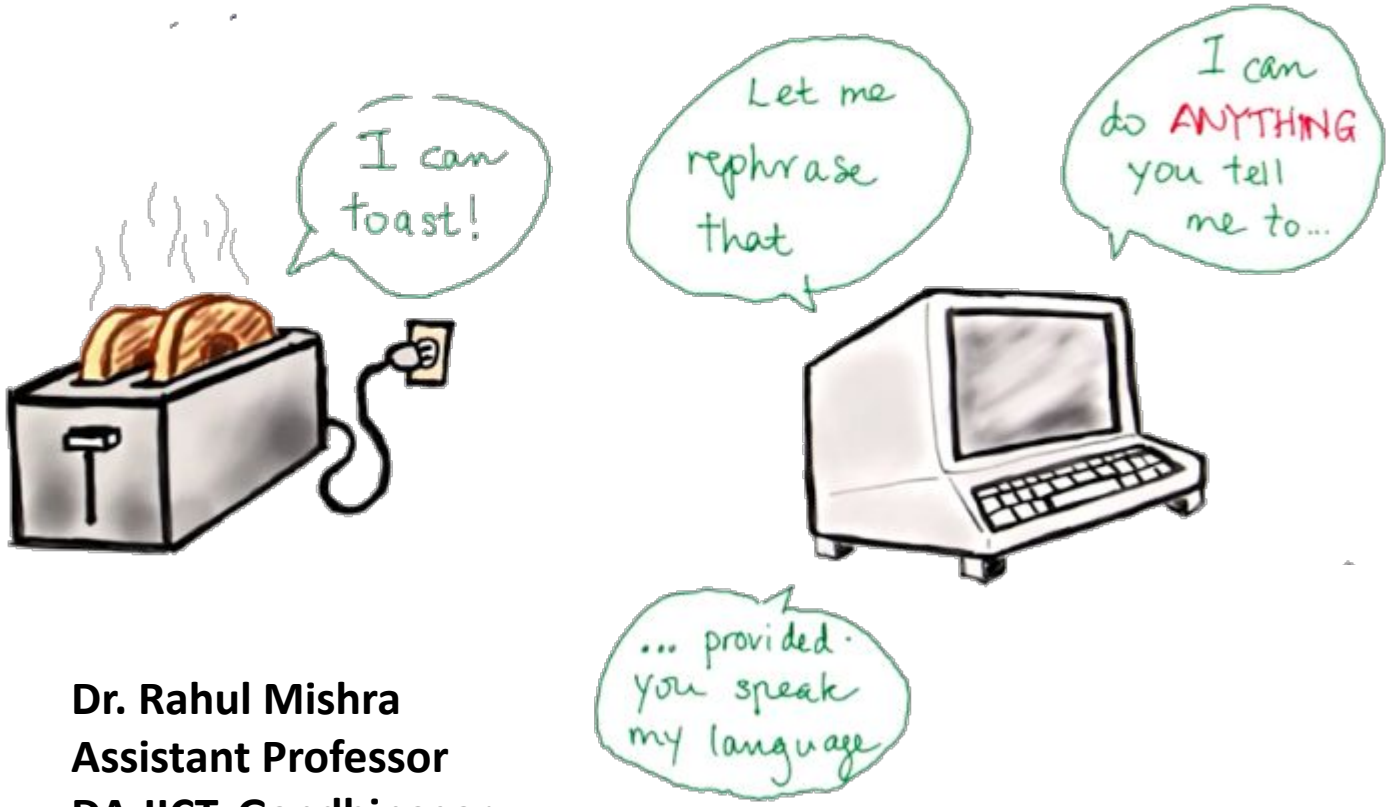
Programming Lab

Autumn Semester

Course code: PC503



Dr. Rahul Mishra
Assistant Professor
DA-IICT, Gandhinagar

Lecture
Django

DJANGO →

1) Today's websites are rich applications that act like fully develop destop applications.

2) Python has a great set of tools called Django for building web applications.

3) Django is a web framework — a set of tools designed to help you build interactive websities.

4) Here, we will learn how to use Django to build a project called Learning Log

5) The Learning log is an online journal system that lets you keep track of information you have learned about particular topics.

6) Our target to create a web-page using Django which can respond to page requests and make it easier to read and write to a database, manage users, and much more.

Setting Up a Project:-

> When begining a project, we first need to describe the project in a specification or spec.
> The we will set up a virtual environment in which to build the project.

* Writing a spec :-

» A full spec details the project goals, describe the project's functionality, and discuss its appearance and user interface.

Spec of our project -

- We will write a web app called Learning Log, that allows users to log the topics they are interested in and to make journal entries as they learn about each topic.
- The Learning Log home page will describe the site and invite users to either register or log in
- Once logged in, a user can create new topics, add new entries, and read and exit existing entries.

Creating a Virtual Environment :-

- To work with Django, we will first set up virtual environment
- A virtual environment is a place on your system where you can install packages and isolate them from all other Python packages.
- Seperating one project's libraries from other projects is beneficial and will be necessary when we deploy Learning Log to a server.

① create a new directory for your project called "learning_log"

② switch to that directory in a terminal, and enter the following code to create a virtual environment:

```
learning_log $ python -m venv ll_env
learning_log $
```

- We are running the view venv environment module and using it to create a virtual environment named ll_env (note that this is ll_env with two lowercase Ls, not two ones.

- IF you use a command such as python3 when running program or installing packages.

