

How to create e-wallet app with python django

Nowday's e-wallet web apps are widely used by everyone all over the world. Paypal is a perfect example of it. From sending money to friends to online shopping it is being used. Have you ever wondered how these apps are built or which technologies have been used or how these are actually developed? If your answer is yes then you are at the right place.

Today we will be building a real world e-wallet app like paypal with python and famous web application framework 'Django' .

Our apps core features :

1. Sending money to anyone.
2. Receiving money.
3. Requesting money .
4. Taking donation from others .

What you will learn :

1. Building a real world project from scratch(html,css,javascript)
2. User registration with custom user model.
3. Django authentication
4. Django message framework.
5. Payment integration using checkout.com API.
6. Complex database queries and relationships.
7. Integrating google re-captcha .
8. All the advance features discussed above.

First of all we need to install django on our machine. But we need a virtualization package to create a virtual environment for our project. First, create a folder with your project name. I am using e-wallet as project folder name. After that navigate to it via cd command and to install virtualenv just type in your terminal 'pip install virtualenv'.

Terminal:

```
shakil@shakil:~/Desktop/project/e-wallet$ pip install virtualenv  
Collecting virtualenv.....
```

After installing virtualenv -in terminal type 'virtualenv env'

Terminal:

```
shakil@shakil:~/Desktop/project/e-wallet$ virtualenv env
```

now you should be able to see a directory inside e-wallet directory called env where all our required package will be installed.

Let's install django real quick----

but before that we need to activate our virtual environment. To activate it just type in the terminal '**source env/bin/activate**' .

Terminal:

```
shakil@shakil:~/Desktop/project/e-wallet$ source env/bin/activate
```

In your terminal you will see a (env) tag before any line. It will look like this

```
shakil@shakil:~/Desktop/project/e-wallet$ source env/bin/activate  
(env) shakil@shakil:~/Desktop/project/e-wallet$
```

We are all set to install django finally . To install django type in the terminal
'**pip install django**'. It will automatically install the latest version of django.

Terminal:

```
(env) shakil@shakil:~/Desktop/project/e-wallet$ pip install django
```

After successfully installing django start a new django project by typing in the terminal

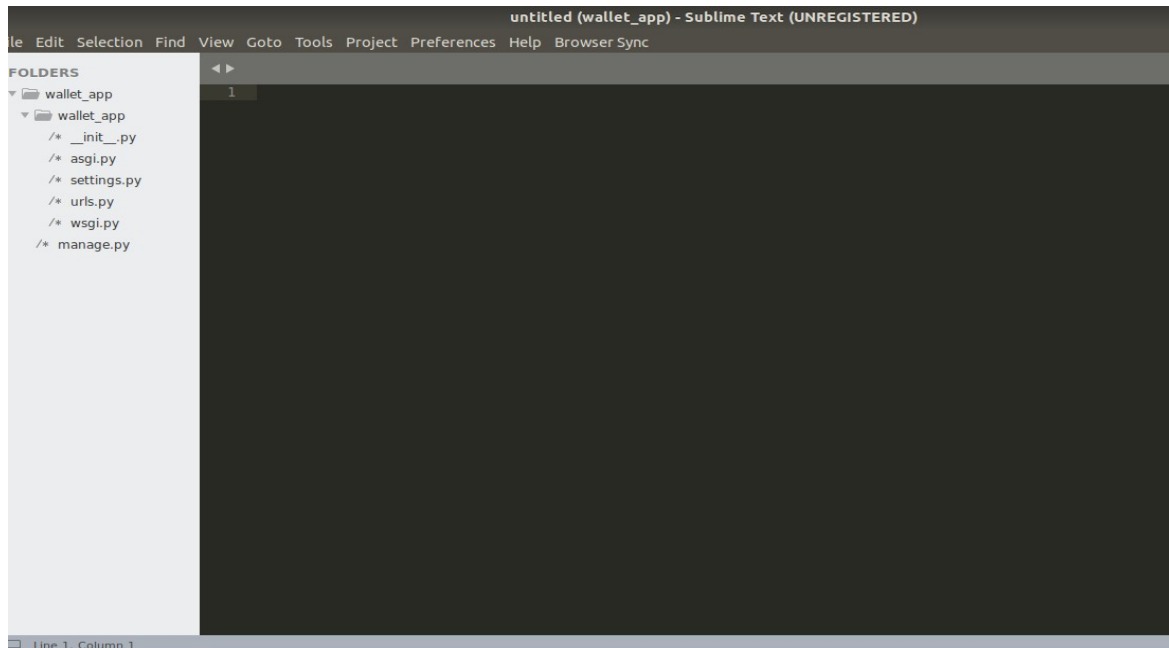
'**django-admin startproject wallet_app**'

Terminal:

```
(env) shakil@shakil:~/Desktop/project/e-wallet$ django-admin startproject wallet_app
```

You will see a folder name wallet_app. Open it or add it as a project folder with your favorite ide or text editor.

I am using sublime-text. My project Structure looks like this:



Complete django configuration:

Open your settings.py and edit the followings:

In the template section you will see this:--

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]
```

Now it looks like this :

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'template')], #new code
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
],
```

add these code at the bottom of the settings.py file :

```
STATIC_URL = '/static/'
```

```
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

```
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'staticfiles')]
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

```
MEDIA_URL = '/media/'
```

these special tag is used to tell django where our static files will be located and how our url structure will look like for media and staticfiles.

In the DIRS[] list we have specified where the template files are located but still we have not created any template directory. So, create 'template' and 'staticfiles' directory inside your main app folder by typing in the terminal 'mkdir template staticfiles'.

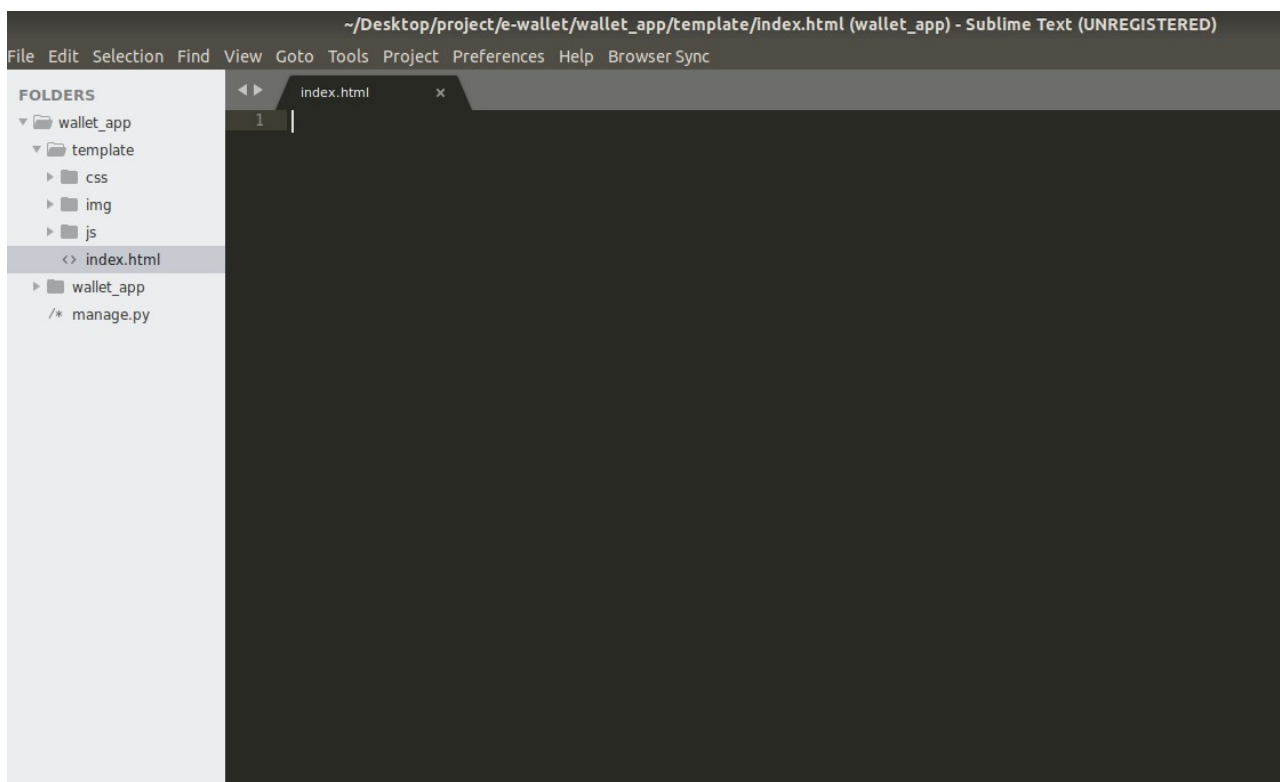
Throughout the tutorial we will be using sqlite database ,so we do not need to customize the database section in settings.py . We are now all set to start our real project.

Designing UI using html and css:

In the template directory create one file 'index.html', three folders 'css', 'js' and 'img'.

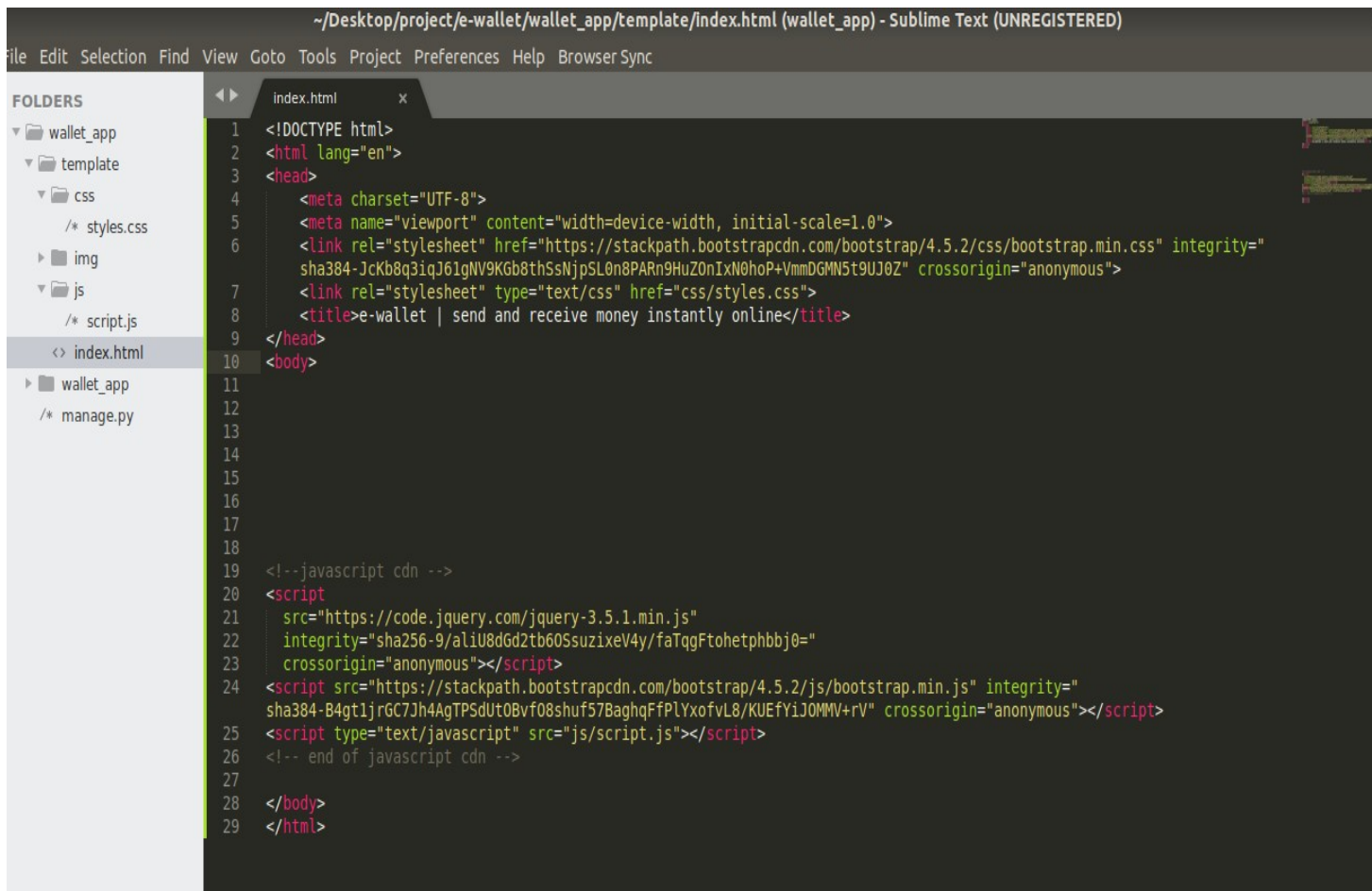
In the css folder create one file style.css and in the js folder create a file 'script.js'.

For temporary UI design we are going to store all the html and static files in this template directory. When we are done with designing we will separate the html and static files to different folder as django suggests. 'img' 'css' and 'js' folder will be moved to the staticfiles folder. But for now let's place it altogether and start designing our first index page. Here is my project structure until now:



I will be using bootstrap and jquery to make our UI development fast and responsive.

Here you can see I have setup a basic html file structure with bootstrap, jquery cdn and our custom css, js files .



```
~/Desktop/project/e-wallet/wallet_app/template/index.html (wallet_app) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help Browser Sync

FOLDERS
└─ wallet_app
  └─ template
    └─ css
      /* styles.css
    └─ img
    └─ js
      /* script.js
  < index.html
  └─ wallet_app
    /* manage.py

index.html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" integrity="
7   sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZ0nIxN0hoP+VmmDGMN5t9UJ0Z" crossorigin="anonymous">
8   <link rel="stylesheet" type="text/css" href="css/styles.css">
9   <title>e-wallet | send and receive money instantly online</title>
10 </head>
11 <body>
12
13
14
15
16
17
18
19 <!-- javascript cdn -->
20 <script
21   src="https://code.jquery.com/jquery-3.5.1.min.js"
22   integrity="sha256-9/aliU8dGd2tb60SsuzixeV4y/faTqgFtohetphbbj0="
23   crossorigin="anonymous"></script>
24 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" integrity="
25 sha384-B4gt1jrGC7Jh4AgTPSdUt08Bvf08shuf57BaghqFfPLYxofvL8/KUEfYiJOMMV+rV" crossorigin="anonymous"></script>
26 <script type="text/javascript" src="js/script.js"></script>
27 <!-- end of javascript cdn -->
28
29 </body>
30 </html>
```

Now let's create the navigation menu bar. Navigation bar should be transparent so that our background image can cover entire navbar and hero text. Type the navbar codes inside the `<body>` tag of `index.html`.

We have created a head section and set the background image. As our navbar is transparent we can see the image through the navbar which looks great. We also created some buttons and added a border at bottom with 1px solid white color and it will only be visible when someone hovers over it. It shows awesome cool effect. The navbar links are dummy for now, we will work on it later.

Note: I will not be able to discuss all the nitty-gritty stuffs of UI design as we are focused only on django and python. So, I will go quick here.....

Index.html:

```

<header>
  <div class="head-section">

<nav class="navbar navbar-expand-lg navbar-dark bg-transparent">
  <div class="container">
    <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarTogglerDemo01" aria-controls="navbarTogglerDemo01"
aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">

      <a class="navbar-brand" href="#">E-WALLET</a>
      <ul class="navbar-nav ml-auto mt-2 mt-lg-0">
        <li class="nav-item active">
          <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
        </li><span class="bar text-white mx-1 my-auto" >|</span>
        <li class="nav-item">
          <a class="nav-link" href="#">Our services </a>
        </li><span class="bar text-white mx-1 my-auto">|</span>
        <li class="nav-item">
          <a class="nav-link" href="#">About us </a>
        </li><span class="bar text-white mx-1 my-auto">|</span>
        <li class="nav-item">
          <a class="nav-link" href="#">Contact us </a>
        </li><span class="bar text-white mx-1 my-auto">|</span>
        <li class="nav-item">
          <a class="nav-link" href="#">Login </a>
        </li><span class="bar text-white mx-1 my-auto">|</span>
        <li class="nav-item">
          <a class="nav-link" href="#">Register</a>
        </li>
      </ul>

    </div>
  </div>
</nav>

<div class="hero-text">

  <h2 class="text-white mb-3">Send Money Worldwide Instantly </h2>
  <p class="lead mb-4 text-white">Send money to your friends and family.
Generate invoice and get paid from your client in a just few clicks.<br>
Secure transection within a second.<br></p>
  <a class="calltoaction btn btn-info btn-lg" href="#">Get started</a>
  <a href="#" class="btn text-white ml-3 btn-lg btn-warning moreinfobtn">More info</a>

</div>

</div>
</header>

```


styles.css:

```
nav ul li a{
    color:white !important ;

}

nav ul li{
    padding-bottom:3px !important;

    transition-duration: 0.5s !important;
}

nav ul li:hover{

    border-bottom: 1px solid white !important;

}

.head-section{
    background-image: linear-gradient(to right,rgba(0,0,0,0.7),rgba(0,0,0,0.7)), url('../img/heroimage-min.jpg');
    background-size: cover;
    height: 100vh;
    width: 100%;
}

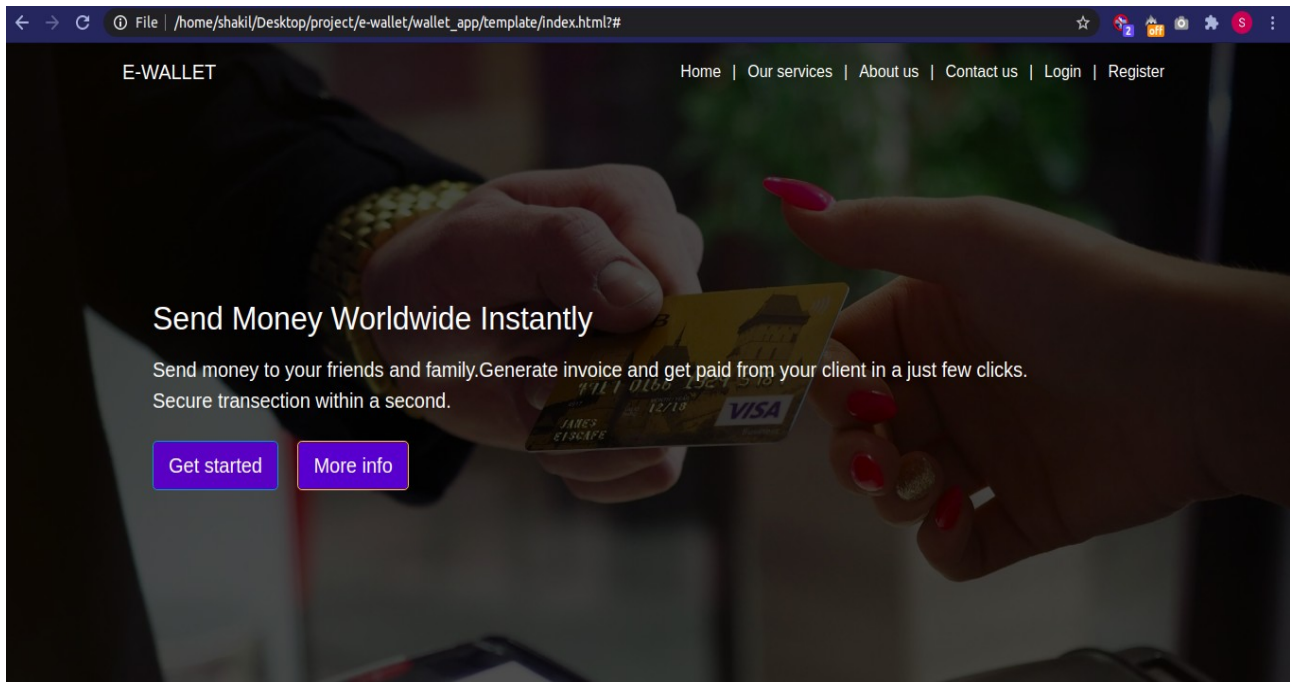
.hero-text{

    margin-left:10rem;
    margin-top: 12rem;
}

.calltoaction{
    background-color: #383CC1;
}

.moreinfobtn{
    background-color: #2827CC;
}
```

Now our website looks exactly like this:



Creating ‘our services’ section-

First of all let's add font-awesome cdn in our project because we need to use some of the related icons in this section. Add the cdn link at the bottom of the index page

```
<!-- javascript cdn -->
<script src="https://kit.fontawesome.com/f7191d8fdb.js" crossorigin="anonymous"></script>
```

Now add a new section with section id our-services.

```
<!-- our services section starts here -->

<section id="our-services">

</section>
```

Place the below codes inside <section id="our-service">

```
<section id="our-services mt-4">

<div class="container-fluid">

<h1 class="section-header text-dark text-center mt-4">Our Services</h1>
<div class="section-border text-center mx-auto"></div>

<div class="row mt-4 justify-content-around">

<div class="col-md-3 mt-3 service-column">
  <div class="col-content mt-4">

    <div class="section-icon-wrap d-flex justify-content-center mb-4">

      <i class="section-icon far text-danger fa-paper-plane"></i>
    </div>

    <h4 class="sub-header text-center mb-4 mt-4">Instant money transfer</h4>
    <p class="laed text-justify">Transfer money to your friends and family globally with a single mouse click.
    Lorem ipsum dolor sit amet consectetur adipisicing elit. Perferendis modi vero vitae mollitia non quaerat porro repellendus, nihil ex nobis. Autem officia error, dolores voluptatem quibusdam vel culpa reprehenderit dolore.
    </p>
  </div>

</div>
```

```
<div class="col-md-3 mt-3 service-column">
  <div class="col-content mt-4">

    <div class="section-icon-wrap d-flex justify-content-center mb-4">

      <i class="section-icon fas text-danger fa-file-invoice-dollar"></i>
    </div>

    <h4 class="sub-header text-center mb-4 mt-4">One click invoice</h4>
    <p class="laed text-justify">Generate invoice and send to your client within minutes and get paid.
    Lorem ipsum dolor sit amet consectetur adipisicing elit. Perferendis modi vero vitae mollitia non quaerat
    </p>

  </div>
</div>
```

```

<div class="col-md-3 mt-3 service-column">
  <div class="col-content mt-4">

    <div class="section-icon-wrap d-flex justify-content-center mb-4">

      <i class="section-icon fas text-danger fa-hand-holding-usd"></i>
    </div>

    <h4 class="sub-header text-center mb-4 mt-4">Request money in seconds</h4>
    <p class="laed text-justify">Request money to your friend.
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Perferendis modi
      vero vitae mollitia non quaerat porro repellendus, nihil ex nobis.
      Autem officia error, dolores voluptatem quibusdam vel culpa reprehenderit dolore.
    </p>

  </div>

</div>
</div>
</div>
</section>

