SomnoHealth Final Report

Client: Chris Crowley and Dave Tobler
Advisor: Mark Baldwin
Team Members: Amber Walker, Demeaus Wong, Reed Huang, and Tyler Braun
12/8/2020

Introduction

SomnoHealth is a company locally based out of Golden which produces products involved in the sleep health industry. Their product, EverSleep, is a wrist-strap device which tracks health telemetry overnight while the user is sleeping. This product can then provide coaching to the user to help improve sleeping. EverSleep is a competitor to products from companies such as FitBit, Apple, et cetera.

Product Vision

- SomnoHealth has found individuals lacking in motivation to pay attention to sleep coaching and consistently use EverSleep, but will do it if there is someone to manage them, such as a company nurse or healthcare provider.
- Allow health providers to manage patients and access their data in a manageable way
- Current sleep trackers like smart watches and Fitbits are managed by an individual, but the goal for this aspect of EverSleep is to develop the ability to manage sleep trackers corporately.

Requirements

Functional Requirements

- A portal for a healthcare provider to view a summary of only their patients' data
- A page where a client can click on a specific patient to see that patient's specific data and more details
- A page where our client can manually add new healthcare providers and a list of new patients (from a CSV) to the portal database
- Store patient data in groups according to their company or assigned healthcare provider.
- Convert raw data and technical terms in current graphs to common terms (provided by client)
- Implement UI/UX design provided by another employee
- Synchronize portal with a copy of the current database (eversleep-clone)

Non-Functional Requirements/Quality Requirements

- When implementing UI/UX for the portal, we must pick frameworks and tools that accommodate large amounts of data (something lightweight).

Potential Project Risks

- We may encounter roadblocks with code or technologies implemented from previous developers that we don't understand, since we don't have contact with any experienced developers on the time
- Other factors out of our control (i.e. "earthquakes" at Eversleep, internet issues)

System Architecture

Functional Interactions

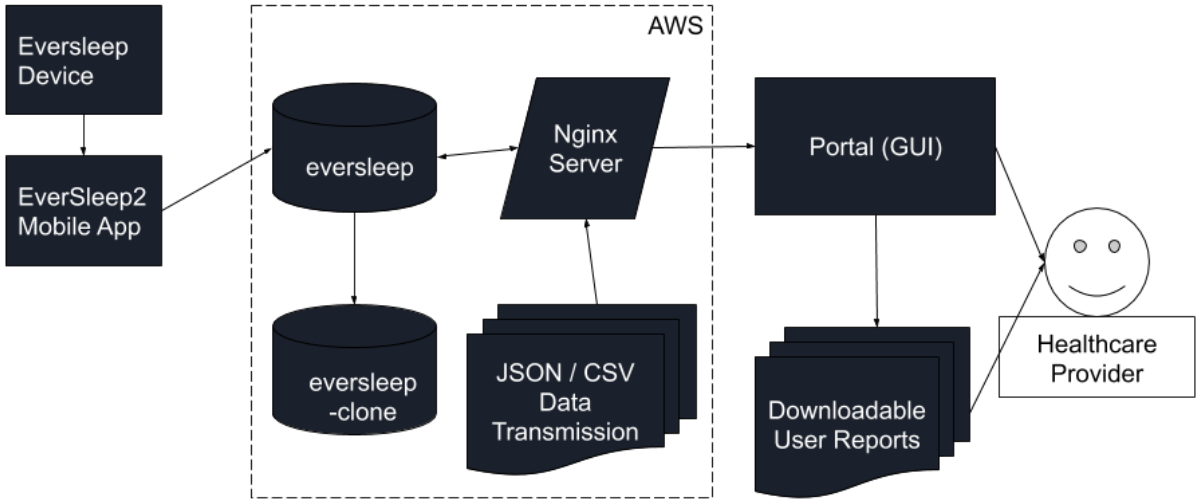

Figure 1: Implementation of Data Flow in the Portal

The Eversleep system originally consisted of the device itself and a mobile application. Later, this system was expanded, and this is reflected in the design of the many interactions between nodes in Figure 1. As the Eversleep device records data while the user is sleeping, the data is transmitted via bluetooth to the mobile device. Then this data is uploaded to the eversleep production database on AWS. The data is then cloned to our backup server, eversleep-clone, that we modify and test in development. However, at some point in time the cloning process stopped working on AWS so eversleep-clone is no longer a direct clone of eversleep, but we still use it for development. The web portal is hosted on development and production servers on AWS as well. The production and development web portals use an Nginx server and data from the respective AWS database. Then, to visit the web portal, there is a series of API requests that return the page requested. Lastly from our portal the user can view the sleep data of their devices and generate user reports in the form of a PDF or CSV to download and view offline.