▶ Activating the Virtual Environment:

```
learning_log$ source ll_env/bin/activate
(ll_env)learning_log $
```

Note: IF you are using Windows, use the command "ll_env\Scripts\activate" (without the word source) to activate the virtual environment. IF you are using Powershell, you might need to captilize Activate

* To deactivate use the following:-

```
(ll_env)learning_log$ deactivate
learning_log$
```

Installing Django

```
(ll_env) learning_log$ pip install django
Collecting django
-- snip --
Installing collected packages: pytz, django
Successfully installed django
(ll_env) learning_log$
```

- We are working in a virtual environment, which is its own self-contained environment, this command is the same on all systems.
- There's no need to use the --user flag, and there's no need to use longer commands, such as python -m pip install package_name.

Creating a Project in Django →

(ll_env) learning_log$ django-admin startproject learning_log .
(ll_env) learning_log$ ls # windows dir
learning_log ll_env manage.py
(ll_env) learning_log$ ls learning_log
__init__.py settings.py urls.py wsgi.py

Creating the Database →

> Django stores most of the information for a project in a database, so next we need to create a database that Django can work with. # Use same active environment.

(ll_env) learning_log $ python manage.py migrate

> operations to perform:

Apply all migrations : admin, auth, contenttypes, sessions

> Running migrations:

Applying contenttypes.0001_initial... OK

Applying auth.0001_initial... OK

--snip--

Applying sessions.0001_initial... OK

(ll_env) learning_log $ ls #dir

db.sqlite3 learning_log ll_env manage.py

Note: ① Anytime we modify a database, we say we are migrating the database.

② Issuing the migrate command for the first time tells Django to make sure that database matches the current state of the project

③ SQLite is a database that runs off a single file; it's ideal for writing simple apps because you won't have to pay much attension to managing the database

* In an active virtual environment, use the command python to run manage.py commands, even if you use something different, like python3, to run the program.

Viewing the Project :-

- Let us make sure that Django has set-up the project properly.
- Enter the runserver command as follows to view the project in its current state:-

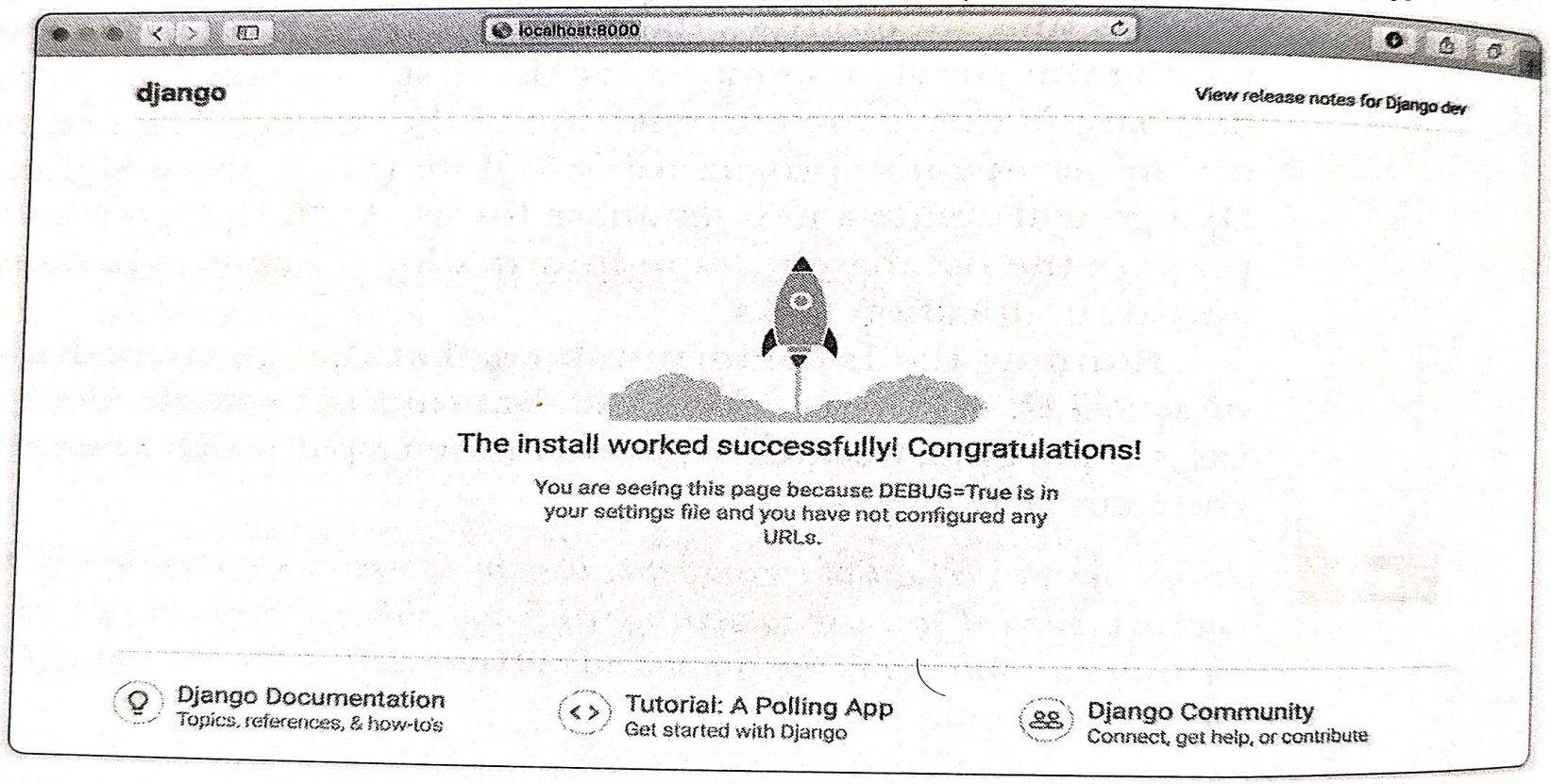
```
(ll_env) learning_log$ python manage.py runserver
Watchman unavailable: pywatchman not installed
Watching for file changes with StatReloader
Performing system checks...

① - System check identified no issue (0 silenced)
② - Django version using settings 'learning_log.settings'
③ - Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C
```

→ Django should start a server called the development server, so you view the project on your system to see its work.

