Project (Social Media)

- Develop self registraation
- Develop user authentication for the app
- Develop a feature such as an upload/update(owned by the authenticated user) social media posts including images with some title, content, and tags.
- Develop a features such as comments and like on posts (per user, per post 1 like and unlimited comments)
- Search posts on the basis of titles and tags
- Develop a feature to filter
- Share email notification via background task
- Allow administration to view the raw data

Week	Session	Mentor	Assignement	Status	Topic	Comments
Week 01 (9th April - 15th April)	Installation and Introduction to Django	Maulik	- VsCode Editor: Install VsCode and familiarize interns with its interface. Customize VsCode settings according to personal preferences. Explore and install popular VsCode extensions for Django development (e.g., Django Template, Python, GitLens). Practice using VsCode's debugging features for Python. - Git & Github: Create a new repository on Github and learn basic Git commands (clone, add, commit, push, pull). Collaborate on a simple project with other interns using Git and Github (create branches, merge changes, resolve conflicts). Implement Git branching strategies like GitFlow or Feature Branching in a sample project. Explore advanced Git features like rebasing and squashing commits. - SQLite3: Set up a Django project with SQLite3 as the database backend. Create models, migrate them to the database, and perform CRUD operations using Django's ORM. Write custom SQL queries using SQLite3 syntax and execute them through Django. - Postman: Install Postman and create a new collection for API testing. Test CRUD operations of a Django REST API using Postman requests. Explore Postman's features like environment variables, scripting, and automated testing. - Virtual Environment: Learn to create and activate virtual environments using venv or virtualenv. Set up a virtual environment for a Django project and install project dependencies. Explore how virtual environments help manage project dependencies and isolate project environments. - DBeaver: Install DBeaver and connect it to a SQLite database. Perform database operations like querying, creating tables, and modifying data using DBeaver's GUI. Connect DBeaver to other database systems like PostgreSQL or MySQL and perform similar operations.		- Learn and explore various web dev tools like VsCode Editor, Git & Github, Sqlite3, Postman, Virtual Environment, DBeaver, and Other Essential Tools - Install Django framework and it's dependencies - Setup Django environment - Create your first sample Django project - Learn about project structure in Django	
Week 02 (16th April - 22nd April)	System Design	Maulik	- System Design: Start with understanding the fundamentals of system design, including scalability, reliability, and performance. Study the components of a typical web application architecture (client, server, database, cache, load balancer, etc.). Analyze and discuss the architecture of popular websites or applications (e.g., Twitter, Facebook) to understand how they handle scalability and reliability. Design a simple system architecture for a web application using Django, considering factors like user authentication, database access, caching, and load balancing. Explore design patterns like MVC (Model-View-Controller) and how they apply to Django projects. - Caching Architecture: Learn about the importance of caching in improving the performance of web applications. Explore different types of caches (e.g., in-memory cache, distributed cache) and their use cases. Implement caching in a Django project using Django's built-in caching framework or third-party libraries like Redis. Experiment with caching strategies like cache invalidation and cache expiration to understand their impact on application performance. Analyze the performance improvements achieved through caching and compare them with uncached scenarios. - Database Connection: Dive deeper into database connections in Django, understanding how Django interacts with various database systems. Experiment with configuring Django to connect to different database backends like PostgreSQL, MySQL, or Oracle. Explore advanced database connection settings in Django settings.py file, such as connection pooling and query optimization. Study techniques for optimizing database connections, such as connection pooling, lazy connections, and read replicas. Investigate Django's support for multiple databases and how to handle data replication and synchronization in a multi-database environment.		- Learn about System Design - Learn about Caching Architecture - Data base connection	

Project (Social Media)

- Develop self registraation
- Develop user authentication for the app
- Develop a feature such as an upload/update(owned by the authenticated user) social media posts including images with some title, content, and tags.
- Develop a features such as comments and like on posts (per user, per post 1 like and unlimited comments)
- Search posts on the basis of titles and tags
- Develop a feature to filter
- Share email notification via background task
- Allow administration to view the raw data

