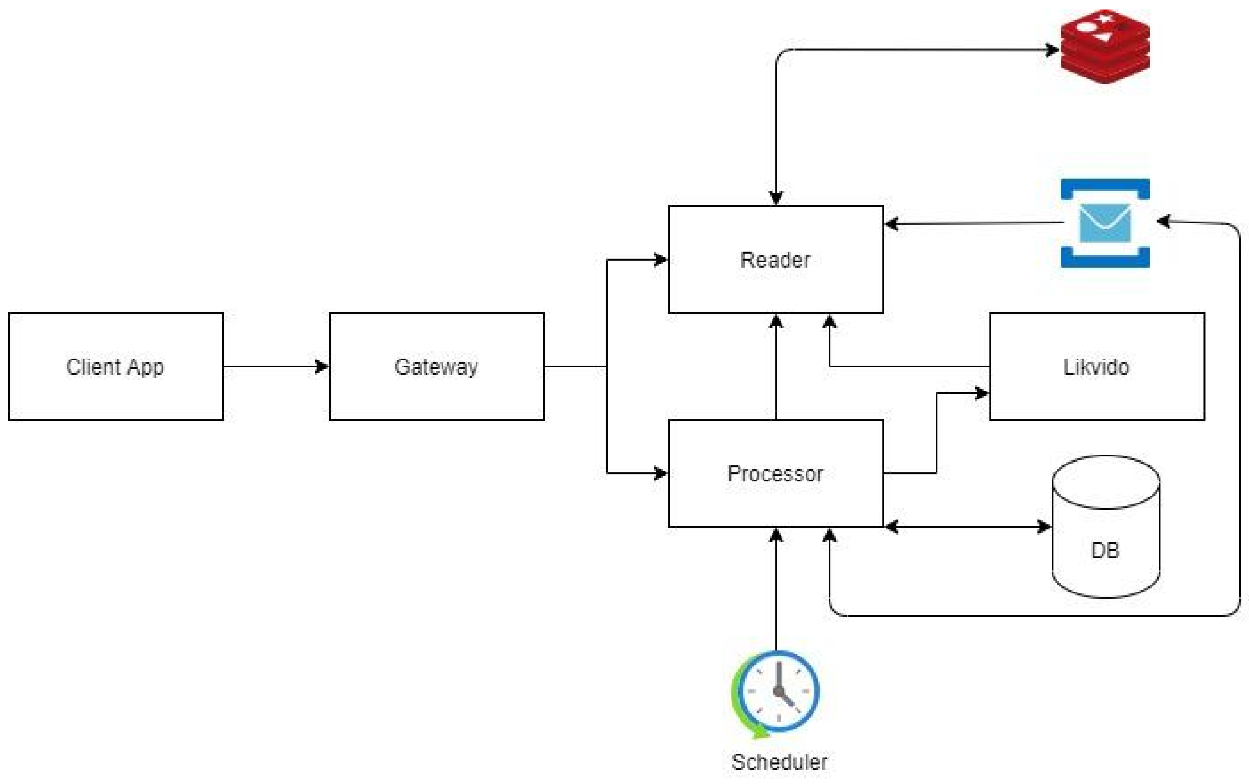
Likvido Invoice Client

A web application to create and read invoices from Likvido.com with fault tolerance and in high performance.

Motivation

- Develop a distributed application with microservices.
- Implement the microservices to provide better performance and the ability to scale independently.
- Ensure audit of the user operations.
- Implement the application with SOLID and DRY principles.
- Create reusable components.
- Develop the application following DDD approach.

Components and their communication



Client Application

A single page application to display the invoice list and create a new one.
Framework and Libraries: Angular 12, Angular Material, NgRx

Gateway

Accessible API to transfer read and write requests to desired microservices.
Framework and Libraries: ASP .NET 5 Web API, Ocelot

Invoice Processor

Microservice to store user requests to Outbox, send the invoice request to Likvdo and, update the outbox. Sending an invoice will run inside a transaction. When Likvido is unreachable, insisting on changing the outbox status, it will reschedule the queue.
Framework and Libraries: ASP .NET 5 Web API, Azure Service Bus, EF Core, MediatR, Automapper

Scheduler

Schedule the execution of the processing pending request command periodically.
Framework and Libraries: Quartz

Database

Store user requests in the outbox and contain user events for better audit and debugging.

