------------API’S------------

A certificate of course

Description automatically generated

[Introduction to the course | Coursera](https://www.coursera.org/learn/apis/lecture/Gg5rw/introduction-to-the-course) 4-5-2024 1555

* Course into

[How are APIs used in the real world? | Coursera](https://www.coursera.org/learn/apis/lecture/TvTcn/how-are-apis-used-in-the-real-world) 4-5-2024 1600

* Day in the life of a software engineer at Meta

[What you know about HTTP | Coursera](https://www.coursera.org/learn/apis/lecture/LS9Jb/what-you-know-about-http) 4-5-2024 1608

**Summary: This video provides a refresher on the HTTP and HTTPS protocols, including methods, requests, responses, and status codes. It emphasizes the importance of these concepts for API development.**

**Acronyms**

* **HTTP:** Hypertext Transfer Protocol - The foundational protocol for data communication on the web.
* **HTTPS:** Hypertext Transfer Protocol Secure - An encrypted version of HTTP for secure data transmission.
* **SDLC:** Software Development Life Cycle - A structured framework for software development, encompassing stages from planning to maintenance.
* **API:** Application Programming Interface - A set of rules and specifications that software programs can follow to communicate with each other.

**Terms**

* **Protocol:** A standardized set of rules governing how computers communicate with each other.
* **Client:** A device or software (like a web browser) that requests and receives data.
* **Server:** A computer or software that receives requests and sends back responses with data or resources.
* **Encryption:** The process of scrambling data to prevent unauthorized access.
* **Decryption:** The process of unscrambling encrypted data to its readable form.

**Technical Words**

* **Resource:** Any piece of data on a server (e.g., image, document, webpage).
* **URL/Path:** The specific address of a resource on the web.
* **Body:** The optional data payload included in an HTTP request or response.
* **Referrer:** Information identifying the previous webpage that linked to the current request.
* **Cookies:** Small text files stored by your browser to remember information about your interactions with websites.
* **ETags:** Identifiers attached to resources to help with caching and determining if content has changed.

**Ideas**

* **Data transmission over the web:** The concept of how information travels between clients and servers.
* **Secure communication:** The importance of encryption (HTTPS) for transmitting sensitive data.
* **HTTP Methods (Verbs):** The different actions used to interact with resources (GET, POST, PUT, PATCH, DELETE).
* **HTTP Requests and Responses:** The structured messages sent between a client and a server.
* **HTTP Status Codes:** Numerical codes indicating the result of a request.

**Keywords**=['HTTP', 'HTTPS', 'methods', 'request', 'response', 'status code', 'data', 'server', 'client', 'encryption', 'API']

Let me know if you have other transcripts or texts to analyze. I'm happy to help! 🤖✨

[RESTfulness | Coursera](https://www.coursera.org/learn/apis/lecture/dzYNs/restfulness) 4-5-2024 1623

**Summary:** This video introduces REST APIs, explaining their architecture, key constraints, and the concept of resources within an API. It emphasizes the importance of RESTful design for building maintainable APIs.

**Acronyms**

* **API:** Application Programming Interface

**Terms**

* **REST:** Representational State Transfer - An architectural style for designing APIs.
* **Resource:** A piece of data or an entity within an API (e.g., order, customer, menu item).
* **Endpoint:** A specific URL that exposes a resource within an API.
* **Stateless:** The server doesn't maintain information about previous client requests.
* **Cacheable:** API responses can be stored to reduce server load.
* **Layered System:** An API architecture can have distinct components (e.g., firewall, server, database).
* **Uniform Interface:** There's a standardized way of interacting with resources in an API.
* **XML:** Extensible Markup Language - A format for representing data.
* **JSON:** JavaScript Object Notation - Another common data representation format.

**Technical Words**

* **Client-Server Architecture:** A model where a client requests data and a server provides it.
* **HTTP Request:** A message sent by a client to initiate an action on the server.

**Rarely Used Words**

* **Decoupled:** Separated into independent components.

**Software Packages**

* **Web browser** (implied; acts as a client)

**Ideas**

* **REST API Constraints:** Rules that define a RESTful API:
  + Client-server separation
  + Statelessness
  + Cacheability
  + Layered system
  + Uniform interface
  + Code on demand (optional)
* **API Resources:** The data entities exposed by an API.
* **API Endpoints:** URLs used to interact with resources.

**Keywords**=['REST', 'API', 'resource', 'client', 'server', 'stateless', 'cache', 'endpoint', 'HTTP']

**Let me know if you'd like a deeper explanation of any of these concepts! 🤖**

[Naming conventions | Coursera](https://www.coursera.org/learn/apis/lecture/1jSCM/naming-conventions)

**Summary:** This video teaches you how to design clear and effective REST API endpoints. It emphasizes the importance of naming conventions for improving developer understanding and overall project maintainability.

**Acronyms**

* **API:** Application Programming Interface
* **URI:** Uniform Resource Identifier
* **URL:** Uniform Resource Locator (a specific kind of URI)
* **CRUD:** Create, Read, Update, Delete

**Terms**

* **Endpoint:** A URL that provides access to a specific API resource.
* **Resource:** A piece of data or object that an API manages.
* **Hierarchical relationship:** A nested structure of related resources within an API.
* **Query string:** Parameters within a URL (following a '?') used for filtering or specifying data formats.

**Technical Words**

* **XML:** Extensible Markup Language - A data representation format.
* **JSON:** JavaScript Object Notation - A common data representation format.
* **Trailing slash:** A forward slash (/) at the end of an endpoint.

**Ideas**

* **Importance of endpoint naming:** Well-named endpoints make APIs easier to understand and use.
* **Hierarchical relationships:** Using forward slashes (/) to show relationships between resources.
* **Nouns for resources:** Endpoints should use nouns to represent the data they provide.
* **HTTP methods for actions:** Use HTTP methods (GET, POST, PUT, DELETE) to manipulate data, not endpoint names.
* **Query parameters:** For filtering, searching, and specifying output formats of API data.

**Best Practices**

* **Lowercase letters and hyphens:** for endpoint names.
* **Nouns for resources.**
* **Forward slashes (/) for hierarchy.**
* **Query parameters for filtering and data formats.**
* **No trailing slashes.**
* **Avoid file extensions.**
* **Consistency is key!**

**Keywords**=['REST', 'API', 'endpoint', 'naming conventions', 'URI', 'resource', 'HTTP']

**Let me know if you'd like more details on any of these concepts or additional tips for building RESTful APIs! 🤖✨**

[Good routes versus bad routes | Coursera](https://www.coursera.org/learn/apis/supplement/wRYeY/good-routes-versus-bad-routes)4-5-2024 1639

* Basic stuff, but read over it anyway.

[Essential tools for API development | Coursera](https://www.coursera.org/learn/apis/lecture/GfIq7/essential-tools-for-api-development) 4-5-2024 1935

**Short Summary**

This video introduces command-line and graphical tools for working with APIs. It covers curl, Postman, and Insomnia, demonstrating how to make HTTP requests and test API endpoints.

**Detailed Breakdown**

* **Command-Line Tools**
* **Graphical API Clients**

**Acronyms**

* **API:** Application Programming Interface (a way for software components to communicate)
* **HTTP:** Hypertext Transfer Protocol (the foundation of web communication)
* **REST:** Representational State Transfer (an architectural style for APIs)

**Technical Words**

* **Client:** A piece of software that makes requests to a server.
* **Endpoint:** A specific URL within an API that handles a particular type of request.
* **Query Argument:** Data passed within a URL using the format ?key=value.

**Software**

* **curl:** A command-line tool for making HTTP requests.
* **Postman:** A graphical API client with a web version.
* **Insomnia:** A desktop-based graphical REST API client.

**Commands/Modules**

* **curl** (e.g., curl https://api.example.com/data)

**Ideas/Techniques**

* **API Testing:** The process of ensuring APIs work correctly and meet expectations.

**Keywords**

Python

keywords = ["API", "HTTP", "GET", "POST", "request", "response", "Insomnia", "curl"]

**Mistakes and Misleading Information**

* The video implies that curl doesn't have a graphical interface. There are, in fact, some GUI wrappers for curl, but they are less common than dedicated tools like Postman and Insomnia.

**Tips**

* Use a graphical API client like Insomnia for most API development and testing.
* Learn curl for quick command-line interactions with APIs.

Written by Gemini, you can call me the API adventurer 🚀

[Installing VS Code | Coursera](https://www.coursera.org/learn/apis/supplement/SvzGJ/installing-vs-code) 4-5-2024 1937

* Don’t need this but its here if someone does.

[Optional: Creating a Django project (steps and code) | Coursera](https://www.coursera.org/learn/apis/supplement/oQrdH/optional-creating-a-django-project-steps-and-code) 4-5-2024 1950

* Simple exercise setting up a project. No coding in the views.

[REST best practices | Coursera](https://www.coursera.org/learn/apis/lecture/VQtkp/rest-best-practices) 4-6-2024 0359

* Best practices. Nothing special that required notes.

[Security and authentication in REST API | Coursera](https://www.coursera.org/learn/apis/lecture/VT1fa/security-and-authentication-in-rest-api) 4-6-2024 0410

**Short Summary**

This video discusses API security techniques to safeguard data and restrict access. It covers concepts like SSL, signed URLs, token-based authentication, CORS, and firewalls.

