# CSCI 387 Final Project Documentation

# Zack Wertz, Chase Rydeen, Jason Stone, Cooper Samples, Abbie Thornton, Ben Corrie, Aaron Moncrief, Abbie Thornton

# Group Factorial

Table of Contents

**Database Design3**

Evolution Over Time3

Full Database Schema5

**Agile Methodology5**

Agile Practices5

Scrum6

Pair/Extreme Programming6

Commit History7

**Software Testing8**

Testing Principles8

Automated Testing8

**Software Design Principles9**

UML Diagrams9

Design Patterns 11

Architectural Models 12

**Requirement Engineering12**

Collecting Requirements12

**Software Evaluation13**

Self-Evaluations13

Peer-Evaluations13

**User Manual14**

Using the Software14

# Database Design

# Evolution Over Time

The design of the database plays a crucial role in the usability of this application. This backend inevitably stores every object submitted by the user, and returns proper information when requested. Limitations on this data must also be observed and followed as to keep the integrity of all the tables of the database. The process of crafting and honing the database to the specific needs of group Factorial took several weeks, and adopted several new strategies as the project progressed.

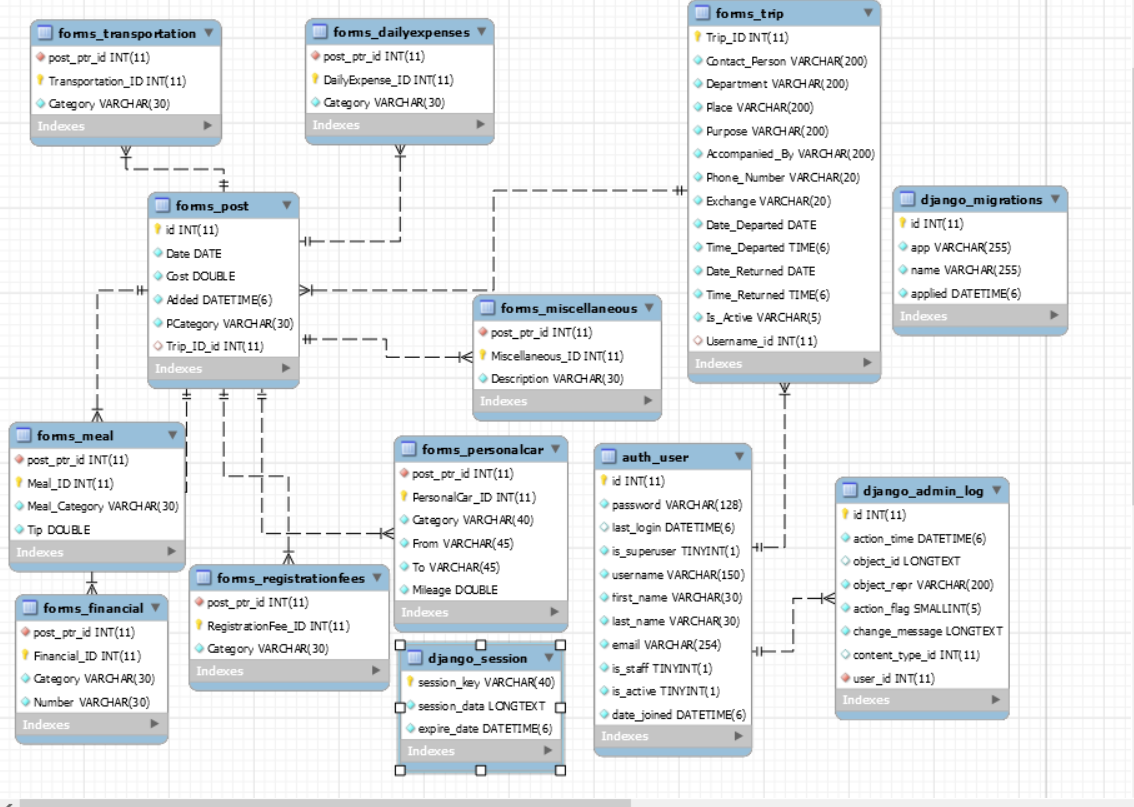
Before the capabilities of the Python based Django web framework were discovered, we installed MySQL Workbench to begin sketching the initial tables sought out by our group. This manifested into a small schema which consisted of a Traveler table, a Trip table, and an Expenses table. At this time, our group knew very little of implementing a starting path for a software project, so this seemed like a good first step. This schema maintained its contents while the group focused on adding templates and getting Django to recognize the correct folder as the templates folder.

After templates were created and front-to-back connection was required by the third sprint, our group collaborated and decided that our current schema would not suffice. We were uncertain of how one table would fare for every single expense type, so group Factorial established separate tables for separate types of expenses. With this schema, we could successfully connect our first ‘Traveler’ through a basic registration html template. This accomplishment warrants a thank you to Django. Django is very efficient at handling models and their views in one spot, thus making it easy to remodel them. Our Traveler table consisted of all the fields of a User: first name, last name, department, email, sap id, and a password. Thus, we declared these fields in our models and then proceeded to worry about front end.