Sleep Records

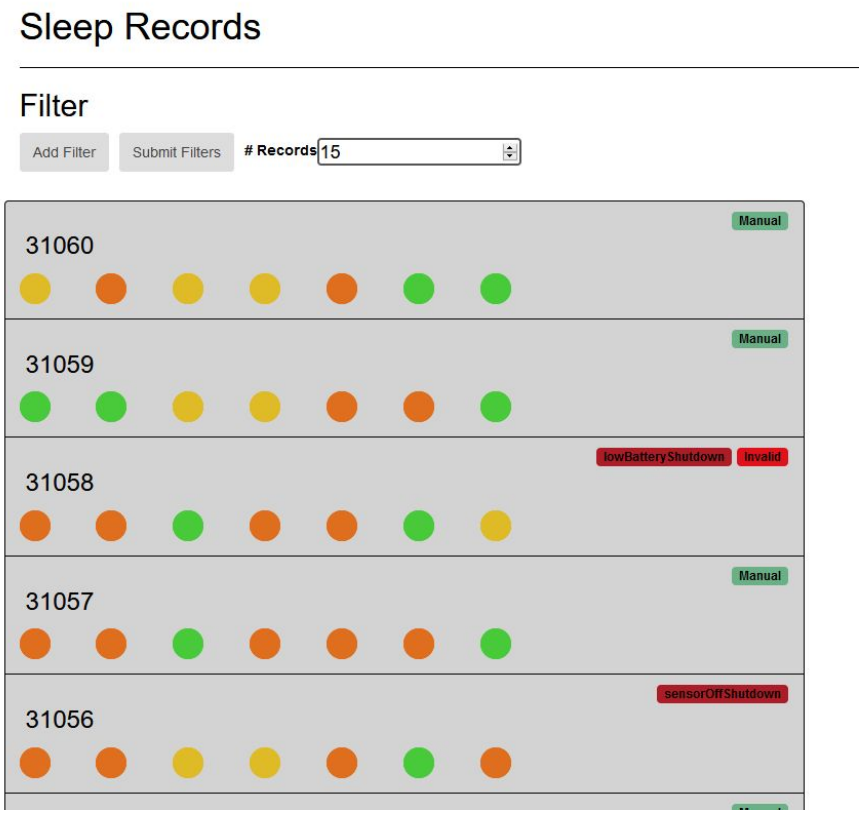

Figure 3: Implementation of fig. 2 in the Portal with sleep instances IDs.

Above, in Figure 3 the reports tab layout is expounded. As data comes in, it is parsed into separate patient profiles and presented in an easily interpretable table. Each six digit number correlates to a different night of data. With the stoplight style of the reports it's easy to see how well or poorly each user slept. There is also information about errors on the right like "lowBattery Shutdown", this shows the user that a bad night of sleep might be caused by the device shutting off and not the user sleeping poorly. By clicking on the report number, you can see the full report for a night of sleep. You can also go to the devices summary from that name.

