PART-5

django Web Framework

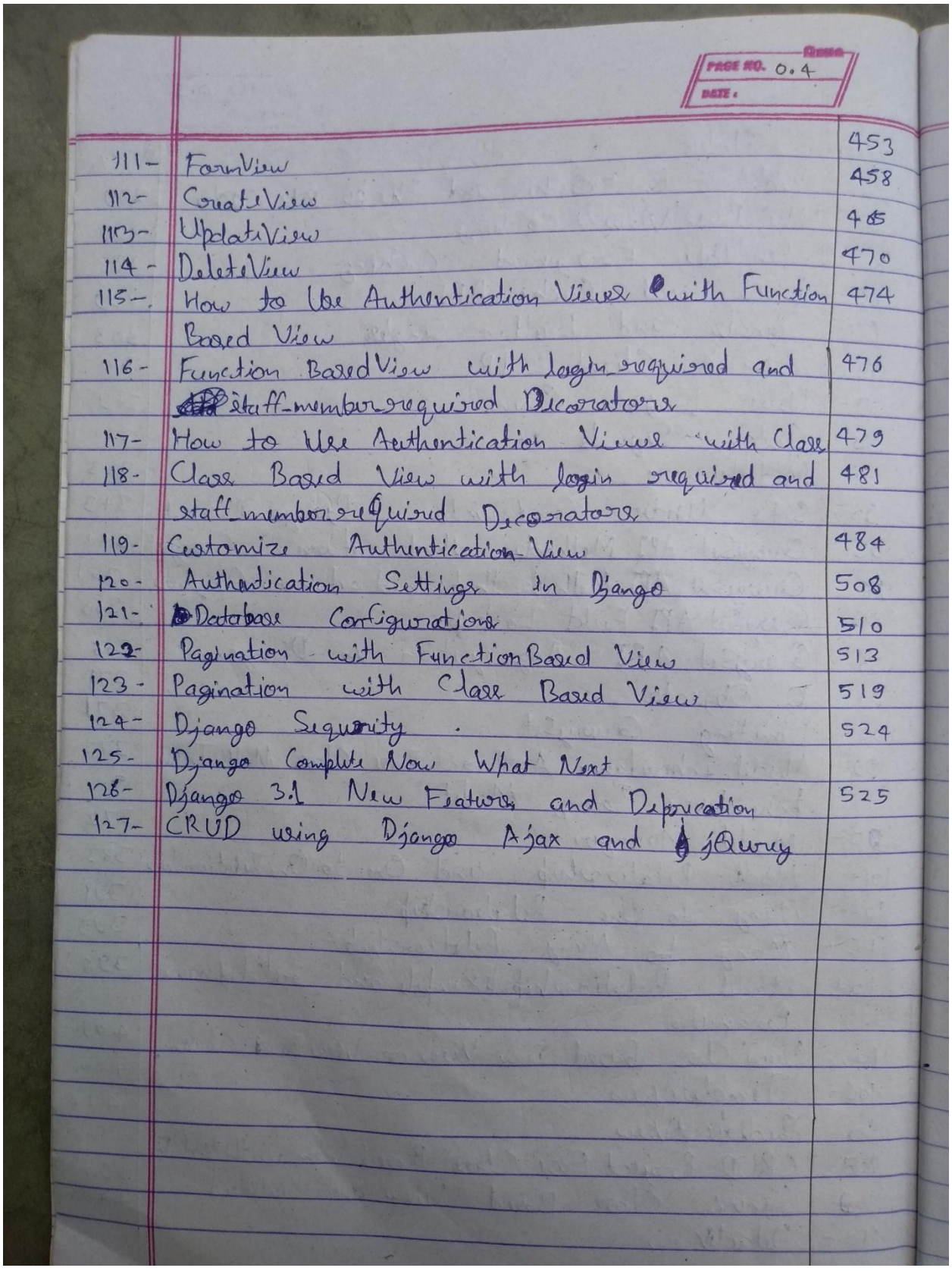
Geeky Shows YouTube Channel Learning Notes

SATYAM SETH

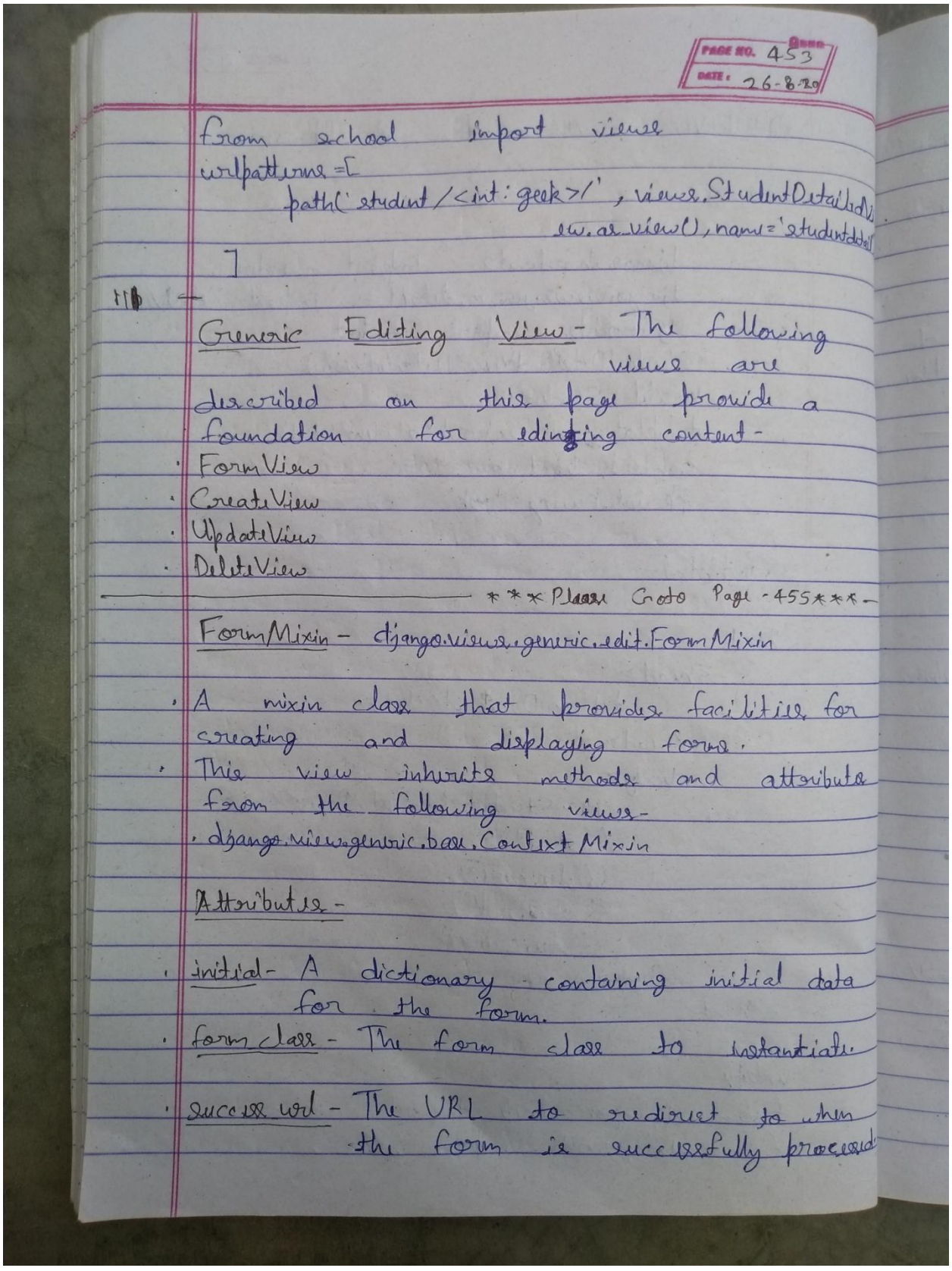
14/09/2020

Source Code – https://github.com/satyam-seth/django_learning

Playlist Link – https://www.youtube.com/playlist?list=PLbGui_ZYuhigchy8DTw4pX4duTTpvqlh6



PAGE NO. 0.4
DATE :



PAGE NO. 453
DATE: 26-8-20

```
from school import views
urlpatterns =[
    path('student/<int:geek>/', views.StudentDetailsVi
                                ew.as_view(), name='studentdetail
]
```

Generic Editing View - The following views are described on this page provide a foundation for editing content -

- FormView
- CreateView
- UpdateView
- DeleteView

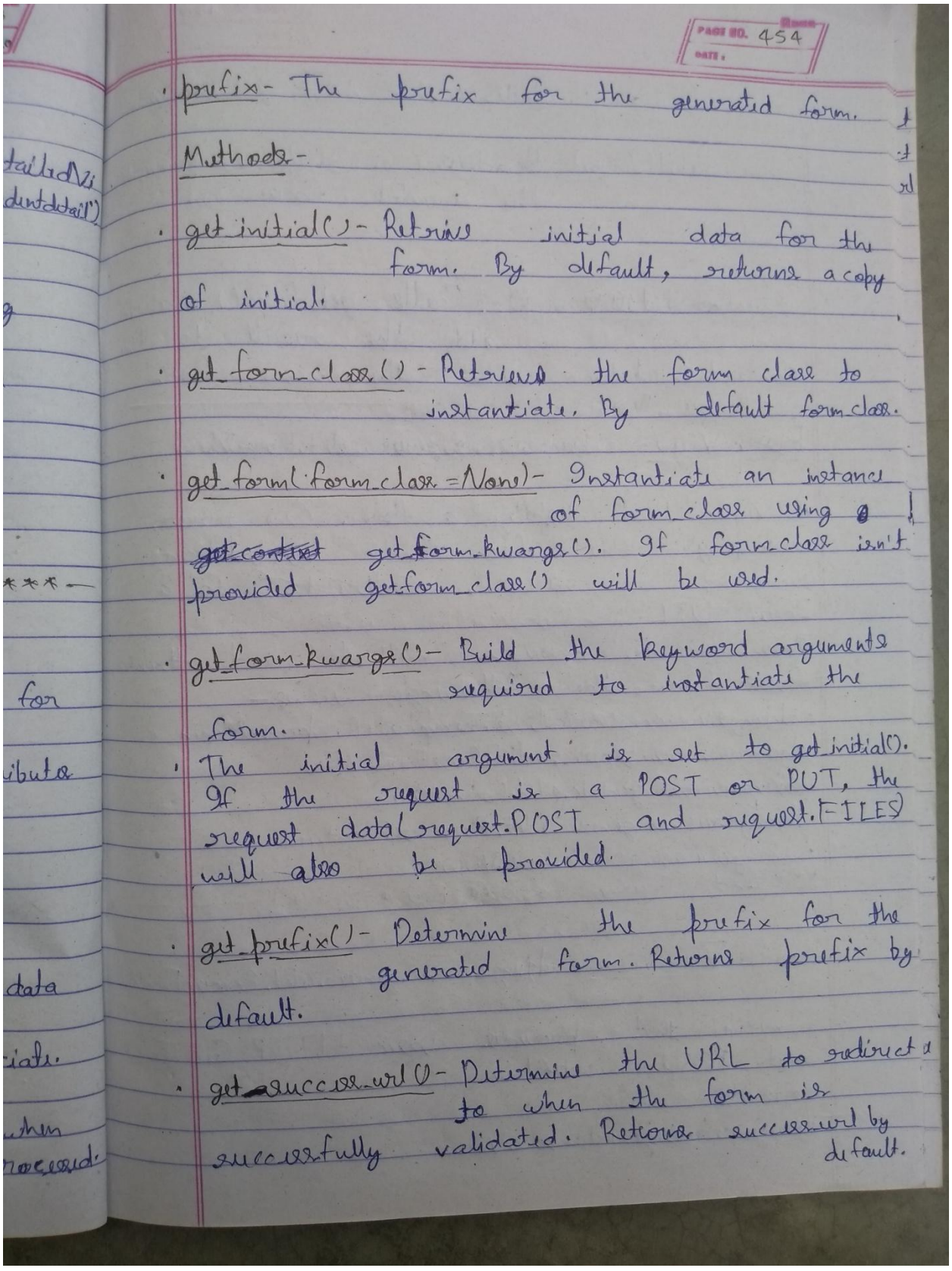
*** Please Goto Page - 455 ***

FormMixin - django.views.generic.edit.FormMixin

- A mixin class that provides facilities for creating and displaying forms.
- This view inherits methods and attributes from the following views -
  - django.view.generic.base.ContextMixin

Attributes -

- initial - A dictionary containing initial data for the form.
- form_class - The form class to instantiate.
- success_url - The URL to redirect to when the form is successfully processed

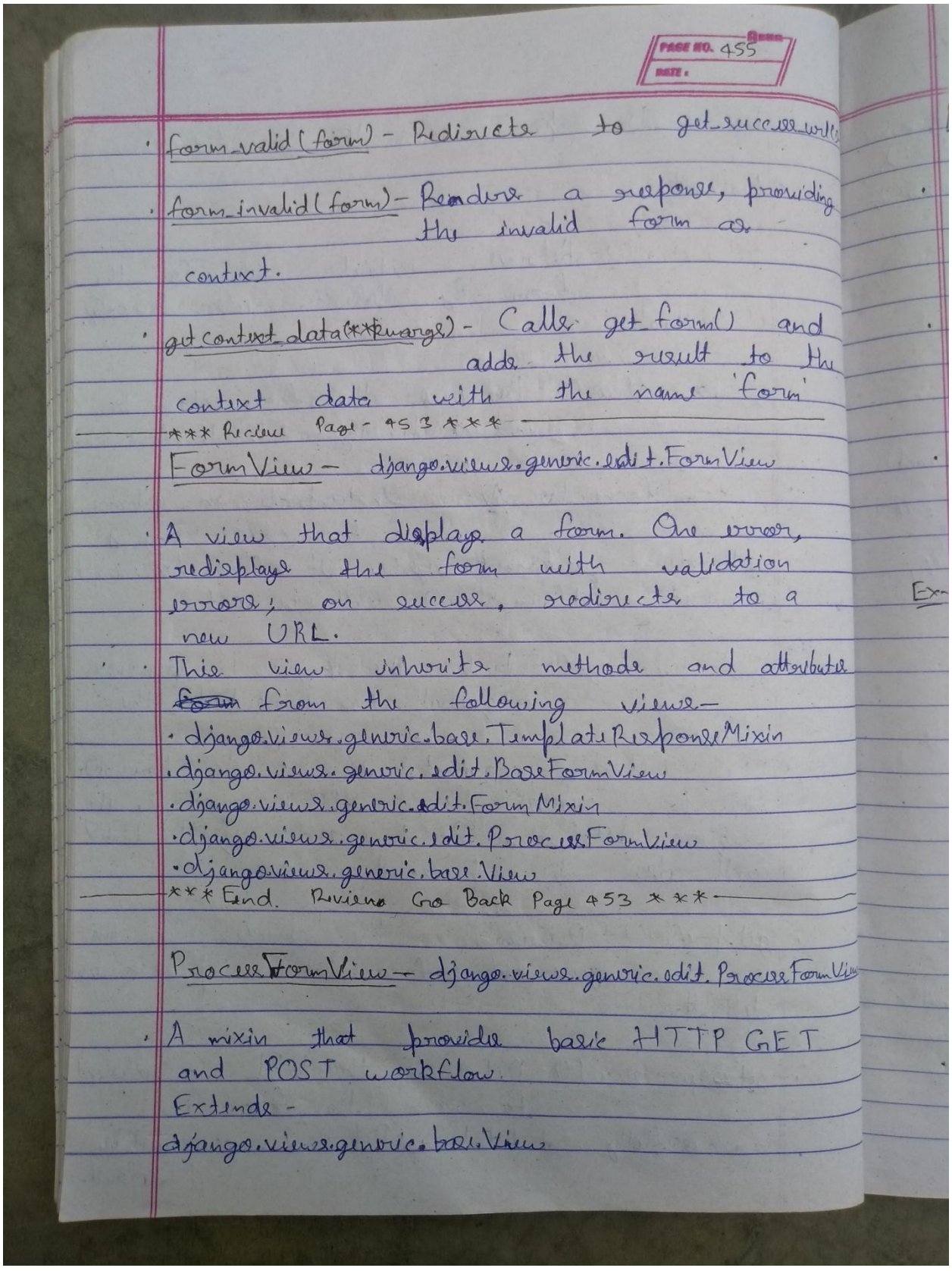


PAGE NO. 454
DATE :

- prefix - The prefix for the generated form.

Methods -

- get_initial() - Retrive initial data for the form. By default, returns a copy of initial.
- get_form_class() - Retrieve the form class to instantiate. By default form_class.
- get_form(form_class=None) - Instantiate an instance of form_class using ~~get_context~~ get_form_kwargs(). If form_class isn't provided get_form_class() will be used.
- get_form_kwargs() - Build the keyword arguments required to instantiate the form.
- The initial argument is set to get_initial(). If the request is a POST or PUT, the request data (request.POST and request.FILES) will also be provided.
- get_prefix() - Determine the prefix for the generated form. Returns prefix by default.
- get_success_url() - Determine the URL to redirect to when the form is successfully validated. Returns success_url by default.



PAGE NO. 455
DATE :

- form_valid(form) - Redirects to get_success_url()
- form_invalid(form) - Renders a response, providing the invalid form as context.
- get_context_data(**kwargs) - Calls get_form() and adds the result to the context data with the name 'form'

*** Review Page - 453 ***

FormView - django.views.generic.edit.FormView

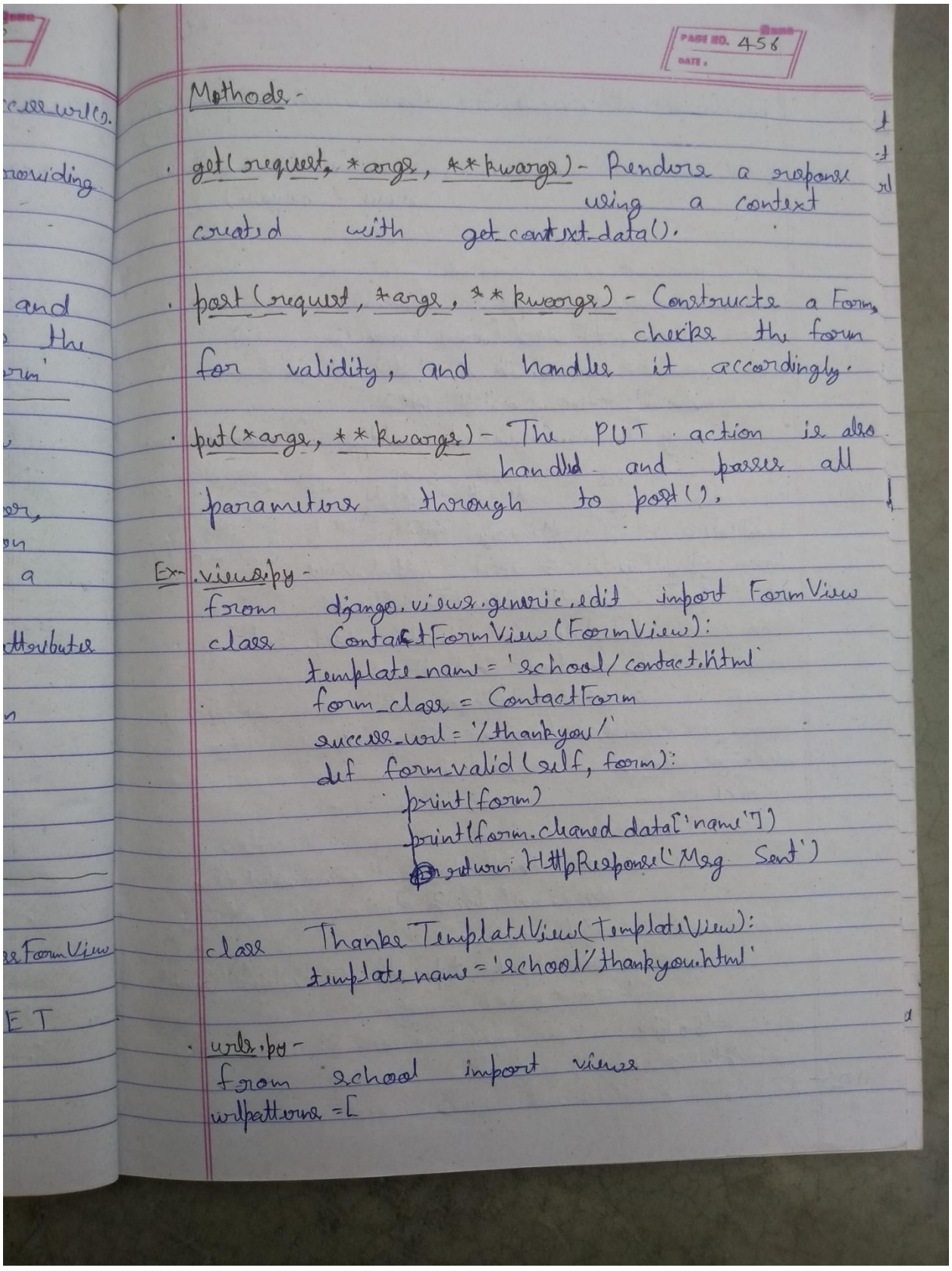
- A view that displays a form. On error, redisplays the form with validation errors; on success, redirects to a new URL.
- This view inherits methods and attributes ~~form~~ from the following views -
  - django.views.generic.base.TemplateResponseMixin
  - django.views.generic.edit.BaseFormView
  - django.views.generic.edit.FormMixin
  - django.views.generic.edit.ProcessFormView
  - django.views.generic.base.View

*** End. Review Go Back Page 453 ***

ProcessFormView - django.views.generic.edit.ProcessFormView

- A mixin that provides basic HTTP GET and POST workflow.

Extends -
django.views.generic.base.View



PAGE NO. 458
DATE :

Methods -

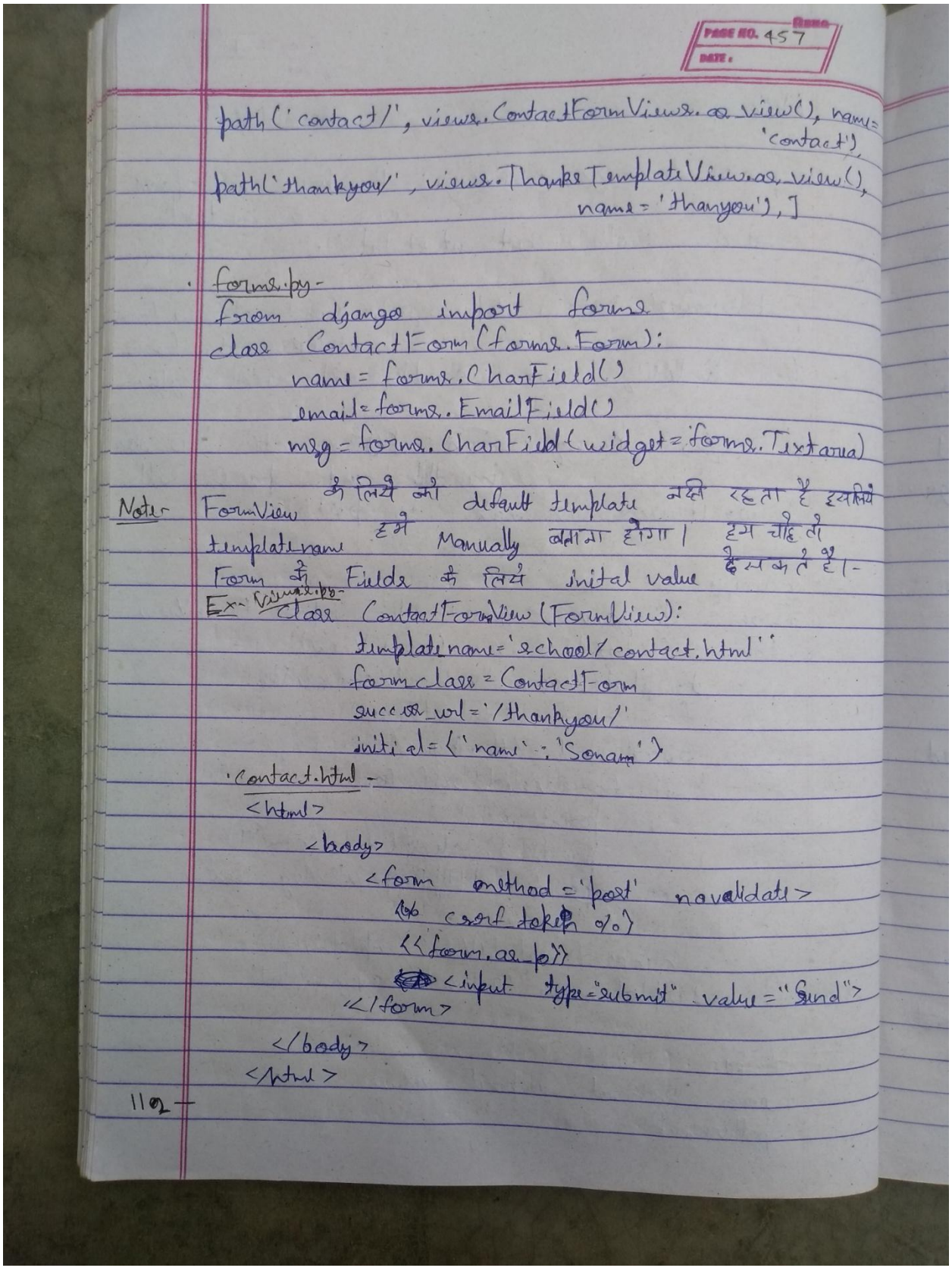
- get(request, *args, **kwargs) - Renders a response using a context created with get_context_data().
- post(request, *args, **kwargs) - Constructs a Form, checks the form for validity, and handles it accordingly.
- put(*args, **kwargs) - The PUT action is also handled and passes all parameters through to post().

Ex. views.py -

```
from django.views.generic.edit import FormView
class ContactFormView(FormView):
    template_name = 'school/contact.html'
    form_class = ContactForm
    success_url = '/thankyou/'
    def form_valid(self, form):
        print(form)
        print(form.cleaned_data['name'])
        return HttpResponse('Msg Sent')

class ThanksTemplateView(TemplateView):
    template_name = 'school/thankyou.html'
```

- urls.py -

```
from school import views
urlpatterns = [
```



PAGE NO. 457
DATE :

path('contact/', views.ContactFormViews.as_view(), name='contact'),
path('thankyou/', views.ThanksTemplateView.as_view(), name='thanyou'),]

- forms.py -

```
from django import forms
class ContactForm(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    msg = forms.CharField(widget=forms.Textarea)
```

Note- FormView के लिये जो default template नही रहता है इसलिये हमे templatename Manually बनाना होगा। हम चाहे तो Form के Fields के लिये inital value भी set कर सकते है।-

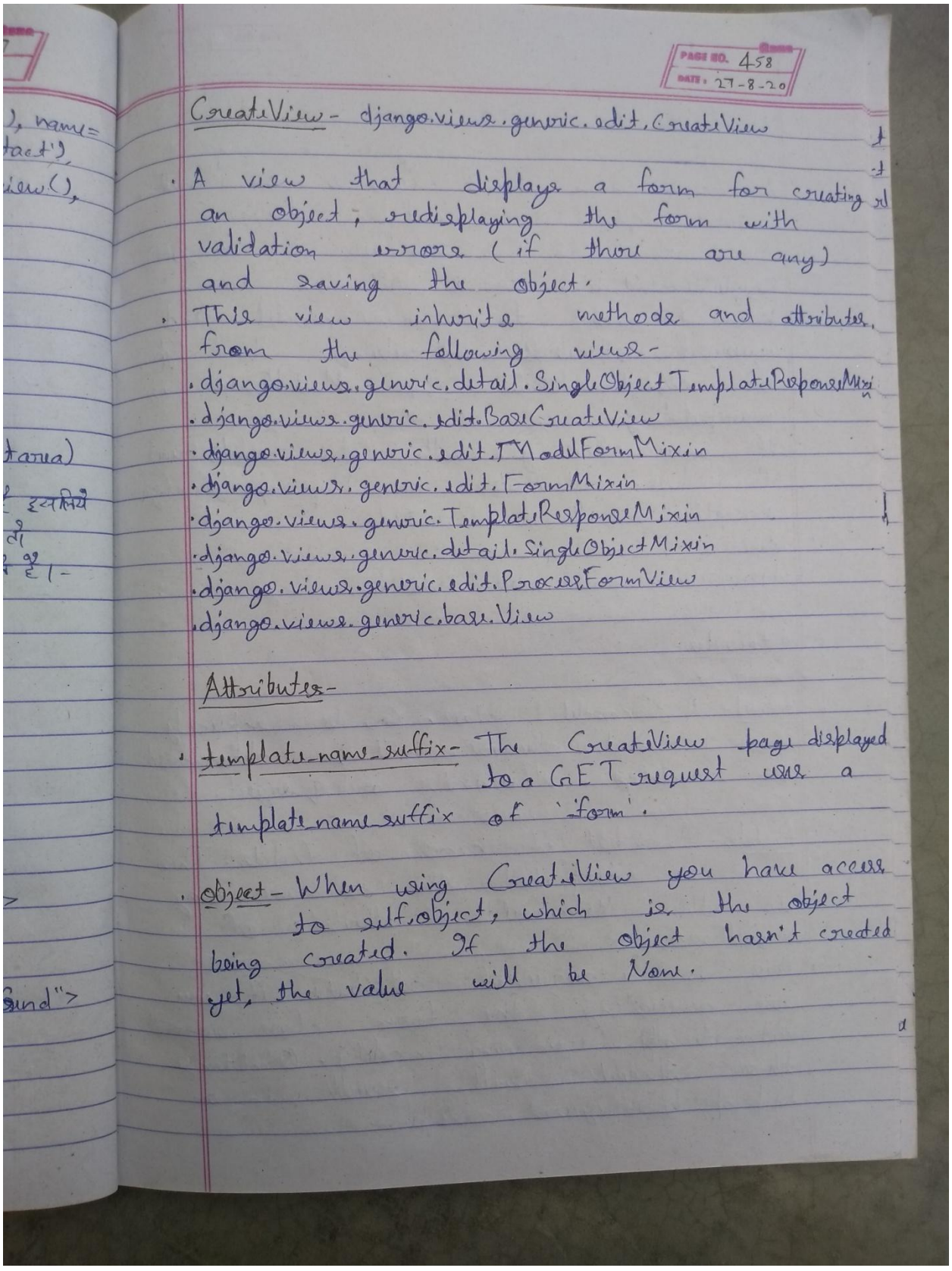
Ex- views.py-

```
class ContactFormView(FormView):
    template_name='school/contact.html''
    form_class = ContactForm
    success_url='/thankyou/'
    initial={'name':'Sonam'}
```

- contact.html -

```
<html>
    <body>
        <form method='post' novalidate>
            {% csrf_token %}
            {{form.as_p}}
            <input type="submit" value="Send">
        </form>
    </body>
</html>
```

11Q-



PAGE NO. 458
DATE : 27-8-20

CreateView - django.views.generic.edit.CreateView

- A view that displays a form for creating an object, redisplaying the form with validation errors (if there are any) and saving the object.
- This view inherits methods and attributes from the following views -
  - django.views.generic.detail.SingleObjectTemplateResponseMixin
  - django.views.generic.edit.BaseCreateView
  - django.views.generic.edit.ModelFormMixin
  - django.views.generic.edit.FormMixin
  - django.views.generic.TemplateResponseMixin
  - django.views.generic.detail.SingleObjectMixin
  - django.views.generic.edit.ProcessFormView
  - django.views.generic.base.View

Attributes -

- template_name_suffix - The CreateView page displayed to a GET request uses a template_name_suffix of '_form'.
- object - When using CreateView you have access to self.object, which is the object being created. If the object hasn't created yet, the value will be None.

PAGE NO. 459
DATE :

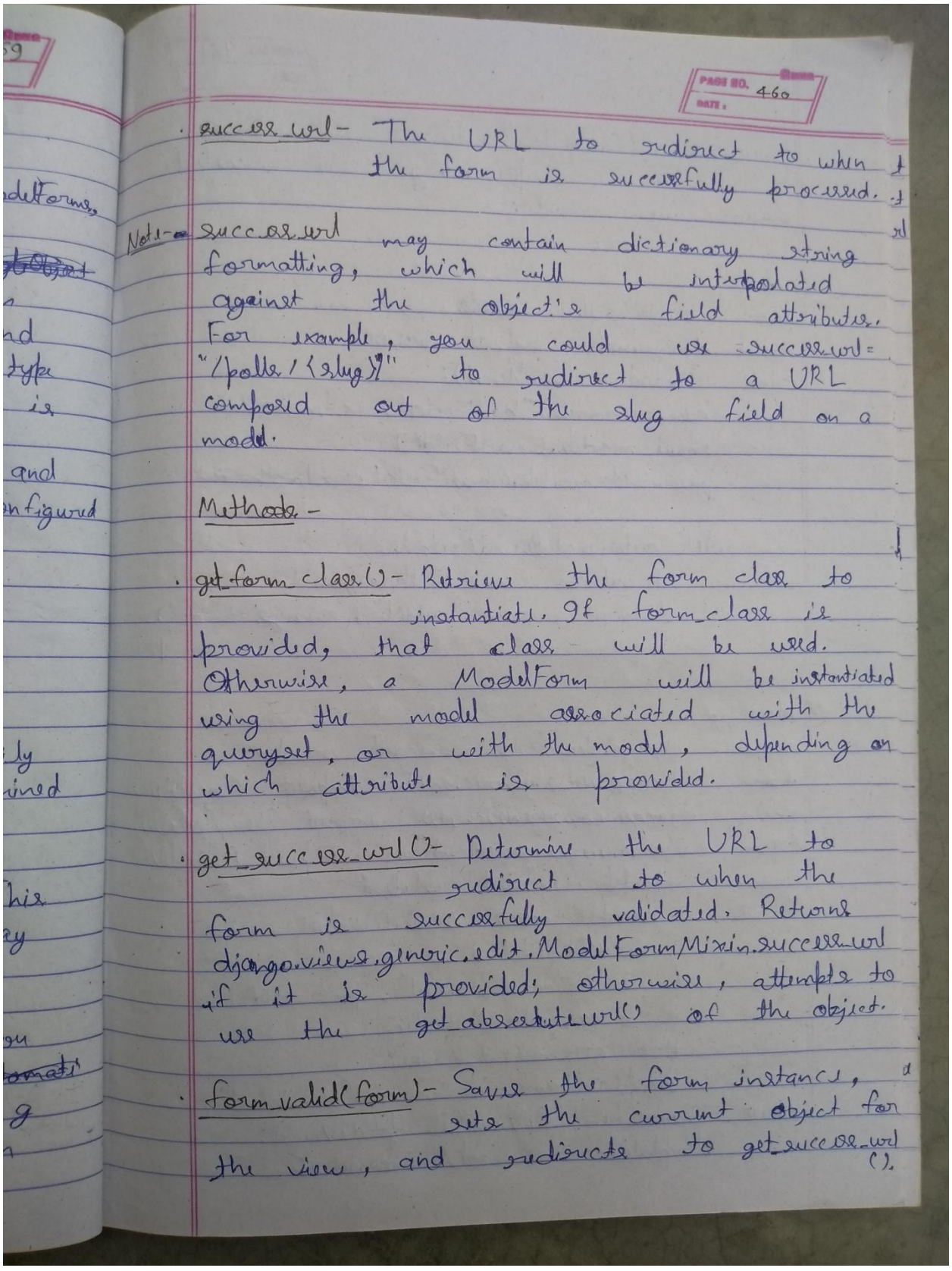
ModelFormMixin-

- A form mixin that works on ModelForms, rather than a standalone form.
- Since this is a subclass of ~~SingleObject~~ SingleObjectMixin, instances of this mixin have access to the model and queryset attributes, describing the type of object that the ModelForm is manipulating.
- If you specify both the fields and form_class attributes, an ImproperlyConfigured exception will be raised.
- Mixin
  'django.views.generic.edit.FormMixin

Note-

Attributes-

- model- A model class. Can be explicitly provided, otherwise will be determined by examining self.object or queryset.
- fields- A list of names of fields. This is interpreted the same way as the Meta.fields attribute of ModelForm.
- This is a required attribute if you are generating the form class automatically (e.g. using model). Omitting this attribute will result in an ImproperlyConfigured exception.



PAGE NO. 460
DATE :

- success_url - The URL to redirect to when the form is successfully processed.

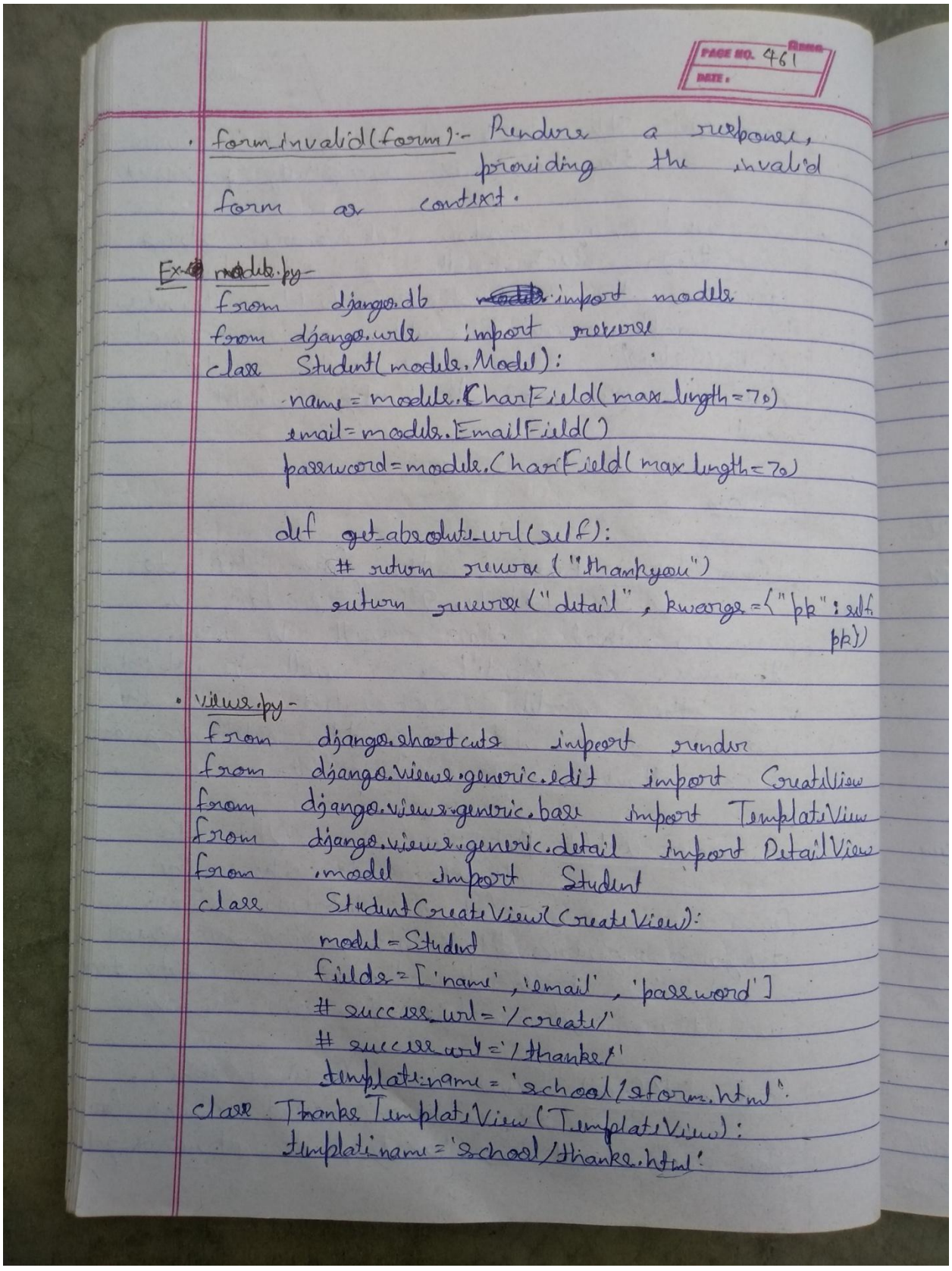
Note - success_url may contain dictionary string formatting, which will be interpolated against the object's field attributes. For example, you could use success_url="/polls/{slug}/" to redirect to a URL composed out of the slug field on a model.