Invoice Reader

Microservice to get data from Likvido, store and invalidate them inside Redis Cache for faster delivery.
Framework and Libraries: ASP .NET 5 Web API, Redis Cache, Azure Service Bus, MediatR, Automapper

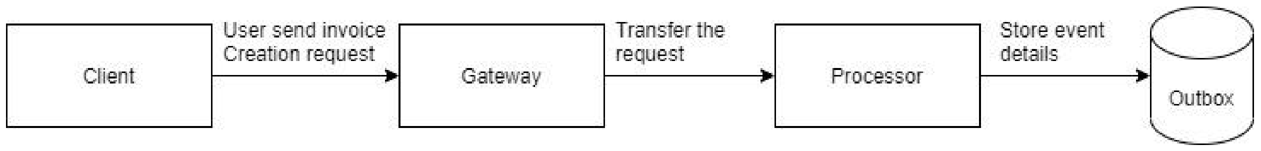
Redis Cache

Store the cache of invoice lists for faster delivery.

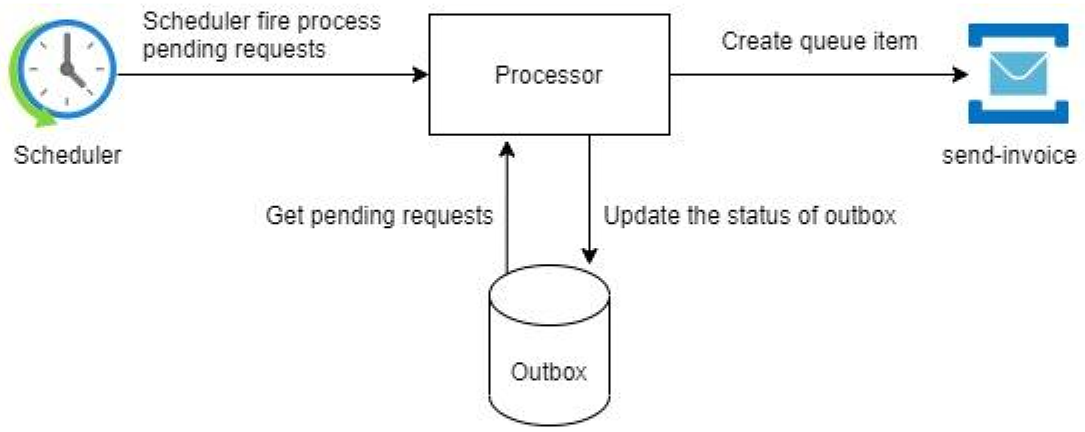
Service Bus

Used as the way of communication among microservie(s) in an asynchronous way.

