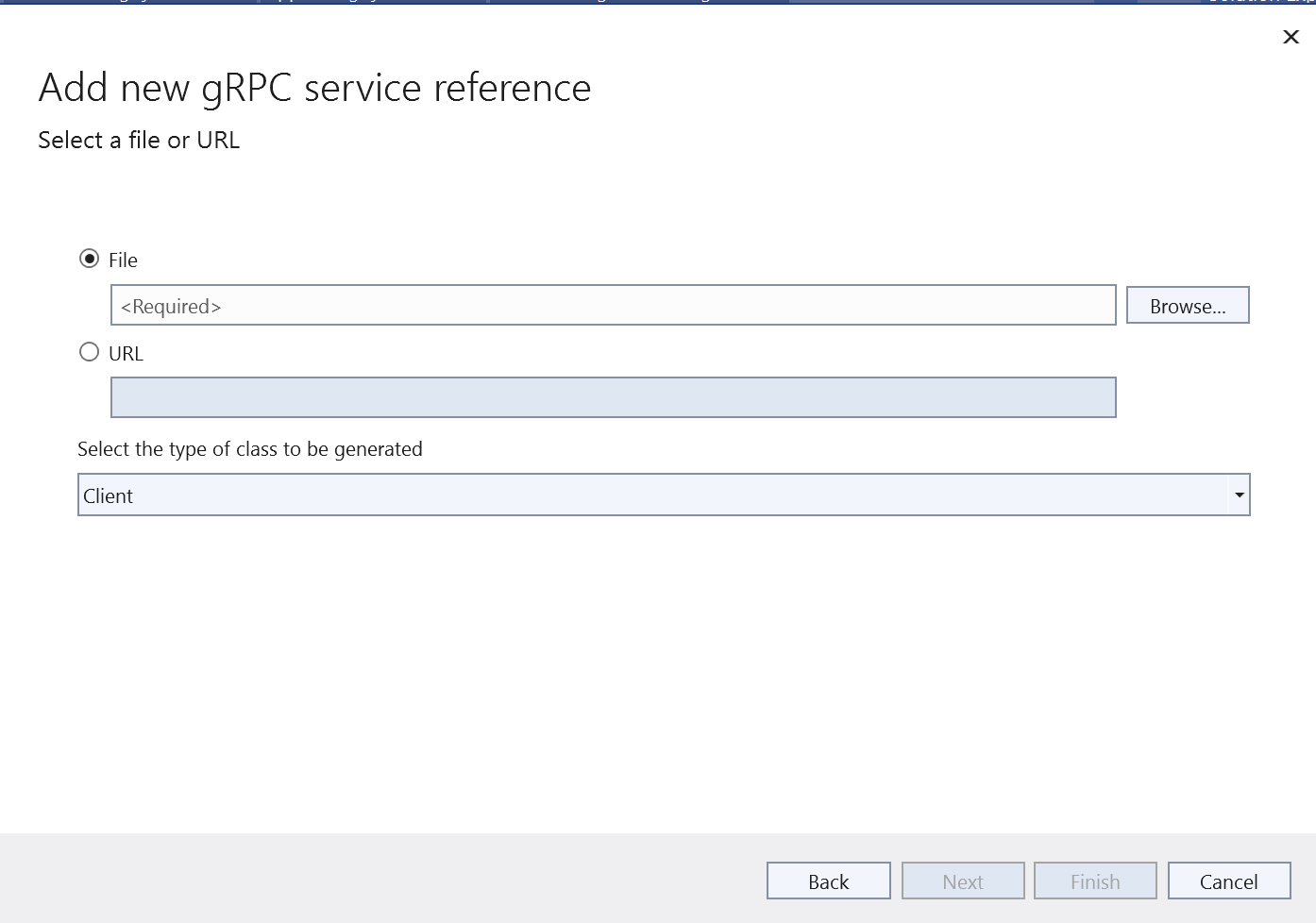
Demo Points:

Provide server service proto file as reference for client project by referencing proto file.

Generated client by providing proto file reference.



Unique Features:

Efficient gRPC Architecture: Our system leverages the gRPC framework for communication between client applications and the image processing service. This choice ensures fast and efficient communication, making it ideal for real-time image processing applications.

Asynchronous Processing: The image processing operations are designed to be asynchronous, allowing the system to handle multiple concurrent requests efficiently. This improves scalability and performance, enabling the system to process large volumes of image data with minimal latency.

Flexible and Extensible API: The API provided by our system supports a wide range of image processing operations, including flipping, rotating, resizing, converting to grayscale, and generating thumbnails. The API is designed to be flexible and extensible, allowing for easy addition of new features and functionality in the future.

Error Handling and Logging: Comprehensive error handling mechanisms are implemented to handle various scenarios, including invalid input parameters, processing failures, and network errors. Additionally, logging is implemented to track the status of image processing requests, monitor system performance, and troubleshoot issues.

Architecture:

Our system follows a microservices architecture, with a gRPC-based image processing service as the core component. The architecture consists of the following components:

Image Processing Service: Responsible for processing image transformation requests. Exposes a gRPC API that clients can use to send image processing requests.

Client Applications: Applications or services that interact with the Image Processing Service to perform various image processing operations. Clients can be web applications, mobile apps, or other backend services.

Storage: Utilizes cloud storage services like AWS S3, Google Cloud Storage, or Azure Blob Storage to store input and output images during the processing pipeline.

Logging and Monitoring: Implements logging and monitoring to track the status of image processing requests, monitor system performance, and troubleshoot issues.

Interesting Design Details:

Use of gRPC: The decision to use gRPC for communication between clients and the image processing service ensures efficient and fast communication, ideal for real-time image processing applications.

Flexible API: The API provided by our system is designed to be flexible and extensible, allowing for easy addition of new features and functionality in the future. This makes the system adaptable to changing requirements and business needs.

Error Handling and Logging: Comprehensive error handling mechanisms and logging are implemented to ensure the reliability and robustness of the system. This includes handling various error scenarios and logging relevant information for monitoring and troubleshooting purposes.

API in Detail:

The gRPC API provided by our system includes the following methods:

ProcessImage: This method takes a ProcessImageRequest containing the raw image data and processing parameters and returns a ProcessImageResponse containing the processed image data.

Little Language (if applicable):

Our system does not include a "little language" component.

Limitations:

Our architecture may have limitations in terms of scalability and fault tolerance under extremely high load conditions. To address this, additional measures such as load balancing and redundancy could be implemented.

The current implementation may have limitations in terms of the variety and complexity of image processing operations supported. Extending the system to support a wider range of operations would require additional development effort and testing.

Error handling and logging mechanisms may not cover all possible error scenarios. Continuous monitoring and feedback from production usage are essential for identifying and addressing any limitations in the implementation.