Arun G. Rangarajan

12.07.14

UrBudget: Design Documentation

438 Leverett House Mail Center, Cambridge MA 02138
Phone: 617-462-1039 E-Mail: arangarajan@college.harvard.edu

Introduction

UrBudget is a simple Ruby on Rails Web Application to allow users to track how they are spending their monthly disposable income. Using the Chartkick gem to display graphs of user spending, the app lets a user see a neat summary of their current financial status.

First, this is a quick overview of the techniques and technology behind the project:

- Database
    a. Sqlite3 development environment
    b. postgreSQL production environment, deployed on Heroku
- Languages
    a. Ruby
    b. Javascript
- Framework + Details
    a. Ruby On Rails
        i. Models
            1. Users
            2. Budgets
            3. Expenses
        ii. Controllers
            1. Application_Controller
            2. Sessions_Controller
            3. Static_Pages_Controller
            4. Users_Controller
            5. Expenses_Controller
            6. Budgets_Controller
        iii. Views
            1. A lot of them
    b. Notable Gems
        i. Chartkick
        ii. Bootstrap

iii. Jquery

iv. Paginate

v. Faker

Explanation of Techniques

Why Ruby on Rails?

I chose to make my budgeting website in Ruby on Rails because I wanted to learn an additional language and framework used commonly outside of class in actual production environments. Rails comes built in with many built in features allowing for the easy creation of MVC –based sites, each of which I took advantage of in my project.

How did I build?

I mostly started the project by beginning Michael Hartl's Tutorial on Rails at railstutorial.org. I completed the user logins, password tests, and static pages (initially) by using Hartl's tutorial, and then ventured off on my own to build the Budget and Expense functionalities. I then worked to add charting features and more relevant displays. Rails has a fairly steep learning curve but I am confident that I now am prepared to implement more advanced and useful apps.

Models:

3 models are outlined below. The relationships between them are fairly simple. The user is the core of the site. Each user has_one budget, which belongs_to the user, and each user has_many expenses, which belong_to the user. I implemented the model this way because it made logical sense to me. The critical data pieces that would be manipulated would be Users, who we can create, read, update, or destroy. Tied to each user would be their budget, which could be created or updated, but never destroyed, as well as their expenses, which could be created or destroyed. These two secondary models all depended on the user's model which I captured in these rails relationships.

User

id	INTEGER
name	varchar(255)
email	varchar(255)
created_at	datetime
updated_at	datetime
password_digest	varchar(255)

Budget

id	INTEGER
income	float
remaining	float
used	float
user_id	integer
created_at	datetime
updated_at	datetime

Expense

id	INTEGER
name	text
category	text
value	float
user_id	integer
created_at	datetime
updated_at	datetime

Controllers:

Each controller has a purpose within MVC. ApplicationController, which is included within every controller, holds a helper ("SessionsHelper") that contains many methods we call elsewhere in other controllers as well as defines a private logged_in_user action which we'll use across other controllers. StaticPagesController only configures options for the home page determining on where in the login/signup process the user is in. UsersController requires the user to be logged into his/her profile with a completed budget prior to any CRUD actions. It passes the parameters necessary for creating, updating, editing, and deleting users into the relevant views & html forms, as well defines various definitions which we use frequently. BudgetsController requires users to be logged into their profiles prior to any CRUD action. Much like UsersController, it passes the parameters necessary for creating and updating a user's budget, as well as adding a few math options. Expenses requires users to be logged in to their correct account prior to any actions being taken, and actually alters both budget and itself with CRUD actions so that adding expenses interacts with the user's budget. SessionsController has been derived from the login section of Michael Hartl's tutorial.

Views:
There are various HTML Forms made with Rails' Form For helper, as well as various static pages and a few dynamic pages (/users/show.html.erb and /static_pages/home.html.erb. One interesting design choice here was creating the dynamic homepage. This was done because I wanted the expense form to appear in the same screen as the expense feed, and not have a logged in user able to access the "sign up" page and button.

Structure:
Most of my design optimizations such as partial forms, shared views, and declarations are derived from the Rails Tutorial by Michael Hartl, as many Rails conventions and optimizations were new to me prior to the project.

Thank you for reading,

[illegible]

Arun Rangarajan