6170
Fall 2014
</tracker>: A new way for budget tracking

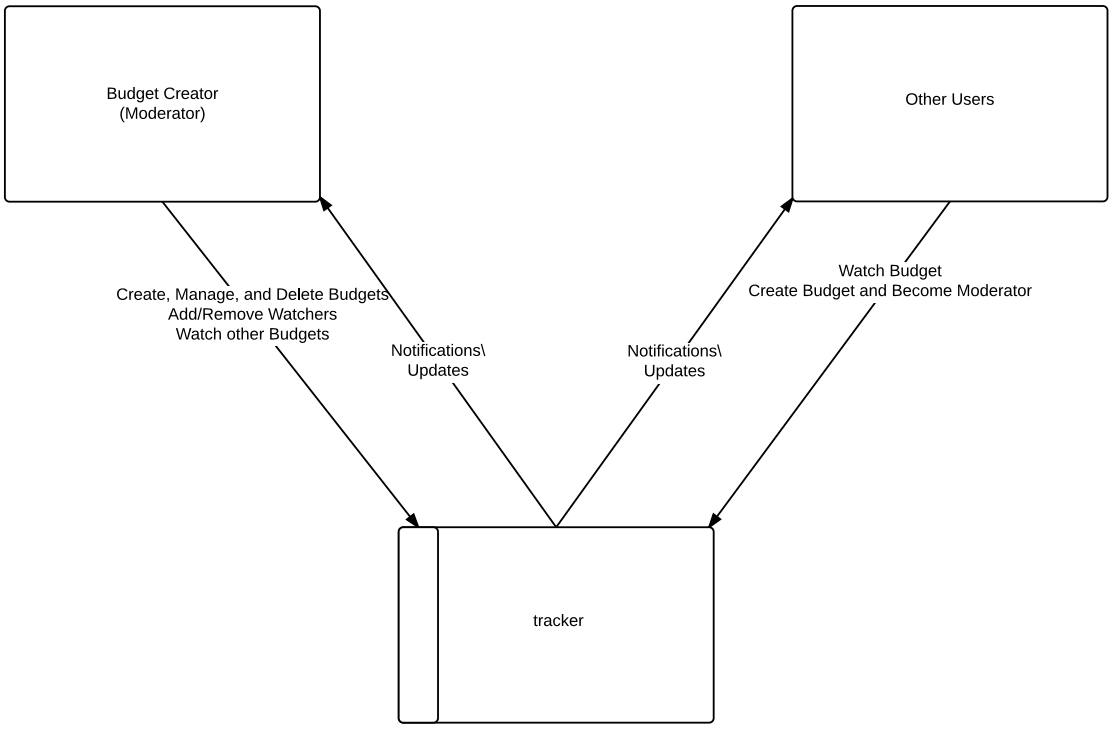
Motivation

btracker is an online application that allows for multiple users to create and maintain budgets. It supports creating budgets, following budgets, updating budgets, labeling purchases with section tags, and closing budgets with a final budget analysis. It does not sync budgets with credit card/bank transactions or sync with Gmail, Apple Mail, or other 3rd party organization tools.

The purposes of tracker are:

- **Create a collaborative environment for maintaining budgets.** Managing budgets should never be a one man job. We plan on creating an environment that allows for a user to create budgets, share to other users, and allow those other users to view and request edits for the budget.
- **Simplify the labeling of purchases when adding items to a budget.** tracker uses a section tagging system to label budget items with one key word. This allows for easy organization of where money is spent in a budget. For example, if you have a monthly budget of income, you can label all of your spendings from Shaw's with a 'food' tag.
- **(Potentially) provide visualization templates for created budgets.** With data provided by the users, tracker could potentially provide templates for visualizing budgets for data analytics. If will use the tags mentioned above for easy search queries.

