Django and React Integration Guide: Building a Full-Stack Web Application

1. Introduction

Overview of Django and React

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the model-view-template architectural pattern and includes an ORM for database operations, authentication mechanisms, and a powerful admin interface.

React is a JavaScript library for building user interfaces, particularly single-page applications. It allows developers to create large web applications that can update without reloading the page. React's component-based architecture makes it easier to build reusable UI components and manage application state.

Benefits of Using Django and React Together

- **Specialized Technologies**: Django excels at backend operations like database management, authentication, and business logic, while React specializes in creating dynamic, responsive user interfaces.
- **Separation of Concerns**: The clear division between frontend and backend allows for independent development and testing.
- **Scalability**: Each part of the application can be scaled separately according to needs.
- **Development Speed**: Django's built-in features speed up backend development, while React's component reusability accelerates frontend development.
- **Maintenance**: Updates or changes to one part of the stack don't necessarily affect the other.

Advantages of a Decoupled Architecture

- **Independent Development**: Frontend and backend teams can work simultaneously without blocking each other.
- **Technology Stack Flexibility**: Future technology upgrades can happen independently on either side.

- **Better Testing**: Isolated components are easier to test thoroughly.
- **Performance Optimization**: Each layer can be optimized independently for better performance.
- **API Reusability**: The backend API can be used by multiple clients (web, mobile, IoT, etc.).

2. Setting Up the Development Environment

Installing Required Software

1. **Python and Django**:
    - Install Python (3.8 or higher recommended)
    - Install Django using pip: `pip install django`
    - Install Django REST Framework: `pip install djangorestframework`

2. **Node.js and React**:
    - Install Node.js and npm (Node Package Manager)
    - Install create-react-app globally: `npm install -g create-react-app`

Creating a New Django Project

1. **Set up a virtual environment**:

```
# Create a virtual environment
python -m venv venv

# Activate it
# On Windows
venv\Scripts\activate
# On macOS/Linux
source venv/bin/activate
```

2. **Create a new Django project**:

```
django-admin startproject django_react_project
cd django_react_project
```

3. **Install necessary packages**:

```
pip install djangorestframework django-cors-headers
```

Setting up a React Application

1. **Create a new React application**:

```
npx create-react-app frontend
cd frontend
```

2. **Install necessary packages for the frontend**:

```
npm install axios react-router-dom
```

Project Structure

Your project should now have a structure similar to this:

```
django_react_project/
├── django_react_project/     # Django project settings
├── venv/                     # Python virtual environment
└── frontend/                 # React application
```

3. Building the Django Backend

Creating a Django App for the API

1. **Create a new Django app**:

```
python manage.py startapp api
```

2. **Add the app to your Django settings**:
   In `django_react_project/settings.py`, add to `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # ...
    'rest_framework',
    'corsheaders',
    'api',
]
```

3. **Configure CORS settings**:
   In `settings.py`, add:

```
MIDDLEWARE = [
    # ...
```

```
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    # ...
]

# Allow all origins for development (restrict this in production)
CORS_ALLOW_ALL_ORIGINS = True
```

Setting up Django REST Framework

1. **Configure REST Framework in settings**:

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.TokenAuthentication',
    ],
}
```

Defining Models

Let's create a simple Todo model:

In `api/models.py`:

```
from django.db import models
from django.contrib.auth.models import User

class Todo(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField(blank=True)
    completed = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE)

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

After defining your model, run:

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

Creating Serializers

Serializers convert model instances to JSON:

Create `api/serializers.py`:

```
from rest_framework import serializers
from django.contrib.auth.models import User
from .models import Todo

class TodoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Todo
        fields = ['id', 'title', 'description', 'completed', 'created_at']

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'email']
        extra_kwargs = {'password': {'write_only': True}}

    def create(self, validated_data):
        user = User.objects.create_user(**validated_data)
        return user
```

Implementing Views

Create views for your API:

In `api/views.py`:

```
from rest_framework import viewsets, permissions
from .models import Todo
from .serializers import TodoSerializer, UserSerializer
from django.contrib.auth.models import User
from rest_framework.response import Response
from rest_framework import status
from rest_framework.decorators import api_view, permission_classes

class TodoViewSet(viewsets.ModelViewSet):
    serializer_class = TodoSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        return Todo.objects.filter(user=self.request.user).order_by('-created_at')

