(PART A – 2 Marks)

1. Enumerate any 4 web application frameworks.

- Django
- Spring
- MERN
- MEAN

2. Mention the files responsible for Model-View-Template pattern in Django Web Framework.

- urls.py
- views.py
- models.py
- template.html

3. What is the reason to choose popular web frameworks for web application development stack?

- Streamlined development
- Speed and Efficiency
- Integrated Security
- Enhanced performance & scalability

4. List the various features of SQLite.

- Free version
- Serverless
- Flexible
- Configuration not required

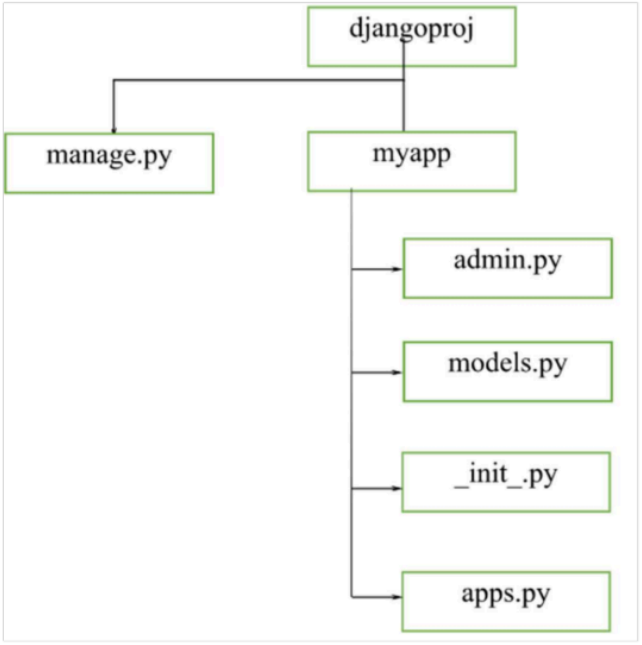
5. Mention the various features of Django framework.

- Rapid development
- Scalability
- Object-relational mapper (ORM)
- REST framework

6. Compare and contrast MERN and MEAN stack.

Parameters	MERN	MEAN
Type	JavaScript framework	JavaScript library
Scalability	Medium	High
Learning Curve	Medim	Low
DOM	Regular	Virtual

7. Draw the project structure of Django for creating bank loan database.



8. Does Django ORM prevent SQL injection? Justify your answer.

Django's ORM automatically escapes user inputs when executing queries, making it inherently resistant to SQL injection. By using parameterized queries and avoiding raw SQL, you can ensure that user inputs are treated as data and not executable code.

(PART B – 13 Marks)

1. Draw the steps involved in developing a web application and explain each.

Web application development is the process of using client-side and server-side programming to develop an application accessible over a web browser.

Step 1 – Find a Genuine App Idea

First, you start with an idea for a web app that solves a problem.

Step 2 – Market Research

Once you have an idea, it's time to understand the users. Both the business perspective and technical direction of the app depend on the people intended to serve.

Step 3 – Define Functionality

One core functionality that your app does best and which is the reason people should use your app must be isolated.

Step 4 – Sketch Your Web App Design

A simple sketch is your first step to web application design. Just be sure to know where the buttons, text and images must be placed. This is also the stage to map out workflow.

Step 5 – Wireframes and Prototypes

A wireframe tells how the app pages will look. The next step would be creating a web app prototype. When some interactivity is added to wireframes with limited functionality, that's called a prototype.

Step 6 – Time to Start Validating

Round up a group of potential users. Get them to try out the web app prototype and begin gathering feedback. This feedback will help you immensely improve the app and make it more usable.

Step 7 – Choose Your Technology

So far, we were dealing with the soft skills of web app development. Once you enter the development phase though, there are larger technical decisions to be made.

Step 8 – Host Your Application on the Web

You will need to choose a server to host your web based application. Buy a domain and set up an SSL certificate. Then, choose a cloud provider like Amazon (AWS) or MS Azure. You could also choose the Google Cloud Platform.

