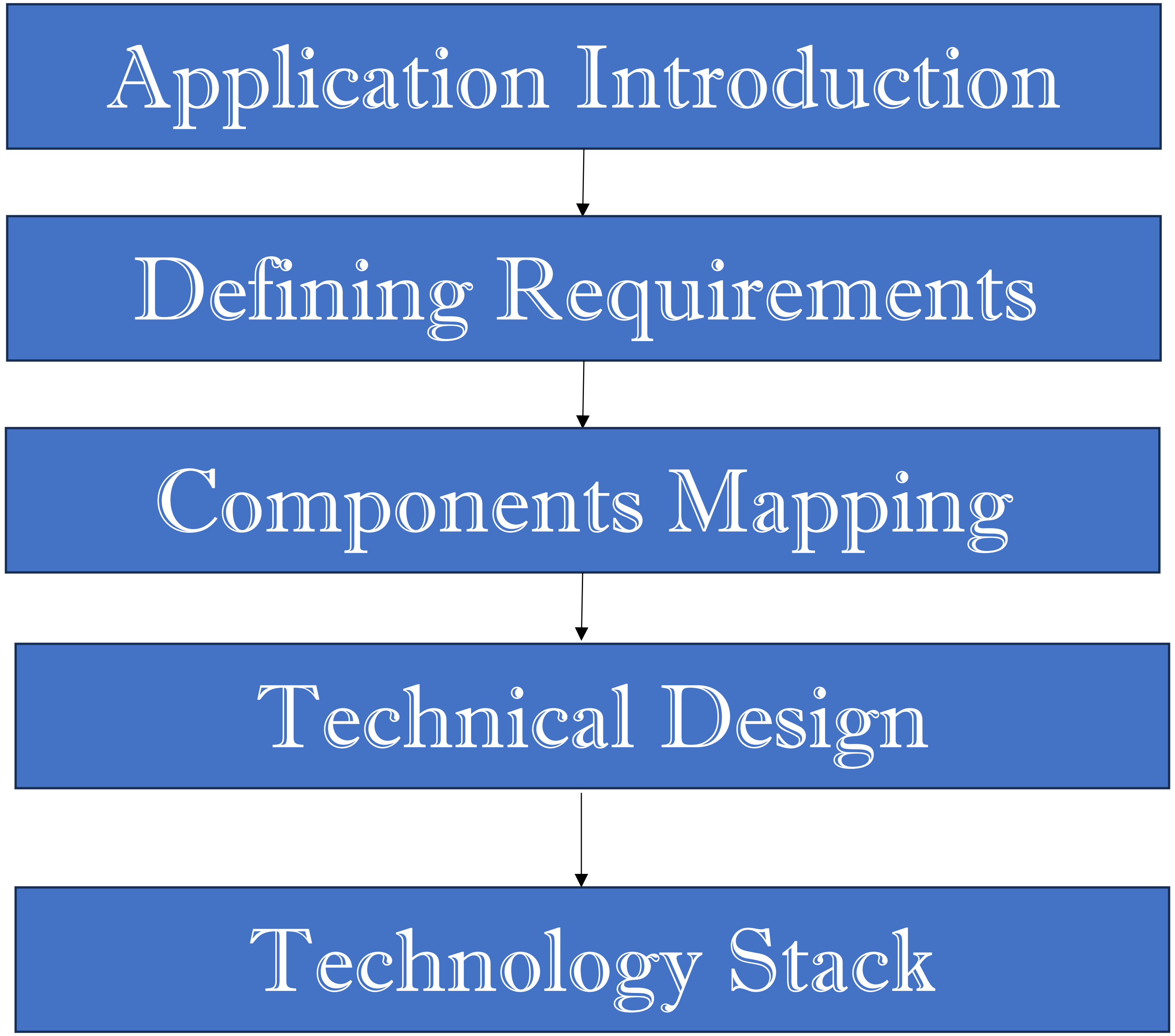
PayPal as payment method

Case Study Overview

- Objective: Integrate PayPal 2.0 as a payment method in the payment gateway.
- Scope: Implement authorization, capture, refund, and reporting functionalities using PayPal SDK.