Methods -

- get_form_class() - Retrieve the form class to instantiate. If form_class is provided, that class will be used. Otherwise, a ModelForm will be instantiated using the model associated with the queryset, or with the model, depending on which attribute is provided.

- get_success_url() - Determine the URL to redirect to when the form is successfully validated. Returns django.views.generic.edit.ModelFormMixin.success_url if it is provided; otherwise, attempts to use the get_absolute_url() of the object.

- form_valid(form) - Saves the form instance, sets the current object for the view, and redirects to get_success_url().



PAGE NO. 461
DATE :

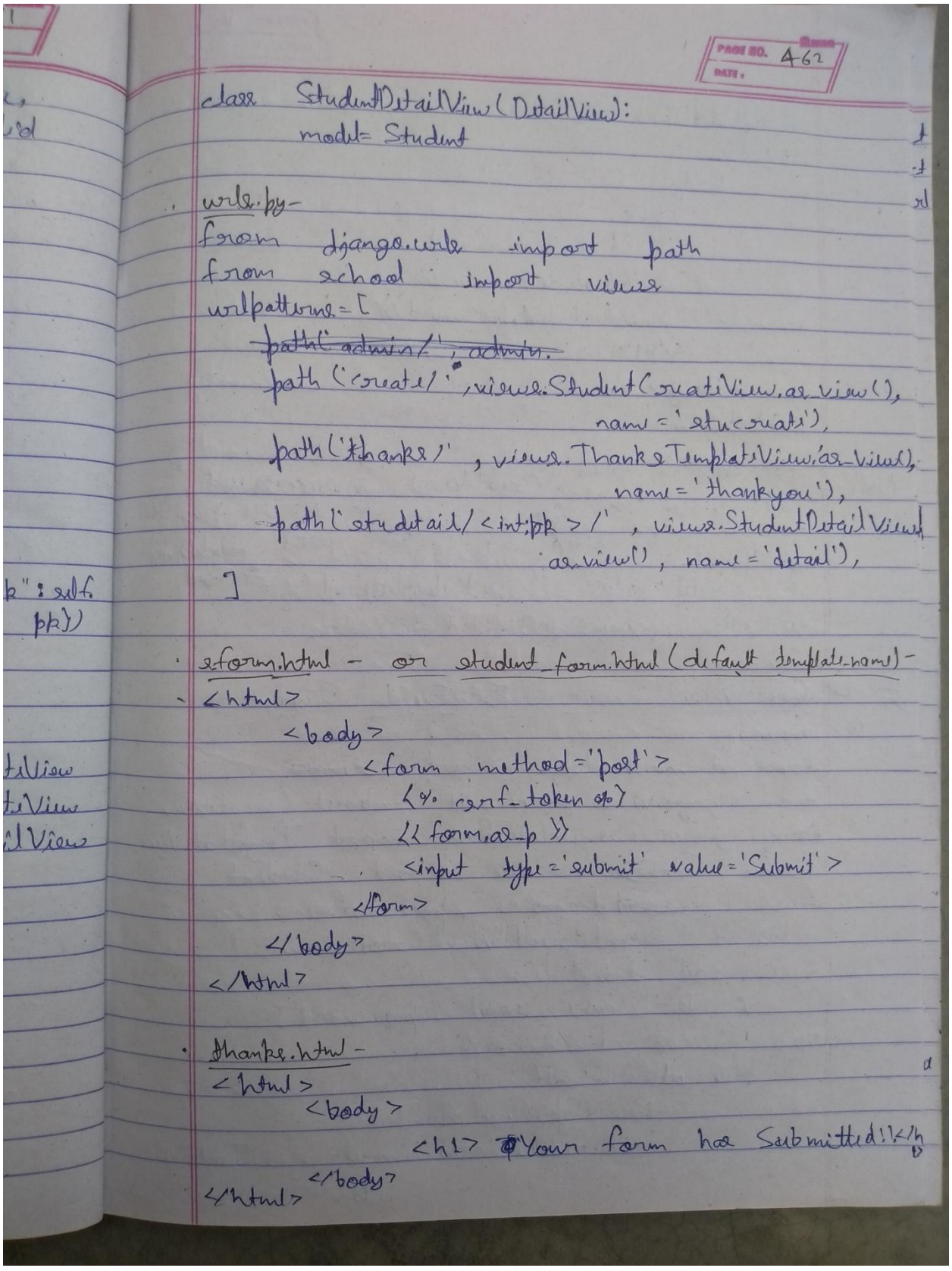
- form_invalid(form):- Renders a response, providing the invalid form as context.

Ex- models.py-

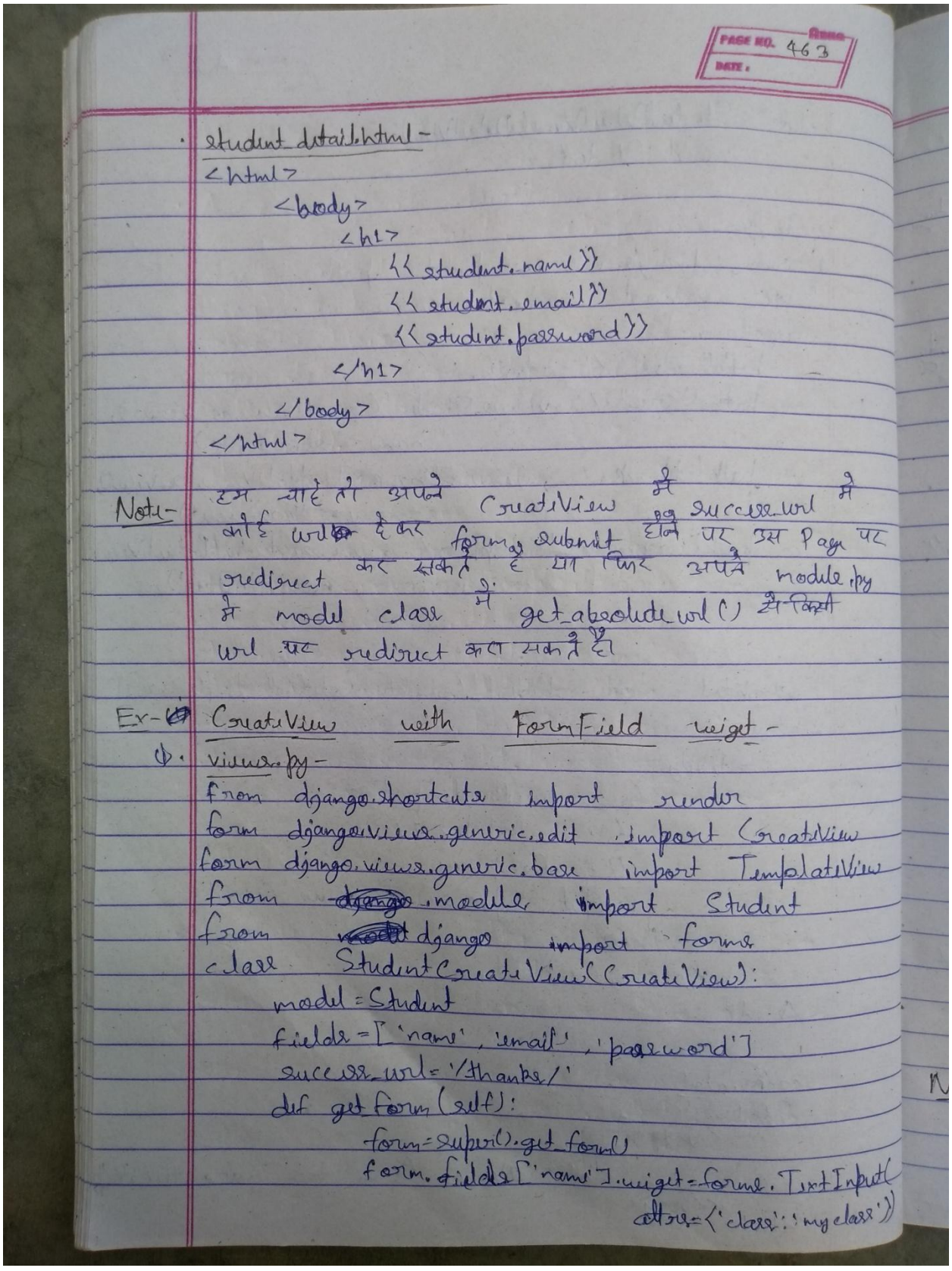
```
from django.db import models
from django.urls import reverse
class Student(models.Model):
    name = models.CharField(max_length=70)
    email = models.EmailField()
    password = models.CharField(max_length=70)

    def get_absolute_url(self):
        # return reverse("thankyou")
        return reverse("detail", kwargs={"pk": self.pk})
```

- views.py-

```
from django.shortcuts import render
from django.views.generic.edit import CreateView
from django.views.generic.base import TemplateView
from django.views.generic.detail import DetailView
from .model import Student
class StudentCreateView(CreateView):
    model = Student
    fields = ['name', 'email', 'password']
    # success_url = '/create/'
    # success_url = '/thanks/'
    template_name = 'school/form.html'
class ThanksTemplateView(TemplateView):
    template_name = 'school/thanks.html'
```



```
class StudentDetailView(DetailView):
    model = Student

. urls.py -
from django.urls import path
from school import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('create/', views.StudentCreateView.as_view(),
                    name='stucreate'),
    path('thanks/', views.ThankTemplateView.as_view(),
                    name='thankyou'),
    path('studetail/<int:pk>/', views.StudentDetailView
                    .as_view(), name='detail'),
]

. sform.html - or student_form.html (default template name) -
<html>
    <body>
        <form method='post'>
            {% csrf_token %}
            {{ form.as_p }}
            <input type='submit' value='Submit'>
        </form>
    </body>
</html>

. thanks.html -
<html>
    <body>
        <h1> Your form has Submitted!</h1>
    </body>
</html>
```



PAGE NO. 463
DATE :

student_detail.html -

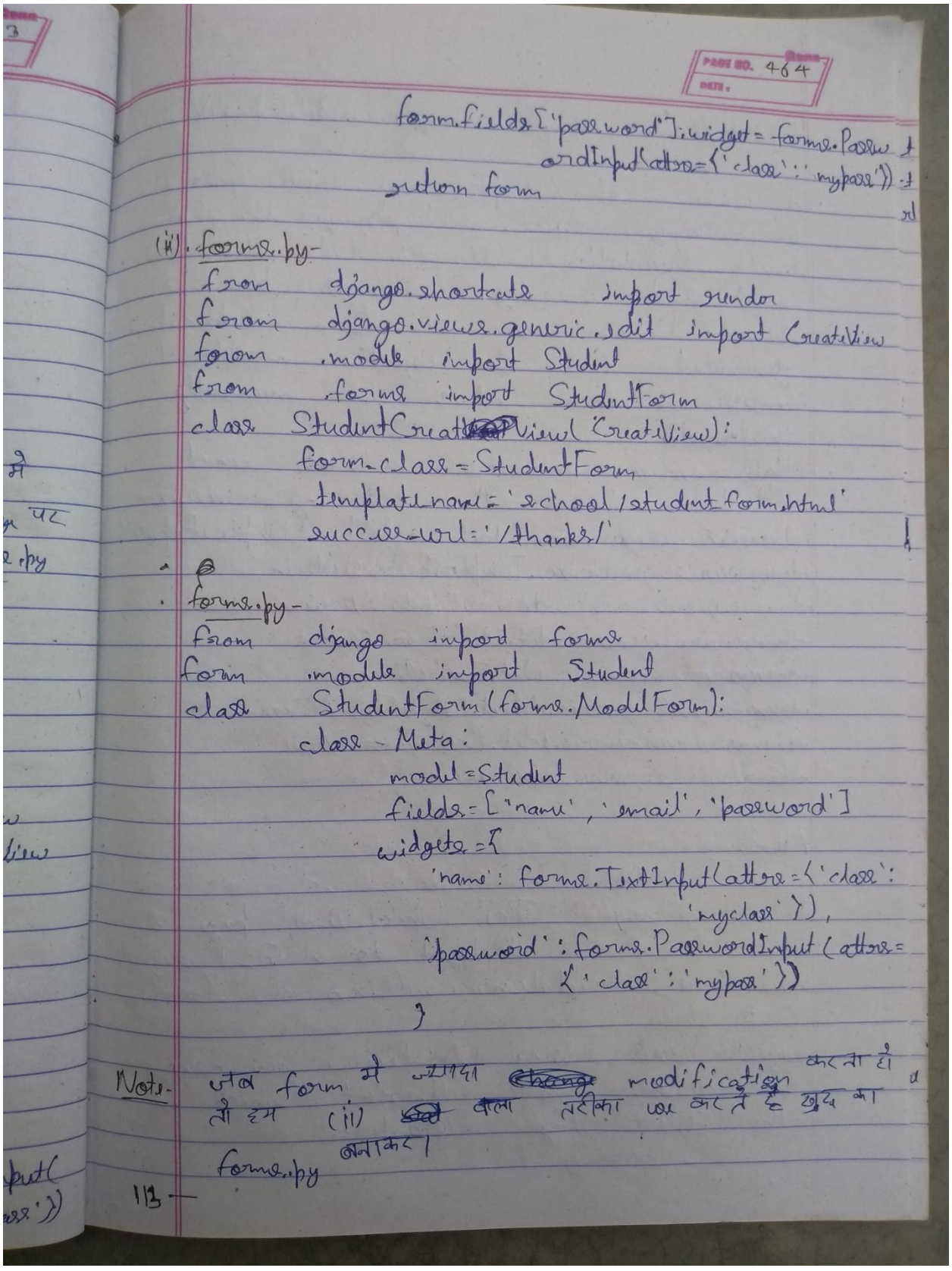
```
<html>
    <body>
        <h1>
            {{ student.name }}
            {{ student.email }}
            {{ student.password }}
        </h1>
    </body>
</html>
```

Note- इस चाहे तो अपने CreateView में success_url में कोई url देकर form submit होने पर उस page पर redirect कर सकते है या फिर अपने module.py में model class में get_absolute_url() से किसी url पर redirect करा सकते है।

Ex-4 CreateView with FormField wiget -

views.py -

```
from django.shortcuts import render
form django.views.generic.edit import CreateView
form django.views.generic.base import TemplateView
from .module import Student
from django import forms
class StudentCreateView(CreateView):
    model = Student
    fields = ['name', 'email', 'password']
    success_url = '/thanks/'
    def get_form(self):
        form = super().get_form()
        form.fields['name'].wiget = forms.TextInput(
            attrs={'class': 'myclass'})
```



form.fields['password'].widget = forms.PasswordInput(attrs={'class':'mypass'})
return form

(ii) forms.py-

```
from django.shortcuts import render
from django.views.generic.edit import CreateView
from .models import Student
from .forms import StudentForm
class StudentCreateView(CreateView):
    form_class = StudentForm
    template_name = 'school/student_form.html'
    success_url = '/thanks/'
```

forms.py-

```
from django import forms
from .models import Student
class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['name', 'email', 'password']
        widgets = {
            'name': forms.TextInput(attrs={'class':
                                    'myclass'}),
            'password': forms.PasswordInput(attrs=
                        {'class':'mypass'})
        }
```

Note- जब form में ज्यादा modification करना हो तो हम (ii) वाला तरीका use करते है सुधि का forms.py बनाकर।

113 —

PAGE NO. 465
DATE : 28-8-20

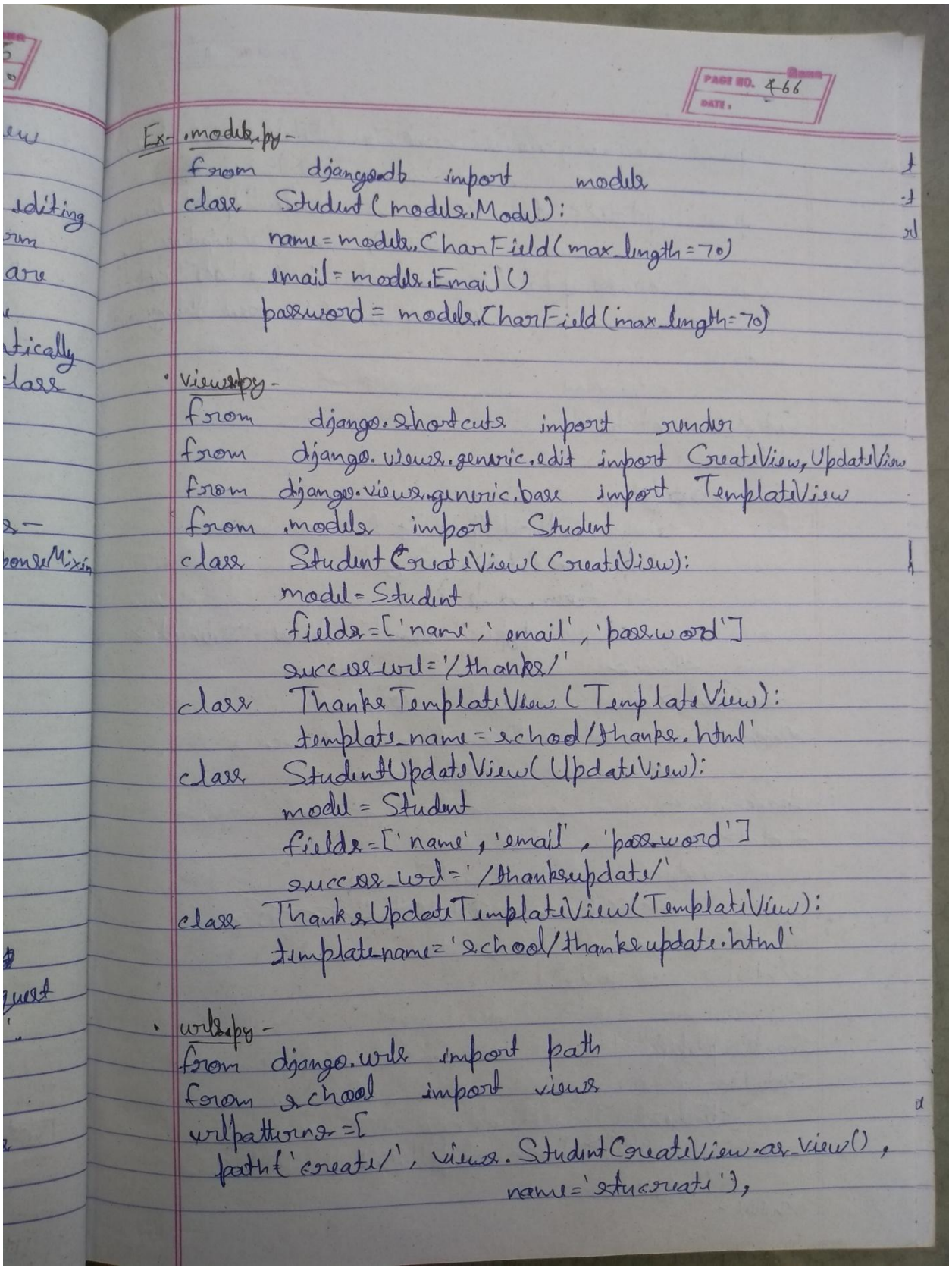
UpdateView - django.views.generic.edit.UpdateView

Ex-

- A view that displays a form for editing an existing object, redisplaying the form with validation errors (if there are any) and saving changes to the object. This uses a form automatically generated from the object's model class (unless a form class is manually specified).
- This view inherits methods and attributes from the following views -
  - django.views.generic.detail.SingleObjectTemplateResponseMixin
  - django.views.generic.base.TemplateResponseMixin
  - django.views.generic.edit.BaseUpdateView
  - django.views.generic.edit.ModelFormMixin
  - django.views.generic.edit.FormMixin
  - django.views.generic.detail.SingleObjectMixin
  - django.views.generic.edit.ProcessFormView
  - django.views.generic.base.View

Attributes -

- template_name_suffix - The UpdateView page displayed to a GET request uses a template_name_suffix of '_form'.
- object - When using UpdateView you have access to self.object, which is the object being updated.



Ex- models.py -

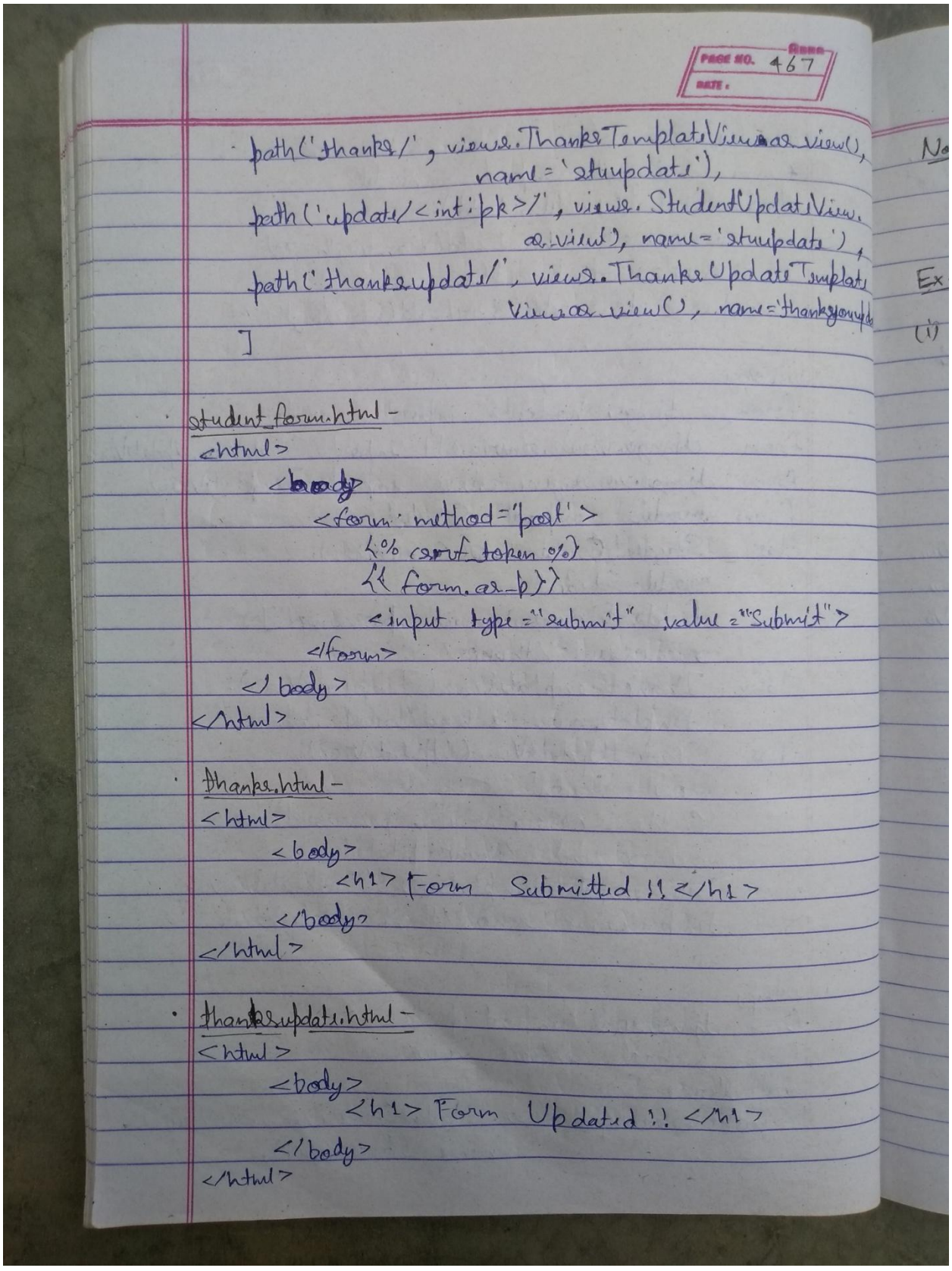
```
from djangodb import models
class Student(models.Model):
    name = models.CharField(max_length=70)
    email = models.Email()
    password = models.CharField(max_length=70)
```

- views.py -

```
from django.shortcuts import render
from django.views.generic.edit import CreateView, UpdateView
from django.views.generic.base import TemplateView
from .models import Student
class StudentCreateView(CreateView):
    model = Student
    fields = ['name', 'email', 'password']
    success_url = '/thanks/'
class ThanksTemplateView(TemplateView):
    template_name = 'school/thanks.html'
class StudentUpdateView(UpdateView):
    model = Student
    fields = ['name', 'email', 'password']
    success_url = '/thanksupdate/'
class ThanksUpdateTemplateView(TemplateView):
    template_name = 'school/thanksupdate.html'
```

- urls.py -

```
from django.urls import path
from school import views
urlpatterns = [
  path('create/', views.StudentCreateView.as_view(),
                  name='stucreate'),
```



PAGE NO. 467
DATE :

```
path('thanks/', views.ThanksTemplateView.as_view(),
                name='stupdate'),
path('update/<int:pk>/', views.StudentUpdateView.
                as_view(), name='stupdate'),
path('thanksupdate/', views.ThanksUpdateTemplate
                View.as_view(), name='thankyouupd
]
```

student_form.html -

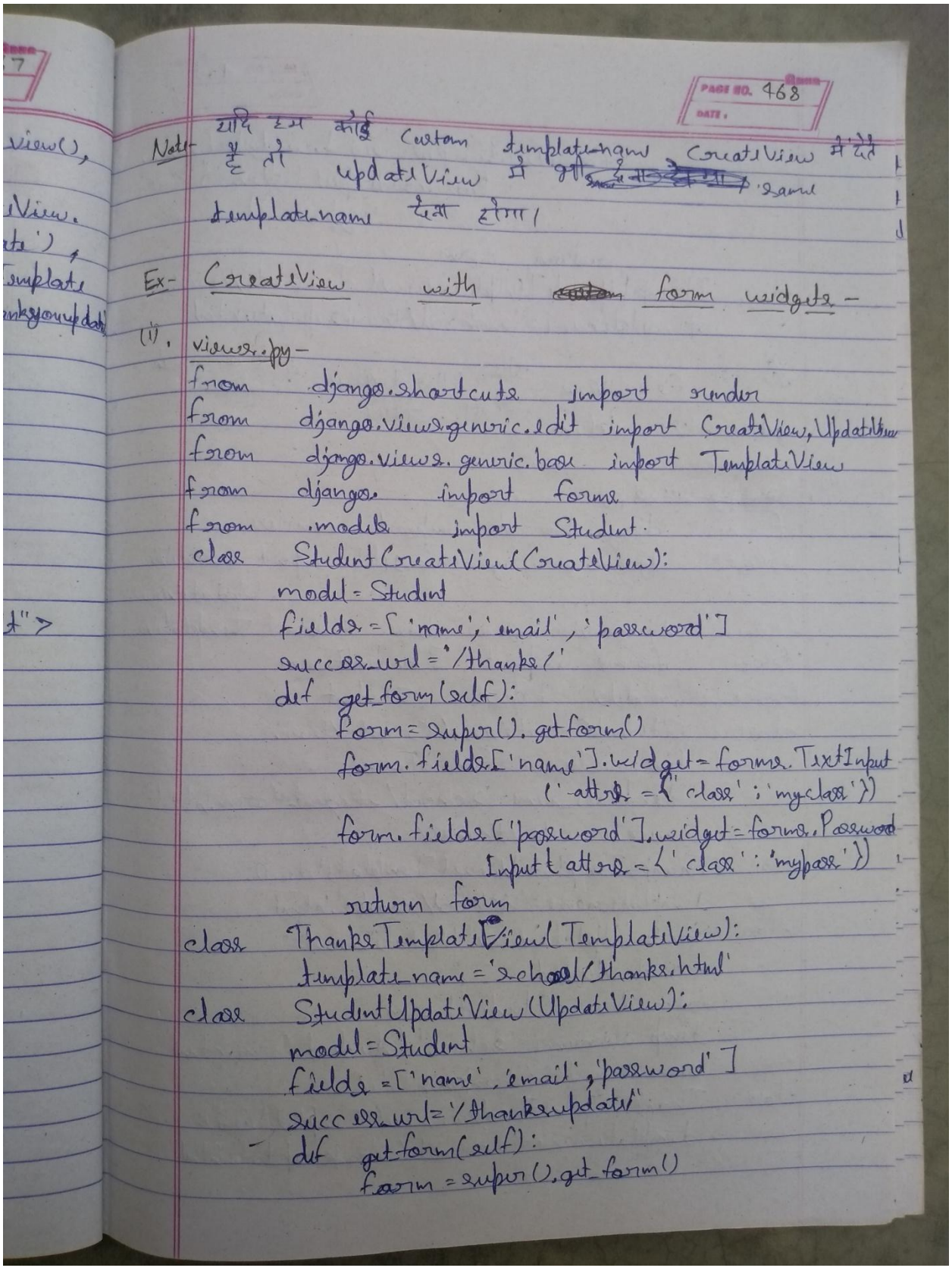
```
<html>
    <body>
        <form method='post'>
            {% csrf_token %}
            {{ form.as_p }}
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

thanks.html -

```
<html>
    <body>
        <h1> Form Submitted !! </h1>
    </body>
</html>
```

thanksupdate.html -

```
<html>
    <body>
        <h1> Form Updated !! </h1>
    </body>
</html>
```

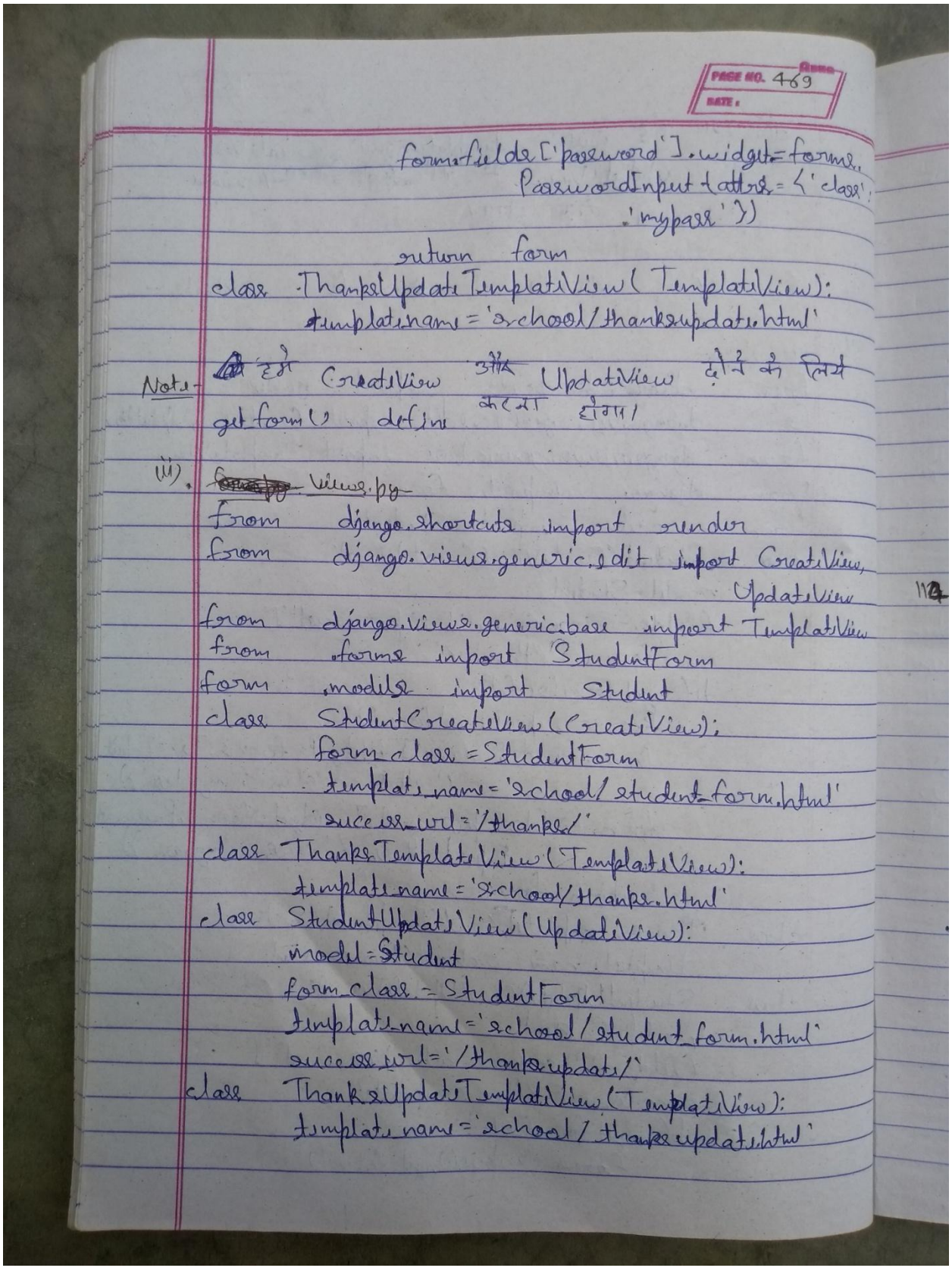


Note- यदि हम कोई custom template name CreateView में देते है तो UpdateView में भी वही same template_name देना होगा।

Ex- CreateView with form widgets -

(i). views.py -

```
from django.shortcuts import render
from django.views.generic.edit import CreateView, UpdateView
from django.views.generic.base import TemplateView
from django. import forms
from .models import Student.
class StudentCreateView(CreateView):
    model = Student
    fields = ['name', 'email', 'password']
    success_url = '/thanks/'
    def get_form(self):
        form = super().get_form()
        form.fields['name'].widget = forms.TextInput
                    ('attrs' = {'class': 'myclass'})
        form.fields['password'].widget = forms.Password
                    Input(attrs = {'class': 'mypass'})
        return form
class ThanksTemplateView(TemplateView):
    template_name = 'school/thanks.html'
class StudentUpdateView(UpdateView):
    model = Student
    fields = ['name', 'email', 'password']
    success_url = '/thanksupdate'
    def get_form(self):
        form = super().get_form()
```



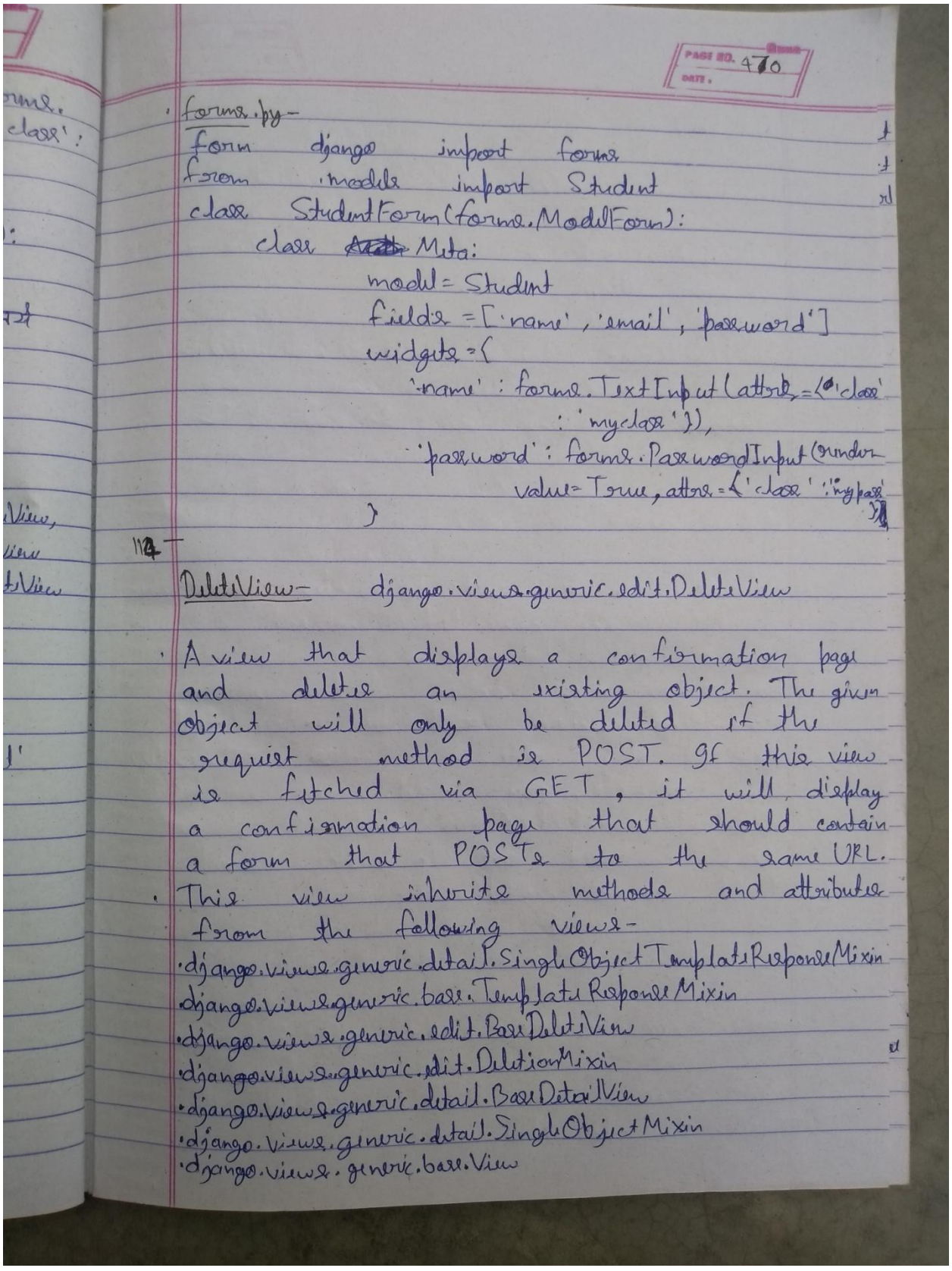
PAGE NO. 469
DATE :

```
        form.fields['password'].widget=forms.
                PasswordInput(attrs={'class':
                    'mypass'})
        return form
class ThanksUpdateTemplateView(TemplateView):
    template_name='school/thanksupdate.html'
```

Note:- हमें CreateView और UpdateView दोनों के लिये get_form() define करना होगा।

(iii). views.py