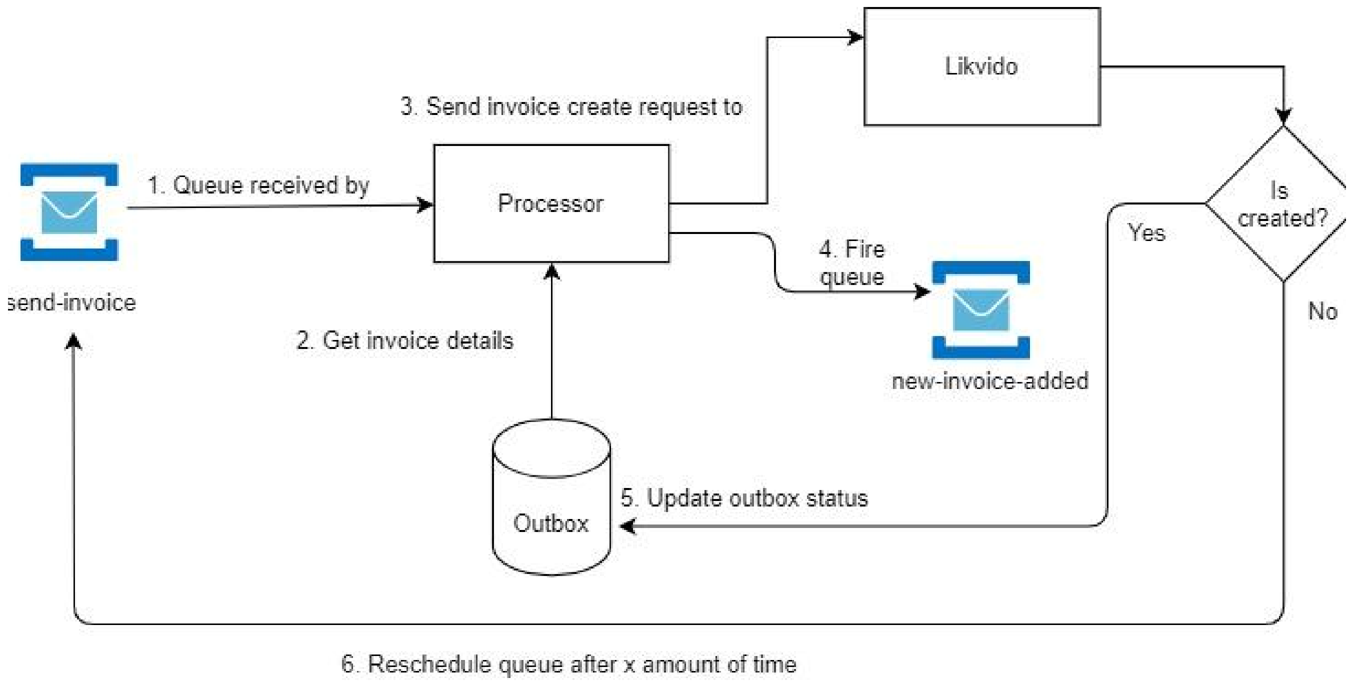
Stack holder and application communication