Our Traveler model consisted of six fields originally. We wanted a way for Django to differentiate between users and also be able to indicate which user is currently active. Fortunately, upon migrating models in Django they provide multiple set up tables to assist with development. One of these tables, ‘Auth\_user’, consists five of the six fields we had originally dedicated to ‘Traveler’. So, from here we had two options, we could either extend the Auth\_user model and include one additional field, ‘department’, or simply have the user include the department in another form. We ended up choosing the latter for simplicity’s sake.

At the beginning of the project we assigned each group member the task of constructing a dummy home page. At our next meeting, we showed each other our pages and decided on a victor. This design had a list of recent expenses on the main page for the user’s pleasure. Our once-revised schema worked for us as we proceeded approached this point, but could not bear the burden for this task. We needed a way to connect all the expenses tables in Django, and it doesn’t have an easy way to filter multiple tables. So, after much research we found a solution. It lied in multi-table inheritance provided by Django. We created a ‘Post’ table that would act as the parent table, including fields required by all expenses: date, cost, and a category which adjusts depending on what type of expense is entered. Then we created child tables of every expense type with fields more specific to that table's’ context, and related them back to the parent table. This scheme helped us tremendously. Each model corresponds to its own database table and can be queried and created individually. In other words, you could search for a meal expense in the ‘Meal’ table or the ‘Post’ table, allowing us for much more versatility. This schema provided enough stability of the tables that we could keep this schema for the rest of the project.

# Database Schema



# Agile Methodology

# Agile Practices

This project would not have been possible without the knowledge of Agile Software Development. Our group went out on a limb and trusted the process and in the end, it paid off. Over this semester, our group primarily concentrated on responding to change rather than following a plan. This seemed the most logical to all of us, especially since this is a team-oriented project. Since none of us had any previous experience in development, responding to change allowed us to expand our capabilities and helped us break through some tough bugs in code. A big agile method that complemented this process was face-to-face interaction. Throughout this semester MANY new, difficult concepts were stumbled upon. There were struggles to learn these new problems, but when a group member was present to help explain and guide the struggling member through it, the entire group gained morale. Even a confidence boost from a fellow member can help get a group over a tough task.

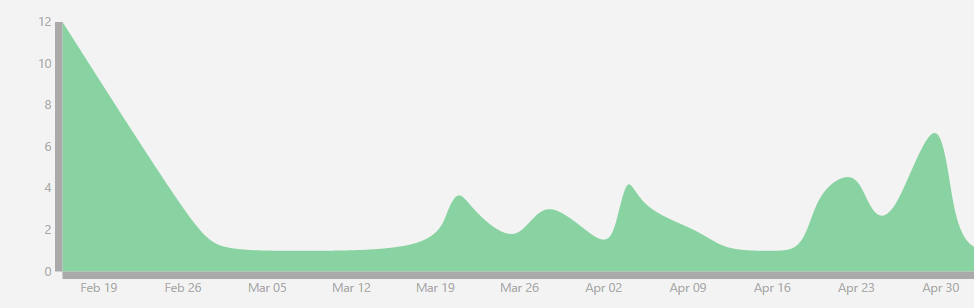
# Scrum

Although our group did not keep up with a burn-down chart or a velocity chart, that does not mean we did not implement practices from the Scrum methodology into our project. At all times our group possessed a backlog of features to be implemented before the final due date. We performed poorly in the aspect of setting deadlines for the backlog list, but they all were completed before turning in our project. Our group did perform well in various other Scrum practices though, including mandatory group meetings and possessing and updating a dynamic product backlog. During group meetings, evaluations of the previous sprints’ work are conducted. After all new material has been presented to the group and discussed, our group would then vote to see if we did want it, if we did not want it, or if we wanted to edit it. At this point in our meeting the group would vote for prioritization of the tasks, and then we would execute accordingly.

# Pair/Extreme Programming

Extreme programming proved to be an essential aspect of our project, another topic learned in class. Our group focused on small iterative releases, lots of manual testing, as well as limited automated testing, and pair programming. This accredits to our final project’s success. The biggest reason we could make continuous progress towards our goal was that we performed small releases constantly to improve the product. These small releases allowed for all members of the group to see improvements and in turn better their understanding of our project. This aspect goes hand in hand with the manual testing that we performed. Our process would be to update a small piece of code, test it with various inputs that would break it, and then from there try to implement the correct logic behind fixing that issue. Once a solution appeared to be completed, other members of the group would be presented the program and try to find ways to break it. The main goal of our pair programming was to get different sets of eyes on different problems and hopefully get us the solution in the shortest amount of time.

# Commit History



The picture shown above displays Group Factorials commit history as per git.cs.olemiss.edu/factorial.

Our groups commit history (77 total) only appears so large at the beginning because everyone in the group pushed test .txt files and we had a small learning curve with Git. Shortly after, our group mainly had meetings to discuss what topics should be researched and how we should go about starting the project. To be honest, true work did not start being applied to this project until after Spring Break. At this point our group quickly realized that we needed to get on top of things, so from March 19th onwards we kept a pretty consistent commit history, until of course we hit deadline week. The last week of the project saw the most commits per day than any of the previous weeks. This can be translated over to our velocity pretty well and shows the relatively steady stream of pushes and the rapid increase as deadlines approached and we focused more on getting our deliverables prepared.