```
from django.shortcuts import render
from django.views.generic.edit import CreateView,
                                      UpdateView
from django.views.generic.base import TemplateView
from .forms import StudentForm
from .models import Student
class StudentCreateView(CreateView):
    form_class=StudentForm
    template_name='school/student_form.html'
    success_url='/thanks/'
class ThanksTemplateView(TemplateView):
    template_name='school/thanks.html'
class StudentUpdateView(UpdateView):
    model=Student
    form_class=StudentForm
    template_name='school/student_form.html'
    success_url='/thanksupdate/'
class ThanksUpdateTemplateView(TemplateView):
    template_name='school/thanksupdate.html'
```

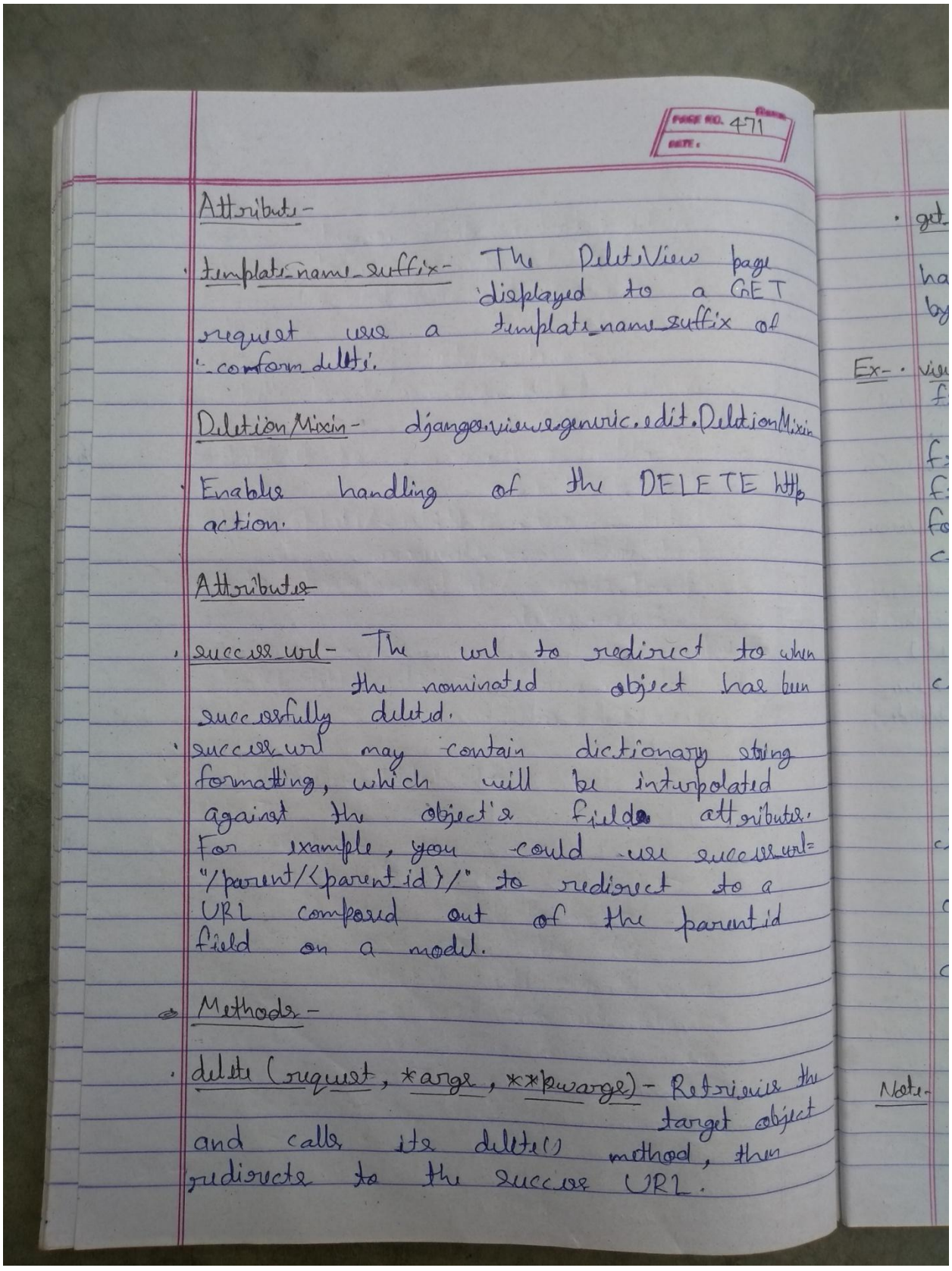


forms.py -

```
form django import forms
from .models import Student
class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        fields = ['name', 'email', 'password']
        widgets = {
            'name' : forms.TextInput(attrs={'class'
                : 'myclass'}),
            'password' : forms.PasswordInput(render_
                value=True, attrs={'class' : 'mypass'
        }
```

DeleteView - django.views.generic.edit.DeleteView

- A view that displays a confirmation page and deletes an existing object. The given object will only be deleted if the request method is POST. If this view is fetched via GET, it will display a confirmation page that should contain a form that POSTs to the same URL.
- This view inherits methods and attributes from the following views -
  - django.views.generic.detail.SingleObjectTemplateResponseMixin
  - django.views.generic.base.TemplateResponseMixin
  - django.views.generic.edit.BaseDeleteView
  - django.views.generic.edit.DeletionMixin
  - django.views.generic.detail.BaseDetailView
  - django.views.generic.detail.SingleObjectMixin
  - django.views.generic.base.View



PAGE NO. 471
DATE :

Attributes -

- template_name_suffix - The DeleteView page displayed to a GET request uses a template_name_suffix of '_confirm_delete'.

DeletionMixin - django.views.generic.edit.DeletionMixin

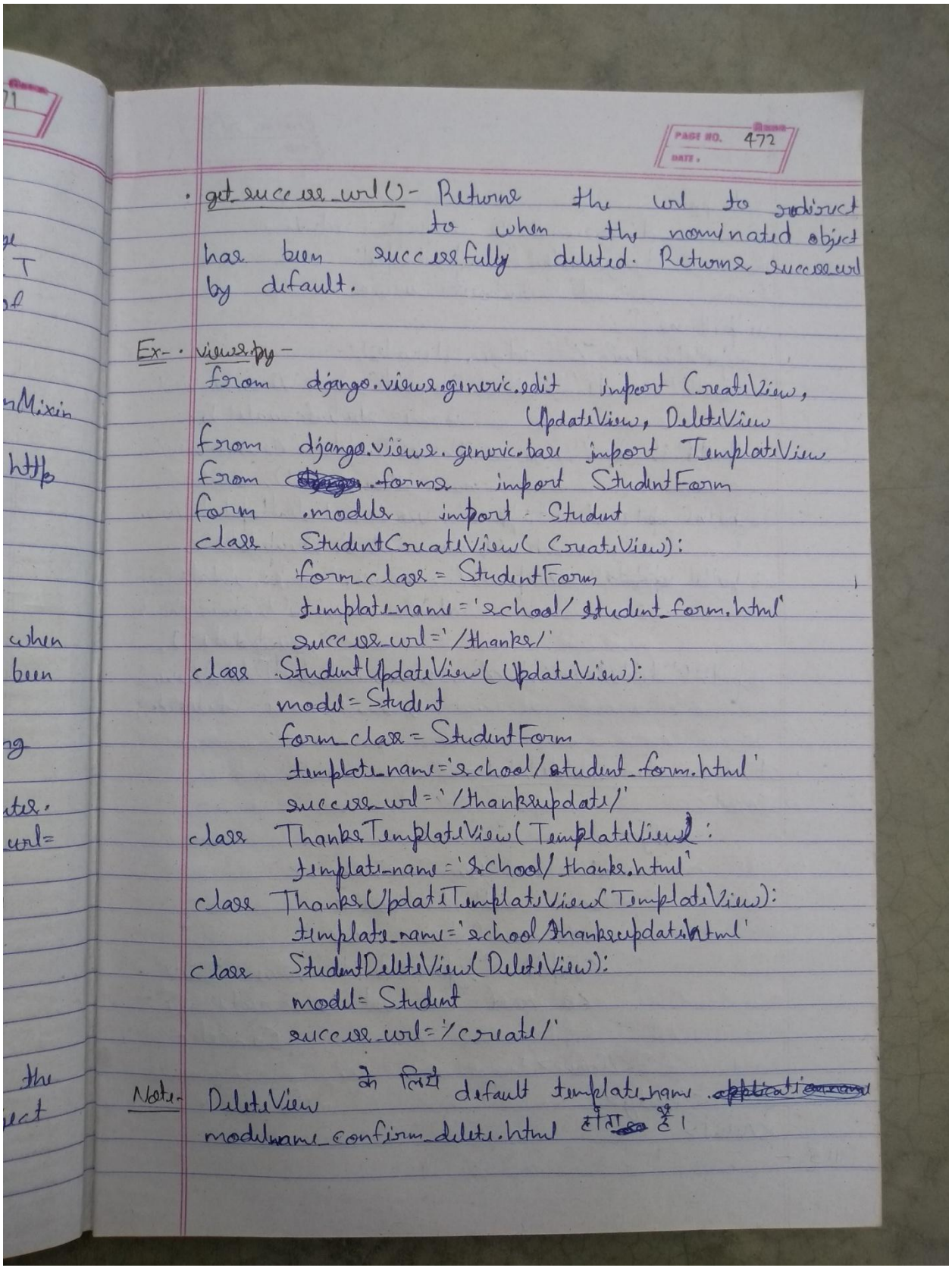
- Enables handling of the DELETE http action.

Attributes

- success_url - The url to redirect to when the nominated object has been successfully deleted.
- success_url may contain dictionary string formatting, which will be interpolated against the object's field attributes. For example, you could use success_url="/parent/{parent_id}/" to redirect to a URL composed out of the parent_id field on a model.

Methods -

- delete(request, *args, **kwargs) - Retrieves the target object and calls its delete() method, then redirects to the success URL.



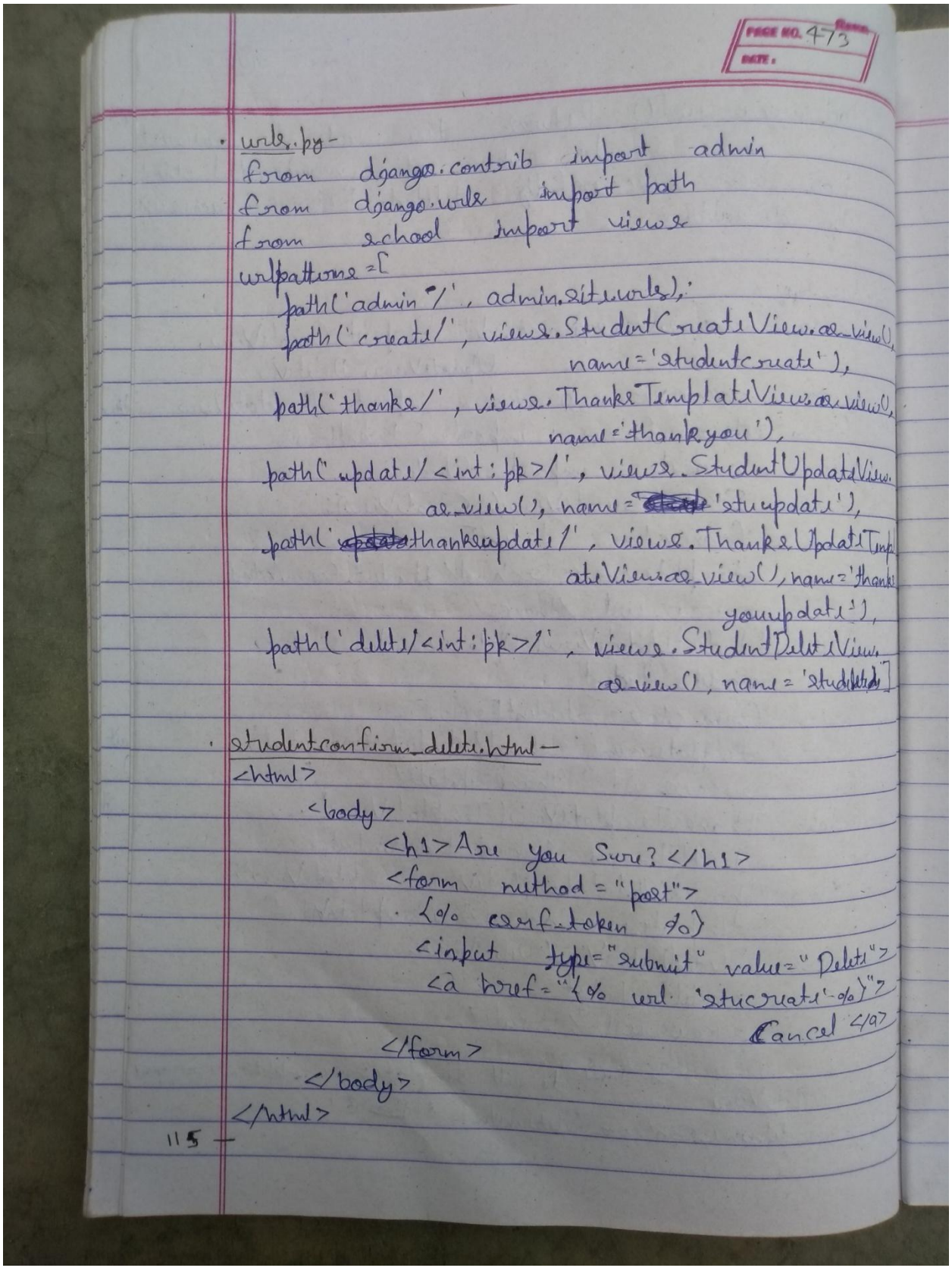
PAGE NO. 472
DATE :

- get_success_url() - Returns the url to redirect to when the nominated object has been successfully deleted. Returns success_url by default.

Ex- · views.py -

```
from django.views.generic.edit import CreateView,
                                      UpdateView, DeleteView
from django.views.generic.base import TemplateView
from .forms import StudentForm
form .models import Student
class StudentCreateView(CreateView):
        form_class = StudentForm
        template_name = 'school/student_form.html'
        success_url = '/thanks/'
class StudentUpdateView(UpdateView):
        model = Student
        form_class = StudentForm
        template_name = 'school/student_form.html'
        success_url = '/thanksupdate/'
class ThanksTemplateView(TemplateView):
        template_name = 'school/thanks.html'
class ThanksUpdateTemplateView(TemplateView):
        template_name = 'school/thanksupdate.html'
class StudentDeleteView(DeleteView):
        model = Student
        success_url = '/create/'
```

Note- DeleteView के लिये default template_name modulname_confirm_delete.html होता है।



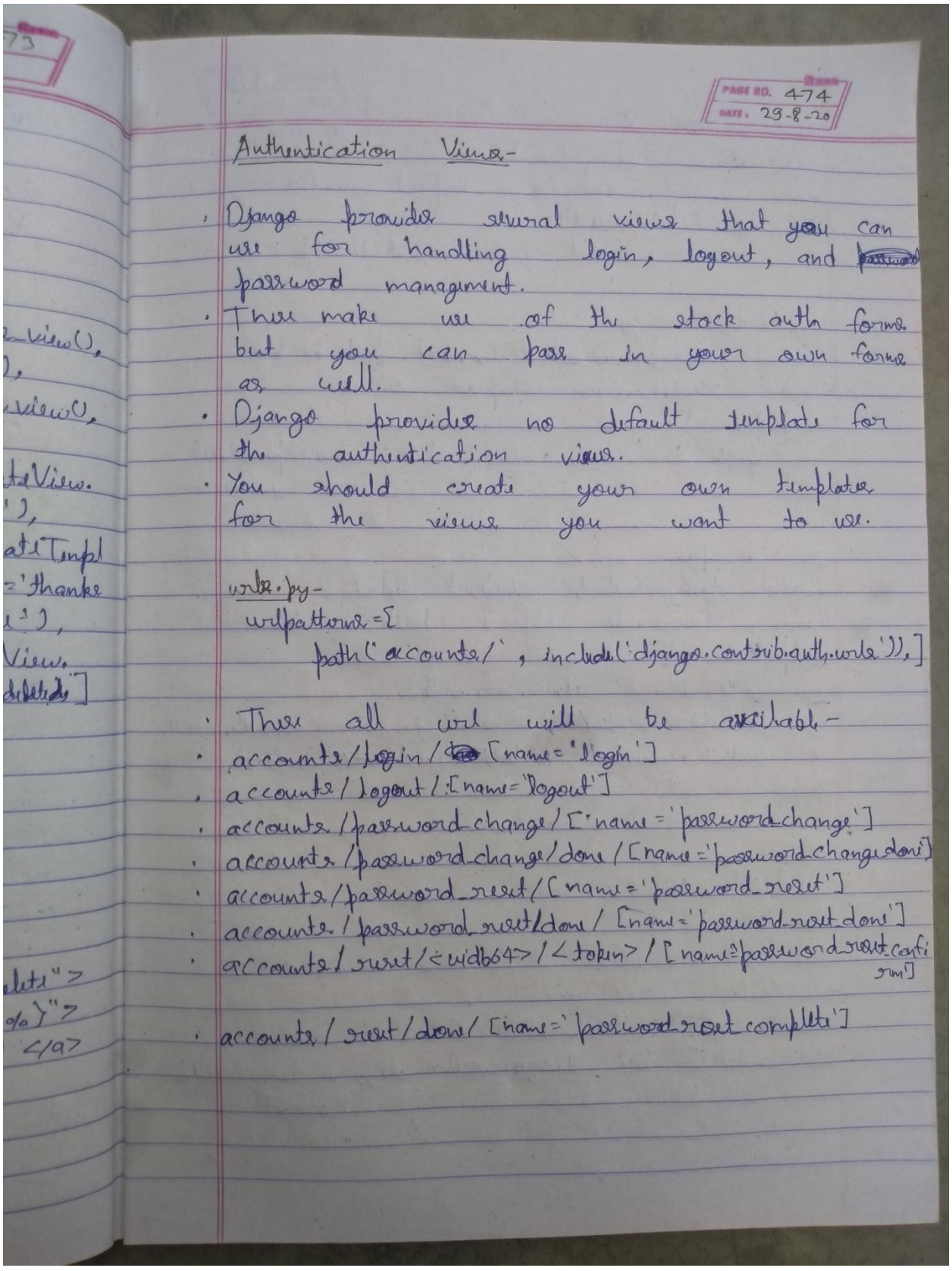
- urls.py -

```
from django.contrib import admin
from django.urls import path
from school import views
urlpatterns =[
    path('admin/', admin.site.urls),
    path('create/', views.StudentCreateView.as_view(),
                    name='studentcreate'),
    path('thanks/', views.ThanksTemplateView.as_view(),
                    name='thankyou'),
    path('update/<int:pk>/', views.StudentUpdateView.
        as_view(), name='studupdate'),
    path('thanksupdate/', views.ThanksUpdateTemp
        ateView.as_view(), name='thanks
                    youupdate'),
    path('delete/<int:pk>/', views.StudentDeleteView.
                    as_view(), name='studdelete')]
```

- student_confirm_delete.html -

```
<html>
    <body>
        <h1>Are you Sure?</h1>
        <form method="post">
        {% csrf_token %}
        <input type="submit" value="Delete">
        <a href="{% url 'studcreate' %}">
                    Cancel </a>
        </form>
    </body>
</html>
```

115 —



PAGE NO. 474
DATE : 29-8-20

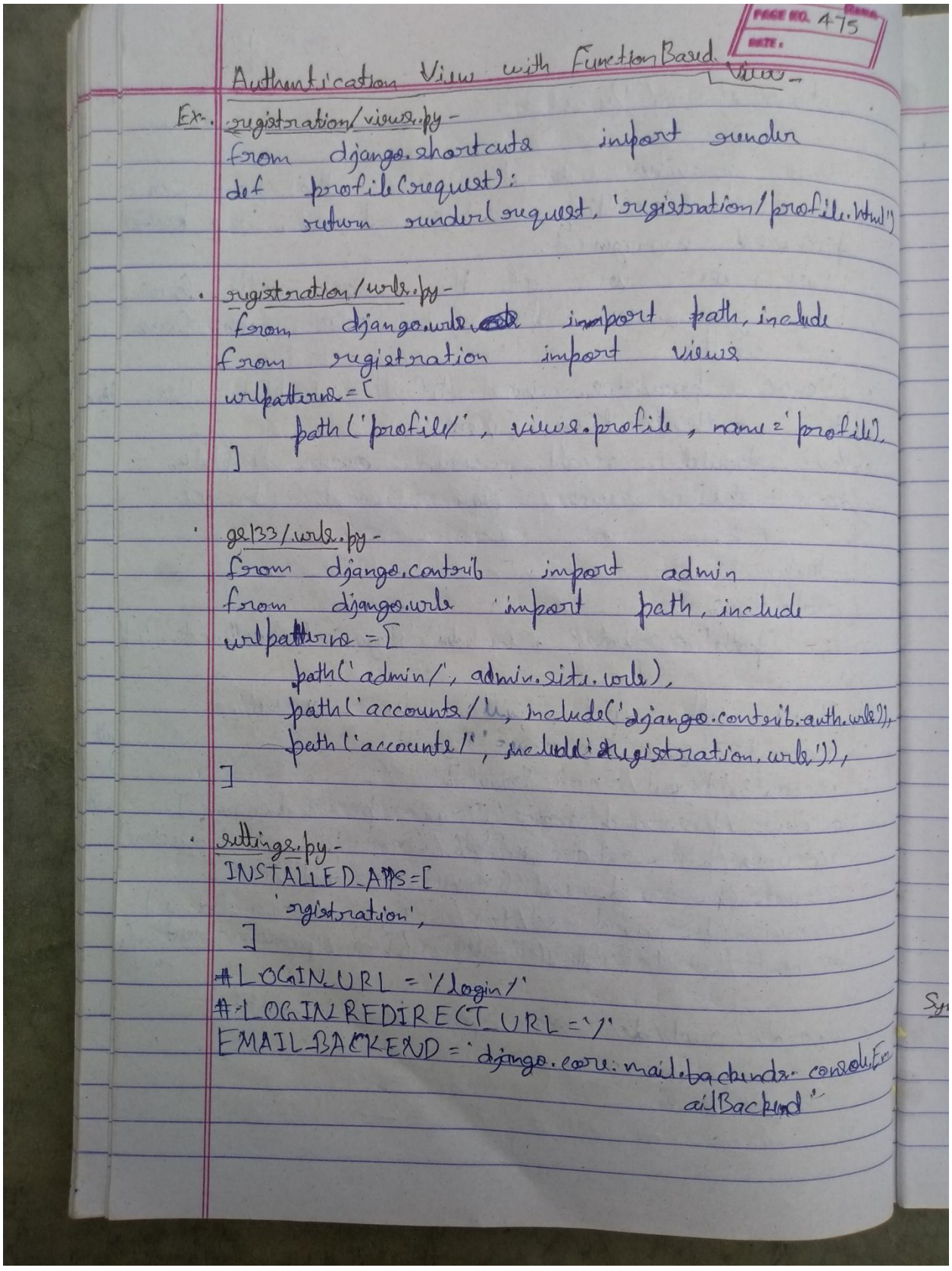
Authentication Views-

- Django provides several views that you can use for handling login, logout, and password management.
- These make use of the stock auth forms but you can pass in your own forms as well.
- Django provides no default template for the authentication views.
- You should create your own templates for the views you want to use.

urls.py-

```
urlpatterns=[
    path('accounts/', include('django.contrib.auth.urls')),]
```

- These all url will be available-
- accounts/login/ [name='login']
- accounts/logout/ [name='logout']
- accounts/password_change/ [name='password_change']
- accounts/password_change/done/ [name='password_change_done']
- accounts/password_reset/ [name='password_reset']
- accounts/password_reset/done/ [name='password_reset_done']
- accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']
- accounts/reset/done/ [name='password_reset_complete']



PAGE NO. 475
DATE :

Authentication View with Function Based View -

Ex. registration/views.py -

```
from django.shortcuts import render
def profile(request):
    return render(request, 'registration/profile.html')
```

- registration/urls.py -

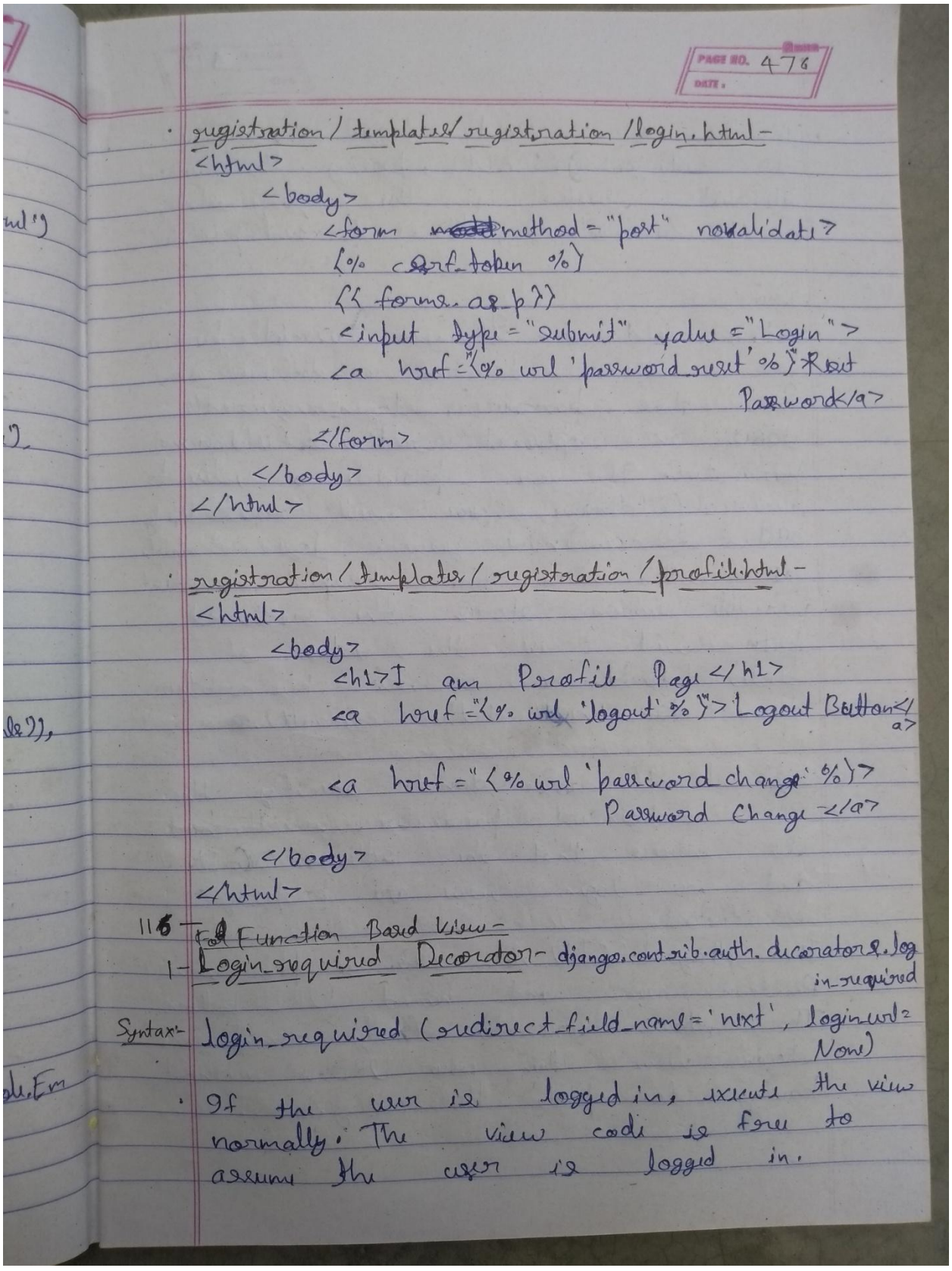
```
from django.urls import path, include
from registration import views
urlpatterns = [
    path('profile/', views.profile, name='profile'),
]
```

- ge133/urls.py -

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('django.contrib.auth.urls')),
    path('accounts/', include('registration.urls')),
]
```

- settings.py -

```
INSTALLED_APPS=[
    'registration',
]
#LOGIN_URL = '/login/'
#LOGIN_REDIRECT_URL = '/'
EMAIL_BACKEND = 'django.core.mail.backends.consoleEmailBackend'
```



PAGE NO. 476
DATE :

· registration / templates / registration / login.html -

```
<html>
    <body>
        <form method="post" novalidate>
        {% csrf_token %}
        {{ form.as_p }}
        <input type="submit" value="Login">
        <a href="{% url 'password_reset' %}">Reset
                                    Password</a>
        </form>
    </body>
</html>
```

· registration / templates / registration / profile.html -

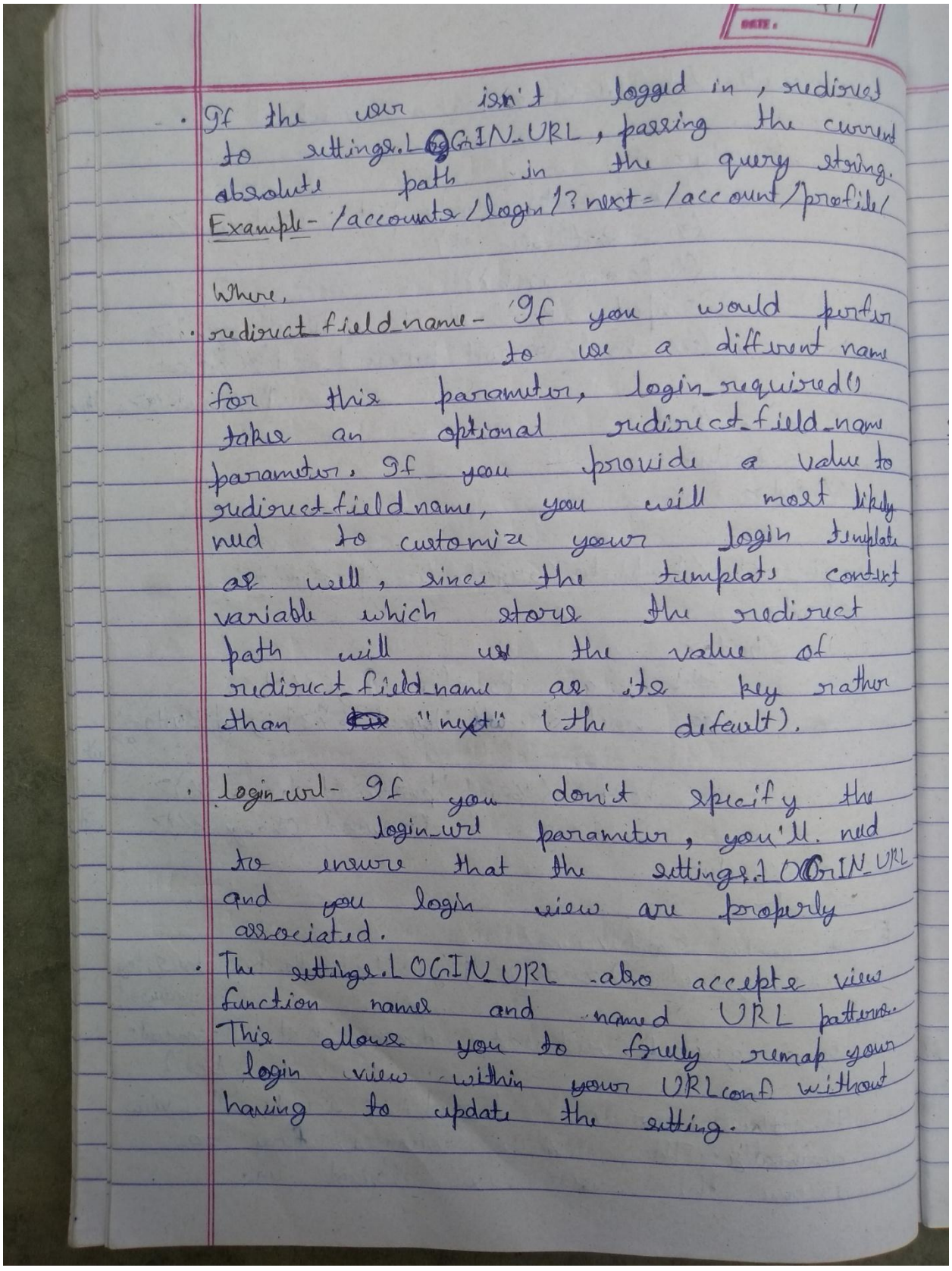
```
<html>
    <body>
        <h1>I am Profile Page</h1>
        <a href="{% url 'logout' %}">Logout Button</
                                    a>

        <a href="{% url 'password_change' %}">
                        Password Change</a>

    </body>
</html>
```

11 6 — For Function Based View -

1- Login_required Decorator - django.contrib.auth.decorators.login_required

Syntax - login_required (redirect_field_name='next', login_url=None)

· If the user is logged in, execute the view normally. The view code is free to assume the user is logged in.



- If the user isn't logged in, redirect to settings.LOGIN_URL, passing the current absolute path in the query string.
Example- /accounts/login/?next=/account/profile/

Where,

- redirect_field_name- If you would prefer to use a different name for this parameter, login_required() takes an optional redirect_field_name parameter. If you provide a value to redirect_field_name, you will most likely need to customize your login template as well, since the template context variable which stores the redirect path will use the value of redirect_field_name as its key rather than "next" (the default).

- login_url- If you don't specify the login_url parameter, you'll need to ensure that the settings.LOGIN_URL and your login view are properly associated.
- The settings.LOGIN_URL also accepts view function names and named URL patterns. This allows you to freely remap your login view within your URLconf without having to update the setting.

PAGE NO. 478
DATE :

Ex- views.py-
```
from django.contrib.auth.decorators import login_required

@login_required
def profile(request):
    return render(request, 'registration/profile.html')
```

2- staff_member_required decorator-

Syntax- staff_member_required(redirect_field_name='next', login_url='admin:login')

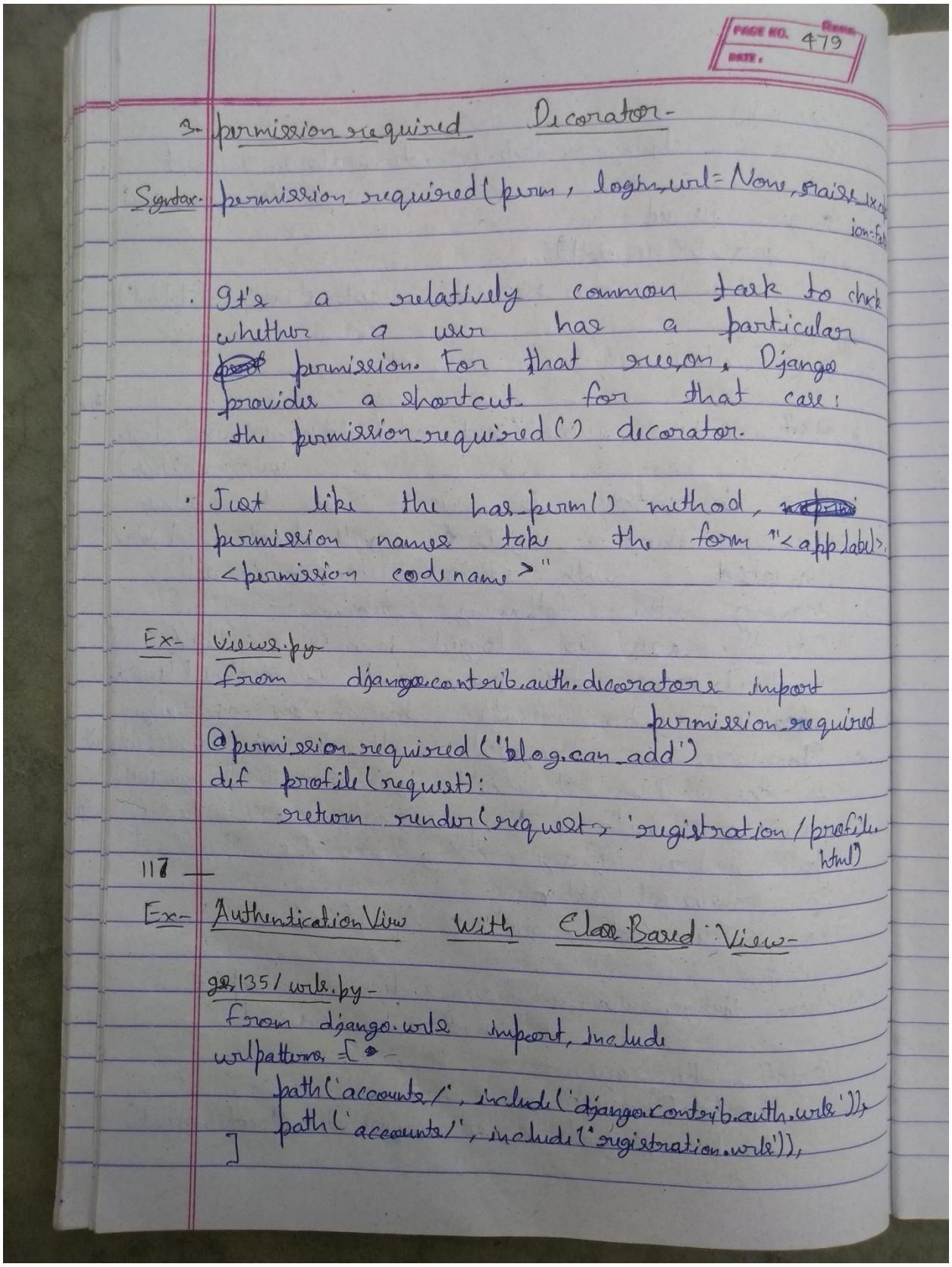
. This decorator is used on the admin views that required authentication. A view decorated with this function will having the following behaviour-

. If the user is logged in, is a staff member (User.is_staff=True), and is active (User.is_active=True), execute the view normally.

. Otherwise, the request will be redirected to the URL specified by login_url parameter, with the originally requested path in a query string variable specified by redirect_field_name.

For Example- /admin/login?next=/profile/

views.py-
Ex-
```
from django.contrib.admin.views.decorators import staff_member_required

@staff_member_required
def profile(request):
    return render(request, 'registration/profile.html')
```



PAGE NO. 479
DATE :

3. permission_required Decorator-

Syntax- permission_required(perm, login_url=None, raise_exception=False)

- It's a relatively common task to check whether a user has a particular permission. For that reason, Django provides a shortcut for that case: the permission_required() decorator.