**Detailed Breakdown**

* **Transport Security**
* **Access Control**
* **API Restrictions**

**Acronyms**

* **API:** Application Programming Interface
* **SSL:** Secure Sockets Layer
* **HTTPS:** Hypertext Transfer Protocol Secure
* **CORS:** Cross-Origin Resource Sharing
* **JWT:** (JSON Web Token) A standard for securely encoding authentication information.
* **HMAC:** (Hash-based Message Authentication Code) A mechanism for verifying data integrity and authenticity using a shared secret key.

**Technical Words**

* **Encryption:** The process of scrambling data to make it unreadable without a secret key.
* **Authentication:** The process of verifying a user's identity.
* **Authorization:** The process of granting specific permissions to an authenticated user.
* **Firewall:** A system that filters network traffic based on defined rules.

**Ideas/Techniques**

* **SSL/TLS Certificates:** Used to encrypt data in transit (HTTPS).
* **Signed URLs:** Verify request origin and limit resource access temporarily.
* **Token-based Authentication:** Send a unique token with requests instead of username/password credentials.
* **CORS Policy:** Controls cross-domain requests from web browsers.

keywords = ["API", "security", "authentication", "authorization", "encryption", "HTTPS", "token", "CORS"]

**Mistakes and Misleading Information**

* **HTTP Basic Authentication:** The video suggests this is less secure than token-based authentication for APIs. While tokens offer advantages, basic authentication can still be reasonably secure if used over HTTPS.
* **Firewall scope:** The video mentions using firewalls to restrict API access by IP address. While true, firewalls are more commonly used at the network level, rather than specifically within the API application code.

**Tips**

* Always use HTTPS to secure API communication.
* Prefer token-based authentication over repeatedly sending user credentials.
* Implement appropriate authorization mechanisms to restrict data access based on user roles.

Written by Gemini, you can call me the API guardian 🛡️

[Access control | Coursera](https://www.coursera.org/learn/apis/lecture/ui6mR/access-control) 4-6-2024 0418

**Short Summary**

This video introduces access control in APIs, explaining how it protects sensitive data. It discusses roles, privileges, and the importance of careful planning in designing an API authorization system.

**Technical Words**

* **Access Control:** The process of restricting access to data and resources based on defined rules.
* **Authorization:** The aspect of access control that determines what actions a user is allowed to perform after they have been authenticated.
* **Authentication:** The process of verifying a user's identity.
* **Privilege:** A specific permission to perform an action on a resource.

**Ideas/Techniques**

* **Role-Based Access Control (RBAC):** A common model where users are assigned roles, and roles define allowed actions (privileges).

keywords = ["API", "authorization", "authentication", "access control", "role", "privilege"]

**Tips**

* Carefully design roles and privileges for granular control over API access.
* Consider how roles might overlap or be combined for complex user permissions.

**Mistakes and Misleading Information**

* The video implies a strict separation between 'authentication' and 'authorization.' In practice, these concepts work together; authorization often relies on the identity established during authentication.

Written by Gemini, you can call me the permission protocol expert 🧐

[Authentication versus authorization | Coursera](https://www.coursera.org/learn/apis/supplement/DesQl/authentication-versus-authorization) 4-6-2024 0421

**Short Summary**

This reading explains the difference between authentication and authorization in API security. It emphasizes that authentication verifies user identity, while authorization determines what actions an authenticated user is allowed to perform.

**Acronyms**

* **API:** Application Programming Interface
* **HTTP:** Hypertext Transfer Protocol

**Technical Words**

* **Authentication:** The process of verifying a user's identity (e.g., with username and password).
* **Authorization:** The process of granting or denying specific permissions to an authenticated user.
* **Token-based authentication:** An authentication mechanism using a unique token to identify users after login.
* **Privilege:** A specific permission to perform an action within a system.
* **Role:** A collection of privileges within an authorization system.

keywords = ["authentication", "authorization", "API", "token", "privilege", "HTTP", "Django"]

**Mistakes and Misleading Information**

* The reading suggests that authentication and authorization are entirely separate processes. In reality, authorization often depends on the identity established during authentication.

Written by Gemini, you can call me the the security specialist 🕵️‍♂️

[Knowledge check: Principles of API development | Coursera](https://www.coursera.org/learn/apis/quiz/9HxZr/knowledge-check-principles-of-api-development) 4-6-2024 0425

* 100%

[Book List API project | Coursera](https://www.coursera.org/learn/apis/lecture/C1RBI/book-list-api-project) 4-6-2024 0434

**Summary**

This video introduces how to build a basic Django API for managing a bookstore's book inventory. It covers concepts of creating Django models, API endpoints, CRUD operations, and working with JSON data.

**Acronyms**

* **API:** Application Programming Interface
* **CRUD:** Create, Read, Update, Delete
* **HTTP:** Hypertext Transfer Protocol
* **JSON:** JavaScript Object Notation

**Technical Words**

* **Django model:** A Python class that represents a database table.
* **API endpoint:** A specific URL within an API that handles a certain type of request.
* **HTTP Methods:** Verbs (GET, POST, PUT, DELETE) that specify the action to be performed on a resource.
* **Payload:** Data sent in the body of an HTTP request.

**Software**

* **Insomnia:** API testing tool.

**Commands/Modules**

* **model\_to\_dict()** : Converts a Django model instance to a Python dictionary.
* **JsonResponse()**: Creates an HTTP response with JSON data.
* **QueryDict**: Parses the HTTP request body into a Python dictionary.

keywords = ["Django", "API", "model", "endpoint", "CRUD", "HTTP", "JSON", "Insomnia"]

**Tips**

* Plan your API structure and endpoints before starting to code.

**Mistakes and Misleading Information**

* The video states that a single book should not be returned within square brackets in the JSON response. While technically accurate, it's common practice for APIs to always return lists, even if containing a single item, for consistency.

by Gemini, your API architect 🏛️

[Organizing an API project | Coursera](https://www.coursera.org/learn/apis/lecture/V4dvn/organizing-an-api-project) 4-5-2024 0735

**Short Summary**

This video explains how to organize Django projects using multiple apps to improve maintainability, scalability, and reusability. It emphasizes techniques like versioning, virtual environments, and code structuring.

**Technical Words**

* **Decoupling:** Reducing dependencies between components of your project.
* **Virtual Environment:** An isolated Python environment to manage project-specific dependencies.
* **Versioning:** Tracking changes to APIs over time, ensuring compatibility.

**Software**

* **pipenv:** Dependency management tool for Python projects.
* **Django-splits-settings:** A tool for splitting Django settings files.

**Commands/Modules**

* **pip freeze > requirements.txt** (Generate a requirements file listing dependencies)

**Ideas/Techniques**

* **Splitting into multiple apps:** Break down large projects into smaller, focused apps.
* **Isolated dependencies:** Use virtual environments (like pipenv) to avoid conflicts.
* **API versioning:** Maintain compatibility when APIs change.
* **Requirements file:** Track project dependencies for deployment and collaboration.
* **Organized resources:** Separate static and template files by app.
* **Split settings:** Break down the Django settings file for manageability.
* **Business logic in models:** Keep models self-contained and reusable.

keywords = ["Django", "app", "dependencies", "virtual environment", "versioning", "API", "maintainability"]

**Tips**

* Plan your project structure in advance.
* Use a virtual environment for each project.
* Document your dependencies (requirements.txt).

**Mistakes and Misleading Information**

* The suggestion to always store requirements in a requirements.txt file is true for pip but not necessarily for pipenv (which uses a Pipfile).

by Gemini, your project architect 🏗️

[Consequences of a poorly designed API project | Coursera](https://www.coursera.org/learn/apis/supplement/KqDmX/consequences-of-a-poorly-designed-api-project) 4-6-2024 0753

* Nothing really to note. Check it out.

[XML and JSON response types | Coursera](https://www.coursera.org/learn/apis/supplement/io8Z2/xml-and-json-response-types) 4-6-2024 0757

* No notes

[Exercise: Your first API | Coursera](https://www.coursera.org/learn/apis/ungradedLab/zwFQt/exercise-your-first-api) 4-6-2024 1245

Very basic

[Debugging your API | Coursera](https://www.coursera.org/learn/apis/lecture/hfpuh/debugging-your-api) 4-6-2024 1305

**Summary**

This video introduces the Visual Studio Code debugger and demonstrates how to use it to find and fix errors in a Django application. It covers breakpoints, the watch list, and the debug toolbar.

**Technical Words**

* **Debugging:** The process of finding and fixing errors (bugs) in code.
* **Breakpoint:** A point in code where execution is intentionally paused for inspection.
* **Watch list:** A panel for monitoring the values of variables during debugging.
* **Debug toolbar:** A set of controls for stepping through code execution.

**Software**

* **Visual Studio Code:** A popular code editor with integrated debugging tools.

keywords = ["Django", "debugging", "breakpoint", "watch", "VS Code"]

**Mistakes and Misleading Information**

**Tips**

* Use the debugger to step through your code line by line when troubleshooting.
* Add variables to the watch list to monitor their values.

Written by Gemini, your debugging detective 🕵️‍♀️

[Browser tools and extensions for API development | Coursera](https://www.coursera.org/learn/apis/lecture/AAsvc/browser-tools-and-extensions-for-api-development) 4-6-2024 1322

**Summary**

This video explains how to use the Chrome browser's developer console to inspect API calls and responses. It covers the Network tab, headers, previews, and using a JSON formatter extension.

**Technical Words**

* **Developer console:** A set of tools within web browsers for debugging and inspecting web applications.
* **HTTP request/response headers:** Metadata associated with an API call.
* **API call:** The act of requesting data from an API endpoint.
* **Caching:** Storing data locally to avoid repeated fetches from the server.
* **JSON:** JavaScript Object Notation – a common format for API data.

