MTD: MOOD TOOLKIT DASHBOARD

A Data Analytical Web Application for Psychological Research Studies

A Project

Presented to

The Faculty of the Graduate School

At the University of Missouri

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

Implemented and Defended by:

Dheeraj P. Srivatsav

Prof. Yi Shang, Advisor

April 2017

# 

TABLE OF CONTENTS

[ACKNOWLEDGEMENTS 5](#_Toc480751373)

[ABSTRACT 6](#_Toc480751374)

[1. INTRODUCTION 7](#_Toc480751375)

[1.1 WEB APPLICATION DASHBOARDS 7](#_Toc480751376)

[1.2 MOOD TOOLKIT 9](#_Toc480751377)

[1.3 PROBLEM DESCRIPTION 10](#_Toc480751378)

[1.4 PROPOSED SOLUTION FOR WEB APPLICATION 11](#_Toc480751379)

[2. BACKGROUND AND RELATED WORK 13](#_Toc480751380)

[3. DESIGN THEORY 16](#_Toc480751381)

[3.1 DASHBOARD BASICS 16](#_Toc480751382)

[3.2 SYSTEM DESIGN 17](#_Toc480751383)

[3.2.1 SYSTEM ARCHITECTURE 17](#_Toc480751384)

[3.2.2 TECHNOLOGY STACK 19](#_Toc480751385)

[3.2.3 MVC FRAMEWORK 20](#_Toc480751386)

[3.2.4 USEFUL ANGULARJS FEATURES 21](#_Toc480751387)

[3.3 DATA OVERVIEW 22](#_Toc480751388)

[4. SYSTEM IMPLEMENTATION 23](#_Toc480751389)

[4.1 Account Management Module 23](#_Toc480751390)

[4.2 File System Hierarchy 25](#_Toc480751391)

[4.3 Overview Module 26](#_Toc480751392)

[4.4 Admin Module 27](#_Toc480751393)

[4.5 Shared Services Module 28](#_Toc480751394)

[4.5.1 Separation of Concerns and Dependency Injection 29](#_Toc480751395)

[4.5.2 API Helper Module 31](#_Toc480751396)

[4.6 Studies Module 31](#_Toc480751397)

[4.6.1 NIMH STUDY 32](#_Toc480751398)

[4.6.2 ALCOHOL STUDY 36](#_Toc480751399)

[4.6.3 SLU WATCH BASIS STUDY 38](#_Toc480751400)

[4.7 File Management Module 43](#_Toc480751401)

[5. CONSLUSION AND FUTURE WORK 45](#_Toc480751402)

[6. REFERENCES 47](#_Toc480751403)

LIST OF FIGURES

Figure 1 – Data to information. 7

Figure 2 - The process of collecting data, processing raw data and then performing cleaning 8

Figure 3 - Data collection, processing and analytics. 9

Figure 4 – Existing applications. 10

Figure 5- MTD - Mood Toolkit Dashboard 10

Figure 6- Current way of analytics. 13

Figure 7 – UI Development mechanism. 13

Figure 8- System architecture of MTD. 17

Figure 9- Popularity of the languages used 19

Figure 10 - MVC Architecture framework 20

Figure 11-Data Binding. 21

Figure 12 - Module level system implementation diagram of MTD 22

Figure 13- Account Management Module. 23

Figure 14 - Directory Listing of the Web 25

Figure 15 - Overview Page of MTD 26

Figure 16 - Admin Module Features. Approve, list, delete and email status. 27

Figure 17 - Separation of Concerns principle 29

Figure 18 - Dependency Injection. 29

Figure 19 - MTD development process for each Study Module 31

Figure 20 –NIMH controller snippets. . 33

Figure 21 - Surveys Complete and Days In study Graph across all participants 34

Figure 22 - Mood change response. 35

Figure 23- NIMH participant table, on click generates the USER view with user specific data 35

Figure 24 - Alcohol Craving Study stats. 37

Figure 25 - Alcohol Study User level view 37

Figure 26 - SLU Watch Study JSON response. 39

Figure 27 - Days in Study and Surveys complete graph for SLU Watch study 40

Figure 28 - Response averages between 1-5. 41

Figure 29 - Substance usage stats. Cigarettes smoked and Drinks 41

Figure 30 - USER 2867 in SLU Watch Study. 41

Figure 31 - Daily survey compliance broken down for the USER 4489 41

Figure 32 - Daily average values of GSR, Heart Rate and Skin Temp 42

Figure 33 - File contents for USER 1510 in SLU 42

Figure 34 - Zipped file download and archived. Implemented with the help of File Management Module 43

# 

# ACKNOWLEDGEMENTS

First and foremost, I would like to sincerely appreciate the unwavering support, mentorship and guidance from my faculty advisor Dr. Yi Shang. His advisory helped me get through every phase of this research project and also my graduate school studies. I would like to thank Dr. Timothy Trull for giving me this wonderful opportunity to collaborate with the MU Department of Psychology on this project and for his continuous support, guidance and important comments at the meetings every week.

I extend my regards to every member of the Distributed and Intelligent Computing Lab for their guidance and help throughout this process. Special mention goes to the brilliant and promising Luke Guerdan, it was an absolute pleasure to work with him on this research topic and I highly value his inputs. Thanks to Peng Sun for his leadership and friendly advice.

I would do no justice if I forgot my mentor, guide and an amazing friend, Nick Wergeles for his strong guidance, selfless support and philanthropic counsel from my first semester here up until today. He has played a crucial part in my growth not just as a student but a computer science professional as well. Highly appreciate his invaluable inputs for this project.

I would like to thank Joe Guilliams for his motivation, co-operation at work and most importantly for giving me the prestigious opportunity to serve as a Teaching Assistant during my time here. The door to Jodie Lenser’s office, was always open whenever I had questions regarding my academic progress, words cannot describe Mrs. Lenser’s affable nature and thoughtful assistance towards graduate students and more specifically International Graduate Students. She was ever present and always willing to help.

Finally, I would like to express my deepest gratitude to my whole family back at Bangalore, India. If it were not for their sacrifices, encouragement and advice I would not have made it this far. Their love and support is a great driving factor towards my success here.

# ABSTRACT

“Data! Data! Data! I can’t make bricks without clay.”- Sherlock Holmes. Data drives decisions. A web application dashboard helps in visualizing data intensive application logic and provides quick access to various business metrics which can influence decision making and research topics. It also provides ease of access to data at one place. A configurable dashboard can allow various user actions and easy maintainability.

 In this research project a Web Application is developed in collaboration with MU Department of Psychology to handle large sets of research survey and sensor data. Some of the datasets under consideration include Alcohol Craving Study, NIMH Study and SLU Basis Watch study. Cleaned data converted to excel sheets add up to 500 files and amounts to 80000 data points across an average of 25 participants for each study. With large amount of data from surveys and sensors there arises a problem of access and visualization. There must be a way to structure, organize, run basic analytics and provide visualization to such data.

MTD provides a Survey schema generalization approach where common application components are identified across different studies, different metrics such as participant compliance, average mood change response, days in study etc. are calculated, analytics on various physiological features are displayed and finally the files are organized by each study on two levels, study level and participant level. Disparate datasets are used and a hierarchical visualization scheme is provided. MTD is a full stack(LAMP) development project consisting of RESTful Web services hosted on Amazon EC-2 and consumed by AngularJS front end framework, which provides modular design features. It utilizes Highcharts.js to display rich and interactive graphs.

This application is a novel addition to a psychological study pipeline from which the researchers will be benefitted due to its analytical features, speed, security and ease of access to research data.

# INTRODUCTION

Sherlock Holmes once said “Data! Data! Data! I can’t make bricks without clay.” The famous fictional detective genius couldn’t arrive to conclusions or draw hypotheses without sufficient data at his disposal. Similarly, as computer science researchers we depend upon the clay, in our case, digital data to make decisions and empower researchers. Data is the main building block of every research community, we have to use it in an optimized way to help us make decisions, solve problems and perform analytics.

“Not everything that can be counted counts, and not everything that counts can be counted.” - Albert Einstein, Physicist.

While data is important, the right data is essential. It’s becoming easier to feel overwhelmed by the increasing amounts of data being collected. Understanding what’s important to the research community, what will help making solving problems easier goes a long way in providing robust solutions and building machine learning models. As the detective Sherlock Holmes recognized, data is essential to solving problems and unlocking insights. The more useful and trustworthy our digital data is; the more impact it will have on the research process.

## WEB APPLICATION DASHBOARDS

In the early days of the world wide web, it was enough to simply have the right information displayed on the web page. The current industry-standard dictates that this process has to be more ambitious. Modern web design has moved on to support features such as utility, usability and beauty. The whole point of the web-based dashboard is that it lets us visualize the Key Performance Indicators and other strategic data for your organization at a glance. Key Performance Indicators (KPI) are measurements, that reflect the critical factors of an organization. The web dashboard allows the user to view KPIs and other critical data without delving into the semantics or the inherent working details of the source system that manages data.

A Web Application Dashboard helps in visualizing the data on a web front rather than an Excel data sheet or Access Database thus providing ease of access across platforms and also provide its users a provides a brief snapshot of the entire business and the ability to perform actions regarding the data flow, control the access levels and make sense of digital 1s and 0s in the form of graphical displays as shown in Figure 1. Dashboards have become a standard business tool over the last decade. The function of a dashboard is to communicate critical information to its audience in a way they can understand, delivered when and where they need the information. Dashboard building involves the whole process of cleaning the data, building a development stack, design principles focused on the end users, thoughtful display of data without providing excess of information, and still make it user-friendly.

Figure – Data to information. Converting data from surveys or raw input into a visual format to make business sense and provide ease of access and consolidated summary is the main purpose of Web Application Dashboard.

Web Applications are dynamic web sites combined with server side programming which provide functionalities such as interacting with users, connecting to back-end databases, and generating results to browsers. Examples of Web Applications are, Online Banking, Social Networking, Online Reservations, e-commerce / Shopping Cart Applications, Interactive Games, Online Training, Online Polls, Blogs.

## MOOD TOOLKIT

mobile ambulatory assessment system mAAS [1] is designed for clinical studies of mood dysregulation, craving, and alcohol. The system is a mobile application, which helps in collecting data from sensors, interacting with the user and collecting survey data.

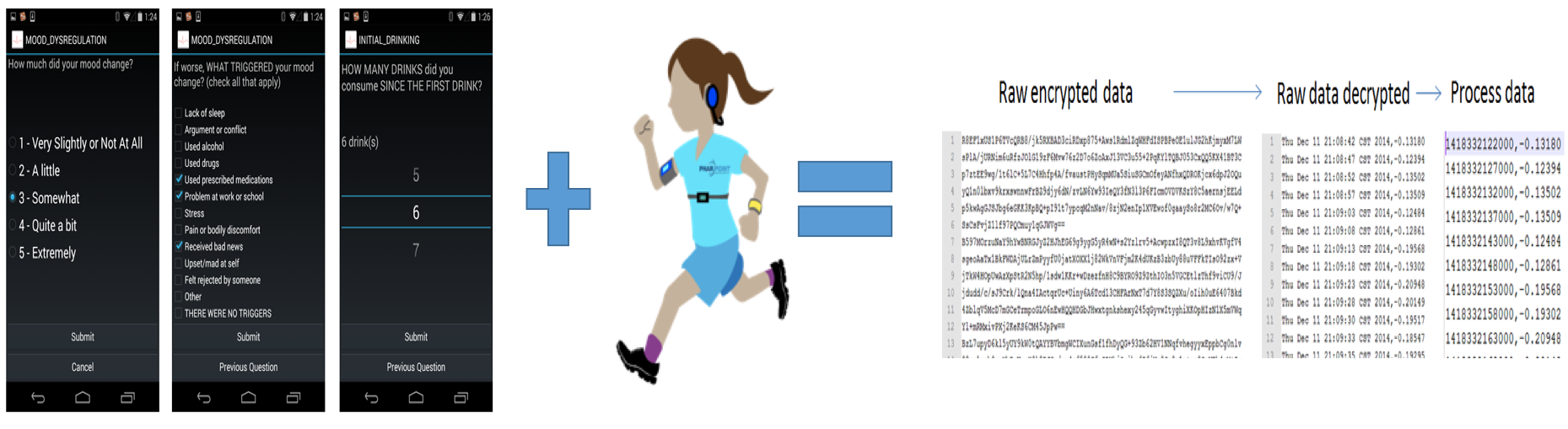


Figure - The process of collecting data from the users and processing raw data and then performing cleaning to use by researchers

Data collected by the smartphone is uploaded to the server either in real-time or when an Internet connection is established. The research lab is undertaking a pipeline where the data is cleaned, grouped across all the patients and individual files across each patient for each day in study is generated so that researchers can have access to it. The result of this generates a collection of excel files with data records from the survey and sensors as shown in Figure 2. This process has been followed for three kinds of studies which are the main focus in this project, Alcohol Craving Study (to determine the alcohol craving based on the sensor data and survey response) [1][2], NIMH (National Institute of Mental Health) funded Mood Dysregulation study and the SLU (St Louis University) Basis watch data is used for HIV study. The data collected from this is processed and segregated and is waiting for it to be analyzed so that key metrics across each study can be visualized for researchers to draw conclusions and as result fine tune this cyclic process of data collection, data processing and data analytics as shown in Figure 3. So Mood Toolkit is a product which implements all the above mentioned processes and helps the MU Department of Psychology in conducting research on patient behavior, patient health patterns, substance craving and emotion change detection.