Step 10 – Deploy Your App

Once you have the code and the host, you are ready to go live. Just deploy your app on to your hosting solution and your app is ready for your users.

2. List all the steps involved in cloning a repository in Github, adding the files changed, commit, setting up remote-url and pushing the contents from Visual Studio Code to the repository back.

Step-1: Activate environment

PS. C:/User/Admin> env1/scripts/activate

Step-2: Clone the repository in GitHub

(env1) C:/User/Admin> git clone https://github.com/sec/repo.git

Step-3: Change directory to the name of the repository cloned.

(env1) C:/User/Admin> cd repo

Step-4: Create a Django Project

(env1) C:/User/Admin/repo> django-admin startproject sam

Step-5: Change directory to the sub-folder

(env1) C:/User/Admin/repo> cd sam

Step-6: Create a new app

(env1) C:/User/Admin/repo/sam> python3 manage.py startapp myapp

Step-7: Click on settings.py and insert “import os”. Then insert ‘myapp’ in the configuration INSTALLED_APPS

Step-8: Check whether Django server is installed successfully

(env1) C:/User/Admin/repo/sam> python manage.py runserver 8000

Go to the URL ‘http://localhost:8000/’

Step-9: Create a migration file that contains code for the database schema of a model

(env1) C:/User/Admin/repo/sam> python manage.py makemigrations

Step-10: Apply all the migrations to the database

(env1) C:/User/Admin/repo/sam> python manage.py migrate

Step-11: Create admin

(env1) C:/User/Admin/repo/sam> python manage.py createsuperuser

Username: admin

Email address: admin@gmail.com

Password:

Password (again):

Superuser created successfully.

Step-12: Run Django admin

(env1) C:/User/Admin/repo/sam> python manage.py runserver 8000

Step-13: Copy the program in README.md and add the screenshot of output

Step-14: Inform Git that you want to include updates to a particular file in the next commit.

(env1) C:/User/Admin/repo/sam> git add -A

Step-15: Configure GitHub username

(env1) C:/User/Admin/repo/sam> git config user.name "USERNAME"

Step-16: Configure GitHub email

(env1) C:/User/Admin/repo/sam> git config user.email "EMAIL"

Step-17: Commit all the updates

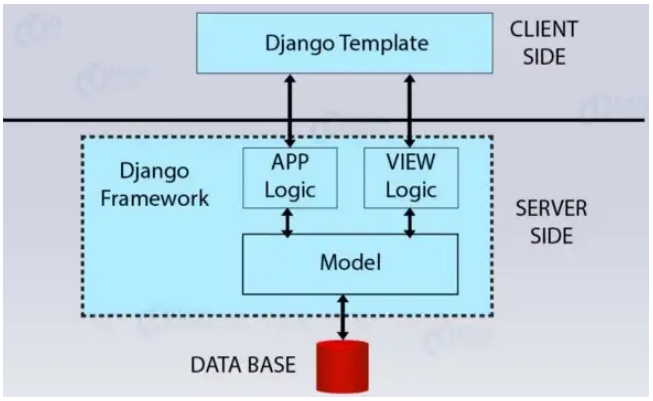
(env1) C:/User/Admin/repo/sam> git commit -m "Success"

Step-18: Push the changes in the branch to the Github repository

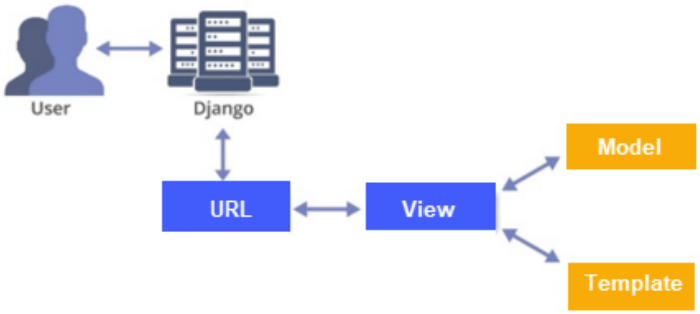
(env1) C:/User/Admin/repo/sam> git push https://github.com/USERNAME/repo.git