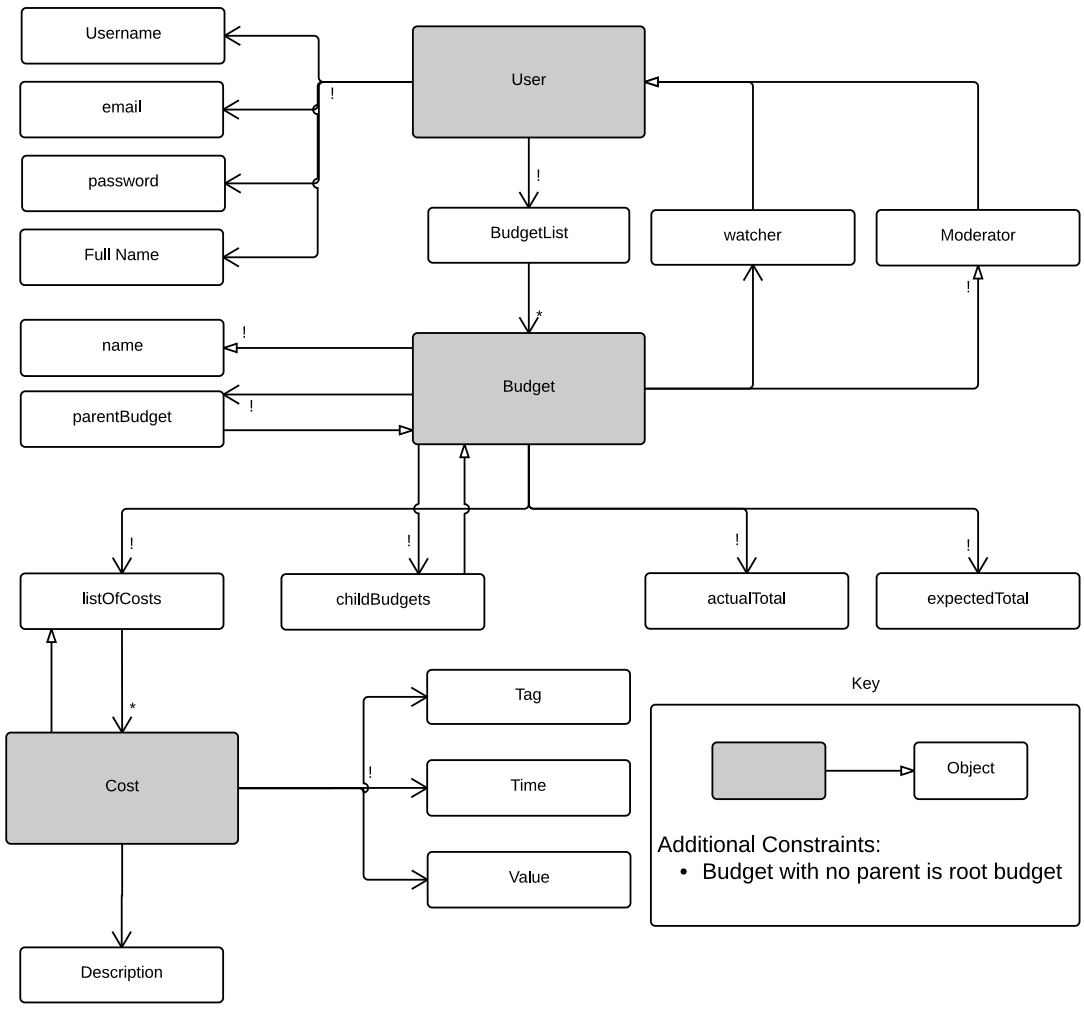
Context Diagram



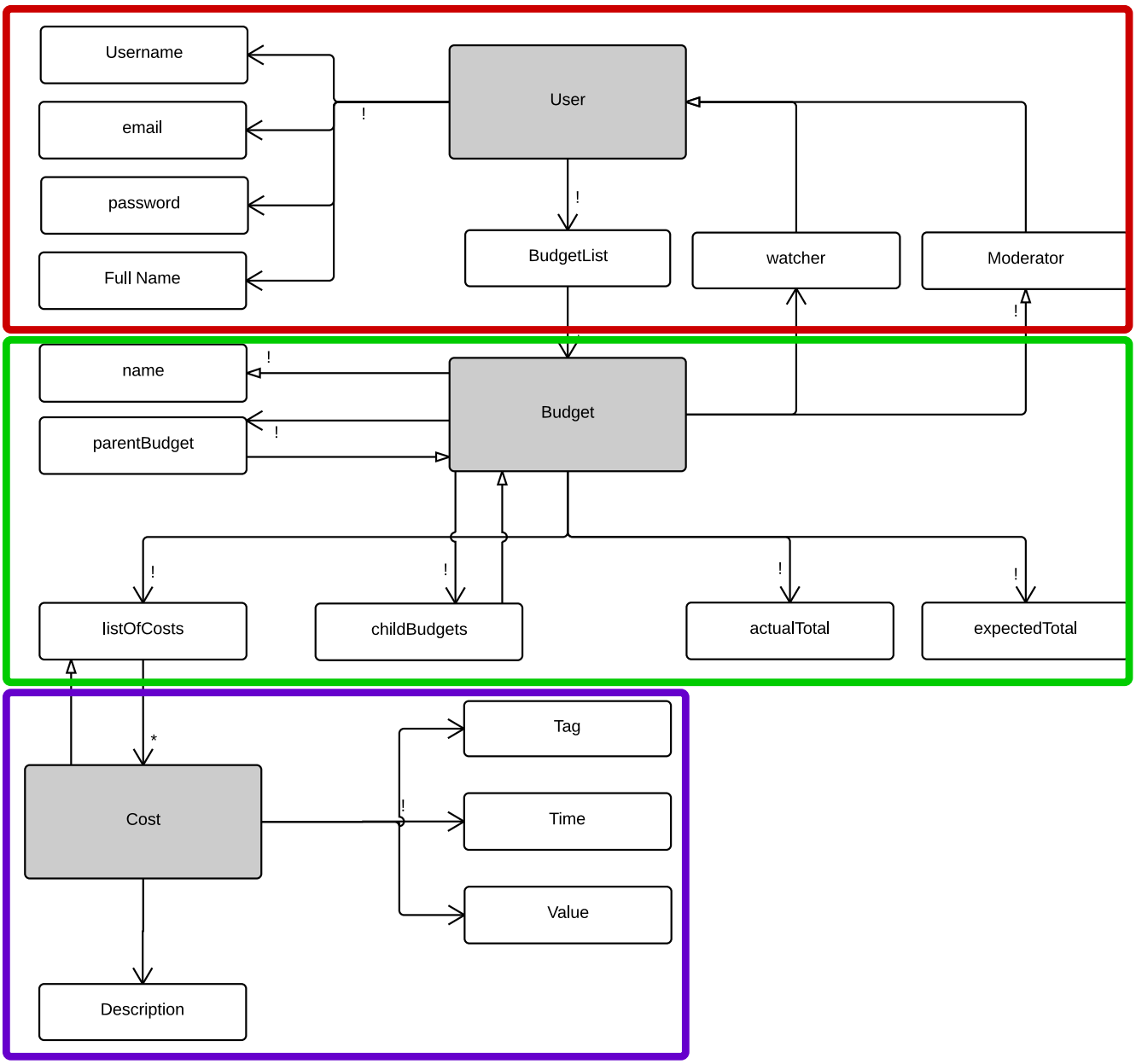
Concepts

Tags: Allows for better, more efficient organization in a budget. Each budget will have a list of section tags with it that corresponds to different sections of spending in the overall budget. These 'sections' are made of smaller budgets and keywords associated with it.
Cost Item: An Item represents an individual purchase that should be accounted for in the budget. All items will have a name, a short description, and a 'tag' to be added in a specific section bin in the budget. All items start with the section tag 'misc.'.
Shared Budget: A shared budget is the representation of the budget being created. It allows for multiple people to view and/or contribute to a single budget.
Budget: A collection of costs with an overall expected total. Each budget will have a name, expected total, and list of costs associated with it. Will also have exactly one moderator that can edit and 0 or more watchers that can view and suggest edits.
Watchers: Watchers are users that are not moderating a specified budget, but are still getting updates and suggesting edits to a budget. It is assuming that if a user is moderating a budget, he/she is also watching it. Users are allowed to watch multiple budgets.
Moderator: Moderators are users that have edit permissions for a specified budget. There is exactly one moderator per budget, but a user can moderate multiple budgets.

