

Alfabolt Backend Training Guide (React / Next.js)

The Intern Operating System (iOS)

* **Daily buddy syncs**: Quick check-ins to share updates, ask questions, and get unblocked.
* **Weekly lead reviews**: Show progress, get feedback, and align on next steps.
* **Friday wrap-ups**: Short presentations to reflect on what you did, learned, and what’s next.
* **Brag list**: Keep a personal record of wins, learnings, and highlights — helps with portfolio building later.
* **Present everything**: Design is communication. Sharing your work is part of the craft.
* **Feedback culture**: We give and receive feedback kindly, clearly, and often.
* **Stay curious**: Ask questions, explore, and share cool things with your team — learning is part of the job.
* **Key Focus Areas:** Python fundamentals, Django framework basics, Django REST framework, database management, API design and development, version control, and effective communication.

Week 1: Python Fundamentals & Setup

**Goal:** Establish a strong understanding of Python basics and set up the development environment.

Topics:

* **Introduction to Python:** History, features, and applications.
* **Setting up the Environment:**
  + Installing Python (latest stable version).
  + Introduction to IDEs and text editors (e.g., VS Code, vim, PyCharm).
  + Using virtual environments (venv).
  + Package management with pip.
* **Python Basics:**
  + Variables, data types (integers, floats, strings, booleans etc).
  + Operators (arithmetic, comparison, logical).
  + Control flow (if-elif-else statements, for/while loops).
  + Basic input/output operations.
* **Data Structures:**
  + Lists, tuples, dictionaries, sets.
  + Operations and methods associated with each.
* **Functions:**
  + Defining functions, arguments, return values.
  + Lambda functions.
  + Decorators
* **Introduction to Version Control with Git:**
  + What is version control? Why Git?
  + Basic Git commands: clone, add, commit, status, log,squash.
  + Setting up a GitHub/GitLab account.\

Tasks & Exercises:

* Install Python and an IDE of your own choice.
* Set up a virtual environment for projects.
* Complete basic Python programming exercises (e.g., simple calculator, string manipulations, list operations).
* Create a GitHub/GitLab repository for your internship assignments.
* Practice basic Git commands.
* Write a data scraper for e-commerce store.

Resources:

* Official Python Documentation:<https://docs.python.org/3/>
* Python for Everybody (Book & Course by Dr. Charles R. Severance):<https://www.py4e.com/>
* PEP 8 -- Style Guide for Python Code:<https://www.python.org/dev/peps/pep-0008/>
* W3 School:<https://www.w3schools.com/python/>
* Learn Python:<https://www.learnpython.org/>
* Git Handbook:<https://guides.github.com/introduction/git-handbook/>
* Learn Git Branching (Interactive Tutorial):<https://learngitbranching.js.org/>

Week 2: Advanced Python & Introduction to Django

**Goal:** Dive deeper into Python concepts and get introduced to the Django framework as a foundation for DRF.

Topics:

* **Object-Oriented Programming (OOP) in Python:**
  + Classes and objects.
  + Inheritance, polymorphism, encapsulation.
  + Constructors (\_\_init\_\_), methods, attributes.
* **Modules and Packages:**
  + Importing modules.
  + Creating and using your own modules.
* **Error Handling:**
  + try-except blocks.
* **File Handling:**
  + Reading from and writing to files.
* **Introduction to Web Concepts for APIs:**
  + HTTP/HTTPS protocols, request/response cycle.
  + Common HTTP methods (GET, POST, PUT, DELETE, PATCH).
  + Status codes (2xx, 3xx, 4xx, 5xx).
  + Client-Server architecture.
  + Difference between frontend and backend.
* **Introduction to Django:**
  + What is Django? MVT (Model-View-Template) architecture overview (View part will be API focused).
  + Installation and setting up a Django project.
  + Understanding manage.py and project structure.
  + Creating your first Django app.
* **Version Control (Continued):**
  + Branching and merging in Git (git branch, git checkout, creating/deleting branches).
  + Understanding the purpose of Pull Requests (or Merge Requests) in collaborative development and for code integration.
  + **How to make a pull request:** Pushing a feature branch to the remote repository (e.g., GitHub) and initiating a PR.
  + **Writing good PR descriptions:** Clearly articulating the "what" (the changes made) and the "why" (the purpose or problem solved) to provide context for the reviewer.
  + **Code comments:** Ensuring your code is well-commented, especially for complex logic, to aid reviewers in understanding your changes.
  + The code review process: What to expect, how to interpret feedback constructively, and the importance of timely reviews.
  + **Following up on PR feedback:** How to address comments, make necessary adjustments, push updates to the PR, and engage in discussion with the reviewer.