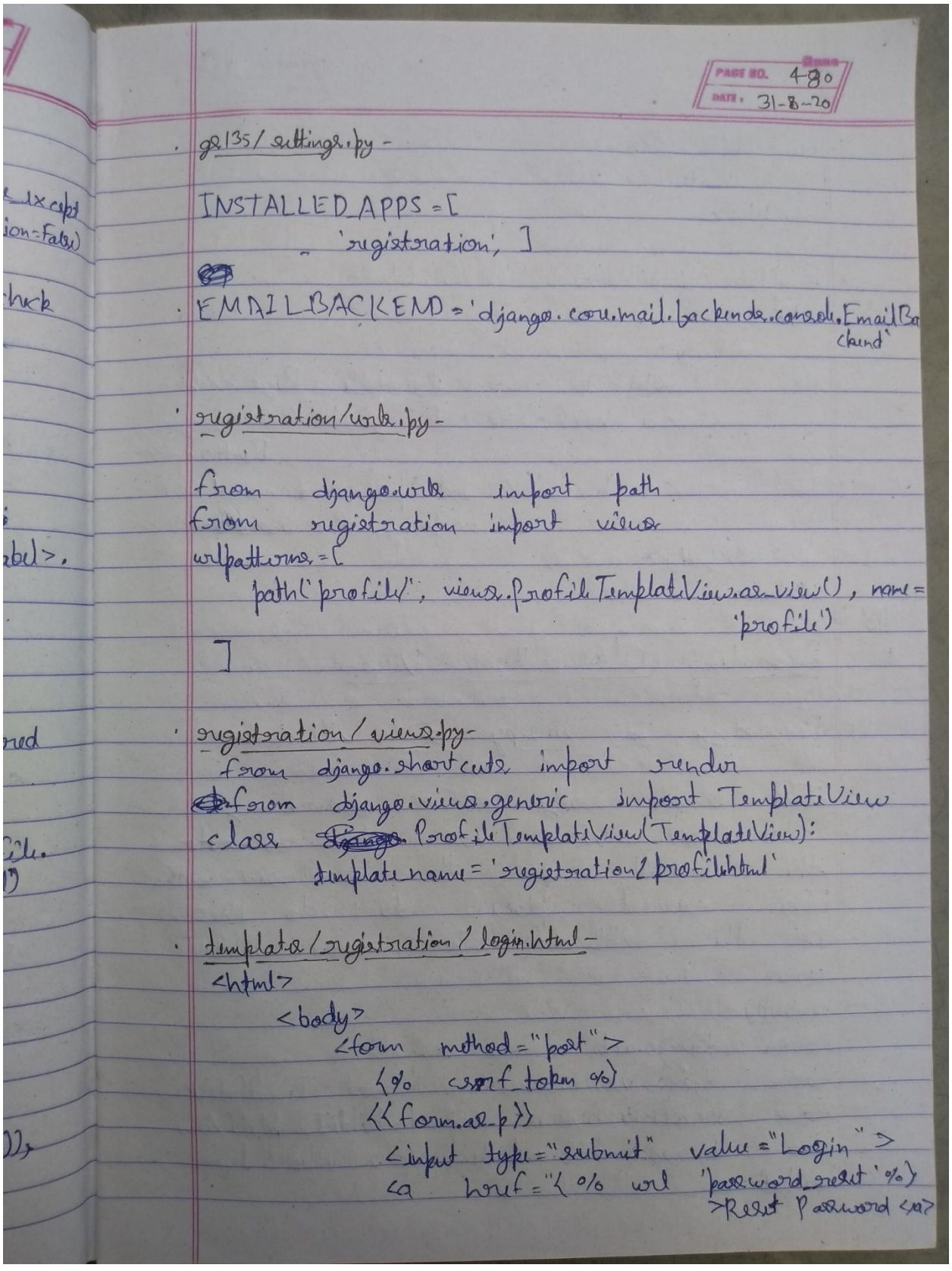
- Just like the has_perm() method, permission names take the form "<app label>.<permission codename>"

Ex- views.py
from django.contrib.auth.decorators import permission_required
@permission_required('blog.can_add')
def profile(request):
    return render(request, 'registration/profile.html')

118 —

Ex- Authentication View With Class Based View-

22/135/urls.py-
from django.urls import path, include
urlpatterns = [
    path('accounts/', include('django.contrib.auth.urls')),
    path('accounts/', include('registration.urls')),
]



PAGE NO. 490
DATE : 31-8-20

- gs135/settings.py -

```
INSTALLED_APPS = [
        'registration', ]
```

- EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'

- registration/urls.py -

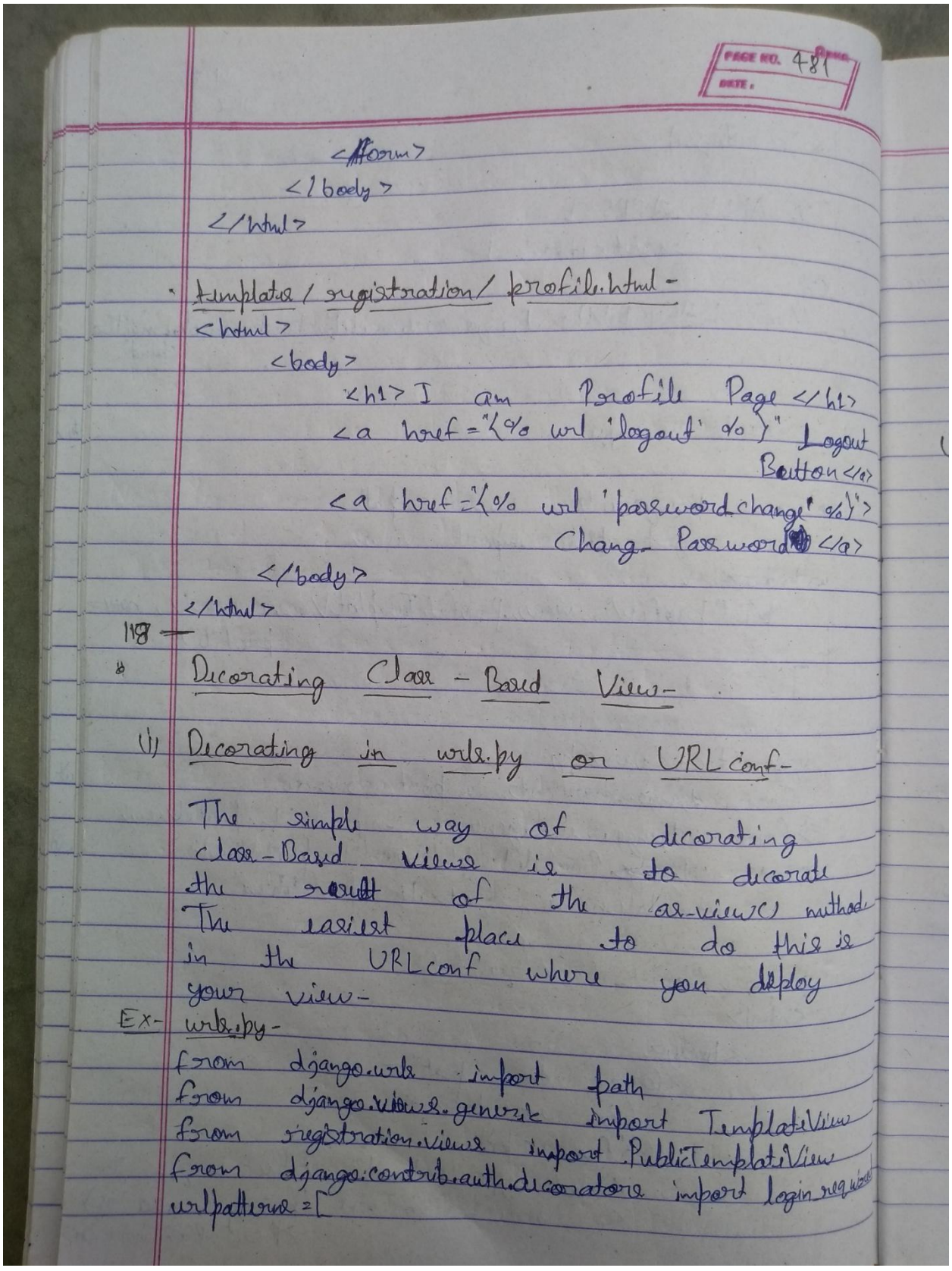
```
from django.urls import path
from registration import views
urlpatterns = [
    path('profile/', views.ProfileTemplateView.as_view(), name='profile')
]
```

- registration/views.py -

```
from django.shortcuts import render
from django.views.generic import TemplateView
class ProfileTemplateView(TemplateView):
    template_name = 'registration/profile.html'
```

- template/registration/login.html -

```
<html>
    <body>
        <form method="post">
            {% csrf_token %}
            {{form.as_p}}
            <input type="submit" value="Login">
            <a href="{% url 'password_reset' %}">Reset Password</a>
```



```
        </form>
    </body>
</html>
```

- templates / registration / profile.html -

```
<html>
    <body>
        <h1> I am Profile Page </h1>
        <a href="{% url 'logout' %}" Logout Button </a>
        <a href="{% url 'password_change' %}"> Change Password </a>
    </body>
</html>
```

118 —

Decorating Class - Based Views -

(i) Decorating in urls.py or URL conf -

The simple way of decorating class-Based views is to decorate the result of the as_view() method. The easiest place to do this is in the URLconf where you deploy your view -

Ex- urls.py -

```
from django.urls import path
from django.views.generic import TemplateView
from registration.views import PublicTemplateView
from django.contrib.auth.decorators import login_required
urlpatterns = [
```

PAGE NO. 482
DATE :

```
path('dashboard/', login_required(TemplateView.as_view(
        template_name='blog/dash.html')), name='dash'),
path('profile/', login_required(ProfileTemplateView.as_vie
        w(template_name='registration/profile.html')
        ), name='profile'),
path('blogpost/', permission_required('blog.can_add')(Blog
        PostView.as_view())),]
```

(ii) method_decorator-

- The method_decorator decorator transforms a function decorator into a method decorator so that it can be used on an instance method.
- A method on a class isn't quit the same as a standalone function, so you can't just apply a function decorator to the method you need to transform it into a method decorator first.

Syntax- @method_decorator(*args, **kwargs)

~~Decorating Class-Based Views~~

- Decorating in the class- To decorate every instance of a class-based view, you need to decorate the class definition itself. To do this you apply the decorator to the dispatch() method of the class.

PAGE NO. 483
DATE :

Ex- write by -

```
from django.utils.decorators import method_decorator

from django.views.generic import TemplateView
class ProfileTemplateView(TemplateView):
    template_name = 'registration/profile.html'
    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super().dispatch(*args, **kwargs)
```

(Tab inside class)

Note- You can decorate the class instead and pass the name of the method to be decorated as the keyword argument name -

Ex-

```
@method_decorator(login_required, name='dispatch')
class ProfileTemplateView(TemplateView):
    template_name = 'registration/profile.html'
```

Note- If you have a set of common decorators used in several places, you can define a list or tuple of decorators and use this instead of invoking method_decorator() multiple times.

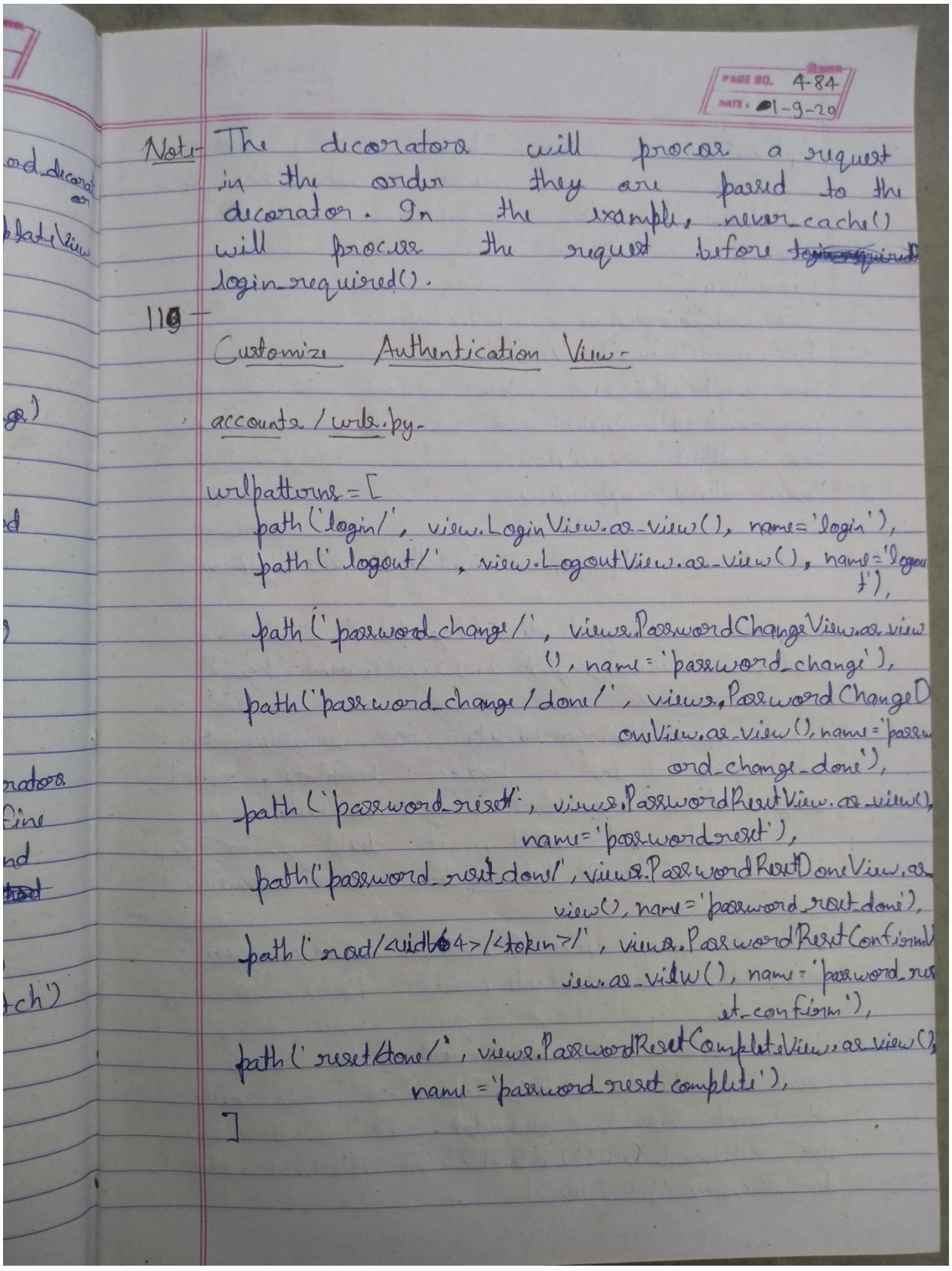
Ex- →

```
@method_decorator(never_cache, name='dispatch')
@method_decorator(login_required, name='dispatch')
class ProfileTemplateView(TemplateView):
    template_name = 'registration/profile.html'
```

Or

→

```
decorators = [never_cache, login_required]
@method_decorator(decorators, name='dispatch')
class ProfileTemplateView(TemplateView):
    template_name = 'registration/profile.html'
```



PAGE NO. 4-84
DATE : 01-9-20

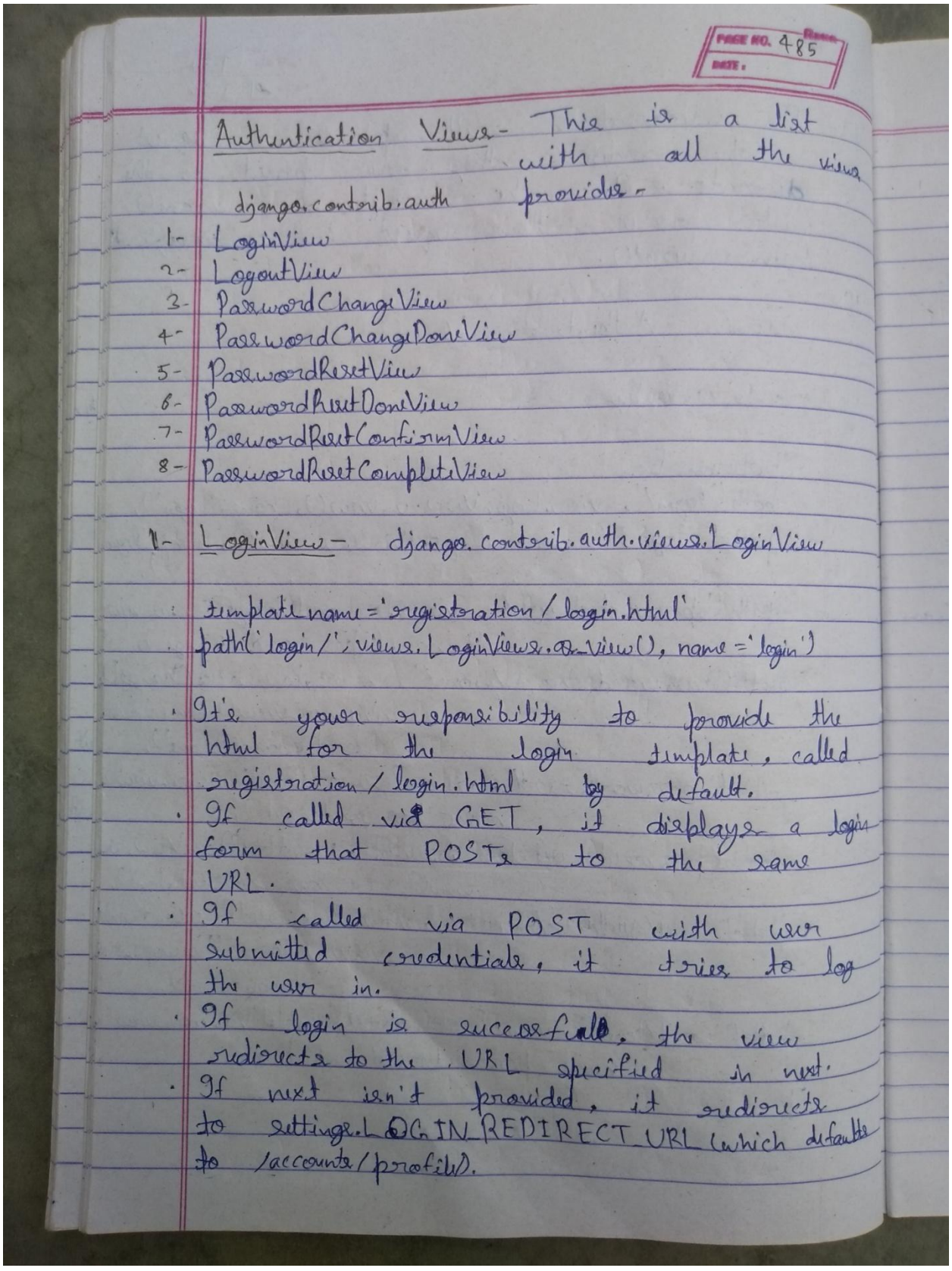
Note- The decorators will procss a request in the order they are passed to the decorator. In the example, never_cache() will process the request before login_required().

110 -

Customize Authentication View -

accounts / urls.py -

```
urlpatterns = [
    path('login/', view.LoginView.as_view(), name='login'),
    path('logout/', view.LogoutView.as_view(), name='logout'),
    path('password_change/', views.PasswordChangeView.as_view(), name='password_change'),
    path('password_change/done/', views.PasswordChangeDoneView.as_view(), name='password_change_done'),
    path('password_reset', views.PasswordResetView.as_view(), name='password_reset'),
    path('password_reset_done', views.PasswordResetDoneView.as_view(), name='password_reset_done'),
    path('reset/<uidb64>/<token>/', views.PasswordResetConfirmView.as_view(), name='password_reset_confirm'),
    path('reset/done/', views.PasswordResetCompleteView.as_view(), name='password_reset_complete'),
]
```



PAGE NO. 485
DATE :

Authentication Views - This is a list with all the views django.contrib.auth provides -

1- LoginView
2- LogoutView
3- PasswordChangeView
4- PasswordChangeDoneView
5- PasswordResetView
6- PasswordResetDoneView
7- PasswordResetConfirmView
8- PasswordResetCompleteView

1- LoginView - django.contrib.auth.views.LoginView

template_name = 'registration/login.html'
path('login/', views.LoginViews.as_view(), name = 'login')

- It's your responsibility to provide the html for the login template, called registration/login.html by default.
- If called via GET, it displays a login form that POSTs to the same URL.
- If called via POST with user submitted credentials, it tries to log the user in.
- If login is successful, the view redirects to the URL specified in next.
- If next isn't provided, it redirects to settings.LOGIN_REDIRECT_URL (which defaults to /accounts/profile).

PAGE NO. 486
DATE :

- If login isn't successful, it redisplays the login form.

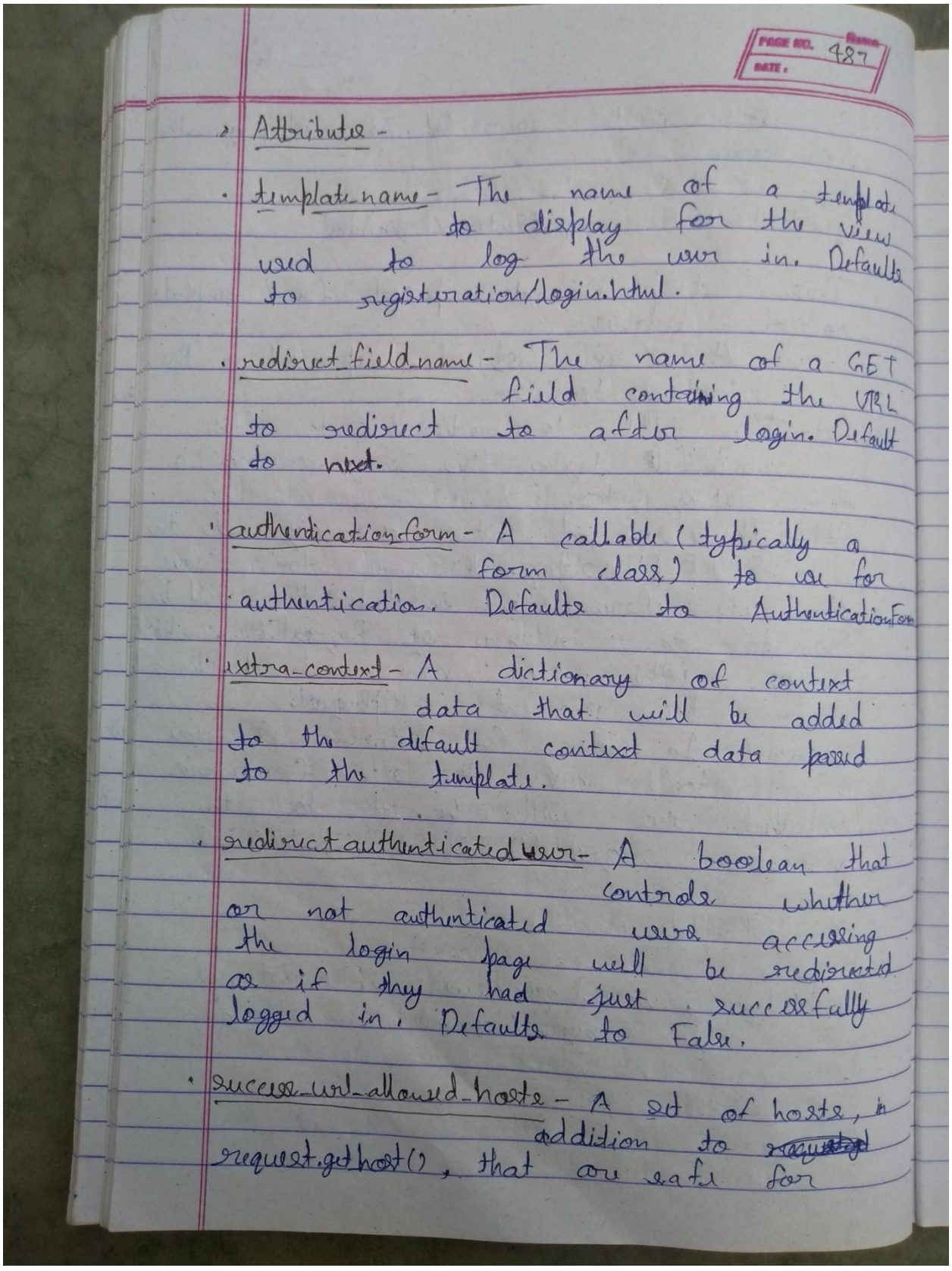
- Default Template- registration/login.html

This template gets passed four template context variables-

- form- A form object representing the AuthenticationForm.
- next- The URL to redirect to, after successful login. This may contain a query string, too.
- site- The current site, according to the SITE_ID setting. If you don't have the site framework installed, this will be set to an instance of RequestSite, which derives the site name and domain from the current HttpRequest.
- site.name- An alias for site.name. If you don't have the site framework installed, this will be set to the value of request.META['SERVER_NAME'].

Custom Template-

```
path('login/', views.LoginViews.view(template_name =
                'myapp/login.html'), name='login')
```



PAGE NO. 487
DATE :

- Attributes -

- template_name - The name of a template to display for the view used to log the user in. Defaults to registration/login.html.

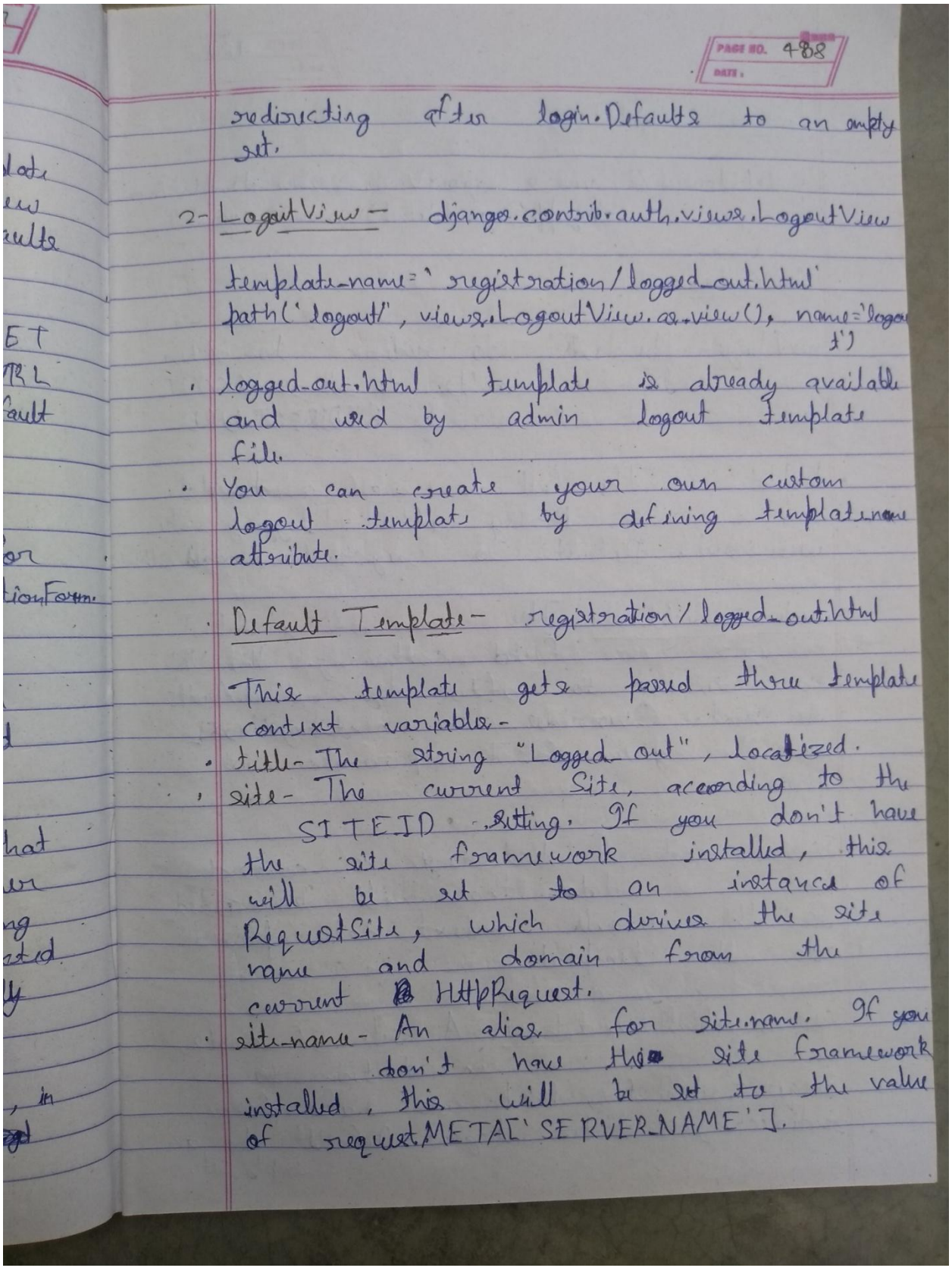
- redirect_field_name - The name of a GET field containing the URL to redirect to after login. Default to next.

- authentication_form - A callable (typically a form class) to use for authentication. Defaults to AuthenticationForm

- extra_context - A dictionary of context data that will be added to the default context data passed to the template.

- redirect_authenticated_user - A boolean that controls whether or not authenticated users accessing the login page will be redirected as if they had just successfully logged in. Defaults to False.

- success_url_allowed_hosts - A set of hosts, in addition to ~~request~~ request.get_host(), that are safe for



PAGE NO. 488
DATE :

redirecting after login. Defaults to an empty set.

2- LogoutView - django.contrib.auth.views.LogoutView

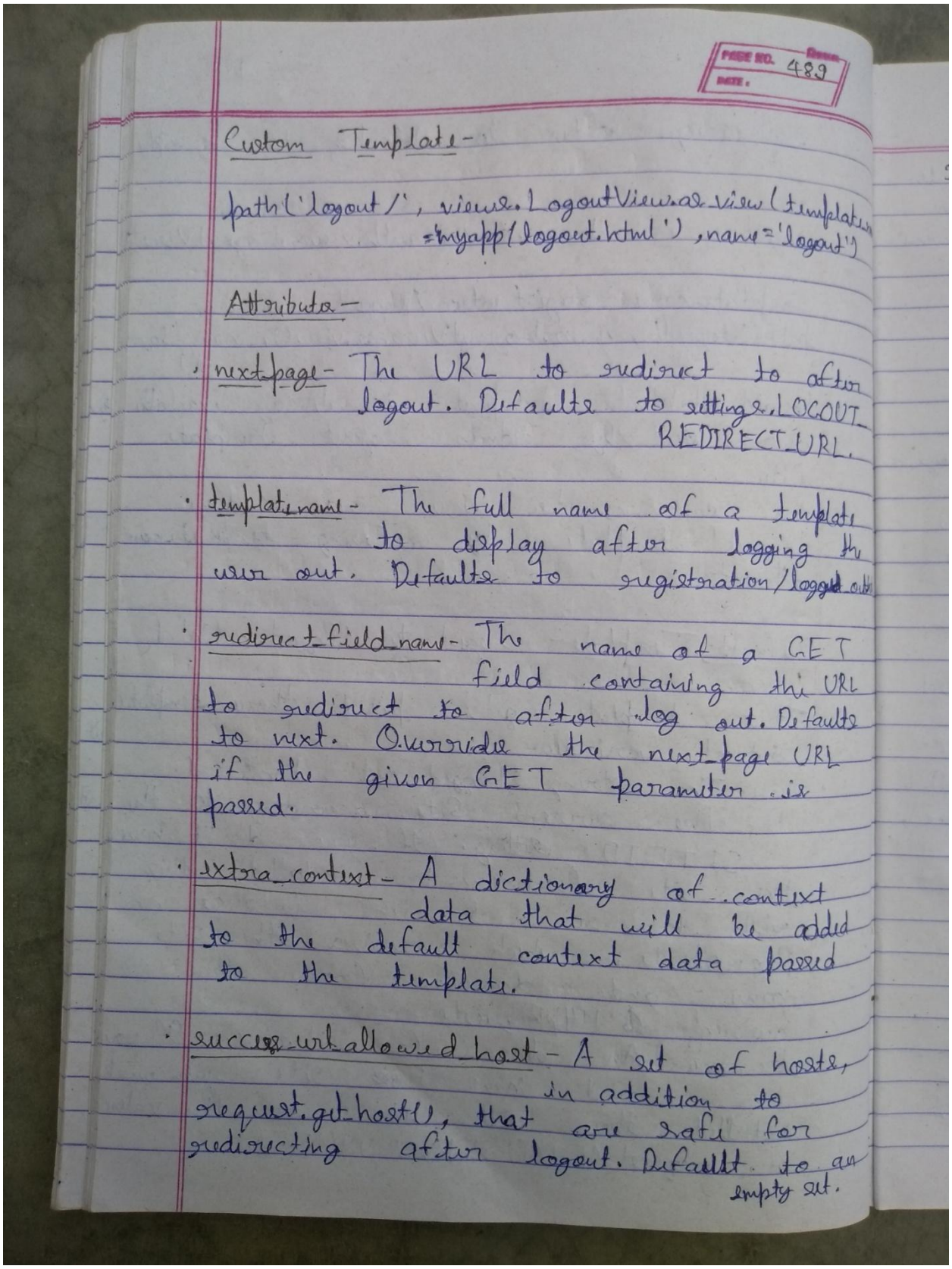
template_name='registration/logged_out.html'
path('logout', views.LogoutView.as_view(), name='logout')

- logged_out.html template is already available and used by admin logout template file.
- You can create your own custom logout template by defining template_name attribute.

- Default Template - registration/logged_out.html

This template gets passed three template context variables -

- title - The string "Logged out", localized.
- site - The current Site, according to the SITEID setting. If you don't have the site framework installed, this will be set to an instance of RequestSite, which derives the site name and domain from the current HttpRequest.
- site_name - An alias for site.name. If you don't have the site framework installed, this will be set to the value of request.META['SERVER_NAME'].



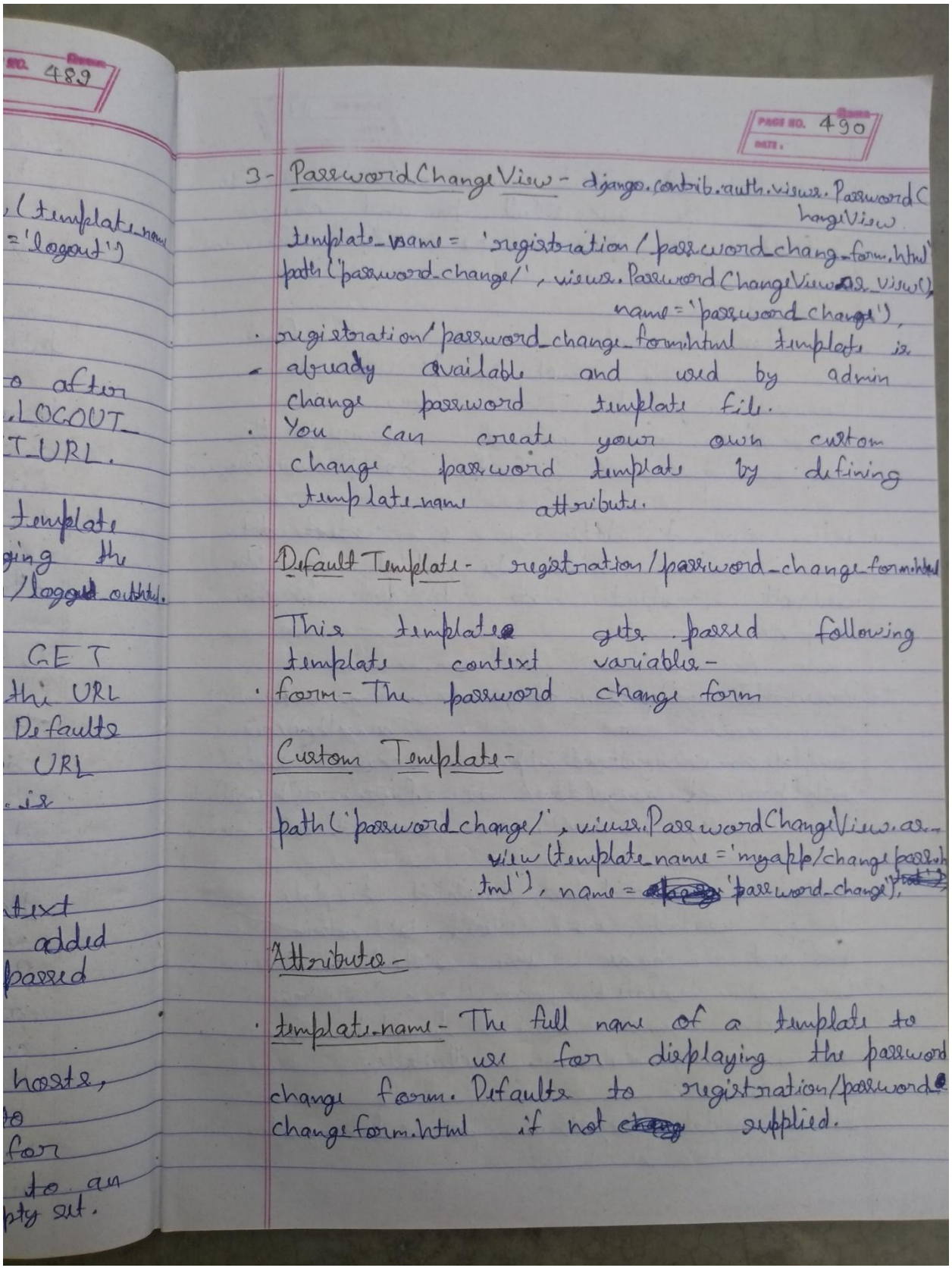
PAGE NO. 489
DATE :

Custom Template -

path('logout/', views.LogoutView.as_view(template_name='myapp/logout.html'), name='logout')

Attributes -

- next_page - The URL to redirect to after logout. Defaults to settings.LOGOUT_REDIRECT_URL.
- template_name - The full name of a template to display after logging the user out. Defaults to registration/logged_out.
- redirect_field_name - The name of a GET field containing the URL to redirect to after log out. Defaults to next. Overrides the next_page URL if the given GET parameter is passed.
- extra_context - A dictionary of context data that will be added to the default context data passed to the template.
- success_url_allowed_hosts - A set of hosts, in addition to request.get_host(), that are safe for redirecting after logout. Default to an empty set.



3- PasswordChangeView - django.contrib.auth.views.PasswordChangeView.

template_name = 'registration/password_change_form.html'

path('password_change/', views.PasswordChangeView.as_view(), name='password_change'),

- registration/password_change_form.html template is already available and used by admin change password template file.
- You can create your own custom change password template by defining template_name attribute.

Default Template - registration/password_change_form.html

This template gets passed following template context variable -

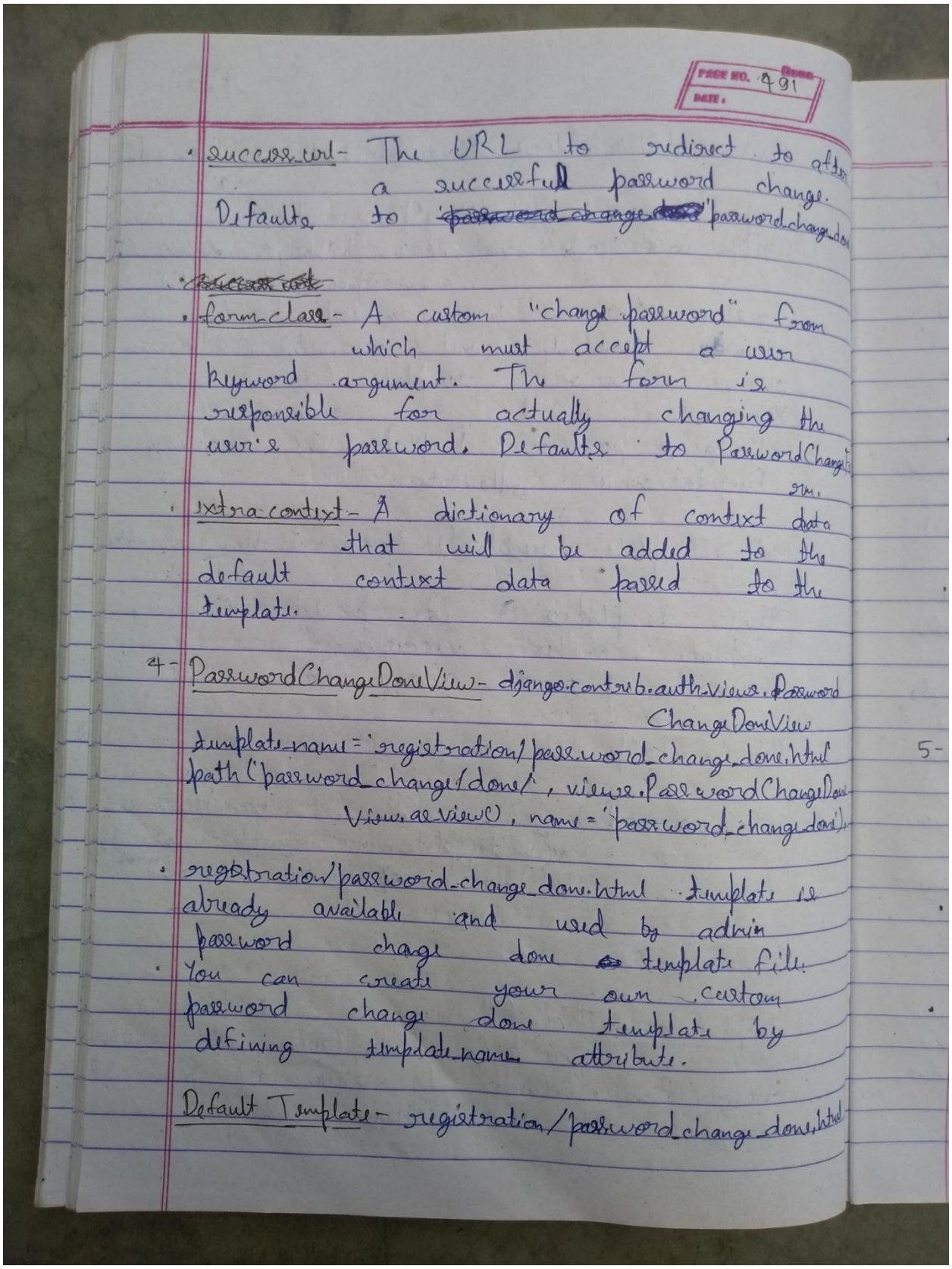
- form - The password change form

Custom Template -

path('password_change/', views.PasswordChangeView.as_view(template_name='myapp/change_password.html'), name='password_change'),

Attributes -

- template_name - The full name of a template to use for displaying the password change form. Defaults to registration/password_change_form.html if not supplied.



PAGE NO. 491
DATE :

- success_url - The URL to redirect to after a successful password change. Defaults to ~~'password_change_done'~~ 'password_change_done'.

- ~~success_url~~
- form_class - A custom "change password" form which must accept a user keyword argument. The form is responsible for actually changing the user's password. Defaults to PasswordChangeForm.
- extra_context - A dictionary of context data that will be added to the default context data passed to the template.