Technical Design

Database Redesign

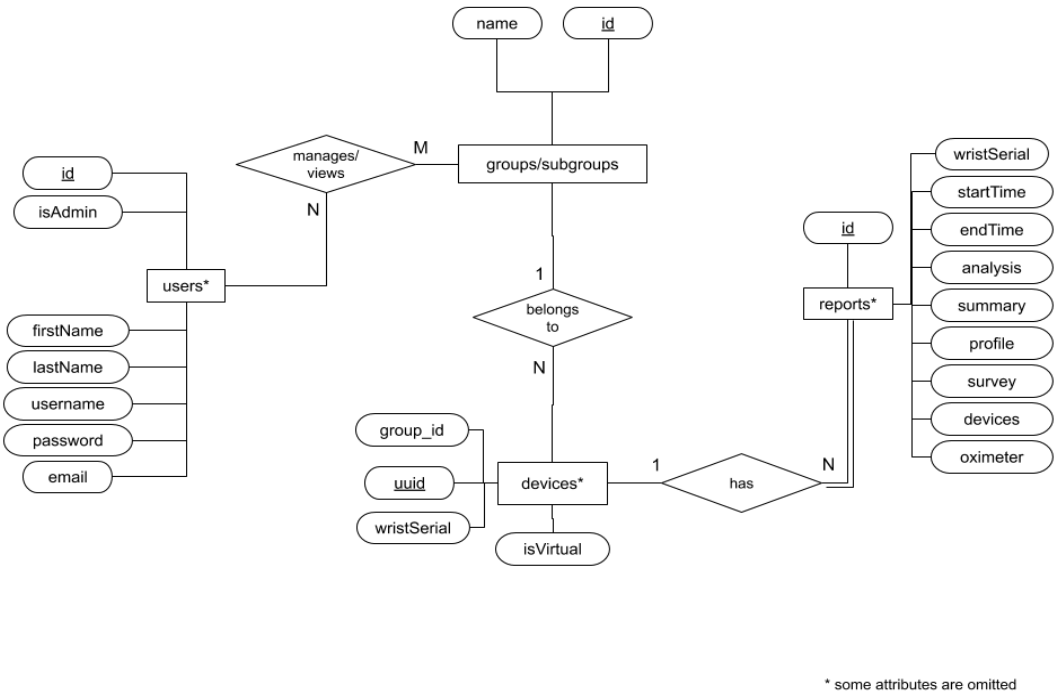

Figure 4: ERD of eversleep database

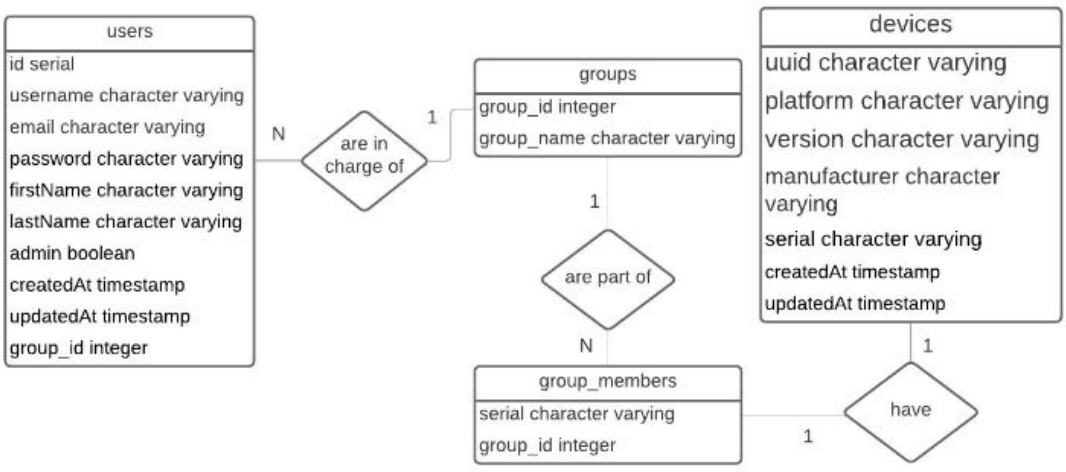

Figure 5: ERD of eversleep from a previous team

One of the major parts of our final project was redesigning the database to accommodate groups. This happened in a couple of stages. First, we created an ERD that aligned with the client's requirements for groups (Figure 4). The team before us began redesigning the database to be able to handle groups using a table that stores the groups and a table that maps devices to a group (Figure 5). We decided to handle groups a bit differently by storing the group that a device belongs to directly in the *devices* table and also by using a *users_groups_xref* that allows the database to handle many-to-many relationships between the *users* table and *groups*. In the context of the Data Portal, users are either managers of groups or admins that manage managers and groups. We made this decision so that a manager might be able to handle multiple subgroups and a group would potentially be able to handle multiple managers. For example, a wellness coach might manage multiple teams of firefighters at multiple stations or a team of wellness coaches might manage a set of firefighters. This is demonstrated in Figure 6. The many-to-many relationship allows the EverSleep team flexibility to decide how they want to store groups in the database.

