## **CS 3750 - Challenge 8-1**

By: Levi Bernards, Hayden Blackmer, Natalie Mueller

For our project, we decided to use a stack that included Express.js with Node.js and MySQL. This stack already implements a few different design patterns, so we utilized those to create clean and maintainable code.

First, and most importantly, Express.js is built around the Middleware Pattern. This pattern creates a kind of a pipeline that places numerous levels of processing in between every request and response. Some examples of middleware that are included in Express.js out-of-the-box are error handling, logging, and cookie parsing. These are added to the “app” variable by using the “app.use()” function, and from there each tool has the ability to access requests and responses. We customized the middleware in our application and included Sessions and a MySQL Session Store. We also exposed Session variables to our templating system (EJS) using a middleware function. Express.js’ Routers are also added to the app as middleware. These take the path of the request, send it to the specified router, and then that file processes the request and returns the page.

The second pattern we used that is already built into Express.js is the Singleton Pattern. The idea behind this pattern is that many programs only need one instance of certain objects, therefore they should be built so that only a single object exists and the program cannot create more of them (such as global variables or static variables.) Express.js’s “app” variable is a great example of creating a singleton using a global variable. When the server starts, it loads the app.js file and this global object is created as the main point of entry. There are many other singletons found in the app.js too, including variables for all of the different routers in the app and variables that manage all of the installed middleware, like error handling, logging, sessions, etc…

And finally, the third pattern we utilized in our project is one commonly found in Node.js projects: The Command Pattern. This pattern says that all of the information and functionality required to perform a command should be encapsulated so that it can be called at any later time to perform the command. The command pattern is useful if we want to repeatedly call a command, send that command to somewhere else in the app to be called, or even schedule the command to be called at a later time. We used this pattern in two different places related to our database. First, we created an object with database credentials that will automatically open up a connection to the database when called. And secondly, we created an object that will use the current database connection to create a database, add all of the necessary tables, and populate these tables with seed data.