**1.) for presentation:**

-no personal introductions at the beginning. Only "hello, I'm Justin Koehler" as I start, and when I hand off to Jason, I go "and Jason will talk to you next about whatever", and he'll go "hi I'm Jason pearson", and so on.

**2.) Slides:**

**a.) Background client** –

Our client is CEAS center, CS department, with Dr. Kapenga advising

**b.) background –**

Talk about how quick the temperature increases in servers, things can go critical in an hour. Also talk about that the rooms only have a single chiller so it's ultra-critical to have sensing. If an AC fails, 1 hour before equipment is physically damaged by heat, and aside from that, any increase in temperature greatly reduces life expectancy of equipment.

Two chillers are expected for redundancy. If they need to do maintenance on one chiller, they have to shut the whole room down. If they had two chillers, the other would (generally) keep things in check while maintenance was done. If one went bad, the other is there to pick up the slack. I'll relate it in the presentation along the lines of "that's the same reason airplanes have two engines, one can provide adequate thrust for the whole plane, but the extra is there to lighten the load and act as a backup and also increase life expectancy of both. "

Will talk about how right now they need portable chillers because the current single chiller isn't actually adequate to protect the whole room.

One of the issues is that the servers at CEAS are the servers for the whole college of engineering. If they go down, the whole building engineering campus goes down.

The current setup is not as flexible as they'd like for sending alerts. They have the Temperature @lert sensors, but for reporting and sending automatic messages, it requires one of the Temperature @lert branded servers.

Site needs to be external, that way if the network is down, the reports still happen.

We have slides about how there might be hot spots in the room, but I will talk about the main issue being that there is one chiller, and if that goes out, then just one sensor alone will do the job of alerting that the room as a whole is increasing in temperature.

**c.) design decisions –**

note that any devices will actually do the trick, but raspberry pi is the fastest, cheapest overall, greatest expandability, has the best development environment.

Arduino and msp234202343330 have awful development environments, makes it harder to develop and maintain.

d.) design decisions- sensors –

digital vs. analog – note that the digital sensors require no calibrating

current implementation is for temp and humidity only but future implementations allow other sensors.

e.) design decisions- environment

laravel vs ruby on rails, raw php, tomcat, asp.net – why we eliminated (tomcat not linux, asp.net too costly, raw php not secure enough or robust. When it came down to ruby on rails or laravel, went with laravel because most supported by community, most documentation.

lamp stack – client wanted linux based (which nixed tomcat) as client has most experience with \*nix

f.) design decisions – site hosting

client insisted it be external for reliability – offsite in case wmu network goes down

pros/cons

where we hosted. Currently at AWS, very reliable, free for a year. All other hosts claimed same reliability but offered one free month or none at all. AWS is slower on it's free tier, but current volume of data will not maximize bandwidth

g.) design decisions – improvement (new slide?)

bootstrap vs foundation vs jquery mobile. Went with bootstrap because it was easy, pretty, little to no work to make work, bootstrap offered much more styling, it's scaffolding seemed to fit in with laravel's coding easily. We really just wanted a scaffolding option and in that sense it seemed as well as foundation and jquery mobile, but I had previous experience with bootstrap so I quickly added it with no learning curve.

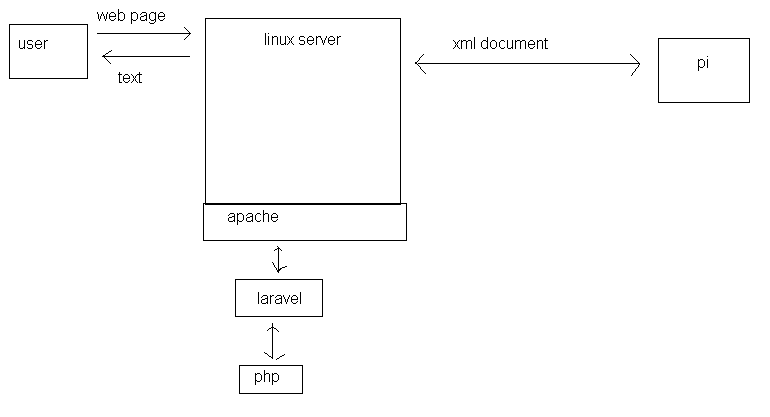
google charts vs highstocks – it came down to a late design decision. Google charts was our choice initially because of it's ease, but it required data in a very specific format, and with MySQL lacking a pivot option, it would have taken a significantly greater amount of time to code for google charts, and highstocks/highcharts took pretty much any data and ran with it.

h.) design decisions – pi driver language

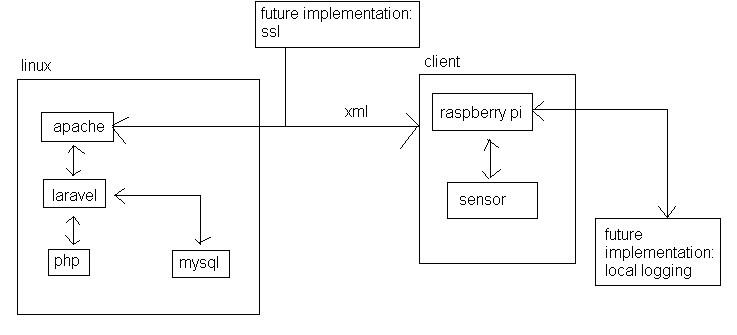
raspberry pi was designed with python in mind. The sensor didn't run well using c code due to poor third party implementation. Ran fast and reliably with third party python code though.

(get rid of web site part of the slide as it's already covered?)

i.) design – overall



j.) design – pi and server



Need to take a picture of the actual xml document (wouldn't come up for me at work)

Need to speak to release 2 – pi will store all logged data locally (has an 8gb card, can store years worth of data without any issue) and in the case it's unreachable, can communicate historical data to server in case anything was missed.

Admit freely that there is currently a small possibility of a race condition with pulling the data (haven't experienced it, but it's a very small possibility) that will be handled in the future with a semaphore.

k.) pi and server communication (left side of graph – duplicate that part)

explain that a cronjob runs at the admin's desired interval (has to be changed manually as it would be insecure to give the site sudo access as it would potentially compromise the entire server.

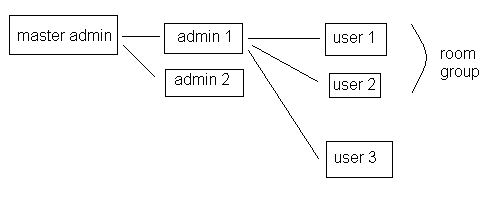
Go on to explain the steps verbatim, including sequence of cron, apache, gets xml, parses, stores in mysql, etc. leave no details out.

l.) new slide with how users work

Explain that user can set mins/maxes, etc.

MOAR SCREENSHOTS of all admin options. Also explain admins can change, users are restricted. Show all screenshots of admin stuff, but then use just one screenshot showing all buttons missing for general user.

Create a flow chart like so (goes under implementation):



Talk about text/email triggers

m.) in later releases –

have pi interface to vid cards/ cpu's, or other hardware directly to monitor those.

n.) testing – all good – no need to change

o.) security –

sql injection: laravel has a parameterized data access layer that prevents sql injection

-also using laravel's supported/recommend password hashing (show screenshot of a hashed password, maybe show screenshot of a filled in new user form and show how it's actually stored?

cross side scripting: laravel includes this prevention by default

p.) maintenance

laravel combined with phpunit is what we use for testing

q.) summary –

can add other devices than raspberry pi's

current sensors can be used with our site (not just the pi's, the pi's are more proof of concept)

talk about the central server notifying the users.