Technical Design Specification

Matthew Lydigsen, Lucas Meira, Jake Bernstein, Raymond Zhu, Shaoxi Ma, Michael Markman

Overview

by Raymond Zhu 4/7/15

- Plannit is an application that engages people of all backgrounds in an effort to give them the opportunity to make their days easier through organization and management. Plannit provides its users with modules that let them organize their lives in as complex or simple a manner as they like. Plannit will contain modules that will be presented in different categories that the user specifies on his or her homepage. Each homepage is customized for a different user and is accessed through that user's login info.
- With Plannit, the user could have something as small as a calendar to put down important dates, or could go as big as a calendar, a reminder system, your entire notes for a class, and even more! The possibilities and customization options are endless when it comes to Plannit, and it is up to our users to find out what works best for them.
- This document will take a look into the design/structure of Plannit and how all of the code will be set up

Revision History

by Matthew Lydigsen 4/7/15

Author	Version	Summary of Changes	Date of Change
Matthew Lydigsen	1.0	Logo for website created	03/05/15
Jake Bernstein, Matthew Lydigsen	1.0	Initial layout of website created	03/06/15-03/08/15
Raymond Zhu, Shaoxi Ma, Lucas	1.0	Default modules are decided	03/07/15
Jake Bernstein	2.0	Budget module added to list of default modules	04/01/15
Matthew Lydigsen	2.0	Layout of all parts of code for express app created	04/05/15
Michael Markman	2.0	List of dependencies	04/09/15

Why MEAN Stack?

by Jake Bernstein 4/7/15

For this app we will be using the MEAN stack to do all of our coding. The MEAN stack uses four pieces of software: MongoDB, ExpressJS, AngularJS, and NodeJS. Using these four tools together will let us create an efficient, well organized, and interactive application quickly.

Since every component of the stack uses JavaScript, we can glide through our web development code seamlessly. Using all JavaScript lets us do some great things like:

- Use JavaScript on the server-side (Node and Express)
- Use JavaScript on the client-side (Angular)
- Store JSON objects in MongoDB
- Use JSON objects to transfer data easily from database to server to client
- A single language across your entire stack increases productivity. Even client side developers that work in Angular can easily understand most of the code on the server side.

With our database, we store information in a JSON like format. We can then write JSON queries on our Node server and send this directly to our front-end using Angular. This is especially useful since we have multiple developers and are working in a team. Server-side code becomes more readable to front-end developers and vice versa. This makes everything a little more transparent and will greatly speed up development time.

The MEAN stack benefits greatly from the strengths of Node. Node let's us build real-time open APIs that we can consume and use with our frontend Angular code. Transferring data for applications that require quick display of real-time data. Plannit will be very front-end heavy and very reliant on displaying our modules in a timely fashion, so this will be very useful.

The MEAN stack will allow us to build our application with a client-server Model in mind. This means our application will be built with two separate parts (Client side and Server side). The providers of a resource or service (servers/backend/Node) will handle the data layer and will provide information to our service requesters (clients/frontend/Angular). This will be beneficial to Plannit as the server will sometimes need to act independently of the user so send

the user reminders and to edit the user's calendar based on unforeseen events like holidays.
Overall, The MEAN stack provides a ton of features that will help us quickly code Plannit, help us iterate through our application versions, help us debug our code quickly, and most importantly help us communicate and understand each other's code.

Below we will talk about the different sections of the MEAN stack and how they fit into our code. All of external libraries we will be using are libraries that we need in order to use the MEAN stack.

External Libraries-

by Michael Markman 4/9/15

- express
    - Express framework for route handling and turning Nodejs into a server
- oauth
    - For user authentication
- mongo
    - For storing data
- gcal
    - For the users calendar
- bodyParser
    - To process http requests
- grunt (development dependency)
    - For server start and checking for changes to files
- bower (development dependency)
    - Manage dependencies
- angular
    - For front end data binding with database (MVC)
- socket.io
    - For real-time event driven communication
- bootstrap
    - Front end CSS library
- errorhandler

    - For processing errors
- morgan
    - http request logging middleware
- ejs
    - Front end templating
- express.js
- angular.js
- node.js (all associated modules)

MongoDB

by Lucas Meira 4/7/15

MongoDB is an easy to use database framework that provides high performance, scalability and availability.

For our application, lib/user.js will initialize MongoDB and get the databases of all users. Every user will have a username, password, email, and a uid. In addition there will be a default database for the “To Do List” and a database used to store that users’ course names. For each course, there will be a designated database for that specific class. These course database will store the modules needed for that respective course. And depending on the module, for example “Notes” and “Upcoming Events”, there will also database for those modules with a list of information.

