USER/DEVELOPER GUIDE

# Introduction

The purpose of this application is to allow users to attempt experiments on the web with absolute convenience rather than having the user to attempt experiments within coordinated lab locations. This document will provide instructions on how to work with the application as well as how to build upon it.

# Getting Started

The minimum required applications and software must be installed to allow the application to function properly, and for the future developer to build upon the system, the required frameworks and libraries must be installed.

## Quick Start

### 2.1.1 User

To properly render the application supported browsers must be used.

Table 1 - Supported browsers to render the application

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MOBILE DEVICES** | Chrome | Firefox | Safari | Internet Explorer | Opera |
| Android | 🗸 | 🗸 | N/A | N/A | N/A |
| IOS | 🗸 | 🗸 | N/A | N/A | N/A |
| **Desktop** |
| Mac | 🗸 | 🗸 | 🗸 | N/A | 🗸 |
| Windows | 🗸 | 🗸 | N/A | 🗸 | 🗸 |

***Note:*** *Although the supported browser may be used, researchers may limit their experiment to work with specific browsers or devices. Supported browsers/devices will appear for each experiment.*

### 2.1.2 Developer

The required and recommended tools must be installed to efficiently build upon the application.

#### Installation of required tools and components

Whether the developer is using a UNIX system or Windows, a virtual environment should be used for best practice (Error handling, package deletion/update, etc.)

If your system contains python, run any virtual environment of your choosing.

|  |
| --- |
| pip install virtualenv |

Within your virtual environment, python 3.5.2 must be installed, as this is the python version that the application was developed with, the following code was used to build the environment folder with the corresponding python version.

|  |
| --- |
| virtualenv -p python3.5.2 env |

Then activate the virtual environment, the following code within the environment folder is used:

|  |
| --- |
| . bin/activate |

Once the virtual environment is activated, the Django framework must be installed:

|  |
| --- |
| pip install Django==1.8.7 |

##### Database:

By this point the developer can upload the application from GitHub within the environment folder, then PostgreSQL must be installed on the computer to handle the database. The original development was made with PostgreSQL 9.6. Use the following link to download PostgreSQL

<https://www.postgresql.org/download/>

##### IDE:

The IDE used for development was originally Eclipse with the pydev extension. Pydev is crucial as it provides code completion, Django integration, code analysis and many other features that would help with coding. The IDE was then switched to Visual Studio Code, as it was more powerful and didn’t experience any technical hiccups.

##### Libraries:

To fully utilize PostgreSQL with python and Django, **psycopg2** must be installed using pip in the command line within the virtual environment:

pip install psycopg2

Throughout the project **bootstrap** **forms** were used over Django forms, they are called in the template using bootstrap\_form, more of this will be discussed later. Install the library using:

pip install django-bootstrap3

A third-party library for allowing multiple upload capabilities was used, it allows researcher accounts to upload multiple files, install it through the command line like so:

pip install django-multiupload

##### Documentation:

Django Documentation: <https://docs.djangoproject.com/en/1.8/>

PostgreSQL: <https://www.postgresql.org/docs/9.6/static/intro-whatis.html>

Django-bootstrap3: <https://pypi.org/project/django-bootstrap3/>

Psycopg2: <http://initd.org/psycopg/docs/>

Django-multiupload: <https://github.com/Chive/django-multiupload>

# Understanding Django Framework

To start developing upon this platform, the base framework, Django, must be understood. This section will go over the bare minimum, but it is highly recommended to check out the documentation and understand it through that. Django itself is a powerful tool that allows developers to quickly develop their web applications rather than having to reinvent the wheel. Not only is it quick to develop, but it has an emphasis on security, the developer does not need to worry about cross-site scripting, cross-site request forgery and many more risks that developers must worry about. The backend of the platform is programmed with python, like many other frameworks, Django follows an architecture like the Model-View-Controller. Except, its known as the Model-View-Template. The view module contains the data that can be used within the platform, while the template handles how the data from the view module is shown to the user. There are multiple applications created within the project. All the applications installed should be part of the INSTALLED\_APPS array within the settings.py file of the project.

