**Three use cases we need asynchronous programming**

**1 - I/O-Bound Operations** (e.g., File, Network, or Database Access)

**Scenario**: Your application needs to read a large file, make an HTTP request to an external API, or query a database.

**Why Asynchronous**?: I/O operations often involve waiting for external resources (disk, network, etc.), which can take milliseconds or even seconds. Without async, the thread handling the request would be blocked, wasting resources and reducing responsiveness.

**Benefit**: The thread is freed up to handle other requests while waiting for the API response, improving scalability in a web server scenario.

**2. Improving UI Responsiveness in Client-Server Applications**

**Scenario**: A backend service provides data to a desktop or mobile app, and the client needs to fetch this data without freezing the UI.

**Why Asynchronous?:** Synchronous calls block the calling thread (e.g., the UI thread in a client app), making the application unresponsive. On the backend, async endpoints ensure the server can handle multiple requests efficiently while the client awaits the response.

**Benefit**: The client app can remain responsive while the backend processes the request, and the server can handle more concurrent users.

**3. Parallel Processing of Independent Tasks (e.g., Batch Operations)**

**Scenario**: Your backend needs to process multiple independent tasks, such as sending emails to a list of users, resizing uploaded images, or performing calculations on a dataset.

**Why Asynchronous**?: Running these tasks synchronously would execute them one-by-one, taking longer and underutilizing CPU resources. Async allows you to run them concurrently, leveraging multiple threads or cores efficiently.

**Example**: Sending emails to multiple users in a .NET backend.