# Expense Tracker

Paul Matthews
@pdmxdd
https://github.com/pdmxdd/expense_tracker

## Description

The Expense Tracker allows users to record, and track their expenses. It gives visibility into spending habits over various time periods. User's can create an account, create categories, and add a category to any given expense. This allows them to gain a better understanding of where their money is going.

## Features

- Prospective Users can create an account.
- Users can create, and view categories.
- Users can create, and view expenses.

## Planning - User Stories

**As a prospective user I can create an account so that I can start tracking my expenses**. This entire project is built on having users in the system. I had to plan my database so that categories, and expenses were somehow tied to a given user. This way only authorized users can access their information.

**As a user I can add a category to an expense**. It's not only important to know how much you spent in a given time period, but it is benefical to know what you spent money on. Adding categories to expenses allows users to better understand where their money goes.

## Planning - Database

I have three tables in my database -- Users, Categories, and Expenses. All three have a primary key (id). Categories is linked to Users through the User_ID foreign key, and Expenses is linked to both Users, and Categories through the User_ID, and Category_ID fields.

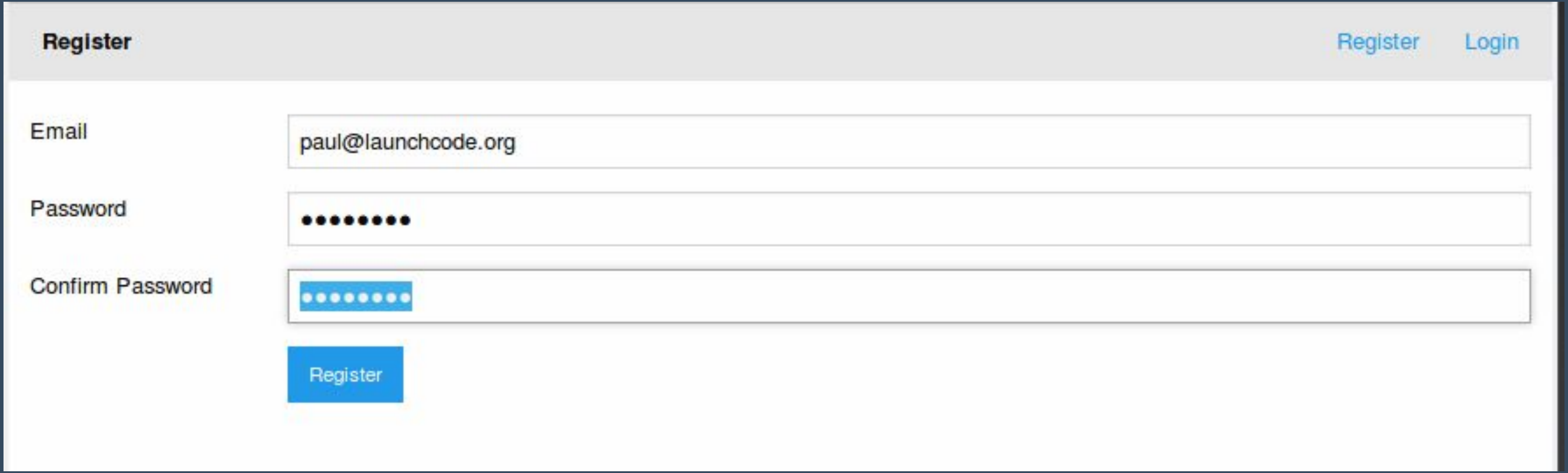
Users have a one to many relationship to Categories, and Expenses. Categories have a one to many relationship with Expenses.

All expenses are timestamped, and have an amount. This allows the ability to figure out how much money was spent in a given time.

## Technology Stack

- Rust
- Rocket
- Tera templating
- Diesel
- FoundationCSS
- Chrono (3rd party crate for Rust that handles time)
- Bcrypt (3rd party crate for Rust that securely encrypts, and decrypts data)