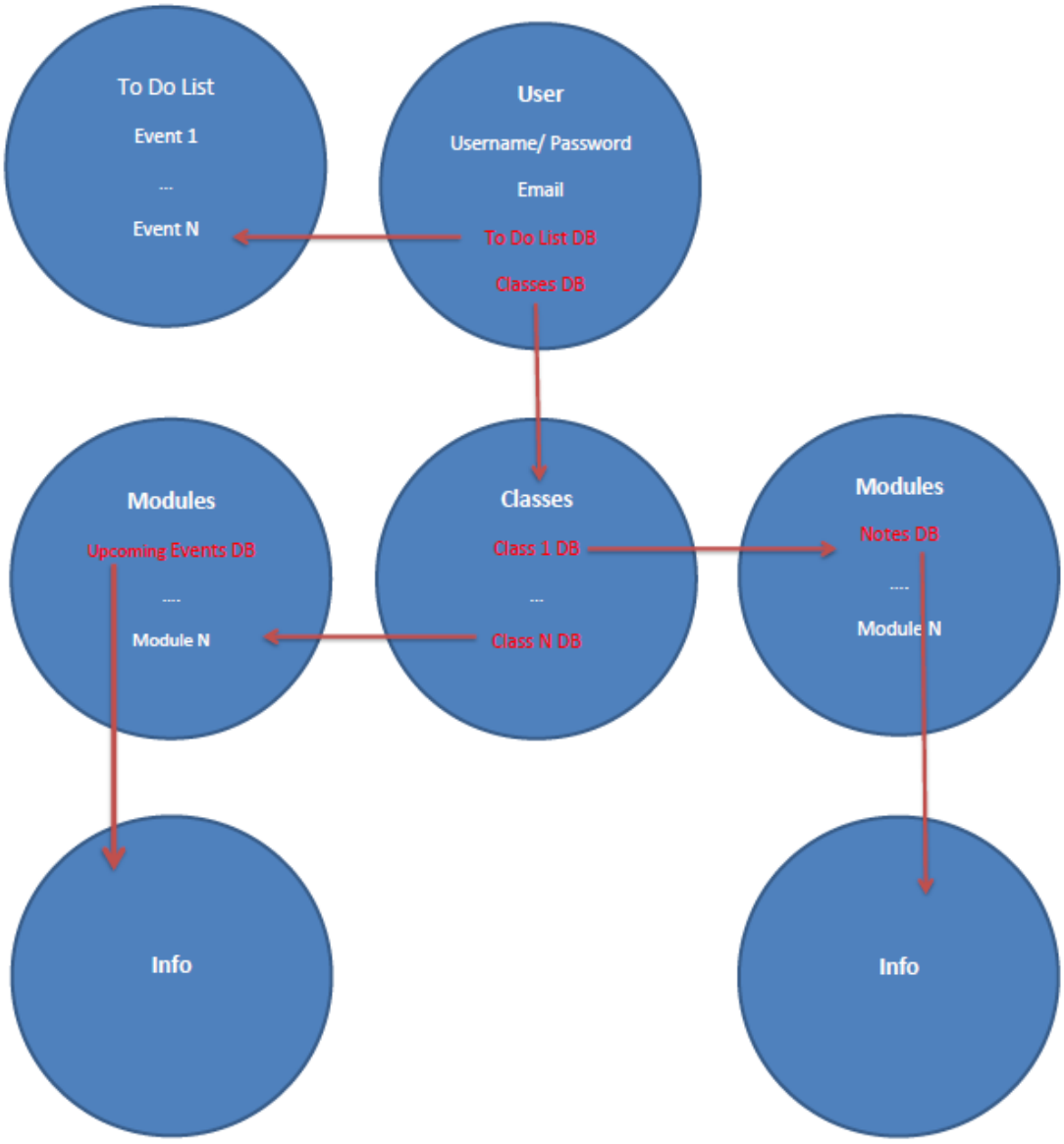
External Libraries (Dependencies)-

- express
    - Express framework for route handling and turning Nodejs into a server
- oauth
    - For user authentication
- mongo
    - For storing data
- bodyParser

    - To process http requests
- grunt (development dependency)
    - For server start and checking for changes to files
- bower (development dependency)
    - Manage dependencies

Birds eye view-

This flow-chart lays out our database structure, circles represent databases and arrows from one database to another mean that database contains a link to another database. For example, all users contain a link to a database that is their to-do list.



Component Breakdown-

- This is a list of functions we will need so that the routes users.js and login.js(see node section) can interact with our database system (along with out views).

- online[]
    - This array will be used to store everyone who is online

- exports.addNewUser = function(username, password, email) { … }
    - This will add a user to the users database and ensure that this user isn't already an existing user

- exports.removeUser = function(username, uid) {...}
    - This will remove the user from the database

- exports.addHomeModule = function(user, nameOfModule) {...}
    - This function will be called to add a unique database that will store all of the users' planners (for example cs326, cs250, etc.)

