Group Quadratic - Brick Basinger, Robert Brennan, Sameer Dhoju, Nikita Drozdovskii, Blake Lewis
Dr. Naeemul Hassan
Csci 387 - Software Design & Development
4 May 2017

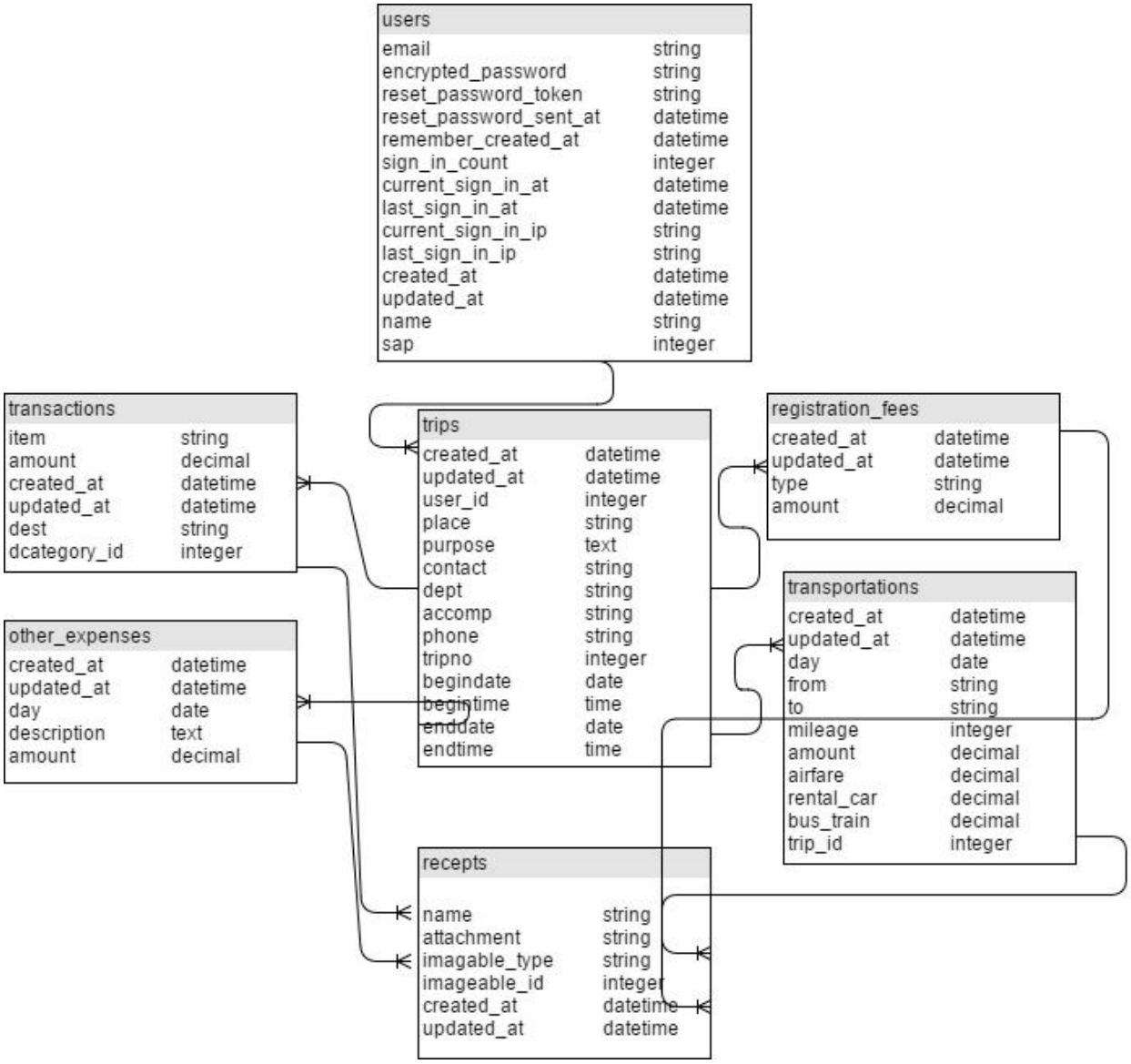
Group Quadratic Project Write Up

I. Database Design and Server Implementation

The beginning of our database design started off rocky. With none of us having any experience using a databases let alone building one, we had to start from square one. We began by finding multiple tutorials online and assigning the database members the task of completing them. With so much information out there this got the ball rolling, but we were still in the dark. We started off with the plan to use Amazon Web Services to host a relational database in the cloud. We ran into problems trying to connect the Ruby on Rails server to the AWS database server. After some more research we found out our problem resided with the rails server not running locally with the database.

We switched gears and decided to implement the server and database on the Ole Miss Turing server. This seemed like the easiest solution, as we were all able to run the database and server on our local machine using basic terminal commands. Even though this seemed like an easy solution, we quickly found out the school has never hosted a Ruby on Rails project. To do this we needed root access every step of the way. We were able to work with Mr. Jeff Lucas to run root commands; however, every time we ran into a problem we would have to troubleshoot and reschedule to try with Mr. Lucas. This became time consuming, and eventually ended with us hitting a wall. We would need to put the root password in a plain text file for everyone to see, which obviously was not possible. This is where we made the pivot to PostgreSQL, and decided to host it on Heroku.

Database Schema



The large-scale logic of schema did not change much since the beginning of our work on the project. The biggest differences between are initial and final schemas are as follows:

- We added Rails-specific fields like *created_at/updated_at* to every table and sign-in related fields to “users” table, like *sign_in_count, current_sign_in_ip, etc.* which were not foreseen at the beginning of the project.
- We removed “totals” table because it was more efficient to have totals calculated when the trip is finalized and inserted directly into the output form.
- We added “receipts” table when adding a receipt functionality.