# Demo - Create User 1

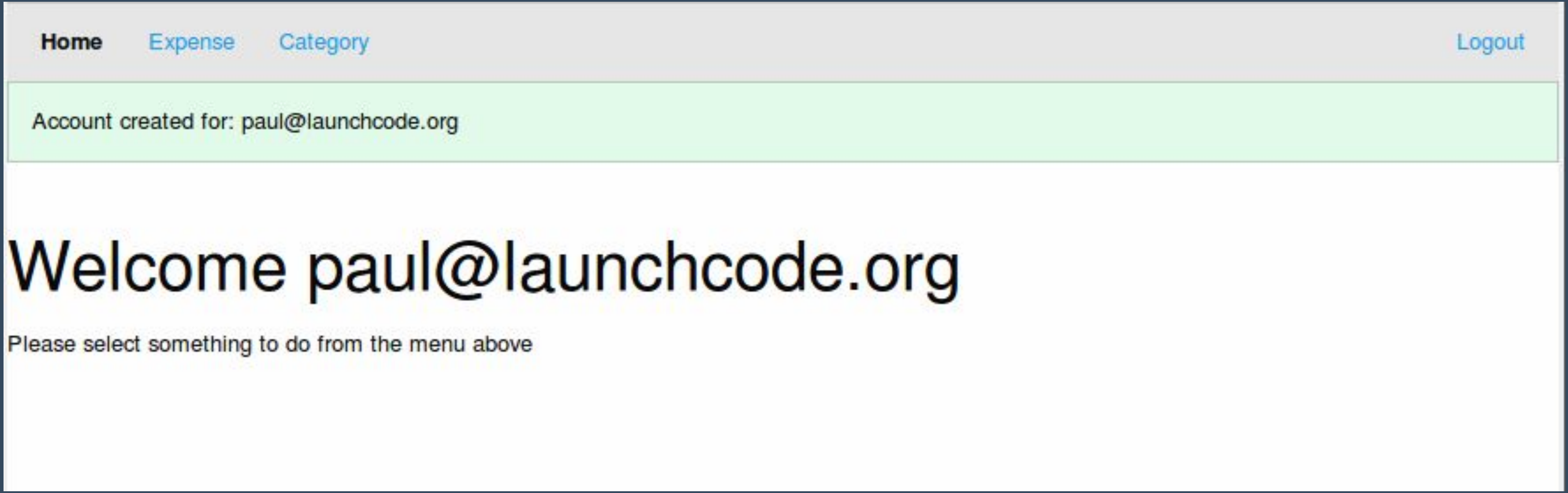


**Register**

Register Login

| | |
|---|---|
| Email | paul@launchcode.org |
| Password | ●●●●●●●●● |
| Confirm Password | ●●●●●●●●● |