Application Introduction
Defining Requirements
Components Mapping
Technical Design
Technology Stack

Pay flexibly with PayPal

Image Credit: https://www.paypal.com/ca/business/resources/logos-and-banners

PayPal

- **Secure Payment method**
- **With over 173 million users globally and operates in 202 countries**
- **Support in 21 different currencies**
- **Zero Subscription Fee**
- **Secure Transfers**
- **Convenient to use**
- **Hassle-free Subscription Services**

Requirements

Functional

What the system should do

- Payment (2 Step integration)
  - Payment Authorization
  - Payment Capture
- Payment – Fastlane required?
- Refund Processing
- Transaction Logging
- Transaction Reporting

Non Functional

What the system should deal with

Non Functional

How many concurrent users?	At least 200 per minute
How many refunds will be processed Per day?	At least 50
How much transaction throughput?	At least 500 per minute
What is the target payment authorization response time?	< 2 Seconds
What is the required system availability?	99.99%
How quickly should refunds be processed?	< 1 hour
Which transactions need to be logged?	All transactions
How many years these data has to be stored?	As per GDPR

…..contd

Non Functional

How scalable should the system be?	Scalable to handle a **50% increase** in peak transaction load
What is the acceptable error rate for transactions?	< 0.5%
Which security standards must be followed?	must comply with **PCI DSS** and implement **SSL/TLS encryption**
When should data backups be performed?	**at least 30 days**.
What is the disaster recovery objective?	recovery time objective (RTO) of **less than 1 hour**.

Client’s Current Payment Landscape:

Assuming Client’s payment gateway has following:

- Handling various payment methods including credit cards, debit cards, Internet Banking

- Has API interaction with VISA, MasterCard and other payment interfaces for direct debit from bank account.

- Has custom data management to store transaction history

- Fraud Detection

- Reporting of transaction for any metric analysis

- Cancellation Page when payment is cancelled or authorization is denied by payment provider.

Client’s Current Payment Landscape:

Assuming Client’s payment gateway does not have:

- Buy Now Pay Later option
- Support for international transactions i.e. only domestic credit/debit cards are accepted.
- Support of any specific card payment network like American Express, Discover etc.

PayPal Integration Service - Purpose

The purpose of developing the PayPal Integration Service is to securely and efficiently integrate PayPal as a payment method within the client’s existing payment gateway, allowing the client to offer PayPal as an additional payment option. This service manages the entire payment lifecycle—authorizing, capturing, and refunding payments—through PayPal's API, ensuring seamless operation within the client’s gateway, compliance with security standards, and real-time transaction updates for an enhanced customer and merchant experience.

<u>PayPal handles the existing client’s challenges by offering</u>

Buy Now, Pay Later Option using “Pay Later” or “Pay after 30 days” feature
Support for International Transactions and
    Enables multi-currency support
    Expands global market reach
Support for Specific Card Networks
    Supports American Express, Discover and all other world renowned payment networks.
    Broadens payment options for customers