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)

@api_view(['POST'])
@permission_classes([permissions.AllowAny])
def register_user(request):
    serializer = UserSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Configuring URLs

Set up URL routing:

Create `api/urls.py`:

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import TodoViewSet, register_user
from rest_framework.authtoken.views import obtain_auth_token

router = DefaultRouter()
router.register(r'todos', TodoViewSet, basename='todo')

urlpatterns = [
    path('', include(router.urls)),
    path('register/', register_user, name='register'),
    path('token/', obtain_auth_token, name='token'),
]
```

Update the main project's `urls.py`:

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

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

4. Developing the React Frontend

Setting Up React Components

Create a component structure:

1. **Create component folders**:

```
mkdir -p src/components/auth
mkdir -p src/components/todos
mkdir -p src/services
```

2. **Create a navigation component** (`src/components/Navigation.js`):

```
import React from 'react';
import { Link } from 'react-router-dom';

const Navigation = ({ isAuthenticated, logout }) => {
  return (
    <nav className="navbar">
      <div className="navbar-brand">Django-React Todo</div>
      <div className="navbar-menu">
```

```
        {isAuthenticated ? (
          <>
            <Link to="/todos">Todos</Link>
            <button onClick={logout}>Logout</button>
          </>
        ) : (
          <>
            <Link to="/login">Login</Link>
            <Link to="/register">Register</Link>
          </>
        )}
      </div>
    </nav>
  );
};

export default Navigation;
```

3. **Create authentication components**:

- `src/components/auth/Login.js`:

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { login } from '../../services/authService';

const Login = ({ setIsAuthenticated }) => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await login(username, password);
      setIsAuthenticated(true);
      navigate('/todos');
    } catch (err) {
      setError('Invalid credentials');
    }
  };

  return (
    <div className="auth-form">
      <h2>Login</h2>
      {error && <div className="error">{error}</div>}
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label>Username:</label>
          <input
            type="text"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
            required
          />
```

```
        </div>
        <div className="form-group">
          <label>Password:</label>
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required
          />
        </div>
        <button type="submit">Login</button>
      </form>
    </div>
  );
};

export default Login;
```

- `src/components/auth/Register.js`:

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { register } from '../../services/authService';

const Register = () => {
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await register(username, email, password);
      navigate('/login');
    } catch (err) {
      setError('Registration failed');
    }
  };

  return (
    <div className="auth-form">
      <h2>Register</h2>
      {error && <div className="error">{error}</div>}
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label>Username:</label>
          <input
            type="text"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label>Email:</label>
```

```
          <input
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label>Password:</label>
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required
          />
        </div>
        <button type="submit">Register</button>
      </form>
    </div>
  );
};

export default Register;
```

4. **Create Todo components**:

- `src/components/todos/TodoList.js`:

```
import React, { useState, useEffect } from 'react';
import { fetchTodos, deleteTodo, updateTodo } from '../../services/todoService';
import TodoForm from './TodoForm';

const TodoList = () => {
  const [todos, setTodos] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  const loadTodos = async () => {
    try {
      const data = await fetchTodos();
      setTodos(data);
      setLoading(false);
    } catch (err) {
      setError('Failed to fetch todos');
      setLoading(false);
    }
  };

  useEffect(() => {
    loadTodos();
  }, []);

  const handleDelete = async (id) => {
    try {
      await deleteTodo(id);
      setTodos(todos.filter(todo => todo.id !== id));
```

```
    } catch (err) {
      setError('Failed to delete todo');
    }
  };

  const handleToggleComplete = async (todo) => {
    try {
      const updatedTodo = { ...todo, completed: !todo.completed };
      await updateTodo(todo.id, updatedTodo);
      setTodos(todos.map(t => t.id === todo.id ? updatedTodo : t));
    } catch (err) {
      setError('Failed to update todo');
    }
  };

  const addTodo = (newTodo) => {
    setTodos([newTodo, ...todos]);
  };

  if (loading) return <div>Loading...</div>;
  if (error) return <div className="error">{error}</div>;

  return (
    <div className="todo-container">
      <h2>My Todos</h2>
      <TodoForm onAddTodo={addTodo} />