**Software**

* **Chrome Browser**
* **JSON Formatter Extension**

**Commands/Modules**

* **fetch()**: A JavaScript function to initiate API requests (example: fetch('https://api.example.com'))

keywords = ["API", "HTTP", "browser", "developer console", "Network", "JSON"]

**Tips**

* Use the Network tab to examine API calls made from your web application.
* Check the 'Disable cache' option for fresh API responses during development.
* Install a JSON Formatter extension for easier viewing of JSON output.

Written by Gemini, your web detective 🔎

[Mock APIs | Coursera](https://www.coursera.org/learn/apis/supplement/4AQWp/mock-apis) 4-6-2024 1327

**Short Summary**

This reading explains the concept of mock APIs and how they streamline development by allowing frontend and backend teams to work independently. It highlights the benefits of mock APIs, such as reduced waiting time and improved maintainability.

**Technical Words**

* **Mock API:** A simulated API that returns predetermined data, mimicking a real API's behavior.
* **API endpoint:** A specific URL within an API that handles a certain type of request.
* **Client application:** Software that interacts with an API (e.g., web app, mobile app).

**Software**

* **Mockaroo:** A tool for generating fake data. (<https://www.mockaroo.com/>)
* **Mockapi:** A service for creating mock API endpoints. (<https://mockapi.io/>)

**Ideas/Techniques**

* **Mocking:** The practice of simulating API behavior to facilitate parallel development.

keywords = ["API", "mock API", "endpoint", "development", "client application"]

**Tips**

* Use mock APIs to decouple frontend and backend development, saving time.

**Let me know if you'd like to explore specific use cases for mock APIs or get hands-on with tools like Mockaroo or Mockapi!** Written by Gemini, your API simulation specialist 🤖

[Module summary: REST APIs | Coursera](https://www.coursera.org/learn/apis/lecture/Ytx1H/module-summary-rest-apis) 4-6-2024 1328

* Summary

[Module quiz: REST APIs | Coursera](https://www.coursera.org/learn/apis/exam/q0HtL/module-quiz-rest-apis) 4-6-2024 1336

80% first try, 100% second

------------Week 2--------------

[What is the Django REST framework (DRF)? | Coursera](https://www.coursera.org/learn/apis/lecture/nfrx2/what-is-the-django-rest-framework-drf) 4-6-2024 1744

* Basic introduction

[Installing and setting up DRF | Coursera](https://www.coursera.org/learn/apis/lecture/EE4hl/installing-and-setting-up-drf) 4-6-2024 1815

**Short Summary**

This video demonstrates how to install and set up the Django Rest Framework (DRF) within a Django project. It covers using pipenv for dependency management, creating API endpoints, and handling different HTTP methods.

**Acronyms**

* **DRF:** Django Rest Framework
* **API:** Application Programming Interface
* **HTTP:** Hypertext Transfer Protocol

**Technical Words**

* **Virtual environment:** An isolated Python environment for project-specific dependencies.
* **Dependency management:** The process of handling the packages a project relies on.
* **Django Rest Framework:** A toolkit for building APIs with Django.
* **pipenv:** Dependency management tool (combines Pip and Virtualenv functionalities).
* **Insomnia:** API testing tool.

**Commands/Modules**

* **pipenv install <package>** Installs a package and manages it within the project's virtual environment.
* **pipenv shell** Activates the project's virtual environment.
* **django-admin startproject <projectname>**: Creates a new Django project.
* **python manage.py startapp <appname>**: Creates a new Django app.

**Ideas/Techniques**

* **API view decorator (@api\_view):** DRF decorator to designate a function as an API view and specify allowed HTTP methods.

keywords = ["Django", "DRF", "API", "pipenv", "virtual environment", "endpoint", "HTTP", "Insomnia"]

**Mistakes and Misleading Information**

* **The video uses 'pip or pip3' interchangeably.** It's important to be consistent and use the appropriate command (pip or pip3) depending on your Python setup.

Written by Gemini, your API setup guide 🚀

[Better API view with decorators | Coursera](https://www.coursera.org/learn/apis/lecture/qSzQ6/better-api-view-with-decorators) 4-6-2024 1838

**Short Summary**

This video explains how the @api\_view decorator in Django Rest Framework (DRF) transforms regular function-based views into robust API endpoints. It highlights features like the browsable API interface, HTTP method specification, and support for concepts like throttling and authentication.

**Acronyms**

* **DRF:** Django Rest Framework
* **API:** Application Programming Interface
* **HTTP:** Hypertext Transfer Protocol

**Technical Words**

* **Decorator:** A Python construct that modifies the behavior of a function.
* **Browsable API:** A user-friendly interface for interacting with an API directly in the web browser.
* **Throttling/Rate-limiting:** Controlling the frequency of API requests to prevent overuse.
* **Authentication:** The process of verifying a user's identity.

**Software**

* **Django Rest Framework**

**Commands/Modules**

* **@api\_view:** The decorator used to designate function-based views as API endpoints (e.g., from rest\_framework.decorators import api\_view)
* **Response:** A DRF class for building HTTP responses (e.g., from rest\_framework.response import Response)

**Ideas/Techniques**

* **API Design with Decorators:** Using @api\_view simplifies the creation of well-structured API endpoints.

keywords = ["Django", "DRF", "API", "decorator", "@api\_view", "browsable API", "HTTP", "throttling", "authentication"]

Written by Gemini, your API enhancement specialist 💫

[Different types of routing in DRF | Coursera](https://www.coursera.org/learn/apis/supplement/cFRCv/different-types-of-routing-in-drf) 4-7-2024 0430

**Short Summary**

This reading explains various ways to define URL patterns (routing) in a Django Rest Framework API project. It covers traditional function-based routing, class-based views, ViewSets, and the use of routers to streamline the process.

**Acronyms**

* **DRF:** Django Rest Framework
* **API:** Application Programming Interface
* **HTTP:** Hypertext Transfer Protocol

**Technical Words**

* **Routing:** The process of mapping URL patterns to specific functions or views within an application.
* **ViewSets:** DRF classes providing actions for common CRUD operations (Create, Read, Update, Delete).
* **Router:** A DRF component that automatically generates URL patterns based on registered ViewSets.

**Commands/Modules**

* **path()**: Django function for defining URL patterns (from django.urls import path)
* **APIView:** Base DRF class for creating API views.
* **ViewSets:** DRF class for defining common RESTful actions.
* **SimpleRouter, DefaultRouter:** DRF classes for automatically creating URL patterns (from rest\_framework.routers import SimpleRouter, DefaultRouter)

keywords = ["Django", "DRF", "API", "routing", "view", "ViewSet", "router", "HTTP"]

**Tips**

* Use ViewSets and Routers for cleaner routing configurations in larger API projects.

Written by Gemini, your API navigation guide 🗺️

I was not clear why default and simple router was any better than the standard way. As you can see below they both look more complicated and take longer to write.

**Scenario 2: Routing with SimpleRouter**urlpatterns = [

from django.urls import path

from .views import BookViewSet, UserViewSet

urlpatterns = [

path('books/', BookViewSet.as\_view({'get': 'list', 'post': 'create'}), name='book-list'),

path('books/<int:pk>/', BookViewSet.as\_view({'get': 'retrieve', 'put': 'update', 'delete': 'destroy'}), name='book-detail'),

path('users/', UserViewSet.as\_view({'get': 'list', 'post': 'create'}), name='user-list'),

path('users/<int:pk>/', UserViewSet.as\_view({'get': 'retrieve', 'put': 'update', 'delete': 'destroy'}), name='user-detail'),

]**Observations:**

* **Verbose:** Requires manual specification of URLs and HTTP methods for each ViewSet action.
* **Repetitive:** Can lead to code duplication and less maintainability with a growing API.

**Scenario 2: Routing with SimpleRouter**

from rest\_framework.routers import SimpleRouter

from .views import BookViewSet, UserViewSet

router = SimpleRouter()

router.register('books', BookViewSet, basename='books')

router.register('users', UserViewSet, basename='users')

urlpatterns = router.urls

**Observations**

* **Concise:** Automatically generates URLs based on ViewSet actions.
* **Cleaner:** Reduces boilerplate code and improves readability.

**Scenario 3: Routing with DefaultRouter**

from rest\_framework.routers import DefaultRouter

from .views import BookViewSet, UserViewSet

router = DefaultRouter()

router.register('books', BookViewSet, basename='books')

router.register('users', UserViewSet, basename='users')

urlpatterns = router.urls

**Observations**

* **API Root:** Provides a discoverable root endpoint (e.g., /api/) that lists all available endpoints.
* **Consistency:** Optionally enforces trailing slashes in URLs if desired.

**Key Points**

* **Both routers** streamline URL configuration and promote consistent URL structures.
* **DefaultRouter** provides the added benefit of an API root for discoverability.

1. **Navigate to http://127.0.0.1:8000/api/ in your web browser.**

**Example Output:**

JSON

{

"books": "http://127.0.0.1:8000/api/books/",

"authors": "http://127.0.0.1:8000/api/authors/"

}

**Observations**

* The API Root lists the available endpoints for both the books and authors resources.
* Each endpoint link provides a starting point to interact with that specific resource in your API.

**Important Notes:**

* Only DefaultRouter includes the API Root functionality automatically. SimpleRouter focuses solely on the core endpoint generation.
* The API Root is a convenient tool for development and exploration, but you might consider customizing its output or even disabling it for security reasons in certain production environments.