→ The URL http://127.0.0.1:8000/ indicates that the project is listening for requests on port 8000 on your computer, which is local host.

→ The term local-host refers to a server that only processes requests on your system.

→ Open a web browser and enter the URL http://localhost:8000/ or http://127.0.0.1:8000/



* If you receive the error message That port is already in use, tell Django to use a different port by entering python manage.py runserver 8001.

Starting an App

> A Django project is organized as a group of individual apps that works together to make the project work as a whole.

> You should leave the development server running in the terminal window opened earlier.

> Open a new terminal window (or tab), and navigate to the directory that contains manage.py

> Activate the virtual environment, and then run the startapp command:

```
learning_log$ ll_env\Scripts\activate   # source ll_env/bin/activate
(ll_env) learning_log$ python manage.py startapp learning_logs
(ll_env) learning_log$ ls #dir
db.sqlite3  learning_log  learning_logs  ll_env  manage.py
(ll_env) learning_log$ ls learning_logs/  # dir learning_logs/
```

> The command startapp appname tells Django to create the infrastructure needed to build an app.

> When you look in the project directory now, we will see a new folder called learning_logs.

> The most important files are models.py, admin.py, and views.py

> We will use model.py to define the data we want to manage in our app.

* Defining Models :-

> Each users will need to create a number of topics in their learning log.
> Each entry they make will be tied to a topic, and these entries will be displayed as text.
> We will also need to store the timestamp of each entry
> Open the file models.py

models.py

```
from django.db import models
class Topic(models.Model):
    """A topic the user is learning about"""
    text = models.CharField(max_length=200)
    date_added = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        """Return a string representation of the model."""
        return self.text
```

→ We have created a class called Topic, which inherits from Model — a parent class included in Django.

* Activating Models -

Open setting.py (in the learning_log/learning_log directory); you will see a section that tell Django which apps are installed and work together in the project.

setting.py

```
-- snip --
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages'
    ]
```

→

```
-- snip --
INSTALLED_APPS = [
    # My apps
    'learning_logs',
    # Default django apps
    .
    .
```

* We need to tell Django to modify the database so it can store information related to the model Topic.

```
(ll_env) learning_log$ python manage.py makemigrations learning_logs
        :
(ll_env) learning_log$
```

* Now, we will apply this migration and have Django modify the database

```
(ll_env) learning_log$ python manage.py migrate
Operations to perform:
  :
  :
Applying learning_logs.0001_initial... OK
```

} Django confirms that the migration for learning_logs worked OK.

Whenever we want to modify the data that Learning Log manages, we will follow these three steps:

> modify model.py
> call make migrations on learning_logs.
> tell Django to migrate the project.

* ⇒ The Django Admin Site

* Django makes it easy to work with your models through the admin site.
* Only the site's administrators are users of the admin site, not general users.

* Setting Up a Superuser :-

→ Django allows you to create a superuser, or user who has all privileges available on the site

> A user's privileges control the actions that user can take.

> The most restrictive privilege settings allow a user to only read public information on the site.

> Registered users typically have the privilege of reading their own private data and some selected information available only to members.

> To effectively administer a web application, the site owner usually needs access to all information stored on the site

> To create a superuser in Django, enter the following command and respond to the prompts:

```
(ll_env) learning_log$ python manage.py createsuperuser
Username (leave blank to use 'eric'): ll_admin
Email address:
Password:
Password (again):
Superuser created successfully.
```

* Some sensitive information can be hidden from a site's administrations.
* Django doesn't store the password you enter; instead, it stores a string derived from the password, called a hash.
* Each time you enter your password, Django hashes your entry and compares it to the stored hash.
* If two hashes match, we are authenticated.

Registering a Model with the Admin site

* Django includes some models in the admin site automatically, such as User and Group, but the models we create need to be added manually.
* When we started the learning_logs app, Django created an admin.py file in the same directory as models.py. Open the admin.py file.

admin.py

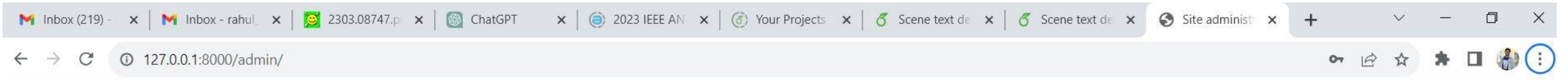
```
from django.contrib import
# Register your model here from .models import Topic
admin.site.register(Topic)
```

* This code first imports the model we want to register, Topic.
* The dot in front of models tells Django to look for model.py in the same directory as admin.py
* The code admin.site.register() tells Django to manage our model through the admin site.

Go to http://localhost:8000/admin/

↳ Image as next slide

IF you see a message in your browser that the web page is not available, make sure you still have the Django server running in a terminal windows.



Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

LEARNING_LOGS		
Topics	+ Add	Change

Recent actions

My actions

None available

Inbox (219) - | Inbox - rahul_ | 2303.08747.p | ChatGPT | 2023 IEEE AN | Your Projects | Scene text de | Scene text de | Add topic | D

127.0.0.1:8000/admin/learning_logs/topic/add/

Django administration

WELCOME, **LL_ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Learning_Logs › Topics › Add topic

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

- Groups + Add
- Users + Add

LEARNING_LOGS

- Topics + Add

Add topic

Text: Chess

SAVE | Save and add another | Save and continue editing

Adding Topics :-

» Now that Topic has been registered with admin site, lets add our first topic.

» Click Topics to go the Topics page, which is mostly empty, because we have no topics to manage yet.

» Click Add Topic, and a form for adding a new topic appears.

↳ Enter Chess in the first box and click save

↳ You will be sent back to the topic admin page

!

!

Next, topic Rock Climbing

Defining the Entry Model :-

> For a user to record what they have been learning about chess and rock climbing, we need to define a model for the kinds of entries users can make in their learning logs.

> Each entry needs to be associated with a particular topic.

> This relationship is called a many-to-one relationship, meaning many entries can be associated with one topic.

models.py

```
from django.db import models
class Topic (models.Model):
    --snip--
class Entry (models.Model):
    """Something specific learned about a topic"""
    topic = models.ForeignKey (Topic, on_delete = models.CASCADE)
    date_added = models.DateTimeField (auto_now_add = True)
    text = models.TextField ()
```

```
class Meta:
    verbose_name_plural = 'entries'

def __str__(self):
    """Return a string representation of the model."""
    return f"{self.text[:50]}..."
```

> The Entry class inherits from Django's base Model class, just as Topic did

> A foreign key is a database term; it's a reference to another record in the database. This is the code that connects each entry to specific topic.

> Each topic is assigned to a key or ID, when it is created.

> When Django needs to establish a connection between two pieces of data, it uses the key associated with each piece of information.

> The `on_delete = models.CASCADE` arguments tells Django that when a topic is deleted, all the entries associated with that topic should be deleted as well.

↳ This is also called cascade delete / cascading.

> Next is an attribute called text, which is an instance of TextField.

- This kind of field doesn't need size limit, because we don't want to limit the size of individual entries.
- The date_added attributes allows up to present entries in the order they were created and to place a timestamp next to each entry.
- At we nest the Meta class inside our Entry class. The "Meta class holds extra information for managing a model;" here, it allows us to set a special attribute telling Django to use "Entries" when it needs to refer to more than one entry.
- __str__() method tells Django which information to show when it refers to individual entries.
- Because an entry can be a long body of text, we tell Django to show just the first 50 characters of text.
- We also add an ellipis to identify that we are not always displaying the entire entry.

Migrating the Entry Model :-

→ Because we have added a new model, we need to migrate the database again.
→ This process will become quite familiar: you modify model.py, run the command

```
(ll_env) learning_log $ python manage.py makemigrations learning_logs
Migrations for 'learning_logs':
  !
  learning_logs/migrations/0002_entry.py
  !
Applying learning_logs.0002_entry... ok
```

A new migration called 0002_entry.py is generated, which tells Django how to modify the database to store information related to the model Entry.

Registering Entry with the Admin site :-

admin.py

```
from django.contrib import admin
from .models import Topic, Entry
admin.site.register(Topic)
admin.site.register(Entry)
```

↳ Go back to http://localhost/admin/, and you should see Entries listed under Learning_Logs.

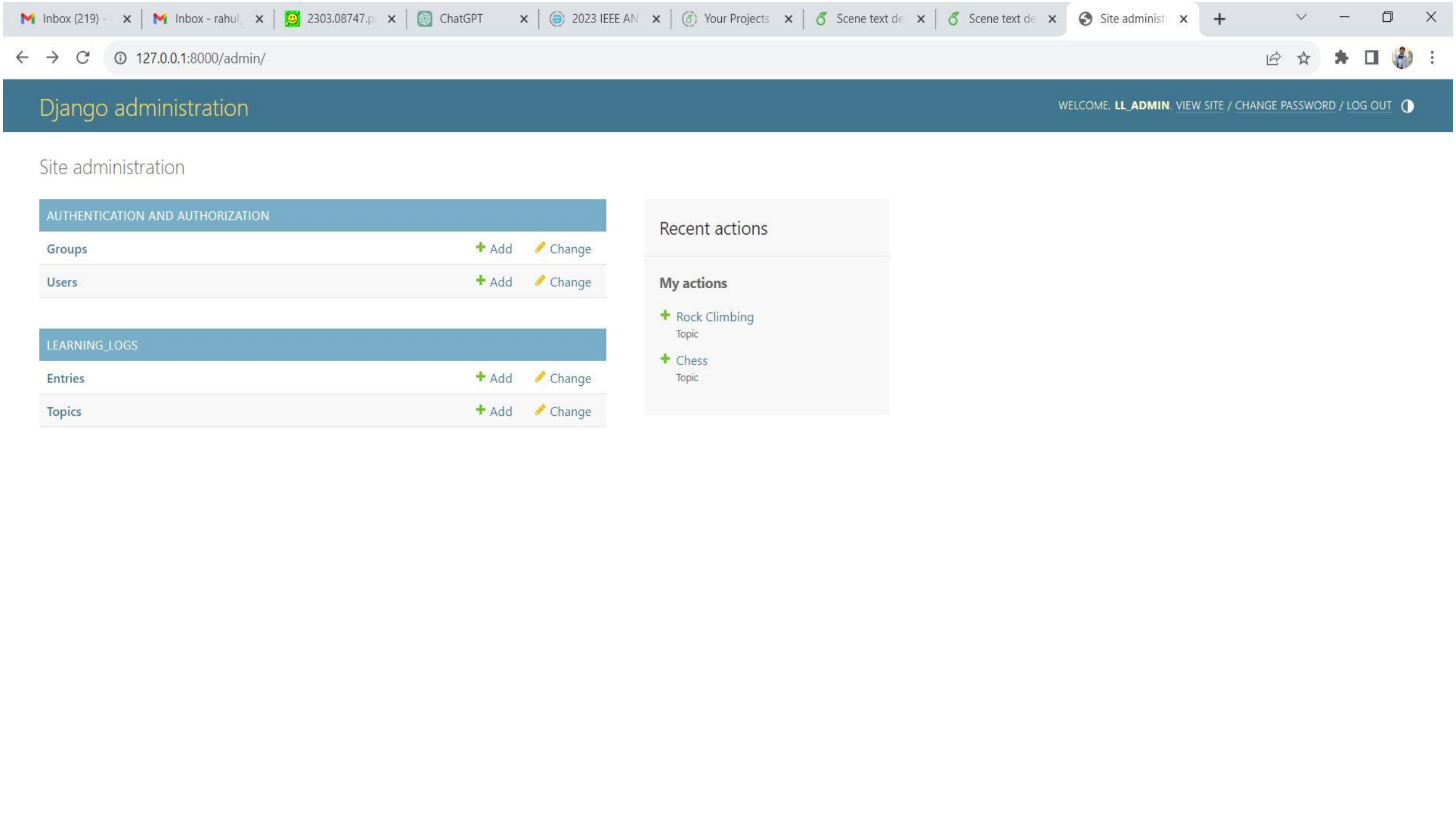
→ Click the Add link for Entries, or click Entries and then choose Add entry.