```

And the css for our service section :

```

/** our service Section Css */

section {
  margin-top:4rem;
}

.section-icon{

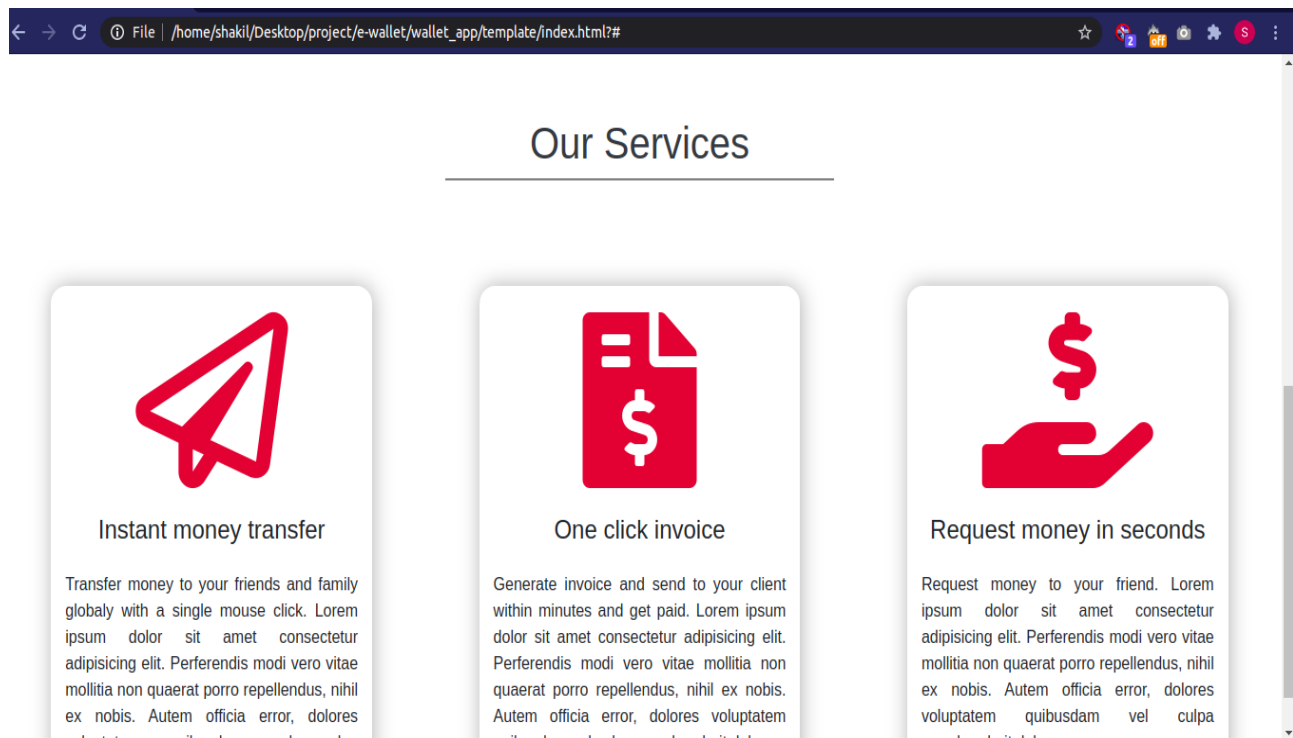
  font-size:10rem;}

.section-border{
  margin-bottom:5rem;
  border-bottom:2px solid gray;
  width:30vw;
}

.service-column{
  box-shadow: 1px 0px 20px rgba(0,0,0,0.3);
  border-radius: 1rem;
}

```

now our services section looks like this :-



Our service has been finished. Let's start the 'about us' section .

Creating 'About us' section :

Create a new section with id about-us . Inside about-us section One main row and two main columns. Inside each column we have nested columns and rows.

About us section has two subsection- One is 'who we are' and another is 'Our activity' .

Now we are all done with about us section. If you want to make it more appealing or add some animation, go ahead and do it. Full project is available on my github page .

Paste the codes inside about us section :

```

<section id="about-us">

<div class="container-fluid">
  <h1 class="section-header text-dark text-center mt-4">About us</h1>
  <div class="section-border text-center mx-auto"></div>
  <div class="row">
    <div class="col-md-7">
      <h3 class="text-center text-dark font-weight-bold mb-4 subsection-head">Who we are-</h3>
      <div class="row">
        <div class="col-md-6">
          
        </div>
        <div class="col-md-6">
          <p class="lead samble-text text-justify">We are the team of developers .....
Lorem ipsum dolor, sit amet, consectetur adipisicing elit. Facere offic
ia quae saepe optio necessitatibus quas excepturi voluptate repudiandae nostrum po
ssimus nulla ab minima accusamus eum eveniet fugiat impedit numquam beatae dignissimos laborum<br>
<br> itaque id hic cumque quo, commodi? Distin
ctio fugit consequuntur aspernatur dolor illum. Lorem ipsum dolor sit amet
consectetur adipisicing elit. Magnam praesentium neque perspiciatis porro rem, nulla obcaecati p>
</div>

</div>

</div>

<div class="col-md-5">
  <h3 class="text-center text-dark font-weight-bold mb-4 subsection-head">Our activity-</h3>
  <ul class="list-group">
    <li class="list-group-item">We help to keep your transection fast and safe</li>
    <li class="list-group-item">Money Lorem ipsum dolor sit, amet consectetur adip. </li>
    <li class="list-group-item">Transferring money to your friends Lorem ipsum</li>
    <li class="list-group-item">Shop worldwide instantly without any hustle Lorem ipsum do
or, sit.</li>
    <li class="list-group-item">World largest money transferring network ipsum dolor sit,
amet consectetur adip. </li>

  </div>

</div>
</div>

</section>

```

Our styles.css files for about-us section is here:

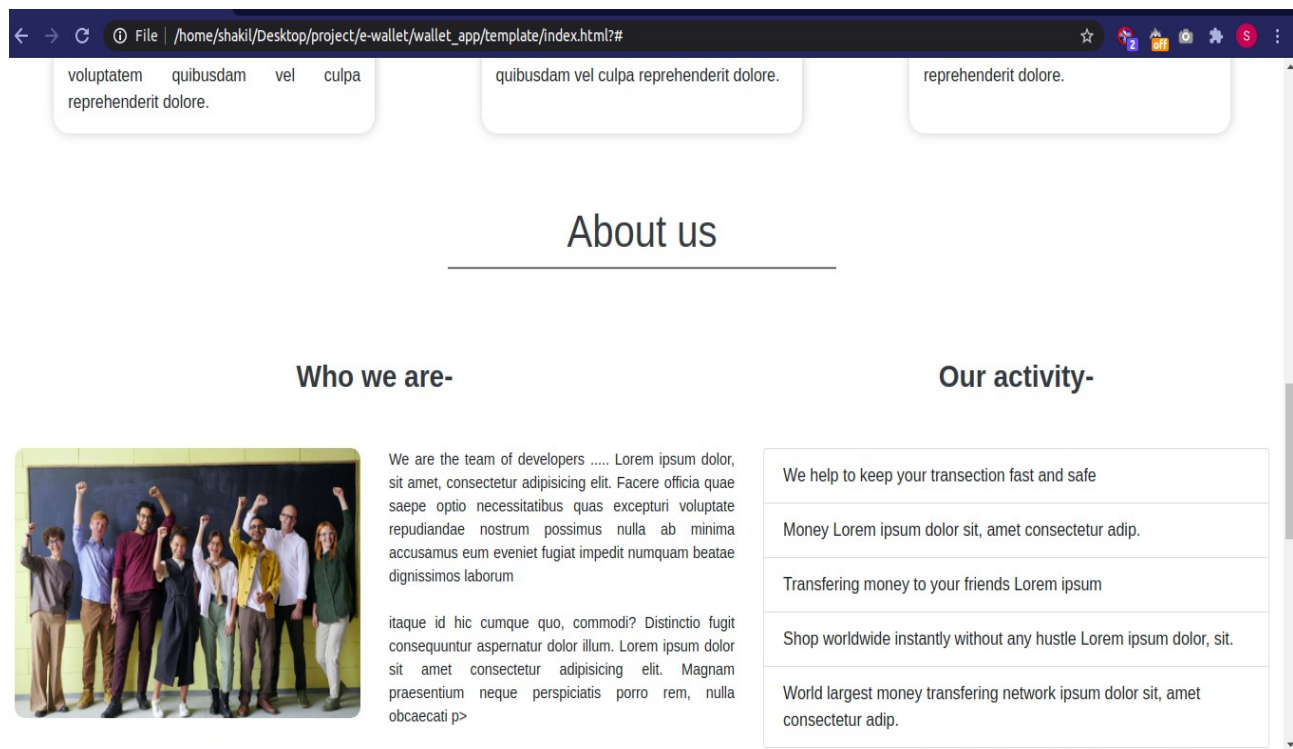
Styles.css :

```
.img-rounded{
    border-radius: 10px !important;
}

.subsection-head{
    margin-bottom: 3rem !important;
}

.samble-text{
    font-size: 0.9rem;
}
```

Now our about us section looks like this:



We are all done with this section. Let's add another section name Faq where all the frequently asked questions will be displayed with bootstrap accordion.

Creating faq section:

Now add a FAQ link to the navbar after abouts us nav item. To do this just type new `` tag with the following code:

```
<li class="nav-item">
  <a class="nav-link" href="#"> Faq </a>
</li><span class="bar text-white mx-1 my-auto"> |</span>
```

Now add a new section with section id 'faq'. Again paste the codes inside this section:
index.html:

```
<section id="faq">
  <div class="container-fluid">
    <h1 class="section-header text-dark text-center mt-4">Frequently asked questions</h1>
    <div class="section-border text-center mx-auto"></div>
    <div class="row justify-content-around">
      <div class="col-md-5">

        <div class="panel-group" id="accordion">
          <!--repeat the code 5 times for this column-->

          <div class="panel panel-default">
            <div class="panel-heading">
              <h6 class="panel-title d-flex justify-content-between" data-toggle="collapse" data-parent="#accordion">
                <a >How much the fee to send money?

                </a>
                <i class="mr-4 arrow-icon fas fa-arrow-circle-down"></i>

              </h6>
            </div>
            <div id="collapse1" class="panel-collapse collapse in">
              <div class="panel-body">Lorem ipsum dolor sit amet, consectetur adipisicing elit,
                sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
                quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
```


inside the col-md-5 we have panel-default repeat the entire code from the html comments 5 times to create 5 faq questions in a column. Now add another column by typing this:

```
<!--another faq column -->

<div class="col-md-5">

<!-- repeat the code below to create multiple questions -->

<div class="panel-group" id="accordion">

<div class="panel panel-default">
<div class="panel-heading">
<h6 class="panel-title d-flex justify-content-between" data-toggle="collapse" data-parent="#accordion">
<a > Can i withdraw money to the bank?

</a>
<i class="mr-4 arrow-icon fas fa-arrow-circle-down"></i>

</h6>
</div>
<div id="collapse6" class="panel-collapse collapse in">
<div class="panel-body">Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</div>
</div>
</div>
</div>
</div>
```

Our styles.css file for this section:

Styles.css:

```
/** faq section **/

.panel-heading a{
    color:white;
    margin-left: 1rem;
}

.panel-title{
    padding-top:1rem;
    padding-bottom: 1rem;
}

.panel-heading a:hover{
    text-decoration: none;
}

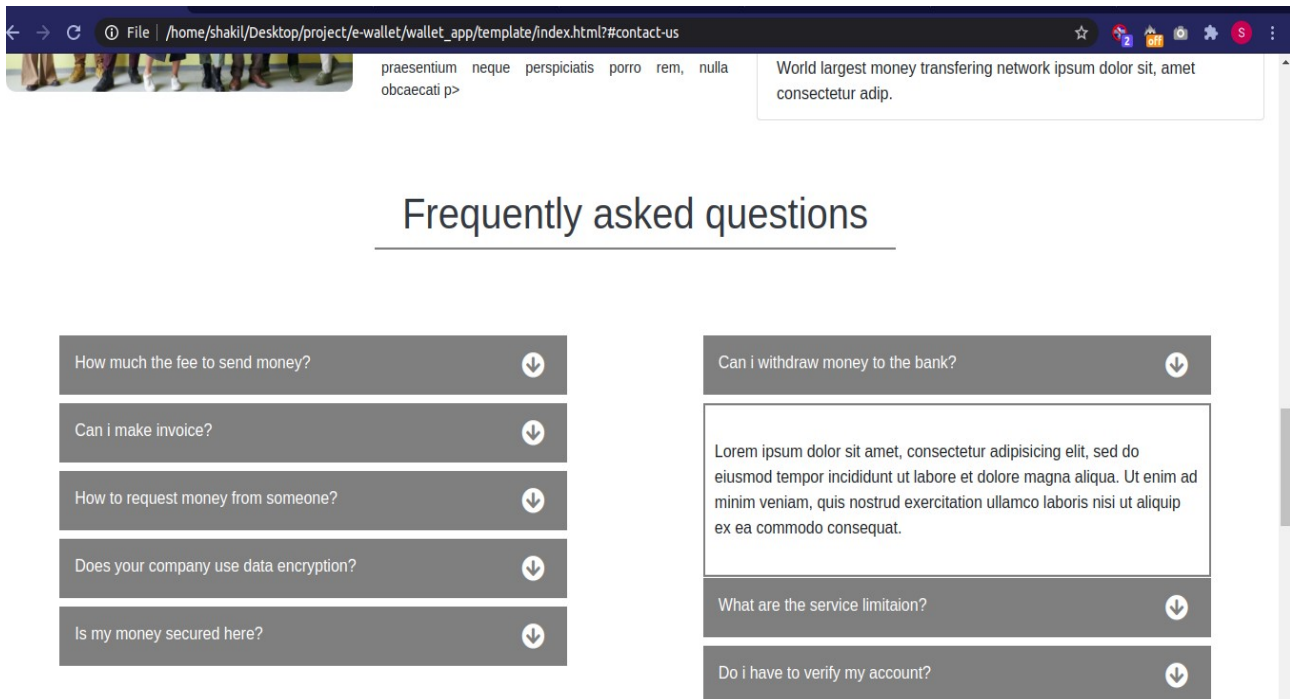
.panel-heading{
    background-color: rgba(0,0,0,0.5);
    color:white;
    margin-bottom:1px !important;
    margin-top:1px;
}

.panel-heading:hover{
    border:1px solid red;
}

.panel-body{
    border:2px solid gray;
    padding-top:2rem;
    padding-bottom: 2rem;
    padding-left:10px;
    padding-right:10px;
}

.arrow-icon{
    font-size: 1.5rem;
}
```

Now our Faq section looks like this :



Creating contact us form:

Contact us section has only a form and some input for now. We will work on this form later .

create another section with id contact-us and paste the codes below:

styles.css:

```
.btn-send{
background-color: #538FFB;
}
```

```

<!-- contact us section starts here -->

<section id="contact-us">
  <div class="container-fluid">
    <h1 class="section-header text-dark text-center mt-4">Contact us</h1>
    <div class="section-border text-center mx-auto"></div>
    <div class="row justify-content-center">
      <div class="col-md-7">

        <form>
          <div class="form-group">
            <label for="name">Name : </label>
            <input type="text" name="name" class="form-control" id="name">
          </div>

          <div class="form-group">
            <label for="email">Email : </label>
            <input type="email" id="email" name="email" class="form-control">
          </div>

          <div class="form-group">
            <label for="name">Subject : </label>
            <input type="text" name="subject" class="form-control">
          </div>
          <div class="form-group">
            <label for="name">Message : </label>
            <textarea name="subject" rows="10" class="form-control" placeholder="Message starts here">

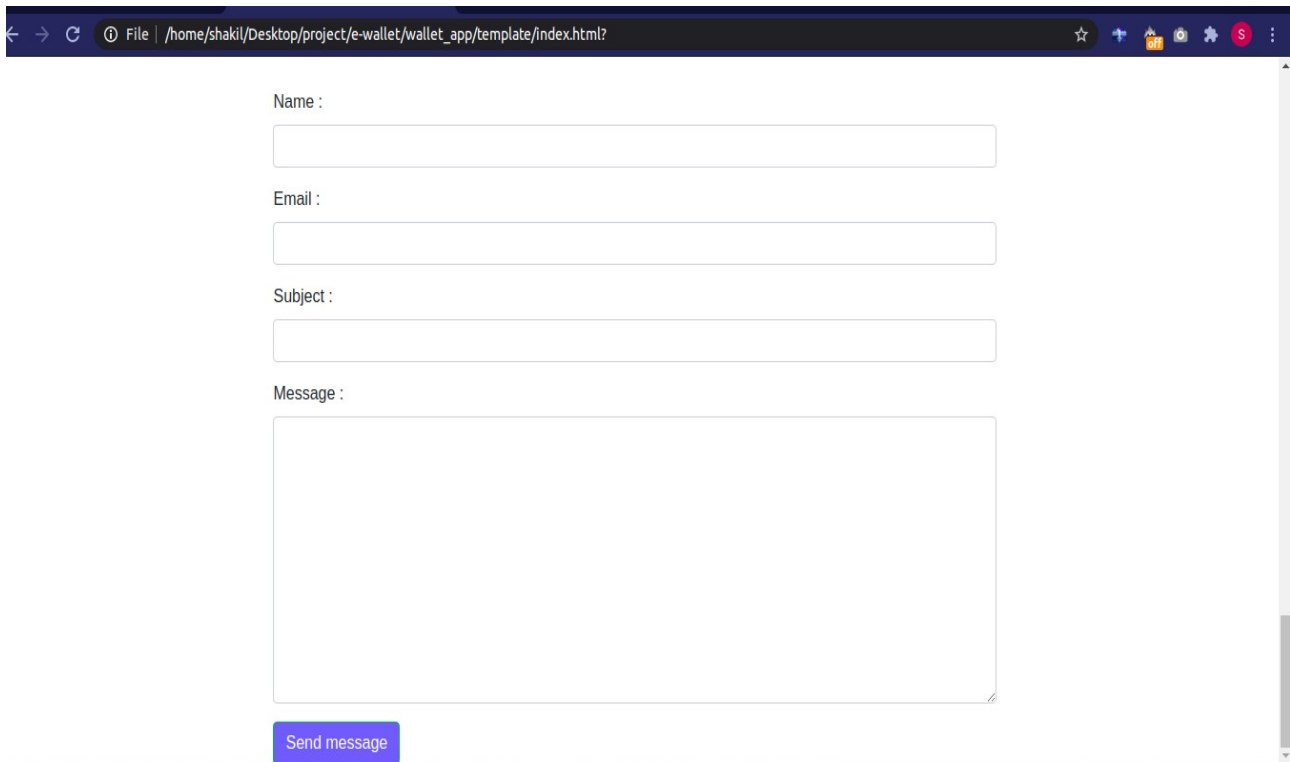
              </textarea>
            </div>
            <button class="btn btn-send btn-success btn-md text-white" type="submit">Send message
          </button>

        </form>

      </div>
    </div>
  </div>
</section>

```

Till now our contact-us section looks like this:



The screenshot shows a web browser window with the address bar displaying the file path: `File | /home/shakil/Desktop/project/e-wallet/wallet_app/template/index.html?`. The browser's address bar also shows standard navigation icons (back, forward, refresh) and a star icon for bookmarks. The main content area of the browser displays a contact form. The form consists of four input fields: 'Name :', 'Email :', 'Subject :', and 'Message :'. Each field is a simple text box with a light gray border. Below the 'Message :' field is a large, empty text area. At the bottom of the form is a blue button with the text 'Send message' in white. The browser's status bar at the bottom is dark gray.

To end the boring stuff of webdevelopment let's add the last footer section and after that we will work on user creating and authentication which will be so much fun than just UI designing.

