LifeSense

Zach Toolson and Dan Willoughby

December 19, 2014

Contents

1 Introduction

Life is full of unexpected surprises. Some days you can come home and find you left the lights on wasting your hard earned cash on an electricity bill. Other days you might find that your dog is laying on your couch, even though you told it not to. So much for those dog treats you used when you thought it was being a good dog. Probably the worst unexpected surprise though, is when you have already poured your favorite cereal only to find that there is not any milk left in the fridge.

The problem that our project solves is providing a way for people to monitor data from their lives and interact with it in a meaningful way. In other words, it helps lessen the unexpectedness of those unexpected surprises. Our project provides a way for you to turn off your lights from anywhere, even after you have left your house (if your house is super small and lit with LEDs). You can detect when that dog jumps up on the couch and be notified via email when it does. You can track the amount of milk left in your fridge based off its weight and check its levels in real-time. Thats right, you can see how much milk is in your fridge in real-time! No more missing out on your favorite cereal.

On a more serious note, our project is a platform for managing sensors and controlling peripherals. Our project involves a printed circuit board (PCB) with wireless network capabilities that we designed, fabricated, and soldered. Since the PCB is wireless it can be placed anywhere, so that data can be gathered in anyplace without having to run a wire to a central control station. Our project provides a complete and functional website to manage multiple PCBs in one place. The website allows users to see a history of sensor data, view real-time updates, get email notifications, configure PCBs, and control peripherals, all from the comfort of a laptop or smartphone. Although its not quite ready to be stocked on Home Depots selves, its functionality is complete enough for any electronic hobbyists that is interested in managing sensors and controlling peripherals via wifi.

2 Background

The background of our project comes from Dan wanting to keep track how much milk he had left in the milk carton in his fridge, and receive an email when the milk was almost gone. We both thought it would be a pretty cool thing to have so we started work on the project in our Embedded Systems class. After we built a prototype in Embedded Systems, we decided to continue working on it for our senior project. We realized that we could expand on the simple prototype we made for simply tracking milk, into a project that would work with any sensor and could be used from any device with a web browser.

Another huge contributor to the decision on our project was we wanted to make something useful. We actually wanted to use the project that we were making, if not to track milk, but to track motion, temperature, or weight. We laid the foundation to work on something we were passionate about and set a goal for the project to have an awesome demo. From the very beginning of the project, we envisioned people on demo day sincerely saying "Wow! Thats cool!", instead of people purposely lying and saying "Ohh... looks great guys."

Planning began the beginning of 2014 and continued for about 4 months. Although we had a general idea of what we wanted to do, the project really evolved over the summer and in the

fall when we started implementing our ideas. We adjusted our plans and even made new ones.

3 Project Implementation

3.1 Printed Circuit Board

Our printed circuit board (PCB) handled most of the input from the sensors and controlled outputs such as LEDs. The PCB needed to have a way to communicate wirelessly and have a powerful enough microcontroller in order to interface with the network processor. We decided to use the Cortex M4 microcontroller and the CC3000 network processor. The main reason for the Cortex M4 was because we had previous experience with it and it was a relatively popular chip. We also owned a Stellaris Launchpad from TI which featured the Cortex M4. We decided to use the CC3000 in our Embedded Systems class. During the class, we made a prototype PCB which just had the CC3000 soldered onto it. We connected the CC3000 with several wires to the Stellaris Launchpad and we were surprised, but relieved when it was fetching data from a website. In terms of first motivating successes, fetching data for the first time with the prototype was one of the firsts. The remainder of our time in Embedded Systems we worked on creating a prototype that was able to collect data from a scale and display it on a very simple website.

The actual creation of the PCB for our senior project was never really planned nor even thought possible. During the spring semester, we never officially decided to fabricate a PCB, but it all started when we found a schematic of the stellaris launchpad and started looking at the data sheet for the CC3000. After some time of studying the schematic, it seemed more of a possibility, but never something wed actually try. We heard from a friend that Kicad was a cool open source tool that was pretty similar to Altium. Since we had fabricated a PCB in Embedded Systems and thought it was cool that we made a board that worked in that class, we decided to learn how Kicad worked. We followed a youtube video series that taught the basics of Kicad and for awhile thats as far as we thought itd go.

One day, we just started looking up footprints on the internet for the parts of the prototype. After finding the footprints for the Cortex M4 and CC3000, as a joke we decided to place both the footprints in a project, just to see how insane it would be to connect the 64 pin microcontroller to a 30 or so pin network processor. Progress on the PCB continued in this way for awhile, just for fun wed add some more components like capacitors or resistors and joke around saying, man wed be nuts to actually connect all these footprints and pins together with traces. Shortly after the fall semester started, we just decided to do it. We said to ourselves, well if the PCB does not work we can just use the prototype, what do we have to lose? We spent about a week finalizing the design (connecting each of the 64 pins on the microcontroller, which took forever), creating some of our own footprints, and determined which parts wed need to order.

The first run of the board did not work completely, but it was successful in some ways. We figured out that our micro-usb footprint was too big and that we had wired up our LEDs backwards. If that had been our only mistake the board might have been usable, but we forgot to connect 4 pins to each other on the CC3000 so we had no networking abilities at all. After we discovered those issues we quickly fixed them and order the second board. The second order

or boards in roughly two weeks. At this time we calculated if the board did not work, we had roughly 2 or 3 more tries to get it to work before we were out of time. We were hopeful it would work, but still in the back of our minds we thought how could it possibly work, its such a complicated board. After it was soldered, we loaded on some sample code and the board worked (details later about what worked means)! It was an awesome feeling. Looking back we were really happy with the decision to make the board. It really enhanced the project and gave it the cool factor.

After we had the boards complete we were able to focus more on the software side of the project. We named the Embedded devices, transmitters. The name was fitting and allowed for the details of the embedded device to be abstracted away from the user and allow it to be configured more easily on the website.

3.2 Ruby on Rails

Ruby on Rails served as the backend of project that kept track of our data and served up our website. Rails is a modern web framework used to create dynamic web pages that is open source and written in the Ruby language. We were both familiar with Rails and we both wanted to get better at developing applications with it. It seemed like the logical choice for our website. Even though Rails did not have anything that was visual for the user to see (such as a GUI), it was extremely important to the success of our project.

Rails made it easy to keep track of all the dynamic information and data. The Rails framework abstracts the tricky details of a database (such as migrations, tables, and columns) and maps database entries to objects. The bonus was that we could create objects by receiving information from the user, manipulate the objects if needed, and then save the information into the database, all within a single line of code (and not just one really long line of code, in under 5 words).

We saved a lot of data in our database. Information about user accounts, sensor data that was collected, and information for the setup of transmitters and sensors. The user login information allowed our website to serve many different users. Each user could create transmitters and sensors and configure them so that they were associated with each other. For example, you could create a transmitter called LIfeSense which had about 20 pin numbers. The user could then create a sensor and assign it to the LifeSense transmitter and a pin number belonging to that transmitter. At a later time, if they moved the sensor to a different transmitter, all that was required was reassigning the transmitter and pin number on the website. The user did not have to reprogram their Embedded Device which is a feature we were quite proud of.

The dynamic functionality of switching sensors between transmitters came at a cost, however. We needed a way to determine which transmitter data was coming from. Our solution was to assign a token to each transmitter that the user would program into the PCB software. Each time the transmitter posted data to the website it would include its token and which pin number the data belonged to. From there we could do a reverse lookup to find the right sensor. To find the sensor, we first found the transmitter object based off its token. A transmitter could list all its pin numbers, so from there we could select the pin number and create a record of the data for the sensor.

The last piece of the puzzle was getting data from the Rails server to our website. We decided fairly early on in our project that we wanted to make an API so that all our data could

be updated, created, deleted, or read from JSON requests. JSON is a convention for specifying data structures in text, so for instance an array would be declared as an open and close bracket with commas separating values and a dictionary or hash would be similar, but use curly braces instead. The main reason for making an API was so we could make smartphone applications (if we had time), but it turned out it made it easier for using a rich user interface javascript framework.

We chose to use Ruby on Rails because it is optimized for programmers happiness and productivity. Rails uses a model, view, controller model which separates concerns and provides a very logical pattern of developing a web server. Rails also has a thriving community that allowed us to use gems, or modules that are written by other programmers very easily. One example of this is using the denkatu gem which was a simple library that provided calculator functionality to apply the sensors formula (which was usually a polynomial) to data coming in. Rails was very helpful in the storage of our data, but we found it lacking in its ability to create a responsive front-end user interface.

3.3 Ember

Ember was the front end of the project that was most of what the user saw, but it was not part of our original plan. We had absolutely no experience with Ember when we decided to use it, but we had a major dilemma. We had spent a month or so implementing a Rails website and all we had been able to create was a website that could track time. It was just overly frustrating to have Rails function as the user interface for dynamically changing content. Our time-tracker was slow and severely lacked functionality and the time tracker was not even a main feature of the project, it was just supposed to be something simple. At about the same time we finished the time tracker in Rails we heard about Ember. Ember.js is a javascript framework used for creating ambitious web applications and follows a lot of the same conventions and ideas as the Ruby on Rails framework. We made the decision to invest the time to learn a new framework, since we had already had some success with our PCB.

Learning Ember took way longer than expected. We spent several weeks following multiple tutorials. Some times we thought we had made the wrong decision switching, especially since we were already a month into development and time was running out.

Our decision paid off after a while as we started to get a hand of the frame work. It allowed us to have formulas to change sensors data on the fly as it displayed in graphs. It helped us in implementing a simple yet intuitive interface for sensors and transmitters. The best part was it provided a local mapping our server data, handling requests to Rails, caching results, and updates, all with little effort on our part. Ember allowed us to integrate with the highcharts graphing library for Javascript allowing our users to view live graphs, and detailed graphs of historical data that they could drag and zoom in on data points. We were able to add animated switches for users to flip on and off LEDs. We owe a lot of the simplistic nature of our website to Ember and a lot of the dynamic functionality that came with it.

3.4 Node

Node was used to tie everything together. We had the Rails server to store all our data, an Embedded Device to collect data, Ember to display and collect data from the user, but we

needed some way to get all these pieces together. We had a simple solution, that required the user to press refresh a lot to see updated data, or to wait quite a while for LEDs to turn on or off. We were at a point where we were almost decided to be content with the simple solution. It was then we thought back to our initial goal of our project. We wanted a sincere, "Wow! Thats cool" instead of "Ohh... looks great guys." So we pressed on with a more complicated solution that would allow for the communication between each piece of our project to be real-time. We just knew the real-time functionality would really give the wow factor we were looking for.

Node.js is a web development framework has great functionality for concurrent connections, so it seemed like the perfect fit for our live communication. We also had some previous experience with node when we made our prototype in the spring so that helped in making the decision. Our alternative was using some gems in Rails, but we quickly became frustrated by the options Rails had available.

The node server was pretty basic in concept but more difficult in implementation. Node kept a connection alive with each transmitter and continuously fetched data from them. We used the http protocol to send requests and responses. The transmitter read its sensor data and posted it to the node server. The node server would then respond in json with the values that the outputs should be. So for example, if pin A0 was an input and pin A1 was an output on the embedded device, every second the embedded device would read pin A0 and send in that value. Then node would tell the embedded device to set pin A1 to high in its response. We implemented the communication this way because it used existing functionality in both node and the embedded devices software libraries.

The other function of the node server was communicating live data to and from the Ember apps. The idea was to have the user see live data points on a graph as it was being read from a sensor, or be able to flip a switch on the web browser and have it turn off an LED instantly. In order to achive the functionality, we used a library called socket.io to keep a web socket open. We had some previous experience implementing sockets so it was relatively easy to implement once we got the some socket.io libraries working. After the sockets connected, the node server could send out the embedded device data based on its token. The web browser or Ember app could then parse the data for the correct graph (again based on the transmitters token) and display it to the user. The LEDs worked in a similar way, when a user flipped the switch, the web browser sent a message with the transmitter token and pin number to node, then node would tell the appropriate device to turn on/off that LED.

The live data part of our project was an unplanned, but an added bonus. We left it till the very end, because we knew it added additional complexity. We knew though, however, without real-time data, the project would be kind of lame, so we worked hard to make it functional with just a few days to spare before demo day.

4 Evaluation

4.1 Printed Circuit Board

We were very happy with our PCB and felt very lucky that it worked. In simplest terms, the PCB did everything we wanted it to. It connected the WiFi, it collected data from sensors, it controlled peripherals, and it was able to be controlled over the WiFi connection. The main

tests we did on the PCB was to see how long a connection could stay alive. We found that if the PCB acted as its own web server, the connection would die quickly, but if it was constantly posting data and receiving responses the connection would last a long time. Our longest period of posting and receiving responses was 13 hours and even then we manually stopped the board to move it.

We had a PCB stop working the day before the demo. Luckily, we had a spare board so we were able to replace it. It was pretty scarey when the board would not connect to WiFi, it was as if our CC3000 completely died. Another thing we noticed was some of the GPIOs did not work very well, but since we had wired up some many of them initially we had plenty left over for our demo.

4.2 Ruby on Rails

Ruby on Rails worked great as a web server that would communicate between the front end Ember.js application and our database. Some of the difficulties that we had using Ruby on Rails were getting the emails to be sent from our local machines. Many web tutorials all pointed to the same way of sending emails, but didnt work. What was required is installing a SMTP server on our local machines to send the email for Ruby on Rails. Another thing that was questionable is that our Ruby on Rails formula evaluating library would evaluate the values slightly differently than the javascript library that we were using on the front end, which caused some irregularities in the email when checking if a value is above or below a certain value. Our testing strategy was mostly using the command line tool curl to verify that the proper data was being served up in JSON format and records, sensors, and transmitters would be saved properly.

