1. Write the unit test cases for the arrayList by implementing exceptions when user tries to access the index which is not there and implement the other edge cases and try to handle them.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class ArrayListTest {
    private ArrayList<String> list;

    @BeforeEach
    public void setUp() {
        list = new ArrayList<>();
        list.add("Item 1");
        list.add("Item 2");
        list.add("Item 3");
    }

    @Test
    public void testGetExistingIndex() {
        assertEquals("Item 1", list.get(0));
        assertEquals("Item 2", list.get(1));
        assertEquals("Item 3", list.get(2));
    }

    @Test
    public void testGetInvalidIndex() {
        assertThrows(IndexOutOfBoundsException.class, () -> {
            list.get(3); // Index 3 is out of bounds (valid indices are 0 to 2)
        });
    }

    @Test
    public void testAddAndSize() {
        assertEquals(3, list.size());
        list.add("Item 4");
        assertEquals(4, list.size());
        assertEquals("Item 4", list.get(3));
    }

    @Test
    public void testRemove() {
        assertEquals(3, list.size());
        assertTrue(list.remove("Item 2"));
        assertFalse(list.remove("Non-existent Item"));
        assertEquals(2, list.size());
        assertEquals("Item 3", list.get(1)); // Check if subsequent items are shifted correctly
    }

    @Test
    public void testEmptyList() {
        list = new ArrayList<>();
        assertEquals(0, list.size());
        assertThrows(IndexOutOfBoundsException.class, () -> {
            list.get(0); // Accessing index 0 in an empty list should throw an exception
        });
    }
}
```

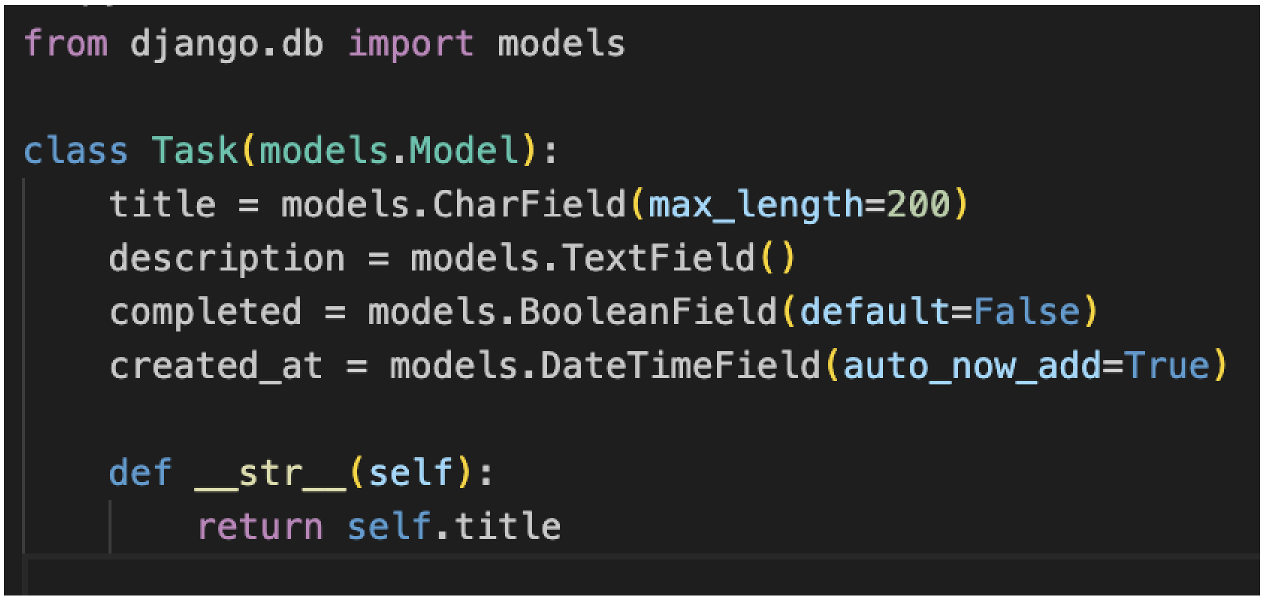
2. Create one spring boot/django/nodejs project (TODO APP) and write the unit test cases on it.
    a. Test the repository methods for saving, retrieving, and deleting tasks.
    b. Test the service layer to ensure business logic is correctly implemented.
    c. Test the controller layer for correct HTTP responses and payloads.

1. **Creating django app -**

