Question 1

(a) How is the Model-View-Controller (MVC) design pattern applied in web development

The **Model-View-Controller (MVC)** design pattern is a widely used architectural pattern in web development that separates an application into three interconnected components: **Model**, **View**, and **Controller**. This separation helps organize code, improve maintainability, and make the application more scalable.

The following describe how MVC is applied in web development:

1. Components of MVC

1. **Model**:
    - Represents the **data and business logic** of the application.
    - Interacts with the database to fetch, insert, update, or delete data.
    - Notifies the View when data changes (in some implementations).
2. **View**:
    - Represents the **user interface (UI)** of the application.
    - Displays data to the user and sends user actions (e.g., button clicks) to the Controller.
3. **Controller**:
    - Acts as an **intermediary** between the Model and the View.
    - Handles user input, processes requests, and updates the Model or View accordingly.

2. How MVC Works in Web Development

1. **User Interaction**:
    - The user interacts with the **View** (e.g., clicks a button to create a new post).
    - The View sends the user's action to the **Controller**.
2. **Controller Processing**:
    - The Controller receives the request and processes it.
    - It interacts with the **Model** to perform the necessary business logic (e.g., saving a new post to the database).

3. **Model Updates**:
    - The Model updates the database and returns the result to the Controller.
4. **View Update**:
    - The Controller updates the **View** with the new data (e.g., displaying the newly created post).

---

3. Benefits of MVC in Web Development

1. **Separation of Concerns**:
    - Each component has a specific responsibility, making the codebase easier to manage and maintain.
2. **Reusability**:
    - Models and Controllers can be reused across different parts of the application.
3. **Scalability**:
    - MVC makes it easier to scale the application by allowing developers to work on different components independently.
4. **Testability**:
    - Each component can be tested in isolation, improving the overall quality of the application.

---

4. Example of MVC in a Blogging Platform

- **Model**: Post.js (handles database operations for posts).
- **View**: A webpage or template that displays posts (e.g., post-list.html).
- **Controller**: PostController.js (handles requests to create, read, update, or delete posts).

(b) Describe five (5) common API security vulnerabilities and how to mitigate them.

1. Injection Attacks

- **Description**: Attackers inject malicious code (e.g., SQL, NoSQL, or OS commands) into API requests to manipulate the database or server.
- **Mitigation**:
    - Use parameterized queries or prepared statements to prevent SQL injection.

- Validate and sanitize all user inputs.
- Use ORM (Object-Relational Mapping) libraries to abstract database interactions.

2. Broken Authentication

- **Description**: Weak authentication mechanisms allow attackers to compromise user accounts or tokens.
- **Mitigation**:
  - Use strong password policies and multi-factor authentication (MFA).
  - Implement secure token-based authentication (e.g., JWT) with short expiration times.
  - Store passwords securely using hashing algorithms like bcrypt.

3. Sensitive Data Exposure

- **Description**: APIs may expose sensitive data (e.g., passwords, credit card numbers) due to lack of encryption.
- **Mitigation**:
  - Use HTTPS to encrypt data in transit.
  - Avoid storing sensitive data unnecessarily.
  - Encrypt sensitive data at rest using strong encryption algorithms.

4. Broken Access Control

- **Description**: Attackers gain unauthorized access to resources due to improper access controls.
- **Mitigation**:
  - Implement role-based access control (RBAC) to restrict access to resources.
  - Validate user permissions for every request.
  - Use middleware to enforce access control rules.

5. Security Misconfiguration

- **Description**: APIs may have insecure configurations (e.g., default credentials, exposed debug endpoints).
- **Mitigation**:

- Regularly update and patch software.
- Disable unnecessary features and endpoints.
- Use security headers (e.g., CORS, Content Security Policy) to protect against common attacks.

(c) Explain the role of authentication and authorization in API security.

1. Authentication

- **Definition**: Authentication verifies the identity of a user or system accessing the API.
- **Role**:
  - Ensures that only legitimate users can access the API.
  - Typically involves validating credentials (e.g., username/password) or tokens (e.g., JWT)..

2. Authorization

- **Definition**: Authorization determines what actions an authenticated user or system is allowed to perform.
- **Role**:
  - Ensures that users can only access resources and perform actions they are permitted to.
  - Typically involves checking roles or permissions (e.g., admin, user).

3. Importance in API Security

- **Authentication** prevents unauthorized access to the API.
- **Authorization** ensures that even authenticated users cannot perform actions beyond their permissions.
- Together, they form the foundation of API security.

(D) Using Django REST Framework, implement authentication and permission controls for API endpoints.

The following steps describe how to use Django re4stframework to implement authentication and permission control for an API endpoint.

Step 1: Install Required Packages

Install the following packages Django, Django REST Framework (DRF), and Simple JWT:

pip install django djangorestframework djangorestframework-simplejwt

Step 2: Configure Django Settings

Add rest_framework and rest_framework_simplejwt to your INSTALLED_APPS in settings.py:

```
INSTALLED_APPS = [
    ...
    'rest_framework',
    'rest_framework.authtoken',
    'rest_framework_simplejwt',
]
```

Then configure authentication and permission settings:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    ),
}
```

Step 3: Create Models

Create a such as UserProfile model:

```
from django.contrib.auth.models import User
from django.db import models
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField(blank=True)
    location = models.CharField(max_length=100, blank=True)
    def __str__(self):
        return self.user.username
```

Run migrations:

```
python manage.py makemigrations
python manage.py migrate
```

Step 4: Create Serializers

Create a serializers.py file to convert models into JSON:

```
from rest_framework import serializers
from django.contrib.auth.models import User
from .models import UserProfile
class UserProfileSerializer(serializers.ModelSerializer):
    class Meta:
        model = UserProfile
        fields = ['user', 'bio', 'location']
```

Step 5: Create API Views

Define your API views in views.py:

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated, AllowAny
from rest_framework.authentication import SessionAuthentication, BasicAuthentication,
TokenAuthentication
from .models import UserProfile
from .serializers import UserProfileSerializer
class UserProfileView(APIView):
    permission_classes = [IsAuthenticated]
    def get(self, request):
        profile = UserProfile.objects.get(user=request.user)
        serializer = UserProfileSerializer(profile)
        return Response(serializer.data)
```

Step 6: Register API Endpoints

Update urls.py:

```
from django.urls import path
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView
from .views import PublicView, ProtectedView, UserDetailView, UserProfileView
```

```
urlpatterns = [
    path('api/public/', PublicView.as_view(), name='public'),
    path('api/protected/', ProtectedView.as_view(), name='protected'),
    path('api/user/', UserDetailView.as_view(), name='user-detail'),
    path('api/profile/', UserProfileView.as_view(), name='user-profile'),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
]
```