4.3 Ember

Ember was a love hate relationship. Even though it provided great tools for fetching and receiving data from our Rails backend, we just had constant headaches when implementing things that were not natively supported. One frustration was updating relationships between pin numbers and sensors. Ember did not provide a way to detect when a relationship had changed, so we had to manually do it ourselves. Another thing was applying the user token in the HTTP headers so that our Rails server would see our JSON request as authenticated. We eventually came up with some hacked together solutions, but each hacked solution cost us tons of time and effort.

Since a majority of our Ember code was for User interface, we simply clicked around on the website to determine the functionality we expect was there.

4.4 Node

Node was perhaps the fastest development for our project. Every aspect went fast and smooth. Node provided all the tools we needed to communicate our real-time data between our Ember app, embedded devices, and our Rails backend. To the credit of all the other frameworks we used, Node had the simplest job to do, so that contributed to the good experience we had.

5 Conclusion

The goal of our final project was to learn modern web frameworks that interact with hardware, and to create a product that people will find useful. Throughout our senior project display, we received lots of feedback on ways that people wanted to use our product which proved our success on creating a product that people are interested in. The other goal of our senior project was to build a modern website with many different frameworks, which we accomplished as well. Our final project can be found at lifesense.herokuapp.com where you can sign up and start using our transmitters to track data.

A Tutorials

http://ember.vicramon.com/
https://www.codeschool.com/courses/warming-up-with-ember-js
https://www.railstutorial.org/book
http://webcloud.info/blog/2014/04/07/emberjs-authentication-the-right-way-javascript-ver

B Discoveries and pitfalls

PCB - start early and even if you think the first iteration of your board is perfect, it will most likely have issues so plan on it.

Learning new frameworks takes time in understanding the conventions (ember, rails, node).

Always read the documentation.

Email in rails locally requires a SMTP server running along with proper settings (found in config/environments/development)

C Ruby on Rails Source Code

C.1 Gemfile

```
source 'https://rubygems.org'
source 'https://rails-assets.org'


# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.1.6'
gem 'ember-rails'
gem 'ember-source'
gem 'emblem-rails' # contains active_model_serializers
gem 'socket.io-rails'
gem 'ffaker'
gem 'puma'
gem 'bcrypt',      '3.1.7'
gem 'clockwork'
gem 'curb'

# For debugging
gem 'pry'

# For evaluting formula's serverside
gem 'dentaku', '~> 1.2.1'

# For email
gem "letter_opener", :group => :development
```

```
gem 'figaro'

# Use SCSS for stylesheets
gem 'sass-rails', '~> 4.0.3'
gem 'bootstrap-sass',       '3.2.0.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.0.0'

# Use jquery as the JavaScript library
gem 'jquery-rails'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.0'
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0',          group: :doc

#Assets
gem 'rails-assets-jquery.cookie'
gem 'rails-assets-ember-list-view'
gem 'rails-assets-highcharts'
gem 'rails-assets-mathjs'
gem 'rails-assets-moment'
gem 'rails-assets-marked'
gem 'rails-assets-bootstrap-switch'

# Spring speeds up development by keeping your application running in the background.
group :development do
  gem 'sqlite3'
  gem 'spring'
end

group :test do
  gem 'assert_json'
end


group :production do
  gem 'pg',             '0.17.1'
  gem 'rails_12factor', '0.0.2'
  gem 'unicorn',        '4.8.3'
end


# Use ActiveModel has_secure_password
```

```
# gem 'bcrypt', '~> 3.1.7'

# Use unicorn as the app server
# gem 'unicorn'

# Use Capistrano for deployment
# gem 'capistrano-rails', group: :development

# Use debugger
# gem 'debugger', group: [:development, :test]
```

C.2 app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception


  def authenticate
    # halts with 401
    authenticate_or_request_with_http_token do |token, options|
      @current_user = User.authenticate_with_token(token)
      return @current_user if @current_user && token == @current_user.token
    end
  end

  def authenticate_token
    #returns a boolean
    authenticate_with_http_token do |token, options|
      @current_user = User.authenticate_with_token(token)
      token == @current_user.token if @current_user
    end
  end


end
```

C.3 app/controllers/api/v2/frequency_periods_controller.rb

```
class Api::V2::FrequencyPeriodsController < ApplicationController
  respond_to :json
  before_action :authenticate

  def index
```

```
    respond_with FrequencyPeriod.all
  end

  def show
    respond_with frequency_period
  end

  def create
    respond_with :api, status: :created, json: FrequencyPeriod.create(frequency_period_p
  end

        def update
                respond_with frequency_period.update(frequency_period_params)
        end

  def destroy
    respond_with frequency_period.destroy
  end

  private

  def frequency_period
    FrequencyPeriod.find(params[:id])
  end

  def frequency_period_params
    params.require(:frequency_period).permit(:id, :name)
  end
end
```

C.4 app/controllers/api/v2/posts_controller.rb

```
class Api::V2::PostsController < ApplicationController
  respond_to :json
  before_action :authenticate, only: [:create, :update, :destroy]

  def index
    respond_with Post.all
  end

  def show
    respond_with post
  end

  def create
```

```
    p = post_params
    tag = p["tag"]
    tag = p["tag"]["id"] if !tag.nil?
    respond_with :api, status: :created, json: Post.create(title: p["title"], content: p
  end

      def update
    p = post_params
    tag = p["tag"]
    tag = p["tag"]["id"] if !tag.nil?
            respond_with post.update(title: p["title"], content: p["content"], tag_i
      end

  def destroy
    respond_with post.destroy
  end

  private

  def post
    Post.find(params[:id])
  end

  def post_params
    params.require(:post).permit(:id, :content, :title, tag: [:id])
  end


end
```

C.5 app/controllers/api/v2/records_controller.rb

```
class Api::V2::RecordsController < ApplicationController
  respond_to :json
  before_action :authenticate, except: [:create]
  skip_before_filter :verify_authenticity_token

  def show
    respond_with record
  end

  def create
    # TODO send error messages
    puts "********************************************"
    puts record_params
```

```ruby
transmitter = Transmitter.find_by(transmitter_token: record_params["transmitter_toke
user = User.where(id: transmitter.user_id).first

pin_number = transmitter.pin_numbers.find_by(name: record_params["pin_number"])
sensor = pin_number.sensor if pin_number
puts sensor

if sensor
  @record = sensor.records.build(x: DateTime.now.to_i * 1000, y: record_params["y"])
  if @record.save

    # only send email if lower is a valid number
    if sensor.lower && (sensor.lower.match(/\A[+-]?\d+?(\.\d+)?\Z/) == nil ? false :
      # evaluate sensor formula
      calculator = Dentaku::Calculator.new
      if sensor.formula
        formula_value = calculator.evaluate(sensor.formula, x: @record.y)
      else
        formula_value = calculator.evaluate("x", x: @record.y)
      end

      # Send email if formula_value is below sensor lower value
      if formula_value && formula_value < sensor.lower.to_f
        RecordMailer.send_record(user, @record, sensor).deliver
      end
    end

    # only send email if upper is a valid number
    if sensor.upper && (sensor.upper.match(/\A[+-]?\d+?(\.\d+)?\Z/) == nil ? false :
      # evaluate sensor formula
      calculator = Dentaku::Calculator.new
      if sensor.formula
        formula_value = calculator.evaluate(sensor.formula, x: @record.y)
      else
        formula_value = calculator.evaluate("x", x: @record.y)
      end

      # Send email if formula_value is above sensor upper value
      if formula_value && formula_value > sensor.upper.to_f
        RecordMailer.send_record(user, @record, sensor).deliver
      end
    end
  end
end
```

```
    respond_with :api, status: :created, json: @record
  end

  def update
    respond_with record.update(transmitter_params)
  end

  def destroy
    respond_with record.destroy
  end

  private

  def record
    Record.find(params[:id])
  end

  def record_params
    # make additional required params
    params.require(:record).require(:transmitter_token)
    params.require(:record).require(:pin_number)
    # setup for hash structure
    params.require(:record).permit(:y, :x, :transmitter_token, :pin_number)
  end

end
```

C.6 app/controllers/api/v2/sensors_controller.rb

```
class Api::V2::SensorsController < ApplicationController
  respond_to :json
  before_action :authenticate
  skip_before_filter :verify_authenticity_token

  def index
    respond_with @current_user.sensors
  end

  def show
    respond_with sensor
  end

  def create
    # TODO send error messages
    @sensor = @current_user.sensors.build(sensor_params)
```

```
  @sensor.save

  respond_with :api, status: :created, json: @sensor
end

def update
  param = sensor_params
  pin = param["pin_number"]
  pin = param["pin_number"]["id"] if !pin.nil?
  cwde = param["clockwork_database_event"]
  puts "*****************************88"
  puts param
  update_clockwork_database_event(cwde)

  respond_with sensor.update_attributes(name: param["name"],
                                        pin_number_id: pin,
                                        formula: param["formula"],
                                        lower: param["lower"],
                                        upper: param["upper"],
                                        led: param["led"])
end

def destroy
  respond_with sensor.destroy
end

private

def sensor
  @current_user.sensors.find(params[:id]) if @current_user
end

def sensor_params

  params.require(:sensor).permit(:id,
                                 :name,
                                 :formula,
                                 { clockwork_database_event:[:at, :frequency_quantity,
                                 :led,
                                 :lower,
                                 :upper,
                                 :user_id,
                                 pin_number: [:id])
end
```

```
  def update_clockwork_database_event(cw_params)
    if cw_params
      cw = ClockworkDatabaseEvent.find_by(cw_params["id"])
      puts cw_params["frequency_period"]
      puts cw_params["frequency_period"]["name"]
      fp = FrequencyPeriod.find_by(name: cw_params["frequency_period"]["name"])
      puts fp
      cw.update_attributes(frequency_quantity: cw_params["frequency_quantity"], frequenc

      puts cw
    end
  end
end
```

C.7 app/controllers/api/v2/tags_controller.rb

```
class Api::V2::TagsController < ApplicationController
  respond_to :json
  before_action :authenticate

  def index
    respond_with Tag.all
  end

  def show
    respond_with tag
  end

  def create
    respond_with :api, status: :created, json: Tag.create(tag_params)
  end

        def update
                respond_with tag.update(tag_params)
        end

  def destroy
    respond_with tag.destroy
  end

  private

  def tag
    Tag.find(params[:id])
  end
```

```
  def tag_params
    params.require(:tag).permit(:name)
  end


end
```

C.8 app/controllers/api/v2/transmitters_controller.rb

```
class Api::V2::TransmittersController < ApplicationController
  respond_to :json
  before_action :authenticate
  skip_before_filter :verify_authenticity_token

  def index
    puts params
    respond_with @current_user.transmitters
  end

  def show
    respond_with transmitter
  end

  def create
    # TODO send error messages
    @transmitter = @current_user.transmitters.build(name: transmitter_params["name"], tr
    @transmitter.save
    [:A0, :A1, :A2, :A4, :A5, :A6, :A7, :PF_1, :PF_2, :PF_3, :PB_3, :PC_4, :PC_5, :PC_6,
      @transmitter.pin_numbers.create(name: pin)
    end
    @transmitter.save

    respond_with :api, status: :created, json: @transmitter
  end

  def update
    respond_with transmitter.update(transmitter_params)
  end

  def destroy
    respond_with transmitter.destroy
  end

  private
```

```
  def transmitter
    @current_user.transmitters.find(params[:id]) if @current_user
  end

  def transmitter_params
    params.require(:transmitter).permit(:name)
  end


end
```

C.9 app/controllers/api/v2/users_controller.rb

```
class Api::V2::UsersController < ApplicationController
  respond_to :json
  before_action :authenticate, except: [:login, :create]
  before_action :admin_user, only: :destroy
  skip_before_filter :verify_authenticity_token



  def login
    @current_user = User.find_by(email: params[:email].downcase) if params[:email]
    if @current_user && @current_user.authenticate(params[:password]) || authenticate_to
      @current_user.generate_token if @current_user
      render status: :ok, json: @current_user
    else
      render status: :unauthorized, json: 'Invalid username/password'
    end
  end

  def current_user
    @current_user
  end

  def index
    respond_with User.all
  end

  def show
    respond_with user
  end

  def create
```

```
    @current_user = User.create(user_params)
    respond_with :api, @current_user
  end

  def update
    respond_with user.update(user_params)
  end

  def destroy
    respond_with user.destroy
  end

  private

  def user
    User.find(params[:id])
  end

  def user_params
    params.require(:user).permit(:name, :email, :password, :password_confirmation)
  end

  def admin_user
    render status: :unauthorized, json: 'Not authorized to do that!' unless @current_use
  end
end
```

C.10 app/constraints/api_constraint.rb

```
class ApiConstraint
  attr_reader :version

  def initialize(options)
    @version = options[:version]
    @default = options[:default]
  end

  def matches?(request)
    @default || request.headers[:accept].include?("version=#{version}")
  end
end
```

C.11 app/mailers/record_mailer.rb

```
class RecordMailer < ActionMailer::Base
#  default from: "data@lifesense.com"
  default :from => 'no-reply@lifesense.herokuapp.com'

  def send_record(user, record, sensor)
    @user = user
    @record = record
    @sensor = sensor

    mail(to: @user.email, subject: 'LifeSense Data Record')
  end

  def hello_world(user)
    @user = user

    mail(to: @user.email, subject: 'LifeSense Data Record')
  end
end
```

C.12 app/models/clockwork_database_event.rb

```
class ClockworkDatabaseEvent < ActiveRecord::Base
  belongs_to :frequency_period
  has_one :sensor