Monolithic vs. Microservices

A Comparative Analysis for PayPal Integration

Pros

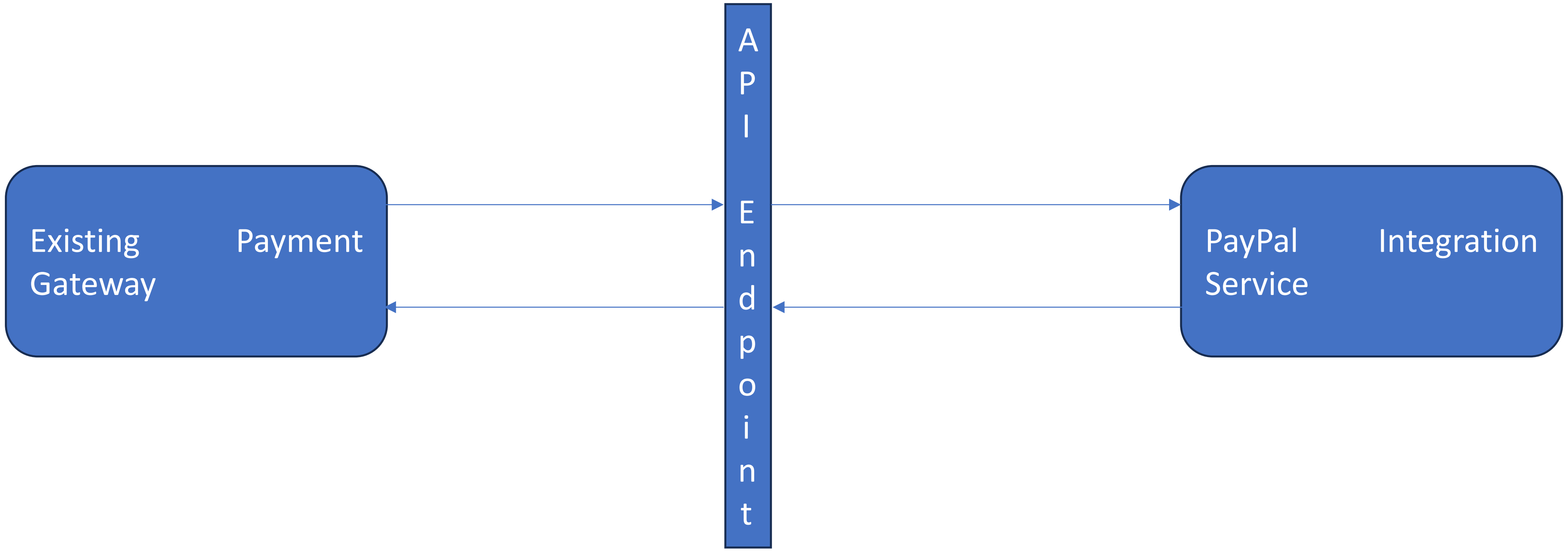
Criteria	Monolithic Architecture	Microservices Architecture
Functional Requirements	- Unified codebase simplifies management of interdependencies	- Modularity allows for clear separation of concerns
	- Easier and faster development of features	- Independent development and deployment of services
	- Consistent data management across the application	- Flexibility to use different technologies for each service
	- Simpler testing as all components are in one place	- Easier to scale specific functions (e.g., payment processing)
Non-Functional Requirements	- Better performance due to in-process communication	- Superior scalability with independent scaling of services
	- Simpler and quicker deployment process	- Enhanced resilience; failure in one service doesn’t impact others
	- Easier to maintain data consistency with a single database	- Fault isolation minimizes risk of cascading failures
	- Lower operational overhead as everything is in one application	- Flexibility and adaptability to changing business requirements
	- Easier logging and monitoring in a single application	- Better suited for continuous delivery and frequent updates

Cons

Criteria	Monolithic Architecture	Microservices Architecture
Scalability	Difficult to scale individual components independently	Increased complexity in managing multiple services
Maintenance	Codebase can become large and complex over time	Requires robust DevOps practices and higher operational overhead
Deployment	Even small changes require redeploying the entire application	More complex deployment processes due to multiple services
Flexibility	Limited flexibility in adopting new technologies due to tight coupling	Challenges in maintaining data consistency across services
Reliability	A failure in one part can bring down the entire system	Network latency due to inter-service communication
Testing	Simpler but affects entire system if issues arise	More complex testing processes for individual services

Integration of an Instant Payment Solution into the Client's IT Payment Landscape

PayPal Integration Service can be developed separately as microservice and the Integration API can be called from the existing gateway. If client payment gateway is monolithic, this service can be integrated as hybrid architecture.



Hybrid (if monolithic) or Microservices Architecture

Hybrid architecture helpful

- Incremental Migration and modernization.
- Flexibility
- Improved Scalability
- Isolation
- Improved Maintenance

Scope of Work:

1.Obtain and Manage OAuth 2.0 Access Tokens:

Securely obtain and manage OAuth 2.0 access tokens from PayPal to authenticate API requests, ensuring all communication with PayPal's services is authorized.

2.Payment Authorization:

Send requests to PayPal to verify and hold funds on the customer's payment method, ensuring that the payment can be captured later.

3.Capture Authorized Payments:

Finalize the payment by capturing the authorized funds, transferring the amount from the customer's account to the merchant's account.

4.Refund Processing:

Handle refund requests by interacting with PayPal to return funds to the customer for previously captured payments.

5.Transaction Logging and Auditing:

Record all transaction details and events for future reference, ensuring that every payment and refund is auditable.

6.Validation, Error Handling and Retry Logic:

Implement mechanisms to gracefully handle errors during API interactions and automatically retry operations when transient issues occur.

7. Security and Compliance

Ensure that all processes and data handling adhere to security standards and compliance requirements, such as PCI DSS, to protect sensitive payment information.

8. API Rate Limiting and Idempotency:

Monitor and manage API request rates to avoid hitting PayPal's limits, and implement idempotency to prevent duplicate transactions.

