Easy Expense Web App

Adam Schlichtmann Adrian Ariffin Micheal Baumfalk

Introduction

In this project, we attempted to solve the problem of keeping track of how much money everyone in a group of people owes each other.

This problem is important because groups of people, such as roommates or people travelling together, will often pay for bills which benefit the entire group, and expect to be paid back at some time in the future. After people pay for other people's bills a number of times, keeping track of who owes who by hand can become difficult. If we can succeed at our goal, keeping track of these expenses can become very easy.

Our objectives for this problem are two fold. First, we hope to create an easy way for people to keep track of expenses in a group. Second, we hope to provide an easy way to consolidate those transactions to reduce the number of transactions that need to be made between the people in the group.

Background

As students, we often live with other students and that includes renting a house together, sharing internet bills, utilities, and many more. Many students face a problem with managing these costs, and often time end up with balances that do not add up.

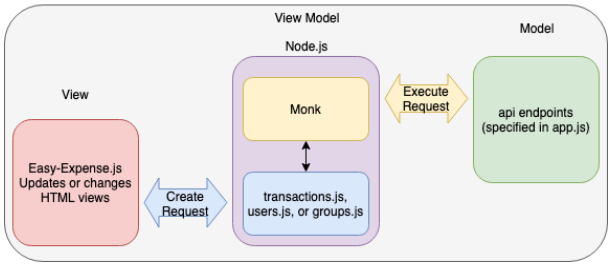
This issue is not just relevant to people who live with roommates, it could also be expanded to splitting costs on a road trip with friends or splitting costs with a group of friends after a night out.

A system that manages your requests and payments with no manual calculation needed would be invaluable to anyone with trouble keeping up with who owes who.

Think of it, what if you owe $10 to John, and John owes you $20? and what if a system would be able to simplify these two requests into one payment where John just pays you $10? Now imagine that same process but with 10 people with a varying number of expenses.

Approach

To create our Web App we first looked to what technologies would be appropriate to use. Our first idea was to use Java servlets and JSP pages. Upon further consideration, we decided that this solution was very outdated and would not be very suitable to a modern design. Additionally, Java servlets uses the MVC architecture, which is not particularly interesting and would not provide us with much benefit, since we have all used this style countless times. After some more research, we decided to use a MEAN stack application with a MVVM architecture style. With our decision made for what technologies to use, we got started working on development.



Above is the current architecture of our system. Using MVVM, we are able to separate the back and front ends very well using the Viewmodel as a liaison between the other two.

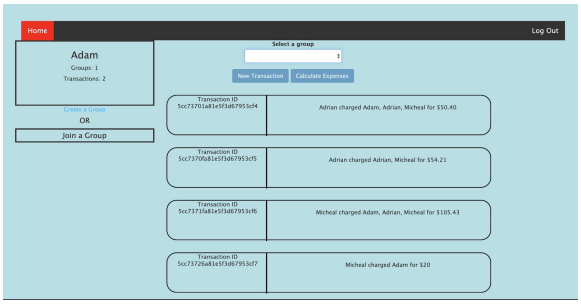
The Viewmodel allows for the View to send HTTP requests through the Viewmodel JavaScript files to the model. The data is then returned to the Viewmodel that binds the data to a variable and allows the View to access it.

```
//Get single user by ID
var User = $resource('api/users/:id', {id: '@_id'});
User.get({id: localStorage['id']}, function(user){
    $scope.user = user;
});
```

The image above is code in the View. A get request is being sent to the Viewmodel which has a handler in the users.js file to send a HTTP request to the API endpoint specified by the User variable above. This example uses the users endpoint, but can easily be translated to transactions or groups by changing what $resource is being asked for. What this means for out application is there is a nice separation of concerns, where the view cannot access the model without asking the Viewmodel to make a connection.

Usefulness

Our main motivation of this tool is to be able to keep track of expenses and easily compute results of a large set of transactions between large groups of people. Both of these goals can easily be achieved through the use of our web app.



The above screenshot shows all transactions relevant to the current user.

If a user needed to add a new transaction they could easily do this by filling out a simple form and selecting users that are involved in the transaction.

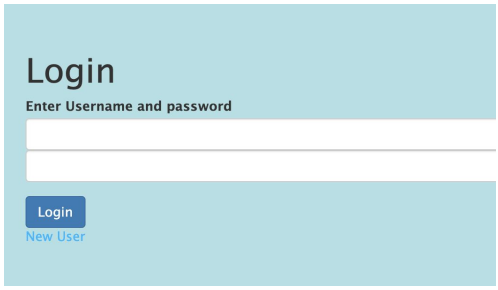
Finally, at the end of the month, the a user can create a month in review which consolidates all transactions in the database for a single group to the minimal number of transactions needed between users.

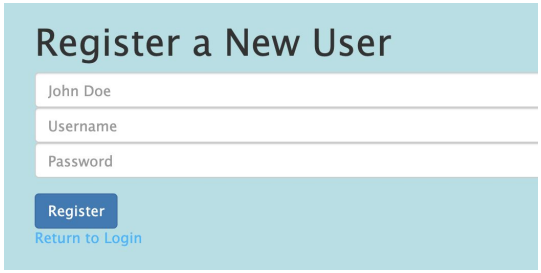
The use of these few pages in our web app accomplishes our original goal of a simple way to track expenses and consolidate a large set of transactions to the minimal number of transactions possible.