Tasks & Exercises:

* Develop small Python programs using OOP concepts.
* Write scripts that read from and write to files.
* Install Django and create a basic Django project and app.
* Explore the Django project structure.
* Practice Git branching: create a feature branch, make commits, and merge it back to the main branch.

Resources:

* Python OOP: Real Python -<https://realpython.com/python3-object-oriented-programming/>
* Official Django Documentation:<https://docs.djangoproject.com/en/stable/>
* Test driven DRF Views Part 1 (Resource for later, good to be aware of):<https://testdriven.io/blog/drf-views-part-1/>
* Django Girls Tutorial (for basic Django setup and concepts):<https://tutorial.djangogirls.org/>
* Pro Git book (Comprehensive Git reference):<https://git-scm.com/book/en/v2>
* Git Tutorial:<https://git-scm.com/docs/gittutorial>
* Learn Git Branching (Interactive):<https://learngitbranching.js.org/>
* Pull requests: <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests>

Week 3: Django Models, Admin & Intro to API Concepts

**Goal:** Understand Django Models for database interaction, utilize the Admin Panel, and begin understanding core API principles.

Topics:

* **Django Models:**
  + Defining models (classes representing database tables).
  + Fields types (CharField, IntegerField, DateTimeField, ForeignKey, ManyToManyField, etc.).
  + Model relationships (One-to-One, One-to-Many, Many-to-Many).
  + Migrations: makemigrations and migrate.
* **Django ORM (Object-Relational Mapper):**
  + Understand why ORM is used and how it differs from Raw SQL.
  + Querying data: filter(), get(), all(), exclude(), Q object.
  + Creating, updating, and deleting objects.
  + Ordering and slicing QuerySets.
  + Aggregations and annotations.
* **Django Admin Panel:**
  + Automatic admin interface for data management.
  + Registering models with the admin site.
  + Customizing the admin display.
* **Database Basics:**
  + Brief overview of relational databases (e.g., SQLite, PostgreSQL).
  + Understanding primary keys, foreign keys, indexing, normalization, denormalization.
* **Introduction to APIs & REST:**
  + What is an API? Why are they important in modern applications?
  + REST principles (Representational State Transfer).
  + Statelessness, Client-Server, Cacheability, Layered System, Uniform Interface.
  + Understanding JSON as a data format for APIs.
  + Understanding Web Communication Protocols such as REST, SOAP, WebSockets.
* **Communication:**
  + Importance of clear and concise communication for technical topics.
  + Asking effective questions for API design/issues.
  + Providing status updates on development tasks.

Tasks & Exercises:

* Create a To-Do List project in Django: set up the basic project structure and define the models.
* Run migrations to create database tables.
* Use the Django shell (python manage.py shell) to practice ORM queries.
* Register your models with the Django admin and explore its functionalities.
* Research and document key REST principles and compare different web communication protocols.
* Practice creating and parsing JSON objects in Python.

Resources:

* Django Models:<https://docs.djangoproject.com/en/stable/topics/db/models/>
* Making Queries:<https://docs.djangoproject.com/en/stable/topics/db/queries/>
* Examples of model relationship API usage:<https://docs.djangoproject.com/en/stable/topics/db/examples/> (Note: Link might be for a specific Django version, ensure to check latest stable docs)
* Django Admin Site:<https://docs.djangoproject.com/en/stable/ref/contrib/admin/>
* REST API Introduction (GeeksforGeeks):<https://www.geeksforgeeks.org/rest-api-introduction/>

Week 4: Django REST framework - Serializers & Views

**Goal:** Dive into Django REST framework (DRF), focusing on serializers for data representation and building API views, starting with Function-Based Views and introducing Class-Based Views via APIView and GenericAPIView.

Topics:

* **Introduction to Django REST framework (DRF):**
  + What is DRF and why use it? Core components.
  + Installation and setup within a Django project.
* **Serializers:**
  + Core concepts
  + Understanding Serializer vs ModelSerializer.
  + Fields in serializers (source, read\_only, write\_only), validation methods.
  + Handling relationships in serializers (PrimaryKeyRelatedField, StringRelatedField, Nested Serializers, SlugRelatedField).
* **Function-Based API Views (FBVs) in DRF:**
  + Using the @api\_view decorator for FBVs.
  + Handling different HTTP methods (GET, POST, PUT, PATCH, DELETE).
  + Returning Response objects.
* **Introduction to Class-Based Views (CBVs) in DRF - The APIView:**
  + Why CBVs? (Organization, inheritance, reusability).
  + Understanding Django's base View class (how it handles request dispatching).
  + DRF's APIView class:
    - Extends Django's View.
    - Provides DRF-specific functionalities: uses DRF's Request and Response objects, handles API-specific exceptions, manages authentication, permissions, and throttling.
    - Defining methods for HTTP verbs (e.g., get(), post(), put(), delete()).
  + Alfabolt's preference: "CBVs is the preferred way to write API views. We at Alfabolt use CBVs."
  + Brief mention: APIView is the base for more specialized Generic Views (covered in Week 5).