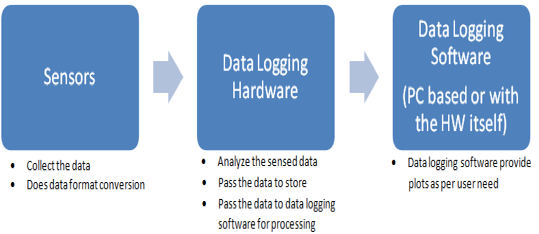


Figure - Data collection, processing and analytics. The results of this process will be beneficial for the research community who are interested in studying the trends in each study, measure certain key metrics. This process is implemented in [3]

## PROBLEM DESCRIPTION

Large amounts of data sets sit idle in the manner of Excel files without a methodology where we can easily visualize them and show certain basic metrics, perform aggregations on a study level and as well as patient level. The data available from the studies is segregated and pre-processed to be used to construct machine learning models. As mentioned in chapter 3.5 of [Nick’s research paper] the researcher had to manually graph the number of days in study for each user and then perform statistical analysis on that. This will be a hindrance if the number of users increases and/or if number of studies increases. There is also a room for error from manual based calculation. Thus, one of the main problem is lack of visualization and lack of use of this data apart from the predictive analytics (machine learning) module. As described in the [3] the web server module is accompanied with a flat file storage system, thus, without any kind of hierarchical classification of files, which would help access all the data at one place per user and study level, the data management will be a problem which might affect decision making. The Mood Toolkit currently doesn’t have a web based service to provide any kind of support for its Mobile based surveys, a visualization schema for each study will provide researchers with ample tools to watch Key Performance Metrics and then make changes to their processes based on the results shown from the web service. The Web application as shown in Figure 4 as part of the Thesis [3] lacks a modular design to support multiple studies, it is difficult to update features and follows a convoluted monolithic development approach.

The data cleaning pipeline described by [2][4] provides an approach for harnessing all the survey and sensor data into a structured file system which that can be separated on Study and user level respectively. The large amount of data from this process needs to be converted into a database accessible format for readily building a web application.

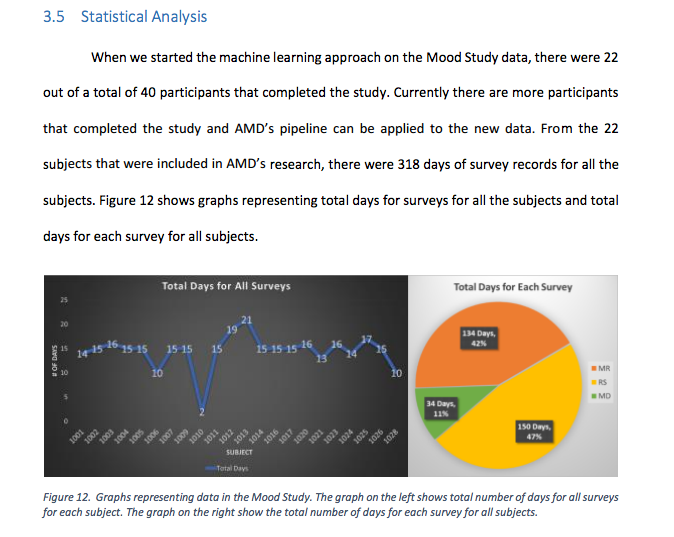
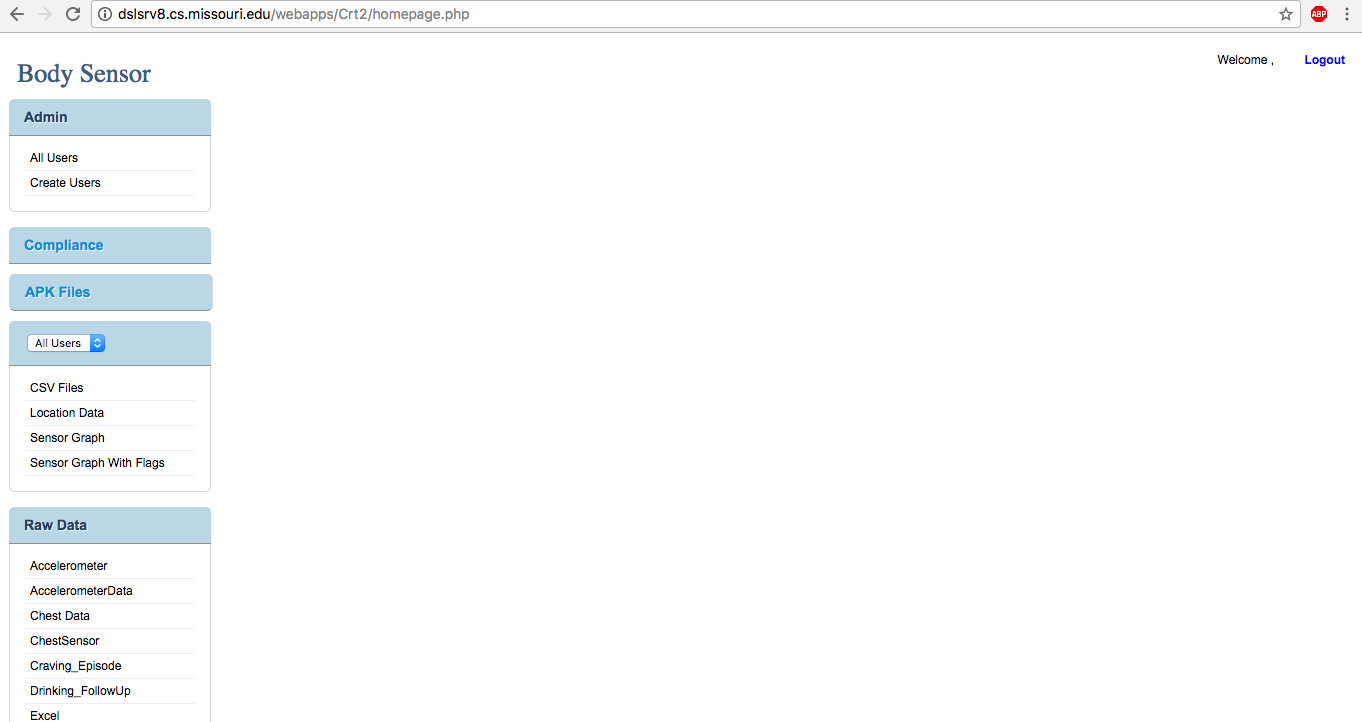


Figure – Existing applications. The left picture denotes the existing web application for alcohol craving study only. The image on right depicts manual method of calculating key metrics of a study which might be slow and can be error-prone.

## PROPOSED SOLUTION FOR WEB APPLICATION

Therefore, in order to help with the different requirements from the research community and provide an everything at one place access for the data files, a data analytical web application dashboard is proposed in this research project. The Figure 5 shows the web software hosted on the amazon-ec2 [14][15]. MTD currently handles 3 different studies. A software engineering approach was followed while implementing this project. In collaboration with the psychology department, different project requirements were discussed and implemented according to the specifications.



Figure - MTD - Mood Toolkit Dashboard, a Data Analytical Dashboard for Psychological Research Studies

Different metrics such as days in study, participant compliance, Number of surveys answered daily, mood change reported and physiological metrics are visually depicted. MTD, follows a modular development methodology by providing separation of concerns [11], abstraction and hierarchical display scheme.

This is a full stack development project, utilizing industry standard technology and frameworks for building scalable web applications and also implements unit-testing features. The development stack is LAMP (Linux, Apache Web server, MySQL and PHP) in the back end and a highly popular client side framework called AngularJS [10] which helps in building robust, single page applications(SPAs) using MVC framework [9]. The application uses a JavaScript library called Highcharts.js to build interactive graphs which convey the information in a visually pleasing and accurate manner. MTD handles Alcohol Craving Study [1][2], SLU Watch Study and NIMH Mood Study, all these are treated as separate modules, which is a noticeable enhancement to the existing web application. Around 800,000 data points across all the studies and an average of 25 participants per each study. Data aggregation is done using Python and all the specific metrics are pulled from the database to be shown on the dashboard.

Increase in studies brings the problem of access levels, MTD tackles this problem by providing restrictive access levels based on the permissions possessed by the user of the application. Each user file is available for download across their field, all their files, sensor and survey data are at one place and can be downloaded as a zipped folder a feature which saves time in downloading individual files and preserves the structured format.

# BACKGROUND AND RELATED WORK

A web-based dashboard is a web application that helps in providing multiple views of a large scale dataset. Visualization is a very important functionality in these applications as they help decision making and they become the core functionality of data driven applications. As the dataset is updated, the visualizations in the dashboard can change to reflect the updated data. Typically, dashboards are customized by developers for specific business requirements. Applications built for research faculty seldom permit end-users to customize the dashboard, make changes or integrate outside data additionally, even if that option is provided it will be limited and programmatically complex [3]

Certain wiki based dashboard systems, such as TWiki [5], included limited visualization capability. These applications include a special wiki syntax that enables the user to precisely define key business metrics such as bar charts or line plots. Data is retrieved either from a table on the wiki page or from a file on the server. The different visualizations options are limited and this drawback will be addressed in our application using a highly robust and developer friendly tool highcharts.js. Highcharts.js [6] is a 3rd party charting library which is completely written in JavaScript which is one of the main building blocks of a web application. This provides an easy way for integrating responsive charts to our web application with over 15 different chart options to choose from. It is free, open source software requiring no server side support or client side installations, just pure JavaScript implementation, making it very developer friendly as it has a simple configuration syntax [6] and meeting the end user results with rich displays over the web.

MTD is motivated by the one mentioned implemented and deployed in the paper[1][3], it is extending the framework set there to accommodate more studies and building a more organized and distributed file system unlike a flat file system for the project directory and thus makes it a modular application which will make handle changes easier, handles data JSON format unlike text based input as mentioned in Figure 6a. MTD proposes to help researchers handle research data in a way to have easier access, in reference to where the researcher had to manually plot and calculate the key metrics of his research topic[7] as shown in Figure 6b, having a easy to access, readily available Web Application with such data will be more helpful and time saving.

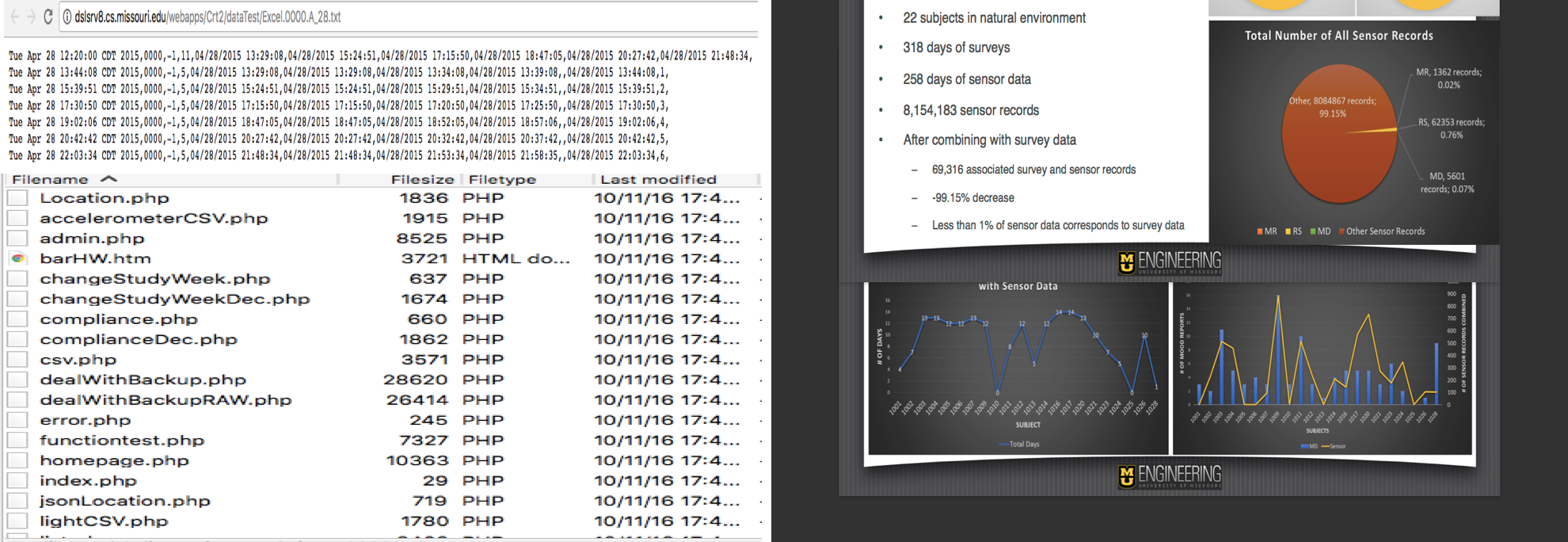


Figure - Current way of analytics. The image on the left(a) denotes the application developed using [3] approach taking in txt and csv files as input data which will be changed to JSON format. The image also denotes a flat file system with very little abstraction and directory structure. The image on the right(b) as part of [7] involved manual calculation which can be overcome using MTD approach

Prior to constructing the information display dashboard, 3 datasets were aggregated from 3 distinct ambulatory assessment studies. Each study contained survey data from each participant based on responses logged through an android application and stored on a server. In addition, some studies also had sensor data from wearable watches and body sensors. Once all data was collected and stored according to a uniform convention, the records were aggregated to form a study-wide dataset with all information for each study stored in a single file. In order to achieve this, relevant sensor data was first combined with survey data and matched over ascending time.