→ You should see a drop-down list to select the topic you are creating an entry for and a text box for adding an entry.

→ Select chess from the drop-down list, and add an entry.

"Write in 200 words about the game chess."

↳ SAVE


Inbox (219) -
Inbox - rahul_
2303.08747.p
ChatGPT
2023 IEEE AN
Your Projects
Scene text de
Scene text de
Site administ
127.0.0.1:8000/admin/
Django administration
WELCOME, LL_ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Site administration
AUTHENTICATION AND AUTHORIZATION
Groups + Add Change
Users + Add Change
LEARNING_LOGS
Entries + Add Change
Topics + Add Change
Recent actions
My actions
+ Rock Climbing
Topic
+ Chess
Topic

⇒ When you back to the main admin page for entries

↳ We will see the benefit of using `text[:50]` as the string representation for each entry; it's match easier

→ Select Rock Climbing

↳ Write in 200 words about Rock climbing ever.

!
!

The Django shell :-

>> With some data entered, we can examine that data programmatically through an interactive terminal session.

>> This interactive environment is called the Django shell, and it is a great environment for testing and trouble shooting project.

```
(ll_env) learning_log$ python manage.py shell
>>> from learning_logs.models import Topic
>>> Topic.objects.all()
<QuerySet [<Topic: chess>, <Topic: Rock Climbing>]>
```

- Here, we import the model Topic from the learning_logs.models module.
- We then use the method Topic.object.all() to get all the instances of the model Topic

```
>>> topics = Topic.objects.all()
>>> for topic in topics:
...     print(topic.id, topic)
...
1 Chess
2 Rock climbing
```

* IF you know the ID of a particular object, you can use the method Topic.object.get() to retrieve that object and examine any attribute

```
>>> t = Topic.objects.get(id=1)
>>> t.text
'chess'
>>> t.date_added
datetime.datetime(2023, 2, 19, 1, 55, 31, 98500, tzinfo=<UTC>)
```

» We can also look at the entries related to a certain topic.

» Earlier we defined the topic attribute for the Entry model

» This was a Foreignkey, a connection between each entry and a topic.

```
>>> t.entry_set.all()
<QuerySet [<Entry: ____>,
           <Entry: ____>]
```

Note :- Each time you need to modify your models, you will need to restart the shell to see the effects of those changes. To exit a shell session, press CTRL-Z and Enter.

⇒ You can develop application for Pizza, your subjects etc.

Making Pages: The Learning Log Home Page

> Making web pages with Django consists of three stages: ① defining URLs
② writing views, and
③ writing templates.

> A URL pattern describes the way the URL is laid out. It also tells Django what to look for when matching a browser request with a site URL so it knows which page to return.

> Each URL then maps to a particular view – the view function retrieves and process the data needed for that page.

> The view function often renders the page using a template, which contains the overall structure of the page.

> Let's make the home page for Learning Log.

> We will define the
- URL for the home page,
- write its view function
- create a simple template

> We will make a simple page for now.
↳ A functioning web app is fun to style when it is complete;

> For now, the home page will display only a title and a brief description.

* → Mapping a URL

> Users request pages by entering URLs into browser and clicking links, so we will need to decide what URLs are needed.

> Base URL :- http://localhost:8000/, returns the default Django site that lets us know the project was set up correctly.

In main learning_log project folder, open the file urls.py

```
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

→ The first two lines import a module and a function to manage URLs for the admin site.

→ The body of the file defines the urlpatterns variable.

→ The code at includes the module admin.site.urls, which defines all the URLs that can be requested from the admin site.

→ We need to include the URLs for learning_logs, so add the following:

urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('learning_logs.urls')),
]
```

⟺ We have added a line to include the module learning_log.urls

* The default urls.py is in the learning_logs folder

↳ Create a new Python file and save it as urls.py in learning_logs, and enter this code into it:

urls.py

```
"""Define URL patterns for learning_logs"""
from django.urls import path
from . import views

app_name = 'learning_logs'
urlpatterns = [
    # Home page
    path('', views.index, name='index'),
]
```

> We then import the path function, which is needed when mappings URLs to views.

