them.

In this directory there is a publications.js inside the client side code that is strictly created during the life cycle of a visit to the application. Once refreshed, the collection is cleared.

### global

client and server functionalities that are globally available. The client is used significantly more during the application.

### public/images

All images that will be publically available. The only image there currently is the logo.

### seed/server

These files are used to seed the database for mostly testing purposes.

### settings

this directory contains all the settings files for deployment and when you run. These are json files that you pass to the site when you run it. Then the information in the json files become available to access.

### shared/client

Contains layouts, navbars, and pages reachable from other client pages

### tests

Contains cucumber tests for functional testing

# Database

Meteor works using the MongoDB on the server side, and utilizing a MiniMongo on the client side, which caches a local copy. The database is broken into collections. In our file structure, each part of the application contains its own collections.js file in its lib folder, which creates a template for each available member of each collection. Each part also contains a publications.js in its server folder, which defines how/what the client side of the application can access.

Client side: Databases are subscribed to, using Meteor.subscribe(), in client side javascript code to give access to specific collections. Methods like <Name\_Of\_Collection>.find() and <Name\_Of\_Collection>.findOne() are used to find specific information in the corresponding collection.

Server side: Databases can be changed from server side code by using the Meteor.publish() call seen in publications.js and then making specific changes which can be seen in Methods.js. Methods like NameOfCollection.remove(element) and NameOfCollection.insert(element) can be used to alter the collection, returning a status code about the result of the function.

Further instructions on the area can be found in the Collections section of [Meteor Documentation](http://docs.meteor.com/).

Collections:

The following are all of the collections in the application, followed by their respective properties:

**Action Items**: Contains the action items from the main tasks page

|  |  |
| --- | --- |
| Title | String containing title of action item, not used as we ended up using description |
| Description | String containing description of the action item |
| Urgency | Optional string containing the priority level of the action item |
| Type | String defining if the action item is for the Toddler, Infant, or Misc section |
| CreatedBy | Optional Regex.Id of user who created action item |
| CreatedAt | Date value of when the action item was created |
| CompletedBy | Optional Regex.Id of user who completed action item |
| CompletedAt | Optional Date value of time when action item was completed |
| IsSystemMessage | Boolean for when action item is created by the system |

**Application**: Contains information of sent applications

|  |  |
| --- | --- |
| expirationDate | Date value of when application expires |
| sentAt | Date value of when application was sent |
| sentBy | Regex.Id of user who sent application |
| sentTo | Regex.Email of which email application was sent to |
| token | Regex.Id of specific application for tracking |
| submittedAt | Optional Date for when application was submitted |
| type | String value of which type of application (Regular, Member, Existing) |

**Classrooms**: Contains information about each classroom

|  |  |
| --- | --- |
| name | String containing name of the classroom |
| type | String containing type of classroom (Toddler/Infant) |
| capacity | Optional number of max number of students in each day |
| teacherId | Optional Regex.Id of teacher of classroom |
| createdAt | Date of when classroom was created |

**Parents:** Contains information about parents of children, this collection is linked to students collection

|  |  |
| --- | --- |
| firstName | String containing parents first name |
| middleName | Optional string containing parents middle name |
| lastName | String containing parents last name |
| dateOfBirth | Optional date of parents birthdate |
| address | String containing parents place of residence |
| image | Optional Regex.Url for avatar of parent |
| email | Regex.Email of parent |
| phoneNumber | String of number to be contacted |
| createdAt | Date of when parent element was created |
| isPrimary | Optional Boolean of if specific parent is primary contact |

**Students:** Contains information about each enrolled and waitlisted student

|  |  |
| --- | --- |
| day | String values of which days to be enrolled |
| flexible | Optional boolean of if student is flexible about days |
| firstName | String of childs first name |
| middleName | Optional string of childs middle name |
| lastName | String of childs last name |
| dateOfBirth | Optional date of childs birth date |
| conceived | Boolean of whether child is conceived or not |
| startDate | Date value of desired start date |
| moveDate | Optional date value of projected move date to next class |
| image | Optional Regex.Url of childs avatar image |
| group | String value of class type, Infant or Toddler |
| status | String type of current status: Waitlist, Application, Enrolled, Partial Enrolled |
| order | Optional Number of current spot on the waitlist |
| type | String value of applcation type: Regular, Member, Existing |
| paidApplicationFee | Boolean of whether application fee is paid or not |
| daysEnrolled | Optional Schemas.Days array of days that are enrolled |
| daysEnrolled | Optional Schemas.Days array of days waitlisted for |
| daysRequested | Schemas.Days array of days requested for enrollment |
| classId | Optional RegexId of associated classId |
| createdAt | Date value of when student element was created |
| color | String value indicatin associated color for classrooms table |
| details | Optional string of extra notes on the student |
| dueDate | Optional due date for unborn child |

