Spring 2019

Estimating And Visualizing Debt Using Debt Calculator

Donald E. Landrum Jr.
University of South Carolina, donlandrum@hotmail.com

Caleb D.H. Parks
University of South Carolina, calebdhparks7@gmail.com

ESTIMATING AND VISUALIZING DEBT USING DEBT CALCULATOR

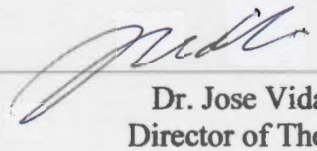
By

Don Landrum & Caleb Parks

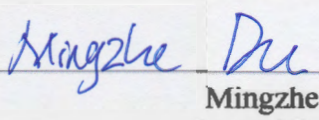
Submitted in Partial Fulfillment
of the Requirements for
Graduation with Honors from the
South Carolina Honors College

May, 2019

Approved:



Dr. Jose Vidal
Director of Thesis



Mingzhe Du
Second Reader

Steve Lynn, Dean
For South Carolina Honors College

Table of Contents

Thesis Summary

Debt Calculator is a web application that our team developed this year to assist users in understanding and visualizing both their current debt sources, such as student loans, car loans, and credit card debt, and the impact that upcoming career choices, such as where they attend graduate school or which job offer they accept, will have on their standards of living and savings accounts. All of the data that our application bases its analysis upon is entered by the user, and none of the algorithms that our application performs are particularly unique or proprietary; rather, the value that our application adds to the lives of its users is in time and effort saved, for our product can distill a complex array of upcoming opportunities, each in different regions of the country and involving different salaries or loan terms, into a ranking of most affordable opportunities in a matter of seconds, saving users the hours of effort that they might have otherwise spent searching for cost-of-living data and relocation costs for each of the cities to which they are considering moving.

Most of the tasks that our team endeavored to complete fell into one of two broad categories: determining which data we needed to collect from our users and how to best store that data, and determining how to show our users the results of our analysis in an intuitive and aesthetically-pleasing manner. In our attempts to address the first category, we first collected much more data than was needed before streamlining the data input sections of Debt Calculator and focusing on the most critical portions of the user's portfolio. Our data collection is divided into three sections: the Profile page, the Opportunities page, and the Debts page. A user's Profile page contains data that informs us about that user's current standard of living, such as their current monthly rent, grocery budget, and savings balance. A user's Opportunities page contains a list of the user's upcoming opportunities, each of which has a title, location, income data. A

user's Debts page contains a list of the user's current debt sources, each of which is either constant across all of their upcoming opportunities or specific to one opportunity.

Determining how to serve results to our users in an appealing way required a number of decisions, such as what color scheme we would use on the website, which graph-generation package we would utilize in our analysis section, and even how we would structure our HTML pages to make our user experience as pleasant as possible. One of the main governing principles of the user experience on the Debt Calculator website is the use of Material Design, which is a style of website layout invented and popularized by Google. By using buttons, cards, and forms influenced by Material Design, we have made our website appear familiar, new, and stylish to our users, which in turn inspires them to spend more time using our tool. We have also striven to give users as much freedom as possible in deciding the types of analysis that they want, which is evidenced by features in Debt Calculator such as charts that display over a user-adjustable length of time and graphs that allow users to toggle the visibility of their input sources. We believe that these enhanced features make our users feel empowered by the insight that our product provides, and we hope that our users enjoy viewing their projections and analyses in such a customized environment.