# Software Testing

# Testing Principles

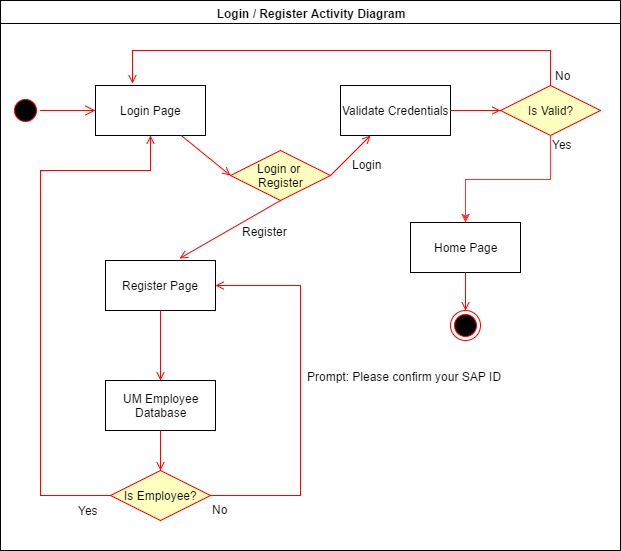
This is an area where Group Factorial may have needed some guidance. It was not until late into the development of the project did we realize that unit test cases should have been created already. Instead we chose to test our software the hard way with various inputs provided manually by our group members. This, clearly, is not a valid way of error checking because one can see how easy it is to forget an input test case. This worked for us to an extent because of the relatively small size of our project, however it is still not ideal. One of our most successful tactics emerged after we began hosting our project on Pythonanywhere.com. Following that date, we asked friends to check out the app and give us feedback. We received feedback on input styles all the way to design layout. This involvement caused us to implement a date calendar widget, something that we think looks much nicer than before.

# Automated Testing

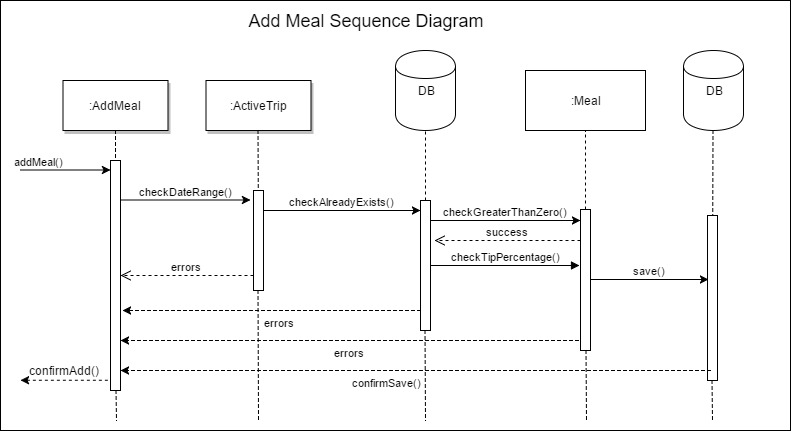
After the presentation and the questions with Dr. Hassan we implemented a new automated testing feature into our project that helped a lot in adding features and checking them immediately, which is something we had struggled with previously. We discovered the python tool called Coverage.py that does automated testing on Python code and were blown away by its effectiveness. When going through and adding error checking features, a quick run of the test cases quickly pointed out any errors or oversights and it made the process of implementing the new error checking features seamless and easy. Coverage.py is a powerful tool and provides a good measurement of test cases to help create good tests to run and it helps increase overall efficiency.

# Software Design Principles

# UML Diagrams



Here is an activity diagram for one of the processes of our application that is important to keeping up User integrity since every single object in our database is linked to a User ID number so the integrity of the login and validation process is pertinent to the functioning of our application. We added in the UM Employee Database check although that has not been implemented into our system yet as that is a next step in the process but we wanted to show it for clarity’s sake.



Here is a sequence diagram for adding a meal to our database. This sequence diagram shows the many levels of error checking that expenses go through before they can be successfully added to the database. A date checking function first confirms that the entered date from POST is within the range given to the Trip at creation. If it passes this step the next step is to check if an expense already exists in the Database by querying it and checking for existence. If it passes here then it goes through two checks regarding the integrity of the amount entered. One function checks to make sure that the entered cost is greater than 0 and the second, which is specific to Add Meal, is a check to make sure that the tip is less than 20% of the total cost of the meal.

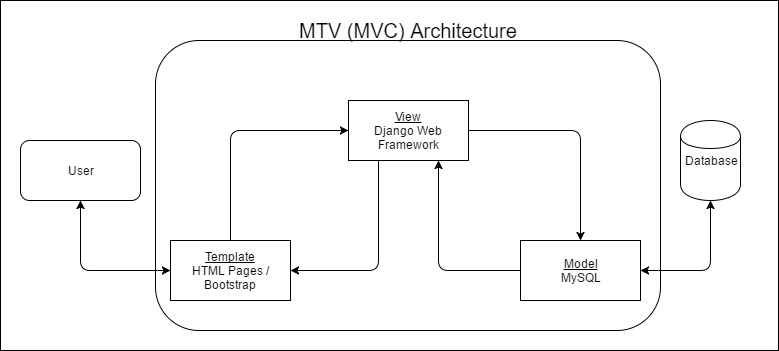
# Design Patterns

The main design pattern that is critical to Django and to our application is the Builder design pattern which is a creational pattern. The foundation of Django lies in its models, a feature we used extensively in our application to keep track of all our data and manipulate it using queries and querysets. The Models are abstracted and they allow us to create many objects using the same construction process to keep model creation consistent for all of our objects and also in our database.

