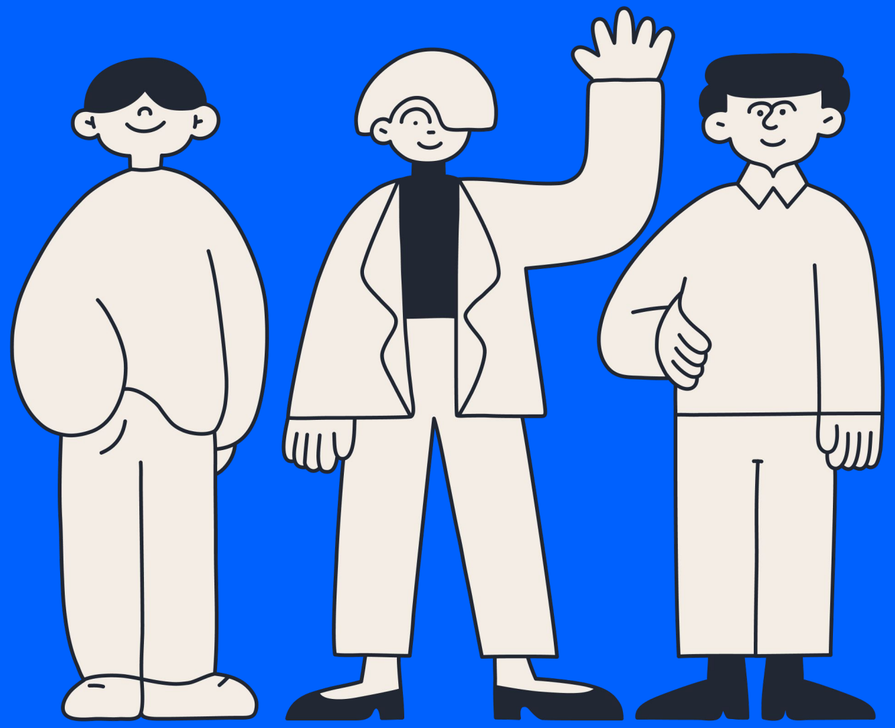
ITE-SPA-06
Integration and Deployment

“Bridging classic enterprise concepts with modern cloud-native practices”



OSOME Havit C. Rovik | Universitas ‘Aisyiyah Yogyakarta (UNISA) | 24.06.2025

Today Agenda

1. Introduction: Why Modern Integration Matters
2. Middleware Platforms in AWS
3. Choosing the Right Integration Platform
4. Wrappers vs Glue Code Integration
5. Frameworks for Integration
6. Enterprise Data Warehouse & AWS
7. Testing & Evaluation in Cloud Integrations
8. A Glimpse of Deployment
9. Summary

Havit C. Rovik

Senior Software Engineer @OSOME

- ❏ 1st Winner BRI x Tech in Asia FutureMakers (2024)
- ❏ 1st Winner OCBC NISP Hack@ON (2022)
- ❏ 1st Winner BCA Finhacks #Codescape (2017)

- Facilitator at “AI-Volution: Machine Learning for Social Impact” (2025)
- Spoke at “The State of AI in Fintech and the Digital Economy” (2025)
- Graduate of the Amplifier Program by 10x1000 Tech for Inclusion (2024)
- Grand Finalist at Mitrais Innovathon (2021)
- Spoke at “Founder Session: React Native to Handle 3 Platforms” (2020)
- Mentor at DILo Developer Class Season 7 “React Native” (2019)
- Author at Level Up Coding (2019)
- Participant at Hacktoberfest (2018 - 2023)
- Contributor at React Native Elements (2018 - 2022)

Why Talk About Integration & Amazon Web Service (AWS) ?

Enterprises rely on distributed systems

In today’s tech landscape,

no enterprise system exists in isolation.

Applications are built with microservices,

SaaS platforms, legacy components, and third-party integrations.

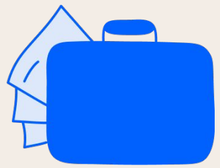
These components are **distributed**

across locations, teams, cloud providers, and data centers.

Middleware and integration are key to system interoperability

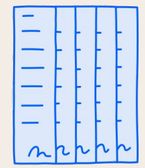
Middleware acts as the “**glue**”

that connects these distributed components.



It enables **interoperability**,

message passing, data transformation, and workflow orchestration.



Classical examples:

ESBs (Enterprise Service Buses), message queues, RPC, etc.

