Ecom – Website

◆ Project Basics

Q1: What is this project about?
A: This is an e-commerce website built with Django. It allows users to browse products, search, view product details, and place orders. The admin interface is also customized for easier product and order management.

◆ Admin Customization

Q2: How have you customized the Django admin interface?
A: I customized the Django admin by:

- Changing the site header, title, and index title using admin.site.site_header, site_title, and index_title.
- Creating a ProductAdmin class to control how products are displayed and edited.
- Implementing a custom action change_category_to_defalut to set a product's category to "default".
- Adding search fields, list display, editable fields, and action functionalities to streamline admin workflows.

Q3: What's the purpose of change_category_to_defalut method?
A: It's a custom admin action that allows bulk updating of the product category to "default" directly from the admin panel. This helps in quickly categorizing multiple products.

◆ Pagination and Filtering

Q4: How did you implement pagination in your views?
A: I used Django’s built-in Paginator class in the index view. It paginates the list of products, displaying 4 items per page. The current page number is fetched using request.GET.get('page'), and the corresponding page object is passed to the template.

Q5: How does search functionality work in your index view?
A: It uses a GET parameter called item_name. If the parameter is provided, the queryset is filtered using title__icontains=item_name, which allows case-insensitive partial matching on product titles.

◆ Order Handling and Checkout

Q6: How do you handle order creation in the checkout view?
A: On a POST request to the checkout view, I collect customer details and the cart data from the form. The cart data is sent as a JSON string and then deserialized using json.loads. A new Order object is created and saved with this data.

Q7: How do you ensure that the cart data is valid JSON?
A: I wrap json.loads(cart_data) in a try...except block to catch and handle json.JSONDecodeError. If the JSON is invalid, I fallback to an empty dictionary.

🔹 Other Django Concepts

Q8: What is get_asgi_application() used for?
A: get_asgi_application() is used to get the ASGI-compatible application object for serving the Django project using ASGI servers like Daphne or Uvicorn. It's necessary for asynchronous features and deploying with ASGI.

Q9: Why do you set the environment variable DJANGO_SETTINGS_MODULE in asgi.py?
A: It tells Django which settings module to use for configuring the application. It's essential for Django to load the correct configurations when the application starts.

🔹 Suggestions for Improvements

Q10: How would you improve this project further?
A:

- Add user authentication for order tracking and personalized experience.
- Implement cart persistence using sessions or database models.
- Add validation and success messages after checkout.
- Implement AJAX-based product filtering and cart updates for better UX.
- Add unit tests for views and models to improve code reliability.

Q11: What is the difference between GET and POST in Django views?
A: GET is used for retrieving data (like displaying products), while POST is used to send data to the server (like submitting orders). In my checkout view, I handle POST to save order data.

Q12: How does Django handle form submission in the checkout view?
A: I use request.POST.get() to extract form data and then manually create an Order object. In a full implementation, I could also use Django Forms for built-in validation.

Q13: What would happen if the product ID doesn't exist in the detail view?
A: It would raise a DoesNotExist error. To handle it gracefully, I can use get_object_or_404(Products, id=id) instead of Products.objects.get(id=id).

Q14: What are some Django security practices you've followed or would add?
A:

- CSRF protection via Django's built-in middleware
- Using json.loads with try-except to avoid crashes
- Input validation in forms
- Session-based cart or authentication for secure checkout

Q15: What is the role of templates in your project?
A: Templates render dynamic HTML pages by injecting data from views (like product lists, details). I use 'shop/index.html', 'shop/detail.html', and 'shop/checkout.html' to create the user interface.

Web Scraper

✓ Project Overview

Q1: What does your web scraper project do?

A: It allows users to input a website URL, fetches the webpage, extracts all the anchor (<a>) tags using BeautifulSoup, and stores the link text and address into a database using Django's ORM. Users can also delete individual or all links via buttons on the interface.

✓ Questions on Packages and Code Logic

Q2: What is the use of requests in your project?

A: The requests library is used to make HTTP GET requests to the user-provided URL. It retrieves the HTML content of the page so it can be parsed for links.

Q3: Why do you use verify=False in requests.get()?

A: This disables SSL certificate verification, which helps avoid errors when scraping sites with expired or self-signed certificates. However, it's **not recommended in production** due to security risks. In a real deployment, I would use verify=True.

Q4: What is BeautifulSoup and how do you use it?

A: BeautifulSoup is a Python library used to **parse HTML or XML documents**. In my project, I use it to parse the HTML content returned from the requests.get() call:

python
CopyEdit
```
soup = BeautifulSoup(page.text, 'html.parser')
```

Then I extract all anchor tags with:

python
CopyEdit
```
soup.find_all('a', href=True)
```

It helps in navigating and searching through HTML elements efficiently.