> We also import the view module; the dot tell Python to import the views.py module from the same directory.

> url/patterns in the module is a list of individual pages that can be requested from the learning_logs app.

> The actual URL pattern is a call to the path() function, which takes three arguments

① The first argument is a string that helps Django route the current request properly.

Django ignores the base URL for the project (http://localhost:8000/) so the empty string ('') matches the base URL

② path ⟶ views.index # index function from the view will be called

③ To refer in other code sections.

Writing a View :-

A view function takes in information from a request, prepare the data needed to generate a page.

"Learning_logs"

views.py

```
from django.shortcuts import render
# create your view here
def index(request):
    """The homepage for Learning Log."""
    return render(request, 'learning_logs/index.html')
```

The render function here passes two arguments - the original request object and a template it can use to build the page.

Writing a Template :-

* The template defines what the page should look like, and Django fills in the relevant data each time the page is requested.

① Inside the learning_logs folder, make a ~~new~~ new folder called templates.

② Inside the templates folder, make another folder called learning_logs.

③ Inside the inner learning_logs folder, make a new file called index.html

The path to the file will be learning_logs/learning_logs/templates/learning_logs/index.html

```
<p> Learning Log </p>

<p> Learning Log helps you keep track of your learning, for any topic you are
    learning about. </p>
```

Go to http://localhost:8000/ →

Learning Log

A learning log is a personal record or journal that individuals use to document their learning experiences, reflections, and progress over a period of time. It serves as a valuable tool for self-assessment, self-reflection, and goal-setting in the learning process. A typical learning log may include entries detailing what was learned, how it was learned, challenges faced, breakthroughs achieved, and plans for future learning. It can cover a wide range of topics, including academic subjects, skills development, professional growth, or personal interests. By keeping a learning log, individuals gain insights into their learning patterns, areas of strength, and areas that may require further attention. This reflective practice can enhance the effectiveness and efficiency of the learning process, making it a valuable tool for learners of all ages and levels.

Google Chrome

Google Chrome is a web browser developed by Google, released in 2008. Chrome is the world's most popular web browser today!

Mozilla Firefox

Mozilla Firefox is an open-source web browser developed by Mozilla. Firefox has been the second most popular web browser since January, 2018.

Microsoft Edge

Microsoft Edge is a web browser developed by Microsoft, released in 2015. Microsoft Edge replaced Internet Explorer.

DAIICT Gandhinagar 2023-2024

Building Additional Pages :-

> We will build two pages that display data: a page that lists all topics and a page that shows all the entries for a particular topic.

① For each page, we will specify a URL pattern, write a view function, and write a template

② Let build a base template that all templates in the project can inherit from.

Template Inheritance:

⇒ When building a website, some elements will always need to be repeated on each page.

⇒ Thus, we can write a base template containing the repeated elements and then have each page inherit from the base.

The Parent Template:

> We will create a template, called base.html in the same directory as index.html
> This file will contain elements common to all pages; every other template will inherits from base.html
> The only element we want to repeat on each page right now is the title at the top.
> Because, we will include this template on every page, let us make the title a link to the homepage.

base.html

```
<p>
<a href="{% url 'learning_logs:index' %}"> Learning Log </a>
</p>
{% block content %}{% endblock content %}
```

* The first part of this file creates a paragraph containing the name of the project, which also acts as a homepage link.

* To generate a link, we have to use a template tag, which is indicated by braces and percent signs {% %}.

* A template tag generates information to be displayed on a page.

* Our template tag {% url 'learning_logs:index' %} generates a URL matching the URL pattern defined in learning_logs/urls.py with the name 'index'

For example:- A link is surrounded by the anchor tag <a>

```
<a href="link_url">link test</a>
```

* Every page in our project will inherit from base.html, so from now on, every page will have a link back to the homepage.

→ We insert a pair of block tags. This block, named content, is a place-holder; the child template will define the kind of information that goes in the content block.

→ A child template doesn't have to define every block from its parent, so you can reserve space in parent templates for as many block as you like.

Notably:- In Python code, we almost always use four spaces when we indent. Template files tend to have more levels of nesting than Python files, so its common to use only two spaces for each indentation level.

The Child Template:

We need to rewrite index.html to inherit from base.html. Add the following code to index.html.

index.html

```
{% extends "learning_logs/base.html" %}

{% block content %}
<p>Learning Log helps you keep track of your learning, for any topic you are learning
    about.</p>
{% endblock content %}
```

- Notably, in a large project, it is common to have one parent template called base.html for the entire site and parent templates for each major section of the site.
- The section templates inherits from base.html, and each page in the site inherits from a section templat.
- Such configuration provides a very effective way to work, and it encourages to steadily updat your site over time.

The Topics Page:

⇒ The General topics pages will show all topics that users have created and its the first page that will involve working with data.

The Topics URL Pattern

→ First, we define the URL for the topics page. It's common to choose a simple URL fragment that reflects the kind of information presented on the page.

→ We will use the word topics, so the URL http://localhost:8000/topics/ will return this page.

urls.py

```
"""Defines URL pattern for learning_logs."""
--snip--
urlpatterns = [
    # Home page.
    path('', views.index, name='index'),
```

```
    # page that shows all topic
    path('topics/', views.topics, name='topics'),
]
```

> When Django examines a requested URL, this pattern will match any URL that has the base URL followed by topics.