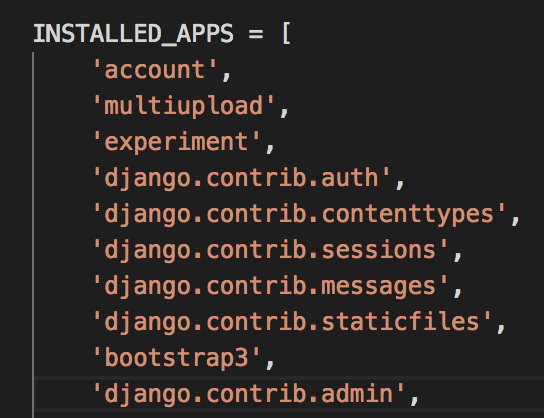


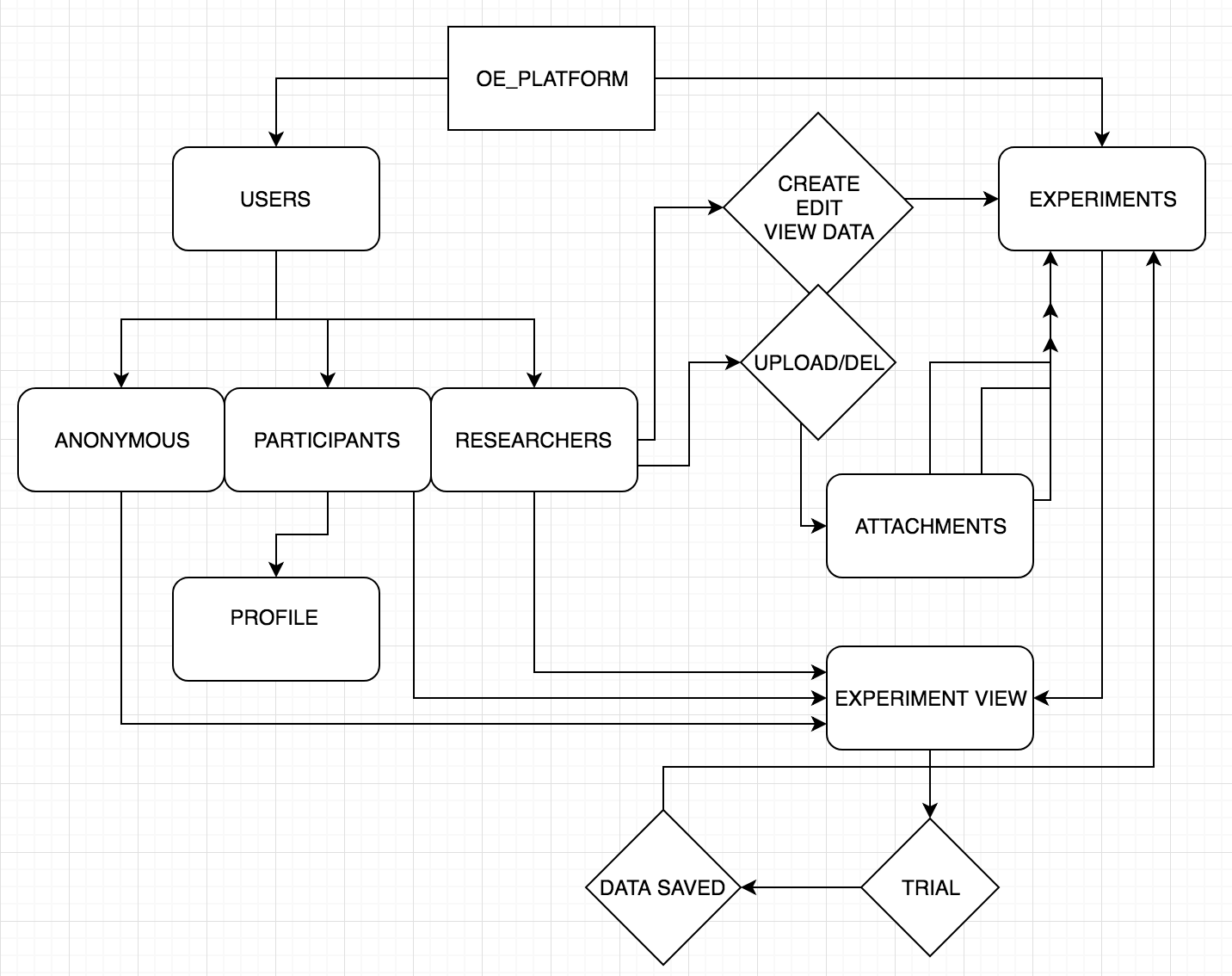
Figure - The array of installed applications within the project, located within the settings.py file

Within this array there are four applications that are not default Django applications, multiupload and bootstrap3 are third-party applications while account and experiment were created from scratch. Account is the application that stores the user and profile models. It handles most tasks related to the user, such as creating an account, editing an account, authenticating, and handles the urls and templates for each task. Experiment handles the experiment model and how it is created, edited, and how it can be rendered on the screen.

If your IDE is opened, you will notice three important files,

|  |  |
| --- | --- |
| Models.py | will allow you to design your models |
| Urls.py | will allow you to write your urls (connect the views to actual urls) |
| Views.py | this allows you to create the views |

Before moving on, you must already understand how databases work and the bare minimum of how they interact with frameworks(RDBMS) and using the command-line. The work flow of the application is shown below:



The simple flow chart will help understand the base of the platform. There are three types of users, Anonymous, Participants, and Researchers. When the participant is created, a profile object is created with it, used to store information of the user that may be used by experiments. The anonymous user is a temporary user that can access the experiments and attempt them, but if the user clears their browser cookies, they lose access to that temporary anonymous account. The researcher can create/edit/view-data of the experiments. There is a many-to-many relationship with the experiments and researchers, so many researchers can belong to an experiment and many experiments can belong to a researcher. The researcher can upload/delete multiple experiment files (CSS/HTML/JS). These uploaded files will render the experiment. When the experiment view is accessed, and an experiment is attempted, a trial model is produced, currently only used for anonymous users, to store trial numbers. Regular users have the data saved directly, without a trial model.

we will first off try to understand how to code models. Objects or models are created to help the interactions with each other. A model is a collection of properties and actions. For example, we created a User model, with the properties, username, password, email, etc. The user has such actions like create, save, clean, etc. “create” allows the user to be created, “save” allows the user that was created to be saved to the database, “clean” is a method in Django used to help validate objects, for our case, we checked whether the username or email were valid. These models are saved within the database (collection of data).