Figure – UI Development mechanism. The image on the left(a) refers to the traditional UI building based on server side rendering which is slow, MTD follows the mechanism suggested on the image on the right(b) with client side rendering using highly robust JavaScript features

The Classic Visualization Application Reference Model defined in [24] is a major influencing factor for the system design of MTD. The flowchart presented in the First chapter of the paper and the recommendation for the web based dashboards is a good design framework for building data intensive and database oriented Web Application Dashboards.

One of the major inspiration for the approach taken for development in this project work is from [8] where the best practices for JavaScript based UI development is echoed. Traditional web UIs which are developed as shown in Figure 7a do not provide novel features for data analysis and web application development as it comes with low interactivity and slow loading due to a full-page load for every user interaction and the requirement of a new server-side request even for a new view of the same data. Modern web UIs as shown in Figure 7b can be highly interactive and provide a much better user experience. Such UIs may be implemented with a number of different technologies and frameworks but using JavaScript and AJAX. Key characteristics of such UIs are single-page interface, data loaded on-demand via AJAX, highly modular design, easily interpretable code logic and client-side view generation. This follows the current industry standard of web application development which involves having the core of the UI code as a client-side model-view-controller (MVC) framework [9] with event-based communication between all the 3 components. This approach, the need for this approach and the advantages is detailed in the future chapters of this paper.

# DESIGN THEORY

This chapter discusses the design factors being considered and discusses the pros and cons of the technology being used and the architecture of the system. The methodology and guidelines recommended [12] has closely been followed to build this application. The features from existing technology which are being used and the customizations provided in the MTD app is explained.

## DASHBOARD BASICS

The main questions needed to be answered for this kind of an application were, how is the dashboard going to add value to the research community, what type of Dashboard is needed, who is the audience and what are the key metrics which needs to be shown. Different factors such as timeliness (how often and how old the data is), interactivity (allow user actions), data intensiveness, the kind of application (web or mobile based) and aesthetic sense(UI) were all given consideration. A flow based approach for the structure shows the change in data over time or gives a measure of different metrics over time, this forms a key component in showing what goes into the dashboard. The other structural consideration was to group related information, in this case, the studies and the users in a study into a hierarchical level. We might lose the vision or the goal of the application if we do not group the right data as that might affect the way dashboard comes out.

The major design principles [13] that were considered right at the beginning of the design of the application were:

1. Compact Design: some applications become large and difficult to handle when we embark on an effort to create a comprehensive view of the dataset or the business process. A software application must be compact, similar to the way we use a tool which only fits our hand, an application must fit into the users thought process, thus building a web app which is compact and not structured in a convoluted manner with respect to all the inherent components
2. Modularity: Dealing with large datasets like the MTD does comes with a major design problem of can the data be broken down into small sized features, as that makes it easy to scale and easy to build. Thus industry standard frameworks for application development were used and a modular approach of separating every study from each other and using a divide and conquer philosophy made the development process both enjoyable and enlightening.
3. Explanation: How much do we let the data speak for itself and how much explanation do we offer to support the data was a good data, this factor helped us pick the right metrics to show and the visualization scheme to follow. Picking the right chart type conveyed a lot of information than text. These trade-offs were well addressed before implementing a graph module. Highcharts.js is one such consideration which adds quality to MTD.
4. Interface Design: Making the application behave interactively yet keep a simplistic design is another challenge for developing web applications. Choosing the right design tools so that it allows minimum development effort and provides optimum UI design features played a great part in choosing the tech stack for the application.

## SYSTEM DESIGN

### SYSTEM ARCHITECTURE

A Web application can be accessed by the users through a Web browser. The browser creates HTTP requests for specific URLs that map to resources on a Web server. The server returns the response via a JSON object to the client, which the browser can display after being parsed by the client-side scripts. The core of a Web application is its server‐side logic. The application can contain several distinct layers. MTD follows a LAMP stack development architecture as shown in Figure 8. As shown the architecture is divided into 3 main parts, a back-end structure consisting of the file system as well, a front-end framework which is client facing and a Database server where data from all the studies reside.

5.Design%20Theory/arch.png

Figure - System architecture of MTD. LAMP stack hosted on AWS supported with AngularJS and HTML front-end technology

LAMP stack is hosted on an ec-2 instance from AWS. This provides many good features such as auto scalability, reliable network, security and web application hosting. The LAMP stack consists of a Linux machine, Apache Web server, MySQL database and PHP as the back-end scripting language which communicates with the Database and the front-end. The front-end architecture follows a MVC pattern where AngularJS provides controllers and services to consume the RESTful Web Service and manipulate the client facing UI using HTML5 directives and Materialize CSS design. Highcharts.js [6] forms a crucial building block of the graphing mechanism in the dashboard.

Using this kind of a layered architecture [19] to develop an application by partitioning the logic into presentation, business, and data access layers. Therefore, the UI is neatly decoupled from the server, which becomes a pure data source and is only used to call the restful web API. This helps you to create maintainable and modular code helps us to optimize the performance of each layer individually, thus, helping us to scale the application without much changes.

### TECHNOLOGY STACK

1. Client Side scripting - Client Side Scripting is the type of code that is executed or interpreted by browsers.

* HTML 5(HyperText Markup Language)
* CSS3 (Cascading Style Sheets) – Materialize by Google
* JavaScript – AngularJS framework by Google, provides increased interactivity and background processing, with fewer page reloads along with modular features.

1. Server Side scripting - Server Side Scripting is the type of code that is executed or interpreted by the web server. Server Side Scripting is not viewable or accessible by any visitor or general public. The code base is one ec2 instance

* PHP5.6 – Popular Web Application Development tool
* Python – for running data aggregation, analytics on all the study files and scripting
* MySQL – Popular relational Database to store all the survey and sensor data

A complete software engineering approach was followed during the implementation. The version control client Bitbucket [14] was used to appropriately version and commit the code modules. Incremental development strategy was followed, known as AGILE as per industry standards. The Figure 9 provides support to the tool selection. Figure 9a shows PHP is the most sought after language for web application development and Figure 9b shows AngularJS is the most popular JavaScript framework utilized for consuming Web services, it provides lot of modular and interesting features which are described ahead and also incorporated by MTD along with its unit testing framework Karma [16].

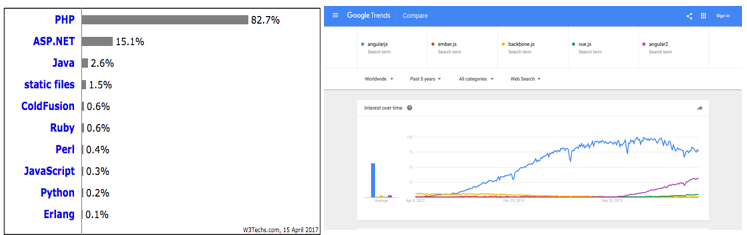


Figure - Popularity of the languages used with respect to Web Application Development. Figure on the left (a) shows PHP's latest popularity and the Figure on the right [17] (b) shows the popularity and growth of AngularJS compared with other JavaScript frameworks(Angular,Ember,backbone,vue,angular2) over the past 5 years

### MVC FRAMEWORK

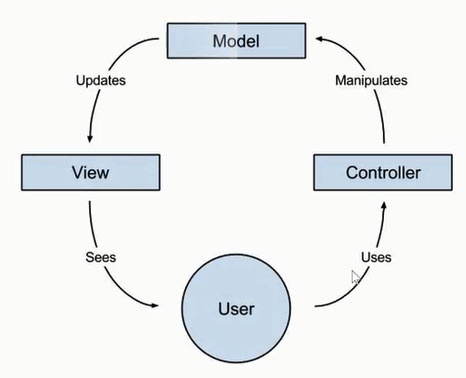
The proposed application implements a common and an effective web application framework called MVC [9]. As shown in the Figure 10 the Model-View-Controller is a modular, robust software architectural pattern for designing and developing user interfaces. The client-side MVC framework provides a configurable model and a plug-in architecture for views. The model contains the session state, i.e. user selected filters, and application data, i.e. JSON data loaded from the server. Views are responsible for updating the model state in response to user interaction, reflecting changes in the model, and updating the model with data loaded from the server, the controller acts as a medium between view and model by providing a pathway for data flow and managing the model state and the view state according to the user actions, many layers of abstraction can be introduced to perform this kind of action.

Figure - MVC Architecture framework

### USEFUL ANGULARJS FEATURES

Use of AngularJS was an inspiration for this application because of its highly modular features and the support it provides to write clean and efficient code. It allows:

* Easy handling of DOM as it runs the markup only after HTML is loaded into the DOM
* It provides a feature called dirty checking where DOM elements are checked over to see if anything has changed
* Supports the execution of Lazy Loading, where a particular call to the back-end is not made until it is necessary, thus not all data has to be loaded in a single page refresh.
* A strong backing for abstraction is provided by the feature Dependency Injection, this gives the MTD web application better modularization. [Separation of Concerns and Dependency Injection].
* Data-binding is the most useful feature in AngularJS. It saves from writing a considerable amount of boilerplate code. According to [18] a web application contains 80% of its code base, dedicated to traversing manipulating, and listening to the DOM objects. This feature as shown in Figure 11 helps in synchronizing the changes between the DOM elements and the model, thus preventing the developer from keeping track of the changes in the code.

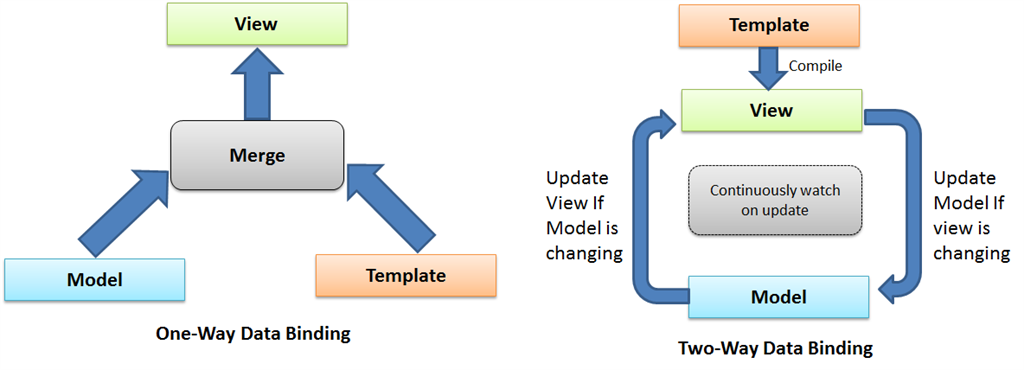


Figure -Data Binding. The figure on the right (a) shows 1-way data binding which takes a toll on the codebase and the developer, but using AngularJS's two-way data binding feature makes MTD more interactive and quick. Image courtesy DotNetCircle

## DATA OVERVIEW

The NIMH Study contained 33 raw survey files, each of which contained individual responses for each participant. The SLU Basis Watch study contained raw survey, watch sensor, and automatically calculated sleep summary data. This study was composed of 31 participants, resulting in 31 raw survey files. The SLU Basis Watch data set also had 56 subject sleep files and 363 raw sensor data files. The Alcohol Craving study contained 24 raw survey files, as well as high-frequency sensor data not aggregated and only provided for download. This resulted in a total of 507 files aggregated across the three studies.

For each study, each participant’s aggregated data was then combined into one file containing all records, with a participant ID uniquely identifying each participant’s records. Next, relevant features were extracted from each dataset and summarized to give summary statistics. In the case of the SLU Basis Watch Study, a physiological data frequency of one record per second was averaged to one sample per minute and combined with sleep summary data at one sample per minute. Because the Alcohol Craving and NIMH studies both employed the same questionnaire, both were summarized using the same procedure. Total number of positive and negative mood changes were calculated by summing mood change indication responses, each type of mood change (positive and negative) composite ratings of each cause were calculated by averaging their values across responses given the respective rating.

# SYSTEM IMPLEMENTATION

This chapter focusses on the implementation of the entire system and detailed explanation of each and every module implemented in the application. A module level design of the MTD application is as shown in the Figure 12. The following topics will discuss special features in the applications, key architectural considerations and its implementation and the overall flow of data from the database to the backend API and to the front end dashboard view.