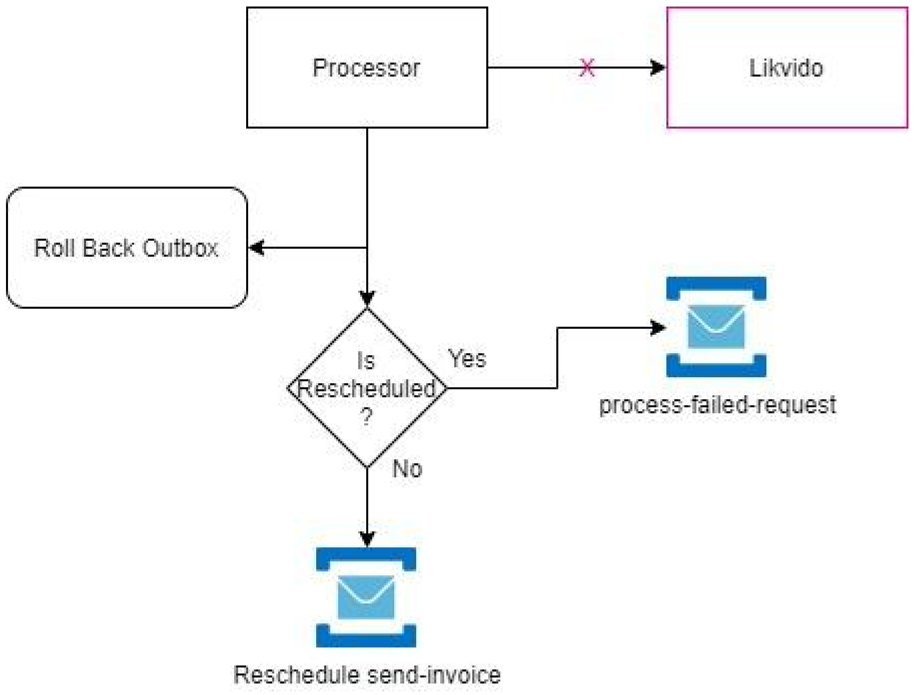
Processing pending requests



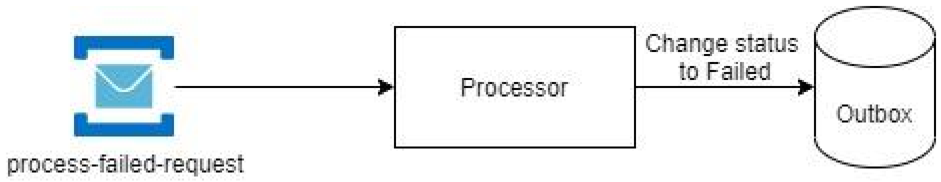
Send request to Likvido.com



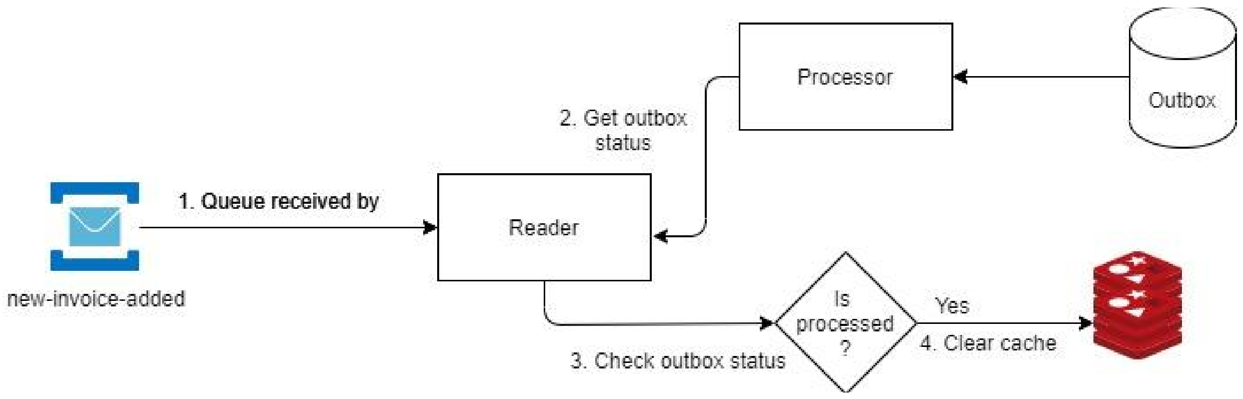
When Likvido is unreachable



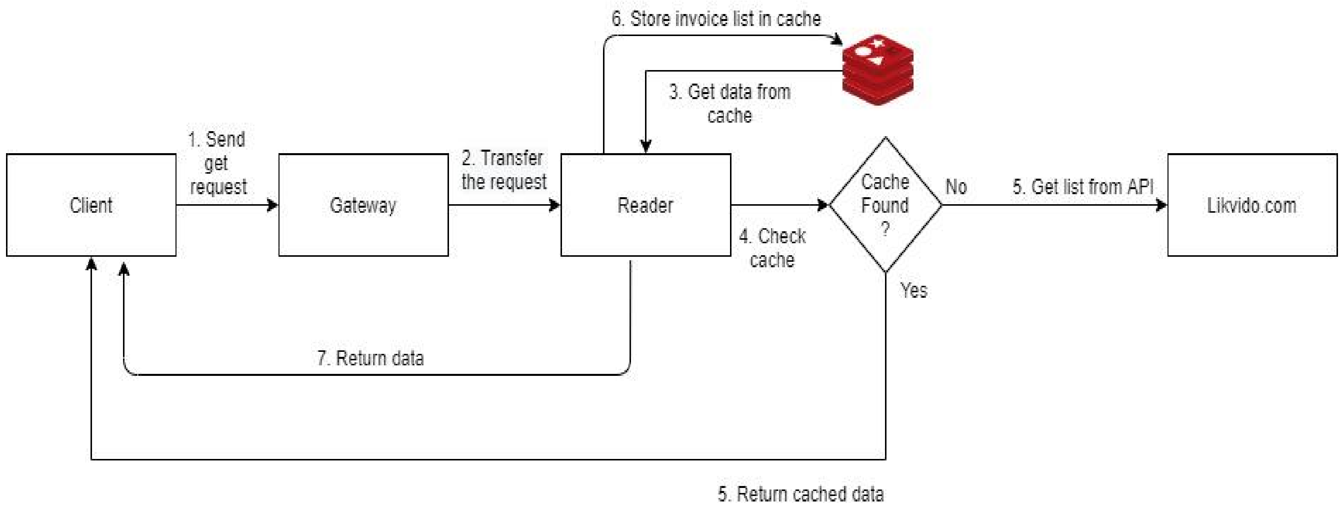
Process failed request



Cache Invalidation



Behind the scenes of retrieving Invoice



Tradeoffs of the Architecture

Advantages:

- Ensure invoice creation requests would be processed, maybe not now, but for sure at later.
- Full audit of the customer operations.
- As we are storing every customer operation event inside our database. So there is no chance of data loss.
- Caching helps us to deliver faster requests.
- Individual teams can autonomously develop/deploy microservices with minimum/no communication among other teams.
- Scale read and write microservice separately, depending on demand.
- Because of Gateway, our client application requires less modification during the internal microservice API changes.

Disadvantages:

- No matter whether Likvido.com is available or not, customers need to wait for a period of time to get their invoices live.
- The logic behind the application gets complex when compared to a simple CRUD applications.
- Creating deployment infrastructure and CI/CD would take more time than monolithic application.
- Because of caching there are some chances to get outdated data when anything gets changed from the back office of Likvido.