Written by Gemini, your API organization expert 🗺️

[Generic views and ViewSets in DRF | Coursera](https://www.coursera.org/learn/apis/supplement/AlxEs/generic-views-and-viewsets-in-drf) 4-7-2024 0555

**Short Summary**

This reading introduces generic views and ViewSets in Django Rest Framework. It explains how these components streamline API development by providing pre-built functionality for common CRUD operations, authentication, and filtering.

**Acronyms**

* **API:** Application Programming Interface
* **CRUD:** Create, Read, Update, Delete
* **HTTP:** Hypertext Transfer Protocol

**Technical Words**

* **ViewSet:** A DRF class providing a set of actions for common CRUD operations.
* **ModelViewSet:** A ViewSet automatically handling database interactions for a model.
* **ReadOnlyModelViewSet:** A ViewSet supporting only read operations.
* **Generic Views:** DRF classes offering pre-built views for specific tasks (e.g., listing, creating resources).
* **Queryset:** A Django object representing a collection of database records.
* **Serializer:** A DRF component for translating data between Python objects and formats like JSON.
* **Authentication:** Verifying a user's identity.

**Commands/Modules**

* **viewsets** (from rest\_framework import viewsets)
* **generics** (from rest\_framework import generics)

keywords = ["Django", "DRF", "API", "CRUD", "ViewSet", "ModelViewSet", "generic views", "serializer", "authentication"]

**Mistakes and Misleading Information**

* **Selective Authentication Description:** The provided example on selective authentication might need more clarity on the role of permission\_classes.

Written by Gemini, your API architect 🏛️

[Function and class-based views | Coursera](https://www.coursera.org/learn/apis/lecture/4BASB/function-and-class-based-views) 4-7-2024 0630

**Short Summary**

This video introduces class-based views in DRF, comparing them to function-based views. It demonstrates how class-based views can streamline API development, handle HTTP methods, and work with query parameters and payloads.

**Acronyms**

* **DRF:** Django Rest Framework
* **API:** Application Programming Interface
* **HTTP:** Hypertext Transfer Protocol
* **JSON:** JavaScript Object Notation

**Technical Words**

* **Class-based View:** A DRF view implemented as a class, with methods corresponding to HTTP verbs (GET, POST, etc.).
* **APIView:** The base class in DRF for creating class-based views.
* **Payload:** Data sent in the body of an API request (often JSON).
* **Query Parameter:** Key-value pairs in a URL used to filter or modify API responses.

**Commands/Modules**

* **APIView** (from rest\_framework.views import APIView)
* **Response** (from rest\_framework.response import Response)

keywords = ["Django", "DRF", "API", "class-based view", "APIView", "HTTP", "GET", "POST", "payload", "query parameters"]

**Mistakes and Misleading Information**

* **Slight Omission:** The video doesn't explicitly mention the need to map class-based views in the urls.py file, which is essential for their functionality.

Written by Gemini, your API design guide 🏛️

[Django debug toolbar | Coursera](https://www.coursera.org/learn/apis/lecture/BJPag/django-debug-toolbar) 4-7-2024 0730

**Short Summary**

This video introduces the Django Debug Toolbar as a valuable debugging and optimization tool for Django projects. It covers the installation, configuration, and exploration of key sections within the toolbar.

**Acronyms**

* **API:** Application Programming Interface

**Technical Words**

* **Middleware:** Django components that process requests and responses.
* **Static Files:** CSS, JavaScript, and images used by a web application.
* **Cache:** A temporary storage mechanism for frequently accessed data.
* **Profiling:** Analyzing code to measure performance and identify bottlenecks.

**Software**

* **Django Debug Toolbar:** A Django add-on for debugging.

**Commands/Modules**

* **pipenv install django-debug-toolbar**

keywords = ["Django", "debugging", "Django Debug Toolbar", "middleware", "SQL", "profiling", "cache", "optimization"]

**Mistakes and Misleading Information**

* **API Focus:** The video states that the toolbar mainly works with "browser API" calls. It's important to clarify that the Django Debug Toolbar works with any Django view, whether rendered in the browser or returning data for API consumption.

**Tips**

* Use the Django Debug Toolbar in your development environment to get insights into your Django project's inner workings.

Written by Gemini, your debugging detective 🕵️‍♀️

[Restaurant menu API project with DRF | Coursera](https://www.coursera.org/learn/apis/lecture/r0j4h/restaurant-menu-api-project-with-drf) 4-7-2024 0800

**Short Summary**

This video demonstrates how to quickly build a functional CRUD (Create, Read, Update, Delete) API within a Django project using the Django Rest Framework's (DRF) generic views. It highlights the efficiency of leveraging these pre-built classes to simplify API development.

**Acronyms**

* **API:** Application Programming Interface
* **CRUD:** Create, Read, Update, Delete

**Technical Words**

* **Model:** A Django class representing a database table.
* **Serializer:** A DRF component for translating data between Python objects and formats like JSON.
* **Generic Views:** Pre-built classes in DRF providing common CRUD functionalities.
* **Queryset:** A Django object representing a collection of database records.

**Software**

* **Django Rest Framework**

**Commands/Modules**

* **generics** (from rest\_framework import generics)
* **ListCreateAPIView**, **RetrieveUpdateDestroyAPIView**

keywords = ["Django", "DRF", "API", "CRUD", "generic views", "serializer", "queryset"]

**Mistakes and Misleading Information**

* **Slight Oversimplification:** While generic views are powerful, it's important to remember that they might not be suitable for every API endpoint, especially when dealing with complex logic or custom behaviors.

**Tips**

* Use generic views as a starting point for basic CRUD operations.
* For more customization, consider extending generic views or creating custom class-based views.

Written by Gemini, your API architect 🏛️

Note: This went well. I think I am getting it.

[Convert booklist API project to DRF | Coursera](https://www.coursera.org/learn/apis/ungradedLab/Zun7V/convert-booklist-api-project-to-drf)4-7-2024 0830

* Nothing new, but a good exercise.

[Knowledge check: Introduction to DRF | Coursera](https://www.coursera.org/learn/apis/quiz/Q1Gld/knowledge-check-introduction-to-drf/attempt) 4-7-2024 1245

* simple

[Serializers | Coursera](https://www.coursera.org/learn/apis/lecture/PW4u4/serializers) 4-7-2024 1300, 4-8-2024 0515, 4-8-2024 1039(Because I did not update version control)

Summary: This video teaches how to use serializers in the Django Rest Framework (DRF) to convert database records into JSON format. It emphasizes the importance of serializers for controlling data exposure and streamlining API development.

* **Technical Words**
  + Serialization - The process of converting data structures or objects into a format suitable for storage or transmission (like JSON or XML).
  + Deserialization - The reverse process of converting serialized data back into native objects.
  + Model - A class in Django that represents a database table, defining its fields and relationships.
  + Queryset - A collection of database query results in Django.
* **Rarely Used Words**
  + Endpoint - A specific URL within an API that allows access to a particular resource or function.
  + Integrity - Ensuring data is accurate, consistent, and free from errors
* **Software Packages**
  + Django Rest Framework (DRF) - A toolkit built on top of Django, designed specifically for creating RESTful web APIs.
* **Commands/Modules**
  + get\_object\_or\_404 - Django shortcut function to retrieve an object by ID or raise a "Http404" exception (Not Found) if it doesn't exist.
* **Ideas/Techniques**
  + Model Serializers - A DRF feature that provides an automatic, efficient way to map Django models to serializers based on model fields.
* **Tips**
  + Use many=True argument when serializing a queryset (a list of objects)
  + Customize serializer fields to control which fields are included in the output.
  + Employ get\_object\_or\_404 for cleaner error handling.
* keywords = ['serializers', 'Django', 'models', 'DRF', 'API', 'JSON', 'data conversion', 'serialization', 'deserialization', 'validation', 'queryset', 'many=True']
* **Mistakes or Misleading Information**
  + The video suggests that DRF automatically handles the conversion of querysets into JSON. While DRF simplifies this, you still need to explicitly use serializers for the conversion.

Written by Gemini, a sometimes sassy but always helpful LLM

Notes:

I through the serializers, model and relational serializers and still felt I had not mastered it. I erased my progress and started from this module again. One of the things I missed the first time around was adding models to the admin site so I could add records to category.

[Model serializers | Coursera](https://www.coursera.org/learn/apis/lecture/wKFBS/model-serializers) 4-7-2024 1425, 4-8-2024 0614

Summary: This video delves into using model serializers in the Django Rest Framework (DRF) to streamline the conversion of Django models into JSON data. It demonstrates how to customize field names, use calculated fields, and efficiently represent model relationships with model serializers.

**Detailed Breakdown:**

* **Acronyms**
  + DRF - Django Rest Framework
  + JSON - JavaScript Object Notation
* **Technical Words**
  + Model Serializer - A specialized type of serializer in DRF that automatically creates serializers based on existing Django models.
  + Instance - A single object of a particular class.
  + Relationship - Connections or associations between different models in a Django database.
* **Ideas/Techniques**
  + Model Serialization Efficiency - Employing model serializers in DRF offers a time-saving way to convert Django models into JSON data.
  + Field Customization - Model serializers allow you to rename fields in the API output for clarity or consistency.
  + Calculated Fields - Model serializers support the inclusion of fields computed dynamically based on model data.
* **Tips**
  + Use the source argument to map serializer fields to existing model fields with different names.
  + Always include new serializer fields within the Meta class's fields list.
* keywords = ['model serializers', 'Django', 'DRF', 'JSON', 'API', 'models', 'serialization', 'relationships', 'calculated fields']
* **Mistakes or Misleading Information**
  + While model serializers significantly simplify the process, it's important to note they don't fully automate the conversion of Django models to JSON. You still need to define the model serializer class.

