We did the movie database for our project. Overall, we tried to make an extensible, robust, and scalable web application using the latest web-based technologies.

The successful use of the server relies on a few assumptions. The foremost of which is the assumption that the server will use the pre-existing database folder located at ./database, and have it running before starting the actual app server. The server also assumes that the database will have the following expected structure: db -> users, movies, newPeople. Furthermore, the web app assumes that the user will be using a sufficiently updated browser in order to support the various jQuery library functionalities. The final assumption is that the user has installed all of the prerequisite NPM modules using the “npm install” command.

To start the server, you must connect to public IP: 134.117.132.15 on Openstack. The username is student and the password is yash1014. FIrstly, you must change directory into the project folder with “cd ~/MovieDatabase” To start the MongoDB server, you must run the command “npm run start-database”, and then start the app with “npm start”.

Some of the key web technologies that we used include jQuery, Bootstrap and MongoDB. With jQuery, we attempted to use its wrappers of native Javascript functions to speed up our workflow. This includes functions such as the $.ajax method that wraps all of the XMLHttpRequest functionality into a more readable and compact form factor. Furthermore, jQuery also has the added functionality of bridging compatibility gaps between browsers. Instead of having to program multiple ways of doing things, the jQuery methods will handle any browser differences in the background.

Bootstrap gave us a comprehensive and responsive starting point for the actual website itself. Similar to jQuery, it takes out a lot of the steps of ensuring browser compatibility and allows us to focus on the actual content of the website itself, instead of focusing solely on the formatting. By using Bootstrap and its grid system with rows and columns, we could easily adapt our website to fit multiple viewport sizes and even mobile devices. To further ensure this mobile-friendliness, we also had to change our logo from an image that also contained text to an image with a separate <h1> element. We tested the responsiveness of our website through multiple methods, including using Chrome Dev Tools on the responsive window sizing and simulating a phone display with Chrome Dev Tools’ mobile tools. As Bootstrap automatically formats webpages to conform and respond to the viewport’s size, everything looked correct.

We primarily used .json files containing our users and movies for most of our project. However, we switched databases to MongoDB in light of the significant advantages of its usage over .json files. With .json files, we had to loop through the objects within the file and check to ensure that the right one was selected. With MongoDB, this process was simplified and performance-optimized as we could instead make simple queries into the database server with its support for things such as Regex.

We did end up using the movie-data.json file as a template for our movie database, but we added a few modifications such as changing ratings to integers (to use to calculate the average on the movie pages) and a review field that contains an array of objects of the following structure: {username: “”, password: “”, profilePic: “http://...”, followUser: [], peopleFollow: [], movies: [], contributor: false, notifications: []}. The password field will never be sent to the client side and will only be used on the server when evaluating the inputted login form. The profilePic field will contain the direct link to an image in order to display it on their user page. The followUser array contains the usernames of all the people that the current user is following, while peopleFollow contains all the people that they are following. The movies field indicates all of the movies that the user has reviewed. Contributor is a boolean that indicates if the user can add movies and people, and if not, they are redirected back to the login-page or homepage. The contributor status can be updated on the personal profile page. Finally, the notifications array acts as a queue for all important messages directed to the user. A string is pushed to the notifications queue when a followed user adds a movie review or a followed person is added to a new movie. When the user with the notifications logs in and checks their personal profile page, the first element of the queue is popped out and displayed as an alert.

To add a movie, the user must first be logged in and be a contributor. These two conditions are checked by the server using query parameters and session variables, and if they are not met they are redirected to the appropriate pages upon a submission attempt. The contributor can then fill out a form with the bare minimum amount of information required. This form then uses the POST method to transfer the information to the server and inserts it as a document in the movies collection.

When looking at a movie page, the user has the option to add a review if they wish. This review consists of a basic rating out of 10, followed by a full-text review. When this form is posted to the server at /movies/addReview/:movie, the server uses the user’s session variable username to determine the reviewer and then pushes an object containing all of the review’s information into the movie’s review field in the database.

Our web app also implements a RESTful API in order to ensure proper separation between client and server and a cleaner codebase. With the GET /movies route, one could search the database using various query parameters such as the movie’s title, genre, year, and minRating and return an array of matching movies. This is used in the “View Full Database” and homepage screens as it populates the movie poster grid. The GET /movies/:movie router allows the user to search for a specific movie using its ID and returns it as a JSON object.

We also have three partner project extensions. These three extensions are mobile responsiveness, the use of MongoDB, and a forgotten password form that emails the user their password.

While the use of mobile-friendly frameworks and MongoDB are described above, the forgotten password form sanitizes and validates both the username and email, ensuring that the username is actually in the database and that the email is in a proper format. Afterwards, it uses the nodemailer NPM module to send an email containing the user’s password to the specified email. An important thing to note is that the email may sometimes be sent to the Spam folder.

We have strived throughout the project to reuse and share code where possible. An example of this is with the dynamically generated navbar in the navbar.js file. Almost all of the pages require the navbar, so instead of adding a new navbar in each HTML file, the navbar.js file is included in each .html file and then runs an initializing function that actually generates the bar.

Overall, we were able to implement most of the functionality described in the specifications. One unimplemented feature set, however, is the addition and managing of people. The server will currently search through the movies to find a person, and if not found in the movies, it will search in a secondary newPeople database. The newPeople database is where contributors can add new people to add in new movies. Currently, the contributor can add movies with people who haven't yet been added to the database. We believe that this may be because of the asynchronous nature of some of the operations within a few helper functions for the people-router. A greater degree of polish could also be used, such as with small visual errors or the opportunity to reuse more code.

After much discussion, we have decided that the aspect we most liked about the project were the new concepts and technologies we learned about and used. For example, neither of us had any previous experience working with MongoDB or other types of databases. We also had no experience with Node.js and Express. But throughout this experience with our final project, we were able to learn more and grow as programmers. We believe that the best feature(s) of our project include the dynamic propagation of the movie, user, and people pages, along with the search database page that is able to efficiently work despite rather large data quantities. These elements were also some of the most difficult, but they had proven very enjoyable.