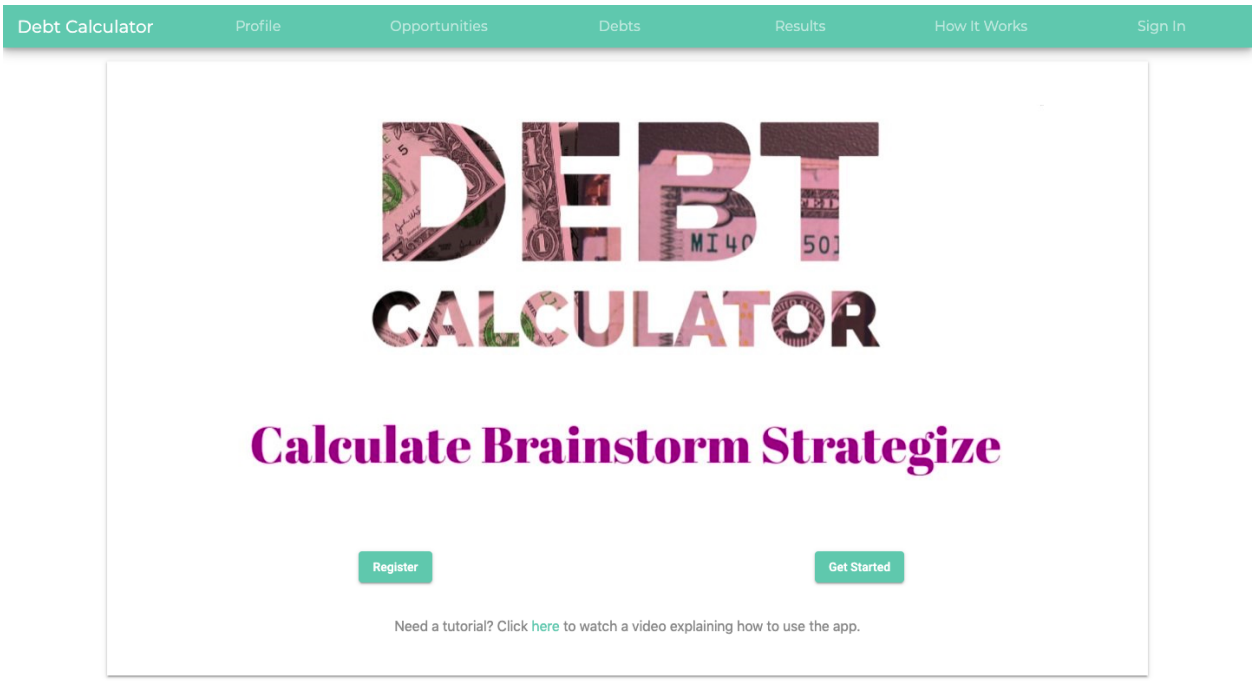
Introduction

This project was designed to satisfy both the Senior Thesis Project requirement of the South Carolina Honors College and the CSE Capstone requirement of the Computer Science and Engineering Department. CSE Capstone projects are divided into "startup" projects, in which students build an original product, and "client" projects, in which students build a product for a client in the community. Debt Calculator is a startup project, so our team brainstormed the ideas behind this product, developed our own goals and milestones, and worked towards the completion of those objectives.

The idea that sparked Debt Calculator came when one of our members was trying to decide to which graduate schools he wanted to apply. He found the process of comparing rent, grocery, and transportation costs across different cities to be cumbersome, and when he tried to also factor in the current undergraduate student debt and projected graduate student debt that each prospective school would add to his financial affairs, he felt as though too much work was required of a college student, like himself, who just wants to make a responsible decision about which graduate school to attend after graduation. After some group discussion, we realized that similar effort is required of college students weighing competing job offers in different cities. Debt Calculator was born as a possible solution to this situation - a tool that college students can use to reduce the number of effortful hours they must spend researching life in different U.S. cities in order to make well-informed fiscal decisions about their professional and academic futures. Though designed with college students in mind, this tool can be used by anyone considering switching jobs, pursuing higher education, or entering a new industry.

Feature Overview

The following image is a screenshot of the landing page of our web application.



The landing page features our Debt Calculator logo, a ‘Register’ button that links to the Registration page, a ‘Get Started’ button that links to the user’s Profile page, and a navigation bar at the top that is present on all pages. The ‘Register’ button switches to an ‘Account’ button, which links to the user’s Account Management page, when a user is signed in.

The navigation bar, which is present on all pages of the website, contains links to the main pages of the app. The ‘Debt Calculator’ link in the leftmost position of the navigation bar serves as a home button and takes the user to the landing page when clicked. The ‘Profile’, ‘Opportunities’, ‘Debts’, and ‘Results’ links connect the user to the four most important pages of Debt Calculator. The ‘Profile’ page is where the user enters personal information about their finances. The ‘Opportunities’ page is where the user can input any job or school options that they are considering. The ‘Debts’ page is where the user’s current debt is taken into consideration.