Written by Gemini, a sometimes sassy but always helpful LLM

[Relationship serializers | Coursera](https://www.coursera.org/learn/apis/lecture/oCEa9/relationship-serializers) 4-8-2024 1218

**Short summary** Summary: This video teaches how to use relationship serializers in Django Rest Framework (DRF) to display connected model data efficiently within JSON responses. It also covers optimizing views to load related data with fewer database queries.

**Technical Terms**

* **API:** Application Programming Interface (A way for software components to interact)
* **Endpoint:** A URL location where an API can be accessed
* **Framework** A structured foundation for building software applications
* **JSON:** JavaScript Object Notation (A data format for exchanging information)
* **Models:** Representations of database tables and their relationships in Django
* **REST:** Representational State Transfer (An architectural style for web APIs)
* **Serializer:** Converts Django model data into formats like JSON for API responses
* **SQL:** Structured Query Language (Used to interact with databases)

**Rarely Used Words**

* **propagate:** To spread or transmit (used in the context of deleting database records)

**Software Packages**

* **Django Rest Framework (DRF):** A toolkit for building APIs with Django

**Commands**

* **pipenv install:** Used to install packages in a Pipenv virtual environment.
* **migrate:** Django command to apply database schema changes

**Modules**

* **models.py:** Django file where database models are defined
* **serializers.py:** Django file where serializers are defined

**Ideas**

* **Relationship Serializers:** A type of DRF serializer for handling nested model data.
* **Foreign Keys:** Fields in a database table that link to another table's primary key creating a relationship.
* **Database Optimization:** Techniques to improve API performance by reducing database queries.

**Techniques**

* **Representing related models in API responses:** Using relationship serializers to include nested data
* **Eager loading of related data:** Optimizing view code to fetch related data in a single query

**keywords** =["Django", "DRF", "serializers", "models", "relationships", "JSON", "API", "endpoints", "views", "optimization"]

**This analysis was brought to you by Gemini, your friendly neighborhood language model!** 🤖

[Deserialization and validation | Coursera](https://www.coursera.org/learn/apis/lecture/qGJQI/deserialization-and-validation) 4-8-2024 1348

**Short summary** Summary: This video teaches how to use Django Rest Framework (DRF) to deserialize data sent to API endpoints, validate the data, and store it in a database. It also covers making fields write-only for specific request types.

**Technical Terms**

* **API:** Application Programming Interface
* **Endpoint:** A specific URL where an API can be accessed
* **Deserializer:** Converts data formats (like JSON) into Python objects
* **Framework** A structured foundation for building software applications
* **HTTP Request:** Message sent from a client (like a web browser) to a server
* **JSON:** JavaScript Object Notation (data format)
* **Model:** Representation of a database table in Django
* **Payload:** The data within an HTTP request or response
* **REST:** Representational State Transfer (architectural style for web APIs)
* **Serializer:** Converts Django models into formats like JSON and vice-versa
* **Validate:** To check if data conforms to rules

**Rarely Used Words**

* **deserialize:** To convert data from a serialized format back into original objects.

**Modules**

* **serializers.py:** Django file for defining serializers

**Ideas**

* **Deserialization:** Converting incoming data into model instances
* **Data Validation:** Verifying data against expected rules
* **Write-only Fields:** Fields that are sent for creation/update but hidden in responses

**Techniques**

* **Deserializing and saving data:** Using DRF serializers to handle incoming data.
* **Validating incoming data:** Using serializer's is\_valid method.
* **Controlling field visibility:** Using the read\_only or write\_only in serializers.

**keywords=** ["Django", "DRF", "API", "endpoints", "deserialization", "validation", "models", "HTTP", "JSON", "payload", "views", "serializers"]

**This breakdown was created by Gemini, your trusty language model assistant! 🤖**

[Renderers | Coursera](https://www.coursera.org/learn/apis/lecture/PyR23/renderers) 4-8-2024 1632

Short summary: This video explores different renderers in Django Rest Framework (DRF) to support diverse output formats like JSON, HTML, and XML. It explains how to control renderers for specific content types using the 'Accept' header.

**Technical Words**

* **Endpoint:** A specific URL location where an API can be accessed
* **HTTP Request:** Message sent from a client (like a web browser) to a server
* **JSON:** JavaScript Object Notation (data format)
* **Renderer:** DRF component responsible for transforming data into different output formats.
* **XML:** Extensible Markup Language (data format)

**Rarely Used Words**

* **Extensible:** Capable of being extended or expanded.

**Software Packages**

* **Django Rest Framework (DRF):** Toolkit for building APIs with Django
* **Django REST Framework XML:** A package extending DRF to support XML output

**Commands/Modules**

* **pipenv install** : Used to install packages in a Pipenv virtual environment. Example: pipenv install djangorestframework-xml

**Ideas/Techniques**

* **Content Negotiation:** The process of determining the best format for API responses based on the client's 'Accept' header.
* **Browsable API Renderer:** A built-in DRF renderer for a user-friendly, interactive API interface.

**Tips**

* Use the 'Accept' header in HTTP requests to specify the desired output format from the API.

keywords = ["Django", "DRF", "API", "renderers", "JSON", "XML", "HTTP", "Accept", "content negotiation"]

**Mistakes or Misleading Information**

* The transcript mentions YAML support. While a YAML renderer likely exists, it's not a built-in feature of DRF.

**Written by Gemini**

[Module quiz: Django REST framework | Coursera](https://www.coursera.org/learn/apis/exam/hyO3l/module-quiz-django-rest-framework) 4-9-2024 0400

80%, 90% and 100% on 3rd try

Week 3

[Filtering and searching | Coursera](https://www.coursera.org/learn/apis/lecture/h7QUx/filtering-and-searching) 4-9-2024 0420

**Short Summary:** This video teaches how to add search and filtering capabilities to Django REST Framework (DRF) APIs. It demonstrates filtering menu items by category, price, and performing case-insensitive searches on menu item names.

**Detailed Breakdown:**

* **Acronyms**
  + URL (Uniform Resource Locator)
* **Technical Words**
  + **Endpoint:** A specific point within an API where data exchange occurs.
  + **Query string:** Parameters within a URL, started with "?", used for filtering or searching.
  + **Client application:** A program that interacts with an API (ex. web browser, mobile app).
  + **Serialization:** Translating data between Python objects and formats like JSON.
  + **Field lookup:** Operators in DRF to specify filtering based on field values (e.g., "startswith").
* **Commands/Modules**
  + **.filter()**: Django method to refine query results. Sample use: items.filter(price\_\_lte=10)
* **Ideas/Techniques**
  + **API Filtering:** Refining API results to match specific search parameters.
  + **Case-insensitive Searching:** Searching without regard to letter case.
* **Tips**
  + Optimize APIs by filtering data on the server side to avoid unnecessary load.
* keywords = ['request', 'query', 'params', 'get', 'if', 'else', 'title', 'lte', 'contains', 'icontains', 'startswith', 'istartswith']
* **Mistakes or Misleading Information**
  + While generally accurate, the video could emphasize that client applications can *optionally* use filtering; it's not strictly mandatory.

Written by Gemini, a witty and sometimes sarcastic AI assistant

Note: At some point I named my category titles differently than the course causing issues when following the course. Used admin to rename.

Lots of good stuff in this video for my personal project.

[Ordering | Coursera](https://www.coursera.org/learn/apis/lecture/gza5P/ordering) 4-9-2024 0449

Short Summary: This video demonstrates how to implement result sorting in Django REST Framework (DRF) APIs. It covers sorting by single and multiple fields, controlling ascending/descending order, and briefly mentions the django-filters package.

Detailed Breakdown:

* Technical Words
  + **Query string:** Parameters within a URL, started with "?", used for filtering or searching.
  + **django-filters:** A package providing advanced filtering functionality for Django.
* Rarely Used Words
  + **Query set:** A collection of database results in Django.
* Commands/Modules
  + **.order\_by()**: Django method for sorting query sets. Sample use: items.order\_by('price', '-inventory')
* Ideas/Techniques
  + **API Sorting:** The process of ordering API results based on specified criteria.
  + **Multiple Field Sorting:** Sorting results by more than one field simultaneously.
* keywords = ['request', 'query', 'params', 'get', 'split', 'order\_by']
* Mistakes or Misleading Information
  + The statement about django-filters being primarily for class-based views is not strictly true. While it integrates easily with class-based views, django-filters can be used with function-based views as well.

Written by Gemini, a witty and sometimes sarcastic AI assistant

[Importance of data validation | Coursera](https://www.coursera.org/learn/apis/supplement/ISsUx/importance-of-data-validation) 4-9-2024 0845

Short Summary: This reading explains the significance of data validation in web applications and demonstrates various validation techniques within the Django REST Framework (DRF). It covers field-level validation, unique validation, and different methods for implementing these validations.

Detailed Breakdown:

* Technical Words
  + **Validation:** Ensuring the correctness, completeness, and security of user-submitted data.
  + **Serializer:** In DRF, serializers convert data between Python objects and formats like JSON.
  + **Field:** Individual elements within a model (e.g., 'title', 'price').
* Commands/Modules
  + **.validate()**: DRF serializer method for validating multiple fields. Sample use: serializer.validate(data)
* Ideas/Techniques
  + **Field-level Validation:** Validating individual data fields (e.g., ensuring price is greater than 2).
  + **Unique Validation:** Enforcing uniqueness of single fields or field combinations to prevent duplicates.
* keywords = ['serializers', 'Decimal','ValidationError']
* Mistakes or Misleading Information
  + While the reading offers valuable insights, it could benefit from these clarifications:
    - **ValidationError:** It's important to note that serializers.ValidationError needs to be imported.
    - **Method Placement:** Validation methods like validate\_price() should be placed *within* the serializer class, not outside.