3. How should a Django project be structured? Discuss briefly.

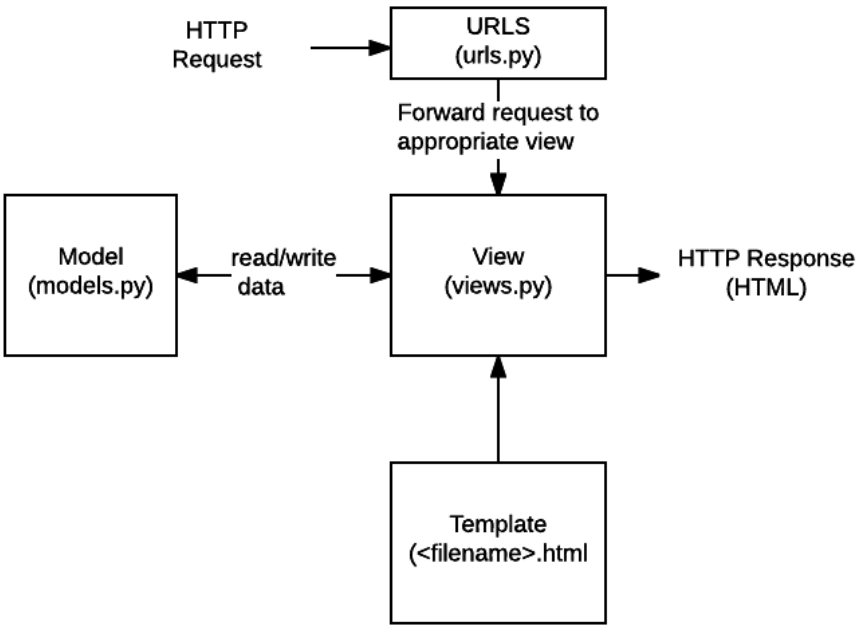
Django is a Python Web Application Framework. It is an open source framework for developing web applications in python. It is based on the Model-View-Template architecture and provides a separation of the data model from the business model and the user interface. Django can work with different databases without changing any code.



Django MVT Pattern



The diagram shows how all the components of the MVT pattern interact with each other to serve specific to a user request. A user requests a resource from Django. Django acts as a controller and checks the available resource in the URL. If URL maps, a view is called which interacts with the model and template. Django then responds to the user and sends a template as a response.



(i) URLs.py

While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.

(ii) view.py

A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via models, and delegate the formatting of the response to templates.

(iii) Models.py

Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.

(iv) Template

A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content.

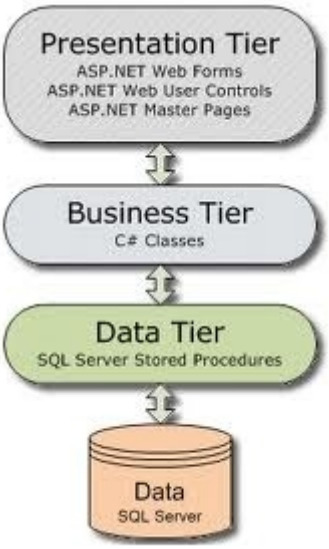
4. Discuss any three web application frameworks.

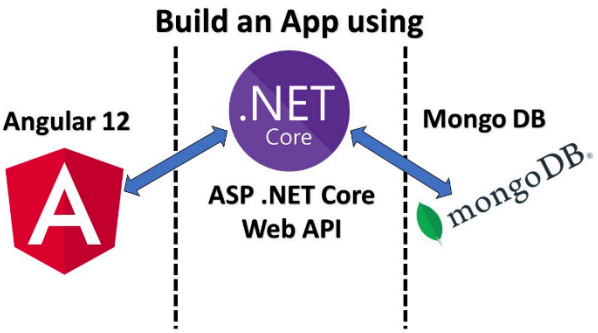
(i).NET core

ASP.NET core is an open-source, server-side web-application framework designed for web development to produce dynamic web pages.

It was developed by Microsoft to allow programmers to build dynamic web sites, applications and services.

The name stands for Active Server Pages Network Enabled Technologies.