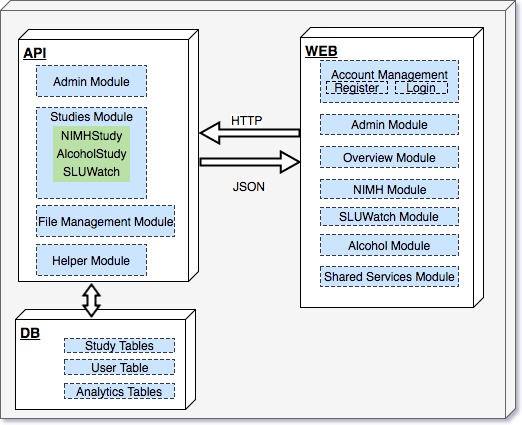


Figure - Module level system implementation diagram of MTD

## Account Management Module

This module implements the account register and login details. As the Figure 13. shows a user can register to use the application by clicking on the register link which pops up the register modal and shows a form with a set of fields. The register form asks for the First Name, Last Name, Username, email-id, a password and the University Affiliation. The university affiliation field denotes the association of the user with the Department of Psychology so that the admin gets an additional field of information before granting access for the user to use the application by using the [Admin Module]. The register form utilizes the *landingController* logic and then posts to the *userInfo* table of the database with the help of *registerHelper.php* file from the helper module in the API. A custom angular directive called *“equals”* was written to validate confirm password feature. Each password is hashed using SHA512 along with a 64bit randomly generated salt. This provides additional security as the data under consideration is non-trivial and a secure authentication and authorization is a must.

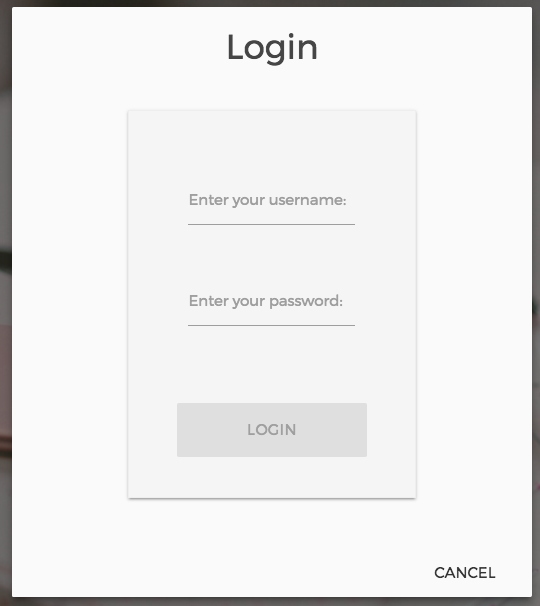
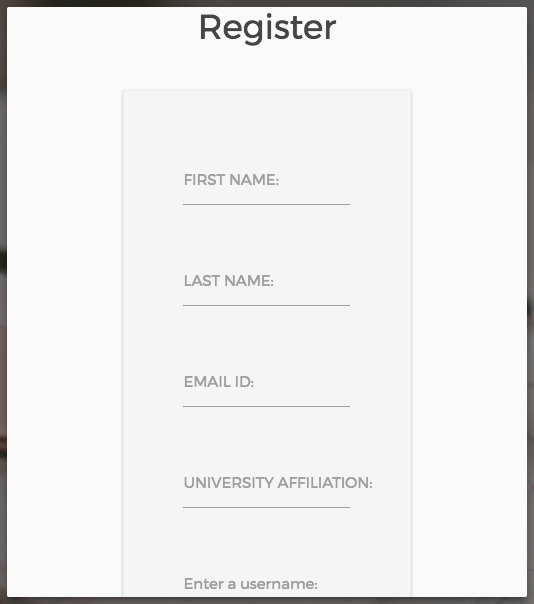
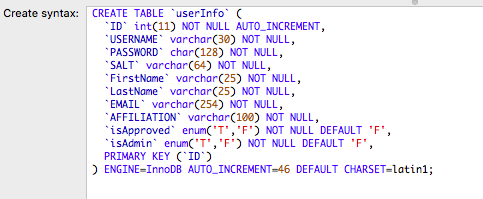


Figure - Account Management Module. Register modal, Login Modal and the Account Management Schema



A user must be approved by the admin before giving access to the application. The database schema as depicted in the Figure 13 describes this module.

The login link pops up the login modal, which is also controlled by the *landingController* logic, the login form posts to *the loginHelper.php* in the helper module and checks if the username and password match to the record in the database. This module also handles error messages in the right way, by giving appropriate error message while registering and logging in, in the form of Materialize Design Toasts. If login is successful, the user gets routed into the overview page. Which shows all the studies currently being analyzed.

## File System Hierarchy

Building modern web applications involves having a file system architecture which supports modularity, having all the components in a single file works well for small application, as the application logic grows the files must be modularized, i.e. That means the components (controllers, filters, directives, views, stylesheets) will live in modules. This allows for easy maintainability and cleaner code practice. The Figure 14 shows the directory listing of the Web part of the app which has multiple components. As seen in the figure, having a well split up layout helps in providing the feature of encapsulation for extending to other modules, provides a context for easy learning and speeds up testability.

One of the main principle of SOLID [20] programming practices is Single Responsibility Principle, similarly a module should have one and only one business logic, thus this allows for singularity in design and an efficient development practice. This principle has been aggressively advocated as one of the best practices in developing multiple page web applications, thus using it as shown in the directory listing of the web front in Figure 14a has made MTD a better designed application than that compared to the web application developed in [3] which had a file structure as shown in Figure 6a.

Figure - Directory Listing of the Web image on the left (a). Image on the right (b)app.js which controls the route and configures the application

The image on the right in Figure 14 shows the code snippet of the app.js file. This along with the *index.html* runs the whole web application. The *ng-route* [16] directive provided by Angular helps us configure the application and its routes by using the *$routeProvider* service, whenever a route change is seen, the app.js loads the required module and its components, thus preventing everything to occur on page load and embrace the functionality of lazy loading. AngularJS splits the life-cycle into configuration phase and run phase and we can provide configuration to our application via the *config* function. All the required dependencies are loaded in the *index.html*.

## Overview Module

This module is the first page or the first view a user is directed after successful log-in. The overview controller checks if the user is an admin or not before showing the admin tab. This is done by checking the status of the *“isAdmin”* property in the user metadata. This page employs a strategy of showing the current studies. As these studies are fixed and known while developing, a feature called constants, for values that do not change and do not come from another service. Thus, overview.constants.js contains the details of each study being displayed and this provider can be injected into the overview controller and then used appropriately.

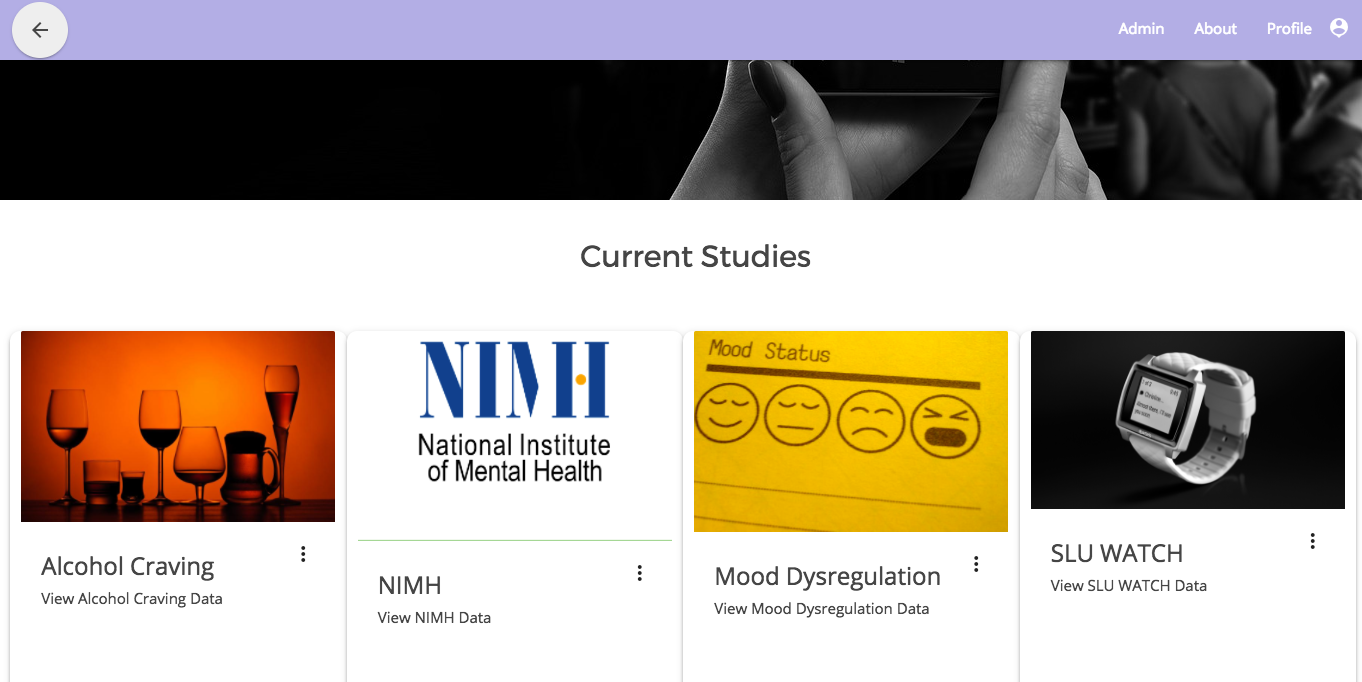
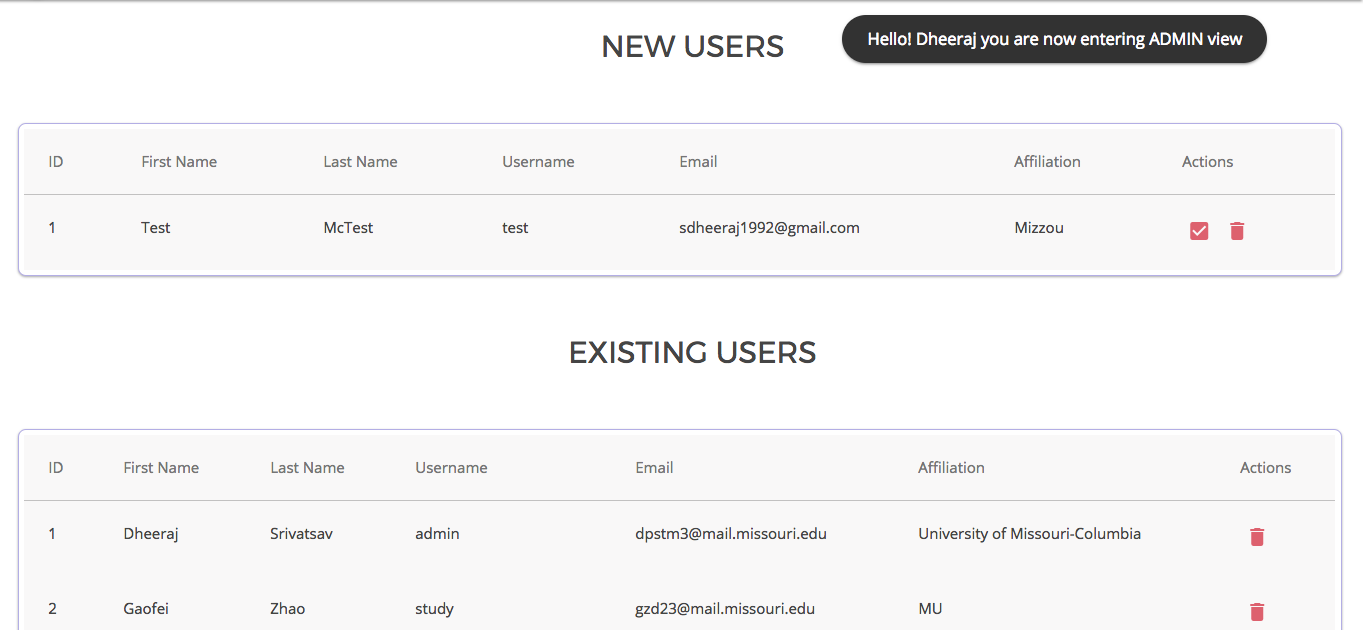


Figure - Overview Page of MTD

## Admin Module

This is one of the modules which allows configurable user actions in the MTD, this is available via the admin Tab on the overview page, clicking that will route the user to the admin page. Figure 16 shows some of the screenshots of the actions performed at admin page. Here all the existing users for the application are listed in a form of a table along with their details. Any new users are shown above the existing users and the admin is provided with two action buttons, one to approve the users and the other to delete the user from the database. This module directly interacts with the admin module at the API via HTTP requests. There are separate end points for each user action, *deleteUser.php*, *approveUser.php, getUsers.php*, which returns an JSON objects (associative array of new users and the approved users) differentiated by the *“isApproved”* flag in the *userInfo* table. There is an email service implemented as well (Figure 16) which sends an email to the admin and the user while registering is done and also when the user is approved. Further enhancements like providing separate view privileges, enabling admin access via this page are being considered/implemented for this module.