      {todos.length === 0 ? (
        <p>No todos yet. Add one!</p>
      ) : (
        <ul className="todo-list">
          {todos.map(todo => (
            <li key={todo.id} className={`todo-item ${todo.completed ? 'completed'
              <span onClick={() => handleToggleComplete(todo)}>
                {todo.title}
              </span>
              <div className="todo-actions">
                <button onClick={() => handleDelete(todo.id)}>Delete</button>
              </div>
            </li>
          ))}
        </ul>
      )}
    </div>
  );
};

export default TodoList;
```

- `src/components/todos/TodoForm.js`:

```
import React, { useState } from 'react';
import { createTodo } from '../../services/todoService';

const TodoForm = ({ onAddTodo }) => {
  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
```

```
  const [error, setError] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!title.trim()) return;

    try {
      const newTodo = await createTodo({ title, description });
      onAddTodo(newTodo);
      setTitle('');
      setDescription('');
      setError('');
    } catch (err) {
      setError('Failed to create todo');
    }
  };

  return (
    <div className="todo-form">
      <h3>Add New Todo</h3>
      {error && <div className="error">{error}</div>}
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label>Title:</label>
          <input
            type="text"
            value={title}
            onChange={(e) => setTitle(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label>Description:</label>
          <textarea
            value={description}
            onChange={(e) => setDescription(e.target.value)}
          />
        </div>
        <button type="submit">Add Todo</button>
      </form>
    </div>
  );
};

export default TodoForm;
```

Creating API Services

Set up services to handle API calls:

1. **API Configuration** (`src/services/apiConfig.js`):

```
import axios from 'axios';

const API_URL = 'http://localhost:8000/api/';
```

```
const api = axios.create({
  baseURL: API_URL
});

// Add request interceptor to include auth token
api.interceptors.request.use(
  config => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers['Authorization'] = `Token ${token}`;
    }
    return config;
  },
  error => Promise.reject(error)
);

export default api;
```

2. **Authentication Service** (`src/services/authService.js`):

```
import api from './apiConfig';

export const login = async (username, password) => {
  const response = await api.post('token/', { username, password });
  const { token } = response.data;

  // Save token to localStorage
  localStorage.setItem('token', token);
  return response.data;
};

export const register = async (username, email, password) => {
  return await api.post('register/', { username, email, password });
};

export const logout = () => {
  localStorage.removeItem('token');
};

export const isAuthenticated = () => {
  return localStorage.getItem('token') !== null;
};
```

3. **Todo Service** (`src/services/todoService.js`):

```
import api from './apiConfig';

export const fetchTodos = async () => {
  const response = await api.get('todos/');
  return response.data;
};

export const fetchTodo = async (id) => {
  const response = await api.get(`todos/${id}/`);
  return response.data;
```

```
};

export const createTodo = async (todoData) => {
  const response = await api.post('todos/', todoData);
  return response.data;
};

export const updateTodo = async (id, todoData) => {
  const response = await api.put(`todos/${id}/`, todoData);
  return response.data;
};

export const deleteTodo = async (id) => {
  return await api.delete(`todos/${id}/`);
};
```

Setting Up Routing

Configure React Router:

Update `src/App.js`:

```
import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Route, Routes, Navigate } from 'react-router-dom
import Navigation from './components/Navigation';
import Login from './components/auth/Login';
import Register from './components/auth/Register';
import TodoList from './components/todos/TodoList';
import { isAuthenticated, logout } from './services/authService';
import './App.css';

function App() {
  const [isAuth, setIsAuthenticated] = useState(isAuthenticated());

  useEffect(() => {
    setIsAuthenticated(isAuthenticated());
  }, []);

  const handleLogout = () => {
    logout();
    setIsAuthenticated(false);
  };

  return (
    <Router>
      <div className="app">
        <Navigation isAuthenticated={isAuth} logout={handleLogout} />
        <main className="container">
          <Routes>
            <Route
              path="/"
              element={isAuth ? <Navigate to="/todos" /> : <Navigate to="/login" /
            />
            <Route
              path="/login"
              element={isAuth ? <Navigate to="/todos" /> : <Login setIsAuthenticat
```