Creating footer section:

Start with the footer tag and inside this tag paste the codes below:

```
<footer class="site-footer bg-dark">

<div class="row justify-content-around">
  <div class="col-md-3">
    <h5 class="text-white text-center font-weight-bold foot-header">E-wallet</h5>
    <div class="section-border text-center mx-auto"></div>
    <p class="lead text-white text-justify foot-text">
      Lorem, ipsum dolor sit amet consectetur adipisicing elit.
      Consequatur ea pariatur similique maxime libero fuga voluptas
      nesciunt minima corrupti nobis sapiente, distinctio, corporis
      impedit maiores debitis fugiat accusantium, earum!. Lorem,
      ipsum dolor sit amet consectetur adipisicing, elit.
      Dolorem ipsum aut qui ex voluptatibus architecto culpa suscipit magni neque deserunt.</p>
    </div>
  </div>
</div>
```

```

<div class="col-md-3">
  <h5 class="text-white text-center font-weight-bold foot-header">Quick links</h5>
  <div class="section-border text-center mx-auto"></div>
  <ul class="list-unstyled foot-text foot-lnk">
    <li class="m-2"><a class="text-white" href="#home">home</a></li>
    <li class="m-2"><a class="text-white" href="#about-us">about us</a></li>
    <li class="m-2"><a class="text-white" href="#faq">faq</a></li>
    <li class="m-2"><a class="text-white" href="#contact-us">contact us</a></li>
    <li class="m-2"><a class="text-white" href="#login">login</a></li>
    <li class="m-2"><a class="text-white" href="#register">register</a></li>

  </ul>
</div>

<div class="col-md-3">
  <h5 class="text-white text-center font-weight-bold foot-header">Contact</h5>
  <div class="section-border text-center mx-auto"></div>
  <ul class="list-unstyled foot-text foot-lnk">
    <li class="mb-2 text-white"><i class="fas fa-home mr-3 text-white"></i>
    New York, NY 10027, US</li>
    <li class="mb-2 text-white"><i class="fas fa-envelope-square mr-3 text-white"></i>
    examlpl@e-wallet.com</li>
    <li class="mb-2 text-white"><i class="fas fa-phone-volume mr-3 text-white"></i>
    +01 234 567 89</li>
    <li class="mb-2 text-white"><i class="fas fa-fax mr-3 text-white"></i>
    +01 234 567 89</li>

  </ul>
</div>

</div>
<div class="row copyright-text">
  <div class="col-md-12 text-center text-white font-weight-bold pt-2 pb-2">
    &copy; 2021 Copyright E-wallet
  </div>
</div>

</footer>

```

styles.css :

```
/* footer section starts here */

footer{
    margin-top:4rem !important;
}

.foot-header{
    margin-top:2rem;
    margin-bottom: 1rem !important;
}

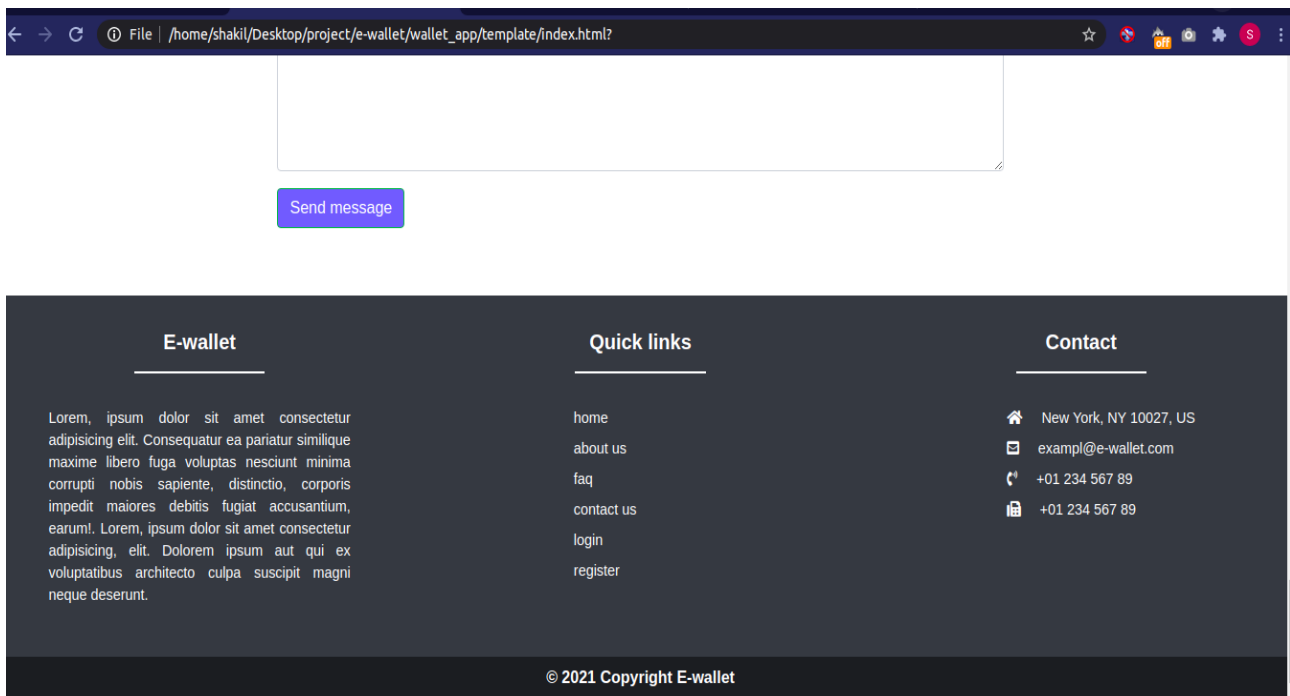
.foot-text{
    font-size: 0.9rem;
    margin-bottom: 3rem;
}

footer .section-border{
    margin-bottom:2rem;
    border-bottom:2px solid white;
    width:10vw;
}

.foot-lnk{
    margin-left: 5rem !important;
    margin-right: auto !important;
}

.copyright-text{
    background-color: rgba(0,0,0,0.5);
}
```

Let's see how it looks like after applying html and css for the footer area:



Now, We are done with the UI of index.html file. Let's now create login and registration page and I promise after that I will start the core development part with django .

Creating registration file:

Create a file with name 'registration.html' in the same directory where the index page is located. Paste the code below in 'registration.html' and don't forget to add header section in the file. You can copy the entire head section from 'index.html' for now. Later on this tutorial we will use django template system.

Registration.html:

```
<body>

<div class="container-fluid">

    <div class="row">
        <div class="col-md-7 reg-img">

        </div>
        <div class="col-md-5">
            <h3 class="text-center mt-4 font-weight-bold">Registration</h3>
            <div class="form-area">
                <form>
                    <div class="form-group">
                        <label for="fullname">Full Name :</label>
                        <input id="fullname" type="text" class="form-control"
                            placeholder="full name">
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
```



```

        <div class="form-group">
            <label for="email">Email :</label>
            <input id="email" type="email" class="form-control"
                placeholder="yourname@email.com">
        </div>
        <div class="form-group">
            <label for="Address">Address :</label>
            <input id="Address" type="text" class="form-control"
                placeholder="address">
        </div>
        <div class="form-group">
            <label for="Password1">Password :</label>
            <input id="Password1" type="password" class="form-control"
                placeholder="password">
        </div>
        <div class="form-group">
            <label for="Password2">Retype Password :</label>
            <input id="Password2" type="password" class="form-control"
                placeholder="confirm password">
        </div>
        <button type="submit" class="btn btn-md btn-info text-white">
            Register</button>
    </form>
    </div>
</div>
<!-- you can add javascript cdn here-->
<body>

```

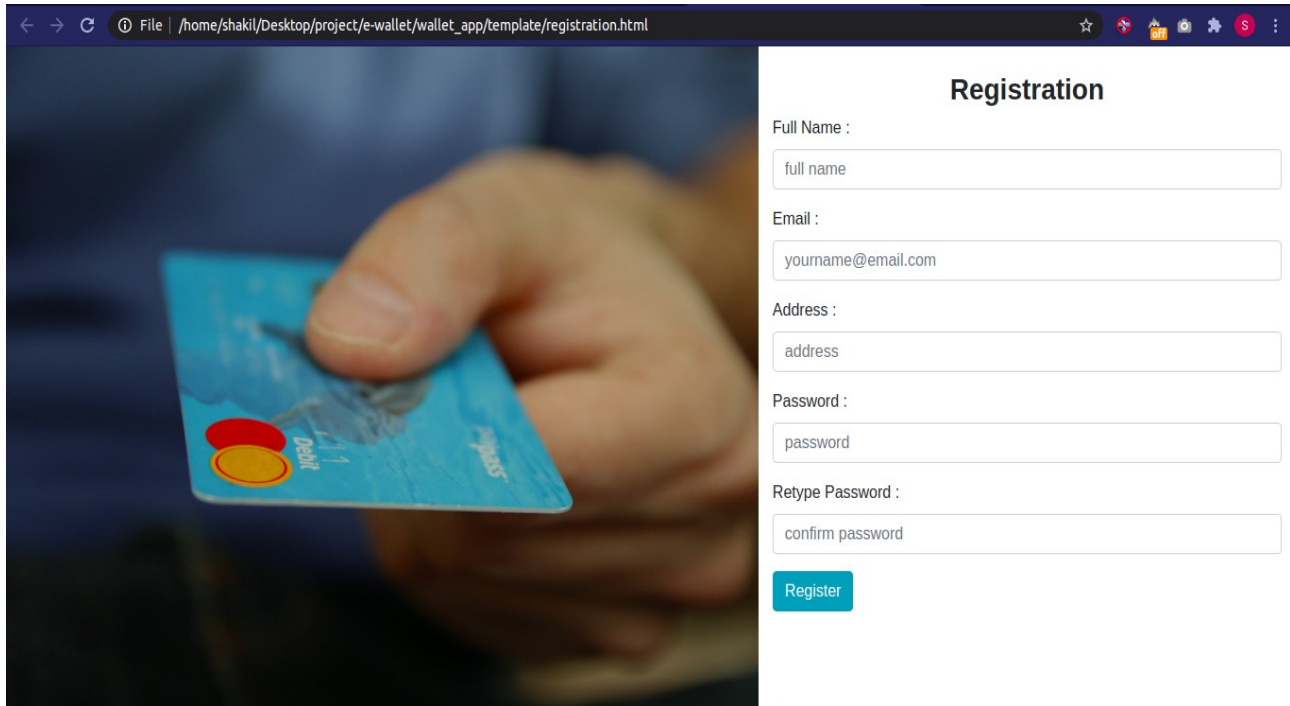
styles.css :

```

.reg-img{
    background-image: linear-gradient(to right,rgba(0,0,0,0.2),rgba(0,0,0,0.2)),
        url('../img/registration-min.jpg');
    background-size: cover;
    height: 100vh;
    width: 100%;
}

```

Now our registration page looks like this:



The screenshot shows a web browser window with the address bar displaying the file path: `File | /home/shakil/Desktop/project/e-wallet/wallet_app/template/registration.html`. The page content features a registration form on the right side, titled "Registration". The form contains the following fields and elements:

- Full Name :** An input field with the placeholder text "full name".
- Email :** An input field with the placeholder text "yourname@email.com".
- Address :** An input field with the placeholder text "address".
- Password :** An input field with the placeholder text "password".
- Retype Password :** An input field with the placeholder text "confirm password".
- Register**: A blue button with white text.

The background of the page is a blurred image of a hand holding a blue credit card with a red and yellow logo.

We will add google re-captcha in this form later. Till now everything looks great.

Creating login page :

I know you are eagerly waiting for the **django** part . Don't worry , after designing the login page I will dive into the django part.

Now let's create the login page. To do that, first create a file with name 'login.html' in the same directory where the registration and index page is located. Again add the head section like registration page.

Styles.css :

```
.loginform{
    margin-top:auto;
    margin-bottom: auto;
}
```

login.html :

```
<body>

<div class="container-fluid">

  <div class="row">
    <div class="col-md-7 reg-img">

      </div>
      <div class="col-md-5 loginform">
        <h3 class="text-center mt-4 font-weight-bold">E-wallet login</h3>
        <div class="form-area ml-4 mr-4">
          <form>

            <div class="form-group">
              <label for="email">Email :</label>
              <input id="email" type="email" class="form-control"
                placeholder="yourname@email.com">
            </div>

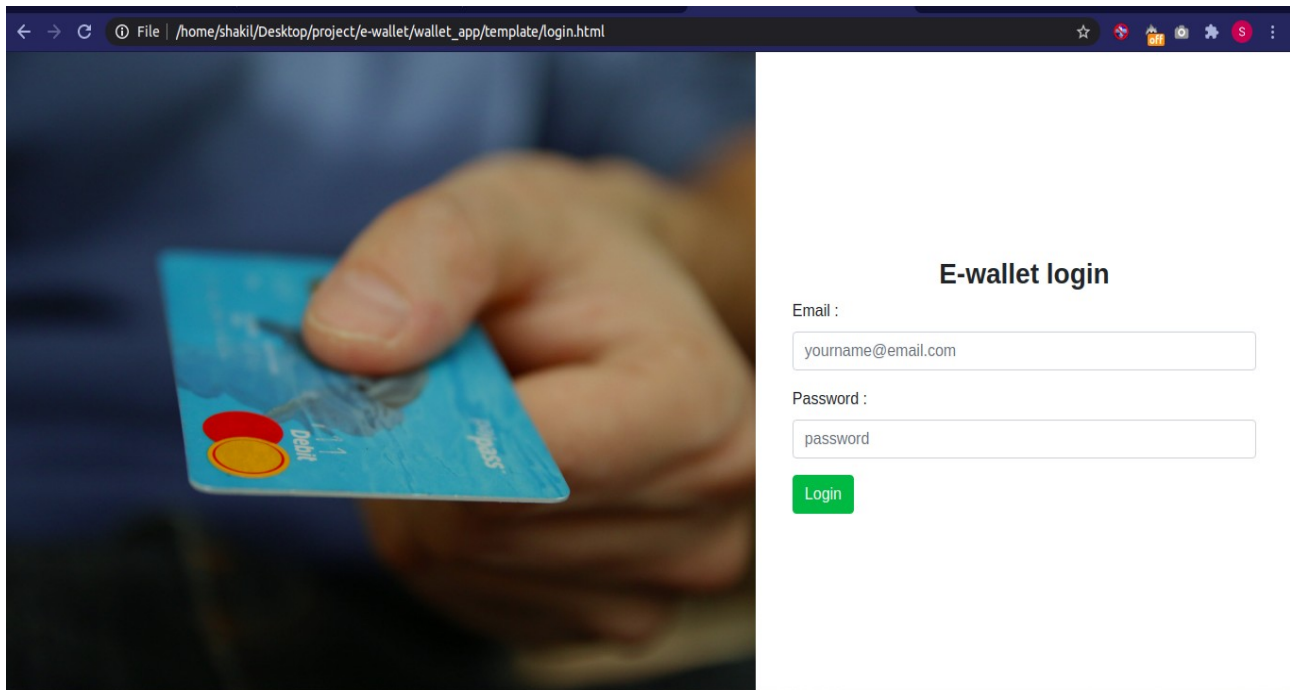
            <div class="form-group">
              <label for="Password1">Password :</label>
              <input id="Password1" type="password" class="form-control"
                placeholder="password">
            </div>

            <button type="submit" class="btn btn-md btn-success text-white">

          </form>
        </div>
      </div>
    </div>
  </div>

</div>
</body>
```

Now our login page looks like this:



We are done with login and registration page for now.

Now without wasting any time let's dive into the django.

At first we need to structure our files in a django way.so let's do it now .

Django file structuring :

Create a folder with name 'staticfiles' where all our static files (image,js,css) will be stored. Cut three folders img,css and js and paste it to the newly created folder.

Now, we need a base file where only header and footer contents will be there.

So, create a 'base.html' file in the template directory.

What is a base file?

Well, a base file is a file which normally contains the header and footer sections or any types of static contents of the website so that we can reuse these contents in another files. Django has a builtin template tag `{% block something %}` which allows developers to render contents dynamically.

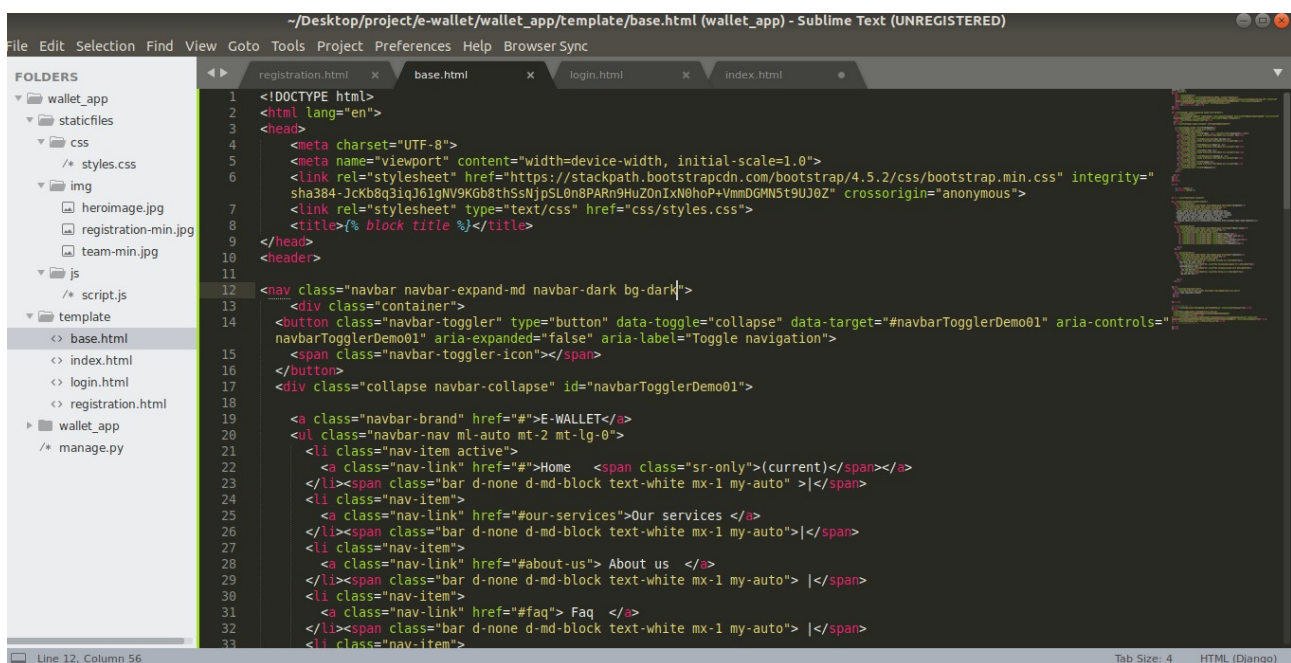
Code inside base.html:

Copy the head section from 'index.html' and paste it Inside 'base.html' file.

To render title dynamically replace the contents of the title tag with

`{% block title %} {% endblock %}` . Again copy the full nav from index.html file and paste it below body tag . But, there is a problem. Our navbar has a class of `bg-transparent` which makes the navbar transparent that we dont want in base file. Because we only need transparent nav in the index file only. So to avoid this problem just replace it with the class '`.bg-dark`' for now.

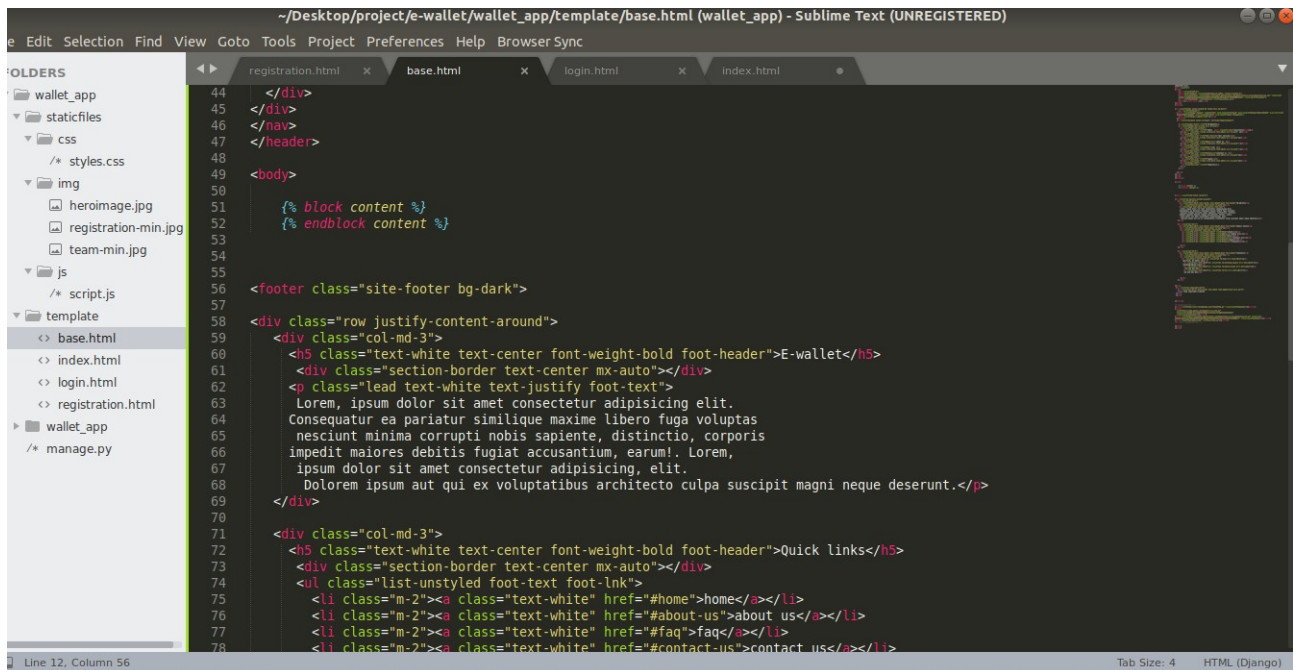
Now it looks like this :