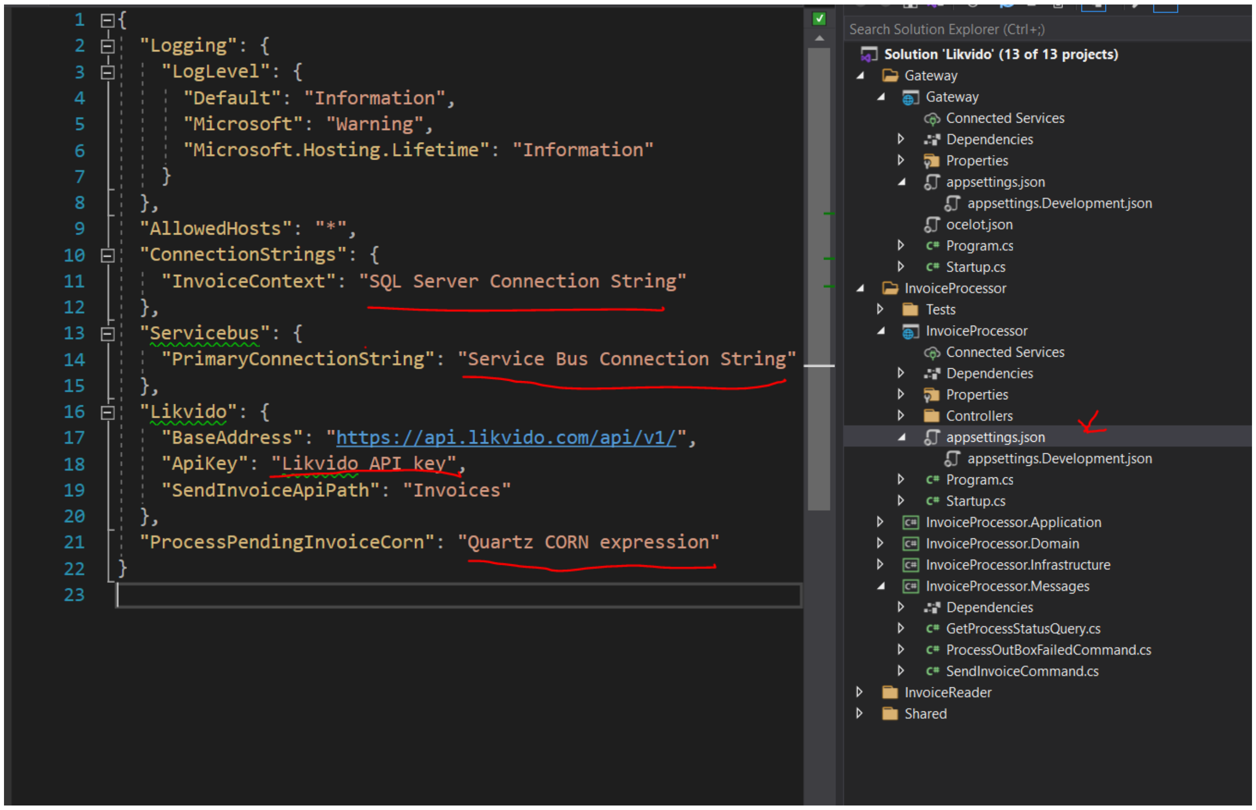
Installation and running locally

Prerequisite:

- .NET 5 SDK and runtime
- Visual Studio / .NET CLI
- MS SQL Server
- Node
- Angular CLI

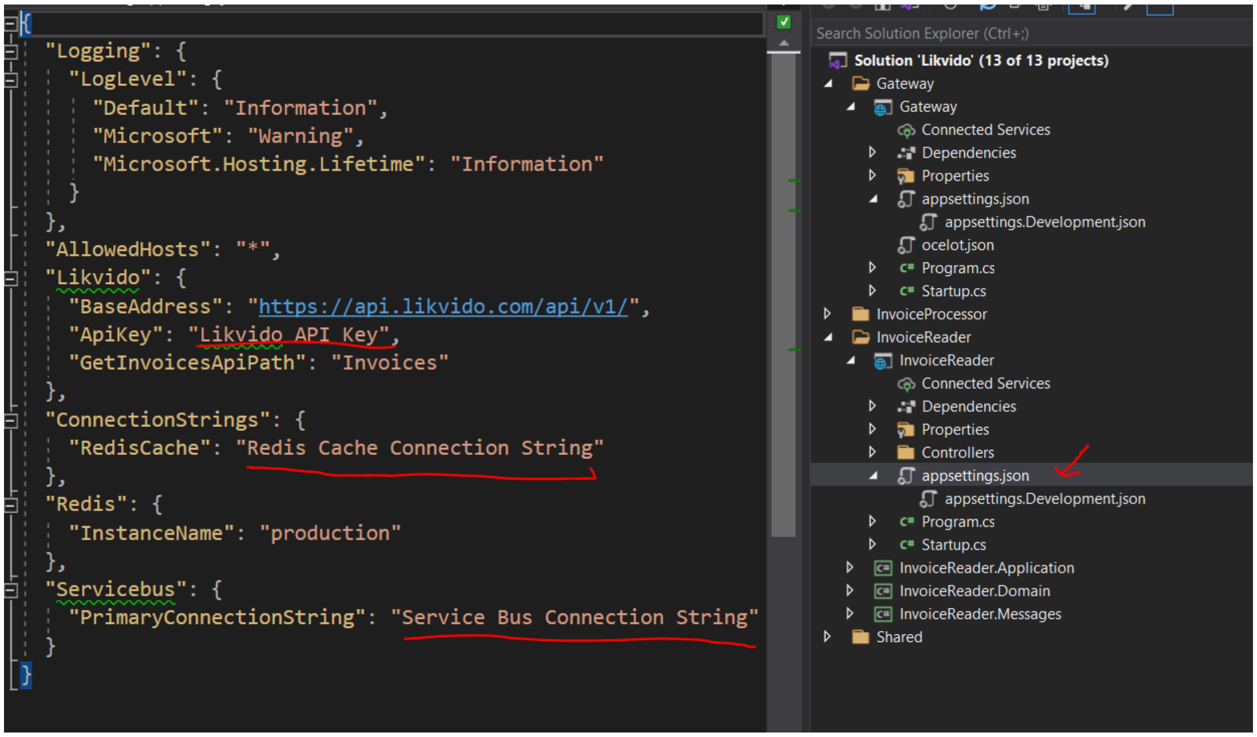
Running Microservies:

1. Open **Likvido.sln** from the root folder.
2. Navigate to the appsetting.json of the InvoiceProcessor project and insert the following data. FYI, in the appsettings.development json file, I have added my test service bus connection string, Likvido API key(Please make sure that is active, previously, one of my accounts was deleted by the Likvido team), and Quartz cron expression. You can generate your corn expression from here.



```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "InvoiceContext": "SQL Server Connection String"
  },
  "Servicebus": {
    "PrimaryConnectionString": "Service Bus Connection String"
  },
  "Likvido": {
    "BaseAddress": "https://api.likvido.com/api/v1/",
    "ApiKey": "Likvido API key",
    "SendInvoiceApiPath": "Invoices"
  },
  "ProcessPendingInvoiceCorn": "Quartz CORN expression"
}
```

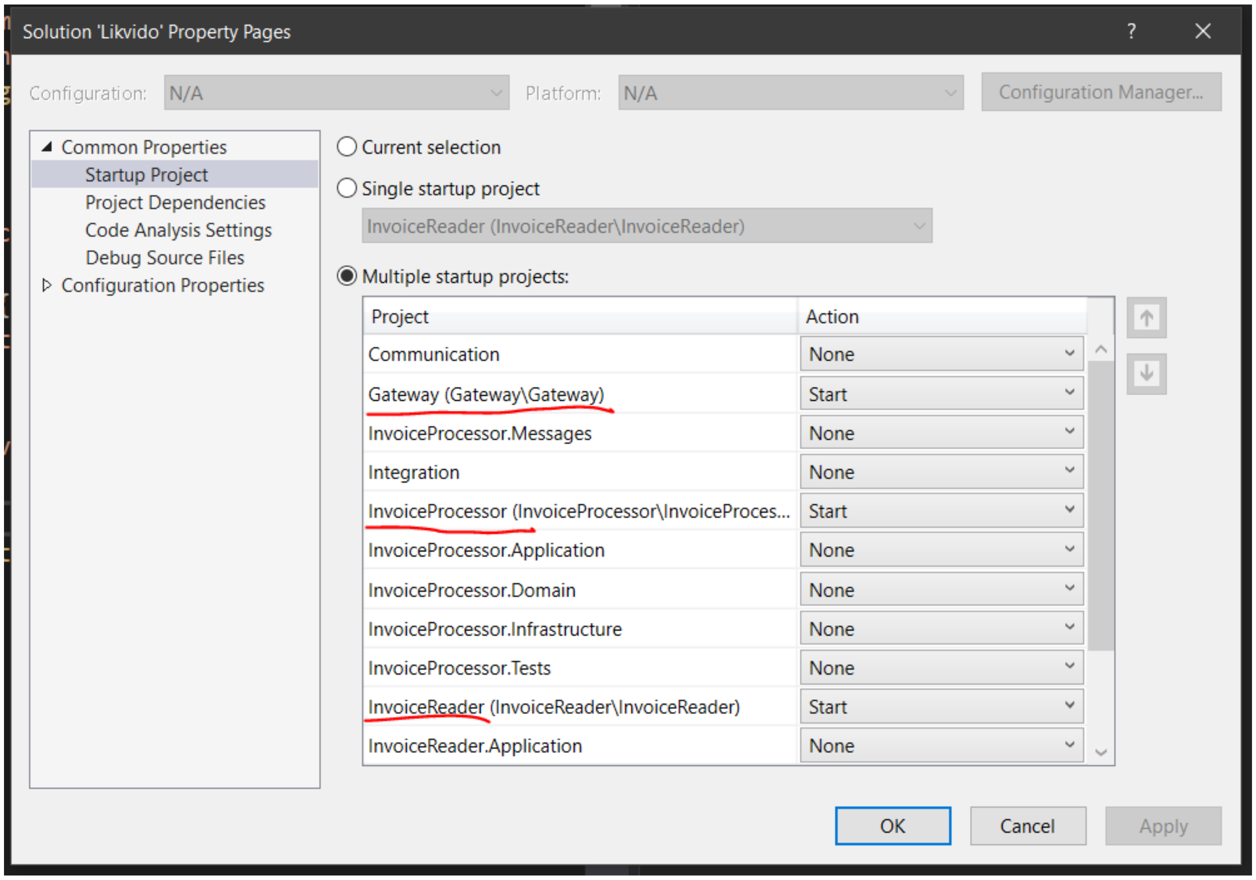
3. Open the appsettings.json of the InvoiceReader project. And provide necessary settings as mentioned below. FYI, you can use the test service bus and Redis connection string that is included inside appsettings.development.json.



```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "Likvido": {
    "BaseAddress": "https://api.likvido.com/api/v1/",
    "ApiKey": "Likvido API Key",
    "GetInvoicesApiPath": "Invoices"
  },
  "ConnectionStrings": {
    "RedisCache": "Redis Cache Connection String"
  },
  "Redis": {
    "InstanceName": "production"
  },
  "Servicebus": {
    "PrimaryConnectionString": "Service Bus Connection String"
  }
}
```

```
Search Solution Explorer (Ctrl+;)
Solution 'Likvido' (13 of 13 projects)
  Gateway
    Gateway
      Connected Services
      Dependencies
      Properties
      appsettings.json
        appsettings.Development.json
      ocelot.json
      Program.cs
      Startup.cs
  InvoiceProcessor
  InvoiceReader
    InvoiceReader
      Connected Services
      Dependencies
      Properties
      Controllers
      appsettings.json
        appsettings.Development.json
      Program.cs
      Startup.cs
    InvoiceReader.Application
    InvoiceReader.Domain
    InvoiceReader.Messages
  Shared
```