# 

# 

# Deployment

To deploy the site, there are some steps that needs to be taken. Here is a list of those items:

1. Clone the repository from Github
2. Build the project on the server
3. Unzip the built version
4. npm install
5. Setup the settings and bring the scripts (may need some enhancements)
6. Start the project

## 1. Clone from repository

In order to clone the repository, you need to first log into the machine that the program is going to be hosted. You can SSH into the machine using the credentials provided. You will have to use the Root user for this step (probably). After you are logged in to the machine, change directory to /var/www/vhosts/ourladyofbethlehem.org/httpdocs/enrollment\_application. In this directory, there is a script -- getLatestFromGithub -- that clones the newest version of the code from Github repository (from a branch called 'Production') into a directory labeled with the date. You can simply run the script by running bash getLatestFromGithub. If this process goes smoothly, you will have a new directory that is named with the following format: %Y%m%d%H%M%S. For example, if the date is Jan 29, 2016 and it is 3:30:45pm, the directory name will be 20130129153045. This directory would contain all the code that is required to for the project.

## 2. Build

After you have successfully downloaded the project into the server, you can start the build process. You can do so by first changing directory into the newly downloaded directory (for example, the 20130129153045 directory) and then executing the following command. meteor build bin You can optionally use the --directory option so you can skip the unzipping part. if the build is complete, you will have a new directory named bin where you will have a zipped file (unless you use the --directory option) inside of it.

## 3. Unzipping the Build Directory (SKIP IF YOU USED --directory IN STEP 2)

When you have the built version of the code in a zipped format inside the bin directory, all you need to do is to unzip the file to be able to use it. You can use the following command to unzip the file: tar -zxvf 20130129153045.tar.gz (replace 20130129153045 with your file name) When execution of this command is done, you will have a new directory called bundle where the real executable code exists.

## 4. NPM Install

Now that we have the executable code, we need to install it. Since meteor will create its executable code versions as a node program, you can install them using the npm. First change directory into bundle/program/server and the execute npm install.

## 5. Setup the Settings.json and the Start Script

In this step, we are going to bring all the necessary files to be able to run the program on the server. One of those files is the settings file for meteor. This file is located in the downloaded code from github (located /var/www/vhosts/ourladyofbethlehem.org/httpdocs/enrollment\_application/<SOME\_DATE>/settings/production) and needs to be copied to in the bundle directory. You can do so using this command (executed from the bundle directory):  
cp ../../settings/production/settings.json . (There is a period at the end of the command indicating where to copy the file to). This command, will copy the settings.json into the bundle directory. You will also need another script called start that would help you start the program. The start script is located in the enrollment\_application directory (cp /var/www/vhosts/ourladyofbethlehem.org/httpdocs/enrollment\_application/start .)

## 6. Start the Site

It's finally time to start the application. We are currently using a service called forever to run the main.js script for a long period of time (hence the name). For future work, other options could be used such as MeteorUp (AKA MUP). The actual command is written in the start script. You can simply run the start script by executing bash start. This command, will start the main.js program on the port 3000 where we have previously setup a proxy server that would correctly reroute the application. This way the client, does not need to put the port number in the address bar to access the site (just because that would be ridiculous). Once that command is ran, you can check the site at www.ourladyofbethlehem.org/waitlistapp.

For testing purposes (on the server -- UAT testing), you can pass the port number to the start script and it will start the application on that port. It's worth noting that all the running sites (on different ports) will be sharing the same database (MongoDB). To check and see what all the forever is running, you can use the forever list command and see how many instances of the main.js is it running. You can stop particular processes running on forever using the forever stop <UID>

# Local Development for Future Development

This is the code that we used to run the application on our local machines when developing. You could simply run meteor run for a simple meteor application, but since we are sending emailing we need a smtp server, and some more options that are added to the run command.

(clear&&MAIL\_URL=smtp://admin@s1088540.instanturl.net:fwbJRScQ8gPFdwQJ@s1088540.instanturl.net:25/ ROOT\_URL=http://localhost:3000 meteor --production --settings settings/development/settings.json run)

You could use any of the email address and password as the smtp url.

# Email Account

1. Admin

email address: admin@s1088540.instanturl.net

password: fwbJRScQ8gPFdwQJ

2. Application

email address: application@ourladyofbethlehem.org

password: Sol314?u

3. Waitlist

email address: waitlist@ourladyofbethlehem.org

password: a4Ed\_Jm^!3#3QThN