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" integrity="
7   sha384-Jckb8q3iqJ61gN9Kgb8thSsNjpsL0n8PARn9HuZ0IXN0hoP+VmmGMNSt9UJ0Z" crossorigin="anonymous">
8   <link rel="stylesheet" type="text/css" href="css/styles.css">
9 </head>
10 <header>
11
12 <nav class="navbar navbar-expand-md navbar-dark bg-dark">
13   <div class="container">
14     <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarTogglerDemo01" aria-controls="
15     navbarTogglerDemo01" aria-expanded="false" aria-label="Toggle navigation">
16       <span class="navbar-toggler-icon"></span>
17     </button>
18     <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
19       <a class="navbar-brand" href="#">E-WALLET</a>
20       <ul class="navbar-nav ml-auto mt-2 mt-lg-0">
21         <li class="nav-item active">
22           <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
23         </li><span class="bar d-none d-md-block text-white mx-1 my-auto" >|</span>
24         <li class="nav-item">
25           <a class="nav-link" href="#our-services">Our services </a>
26         </li><span class="bar d-none d-md-block text-white mx-1 my-auto" >|</span>
27         <li class="nav-item">
28           <a class="nav-link" href="#about-us">About us </a>
29         </li><span class="bar d-none d-md-block text-white mx-1 my-auto" >|</span>
30         <li class="nav-item">
31           <a class="nav-link" href="#faq">Faq </a>
32         </li><span class="bar d-none d-md-block text-white mx-1 my-auto" >|</span>
33         <li class="nav-item">
```

After the navbar we want to show contents from another file which extends the base file. So, below navbar use the django template tag

{%block content %} {% endblock %}'. After that, copy footer section and rest of the elements from 'index.html' and paste it after {%block content%} {%endblock%} tag. Which looks like this:



```
44 </div>
45 </div>
46 </nav>
47 </header>
48
49 <body>
50
51     {% block content %}
52     {% endblock content %}
53
54
55
56 <footer class="site-footer bg-dark">
57
58     <div class="row justify-content-around">
59
60         <div class="col-md-3">
61             <h5 class="text-white text-center font-weight-bold foot-header">E-wallet</h5>
62             <div class="section-border text-center mx-auto"></div>
63             <p class="lead text-white text-justify foot-text">
64                 Lorem, ipsum dolor sit amet consectetur adipisicing elit.
65                 Consequatur ea pariatur similique maxime libero fuga voluptas
66                 nesciunt minima corrupti nobis sapiente, distinctio, corporis
67                 impedit maiores debitis fugiat accusantium, earum! Lorem,
68                 ipsum dolor sit amet consectetur adipisicing, elit.
69                 Dolorem ipsum aut qui ex voluptatibus architecto culpa suscipit magni neque deserunt.</p>
70         </div>
71
72         <div class="col-md-3">
73             <h5 class="text-white text-center font-weight-bold foot-header">Quick links</h5>
74             <div class="section-border text-center mx-auto"></div>
75             <ul class="list-unstyled foot-text foot-lnk">
76                 <li class="m-2"><a class="text-white" href="#home">home</a></li>
77                 <li class="m-2"><a class="text-white" href="#about-us">about us</a></li>
78                 <li class="m-2"><a class="text-white" href="#faq">faq</a></li>
79                 <li class="m-2"><a class="text-white" href="#contact-us">contact us</a></li>
```

Now we are ready to use this base file anywhere we want. But throughout the tutorial we will not use it for this project . This was for the educational purpose only .

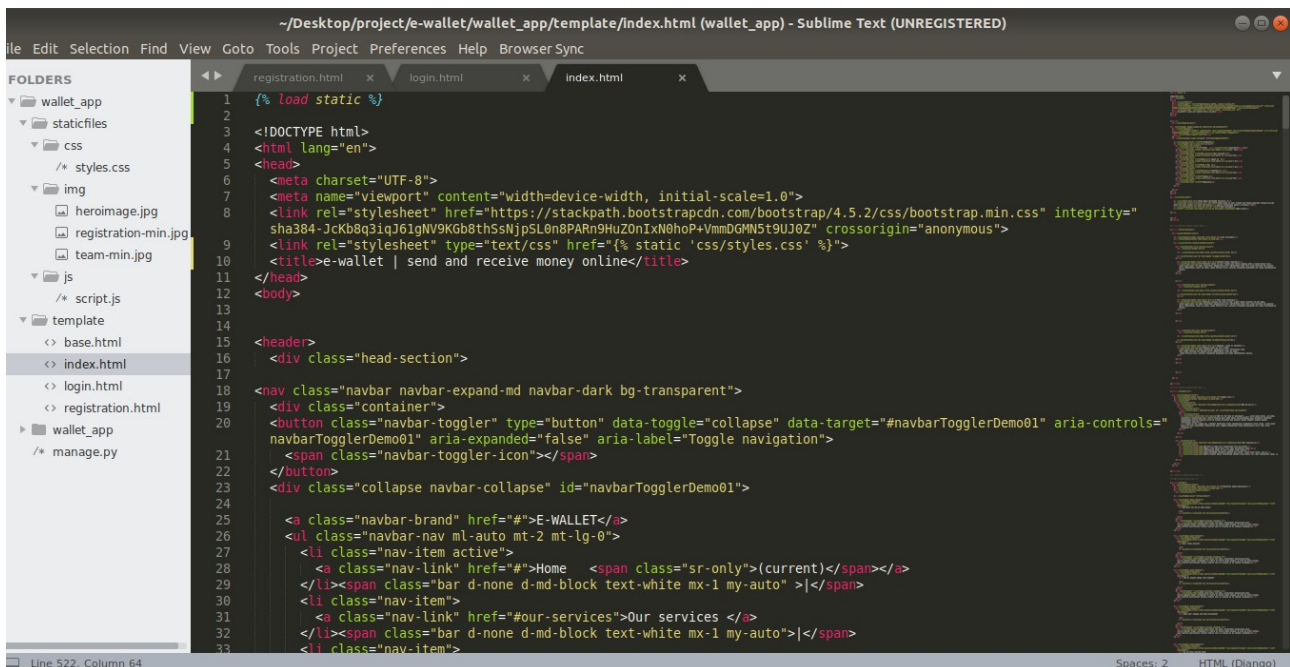
Configuring static path :

Before moving our static files to staticfiles directory we used simple structural file path in our html files . As ,it is now moved to staticfiles folder django will now handle these files by creating a static file path.

To configure the paths of html files we need to first declare a template tag, '{% load static %}' above all of the html codes .

After that we have to update all of our local links with {% static 'localpath' %} django template tag. Let's do it for index.html first.

Index.html:



```
1 {% load static %}
2
3 <!DOCTYPE html>
4 <html lang="en">
5 <head>
6 <meta charset="UTF-8">
7 <meta name="viewport" content="width=device-width, initial-scale=1.0">
8 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" integrity="
sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMW5t9UJ0Z" crossorigin="anonymous">
9 <link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">
10 <title>e-wallet | send and receive money online</title>
11 </head>
12 <body>
13
14 <header>
15 <div class="head-section">
16
17 <nav class="navbar navbar-expand-md navbar-dark bg-transparent">
18 <div class="container">
19 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarTogglerDemo01" aria-controls="
navbarTogglerDemo01" aria-expanded="false" aria-label="Toggle navigation">
20 <span class="navbar-toggler-icon"></span>
21 </button>
22 <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
23
24 <a class="navbar-brand" href="#">E-WALLET</a>
25 <ul class="navbar-nav ml-auto mt-2 mt-lg-0">
26 <li class="nav-item active">
27 <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
28 </li><span class="bar d-none d-md-block text-white mx-1 my-auto" ></span>
29 <li class="nav-item">
30 <a class="nav-link" href="#">Our services </a>
31 </li><span class="bar d-none d-md-block text-white mx-1 my-auto"></span>
32 <li class="nav-item">
33
```

update all the local links with this format for index.html , login.html and registration.html files and don't forget to init the {% load static %} tag.

Now collect the static files to staticfiles folder by typing a command in terminal terminal:

```
(env) shakil@shakil:~/Desktop/project/e-wallet/wallet_app$ python manage.py collectstatic
137 static files copied to '/home/shakil/Desktop/project/e-wallet/wallet_app/static'.
```

We are all done to write our first django code.

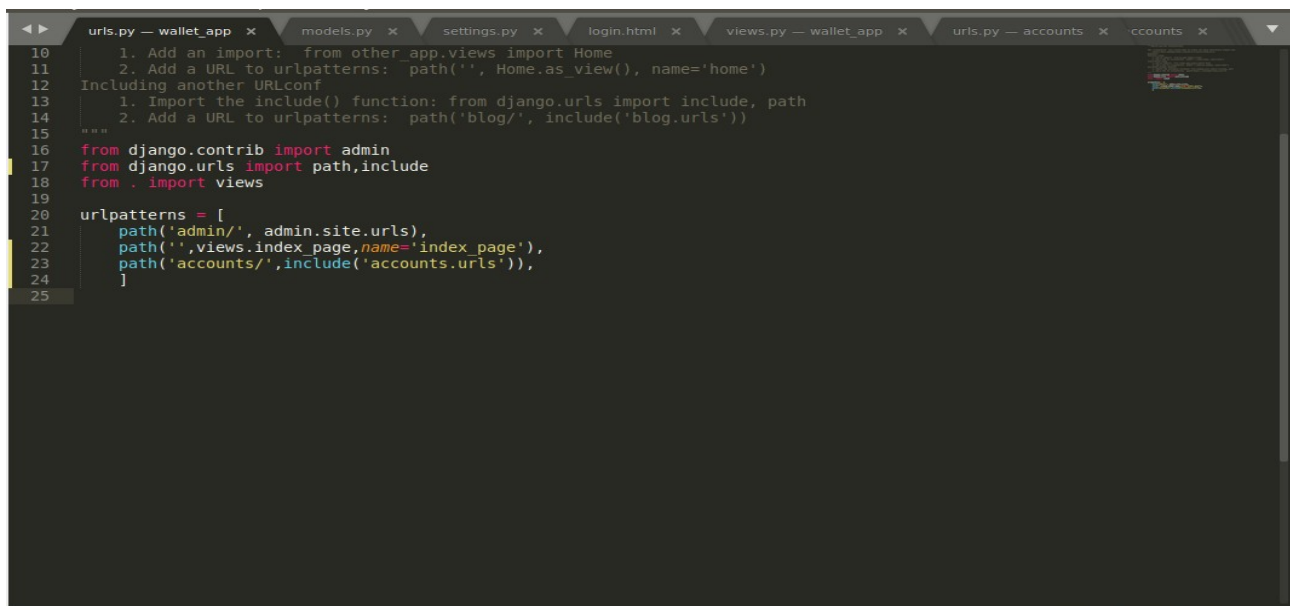
Routing the urls for template rendering:

In the 'wallet_app' directory you will see a python file name 'urls.py'. Open it with the text editor and first import the views.py file by typing

‘from . import views’ . Although we have not created views.py file but we will create it later . Now add these lines to urlpatterns=[] :

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('',views.index_page,name='index_page'),  
    path('accounts/',include('accounts.urls')),  
]
```

Now our urls.py file looks like:

A screenshot of a code editor with multiple tabs. The active tab is 'urls.py — wallet_app'. The code in the editor is as follows:

```
10 1. Add an import: from other_app.views import Home  
11 2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')  
12 Including another URLconf  
13 1. Import the include() function: from django.urls import include, path  
14 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))  
15  
16 from django.contrib import admin  
17 from django.urls import path,include  
18 from . import views  
19  
20 urlpatterns = [  
21     path('admin/', admin.site.urls),  
22     path('',views.index_page,name='index_page'),  
23     path('accounts/',include('accounts.urls')),  
24 ]  
25
```

Now create a views.py file in the same directory and add the following lines:

views.py:

```
from django.shortcuts import render  
  
def index_page(request):  
    return render(request,'index.html')
```


At first we imported render method from django shortcuts and then created some functions (index_page, registration_page, login_page) as we imported in the urls.py file and render our destination page 'index.html', 'registration.html' and 'login.html'.

Note: you can use TemplateView in the urls.py to render the pages without using any function. Just import it by typing

'from django.views.generic import TemplateView'

and urls.py :

urlpatterns=[path('', TemplateView.as_view(template_name='index.html'), name='index_page'),]

Testing our urls:

Run the command below in the terminal to start the local server :

terminal :

```
(env) shakil@shakil:~/Desktop/project/e-wallet/wallet_app$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
```

```
System check identified no issues (0 silenced).
```

```
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations. Run 'python manage.py migrate' to apply them.
```

```
January 01, 2021 - 09:25:50
```

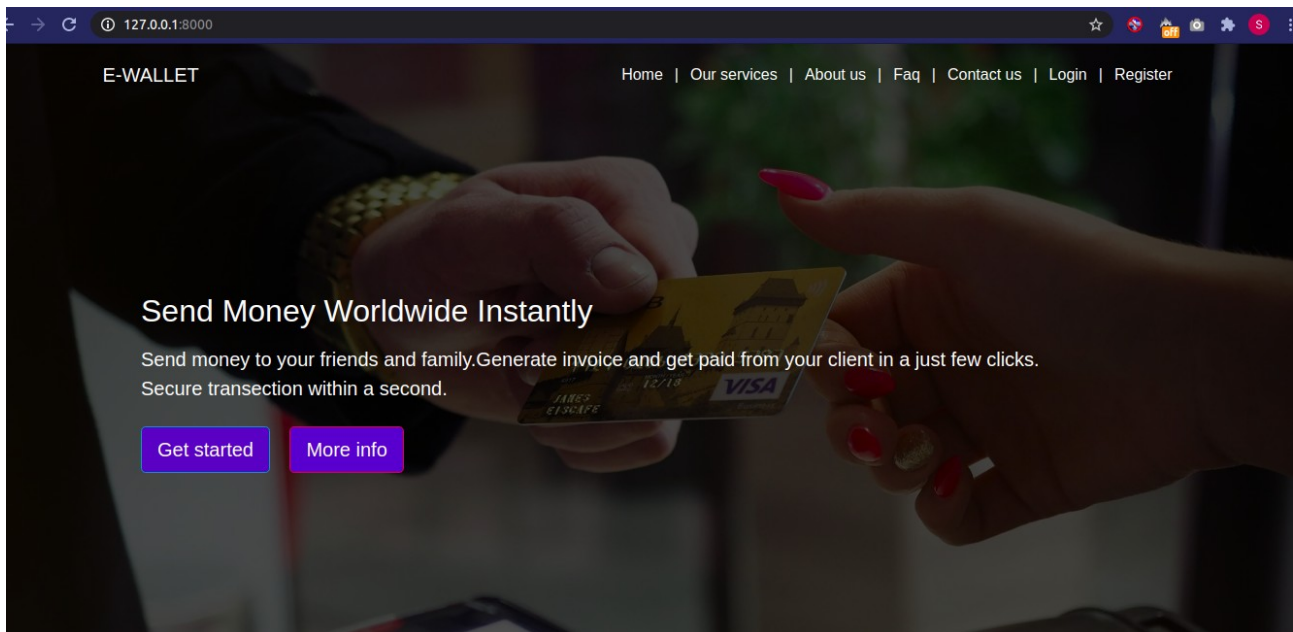
```
Django version 3.1.4, using settings 'wallet_app.settings'
```

```
Starting development server at http://127.0.0.1:8000/
```

```
Quit the server with CONTROL-C.
```

visit the site by typing in the address bar '<http://127.0.0.1:8000/>'. It will render our index.html file.

<http://127.0.0.1:8000/> :



let's update our links in the index.html, login.html and registration.html files.

In the index.html rewrite the login and registration page link with django template tag . First in the nav-bar update these ``: values with `{% url 'name' %}`

```
<li class="nav-item">
  <a class="nav-link" href="{% url 'login_page' %}">Login </a>
</li><span class="bar d-none d-md-block text-white mx-1 my-auto">|</span>
<li class="nav-item">
  <a class="nav-link" href="{% url 'registration_page' %}">Register</a>
</li>
```

and ofcourse in the footer:

```
<ul class="list-unstyled foot-text foot-lnk">
  <li class="m-2"><a class="text-white" href="#home">home</a></li>
  <li class="m-2"><a class="text-white" href="#about-us">about us</a></li>
  <li class="m-2"><a class="text-white" href="#faq">faq</a></li>
  <li class="m-2"><a class="text-white" href="#contact-us">contact us</a></li>
  <li class="m-2"><a class="text-white" href="{% url 'login_page' %}">login</a></li>
  <li class="m-2"><a class="text-white" href="{% url 'registration_page' %}">register</a></li>
</ul>
```

Again update the links in the registration.html and login.html files.

Login.html:

```
<p class="text-success mb-2 mt-2 ">Don't have an account?  
<a class="text-success font-weight-bold "  
  href="{% url 'registration_page' %}">Register</a></p>
```

Registration.html:

```
<p class="text-success mb-2 mt-2 ">Already have an account?  
<a class="text-success font-weight-bold "  
  href="{% url 'login_page' %}">login</a></p>
```

Note: We have not created accounts/urls.py yet so, dynamic link tag will not work. You can remove this and test the url for now and add this later.

User registration :

To register user we will use custom user model because the default user model only contains username and password fields. We need to customize all the fields to use email address as username and to add additional fields.

Create a new app by typing in the terminal “python manage.py startapp accounts’ and open ‘models.py’ with your text editor/ide .Don’t forget to register this app at the settings.py .

Import some classes and methods which will help us to create custom user fields by extending those.

Models.py:

```
from django.db import models
from django.contrib.auth.base_user import AbstractBaseUser, BaseUserManager
from django.utils.translation import gettext_lazy as _
from django.contrib.auth.models import PermissionsMixin
import datetime
from django.utils import timezone
```

AbstractBaseUser and BaseUserManager are used to customize user model. Don't worry, you don't need to memorize these because all codes are available in your python site-packages and inside django folder. If you just follow the path 'django>contrib>auth>models.py' you will find all the codes. Just copy and paste for now. To make this tutorial beginner friendly I will skip the in dept discussion of these special classes.

Now extends the 'BaseUserManager' to create our custom model manager for user.

Models.py :

```
class UserManager(BaseUserManager):
    use_in_migrations = True

    def _create_user(self, email, password, **extra_fields):
        """
        Create and save a user with the given username, email, and password.
        """
        if not email:
            raise ValueError("The given email must be set")

        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user
```

```

def create_user(self, email,password, **extra_fields):
    extra_fields.setdefault('is_staff', False)
    extra_fields.setdefault('is_superuser', False)
    return self._create_user(email,password, **extra_fields)

def create_superuser(self, email, password=None, **extra_fields):
    extra_fields.setdefault('is_staff', True)
    extra_fields.setdefault('is_superuser', True)

    if extra_fields.get('is_staff') is not True:
        raise ValueError('Superuser must have is_staff=True.')
    if extra_fields.get('is_superuser') is not True:
        raise ValueError('Superuser must have is_superuser=True.')

    return self._create_user(email,password, **extra_fields)

def with_perm(self, perm, is_active=True, include_superusers=True, backend=None, obj=
    if backend is None:
        backends = auth._get_backends(return_tuples=True)
        if len(backends) == 1:
            backend, _ = backends[0]
        else:
            raise ValueError(
                'You have multiple authentication backends configured and '
                'therefore must provide the `backend` argument.'
            )
        elif not isinstance(backend, str):
            raise TypeError(
                'backend must be a dotted import path string (got %r).'
                % backend
            )
        else:
            backend = auth.load_backend(backend)
        if hasattr(backend, 'with_perm'):
            return backend.with_perm(
                perm,
                is_active=is_active,
                include_superusers=include_superusers,
                obj=obj,
            )
        return self.none()

```

model manager has been created . Now create our custom user model.

Models.py:

```
class User(AbstractBaseUser, PermissionsMixin):
    full_name=models.CharField(_('full_name'),max_length=25,blank=False,null=False)
    email = models.EmailField(_('email'), blank=False,unique=True)
    address=models.CharField(_('address'),max_length=45,blank=False,null=False)

    is_staff = models.BooleanField(
        _('staff status'),
        default=False,
        help_text=_('Designates whether the user can log into this admin site.'),
    )
    is_active = models.BooleanField(
        _('active'),
        default=True,
        help_text=_(
            'Designates whether this user should be treated as active. '
            'Unselect this instead of deleting accounts.'
        ),
    )
    date_joined = models.DateTimeField(_('date joined'), default=timezone.now)

    objects = UserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['full_name','address']

    class Meta:
        verbose_name = _('users')
        verbose_name_plural = _('users')
        swappable = 'AUTH_USER_MODEL'

    def get_full_name(self):
        """
        Return the first_name plus the last_name, with a space in between.
        """
        full_name = '%s %s' % (self.first_name,self.last_name)
        return first_name+"last_name

    def get_short_name(self):
        """Return the short name for the user."""
        return self.first_name

    def email_user(self, subject, message, from_email=None, **kwargs):
        """Send an email to this user."""
        send_mail(subject, message, from_email, [self.email], **kwargs)
```

In the settings.py add this line to define our user model:

```
AUTH_USER_MODEL='accounts.User' .
```

we are all set with custom user model. Now create the migration by typing in the terminal :

```
(env) shakil@shakil:~/Desktop/project/e-wallet/wallet_app$ python manage.py makemigrations
Migrations for 'accounts':
  accounts/migrations/0001_initial.py
  - Create model User
```

And migrate it :

```
(env) shakil@shakil:~/Desktop/project/e-wallet/wallet_app$ python manage.py migrate
```

Now our custom user model has been created (:--

Creating user creation form:

create a file with name 'forms.py' at the accounts app directory.

In this forms.py file we will create user creation form to take input from registration page . I used widget property to give it a custom design.