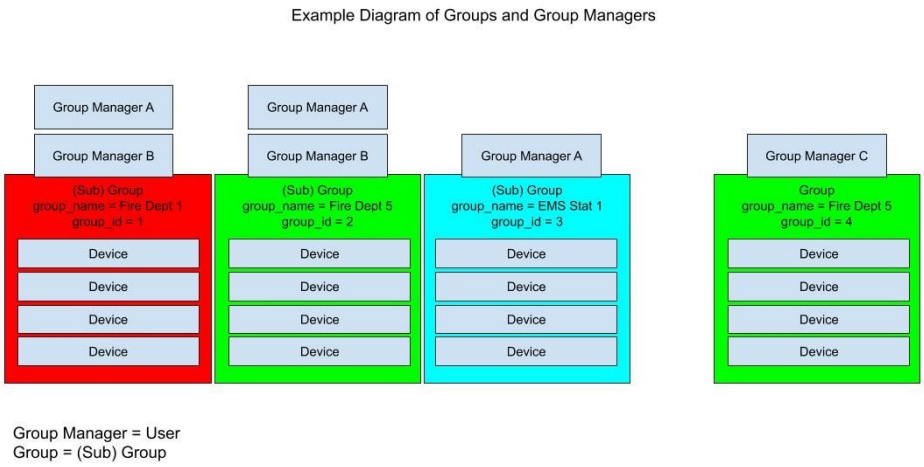

Figure 6: Example of Groups and Group Managers Use Case

After creating the ERD, we started modifying eversleep-clone to match the ERD. We removed the *groups_members* table from the database and tried to remove *group_id* from the *users* table, but encountered some issues (see Database Notes in the Appendix). We removed constraints, added tables, added constraints. We then had to adapt server and client-side code to interact with the new database structure, as well as test functionality.

The last part of the redesign will be to replicate these changes onto the production database, eversleep, once the groups functionality is completed on the development server. This has not been completed at the time of writing.

Quality Assurance

In all honesty, we did not implement as much testing as we had hoped, due to needing to prioritize other tasks, but we tried to evaluate and keep the quality of our software in other ways.

Documentation

We added more documentation to help future teams set-up their own development environment and understand the reasoning behind the existing architecture of the web portal.

Code Review

Our workflow consisted of creating branches for each independent task, making frequent commits to that branch, and then pushing that code to the remote. Before merging the branch into a main branch, the code must have been reviewed by another teammate.

Results

Accomplishments

- Created groups functionality to add/remove group, managers from groups, and devices from groups
- Changed database to accommodate groups logic
- Created the ability to assign two levels of account credentials (admin vs manager)
- Created two views for the website (admin vs manager)
- Redesigned the reports tab to show a stoplight style view

Future Work

- We need to finish documenting changes to the product as well as how to replicate changes to the production database.

List of features not implemented

- Labels for the stoplights in the reports tab
- Move PDF generation button
- Filters that limit and specify valid input
- Filtering based on groups

Summary of Testing

- Our changes to the product have been limited, so our focus has honestly been more on implementing the features wanted by the client over testing them.

Performance Testing Results

- Deployed to a production server for the client to interact with the product

Results of Usability Tests

- We tested in Chrome, Firefox, and Safari
- Received feedback from clients about design

Lessons Learned

- Documentation is extremely important. Without it, the transition time and learning curve for a new team is much longer. We struggled for a long time to understand the thought process and intention of previous teams before we could begin to code, so we learned how important it is to leave the future teams as much information as possible.
- Along the same line, making code easily extensible is very important for allowing new groups/programmers to quickly add new features in the long run.
- Communication is extremely important to be able to work as a supportive team
- Asking for help is important because another teammate might be able to help
- The client will likely change their mind and may not really know what they want until asked
- It's really difficult to guess how long a task will take because things are often out of control in terms of time.

Appendices

README*

*links have been removed for simplicity and to avoid linking to resources crucial to the project

Eversleep Dashboard Operating Procedures

Welcome to the Eversleep Dashboard! This was initially created as a part of the Colorado School of Mines Field Session Project in the Summer of 2018. With only 6 weeks, we made amazing progress, especially given our limited experience with web technologies. There are some areas that do need some love, including the deployment process. Below is a crash course on how we go about deploying code that we have developed locally, so hopefully you can follow along and deploy changes as well. If there are any questions, feel free to reach out to me at lvalentine@mymail.mines.edu, for inquiries within reason of course.