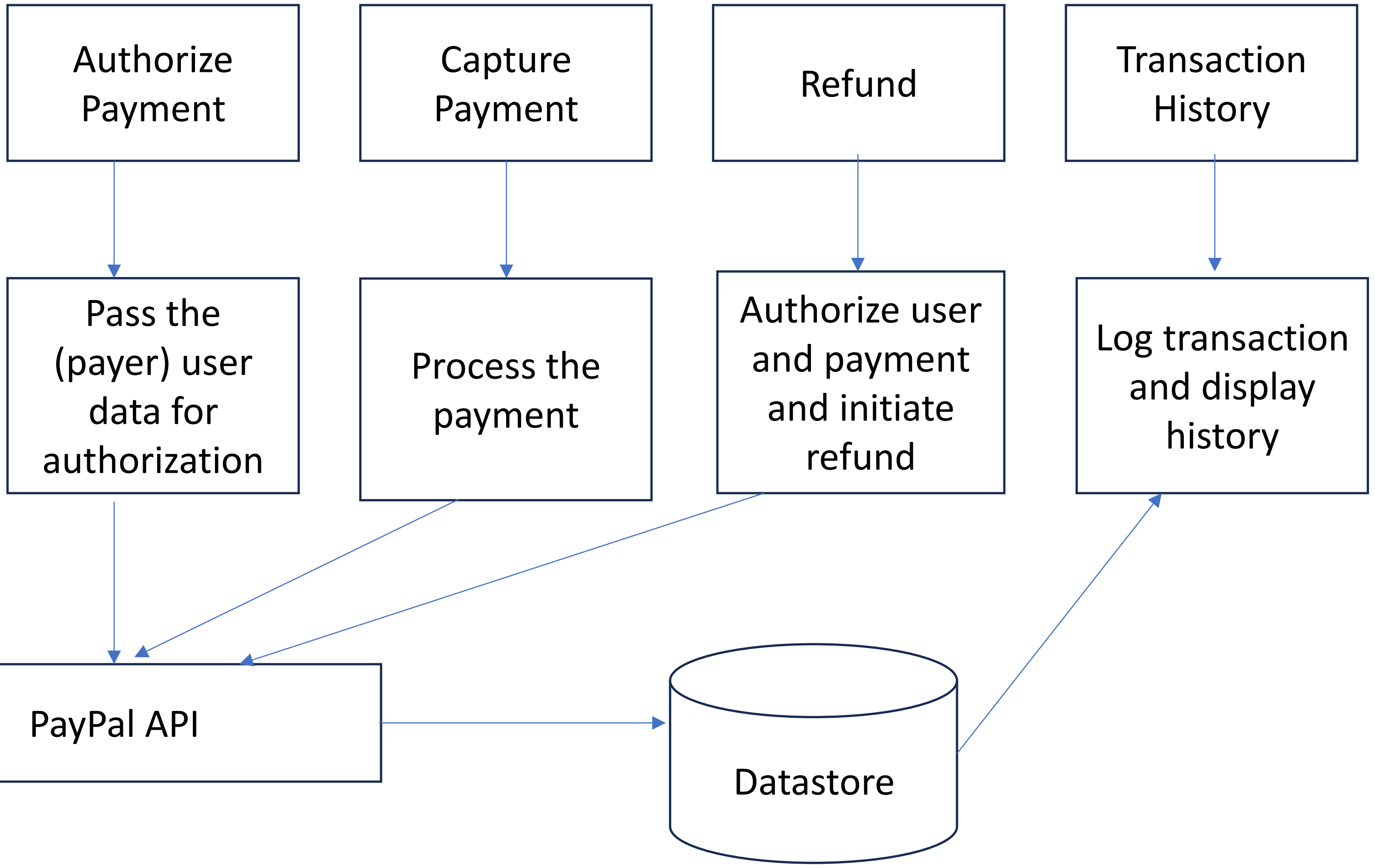
9. **Webhook Integration:**

Set up webhook listeners to receive real-time notifications from PayPal about payment status updates, disputes, and other important events

PayPal Integration API - Key Components

Based on requirements:

1. Authorize Payment
2. Capture Payment
3. Refund Processing
4. Transaction History Report


Authorize Payment
Capture Payment
Refund
Transaction History
Pass the (payer) user data for authorization
Process the payment
Authorize user and payment and initiate refund
Log transaction and display history
PayPal API
Datastore

Authorization Service: What It Does

Request:

•Sends user data and payment amount to the PayPal API for authorization.

•Verifies buyer's funds and reserves the amount for debit.

Response:

•Returns an authorization status: Approved or Denied, along with an authorization ID.

•Provides a response message:

- **Success:** User is authorized, and sufficient funds are available.
- **Error:** User is not authorized due to insufficient funds, or no card/account is linked.