# Demo - Create User 2

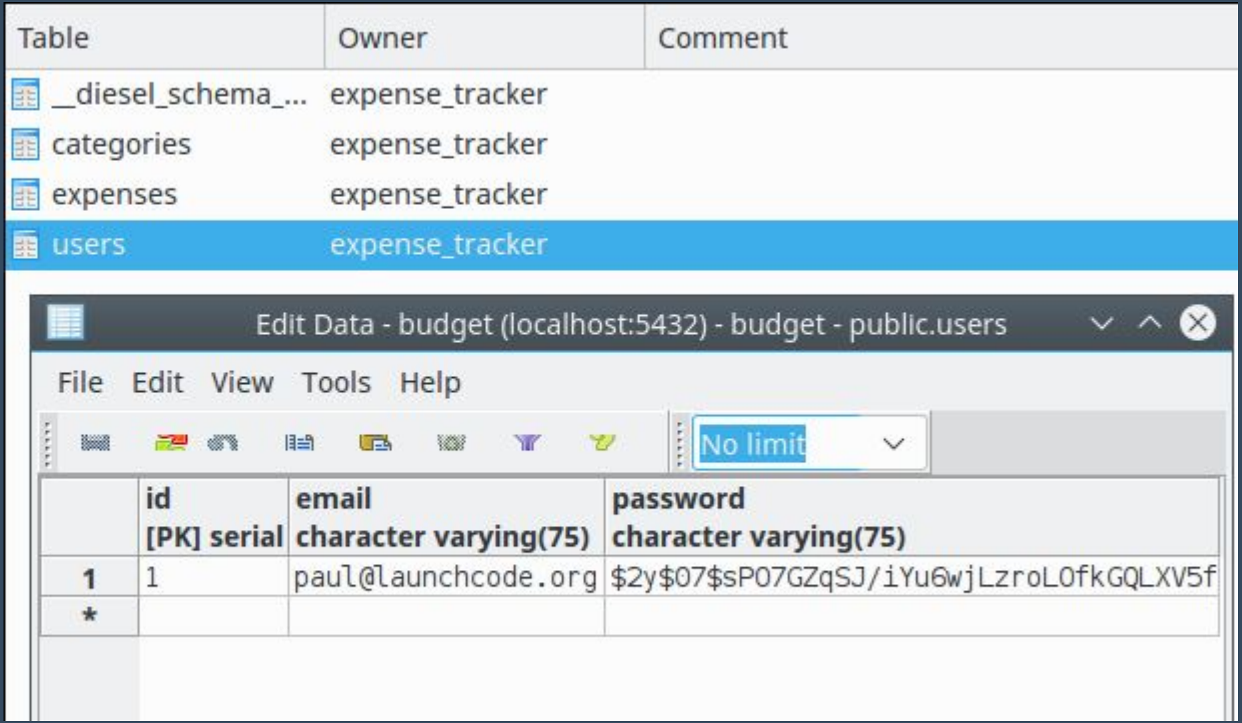


Account created for: paul@launchcode.org

Welcome paul@launchcode.org

Please select something to do from the menu above

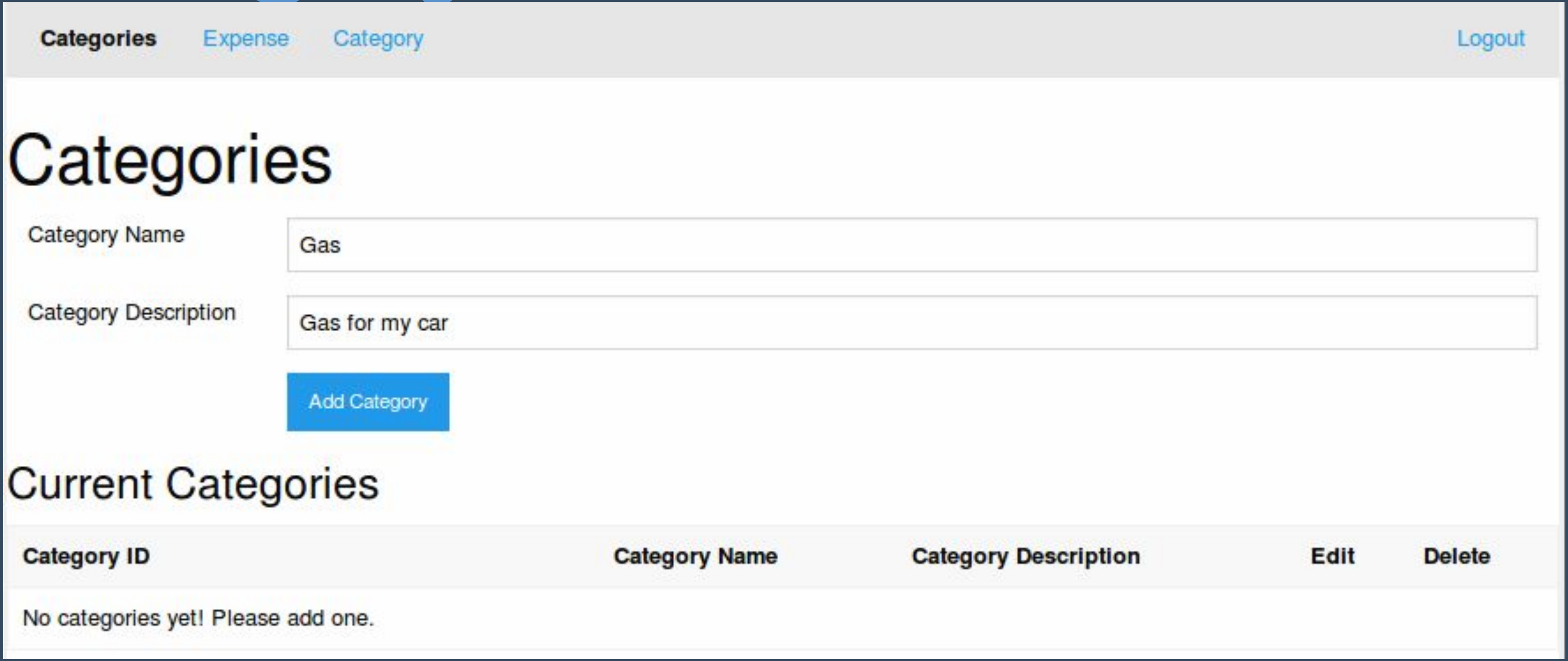
# Demo - Create User 3



| Table | Owner | Comment |
|---|---|---|
| _diesel_schema_... | expense_tracker | |
| categories | expense_tracker | |
| expenses | expense_tracker | |
| users | expense_tracker | |

| | id<br>[PK] serial | email<br>character varying(75) | password<br>character varying(75) |
|---|---|---|---|
| 1 | 1 | paul@launchcode.org | $2y$07$sPO7GZqSJ/iYu6wjLzroLOfkGQLXV5f |
| * | | | |