Written by Gemini, a witty and sometimes sarcastic AI assistant

Note:

In method 2 when adding min stock you need to remove “stock = serializers.IntegerField(source='inventory')” and add it back when doing method 3.

[Data sanitization | Coursera](https://www.coursera.org/learn/apis/supplement/52kdB/data-sanitization) 4-9-2024 1030

Short summary: This reading explains data sanitization in Django Rest Framework (DRF) to protect against security threats like script injection and SQL injection. It provides instructions on using the 'bleach' library for HTML/JavaScript sanitization and parameterized queries to prevent SQL injection.

**Technical Words**

* **Data Sanitization:** The process of cleaning data to remove potential security threats.
* **Data Validation:** Checking if data conforms to expected rules.
* **SQL injection:** A security exploit where malicious SQL code is injected into input fields to manipulate a database.

**Software Packages**

* **Django Rest Framework (DRF):** Toolkit for building APIs with Django.
* **bleach:** A Python package for sanitizing HTML.

**Commands/Modules**

* **MenuItem.objects.raw()** : Django ORM method to execute raw SQL queries. Example: MenuItem.objects.raw('SELECT \* FROM myapp\_menuitem LIMIT %s', [10])

**Ideas/Techniques**

* **HTML/JavaScript Sanitization:** Preventing script injection attacks by cleaning HTML and JavaScript from user input.
* **Parameterized Queries:** A method to prevent SQL injection by using placeholders for dynamic values in SQL queries.

**Tips**

* Sanitize user input to prevent script injection and SQL injection attacks.
* Use the 'bleach' library to sanitize HTML/JavaScript content.
* Always use parameterized queries in raw SQL to prevent SQL injection.

keywords = ["Django", "DRF", "API", "data sanitization", "SQL injection", "script injection", "HTML", "JavaScript", "bleach", "security"]

**Mistakes or Misleading Information**

* The reading suggests that raw SQL queries are sometimes necessary. While there might be rare cases, it's generally best to avoid raw SQL if possible and use Django's ORM for database interactions to minimize security risks.

**Written by Gemini**

[Pagination | Coursera](https://www.coursera.org/learn/apis/lecture/mEYFj/pagination) 4-9-2024 1200

Short summary: This video explains the importance of pagination in APIs to improve performance and teaches how to implement it in a Django Rest Framework (DRF) project using Django's built-in paginator module.

**Technical Words**

* **API Endpoint:** A specific URL where an API can be accessed.
* **Client:** A software component that makes requests to an API.
* **HTTP Status Code:** A code indicating the outcome of an HTTP request (e.g., 200 for success, 400 for bad request).
* **Pagination:** The practice of dividing results into manageable pages.
* **Query String:** Parameters added to a URL to send information to the server.
* **Server:** A computer or software that provides data or services to other devices.

**Software Packages**

* **Django Rest Framework (DRF):** Toolkit for building APIs with Django.

**Modules**

* **django.core.paginator:** Django module providing pagination functionality.

**Ideas**

* **API Performance Optimization:** Techniques to improve the speed and efficiency of APIs.

**Techniques**

* **Implementing Pagination:** Using Django's 'Paginator' class to divide API responses into pages.
* **Handling Invalid Page Requests:** Gracefully managing requests for non-existent pages.

keywords = ["Django", "DRF", "API", "pagination", "query string", "server", "client"]

**Written by Gemini, your trusty language model assistant! 🤖**

[More on filtering and pagination | Coursera](https://www.coursera.org/learn/apis/supplement/oCL3M/more-on-filtering-and-pagination) 4-9-2024 1318

Short summary: This reading teaches how to easily implement filtering, searching, and pagination functionalities in Django Rest Framework (DRF) class-based views using built-in DRF classes.

**Acronyms**

* API: Application Programming Interface
* CRUD: Create, Read, Update, Delete (basic database operations)

**Technical Words**

* **API Endpoint:** A specific URL where an API can be accessed.
* **Class-Based View:** A way to structure API views in Django using classes, offering reusability and organization.
* **Filtering:** Selecting a subset of data based on criteria.
* **Lookup Field:** The type of search comparison used (e.g., 'icontains' for case-insensitive substring matching).
* **ModelViewSet:** A DRF class providing common CRUD actions for a model.
* **Nested Field:** A field in a model referencing another model.
* **Ordering/Sorting:** Arranging data in a specific sequence.
* **Pagination:** Dividing results into manageable pages.
* **Query String:** Parameters added to a URL to send information to the server.
* **Serializer:** Converts Django models into formats like JSON.

**Rarely Used Words**

* **Scaffolding:** Setting up a basic project structure.

**Modules**

* **rest\_framework.viewsets:** Provides the ModelViewSet class in DRF.
* **django\_filters.rest\_framework:** Provides filtering functionality in DRF.

**Ideas**

* **API Filtering:** Allowing clients to request specific subsets of data.
* **API Searching:** Providing a way for clients to find data based on keywords.

**Techniques**

* **Using DRF's OrderingFilter:** Implementing API result sorting.
* **Using DRF's SearchFilter:** Enabling search functionality in API results.
* **Using DRF's Pagination Classes:** Dividing API responses into pages.
* **Searching Nested Fields :** Filtering data based on fields in related models

keywords = ["Django", "DRF", "API", "filtering", "pagination", "searching", "viewsets", "models", "serializers"]

**Written by Gemini, your friendly neighborhood language model! 🤖**

[Caching | Coursera](https://www.coursera.org/learn/apis/lecture/Dd5C1/caching) 4-9-2024 1350

Short summary: This video explains caching as a technique to improve web application performance. It discusses different caching layers (client, reverse proxy, web server, database) and their roles in reducing server load and speeding up responses.

**Acronyms**

* REST: Representational State Transfer

**Technical Words**

* **API Infrastructure:** The hardware, software, and network components supporting an API.
* **Bandwidth:** The amount of data that can be transferred over a network connection in a given time.
* **Cache/Caching:** Storing frequently accessed data in a temporary location for faster retrieval.
* **Client:** A software or device that makes requests to a server.
* **Database:** An organized collection of data.
* **Firewall:** A network security system that controls incoming and outgoing traffic.
* **Query:** A request for data from a database.
* **Reverse Proxy** A server that sits in front of web servers, routing traffic and providing an additional layer of security.
* **Server:** A computer or software that provides data or services to other devices.
* **Storage:** Refers to devices or technologies used to store data persistently.
* **Traffic:** The flow of data over a network.

**Software Packages**

* **Redis:** An in-memory data store often used for caching.
* **Memcached:** Another popular in-memory caching system.

**Ideas**

* **Layered Architecture** A design principle where software is divided into self-contained components.
* **Performance Optimization:** Techniques to improve application speed and responsiveness.

**Techniques**

* **Database Caching:** Storing query results in memory to avoid repeated database access.
* **Web Server Caching:** Storing generated responses in memory or external storage (like Redis) to avoid redundant processing.
* **Reverse Proxy Caching:** Having a reverse proxy store responses based on cache headers, reducing load on web servers.
* **Client-Side Caching:** Instructing browsers or applications to store responses locally for a specified duration.

keywords = ["caching", "performance", "web server", "database", "reverse proxy", "client", "API", "HTTP"]

**Written by Gemini, your language model for all things tech! 🤖**

[Exercise: Restaurant menu API - filtering, ordering and searching | Coursera](https://www.coursera.org/learn/apis/ungradedLab/6qoGW/exercise-restaurant-menu-api-filtering-ordering-and-searching) 4-9-2024 1420

* Lab was already completed.

[Knowledge check: Essential API tasks | Coursera](https://www.coursera.org/learn/apis/quiz/9zktc/knowledge-check-essential-api-tasks) 4-9-2024 1424

* Pretty easy

[Token-based authentication in DRF | Coursera](https://www.coursera.org/learn/apis/lecture/MJTLM/token-based-authentication-in-drf) 4-9-2024 1614

Short summary: This video explains authentication in Django Rest Framework (DRF), focusing on token-based authentication to protect API endpoints. It covers how to generate tokens and use them to access secured resources.

**Technical Words**

* **API Endpoint:** A specific URL where an API can be accessed.
* **Authentication:** Verifying the identity of a user or client.
* **Authorization:** Determining what permissions a verified user has.
* **Credentials:** Information used for authentication, usually a username and password.
* **Token:** A unique string used in token-based authentication to represent a user's session.

**Commands**

* **python manage.py createsuperuser:** Django command to create an administrative user.
* **pipenv shell:** Activates a Pipenv virtual environment

**Modules**

* **rest\_framework.authtoken:** DRF module providing token-based authentication.
* **rest\_framework.authtoken.views:** Provides the obtain\_auth\_token view for generating tokens.

**Ideas**

* **API Security:** Protecting APIs from unauthorized access.
* **Password-Based Authentication:** Authenticating with a username and password.
* **Token-Based Authentication:** Authenticating using a unique token sent with API requests.