Finally, the ‘Results’ page digests the compiled information from the other three pages and uses it to provide insight into the affordability of each opportunity. ‘How It Works’ is the next navigation bar link, and it sends the user to an information page that explains the financial equations and mathematical significance of various fields in the other pages of the website. If a user is logged in, ‘Account’ is the penultimate link, and it sends the user to the Account Management page, from which username and password changes are possible. The last link in the navigation bar is dependent upon the state of the user. If a user is logged in, then the last link reads ‘Sign Out’, and clicking it will sign the user out of the application. If there is no user logged in, then the last link reads ‘Sign In’, and clicking it takes the user to the Sign In page. When a user accesses Debt Calculator from a mobile device, every link except for the home button will condense into a menu button. When clicked, the menu button expands down overtop of the page, and its contents are identical to those described above, but the links follow one another vertically instead of horizontally.

The first thing a new user might do is click the ‘Register’ button on the home page, leading them to the Registration page. On this page, there is a field for a username, a field for a password, and a button labeled ‘Register!’. The button will not be clickable until both the username and password fields are filled. Once both fields are filled and the user clicks the ‘Register!’ button, our app checks to see if the given username is available. If so, the account is created; if not, the user is notified and asked to select a different username. When creating the user’s account, Debt Calculator practices proper password protocol and does not store the user’s actual password; instead, our system uses the strong SHA-512 encryption algorithm to hash the user’s password and then stores the hash. Each subsequent time that the user provides their password while signing in or approving major account changes, the password will be hashed

again and compared against our saved password hash, not against the user's password itself. Should our database be compromised, this encryption scheme will protect the privacy of our users. After the user has successfully created an account, they are redirected to the landing page.

Upon visiting the landing page again, the user can click the 'Get Started' button to begin entering information on the Profile page. Here our tool collects information about the user's expenses. We prompt the user to enter their monthly rent, their monthly grocery and food budget, their miscellaneous monthly spending, and their current savings. These are used to get an idea of the lifestyle of the user so that we can compare the costs of similar lifestyles in different areas when analyzing the user's opportunities later. To this end, the Profile page also asks the users to enter the state and geographic region within that state in which they currently live. Once the user has filled out each of these inputs, in the bottom rightmost corner of the page is a 'Submit Profile' button, which, when clicked, stores the user's personal information in our database and forwards the user to the Opportunities page.

The Opportunities page is where the user can enter the essential information of their various opportunities. Each opportunity has a name, a city, state, and region in which it is located, and a projected income. Lastly, we ask the user two boolean questions: "Is the type of opportunity a job or a graduate school?", and "Does the user need to relocate for this opportunity or not?". Once the user enters the required information, they can click the 'Add Opportunity' button to add this opportunity to the database and to the table of opportunities displayed at the bottom of the page. In the table, there are edit and delete options for each opportunity, so the user can update or remove their opportunities in the database after creating them. There is no link to the Debts page on the Opportunities page, as we believe that such a button might confuse the

user by providing them with too many options on one page. Instead, when the user is ready, they can navigate to the Debts page by clicking the ‘Debts’ link in the navigation bar.

The Debts page is where the user can enter all of the existing debt that they have. Each debt has a name, a principal amount, a yearly interest rate, a number of monthly payments, and a number of times per year that the debt is compounded. If the debt has simple interest, the compound frequency is simply zero. Additionally, each debt is associated with either one specific opportunity, or with all opportunities. This selection affects how the debt will be treated in the analysis on the Results page. Much like on the Opportunities page, once the user enters the required information, they can click the ‘Add Debt’ button to add this debt to the database and to the table of debts displayed at the bottom of the page. In the table, there are edit and delete options for each debt, so the user can update or remove their debts in the database after creating the debt sources. Also like on the Opportunities page, there is no link to the Results page on the Debts page, as we believe that such a button might confuse the user by providing them with too many options on one page. Instead, when the user is ready, they can navigate to the Results page by clicking the ‘Results’ link in the navigation bar.