# Demo - Create/View Category 1



Categories

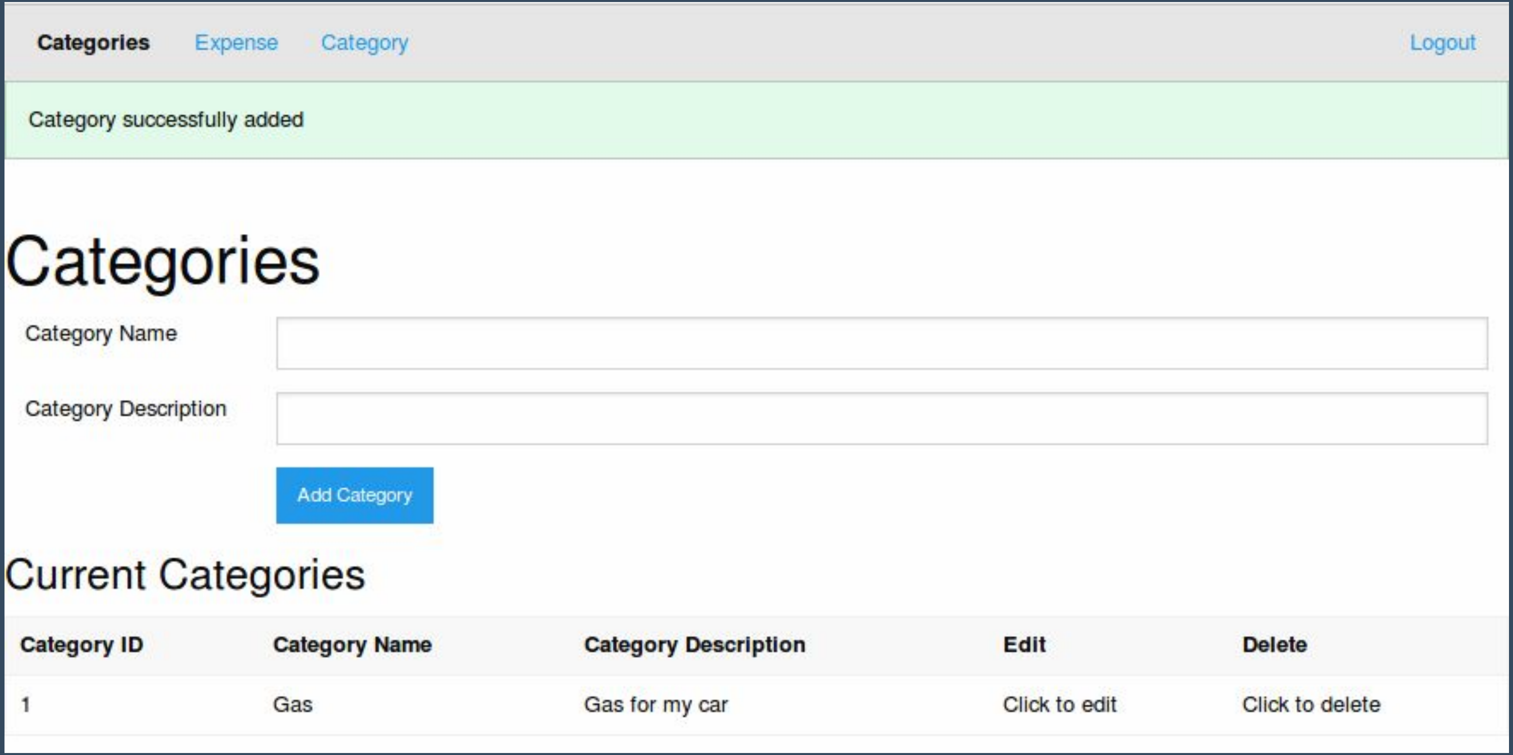
Category Name: Gas

Category Description: Gas for my car

Current Categories

| Category ID | Category Name | Category Description | Edit | Delete |
|---|---|---|---|---|
| No categories yet! Please add one. | | | | |