Data Model



Data Design



User {
  fullName: String,
  username: String,
  email: String,
  watchedBudgets: [{type: String, ref: 'Budget'}],
  moderatedBudgets: [{type: String, ref: 'Budget'}],
  password: String,
}

Budget {
  name: String,
  admin : {type: String, ref: 'User'},
  watchers: [{type: String, ref: 'User'}],
  parentBudget: { type: String, ref:'User', default: ""},
  childBudgets: [{type: String, ref: 'Budget'}],
  expectedTotal: { type: Number, default: 0 },
  actualTotal: { type: Number, default: 0 },
  listOfCosts: [{type: String, ref: 'Cost'}],
}

Cost {
  name: String,
  value: Number,
  time: String,
  tag: String,
  description: { type: String, default: "" },
}

Design Challenges

How often should a budget update when multiple people are collaborating? This would be a more dire concern if the budget was a real time update, instead of a 'per budget item created' structure, but is still a concern.

Potential Solution:

- Send a push notification to all users viewing/contributing to the budget when a new budget item is added to the budget.
- Real-time updating of the budget. This is easier if the moderator is the only one with edit permissions.

What happens if multiple people are attempting to edit a single budget item? With any collaborate web application comes concurrency issues. The potential solutions to this problem can be also applied to any other potential concurrency issue in the application.

Potential Solutions:

- Multi-treading. I am under the impression that javascript does not support multithreading, but I am not familiar with 3rd party frameworks that could support this.
- Only the original author can edit the create item. This does reduce collaboration abilities, but removes a large burden in regard to potential concurrency issues.
- When someone edits a budget item, they remove the old one and create a new one (all budget items are immutable). Memory management becomes problem as budgets get larger, and btracker will also have to decide when budget items are deleted/ replaced. Can no longer use object id's for identification because new object is created. Greatly increases the number of database calls for replacing an object.
- Simulate the Github model. Have only the moderator have the ability to officially edit the budget. And watchers are allow to 'commit' edits to budget, with the moderator accepting request when done correct.

Should cost be associated with more than one budget? The cost item is created while focused on a budget. This is primarily a scoping design challenge.

Potential Solutions:

- Allow for filtering using the tags. This allows for both aspects of the tag scoping to be shown.
- Constrain to one tag per cost item. This removes the potential of confusing user of where money is being allocating and accidentally double budgeting.
- Create the cost item in an independent field. After it is created, the user currently logged in has the option to add the cost to any budget that he/she is currently moderating. Separates the cost and budgets objects in regard to dependency and allows for cost to stand as their own feature. Creates slight confusion on what should come first (chicken/egg).

What happens if a user deletes his/her account while they are moderating budgets? With our current design strategy, only one moderator exist per budget and cannot be changed. How do we deal with the now 'idle' budgets?

Potential Solutions:

- Ask moderator to select one of the watchers to become the new moderator. Allowing for constant editing to take place.
- Remove the budget as well. This would keep the idea of one moderator per budget and also removes the garbage data, since the budgets are only viewable now.
- Leave budgets and consider them 'closed.' This allows the design to keep the idea of a closed budget, which means that the budget has been completed and will no longer be edited. With large amounts of budgets this can lead to a memory issue, which can be solved by putting an expiration date on closed budgets.

What should the URL structure be for the cost object? Although it is a separate object, each cost is directly associated with a budget, and will not be seen as it's own entity in tracker. So how should the URL reflect this?

- All cost actions will be post and redirect back to the budget. Since the budget is storing all of the associated tags, it should branch off of budget.
- Create it's own branch from root ('/cost'). This illustrates that cost are separate objects, however adds difficulty in drawing the relationship between cost and budget items.

Minimum Viable Product

To illustrate the basic functionality of tracker, it will have the following features and constants:

- Basic user login implementation.
- Only the creator of the budget can edit the budget.
- Moderator adds users that can watch the budget.
- Creating cost objects and adding them to a budget.

Lessons Learned

While building tracker, we learned that API design takes a lot more focus and thought than anticipated. Throughout the 2nd week, we saw ourselves adding and removing a few methods, which on this project doesn't mean much but can be much more damaging for a larger scale project. Communication is key, we did a good job of keeping in contact with one another and making sure that everyone is on the same page in regard to the project design and functionality. Lastly, we learned how difficult it is to stay in the scope of the project. It is easy to want to add features, and you never know how long it will take until it's 3am on a Sunday and there is 'just one more bug to fix.'