The bulk of our website’s computations are triggered when a user visits the Results page. The first item on the page is a listing of all the user’s opportunities sorted by affordability. Affordability is a somewhat nebulous concept, so for the purpose of our product, we have decided to define it in relation to potential savings. For each opportunity, we project the potential savings that the user might accumulate each month over the next 60 months if they select that opportunity, and we record the minimum and maximum potential monthly savings values for each opportunity. The opportunities with the highest minimum potential savings are ranked highest, and in the case of a tie between opportunities, the opportunity with the highest

maximum potential savings is preferred. The potential savings projection is the most aggregate feature on the Results page, which is why it appears first. To project the user's potential savings for a certain opportunity across successive months, the following process is applied. Beginning with the user's current savings, the monthly income of the opportunity is added, then the user's monthly debt payments, as well as an estimate of their monthly expenses, are subtracted. This estimate takes into consideration the user's current spending habits as well as cost-of-living data from the United States Department of Labor Bureau of Labor Statistics API (BLS API). The resulting estimate of monthly savings is recorded, and the cycle is repeated for each month in the desired time window. Once the opportunities have been properly sorted, they are displayed in a table that lists some critical opportunity details followed by the projected savings after one and three years.

The next feature of the Results page is a color-coordinated line graph to help users visualize their existing debts that are opportunity-agnostic. This graph displays each debt that was entered on the Debts page and that is associated with all opportunities in a different color and projects the debt amounts for a set number of future months (assuming that all monthly payments are paid on time). By default, this graph projects 60 months into the future, but the user can edit this timeframe using the 'Update Charts!' button, with the minimum value being two months. This graph also features a total debt line that always appears in red and is the sum of all the other debts. Any of the debts can be toggled on and off by clicking the name of the debt in the legend of the graph. Toggling a debt causes the y-axis of the graph, measured in dollars, to scale up and down in real time.

The following feature of the Results page is a similar color-coordinated line graph that helps users visualize their potential savings. Like the projected debt graph above, this graph

shows the projected savings generated by each opportunity as a line in a different color. Also like the projected debt graph, individual lines can be toggled, and the range over which they are viewed can be adjusted using the 'Update Charts!' button. The potential savings that each opportunity generates is computed in the same manner used by the overall affordability graph: beginning with the last month's savings, monthly income is added, and monthly debt payments and an estimate based on spending habits and cost-of-living are removed to produce this month's savings. Just like the projected debt graph, the projected savings graph provides a visual representation of one facet of the user's fiscal near-future, and viewing the impact that each opportunity will have on the user in comparison to its peers may help users understand the large disparities that seemingly small differences can produce over time.

The next feature on the Results page is a bar graph displaying the relative purchasing power of each city in which one or more of the user's opportunities is located. The values in this graph are generated by dividing the cost-of-living in the city in question by the cost-of-living in the average American city, so values above $1 indicate strong purchasing power, while values below $1 indicate weaker purchasing power. The data used to create this graph is obtained from the BLS API.

The penultimate feature present on the Results page is a table containing the average price of purchasing a home in the city in which each opportunity is located. This table is generated using data obtained from the Zillow API. Zillow is a real estate company whose online database contains quite robust data about home sales across the country.

The final feature on the Results page is a Twitter integration. Users can tweet about Debt Calculator using the official Twitter-branded buttons at the bottom of the page. A user can tweet