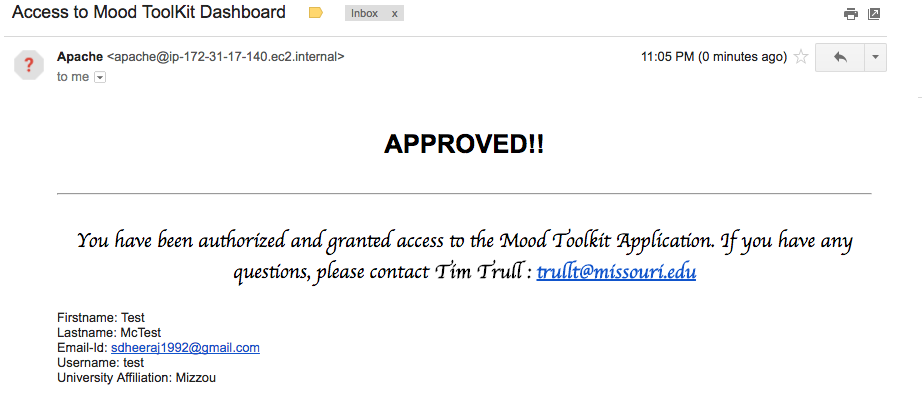
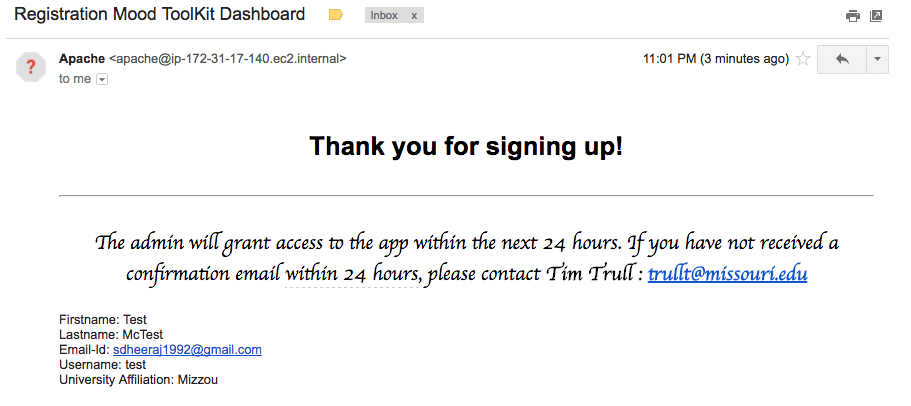


Figure - Admin Module Features. Approve, list, delete and email status.

## Shared Services Module

This is one of the key components of MTD which holds and stitches together the other components which share data across themselves and like to access the same component. As mentioned in the design theory and in the file system hierarchy, modularizing the design allows having the shared components in a separate location making it easily accessible and most importantly easily injectable as a dependency to other components.

### Separation of Concerns and Dependency Injection

Separation of concerns [11] Figure 17 denotes that a module:

1. Should be responsible for a single feature or a function
2. Has to have a unique role and handle its own actions
3. Must be located in a single file

Keeping this in mind, certain features are meant to be shared across the applications, the services located under *web/resources/sharedServices* are meant to be used as a dependency for every other module. The *highcharts-ng.js* [21] is an angular wrapper for using *Highcharts.js* [6] this was an open source solution which was incorporated to draw the charts in MTD, this makes utilizing Highcharts very easy as it provides a directive *(<highcharts> </highcharts>)* on the view which just needs a configuration object for the chart with all the properties. Utilizing this made charting very easy to develop.

The MTD application is built on the philosophy of Survey Schema generalization. This involves:

1. Finding common application components among different studies
2. Identifying shared components
3. Implementing the shared services for overall use

Although MTD uses disparate datasets, one of the main tasks involved finding similarity in different studies so the development process could be benefitted. Calculation of certain metrics such as total days in study, number of surveys taken, number of surveys completed, what is the average compliance of all the participants in the study, daily compliance etc. can be abstracted and implemented in the form of a service with various functions, this service could be then injected to a particular study module(controller) to get its own results.

This methodology was also extended while drawing the analytical graphs for different studies, for example, days in study graphs remains the same across all the studies, calculation of mood change graph remains the same across NIMH and Alcohol Craving Study, the graph which shows the breakdown of causes for mood change was studied the same way, this kind of design allowed the application to have an *aggregateService.js, graphService.js* and other shared services were created and then extended using dependency injection principle Figure 18, where a singleton instance of the service is provided for a controller to utilize. This controller will behave independently when compared to another controller extending the same service. The views will be independent as well, thus implementing such a feature has given MTD robust testing and easy maintainability features.

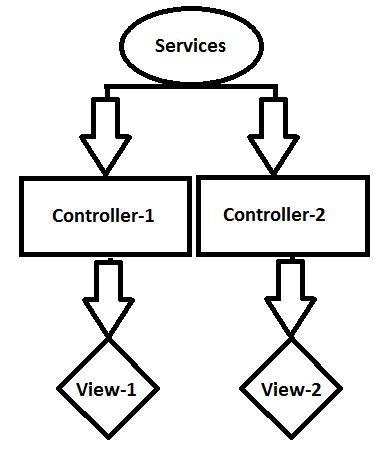


Figure - Separation of Concerns principle



Figure - Dependency Injection [22] in action. The code snippet denotes the process in SLUWatch Study.

### API Helper Module

The corresponding shared service module on the back-end which helps construct the data in a manner to be parsed by the front-end code is the Helper module, the helper module has:

1. *config.php*, sets up the database configuration and returns a connection object, also sets up query execution and parsing functionality to return results from DB in the form of an array.
2. a csv-to-MySQL python script which helps converting all the study files into MySQL database tables, the execution of this script is as in *README.md [link to bitbucket]*.
3. Other helper files which are used are *registerHelper.php and loginHelper.php.* These files take care of the logging, registering, authentication and database operations. Thus, making these files reside in a Helper Module allows it to be included in other scripts.

## Studies Module

This module is the core part of the MTD application, the individual studies described below communicates with the corresponding modules in the front-end. Thus, this approach of layered architecture helped the development process and also make the application easy to test and maintain for future changes. The Figure 19 shows the general approach going into developing each module, as mentioned before Survey schema generalization technique allows to approach each study in the similar manner, of course the problems faced with the disparate datasets makes the development process challenging.

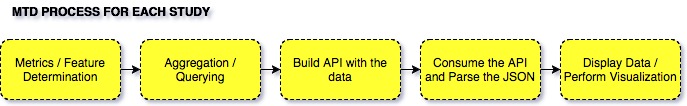


Figure - MTD development process for each Study Module

At first, the requirements are collected from the administrators who determine showing what kind of metrics makes the study more meaningful, the research community also state their inputs on the kind of key factors which has to be displayed which might help their research. Then, based on these inputs, the analytics are performed on the dataset, the results are queried from the study table or the stats table for each study. These results are used to construct the response in a format where it will be easy to parse in the front-end and also have a hierarchical approach of showing data, features which are meant to be shown on a study level are added as properties, the one which needs to be shown at user level are appended to the user property. This response generated in JSON format needs to be consumed using the angular services in the front-end whenever there is GET request. Once the response arrives as a promise, the controller for each study will use it as and when required and generate the visualizations and display which matches the original requirements. This approach has allowed the metadata generation for each user for each study to be stored, this means that further requests for additional features has a working blueprint, which can be followed and the requested feature can be appended as a property to the response object.

Next, different study modules implemented along with the challenges, approaches and results from the dashboard are shown.

### NIMH STUDY

This study funded by the National Institute of Mental Health(NIMH) involves studying for Mood Change pattern and change in behavior of a participant. The database description of this study involves 2 tables, *nimhTest* and *nimhTestStats*. The *nimhTest* has over 5200+ records of cleaned survey data. With 148 features for each record. With these many columns the querying and aggregation needed to be optimized. Using appropriate python libraries and SQL indexing helped get all the required metrics.

The major metrics which was required were total days of the study, average participant compliance, total missed surveys, total surveys taken, days in study for each user, user compliance, start and stop date, total survey count, negative and positive changes, etc. All this information was collected from the administrators, the approach of retrieval was discussed and the one which made application more stable and structured and the one which was development friendly was chosen to get the data and build the NIMH API.

Number of participants in this study were 33. The study overview stats such total days and number of completed surveys, participant compliance were calculated using querying method as opposed to done manually in [7]. The field *surveylabel* with a value “Suspension” meant that a user didn’t take part in that survey prompt. This result counts against the user during compliance calculation. The *nimhTestStats* table is a consolidated table of all the individual statistics of every participant in the study, it mainly contains the mood change stats and the reason for mood changes with the values aggregated. Some of the querying involved using DATE functions on MySQL as the format in the csv file was not human readable date format. Grouping Date related queries in one single query helped get all the results in an efficient manner.

API building involved executing queries and constructing the result object as shown in Figure 20(c), using inbuilt PHP functions helped parse through query results and generate a JSON object.

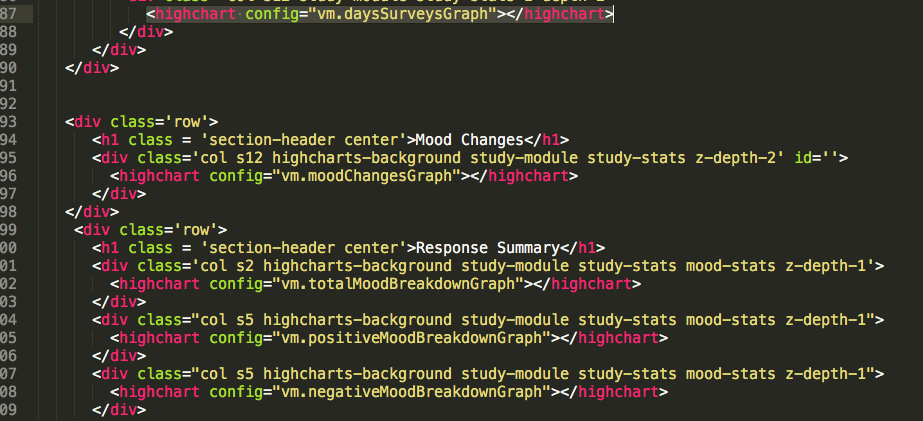


Figure –NIMH controller snippets. Clockwise. (a)Service oriented functionality in the controller (b) Highcharts directive from angular in use (c) JSON response from NIMH API .

Initially, relay of this object across the http request had a lot of delay, thus an alternate approach of using writing the response to a cached json file and updating the file by periodically running the PHP scripts helped reduce the time of transfer by 60%. This was a vital design consideration which affects the running of the application in a positive manner.

As each record in the table corresponds to a survey response, by brute force approach the average responses per each patient should be around 150. This is accurately displayed and verified with the help of days in study and survey responses graph. A couple of users had a not so good response rate. Result of this, the researcher can easily ignore this participant responses during analysis.

Similarly, graphs for total mood changes, the breakdown for number of positive and negative mood changes, the reason for mood changes all are shown on the overview page. Some of the graphing and aggregation on the front end involved implementation of Dependency Injection [22]. The way this was done by using *graphService* and *aggregateService* is shown via a code snippet in Figure 20b. The view for NIMH module has a highcharts directives which took the configuration objects as the input and rendered the interactive graphs on the screen. Figure 20b

Different graphs shown in this study, some of the screenshots are shown below.

1. Days in Study and Study Response Graph. Figure 21
2. Mood Changes. Figure 22
3. Response Summary
4. Overall Mood Change Data
5. Positive Mood Change Triggers
6. Negative Mood Change Triggers
7. Overall Mood Change Data – User Level. This appears on click of each records in the List of Participants Table.

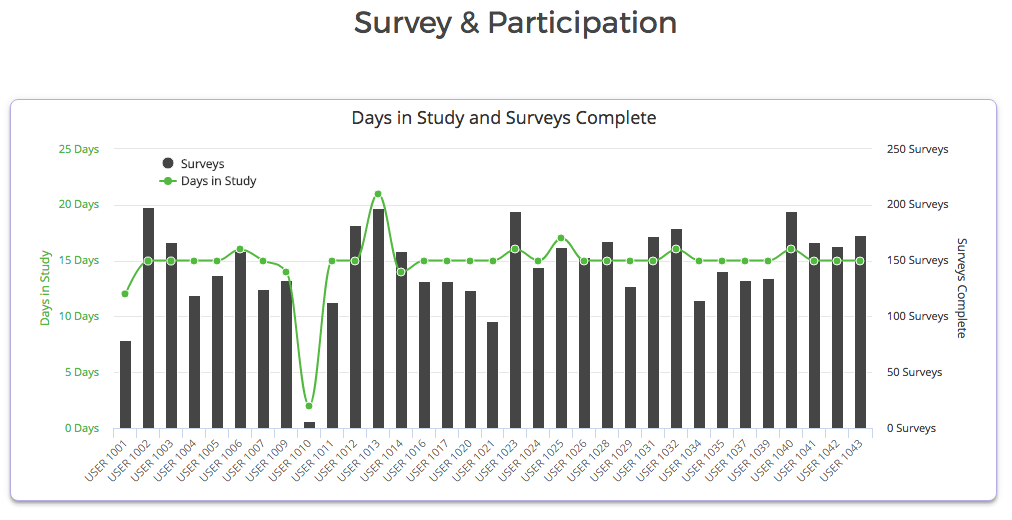


Figure - Surveys Complete and Days In study Graph across all participants

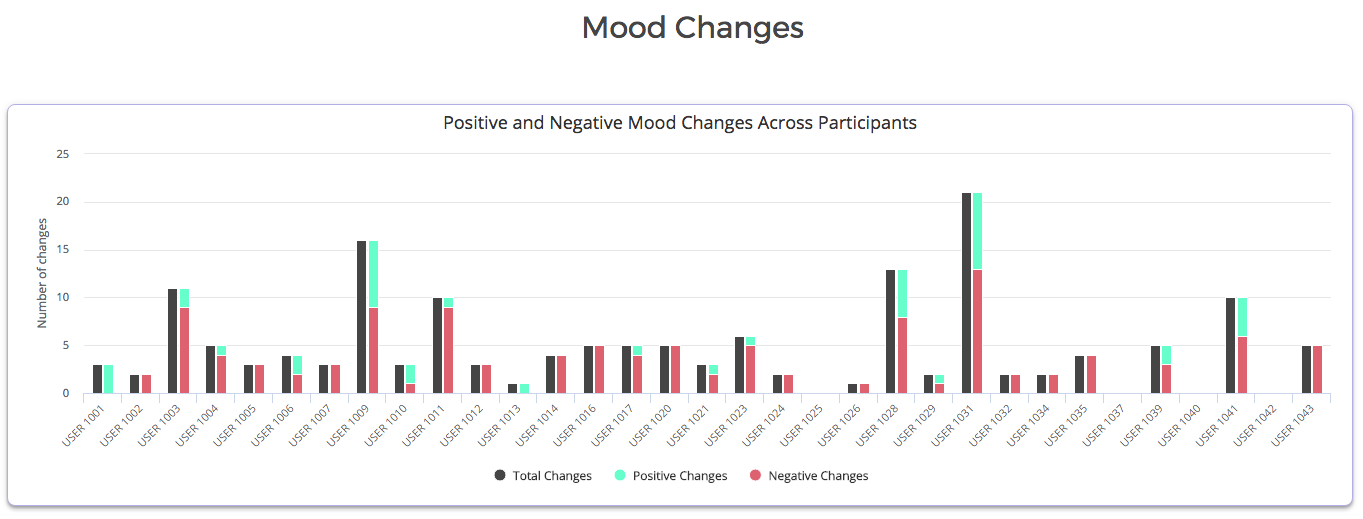


Figure - Mood change response. Total denoted by black, positive denoted by green and negative change denoted by red