Creating models is simple, and there is no need to worry about creating the necessary SQL code to build the models to the database, Django handles everything.

To create a simple Profile model with different attributes we write the following:

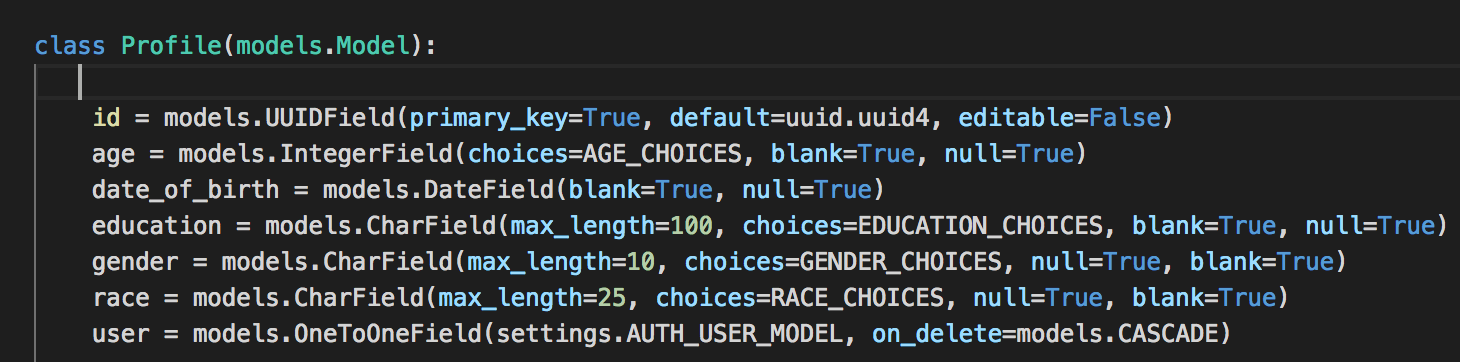


Figure - Profile Model Example

Normally the “id” field does not need to be declared as so, but because we wanted to use UUID field we declared it within the model and made sure the primary\_key option was set to true. The age field used the field option “choices”, meaning you are limited to a set of choices that are declared. AGE\_CHOICES was set to the following: This can be easily changed anytime.



Figure – Set of choices that age will be limited to

There are many fields and their following options that can be used for the models, they are listed within, it is highly recommended to check out the fields you are limited to. <https://docs.djangoproject.com/en/1.8/ref/models/fields/#django.db.models.Field>

The last field “user” is a one-to-one field linked with the profile. “settings.AUTH\_USER\_MODEL” is the user model that is set within the “settings” module. The class Profile inherited from the class models.Model, which is the base Django object model, so Django knows it should be saved in the database. There are times when you need to inherit from other classes, for example within our user object we have:

Class User(AbstractBaseUser, PermissionsMixin):

The User class in this case inherited from AbstractBaseUser, rather than the typical choice of models.User. The reason to inherit from the AbstractBaseUser is the fact that we did not want our user model to contain all the fields that come with it, we wanted a more basic user model, so inherited from the AbstractBaseUser, which models.User also inherits from. Now PermissionMixin was also inherited, now for Django a Mixin is a class that can be inherited to share all the methods.

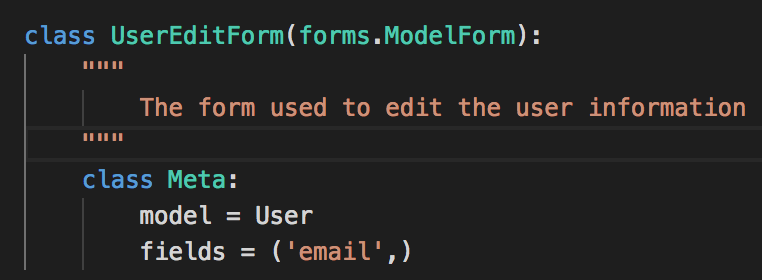


Figure - Creating a simple login form

Another powerful tool within Django is the ability to build forms from models easily, setting up the way as shown in figure 4, allows the form to map closely to the model. The code is found within forms.py, will be used within the view module and used to change the email of a user.