* **Django URL Routing for APIs:**
  + Configuring urls.py for both FBVs and CBVs (.as\_view() for CBVs).
* **Version Control (Advanced Git):**
  + Working with remote repositories: push, pull, fetch.
  + Understanding merge vs rebase vs squash merge (conceptual overview).
  + Local branches vs. remote tracking branches (origin/main).
  + Understanding and resolving merge conflicts.

Tasks & Exercises:

* Install DRF and integrate it into your Django project.
* Create ModelSerializer classes for your existing models, experimenting with different field types and relationship handling.
* Write function-based API views for basic CRUD operations on a model.
* Rewrite one of your FBVs as a Class-Based View using APIView.
* Test your API endpoints thoroughly using tools like Postman.
* Practice advanced Git workflows: create a feature branch, make changes, rebase it onto main (locally), and then merge. Simulate and resolve a merge conflict.

Resources:

* Django REST framework Official Tutorial (Sections 1 & 2 - Serialization, Requests & Responses):<https://www.django-rest-framework.org/tutorial/1-serialization/> &<https://www.django-rest-framework.org/tutorial/2-requests-and-responses/>
* DRF Tutorial Section 3 (Class-Based Views - focus on APIView initially):<https://www.django-rest-framework.org/tutorial/3-class-based-views/>
* DRF API Guide - Serializers:<https://www.django-rest-framework.org/api-guide/serializers/>
* DRF API Guide - Requests:<https://www.django-rest-framework.org/api-guide/requests/>
* DRF API Guide - Responses:<https://www.django-rest-framework.org/api-guide/responses/>
* DRF API Guide - Views (Function Based Views & APIView):<https://www.django-rest-framework.org/api-guide/views/>
* Atlassian Git Tutorials (for merge, rebase concepts):<https://www.atlassian.com/git/tutorials>
* Video tutorial to understand git merge vs rebase: <https://www.youtube.com/watch?v=zOnwgxiC0OA&ab_channel=TheModernCoder>

Week 5: Advanced DRF - Generic Views, ViewSets & Authentication/Permissions

**Goal:** Explore more advanced DRF features including Generic Class-Based Views for common patterns, ViewSets for rapid CRUD API development, and an introduction to API authentication and permissions. Start planning the final project.

Topics:

* **Generic Class-Based Views (CBVs) in DRF:**
  + Building on APIView: The GenericAPIView class (provides core functionality like get\_queryset(), get\_serializer\_class()).
  + Mixin Classes for common actions:
    - ListModelMixin, CreateModelMixin
    - RetrieveModelMixin, UpdateModelMixin, DestroyModelMixin
  + Concrete Generic Views (combining GenericAPIView with mixins):
    - ListAPIView, CreateAPIView, ListCreateAPIView
    - RetrieveAPIView, UpdateAPIView, DestroyAPIView
    - RetrieveUpdateAPIView, RetrieveDestroyAPIView, RetrieveUpdateDestroyAPIView
* **ViewSets & Routers:**
  + ViewSet class (combines logic for a set of related views).
  + GenericViewSet (inherits from GenericAPIView and ViewSetMixin).
  + ModelViewSet (inherits from GenericViewSet and includes implementations for all CRUD actions by default using mixins).
  + Using Routers (SimpleRouter, DefaultRouter) for automatic URL configuration for ViewSets.
* **Authentication in DRF:**
  + How authentication works in APIs (identifying the user making the request).
  + Common schemes: BasicAuthentication, SessionAuthentication.
  + TokenAuthentication (stateless, widely used for APIs).
  + Setting default authentication schemes.
* **Permissions in DRF:**
  + Controlling access to API endpoints based on user identity or role.
  + Built-in permission classes: AllowAny, IsAuthenticated, IsAdminUser, IsAuthenticatedOrReadOnly.
  + Object-level permissions.
  + Creating custom permissions (brief overview).
* **Middleware in Django (Relevance to APIs):**
  + What is middleware? Its role in the request/response cycle (processing requests before they reach the view and responses before they leave).
  + How to write and use custom middleware (e.g., for custom headers, advanced logging, request processing).
  + Django documentation covers middleware quite well.
* **Project Planning (API Focused):**
  + Define scope for a small project: a CRUD API for a specific resource, potentially with some business logic, authentication, and permissions.
  + Design API endpoints (URLs, HTTP methods, expected request/response formats).
  + Define database schema (models) and serializers.
  + Break down the project into smaller tasks.

