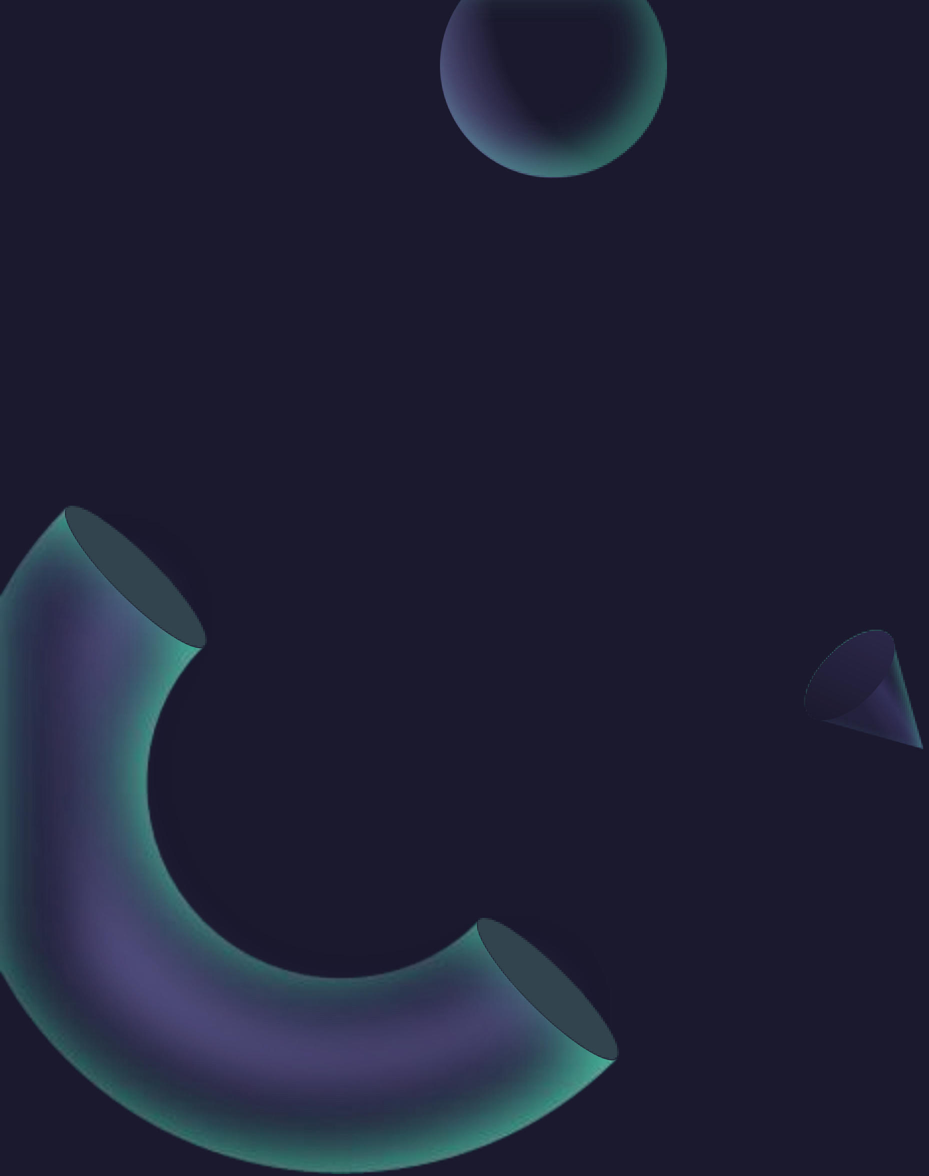

Revolut Integration API Documentation

Project Structure and Architecture CLEAN ARCHITECTURE

API Layer (RevolutIntegration.Api):

- Contains the entry points for API endpoints, including controllers like `AccountController`, `TransactionController`, and `InitiatePaymentController`.
- This layer is responsible for handling HTTP requests and responses, using `IMediator` to handle requests and manage dependencies.
- Uses `Swagger` for API documentation and testing.

Application Layer (RevolutIntegration.Application):

- Implements application logic using **MediatR** to decouple the domain and infrastructure layers.
- Commands and queries are defined here, such as `InitiatePaymentCommand` and `GetAllAccountsQuery`.
- Includes **DTOs** for data transfer between layers and **command/handler classes** to handle business logic.

Domain Layer (RevolutIntegration.Domain):

- Contains domain models and interfaces (e.g., `IPaymentService`, `IOAuthService`, `IAccountService`) which define the core business functionality.
- Domain models like `AccountModel`, `TransactionDto`, and `PaymentModel` represent data structures for accounts, transactions, and payments.

Infrastructure Layer(RevolutIntegration.Infrastructure):

- Handles authentication and authorization via the `OAuthService`, which retrieves and manages access tokens.

Setup

Prerequisites:

- **ASP.NET Core 8.0**
- **Swagger**
- **AutoMapper** for object-to-object mapping
- **MediatR** for CQRS pattern implementation
- **XUnit** and **Moq** for unit testing
- Microsoft.Extensions.Logginf.Abstractions for logging

Configuration

The configuration settings for the Revolut API integration is added to the `appsettings.json`

Error Handling

Error handling is implemented to ensure that the service behaves gracefully in the face of different kinds of failures. The key error types we handle in `TransactionService` are:

-HTTP Errors (`HttpRequestException`):

Cause: This exception is thrown when there are issues related to HTTP requests, such as network errors, unreachable endpoints, or unsuccessful HTTP responses (e.g., 4xx or 5xx status codes).

-Deserialization Errors (`JsonException`):

Cause: This exception occurs when the response content cannot be deserialized into the expected data structure (e.g., `List<TransactionModel>`), usually due to an unexpected format in the API response.

-General Exceptions (`Exception`):

Cause: Catches any unexpected exceptions that may occur, such as null references or logic errors.

Logging

Contextual Information:

Each log entry includes context-specific information such as `accountId`, which helps identify which particular transaction retrieval operation the log refers to.This ensures that logs are easy to trace and diagnose.

Log Format:

Log entries are structured in a clear and informative format, providing both the context (e.g., account ID) and the outcome (e.g., success or error status) of the operation. This makes logs human-readable and aids debugging.

Log Levels:

1. **`LogInformation`**: Logs are created for successful operations and important milestones (e.g., beginning or completing a transaction retrieval).
2. **`LogWarning`**: Logs warning-level events like API failures or unexpected status codes.
3. **`LogError`**: Logs error-level events such as HTTP errors or deserialization issues that prevent the transaction retrieval from completing as expected.

Conclusion

This Revolut Integration API project follows a modular, testable architecture. The layered approach with well-defined services and controllers, combined with comprehensive unit testing, ensures a maintainable codebase. The MediatR setup allows easy extension and modification of business logic, while dependency injection facilitates mock testing, making the project suitable for robust Revolut API integrations in a production environment.