- exports.removeHomeModule = function(user, nameOfModule) {...}
    - This function will remove a planner from the users database of planners. This can be used in cases where a planner becomes obsolete and the user no longer needs it.

- exports.addPageModule = function (user, nameOfModule, newPageModule, pageModuleData) {...}
    - This function is used in order to create a new, unique page module database that is specific to a user and one of their planners.
    - The newPageModule variable will be the name of the new database
    - The pageModuleData will be the data you store in that newPageModule, such as notes you may have or possibly a class link.
    -
- exports.editPageModule = function(user, nameOfModule, pageModule, pageModuleData) {...}
    - This function is used to edit a page module data. This can be used for changing certain aspects of a module that needs to be updated. For example, if you want to add to your notes or correct your list of upcoming events, then this function will be useful.

- exports.removePageModule = function(user, nameOfModule, pageModule) {...}

- This function removes a specified page module and all data associated with it. If a page module becomes obsolete, then the user has the ability to remove that page module by using this function. For example, if the user no longer needs a budget module for one of his/her pages that than he/she can easily remove it.

Team members-

- The team members responsible for this part of the code are Matthew Lydigsen, Raymond Zhu, and Lucas Meira

Express

by Shaoxi Ma 4/7/15

Node.js is a platform that provides a free environment to develop server side functions. Express is the web application development framework that we are using to build Plannit based on Node.js platform. It helps us set up our server environment, generates the barebones of the app, and provides us a friendly interface to design our own app. So we can adjust these settings easily, according to our needs.

Routing is a main speciality that Express provides us to perform the communication between server side and client side. Routers can handle the request from the client, get the data from database, then send the response back to the client. Middlewares are also useful for some data pre-processing work, which helps routers handle part of their work.

External Libraries-

- For this part of the app we need to install Express.js

Birds eye view-

app.js is our main entrance to our application on the server side. Thus, it is a core component that builds the base for our app. It is very important because it sets up the HTTP server that allows users to connect to, handles some connection errors for routers, then call routers to process the request according to the routes.

Modules:
We will use most of the default modules that Express generates for us, such as:

- express (Express basic functions)
- path (path manipulation)
- serve-favicon (Plannit logo)
- morgan (logger)
- cookie-parser (handling cookies)
- body-parser (parsing different type of data)

We will also include two main routers in app.js; login.js and users.js (see Node section to get more information). Ejs is the default view engine of our app. More modules may be added to app.js according to our needs.

Middlewares:
We will use middlewares mentioned above to preprocess the received data. Basic error handlers will catch different error codes, then redirect the users to the appropriate urls and display the error message.

Most of the express code does not need to be changed for our application so for the time being we will use the default layout that the express app generates for us

Angular

by Jake Bernstein 4/7/15

Angular.js is a Javascript framework that was built by google to provide fast and dynamic front-end deployment. Angular allows you to build normal HTML applications and then extend the markup to create dynamic components. Angular will help us prevent some commonplace problems such as data binding, form validation, DOM event handling, and much more. Angular has two main features that will assist in the creation of Plannit. These features are data binding, which deals with how we handle data in our application, and dependency injection, which deals with how we architect our data.

Our app Plannit will require a lot of different modules, functions, and user storage, which would normally require us to store our data in a lot of different locations. This would end up making the coding hard and finding the right data even harder. But Angular keeps this simple for us. Angular has something similar

to a model view controller that keeps all the data in one spot so it is easy to find and edit! When looking for specific data, it will be easy to be sure we are looking at the correct information thanks to the fact that changing data in our view (HTML files) or controller (Javascript files) changes the data everywhere. (Thanks to data binding)

Data binding takes care of storing and locating all the different data for our various modules, but what about actually programming these modules? Won't it be hard to program each module and test it without having to rely on all the other code components and modules? It is especially important to solve this problem since we are programming in a team and it is hard for one programmer to pick up where another programmer left off or to understand that programmer's code. Dependency Injection allows us to compartmentalize and modularize our application. We can reuse our modularized pieces across many different parts of our application and even many different projects. We can even pull in modules that other developers have already created. Injecting these modules into our application will allow us to test each module separately. This could not cater anymore perfectly to the overall design of our Web app since Plannit is almost entirely based off modules that are provided for the users. And debugging will be a breeze since dependency injection will allow us to easily determine what parts of our application are failing and more easily hone in on the nasty bugs in our code.

Since Angular helps with our html coding we will discuss below the different pieces of html code we will need for this app.

External Libraries-

- To use Angular we must install angular.js

Birds eye view-

Below is a list of all of our ejs files which will handle the html/css side of the code. Our routes files users.js and login.js (in node section) will render these files at specific routes and then when the client has to send data back to the server

these files will go to a different route in either login.js or user.js which will then decide what to do with that data.