Q5: What is urljoin used for in your project?

A: Many websites use **relative URLs** (like /about), so I use urljoin(site, href) to convert them to **absolute URLs** (like https://example.com/about). This ensures all links are valid and complete.

Q6: What is the purpose of Link.objects.create(...)?

A: It saves each scraped link into the database using Django's ORM. Each link has two fields: name (the visible text) and address (the actual URL).

Q7: What does Link.objects.last() do?

A: It returns the **most recently added** Link object from the database. I use it when the user clicks "Delete Last" to remove the latest entry.

Q8: Why do you use HttpResponseRedirect('/') instead of render() after POST?

A: It's a common best practice called **Post/Redirect/Get pattern (PRG)**. It prevents form resubmission if the user refreshes the page after submitting a form. HttpResponseRedirect('/') redirects the user to the homepage after scraping or deleting.

Q9: Why do you use print() statements in your scraper?

A: For debugging during development. They help verify what links are being scraped. In production, logging should be used instead of print statements.

Q10: How do you handle errors like bad URLs or failed requests?

A: I wrap the requests.get() call in a try-except block to catch requests.exceptions.RequestException. If an error occurs (e.g., invalid URL or timeout), it prints the error without crashing the app.

✅ Model & Database

Q11: What does the __str__ method do in your Link model?

A: It defines how each Link object is displayed in Django admin or shell. I return the name of the link, or "Unnamed Link" if the name is empty.

Q12: Why did you use blank=True, null=True in your model fields?

A: This allows fields to be optional. null=True allows NULL values in the database, and blank=True allows the Django forms to accept empty inputs.

✅ Additional / Advanced Questions

Q13: How would you improve this project?

A:

- Add validation to check if the URL is valid before scraping.
- Store only unique links to avoid duplication.
- Display error or success messages in the UI.
- Add user authentication to make link data user-specific.
- Use Django messages framework for user feedback.

Q14: How can you avoid scraping duplicate links?

A: Before saving, I can check if the link already exists in the database using:

```
python
CopyEdit
if not Link.objects.filter(address=link_address).exists():
    Link.objects.create(...)
```

Q15: What are some ethical considerations when web scraping?

A: It's important to:

- Respect the website's robots.txt.
- Avoid scraping content behind login or paywalls.
- Not overload the server with too many requests (use rate limiting).
- Give proper credit or usage according to the site's terms of service.

Summary of Tools Used

Tool / Library	Purpose
requests	Fetches HTML from the web
BeautifulSoup	Parses HTML to extract elements
urllib.parse.urljoin	Converts relative URLs to absolute URLs
urllib3.disable_warnings	Temporarily suppresses SSL warnings (for dev only)
Django Models (Link)	Stores and manages scraped links
HttpResponseRedirect	Implements PRG pattern after POST
Django ORM	Saves and retrieves link objects from the DB

Expense Tracker

✅ Project Overview

Q1: What is your Expense Tracker project about?

A: It’s a Django-based web application where users can add, view, edit, and delete expenses. It also provides summaries (weekly, monthly, yearly) and visualizes expenses by date and category using graphs generated with matplotlib.

✅ Core Functionality

Q2: How do you calculate the total, weekly, monthly, and yearly expenses?

A: I use Django’s ORM and datetime to filter expenses:

python
CopyEdit
```
last_year = datetime.date.today() - datetime.timedelta(days=365)
yearly_expense = Expense.objects.filter(date__gt=last_year).aggregate(Sum('amount'))
```

Similar logic is used for monthly and weekly expenses using timedelta(30) and timedelta(7).

Q3: How are the expenses visualized?

A: Using matplotlib, I generate two graphs:

- **Daily Expense Graph**: Line chart showing how much was spent each day.
- **Category Expense Graph**: Line chart showing total spend per category.

The charts are rendered in memory using io.BytesIO() and encoded with base64 to embed them in HTML.

Q4: How is data saved in the application?

A: Data is saved through a Django ModelForm called ExpenseForm. When a user submits the form, it’s validated and saved using Django's ORM.

Q5: How do you handle expense editing and deleting?

A:

- **Editing** uses a form prefilled with the instance via:

  python
  CopyEdit

ExpenseForm(instance=expense)

- **Deleting** is done using Expense.objects.get(id=id).delete() after a POST request.

☑ Code & Structure

Q6: What does the generate_plot() function do?