```
            />
            <Route
              path="/register"
              element={isAuth ? <Navigate to="/todos" /> : <Register />}
            />
            <Route
              path="/todos"
              element={isAuth ? <TodoList /> : <Navigate to="/login" />}
            />
          </Routes>
        </main>
      </div>
    </Router>
  );
}

export default App;
```

Add some basic styles in `src/App.css`:

```
.app {
  font-family: Arial, sans-serif;
  max-width: 1200px;
  margin: 0 auto;
  padding: 20px;
}

.navbar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px 0;
  margin-bottom: 30px;
  border-bottom: 1px solid #eee;
}

.navbar-brand {
  font-size: 1.5rem;
  font-weight: bold;
}

.navbar-menu a, .navbar-menu button {
  margin-left: 15px;
  text-decoration: none;
  color: #333;
}

.container {
  padding: 20px;
}

.auth-form {
  max-width: 400px;
  margin: 0 auto;
}

.form-group {
```

```
  margin-bottom: 15px;
}

.form-group label {
  display: block;
  margin-bottom: 5px;
}

.form-group input, .form-group textarea {
  width: 100%;
  padding: 8px;
  border: 1px solid #ddd;
  border-radius: 4px;
}

button {
  padding: 8px 15px;
  background-color: #4285f4;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

button:hover {
  background-color: #3367d6;
}

.error {
  color: red;
  margin-bottom: 10px;
}

.todo-container {
  max-width: 600px;
  margin: 0 auto;
}

.todo-list {
  list-style: none;
  padding: 0;
}

.todo-item {
  padding: 15px;
  border-bottom: 1px solid #eee;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.todo-item.completed span {
  text-decoration: line-through;
  color: #888;
}

.todo-actions {
  display: flex;
```

```
}

.todo-actions button {
  margin-left: 10px;
}
```

5. Integrating Django and React

Connecting Frontend and Backend

The integration between Django and React happens primarily through API calls. We've already set up the services to handle these API calls in the React application.

Key points in the integration:

1. **Proxying in Development**:
   Add a proxy to `package.json` in your React app:

   ```
   "proxy": "http://localhost:8000"
   ```

   This allows you to make API calls without specifying the full URL.

2. **Authentication Flow**:
   - User registers or logs in through React forms
   - Django authenticates and returns a token
   - React stores the token in localStorage
   - All subsequent requests include the token in the Authorization header

3. **Data Flow**:
   - React components request data through service functions
   - Services make API calls to Django endpoints
   - Django processes requests, interacts with the database, and returns data
   - React updates its state with the returned data and re-renders components

Error Handling

Implement consistent error handling throughout your application:

1. **Backend Error Handling**:
   Django REST Framework provides built-in error handling, but you can customize responses:

```
from rest_framework.views import exception_handler

def custom_exception_handler(exc, context):
    response = exception_handler(exc, context)

    if response is not None:
        response.data['status_code'] = response.status_code

    return response
```

Add to `settings.py`:

```
REST_FRAMEWORK = {
    # ...
    'EXCEPTION_HANDLER': 'api.utils.custom_exception_handler',
}
```

2. **Frontend Error Handling**:
    - Use try/catch blocks around API calls
    - Display meaningful error messages to users
    - Consider implementing a global error handling mechanism

6. Authentication and Authorization

Setting Up Token Authentication in Django

1. **Add Django REST Framework's token authentication**:
   Update `settings.py`:

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

2. **Run migrations**:

```
python manage.py migrate
```

3. **Protect API views**:
   We've already added permission classes to our views, but you can customize further:

```
from rest_framework.permissions import IsAuthenticated, IsAuthenticatedOrReadO
```

```
# Different permission levels for different views
class TodoViewSet(viewsets.ModelViewSet):
    permission_classes = [IsAuthenticated]
    # ...
```

Implementing JWT Authentication (Alternative)

For more complex authentication needs, JWT (JSON Web Tokens) provides a stateless solution:

1. **Install Django REST Framework JWT**:

```
pip install djangorestframework-simplejwt
```

2. **Configure in settings**:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
}
```