**Techniques**

* **Protecting API endpoints:** Using DRF's permission classes to restrict access.
* **Generating authentication tokens:** Using obtain\_auth\_token for token creation.
* **Using tokens for authentication:** Including the token in API request headers.

keywords = ["Django", "DRF", "API", "authentication", "token", "security", "credentials", "endpoints"]

**Written by Gemini, your AI assistant for understanding APIs! 🤖**

[User roles | Coursera](https://www.coursera.org/learn/apis/lecture/12F4H/user-roles) 4-10-2024 0512

Short summary: This video explains how to implement authorization in Django Rest Framework (DRF) using Django's built-in user groups. It teaches how to restrict API access based on user roles to protect sensitive data.

**Technical Words**

* **API Endpoint:** A specific URL where an API can be accessed.
* **Authentication:** Verifying the identity of a user or client.
* **Authorization:** Determining what permissions a verified user has.
* **Group:** A way to organize users with similar permissions in Django.
* **HTTP Status Code:** A code indicating the outcome of an HTTP request (e.g., 403 for Forbidden).

**Ideas**

* **API Authorization:** Controlling access to API endpoints based on user permissions.
* **Role-Based Access Control (RBAC):** An authorization model where permissions are assigned to roles, and users are assigned roles.

**Techniques**

* **Creating User Groups:** Using the Django admin to define groups (e.g., "manager").
* **Assigning Users to Groups:** Adding users to groups in the Django admin.
* **Checking Group Membership:** Using user.groups.filter(name='manager').exists() to verify if a user belongs to a specific group.

keywords = ["Django", "DRF", "API", "authentication", "authorization", "groups", "roles", "permissions"]

**Written by Gemini, your language model for explaining technical concepts! 🤖**

**Note: Biggest issue was not checking the spelling in the settings.py file. Also capitalization.**

[Setting up API throttling | Coursera](https://www.coursera.org/learn/apis/lecture/rPE4B/setting-up-api-throttling)

**Short Summary:** This video tutorial explains how to use Django REST Framework's built-in throttling mechanisms to protect API endpoints from abuse. It covers rate limiting for anonymous and authenticated users, as well as custom throttling rules.

**Detailed Breakdown:**

* **Technical Words**
  + Throttling: Intentionally limiting the rate of requests to manage resource usage.
  + Token: A piece of data used for authentication.
  + Endpoint: A specific URL within an API that allows interaction.
  + Rate Limiting: Synonym for throttling.
* **Rarely Used Words**
  + Cluster (in this context): A group of related settings or rules within the framework.
  + Anonymous (in this context): Refers to users who have not logged in or authenticated.
* **Commands/Modules**
  + AnonRateThrottle: DRF module to throttle anonymous requests (from rest\_framework.throttling import AnonRateThrottle)
  + UserRateThrottle: DRF module to throttle authenticated user requests (from rest\_framework.throttling import UserRateThrottle)
* **Ideas/Techniques**
  + API Throttling: The concept of limiting requests to an API to prevent abuse or excessive resource usage.
  + Scope-based Throttling: Applying different throttling rates based on custom categories or 'scopes' defined within the configuration.
* **Tips**
  + Choose appropriate throttling rates based on your API's expected usage patterns and resource availability.
  + Use authentication to enforce different throttling limits for logged-in users.
  + Consider custom scopes for fine-grained control over throttling behavior.
* keywords = ['AnonRateThrottle', 'UserRateThrottle', 'throttle\_classes', 'scope', 'API', 'token', 'authenticate', 'endpoint', 'throttle', 'rate-limit']

**Mistakes or Misleading Information**

* The transcript suggests throttling rates must be expressed with units like "/minute" or "/day". While convenient, DRF also allows numeric rates (e.g., 10), which would be interpreted as "10 per second".

Written by Gemini, you can rely on me to provide clear and concise summaries of tech content

[API throttling for class-based views | Coursera](https://www.coursera.org/learn/apis/supplement/1h6WO/api-throttling-for-class-based-views) 4-10-2024 0623

Short Summary: This reading covers the implementation of API throttling within class-based views using the Django REST Framework. It discusses how to set up throttling, apply conditional throttling based on HTTP methods, and utilize custom throttling rules.

Detailed Breakdown:

* **Acronyms**
  + API: Application Programming Interface
  + CRUD: Create, Read, Update, Delete
  + DRF: Django REST Framework
* **Technical Words**
  + Throttling: Controlling the rate of requests to manage resource usage.
  + Rate Limiting: Synonym for throttling.
  + Endpoint: A specific URL within an API that allows interaction.
  + Class-based View: A style of defining views in Django that groups related logic into a class.
  + Conditional Throttling: Applying throttling selectively based on factors like HTTP method.
* **Software Packages**
  + Django REST Framework (DRF): A toolkit for building RESTful APIs within the Django web framework.
* **Commands/Modules**
  + ModelViewSet: A class in DRF that combines common logic for handling multiple model actions (list, create, update, etc.) (from rest\_framework import viewsets)
  + AnonRateThrottle: A DRF class for throttling anonymous requests (from rest\_framework.throttling import AnonRateThrottle)
  + UserRateThrottle: A DRF class for throttling requests from authenticated users (from rest\_framework.throttling import UserRateThrottle)
* **Ideas/Techniques**
  + API Throttling: The concept of limiting requests to an API to prevent abuse or excessive resource usage.
* **Tips**
  + Customize throttling rates for different user types (anonymous vs. authenticated).
  + Use conditional throttling to apply limits selectively to certain HTTP methods.
  + Create custom throttling classes for fine-grained control.
* **Keywords** keywords = ['viewsets', 'ModelViewSet', 'throttling', 'AnonRateThrottle', 'UserRateThrottle', 'GET', 'POST', 'create', 'list', 'retrieve']
* **Mistakes or Misleading Information**
  + **Imprecision:** The reading uses the terms "throttling" and "rate-limiting" interchangeably. While related, they can have subtle technical distinctions.

Written by Gemini, you can rely on me to provide clear and concise summaries of tech content

[Introduction to Djoser library for better authentication | Coursera](https://www.coursera.org/learn/apis/lecture/bldmJ/introduction-to-djoser-library-for-better-authentication) 4-10-2024 1030

**Summary: This video teaches how to use Djoser, a package that simplifies user authentication in Django REST Framework projects. Djoser provides endpoints for user creation, login, and token-based authentication.**

**Short summary: This video explains how to use the Djoser package to easily add user authentication features to Django REST Framework applications.**

**Acronyms:**

* **JWT:** JSON Web Token (a standard for securely transmitting information as JSON objects).

**Terms:**

* **Authentication:** Verifying the identity of a user or system.
* **Authorization:** Determining what a verified user is allowed to do.
* **Data breach:** An incident where sensitive information is exposed without permission.
* **REST API:** An API that follows the principles of the REST architectural style, using HTTP methods for data manipulation.

**Technical Words:**

* **Framework:** A basic structure of abstractions that provides a foundation and common practices for building software.
* **Endpoint:** A specific address (URL) within an API where a client application can access particular resources or perform actions.
* **HTTP Methods:** Verbs like GET, POST, PUT, DELETE that define the type of action to be performed on a resource in a REST API.
* **Virtual environment:** An isolated environment within your project directory to manage dependencies without affecting your system's overall Python installation.

**Software Packages:**

* **Djoser:** A Django REST Framework library specializing in authentication.
* **Insomnia:** A cross-platform API client for testing and prototyping API interactions.

**Commands:**

* **pipenv install Djoser:** Installs the Djoser package.

**Ideas:**

* **Token-based authentication:** A way of authenticating users by issuing them a unique token that they include in subsequent API requests.
* **Browsable API:** A graphical interface provided by Django REST Framework for visualizing and interacting with API endpoints.

**Techniques:**

* **API endpoint configuration:** The process of defining URL patterns and associated methods for API endpoints in Django.

**keywords=** ['Authentication', 'Authorization', 'Django', 'DRF', 'Djoser', 'endpoint', 'token', 'API', 'user']

**This analysis was brought to you by Gemini, your friendly neighborhood language model!**

[Registration and authentication endpoints with JWT | Coursera](https://www.coursera.org/learn/apis/lecture/Vfw28/registration-and-authentication-endpoints-with-jwt) 4-10-2024 1100

**Summary: This video teaches how to use JSON Web Tokens (JWT) for authentication in Django REST Framework (DRF) projects. It covers JWT generation, token refreshing, and how to blacklist tokens for security purposes.**

**Short summary: This video explains how to implement JWT-based authentication in Django REST Framework applications, including handling token expiration and security measures.**

**Acronyms:**

* **JWT:** JSON Web Token (a standard for securely transmitting information as JSON objects).

**Terms:**

* **Authentication:** Verifying the identity of a user or system.
* **Token-based authentication:** A way of authenticating users by issuing them a unique token that they include in subsequent API requests.
* **API Endpoint:** A specific address (URL) within an API where a client application can access particular resources or perform actions.
* **JSON Web Token (JWT):** A compact, self-contained way of representing information about a user or entity that can be used for authentication.
* **Access Token:** A short-lived JWT used for authorizing access to protected API endpoints.
* **Refresh Token:** A longer-lived JWT used to obtain new access tokens without requiring full login credentials.

**Technical Words:**

* **Blacklisting:** In the context of JWT, it means marking a refresh token as invalid so that it can no longer be used to generate new access tokens. This can be used as a security measure.

**Software Packages:**

* **Django REST Framework:** A toolkit built on top of Django, designed for creating REST APIs.
* **djangorestframework-simplejwt:** A package to simplify JWT implementation in DRF projects.
* **Insomnia:** A cross-platform API client for testing and prototyping API interactions.

**Commands:**

* **pipenv install djangorestframework-simplejwt ~= 5.2.1:** Installs the djangorestframework-simplejwt package.
* **pipenv shell:** Activates your project's virtual environment.
* **python manage.py migrate:** Applies database migrations for the Django project.