II. Agile Methodology

At the very beginning of our project, we took two week sprints and accomplished individual goals that were organized by GitHub issues. However, shortly after the beginning of the semester, our Management person suddenly dropped the class. In order to make up for a lack of a management person, we thought that each being self-sufficient would work for our group's needs. This turned out to be extremely inefficient. So after having a "wake up call" with Armando, we decided that we would have a weekly SCRUM leader who would decide what each person worked on for the sprint. Then the next SCRUM leader would be decided and do the same thing by assigning weekly tasks for each person.

Our group realized that pair programming among the database and frontend teams made for a much more efficient work flow. Depending on the issues assigned for the week, the database people and frontend people would meet separately to work on the issue for that week. Once the paired programming subgroups met, we would meet as all together at least once a week. In these bigger meetings, the subgroups would get everyone up to speed by briefly going over what they had worked on in their prior meeting. This process helped us thrive in terms of staying current with the overall development of the project and also kept us from falling behind.

III. Software Testing

Beta testing was not one of our main priorities. If we the product was working and the minimum results we wanted were being outputted, then we quickly moved to the next phase. In retrospect, if we were allowed more time for this project, then we would be able to more thoroughly test our product for our clients as well as make the design a bit more user friendly. We also took less time doing any kind of automated testing. Consequently, the bulk of the testing we utilized was Development Testing, since we did not have a separate team for testing and we didn't do user testing.

Our testing system ran a little like this: when a new feature was added to our application, we would each test different components and units of the feature to target and fix bugs. These bugs would then be incorporated into the next week of issues to be assigned to each person. Any bug that could not be fixed individually would be brought to paired programming where two minds could be used to solve it instead of one. If somehow that method failed, then the whole group would focus on that specific issue, but fortunately the bugs were never too complicated that the paired programming method could not solve them.

IV. Software Design Principles

We started by using some UML diagrams that gave a good outline of how the process needed to run. Halfway into the project, we realized that making a database schema was much more efficient and also one of the required sprints for the class. At that point, we designed a database schema and used that as a tool for modeling our system.

Since we decided to use Ruby on Rails as our framework, much of the design was already taken care of us with RoR. The architectural model we decided to follow was Model View Controller (MVC). MVC aligned really well with our choice of Ruby on Rails because of Rails' views and controllers, and this architectural model was easy for all of us to understand and design around.

V. Requirement Engineering

In order to develop the system requirements for our software, we first needed to fully understand what the user required from the program. At our first group project meeting at the beginning of the semester, we determined that the user requirement is the following:

The software is a mobile responsive web app that tracks various travel expense items which are associated with a trip, and stores them until the end of the trip. At the end of the trip, it compiles the data into PDF State Travel Reimbursement form for signature and submission. Based on the User Requirement we came up with at the first meeting, we then discussed what our system would need to do in order to satisfy the requirements The following are the System Requirements we came up with:

- *Software needs to have login system to accommodate multiple users.*
- *Each user profile needs to accommodate multiple trips.*
- *Software needs to divide expenses into following categories:*
    - *Daily travel expenses, Transportation, Expenditures*
- *Software needs to have a "finalize" button which would process the data into a PDF file in a format of State Travel Reimbursement form.*
- *All the data information needs to be uploaded to University of Mississippi server for future retrieval.*
- *Software should provide statistical data for analysis by University management.*

The above requirements are what guided us throughout designing and developing our software.

VI. Software Evaluation

We finished using most of the software close to the deadline, so evaluating them has been limited. As stated above, we tested our software by trying different inputs in the system to see if they would cause any errors. For front-end, we used bootstrap with html, javascript and

cascading style sheets. This was all incorporated with the coding language of ruby, which was brand new to all of us so a little difficult to utilize.

VII. Technologies Used

- **Front-End**
    - *HTML/CSS*: HTML was used for organizing each page to look exactly how we wanted it to and we used it to do everything from the banner width and height to how each text box would appear on the page. CSS and a small amount of Javascript were used to style all the forms and small details of the pages or "views"
    - *Bootstrap/Sass*: Bootstrap and Sass were implemented in the middle of the project as Gems to help make the site more mobile friendly.
- **Database**
    - *PostgreSQL*: PostgreSQL is the relational database we used for our website. It is open source and worked well with the hosting we used.
    - *Workbench*: We used MySQL Workbench to create the database schema.
- **Serverside**
    - *Ruby on Rails:* RoR was used as the framework and really shaped how we implemented the Model View Controller architecture.
- **Gitlab via Ole Miss Server**

VIII. User Manual

- **Sign Up -** To create an account enter your Name, School ID number, Email address and a password that is at least 6 characters.
- **Sign In -** To sign in use your Email/username and the created password, you can check remember me to stay logged in.
- **Landing Page -** The landing page after signing in has the options to see your trips, return to home, and log out. Click My Trips to begin.
- **Create Trip Page -** The Trips page has the options to Add New Trip, and view previous trips. Click the show button to see previous trips. Click the Add New Trip button to begin a new trip.
- **Add New Trip -** The Add New Trip page requires all fields be filled out. You can put N/A in any fields you do not know the correct information for. Make sure all fields are filled out and select Create Trip.

- **Trip Page -** On the trip page you can add expenses to your trip. You have Daily Expenses, Transportation, Registration and Other Expenses. Select the appropriate option and click either View Expenses to see current added expenses or Add New Expense to add expenses to the current trip.
- **Add Expense -** On the add expense page you should enter the Date the expense occurred. The type dropdown menu has a list of available expenses, please pick the most correct one. The detail section should include the details of the expense. The amount section should include the total amount of the expense including tips.
- **Export -** The export page has two options to export depending on the file type required. Both will download a document to your computer in the selected file format.