  # Used by clockwork to schedule how frequently this event should be run
  # Should be the intended number of seconds between executions
  def frequency
    frequency_quantity.send(frequency_period.name.pluralize)
  end

  def at
    return nil
  end

  def do_some_work
    c = Curl::Easy.perform("http://energia.nu/hello.html")
    c.perform
    puts c.body_str
  end
end
```

C.13 app/models/frequency_period.rb

```
class FrequencyPeriod < ActiveRecord::Base
end
```

C.14 app/models/pin_number.rb

```
class PinNumber < ActiveRecord::Base
  belongs_to :transmitter
  has_one :sensor
end
```

C.15 app/models/post.rb

```
class Post < ActiveRecord::Base
  belongs_to :tag
  default_scope -> { order('created_at DESC') }
end
```

C.16 app/models/record.rb

```
class Record < ActiveRecord::Base
  belongs_to :sensor
        validates :sensor_id, presence: true
  validates :x, presence: true
  validates :y, presence: true
end
```

C.17 app/models/sensor.rb

```
class Sensor < ActiveRecord::Base
  belongs_to :user
  belongs_to :pin_number
  belongs_to :clockwork_database_event
  accepts_nested_attributes_for :pin_number
  accepts_nested_attributes_for :clockwork_database_event
  has_many :records
  validates :user_id, presence: true
  validates :name, presence: true, length: {maximum: 40 }
end
```

C.18 app/models/tag.rb

```
class Tag < ActiveRecord::Base
  has_many :posts
end
```

C.19 app/models/transmitter.rb

```
class Transmitter < ActiveRecord::Base
  belongs_to :user
        has_many :pin_numbers, dependent: :destroy
  validates :user_id, presence: true
  validates :name, presence: true, length: {maximum: 40 }

  def Transmitter.new_token
    SecureRandom.urlsafe_base64
  end
end
```

C.20 app/models/user.rb

```
class User < ActiveRecord::Base
  has_many :transmitters, dependent: :destroy
        has_many :sensors, dependent: :destroy
  before_save { self.email = email.downcase }
  validates :name,  presence: true, length: { maximum: 50 }
  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i
  validates :email, presence: true, format: { with: VALID_EMAIL_REGEX },
            uniqueness: { case_sensitive: false}

  has_secure_password

  # Returns a random token.
  def User.new_token
    SecureRandom.urlsafe_base64
  end

  def User.authenticate_with_token(token)
    User.find_by_token(token)
  end

  def generate_token
    return if self.token.present?
    update_attribute(:token, User.new_token)
  end
end
```

C.21 app/serializers/clockwork_database_event_serializer.rb

```
class ClockworkDatabaseEventSerializer < ActiveModel::Serializer
  attributes :id, :at, :frequency_quantity
```

```
  has_one :frequency_period
end
```

C.22 app/serializers/frequency_period_serializer.rb

```
class FrequencyPeriodSerializer < ActiveModel::Serializer
  attributes :id, :name
end
```

C.23 app/serializers/pin_number_serializer.rb

```
class PinNumberSerializer < ActiveModel::Serializer
  attributes :id, :name
end
```

C.24 app/serializers/post_serializer.rb

```
class PostSerializer < ActiveModel::Serializer
  attributes :id, :title, :content, :updated_at

  has_one :tag
end
```

C.25 app/serializers/record_serializer.rb

```
class RecordSerializer < ActiveModel::Serializer
  attributes :id, :x, :y, :sensor_id
end
```

C.26 app/serializers/sensor_serializer.rb

```
class SensorSerializer < ActiveModel::Serializer
  attributes :id, :name, :formula, :led, :lower, :upper

  has_one :pin_number
  has_one :clockwork_database_event
  has_many :records
end
```

C.27 app/serializers/tag_serializer.rb

```
class TagSerializer < ActiveModel::Serializer
  attributes :id, :name
end
```

C.28 app/serializers/transmitter_serializer.rb

```
class TransmitterSerializer < ActiveModel::Serializer
  attributes :id, :name, :transmitter_token

  has_many :pin_numbers


end
```

C.29 app/serializers/user_serializer.rb

```
class UserSerializer < ActiveModel::Serializer
  attributes :id, :name, :email, :admin #, :links

  has_many :transmitters
  has_many :sensors

  def attributes
    data = super
    # scope defaults to current_user in UsersController
    data[:token] = object.token if scope.id == object.id
    data
  end

  #def links
    #{ transmitters: "/api/users/#{object.id}/transmitters" }
  #end
end
```

C.30 config/routes.rb

```
Rails.application.routes.draw do
  root to: 'home#index'

  namespace :api, format:false,  defaults: {format: 'json'} do
    scope module: :v1, constraints: ApiConstraint.new(version: 1) do
      resources :users

    end
    scope module: :v2, constraints: ApiConstraint.new(version: 2, default: :true) do
      resources :users
      resources :transmitters
      resources :sensors
      resources :records, except: [:index]
```

```
      resources :posts
      resources :tags
      resources :frequency_periods
      post 'login'  => 'users#login'
      post 'signup'  => 'users#create'
    end
  end


  # Catch-all Rails route for arbitrary Ember routes
  get '*path', to: 'home#index'
end
```

C.31 config/routes.rb

```
Rails.application.routes.draw do
  root to: 'home#index'

  namespace :api, format:false,  defaults: {format: 'json'} do
    scope module: :v1, constraints: ApiConstraint.new(version: 1) do
      resources :users

    end
    scope module: :v2, constraints: ApiConstraint.new(version: 2, default: :true) do
      resources :users
      resources :transmitters
      resources :sensors
      resources :records, except: [:index]
      resources :posts
      resources :tags
      resources :frequency_periods
      post 'login'  => 'users#login'
      post 'signup'  => 'users#create'
    end
  end


  # Catch-all Rails route for arbitrary Ember routes
  get '*path', to: 'home#index'
end
```

C.32 config/environment.rb

```
# Load the Rails application.
require File.expand_path('../application', __FILE__)
```

```
# Initialize the Rails application.
Rails.application.initialize!

ActionMailer::Base.smtp_settings = {
  :address        => 'smtp.sendgrid.net',
  :port           => '587',
  :authentication => :plain,
  :user_name      => ENV['SENDGRID_USERNAME'],
  :password       => ENV['SENDGRID_PASSWORD'],
  :domain         => 'heroku.com',
  :enable_starttls_auto => true
}
```

C.33 db/schema.rb

```
# encoding: UTF-8
# This file is auto-generated from the current state of the database. Instead
# of editing this file, please use the migrations feature of Active Record to
# incrementally modify your database, and then regenerate this schema definition.
#
# Note that this schema.rb definition is the authoritative source for your
# database schema. If you need to create the application database on another
# system, you should be using db:schema:load, not running all the migrations
# from scratch. The latter is a flawed and unsustainable approach (the more migrations
# you'll amass, the slower it'll run and the greater likelihood for issues).
#
# It's strongly recommended that you check this file into your version control system.

ActiveRecord::Schema.define(version: 20141212043054) do

  create_table "clockwork_database_events", force: true do |t|
    t.integer  "frequency_quantity"
    t.integer  "frequency_period_id"
    t.string   "at"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "clockwork_database_events", ["frequency_period_id"], name: "index_clockwork

  create_table "frequency_periods", force: true do |t|
    t.string   "name"
    t.datetime "created_at"
    t.datetime "updated_at"
```

```
end

create_table "pin_numbers", force: true do |t|
  t.string   "name"
  t.integer  "transmitter_id"
  t.datetime "created_at"
  t.datetime "updated_at"
end

add_index "pin_numbers", ["transmitter_id"], name: "index_pin_numbers_on_transmitter_i

create_table "posts", force: true do |t|
  t.string   "title"
  t.text     "content"
  t.integer  "tag_id"
  t.datetime "created_at"
  t.datetime "updated_at"
end

add_index "posts", ["tag_id"], name: "index_posts_on_tag_id"

create_table "records", force: true do |t|
  t.integer  "x"
  t.float    "y"
  t.integer  "sensor_id"
  t.datetime "created_at"
  t.datetime "updated_at"
end

add_index "records", ["sensor_id", "x"], name: "index_records_on_sensor_id_and_x"
add_index "records", ["sensor_id"], name: "index_records_on_sensor_id"

create_table "sensors", force: true do |t|
  t.string   "name"
  t.string   "formula"
  t.integer  "user_id"
  t.datetime "created_at"
  t.datetime "updated_at"
  t.integer  "pin_number_id"
  t.string   "lower"
  t.boolean  "led",                          default: false
  t.integer  "clockwork_database_event_id"
  t.string   "upper"
end
```

```
  add_index "sensors", ["clockwork_database_event_id"], name: "index_sensors_on_clockwor
  add_index "sensors", ["pin_number_id"], name: "index_sensors_on_pin_number_id"
  add_index "sensors", ["user_id"], name: "index_sensors_on_user_id"
  add_index "sensors", ["user_id"], name: "index_sensors_on_user_id_and_transmitter_id"

  create_table "tags", force: true do |t|
    t.string   "name"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "transmitters", force: true do |t|
    t.string   "name"
    t.string   "transmitter_token"
    t.integer  "user_id"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "transmitters", ["user_id", "transmitter_token"], name: "index_transmitters_
  add_index "transmitters", ["user_id"], name: "index_transmitters_on_user_id"

  create_table "users", force: true do |t|
    t.string   "name"
    t.string   "email"
    t.datetime "created_at"
    t.datetime "updated_at"
    t.string   "password_digest"
    t.string   "token"
    t.boolean  "admin",           default: false
  end

  add_index "users", ["email"], name: "index_users_on_email", unique: true
  add_index "users", ["token"], name: "index_users_on_token"

end
```

C.34 db/seeds.rb

```
#This file should contain all the record creation needed to seed the database with its
# The data can then be loaded with the rake db:seed (or created alongside the db with
#
# Examples:
#
#   cities = City.create([{ name: 'Chicago' }, { name: 'Copenhagen' }])
```

```ruby
#   Mayor.create(name: 'Emanuel', city: cities.first)

[:second, :minute, :hour, :day, :week, :month].each do |period|
    FrequencyPeriod.create(name: period)
end

User.create!(name:  "dan",
             email: "dan@dan.com",
             password:              "dandan",
             password_confirmation: "dandan",
             admin: true,
             token: User.new_token)

User.create!(name:  "zach",
             email: "zach.toolson@gmail.com",
             password: "zach",
             password_confirmation: "zach",
             admin: true,
             token: User.new_token)

20.times do |n|
  name  = Faker::Name.name
  email = "example-#{n+1}@lifesense.com"
  password = "password"
  token = User.new_token
  User.create!(name:  name,
               email: email,
               token: token,
               password:              password,
               password_confirmation: password)
end

users = User.order(:created_at).take(6)
5.times do
    users.each do |user|
      name = Faker::Company.name
      token = Transmitter.new_token
      trans = user.transmitters.create!(name: name, transmitter_token: token)
      [:A0, :A1, :A2, :A4, :A5, :A6, :A7, :PF_1, :PF_2, :PF_3, :PB_3, :PC_4, :PC_5, :PC_
        trans.pin_numbers.create!(name: pin)
      end
    end
end

5.times do
```

```
    users.each do |user|
      name = Faker::Color.name
      formula = Faker::Education.degree
      lower_bound = rand(3000)
      upper_bound = rand(3000)
      user.sensors.create!(name: name, formula: formula, lower: lower_bound, upper: uppe
    end
end

day = 1
5.times do
    users.each do |user|
      user.sensors.each do |sensor|
        y = rand(3000)
        x = day.days.ago.to_i * 1000
        sensor.records.create!(time_stamp: time_stamp, value: value)
        day += 1
      end
    end
end
```

D Source code of Ember

D.1 js/application.js.coffee

```
#= require jquery
#= require jquery_ujs
#= require bootstrap
#= require bootstrap-switch
#= require handlebars
#= require ember
#= require ember-data
#= require highcharts
#= require mathjs
#= require moment
#= require socket.io
#= require ember-sockets
#= require jquery.cookie
#= require ember-list-view
#= require marked
#= require_self
#= require app

# for more details see: http://emberjs.com/guides/application/
```

```
window.App = Ember.Application.create(
  rootElement: '#ember-app'

  Socket: EmberSockets.extend({
    #host: "lifesense-node.herokuapp.com"
    host: window.location.hostname
    port: 4033
    controllers: ['socket', 'sensors_sensor', 'user_live', 'transmitters_transmitter']
    autoConnect: true
  })
)

$.cookie.json = true


window.mobilecheck = ->
  check = false
  ((a, b) ->
    check = true  if /(android|bb\d+|meego).+mobile|avantgo|bada\/|blackberry|blazer|com
    return
  ) navigator.userAgent or navigator.vendor or window.opera
  check
```

D.2 js/components/bootstrap-switch-led.js.coffee

```
App.BootstrapSwitchLedComponent = Ember.Component.extend
  contentChanged: (->
    @$("[type='checkbox']").bootstrapSwitch('state', @get('isLED'), true)
  ).observes('isLED')

  setLeds: (->
    leds = @get('leds')
    pinName = @get('pinName')
    transToken = @transToken
    if leds[transToken]
      value = leds[transToken][pinName]
      @$("[type='checkbox']").bootstrapSwitch('state', value, true)
  ).observes('leds')


  afterRenderEvent: ->
    @$("[type='checkbox']").bootstrapSwitch('size', @get('size'))
    @$("[type='checkbox']").bootstrapSwitch('labelText', @get('labelText'))
    @$("[type='checkbox']").bootstrapSwitch('state', @get('isLED'))
```

```
    self = @
    @$("[type='checkbox']").on "switchChange.bootstrapSwitch", (e, data)->
      self.sendAction('action', data, self.get('pinName'), self.get('transToken'))
```

D.3 js/components/line-time-series-chart.js.coffee