Another important design pattern utilized by our application is Lazy Initialization through the use of our query sets which are used extensively for error testing and checking and to map values after completion of a trip. The query sets aren’t evaluated until they are needed and that feature can be utilized to test for existing values and also calling values to store in the output PDF file.

Lastly, a useful design pattern that is implemented by Django is the Adapter pattern, which is a structural pattern wherein interfaces of a class are converted into other interface that clients expect. Django manages all of its files within itself and the framework links them all together using native connection features. We utilized this property heavily in the implementation of our database since we initially had a strict MySQL set-up but Django manages the database creation itself and does its own migrations since it knows exactly what tables to use by looking at what Models have been created.

# Architectural Models



The MVC architecture of Django. Django prefers to name it MVC and switches up the designations slightly, although it still holds the premise of an MVC architecture. The M still refers to database models, the view of Django actually acts as the controller managing both front and back end, and the templates of Django act as the view, which is where information is displayed.

# Requirements Engineering

# Collecting Requirements

The process of collecting requirements of the system, also known as requirements elicitation, took us roughly seven days as we sought out teachers, the internet, and fellow classmates for answers. Normally during this procedure, we would seek the individual(s) that requested the service from us. However, since this project was issued by the school and had some concrete demands we were unable to collaborate with the individuals who requested the service. Requirements yielded from this program included two types, functional and nonfunctional. Initially for our functional requirements, we prioritized authentication and our basic html registration template working correctly to allow for specific functions, like add user, to behave correctly. Some of our nonfunctional requirements included expenses not being able to be added with dates outside of the trip date range, and that the edit expense button must load the correct form. We created a document consisting of all of our requirements and pushed it to our git. From here everyone could review and analyze the requirements, adjusting them as seen fit or completing them to the best of our groups ability.

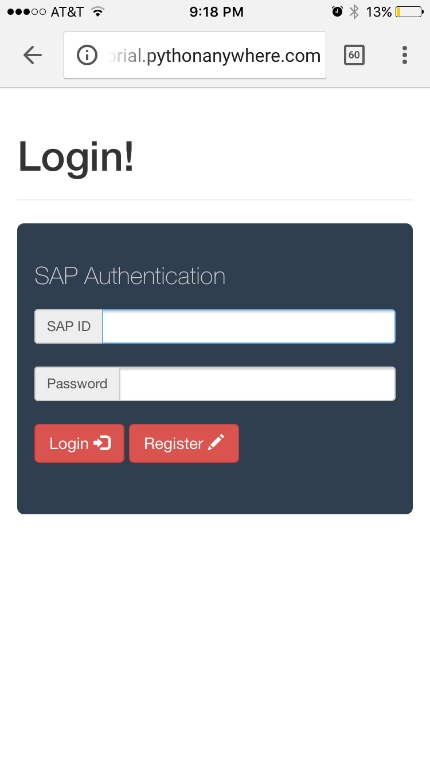
# Software Evaluation

# Evaluations

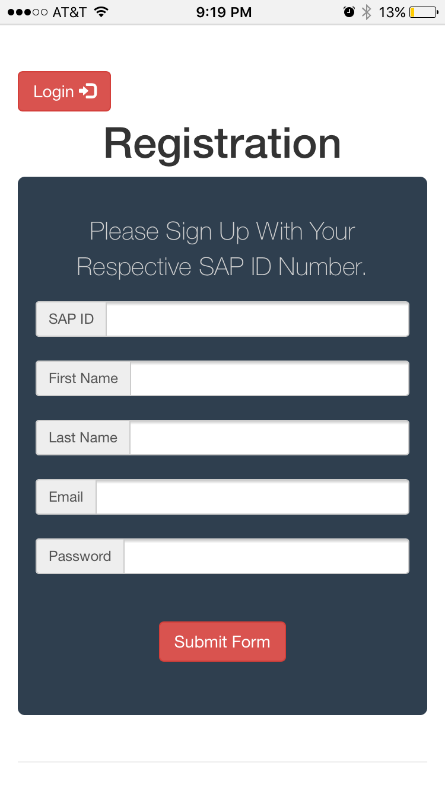
As our group quickly approached the deadline for final presentations, we looked back at our work and decided to evaluate our progress. From a visual standpoint, our project possessed a good color scheme and is easily navigable. However, we missed several key logic errors that were corrected for us by professor Hassan in the final demonstration. This was our first time to evaluate something of this magnitude so most of us were just excited it worked. Not until after the joy of ‘finishing’ did we truly realize that much more must be emphasized to create a piece of software that will be usable by the public. We did not finish the project soon enough to give us ample time to present a Beta testing period to users, which would then have given us more requirements and bugs to fix. However, after the final presentation we went back to our code and analyzed our program from top to bottom. We fixed all the errors we could think of and improved them. Shortly after this we added some basic automated tests to check for completeness.