A: It generates a matplotlib line chart from input data and labels, styles it with titles, markers, and axis labels, and returns a base64 string of the image that can be directly embedded into a Django template using <img> tags.

Q7: Why do you use aggregate(Sum('amount')) instead of looping?

A: aggregate() performs the sum at the database level, which is more efficient than looping through results in Python. It reduces memory usage and improves performance.

Q8: Why do you use auto_now=True in the date field?

A: It automatically sets the current date every time the record is saved. Though it's good for this use case, auto_now_add=True might be better if I want to keep the original creation date unchanged.

☑ Design & UX

Q9: How does your app handle invalid form submissions?

A: Django forms automatically handle validation. If the form is invalid, it doesn't save the data and can display errors in the template.

Q10: What are some improvements you'd like to make?

A:

- Add **user login and session-based expenses**.
- Convert graphs to **bar or pie charts** for better category comparison.
- Add **filters** for date ranges or categories.
- Export data to **CSV or Excel**.

✅ Extra/Advanced Questions

Q11: What are the advantages of using ModelForm?

A: ModelForm automatically links a Django model to the form, reducing repetitive code and handling both validation and saving.

Q12: How would you make this a multi-user system?

A: Add a ForeignKey to User in the Expense model:

python
CopyEdit
```
from django.contrib.auth.models import User
user = models.ForeignKey(User, on_delete=models.CASCADE)
```

Then filter/query expenses by the currently logged-in user.

Q13: Why do you use get_object_or_404()?

A: It's safer than Expense.objects.get() because it returns a 404 page if the object doesn't exist, avoiding app crashes.

Summary of Tools & Concepts Used

Tool / Concept	Purpose
Django ORM	For saving and querying expense data
ModelForm	Automatically builds forms from models
Matplotlib	For generating expense graphs
base64 + BytesIO	To embed graphs in HTML without saving images
aggregate(Sum(...))	Efficiently calculate totals in the database
datetime	Time-based filtering (7 days, 30 days, etc.)

Q1: How would you normalize the Expense model (e.g., split category into its own table)?

Answer:

To normalize the Expense model, I would extract the category field into a separate model and use a ForeignKey relationship. This reduces data duplication and ensures category consistency.

Updated models:

python
CopyEdit
```
class Category(models.Model):
    name = models.CharField(max_length=100, unique=True)

    def __str__(self):
        return self.name

class Expense(models.Model):
    name = models.CharField(max_length=100)
    amount = models.IntegerField()
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    date = models.DateField(auto_now=True)
```

Benefits:

- Data integrity (no typos or inconsistencies in categories)
- Easier filtering, grouping, and analytics
- Dynamic category management

💡 Q2: How would you implement pagination or search for large expense lists?

Answer:

For large datasets, I would use Django's built-in Paginator class to break the expense list into manageable pages. I'd also use a simple keyword filter for expense names or categories.

Example code:

python
CopyEdit
```
from django.core.paginator import Paginator

def index(request):
    query = request.GET.get('q')
    expenses = Expense.objects.all()

    if query:
        expenses = expenses.filter(name__icontains=query)
    paginator = Paginator(expenses, 10)  # 10 per page
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)

    return render(request, 'expensetracter/index.html', {'page_obj': page_obj})
```

Bonus tip: For better UX, add a search bar and next/prev buttons in your HTML.

💡 Q3: How would you make each user’s expenses private (user authentication)?

Answer:

I’d use Django’s built-in User model and add a ForeignKey to link each expense to the authenticated user. This way, each user can only see and manage their own data.

Model update:

```
python
CopyEdit
from django.contrib.auth.models import User

class Expense(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    amount = models.IntegerField()
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    date = models.DateField(auto_now=True)
```

Views:

```
python
CopyEdit
def index(request):
    if request.user.is_authenticated:
        expenses = Expense.objects.filter(user=request.user)
        ...
```

Why this is good:

- Protects user data
- Prepares app for multi-user environments

💡 Q4: If asked about Class-Based Views (CBVs), what would you say?

Answer:

I started with function-based views for simplicity, but I would use class-based views for larger projects. CBVs improve code reusability and follow DRY principles. For example, I'd use ListView for listing expenses and CreateView for adding them.

Example (CBV for listing expenses):

```
python
CopyEdit
from django.views.generic import ListView
from .models import Expense

class ExpenseListView(ListView):
    model = Expense
    template_name = 'expensetracter/index.html'
    context_object_name = 'expenses'

    def get_queryset(self):
        return Expense.objects.filter(user=self.request.user)
```

Benefits:

- Less repetitive code
- Cleaner structure with built-in support for pagination, filtering, etc.