Extra resources:

- Google Drive started by the Fall 2020 Team
- Past Team Reports

How to Set-up a Local Development Environment

Run the application locally

1. If you don't have a text editor yet, this is a great time to get one. We recommend Visual Studio Code.
2. Get access to a Linux-based OS. The pathnames in the project assume you are using a linux-based machine. If you are using Linux or Mac, you should be set. For Windows, here are some options:
    - Install Linux Bash Shell
    - ssh into Mines Linux Server
3. Clone the project.
    - Navigate to a location on your machine where you would like to store the project in your terminal.
    - Clone the project with git clone https://gitlab.somnohealthinc.com/eversleep/dataPortal.git
4. Install Dependencies
    - Make sure you have node and npm installed. You can google how to check if you have them installed or how to install them.
    - Install dependencies for the project running npm i in the project.
5. Run the project locally

- Run export NODE_ENV=development; npm start
- The terminal should confirm that you were able to connect to the database successfully and the address that your project is hosted on. If you go to localhost:3000 in your browser, you should be able to visit the locally hosted project.
- To shutdown the instance, use ctrl+c
- After running the export command once, you should be able to start the instance with npm start going forward.
- There is more info on the different environments in the section about deploying to the AWS server.
-

How to View the Databases

Database Clients
The database clients allow you to read and write to the databases hosted on the AWS server. You can run queries with them to make changes to the database.

1. Visit the link above for two possible clients. In the past, some teams have used MySQL Workbench, but the team that is writing this did not have success using it. The following instructions assume you are following the instructions for using SQuirreLSQL.
2. Ask Dave for the credentials to the databases. You will need them for when you add an alias to SQuirreLSQL.
3. Find the database URL for the database you want to connect to.
    - There are multiple databases that are hosted on the database. At the time of writing (11/24/2020), there is eversleep, eversleep-clone, eversleep-clone-1, and eversleep-clone-latest.
    - The databases currently being hosted can be found here: AWS RDS
    - Each database has its own address. If you click on the database name, you will be able to see the address labelled as the endpoint for the database.
4. Connect to a database
    - If you are following the instructions from the CSCI 403 website, you will need to add a URL to add an alias.
    - Preface the endpoint with jdbc:postgresql://.
    - For example, to connect to the eversleep-clone database, the URL looks like jdbc:postgresql://eversleep-clone.cjvc6lbtyjwi.us-west-2.rds.amazonaws.com:5432/eversleep
5. You may want to add an alias for each database for convenience. However, if you are developing, do not use the eversleep database. That is for production only.
6. The databases store a lot of json files. To be able to view jsons instead of <Other>, use these instructions.

Development Workflow

- For the most part, don't touch the master branch. We used dev as our main branch and developed each feature on an independent branch.

How to Deploy to the Production or Development Server (AWS EC2 Instance)

Deploying to an AWS server is similar to running the project locally.

SSH to AWS EC2 Instance Locally

This is how you access the AWS server to update or deploy code to it.

1. Please get the csm-group.pem file from Chris or David. This will be needed to login to the EC2 and for security reasons, cannot be stored on the git repo. PLEASE MAKE SURE THIS STAYS PRIVATE. It is the only security this EC2 instance has.
2. Change the access rights to the pem file once you have a copy of it saved locally.
    - chmod 600 csm-group.pem
3. Identify the IP address of the instance that you want to connect to.
    - At the time of writing, there is a production (54.214.193.247) and a development server (34.209.240.196). In the case that the IP address or the instances change, you can check the IPv4 addresses on the AWS server here.
4. Then, you need to ssh into the domain using the appropriate IP address with the following command (with the csm-group.pem file somewhere local on your machine):
    - ssh ubuntu@<IP Address> -i <path to local csm-group.pem>

You should be logged into the instance.

Make sure the code on the server is up to date

This is pretty simple. You just need to go into the directory, and pull down your changes.

```
cd ~/dataPortal
git pull
npm install
```

Alright, your changes are pulled down, any new dependencies were installed, let's deploy it.

Kill the old deployment

1. The website is currently running on a detached process you will learn how to do later. We need to kill that old process before we can start a new one.