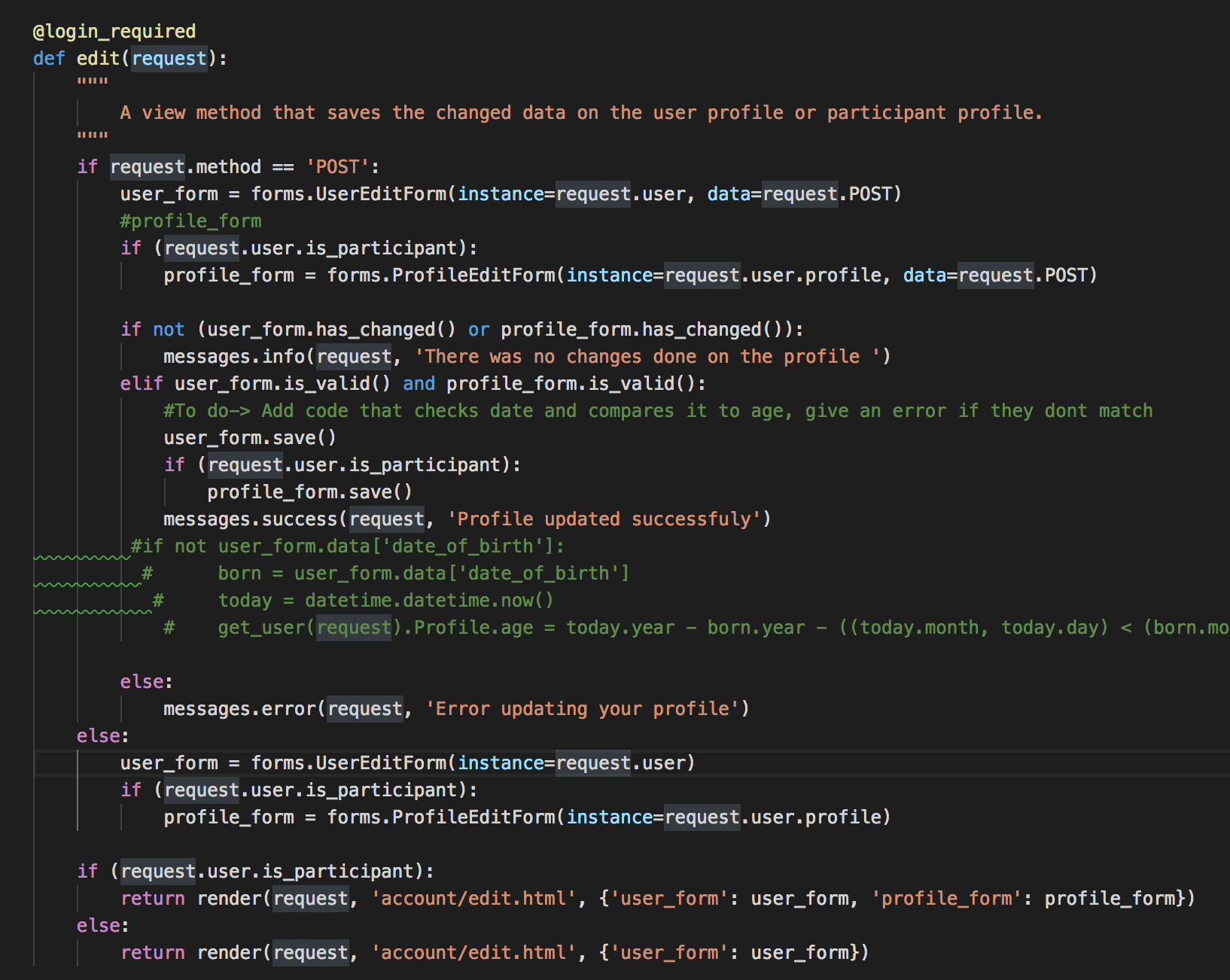


Figure - the user edit view

Figure 5 shows how the UserEditForm is connected to the view. So initially the form is declared as user\_form = forms.UserEditForm(instance=request.user.profile). The instance is the single occurrence of the model, we set user\_form to the pulled the data of the profile of the user. Upon accessing the view, the request sends information regarding the current user. To pull the model of the current user we use request.user.

return render(request, ‘account/edit.html’, {‘user\_form’: user\_form})

Within the view we were able to return render the template ‘edit.html’ located under the ‘account’ folder. We were able to pass a form model called user\_form and given a name ‘user\_form’ which can be used within the ‘edit.html’ template. Curly brace is used when we want to pass Django variables to the template, which is shown in this case.

Within the view we can pass any variables, if were passing an INT variable it would look the same {‘int\_ex’, int\_variable}, and within the template, to show the INT value, you would use ‘int\_ex’.

Within the urls.py file, we have the following:

urlpatterns = [

…

url(r'^edit/$', views.edit, name='edit'),

... ]

As shown, to connect a view to url, we store in an array called urlpatterns using the url() method. “r’^edit/$” is the url for the edit view, the view name is set to ‘edit’ so can be easily accessed elsewhere. When we access this link, it is relative to the app. So the current app is called ‘account’, when we access the url, it is under the following url: ‘server\_ip/account/edit/’.

When storing the template file, they must be stored in a specific folder, so Django can use its indexing capabilities to pull the file.