4- PasswordChangeDoneView - django.contrib.auth.views.PasswordChangeDoneView

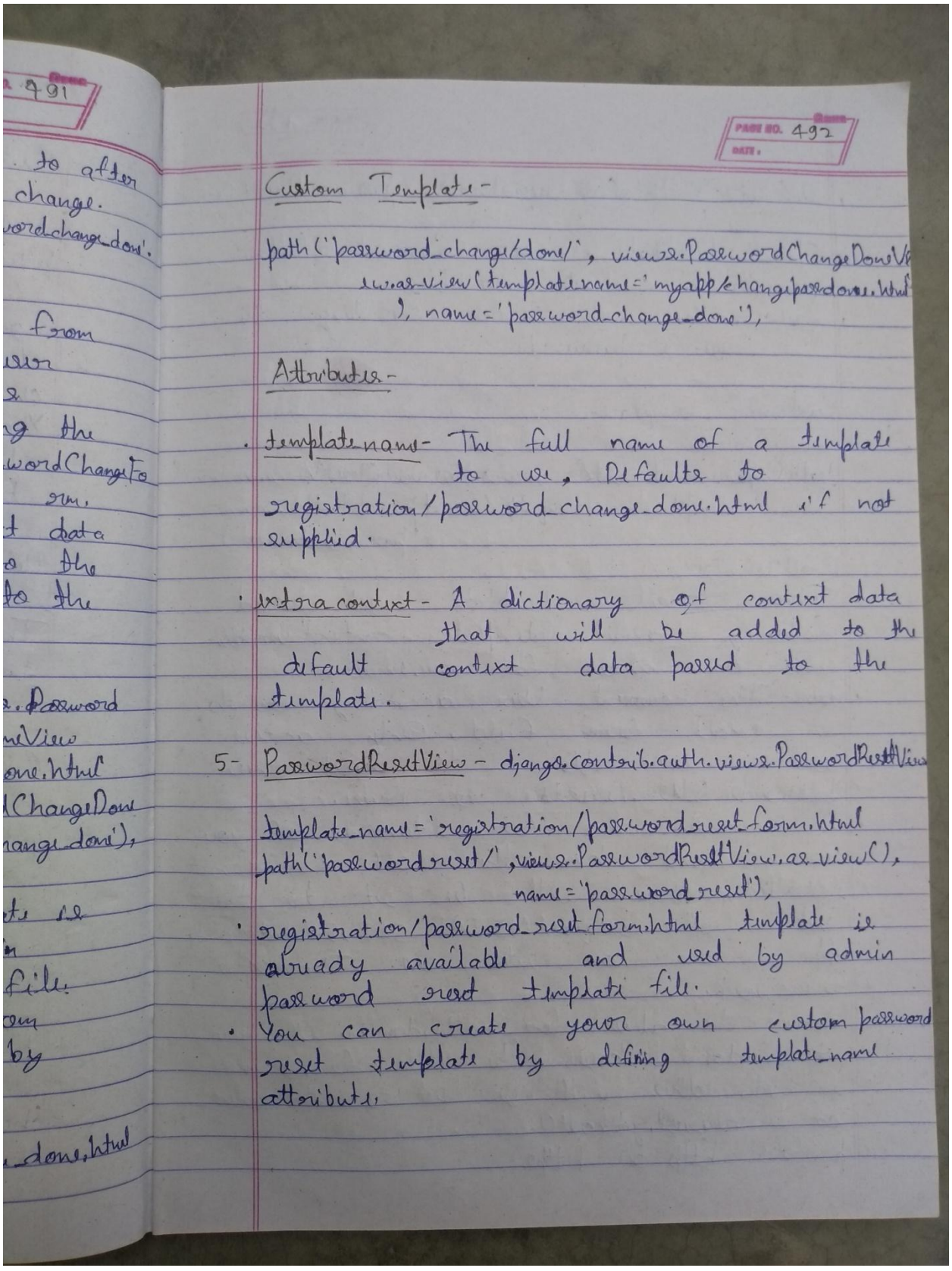
template_name = 'registration/password_change_done.html'

path('password_change/done/', views.PasswordChangeDoneView.as_view(), name='password_change_done')

5-

- registration/password_change_done.html template is already available and used by admin password change done template file.
- You can create your own custom password change done template by defining template_name attribute.

Default Template - registration/password_change_done.html



Custom Template -

path('password_change/done/', views.PasswordChangeDoneVi
ew.as_view(template_name='myapp/changepassdone.html
'), name='password_change_done'),

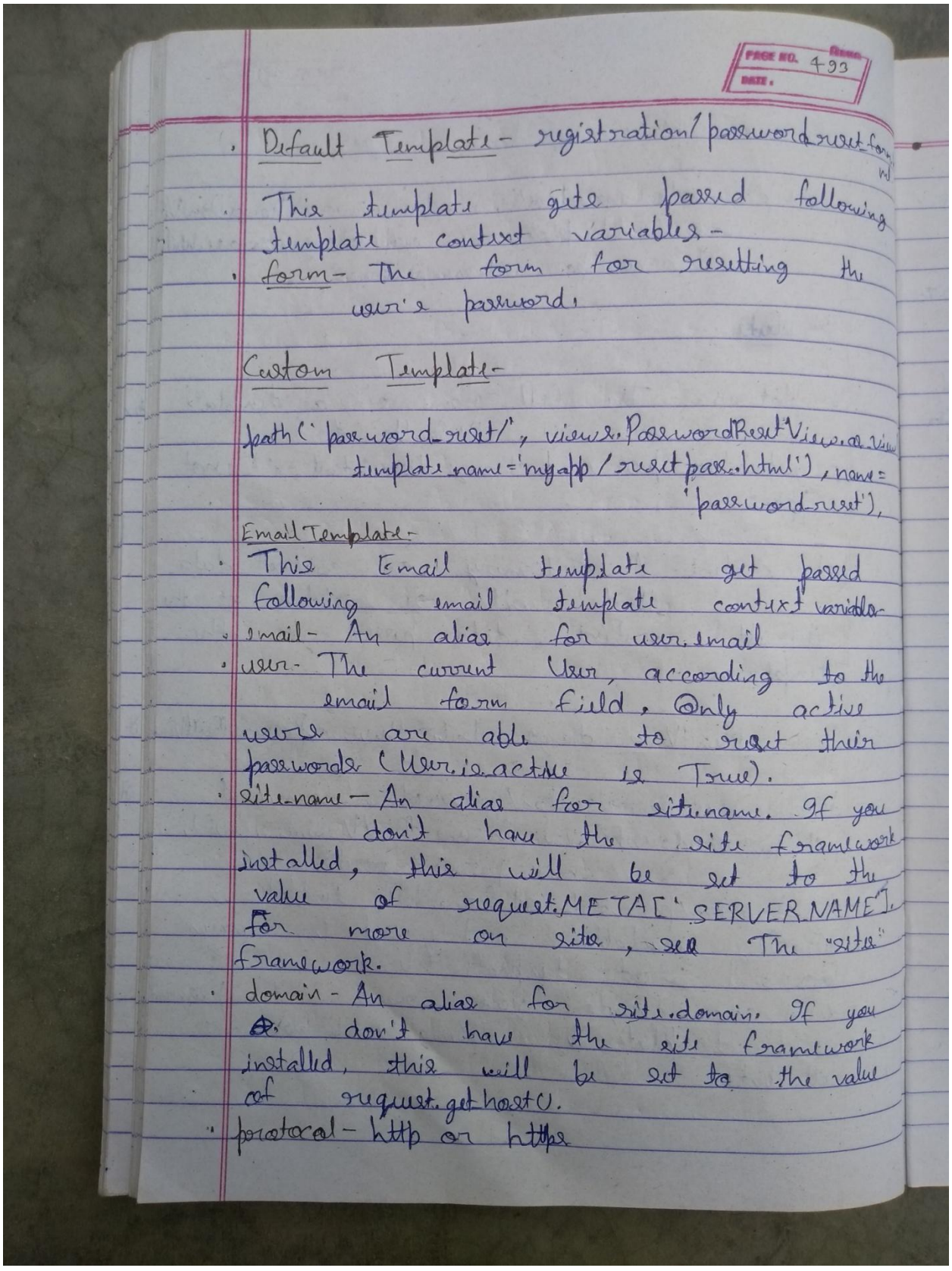
Attributes -

- template_name - The full name of a template to use, Defaults to registration/password_change_done.html if not supplied.
- extra_context - A dictionary of context data that will be added to the default context data passed to the template.

5- PasswordResetView - django.contrib.auth.views.PasswordResetVie

template_name='registration/password_reset_form.html
path('passwordreset/', views.PasswordResetView.as_view(),
name='password_reset'),

- registration/password_reset_form.html template is already available and used by admin password reset template file.
- You can create your own custom password reset template by defining template_name attribute.



PAGE NO. 493
DATE :

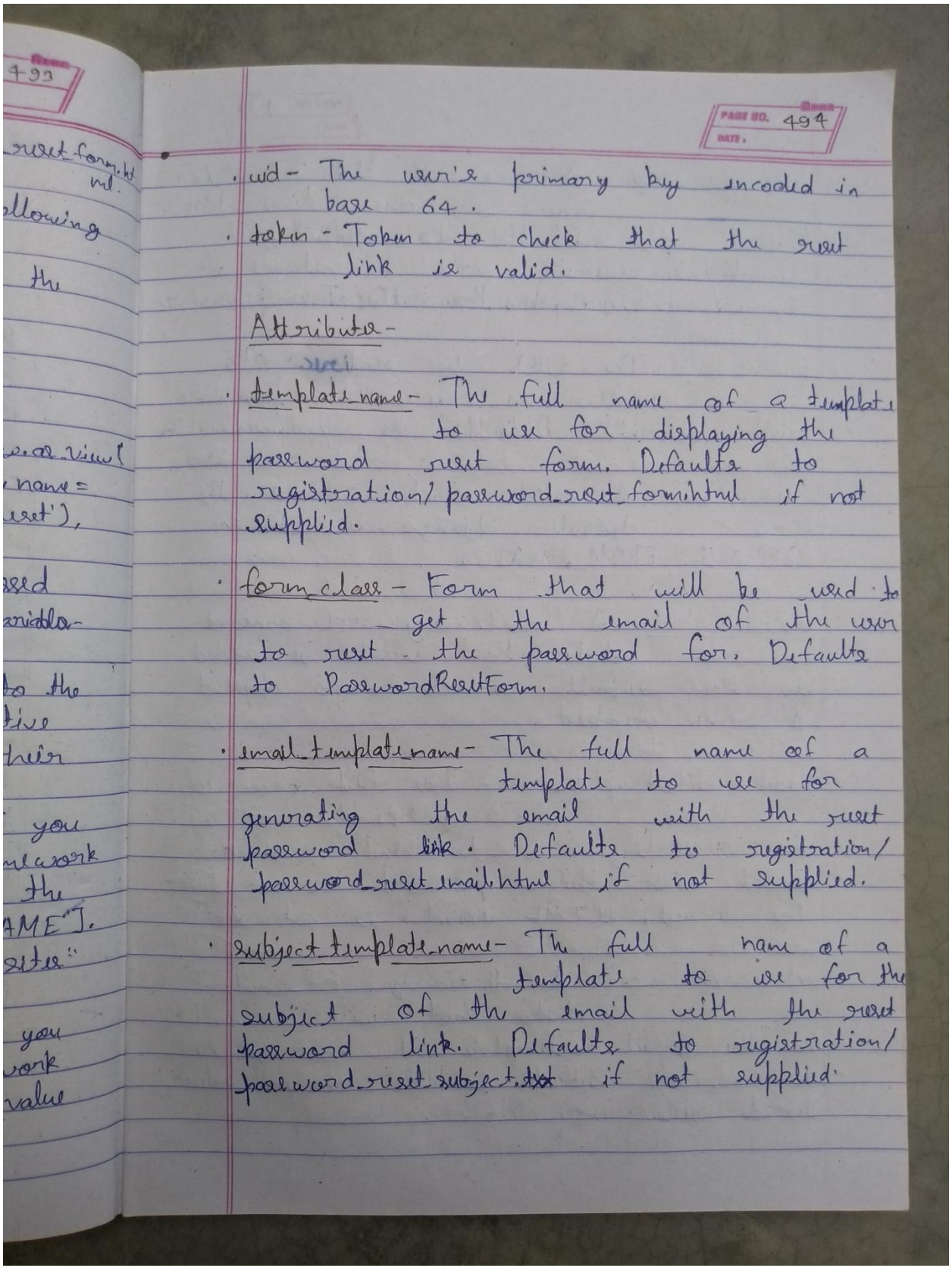
- Default Template - registration/password_reset_form.html
- This template gets passed following template context variables -
- form - The form for resetting the user's password.

Custom Template -

path('password_reset/', views.PasswordResetView.as_view(template_name='myapp/resetpass.html'), name='password_reset'),

Email Template -

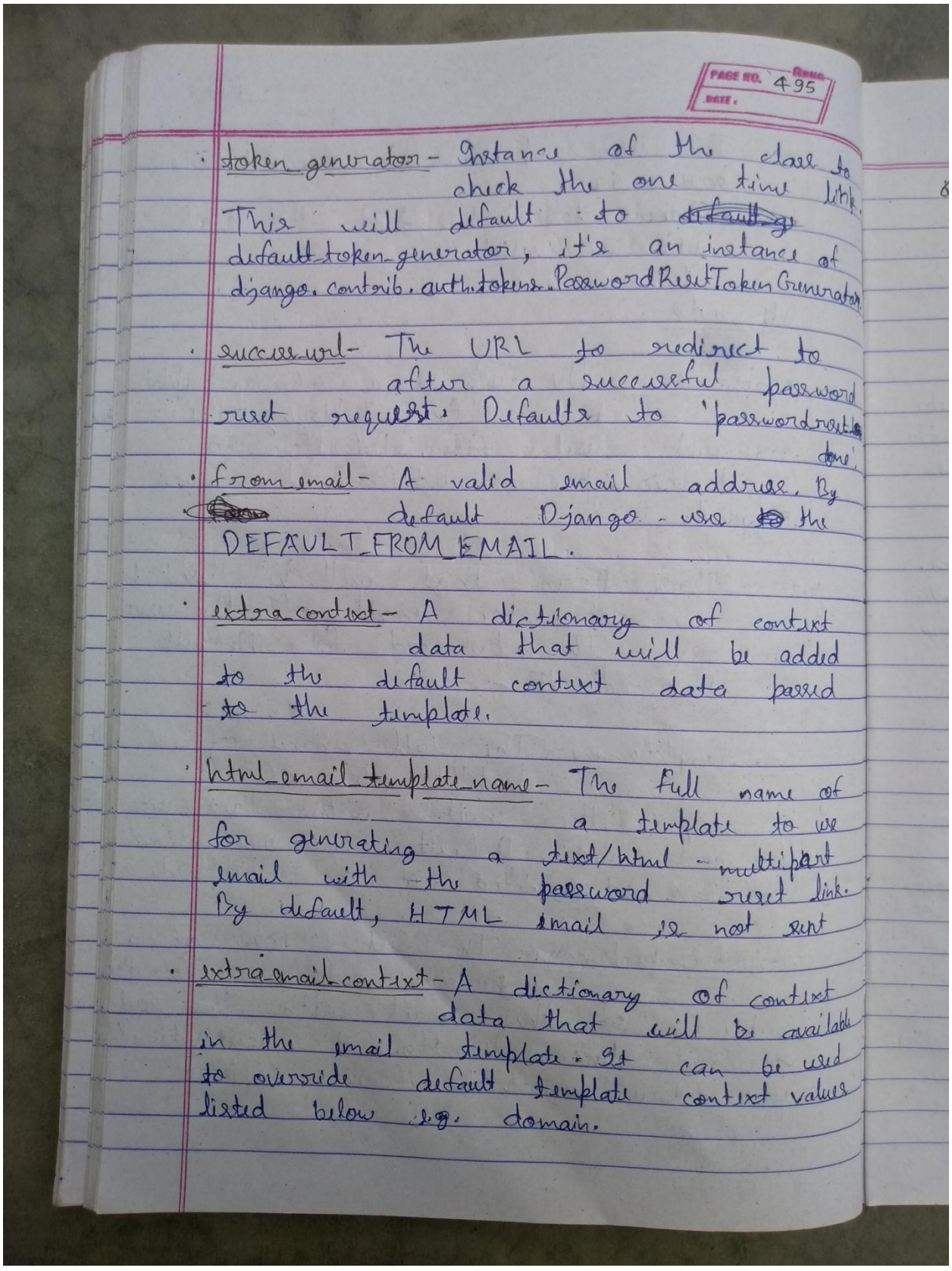
- This Email template get passed following email template context variable -
- email - An alias for user.email
- user - The current User, according to the email form field. Only active users are able to reset their passwords (User.is_active is True).
- site_name - An alias for site.name. If you don't have the site framework installed, this will be set to the value of request.META['SERVER_NAME']. For more on sites, see The "sites" framework.
- domain - An alias for site.domain. If you don't have the site framework installed, this will be set to the value of request.get_host().
- protocol - http or https



- uid - The user's primary key encoded in base 64.
- token - Token to check that the reset link is valid.

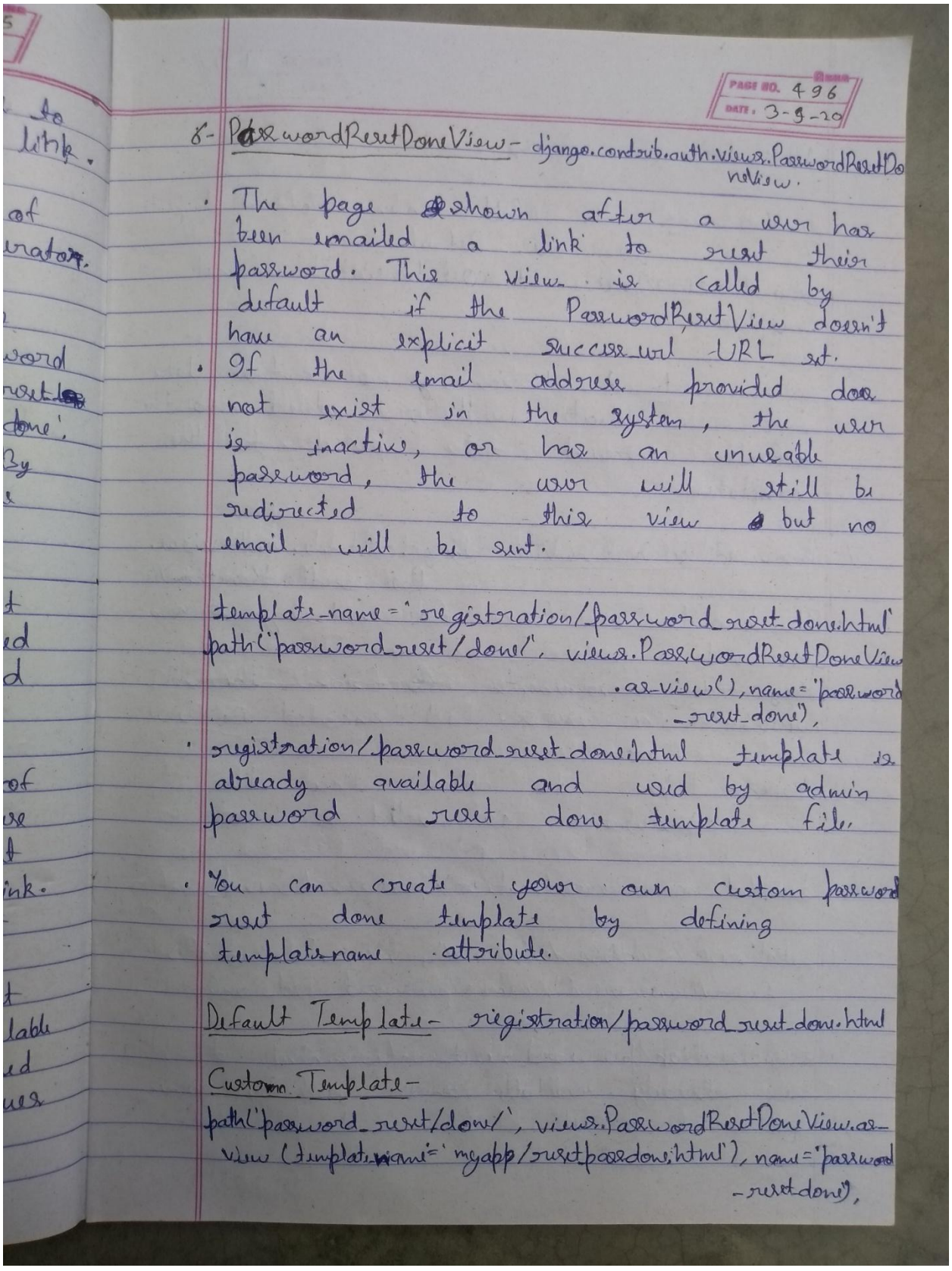
Attributes -

- template_name - The full name of a template to use for displaying the password reset form. Defaults to registration/password_reset_form.html if not supplied.
- form_class - Form that will be used to get the email of the user to reset the password for. Defaults to PasswordResetForm.
- email_template_name - The full name of a template to use for generating the email with the reset password link. Defaults to registration/password_reset_email.html if not supplied.
- subject_template_name - The full name of a template to use for the subject of the email with the reset password link. Defaults to registration/password_reset_subject.txt if not supplied.



PAGE NO. 495
DATE :

- token_generator - Instance of the class to check the one time link. This will default to ~~default~~ default_token_generator, it's an instance of django.contrib.auth.tokens.PasswordResetTokenGenerator.
- success_url - The URL to redirect to after a successful password reset request. Defaults to 'password_reset_done'.
- from_email - A valid email address. By default Django uses the DEFAULT_FROM_EMAIL.
- extra_context - A dictionary of context data that will be added to the default context data passed to the template.
- html_email_template_name - The full name of a template to use for generating a text/html multipart email with the password reset link. By default, HTML email is not sent
- extra_email_context - A dictionary of context data that will be available in the email template. It can be used to override default template context values listed below e.g. domain.



PAGE NO. 496
DATE : 3-9-20

8- PasswordResetDoneView - django.contrib.auth.views.PasswordResetDoneView.

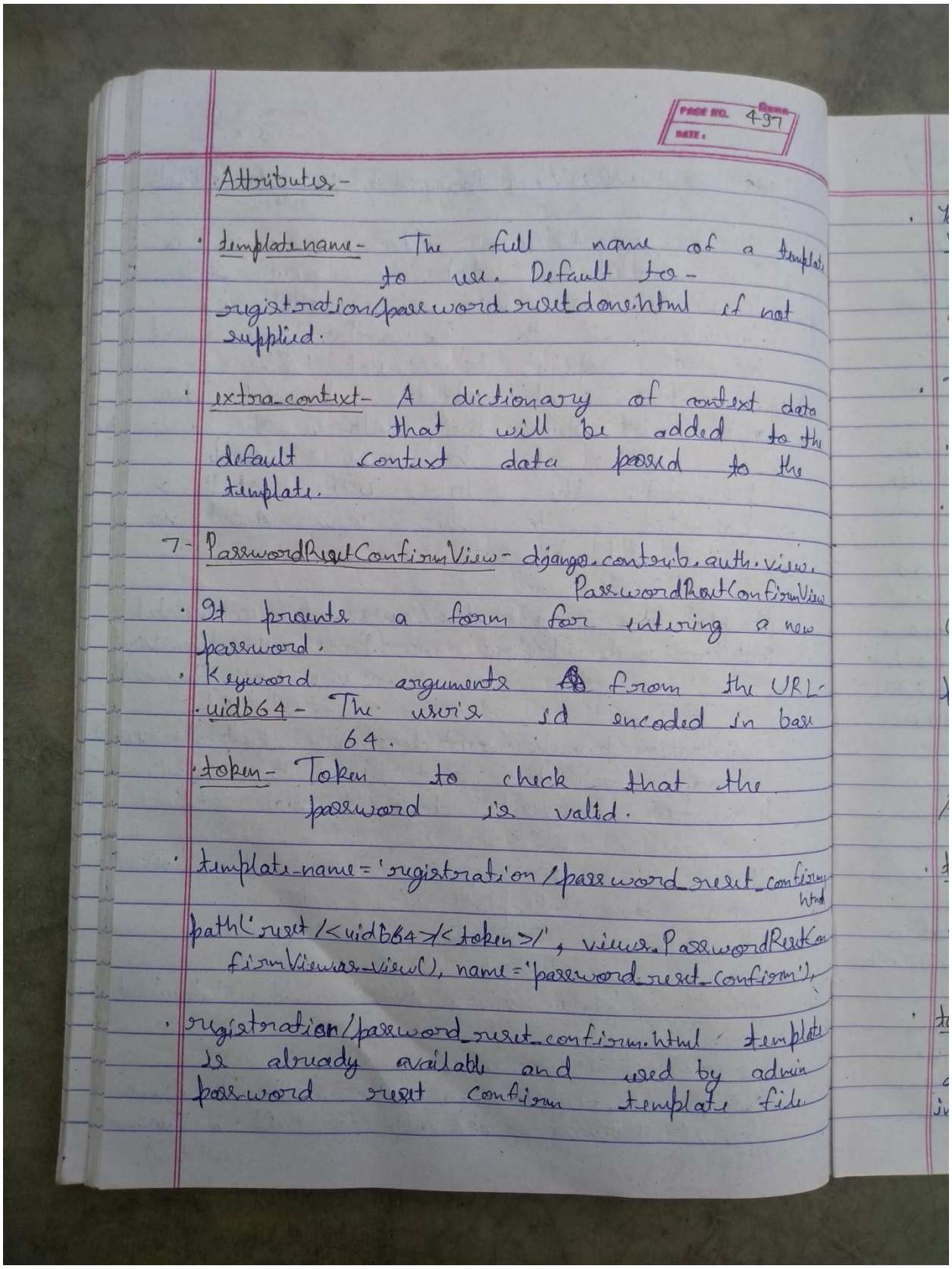
- The page shown after a user has been emailed a link to reset their password. This view is called by default if the PasswordResetView doesn't have an explicit success_url URL set.
- If the email address provided does not exist in the system, the user is inactive, or has an unusable password, the user will still be redirected to this view but no email will be sent.

template_name = 'registration/password_reset_done.html'
path('password_reset/done/', views.PasswordResetDoneView.as_view(), name='password_reset_done'),

- registration/password_reset_done.html template is already available and used by admin password reset done template file.
- You can create your own custom password reset done template by defining template_name attribute.

Default Template - registration/password_reset_done.html

Custom Template -
path('password_reset/done/', views.PasswordResetDoneView.as_view(template_name='myapp/resetpassdone.html'), name='password_reset_done'),



PAGE NO. 497
DATE :

Attributes -

- template_name - The full name of a template to use. Default to - registration/password_reset_done.html if not supplied.

- extra_context - A dictionary of context data that will be added to the default context data passed to the template.

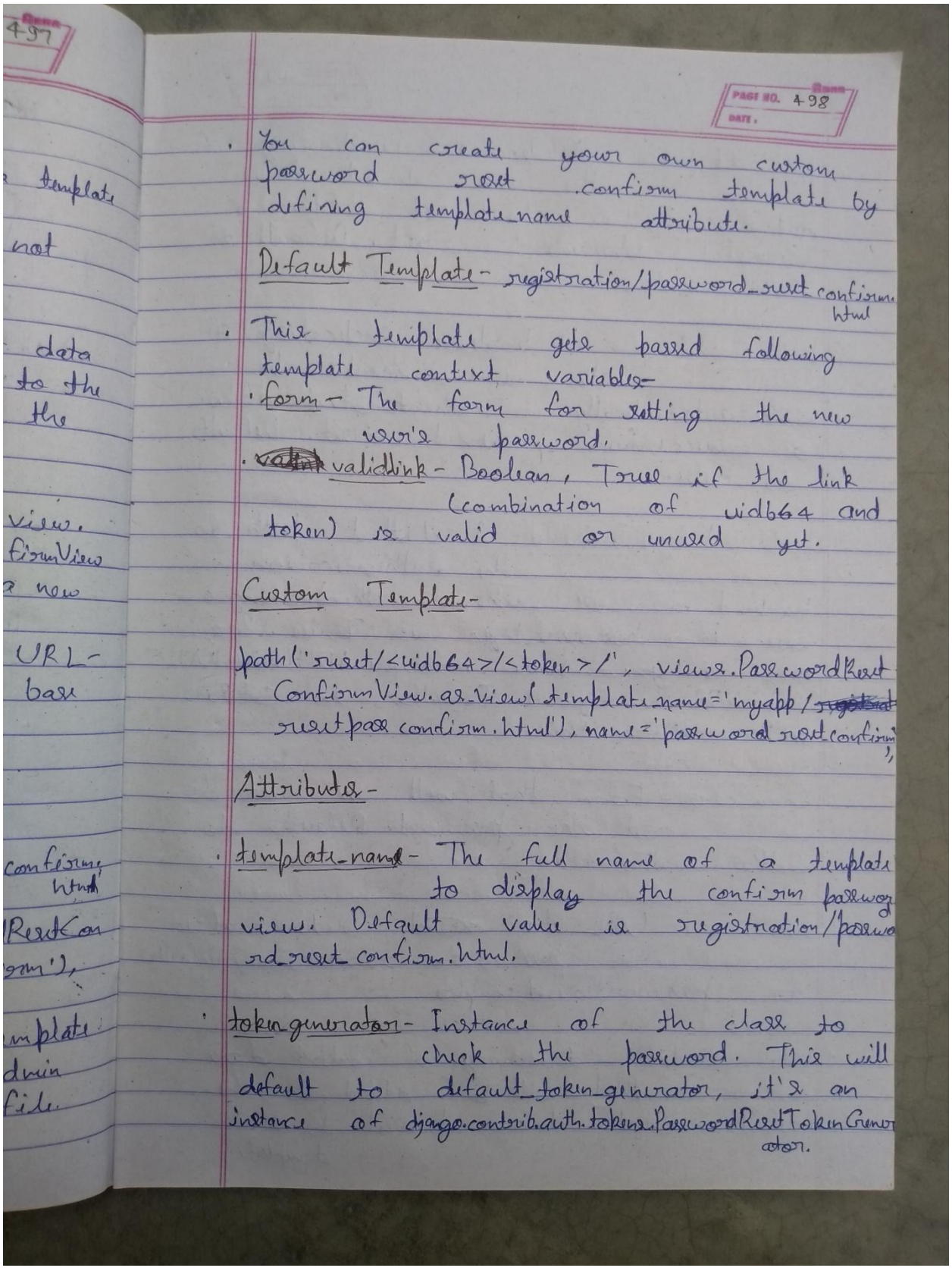
7- PasswordResetConfirmView - django.contrib.auth.views.PasswordResetConfirmView

- It prompts a form for entering a new password.
- Keyword arguments from the URL -
- uidb64 - The user's id encoded in base 64.
- token - Token to check that the password is valid.

- template_name = 'registration/password_reset_confirm.html'

path('reset/<uidb64>/<token>/', views.PasswordResetConfirmView.as_view(), name='password_reset_confirm'),

- registration/password_reset_confirm.html template is already available and used by admin password reset confirm template file



497

PAGE NO. 498
DATE :

- You can create your own custom password reset confirm template by defining template_name attribute.

Default Template - registration/password_reset_confirm.html

- This template gets passed following template context variables-
  - form - The form for setting the new user's password.
  - validlink - Boolean, True if the link (combination of uidb64 and token) is valid or unused yet.

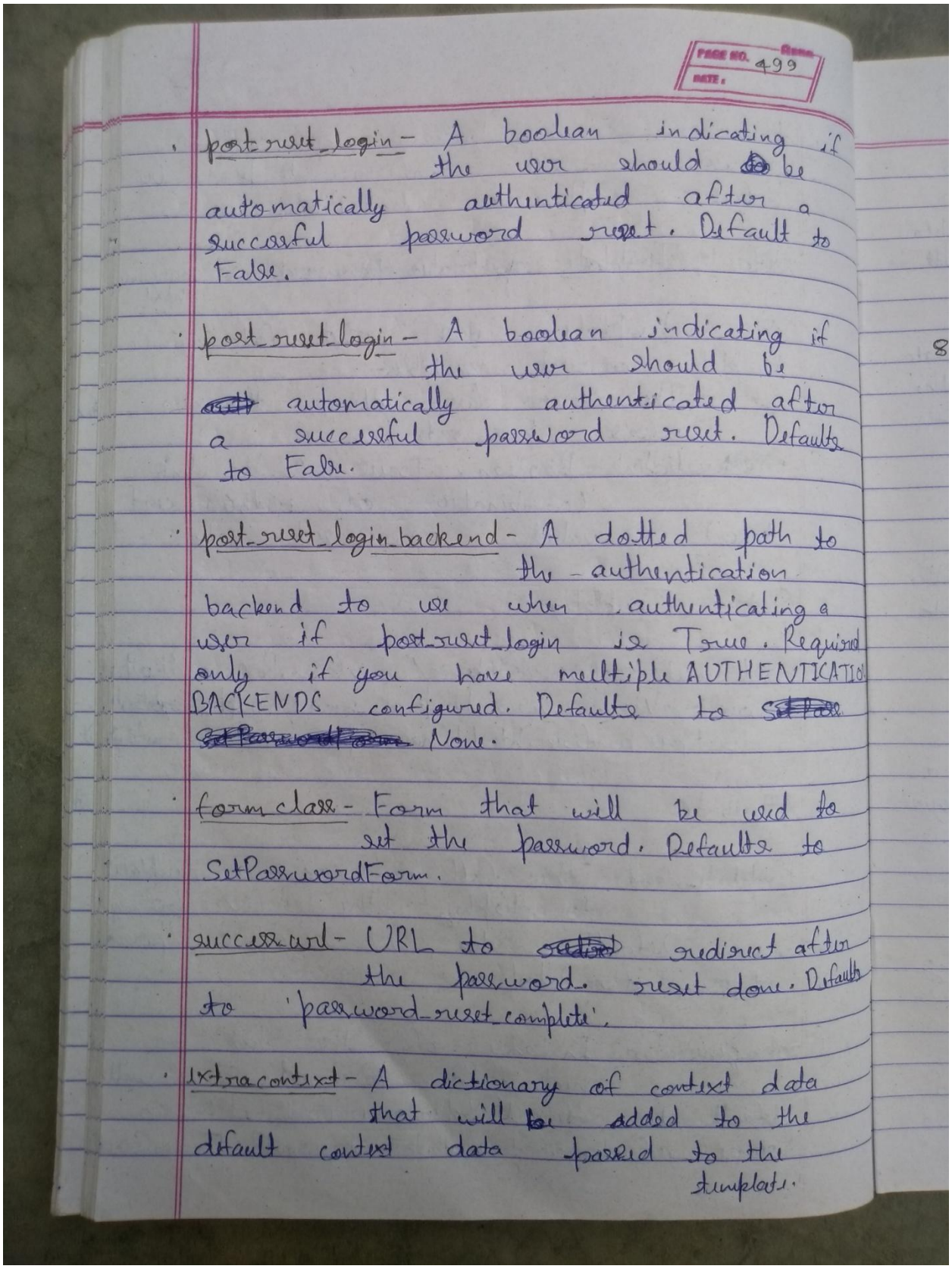
Custom Template -

path('reset/<uidb64>/<token>/', views.PasswordResetConfirmView.as_view(template_name='myapp/~~registration~~ resetpass_confirm.html'), name='password_reset_confirm'),

Attributes -

- template_name - The full name of a template to display the confirm password view. Default value is registration/password_reset_confirm.html.

- token_generator - Instance of the class to check the password. This will default to default_token_generator, it's an instance of django.contrib.auth.tokens.PasswordResetTokenGenerator.



PAGE NO. 499
DATE :

- post_reset_login - A boolean indicating if the user should be automatically authenticated after a succesful password reset. Default to False.

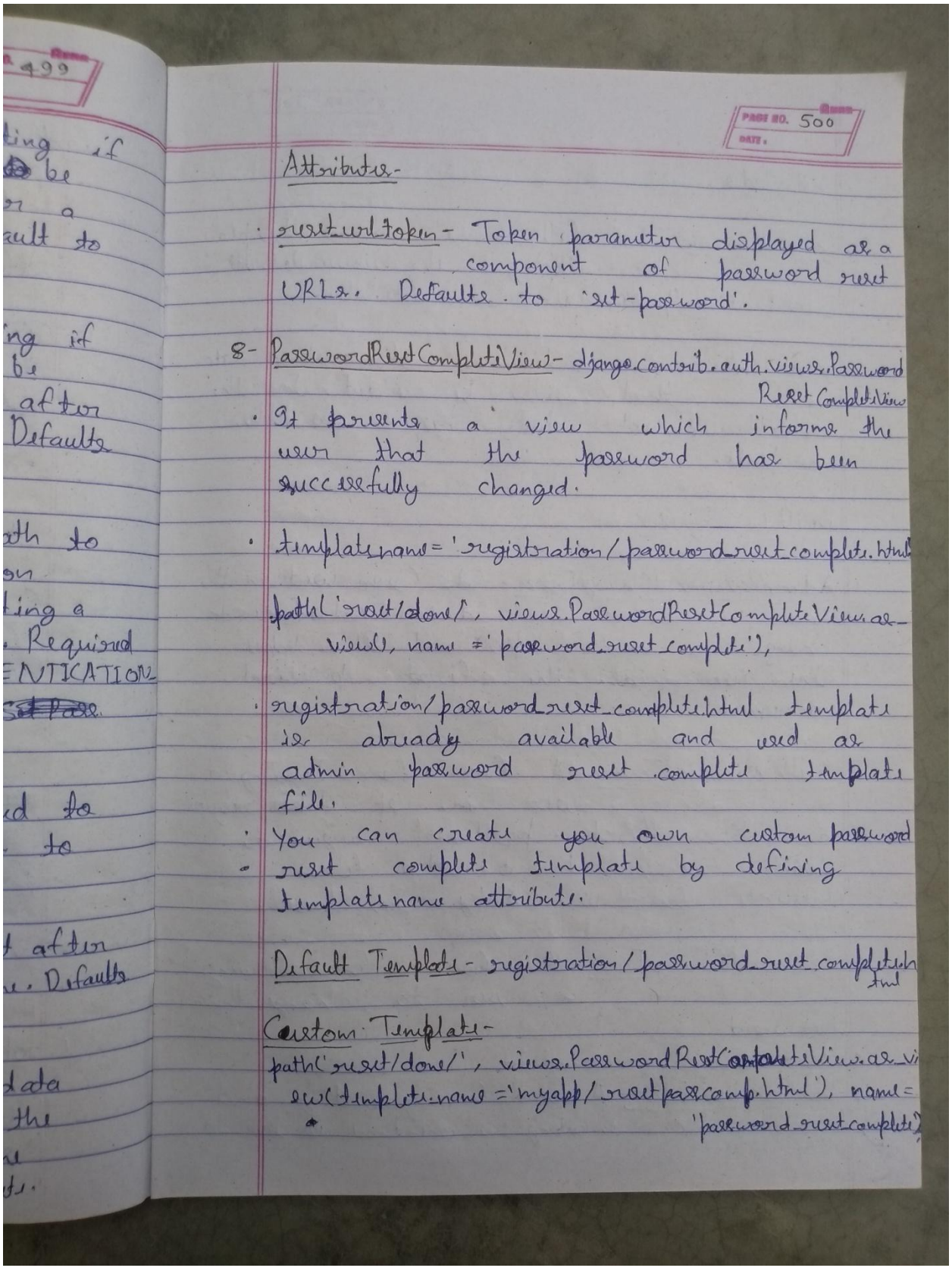
- post_reset_login - A boolean indicating if the user should be automatically authenticated after a successful password reset. Defaults to False.

- post_reset_login_backend - A dotted path to the authentication backend to use when authenticating a user if post_reset_login is True. Required only if you have multiple AUTHENTICATION BACKENDS configured. Defaults to None.

- form_class - Form that will be used to set the password. Defaults to SetPasswordForm.

- success_url - URL to redirect after the password reset done. Defaults to 'password_reset_complete'.

- extra_context - A dictionary of context data that will be added to the default context data passed to the template.



Attributes-

- reset_url_token - Token parameter displayed as a component of password reset URLs. Defaults to 'set-password'.

