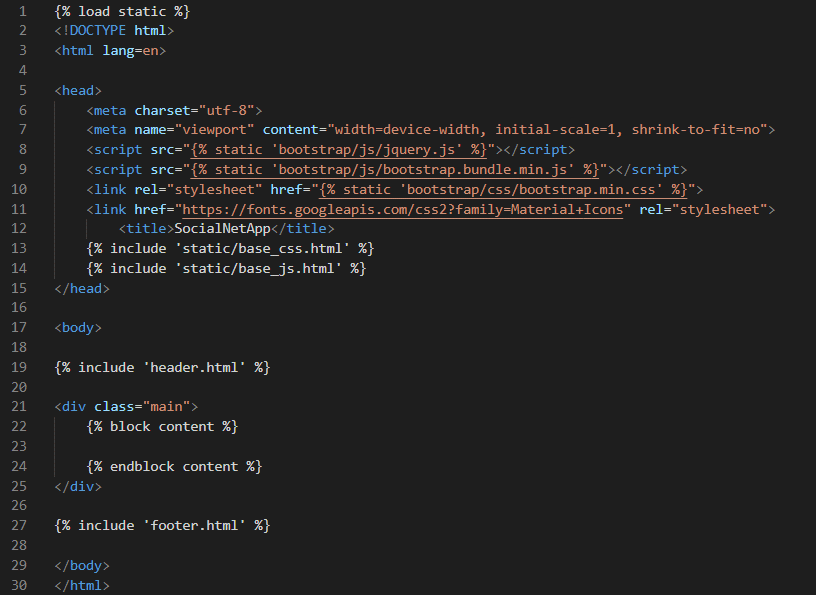
D1: Django code in standard ZIP format

D2: A report in PDF format. Including how to unpackage and run your application and how to run the tests for your application.

# **Introduction:**



# **Requirements:**

## **R1: The application contains the functionality required**

### a) Users can create accounts



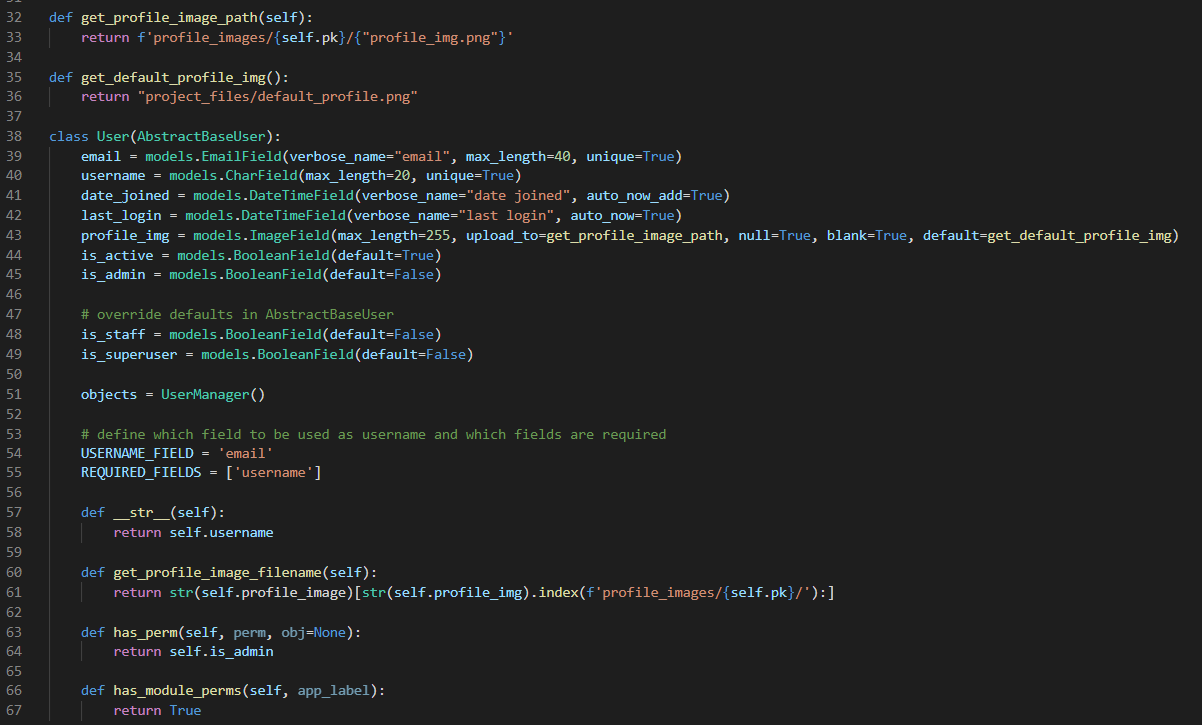
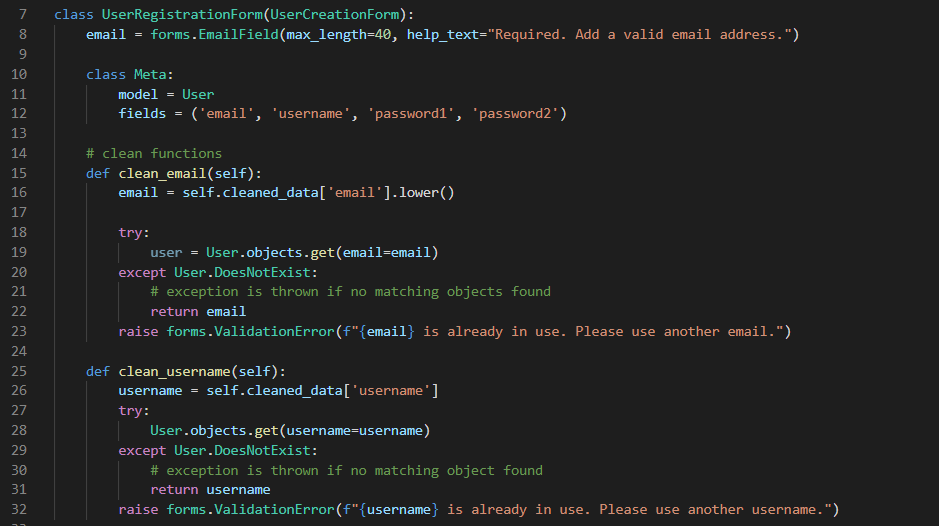