Now our forms.py looks like this:

```
from django import forms

class UserForm(forms.Form):

    full_name=forms.CharField(max_length=20,required=True,
                               widget=forms.TextInput(attrs={'class':'form-control','placeholder':'full name'}))
    email=forms.EmailField(max_length=50,required=True,
                            widget=forms.TextInput(attrs={'class':'form-control','placeholder':'email'}))
    address=forms.CharField(max_length=60,required=True,
                             widget=forms.TextInput(attrs={'class':'form-control','placeholder':'address'}))
    password1=forms.CharField(max_length=20,required=True,
                               widget=forms.PasswordInput(attrs={'class':'form-control','placeholder':'Password'}))
    password2=forms.CharField(max_length=20,required=True,
                               widget=forms.PasswordInput(attrs={'class':'form-control','placeholder':'Confirm password'}))
```

Import this form to views.py file located in the ‘accounts’ directory(not in the wallet_app directory)

note:you can use the ‘wallet_app’ views.py file to do this. Although it is not recommended .

accounts/views.py :

```
from django.shortcuts import render
from accounts.forms import UserForm
def index_page(request):
    return render(request,'index.html')

def registration_page(request):
    if request.method=='GET':
        UserCreationForm=UserForm
        return render(request,'registration.html',{'form':UserCreationForm})
```

At first we need to handle the get request that’s why I checked if the request is get or post. If it is a get request I want to pass the form in the context.

Now we need to render the form fields properly at the registration.html file.

At first remove all the input field and add the 'form.name' to display desired form field.

Here is the registration.html:

```
<form method="post" action="">
<div class="form-group">
<label for="fullname">Full Name :</label>
{{ form.full_name }}
</div>
<div class="form-group">
<label for="email">Email :</label>
{{ form.email }}
</div>
<div class="form-group">
<label for="Address">Address :</label>
{{ form.address }}

</div>
<div class="form-group">
<label for="Password1">Password :</label>
{{ form.password1 }}
</div>
<div class="form-group">
    <label for="Password2">Retype Password :</label>
    {{ form.password2 }}
</div>
<p class="text-success mb-2 mt-2 ">Already have an account?
<a class="text-success font-weight-bold " href="{% url 'login_page' %}">login</a></p>

<button type="submit" class="btn btn-md btn-info text-white">Register</button><br>
</form>
```

Open terminal and run server again. Go to registration page and you will see everything remains same, that means everything is working fine so far .

Integration of google Re-captcha :

Go to google recaptcha and fill out all the form .After completion google will give you a site key and a secret key. Just copy the keys and save it to a text

document. We will need this later.

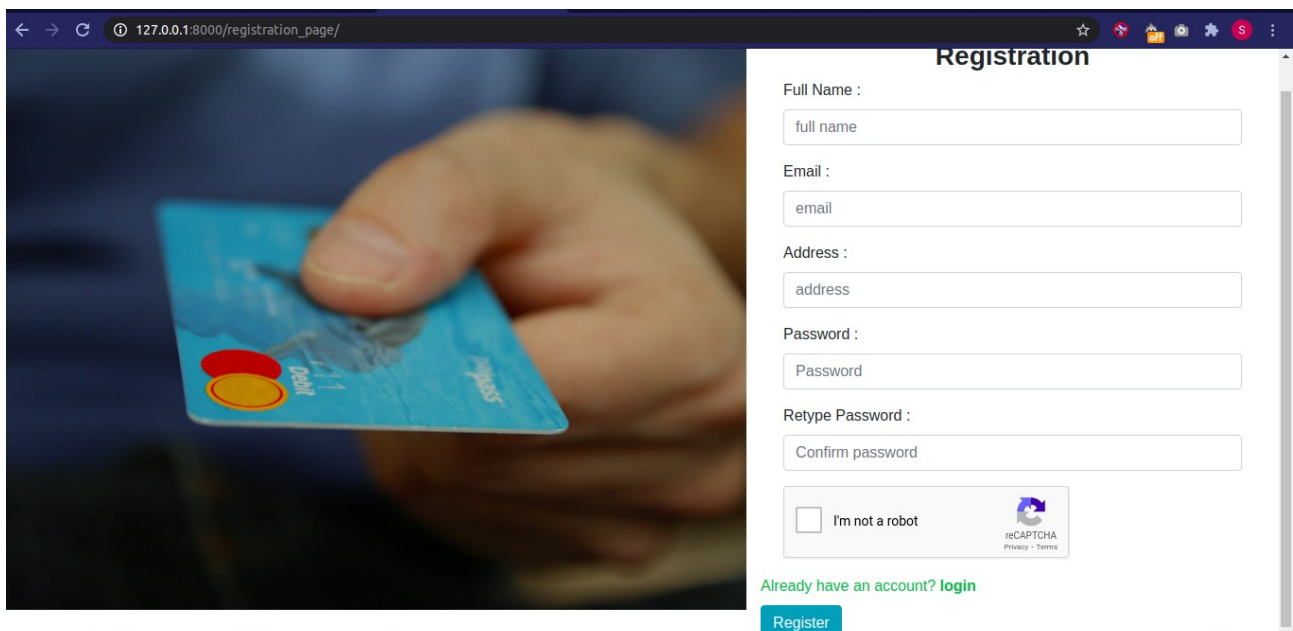
Now to add the recaptcha challenge in our registration form paste the script tag at the head section of the register page>

```
<script src="https://www.google.com/recaptcha/api.js" async defer></script>
```

add this code at the bottom of our form :

```
<div class="form-group">  
  <div class="g-recaptcha" data-sitekey="your_site_key"></div>  
</div>
```

Now reload the registration page and you will see the result like this:



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/registration_page/". The page content is titled "Registration" and contains a form with the following fields:

- Full Name :
- Email :
- Address :
- Password :
- Retype Password :

Below the password fields, there is a checkbox labeled "I'm not a robot" and a reCAPTCHA logo. At the bottom of the form, there is a link "Already have an account? login" and a blue "Register" button.

But that's not the end. We need to process and validate all the user data and recaptcha. To do this let's go to views.py again and handle the post request .

accounts/views.py:

```
def registration_page(request):
    if request.method=='GET':
        UserCreationForm=UserForm
        return render(request,'registration.html',{'form':UserCreationForm})

    if request.method=='POST':
        UserCreationForm=UserForm(request.POST)
        if UserCreationForm.is_valid():
            context={}
            full_name=UserCreationForm.cleaned_data['full_name']
            email=UserCreationForm.cleaned_data['email']
            address=UserCreationForm.cleaned_data['address']
            password1=UserCreationForm.cleaned_data['password1']
            password2=UserCreationForm.cleaned_data['password2']
            Captcha_response=request.POST['g-recaptcha-response']
            secret_key='6Lco6RwaAAAAAPplcNfJsiKRBXvHXQzFRyPzXesO'
            sending_request=requests.post('https://www.google.com/recaptcha/api/siteverify',
                data={'response':Captcha_response,'secret':secret_key})
            status=json.loads(sending_request.text)['success']

            if password1 != password2 :
                context['passerror']='password did not match'
                context['form']=UserForm
                return render(request,'registration.html',context)

            elif status == False :
                context['form']=UserForm
                context['captcha_error']='captcha validation failed'
                return render(request,'registration.html',context)

            else:
                try:
                    User.objects.get(email=email)

                    return render(request,'registration.html',
                        {'form':UserForm,'userexist':'user already exist'})
                except User.DoesNotExist:
                    create_user=User.objects.create_user(password=password1,
                        full_name=full_name,email=email,
                        address=address)
                    return render(request,'registration.html',
                        {'form':UserForm,'success':'account created successfully
                        . You can login now'})
```

You need to import json and requests module at first. After that, we checked if the request is post or not. If the request is post then check if the form is valid or not. You can check it with (form.is_valid()) method where the form is your context form filled with post data. We extract all the data by a method name (form.cleaned_data['input_name']) and stored it into variables.

After we checked the password whether it has matched or not. If not then we can send the user back to the registration page with some context data.

Next we checked the captcha value which we got from the registration page and store the value to a variable name Captcha_response. After we sent the request to the google re-captcha api with context data such as 'secret:secret_key' and 'response:Captcha_response'. In the response of the request we will get a json object with a key name 'success'. If the request is successful then it will return True otherwise False.

Finally we checked if the user exists or not. If it exists then it will resend the user to the registration page with context message.

If the user does not exist then it will create new user and send the user back to the registration page with context message that user registration was successful.

Let's see the registration page template to figure out how it handles the context data :

registration.html:

```
<h3 class="text-center mt-4 font-weight-bold">Registration</h3>
    <div class="alert-area mb-2 mt-2">
        {% if passerror %}
        <div class="alert alert-danger">{{passerror}}</div>
        {% elif captcha_error %}
        <div class="alert alert-danger">{{captcha_error}}</div>

        {% elif userexist %}
        <div class="alert alert-danger">{{userexist}}</div>

        {% elif success %}
        <div class="alert alert-success">{{success}}</div>
        {% else %}
```

Here conditional django template tag has been used. The conditional tag is

```
{% if something %}
```

```
{% elif something %}
```

```
{% else %}
```

pretty simple ,right?

We are all done with the user registration part. Let's dive into login and user authentication part.

User login and authentication :

In the 'login.html' file give the form method to post and action to the current url and set the name attribute for each input tag.

login.html :

```
<div class="form-area ml-4 mr-4">
  <form method="post" action="">

    <div class="form-group">
      <label for="email">Email :</label>
      <input id="email" type="email" class="form-control"
        placeholder="yourname@email.com" name="email">
    </div>

    <div class="form-group">
      <label for="Password1">Password :</label>
      <input id="Password1" type="password" class="form-control"
        placeholder="password" name="password">
    </div>
    <p class="text-success mb-2 mt-2 ">Don't have an account?
    <a class="text-success font-weight-bold " href="{% url 'registration_page' %}"
">Register</a></p>

    <button type="submit" class="btn btn-md btn-success text-white">Login</button>

  </form>
</div>
```

```
in the views.py import some classes
from django.http import HttpResponse
from django.contrib import auth
```

and here is our modified accounts/views.py login view codes:

```
def login_page(request):

    if request.method == 'GET':
        return render(request,'login.html')

    if request.method == 'POST':
        email = request.POST['email']
        password = request.POST['password']

        authenticate=auth.authenticate(request,email=email,password=password)
        if authenticate is not None :
            auth.login(request,authenticate)
            return HttpResponse('<h1>user is authenticated</>')

        else:
            return render(request,'login.html',{'message':'username or password is incorrect'})
```

At first it checks if it is a get or post request. If it is a get request it will render 'login.html' page and if it is a post then it will extract the email and password and store it in two variables name email and password .

Then it will try to authenticate the user by (auth.authenticate) method. If user exists with his/her email and password then it will return the user object otherwise it will return None .

After that, if it is not none then will ensure that user can be logged in .

Auth.login() function receives two arguments , one is request and another is authenticated object . In this case our variable is 'authenticate' .

For now it returns a http response with message 'user is authenticated' but later when we will be working on dashboard it will be modified to redirect user to

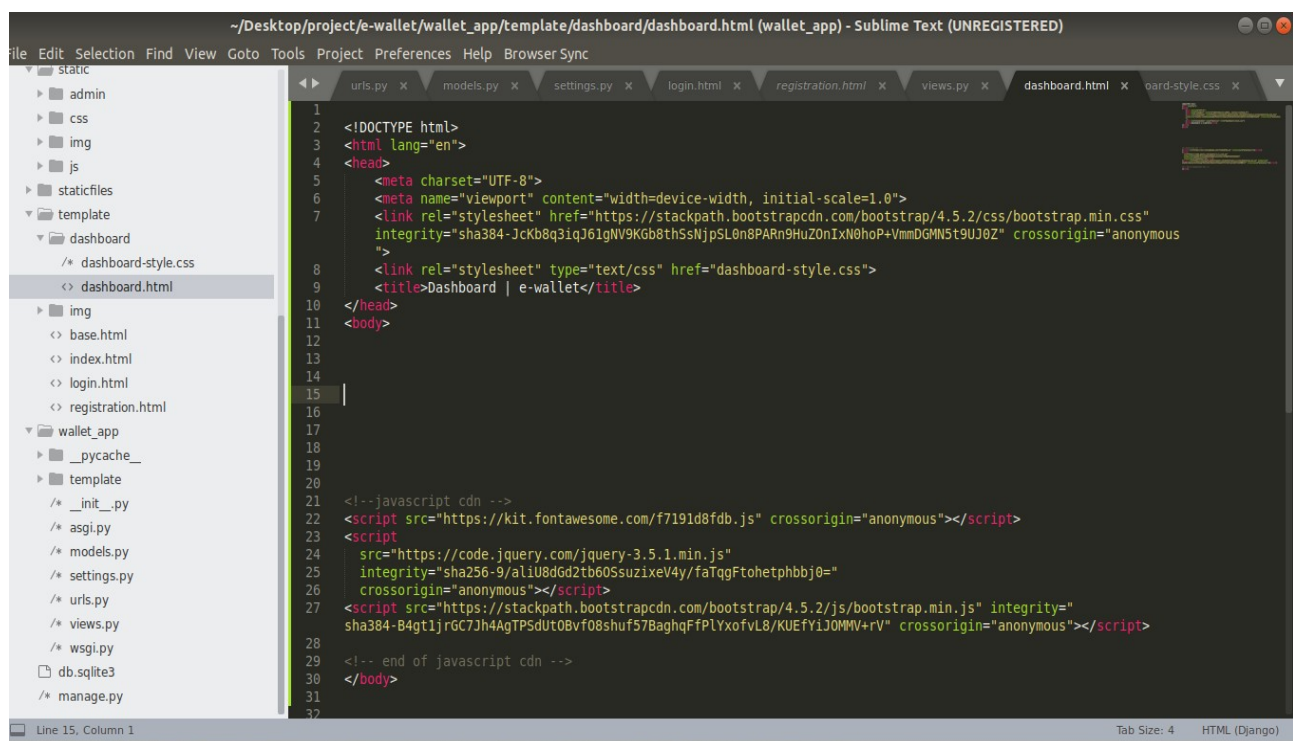
the dashboard. If the user is not authentic by the `auth.authenticate()` method , it will render the login page again with a context message that ‘username or password is incorrect ! ‘ . We are done with user authentication and login part.

Let’s move on to the dashboard .

Building the dashboard ui :

create a new app ‘dashboard’ by typing in the terminal ‘python manage.py startapp dashboard’ and add this app to the `INSTALLED_APP` list in `settings.py`. Now create a directory with name ‘dashboard’ in the template directory to separate our dashboard template files. Inside this directory create two files one is ‘dashboard.html’ and another is ‘dashboard-style.css’ .

Here is our initial setup for dashboard.html :



```
~/Desktop/project/e-wallet/wallet_app/template/dashboard/dashboard.html (wallet_app) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help Browser Sync
Static
├── admin
├── css
├── img
├── js
├── staticfiles
└── template
    └── dashboard
        ├── dashboard-style.css
        └── dashboard.html
    ├── img
    │   ├── base.html
    │   ├── index.html
    │   ├── login.html
    │   └── registration.html
    └── wallet_app
        ├── __pycache__
        ├── template
        ├── __init__.py
        ├── asgi.py
        ├── models.py
        ├── settings.py
        ├── urls.py
        ├── views.py
        ├── wsgi.py
        ├── db.sqlite3
        └── manage.py

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
8          integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpsL8n8PARn9HuZ0n1XN9hoP+VmmDGMN5t9UJ0Z" crossorigin="anonymous"
9      >
10     <link rel="stylesheet" type="text/css" href="dashboard-style.css">
11     <title>Dashboard | e-wallet</title>
12 </head>
13 <body>
14
15
16
17
18
19
20
21 <!-- javascript cdn -->
22 <script src="https://kit.fontawesome.com/f7191d8fdb.js" crossorigin="anonymous"></script>
23 <script
24     src="https://code.jquery.com/jquery-3.5.1.min.js"
25     integrity="sha256-9/aliU8dGd2tb60SsuzixeV4y/faTqgFtohetphbbj0="
26     crossorigin="anonymous"></script>
27 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" integrity="
28 sha384-B4gt1jrGC7Jh4AgTPSdU0BvF08shuf57BaghqFfPLYxofvL8/KUEfYi3OMMM+rV" crossorigin="anonymous"></script>
29 <!-- end of javascript cdn -->
30 </body>
31
32
```

let’s create dashboard sidebar first. I will use bootstrap navbar with flex-direction to column for creating sidebar. Paste the codes below inside `<body>` tag .

Dashboard.html:

```
<body>

<div class="container-fluid m-0 pl-0">
  <div class="row ml-0">
    <div class="col-md-2 m-0 p-0">

      <nav class="navbar navbar-expand-md navbar-dark bg-success mt-0 pt-0 ml-0">
        <div class="sidebarcol">

          <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#nav
          <span class="navbar-toggler-icon"></span>
        </button>
        <h3 class="logo text-center text-white mt-4">E-WALLET</h3>
        <div class="collapse navbar-collapse mt-4" id="navbarTogglerDemo01">

          <ul class="navbar-nav flex-column">

            <li class="nav-item">
              <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">Add fund</a>
            </li>

            <li class="nav-item">
              <a class="nav-link" href="#">Send money </a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#"> Request money </a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#"> Take donations </a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">Pending requests
              <span class="badge badge-pill badge-secondary">2</span> </a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="#">Logout </a>
            </li>

          </ul>

        </div>
      </nav>
    </div>
  </div>
</div>
```


dashboard-style.css:

```
body{
  margin:0;
  box-sizing: border-box;
}

.sidebarcol{
  height: 100vh !important;
  width: 100%;
}

nav{
  margin:0px !important;
  padding:0px !important;
}
nav ul{
  width: 100% !important;
  margin-top:2rem !important;
}
nav ul li{
  margin-top:0rem;
  border-top:1px solid white;
  padding-top:0.5rem;
  padding-bottom: 0.5rem;
}
nav ul li:last-child{
  border-bottom: 1px solid white;
}
nav ul li a{
  margin-left: 2rem !important;
  color:white !important;
}
nav ul li:hover{
  background-color: black;
}
```

Now our dashboard looks like this :



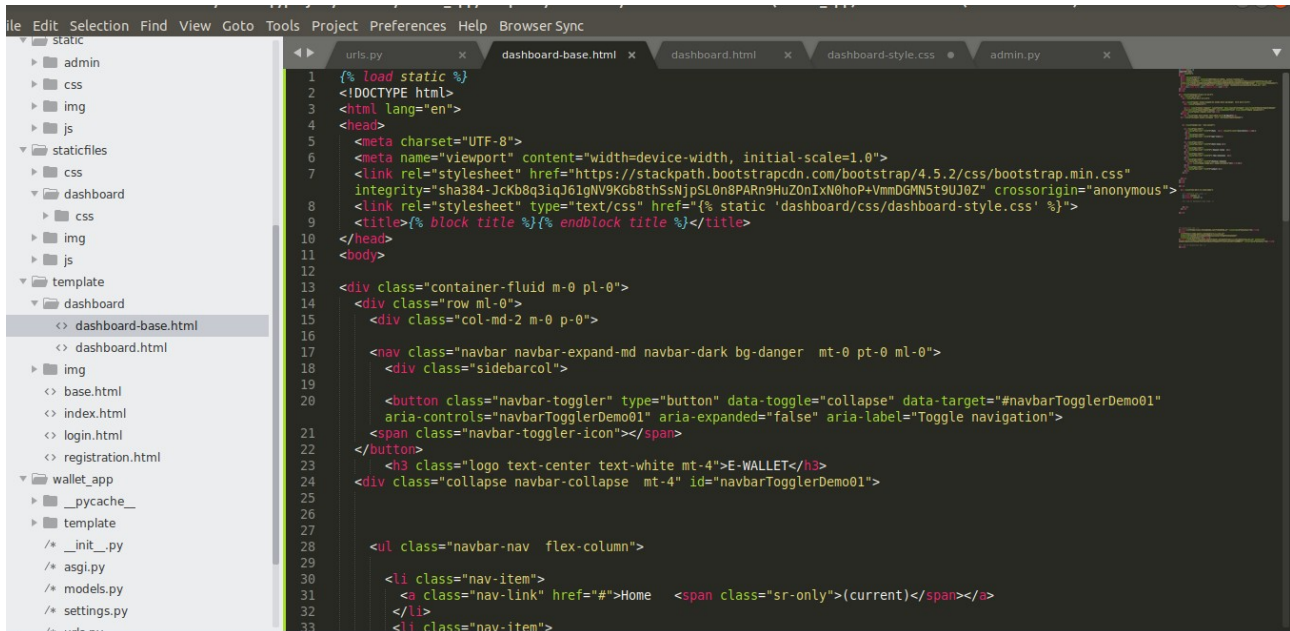
Creating base file for dashboard :

Every page for dashboard will contain the same sidebar. So, copy & pasting this sidebar to all the pages is not a good idea . To cope with this problem we need to create a dashboard-base.html file which will contain only the sidebar and the blank dashboard showcase area .