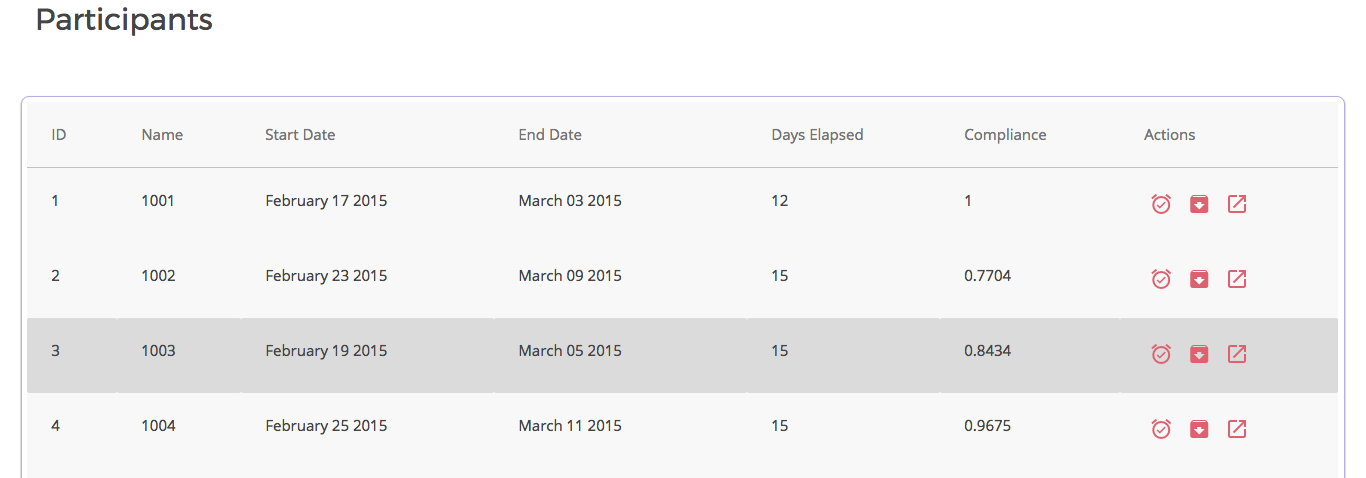


Figure - NIMH participant table, on click generates the USER view with user specific data

Each participant in the table Figure 23. has been provided with 2 action buttons, one for data download and the other to link to the user level view. The user level view is set up in a way to provide more individual level information about each participant.

### ALCOHOL STUDY

The Alcohol Craving Study involves the similar type of questionnaire as the NIMH Mood Dysregulation Study [7], thus a similar approach was used for summarizing the data on the dashboard. This is where the survey schema generalization came in very handy, the common features and the similarity of metrics being studied made the implementation of this module the quickest among all the three study modules.

This study composes of the *alcoholStudy* and the *alcoholStudyStats* table in the MySQL database. The schema was almost similar to the one in NIMH but the number of participants and the data collected differed, which meant making changes to the API development.

The number of participants, in this study were 24. The total number of data points or records in the table was 13,164. All these data points correspond to the survey from every user. This data is after the cleaning up of the combined Sensor and Survey data. The number of columns in this study is 192. The key metrics which was being considered were total days of the study, average participant compliance, total missed surveys, total surveys taken, days in study for each user, user compliance, start and stop date, total survey count, negative and positive changes, etc. The same aggregation principles were used, some of the querying were different because of the data being different. Thus, having a working framework in the form of NIMH helped a lot in building this study module. Using SQL aggregation functions and nested queries helped get Total days in study, total surveys and missed surveys. The ratio of total completed surveys over total surveys gave us the compliance for each user.

The JSON return object looks similar to the one in Figure 20c. One of the major testing tools used to verify if the JSON object constructed was correct was an online JSON verification tool called *jsonlint [23]*. A highly beneficial tool, used extensively during [SLU WATCH BASIS STUDY]

The front-end controller logic makes the GET request call for the json response once the route changes to *“alcoholStudy”*. This is followed by parsing the response and splitting the individual properties into separate arrays so that aggregation in the front-end using the *aggregateService* is possible. All the stats Figure 24a are rendered on the page along with the similar graphs as shown in Figure 21 22 23. One of the graphs involves a pie-chart which gives a breakdown of all the different reasons causing the mood changes as received from the survey. Different causes are totaled and the percentage of each reason is shown in the form a pie-chart as seen in Figure 24b.

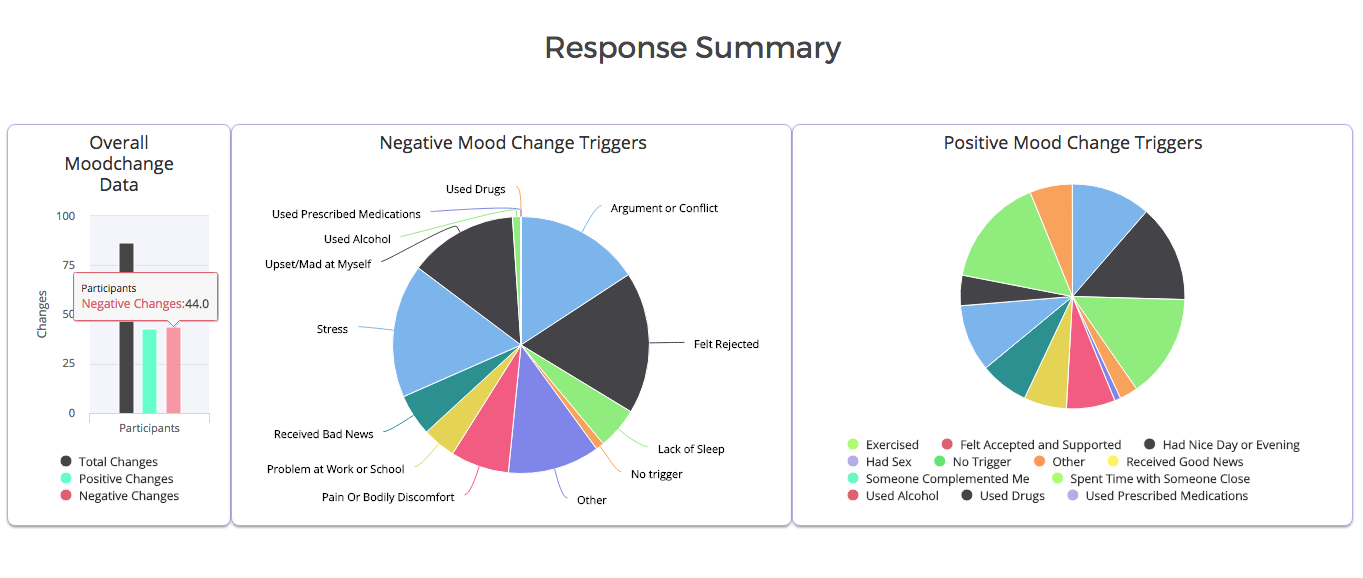
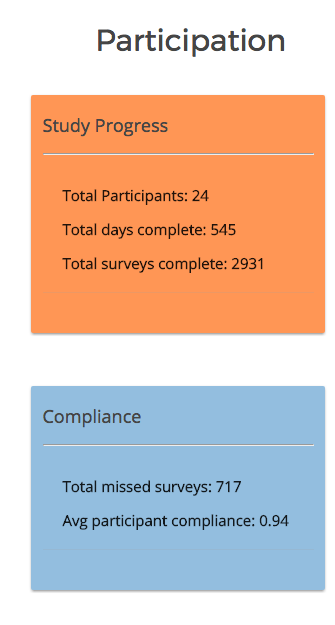


Figure - Alcohol Craving Study stats. The image on the left (a) Shows Participation on the overview page stats. The image on right (b) Shows Mood change response causes as a pie-chart

The users table has a download raw file option which is in the csv format option for every user. It also has a link to the user level view where the stats and graphs for that particular user is shown as seen in the Figure 25.

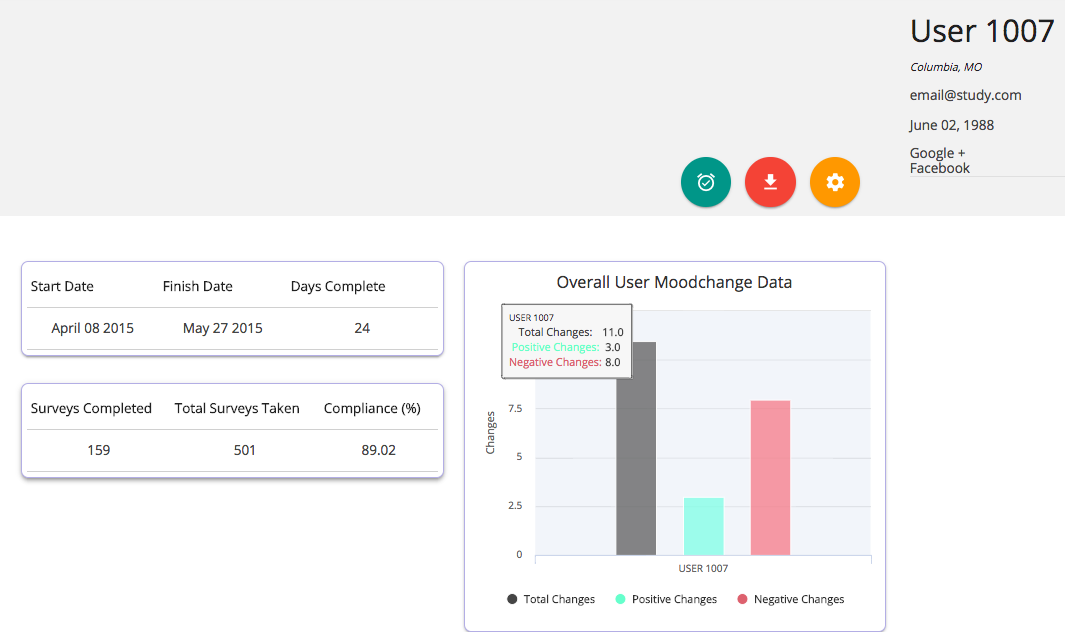


Figure - Alcohol Study User level view

### SLU WATCH BASIS STUDY

This study was conducted using a watch which acted as a sensor for calculating physiological data accompanied with a mobile based survey. The SLU Basis Watch study contained raw survey, watch sensor, and automatically calculated sleep summary data. This study was consisted of 31 participants, resulting in 31 raw survey files. The database tables for this study are *sluWatch* and *sluWatchStats* The SLU Basis Watch data set also had 56 subject sleep files and 363 raw sensor data files.

The number of data points in the *sluWatch* table is 722,731 records of consolidated survey and sensor data. The fact that this has a combined data set made the development process slightly complicated because of the additional variables and the increase in the different metrics being calculated. The number of columns in this study is 124. The metrics being studied were:

1. Compliance – How many surveys are completed for every prompt
2. For the study as a whole.
3. For every participant.
4. For each day in study for every participant.
5. Days in study
6. Missed surveys and surveys complete
7. Average response for mood change – positive, negative and impulsivity
8. Physiological Stats for each participant
9. GSR – Galvin Skin Response – measured in (μS)
10. Heart Rate – measured in bpm
11. Skin Temperature – measured in °F
12. Substance consumption stats
13. Average usage of substance (drinks and cigarette) for each participant
14. Total quantity of consumption for each participant.

The sheer quantity of data requires MTD to have an efficient architecture to render the display, all the above mentioned metrics need to split into overview level and user level in a way it makes it possible to use all the front-end services written in *sharedServicesModule* to be effectively used.