Week	Session	Mentor	Assignement	Status	Topic	Comments
Week 03	Routing and Views in Django	Mustafa	- Mapping web URLs to view functions: Study Django's URL dispatcher and how it maps URLs to view functions within a Django project. Create a Django project and define URL patterns using regular expressions and the path() or re_path() functions in urls.py. Practice mapping URLs to view functions with different URL patterns, including capturing URL parameters. Explore Django's namespace URL patterns for organizing URL patterns within your project. - Familiarize with various HTTP methods: Learn about the different HTTP methods (GET, POST, PUT/PATCH, DELETE) and their purposes. Create Django views that handle requests for each HTTP method. Use Django's @require_http_methods decorator to restrict views to specific HTTP methods. Implement CRUD (Create, Read, Update, Delete) operations in Django views, mapping each operation to the appropriate HTTP method. Creating simple views in Django with HTTP response or JSON response: - Create Django views that return HTTP responses using Django's HttpResponse class. Practice creating views that return JSON responses using Django's JsonResponse class. Explore different ways to serialize data into JSON format, such as using Django's serializers or third-party libraries like Django REST Framework. Learn about different HTTP status codes and their meanings (e.g., 200 for success, 404 for not found, 500 for server error).		- Map web URLs to view functions - Familiarize with various HTTP methods, including GET, POST, PUT/PATCH, and DELETE - Create simple views in django with HTTP response or JSON response and learn about different status codes	
Week 04	Databases and Model layer	Maulik	- Connecting to a relational database in Django: Set up a Django project with a relational database backend such as PostgreSQL, MySQL, or SQLite. Configure the database connection settings in Django's settings.py file. Learn about Django's database abstraction layer and how it facilitates connecting to different database systems. Experiment with different database configurations, such as specifying database host, port, username, and password. - Learning about Django migrations: Study the concept of database migrations and their importance in managing database schema changes in Django projects. Create Django models representing different data entities and relationships. Generate and apply migrations using Django's manage.py command to synchronize the database schema with changes made to models. Explore advanced migration scenarios, such as renaming fields, adding indexes, or creating custom migrations. - Exploring Django ORM (Object Relational Mapping): Understand the basics of Django models and how they represent database tables. Learn about different types of model fields provided by Django (e.g., CharField, IntegerField, ForeignKey). Define relationships between Django models using ForeignKey, OneToOneField, and ManyToManyField. Practice querying data from the database using Django's QuerySet API, including filtering, ordering, and aggregating data. Experiment with advanced querying techniques like annotations, prefetch_related, and select_related to optimize database queries.		- Learn about connecting to a relational database in Django. - Learn about Django migrations, and how to use them to update your database schema. - Explore Django ORM (Object Relational Mapping). Understand Django models, relationship between models, learn how to query on models.	
Week 05	Lear dynamic request response	Mustafa	- Building dynamic Django templates: Create Django templates (*.html files) within your Django project's template directory. Use Django template language (DTL) to insert dynamic data into HTML elements. Practice passing data from views to templates using Django's template context. Build templates that display data retrieved from Django models, such as user profiles or blog posts. Implement template inheritance to create a base template that defines common layout elements (header, footer, navigation) shared across multiple pages. - Adding conditions and loops in templates: Learn about Django template tags and filters for adding logic to templates. Use {% if %} and {% else %} tags to conditionally display content based on data or user permissions. Implement {% for %} loops to iterate over lists or querysets and dynamically render HTML elements for each item. Explore advanced template tags like {% block %} and {% include %} for creating modular and reusable templates. Practice using template tags and filters for common tasks like date formatting, string manipulation, and HTML escaping. - Building common layouts: Create a base template that defines the overall layout structure of your website (e.g., header, footer, sidebar). Extend the base template to create specialized templates for different sections of your site. Implement template inheritance to override specific blocks or sections in child templates while inheriting the rest from the parent template. Use {% include %} to include common elements like navigation menus or call-to-action buttons across multiple templates. Experiment with CSS frameworks like Bootstrap or Foundation to style your templates and create responsive layouts.		- Build Dynamic Django templates that display the HTML GUI seen by clients, using server-side Python - Add conditions and loops in templates to avoid repetitions, build common layouts -	

Project (Social Media)

- Develop self registraation
- Develop user authentication for the app
- Develop a feature such as an upload/update(owned by the authenticated user) social media posts including images with some title, content, and tags.
- Develop a features such as comments and like on posts (per user, per post 1 like and unlimited comments)
- Search posts on the basis of titles and tags
- Develop a feature to filter
- Share email notification via background task
- Allow administration to view the raw data

Week	Session	Mentor	Assignement	Status	Topic	Comments
Week 06	Django template	Mustafa	- Building forms in Django templates: Learn about Django's forms framework and how it simplifies the process of building HTML forms in Django templates. Create Django forms using Django's forms.py module, defining form fields and validation rules. Render forms in Django templates using the {{ form }} template variable or by manually iterating over form fields. Experiment with different form field types provided by Django (e.g., CharField, IntegerField, ChoiceField) and customize their appearance using widget options. - Handling form submissions: Understand the HTTP request-response cycle in Django and how form submissions are handled. Create Django views to handle form submissions, including validating form data and processing form submissions. Use Django's request.POST or request.GET attributes to access form data submitted via POST or GET requests. Implement form validation using Django's built-in form validation mechanisms or custom validation logic in views. Practice redirecting users to different pages or displaying error messages based on the outcome of form submission processing. - Validating forms in Django templates: Explore Django's form validation features, including built-in validators and custom validation methods. Implement client-side form validation using JavaScript frameworks like jQuery or libraries like Django's built-in form validation. Use Django's form error handling mechanisms to display validation errors in Django templates. Experiment with conditional formatting in templates to highlight invalid form fields or display error messages next to form inputs. - Submitting forms in Django templates: Understand how form submissions are initiated by users through HTML form elements. Create HTML form elements in Django templates using <form> tags and form input elements like <input>, <select>, and <textarea>. Implement form submission handling in Django views, including processing form data and redirecting users after form submission. Explore techniques for securing form submissions in Django, such as using CSRF tokens and implementing form field validation on the server-side.		- Understand the Django approach of building, handling, validating and submitting on Django template	