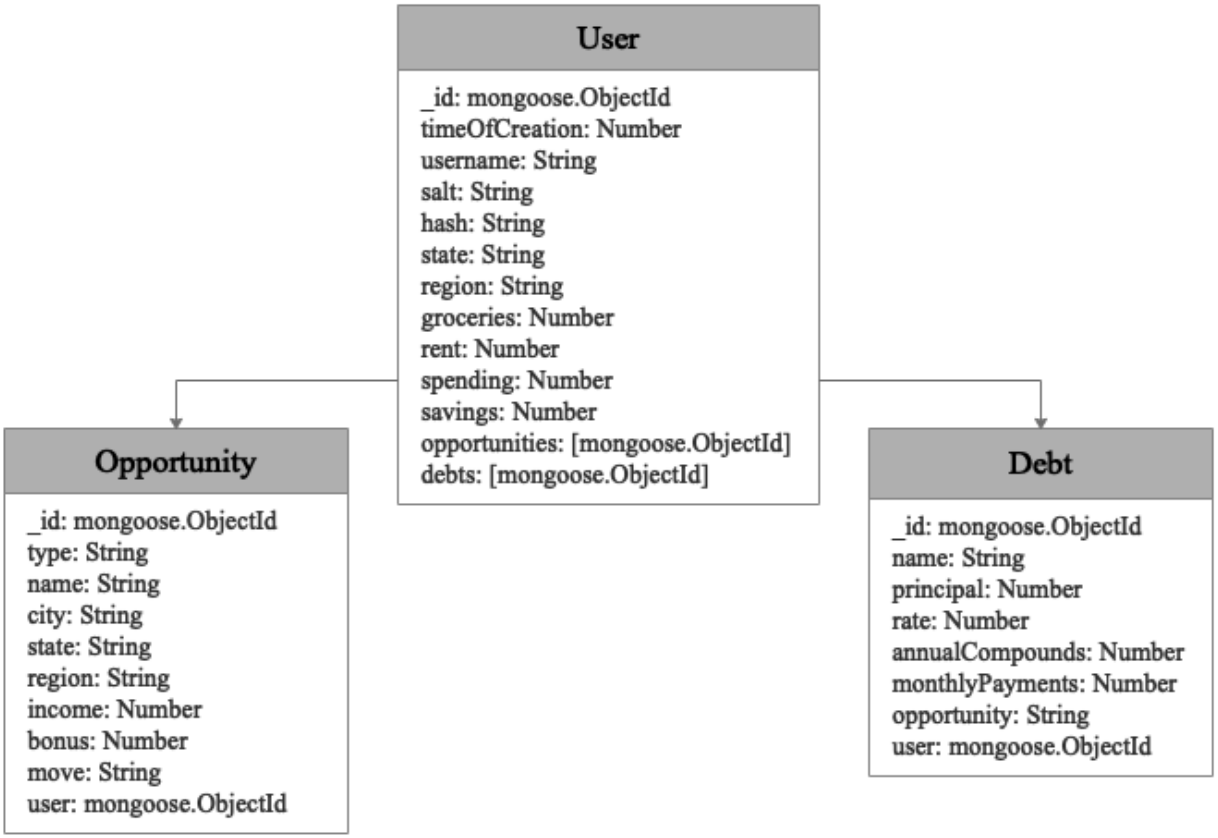
a link to the Debt Calculator website, tweet using the hashtag #DebtCalculator, or tweet at the official Twitter page for Debt Calculator.

The How It Works page is used to describe each of the other pages in the app. It contains a video tutorial that guides users through our app's functions, as well as definitions of some of the terms used in our app. It also declares some of the mathematical equations used in the analysis that our product performs. This page's purposes are to orient first-time users about how to operate Debt Calculator and to inform all users about the computations that take place in the non-visible sections of our product's functionality.

The last important page of the Debt Calculator website, the Account Management page, allows users to edit their account information. Users can change their usernames, change their passwords, or delete their accounts on this page. Each of these actions requires password verification to ensure that the user's information is not compromised due to a simple mistake like forgetting to log out of a shared computer.

Database Schema

The schema of our database, which is hosted by mLab, a Database-as-a-Service website owned by MongoDB, is illustrated below. Our attribute names are listed alongside their data types. The type mongoose.ObjectId is a special type used to identify objects in a database that is running Mongoose middleware, such as ours. For example, by storing the mongoose.ObjectId parameter of each Opportunity that a User owns in an array, we can easily fetch these Opportunity objects and populate an array with them.



Technical Obstacles Overcome

The Debt Calculator website is built on the MEAN stack, which is an acronym for MongoDB (the database), Express (the framework running on top of Node), Angular (the frontend framework), Node (the backend server). We selected the MEAN stack for our app because we knew that each of these four technologies was easily accessible using JavaScript, which was a language with which multiple of our members already had familiarity, and we wanted to spend as little time as possible learning new languages so that we could spend more time handling the intricacies of the frameworks and tools that our app used. Debt Calculator is hosted by Google Cloud Platform (GCP) at https://debt-calculator-csce-490.appspot.com/. We chose to use GCP for our hosting needs because Dr. Vidal procured a trial membership for our team that allowed us to experiment with production-quality resources for free. GCP was a great solution for us because it made common deployment issues, such as HTTPS certificate management and database integration, quite easy. It also has security and scalability benefits that our app would rely upon if it ever accumulated significant traffic. While working with these technologies, we encountered numerous technical roadblocks that we had to resolve.