3. **Update URLs**:

```
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshVi

urlpatterns = [
    # ...
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh
]
```

4. **Adjust frontend service**:

```
export const login = async (username, password) => {
  const response = await api.post('token/', { username, password });
  const { access, refresh } = response.data;

  localStorage.setItem('accessToken', access);
  localStorage.setItem('refreshToken', refresh);
  return response.data;
};

// Add a function to refresh tokens
export const refreshToken = async () => {
  const refresh = localStorage.getItem('refreshToken');
  const response = await api.post('token/refresh/', { refresh });
```

```
  localStorage.setItem('accessToken', response.data.access);
  return response.data;
};
```

7. Deployment

Preparing Django for Production

1. **Update settings for production**:
   Create a `production.py` settings file:

```
from .settings import *

DEBUG = False

ALLOWED_HOSTS = ['your-domain.com', 'www.your-domain.com']

# HTTPS settings
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True

# Database settings (e.g., PostgreSQL)
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'your_db_name',
        'USER': 'your_db_user',
        'PASSWORD': 'your_db_password',
        'HOST': 'your_db_host',
        'PORT': '5432',
    }
}

# Static files
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATIC_URL = '/static/'

# Configure Django to serve React build
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'frontend/build/static'),
]

# Template directory for serving React's index.html
TEMPLATES[0]['DIRS'] = [os.path.join(BASE_DIR, 'frontend/build')]
```

2. **Configure Django to serve the React build**:
   Update the main `urls.py` to serve the React app:

```
from django.views.generic import TemplateView
```

```
urlpatterns = [
    # ... other URLs

    # This should be the last pattern
    path('', TemplateView.as_view(template_name='index.html')),
]
```

3. **Set up a custom middleware to handle React routing**:

   Create `api/middleware.py`:

```
class SPAMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)

        if response.status_code == 404 and not request.path.startswith('/api/'
            return render(request, 'index.html')

        return response
```

   Add to `MIDDLEWARE` in settings:

```
MIDDLEWARE = [
    # ...
    'api.middleware.SPAMiddleware',
]
```

Building the React Application

1. **Create a production build**:

```
cd frontend
npm run build
```

2. **Configure package.json**:

   Add a "homepage" field to `package.json`:

```
"homepage": "/",
```

Deploying to Heroku

1. **Create a Procfile**:

```
web: gunicorn django_react_project.wsgi --log-file -
```

2. **Install Gunicorn**:

```
pip install gunicorn
```

3. **Create requirements.txt**:

```
pip freeze > requirements.txt
```

4. **Create runtime.txt**:

```
python-3.9.7
```

5. **Initialize Git repository**:

```
git init
git add .
git commit -m "Initial commit"
```

6. **Create a Heroku app and deploy**:

```
heroku create your-app-name
heroku addons:create heroku-postgresql:hobby-dev
heroku config:set DJANGO_SETTINGS_MODULE=django_react_project.production
heroku config:set SECRET_KEY='your-secret-key'
git push heroku master
heroku run python manage.py migrate
```

Alternative Deployment Options

1. **AWS**:
   - Deploy Django on EC2 or Elastic Beanstalk
   - Use RDS for the database
   - Serve static files from S3
   - Use CloudFront as a CDN

2. **DigitalOcean**:
   - Deploy using App Platform
   - Or set up a Droplet with Nginx and Gunicorn

3. **Docker**:
   Create a `Dockerfile`:

```
FROM python:3.9

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

# Build React app
WORKDIR /app/frontend
RUN apt-get update && apt-get install -y nodejs npm
RUN npm install
RUN npm run build

WORKDIR /app

EXPOSE 8000

CMD ["gunicorn", "django_react_project.wsgi:application", "--bind", "0.0.0.0:8
```

8. Conclusion

Recap

In this guide, we've built a full-stack web application using Django and React. We:

- Set up a Django project with Django REST Framework to create a robust API
- Built a React frontend with components for todos and authentication
- Connected the frontend and backend through API calls
- Implemented authentication using tokens
- Prepared the application for production deployment

Scalability and Maintainability Considerations

- **Database Optimization**: As your application grows, consider optimizing database queries and adding indexes
- **Caching**: Implement Redis or