The table *sluWatchStats* has 756 rows of consolidated, aggregated data, unlike the stats tables for other 2 studies this one is bulked up because the original data’s sampling rate was 1 record per each second, this was averaged to get *sluWatch* table with 1 record per each minute by [link work], thus while aggregating we got 756 rows having daily records for each participant, which approximates to 25 days per each participant which is accurately captured in the Days in study graph in Figure 27.

Using these two tables all the features were derived using queries, some of them were nested queries, thus SQL performance became an issue. Querying in a way to group by dates meant the query execution was longer, but grouping the fields by user and then by day and then by hour gave quicker results. This still wasn’t as quick as we anticipated it to be as there were lot of SQL aggregation function in use. Thus, indexing the table by the *participant* and then by the Dates made query execution extremely quicker, for example the queries to get the daily and hourly physiological data individually took about 3-6s, after indexing it came down to around 750ms, that is a roughly 400% decrease in SQL execution time.

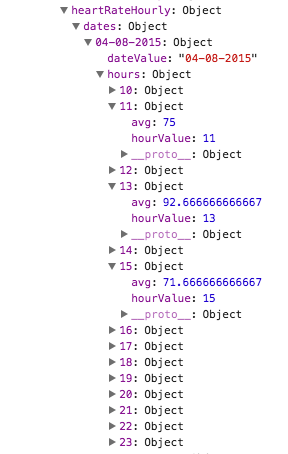
 

Figure - SLU Watch Study JSON response. For participant ID 1510, the image on the left shows all the various features chosen and stored in json format, the image on right shows the nested formatting of daily and hourly metrics of Physiological data

Subsequently, all the queries were built and optimally executed using PHP. Functions were written to separate and structure the physiological data per user on an hourly and daily basis.

The response object was constructed as shown in Figure 26 and returned using the similar technique for a GET request on page load via a JSON file.

The controller at the sluWatch study module checks for every property being loaded in the response object and then parses through it by creating separate storage data structures which will help in drawing the graphs. The highcharts graphs has a series property which takes the arrays/lists for x axis and y axis and draws the plots accordingly. Thus, parsing the data this way makes every graph tell the required story in an analytically convincing manner.

This study too provides action for viewing user level data in the table consisting of participants and allows download of a zipped file containing all the study files having raw data in csv format for every user. There are a lot of files per each participant, counting up to 500 for this study alone. This is one of the key features in this module, implementation of this feature is explained in the description of Chapter 4.7.

The following screenshots show the JSON response being parsed, the graphs being drawn using Highcharts.js and the hierarchical classification of data in both study level and user level with more emphasis placed on the user level view for this study due to the cleaned sensor data available.

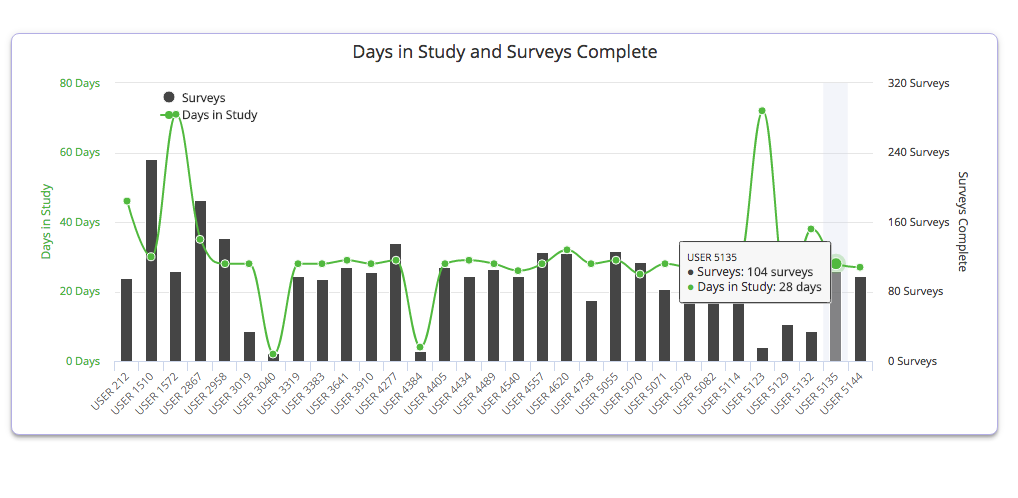


Figure - Days in Study and Surveys complete graph for SLU Watch study

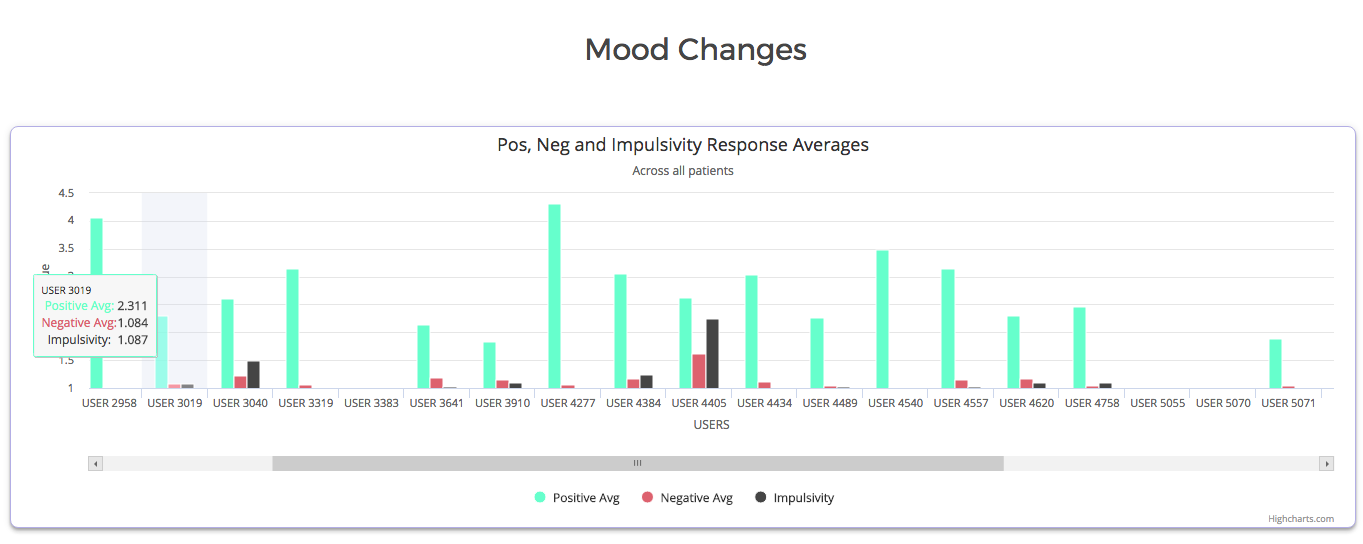


Figure - Response averages between 1-5. Positive, negative and impulsive responses received via the survey

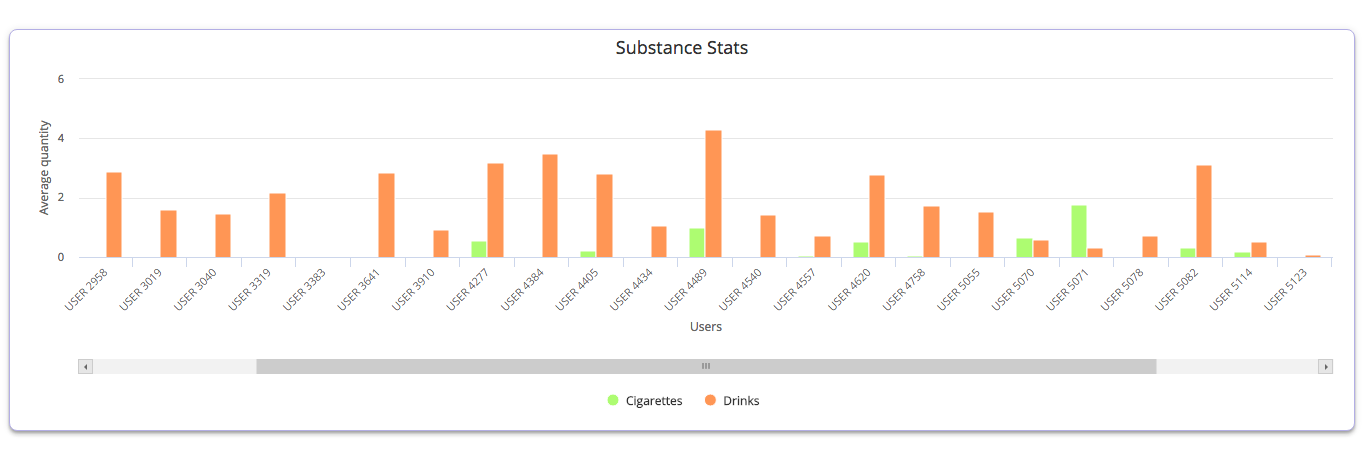


Figure - Substance usage stats. Cigarettes smoked and Drinks consumed average values across all user in the study

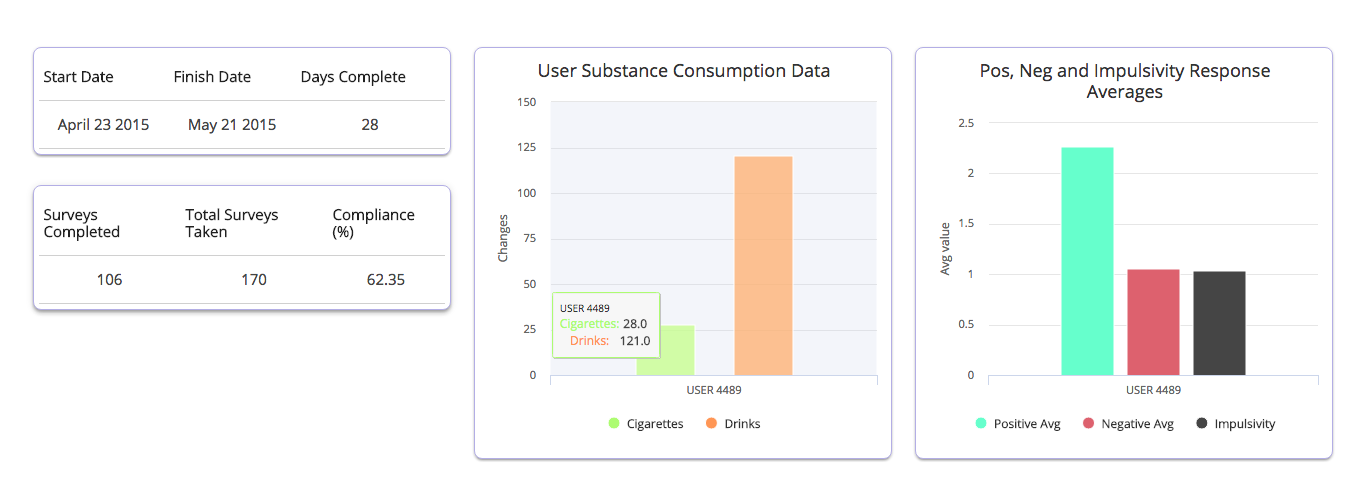


Figure - USER 2867 in SLU Watch Study. Individual statistics regarding survey completion, total substance consumption and mood change response average