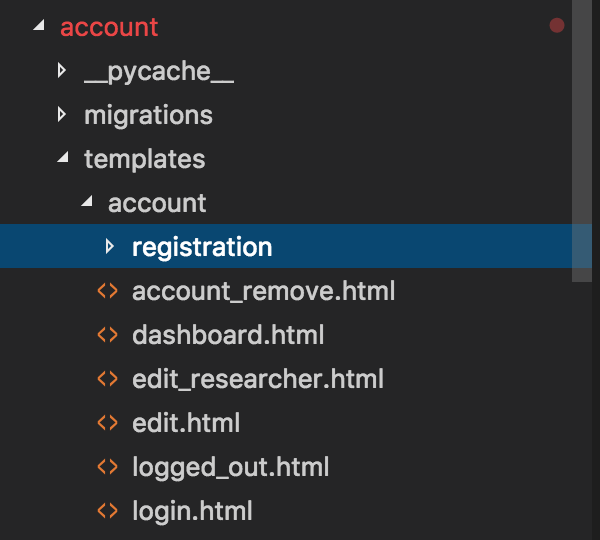


Figure 6 - template folder storage

As shown in figure 6, under the app account, we have a ‘templates’ folder, within that folder must be the app name. /templates/account/------. Within /account, is where all the template files should be stored. As shown previously, to render the edit.html file, we accessed it with the following: ‘account/edit.html’.

We will now quickly glance over the base template of the application, which can be found under the templates folder of the base folder (OE\_PLATFORM).

{% load staticfiles %}

{% load bootstrap3 %}

<title>{% block title %}{% endblock %}</title>

under the template, when we want to access a Django’s template methods we use the curly brace and the percentage sign. The first line ‘load staticfiles’ loads all static files within the base folder static, and all the folders within it. ‘load bootstrap3’ loads the Django bootstrap static files to help with rendering the forms and other front end related objects that use bootstrap. Within <title> a block was used with the name title. The reason this was set was so every template that inherits from this template will be allowed to change the title. If I included the base template in another template, I would simple use {% block title %} title of url {% endblock %}, for example under the dashboard template within the account app, we have:

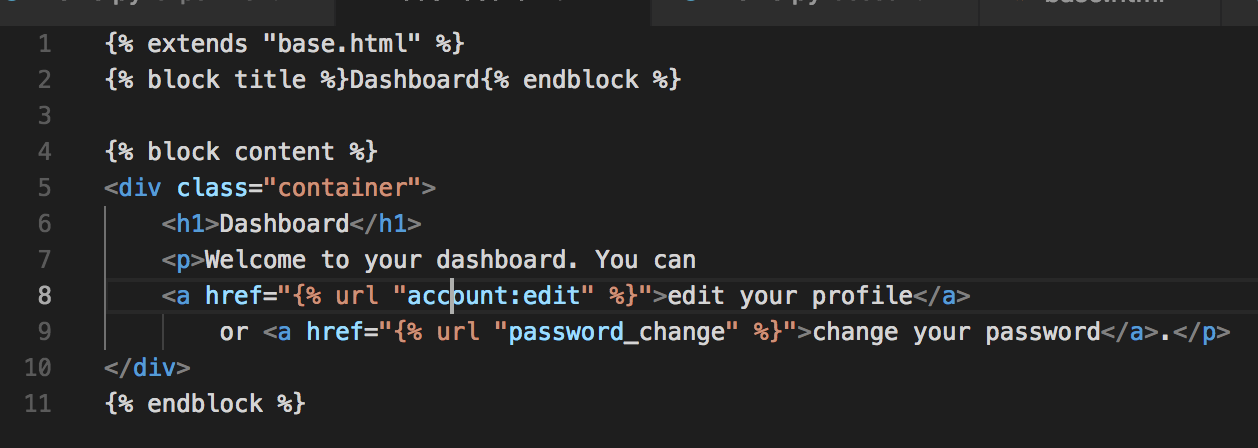


Figure 7 – account/templates/account/dashboard.html - template inheritance example

When the url is accessed for this template, the title shown will be “Dashboard”. To inherit from the base.html, we use the extends method. All the content for each different template is added within the content block.

When developing the views, it is recommended to try to develop class based views (CBV), rather than the traditional way (the edit view built above). CBV is a more modern, and efficient way to develop the views, but since we are using an older version of Django, we will only be limited to some CBVs, but that shouldn’t affect it too much. Under the Experiment views, there are many CBVs.