```
App.LineTimeSeriesChartComponent = Ember.Component.extend
  tagName: 'div'
  classNames: ['highcharts']

  contentChanged: (->
    @rerender()
  ).observes('data', 'formula')

  didInsertElement:( ->
    Highcharts.setOptions({
      global: {
        useUTC: false
      }
    })
    @draw()
  )

  updateTitle: (->
    chart = $("##{@chartId}").highcharts()
    chart.setTitle({
      text: @get('title')
    }
    )).observes('title')

  draw: ->
    $("##{@chartId}").highcharts({
      chart: {
        type: 'spline',
        animation: Highcharts.svg,
        marginRight:10,
      },
      title: { text:  @get('title') }, #{ text: 'Non Linear Sample Data' }, # Possibly
      subtitle: {
        text: 'Click and drag the plot area to zoom in'
      },
      xAxis: {
        type: 'datetime',
        tickPixelInterval: 150
        #minRange: 10 * 24 * 3600000 # 10 TODO days could make smaller?
```

```
      #
    },
    yAxis: {
      title: {
        text: 'Data Values Units' # TODO make dynamic based on units
      },
    },
    legend: { enabled: false },
    series: [{
      name: 'Data Value', # TODO dynamically get property
      data: @data
    }]
  })
```

D.4 js/components/live-time-series-chart.js.coffee

```
App.LiveTimeSeriesChartComponent = Ember.Component.extend
  tagName: 'div'
#  calculated_value: null
  classNames: ['highcharts']
  sensor: Ember.computed.alias('pin.sensor')


  actions: {
    recordValue: ->
      transToken = @get('pin.transmitter.transmitter_token')
      pinName = @get('pin.name')
      @sendAction('recordValue', pinName, transToken)

    sensor: ->
      sensor = @get('pin.sensor')
      @sendAction('action', sensor)
  }

  didInsertElement:( ->
    Highcharts.setOptions({
      global: {
        useUTC: false
      }
    })
    @draw()
  )

  updateTitle: (->
    chart = $("##{@chartId}").highcharts()
```

```
    chart.setTitle({
      text: @get('title')
    }
    )).observes('title')

  loadData: (->
    dataTransToken = @get('sensorData.transmitter_token')
    transToken = @get('pin.transmitter.transmitter_token')
    if dataTransToken != transToken
      return

    chart = $("##{@chartId}").highcharts()
    if typeof chart == 'undefined'
      return
    series = chart.series[0]
    pin = @get('pin.name')
    sensorData = @get('sensorData')
    time = (new Date).getTime()
    value = sensorData[pin]
    scope = { x : value }
    calculated_value = null
    # apply the sensor formula
    try
      calculated_value = math.eval(@formula, scope)
    catch
      calculated_value = value

    series.addPoint([time, calculated_value],true, true)
  ).observes('sensorData')

  draw: ->
    $("##{@chartId}").highcharts({
      chart: {
        type: 'spline',
        animation: Highcharts.svg,
        marginRight:10,
      },
      title: { text:  @get('title') }, #{ text: 'Non Linear Sample Data' }, # Possibly
      subtitle: {
        text: @get('pin.transmitter.name')
      },
      xAxis: {
        type: 'datetime',
        tickPixelInterval: 150
        #minRange: 10 * 24 * 3600000 # 10 TODO days could make smaller?
```

```
      #
    },
    yAxis: {
      title: {
        text: 'Data Values Units' # TODO make dynamic based on units
      },
      min: 0
    },
    legend: { enabled: false },
    series: [{
      name: 'Data Value', # TODO dynamically get property
      data: (->
        data = []
        time = (new Date()).getTime()
        for i in [-19..0]
          data.push({
            x: ( time + i * 1000),
            y: 0
          })
        return data
      )()
    }]
  })
```

D.5 js/controllers/application.js.coffee

```
App.ApplicationController = Ember.Controller.extend
  needs: ['login'],

  isAuthenticated: (->
    !Ember.isEmpty(@get('controllers.login.token'))
  ).property('controllers.login.token')

  currentUser: (->
    @get('controllers.login.currentUser')
  ).property('controllers.login.currentUser')

  isAdmin: (->
    user = @get('controllers.login.currentUser')
    if !Ember.empty(user)
      return user.admin
    else
      return false

  ).property('controllers.login.currentUser')
```

D.6 js/controllers/home.js.coffee

```
App.HomeController = App.ApplicationController
```

D.7 js/controllers/login.js.coffee

```
App.LoginController = Ember.Controller.extend
  init: ->
    @_super()
    if Ember.$.cookie('access_token')
      Ember.$.ajaxSetup
        headers: { 'Authorization': 'Token token=' + Ember.$.cookie('access_token') }

  reset: ->
    @setProperties({
      email: null,
      password: null,
      errorMessage: null,
    })

  attemptedTransition: null

  resetToken: ->
    @set('token', null)
    @set('currentUser', null)
    Ember.$.ajaxSetup
      headers: { 'Authorization': 'Token token=' }

  token: Ember.$.cookie('access_token')
  currentUser: Ember.$.cookie('auth_user')

  tokenChanged: (->
    if Ember.isEmpty(@get('token'))
      Ember.$.removeCookie('access_token')
      Ember.$.removeCookie('auth_user')
    else
      Ember.$.cookie('access_token', @get('token'))
      Ember.$.cookie('auth_user', @get('currentUser'))
  ).observes('token', 'currentUser')


  actions:
    login: ->
      self = @
```

```
    data = @getProperties('email', 'password')

    self.set('errorMessage', null)

    Ember.$.post('/api/login', data).then((response)->
      Ember.$.ajaxSetup
        headers: { 'Authorization': 'Token token=' + response.user.token }
      console.log(response)
      self.set('token', response.user.token)
      # only save important stuff
      self.set('currentUser',
        {
          admin: response.user.admin,
          id: response.user.id
        })
      self.set('errorMessage', null)
      attemptedTransition = self.get('attemptedTransition')

      if (attemptedTransition)
        self.transitionToRoute(attemptedTransition.targetName)
        self.set('attemptedTransition', null)
      else
        self.transitionToRoute('user.live')

    , (value) ->
      self.set('errorMessage', value.responseText)
    )
```

D.8 js/controllers/posts.js.coffee

```
App.PostsController = Ember.ArrayController.extend
  needs: ['application']

  tags: null

  filterName: 'all'

  isAdmin: Ember.computed.alias('controllers.application.isAdmin')

  height: (->
    if window.mobilecheck()
      return 100
    else
      return 500
  )()
```

```
setupTags: ->
  tags = @store.all('tag')
  @set('tags', tags)

filteredContent: (->
  content = @get('content.content')
  filter = @get('filterName')

  if filter == 'all'
    return content

  content.filter (item)->
    item.get('tag.name') == filter

).property('content.isLoaded', 'content.@each', 'filterName')

isAll: (->
  if @get('filterName') == 'all'
    return 'active'
).property('filterName')

isSoftware: (->
  if @get('filterName') == 'Software'
    return 'active'
).property('filterName')

isHardware: (->
  if @get('filterName') == 'Hardware'
    return 'active'
).property('filterName')

isMeeting: (->
  if @get('filterName') == 'Meeting'
    return 'active'
).property('filterName')

isChallenges: (->
  if @get('filterName') == 'Challenges'
    return 'active'
).property('filterName')

actions:
  all: ->
    @set('filterName', 'all')
```

```
    software: ->
      @set('filterName', 'Software')

    hardware: ->
      @set('filterName', 'Hardware')

    meeting: ->
      @set('filterName', 'Meeting')

    challenges: ->
      @set('filterName', 'Challenges')
```

D.9 js/controllers/posts_new.js.coffee

```
App.PostsNewController = Ember.Controller.extend
  title: null
  content: null
  tag: null
  tags: null

  setupTags: ->
    @store.find('tag').then (tags)=>
      console.log(tags.content)
      @set('tags', tags.content)

  actions:
    createPost: ->
      data = @getProperties(
        'title',
        'content',
        'tag'
      )
      self = @
      post = @store.createRecord('post', data)

      post.save().then =>
        @setProperties(
          title: null
          content: null
          tag: null
        )
        self.transitionToRoute "posts.post", post
```

D.10 js/controllers/posts_post_edit.js.coffee

```
App.PostsPostEditController = Ember.Controller.extend
  actions:
    saveChanges: ->
      @get('model').save().then =>
        @transitionToRoute 'posts.post', @get('model')

    cancel: ->
      @get('model').rollback()
      @transitionToRoute 'posts.post', @get('model')
```

D.11 js/controllers/posts_post.js.coffee

```
App.PostsPostController = Ember.Controller.extend
  needs: ['application']

  isAdmin: Ember.computed.alias('controllers.application.isAdmin')

  isEditing: false

  actions:
    delete: ->
      @get('model').destroyRecord().then =>
        @transitionToRoute 'posts'
```

D.12 js/controllers/sensors.js.coffee

```
App.SensorsController = Ember.ArrayController.extend
  sortProperties: ['pin_number.transmitter.name', 'name']

  height: (->
    if window.mobilecheck()
      return 200
    else
      return 700
  )()
```

D.13 js/controllers/sensors_new.js.coffee

```
App.SensorsNewController = Ember.Controller.extend
  needs: ['login']

  actions:
    createSensor: ->
```

```
    data = @getProperties(
      'name',
      'formula'
      'pin_number'
    )
    user = @get('controllers.login').get('currentUser')
    sensor = @store.createRecord 'sensor', data

    self = @
    @store.find('user', user.id).then (user)=>
      sensor.set('user', user)

    sensor.save().then =>
      @setProperties
        name: null,
      self.transitionToRoute "sensors.sensor", sensor
```

D.14 js/controllers/sensors_sensor.js.coffee

```
App.SensorsSensorController = Ember.Controller.extend
  needs: ['application']

  theFormula: null
  data: null
  frequencyPeriods: null
  transmitters: null
  transmitter: null
  pinNumbers: null
  pinNumber: null
  record: null



  isLED: Ember.computed.alias('model.led')

  # emberdata doesn't track dirt on relationships
  isRelationDirty: false

  getFrequencyPeriods: ->
    @store.find('frequency_period').then (frequencyPeriods)=>
      console.log frequencyPeriods
      @set('frequencyPeriods', frequencyPeriods.content)

  getTransmitters: ->
    userid = @get('controllers.application.currentUser').id
```

```
    @set('transmitter', null)

    @store.find('transmitter', { user_id: userid }).then (transmitters)=>
      @set('transmitters', transmitters)

    pin_number = @get("model.pin_number")
    if (!Ember.empty(pin_number) && !Ember.empty(pin_number.get('transmitter')))
      @set('transmitter', pin_number.get('transmitter'))

    @set('isRelationDirty', false) # still dirty since it was just set from the model

  getPinNumbers: ->
    transmitter = @get('transmitter')
    if (!Ember.empty(transmitter))
      @set('pinNumbers', transmitter.get('pin_numbers'))
      @set('pinNumber', @get('model.pin_number'))

  relationChange: (->
    @set('isRelationDirty', true)
    @getPinNumbers()
  ).observes('transmitter', 'pinNumber')

  setupData: (->
    formula = @get('model.formula')
    @get('model.records').then ((records)=>
      data = []
      for record in records.content
        value = record.get('y')

        scope = { x : value }
        calculated_value = null
        # apply the sensor formula
        try
          calculated_value = math.eval(formula, scope)
        catch
          calculated_value = value

        time = record.get 'x'
        data.push({x:time,y: calculated_value})

      @set('data', data)
#
    )
  ).observes('theFormula', 'record')
```

```
  showUnsavedMessage: ( ->
    !@get('model.isSaving') and (@get('model.isDirty') or @get('isRelationDirty'))
  ).property('model.isDirty', 'model.isSaving', 'isRelationDirty')

  # Possibly Keep and modify to delete and individual record
  actions:
    delete: ->
      @get('model').destroyRecord().then =>
        @transitionToRoute 'sensors'

    saveChanges: ->
      if @get('model.isDirty') or @get('isRelationDirty')
        @get('model').save().then =>
          @set('isRelationDirty', false)
          @set('theFormula', @get('model.formula'))

    led: (isChecked)->
      @set('model.led', isChecked)

  sockets:
    test: (data) ->
      data = data.record

      @store.find('sensor', data.sensor_id).then (sensor)=>
        record = {
          sensor: sensor
          time_stamp: (new Date).getTime()
          value: data.value
        }
        #@store.push('record', record)
        console.log(record)
        if @get('model').id == sensor.id
          @set('record', record)
```

D.15 js/controllers/signup.js.coffee

```
App.SignupController = Ember.Controller.extend
  needs: ['login']

  actions:
    createUser: ->
      data = @getProperties(
        'name',
        'email',
        'password',
```

```
      'password_confirmation'
    )
    user = @store.createRecord 'user', data

    user.save().then =>
      @setProperties
        name: null,
        email: null,
        password: null,
        password_confirmation: null

      loginController = @get('controllers.login')
      loginController.setProperties
        email: data.email,
        password: data.password

      loginController.send('login')
```

D.16 js/controllers/socket.js.coffee

```
App.SocketController = Ember.Controller.extend
  sensorData: null
  leds: null

  getLeds: ->
    @socket.emit('get_leds')

  actions:
    led: (isOn, pinName, transToken) ->
      @socket.emit('led', { transmitter_token: transToken, pin_name: pinName, value: isO


  sockets:
    leds: (data) ->
      @set('leds', data)

    live: (data) ->
      @set('sensorData', data)

    connect: ->
      console.log('Socket connected in socket')

    disconnect: ->
      console.log('Socket disconnected')
```

D.17 js/controllers/transmitters.js.coffee