**Ideas:**

* **JWT Authentication:** A security mechanism using tokens for authorizing access to APIs.
* **Token Expiration:** For security purposes, JWT access tokens have a limited lifespan and need to be refreshed.
* **Token Blacklisting:** A way of revoking a user's access by preventing their refresh token from being used.

**K\keywords=**['authentication', 'JWT', 'JSON', 'token', 'Django', 'DRF', 'API', 'access token', 'refresh token', 'expire', 'blacklist', 'security']

**This analysis was crafted by Gemini, your trusty language model assistant! 🤖**

[User account management | Coursera](https://www.coursera.org/learn/apis/lecture/1rLGu/user-account-management) 4-10-2024 1300

**Summary: This video teaches how to use Djoser for user registration and create a super admin-only API to manage user groups in Django REST Framework.**

**Short summary: This video explains how to implement user registration with Djoser and create a protected API endpoint for super admins to add/remove users from groups.**

**Terms:**

* **Authentication:** Verifying the identity of a user or system.
* **Token-based authentication:** A way of authenticating users by issuing them a unique token that they include in subsequent API requests.
* **API Endpoint:** A specific address (URL) within an API where a client application can access particular resources or perform actions.
* **Super admin:** A user account with elevated privileges and access to all administrative functions of a system.
* **Group:** In the context of Django, a way to categorize users and assign permissions collectively.

**Technical Words:**

* **Framework:** A basic structure of abstractions that provides a foundation and common practices for building software.
* **HTTP Methods:** Verbs like GET, POST, PUT, DELETE that define the type of action to be performed on a resource in a REST API.
* **API view:** In Django REST Framework, a function or class that handles incoming HTTP requests and generates responses.
* **Decorator:** In Python, a way to modify the behavior of a function without changing its code directly.

**Rarely Used Words (for a 12-year-old):**

* **Endpoint:** A specific location within an API where you can access resources or perform actions.

**Software Packages:**

* **Django:** A Python-based web framework for building web applications.
* **Django REST Framework:** A toolkit built on top of Django, designed for creating REST APIs.
* **Djoser:** A Django REST Framework library specializing in authentication.
* **Insomnia:** A cross-platform API client for testing and prototyping API interactions.

**Ideas:**

* **User registration:** The process of creating new user accounts.
* **Authentication endpoints:** API endpoints specifically for handling user login/logout.
* **Authorization endpoints:** API endpoints that require special permissions (e.g., super admin)

**Techniques:**

* **API endpoint configuration:** The process of defining URL patterns and associated methods for API endpoints in Django.
* **Using Djoser for authentication:** How to leverage the Djoser library to simplify user registration.
* **Creating permission-based endpoints:** Protecting API endpoints based on user roles

**Keywords:** ['authentication', 'Django', 'DRF', 'API', 'user', 'registration', 'token', 'super admin', 'group', 'endpoint']

**This analysis was brought to you by Gemini, your AI-powered assistant for understanding code! 🤖**

**Note This module did not work 100% for me. I will come bac to this in a few weeks to retry. Not sure why I could not add and delete like in the video, but to be honest the video did not confirm it worked for them so who knows.**

[Exercise: User account management | Coursera](https://www.coursera.org/learn/apis/supplement/sIhjD/exercise-user-account-management) 4-10-2024 1619

**Short summary: This exercise guides you through securing a Django REST Framework rating system. You'll learn to use Djoser, token authentication, and validation to prevent fake reviews.**

**Terms:**

* **Authentication:** Verifying the identity of a user or system.
* **Token Authentication:** Authenticating users by issuing them a unique token to include in API requests.
* **API Request:** A structured way to interact with a web-based API, sending data and instructions.
* **JSON:** JavaScript Object Notation. A common format for sending and receiving data in web APIs.

**Technical Words:**

* **Framework:** A basic structure of abstractions that provides a foundation and common practices for building software.
* **Form Validation:** Checking if user-submitted data meets specific rules or format requirements.
* **Throttling:** Intentionally limiting the rate at which something can happen, often used to prevent abuse of API endpoints.

**Software Packages:**

* **Django REST Framework:** A toolkit built on top of Django, designed for creating REST APIs.
* **Djoser:** A Django REST Framework library specializing in authentication.
* **Insomnia:** A cross-platform API client for testing and prototyping API interactions.
* **Pipenv:** A tool for managing Python package dependencies and virtual environments.

**Commands:**

* **pipenv shell:** Activates your project's virtual environment
* **pipenv install:** Installs required packages
* **python3 manage.py createsuperuser:** Creates an administrative user for your Django project
* **python3 manage.py makemigrations:** Generates database changes based on model updates
* **python3 manage.py migrate:** Applies pending database changes

**Ideas:**

* **User Authentication:** Verifying the identity of users before they access protected resources.
* **Token-based authentication:** Using tokens to verify user identity in API calls.
* **API Rate Limiting:** Preventing API abuse by restricting request frequency.

**Techniques:**

* **Configuring Django REST Framework:** Setting up authentication classes and permissions.
* **Generating Tokens:** Using Djoser or the authtoken app to create user tokens.
* **Using Insomnia for API Testing:** Creating API requests, setting authentication headers, and sending data.

**keywords =** ['Django', 'DRF', 'API', 'authentication', 'token', 'rating', 'validation', 'throttling', 'user']

**This analysis was brought to you by Gemini, your trusty language model companion! 🤖**

Notes: Am I just missing something or is it the intent of the instructors to leave out important information. The following code was needed to have the records print at the end.  
REST\_FRAMEWORK = {

    'DEFAULT\_RENDERER\_CLASSES': (

        'rest\_framework.renderers.JSONRenderer',

    )

}

[Module quiz: Securing an API in Django REST framework | Coursera](https://www.coursera.org/learn/apis/exam/cAEk2/module-quiz-securing-an-api-in-django-rest-framework)

* This test ate me up. Definitely retake.

Week 4

[Course recap: APIs | Coursera](https://www.coursera.org/learn/apis/lecture/TfCWx/course-recap-apis) 4-11-2024 0408

1. Was the user added to the manager group with an admin token? Yes
2. Is there a list of users from the manager group in the API output? Yes
3. Can the admin add menu items? Yes
4. Can the admin add categories? Yes
5. Is the access token in the API output? Yes
6. Did the value of the featured field update for this particular menu item? Yes

Can the manager assign users to the delivery crew group? Yes. Originally had user in manager group thinking that was good enough, but the user must also be a staff member as it checks for permission\_classes = [IsAdminUser]first.

1. Can the managers assign orders to a delivery crew? Yes, but by pk not name which was weird.
2. Can the delivery crew browse orders that were assigned to them? Yes, they see all orders given to anyone.
3. Can the delivery crew update the order status? Yes, toggled on and off or rather true or false.
4. Can customers register using this endpoint? Yes
5. Is the access token visible in the API output? Yes
6. Are the categories visible to customers? Yes
7. Are the menu items visible to customers? Yes
8. Do the menu items in the category display for customers? Yes
9. Do the menu items display proper pagination for customers? Yes
10. Are the menu items properly sorted by price in ascending or descending order for customers? Yes
11. Can customers add menu items to the cart? Yes
12. Are previous items added to the cart visible for customers? Yes
13. Can customers successfully place an order? No

F

I was unable to get ordering working. This project seemed like a huge jump from super basic to extensive integration of views, urls and models. I’ll come back to it.

Notes not module specific

I had a lingering issue with DRF not loading in pipenv. I used ‘get-command pip’ and learned the virtual environment was not the one I was working in. ‘.virtualenvs\Little-Lemon-Booking-System-mcIiThNE/’ I then did a pip install Django. None of that worked. I then   
**Unset the Virtual Environment**: Since **pipenv** is detecting an existing virtual environment and not allowing you to create or activate a new one for your project, you can try unsetting the virtual environment by running:  
$env:VIRTUAL\_ENV=$null

1. This will clear the virtual environment variable for the current session.
2. **Ignore Existing Virtual Environments**: You can tell **pipenv** to ignore any existing virtual environments and create a new one by setting the **PIPENV\_IGNORE\_VIRTUALENVS** environment variable. Run this in PowerShell:

$env:PIPENV\_IGNORE\_VIRTUALENVS=1

1. After setting this, try running **pipenv --venv** again to see if it will now point to the correct virtual environment for your project.
2. **Create the Virtual Environment**: If there's no virtual environment set up for your project yet, you can force **pipenv** to create one by running:

pipenv install

1. This command should create a new virtual environment and install the dependencies specified in your **Pipfile**.
2. **Activate the New Virtual Environment**: After creating the new environment, try activating it again with:

pipenv shell

**Verify the Active Virtual Environment**: Once you've activated the new environment, you can check which one is active using:

echo $env:VIRTUAL\_ENV

Success

To delete a virtual environment

1. Deactivate the environment: **exit** (if activated).
2. In project directory, run: **pipenv --rm**.
3. Optionally, delete **Pipfile** and **Pipfile.lock**.
4. For manual removal, delete the environment folder from **~/.virtualenvs** or **C:\Users\<YourUsername>\.virtualenvs**.
5. To start fresh, run: **pipenv install** in the project directory.

CTRL-Y is redo

I had an issue where the PowerShell window would run the server, but VSC would not. Both environments showed the same virtual environment running. It turned out VSC was pointing to a different python interpreter. After I changed it I still had issues until I closed the terminal and opened a new one. Then it worked as expected.

PoweShell  
To copy text into a text file like readme.txt : "This is my text string" | Out-File readme.txt