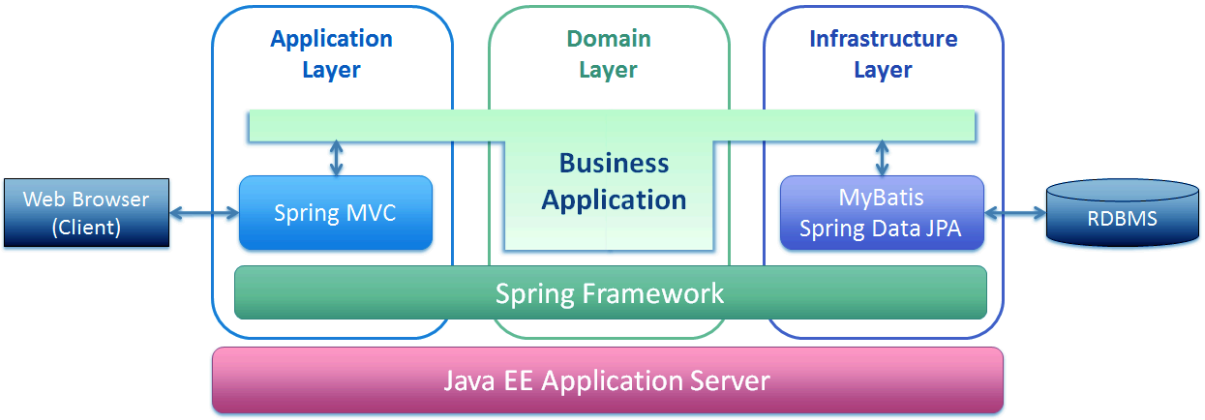


(ii) Spring Framework

Spring Framework is an application framework and inversion of control container for the Java platform.

The framework's core features can be used by any Java application

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.