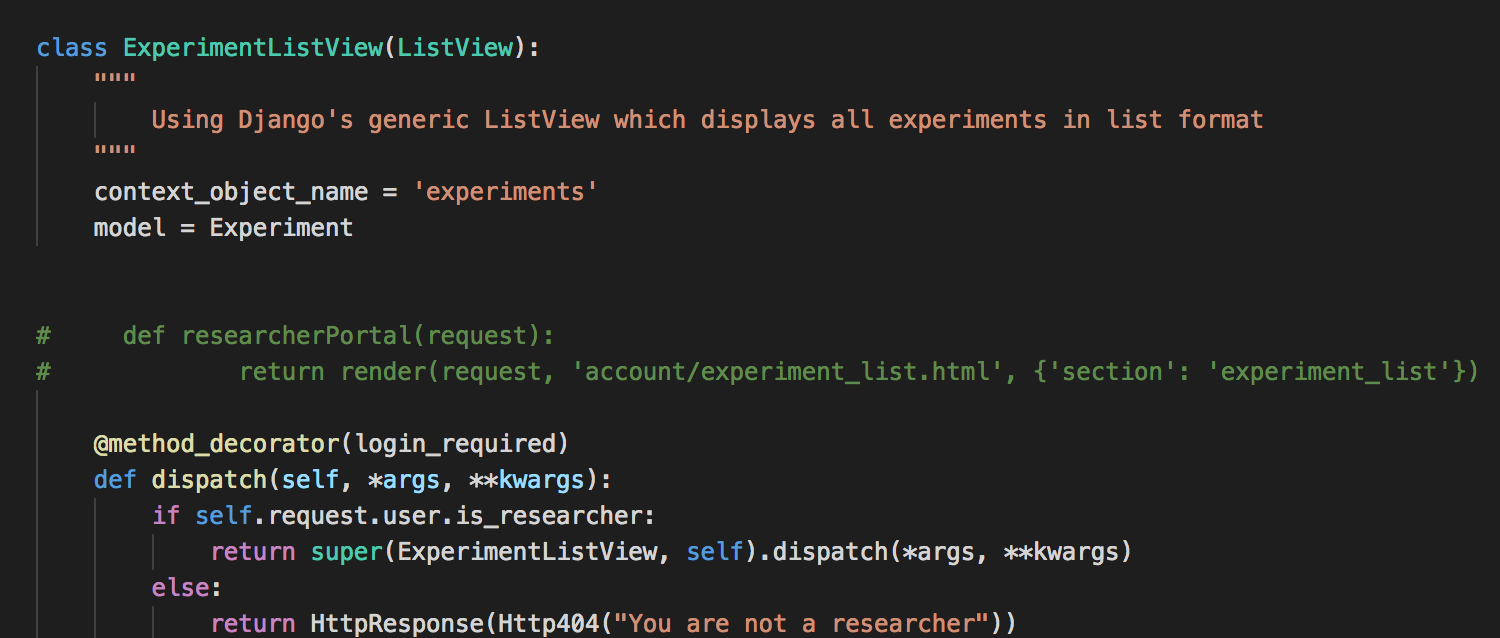


Figure 8 - experiment/templates/experiment/ experiment.html - CBV list example

Figure 8 demonstrates how to build a view that produces a list of all experiments owned by that user. The method dispatch was used to limit the view only to researchers.

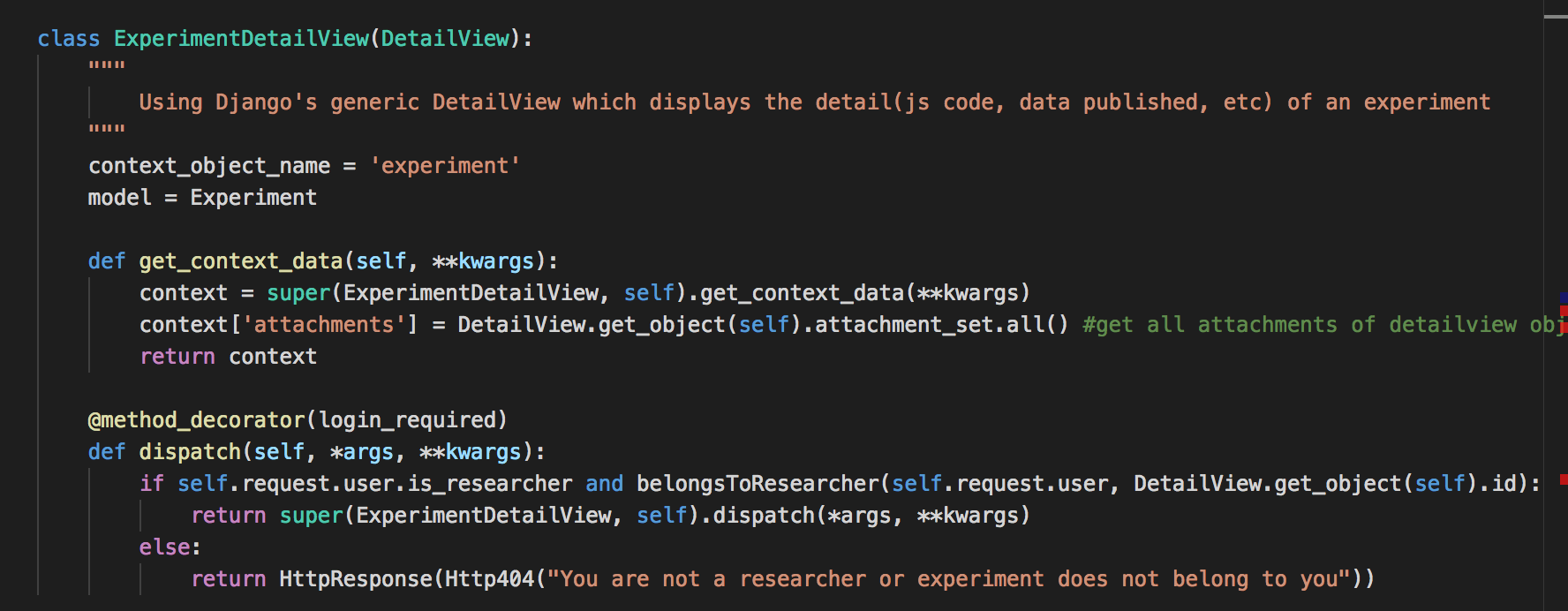


Figure 9 - experiment/templates/experiment/ experiment.html - Produce more data within a CBV

Figure 9 shows how to add extra data to the CBV. The detail view for an experiment is produced here, get\_context\_data will help us produce extra data. The variable context is an array of data that is used within the template. Initially only ‘experiment’ is stored within it.