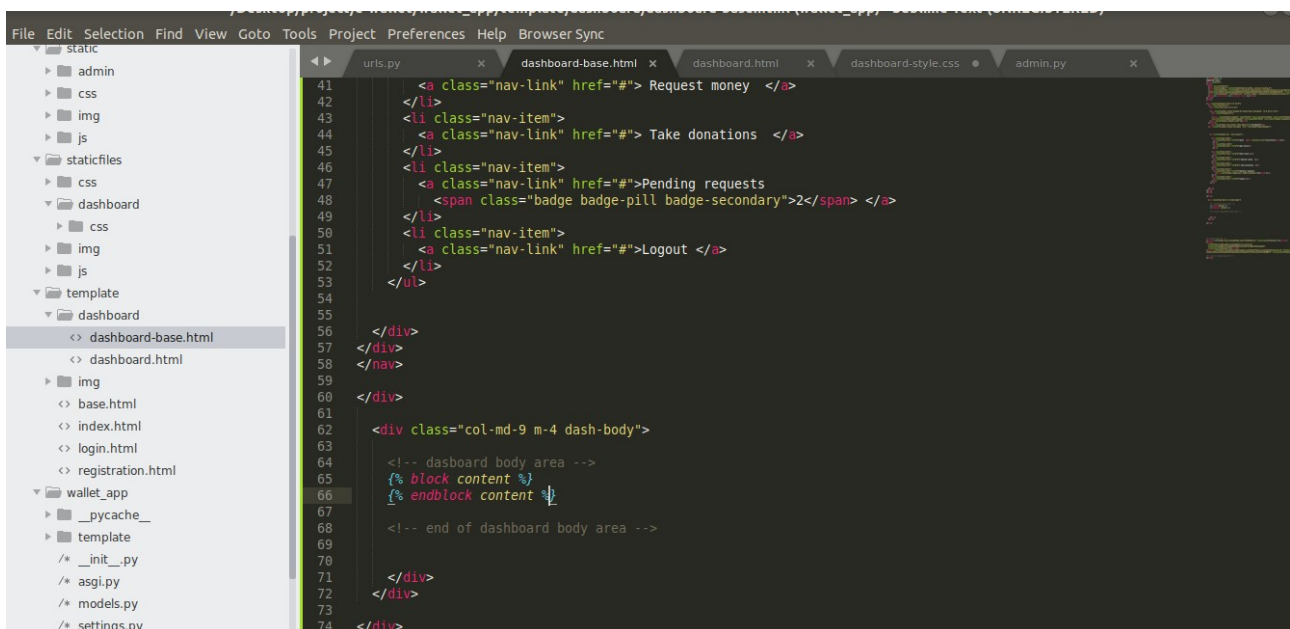
At first create a file with name 'dashboard-base.html' in the 'template/dashboard' folder .Copy base codes from 'dashboard.html' file and paste it into 'dashboard-base.html' file . Now use template tag {% load static %} and {% static 'path' %} (discussed earlier) to load static files .

Next, create a folder in the 'staticfiles' directory with name 'dashboard' and inside it create another folder with name 'css'. Now, cut the 'dashboard-style.css' file from the 'template/dashboard' directory and paste it into 'staticfiles/dashboard/css' directory .

Dashboard-base.html :



```
1 {% load static %}
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5 <meta charset="UTF-8">
6 <meta name="viewport" content="width=device-width, initial-scale=1.0">
7 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
8 integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ8Z" crossorigin="anonymous">
9 <link rel="stylesheet" type="text/css" href="{% static 'dashboard/css/dashboard-style.css' %}">
10 </head>
11 <body>
12
13 <div class="container-fluid m-0 pl-0">
14 <div class="row ml-0">
15 <div class="col-md-2 m-0 p-0">
16
17 <nav class="navbar navbar-expand-md navbar-dark bg-danger mt-0 pt-0 ml-0">
18 <div class="sidebarcol">
19
20 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarToggleDemo01"
21 aria-controls="navbarToggleDemo01" aria-expanded="false" aria-label="Toggle navigation">
22 <span class="navbar-toggler-icon"></span>
23 </button>
24 <h3 class="logo text-center text-white mt-4">E-WALLET</h3>
25 <div class="collapse navbar-collapse mt-4" id="navbarToggleDemo01">
26
27 <ul class="navbar-nav flex-column">
28
29 <li class="nav-item">
30 <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
31 </li>
32 <li class="nav-item">
```



```
41 <a class="nav-link" href="#"> Request money </a>
42 </li>
43 <li class="nav-item">
44 <a class="nav-link" href="#"> Take donations </a>
45 </li>
46 <li class="nav-item">
47 <a class="nav-link" href="#"> Pending requests
48 <span class="badge badge-pill badge-secondary">2</span> </a>
49 </li>
50 <li class="nav-item">
51 <a class="nav-link" href="#"> Logout </a>
52 </li>
53 </ul>
54
55 </div>
56 </div>
57 </div>
58 </nav>
59
60 </div>
61
62 <div class="col-md-9 m-4 dash-body">
63
64 <!-- dashboard body area -->
65 {% block content %}
66
67
68 <!-- end of dashboard body area -->
69
70 </div>
71 </div>
72
73
74 </div>
```

Now base file has been created. To create the dashboard index we need to just extend this base file and that's it. To do this, go to dashboard.html in the 'template/dashboard' directory and extend it like this:

```

{% extends 'dashboard/dashboard-base.html' %}
{% block title %} Dashboard showcase {% endblock title %}

{% block content %}
    <!-- content area -->
    <div class="row justify-content-around mt-4">

        <div class="card bg-warning" style="width:12rem">
            <div class="card-header text-white text-center">
                Balance
            </div>
            <div class="card-body text-center font-weight-bold text-success">
                1024.00 <span>$</span>
            </div>
        </div>
        <div class="card bg-warning" style="width:15rem">
            <div class="card-header text-white text-center">
                money requests(pending)
            </div>
            <div class="card-body text-center font-weight-bold text-success">
                2
            </div>
        </div>
        <div class="card bg-warning" style="width:18rem">
            <div class="card-header text-white text-center">
                your requested money(pending)
            </div>
            <div class="card-body text-center font-weight-bold text-success">
                10
            </div>
        </div>

    </div>

<!-- end of content area -->
{% endblock content %}

```

To copy all the static files run the command in the terminal ‘python manage.py collectstatic’ .

Url routing for dashboard :

In the 'dashboard' directory of the dashboard app create a new file with name 'urls.py' and paste the codes below:

```
from django.urls import path
from . import views

urlpatterns = [

    path("",views.dashboard_index,name='dashboard_index'),

]
```

Now we need to notify our main 'urls.py' that we want to route traffic from a specific url prefix. To do this, go to project root folder 'wallet_app' and open the 'urls.py' .

wallet_app/urls.py :

```
from django.contrib import admin
from django.urls import path,include
from . import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path("",views.index_page,name='index_page'),
    path('accounts/',include('accounts.urls')),
    path('dashboard/',include('dashboard.urls')),
]
```

We included the urls from 'dashboard/urls.py'. When user looks for a url 'oursite.com/dashboard' it will automatically include and route the traffic to the dashboard urls.py file.

Url routing is complete for dashboard index page.

Go to dashboard/views.py file and type this to render dashboard index page.

```
from django.shortcuts import render
from django.contrib.auth.decorators import login_required

@login_required(login_url='/accounts/login/')
def dashboard_index(request):
    return render(request,'dashboard/dashboard.html')
```

But we will only allow authenticated users to visit dashboard . To do that import login_required decorator and use it on the function with login_url parameter to redirect user to this path if the user is not logged in .

Till now after successful user login it shows only a http response without redirecting to dashboard.To fix this go to views.py in ‘accounts’ directory and modify login_page function to redirect users to the dashboard after authentication.

At first import ‘redirect’ by typing above the views file

from django.shortcuts import redirect and modify the old codes like this :

```
def login_page(request):

    if request.method == 'GET':
        return render(request,'login.html')

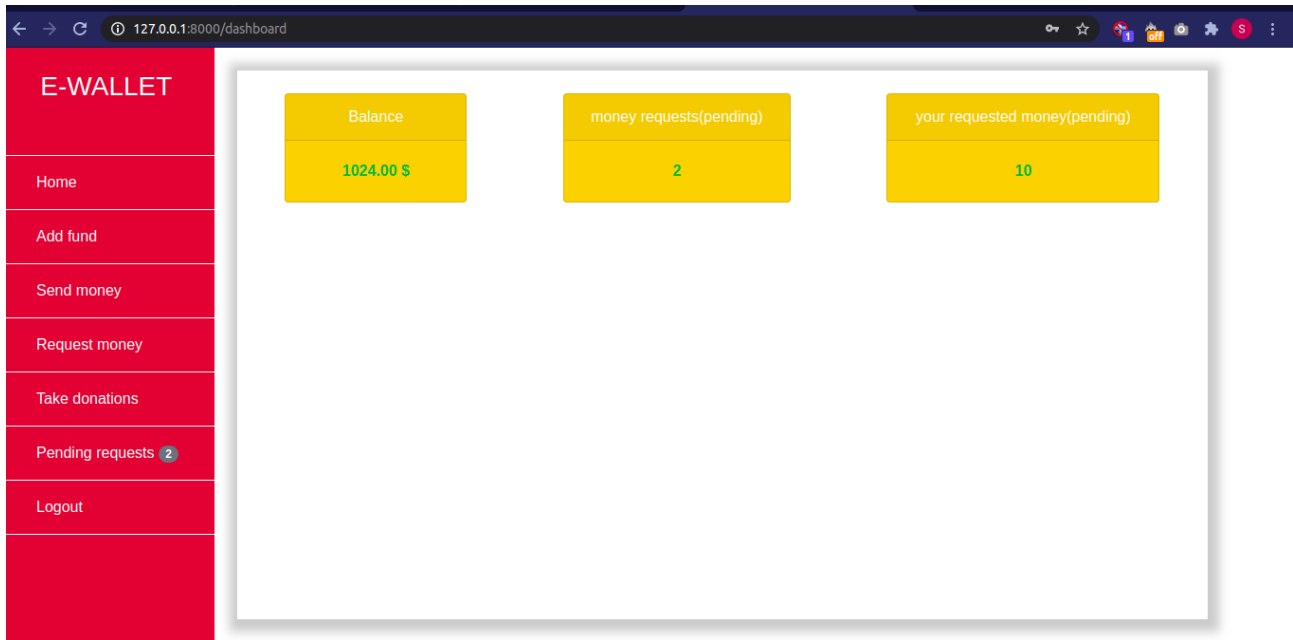
    if request.method == 'POST':
        email = request.POST['email']
        password = request.POST['password']

        authenticate=auth.authenticate(request,email=email,password=password)
        print(authenticate)
        if authenticate is not None :
            auth.login(request,authenticate)

            return redirect('dashboard_index') #new code, instead of just HttpResponseRedirect

    else:
        return render(request,'login.html',{'message':'username or password is incorrect !'})
```

Look at this , our url routing and authentication is working for dashboard.



But hey! How can I log out?

Logout button is not working , we need to fix this issue .

In the accounts directory open `views.py` and create new function with name `user_logout` and paste this code inside it.

```
def user_logout(request):  
    auth.logout(request)  
    return redirect('index_page')
```

And in the `urls.py` :

```
urlpatterns = [  
  
    path('registration/', views.registration_page, name='registration_page'),  
    path('login/', views.login_page, name='login_page'),  
    path('logout/', views.user_logout, name='user_logout')  
]
```


Now add this ‘user_logout’ link to the base template .

Dashboard-base.html:

```
<li class="nav-item">
  <a class="nav-link" href="{% url 'user_logout' %}">Logout </a>
</li>
```

Now logout button is working fine, if you click on the logout button you will be logged out and redirected to the index page .

Developing add fund functionality:

Create a new file with name ‘add-fund.html’ in the ‘template/dashboard’ folder and extend ‘dashboard-base.html’ file and inside content block paste the code below to create a basic add fund form.

Add-fund.html:

```
<div class="row">
  <div class="col-md-5 .fund-form ml-auto mr-auto mt-4">
    <h3 class="font-weight-bold mb-4 mt-auto">Add fund to your account </h3>
    <div class="form form-input">
      <form id="payment-form" class="m-4" action="#" method="post">
        {% csrf_token %}
        <div class="form-group">
          <input type="number" placeholder="amount" name="amount" class="form-control">
        </div>

        <div class="one-liner">
          <div class="card-frame">
            </div>
            <input id="token" type="hidden" name="token">
          </div>
          <input id="pay-button" type="button" value="Add fund" class="btn btn-primary">
        </form>
      </div>
    </div>
  </div>
```

You may ask where from I got this .one-liner and .card-frame css classes?

Well, It is from <https://checkout.com> frame documentation .check it out here <https://docs.checkout.com/quickstart/integrate/frames> .

The processing flow of payment integration is > generate client side token > Charge card with this token on server.

I just have generated client side form with their javascript frame . Then included a hidden input field to store the generated token in this as a value . After that I will this this genetated token and amount at the backend to charge the card which you will see later . Below the form paste this javascript which I got from their documentation .

Add-fund.html:

```
<script src="https://cdn.checkout.com/js/framesv2.min.js"></script>
<script type="text/javascript">

    var payButton = document.getElementById("pay-button");
    var form = document.getElementById("payment-form");

    Frames.init("pk_test_b8c1baa3-0ba8-4726-9d8c-672f0b621b6b");

    Frames.addEventHandler(
        Frames.Events.CARD_VALIDATION_CHANGED,
        function (event) {

            payButton.disabled = !Frames.isCardValid();
        }
    );

    Frames.addEventHandler(
        Frames.Events.CARD_TOKENIZED,
        function (event) {
            var el = document.getElementById("token");
            el.setAttribute("value",event.token);
            form.submit();

        }
    );
};
```

```

payButton.addEventListener("click", function (event) {
    event.preventDefault();
    payButton.value="proccesing.."
    Frames.submitCard();
    payButton.value="wait...."

});

</script>

{% endblock content %}

```

we are done with form creation and token generation . But we have not customized the urls.py and sidebar to view this page. To do this, go to urls.py and add a new line as below :

dashboard/urls.py :

```

from django.urls import path
from . import views

urlpatterns = [

    path("",views.dashboard_index,name='dashboard_index'),
    path('add_fund/',views.add_fund,name='add_fund'), #new line

]

```

dashboard/views.py :

add a new function with name 'add_fund.html' .

```

def add_fund(request):
    if request.method == 'GET':
        return render(request,'dashboard/add-fund.html')

```

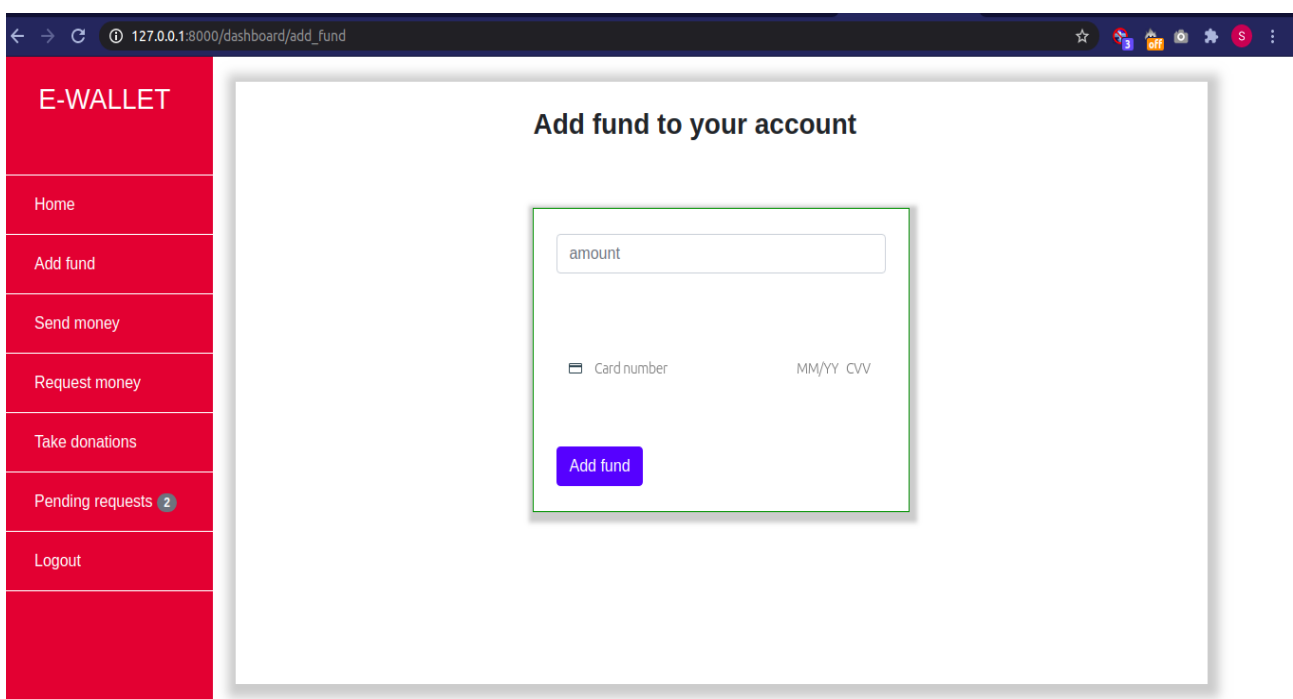
Add this link to the ‘dashboard-base.html’ :

```
<li class="nav-item">
  <a class="nav-link" href="{% url 'add_fund' %}">Add fund</a>
</li>
```

staticfiles/dashboard/css/dashboard-style.css:

```
.fund-form{
    height: 100vh ;
}
.form-input{
    margin-top:4rem;
    border:1px solid green;
    box-shadow: 2px 4px 4px 6px rgba(0,0,0,0.2);
}
```

Now ‘add-fund’ page looks like this :



Open models.py in 'dashboard/models.py' and create a new model with name 'UserBalance' where all the users balance data will be stored.

dashboard/models.py:

```
from django.db import models
from accounts.models import User
class UserBalance(models.Model):
    user = models.ForeignKey(User,on_delete=models.CASCADE)
    balance=models.FloatField(blank=False,null=False)
```

make the migration by typing in the terminal:

```
(env) shakil@shakil:~/Desktop/project/e-wallet/wallet_app$ python manage.py makemigrations dashboard
Migrations for 'dashboard':
  dashboard/migrations/0001_initial.py
  - Create model UserBalance
```

and migrate it by typing in the terminal:

```
(env) shakil@shakil:~/Desktop/project/e-wallet/wallet_app$ python manage.py migrate dashboard
Operations to perform:
  Apply all migrations: dashboard
Running migrations:
  Applying dashboard.0001_initial... OK
```

Database table has been created for storing user balance.

Now install checkout-sdk by typing in the terminal :

```
(env) shakil@shakil:~/Desktop/project/e-wallet/wallet_app$ pip install checkout-sdk==2.0b8
Collecting checkout-sdk==2.0b8
```

Import the checkout sdk to 'dashboard/views.py' by typing:

```
'import checkout_sdk as sdk
```

dashboard/views.py:

```
@login_required(login_url='/accounts/login/')
def add_fund(request):
    if request.method == 'GET':
        return render(request,'dashboard/add-fund.html')

    if request.method == 'POST':
        amount=int(request.POST['amount'])
        token=request.POST['token']
        print('amount',amount,'token',token)
        if amount == " " or token == " ":
            return render(request,'dashboard/add-fund.html',
                {'BlankError':'All fields are required'})
        else:
            api = sdk.get_api(secret_key=
                'sk_test_f2a38861-4256-413b-a95c-4c3822b8f508')

            try:
                payment = api.payments.request(
                    source={
                        'token': token,
                    },
                    amount=amount*100,
                    currency=sdk.Currency.USD,
                )
```

continued:

```
if payment.approved == True and payment.status == "Authorized":
    try :
        user_bal = UserBalance.objects.get(user=request.user)
        curr_bal = user_bal.balance
        new_bal=float(curr_bal + amount )
        user_bal.balance = new_bal
        user_bal.save()
        return render(request,'dashboard/add-fund.html',
                        {'credited':'Fund added successfully !'})
    except UserBalance.DoesNotExist:
        UserBalance.objects.create(user=request.user,balance=amount)
        return render(request,'dashboard/add-fund.html',
                        {'credited':'Fund added successfully !'})

    else:
        return render(request,'dashboard/add-fund.html',
                        {'NoPayment':'your card can not be authorized !'})
except sdk.errors.CheckoutSdkError as e:
    return render(request,'dashboard/add-fund.html',
                  {'PaymentError':'Payment Failed ! Try again later'})
```

Dont worry, I know it is too much codes but I will discuss everything piece by peiece.

First of all it checks if the request is 'GET' or 'POST' . If it is a POST request , it will gather all the information passed from 'add-fund.html' file. It stores the value of the post parameters by this code

```
amount=int(request.POST['amount'])
```

```
token=request.POST['token']
```

In the amount variable it will store user entered amount and in the token variable the token generated by checkout frame will be stored.

Next it will validate whether the values are blank or not. If one of the values is blank it will re render the add-fund page with a context message that every field is required.

If two values are not blank it will go to the next step. Earlier we imported the checkout SDK and now it is the time to use it.

It will initiate the api :

```
api = sdk.get_api(secret_key='sk_test_f2a38861-4256-413b-a95c-4c3822b8f508')
```

You may ask me where I got this secret key? Well, login to your checkout.com sandbox account and you will find yours.

Next, It will try to process the payment by this code :

```
try:
    payment = api.payments.request(
        source={
            'token': token,
        },
        amount=amount*100,
        currency=sdk.Currency.USD,
    )
```

If this is successful , all our payment details will be available at the payment variable as response.

Typical response:

```
{
  "id": "pay_mbabizu24mvu3mela5njyhpit4",
  "action_id": "act_mbabizu24mvu3mela5njyhpit4",
  "amount": 6540,
  "currency": "USD",
  "approved": true,
  "status": "Authorized",
  "auth_code": "770687",
  "response_code": "10000",
  "response_summary": "Approved",
  "3ds": {
    "downgraded": true,
    "enrolled": "N"
  },
}
```


Now you can access all the status of the payment to ensure the payment is made.

Next, it will confirm the payment using this code:

```
if payment.approved == True and payment.status == "Authorized":
    try :
        user_bal = UserBalance.objects.get(user=request.user)
        curr_bal = user_bal.balance
        new_bal=float(curr_bal + amount )
        user_bal.balance = new_bal
        user_bal.save()
        return render(request,'dashboard/add-fund.html',
                        {'credited':'Fund added successfully !'})
    except UserBalance.DoesNotExist:
        UserBalance.objects.create(user=request.user,balance=amount)
        return render(request,'dashboard/add-fund.html',
                        {'credited':'Fund added successfully !'})
```

As I said earlier that you can access the payment data like a normal python dictionary . It will check the 'approved' key and 'status' key for confirmation. If approved key has a value 'True' that means the payment has been approved and if the status is 'Authorized' that means the payment is also successfully authorized .