8- PasswordResetCompleteView - django.contrib.auth.views.PasswordResetCompleteView

- It presents a view which informs the user that the password has been successfully changed.
- template_name = 'registration/password_reset_complete.html'

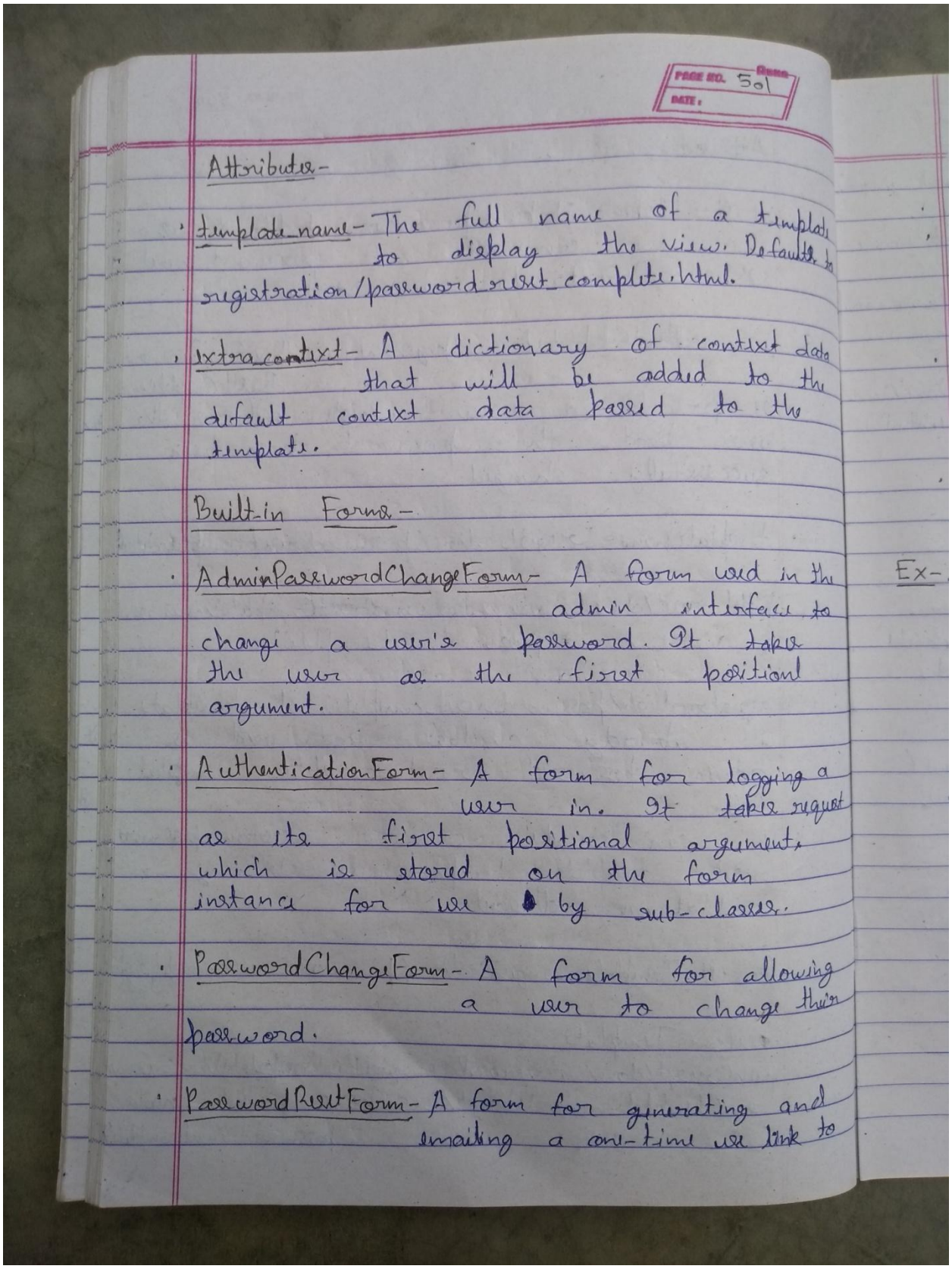
path('reset/done/', views.PasswordResetCompleteView.as_view(), name = 'password_reset_complete'),

- registration/password_reset_complete.html template is already available and used as admin password reset complete template file.
- You can create you own custom password reset complete template by defining template_name attribute.

Default Template - registration/password_reset_complete.html

Custom Template -

path('reset/done/', views.PasswordResetCompleteView.as_view(template_name = 'myapp/reset/passcomp.html'), name = 'password_reset_complete')



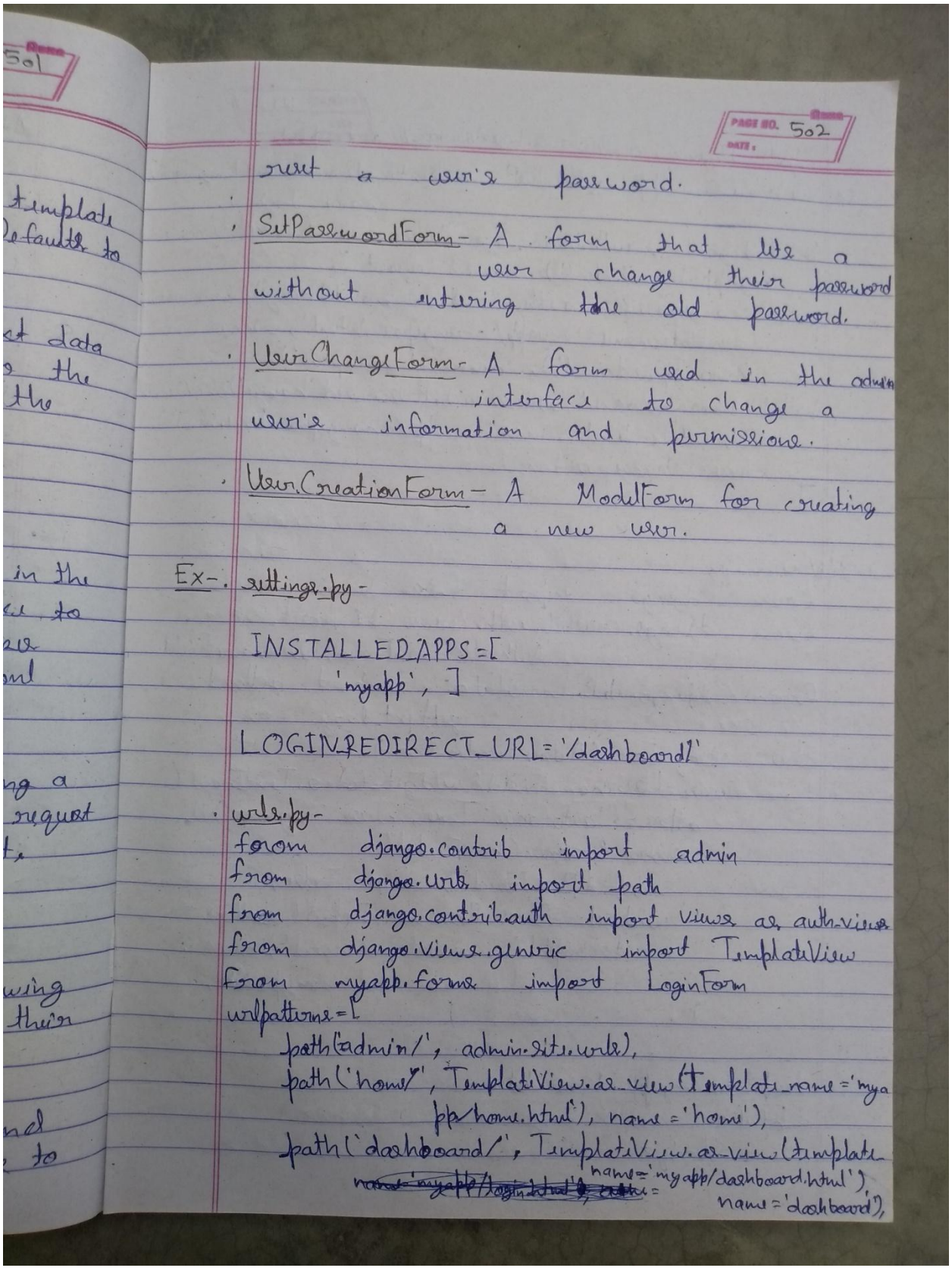
PAGE NO. 501
DATE :

Attributes -

- template_name - The full name of a template to display the view. Defaults to registration/password_reset_complete.html.
- extra_context - A dictionary of context data that will be added to the default context data passed to the template.

Built-in Forms -

- AdminPasswordChangeForm - A form used in the admin interface to change a user's password. It takes the user as the first positional argument. Ex-
- AuthenticationForm - A form for logging a user in. It takes request as its first positional argument, which is stored on the form instance for use by sub-classes.
- PasswordChangeForm - A form for allowing a user to change their password.
- PasswordResetForm - A form for generating and emailing a one-time use link to



reset a user's password.

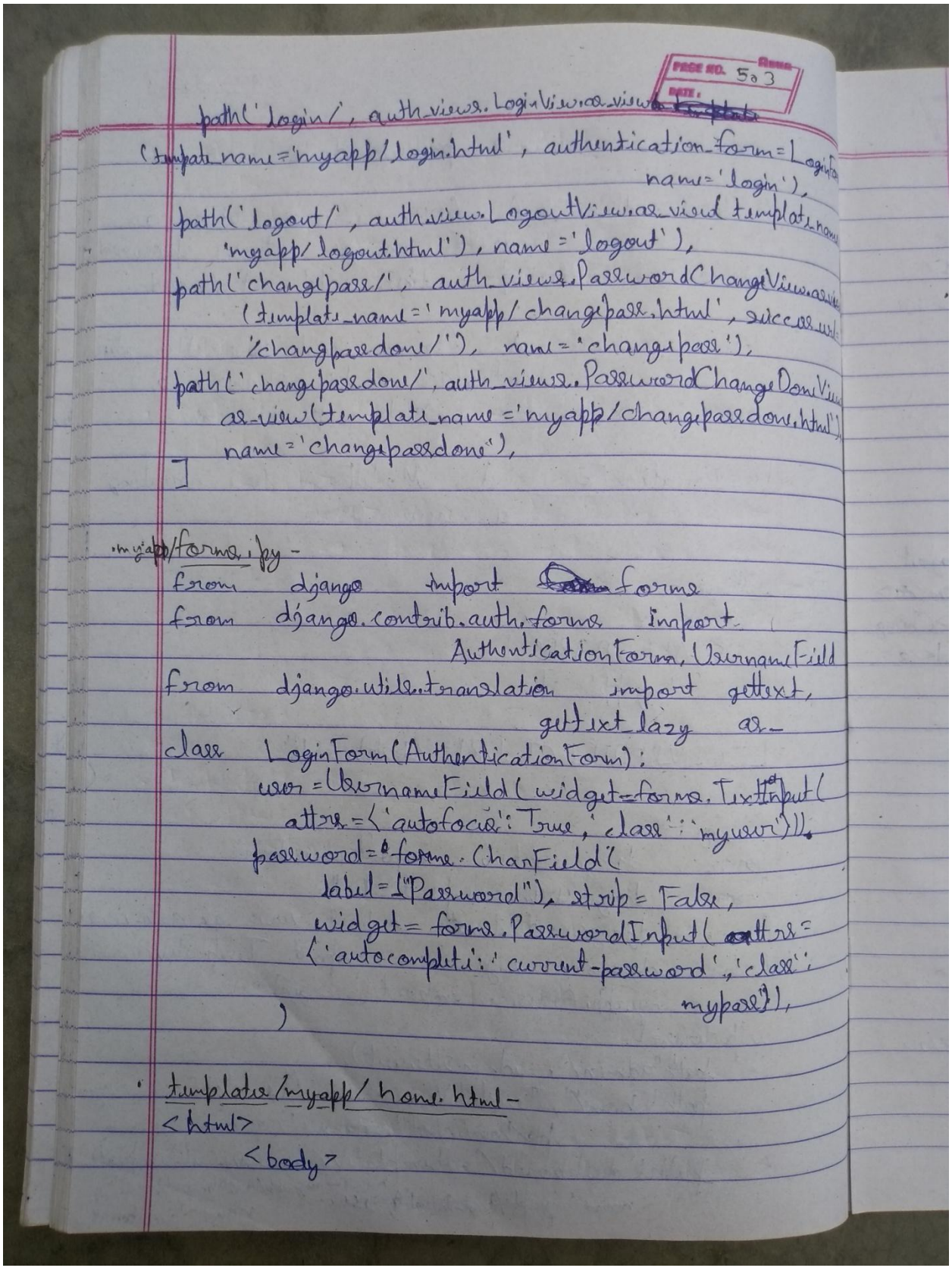
- SetPasswordForm - A form that lets a user change their password without entering the old password.
- UserChangeForm - A form used in the admin interface to change a user's information and permissions.
- User.CreationForm - A ModelForm for creating a new user.

Ex- settings.py -

```
INSTALLED_APPS=[
    'myapp', ]

LOGIN_REDIRECT_URL='/dashboard/'
```

- urls.py -

```
from django.contrib import admin
from django.urls import path
from django.contrib.auth import views as auth_views
from django.views.generic import TemplateView
from myapp.forms import LoginForm
urlpatterns=[
    path('admin/', admin.site.urls),
    path('home/', TemplateView.as_view(template_name='myapp/home.html'), name='home'),
    path('dashboard/', TemplateView.as_view(template_name='myapp/dashboard.html'), name='dashboard'),
```



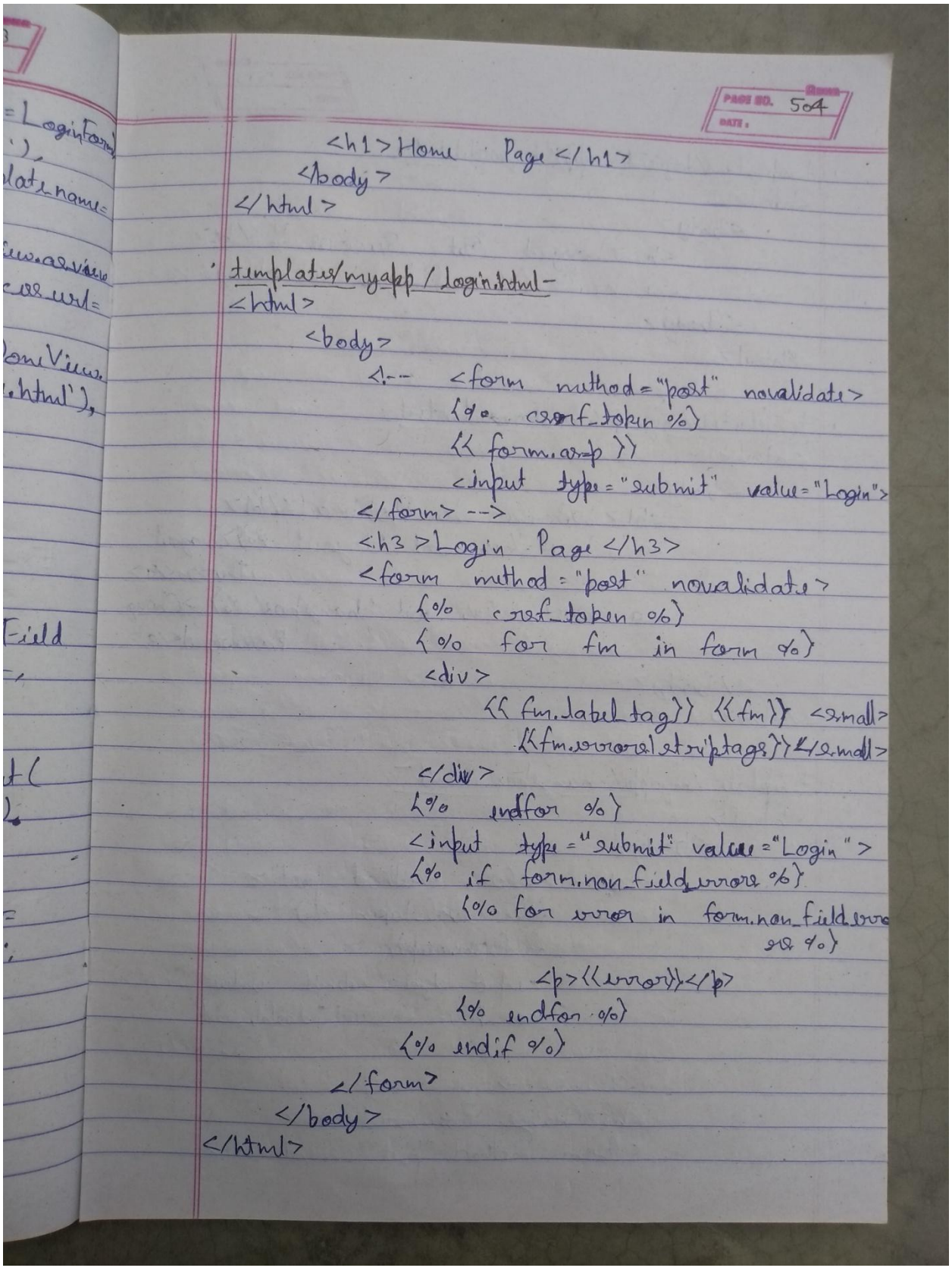
path('login/', auth_views.LoginView.as_view(template_name='myapp/login.html', authentication_form=LoginForm, name='login'),
path('logout/', auth_view.LogoutView.as_view(template_name='myapp/logout.html'), name='logout'),
path('changepass/', auth_views.PasswordChangeView.as_view(template_name='myapp/changepass.html', success_url='/changepassdone/'), name='changepass'),
path('changepassdone/', auth_views.PasswordChangeDoneView.as_view(template_name='myapp/changepassdone.html'), name='changepassdone'),
]

• myapp/forms.py -

```
from django import forms
from django.contrib.auth.forms import
        AuthenticationForm, UsernameField
from django.utils.translation import gettext,
        gettext_lazy as _
class LoginForm(AuthenticationForm):
    user = UsernameField(widget=forms.TextInput(
        attrs={'autofocus': True, 'class': 'myuser'})),
    password = forms.CharField(
        label=("Password"), strip=False,
        widget=forms.PasswordInput(attrs=
        {'autocomplete': 'current-password', 'class':
                'mypass'}),
    )
```

• templates/myapp/home.html -

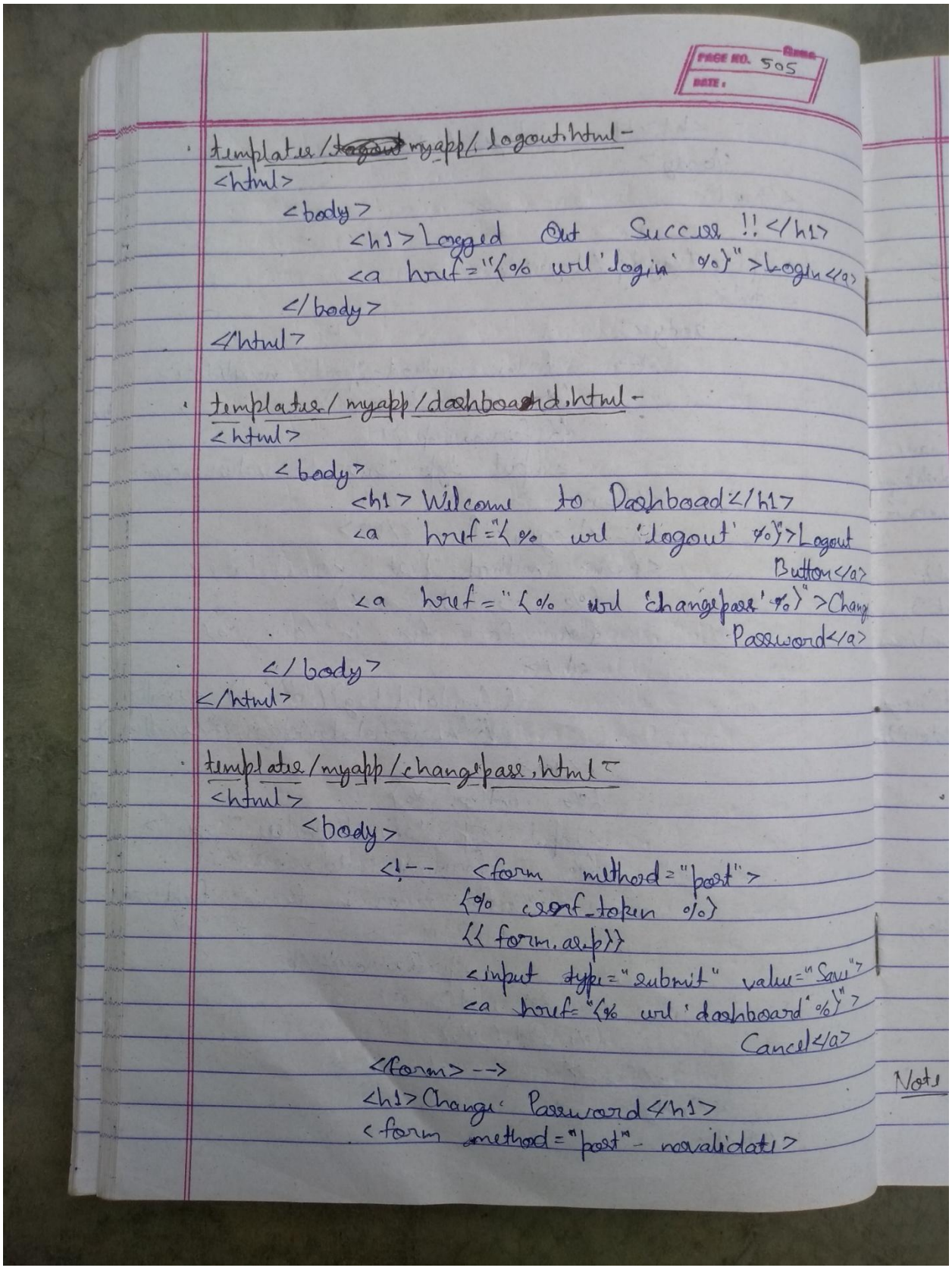
```
<html>
    <body>
```



```
        <h1> Home Page </h1>
    </body>
</html>
```

templates/myapp/login.html -

```
<html>
    <body>
        <!-- <form method="post" novalidate>
            {% csrf_token %}
            {{ form.as_p }}
            <input type="submit" value="Login">
        </form> -->
        <h3> Login Page </h3>
        <form method="post" novalidate>
            {% csrf_token %}
            {% for fm in form %}
            <div>
                {{ fm.label_tag }} {{ fm }} <small>
                {{ fm.errors|striptags }} </small>
            </div>
            {% endfor %}
            <input type="submit" value="Login">
            {% if form.non_field_errors %}
                {% for error in form.non_field_errors %}
                    <p> {{ error }} </p>
                {% endfor %}
            {% endif %}
        </form>
    </body>
</html>
```



templates/~~logout~~myapp/logout.html -

```
<html>
    <body>
        <h1> Logged Out Succes !! </h1>
        <a href="{% url 'login' %}" >Login</a>
    </body>
</html>
```

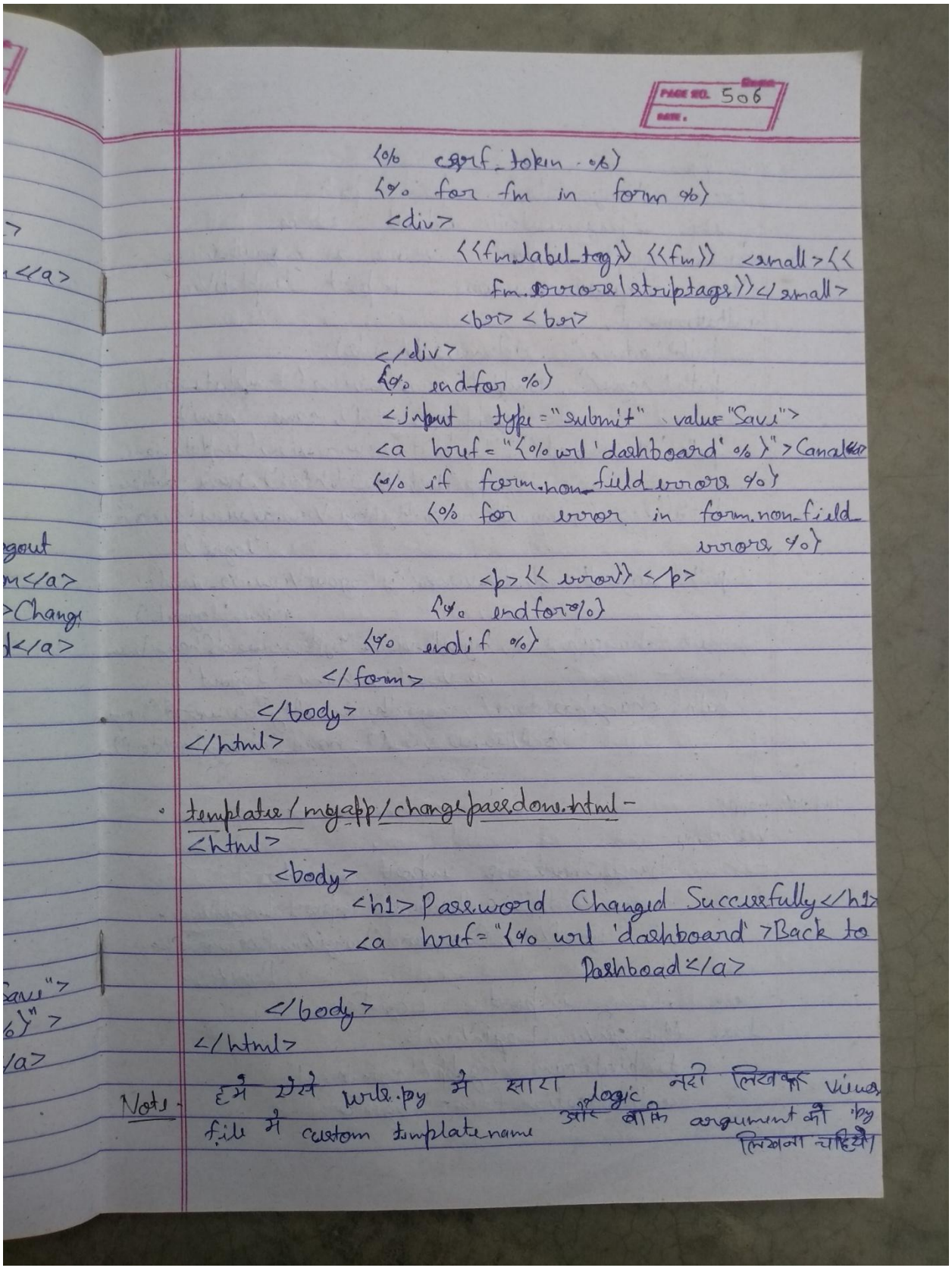
templates/myapp/~~dashboard~~dashboard.html -

```
<html>
    <body>
        <h1> Welcome to Dashboard</h1>
        <a href="{% url 'logout' %}">Logout
                                    Button</a>
        <a href="{% url 'changepass' %}" >Chang
                                    Password</a>
    </body>
</html>
```

templates/myapp/changepass.html -

```
<html>
    <body>
        <!-- <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <input type="submit" value="Save">
            <a href="{% url 'dashboard' %}">
                                    Cancel</a>
        </form> -->
        <h1> Change Password </h1>
        <form method="post" novalidate>
```

Note

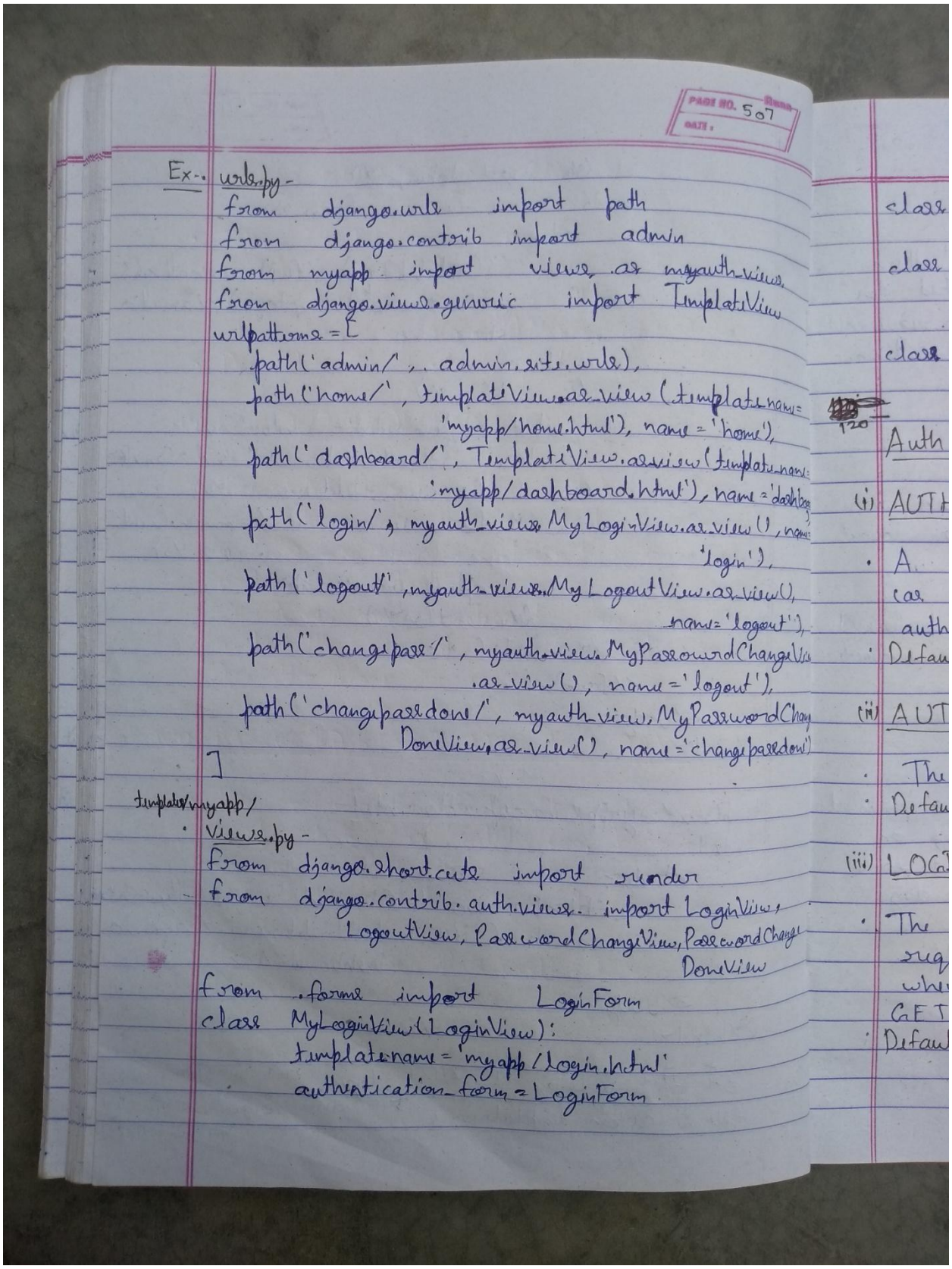


```
{% csrf_token %}
{% for fm in form %}
<div>
    {{fm.label_tag}} {{fm}} <small>{{
    fm.errors|striptags}}</small>
    <br><br>
</div>
{% endfor %}
<input type="submit" value="Save">
<a href="{% url 'dashboard' %}">Cancel</a>
{% if form.non_field_errors %}
    {% for error in form.non_field_
                                errors %}
        <p>{{ error }}</p>
    {% endfor %}
{% endif %}
</form>
</body>
</html>
```

- templates/myapp/changepassdone.html -

```
<html>
    <body>
        <h1>Password Changed Successfully</h1>
        <a href="{% url 'dashboard' %}">Back to
                                Dashboad</a>
    </body>
</html>
```

Note - हमें अपने urls.py में सारा logic नहीं लिखना और views.py file में custom template name और बाकि argument को pass लिखना चाहिये।



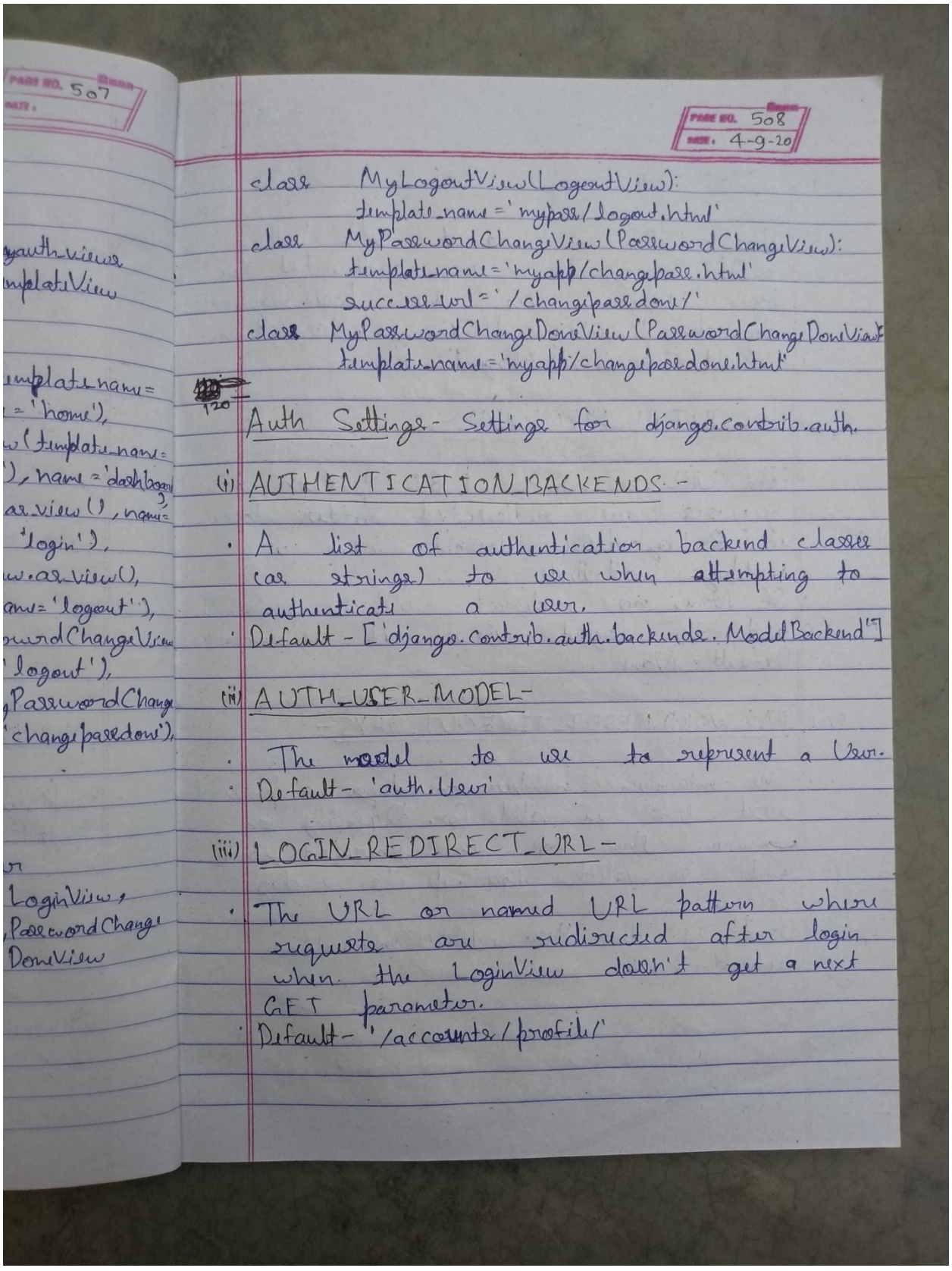
Ex.. urls.py -

```
from django.urls import path
from django.contrib import admin
from myapp import views, as myauth_views.
from django.views.generic import TemplateView
urlpatterns = [
    path('admin/', admin.site.urls),
    path('home/', TemplateView.as_view(template_name=
        'myapp/home.html'), name = 'home'),
    path('dashboard/', TemplateView.as_view(template_name=
        'myapp/dashboard.html'), name = 'dashboard'),
    path('login/', myauth_views.MyLoginView.as_view(), name=
        'login'),
    path('logout', myauth_views.MyLogoutView.as_view(),
        name='logout'),
    path('changepass/', myauth_views.MyPasswordChangeVi
        .as_view(), name='logout'),
    path('changepassdone/', myauth_views.MyPasswordChan
        DoneView.as_view(), name='changepassdone')
]
```

templates/myapp/

- views.py -

```
from django.shortcuts import render
from django.contrib.auth.views import LoginView,
        LogoutView, PasswordChangeView, PasswordChange
        DoneView

from .forms import LoginForm
class MyLoginView(LoginView):
    template_name = 'myapp/login.html'
    authentication_form = LoginForm
```



PAGE NO. 507

PAGE NO. 508
DATE: 4-9-20

```
class MyLogoutView(LogoutView):
    template_name = 'mypass/logout.html'
class MyPasswordChangeView(PasswordChangeView):
    template_name = 'myapp/changepass.html'
    success_url = '/changepassdone/'
class MyPasswordChangeDoneView(PasswordChangeDoneView):
    template_name = 'myapp/changepassdone.html'
```

120

Auth Settings - Settings for django.contrib.auth.