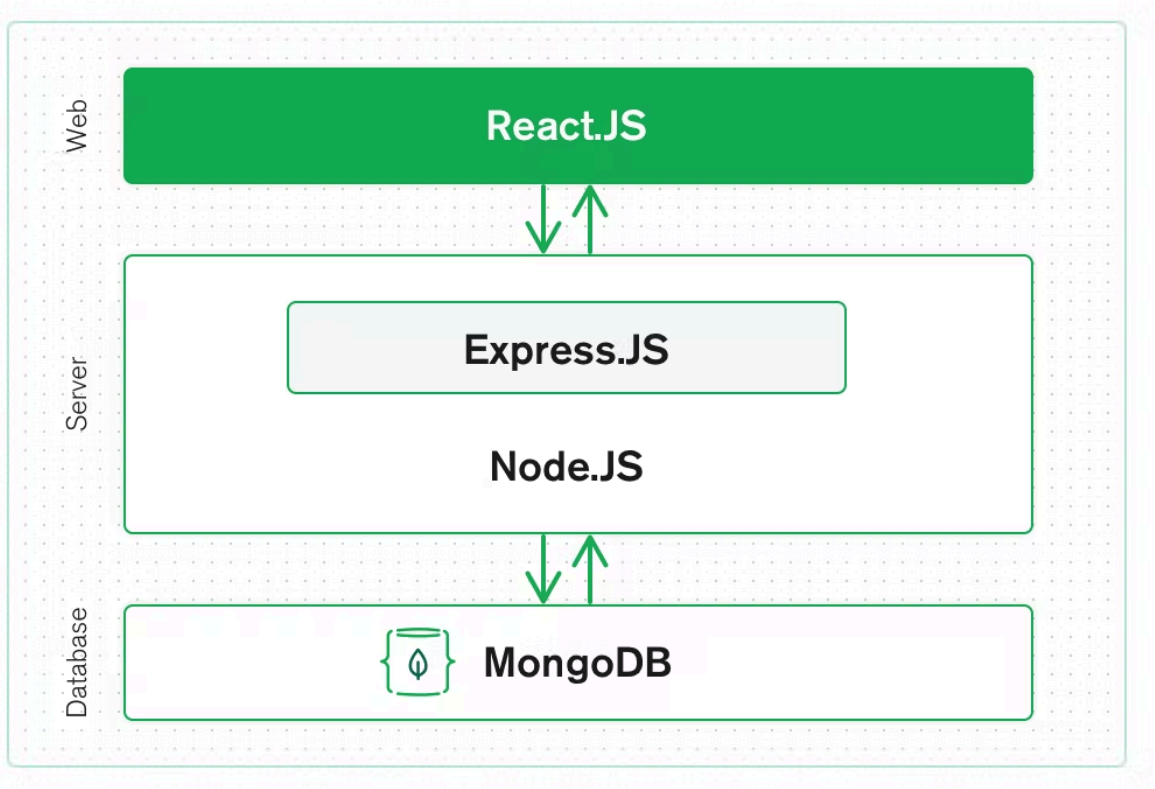
(iii) MERN Stack

MERN Stack is a Javascript Stack that is used for easier and faster deployment of full-stack web applications. MERN Stack comprises of 4 technologies namely:

MongoDB, Express, React and Node.js.

It is designed to make the development process smoother and easier.

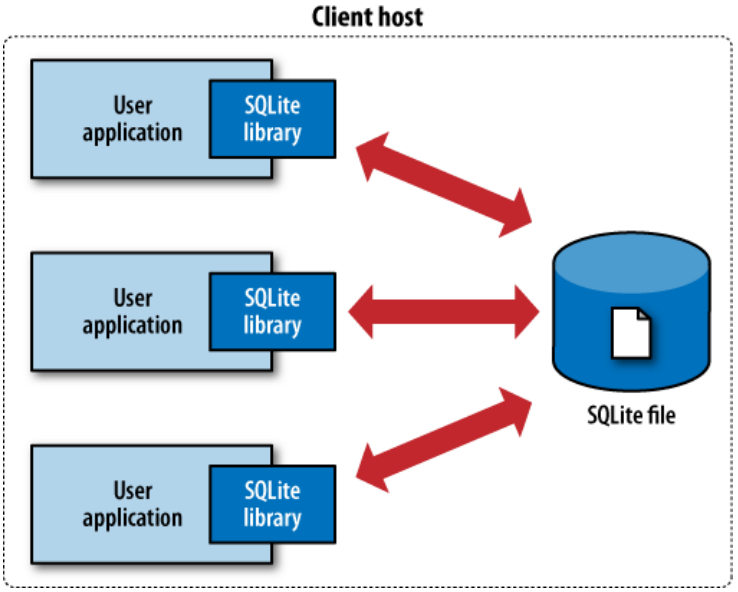
The MERN architecture allows you to easily construct a three-tier architecture (front end, back end, database) entirely using JavaScript and JSON.



5. Explain briefly about the default database engine used as a back end in Django.

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers. SQLite reads and writes directly to ordinary disk files.

A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files.



SQLite Browser

DB Browser for SQLite Portable is a visual tool used to create, design and edit database files compatible with SQLite.

- Create and compact database files
- Create, define, modify and delete tables
- Create, define and delete indexes
- Browse, edit, add and delete records
- Search records
- Import and export records as text
- Import and export tables from/to CSV files
- Import and export databases from/to SQL dump files
- Issue SQL queries and inspect the results
- Examine a log of all SQL commands issued by the application

Configuring the Database

- Set up a database server, activate it, and create a database within it (e.g., using a CREATE DATABASE statement).
- Database configuration lives in the Django in settings file, called settings.py by default. Edit that file and look for the database settings:
    - DATABASE_ENGINE = ''
    - DATABASE_NAME = ''
    - DATABASE_USER = ''
    - DATABASE_PASSWORD = ''
    - DATABASE_HOST = ''
    - DATABASE_PORT = ''

- DATABASE_ENGINE tells Django which database engine to use.