## Researcher Guide

Account creation:

To create a researcher account, the following URL must be manually inputted in the browser: “IP/account/researcher/register/”. Once the researcher account is created, the researcher account can upload HTML, CSS, and JS files. Another important field must be filled in to properly create the experiment, the query field. The query field will help create the tables for the experiments. Ex.

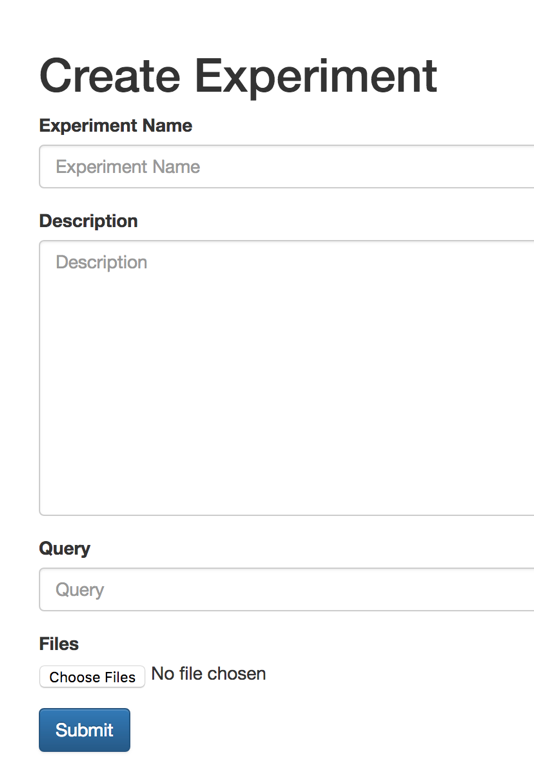


Figure - experiment creation

“CREATE TABLE {} (task\_num INT, task\_name TEXT, trial\_type TEXT, trial\_num INT, terminalfeedback\_bool TEXT, rotation\_angle REAL, targetangle\_deg REAL, targetdistance\_percmax REAL, homex\_px text, homey\_px REAL, targetx\_px REAL, targety\_px REAL, time\_s REAL, mousex\_px REAL, mousey\_px REAL, cursorx\_px REAL, cursory\_px REAL);”

This query will be used to create a table with the set columns and their set data types. The table name must be set to “{}” as the table name changes based on the user that attempts the experiment.

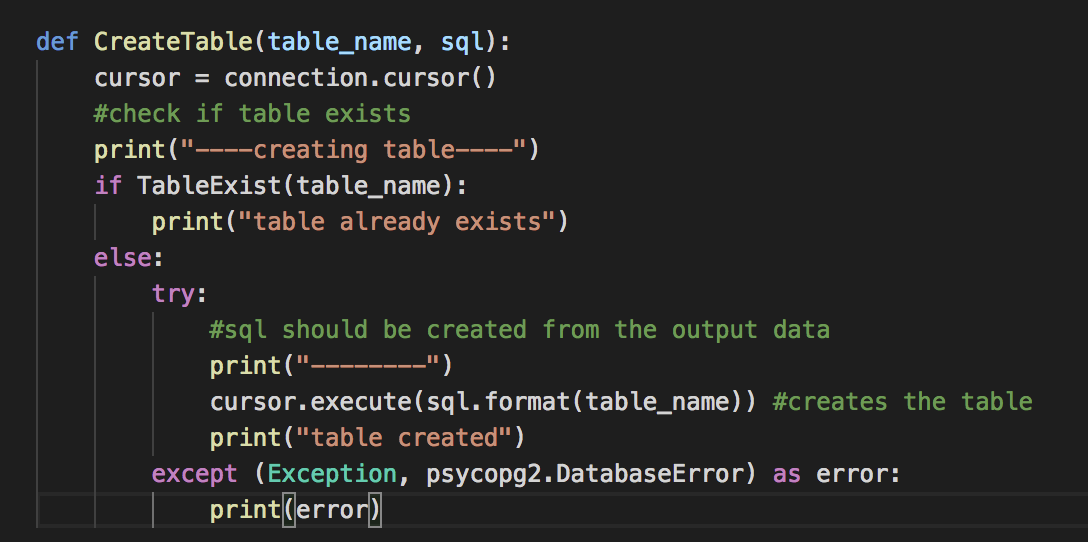
Within the query, the “{}” is added so it can be replaced by the actual table name from the CreateTable function as shown in figure 11. The function takes two inputes, the table name and the query (sql). Python allows you to add variables within the sql using the ‘format’ method. The format method allows you to replace the {} by the variable, this is safe from SQL injections because the table name is set based on user id.

Figure – experiment/views.py - Create table function

### Creating the experiment

When making the experiment, the developer must take into account the functions present by the application. To send data from the front end to the backend, the function PostData() must be used. The data must then be passed as JSON as input. Figure 12 shows the asynchronous ajax function used to send data to the backend.