Project (Social Media)

- Develop self registraation
- Develop user authentication for the app
- Develop a feature such as an upload/update(owned by the authenticated user) social media posts including images with some title, content, and tags.
- Develop a features such as comments and like on posts (per user, per post 1 like and unlimited comments)
- Search posts on the basis of titles and tags
- Develop a feature to filter
- Share email notification via background task
- Allow administration to view the raw data

Week	Session	Mentor	Assignement	Status	Topic	Comments
Week 07 Week 08	Learn about various core concepts in Django REST Framework	Shahnavaz	- Learn about different status codes in APIs: Study the standard HTTP status codes and their meanings (e.g., 200 for success, 404 for not found, 500 for server error). Understand how APIs use status codes to communicate the outcome of API requests to clients. Practice returning appropriate status codes from Django views based on the outcome of API operations. - Learn to implement multiple versions of an API: Explore strategies for versioning APIs, such as URL versioning, parameter versioning, or content negotiation. Implement versioning for an existing Django API by prefixing URLs with version numbers or using custom HTTP headers. Learn how to maintain backward compatibility when introducing new API versions and communicate changes to API consumers. - Learn to implement exception handling in APIs: Study common error scenarios in API development and how to handle them gracefully. Implement exception handling in Django views to catch and handle errors like database exceptions or validation errors. Customize error responses returned by the API to provide meaningful error messages to clients. - Authentication: Explore different authentication methods supported by Django REST Framework (e.g., token authentication, session authentication, OAuth). Implement authentication for your Django API using Django REST Framework's built-in authentication classes. Practice securing API endpoints with authentication and restricting access based on user permissions. - Serializers: Understand the role of serializers in Django REST Framework for converting complex data types (e.g., Django models) into native Python data types and vice versa. Create serializers for Django models to define how model data should be represented in API responses. Experiment with serializers to handle nested data structures, write custom serializer fields, and validate input data. - Views: Learn about Django REST Framework's class-based views and function-based views for building API endpoints. Implement CRUD (Create, Read, Update, Delete) operations in Django views using Django REST Framework's generic views or mixins. Practice writing custom views for handling complex API logic or non-standard API endpoints. - Permissions: Study Django REST Framework's permission classes for controlling access to API endpoints. Implement permission classes to enforce access control rules based on user roles, group membership, or custom logic. Experiment with different permission classes like IsAuthenticated, IsAdminUser, or custom permissions tailored to your application's requirements. - Relationships: Understand how to represent relationships between resources in a RESTful API. Implement nested serializers and views to handle relationships between Django models. Explore different types of relationships (e.g., one-to-many, many-to-many) and how to represent them in API responses.		- Learn about different status codes in API - Learn to implement multiple versions of an API - Learn to implement exception handling in API's - Authentication - Serializers - Views - Permissions - Relationships	

Project (Social Media)

- Develop self registraation
- Develop user authentication for the app
- Develop a feature such as an upload/update(owned by the authenticated user) social media posts including images with some title, content, and tags.
- Develop a features such as comments and like on posts (per user, per post 1 like and unlimited comments)
- Search posts on the basis of titles and tags
- Develop a feature to filter
- Share email notification via background task
- Allow administration to view the raw data