```
App.TransmittersController = Ember.ArrayController.extend
  sortProperties: ['name']

  height: (->
    if window.mobilecheck()
      return 200
    else
      return 500
  )()
```

D.18 js/controllers/transmitters_new.js.coffee

```
App.TransmittersNewController = Ember.Controller.extend
  needs: ['login']

  actions:
    createTransmitter: ->
      data = @getProperties(
        'name',
      )
      user = @get('controllers.login').get('currentUser')
      transmitter = @store.createRecord 'transmitter', data

      @store.find('user', user.id).then (user)=>
        transmitter.set('user', user)

      self = @
      transmitter.save().then =>
        @setProperties
          name: null,
        self.transitionToRoute "transmitters.transmitter", transmitter
```

D.19 js/controllers/transmitters_transmitter.js.coffee

```
App.TransmittersTransmitterController = App.SocketController.extend
  showUnsavedMessage: ( ->
    @get('model.isDirty') and !@get('model.isSaving')
  ).property('model.isDirty', 'model.isSaving')

  actions:
    delete: ->
      @get('model').destroyRecord().then =>
        @transitionToRoute 'transmitters'
```

```
    saveChanges: ->
      @get('model').save() if @get('model.isDirty')
```

D.20 js/controllers/user.js.coffee

```
App.UserController = Ember.ObjectController.extend
  needs: ['login']

  isAdmin: (->
    user = @get('controllers.login.currentUser')
    return user.admin
  ).property('controllers.login.currentUser')

  actions:
    delete: ->
      @get('model').destroyRecord().then =>
        @transitionToRoute 'users'
```

D.21 js/controllers/user_live.js.coffee

```
App.UserLiveController = App.SocketController.extend
  needs: ['user_live_transmitters']

  transmitters: Ember.computed.alias('controllers.user_live_transmitters')

  actions:
    recordValue: (pinName, transToken) ->
      console.log('recordValue in socket')
      @socket.emit('record_value', { transmitter_token: transToken, pin_name: pinName})

    redirectToHistory: (sensor) ->
      @transitionToRoute('user.sensors.sensor', sensor)
```

D.22 js/controllers/user_live_transmitters.js.coffee

```
App.UserLiveTransmittersController = Ember.ArrayController.extend
  sortProperties:['name']
```

D.23 js/controllers/users.js.coffee

```
App.UsersController = Ember.ArrayController.extend
  needs: ['login']

  sortProperties: ['name']
```

D.24 js/controllers/users_new.js.coffee

```
App.UsersNewController = Ember.Controller.extend

  actions:
    createUser: ->
      data = @getProperties(
        'name',
        'email',
        'password',
        'password_confirmation'
      )
      user = @store.createRecord 'user', data
      user.save().then =>
        @setProperties
          name: null,
          email: null,
          password: null,
          password_confirmation: null
        @transitionToRoute 'user', user
```

D.25 js/helpers/format_date.js.coffee

```
Ember.Handlebars.helper('format-date', (date, format)->
  moment(date).format(format))
```

D.26 js/helpers/get_element.js.coffee

```
Ember.Handlebars.helper('get-element', (record, index)->
  record[index]
)

Ember.Handlebars.helper('get-first-element-formatted', (record)->
  moment(record[0]).format('LLL')
)
```

D.27 js/helpers/markdown.js.coffee

```
Ember.Handlebars.registerBoundHelper('markdown', (content) ->
  new Handlebars.SafeString(marked(content))
)
```

D.28 js/helpers/sensor_name.js.coffee

```
Ember.Handlebars.helper('trans-sensor', (transmitterId, pinName, sensorName)->
  if transmitterId == null || pinName == null
    return "(unassigned) " + sensorName
  else
    return "(" + transmitterId + ":" + pinName + ") " + sensorName
)
```

D.29 js/models/clockwork_database_event.js.coffee

```
App.ClockworkDatabaseEvent = DS.Model.extend
  at: DS.attr('string')
  frequency_quantity: DS.attr('number')
  frequency_period: DS.belongsTo('frequency_period')
```

D.30 js/models/frequency_period

```
App.FrequencyPeriod = DS.Model.extend
  name: DS.attr('string')
```

D.31 js/models/pin_number

```
App.PinNumber = DS.Model.extend
  name: DS.attr('string')
  transmitter: DS.belongsTo('transmitter')
  sensor: DS.belongsTo('sensor')
```

D.32 js/models/post.js.coffee

```
App.Post = DS.Model.extend
  title: DS.attr('string')
  content: DS.attr('string')
  updated_at: DS.attr('date')
  tag: DS.belongsTo('tag')
```

D.33 js/models/record.js.coffee

```
App.Record = DS.Model.extend
  x: DS.attr('number')
  y: DS.attr('number')
  sensor: DS.belongsTo('sensor')
```

D.34 js/models/sensor.js.coffee

```
App.Sensor = DS.Model.extend
  name: DS.attr('string')
  pin_number: DS.belongsTo('pin_number')
  clockwork_database_event: DS.belongsTo('clockwork_database_event')
  formula: DS.attr('string')
  lower: DS.attr('string')
  upper: DS.attr('string')
  led: DS.attr('boolean')
  user: DS.belongsTo('user')
  records: DS.hasMany('records', {async: true})
```

D.35 js/models/tag.js.coffee

```
App.Tag = DS.Model.extend
  name: DS.attr('string')
  posts: DS.hasMany('posts', {async: true})
```

D.36 js/models/transmitter.js.coffee

```
App.Transmitter = DS.Model.extend
  name: DS.attr('string')
  transmitter_token: DS.attr('string')
  pin_numbers: DS.hasMany('pin_numbers', {async: true})
  user: DS.belongsTo('user')
```

D.37 js/models/user.js.coffee

```
App.User = DS.Model.extend
  name: DS.attr('string')
  email: DS.attr('string')
  token: DS.attr('string')
  admin: DS.attr('boolean')
  password: DS.attr('string')
  password_confirmation: DS.attr('string')
  transmitters: DS.hasMany('transmitter', {async: true})
  sensors: DS.hasMany('sensors', {async: true})
```

D.38 js/models/router.js.coffee

```
# For more information see: http://emberjs.com/guides/routing/
#
App.Router.reopen
  #location: 'auto' # gets ride of the hash
  #rootURL: '/'
```

```
App.Router.map ()->
  @route 'home'
  @route 'help'
  @route 'login'
  @route 'signup'

  @resource 'blog', ->
    @resource 'posts', ->
      @route 'post', path: "/:post_id", ->
        @route 'edit'
      @route 'new'

  @resource 'user', path: '/', ->
    @resource 'transmitters', ->
      @route 'transmitter', path: "/:transmitter_id"
      @route 'new'
    @resource 'sensors', ->
      @route 'new'
      @route 'sensor', path: "/:sensor_id"
    @route 'live'
```

D.39 js/routes/application.js.coffee

```
App.ApplicationRoute = Ember.Route.extend
  actions:
    logout: ()->
      @controllerFor('login').resetToken()
      @transitionTo('login')
```

D.40 js/routes/posts_new.js.coffee

```
App.PostsNewRoute = Ember.Route.extend
  setupController: (controller, model)->
    @_super(controller, model)
    controller.setupTags()
```

D.41 js/routes/transmitters_transmitter.js.coffee

```
App.TransmittersTransmitterRoute = App.AuthenticatedRoute.extend
  model: (params) ->
    @store.find('transmitter', params.transmitter_id)

  setupController: (controller, model) ->
```

```
    @_super(controller, model)
    controller.getLeds()
```

D.42 js/routes/authenticated.js.coffee

```
App.AuthenticatedRoute = Ember.Route.extend
  beforeModel: (transition)->
    if Ember.isEmpty(@controllerFor('login').get('token'))
      @redirectToLogin(transition)

  redirectToLogin: (transition)->
    @controllerFor('login').set('attemptedTransition', transition)
    @transitionTo('login')

  actions:
    error: (error, transition) ->
      if (error.status is 401)
        @redirectToLogin(transition)
      else
        console.log(error)
        alert('Something went wrong')
```

D.43 js/routes/posts_post_edit.js.coffee

```
App.PostsPostEditRoute = Ember.Route.extend

  activate: ->
    controller = @controllerFor('posts.post')
    controller.set 'isEditing', true

  deactivate: ->
    controller = @controllerFor('posts.post')
    controller.set 'isEditing', false
```

D.44 js/routes/user.js.coffee

```
App.UserRoute = App.AuthenticatedRoute.extend
  model: (params) ->
    user = Ember.$.cookie('auth_user')
    if (user)
      return @store.find('user', user.id)

  beforeModel: (transition) ->
    if Ember.isEmpty(@controllerFor('login').get('token'))
```

```
      @transitionTo('home')
    else if (Ember.get(transition, 'targetName') == 'user.index')
      @transitionTo('sensors')
```

D.45 js/routes/home.js.coffee

```
App.HomeRoute = Ember.Route.extend
  beforeModel: ->
    if (!Ember.isEmpty(@controllerFor('login').get('token')))
      @transitionTo('sensors')
```

D.46 js/routes/sensors.js.coffee

```
App.SensorsRoute = App.AuthenticatedRoute.extend
  model: ->
    @modelFor('user').get('sensors')

  #afterModel: (sensors, transition) ->
  #if (sensors.get('length') >= 1)
  #@transitionTo('sensors.sensor', sensors.get('firstObject'))
```

D.47 js/routes/user_live.js.coffee

```
App.UserLiveRoute = App.AuthenticatedRoute.extend
  model: ->
    @modelFor('user').get('transmitters')

  setupController: (controller, model) ->
    @_super(controller, model)
    controller.getLeds()
    @controllerFor('user_live_transmitters').set('model', model)
```

D.48 js/routes/login.js.coffee

```
App.LoginRoute = Ember.Route.extend
  beforeModel: (transition) ->
    if (!Ember.isEmpty(@controllerFor('login').get('token')))
      @transitionTo('sensors')

  setupController: (controller, context) ->
    controller.reset()

  actions:
    error: (error, transition) ->
```

```
      if (error.status == 401)
        console.log("401!!!")
```

D.49 js/routes/sensors_sensor.js.coffee

```
App.SensorsSensorRoute = App.AuthenticatedRoute.extend
  model: (params) ->
    @store.find('sensor', params.sensor_id)

  setupController: (controller, model) ->
    controller.set('model', model)
    controller.getTransmitters()
    controller.getFrequencyPeriods()
    controller.setupData()
```

D.50 js/routes/users.js.coffee

```
App.UsersRoute = App.AuthenticatedRoute.extend
  model: -> @store.find 'user'
```

D.51 js/routes/posts.js.coffee

```
App.PostsRoute = Ember.Route.extend
  model: ->
    @store.find('post')

  setupController: (controller, model)->
    @_super(controller, model)
    controller.setupTags()
```

D.52 js/routes/transmitters.js.coffee

```
App.TransmittersRoute = App.AuthenticatedRoute.extend
  model: ->
    @modelFor('user').get('transmitters')

  afterModel: (transmitters, transition) ->
    if (transmitters.get('length') >= 1)
      @transitionTo('transmitters.transmitter', transmitters.get('firstObject'))
```

D.53 js/store.js.coffee

```
# http://emberjs.com/guides/models/#toc_store
# http://emberjs.com/guides/models/pushing-records-into-the-store/
```

```
App.ApplicationStore = DS.Store.extend()

DS.RESTAdapter.reopen
  namespace: 'api'

# Override the default adapter with the `DS.ActiveModelAdapter` which
# is built to work nicely with the ActiveModel::Serializers gem.
App.ApplicationAdapter = DS.ActiveModelAdapter.extend()

App.UserSerializer = DS.RESTSerializer.extend(DS.EmbeddedRecordsMixin, {
  attrs: {
    transmitters: { embedded: 'always' }
    sensors: { embedded: 'always' }
  }
})

App.SensorSerializer = DS.RESTSerializer.extend(DS.EmbeddedRecordsMixin, {
  attrs: {
    records: { embedded: 'always' }
    pin_number: { embedded: 'always' }
    clockwork_database_event: { embedded: 'always' }
  }
})

App.TransmitterSerializer = DS.RESTSerializer.extend(DS.EmbeddedRecordsMixin, {
  attrs: {
    pin_numbers: { embedded: 'always' }
  }
})

App.ClockworkDatabaseEventSerializer = DS.RESTSerializer.extend(DS.EmbeddedRecordsMixin,
  attrs: {
    frequency_period: { embedded: 'always' }
  }
})

App.PostSerializer = DS.RESTSerializer.extend(DS.EmbeddedRecordsMixin, {
  attrs: {
    tag: { embedded: 'always' }
  }
})

Ember.View.reopen({
  didInsertElement: ->
    @_super()
```

```
    Ember.run.scheduleOnce "afterRender", @, @afterRenderEvent
    return

  afterRenderEvent: ->
})
```

D.54 js/templates/application.js.emblem

```
header.navbar.navbar-fixed-top.navbar-inverse
  meta name="viewport" content="width=device-width, initial-scale=1, maxium-scale=1"
    .container
      .navbar-header
        button.navbar-toggle data-target=".bs-navbar-collapse" data-toggle="collapse" ty
          span.sr-only Toggle Navigation
          span.icon-bar
          span.icon-bar
          span.icon-bar
        if isAuthenticated
          link-to "sensors" id='logo' | LifeSense
        else
          link-to "home" id='logo' | LifeSense
      .collapse.navbar-collapse.bs-navbar-collapse
        nav
          ul.nav.navbar-nav.navbar-right
            if isAuthenticated
              link-to "user.live" tagName="li"
                a Live
              link-to "sensors" tagName="li"
                a Sensors
              link-to "transmitters" tagName="li"
                a Transmitters
            else
              link-to "home" tagName="li"
                a Home
            link-to "posts" tagName="li"
              a Blog
            link-to "help" tagName="li"
              a Help
            if isAuthenticated
              li
                a click="logout" Logout
            else
              link-to "login" tagName="li"
                a Log in
```