# User Manual

# Using the Software

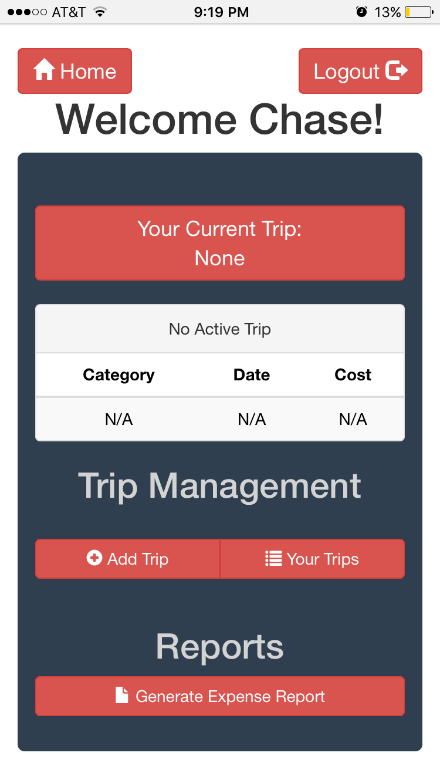


Login Page

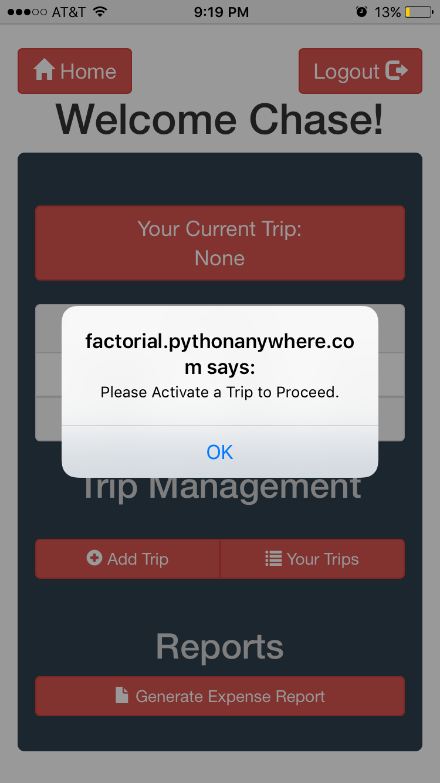


Registration Page

* *Login Page*
  + The login page is what you are greeted with when you start the application. If you try to navigate to other pages without first logging in then you will automatically be redirected back to this page.
  + From this page you can:
    - Login – verifies your credentials and either allows you access to your home page or denies you and displays an error message.
    - Register – Takes you to a different page that allows you to register to use the site
* *Registration Page*
  + If you click register from the home page you are taken here
  + Here you will fill out your information to create an account
  + Once you click submit your SAP ID will be sent to the UM Employee Database to verify employment and once verified an account will be created.