Week	Session	Mentor	Assignement	Status	Topic	Comments
Week 09	Filtering, Pagination and Throttling	Shahnavaz	- Create an API list view: Define a Django view that returns a list of objects serialized as JSON, representing resources in your API. Use Django REST Framework's APIView class or APIView subclass to create the API view. Implement logic in the view to fetch the list of objects from the database or other data sources. - Filter API list view using query params: Extend the API list view to support filtering based on query parameters passed in the request URL. Define query parameters for filtering specific fields or attributes of the resource. Implement filtering logic in the API view to apply filters to the queryset based on the provided query parameters. - Implement search feature inside API list view: Extend the API list view to support searching for resources based on a search query. Define a search parameter in the request URL to specify the search query. Implement search logic in the API view to filter the queryset based on the provided search query using full-text search or other search algorithms. - Paginate API list view: Enable pagination for the API list view to limit the number of resources returned in each API response. Configure pagination settings such as the page size and pagination style. Implement pagination logic in the API view to slice the queryset into pages and return paginated results in the API response. - Control request rate from client side using Throttling: Configure throttling settings in Django REST Framework to limit the rate of requests from clients. Define throttling classes to specify different throttling policies for different views or user groups. Apply throttling to the API list view to enforce rate limits on requests based on client IP address, user, or other criteria.		- Create an API list view - Filter API list view using query params - Implement search feature inside API list view - Paginate API list view - Control request rate from client side using Throttling	
Week 10	Django admin portal	Shahnavaz	- Acquire the knowledge to reuse initial built-in Django applications: Explore the built-in Django applications such as auth, admin, and contenttypes. Study the purpose and functionality of each built-in application. Experiment with integrating these applications into your Django projects to leverage their functionality. Learn about customizing and extending built-in applications to meet specific project requirements. - Learn about Django admin: Familiarize yourself with the Django admin interface and its capabilities for managing Django projects. Set up the Django admin interface in your project and register models to make them editable in the admin interface. Customize the appearance and behavior of the admin interface using Django admin options and customizations. Experiment with advanced features of the Django admin interface such as custom actions, inlines, and list filters. - Learn about various Django commands: Explore Django's built-in management commands and their purposes. Study common management commands such as startproject, startapp, makemigrations, migrate, runserver, and createsuperuser. Practice using Django management commands to perform tasks like database migrations, running the development server, and creating administrative users. Learn how to create custom management commands to automate repetitive tasks specific to your Django project. - Learn about Django Shell: Understand the Django shell and its usefulness for interactive development and debugging. Start the Django shell using the python manage.py shell command. Practice using the Django shell to interact with Django models, querysets, and database objects. Experiment with running Python code snippets and testing Django features directly from the shell.		- Acquire the knowledge to reuse initial built-in Django applications - Learn about Django admin - Learn about various Django Commands - Learn about Django Shell	

Project (Social Media)

- Develop self registraation
- Develop user authentication for the app
- Develop a feature such as an upload/update(owned by the authenticated user) social media posts including images with some title, content, and tags.
- Develop a features such as comments and like on posts (per user, per post 1 like and unlimited comments)
- Search posts on the basis of titles and tags
- Develop a feature to filter
- Share email notification via background task
- Allow administration to view the raw data

Week	Session	Mentor	Assignement	Status	Topic	Comments
Week 11	Django Cronjobs, Workers, Signals, Job Queues and Testing	Maulik	- Cronjobs: Study the concept of cron jobs and their usage in scheduling recurring tasks. Implement cron jobs in Django using third-party packages like django-cron or Celery's periodic tasks. Schedule tasks such as sending periodic emails, generating reports, or updating database records using cron jobs. - Workers: Learn about asynchronous task processing and the role of workers in Django applications. Experiment with task queues and workers using Celery, a popular distributed task queue. Set up Celery in a Django project and define asynchronous tasks to be executed by workers. Monitor task execution and troubleshoot issues related to workers and task processing. - Signals: Understand Django's signals framework and its role in decoupling components within a Django application. Implement signal handlers to perform actions in response to specific events such as model save, delete, or user authentication. Explore built-in signals provided by Django and create custom signals for inter-app communication. - Job Queues: Learn about job queues and their importance in managing and processing background tasks in web applications. Explore different job queue systems such as Celery, RQ (Redis Queue), or Kafka. Integrate a job queue system with Django to offload time-consuming tasks from the main application thread. Experiment with prioritizing, scheduling, and retrying failed tasks in the job queue. - Integration of other technologies with Django such as Celery/Kafka: Study the integration of Celery and Kafka with Django for handling asynchronous task processing and message passing. Set up Celery and Kafka in a Django project and configure them to work together seamlessly. Implement real-time data processing and messaging features using Kafka and Celery in Django applications. Explore advanced integration scenarios such as using Kafka for event sourcing and Celery for task orchestration. - Testing in Django using Unit test cases and API Request Factory: Understand the importance of testing in software development and different testing methodologies. Study Django's built-in testing framework and learn about unit tests, integration tests, and functional tests. Write unit test cases for Django models, views, and utility functions using Django's TestCase class. Explore Django's API Request Factory for creating test requests and writing test cases for API views and endpoints. Practice writing test cases to cover different scenarios, including edge cases, error handling, and boundary conditions.		- Cronjobs - Workers - Signals - Job Queues - Integration of other technologies with Django such as celery/kafka - Testing in django using Unit test cases and API Request Factory	

Topics can be added
Django Life cycle (MVT), Middlewares, custom middlewares, Django Channels, Viewsets, API caching, CORS, CSRF, JWT, Django Debug ToolBar, simple dockerizing Django App