Next, We confirmed the payment now we need to update the balance of the user.

So, first it will try to select the row where the user is current logged user , if it finds user does not exist at this balance table, it will go to the except condition and authentically create a row with the data 'user' and 'amount' .

If it finds there is a row with a same user, it will just update the user balance data by adding the previous balance and current submitted amount and finally save the row. After, it will go to 'add-fund' page with a message that 'fund added successfully ' . We will use template conditional tag to catch and show these messages.

If the payment information is incorrect, our 'else' block will be executed and it will take the user to the 'add-fund' page with a message 'your card can not be authorized' .

If the checkout sdk has an error , it will also executed in the ‘except’ block and re render the page with a message. Here is the code for that :

```
else:
    return render(request,'dashboard/add-fund.html',
                  {'NoPayment':'your card can not be authorized !'})
except sdk.errors.CheckoutSdkError as e:
    return render(request,'dashboard/add-fund.html',
                  {'PaymentError':'Payment Failed ! Try again later'})
```

Let’s capture the messages from this view in our template. To do this add the following codes in your ‘add-fund.html’ file:

Note:You can use ‘redirect’ or ‘django message framework’ to pass messages.

```
<h3 class="font-weight-bold mb-4 mt-auto">Add fund to your account </h3>
{% if BlankError %}
<div class="alert alert-danger">{{ BlankError }}</div>
{% endif %}
{% if PaymentError %}
<div class="alert alert-danger">{{ PaymentError }}</div>
{% endif %}
{% if NoPayment %}
<div class="alert alert-danger">{{ NoPayment }}</div>
{% endif %}
{% if credited %}
<div class="alert alert-success">{{ credited }}</div>
{% endif %}
```

It is just some general django {% if ‘condition’ %} {% endif %} and nothing else .

Now you can test the ‘add fund’ feature by entering some credit card informations, But don’t worry you don’t need to enter your own. Here are the list of testable cards from checkouts official page : <https://docs.checkout.com/testing/test-card-numbers>

Our feature has been added!

Showing balance on the dashboard :

Till now users can only add fund to their account, but it doesn't show the available balance anywhere . To show the balance go to 'dashboard/views.py' and edit the 'dashboard_index' function like this :

current 'dashboard_index' function:

```
@login_required(login_url='/accounts/login/')
def dashboard_index(request):
    return render(request,'dashboard/dashboard.html')
```

After 'dashboard_index' function :

```
@login_required(login_url='/accounts/login/')
def dashboard_index(request):
    user_bal = 0.00
    try :
        curr_bal = UserBalance.objects.get(user=request.user)
        user_bal = curr_bal.balance
        return render(request,'dashboard/dashboard.html',{'balance':user_bal})
    except:
        pass

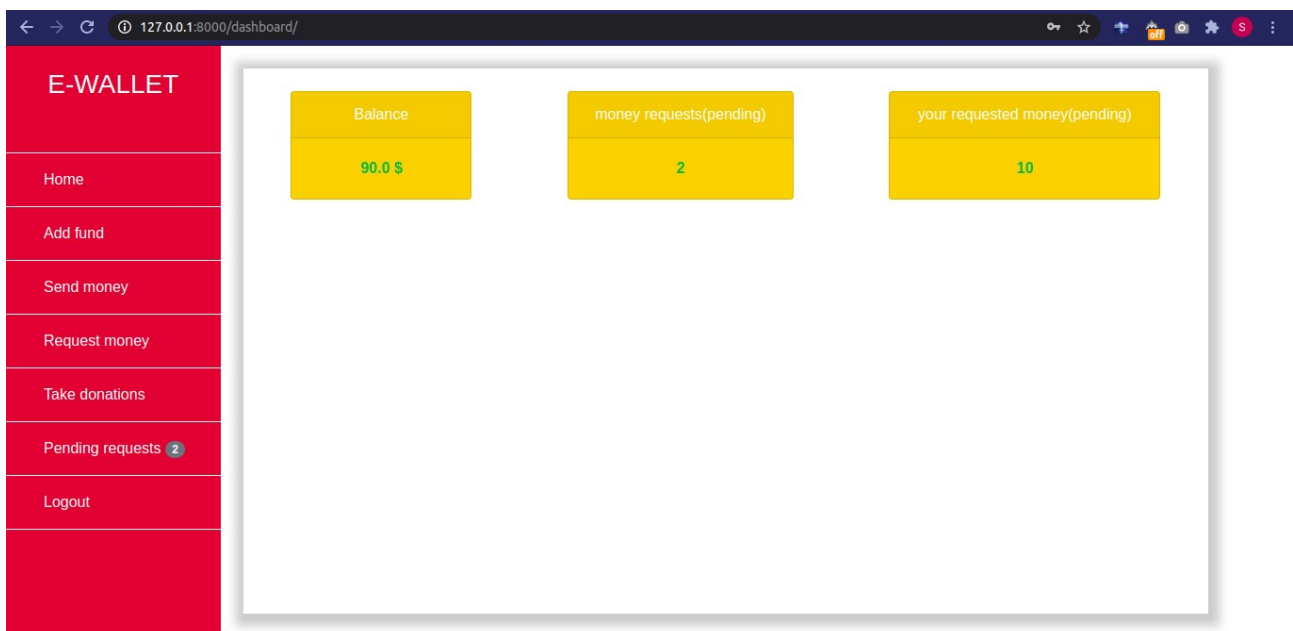
    return render(request,'dashboard/dashboard.html',{'balance':user_bal})
```

At first it will set the initial user_balance to 0.00 assuming that user has not any balance record. Then, it will try to get the balance by `UserBalance.objects.get(user=request.user)` method and set the balance to 'user_bal' variable . If the 'try' fails it means the balance data for the user does not exist and it will then pass the initial 0.00 balance with template context `{'balance':user_bal}` but you need to capture this in the 'dashboard/dashboard.html' file .

dashboard.html :

```
<div class="card bg-warning" style="width:12rem">
  <div class="card-header text-white text-center">
    Balance
  </div>
  <div class="card-body text-center font-weight-bold text-success">
    {{ balance }} <span>$</span>
  </div>
</div>
```

Now login to your account and go to dashboard and you will see your balance.



As we added 50\$ and 40\$ earlier it is showing total balance 90.0\$. Don't worry about 'money request' and 'your requested money' for now, it is just showing hard coded values. We will take care of this later .

Let's work on the 'send money' feature .

Adding send money feature :

Create a new page with name 'send-money.html' in the 'dashboard/' directory .
Extend dashboard-base file and paste this code in the content block .

dashboard/send-money.html :

```
<div class="row">

  <div class="col-md-5 ml-auto mr-auto mt-4">
    <h2 class="text-center">One click send money </h2>
    <div class="form form-input">
      <form action="" method="post" class="m-4">
        <div class="form-group">
          <label for="email">Recipient email </label>
          <input type="email" name="email"
            placeholder="recipient@example.com" class="form-control">

        </div>
        <div class="form-group">
          <label for="amount">Amount </label>
          <input type="number" name="amount"
            placeholder="00.00" class="form-control">

        </div>
        <button type="submit" class="btn btn-info">Send now</button>
      </form>
    </div>
  </div>
</div>
```

Now go to 'dashboard/urls.py' and add this routing:

dashboard/urls.py:

```
path('send_money',views.send_money,name='send_money')
```

And dashboard/views.py to load this file :

```
def send_money(request):  
    if request.method == 'GET':  
        return render(request, 'dashboard/send-money.html')
```

Now it looks like this:

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/dashboard/send_money'. The page features a red sidebar on the left with the title 'E-WALLET' and a list of navigation links: Home, Add fund, Send money, Request money, Take donations, Pending requests (with a blue badge showing '2'), and Logout. The main content area is white and titled 'One click send money'. It contains a form with two input fields: 'Recipient email' (with the placeholder text 'recipient@example.com') and 'Amount' (with the placeholder text '00.00'). Below these fields is a blue button labeled 'Send now'.

In order to make this form works open 'dashboard/views.py and add the post method :

dashboard/views.py :

```
def send_money(request):
    if request.method == 'GET':
        return render(request, 'dashboard/send-money.html')

    if request.method == 'POST':
        email = request.POST['email']
        amount = request.POST['amount']

        if email == "" or amount == "":
            return render(request, 'dashboard/send-money.html',
                          {'BlankError': 'All fields are required'})
```

First of all this send-money function checks whether it is a GET or POST request. If it is a get request then it will render the 'send-money' page but if it is a post request, it will store the email and amount into two variables 'email and amount'. Next, it checks if these are blank or not by (if email== '' or amount == '').

If it is blank then it will render the 'send-money' page with a context message with key 'BlankError' and value 'All fields are required'.

next codes:

```
else :

    try :
        recipient = u.objects.get(email=email)
        recipient_bal_data_check = UserBalance.objects.filter(user=recipient).count
        user_bal_data_check = UserBalance.objects.filter(user=request.user).count
        if recipient_bal_data_check == 0:
            UserBalance.objects.create(user=recipient, balance=0)
            pass
        if user_bal_data_check == 0 :
            UserBalance.objects.create(user=request.user, balance=0)
            return render(request, 'dashboard/send-money.html',
                          {'FundError': 'Not enough funds !'})

        elif int(amount) > UserBalance.objects.get(user=request.user).balance :
            return render(request, 'dashboard/send-money.html',
                          {'FundError': 'Not enough funds !'})

    else:

        recipient_bal = UserBalance.objects.get(user=recipient)
        recipient_bal.balance = (recipient_bal.balance + int(amount))
        user_bal = UserBalance.objects.get(user=request.user)
        user_bal.balance = (user_bal.balance - int(amount))
        recipient_bal.save()
        user_bal.save()
        return render(request, 'dashboard/send-money.html',
                      {'success': 'Send money successful !'})
```

If these fields are not blank then it will move to else block. In the else block it will try to get the recipient by email from the User model . Next it will check whether the balance data for recipient exists or not. If it exists it will return 1 or more otherwise 0. After that, it will again check whether the balance data for current user in the UserBalance table exists or not .The count() method counts the existed data according to the filter query .

Next, it confirms if the recipient balance is 0 or not. If it is 0 that means the balance data of the user doesn't exist so it will create a new UserBalance object with user 'recipient' and 'balance' of 0 because of first time creation .

At the same format it checks the current users ' UserBalance ' data existence and if it doesn't exist then it will create one with user 'current user' and initial balance zero and render the 'send-money' page with context message 'Not enough funds' .

If the object exists then it will move to the next ' elif ' block and this block will check if the amount ,the user is willing to send is greater than the current balance . If it is true then it will render the 'send-money' page with context message 'Not enough funds' .

Now if all the checks are passed, it will move to the 'else' block and deduct the balance from the current user and add it to the recipients balance .

Do you remember that we tried to get the recipient at the first line? Now, what if the recipient does not exist in the database ? Well, it will throw an error ' DoesNotExist ' . Now we need to catch this error by the 'except User.DoesNotExist' and re render the page with context message 'Recipient does not exist' .

Here is the rest of the codes:

```
except User.DoesNotExist :  
    return render(request,'dashboard/send-money.html',  
                  {'recipient_not_found':'Recipient does not exist !'})
```


To show the error messages passed from this view in the template ,go to ‘send-money.html’ and add some lines as follows :

```
<h2 class="text-center">One click send money </h2>
    <div class="form form-input">
        {% if BlankError %}
        <div class="alert alert-danger text-center">{{BlankError}}</div>
        {% endif %}
        {% if FundError %}
        <div class="alert alert-danger text-center">{{FundError}}</div>
        {% endif %}
        {% if success %}
        <div class="alert alert-success text-center">{{success}}</div>
        {% endif %}
        {% if recipient_not_found %}
        <div class="alert alert-danger text-center">{{recipient_not_found}}</div>
        {% endif %}
    <form action="" method="post" class="m-4">
```

Now we are done with adding send money feature !

Developing request money feature :

Create a new model at ‘dashboard/models.py’ with name ‘money_requests’ and paste the code below :

```
class money_requests(models.Model) :
    date = models.DateField(blank=False,null=False,default=datetime.date.today)
    request_from = models.ForeignKey(User,on_delete=models.CASCADE,related_name='request_from')
    request_to = models.ForeignKey(User,on_delete=models.CASCADE,related_name='request_to')
    amount = models.FloatField(blank=False,null=False,default=0)
    is_pending = models.BooleanField(blank=False,default=True)

    def __str__(self):
        return str(self.request_from.full_name + ' to ' +self.request_to.full_name)
```

make migration file and migrate it by typing the following commands in the terminal :

```
(env) shakil@shakil:~/Desktop/project/e-wallet/wallet_app$ python manage.py makemigrations dashboard
(env) shakil@shakil:~/Desktop/project/e-wallet/wallet_app$ python manage.py migrate dashboard
```

Create a page with name 'request-money.html' in the 'template/dashboard/' directory and paste the code below :

dashboard/request-money.html:

```
{% extends 'dashboard/dashboard-base.html' %}

{% block title %}Send money | e-wallet {% endblock title %}

{% block content %}
<div class="row">
  <div class="col-md-5 ml-auto mr-auto mt-4">
    <h2 class="text-center">One click send money </h2>
    <div class="form form-input">

      <form action="" method="post" class="m-4">
        {% csrf_token %}
        <div class="form-group">
          <label for="email">Recipient email : </label>
          <input type="email" name="email"
            placeholder="recipient@example.com" class="form-control"></div>
          <div class="form-group">
            <label for="amount">Amount : </label>
            <input type="number" name="amount"
              placeholder="00.00" class="form-control">
          </div>
          <button type="submit" class="btn btn-warning">Send now</button>
        </form>
      </div>
    </div>
  </div>
{% endblock content %}
```

I am assuming that you now understand how django templating really works,so I am not going to explain all the details now .

Create a url routing for this in the ‘dashboard/urls.py’ :

```
urlpatterns = [  
    path("",views.dashboard_index,name='dashboard_index'),  
    path('add_fund',views.add_fund,name='add_fund'),  
    path('send_money',views.send_money,name='send_money'),  
    path('request_money',views.request_money,name='request_money') #new  
]
```

And the dashboard/views.py:

```
def request_money(request):  
    if request.method == 'GET':  
        return render(request,'dashboard/request-money.html')
```

Now add the url in the base file .

dashboard-base.html:

```
<li class="nav-item">  
    <a class="nav-link" href="{% url 'request_money' %}"> Request money </a>  
</li>
```

modify the ‘dashboard/views.py’ to handle the post request and store the data .

To do that, first of all we need to check whether the data blank or not , if it is blank then we need to redirect the user with message that ‘All fields are required’ . If it is not blank then we need to check whether the sender exists or not in our database . If it does not exist it will redirect the user to the ‘request-money’ page with a message ‘sender does not exist’ . If the sender exists then it will select the sender and store in a variable called sender and create an object

in the 'money_requests' table and store the provided data . As default value of the column 'pending' is true , it will automatically set the record in pending state which we will use later .

dashboard/views.py :

```
def request_money(request):
    if request.method == 'GET':
        return render(request,'dashboard/request-money.html')

    if request.method == 'POST':
        email=request.POST['email']
        amount=request.POST['amount']

        if email==" or amount == " :
            return render(request,'dashboard/request-money.html',
                {'BlankError':'All fields are required'})

        else :

            Sender_sel = User.objects.filter(email=email).count()
            if Sender_sel == 0:
                return render(request,'dashboard/request-money.html',
                    {'doesnotexist':'Sender does not exist'})

            else:
                sender = User.objects.get(email=email)
                money_requests.objects.create(request_from=request.user,
                    request_to=sender,amount=amount
                )
                return render(request,'dashboard/request-money.html',
                    {'request_success':'request successful, wait for approval'})
```

To capture the messages add some conditional template tag in the 'request-money.html' file as follows :

```
<div class="form form-input">
    {% if BlankError %}
    <div class="alert alert-danger">{{ BlankError }}</div>
    {% endif %}
    {% if doesnotexist %}
    <div class="alert alert-danger">{{ doesnotexist }}</div>
    {% endif %}
    {% if request_success %}
    <div class="alert alert-success">{{ request_success }}</div>
    {% endif %}
</form action="" method="post" class="m-4">
```

Till now request money feature is done, but that's not the end. Users need to accept the request or decline . We will add this functionality in another page where all the money requests will be shown .

Showing all the pending money requests :

Create a new page in the 'template/dashboard' directory with name 'pending-requests.html' . Again the same process, extend the 'dashboard-base.html' file and add a table with table header 'date', 'time', 'with', 'amount', 'action' .

Create a route in the 'dashboard/urls.py' and add the following line:

```
path('pending_requests', views.pending_requests, name='pending_requests'),
```

add a new function at 'dashboard/views.py' :

```
@login_required(login_url='/accounts/login/')
def pending_requests(request):
    sel_requests = money_requests.objects.filter(Q(request_from=request.user) |
        Q(request_to=request.user))
    return render(request, 'dashboard/pending-requests.html',
        {'request_data': sel_requests, 'count': sel_requests.count()})
```

In this function it will filter all the data which contains this logged user id either as request_from or as request-to because we need to show all the requests data associated with current logged in user . Then it will pass the objects as context with key 'request_data' . So, what about the count? As the name suggests count() will count the objects and pass the value through context . We will use this information to render conditional data in our template .

Open 'money-requests.html' and paste the code :

```
{% extends 'dashboard/dashboard-base.html' %}
{% block title %} {% endblock title %}
{% block content %}

<div class="row">

    <div class="col-md-8 mx-auto mt-4">
        <h2 class="font-weight-bold text-center mb-4">
            Pending money requests</h2>
        {% if count == 0 %}
        <h3 class="text-center text-danger">No available data !</h3>
        {% else %}
        {% if messages %}
            {% for message in messages %}
            <div class="alert alert-success mt-4 alert-dismissible fade show">
                <strong>Success!</strong> {{ message }}
                <button type="button" class="close" data-dismiss="alert">&times;</button>
            </div>
            {% endfor %}
            {% endif %}

        <table class="table table-stripedS mt-4">
            <thead class="bg-dark text-white">
                <tr><th class="text-center">Date</th>
                    <th class="text-center">Type</th>
                    <th class="text-center">With</th>
                    <th class="text-center">Amount</th>
                    <th class="text-center">Action</th>
                </tr>
            </thead>

    </div>

    </div>
```

We will use django message framework so why I added this {% if messages %} to show messages conditionally .

Next:

```

<tbody class="bg-dark text-white">
    {% for data in request_data %}
    <tr>
        <td>{{ data.date }}</td>
        {% if data.request_from == request.user %}
        <td>Your request</td>
        <td>{{ data.request_to.full_name }}</td>
        {% else %}
        <td>peoples request</td>
        <td>{{ data.request_from.full_name }}</td>
        {% endif %}

        <td>{{ data.amount }}</td>
        <td><a href="{% url 'delete_request' data.id %}" class="pb-1 mt-4">
            <i class="far fa-trash-alt text-danger"></i></a>
            {% if not data.request_from == request.user %}
            <a href="{% url 'approve_request' data.id %}" class="ml-4 pb-1 my-auto mt-2">
            <i class="fas fa-check-square text-success"></i></a>
            {% endif %}
        </td>
    </tr>
    {% endfor %}
</tbody>
</thead>
</table>
{% endif %}
</div>

</div>
<style type="text/css">
    table {font-size: 1.2rem}
    tr i{font-size: 2rem !important;

    }
    tr a:hover{
        border-bottom:3px solid green;

    }
</style>

{% endblock content %}

```

These conditional statements will render the objects conditionally which I discussed earlier .

But we need to be able to delete the request . To do this go to 'dashboard/views.py' and add a new function with name 'delete_request' . And add a url routing at 'dashboard/urls.py' :

```
path('delete_request/<int:id>',views.delete_request,name='delete_request'),
```

This url will accept an integer value as id and it will be passed to function 'delete_request()' at the views.py .

dashboard/views.py :

```
@login_required(login_url='/accounts/login/')
def delete_request(request,id):
    try :
        sel_req=money_requests.objects.get(id=id)
        if sel_req.request_from == request.user or sel_req.request_to == request.user :

            sel_req.delete()
            messages.success(request, "Money request aborted ")
            return redirect('pending_requests')
        else :
            messages.success(request,"Not allowed !")
            return redirect('pending_requests')

    except money_requests.DoesNotExist:
        return redirect('pending_requests')
```


First of all it will try to select the request in the 'money_requests' table with id .

Next, it will check whether the user is present in the request or not. Why it is needed? To make sure that other people who are not involved in this request would not be able to delete any request data. It's only for security purpose .

If the logged user is present it will then delete the record with objects.delete() method and add a message that 'money request aborted' . To use this message feature just at this at the top of the views file:

from django.contrib import django messages . If the user is not part of the request it will redirect him with a message 'not allowed' .

So far we have added 'delete request' functionality . But how about request approval?

Well ,we will do this now.

If a user accepts a request , his balance will be deducted according to the request amount and will be added to the requester account .