# Demo - Create/View Category 2



Category successfully added

## Categories

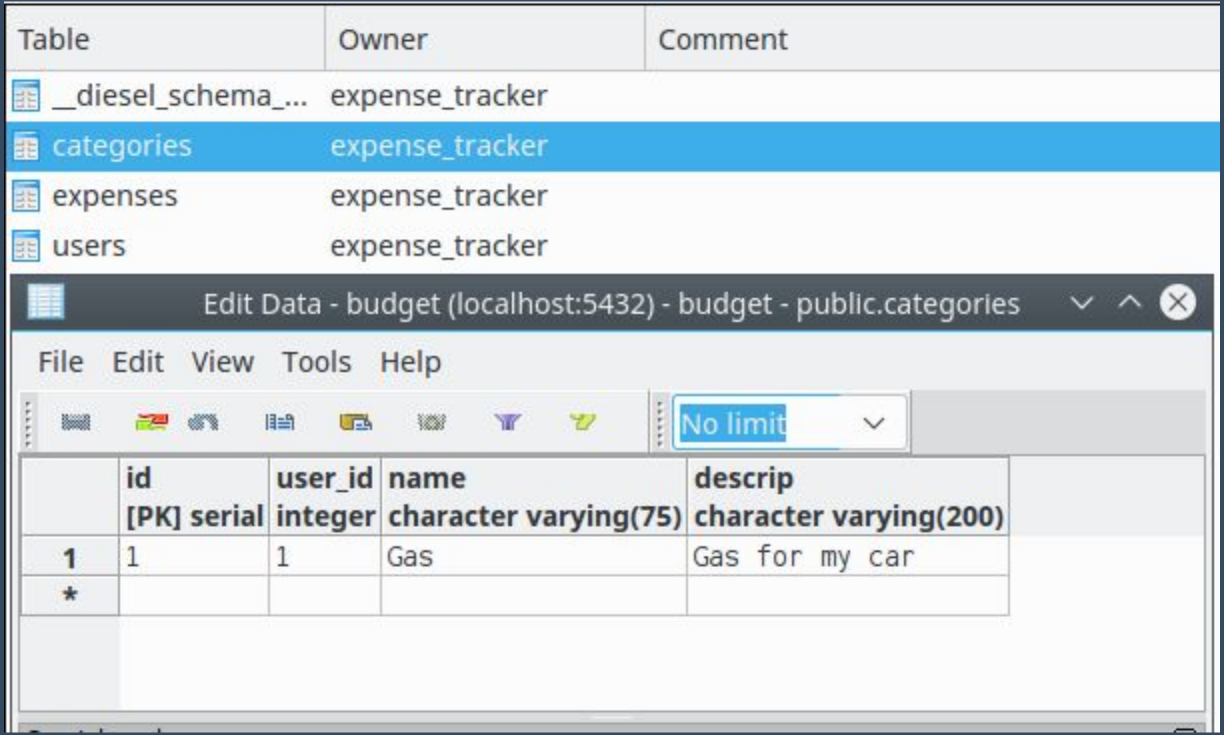
Category Name

Category Description

### Current Categories

| Category ID | Category Name | Category Description | Edit | Delete |
|---|---|---|---|---|
| 1 | Gas | Gas for my car | Click to edit | Click to delete |

# Demo - Create/View Category 3

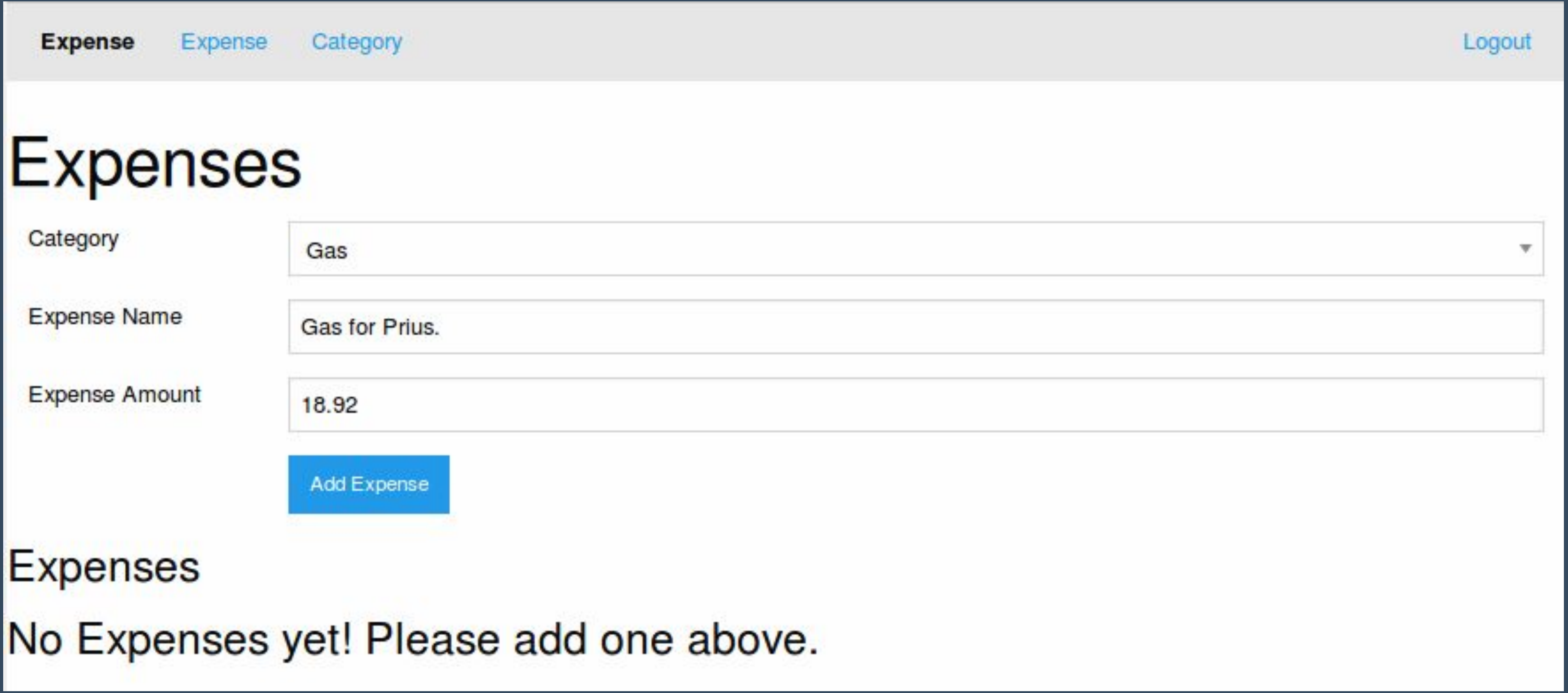


| Table | Owner | Comment |
|---|---|---|
| __diesel_schema_... | expense_tracker | |
| categories | expense_tracker | |
| expenses | expense_tracker | |
| users | expense_tracker | |