(i) AUTHENTICATION_BACKENDS -

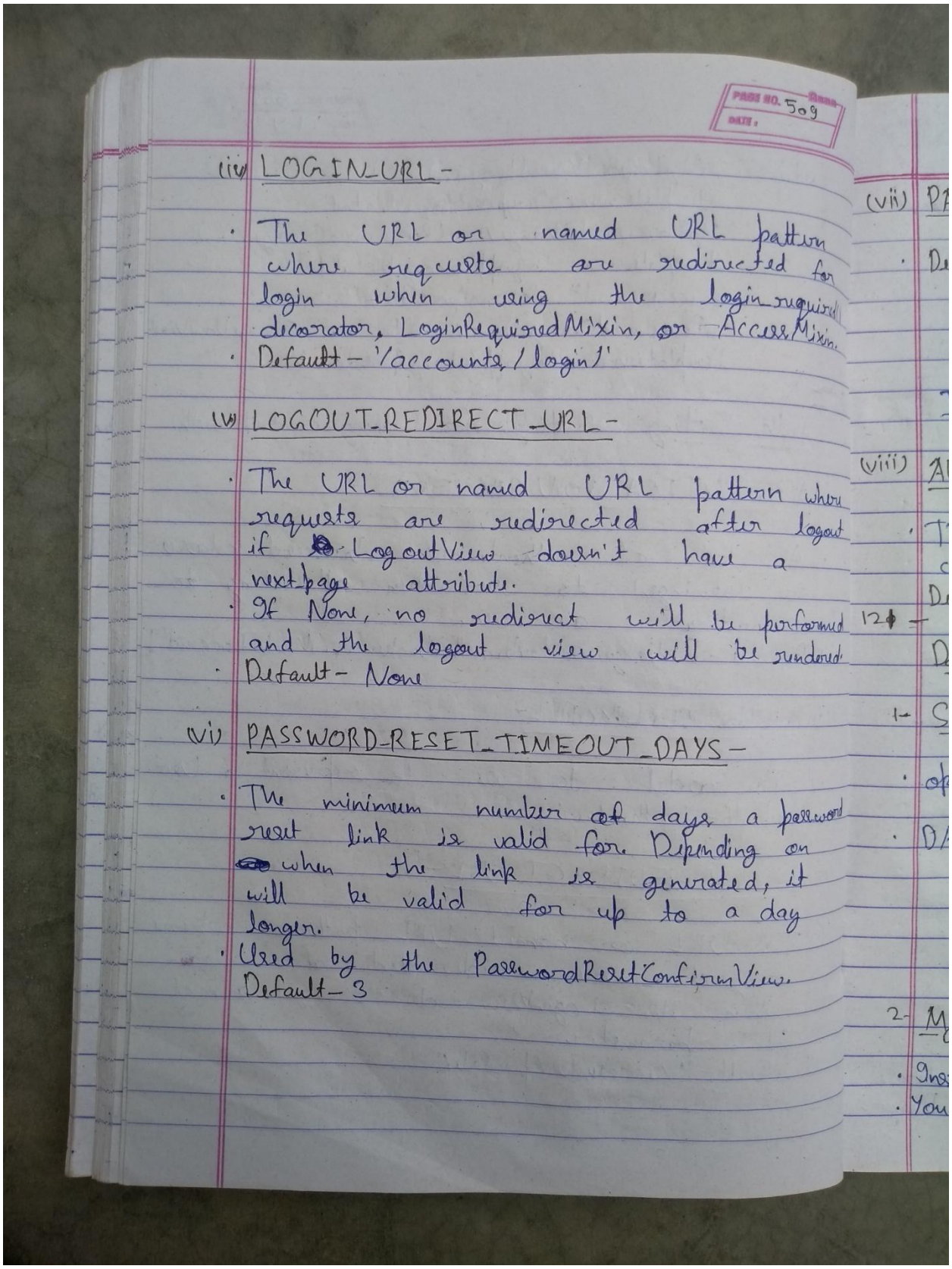
- A list of authentication backend classes (as strings) to use when attempting to authenticate a user.
- Default - ['django.contrib.auth.backends.ModelBackend']

(ii) AUTH_USER_MODEL -

- The model to use to represent a User.
- Default - 'auth.User'

(iii) LOGIN_REDIRECT_URL -

- The URL or named URL pattern where requests are redirected after login when the LoginView doesn't get a next GET parameter.
- Default - '/accounts/profile/'



(iv) LOGIN_URL -

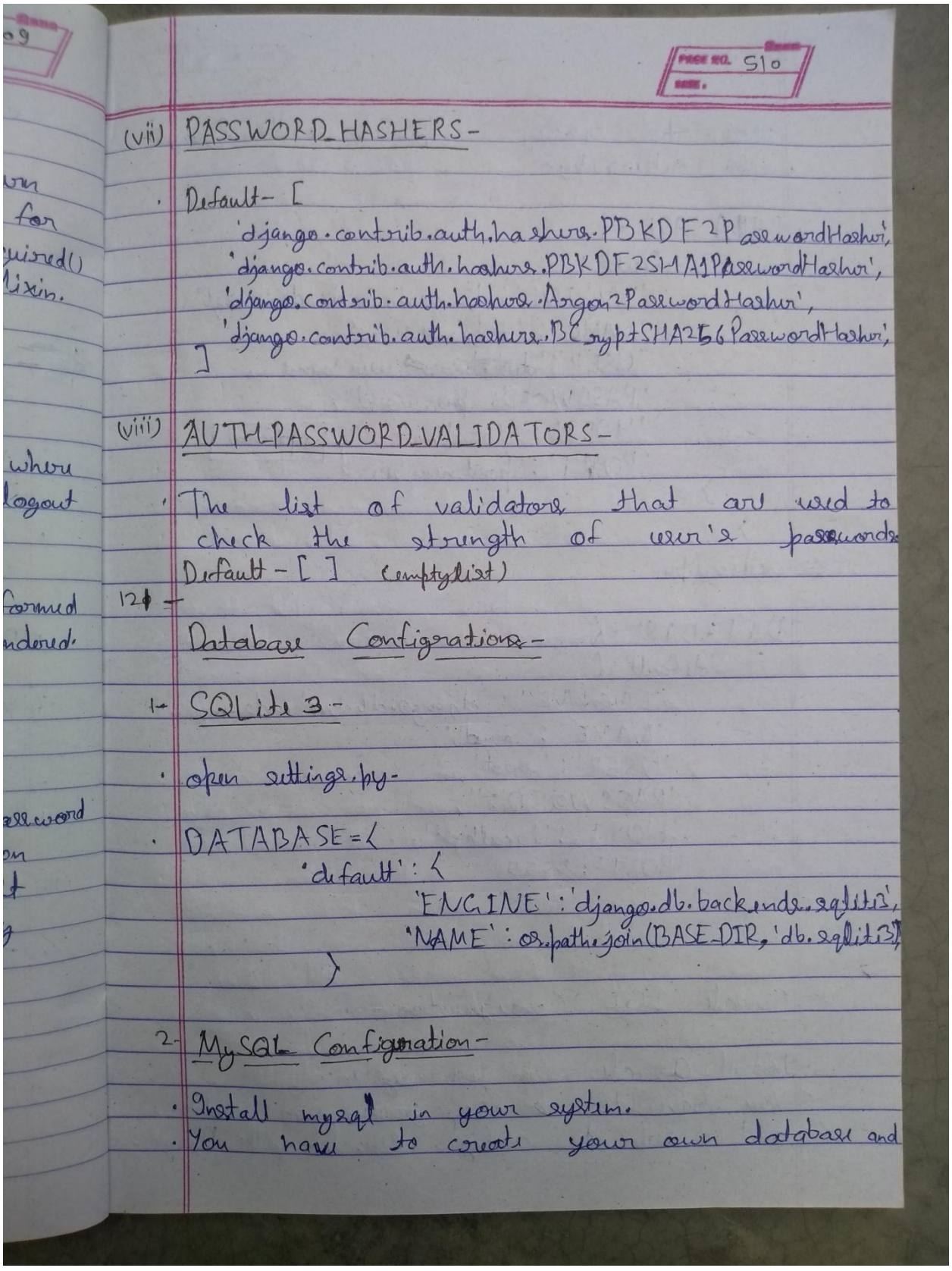
- The URL or named URL pattern where requests are redirected for login when using the login_required decorator, LoginRequiredMixin, or AccessMixin.
- Default - '/accounts/login/'

(v) LOGOUT_REDIRECT_URL -

- The URL or named URL pattern where requests are redirected after logout if LogoutView doesn't have a next_page attribute.
- If None, no redirect will be performed and the logout view will be rendered.
- Default - None

(vi) PASSWORD_RESET_TIMEOUT_DAYS -

- The minimum number of days a password reset link is valid for. Depending on when the link is generated, it will be valid for up to a day longer.
- Used by the PasswordResetConfirmView.
- Default - 3



(vii) PASSWORD_HASHERS -

- Default - [
  'django.contrib.auth.hashers.PBKDF2PasswordHasher',
  'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
  'django.contrib.auth.hashers.Argon2PasswordHasher',
  'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
  ]

(viii) AUTH_PASSWORD_VALIDATORS -

- The list of validators that are used to check the strength of user's passwords.
  Default - [ ] (empty list)

12) -

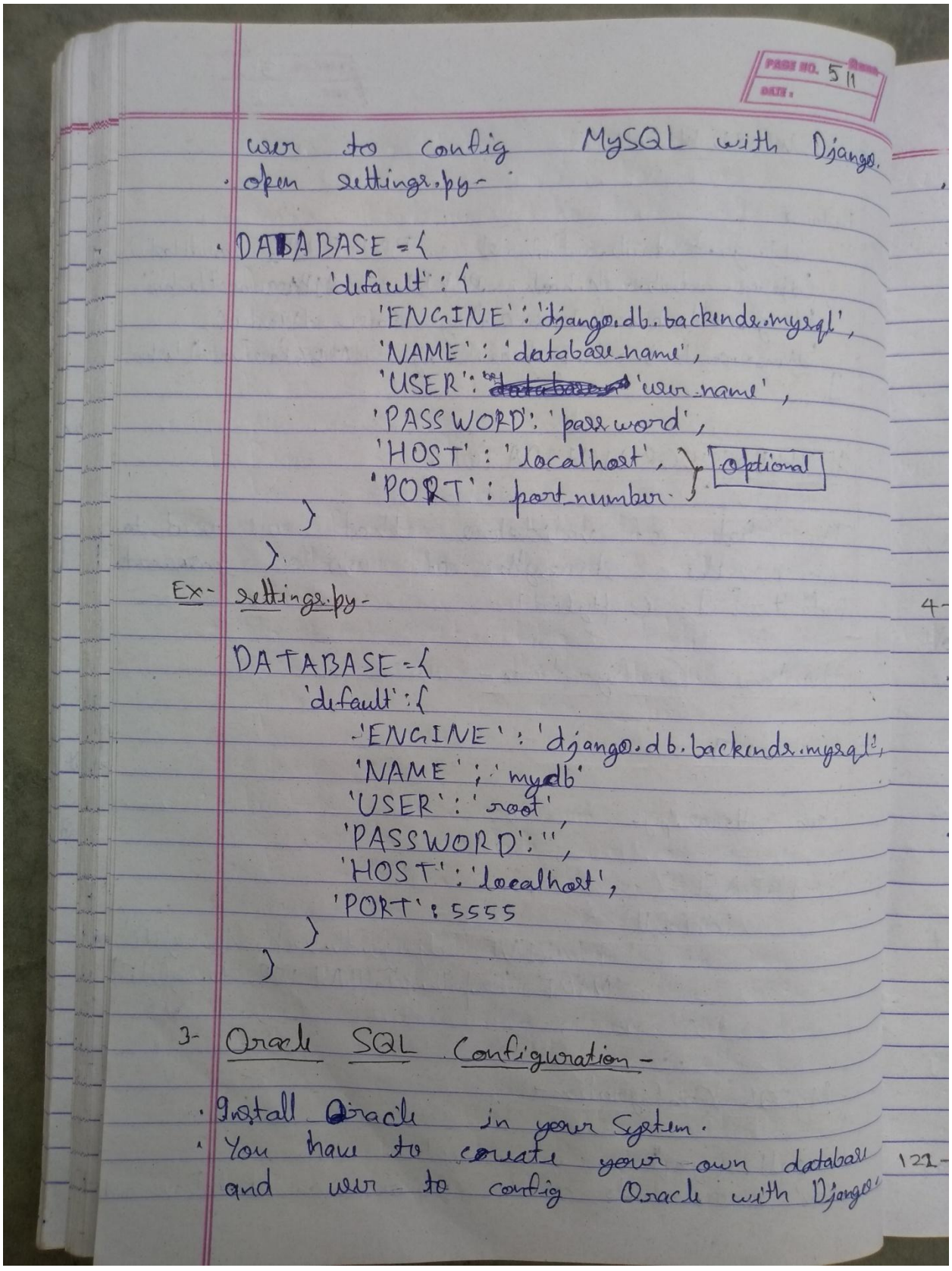
Database Configurations -

1- SQLite 3 -

- open settings.py -
- DATABASE = {
  'default': {
  'ENGINE': 'django.db.backends.sqlite3',
  'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
  }

2- MySQL Configuration -

- Install mysql in your system.
- You have to create your own database and



PAGE NO. 511
DATE :

use to config MySQL with Django.

- open settings.py -

```
DATABASE = {
    'default' : {
        'ENGINE' : 'django.db.backends.mysql',
        'NAME' : 'database_name',
        'USER' : 'user_name',
        'PASSWORD' : 'password',
        'HOST' : 'localhost',
        'PORT' : port_number
    }
}
```

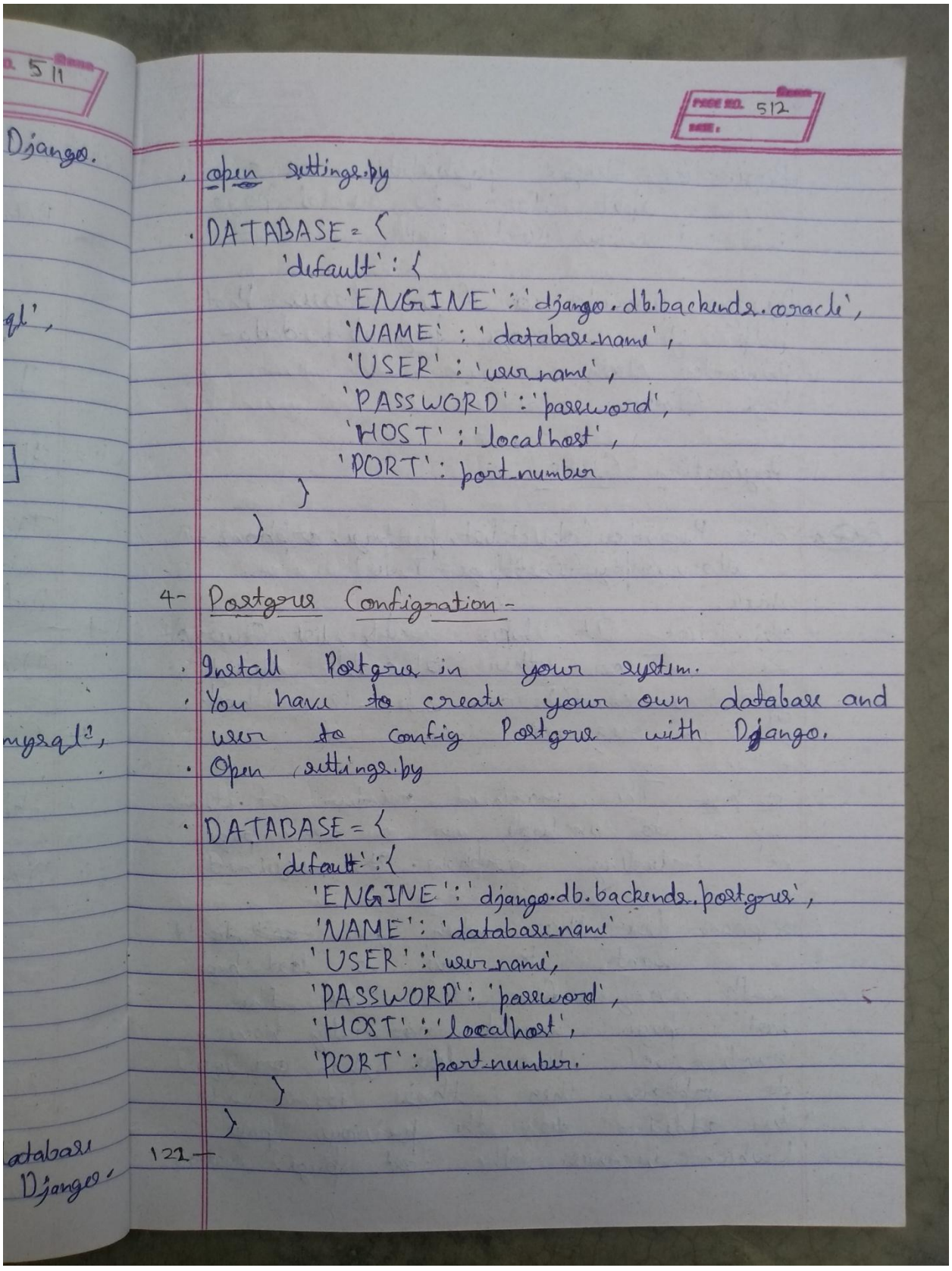
optional

Ex- settings.py -

```
DATABASE = {
    'default' : {
        'ENGINE' : 'django.db.backends.mysql',
        'NAME' : 'mydb'
        'USER' : 'root'
        'PASSWORD' : '',
        'HOST' : 'localhost',
        'PORT' : 5555
    }
}
```

3- Oracle SQL Configuration -

- Install Oracle in your System.
- You have to create your own database and user to config Oracle with Django.



. open settings.py

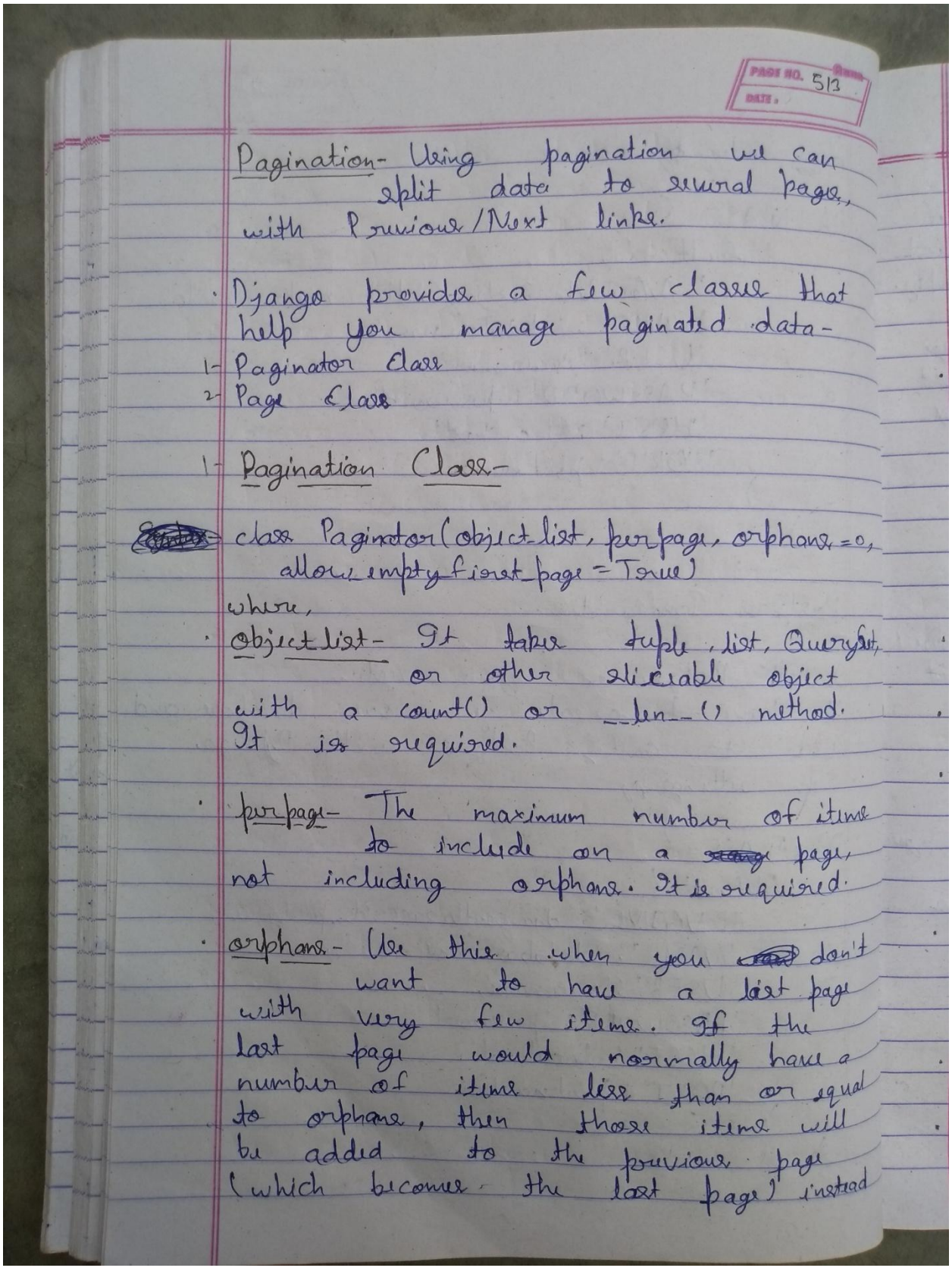
```
. DATABASE = {
      'default' : {
          'ENGINE' : 'django.db.backends.oracle',
          'NAME' : 'database_name',
          'USER' : 'user_name',
          'PASSWORD' : 'password',
          'HOST' : 'localhost',
          'PORT' : port_number
      }
  }
```

4- Postgres Configration -

. Install Postgres in your system.
. You have to create your own database and user to config Postgres with Django.
. Open settings.py

```
. DATABASE = {
      'default' : {
          'ENGINE' : 'django.db.backends.postgres',
          'NAME' : 'database_name'
          'USER' : 'user_name',
          'PASSWORD' : 'password',
          'HOST' : 'localhost',
          'PORT' : port_number.
      }
  }
```

121 —



PAGE NO. 513
DATE :

Pagination- Using pagination we can split data to several page, with Previous/Next links.

- Django provides a few classes that help you manage paginated data-

1- Paginator class
2- Page class

1- Pagination Class-

class Paginator(object_list, per_page, orphans=0, allow_empty_first_page=True)

where,

- object_list- It takes tuple, list, QuerySet or other sliceable object with a count() or __len__() method. It is required.
- per_page- The maximum number of items to include on a page, not including orphans. It is required.
- orphans- Use this when you don't want to have a last page with very few items. If the last page would normally have a number of items less than or equal to orphans, then those items will be added to the previous page (which becomes the last page) instead

of having the items on a page by themselves. orphans defaults to zero, which means pages are never combined and the last page may have one item. It is optional.

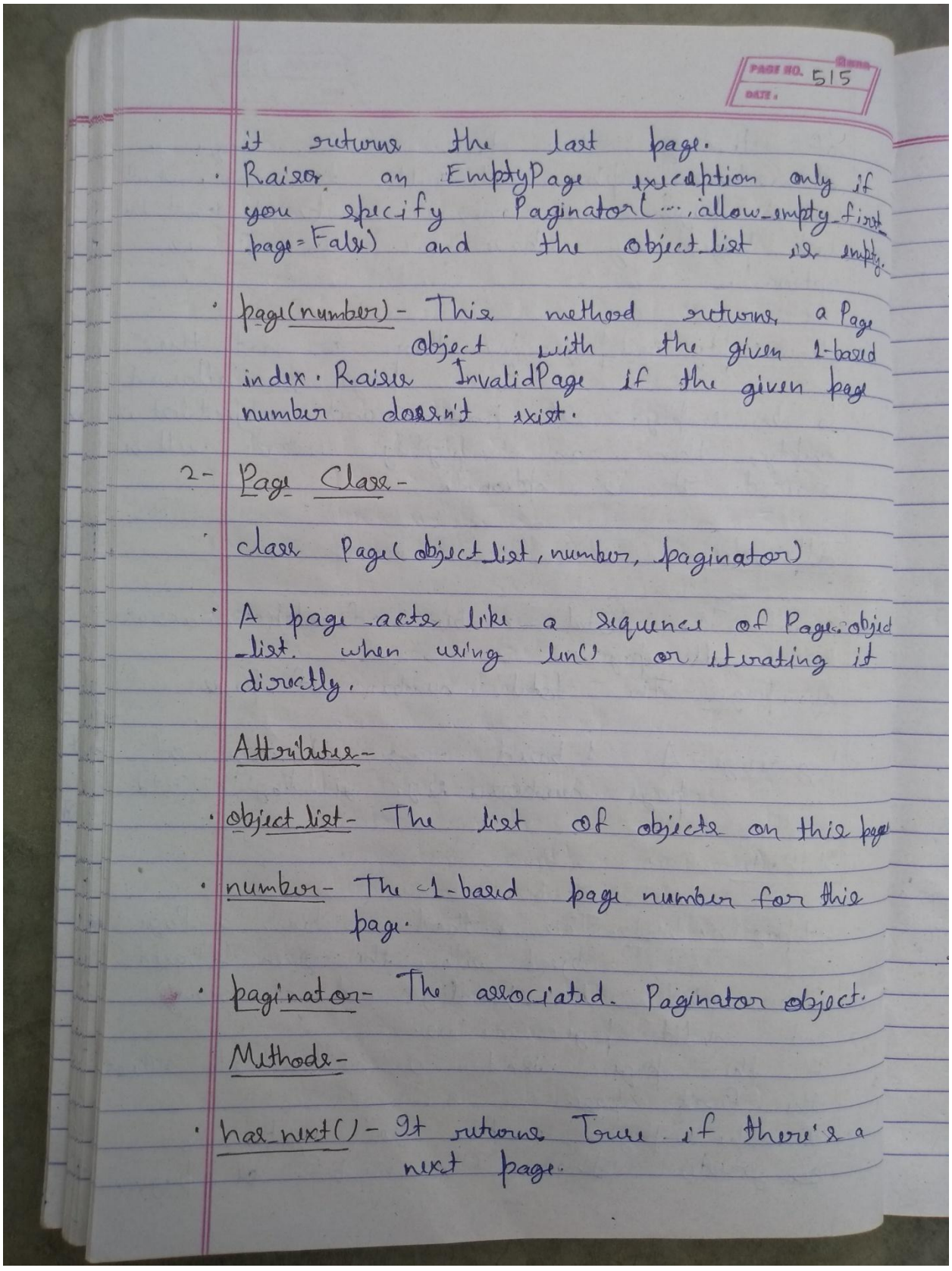
- allow_empty_first_page - Whether or not the first page is allowed to be empty. If False and object_list is empty, then an EmptyPage error will be raised. It is optional.

Attributes -

- count - The total number of objects, across all pages.
- num_pages - The total number of pages
- page_range - A 1-based range iterator of page numbers, e.g. yielding [1, 2, 3, 4].

Methods -

- get_page(number) - This method returns a Page object with the given 1-based index, while also handling out of range and invalid page numbers.
- If the page isn't a number, it returns the first page.
- If the page number is negative or greater than the number of pages,



it returns the last page.

- Raise an EmptyPage exception only if you specify Paginator(..., allow_empty_first_page=False) and the object_list is empty.

- page(number) - This method returns a Page object with the given 1-based index. Raise InvalidPage if the given page number doesn't exist.

2- Page Class -

- class Page(object_list, number, paginator)

- A page acts like a sequence of Page.object_list. when using len() or iterating it directly.

Attributes -

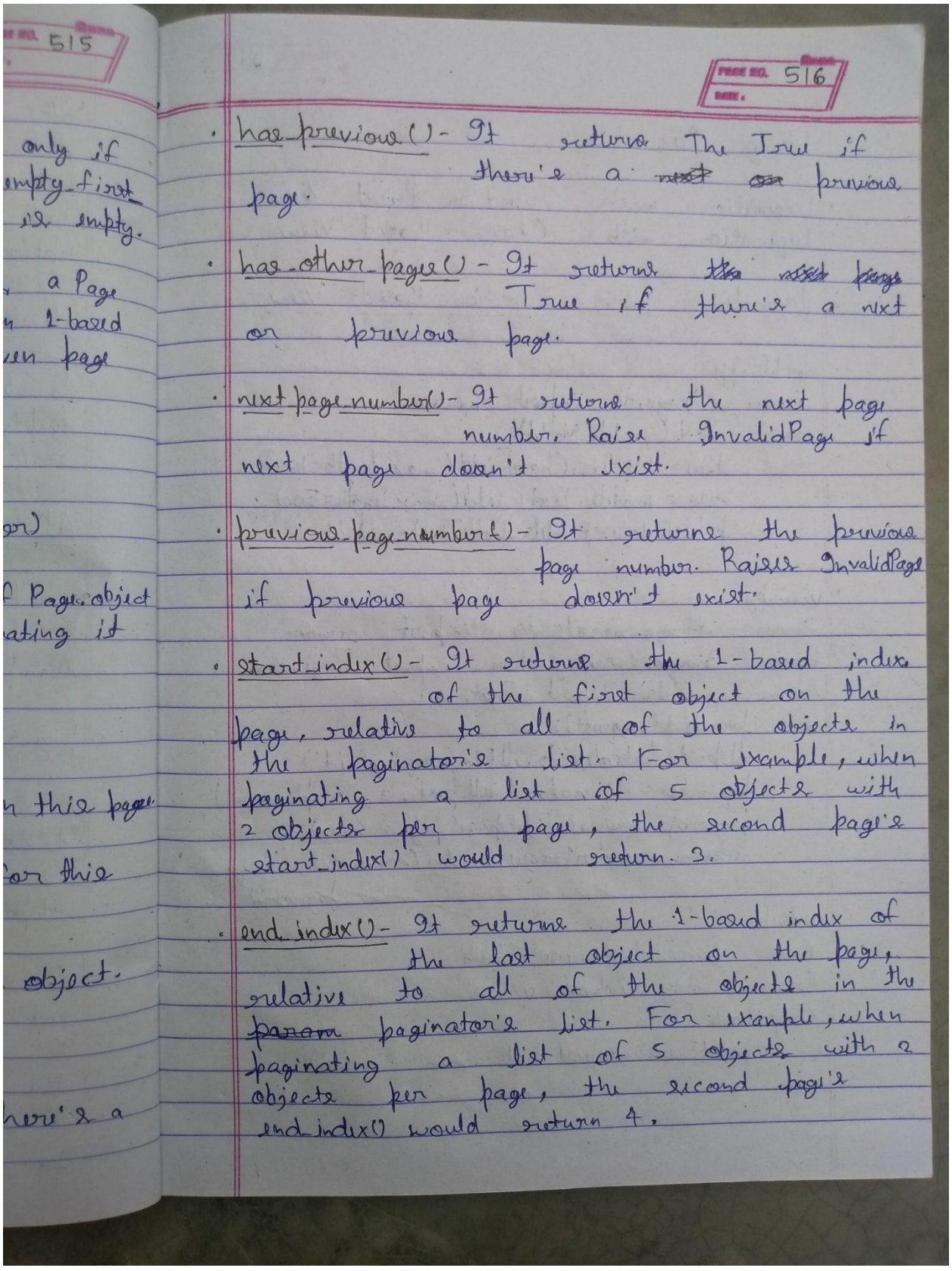
- object_list - The list of objects on this page

- number - The 1-based page number for this page.

- paginator - The associated Paginator object.

Methods -

- has_next() - It returns True if there's a next page.



PAGE NO. 515

only if
empty_first
is empty.

a Page
1-based
en page

or)

Page.object
ating it

n this page.

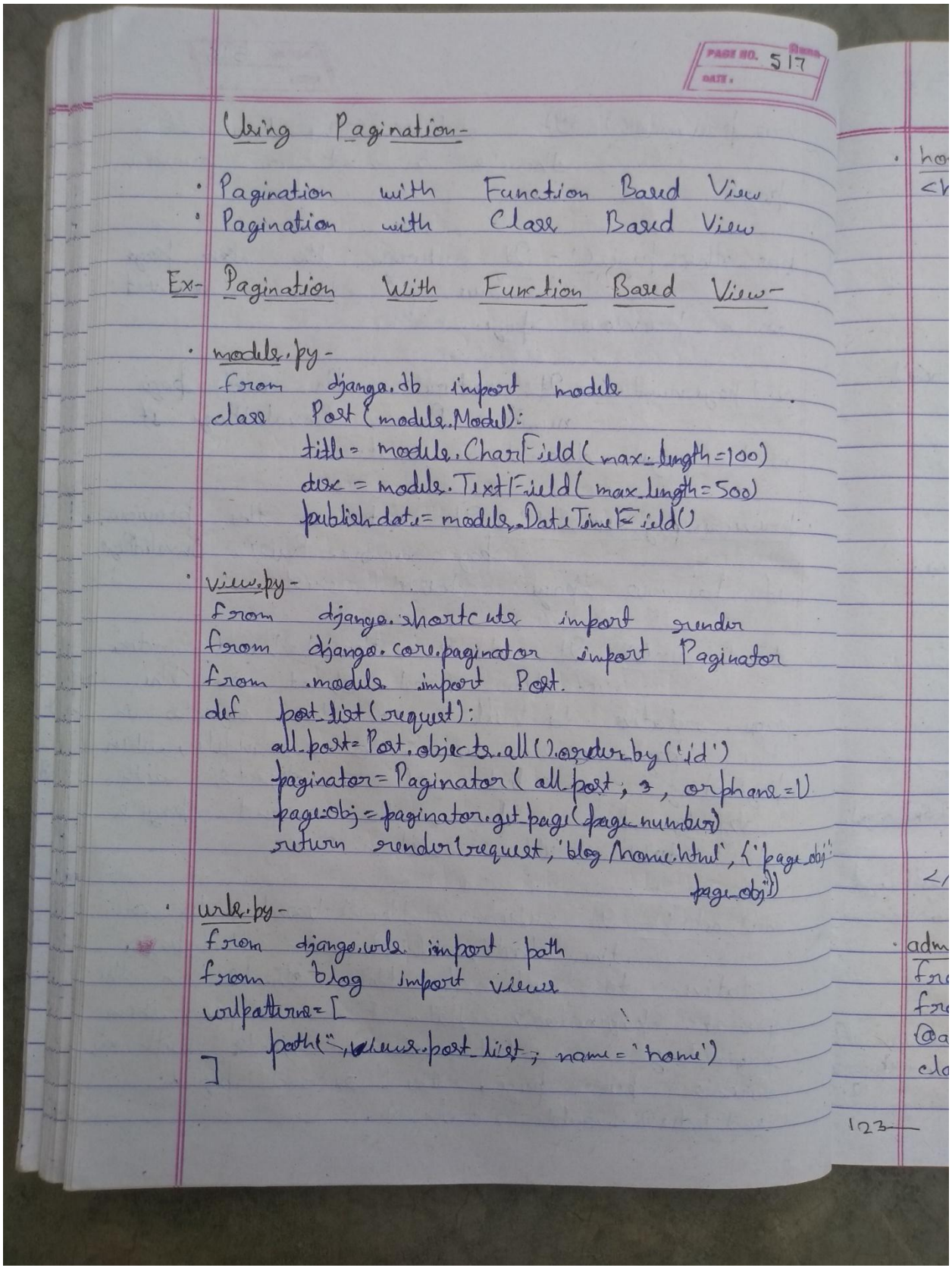
For this

object.

here's a

PAGE NO. 516

- has_previous() - It returns True if there's a ~~next~~ ~~or~~ previous page.
- has_other_pages() - It returns ~~the next page~~ True if there's a next or previous page.
- next_page_number() - It returns the next page number. Raise InvalidPage if next page doesn't exist.
- previous_page_number() - It returns the previous page number. Raises InvalidPage if previous page doesn't exist.
- start_index() - It returns the 1-based index of the first object on the page, relative to all of the objects in the paginator's list. For example, when paginating a list of 5 objects with 2 objects per page, the second page's start_index() would return 3.
- end_index() - It returns the 1-based index of the last object on the page, relative to all of the objects in the ~~param~~ paginator's list. For example, when paginating a list of 5 objects with 2 objects per page, the second page's end_index() would return 4.



PAGE NO. 517
DATE :

Using Pagination-

- Pagination with Function Based View
- Pagination with Class Based View

Ex- Pagination With Function Based View-

- models.py-

```
from django.db import models
class Post(models.Model):
    title = models.CharField(max_length=100)
    desc = models.TextField(max_length=500)
    publish_date = models.DateTimeField()
```

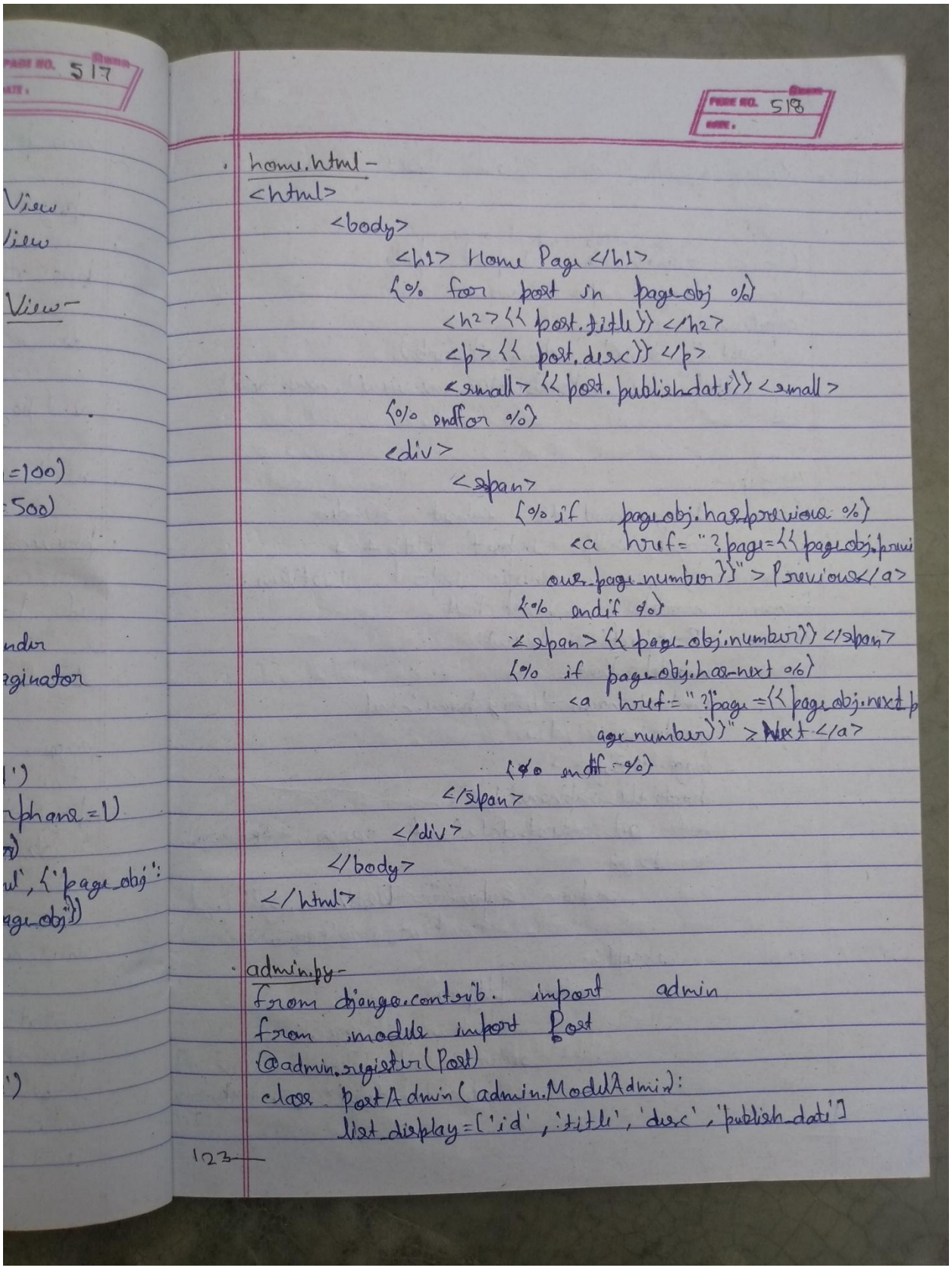
- views.py-

```
from django.shortcuts import render
from django.core.paginator import Paginator
from .models import Post.
def post_list(request):
    all_post = Post.objects.all().order_by('id')
    paginator = Paginator(all_post, 3, orphans=1)
    page_obj = paginator.get_page(page_number)
    return render(request, 'blog/home.html', {'page_obj':
                                              page_obj})
```

- urls.py-

```
from django.urls import path
from blog import views
urlpatterns = [
    path('', views.post_list, name='home')
]
```

123



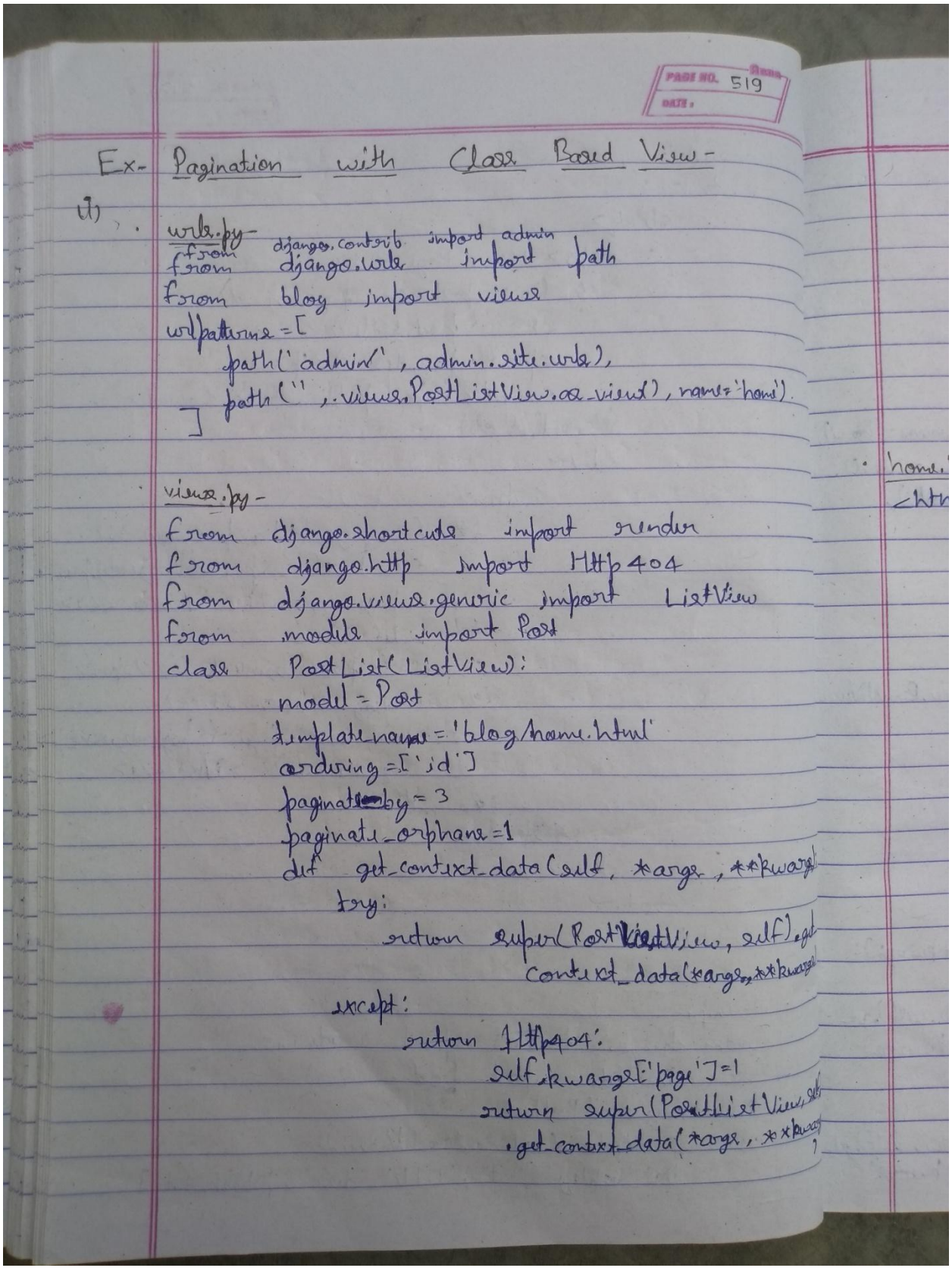
home.html -

```
<html>
    <body>
        <h1> Home Page </h1>
        {% for post in page_obj %}
            <h2> {{ post.title }} </h2>
            <p> {{ post.desc }} </p>
            <small> {{ post.publish_date }} <small>
        {% endfor %}
        <div>
            <span>
                {% if page_obj.has_previous %}
                <a href="?page={{ page_obj.previous_page_number }}"> Previous </a>
                {% endif %}
                <span> {{ page_obj.number }} </span>
                {% if page_obj.has_next %}
                <a href="?page={{ page_obj.next_page_number }}"> Next </a>
                {% endif %}
            </span>
        </div>
    </body>
</html>
```

admin.py -

```
from django.contrib. import admin
from .models import Post
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display=['id', 'title', 'desc', 'publish_date']
```