- addUser.ejs
- headerLogin.ejs
- home.ejs
- module.ejs
- login.ejs
- headerUser-ModulePage.ejs

Component Breakdown-

- Our login.ejs file will display the login view which is where everyone starts before they are logged in
- addUser.ejs will provide a view to add new users. Once a user has entered all of their information the form will reroute to their new homepage and enter their data into our users database
- HeaderLogin.ejs and headerUser-ModulePage.ejs will be our two default header classes. headerLogin.ejs will be the header for the login and addUser pages, which will not have a logout button since the user isn't logged in. headerUser-ModulePage.ejs will be the header we use for every other page on the site. This header will include a logout button
- Once a user has logged in the login.js file will reroute them to their new home page which will be displayed using the home.ejs file. This view will include the users to-do list which they can edit and this will update the users to-do list database. This view will also include all of the planners that are associated with a particular user. The route will pass in the users planner database and list all available databases. They can add planners and remove them whenever they want. When they add a new planner or click on an existing one they will be re-routed to that particular planners page
- module.ejs will be our last view and it will be the view for all planners and modules associated with those planners. Here is where the user gets to customize their planners and add from our list of default modules whatever they want whether it be our notes module or our new budget module. When a user adds a new module the data will be sent back to the users database for that particular planner and will then be added to that database

Team members-

- The team members responsible for this part of the code are Jake Bernstein, Harry Ma, and Michael Markman

Node

by Matthew Lydigsen 4/7/15

Node is a javascript platform that is perfect for building network applications easily and quickly. Node helps us run all of our javascript code so in this section I will list out our routes files and how they will integrate our database code with our html code as well as how it interacts with our databases.

External Libraries-

- For Node we must install Node.js and all associated modules that Node uses

Birds eye view-

- Classes involved are users.js and login.js which are in our routes folder
- These two classes provide all of the routes that we will use to navigate through our website.
    - login.js will be our home page and access to all user pages is restricted until the user is logged in.
    - users.js will provide all routes for the users individual page based off of the users database.

Component Breakdown-

- login.js will include only a few routes which include /login, /login/newUser, and login/addNewUser. login/addNewUser is only used to actually add the new user to the database and then redirect to their new home page
    - /login will be the default screen that everyone starts in and a user can only go to this page and /login/newUser page until they are signed in

    - /login/newUser will have a simple form that you fill out and then it will redirect to /addNewUser to actually add you as a new user and verify you are new user
- users.js will include a few more routes but still not too many overall. These routes will include /users/home, /users/addHomeModule, /users/editHomeModule, /users/removeHomeModule, /users/editToDoList, /users/logout, /users/nameOfHomeModule, /users/nameOfHomeModule/addPageModule, /users/nameOfHomeModule/removePageModule, and /users/nameOfHomeModule/editPageModule
    - /users/home will be the home page for all of our users and will display all of their planners they have created and their to-do list
    - /users/addHomeModule, /users/removeHomeModule, and /users/editHomeModule will edit the current planners listed on the users home page
    - /users/editToDoList will modify the to-do list on the users home page
    - /users/logout will logout the user
    - /users/nameOfHomeModule this path will take you to a page that contains one of your planners. The path name nameOfHomeModule will be replaced with the specific planners title. On this page we will put every module that the user has added to the page in this view. This is the part of the website where users can truly build their planner the way they want to.
    - /users/nameOfHomeModule/addPageModule will add a module to the planner you are in depending on which module you specify
    - /users/nameOfHomeModule/removePageModule will remove the specified module
    - /users/nameOfHomeModule/editPageModule will edit the module you specify possibly the notes section or one of the other modules

Team members and their responsiblities-

- Matthew Lydigsen - Team captain, in charge of overseeing the project. Handling a lot of database work with MongoDB.
- Jake Bernstein - Working largely with angular to take car of the front end. Using angular to make the modules dynamic and aesthetically pleasing. Also helping with the rest of angular for the website (such as the homepage and the login)

- Lucas Meira - Working with MongoDB to help handle the backend. Handling a bit of database work, and a lot of routing work
- Raymond Zhu - Working on the backend with MongoDB. Helping with the routing, and handling most of the API calls
- Michael Markman - Doing the research on what external libraries, frameworks, and other technologies would be best used for our web application
- Harry Shaoxi- Handling part of the front end. Using angular to create a pleasing login page, and to correctly pass data between pages. Also working on any other front end areas where he sees a need for improvement.

Possible implementation problems-

- Learning how to best integrate the different components of the MEAN stack together
- Using angular to make the front-end of the web app dynamic and aesthetically pleasing
- Handling all the databases that we need to store for each user and his or her modules
- Being able to create a great finished project and also let each team member express their ideas