Edit Data - budget (localhost:5432) - budget - public.categories

| | id<br>[PK] serial | user_id<br>integer | name<br>character varying(75) | descrip<br>character varying(200) |
|---|---|---|---|---|
| 1 | 1 | 1 | Gas | Gas for my car |
| * | | | | |

# Demo - Create/View Expense 1



Expenses

Category: Gas

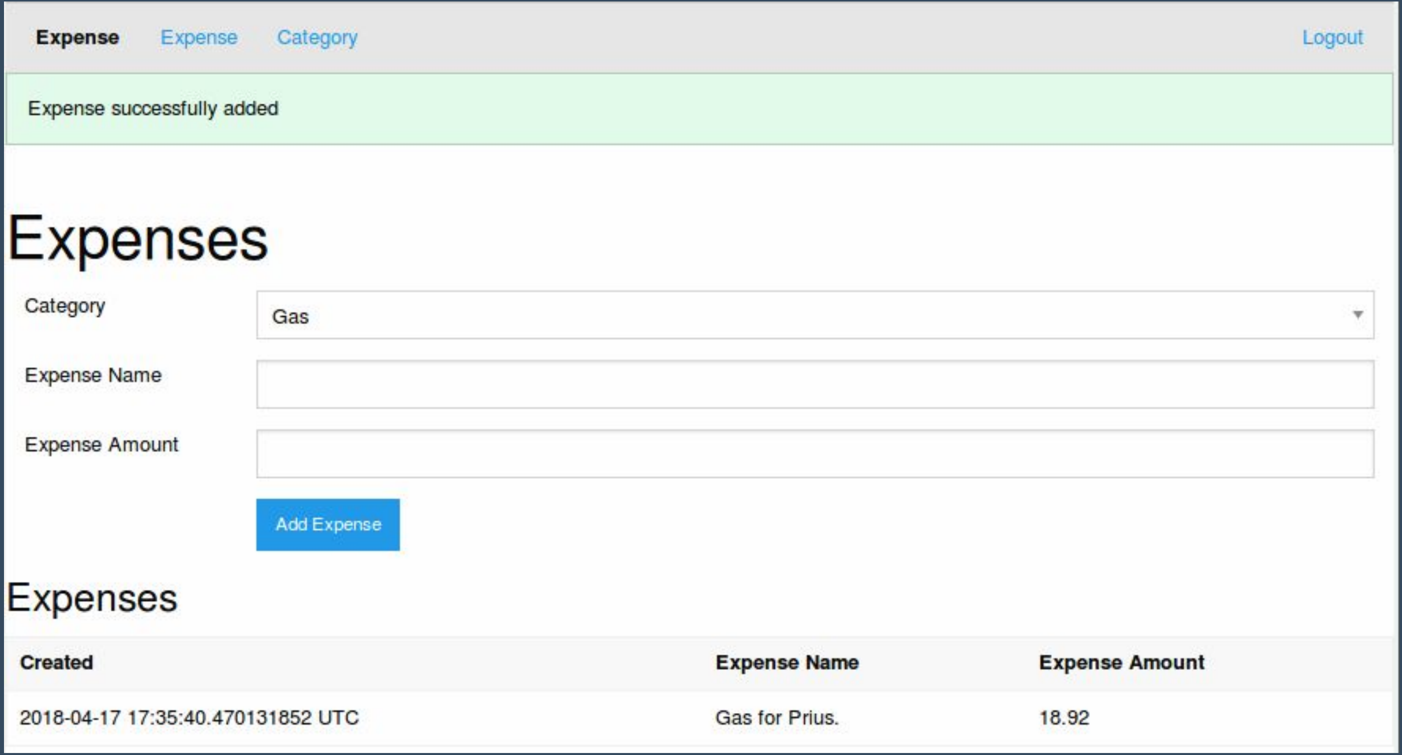
Expense Name: Gas for Prius.

Expense Amount: 18.92

Expenses

No Expenses yet! Please add one above.