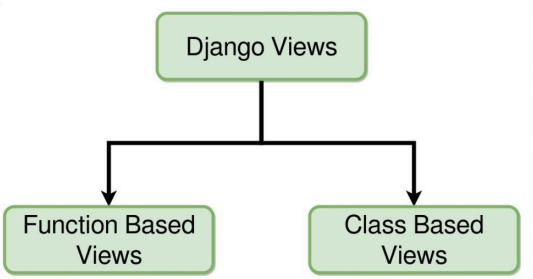
Database	Engine
PostgreSQL	postgresql
MySQL	mysql
SQLite	sqlite3
Oracle	oracle

- DATABASE_NAME tells Django the name of your database.
- DATABASE_USER tells Django which username to use when connecting to your database.
- DATABASE_PASSWORD tells Django which password to use when connecting to your database.
- DATABASE_HOST tells Django which host to use when connecting to your database.
- DATABASE_PORT tells Django which port to use when connecting to your database.

6. Discuss in detail about the role of views in Django.

A view is a callable which takes a request and returns a response. The database operations alone cannot display the application's data to a user. We need a way to display all the stored information in a meaningful way to the user.This is where Django views, templates, and URL mapping come into play. Views are the part of a Django application that takes in a web request and provides a web response. Views are one of the most important parts of a Django application, where the application logic is written. This application logic controls interactions with the database, such as creating, reading, updating, or deleting records from the database. It also controls how the data can be displayed to the user. This is done with the help of Django HTML templates.

Django views can be broadly classified into two types, function-based views and class-based views.



A) Function-based views

Function-based views are implemented as Python functions. A view function, or view for short, is simply a Python function that takes a Web request and returns a Web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image or anything, really. The view itself contains whatever arbitrary logic is necessary to return that response.

Example

```
from django.http import HttpResponse
from django.shortcuts import render

def MyView(request):
    return HttpResponse("Hello world!")
```

B) Class-based Views

Class-based views are implemented as Python classes. Using the principles of class inheritance, these classes are implemented as subclasses of Django's generic view classes. Unlike function-based views, where all the view logic is expressed explicitly in a function, Django's generic view classes come with various pre-built properties and methods that can provide shortcuts to writing clean, reusable views.

This property comes in handy quite often during web development; for example, developers often need to render an HTML page without needing any data inserted from the database, or any customization specific to the user. Developers often need to render an HTML page. Class-based views provide an alternative way to implement views as Python objects instead of functions. They do not replace function-based views, but have certain differences and advantages when compared to function-based views:

Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching. Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components. The major advantage of using class-based views is that fewer lines of code need to be used to implement the same functionality as compared to function-based views. Also, by inheriting Django's generic views, we can keep the code concise and avoid the duplication of code.

However, a disadvantage of class-based views is that the code is often less readable for someone new to Django, which means that learning about it is usually a longer process, as compared to function-based views.

Example

```
from django.http import HttpResponse
from django.shortcuts import render
from django.views import View

def MyView(request):
   return HttpResponse("Hello world!")

class HomePage(View)
        greeting = ‘Good Day’

        def get(self,request):
                return HTTPResponse(self.greeting)
```

7a. Create a model for a bank loan which stores details of 10 customers. Create at least 7 fields.

models.py

```
from django.db import models
from django.contrib import admin
class Loan (models.Model):
    Customer_id=models.CharField(max_length=20, primary_key=True)
    Customer_name=models.CharField(max_length=100)
    Mobile_no=models.IntegerField( )
    Age=models.IntegerField( )
    Email=models.EmailField( )
    DoB=models.DateField( )
    Loan_amount=models.IntegerField( )
class LoanAdmin(admin.ModelAdmin):
    list_display=('Customer_id, 'Customer_name', 'Mobile_no, 'Age', 'Email',
                  'DoB', 'Loan_amount')
```

admin.py

```
from django.contrib import admin
from .models import Loan, LoanAdmin
admin.site.register(Loan, LoanAdmin)
```

7b. Explain about Django model data types and fields list.

A. CharField

This field is used to store strings of up to a specified length. For example, a CharField might be used to store the name of a person, with a maximum length of 50 characters.

B. IntegerField

This field is used to store integers. For example, an IntegerField might be used to store the age of a person.

C. FloatField

This field is used to store floating-point numbers. For example, a FloatField might be used to store the height of a person.

D. BooleanField

This field is used to store boolean values. For example, a BooleanField might be used to indicate whether a user has agreed to the terms and conditions of a website.