Figure 1 - User Model (users/models.py)

We will be creating a custom User model which inherits from AbstractBaseUser along with the use of UserManager which has functions such as create\_user() and create\_superuser() which we could customise and utilise to create our users.

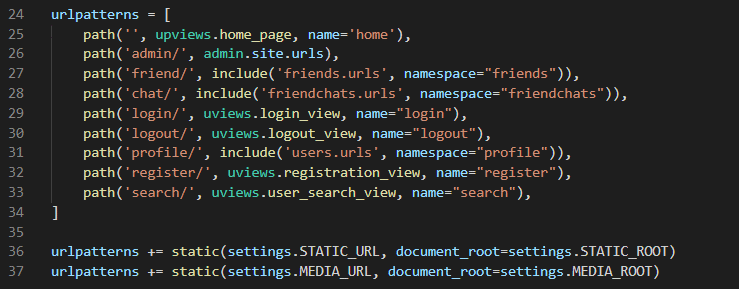
Some default fields such as is\_active, is\_admin, is\_staff and is\_superuser will be overridden and we will also set our USERNAME\_FIELD (‘email’) and REQUIRED\_FIELDS (‘username’) accordingly. I have selected email to be the username field instead of the username as the username can be changed and updated while the email may not be changed. In an actual development application, it would be ideal to allow users to change the email but for the purpose of this assignment, users will not be able to do this. Password field need not be specified.



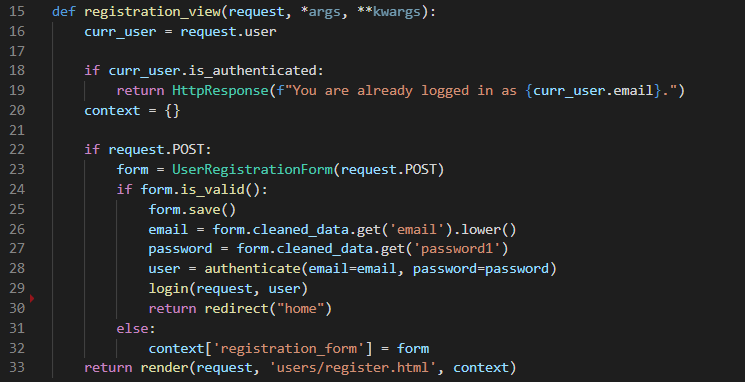
For the UserRegistrationForm, we will specify the email field using the forms.EmailField and I have chosen to set the max\_length of the email to 40 as it is uncommon for anyone to have an email longer than that.

We will be setting the model of this form based on the User model with an additional password1 and password2 fields to make users re-enter their password to confirm their password during registration.