4. Set **Gateway, InvoiceReader and InvoiceProcessor** as startup projects.



Solution 'Likvido' Property Pages

Configuration: N/A   Platform: N/A   Configuration Manager...

Common Properties
- Startup Project
- Project Dependencies
- Code Analysis Settings
- Debug Source Files

Configuration Properties

Current selection
Single startup project: InvoiceReader (InvoiceReader\InvoiceReader)
[✓ Multiple startup projects:]

Project	Action
Communication	None
Gateway (Gateway\Gateway)	Start
InvoiceProcessor.Messages	None
Integration	None
InvoiceProcessor (InvoiceProcessor\InvoiceProces...	Start
InvoiceProcessor.Application	None
InvoiceProcessor.Domain	None
InvoiceProcessor.Infrastructure	None
InvoiceProcessor.Tests	None
InvoiceReader (InvoiceReader\InvoiceReader)	Start
InvoiceReader.Application	None

OK   Cancel   Apply

5. Run the application by pressing F5 or from the toolbar.
6. Make sure Invoice Reader is running on 5002 port by visiting http://localhost:5002/swagger/, Invoice Processor running on 5001 port by browsing http://localhost:5001/swagger/ . And, finally Gateway is running on 44300 by visiting https://localhost:44300/swagger/ .

Running Front End

1. Open terminal from the **ClientApp** folder
2. Execute **yarn install** command
3. Run the application by executing **ng serve** command. You can find the application at http://localhost:4200

Possible Improvements

- Remove hard-coded values from the application(Mostly in the client application).
- Add proper validation in invoice creation.
- Create a circuit breaker, when Likvido fails to process requests continuously.
- Better naming and folder structure(mostly in client application).
- Add more unit, integration and E2E test cases.
- Impose better logging.