The following command will generate a list of the current processes that are running and their ids.

ps -eaf | grep node

Kill all the processes that are listed there (except for the color-related one). You can also learn how to kill these processes on Google.

kill <pid>

Restart the server, and detach from it

Alright here is the issue. We start the server with npm start, which as you probably know is fragile. We had 6 weeks, we did as much as we could, and if we had more time we would have found a better, more resilient way to host the website. You can specify the node by typing NODE_ENV=devAWS npm start for example, but this is temporary.

Ideally you use something like elastic beanstalk, but we ran out of time unfortunately.

Currently, npm start will start the server in the development environment by default. Here are the other environments you can manually use:

Environment	
development	Connects to eversleep-clone.
test	Not defined yet.
devAWS	Connected to an independent instance of a DB that is writable. Not synced with eversleep.
production	Connects to eversleep. Only for deployment. Ignores all packages listed under devDependencies in package.json file. Also can be used with npm start --production.

For more information, see dataPortal/config/config.json.

Note: The development mode that the app runs in by default is different from when you manually set the NODE_ENV to development. We can't figure out why this is.

tmux is used to permanently deploy by setting the bash profile of the instance. If run npm start directly in the instance, once you close the terminal the server will close.

1. Start a tmux session:

tmux new

ctrl+b (nothing changes, you probably typed it right)

export NODE_ENV=development; npm start (You can use the production or uat NODE_ENV as well, look at /config in the repo)

Hit the enter key

ctrl+b then type d. It should exit the mux shell.

Travel to (http://34.209.240.196:3000) and the site should be live. You can now exit the shell.

Going forward

As I mentioned before, this application was created in 6 weeks by a group of 4 students, three of which had never touched a database or web framework before. There are spots this website needs love, like a custom domain and better deployment process.

Should AWS restart the server like they did on Black Friday, I created a cron job to restart the server when the instance starts. If it's easier, you can pull down your changes and just restart the server via the EC2 instance, and that should deploy your code.

Database Notes

Database Notes as of 11/24/20:

- During the Fall 2020 Semester our team worked to change the eversleep-clone database to match this ERD.
- Eventually the changes made to eversleep-clone need to be replicated on eversleep.

Changes we did make:

- We added a lot of database constraints and update and delete rules.
- When it is time to update the eversleep database, those constraints can be copied by viewing them on eversleep clone and replicating them on eversleep
- The database is probably missing some constraints that would be beneficial to ensure good input.
- The client side is also missing a lot of data validation to ensure good input. Things to consider:
    - A new user is trying to be added with a username that already exists
    - A new group is trying to be added with a group name that already exists
    - A new manager is trying to be added to a group when they already are in it
    - A new device is trying to be added to a group when they already are in it
    - A manager is trying to be deleted from a group when they are the only one in it

Changes that we could not make:

- We ran into a couple of issues that prevented us from making all the necessary changes. First of all, in the users table, there is still a group_id attribute. This is unnecessary anymore because of the users_groups_xref table that keeps track of which users belong to which groups. We tried to delete this attribute and around the same time, we discovered that somehow visiting either the systems page on the 54 or the 34 server would crash the instances. We do not know if these issues are correlated. We have a temporary fix on the 34 server (see the module.exports.system at /dataPortal/app_server/controllers/others.js) but do not have permission to GitLab permission to change the deployment in the 54. If the systems page is visited on the 54 and it crashes, it will need to be redeployed

Issues with eversleep-clone:

- The devices id currently is not stored in the devices table; they are believed to be stored in the oximetry column in the reports table. The uuid in the device table isn't the actual id attached to the physical device, which is the id we want to use. As for right now, the devices table is "empty". There used to be data in the table, but, for some reasons, now nothing shows up and nothing can be accessed with our queries. At the beginning of the semester, when we visited the devices table in all of the databases, we could see the information but none of the databases appear to have data in the devices table.

- Our temporary fix for not having any (at least visible) information in the devices table is to create a new fake devices table that has basic information that we desire for implementing some of our functionalities. The table is called devices_test, which has two columns, the devices id (it is called uuid in the database but it represents the physical device id such as 'V2-1b4fe90d-b9ef-4b8d-ba16-ea79591f77ce') and the group_id. This table was created to help testing the 'add' and 'remove' device from

groups functionalities. This is just meant to help development until it can be figured out why information from the devices table is not there or appearing. As far as we can tell, no functionality related to the devices table is not working.

- This is what the devices table looked like when it worked.