Evaluation

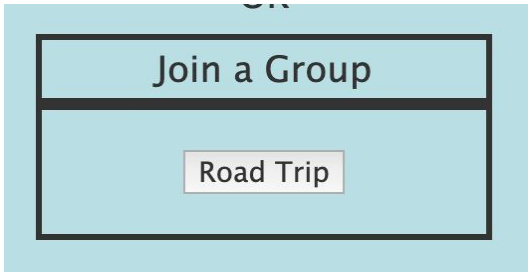
To evaluate the effectiveness of our solution we conducted three users tests with potential users of the system. We had the users complete three tasks with no help or hints from us other than the prompt for what they were supposed to accomplish.

The first task was to create a new account and login with the account. All users were able to complete this task with no issue.



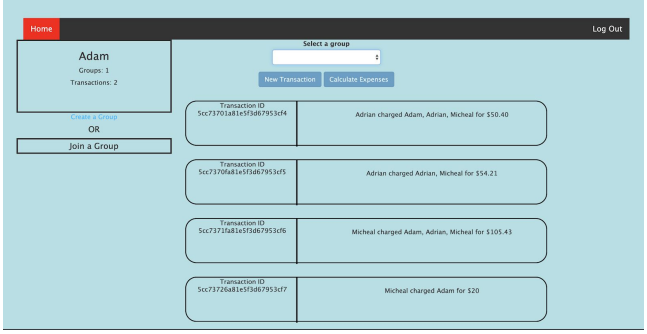


The second task was to join an existing group then create a new transaction. All users were able to join a group. However, we did encounter an unexpected result. When the user clicks to join a existing group, the page does not refresh. This caused some confusion among the users, as they did not know what to do next. We eventually had to tell the user to refresh the page in order for them to continue the task. This is an error that needs to be investigated further. In the final version of the project, this will be fixed. Once the page was refreshed, the users had no issue in adding a new transaction.



The third and final task was to calculate a month in review for any group they wanted to. This task uncovered a strange issue that we did not expect. The first user was not able to figure out what button to click and ended up taking over a minute to find the button and complete the task. The second user knew what button to click, but did not know that they had to select a group before they could click the button. When the user clicked the Calculate Expenses button

without selecting a group, the user was redirected to the login page. It took three attempts before the user figured out to select a group before trying to calculate the expenses. User 3 had no issues with this task.



In summary, all users agreed that the web app was easy to use after they had a chance to use it once and thought it would be a viable solution to keeping track of their expenses. Overall, we take the results of this user study as positive. A few bugs were found, but there was not any major issues to be found with the general use of the web app. With more time for development we believe we could further make the application even easier to use.

Conclusion

Our application provides users with a easily solution to keeping track of their expenses. With our application users can now track expenses and easily compute results of a large set of transactions between large groups of people. The overall, contribution of this project is a simplified system for the everyday person to be more on top of their money. Additionally, for us as developers, we were able to gain experience with a modern architecture that is used in the real world. We feel that this is an invaluable experience that would be beneficial for anyone looking to further themselves as a developer.

Future work

What we have done so far is a proof of concept that definitely shows potential. Our next step would be to scale this application to meet current standards and expectations. With our current decisions for our language and Framework, we could easily scale this application given a necessary amount of time.

With our current technologies, we could translate this web app to a mobile platform. Node.js allows a very convenient way of building mobile apps which allows us to have a more unified system. This idea allows for making finances easily accessible and managed. We do hope to eventually have this web app hosted somewhere that it can be used by at least us and our friend groups.

We feel there are several other features that would be beneficial to have in the web app, such as email invites for groups instead of selecting from a list of groups. Additionally, we could incorporate services like Venmo or PayPal to let users pay each other within the application. The integration of third-party APIs is something we have been considering since the early stages of our idea and is something we definitely look forward to as the future of this application.

Given another month of development time, we would like to implement a few more features. First, a email service to invite users to groups. Second, allow for users to save monthly review results to access them later. Third, add a date parameter to the calculate expense button which would allow users to make a expense report for a certain time frame. Lastly, make the user interface better by adding more CSS to the application to provide a better user experience.

Division of Labor

Initially, we had agreed upon a vertical slice at developing this project but, with the time given and complications faced during the planning phase, we decided to go to a more horizontal approach to avoid conflicts on code and development.

The division of labor was delegated based on strengths. As Adam and Adrian have had experience with both MVC and MVVM architectures, they knew a lot of the technical aspects of this project. Micheal had a lot of experiences in overall design and architecture that he took charge of the designs, features, and documentation.

To further elaborate and give an idea on this, we have created a list of tasks to accomplish for each member.

Adam	• Backend Development • Expenses Summary Algorithm • Transactions feature • Group feature

	• Bug Fixes
Adrian	• Front-end development • Transaction Deletion feature • Amazon Web-Services Deployment (AWS) research • Bug Fixes
Micheal	• Deployment research • Documentation • Architectural development