Tasks & Exercises:

* Refactor existing APIViews to use Generic Class-Based Views where appropriate.
* Implement CRUD operations for a model using ModelViewSet and Routers.
* Secure an API endpoint using TokenAuthentication.
* Apply different permission classes (IsAuthenticated, IsAdminUser) to your views.
* Write a simple custom middleware (e.g., to log specific request headers or measure request time).
* Outline the features, API endpoints, and database schema for your final project.

Resources:

* DRF Tutorial (Sections 3, 4, 5, 6 - CBVs, Mixins, ViewSets, Authentication & Permissions):<https://www.django-rest-framework.org/tutorial/3-class-based-views/> onwards.
* DRF API Guide - Generic views:<https://www.django-rest-framework.org/api-guide/generic-views/>
* DRF API Guide - ViewSets:<https://www.django-rest-framework.org/api-guide/viewsets/>
* DRF API Guide - Routers:<https://www.django-rest-framework.org/api-guide/routers/>
* DRF API Guide - Authentication:<https://www.django-rest-framework.org/api-guide/authentication/>
* DRF API Guide - Permissions:<https://www.django-rest-framework.org/api-guide/permissions/>
* Django Middleware:<https://docs.djangoproject.com/en/stable/topics/http/middleware/>

Week 6: Project Development & API Best Practices

**Goal:** Develop the planned API project, focusing on applying all learned DRF concepts, and learn about API design best practices, testing, and documentation.

Topics & Activities:

* **Project Development (API Focused):**
  + Implement the models, serializers, and API views (Generic Views preferred) for your project.
  + Ensure proper database handling and data validation through serializers.
  + Implement authentication and permissions as planned.
  + Utilize version control (Git) throughout the development process with meaningful commits and branches if needed.Follow best practices.
  + Implement any custom middleware if planned.
* **API Design Best Practices:**
  + Consistent URL naming (plural nouns for resources, clear hierarchy).
  + Proper use of HTTP methods (idempotency of GET, PUT, DELETE).
  + Effective use of HTTP status codes for different scenarios.
  + API Versioning strategies (URL, header, query parameter - brief introduction).
  + Pagination for list endpoints (limit/offset, cursor-based).
  + Filtering and ordering capabilities for collections.
  + Handling errors gracefully and providing informative error messages.
* **Testing APIs (Basic to Intermediate):**
  + Manually test all API endpoints thoroughly using Postman.
  + Introduction to automated testing with DRF's APITestCase or pytest-django.
  + Writing Unit tests for apis  (testing status codes, response data, side effects).
* **API Documentation:**
  + Importance of API documentation for consumers.
  + Writing a comprehensive README.md explaining API setup, authentication, endpoints (with request/response examples).
  + You can use swagger or postman for api documentation.
* **Code Quality & Best Practices:**
  + Writing clean, readable, and maintainable code (DRY principle).
  + Adding meaningful comments and docstrings.
  + Following PEP 8 guidelines for Python.
  + Write Pythonic code.
  + Code linting and formatting (e.g., Flake8, Black).
* **Project Presentation & Code Review:**
  + Prepare a short presentation of your API project (demonstrate endpoints, explain design choices).
  + Participate in a code review session with your lead and buddy, focusing on best practices and lessons learned.

Tasks & Exercises:

* Complete your assigned API project.
* Regularly commit your changes to Git and push to your remote repository.
* Implement pagination and filtering for key list endpoints in your project.
* Write basic automated tests for at least one CRUD operation in your API.
* Write a comprehensive README.md for your API project, detailing setup, authentication, and all endpoints with examples.
* Prepare for the final project presentation.

Resources:

* All previously mentioned Django and DRF documentation.
* API Design Best Practices (various articles, e.g., from Microsoft Docs, Google Cloud API Design Guide, or RESTful API Design on restapitutorial.com).
* DRF Pagination:<https://www.django-rest-framework.org/api-guide/pagination/>
* DRF Filtering:<https://www.django-rest-framework.org/api-guide/filtering/>
* DRF Testing:<https://www.django-rest-framework.org/api-guide/testing/>
* Python best practices: <https://docs.python-guide.org/writing/style/>

General Guidelines & Expectations

* **Proactiveness:** Take initiative, ask questions, and explore beyond the planned topics.
* **Collaboration:** Work with your buddy and other interns. Share knowledge and help each other.
* **Time Management:** Manage your time effectively to cover the weekly topics and complete assignments.
* **Professionalism:** Maintain professional conduct in all communications and interactions.
* **Feedback:** Be open to receiving and providing constructive feedback.

We hope this guideline provides a clear path for your internship. Welcome aboard, and let's build some great Backend together!