```
django-admin startproject todo_project
cd todo_project
python manage.py startapp todoapp
```

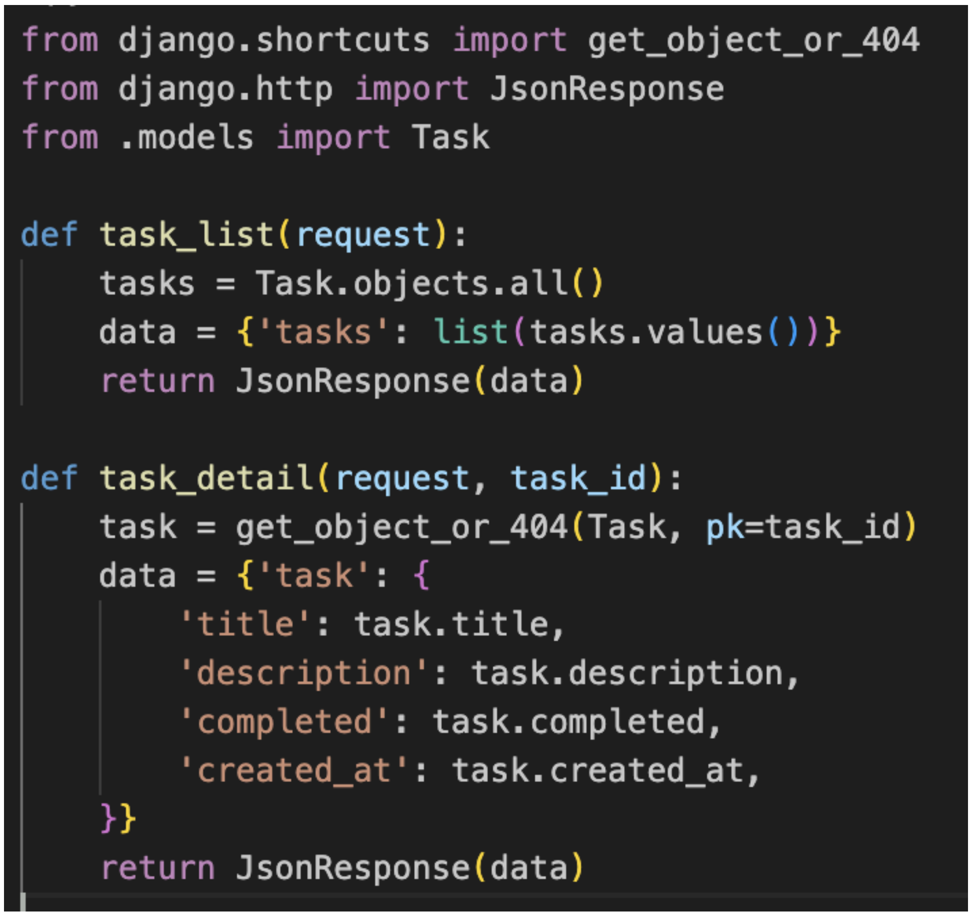
2. **Defining models and views**

models.py -



```python
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    completed = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

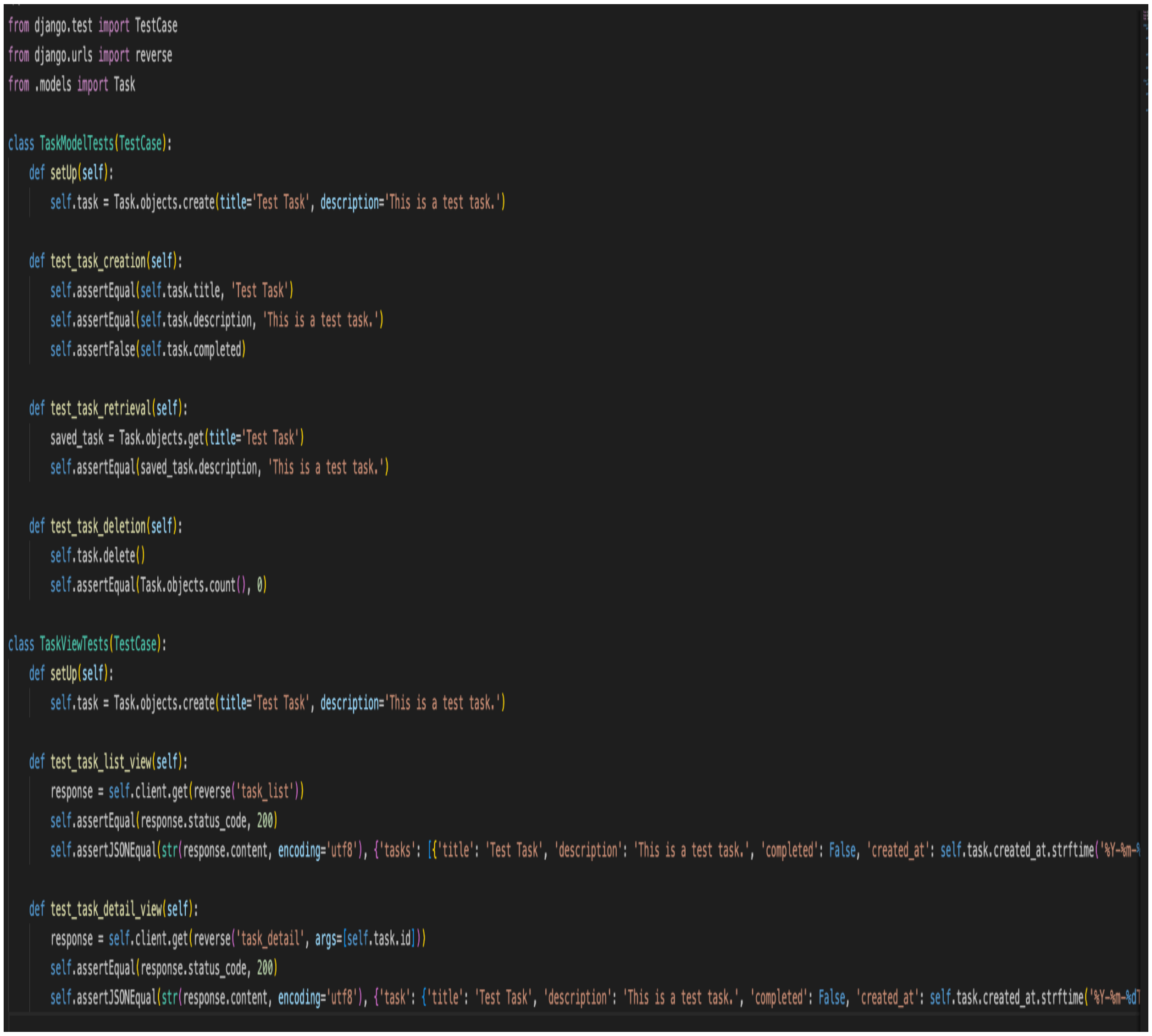
views.py -



```python
from django.shortcuts import get_object_or_404
from django.http import JsonResponse
from .models import Task

def task_list(request):
    tasks = Task.objects.all()
    data = {'tasks': list(tasks.values())}
    return JsonResponse(data)

def task_detail(request, task_id):
    task = get_object_or_404(Task, pk=task_id)
    data = {'task': {
        'title': task.title,
        'description': task.description,
        'completed': task.completed,
        'created_at': task.created_at,
    }}
    return JsonResponse(data)
```