```
section#main
  = outlet

footer
```

D.55 js/templates/home.js.emblem

```
.jumbotron.container.center
  unless isAuthenticated
    h1
      | Welcome to LifeSense

    h2
      | This is where everything that is awesome happens.

    link-to "signup" class="btn btn-lg btn-primary" | Sign up
```

D.56 js/templates/posts.js.emblem

```
.navbar.navbar-default.span5
  ul.nav.navbar-nav
    li{bind-attr class=isAll}
      a click=all All
    li{bind-attr class=isSoftware}
      a click=software Software
    li{bind-attr class=isHardware}
      a click=hardware Hardware
    li{bind-attr class=isMeeting}
      a click=meeting Meeting
    li{bind-attr class=isChallenges}
      a click=challenges Challenges
.row
  .col-sm-4
    if isAdmin
      link-to "posts.new" classNames="new btn btn-primary" | New Post
      br
    .list-group
      collection Ember.ListView contentBinding="filteredContent" height=height rowHeight
        link-to 'posts.post' content classNames="list-group-item"
          | #{=format-date updated_at "MMM DD" } #{title}
  .col-sm-8
    outlet
```

D.57 js/templates/blog.js.emblem

```
.container
  outlet
```

D.58 js/templates/components/bootstrap-switch-led.js.emblem

```
input type="checkbox"
```

D.59 js/templates/components/line-time-series-chart.js.emblem

```
div id=chartId style='width 100%'
```

D.60 js/templates/components/live-time-series-chart.js.emblem

```
div id=chartId style='width 100%'
if sensor
  .col-xs-6
    a.btn.btn-primary click="recordValue" Record
  /.col-xs-6
    link-to 'user.sensors.sensor' sensor
```

D.61 js/templates/loading.js.emblem

```
.loading-wrapper
  h1
    | Loading
    i.ellipsis
      i
        | .
      i
        |.
      i
        |.
```

D.62 js/templates/transmitters.js.emblem

```
.container
  .row
    .col-sm-3
      link-to "transmitters.new" classNames="new btn btn-primary" | New Transmitter
      br
      .list-group
        collection Ember.ListView contentBinding="controller" height=height rowHeight=1
          link-to 'transmitters.transmitter' content classNames="list-group-item"
```

```
            name
    .col-sm-9
      outlet
```

D.63 js/templates/users.js.emblem

```
outlet
```

D.64 js/templates/login.js.emblem

```
.container
  if errorMessage
    .alert.bg-danger
       errorMessage
  h1
    | Log in
  .row
    .col-md-6.col-md-offset-3
      form submit="login"
        / the '=' is for the input helper instead of the HTML element
        =input value=email type="text" placeholder="E-mail"
        =input value=password type="password" placeholder="Password"
        input.btn.btn-primary type="submit" value="Log in"
```

D.65 js/templates/sensors.js.emblem

```
.container
  .row
    .col-sm-4
      link-to "sensors.new" classNames="new btn btn-primary" | New Sensor
      br
      p (Transmitter id:Pin name) sensor

      .list-group
        collection Ember.ListView contentBinding="controller" height=height rowHeight=1
          link-to 'sensors.sensor' content classNames="list-group-item"
            trans-sensor pin_number.transmitter.id pin_number.name name
    .col-sm-8
      outlet
```

D.66 js/templates/help.js.emblem

```
.container.center.jumbotron
  h1
    | Help
```

```
  p
    | Here is where you get help for things.
```

D.67 js/templates/signup.js.emblem

```
.container
  h1 New User

  form
    fieldset
      dl
        dt: label Name:
        dd: view Ember.TextField value=name
      dl
        dt: label Email:
        dd: view Ember.TextField value=email
      dl
        dt: label Password:
        dd: view Ember.TextField type="password" value=password
      dl
        dt: label Password Confirmation:
        dd: view Ember.TextField type="password" value=password_confirmation
      fieldset.actions
        input.btn.btn-lg.btn-primary type='submit' value='Create User' click='createUser
```

D.68 js/templates/user.js.emblem

```
outlet
```

D.69 js/templates/posts/post.js.emblem

```
outlet

unless isEditing
  h5.to-right
    =format-date model.created_at "MMM DD, YYYY"
  br
  h1.center
    model.title

  markdown model.content

  br
```

```
  br
  if isAdmin
    link-to 'edit' 'posts.post.edit' classNames="btn btn-primary"

    a.delete.btn.btn-primary click="delete" Delete
```

D.70 js/templates/posts/new.js.emblem

```
h1 New Post

form
  fieldset
    dl
      dt: label Title:
      dd: view Ember.TextField value=title

    dl
      dt: label Tag:
      dd: view Ember.Select content=tags optionValuePath="content" optionLabelPath="cont

    dl
      dt: label Content:
      dd: view Ember.TextArea value=content rows=15
    fieldset.actions
      input.btn.btn-primary type='submit' value='Create Post' click='createPost'
```

D.71 js/templates/posts/post/edit.js.emblem

```
dl
  dt: label Title:
  dd: view Ember.TextField value=model.title

dl
  dt: label Content:
  dd: view Ember.TextArea value=model.content rows=25

a.btn.btn-primary type='submit' value='Save Changes' click='saveChanges' Save
a.delete.btn.btn-primary click="cancel" cancel
```

D.72 js/templates/sensors/new.js.emblem

```
h1 New Sensor

form
  fieldset
```

```
    dl
      dt: label Name:
      dd: view Ember.TextField value=name
    dl
      dt: label Pin:
      dd: view Ember.TextField type="number" value=pin_number
    dl
      dt: label Formula:
      dd: view Ember.TextField value=formula
    dl
      dt: label Lower Bound:
      dd: view Ember.TextField value=lower
    dl
      dt: label Upper Bound:
      dd: view Ember.TextField value=upper
    fieldset.actions
      input.btn.btn-lg.btn-primary type='submit' value='Create Sensor' click='createSens
```

D.73 js/templates/sensors/sensor.js.emblem

```
unless isLED
  line-time-series-chart chartId='sensed' title=model.name data=data formula=theFormula

p
  label Name:
  view Ember.TextField value=model.name

  label Transmitter:
  view Ember.Select content=transmitters optionValuePath="content" optionLabelPath="cont

  label Pin Name:
  view Ember.Select content=pinNumbers optionValuePath="content" optionLabelPath="conten

  bootstrap-switch-led isLED=model.led action="led" labelText="Output" size="medium"
  br


  unless isLED
    label Formula:
    view Ember.TextField value=model.formula

    label Frequency Quantity:
    view Ember.TextField value=model.clockwork_database_event.frequency_quantity

    label Frequency Period:
```

```
    view Ember.Select content=frequencyPeriods optionValuePath="content" optionLabelPath

    label Lower Bound:
    view Ember.TextField value=model.lower

    label Upper Bound:
    view Ember.TextField value=model.upper

a.delete.btn.btn-primary click="delete" Delete
a.btn.btn-primary click="saveChanges" Save
if showUnsavedMessage
  .unsaved unsaved changes
if model.isSaving
  .saving saving...

br
br


/.table-responsive
  table.table.table-striped.table-bordered.table-hover.table-condensed
    thead
      tr
        th Timestamp
        th Record
    tbody
      each record in data
        tr
          td
            =get-first-element-formatted record
          td
            =get-element record 1
```

D.74 js/templates/transmitters/new.js.emblem

```
h1 New Transmitter

form
  fieldset
    dl
      dt: label Name:
      dd: view Ember.TextField value=name
    fieldset.actions
      input.btn.btn-lg.btn-primary type='submit' value='Create Transmitter' click='creat
```

D.75 js/templates/transmitters/transmitter.js.emblem

```
p
  label Name:
  view Ember.TextField value=model.name

p
  | Token: #{model.transmitter_token}

a.delete.btn.btn-primary click="delete" Delete
a.btn.btn-primary click="saveChanges" Save
if showUnsavedMessage
  .unsaved unsaved changes
if model.isSaving
  .saving saving...

.row
  h3 Outputs
  each pin in model.pin_numbers
    if pin.sensor.led
      .col-sm-4
        label
          pin.sensor.name
        bootstrap-switch-led isLED=pin.sensor.led size="large" pinName=pin.name action="
.row
  h3 Sensors
  ul
  each pin in model.pin_numbers
    if pin.sensor
      unless pin.sensor.led
        li
          pin.sensor.name
```

D.76 js/templates/user/live.js.emblem

```
h1 Live Sensors

.col-sm-4
  each transmitter in transmitters
    .row.center
      br
      label
        transmitter.name
      br
      each pin in transmitter.pin_numbers
```

```
        if pin.sensor.led
          .col-xs-6
            label
              pin.sensor.name
            bootstrap-switch-led isLED=pin.sensor.led size="large" pinName=pin.name acti
.col-sm-8
  each transmitter in transmitters
    each pin in transmitter.pin_numbers
      unless pin.sensor.led
        .col-sm-6
          live-time-series-chart chartId=pin.sensor.id title=pin.sensor.name sensorData=
```

D.77 js/templates/user/sensors.js.emblem

```
h1
  | Sensors
```

D.78 js/templates/users/new.js.emblem

```
h1 New User

form
  fieldset
    dl
      dt: label Name:
      dd: view Ember.TextField value=name
    dl
      dt: label Email:
      dd: view Ember.TextField value=email
    dl
      dt: label Password:
      dd: view Ember.TextField type="password" value=password
    dl
      dt: label Password Confirmation:
      dd: view Ember.TextField type="password" value=password_confirmation
    fieldset.actions
      input.btn.btn-lg.btn-primary type='submit' value='Create User' click='createUser'
```

D.79 js/templates/views/bootstrap-switch.js.emblem

```
input type="checkbox"
```

D.80 js/views/bootstrap_switch.js.coffee

```
App.BootstrapSwitch = Ember.View.extend(
  classNames: ['switch']

  templateName: 'views/bootstrap-switch'

  afterRenderEvent: ->
    @$("[type='checkbox']").bootstrapSwitch('size', 'small')
    controller = @controller
    action = @action
    @$("[type='checkbox']").on "switchChange.bootstrapSwitch", (e, data)->
      controller.send(action, data)
)
```

E Source code of PCB software

```
#include <SPI.h>
#include <WiFi.h>

#define DEBUG 0

const IPAddress INADDR_NONE(0,0,0,0);
char ssid[] = "cc3000";
char pass[] = "coolestever";
const String token = "NGyWTLYByALMCycdrZKgwg";

// Settings
IPAddress hostIp(192,168,150,1);

// Variable Setup
long lastConnectionTime = 0;
boolean lastConnected = false;
int failedCounter = 0;
int PORT = 4033;

// Initialize WiFi Client
WiFiServer server(80);
WiFiClient client;
int statusConfig = 0;

int INPUT_PINS_SIZE = 7;
// These are the names we will send to the server
char* INPUT_PINS_NAME[] = {"A0","A1","A2","A4","A5","A6","A7"};
int INPUT_PINS[] = {A0,A1,A2,A4,A5,A6,A7};
```

```
int   OUTPUT_PINS_SIZE   = 8;
// These are the names we are expecting the server to give us.
char* OUTPUT_PINS_NAME[] = {"PF_1","PF_2","PF_3","PB_3","PC_4","PC_5","PC_6","PC_7","PD_
int   OUTPUT_PINS[]      = { PF_1 , PF_2 , PF_3 , PB_3 , PC_4 , PC_5 , PC_6 , PC_7 , PD_

void setInputPins() {
  if (DEBUG) Serial.println("Input Pins:  ");
  for (int i=0;i<INPUT_PINS_SIZE;i++) {
    if (DEBUG) Serial.println(INPUT_PINS[i]);
    pinMode(INPUT_PINS[i], INPUT);
  }
  if (DEBUG) Serial.println("");
}

void setOutputPins() {
  if (DEBUG) Serial.println("Output Pins:  ");
  for (int i=0;i<INPUT_PINS_SIZE;i++) {
    if (DEBUG) Serial.println(OUTPUT_PINS[i]);
    pinMode(OUTPUT_PINS[i], OUTPUT);
    digitalWrite(OUTPUT_PINS[i],LOW);
  }
  if (DEBUG) Serial.println("");
}

void setup()
{
  // Start Serial for debugging on the Serial Monitor
  Serial.begin(115200);

  setInputPins();
  setOutputPins();

  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(BLUE_LED, OUTPUT);
  digitalWrite(BLUE_LED, LOW);
  digitalWrite(RED_LED, LOW);
  digitalWrite(GREEN_LED, LOW);

  // Set communication pins for CC3000
  WiFi.setCSpin(18);  // 18: P2_2 @ F5529, PE_0 @ LM4F/TM4C
  WiFi.setENpin(2);   //  2: P6_5 @ F5529, PB_5 @ LM4F/TM4C
  WiFi.setIRQpin(19); // 19: P2_0 @ F5529, PB_2 @ LM4F/TM4C
```

```
  // Start WiFi
  startWiFi();
}

void loop()
{
  if (client.connected()) {
    // Start building our JSON data payload
    String data = "{ \"transmitter_token\":";
    data += "\""+ token +"\",";
    readInputPins(data);
    data += "}\n\n";
    pushUpdate(data);

    // Read the response
    parseResponse();
  }
  else {
    reconnect();
  }

  // Check if WiFi needs to be restarted
  if (failedCounter > 3 ) {
    failedCounter = 0;
    Serial.println("Greater than 3");
    startWiFi();
  }
}