> Any request with a URL that matches this pattern will then be passed to the function topics() in views.py

* The Topics View

The topics() function needs to retrieve some data from the database and send it to the template. Here, we need to add to views.py.

views.py

```
from .models import Topic

def index(request):
    --snip--

def topics(request):
    """Show all topics."""
    topics = Topic.object.order_by('date_added')
    context = {'topics': topics}
    return render(request, 'learning_logs/topics.html', context)
```

» We first import the model associated with the data we need.

» The topics() function needs one parameter: the request object Django received from the server.

» We query the database by asking for the Topic objects, sorted by the date_added attribute.

The Topics Template :-

> The template for the topics page receives the context dictionary, so the template can use the data that topics() provides.

> Make a file called topics.html in the same directory as index.html.

topics.html

```
{% extends "learning_logs/base.html" %}

{% block content %}

  <p>Topics</p>

  <ul>

    {% for topic in topics %}
      <li>{{ topics }}</li>
    {% empty %}
      <li>No topics have been added yet.</li>
    {% endfor %}

  </ul>

{% endblock content %}
```

=> We use the {% extends %} tags to inherit from base.html, just as the index template does, and open a content block.

Loop writing in the template

```
{% for item in list %}
    do something with each item
{% endfor %}
```

=> Inside the loop, we want to turn each topic into an item in the bulleted list.

=> To print a variable in a template, wrap the variable name in double braces.

=> {% empty %} template tag, which tell Django what to do if there is no items in the list.

Now, we need to modify the base template to include a link to the topics page.

base.html

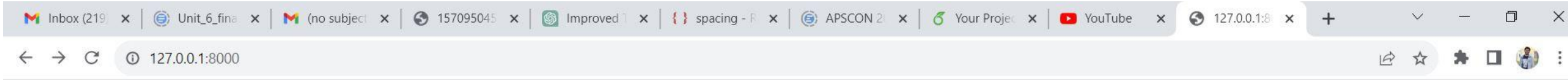
```
<p>
    <a href="{% url 'learning_logs:index' %}">Learning Log</a> —
    <a href="{% url 'learning_logs:topics' %}">Topics</a>
</p>

{% block content %}{% endblock content %}
```

This line tells Django to generate a link matching the URL pattern with the name 'topics' in learning_logs/urls.py.

Refer to the home page content.

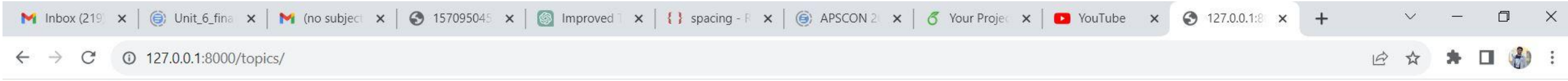
localhost:8000/topics/

Image →



Learning Log - Topics

A learning log is a personal record or journal that individuals use to document their learning experiences, reflections, and progress over a period of time. It serves as a valuable tool for self-assessment, self-reflection, and goal-setting in the learning process. A typical learning log may include entries detailing what was learned, how it was learned, challenges faced, breakthroughs achieved, and plans for future learning. It can cover a wide range of topics, including academic subjects, skills development, professional growth, or personal interests. By keeping a learning log, individuals gain insights into their learning patterns, areas of strength, and areas that may require further attention. This reflective practice can enhance the effectiveness and efficiency of the learning process, making it a valuable tool for learners of all ages and levels.



Learning Log - Topics

Topics

- Chess
- Rock Climbing

• Individual Topic Pages :-

» Next, we need to create a page that can focus on a single topic, showing the topic name and all the entries for that topic.

» We will again define a new URL pattern, write a view, and create a template.

» We will also modify the topics page so each item in the bulleted list links to its corresponding topic page.

• The Topic URL Pattern :-

The URL pattern for the topic page is a little different than the prior URL patterns because it will use the topic's id attribute to indicate which topic was requested

For example, if the user wants to see the details page for the chess topic, where the id is 1.

The URL will be http://localhost:8000/topics/1/.

Here's a pattern to match this URL, which you should place in learning_logs/urls.py

urls.py
```
--snip--
urlpatterns = [
    --snip--
    # Detail page for a single topic
    path('topics/<int:topic_id>/', views.topic, name='topic'),
]
```

When Django finds a URL that matches this pattern, it calls the view function topic() with the value stored in topic_id as an argument.

The topic View :-

The topic() function needs to get the topic and all associated entries from the database :

views.py
```
--snip--
def topic(request, topic_id):
    topic = Topic.objects.get(id=topic_id)
    entries = topic.entry_set.order_by('-date_added')
    context = {'topic': topic, 'entries': entries}
    return render(request, 'learning_logs/topic.html', context)
```

The Topics Template

> The template needs to display the name of the topics and the entries
> We also need to inform the user if no entries have been made yet for this topic.

topic.html

```
{% extends 'learning_logs/base.html' %}
{% block content %}
<p> Topic : {{ topic }} </p>
<p> Entries : </p>
<ul>
{% for entry in entries %}
    <li>
    <p> {{ entry.date_added|date: 'M d, Y H:i' }} </p>
    <p> {{ entry.text|linebreaks }} </p>
    </li>
```

```
    </li>
    {% empty %}
    <li>There are no entries for this topic yet.</li>
  {% endfor %}
  </ul>

{% endblock content %}
```

> It shows the topic that's currently being displayed → stored in the template variable {{ topic }}

> We start a bolleted list to show each of the entries and loop through them as we did the topics earlier.