E. DateField

This field is used to store dates. For example, a DateField might be used to store the date of birth of a person.

F. DateTimeField

This field is used to store dates and times. For example, a DateTimeField might be used to store the date and time a blog post was published.

G. TimeField

This field is used to store times. For example, a TimeField might be used to store the time that a movie starts.

H. TextField

This field is used to store large text blocks.

I. EmailField

This field is used to store email addresses.

J. URLField

This field is used to store URLs.

K. FileField

This field is used to upload files to the server.

L. ImageField

This field is used to upload image files to the server. For example, an ImageField might be used to store the profile picture of a user.

8. Describe the concept of managing users in Django admin.

Users represent the different types of people that interact with your site. These users might not have the same privileges as the admin, who has complete access to the website. Through the admin/superuser, the following can be achieved,

- Add or delete users
- Edit users' status
- Edit existing users
- Create user groups
- Add users to groups
- Add or remove user permissions
- Reset passwords

A) Creating users

We can create users by using the create_user() helper function, as follows,

```
from django.contrib.auth.models import User
user = User.objects.create_user('XYZ', 'sample@mail.com')

# The user has been successfully created
# We can change its attributes

user.last_name = 'ABC'
user.save()
```

We can also add users interactively using the admin index page. Simply click the green + sign next to the Users entry, enter a username and password, and save.

(i) Creating superusers

A superuser is the admin user. Through the superuser, we can manage the entire website or application. Running the following command from inside your virtual environment creates a superuser:

```
python manage.py createsuperuser
```

Following this command, you will be prompted to enter a username, mail and password for the superuser. After confirming your password, the superuser will be created.

B) User permissions and status

We can also use the admin index page to edit user status. For a normal user, check the boxes that indicate Active and Staff. Do not check the box Superuser if you wish to restrict access to the user. When a user is created, they have no permissions. These permissions are granted by the superuser and are user-specific.

These permissions can be added using the helper functions,

- user_permissions.set([permission_list])
- user_permissions.add(permission, permission, ...)
- user_permissions.remove(permission, permission, ...)

We can also add user permissions using the admin index. Simply click on a user, look for the User Permissions option, and use the drop-down list to add/remove permissions.

C) Managing groups

Groups are a means of categorizing users. This allows for granting permissions to a specific group. It also saves a lot of time, as compared to granting permissions to each individual user. A user may belong to any number of groups and automatically has all the permissions granted to that group.

(i) Creating groups

We create groups in a similar manner to users. Simply navigate to the Add button next to the Groups, enter the credentials, and save. We can also add users to the Group using the admin index.

(ii) Group permissions

Think of group permissions as a way of assigning user permissions in bulk.. We assign permissions to a specific group, for example, by allowing Editors to edit the home screen and then adding users to the Editors group. This is a much more efficient and practical way of assigning user permissions.

PART C – 15 Marks

1. Construct a Django model for electricity bill management system with five different columns in which one field should be the primary key and prepare the required python models to insert unique rows into the table. Write the steps involved to set up the same in the admin interface.

Step-1: Activate environment

PS. C:/User/Admin> env1/scripts/activate

Step-2: Clone the repository in GitHub

(env1) C:/User/Admin> git clone https://github.com/AAA/repo.git

Step-3: Change directory to the name of the repository cloned.

(env1) C:/User/Admin> cd repo

Step-4: Create a Django Project

(env1) C:/User/Admin/repo> django-admin startproject sam

Step-5: Change directory to the sub-folder

(env1) C:/User/Admin/repo> cd sam

Step-6: Create a new app

(env1) C:/User/Admin/repo/sam> python3 manage.py startapp myapp

Step-7: Edit settings.py for allowing all hosts and include ‘myapp’ in ‘INSTALLED_APPS’

Step-8: Check whether Django server is installed successfully

(env1) C:/User/Admin/repo/sam> python manage.py runserver 8000

Go to the URL ‘http://localhost:8000/’ and check.