# Demo - Create/View Expense 2



Expense successfully added

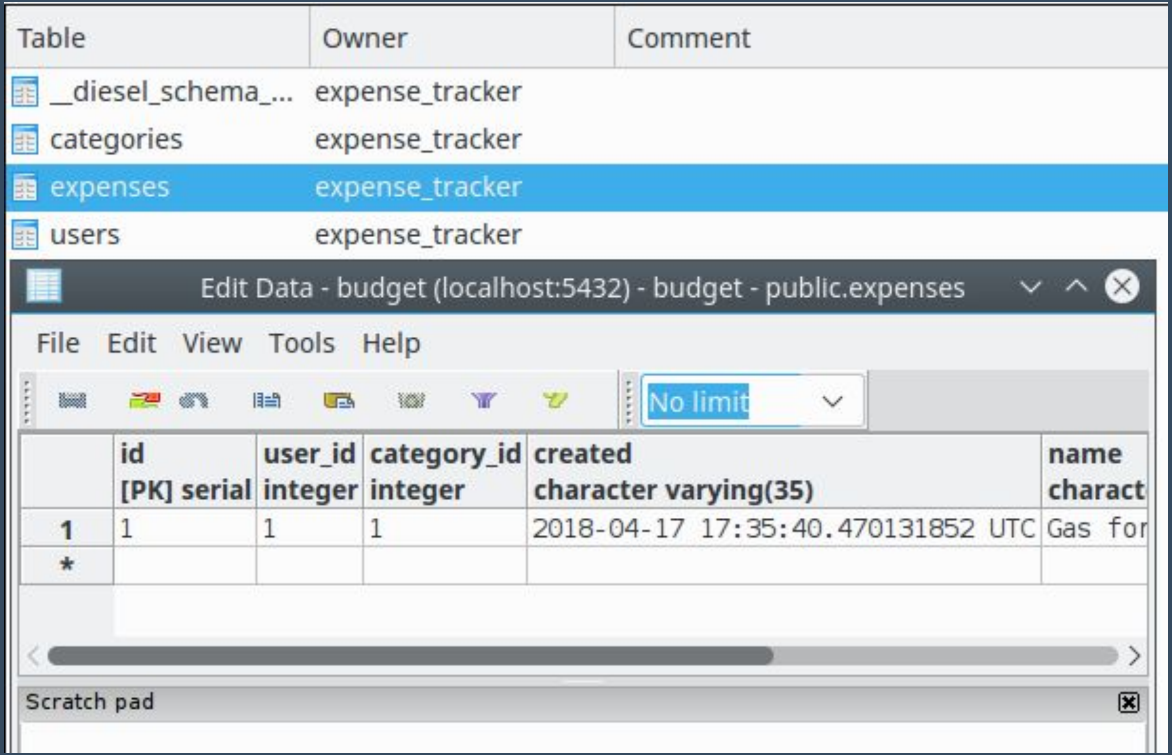
## Expenses

| | |
|---|---|
| Category | Gas |
| Expense Name | |
| Expense Amount | |

### Expenses

| Created | Expense Name | Expense Amount |
|---|---|---|
| 2018-04-17 17:35:40.470131852 UTC | Gas for Prius. | 18.92 |