2 additional clean functions are defined to process the email field to make it case insensitive and that the email is unique. I have made the username field case sensitive, but it must also be unique. When any form validation errors are raised, the errors are displayed.



When the user is redirected or visits the registration url (‘register/’), the registration\_view in users/views.py will be called.



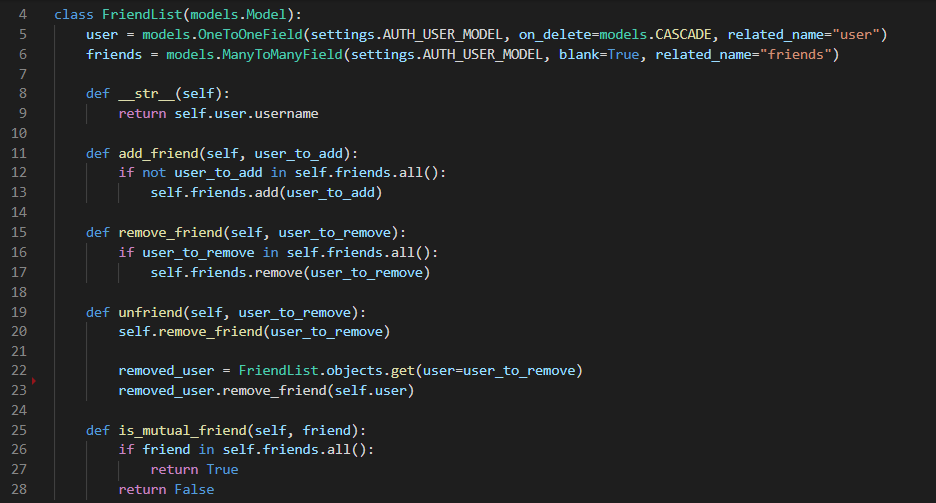
When the registration page is visited, we must first check if the user is already authenticated. Authenticated users will still be able to visit the registration page by entering the url into the browser despite the register button being not available to them.

If the user first visits the page, the request method will be “GET” and we will simply render the   
“users/register.html” page with an empty context. If the users clicks the submit button in the registration form in register.html page, the request method will be “POST” and we will proceed to validate the UserRegistrationForm.

If any form validation errors are raised, the validation errors will be passed to the context before re-rendering the register.html page with the updated context. If the form is valid, the form is saved, and user will be created and authenticated. The user will then be logged in before redirecting the user to the home page.

The registration page can be visited by clicking the register

### b) Users can log in and log out

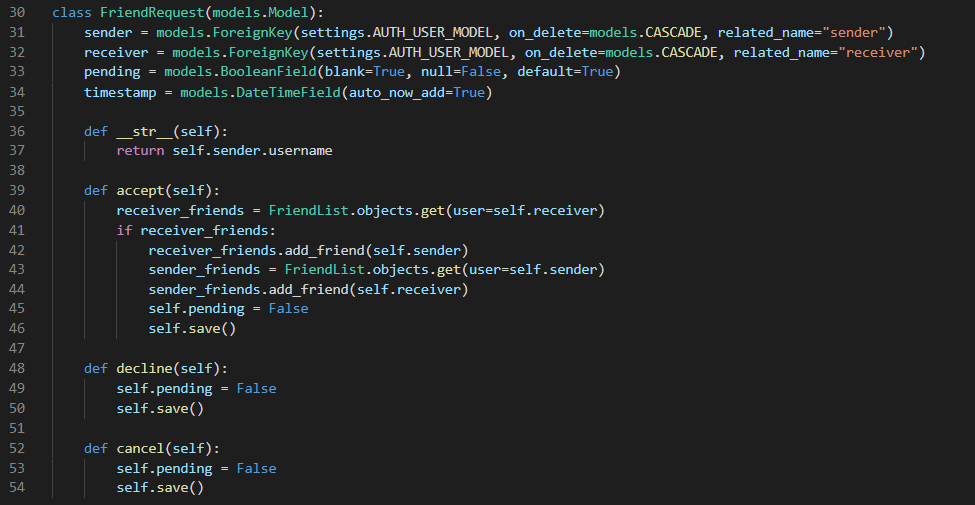


The FriendList model consists of user and friends field where the user field has a one to one field relationship with the settings.AUTH\_USER\_MODEL which is the User model which we have defined in settings.py. Each friend list can have none to many Users as its friends and each User can belong to none or many friend lists.