Create a new function in 'dashboard/views.py' with name approve_request:

```
@login_required(login_url='/accounts/login/')
def approve_request(request,id):
    try :
        sel_req=money_requests.objects.get(id=id)
        if (sel_req.request_to != request.user):
            messages.success(request,"Permission denied !")
            return redirect('pending_requests')

        if not UserBalance.objects.filter(user=sel_req.request_from).exists():
            UserBalance.objects.create(user=sel_req.request_from,balance=0)
            pass
        if not UserBalance.objects.filter(user=sel_req.request_to).exists():
            UserBalance.objects.create(user=sel_req.request_to,balance=0)
            pass
```

At first it will try to select the request with id and after it will check whether the user is from this transaction or not. Then it will check the existence of a balance record for the user at UserBalance model . If it finds it then it will move to the next part otherwise it will create one .

```
user_bal_obj=UserBalance.objects.get(user=request.user)
rec_bal_obj = UserBalance.objects.get(user=sel_req.request_from)
if user_bal_obj.balance < sel_req.amount :
    messages.success(request,"Not enough balance ")
    return redirect('pending_requests')
else :
    user_bal_obj.balance=(user_bal_obj.balance-sel_req.amount)
    user_bal_obj.save()
    rec_bal_obj.balance=(rec_bal_obj.balance + sel_req.amount)
    rec_bal_obj.save()
    sel_req.delete()
    messages.success(request,"Request accepted and paid !")
    return redirect('pending_requests')

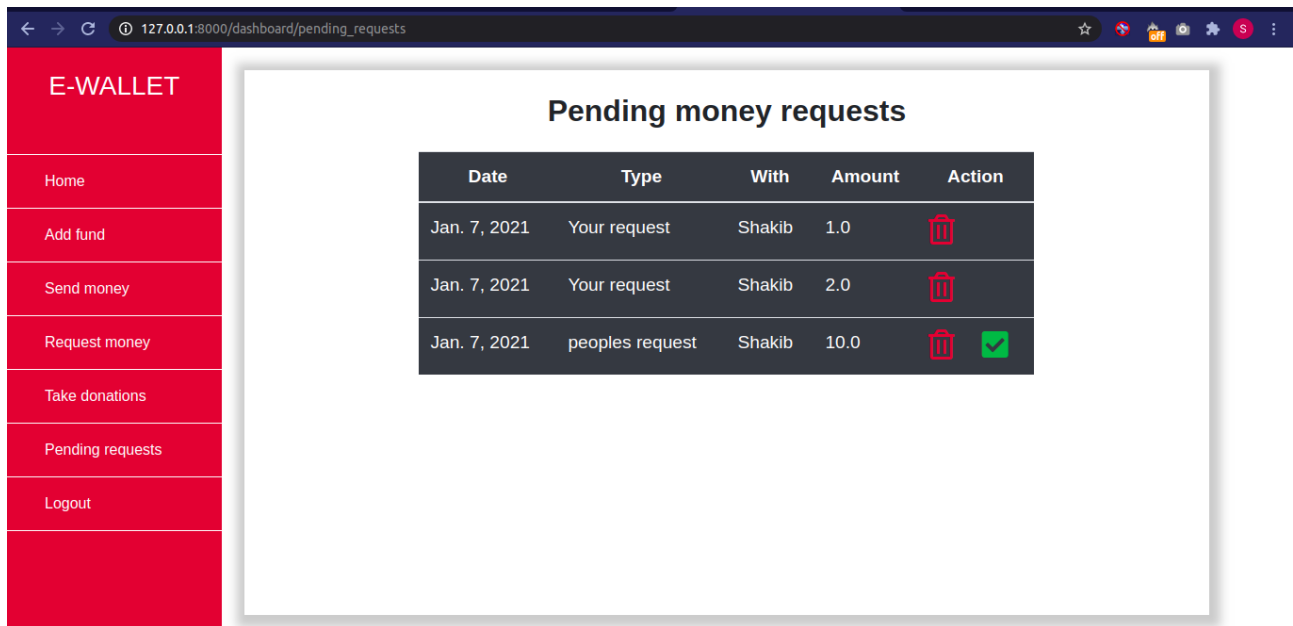
except money_requests.DoesNotExist:
    messages.success(request,"Request not found")
    return redirect('pending_requests')
```

After , it will check the select the user balance and receiver balance objects and check if the user balance is less than the requested amount. If it is true then it will add a message ‘Not enough balance’ and redirect the user to the ‘pending-requests.html’ page . If not then it will deduct the balance from the user account and add it to the receiver account and save the updated data .

Last thing, at another url router for this at ‘dashboard/urls.py’ :

```
path('approve_request/<int:id>',views.approve_request,name='approve_request')
```

That's all you need to do to add this feature ! . Now our 'pending-requests' page looks like this :



To show the total pending requests and user pending requests on dashboard just add these code in 'dashboard/views.py' (dashboard_index()):

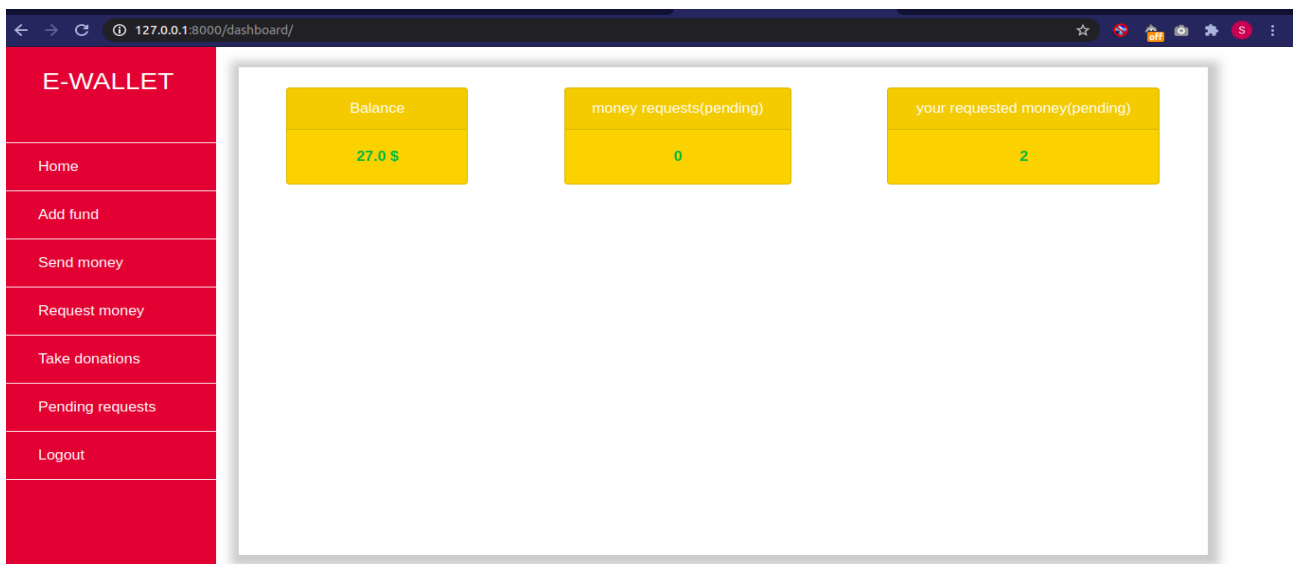
```
@login_required(login_url='/accounts/login/')
def dashboard_index(request):
    user_bal = 0.00
    try :
        curr_bal = UserBalance.objects.get(user=request.user)
        user_bal = curr_bal.balance
        pending_req=money_requests.objects.filter(request_to=request.user).count() #new
        pending_user_req=money_requests.objects.filter(request_from=request.user).count() #new
        return render(request,'dashboard/dashboard.html',
            {'balance':user_bal,'pending_req':pending_req,'pending_user_req':pending_user_req})
    except :
        pass

    return render(request,'dashboard/dashboard.html',{'balance':user_bal})
```

We just have filtered the request objects and pass the count with the context. In the template add this (dashboard.html) :

```
<div class="card bg-warning" style="width:15rem">
    <div class="card-header text-white text-center">
        money requests(pending)
    </div>
    <div class="card-body text-center font-weight-bold text-success">
        {% if pending_req %}
        {{pending_req}}
        {% else %}0
        {%endif %}
    </div>
</div>
<div class="card bg-warning" style="width:18rem">
    <div class="card-header text-white text-center">
        your requested money(pending)
    </div>
    <div class="card-body text-center font-weight-bold text-success">
        {% if pending_user_req %}
        {{pending_user_req}}
        {% else %}0
        {% endif %}
    </div>
</div>
```

Now our dashboard:



Hurrah! It is showing real pending request count dynamically .

Adding donation taking feature :

To accept donation we need to generate a unique donation link for receiving donation from outside of our site network . First create a new page in the 'template/dashboard/' with name donation-link.html . As I said earlier you need to extend the base template and paste the code :

```
{% extends 'dashboard/dashboard-base.html' %}
{% block title %}donation link {% endblock title %}
{% block content %}

<div class="row">
  <div class="col-md-5 mx-auto mt-4">
    <h2 class="text-center mb-4">Your donation link</h2>
    <div class="text-center donate-link font-weight-bold">
      <div id="unique-link"><h6 class="d-inline" id="lnk">
        http://{ { request.META.HTTP_HOST } }/donate/?user={ {request.user.id} }</h6>
        <i class="ml-auto fas fa-copy" id="copy-text" data-clipboard-target="#lnk"></i>
      </div>
    </div>
  </div>
</div>

<style type="text/css">
  .donate-link{
    margin-top: 7rem ;
    padding:3rem;
    box-shadow: 2px 3px 5px 4px rgba(0,0,0,0.2);
  }
  .fa-copy{
    font-size: 2rem;
  }
</style>

{%endblock content %}
```

It is just a simple link showing page with one click copy feature . You may be wondering what is this `{{ request.META.HTTP_HOST }}` , This is another django template tag to get the domain name dynamically . I have added a icon as a click to copy button . I am using jquery clipboard plugin . Just paste the code in the 'dashboard-base.html' at the end -

```
<script src="https://cdn.jsdelivr.net/clipboard.js/1.5.12/clipboard.min.js"></script>

<script type="text/javascript">
$(function(){
  new Clipboard('#copy-text');
});
</script>
```

So, how does it work? - #copy-text is our button id ,in our case it is icon and in the html file you see that i tag has an attribute data-clipboard-target='#lnk' . This attribute tells the script to copy the text in the clipboard which element has id ='link' . That's it for one click copy .

```
<i class="ml-auto fas fa-copy" id="copy-text" data-clipboard-target="#lnk"></i>
```

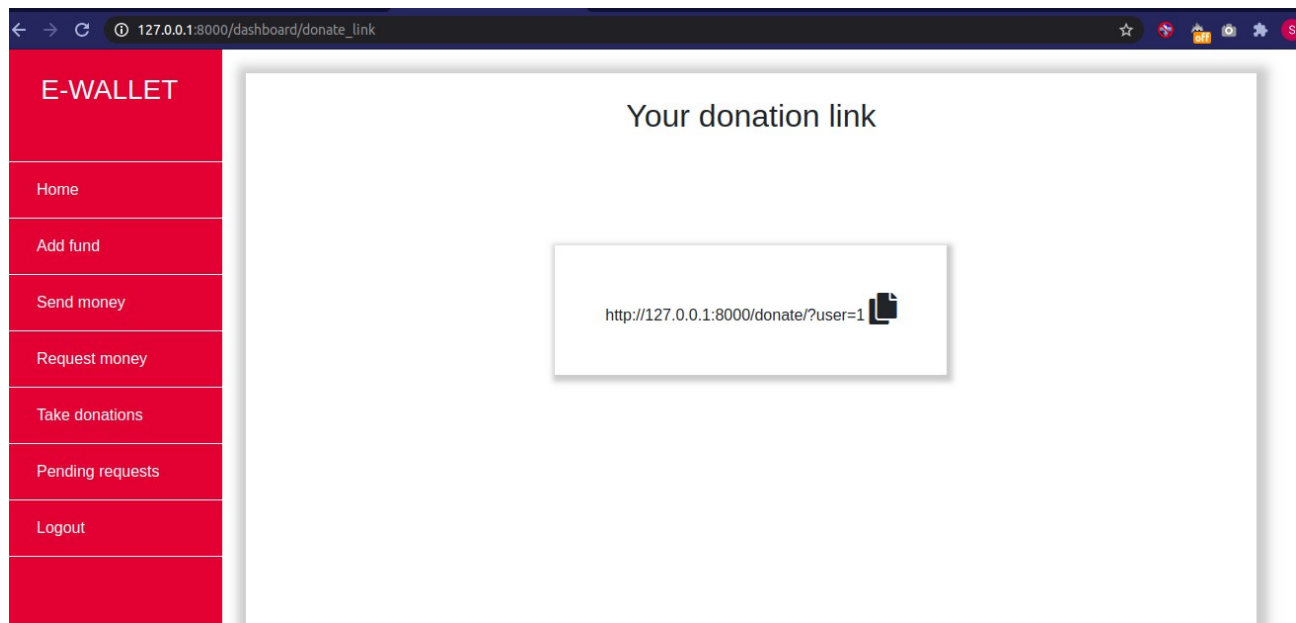
Create a routing in 'dashboard/urls.py' :

```
path('donate_link',TemplateView.as_view(template_name='dashboard/donation-link.html'),name='donate_link')
```

We don't need to use view for this routing. Just import the Template view by this : `from django.views.generic import TemplateView` .

This TemplateView will render the template . Now add this url in the 'dashboard-base.html' file and you are ready to move to the next step.

Donation-link.html page:



Now we need to create a donation taking form to accept donation from others.

Create a new page at 'template' directory with name 'donation-page.html'

You don't need to rewrite into this , just copy the codes from 'dashboard/add-fund.html' and modify like this :

donation-page.html :

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <link rel="stylesheet" type="text/css" href="{% static 'css/styles.css' %}">
  <title>Donate now</title>
</head>
<header>
  <body>
    <div class="container">
      <div class="row">
        <div class="col-md-5 .fund-form ml-auto mr-auto mt-4">
          <h3 class="font-weight-bold mb-4 mt-auto text-center"> Donate
            {{name}} with one click</h3>
```

First of all change the title of the page and the h3 text . Inside the h3 I have used a template tag `{{}}` to show the user full name which will be passed from `views.py`.

```
<div class="form form-input">
<form id="payment-form" class="m-4" action="#" method="post">
    {% csrf_token %}
    <div class="form-group">
        <input type="number" placeholder="amount" name="amount" class="form-control">
    </div>

    <div class="one-liner">
        <div class="card-frame">
<!-- form will be added here -->
        </div>

    </div>
    <input id="token" type="hidden" name="token">
<input type="hidden" name="id" value="{{id}}">
        <input id="pay-button" type="button" class="btn btn-primary" value="Donate now">
    </form>
</div>
</div>
</div>
</div>
```

Everything is same but just add a new field with type hidden and value = `{{id}}` . Change the button text to ‘Donate now’ . Now where does it get this id from ? Well, it will come from our views file .

Rest of the code is same as add-fund file so I don’t want to include it here .

Create a url routing for it in ‘`wallet_app/urls.py`’ :

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.index_page, name='index_page'),
    path('accounts/', include('accounts.urls')),
    path('dashboard/', include('dashboard.urls')),
    path('donate/', views.donate, name='donate') #new
]
```


rendering donation-page with get request.

In 'wallet_app/views.py' :

```
def donate(request):
    if request.method == 'GET':
        rec_id=request.GET['user']
        user=get_object_or_404(User,id=rec_id)
        return render(request,'donate-page.html',{'id':user.id,'name':user.full_name})
```

Firstly it checks if it is a GET request or POST . If it is GET then it will extract the donation receiver id from the url and store it in 'rec_id' variable and it will select it by the method 'get_object_or_404(model,condition)'. You need to import it first by 'from django.shortcuts import get_object_or_404' at the top of the views .It will try the select the user ,but if it fails it will raise an 404 not found error .After it will render the donation page with context 'id' and 'name' which we have used in the donation-page .

Now we need to handle the post request from donation-page.

wallet_app/views.py :

```
if request.method == 'POST':
    amount=int(request.POST['amount'])
    token=request.POST['token']
    userid = request.POST['id']
    if amount == " " or token == " ":
        return render(request,'donate-page.html',{'BlankError':'All fields are required'})
    else:
        api = sdk.get_api(secret_key='sk_test_f2a38861-4256-413b-a95c-4c3822b8f508')

        try:
            payment = api.payments.request(
                source={
                    'token': token,
                },
                amount=amount*100,
                currency=sdk.Currency.USD,
            )
```

If it is a post request it will extract the amount,id and token from the donation page. Next, it will check whether the requests are blank or not . If it is blank it will move to the next block and try to use the sdk to complete the payment.

```
if payment.approved == True and payment.status == "Authorized":
    try :

        user_bal = UserBalance.objects.get(user__id=int(userid))
        curr_bal = user_bal.balance
        new_bal=float(curr_bal + int(amount) )
        user_bal.balance = new_bal
        user_bal.save()
        return render(request,'donate-page.html',
                        {'credited':'Donation successful !'})
    except UserBalance.DoesNotExist:
        UserBalance.objects.create(user_id=userid,balance=int(amount))
        return render(request,'donate-page.html',
                        {'credited':'Donation successful !'})

    else:
        return render(request,'donate-page.html',
                        {'NoPayment':'your card can not be authorized !'})
except sdk.errors.CheckoutSdkError as e:

    return render(request,'donate-page.html',
                  {'PaymentError':'Payment Failed ! Try again later'})
```

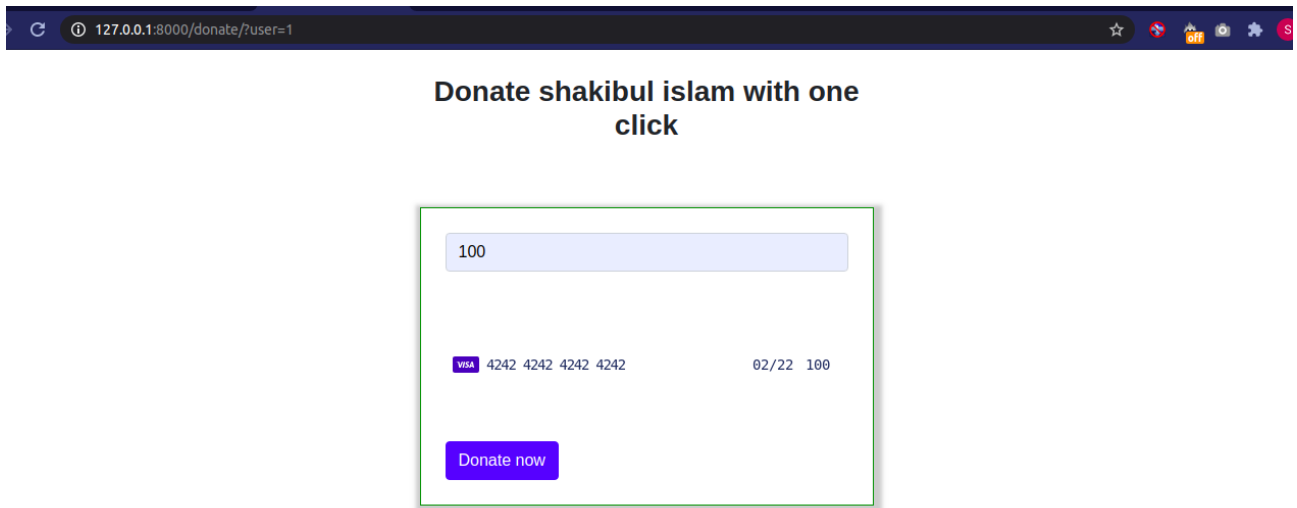
If the payment is approved then it will add the balance to the donation taker UserBalance table and if the user has no UserBalance object then it will create one and store the amount . If the payment is not approved , it will re render the donate page with context message ‘your card can not be authorized’ . If the try block fails to successfully process the payment it will go to the except block and raise sdk error and render the donation-page with context ‘payment-error’ .

Now you can test the donation functionality by using test cards from checkout.com .

Donation-page :

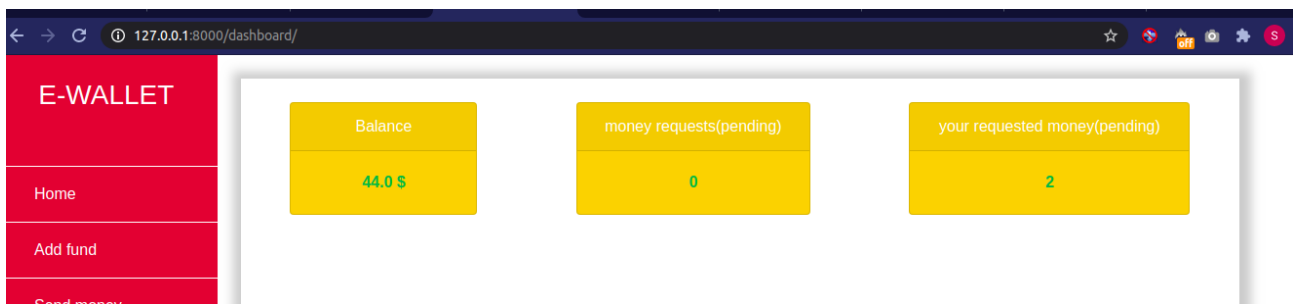
Go to your dashboard and find your donation link . Just visit it and donate with test card.

Filling up the information:



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/donate/?user=1". The page title is "Donate shakibul islam with one click". The form contains a text input field with the value "100", a Visa card number "4242 4242 4242 4242", an expiration date "02/22", and a cardholder name "100". A purple "Donate now" button is at the bottom.

Current balance :




The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/dashboard/". The page title is "E-WALLET". The dashboard has a red sidebar with links: "Home", "Add fund", and "Send money". The main content area has three yellow boxes: "Balance" with "44.0 \$", "money requests(pending)" with "0", and "your requested money(pending)" with "2".

Now hit the Donate now button :

Donate with one click

Donation successful !

 Card number

MM/YY CVV

Donate now

Check the balance again :

E-WALLET

Home

Add fund

Send money

Balance

144.0 \$

money requests(pending)

0

your requested money(pending)

2

Donation money has been added !

Congratulations, you have just added another feature !

You did a lot with this tutorial and finally have completed the entire project. If you want to modify or add some additional features into it ,feel free to do it. This project is available on github .

Link:<https://github.com/iamshakibulislam/online-ewallet>

Thanks for staying with me ...