// Read the input pins specified in INPUT_PIN and append the json to
// the result string.
//
// Adding to the result string like this is not the most efficient, but it
// is probably fast enough for 8 values.
void readInputPins(String &result) {
  for (int i=0;i<INPUT_PINS_SIZE;i++) {
    result = result + "\"" + INPUT_PINS_NAME[i] + "\":";
    String value = String(analogRead(INPUT_PINS[i]), DEC);
    //String value = String(i * 10000);
    result += value;
    if (i!=(INPUT_PINS_SIZE-1)) {
      result += ",";
    }
  }
}
```

```
void reconnect() {
  Serial.println("Connecting!");
  Serial.println();

  client.stop();
  if (client.connect(hostIp, PORT)) {
    Serial.println("Connected!");
    digitalWrite(RED_LED, LOW);
    failedCounter=0;
  }
  else {
    Serial.println("Connection failed.");
    digitalWrite(RED_LED, HIGH);
    digitalWrite(BLUE_LED, LOW);
    digitalWrite(GREEN_LED, LOW);

    failedCounter++;
    delay(500);
  }
}

void pushUpdate(String tsData)
{
  if (DEBUG) Serial.println("Trying to push: ");

  if (DEBUG) Serial.println(tsData);

  client.print("POST /scale HTTP/1.1\n");
  client.print("Host: localhost:8080\n");
  client.print("Accept: \*/\*\n");
  client.print("Content-Type: application/json\n");
  client.print("Content-Length: ");

  client.print(tsData.length());
  client.print("\n\n");
  client.print(tsData);

  if (DEBUG) Serial.println("pushed an update!");
}

void setOutput(String key, String value) {
  if (DEBUG) Serial.println("setOutput:  " + key + " , " + value);
  // This is an ugly search, but is probably fast enough for such low values
  for(int i=0;i<OUTPUT_PINS_SIZE;i++) {
```

```
    // Converting these to strings everytime is probably slow too.
    if (String(OUTPUT_PINS_NAME[i]) == key) {
      if (value == "true") {
        if (DEBUG) Serial.println("Setting " + String(OUTPUT_PINS[i]) + " true.");
        digitalWrite(OUTPUT_PINS[i], HIGH);
      }
      else if (value == "false") {
        if (DEBUG) Serial.println("Setting " + String(OUTPUT_PINS[i]) + " false.");
        digitalWrite(OUTPUT_PINS[i], LOW);
      }
      else {
        Serial.println("Got unknown OUTPUT value '" + value + "' for key '" + key + "'."
      }
      return;
    }
  }
  Serial.println("Could not find key '" + key + "' in OUTPUTS.");
}

// This will read and parse a JSON response with key:value pairs
//
// It is NOT a true JSON parser, and does dumb things like
// ignore all spaces, so don't expect it to handle arbitrary JSON.
void parseResponse() {
  String currentLine  = "";
  String currentKey   = "";
  String currentValue = "";
  int foundJSON = 0;
  while (client.available()) {
    char c = client.read();
    if (c == '{') foundJSON = 1;
    if (!foundJSON) continue;
    if (c == ' ' ||  // ignore ALL whitespace
        c == '{' ||  // and some JSON characters
        c == '"') continue;

    if (DEBUG) Serial.print(c);
    if (c == '\n') {
      currentLine = "";
    }
    else if (c == '\r') {
    }
    else if (c == ':') {
      currentKey = currentLine;
      currentLine = "";
```

```
    }
    else if (c == ',' || c == '}') {
      currentValue = currentLine;
      setOutput(currentKey, currentValue);
      currentKey  = "";
      currentValue= "";
      currentLine = "";
    }
    else {
      currentLine += c;      // add it to the end of the currentLine
    }
  }
  if (DEBUG) Serial.println();
  if (DEBUG) Serial.println();
}

void startWiFi()
{
  digitalWrite(RED_LED, HIGH);

  WiFi.disconnect();
  client.stop();

  Serial.print("Connecting LaunchPad to SSID:");
  Serial.print(ssid);
  Serial.println();

  // Connect to network and obtain an IP address using DHCP

  if (WiFi.begin(ssid, pass) == 0) {
    Serial.println("Connected to WiFi!");
    digitalWrite(GREEN_LED, HIGH);
    digitalWrite(RED_LED, LOW);
    while ( WiFi.status() != WL_CONNECTED) {
      // print dots while we wait to connect
      blinkLed(GREEN_LED);
      Serial.print("*");
    }

    int failedCountMax = 15;
    int failedCount = 0;
    IPAddress ip = INADDR_NONE;
    while (ip == INADDR_NONE) {
      // print dots while we wait for an ip addresss
      ip = WiFi.localIP();
```

```
      delayForTime(1000);
      if (failedCount++ > failedCountMax) {
        startWiFi();
        return;
      }
      Serial.print(".");
    }

    digitalWrite(BLUE_LED, HIGH);
    digitalWrite(GREEN_LED, HIGH);
    printWifiStatus();
    Serial.println();
  } else {
    Serial.println("LaunchPad connected to network using DHCP");
    Serial.println();
  }

  delay(1000);
}

void delayForTime(int time) {
  int toDelay = time;
  int beenDelayed = 0;
  while (beenDelayed < toDelay) {
    blinkLed(GREEN_LED);
    beenDelayed += 1000;
  }
}


void blinkLed(int LED) {
  digitalWrite(LED, HIGH);
  delay(500);
  digitalWrite(LED, LOW);
  delay(500);
}

void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
```

```
  Serial.println(ip);
}
```

F Source code of Node

```
express = require('express')
app = express()
bodyParser = require('body-parser')
http = require('http')
server = http.createServer(app)
request = require('request')

io = require('socket.io').listen(server)

leds = {}
sensors = {}

app.use(bodyParser.urlencoded({
  extended: true
}))
app.use(bodyParser.json())

recordsUrl = 'http://localhost:3000/api/records'
#recordsUrl = 'lifesense-herokuapp.com/api/records'
server.listen(process.env.PORT || 4033)
console.log("Listening on port 4033")

getTransToken = (data)->
  transToken = data.transmitter_token # always being with a letter

sendToRails = (transToken, pin) ->
  if sensors[transToken]
    y = sensors[transToken][pin]
    postData = {
      record:{
        y: y
        transmitter_token: transToken
        pin_number:pin}
    }
    options = {
      method: 'post',
      body:postData,
      json: true,
      url: recordsUrl
```

```
    }
    request options, (error, res, body) ->
      if (!error && res.statusCode== 201)
        console.log(body)


io.sockets.on 'connection', (socket) ->
  console.log("WE have a connection yo")

  app.post '/value', (req, res)->
    transToken = req.body.transmitter_token
    pin = req.body.pin
    sendToRails(transToken, pin)


  app.post '/scale', (req, res)->
    #console.log("post" + req.connection.remoteAddress)
    sensors[req.body.transmitter_token] = req.body
    io.sockets.emit 'live', req.body
    token = getTransToken req.body
    res.send(leds[token])
    res.end

  socket.on 'get_leds', (data) ->
    console.log("GET LEDS")
    socket.emit 'leds', leds

  socket.on 'record_value', (data) ->
    console.log("record value")
    console.log data

    sendToRails(data.transmitter_token, data.pin_name)