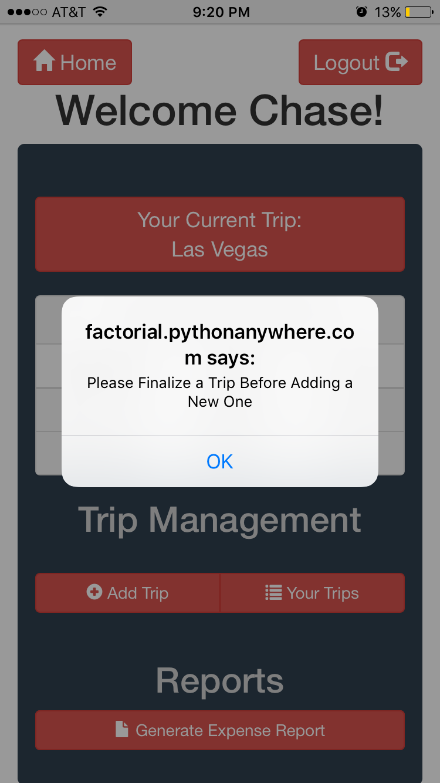


Main Page

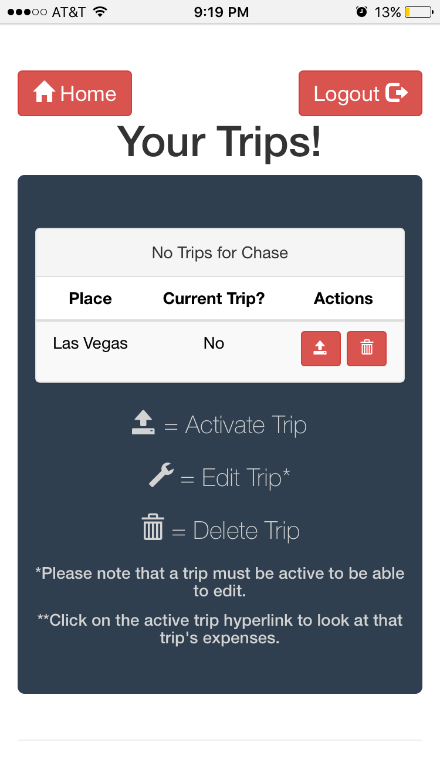


No Active Trip Error

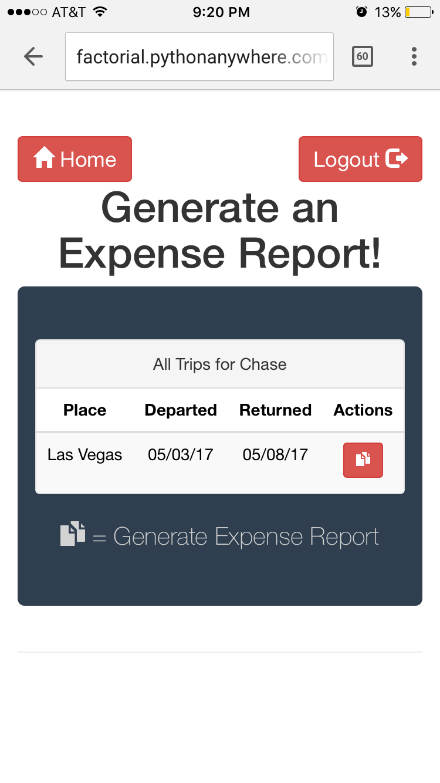
* *Home Page*
  + Once you log in you will be taking to the main page of TripExpenser which allows you to access the rest of the features of the application
  + The most useful here is the “Your Current Trip” button that allows for quick navigation to your current active Trip.
    - If you click this button without an active trip then an error will popup as shown below.
  + From here you can access your main Trip Page (if you have an active trip), Add a new trip (if you don’t already have an active trip), look at a list of all of your trips, or go to a special report list to generate a PDF report of your trip by clicking on the “Generate Expense Report” button



Active Trip and trying to add another trip error

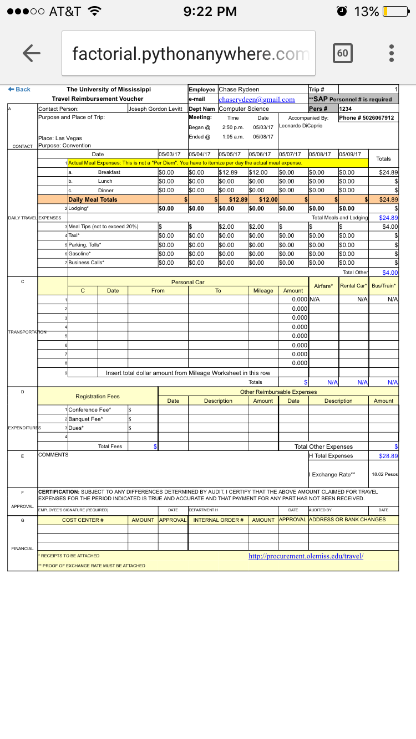


List of all Trips

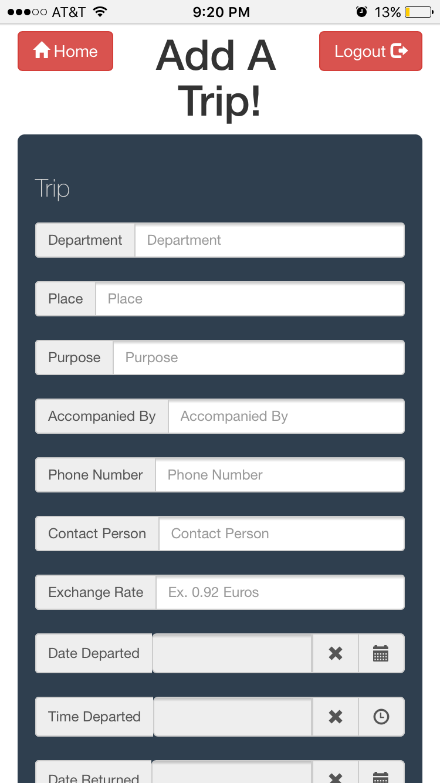


Report List Page

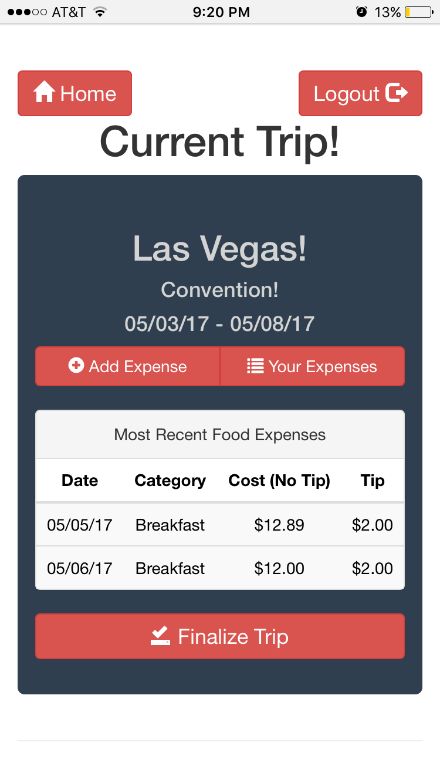
* *Trip List Page*
  + On this page you can look at all of your past and present trips and edit their information
  + If you click on the wrench you can edit the information about a trip but note that can only be done if the trip is active
  + If a trip is inactive you can hit the activate trip button and it will make it your new active trip
  + If you click the trash can next to a trip you will be prompted a confirm delete message
* *Report List Page*
  + This page lists all of your active trips and allows you to view expense reports for all past and current trips
  + You can utilize this functionality on the go to make sure that all of your expenses are showing up correctly on the form and verify everything is correct before you convert it to a PDF