123



PAGE NO. 519
DATE :

Ex- Pagination with Class Based View -

(1)

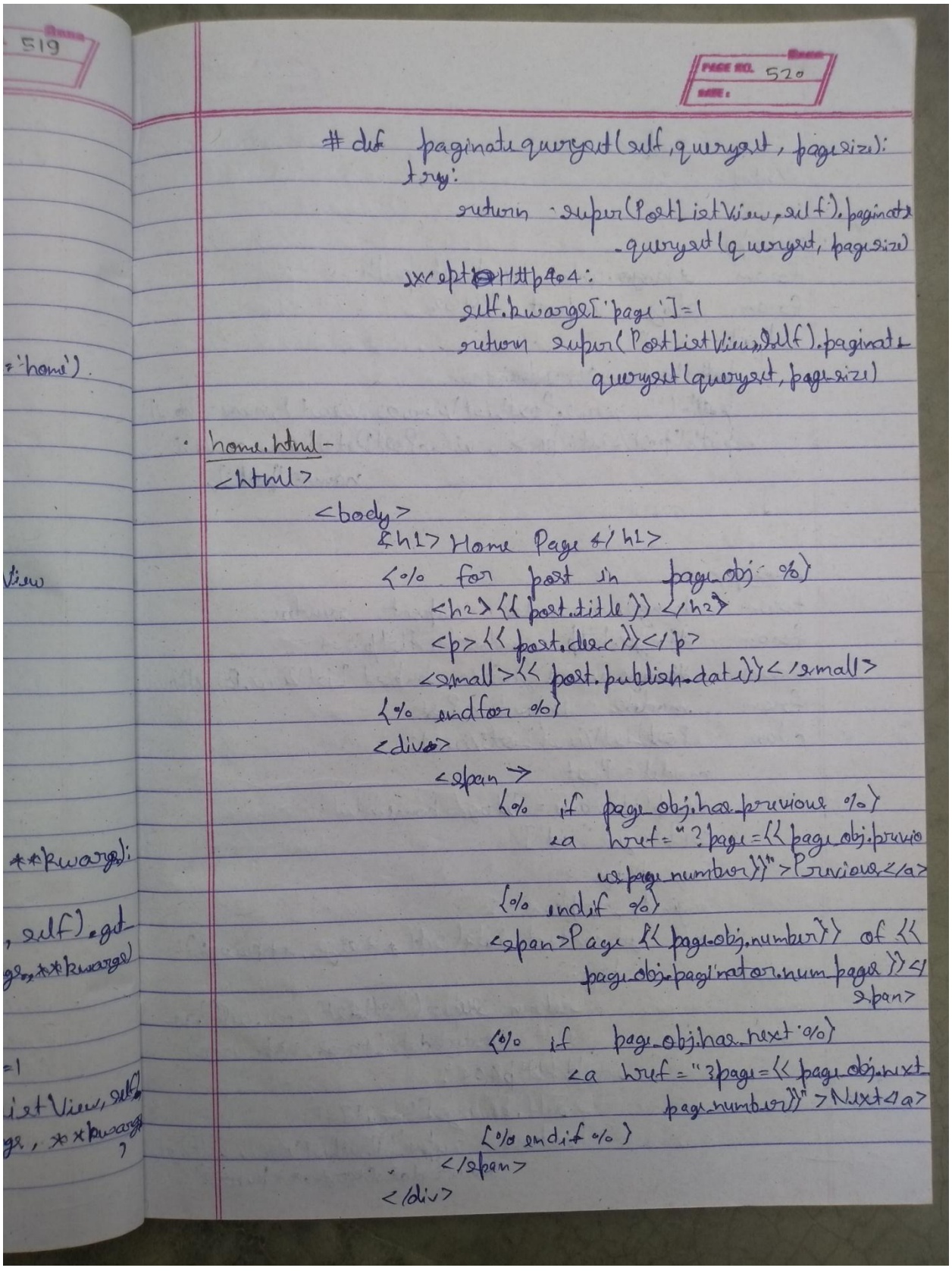
urls.py -

```
from django.contrib import admin
from django.urls import path
from blog import views
urlpatterns = [
    path('admin', admin.site.urls),
    path('', views.PostListView.as_view(), name='home'),
]
```

views.py -

```
from django.shortcuts import render
from django.http import Http404
from django.views.generic import ListView
from .models import Post
class PostList(ListView):
    model = Post
    template_name = 'blog/home.html'
    ordering = ['id']
    paginate_by = 3
    paginate_orphans = 1
    def get_context_data(self, *args, **kwargs
        try:
            return super(PostListView, self).get
                context_data(*args, **kwarg
        except:
            return Http404:
                self.kwargs['page'] = 1
                return super(PostListView, self
                .get_context_data(*args, **kwarg
```

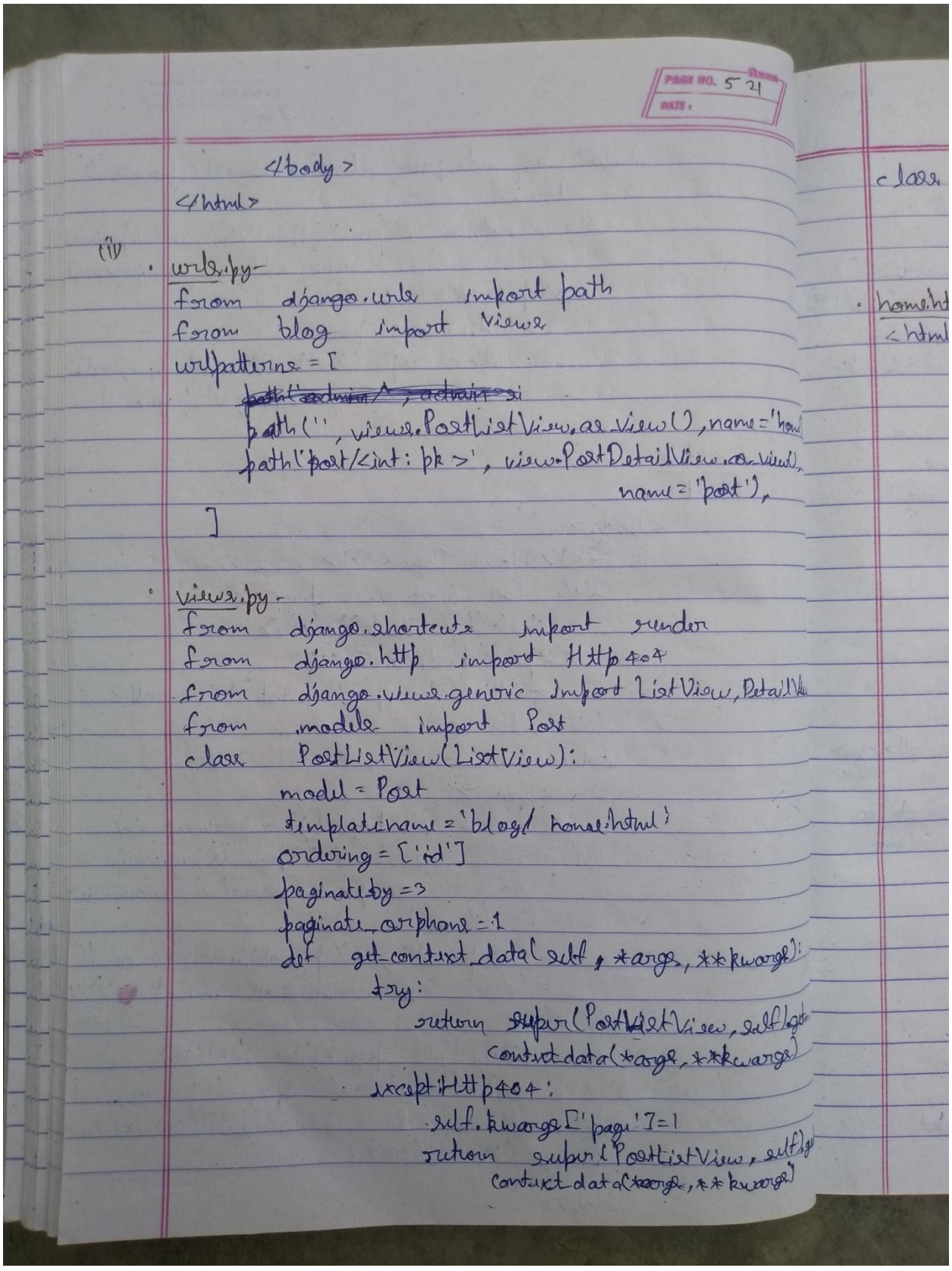
'home.
<htm



```
# def paginate_queryset(self, queryset, page_size):
    try:
        return super(PostListView, self).paginate_
            _queryset(queryset, page_size)
    except Http404:
        self.kwargs['page'] = 1
        return super(PostListView, self).paginate_
            queryset(queryset, page_size)
```

home.html -

```
<html>
    <body>
        <h1> Home Page </h1>
        {% for post in page_obj %}
            <h2> {{ post.title }} </h2>
            <p> {{ post.desc }} </p>
            <small> {{ post.publish_date }} </small>
        {% endfor %}
        <div>
            <span>
                {% if page_obj.has_previous %}
                    <a href="?page={{ page_obj.previo
                    us_page_number }}"> Previous </a>
                {% endif %}
                <span> Page {{ page_obj.number }} of {{
                    page_obj.paginator.num_pages }} </
                    span>
                {% if page_obj.has_next %}
                    <a href="?page={{ page_obj.next_
                    page_number }}"> Next </a>
                {% endif %}
            </span>
        </div>
```



```
        </body>
</html>
```

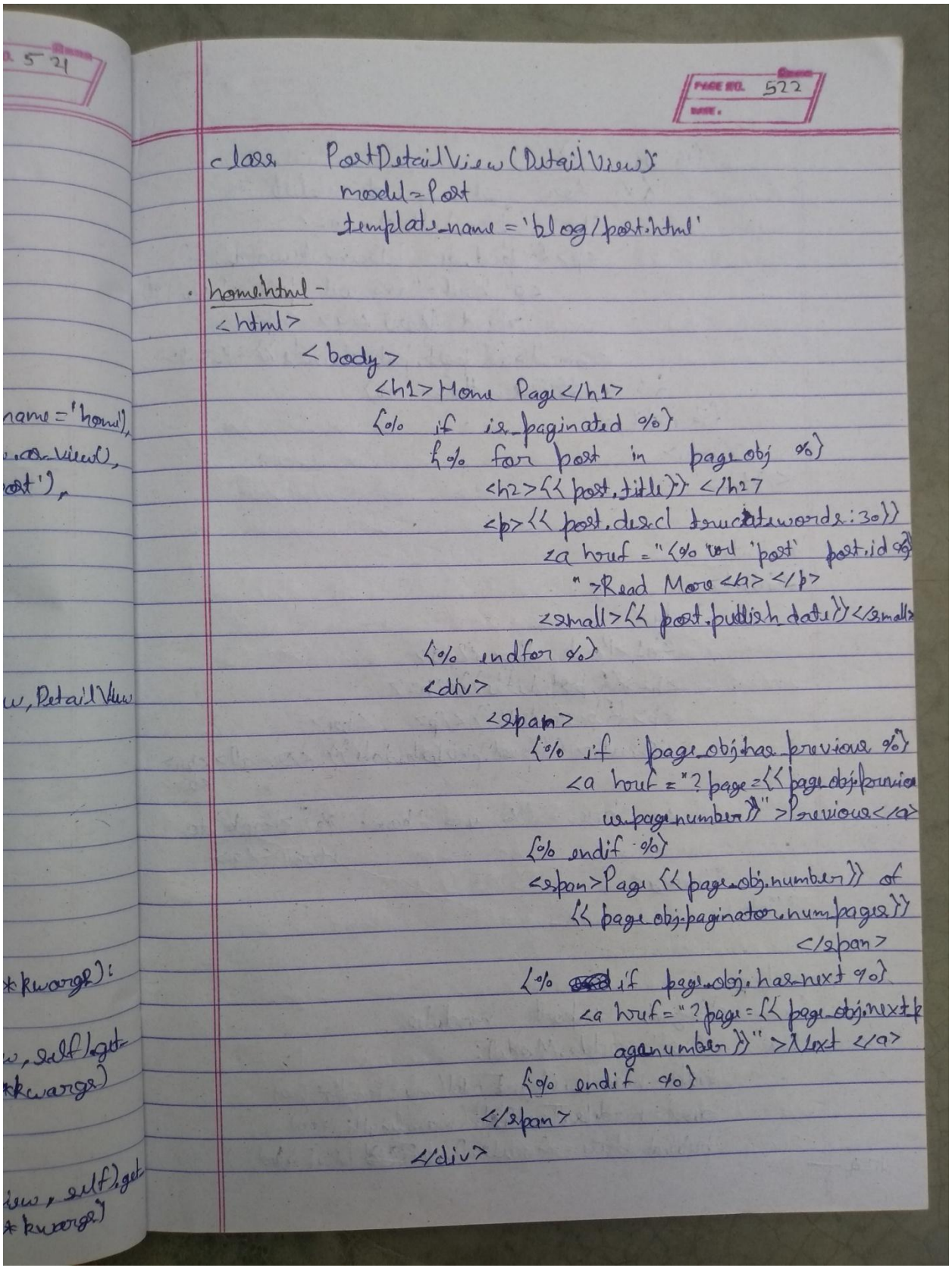
(iv)

- urls.py -

```
from django.urls import path
from blog import views
urlpatterns = [
    path('', views.PostListView.as_view(), name='home'),
    path('post/<int:pk>', views.PostDetailView.as_view(),
                          name='post'),
]
```

- views.py -

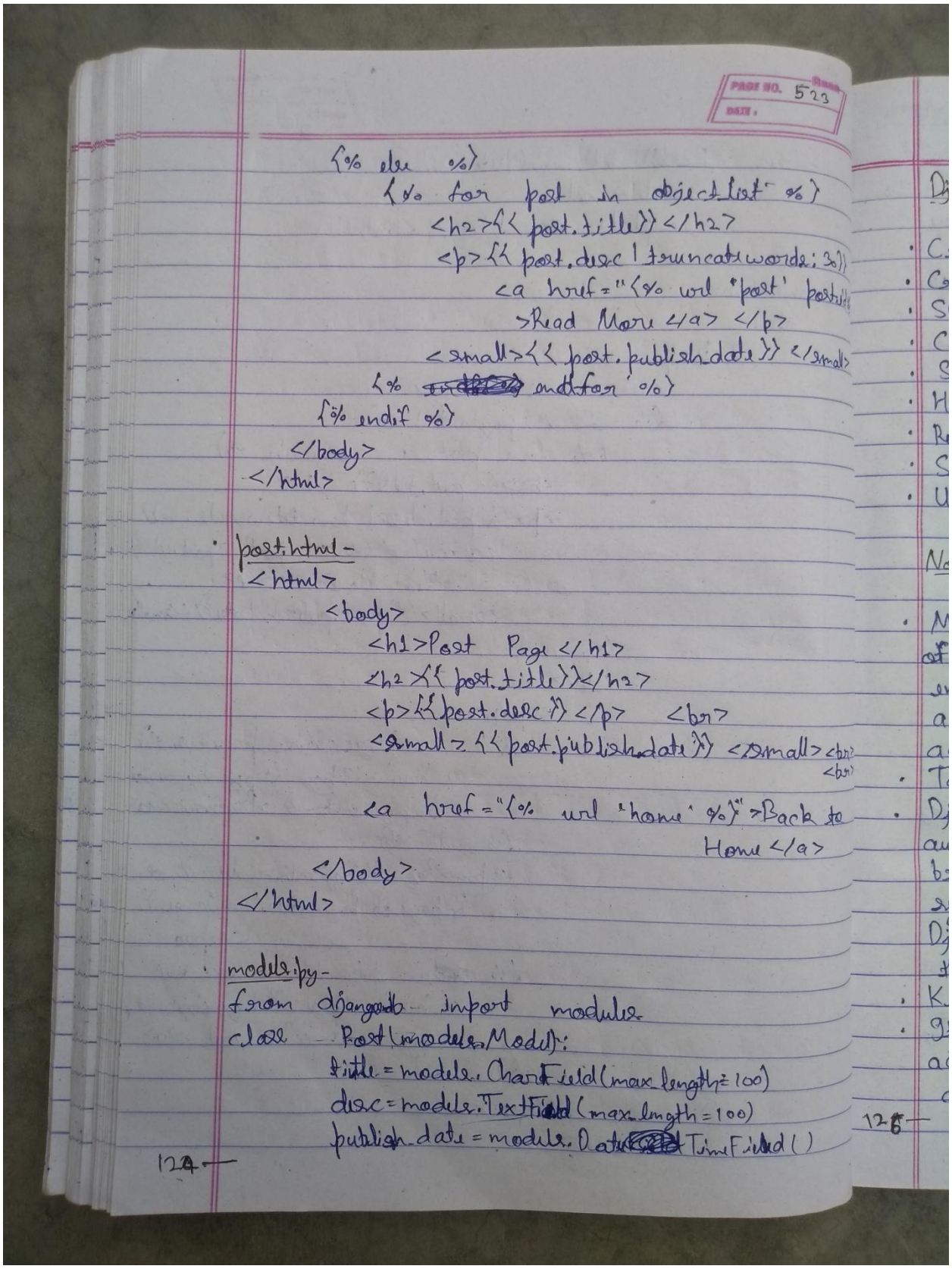
```
from django.shortcuts import render
from django.http import Http404
from django.views.generic import ListView, DetailView
from .models import Post
class PostListView(ListView):
    model = Post
    template_name = 'blog/home.html'
    ordering = ['id']
    paginate_by = 3
    paginate_orphans = 1
    def get_context_data(self, *args, **kwargs):
        try:
            return super(PostListView, self).get_
                context_data(*args, **kwargs)
        except Http404:
            self.kwargs['page'] = 1
            return super(PostListView, self).get_
                context_data(*args, **kwargs)
```



```
class PostDetailView(DetailView):
    model = Post
    template_name = 'blog/post.html'
```

- home.html -

```
<html>
    <body>
        <h1>Home Page</h1>
        {% if is_paginated %}
            {% for post in page_obj %}
                <h2>{{ post.title }}</h2>
                <p>{{ post.descl truncatewords:30 }}
                    <a href = "{% url 'post' post.id %}
                    ">Read More</a></p>
                <small>{{ post.publish_date }}</small>
            {% endfor %}
            <div>
                <span>
                    {% if page_obj.has_previous %}
                        <a href = "?page={{ page_obj.previous
                        .page_number }}" >Previous</a>
                    {% endif %}
                    <span>Page {{ page_obj.number }} of
                        {{ page_obj.paginator.num_pages }}
                                    </span>
                    {% if page_obj.has_next %}
                        <a href = "?page={{ page_obj.next_p
                        age_number }}" >Next</a>
                    {% endif %}
                </span>
            </div>
```



```
{% else %}
    {% for post in object_list %}
        <h2>{{ post.title }}</h2>
        <p>{{ post.desc | truncatewords:30 }}
            <a href="{% url 'post' post.title %}
            >Read More</a></p>
        <small>{{ post.publish_date }}</small>
    {% endfor %}
{% endif %}
</body>
</html>
```

post.html -

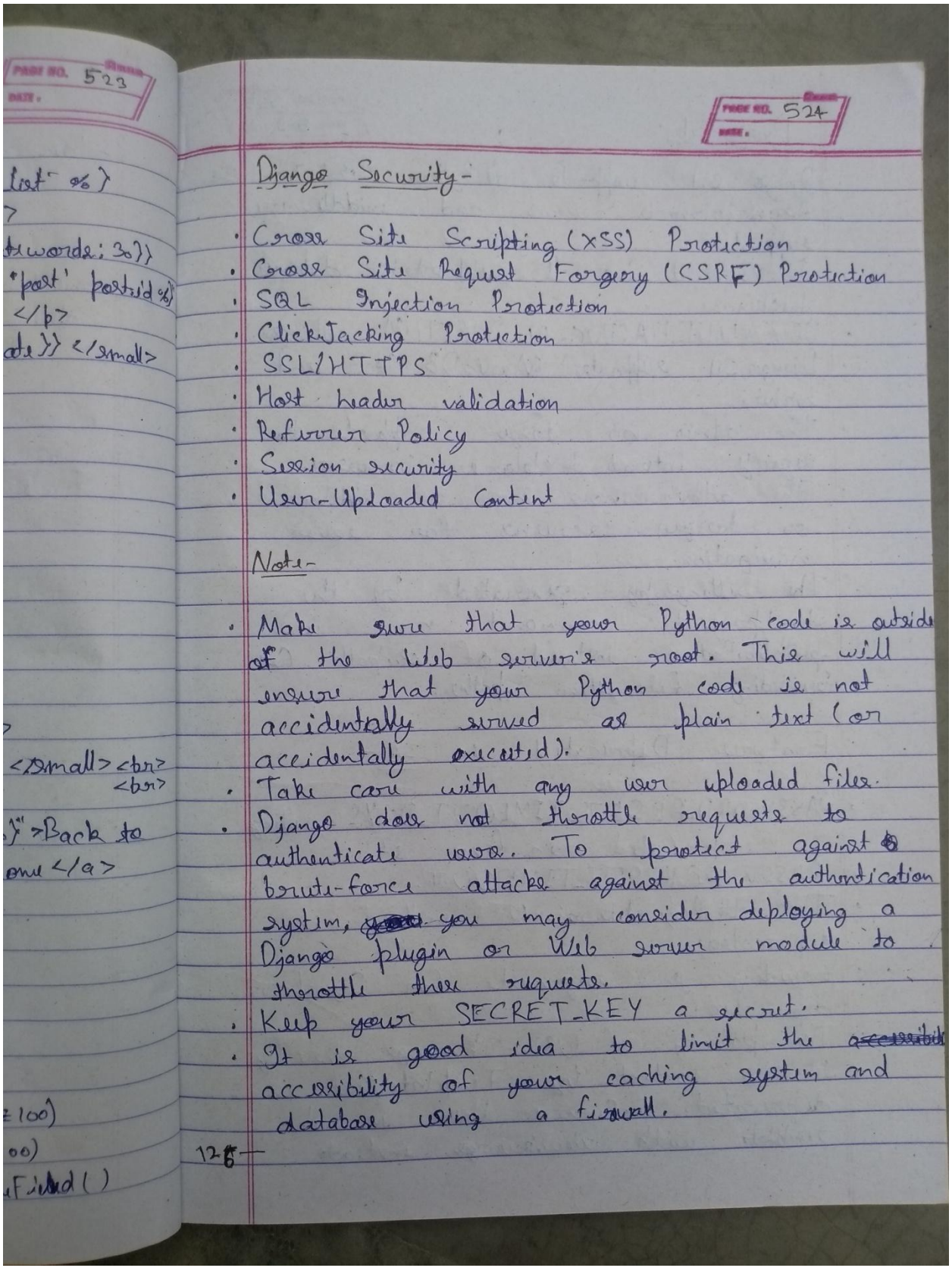
```
<html>
    <body>
        <h1>Post Page</h1>
        <h2>{{ post.title }}</h2>
        <p>{{ post.desc }}</p> <br>
        <small>{{ post.publish_date }}</small> <br> <br>

        <a href="{% url 'home' %}">Back to
                                    Home</a>
    </body>
</html>
```

models.py -

```
from django.db import models
class Post(models.Model):
    title = models.CharField(max_length=100)
    desc = models.TextField(max_length=100)
    publish_date = models.DateTimeField()
```

124

126



PAGE NO. 523

list %}
>
t words:30 }}
'post' post.id %}

ate }} </small>
>
<small>

}">Back to
one
=100)
00)
Find()

PAGE NO. 524

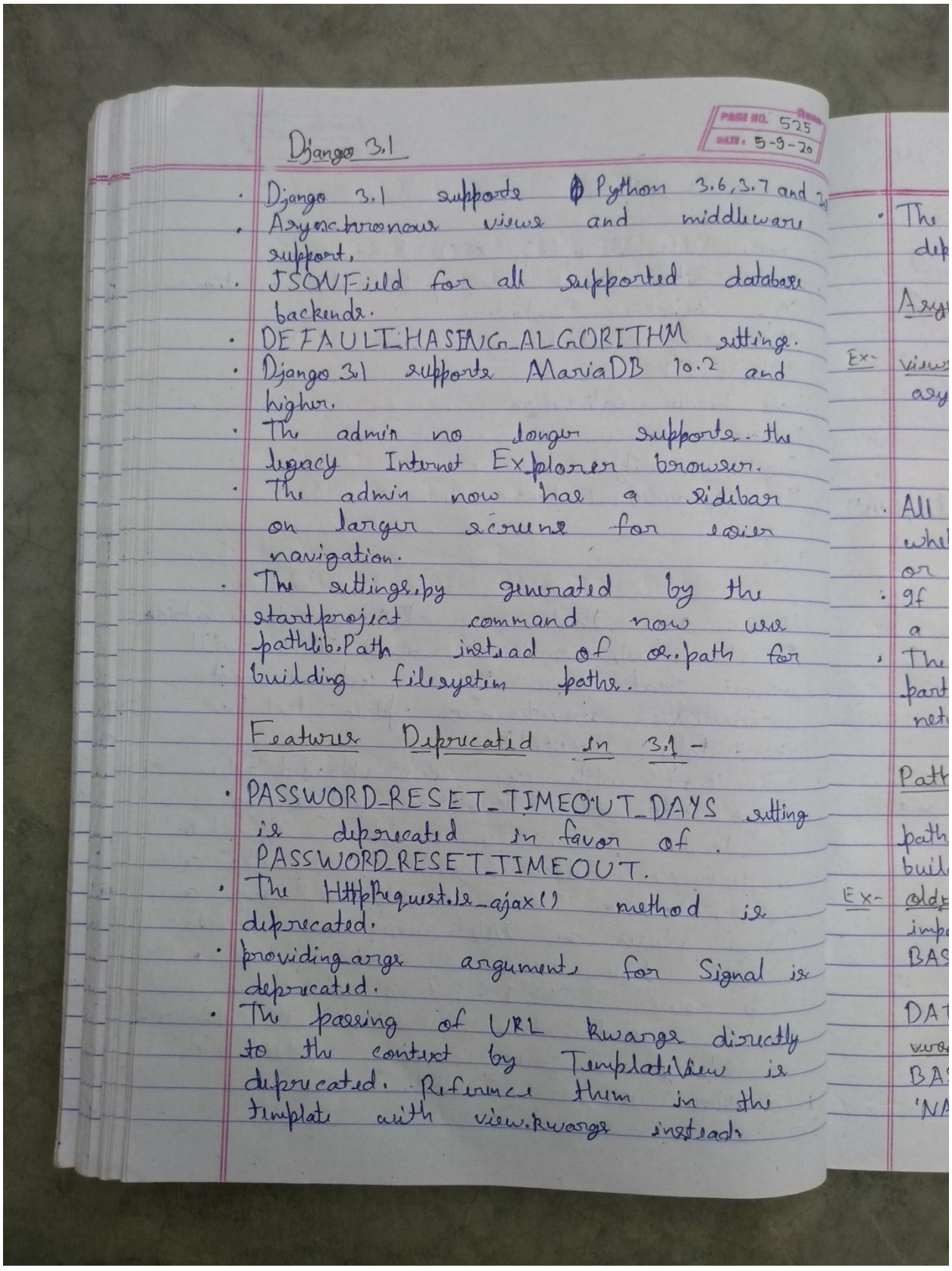
Django Security -

- Cross Site Scripting (XSS) Protection
- Cross Site Request Forgery (CSRF) Protection
- SQL Injection Protection
- ClickJacking Protection
- SSL/HTTPS
- Host header validation
- Referrer Policy
- Session security
- User-Uploaded Content

Note -

- Make sure that your Python code is outside of the web server's root. This will ensure that your Python code is not accidentally served as plain text (or accidentally executed).
- Take care with any user uploaded files.
- Django does not throttle requests to authenticate users. To protect against brute-force attacks against the authentication system, you may consider deploying a Django plugin or Web server module to throttle these requests.
- Keep your SECRET_KEY a secret.
- It is good idea to limit the accessibility of your caching system and database using a firewall.

125

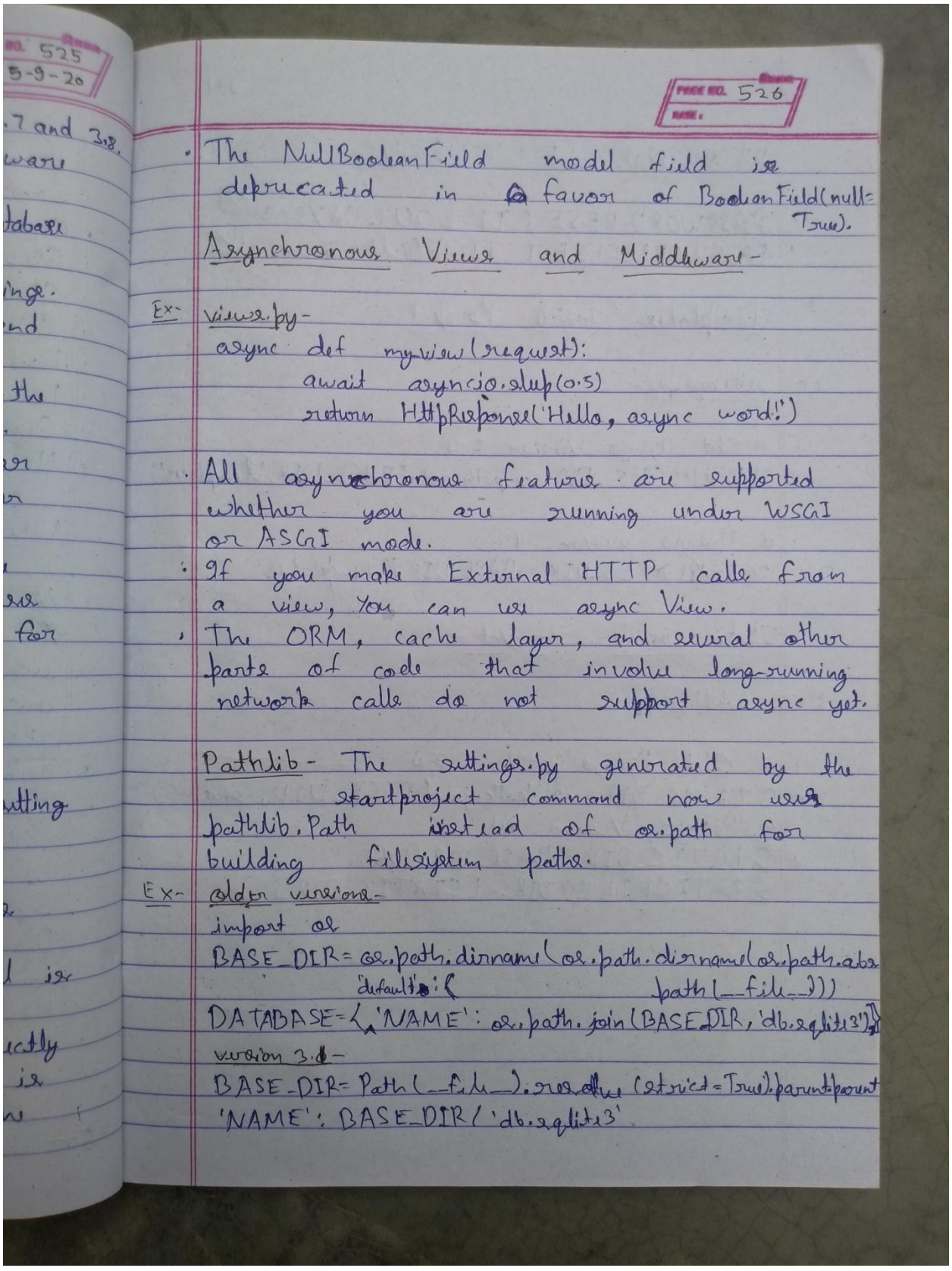


Django 3.1

- Django 3.1 supports Python 3.6, 3.7 and 3.
- Asynchronous views and middleware support.
- JSONField for all supported database backends.
- DEFAULT_HASHING_ALGORITHM setting.
- Django 3.1 supports MariaDB 10.2 and higher.
- The admin no longer supports the legacy Internet Explorer browser.
- The admin now has a sidebar on larger screens for easier navigation.
- The settings.py generated by the startproject command now use pathlib.Path instead of os.path for building filesystem paths.

Features Deprecated in 3.1 -

- PASSWORD_RESET_TIMEOUT_DAYS setting is deprecated in favor of PASSWORD_RESET_TIMEOUT.
- The HttpRequest.is_ajax() method is deprecated.
- providing args argument for Signal is deprecated.
- The passing of URL kwargs directly to the context by TemplateView is deprecated. Reference them in the template with view.kwargs instead.



- The NullBooleanField model field is deprecated in favor of BooleanField(null=True).

Asynchronous Views and Middleware -

Ex- views.py -

```
async def my_view(request):
    await asyncio.sleep(0.5)
    return HttpResponse('Hello, async world!')
```

- All asynchronous features are supported whether you are running under WSGI or ASGI mode.
- If you make External HTTP calls from a view, You can use async View.
- The ORM, cache layer, and several other parts of code that involve long-running network calls do not support async yet.

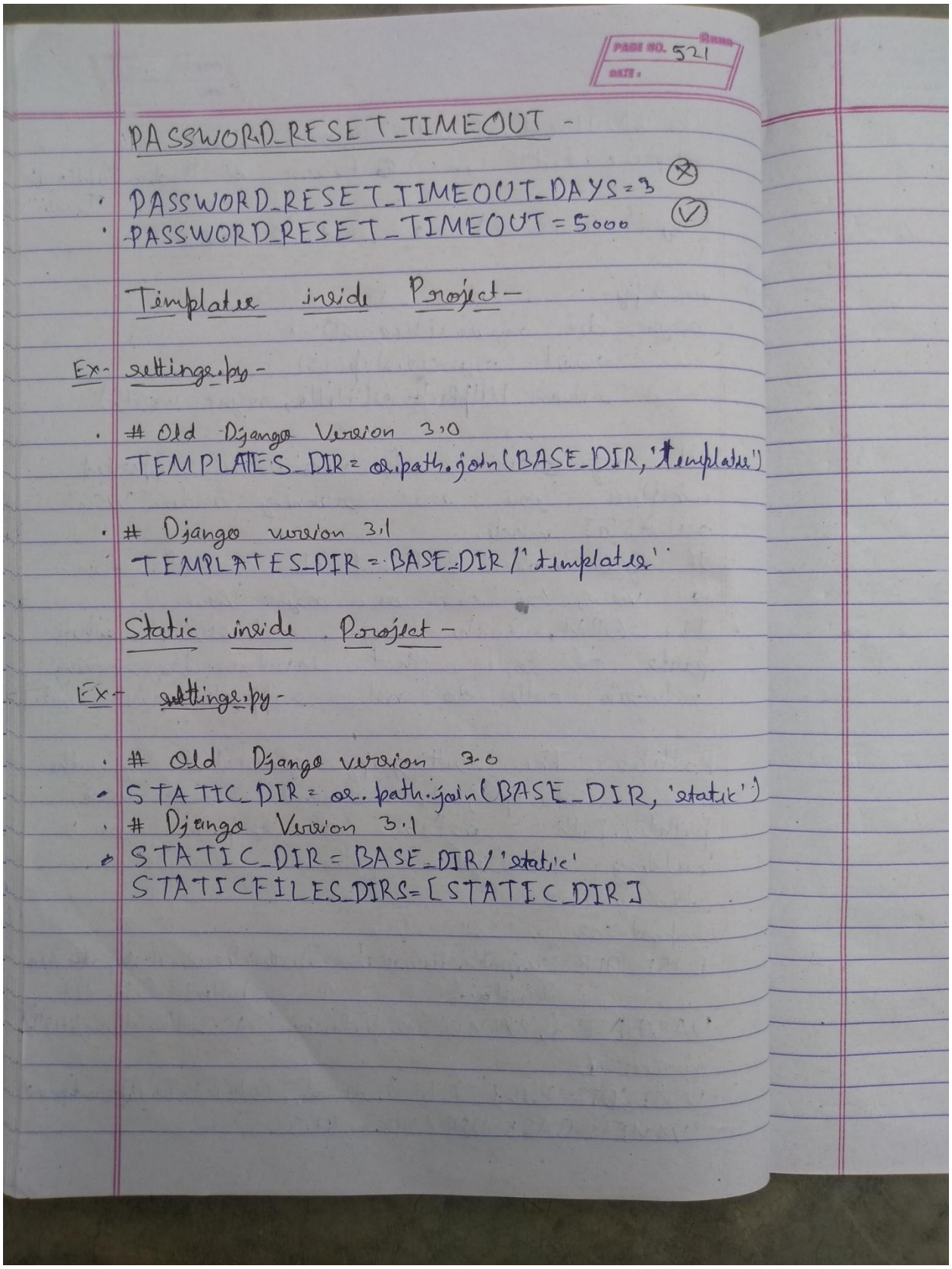
Pathlib - The settings.py generated by the startproject command now uses pathlib.Path instead of os.path for building filesystem paths.

Ex- older versions -

```
import os
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
DATABASES = {'default': {'NAME': os.path.join(BASE_DIR, 'db.sqlite3')}}
```

version 3.0 -

```
BASE_DIR = Path(__file__).resolve(strict=True).parent.parent
'NAME': BASE_DIR / 'db.sqlite3'
```



PAGE NO. 521
DATE :

PASSWORD_RESET_TIMEOUT -

- PASSWORD_RESET_TIMEOUT_DAYS = 3 ✗
- PASSWORD_RESET_TIMEOUT = 5000 ✓

Templates inside Project -

Ex- settings.py -

```
# Old Django Version 3.0
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')

# Django version 3.1
TEMPLATES_DIR = BASE_DIR / 'templates'
```

Static inside Project -

Ex- settings.py -

```
# Old Django version 3.0
STATIC_DIR = os.path.join(BASE_DIR, 'static')
# Django Version 3.1
STATIC_DIR = BASE_DIR / 'static'
STATICFILES_DIRS = [STATIC_DIR]
```