3. **Writing unit tests -**

```
from django.test import TestCase
from django.urls import reverse
from .models import Task

class TaskModelTests(TestCase):
    def setUp(self):
        self.task = Task.objects.create(title='Test Task', description='This is a test task.')

    def test_task_creation(self):
        self.assertEqual(self.task.title, 'Test Task')
        self.assertEqual(self.task.description, 'This is a test task.')
        self.assertFalse(self.task.completed)

    def test_task_retrieval(self):
        saved_task = Task.objects.get(title='Test Task')
        self.assertEqual(saved_task.description, 'This is a test task.')

    def test_task_deletion(self):
        self.task.delete()
        self.assertEqual(Task.objects.count(), 0)

class TaskViewTests(TestCase):
    def setUp(self):
        self.task = Task.objects.create(title='Test Task', description='This is a test task.')

    def test_task_list_view(self):
        response = self.client.get(reverse('task_list'))
        self.assertEqual(response.status_code, 200)
        self.assertJSONEqual(str(response.content, encoding='utf8'), {'tasks': [{'title': 'Test Task',
'description': 'This is a test task.', 'completed': False, 'created_at':
self.task.created_at.strftime('%Y-%m-%dT%H:%M:%SZ')}]})

    def test_task_detail_view(self):
        response = self.client.get(reverse('task_detail', args=[self.task.id]))
        self.assertEqual(response.status_code, 200)
        self.assertJSONEqual(str(response.content, encoding='utf8'), {'task': {'title': 'Test Task',
'description': 'This is a test task.', 'completed': False, 'created_at':
self.task.created_at.strftime('%Y-%m-%dT%H:%M:%SZ')}})
```



```python
from django.test import TestCase
from django.urls import reverse
from .models import Task

class TaskModelTests(TestCase):
    def setUp(self):
        self.task = Task.objects.create(title='Test Task', description='This is a test task.')

    def test_task_creation(self):
        self.assertEqual(self.task.title, 'Test Task')
        self.assertEqual(self.task.description, 'This is a test task.')
        self.assertFalse(self.task.completed)

    def test_task_retrieval(self):
        saved_task = Task.objects.get(title='Test Task')
        self.assertEqual(saved_task.description, 'This is a test task.')

    def test_task_deletion(self):
        self.task.delete()
        self.assertEqual(Task.objects.count(), 0)

class TaskViewTests(TestCase):
    def setUp(self):
        self.task = Task.objects.create(title='Test Task', description='This is a test task.')

    def test_task_list_view(self):
        response = self.client.get(reverse('task_list'))
        self.assertEqual(response.status_code, 200)
        self.assertJSONEqual(str(response.content, encoding='utf8'), {'tasks': [{'title': 'Test Task', 'description': 'This is a test task.', 'completed': False, 'created_at': self.task.created_at.strftime('%Y-%m-

    def test_task_detail_view(self):
        response = self.client.get(reverse('task_detail', args=[self.task.id]))
        self.assertEqual(response.status_code, 200)
        self.assertJSONEqual(str(response.content, encoding='utf8'), {'task': {'title': 'Test Task', 'description': 'This is a test task.', 'completed': False, 'created_at': self.task.created_at.strftime('%Y-%m-%dT
```

4. **Staring app and Running tests -**

    python manage.py startapp todoapp

    python manage.py test todoapp