Example of how the Report shows up on mobile

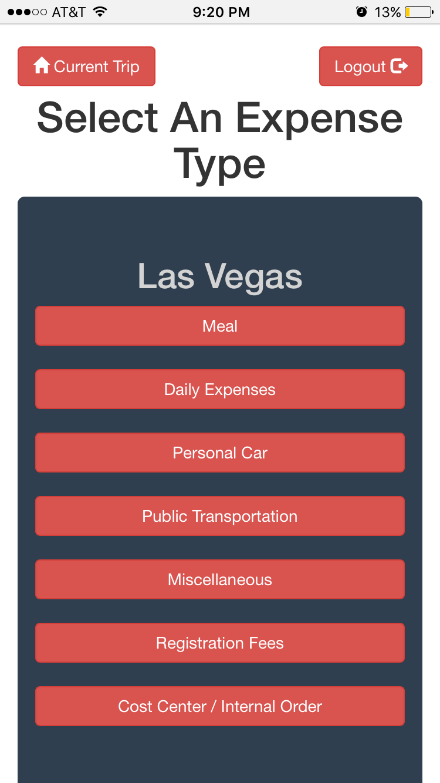


Add Trip Page

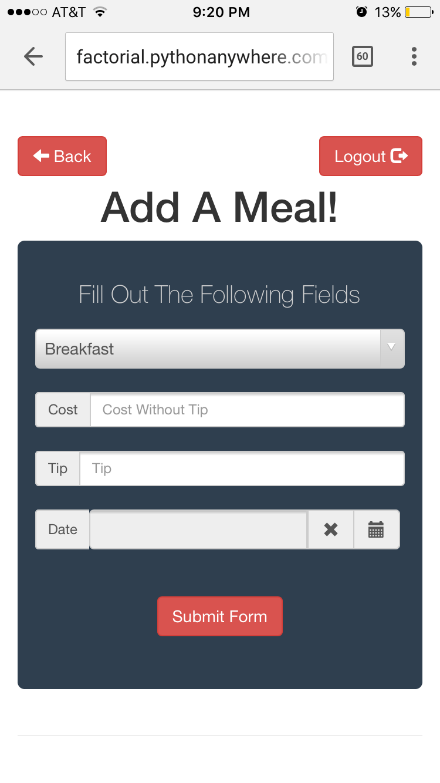


Current Trip Page

* *Add Trip Page*
  + If you click on the add trip page from the home screen then you are taken to this screen that allows you to add a trip to your profile
  + Labels as well as placeholder values indicate how to fill out all of the fields
  + Exchange rate is there so you can note if you go on international travel, if you are travelling within the United States you can just put in a 1.
* *Current Trip Page*
  + This page will show your most recent expenses across all expense categories for quick reference to remind yourself of past input expenses
  + From this page you can return Home via the button in the top left corner, add an expense to your trip, look at a list of all of your expenses for the current selected trip, and at the bottom you can finalize your trip.