> Each bullet lists two pieces of information: the timestamp and the full text of each entry.

Links from the Topics Page :-

→ Before we look at the topic page in a browser, we need to modify the topics template so each topic links to the appropriate page.

topics.html

```
--snip--
{% for topic in topics %}
<li>
<a href="{% url 'learning_logs:topic' topic.id %}">{{ topic }}</a>
</li>
{% empty %}
--snip--
```

→ We use the URL template tag to generate the proper link, based on the URL match pattern in learning_logs with the name 'topic'.

- This URL pattern requires a 'topic_id' argument, so we add the attribute topic.id to the URL template tag.
- Each topic in the list of topics is a link to a topic page, such as

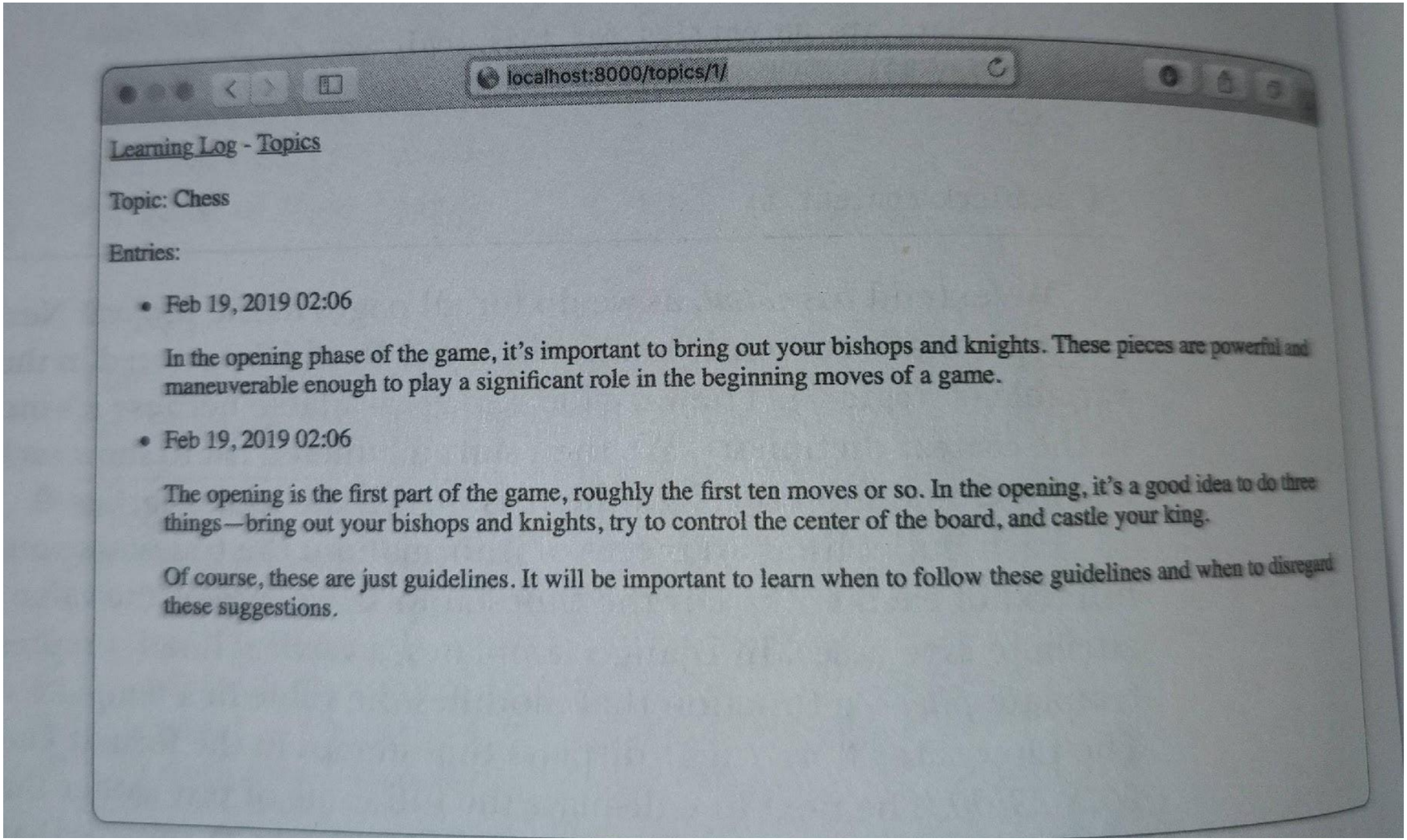
  http://localhost:8000/topics/1/.

NOTE: → There is a subtle but important difference between topic.id and topic_id.

→ The expression topic.id examines a topic and retrieves the value of the corresponding ID.

→ The variable topic_id is a reference to that ID in the code

↳ Web page ahead –


localhost:8000/topics/1/
Learning Log - Topics
Topic: Chess
Entries:
Feb 19, 2019 02:06
In the opening phase of the game, it's important to bring out your bishops and knights. These pieces are powerful and maneuverable enough to play a significant role in the beginning moves of a game.
Feb 19, 2019 02:06
The opening is the first part of the game, roughly the first ten moves or so. In the opening, it's a good idea to do three things—bring out your bishops and knights, try to control the center of the board, and castle your king.
Of course, these are just guidelines. It will be important to learn when to follow these guidelines and when to disregard these suggestions.

This was only preliminaries of Django: more to be covered.