Step-9: Open models.py and type the following

```
from django.db import models
from django.contrib import admin
class Electricity_Bill (models.Model):
    Customer_ID=models.CharField(max_length=20, primary_key=True)
    Customer_name=models.CharField(max_length=100)
    Bill_no=models.IntegerField()
    Bill_amount=models.IntegerField()
    Bill_date=models.DateField()
class Electricity_BillAdmin(admin.ModelAdmin):
list_display=('Customer_ID','Customer_name','Bill_no','Bill_amount','Bill_date')
```

Step-10: Open admin.py and type the following

```
from django.contrib import admin
from .models import Electricity_Bill, Electricity_BillAdmin
admin.site.register(Electricity_Bill, Electricity_BillAdmin)
```

Step-11: Create a migration file that contains code for the database schema of a model

```
(env1) C:/User/Admin/repo/sam> python manage.py makemigrations
```

Step-12: Apply all the migrations to the database

```
(env1) C:/User/Admin/repo/sam> python manage.py migrate
```

Step-13: Create admin

```
(env1) C:/User/Admin/repo/sam> python manage.py createsuperuser
Username: admin
Email address: admin@gmail.com
Password:
Password (again):
Superuser created successfully.
```

Step-14: Migrate database schema of Book model to ‘myapp’

(env1) C:/User/Admin/repo/sam> python manage.py makemigrations myapp

Step-15: Apply all the migrations to the database in ‘myapp’

(env1) C:/User/Admin/repo/sam> python manage.py migrate myapp

Step-16: Run Django admin

(env1) C:/User/Admin/repo/sam> python manage.py runserver 8000

Go to the URL ‘http://localhost:8000/admin’

Step-17: After logging into the Admin account, create users.

2. Explain in detail about various options in modelAdmin.

The ModelAdmin is very flexible. It has several options for dealing with customizing the interface. All options are defined on the ModelAdmin subclass.

A. ModelAdmin.actions

A list of actions to make available on the change list page.

B ModelAdmin.actions_on_top

Controls where on the page the actions bar appears. By default, the admin changelist displays actions at the top of the page.

Example

actions_on_top = True;

C. ModelAdmin.actions_on_bottom

It displays the admin changelist actions at the bottom of the page.

Example

actions_on_bottom = True;

D. ModelAdmin.actions_selection_counter

Controls whether a selection counter is displayed next to the action dropdown. By default, the admin changelist will display it.

Example

actions_selection_counter = True;

E. ModelAdmin.date_hierarchy

Set date_hierarchy to the name of a DateField or DateTimeField in your model, and the change list page will include a date-based drill down navigation by that field.

Example

date_hierarchy = "pub_date"

F. ModelAdmin.exclude

This attribute, if given, should be a list of field names to exclude from the form. For example, let's consider the following model,

```
class Author(models.Model):
    name = models.CharField(max_length=100)
    title = models.CharField(max_length=3)
    birth_date = models.DateField()
```

If you want a form for the Author model that includes only the name and title fields, you would specify fields or exclude like this:

```
class AuthorAdmin(admin.ModelAdmin):
    fields = ["name", "title"]
        (or)
class AuthorAdmin(admin.ModelAdmin):
    exclude = ["birth_date"]
```

G. ModelAdmin.list_display

Set list_display to control which fields are displayed on the change list page of the admin.

Example

```
list_display = ["first_name", "last_name", "regno"]
```

H. ModelAdmin.list_display_links

Use list_display_links to control which fields in list_display should be linked to the "change" page for an object.

Example

```
list_display_links = ["regno"]
```

I. ModelAdmin.fieldsets

Set fieldsets to control the layout of admin "add" and "change" pages. fieldsets is a list of two-tuples, in which each two-tuple represents a <fieldset> on the admin form page. The two-tuples are in the format (name, field_options), where name is a string representing the title of the fieldset and field_options is a dictionary of information about the fieldset, including a list of fields to be displayed in it.

Example

```
class sampleAdmin(admin.ModelAdmin):
    fieldsets = [
        None,
        {
            "fields": ["regno", "name", "age"],
        },
        "Advanced options",
        {
            "fields": ["address", "mobile"],
        },
        ]
```