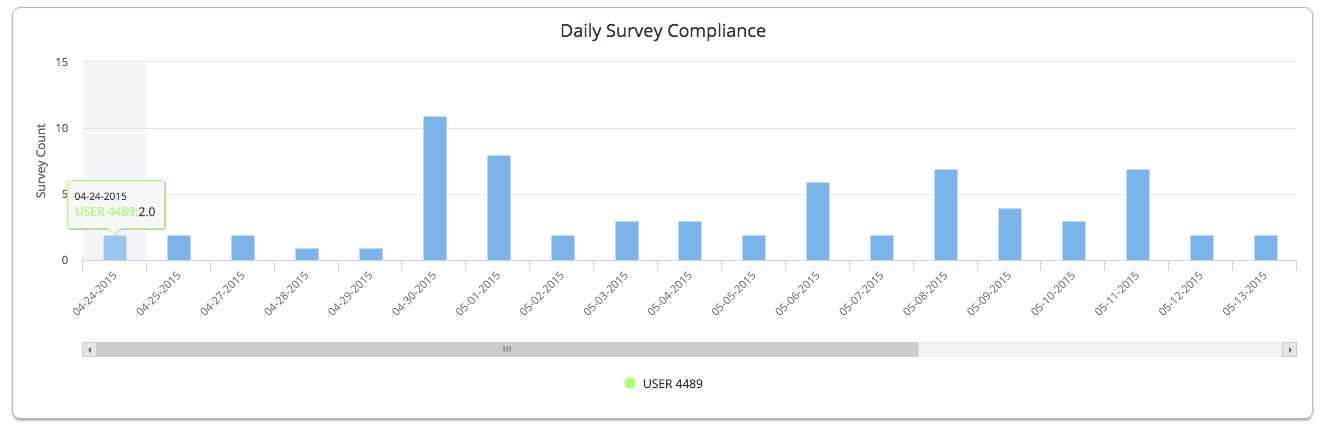


Figure - Daily survey compliance broken down for the USER 4489

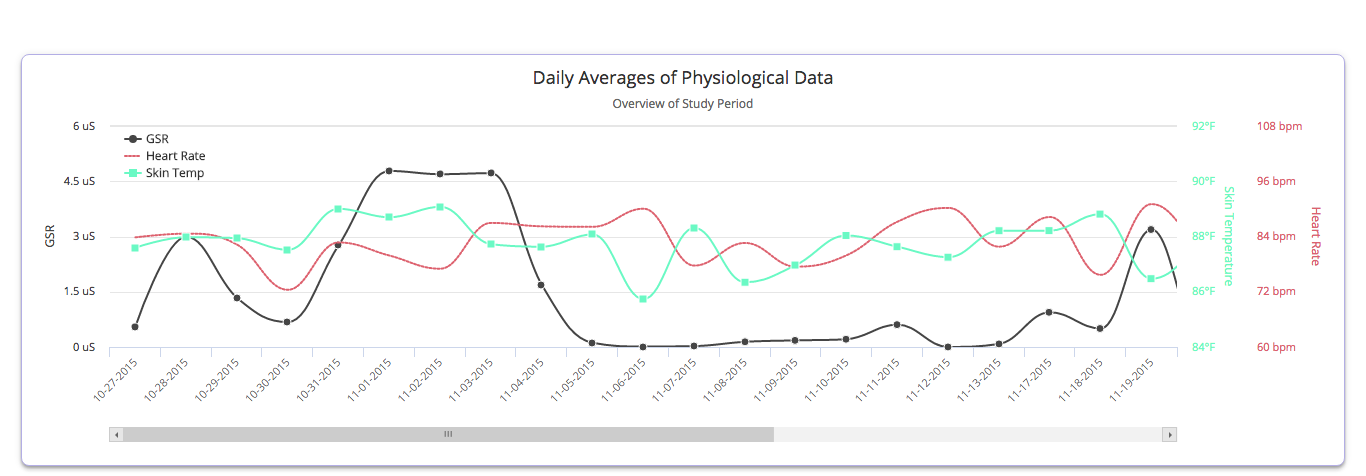


Figure - Daily average values of GSR, Heart Rate and Skin Temp

## File Management Module

The total number of raw files in Alcohol Study is 24 and that in NIMH is 33, as these amount to just the csv files of every participant’s survey data. But with SLU Watch Basis data we considered Sensor + Survey data, this amounts to 500+ files as it has sleep summary, sensor data for each day and corresponding survey data. As mentioned in Chapter [1][2], data access is one of the major problem, centralized data access is what MTD offers, all the respective data files available to access at one place. The download option on the overview and the user page is the result of this module, on download we get the file stored in the file system on the server.

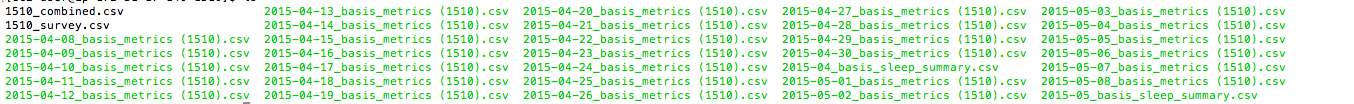


Figure - File contents for USER 1510 in SLU

The back-end project has a *zipHelper.php* which handles this functionality of zipping all the files in a folder and then making it available as a .zip extension in the front-end. This helps SLU Watch study tremendously, but the module was implemented to help all the studies even in the future and for this to work effectively. The script checks for additional files added to any folder and then appends that file to an already existing zip folder. Thus, preventing the overhead of zipping all the files every time a new file is added. The only requirement is the files need to be added to the correct folder and the path specified must be cross-checked. This module utilizes inbuilt PHP classes *ZipArchive* and *DirectoryIterator* to recursively find the .csv files only and then convert them into zip. This script is placed in a cron job so that an addition of a file will be taken care when this job/process gets executed.

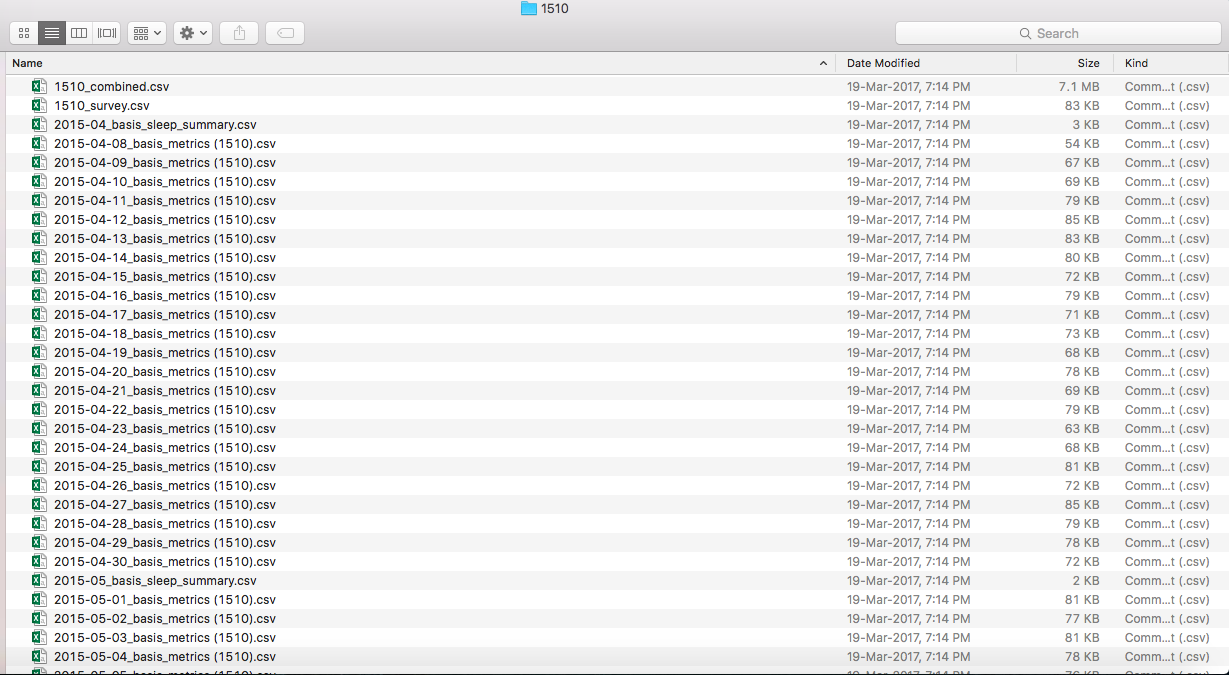
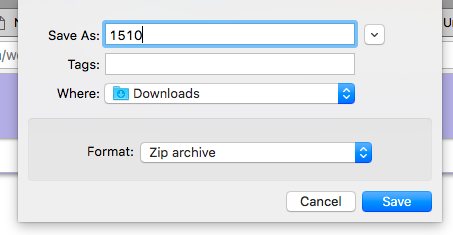


Figure - Zipped file download and archived. Implemented with the help of File Management Module

All the files are separated based on studies and saved in study folder, this structure helps the running of this module, the key consideration is each file is denoted by the participant ID only, thus making it the unique identifier helps access these files inside the scripts as well as in the front-end view easy. The Figure [33][34] show the download of zip, the file contents and archived folders.

# CONSLUSION AND FUTURE WORK

MTD is a web application for showing a dashboard view of large sets of psychological survey and sensor data. This project encapsulates the full stack development methodology in developing modular multi page web applications. MTD has successfully incorporated disparate data sets and has provided multi-level displays.

This is a noticeable addition to an existing pipeline where the researchers collect, clean and analyze this kind of data to build predictive models using machine learning techniques. MTD stand out with its performance when compared to the web service shown in [3], by the fact that this supports multiple studies all at one place.

The speed of website loading is 3 times faster than the one even though this supports multiple data sets.

This system provides a secure approach for access of the data, a 2-layer authentication approach, register and approval via an admin is a must before logging in. The passwords are securely encrypted and stored in databases unlike storing in a file.

This provides better reliability comparatively because it encapsulated the services provided by the Amazon Web Services as it is hosted on an ec-2 instance.

One of the less tangible but the most important feature of this application is its modularity. Every study is treated as a single entity but the shared components are all stored in a separate module to be used across all the studies. Using web development frameworks makes a complicated architecture designed in an easier manner. AngularJS [10] is a current industry standard, using it has made the developers better at it as it has thrown many challenges and questions. As displayed in the Figure 14c, such kind of file structure allows easy maintainability and provides room for extension in the future. MTD employs unit testing features which is a good way of testing an application, as that tests the working of every module individually. Separating all the concerns by breaking down the system into high level modules which consists of individual modules is the most recommended way of development and this practice was followed in both stacks, at the front end as well as in the back-end. Using databases for storing data is another plus point of MTD when comparing this with similar application which used *.txt* files as input for drawing graphs. Currently MTD supports 3 datasets and all the systems are running as expected in a stable environment.

Developing MTD has involved using some of the newest and popular web development technologies [6][10], handling large datasets in terms of relational databases and reducing query complexities, all of which has been an enriching process in terms of knowledge gained about web applications architecture and dashboard development.

Some of the future enhancements for MTD will be adding restrictive access to the study modules based on the role and affiliation, this means users other than researchers can get access to select set of data. This will be an addition to the admin module. Addition of more studies is a very good possibility, the pilot version involves 3 studies, more sensor based studies are ready to be deployed in the horizon, thus those can be supported via a dashboard system.

Improving the pipeline by adding more automation and real time features will be an important addition. Using Highstocks.js for more detailed interactive graphs is a possibility.

The unit testing implemented is a basic version of Karma (via AngularJS), a more sophisticated implementation will make the application more robust.

This application is a good base for research studies as it involves all the stake holders and provides a platform by converting raw data collected via surveys and sensors.

into information via visualization and analytics.

# REFERENCES

1. R. Shi, C. Zhang, H. Wang, P. Sun, T. Trull and Y. Shang, "mAAS - A Mobile Ambulatory Assessment System for Alcohol Craving Studies," Computer Software and Applications Conference (COMPSAC), pp. 282-287, 2015.
2. P. Sun, N. M. Wergeles and C. Zhang, "ADA - Automatic Detection of Alcohol Usage for Mobile Ambulatory Assessment," in Smart Computing (SMARTCOMP), 2016 IEEE International Conference, 2016.
3. Ruiqi Shi, “An Enhanced Mobile Ambulatory Assessment System for Alcohol Craving Studies”, University of Missouri, Department of CS in Master’s Thesis, April 2015.
4. Chen Zhang, “Wearable Sensing Analysis – Identifying Alcohol Drinking from Daily Physiological Data”, University of Missouri, Department of CS in Master’s Thesis, July 2016.
5. Twiki- <http://twiki.org/cgi-bin/view/Plugins/ChartPlugin>
6. Highcharts.js – JS <http://www.highcharts.com/products/highcharts/#non-commercial> 13 Apr. 2017.
7. N. M. Wergeles, “AMD: ANALYSIS OF MOOD DYSREGULATION A Machine Learning Approach”, University of Missouri, Department of CS in Master’s Thesis, 2016.
8. J Andreeva, I Dzhunov, E Karavakis, L Kokoszkiewicz, M Nowotka, P Saiz, D Tuckett, “Designing and developing portable large-scale JavaScript web applications within the Experiment Dashboard framework” in (CHEP2012), 2012.
9. "Model–view–controller." - Wikipedia. Wikimedia Foundation. Web. 13 Apr. 2017.
10. AngularJS - <https://docs.angularjs.org/api> - Web. 23 Apr 2017.
11. “Separation of Concerns” - <https://en.wikipedia.org/wiki/Separation_of_concerns> - Wikipedia.
12. “Whitepaper - Guide to creating Dashboards” – Web. Juiceanalytics.
13. “Software Design Principles” – Article. Web. Eric Steve Raymond
14. <https://bitbucket.org/lguerdan/researchdash/> - Bitbucket Repository for MTD.
15. <http://ec2-54-147-173-237.compute-1.amazonaws.com/web/#/login> - URL for MTD.
16. Karma - <https://docs.angularjs.org/guide/unit-testing> - Web. 23 Apr. 2017.
17. "Usage of Server-side Programming Languages for Websites." Usage Statistics and Market Share of Server-side Programming Languages for Websites, April 2017. W3Techs, Web. 13 Apr. 2017.
18. Nilesh Jain, Priyanka Mangal and Deepak Mehta, “AngularJS: A Modern MVC Framework in JavaScript”, in Journal of Global Research in Computer Science. 2014.
19. <https://msdn.microsoft.com/en-us/library/ee658099> -   
    “Designing Web Applications” – Microsoft Guidelines Web. 29 Mar. 2017.
20. SOLID – <https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)> .20 Apr. 2017.
21. Highcharts-ng <https://github.com/pablojim/highcharts-ng> . GitHub. 20 Apr 2017.
22. Dependency Injection - <https://en.wikipedia.org/wiki/Dependency_injection> Web. 20 Apr. 2017.
23. JSONLINT – Online Verification Tool <http://jsonlint.com/>
24. Matt McKeon, “Harnessing the Information Ecosystem with Wiki-based Visualization Dashboards”, IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 15, NO. 6, NOVEMBER/DECEMBER 2009