  socket.on 'led', (data) ->
    transToken = getTransToken(data)
    transmitter =leds[transToken]
    if typeof transmitter == 'undefined'
      leds[transToken] = {}
      transmitter =leds[transToken]
      transmitter[data.pin_name] = data.value
    else
      transmitter[data.pin_name] = data.value
    data = {}
    data[transToken] = transmitter
    io.sockets.emit 'leds', data
```

G PCB schematic

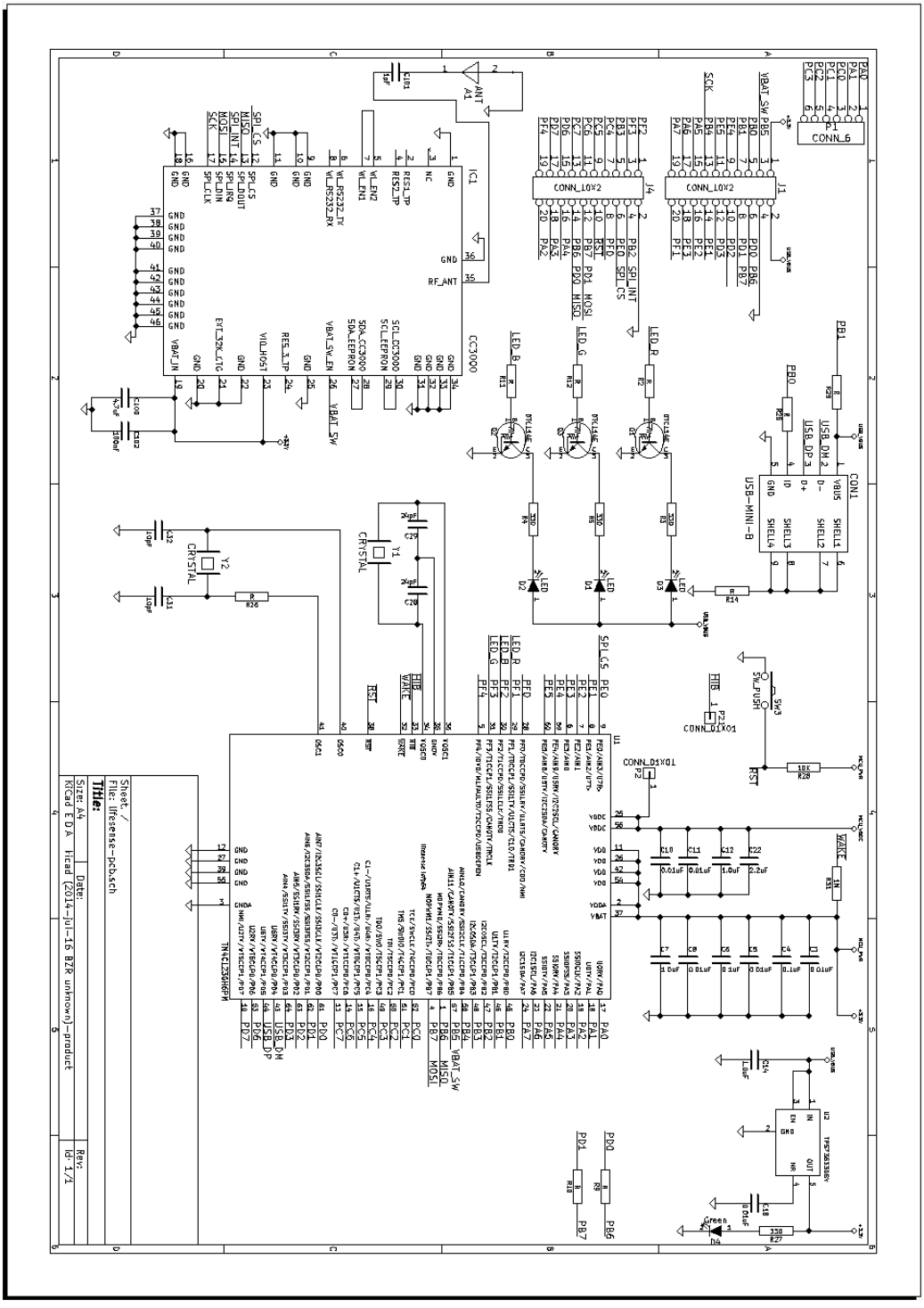

Figure 1: PCB schematic

H PCB Top layer

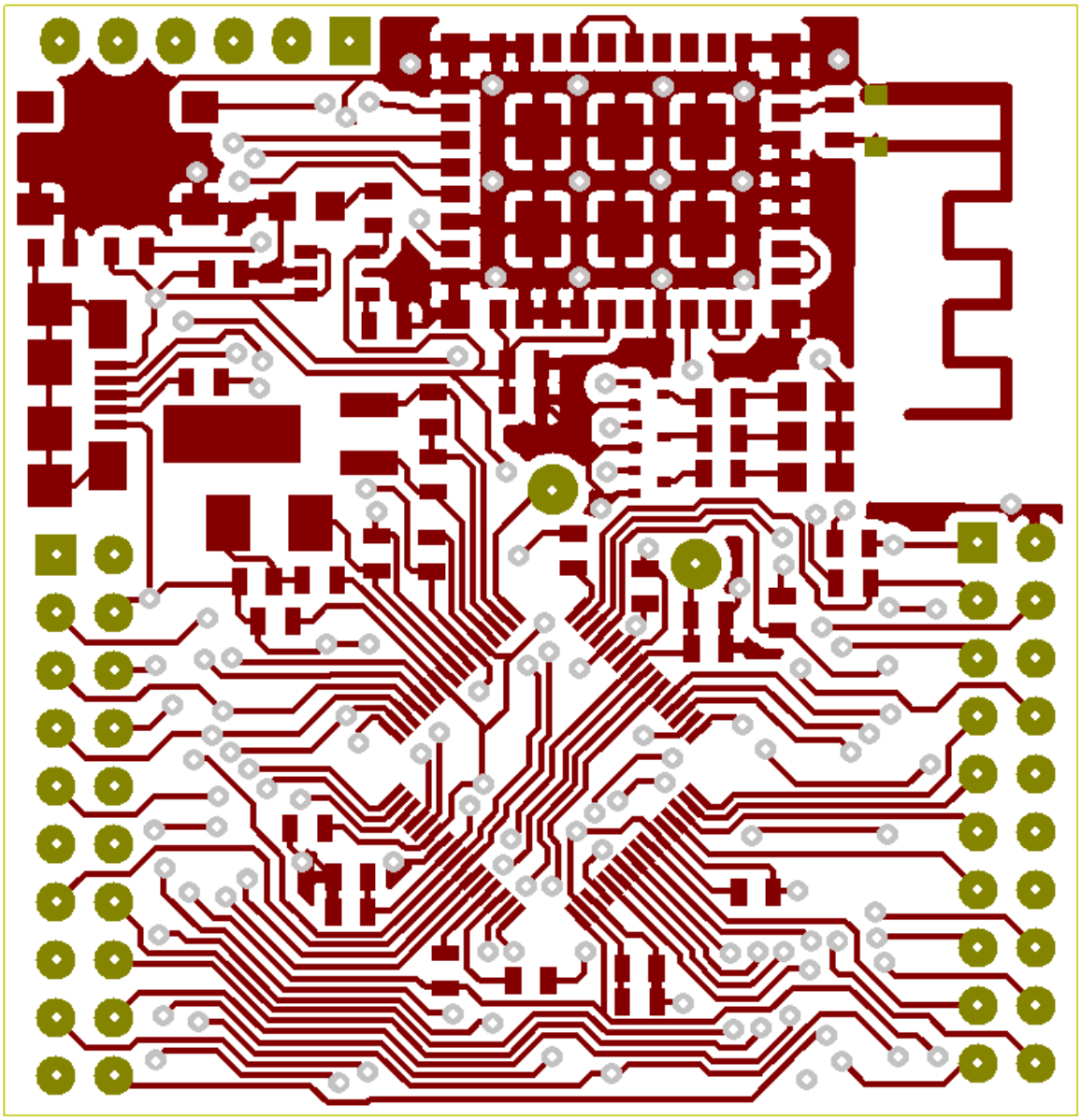

Figure 2: PCB Top layer

I PCB Bottom layer

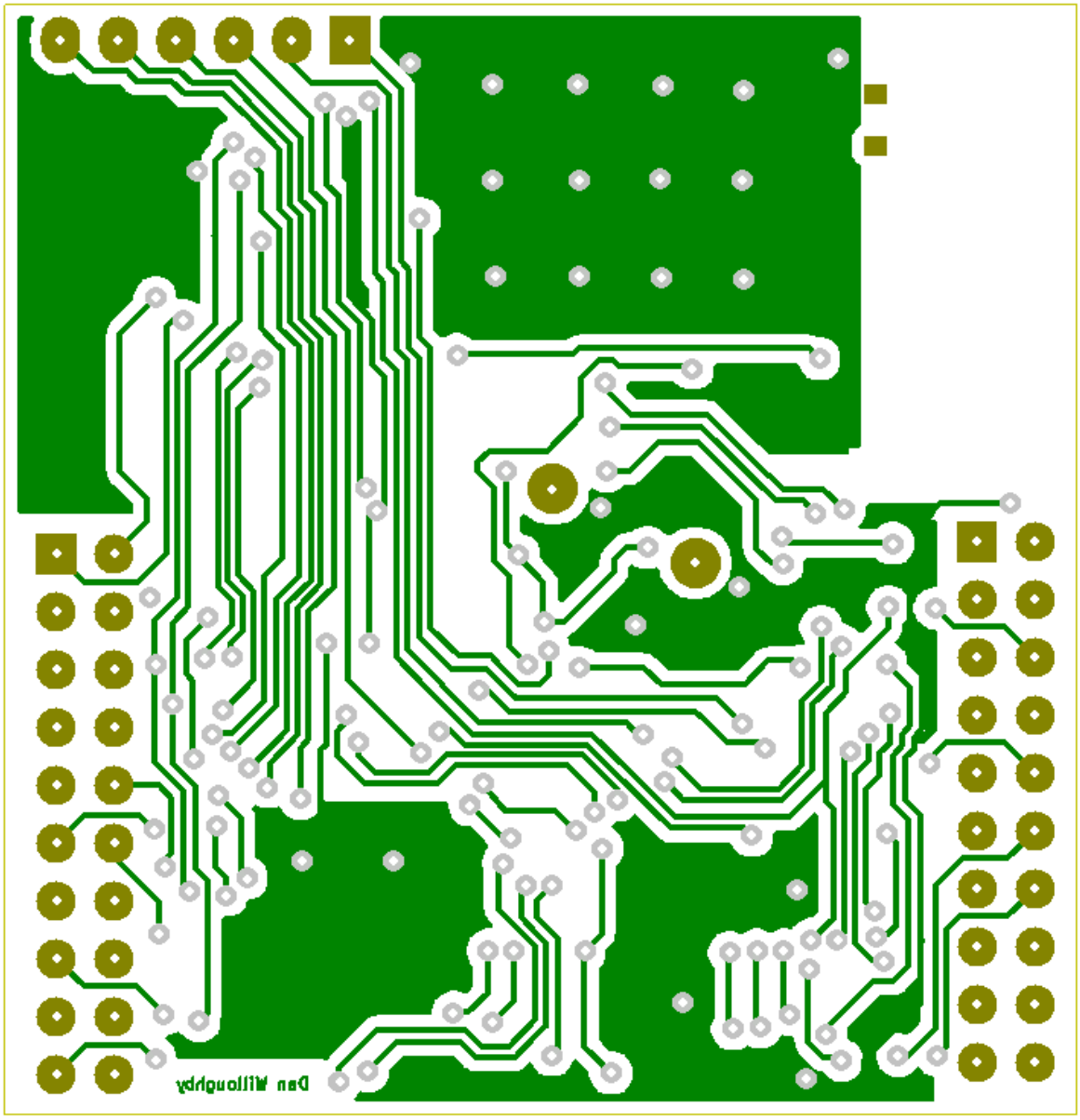

Figure 3: PCB Bottom layer

J Poster

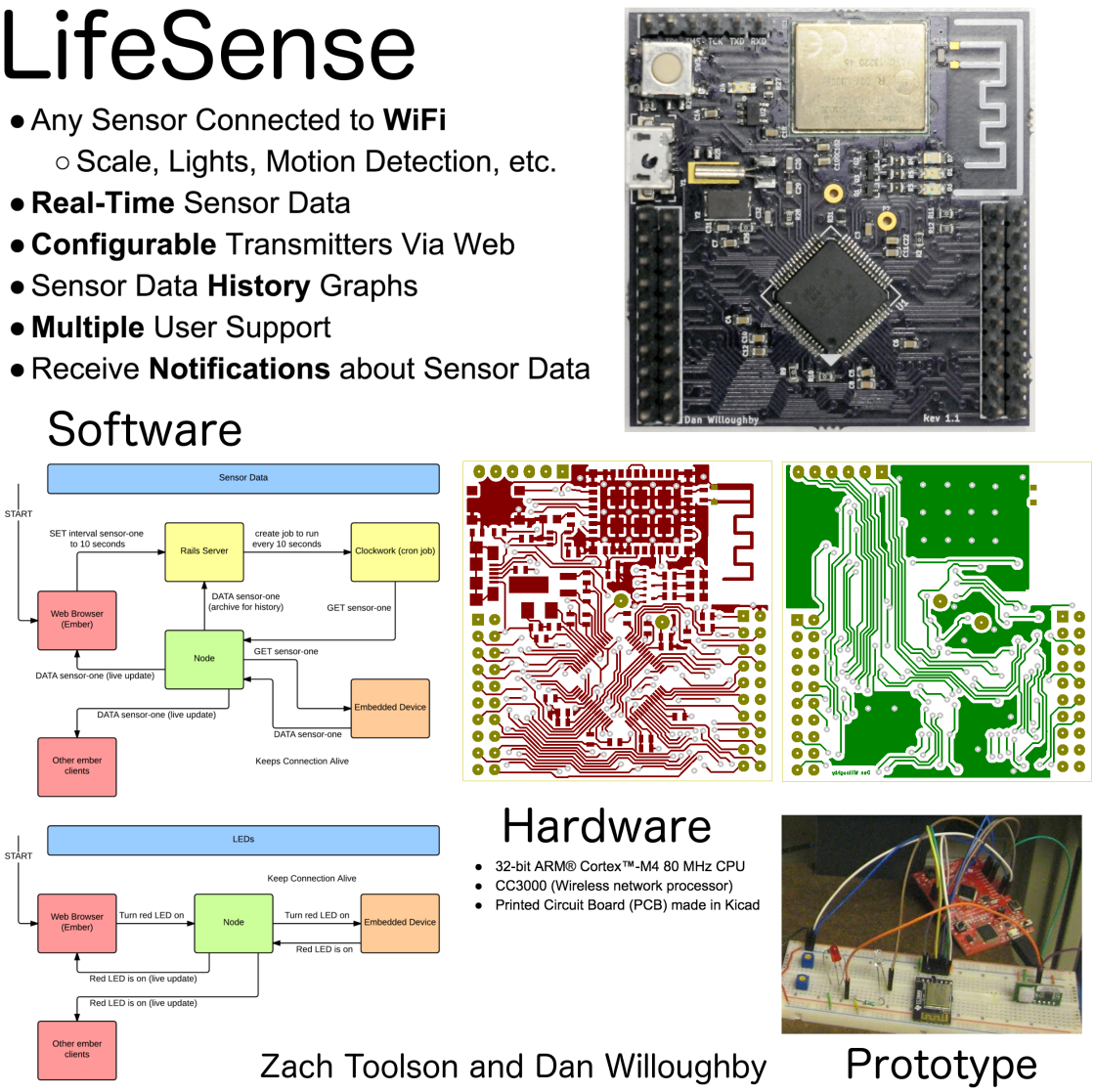

Figure 4: Poster

K BOM

Part		Price quantity	Price	Quantity	Total price
TPS73633DBV	regulator	1	$0.00	1	$0.00
AB26TRB-32.768KHZ-T	crystal	1	$0.54	1	$0.54
ABM3-16.000MHZ-B2-T	crystal	1	$1.04	1	$1.04
C7, C12, C14	1.0uF	1	$0.11	3	$0.33
C31-32	10 pF	1	$0.10	2	$0.20
C28-29	22pF	1	$0.10	2	$0.20
C3, C5, C8, C18	0.01uF	1	$0.10	4	$0.40
C4, C6, C10-11	0.1uF	1	$0.10	4	$0.40
C22	2.2uF	1	$0.20	1	$0.20
D4, D1	green	1	$0.30	2	$0.60
D2	blue	1	$0.39	1	$0.39
D3	red	1	$0.32	1	$0.32
J1, J3	header 2x10	1	$0.73	2	$1.46
J9	Mirco Usb	1	$1.02	1	$1.02
Q1-3	transistors	1	$0.15	3	$0.45
RESET SW1, SW2	switches	1	$0.76	1	$0.76
R2, R9-12, R14, R26, R29(omit)	0 ohm	1	$0.10	8	$0.80
R28	10 k ohm	1	$0.10	1	$0.10
R3-5, R27	330 ohm	1	$0.13	4	$0.52
R31	1 M ohm	1	$0.10	1	$0.10
C101	1pF				
C102	4.7uF				
c103	100nF				

9.83

Footprint	Man. Part number
DONE	
DONE	
DONE	
DONE	250R14X105KV4T
DONE	500R14N100JV4T
DONE	500R14N220JV4T
DONE	500R14W103KV4T
DONE	160R14W104KV4T
DONE	GRM188R61C225KE15D
DONE	LTST-C171GKT
DONE	LTST-C170TBKT
DONE	LTST-C171KRKT
NO	68602-116HLF
DONE	ZX62-AB-5PA(11)
DONE	DTC114EET1G
DONE	B3S-1000
DONE	ERJ-3GEY0R00V
DONE	RC0603JR-0710KL
DONE	KTR03EZPJ331
DONE	RC0603FR-071ML

Link
http://www.ti.com/product/TPS73633/samplebuy
http://www.digikey.com/product-detail/en/AB26TRB-32.768KHZ-T/535-9033-1-ND/675581
http://www.digikey.com/product-search/en?KeyWords=ABM3-16.000MHZ-B2-T&WT.z_header=search_go
http://www.digikey.com/product-detail/en/250R14X105KV4T/709-1224-1-ND/2074916
http://www.digikey.com/product-detail/en/500R14N100JV4T/709-1140-1-ND/1859472
http://www.digikey.com/product-detail/en/500R14N220JV4T/709-1143-1-ND/1859475
http://www.digikey.com/product-detail/en/500R14W103KV4T/709-1160-1-ND/1859492
http://www.digikey.com/product-detail/en/160R14W104KV4T/709-1153-1-ND/1859485
http://www.digikey.com/product-detail/en/GRM188R61C225KE15D/490-3296-1-ND/702837
http://www.digikey.com/product-detail/en/LTST-C171GKT/160-1423-1-ND/386792
http://www.digikey.com/product-detail/en/LTST-C170TBKT/160-1579-1-ND/564889
http://www.digikey.com/product-detail/en/LTST-C171KRKT/160-1427-1-ND/386800
http://www.digikey.com/product-detail/en/68000-436HLF/609-3505-ND/1487451
http://www.digikey.com/product-detail/en/ZX62-AB-5PA(11)/H11635CT-ND/1993370
http://www.digikey.com/product-detail/en/DTC114EET1G/DTC114EET1GOSCT-ND/1967041
http://www.digikey.com/product-detail/en/B3S-1000/SW415-ND/20686
http://www.digikey.com/product-detail/en/ERJ-3GEY0R00V/P0.0GCT-ND/134711
http://www.digikey.com/product-detail/en/RC0603JR-0710KL/311-10KGRCT-ND/729647
http://www.digikey.com/product-detail/en/KTR03EZPJ331/RHM330AZCT-ND/2290050
http://www.digikey.com/product-detail/en/RC0603FR-071ML/311-1.00MHRCT-ND/729791