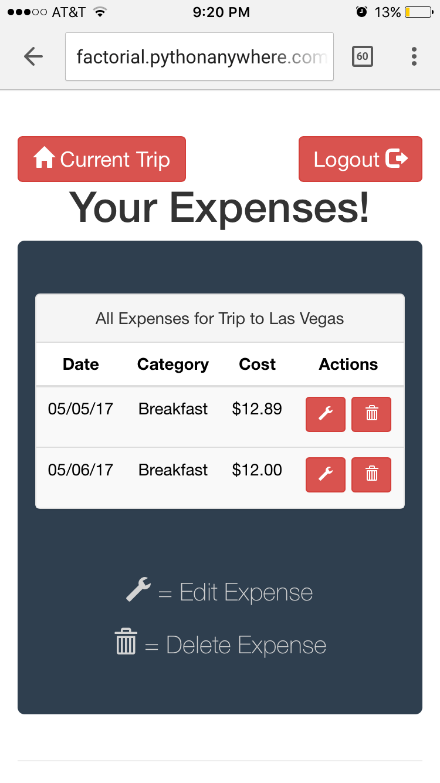


Add Expense Page

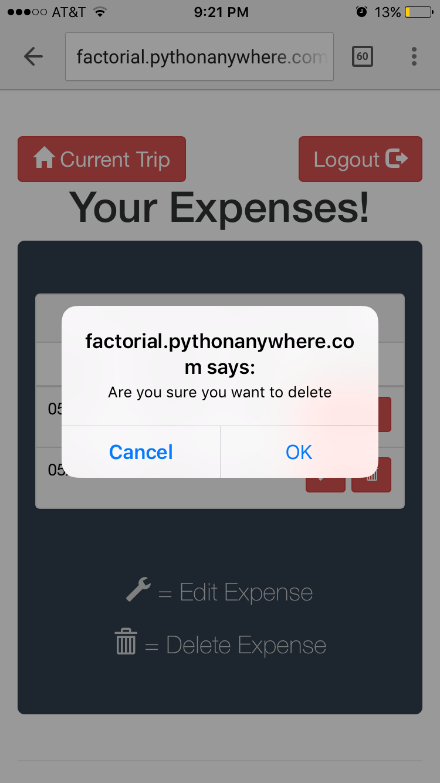


Add Meal Page

* *Add Expense Page*
  + From here you can select what type of expense you would like to add
  + The categories are broken up by what the categories are on the Expense Report Page
  + Meal
    - Breakfast
    - Lunch
    - Dinner
  + Daily Expenses
    - Lodging
    - Taxi
    - Parking, Tolls
    - Gasoline
    - Business Calls
  + Personal Car
  + Public Transportation
    - Airfare
    - Rental Car
    - Bus/Train
  + Miscellaneous
    - Fill out description of what you are doing
  + Registration Fees
    - Conference Fees
    - Banquet Fees
    - Dues
  + Cost Order / Internal Order
* *Add Meal (Expense) Page*
  + On the specific add expense pages are category selectors where you can choose what type of expense it is within that specific category and then select the date and cost
  + There are checks implemented here that confirm that your date selected is within the range of dates of your trip as well as confirming that you don’t have overlapping expenses by accident and you will be warned if such an issue is encountered.

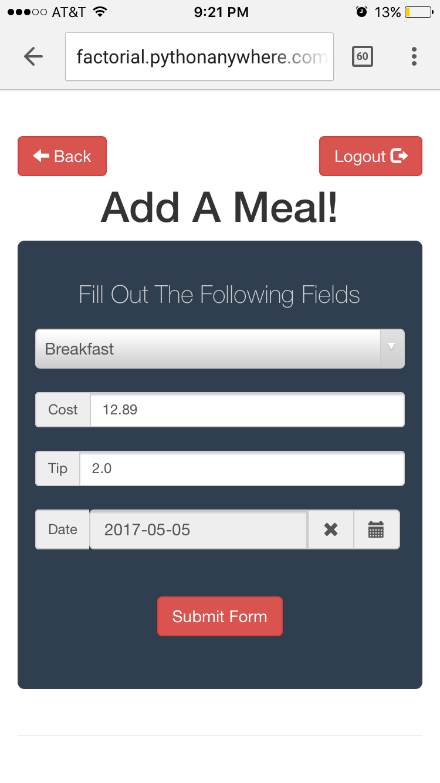


Expense List Page

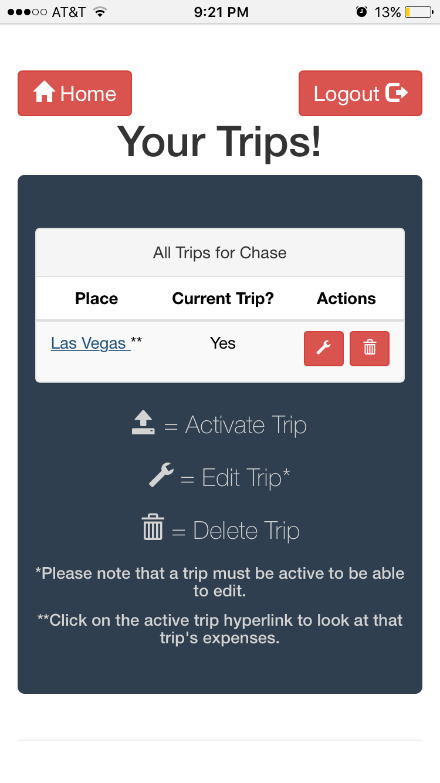


Confirm Delete Message

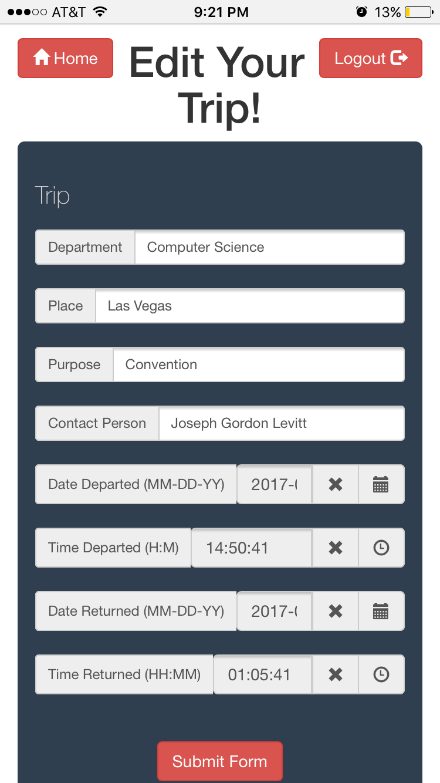
* *Expense List Page*
  + This page will list all of your expenses and allow you to make changes to them if you happened to make an error when you were first submitting the expense
  + You can click the wrench icon and it will allow you to edit specific information about the expense and it will have the added precaution of error checking your input there as well to keep you within the date range of your trip and any other checks needed.
  + If you click on the trash can icon next to an expense then it will pop up a confirm delete message and you must click “OK” to confirm the deletion of the item before it is deleted.



Edit Expense shown with populated existing information to edit



Trip List with Active Trip



Edit Trip Page

* *Trip List Page revisited*
  + Here is the trip list page once again, this time with a current active trip
  + You can notice the difference in that the active button has now changed in to a wrench so that you can edit the trip information.
  + Also a hyperlink is included here so you can click on the trip name and it would take you right to the current trip page to save time.
  + Below you can see the edit trip page with all of the existing information populated and ready for you to change as needed.