Add\_friend() function will add the specific user only if the user is not already in the user’s friend list while remove\_friend() function ensures that the specific user is in the user’s friend list before removing him/her.

The unfriend() function we call upon the remove\_friend() function to remove the specific user from the current user’s friend list and remove the current user from the specific user’s friend list.

The is\_mutual\_friend() function will be used when a user is visiting another user’s friend list. If another user’s friend list contains a mutual friend with the current user, the function returns true.



The FriendRequest model will require 4 fields, sender, receiver, pending and timestamp. Pending field will be used to determine if the friend request is still pending action from the receiver. Timestamp field has an auto\_now\_add parameter set to True so that whenever a new friend request is made, the timestamp is added automatically based on the time of request.

The accept() function will add the sender of the request into the receiver’s friend list and receiver of the request into the sender’s friend list. The pending status of the request is then set to False. The decline() or cancel() function simply sets the request status to False.

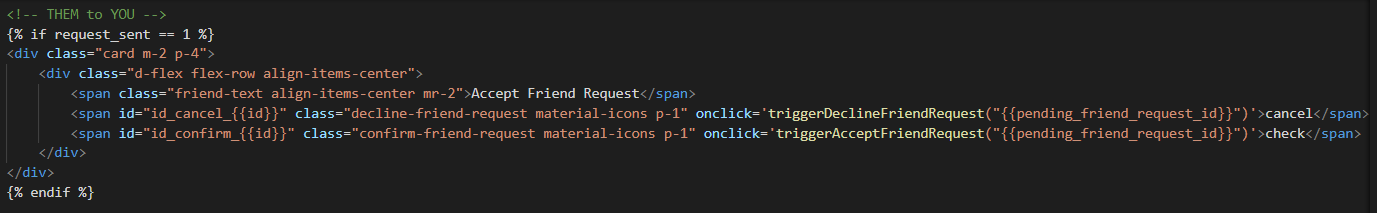


Figure 1 – profile.html line 115 – 123 (users/templates/users/profile.html)

The profile.html checks if the request\_sent status is equal to 1. The friend request statuses can be seen below in the enum class FriendRequestStatus

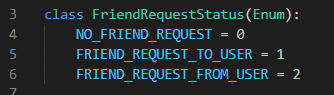
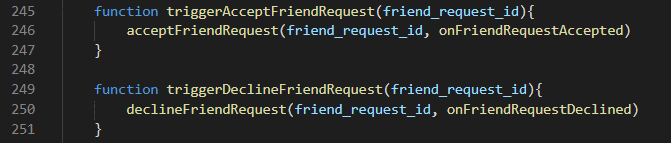


Figure 1 – FriendRequestStatus enum class (friends/friend\_request\_status.py)

When a friend request is sent to a user, they may go to the sender’s profile to accept or decline the friend request by clicking the span “check” and “cancel” icon respectively. Clicking them triggers the following two function.



This 2 functions are javascript functions that throws ajax responses.