# Demo - Create/View Expense 3

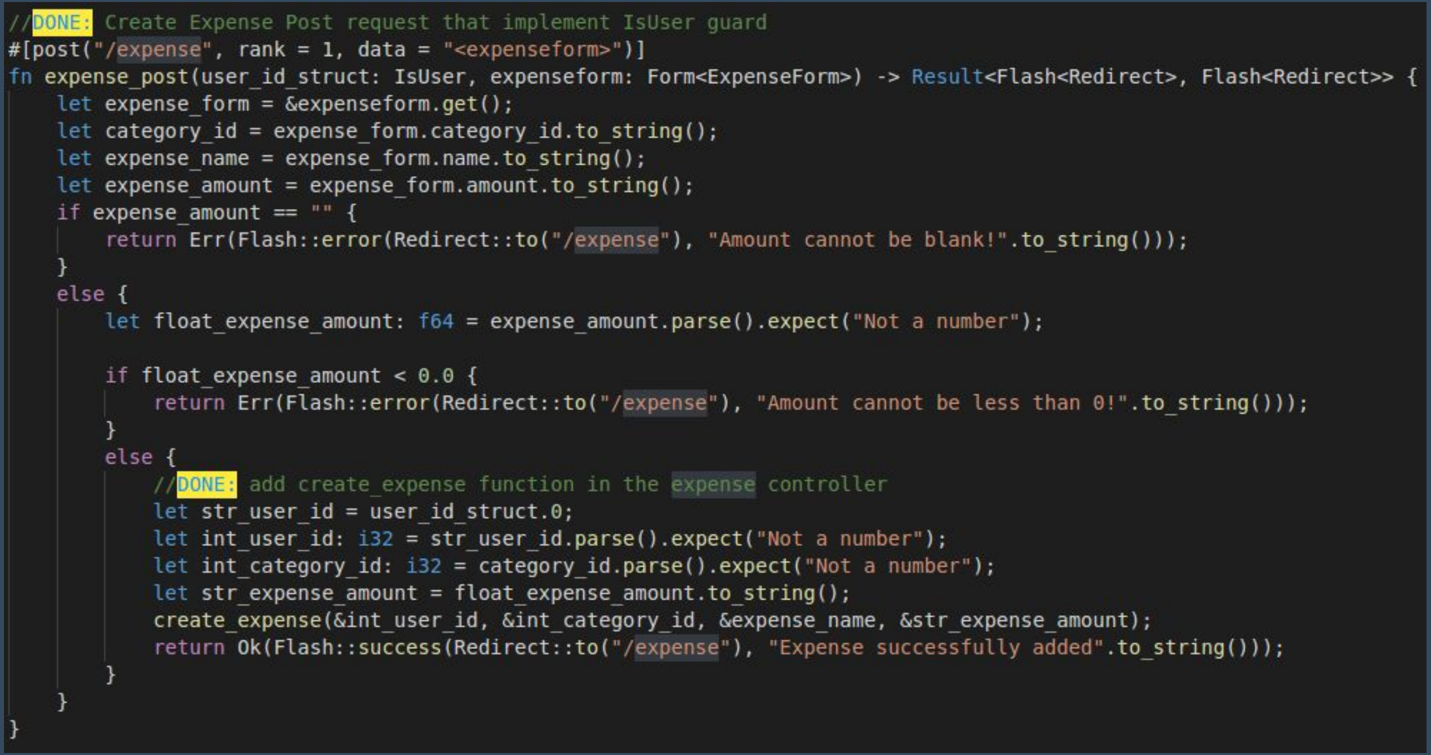


| Table | Owner | Comment |
|---|---|---|
| __diesel_schema_... | expense_tracker | |
| categories | expense_tracker | |
| expenses | expense_tracker | |
| users | expense_tracker | |

Edit Data - budget (localhost:5432) - budget - public.expenses
File
Edit
View
Tools
Help
No limit
Scratch pad

| | id [PK] serial | user_id integer | category_id integer | created character varying(35) | name charact |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2018-04-17 17:35:40.470131852 UTC | Gas for |
| * | | | | | |

# Demo - Code Example 1



```
//DONE: Create Expense Post request that implement IsUser guard
#[post("/expense", rank = 1, data = "<expenseform>")]
fn expense_post(user_id_struct: IsUser, expenseform: Form<ExpenseForm>) -> Result<Flash<Redirect>, Flash<Redirect>> {
    let expense_form = &expenseform.get();
    let category_id = expense_form.category_id.to_string();
    let expense_name = expense_form.name.to_string();
    let expense_amount = expense_form.amount.to_string();
    if expense_amount == "" {
        return Err(Flash::error(Redirect::to("/expense"), "Amount cannot be blank!".to_string()));
    }
    else {
        let float_expense_amount: f64 = expense_amount.parse().expect("Not a number");

        if float_expense_amount < 0.0 {
            return Err(Flash::error(Redirect::to("/expense"), "Amount cannot be less than 0!".to_string()));
        }
        else {
            //DONE: add create_expense function in the expense controller
            let str_user_id = user_id_struct.0;
            let int_user_id: i32 = str_user_id.parse().expect("Not a number");
            let int_category_id: i32 = category_id.parse().expect("Not a number");
            let str_expense_amount = float_expense_amount.to_string();
            create_expense(&int_user_id, &int_category_id, &expense_name, &str_expense_amount);
            return Ok(Flash::success(Redirect::to("/expense"), "Expense successfully added".to_string()));
        }
    }
}
```

# Demo - Code Example 2

The code from the previous slide handles an expense post request.

- The route contains 2 request guards (IsUser, and Form<ExpenseForm>). This is Rocket’s way of validating the user has permission (they are a logged in user), and the necessary data to access this route (they have a fully filled out ExpenseForm).
- Then the code parses the form, storing it’s information in variables to be used later. If the information is still in an incorrect format, it returns a flash message to the expense get route to notify the user of mistakes.
- Then the code gets the user id from the IsUser object, and converts the data into the format the Database requires, and then writes the new object to the Database by sending it to the ExpenseController.
- Finally a flash message, and redirect are returned to notify the user of the successfully added expense.

## What I Learned

- This is the largest project I have built with Rust to date. Rust is a multiparadigm programming language, and to build this project I had to learn more about Enums, Structs, and I became much more proficient at reading documentation.
- This is the first project I have built with Rocket. Rocket does some things that are very familiar to web frameworks I have worked with, but has some of its own unique features. I.e Request Guards that allow any HTTP requests to be defined with extra incoming data -- so you can verify users, or API keys easily. You can also define routes with the same end point, that don’t contain the required information to essentially overload any defined route.

## What I Learned continued

- Diesel, an ORM for Rust. I haved used Postgres before, but have never used Diesel, and had to learn how it handles schemas, and migrations. After picking up the basics it was easy to add new tables, and create controllers for my application.
- Foundation 6 significantly changed alert messages from Foundation 5. I was using Foundation 6, and tried to use alert messages how I had in the past, but their functionality is quite different from the last time I worked with them.

## What’s Next

- Expenses can be edited, and deleted.
- Categories can be edited, and deleted.
- Users can change their passwords, or email addresses.
- Users can view reports of their expenses, either by a set amount of time, or by a specific category.
- A monthly expense report is emailed to the user for the previous month, at the start of each month. I will work with the gmail API to accomplish this.
- The database resource pool is handled by R2D2, I will have to learn how to work with R2D2 and Diesel together.