Some of the most bothersome and frequent technical issues that we encountered arose from disagreements between our Material Design user interface components and the rest of our frontend Angular code. Angular has developed a collection of Material Design inspired HTML elements, and we were eager to use these sleek, professional-looking features in our website. However, these HTML elements are slightly different from the standard ones that they replace, which means that some rather critical functionality that is present in normal, non-styled HTML elements is incompatible with the Angular Material Design elements. For example, the dropdown boxes on Debt Calculator's Profile and Opportunities pages are dependent upon one

another; that is, the items in the second dropdown box depend on which item is selected from the first dropdown box. Creating a system like this one is a standard problem in form-based website design, and it has several standard solutions, but many of them would not work with the Material Design elements that our team wanted to use on these pages. As such, we had to craft a workaround solution to this issue. By calling a function that dynamically populates the second list whenever the value of the first list changes, we were able to achieve the same functionality, but our solution required more latency and lines of code than it would have required if the Angular Material Design components were imbued with more backwards compatibility.

Another issue that we encountered with the Angular Material Design components was their inconsistent compatibility with the Angular Forms module. On our website's Debts and Opportunities pages, we have forms that are allowed to be submitted multiple times without ever leaving the page. Naturally, upon pressing the submit button each time, the user wants to see the form reset to its null values with no errors. However, it seems as though Angular Forms are only designed to be submitted once, as a Form that has been reset behaves differently from a Form that has just been loaded and has never been populated or submitted. When this particular Form happens to contain Material Design components, this different behavior manifests itself with red error notifications, which are highly undesirable for a user that has done nothing wrong. After scouring the internet for a solution to this challenge, we discovered a workaround that allows us to manually disable the Form's error notifications immediately after we reset it, which prevents the red errors from displaying to the user.

Another frontend complication that our team had to navigate was the customization of Chart.js, which is the package that we used to generate the colorful graphs on our Results page. Chart.js had a number of functions that did not seem to work properly. For example, the

documentation claims that every chart object has a “destroy” function that clears out the data in the chart while keeping the reference to the object, but this function does not seem to exist. Without the use of this function, our team had to find out exactly where the data was stored and clear it manually without corrupting the chart object itself. Our first few incorrect attempts at this process caused old data and new data to be rendered on top of each other. Only after we had found a way to remove the old data and replace it with new data were we able to change the scale of the charts based on user inputs.

One final obstacle that our team encountered was deployment. Our initial attempt at deployment was quite difficult and time-consuming, as none of our team’s members had any experience with GCP at the time. Some of the issues that we faced during our initial deployment were the need to move our assets directory to a new location, the need to create a new file specific to the GCP deployment software, the need to edit our app’s startup code to connect it to our production database, and the need to modify the way that our app handles unrecognized URL extensions. Though none of these issues are incredibly difficult to rectify, what made each of them so taxing was the lack of documentation and community support that we were able to locate while trying to address them. Whereas most of the issues that we encountered while writing code had answers buried deep within the Angular documentation pages, the deployment issues that we faced were so personalized to our application that we had to solve them through educated-guess-and-check, making them much more time-consuming than they needed to be.

Conclusion

By building the Debt Calculator web application, we learned many lessons about the software development process. Meeting deadlines, holding team meetings and brainstorm sessions, and documenting our code were activities that we had performed in other classes, but never with such autonomy. Our team had communication issues from time to time, and working through these issues gave us practice for doing so with coworkers in the real world. Becoming comfortable with Github and the more general idea of version control was another valuable skill that we gained while developing this app. Though we did not adhere to all of the formal rules of the Agile software development approach, practicing continual improvement and emulating the quick pace of professional software development helped us gain a more realistic picture of what our first jobs out of college might be like.

Developing the Debt Calculator web application has been an educational and enjoyable experience for everyone on the team, and we are very happy to have fully implemented the originally-conceived core functionality of our product: to develop a tool that distills complex data from multiple sources about various opportunities into a single fiscal recommendation with supporting analysis and explanation. Were this project to continue, the next steps that we would take to further improve the product might include things like using a web search to assist users in finding apartments within their projected budgets or using an intelligent algorithm to discover overspending trends in users' data and provide tips on how to curtail frivolous spending. However, major, labor-intensive enhancements such as these aside, we are quite happy with the product that we have created, which has fulfilled all of the requirements that we originally set for ourselves as well as some additional bonus requirements.