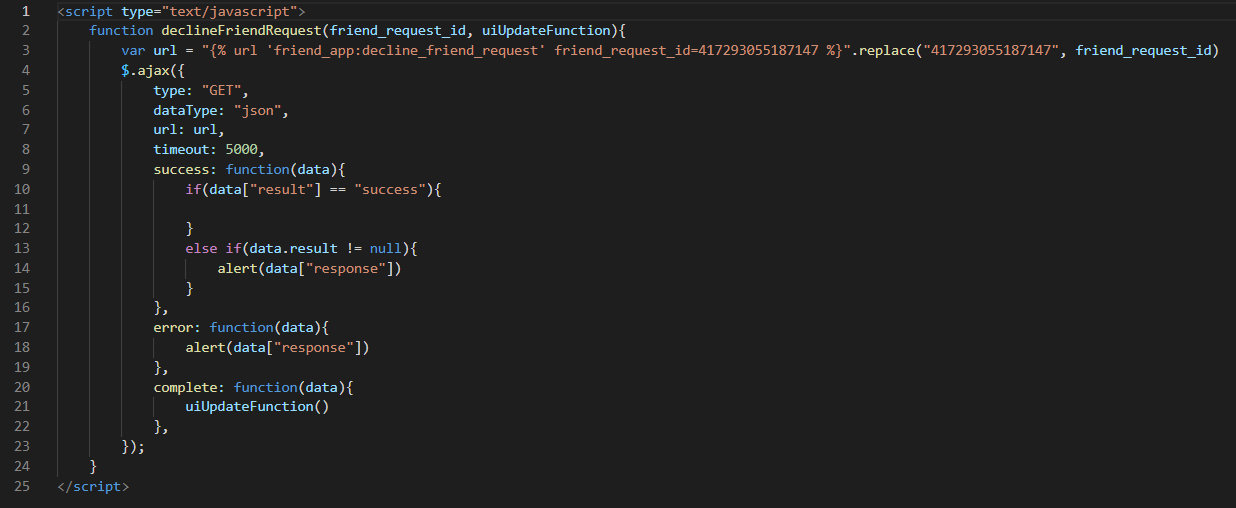
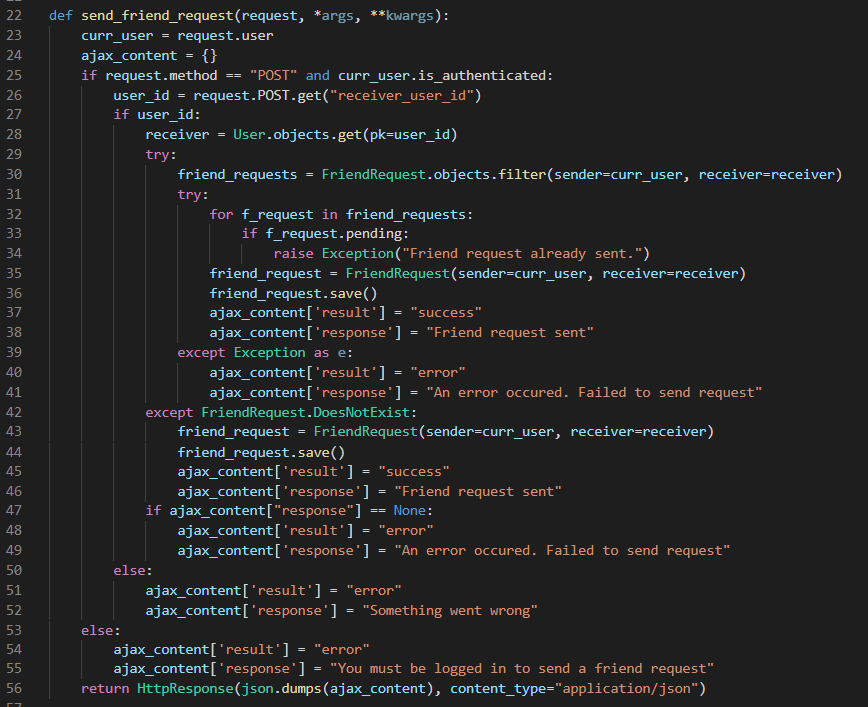




Figure 1 – profile.html line 125 – 156 (users/templates/users/profile.html)





### c) Users can search for other users

### d) Users can add other users as friends

### e) Users can chat in realtime with friends

### f) Users can add status updates to their home page

### g) Users can add media (such as images to their account and these are accessible via their home page

### h) correct use of models and migrations

### i) correct use of form, validators and serialisation

### j) correct use of django-rest-framework

### k) correct use of URL routing

### l) appropriate use of unit testing

### m) An appropriate method for storing and displaying media files is given

## **R2: Implements and appropriate database model to model accounts, the stored data and the relationships between accounts**

## **R3: Implementation of appropriate code for a REST interface that allows users to access their data**

## **R4: Implementation appropriate tests for the server side code**

## **C1: Code is clearly organised into appropriate files (i.e. view code is placed in an appropriate view.py or api.py file, models are placed in an appropriate models.py file)**

## **C2: Appropriate comments are included to ensure the code is clear and readable**

## **C3: Code is laid out clearly with consistent indenting, ideally following python pep8 standard**

## **C4: Code is organised into appropriate functions with clear, limited purpose**

## **C5: Functions, classes and variables have meaningful names, with a consistent naming style**

## **C6: Appropriate tests to cover the API functionality are provided**

Case insensitive for authentication:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

edit\_user.html

explain enctype=”multipart/form-data”

https://stackoverflow.com/questions/4526273/what-does-enctype-multipart-form-data-mean

accept\_friend\_request.html

// inserting friend\_request\_id directly produces error,

// therefore, we will enter a random unguessable string of integers

// and replace it with the friend\_request\_id

var url = "{% url 'friend\_app:accept\_friend\_request' friend\_request\_id=417293055187147 %}".replace("417293055187147", friend\_request\_id)

note: it must be all filled with numbers and no string

Superuser:

Email: admin@superemail.com

Username: admin

Password: theadminpassword