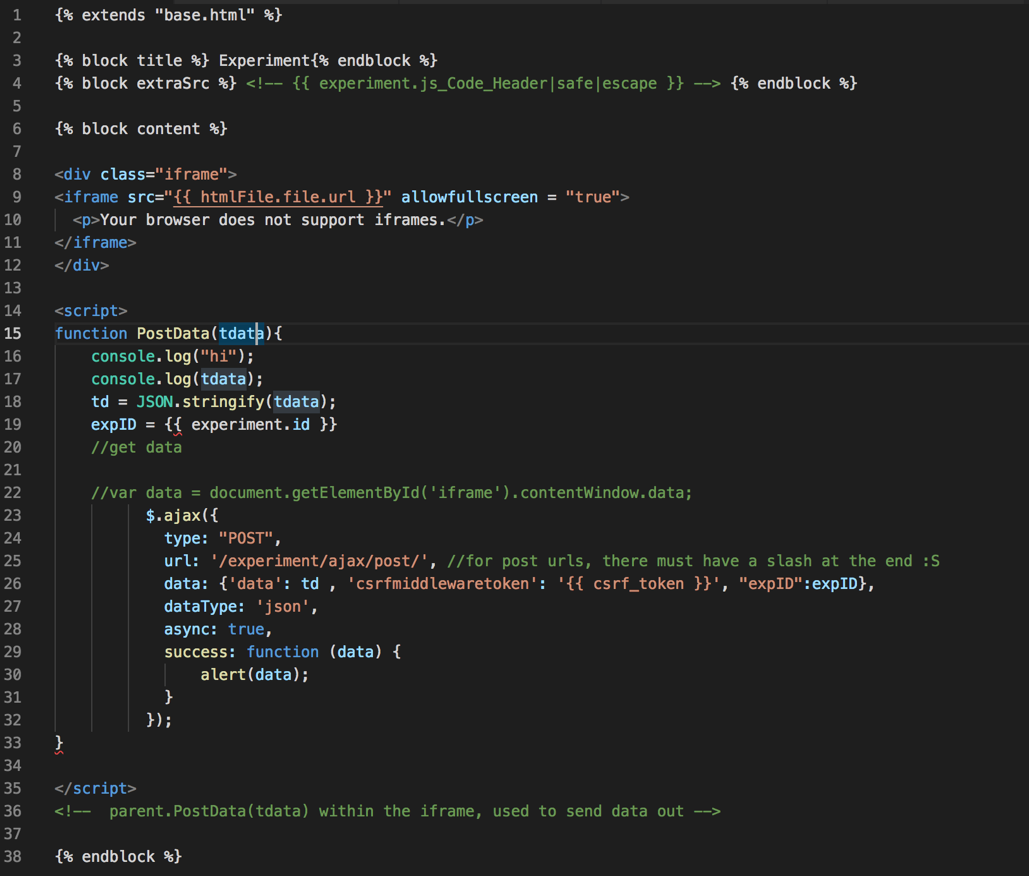


Figure – experiment/templates/experiment/ experiment.html - Function that will store the data

### Example Experiment Uploaded:



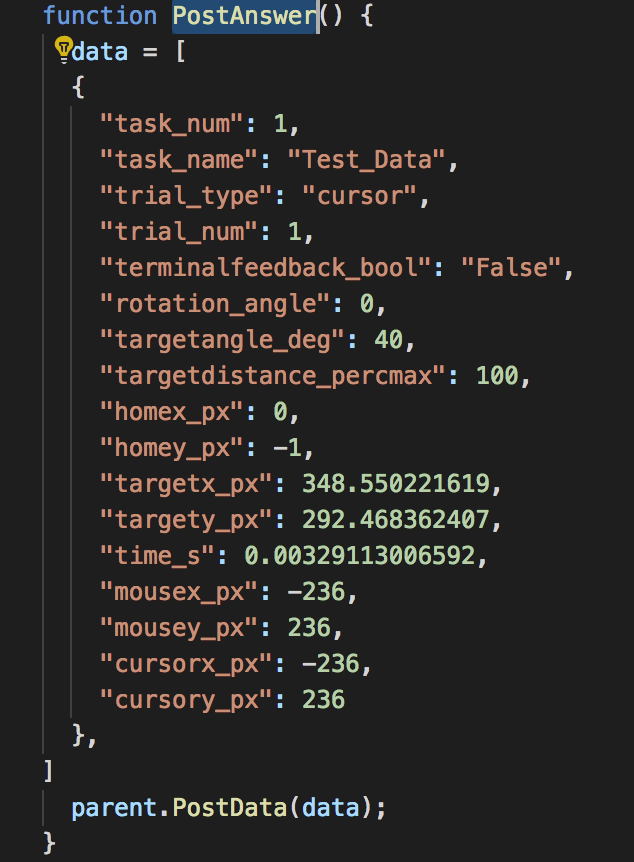
The first block shows the HTML created, a button is created to send data to the backend. The second block shows the JS side. Data was preset for example purposes. To access the PostData() function from the experiment, parent.PostData() must be used. Once the button is clicked, this set of data is sent to the backend and is associated with the user that attempted the experiment.

Figure - example experiment uploaded