What It Doesn't Do:

•Does not capture the payment.

•Does not persist any information.

What it does?

Payment Service:

1. Request:

Send the authorization data fetched from authorization Service to PayPal API

2. Response

    Transaction Status from payment provider

    Status can be

        Payment success

        Payment failed

        Payment Process Cancelled by the user.

3. Persists the transaction information to the persistence unit

What it does?

Refund Service:

1. Request:
   - Sends transaction data such as ID, refund amount, reason (optional)
2. Response
   - Status of refund processing with message

3. Persists the transaction information

What it does?

Transaction reporting Service:

1. Fetches the transaction data based on criteria

   Ex –

   By Geographical location

   By Date and time

   By User

   By type (refund or normal payment)

   etc

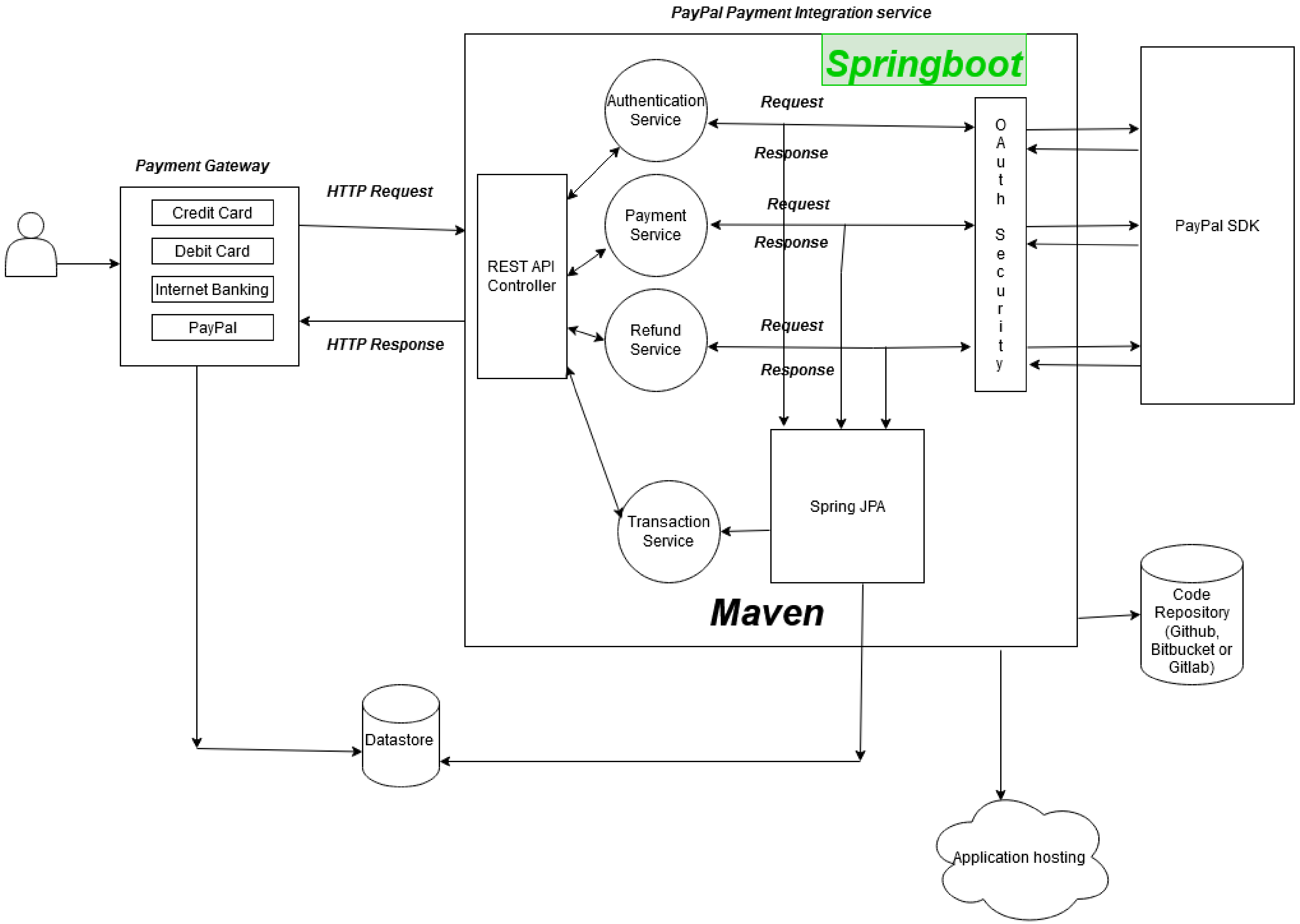
Application Type

• Web App & Web API	✓
• Mobile App	✗
• Console	✗
• Service	✓
• Desktop App	✗

Technical Design

Considerations:

- **Should be able to connect to PayPal SDK**
- **Should be able to communicate to the client payment gateway services.**


PayPal Payment Integration service
Springboot
Payment Gateway
Credit Card
Debit Card
Internet Banking
PayPal
HTTP Request
HTTP Response
REST API Controller
Authentication Service
Payment Service
Refund Service
Transaction Service
Request
Response
Request
Response
Request
Response
OAuth Security
PayPal SDK
Spring JPA
Maven
Code Repository (Github, Bitbucket or Gitlab)
Datastore
Application hosting

Technology Stack

Java

spring

PostgreSQL

PayPal

VISA MasterCard AMERICAN EXPRESS DISCOVER NETWORK

Maven

Elasticsearch + Logstash + Kibana

Technology Stack

- **Business Layer Implementation (Java, Spring Boot, Rest API)**
- **Dependency Management - Maven**
- **Security - OAuth2**
- **Payment Gateway Integration: Paypal SDK**

Technology Stack

- Testing – Spring Test, Mockito
- Messaging/Asynchronous Processing - Apache Kafka, RabbitMQ
- Monitoring and Logging – ELK Stack for distributed environment
- Deployment – Based on existing Client application (explained in later slide)
- Infrastructure – Based on existing client application (explained later)
- Versioning – Same version management as client's application

Technology Stack

What about persisting the transaction history?

Technology Stack

What about persisting the transaction history?

The service can utilise the transaction history storage unit of the Client’s payment gateway.

This helps in
- Reducing inconsistency
- Data duplication
- Simplified data management
- Ease of Integration

Technology Stack

What about persisting the transaction history?

The service can utilise the transaction history storage unit of the Client’s payment gateway.

Alternatives:
Relational Databases – PostGRESQL
NoSQL - MongoDb

DevOps Alternatives

Scenario	Existing Client Application	New Payment Service as Microservice (Alternatives)
Infrastructure	On-premises servers or VMs running the monolithic application	Deploy the new service as a separate VM Or Containerized environment (Docker) Or Cloud-based infrastructure Or Use cloud VMs like EC2 integrated with on-premises VMs
	Containerized environment (Docker) Or Cloud-based infrastructure	Similar infrastructure extending the existing infrastructure

DevOps Alternatives

Scenario	Existing Client Application	New Payment Service as Microservice (Alternatives)
Deployment	Deployed on-premises with traditional CI/CD pipelines	- CI/CD pipelines for containerized deployment (e.g., Jenkins, GitLab CI)
	Deployed in VMs with manual updates	- Cloud-based CI/CD (AWS CodePipeline, Azure DevOps) - Automated deployments with Kubernetes Helm or Docker Compose - Serverless deployment (AWS Lambda with automated deployments)

DevOps Alternatives

Scenario	Existing Client Application	New Payment Service as Microservice (Alternatives)
Load Balancing	On-premises load balancers	Cloud-based load balancers (AWS ELB, Azure Load Balancer) Or Kubernetes Ingress or Service Mesh for internal load balancing Or API Gateway for external load balancing and routing Or Hybrid load balancing using a combination of on-prem and cloud
	Cloud Based Or Kubernetes	Extending the similar load balancing for the new service

DevOps Alternatives

Scenario	Existing Client Application	New Payment Service as Microservice (Alternatives)
Scaling	On-premises servers or VMs with manual scaling	Auto-scaling with Kubernetes or Docker Swarm Or Cloud auto-scaling (AWS Auto Scaling, Azure Scale Sets Or Serverless auto-scaling (e.g., AWS Lambda scales automatically)
	Cloud Based Or Containerized environment with manual scaling	Horizontal scaling by adding more containers Or Hybrid approach using cloud burst for peak demand

References:

https://developer.paypal.com/docs/checkout/standard/customize/authorization/

https://developer.paypal.com/docs/api/payments/v2/

https://spring.io/projects/spring-restdocs

https://medium.com/@umasree.kollu/building-payment-functionality-with-spring-boot-step-by-step-guide-8f2210101c5c

https://kennybrast.medium.com/how-to-explain-hybrid-devops-to-the-c-suite-without-getting-fired-95541673c023