Problem Set 3

# A. Scheduling Periodic Tasks

For scheduling periodic tasks like downloading a list of ISINs every 24 hours, my choice would be Celery with Redis or RabbitMQ as a message broker. Celery is a distributed task queue system that is widely used for scheduling and executing asynchronous tasks in production environments.

## Why Celery?

1. Reliability: Celery is highly reliable due to its task retry mechanism, error handling, and support for robust message brokers like RabbitMQ or Redis. If a task fails (e.g., due to a network issue), Celery can retry it based on a defined policy.

2. Scheduling Support: Celery can schedule periodic tasks with the help of Celery Beat, which is perfect for tasks that need to be triggered every day or at specific intervals.

3. Scalability: Celery can scale across multiple workers, allowing it to handle high loads efficiently. This is essential when handling a large volume of tasks, such as downloading and processing a huge number of ISINs.

4. Asynchronous Task Execution: Celery allows you to decouple task execution from the web process. This means your web application won’t block while waiting for tasks to complete.

## Drawbacks:

1. Complexity: Celery adds additional complexity to the system, including the need for a message broker and periodic task scheduler (Celery Beat). For small or simpler projects, this may be overkill.

2. Latency: While Celery is scalable, there is a slight latency when dispatching tasks through a broker compared to using a local scheduler like Cron.

## Alternative for Smaller Scale:

If scalability is not a concern and you want simplicity, you can use Cron Jobs or APScheduler (a Python scheduler library). Cron is simple to set up and works well for periodic tasks, but it lacks the robustness and scalability that Celery offers.

## At Scale:

In a high-scale production environment, Kubernetes CronJobs can be a solid alternative, especially if you’re already using Kubernetes for container orchestration. Kubernetes CronJobs can handle large-scale periodic tasks and scale well with the rest of your microservices infrastructure.

# B. Flask vs Django

## Flask:

Circumstances to Use Flask:

1. Lightweight Applications: Flask is ideal for small or micro-applications where you need more control over components and a simple request/response cycle.

2. Microservices: Flask is often used for building microservices due to its flexibility, allowing developers to structure the project as needed.

3. Customization: Flask is highly customizable, and you can choose the libraries or extensions you want. This makes it suitable for highly custom applications where Django’s all-in-one framework could be limiting.

4. Rapid Prototyping: Flask has minimal setup and allows quick development, making it great for prototyping.

## Pros:

1. Lightweight and flexible.

2. Minimal boilerplate.

3. Highly modular.

## Cons:

1. Lacks built-in features (e.g., authentication, ORM) that Django provides.

2. Manual configuration required for database migrations, authentication, etc.

## Django:

Circumstances to Use Django:

1. Full-Scale Applications: Django is best for large-scale, feature-rich applications that require an integrated system (e.g., admin dashboard, ORM, authentication system).

2. Rapid Development: With its 'batteries-included' philosophy, Django provides everything out of the box (ORM, forms, admin, etc.), making it suitable for large projects where you need to avoid reinventing the wheel.

3. Monolithic Applications: Django is ideal when you need an all-in-one solution for large applications with tight coupling between components.

4. Enterprise Systems: Django’s admin interface, security features, and ORM make it a good fit for enterprise-level projects where consistency and stability are required.

## Pros:

1. Comes with built-in features like ORM, admin interface, and security modules.

2. Rapid development and deployment.

3. Extensive documentation and a large community.

## Cons:

1. Less flexible due to its monolithic nature.

2. Overhead for smaller or simple applications.