Timeless Concepts

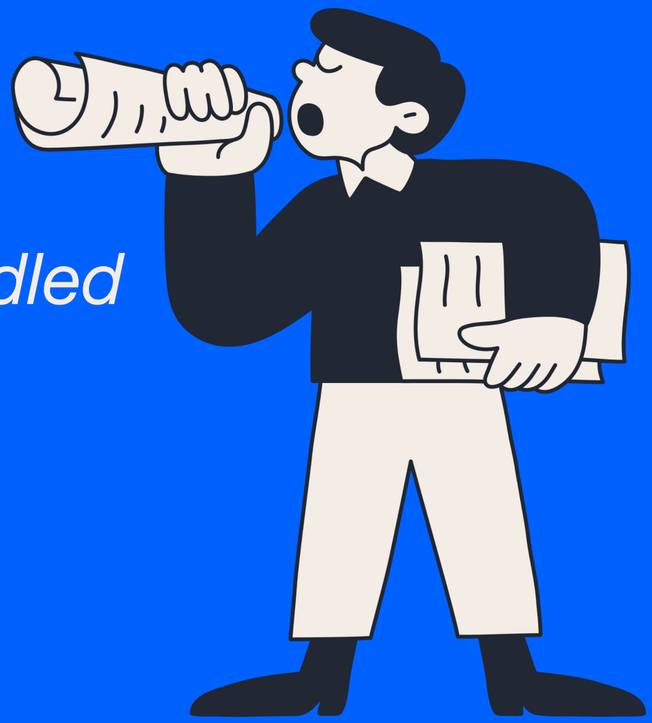
Today's goal

“Translate those classroom concepts *into* modern, AWS-native implementations”

OSOME

*“So **integration** isn’t just a tech buzzword. It’s the **foundation** of how modern systems stay connected and responsive.*

*Let’s now explore how **middleware** platforms are handled in the **AWS** ecosystem.”*



Middleware Platforms in AWS

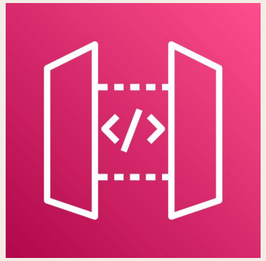
📘 What is it?

Middleware is software that acts as a **bridge** between different applications, services, or systems.

“It is helping them communicate, exchange data, and work together effectively.”

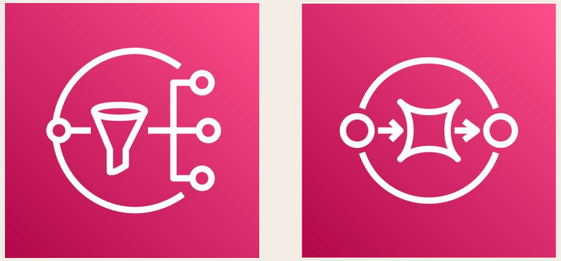
offers **API** & **Event-driven** tools as modern **middleware**.

AWS has **redefined** middleware through managed services



API Gateway

for **synchronous** REST/HTTP-based communication



SNS/SQS

for **asynchronous** event-based communication

Lambda

for orchestration and observability

📘 What is an API?

API stands for **Application Programming Interface**.

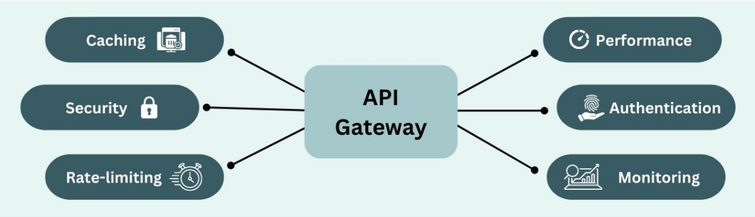
“It is a set of rules and definitions that allows software applications to communicate with each other.”

In the context of AWS and middleware,

APIs are used to:

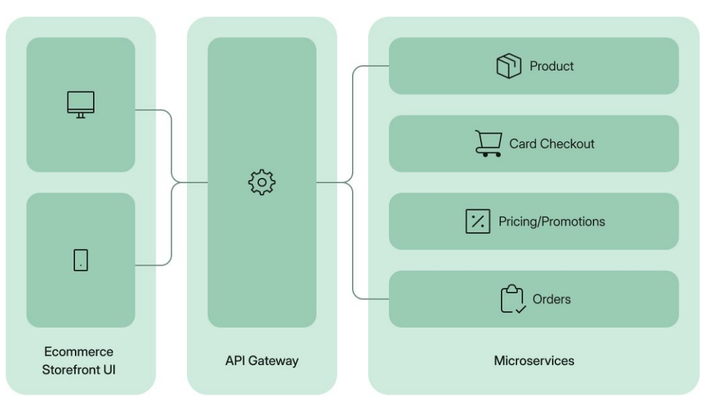
Expose backend services

- e.g., via Amazon API Gateway



Enable integration

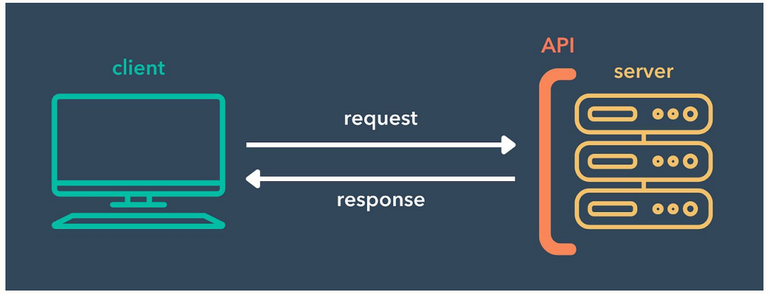
- between microservices



Allow external systems

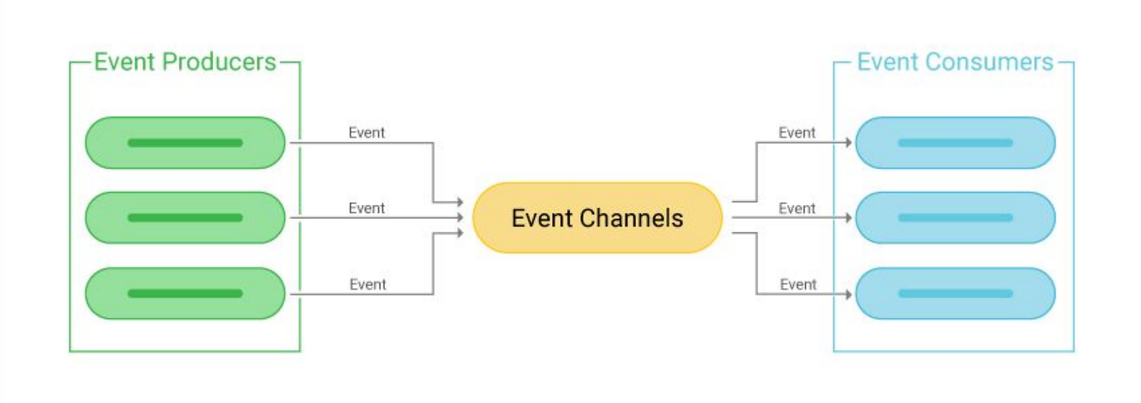
(like web/mobile apps)

- to access our cloud-based logic and data



🔄 What is Event-Driven Architecture?

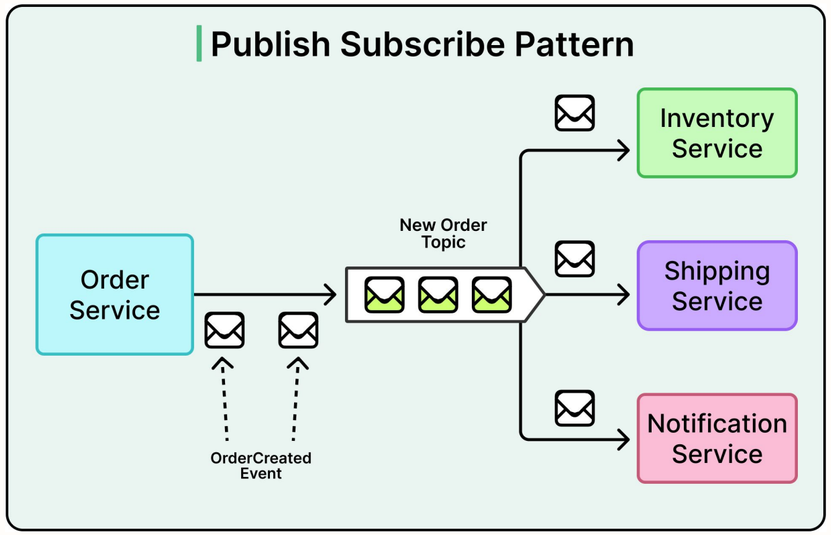
Event-driven architecture (EDA) is a design paradigm where services communicate by **producing** and **consuming** events, rather than through *direct* API calls.



AWS Event-Driven Middleware Tools

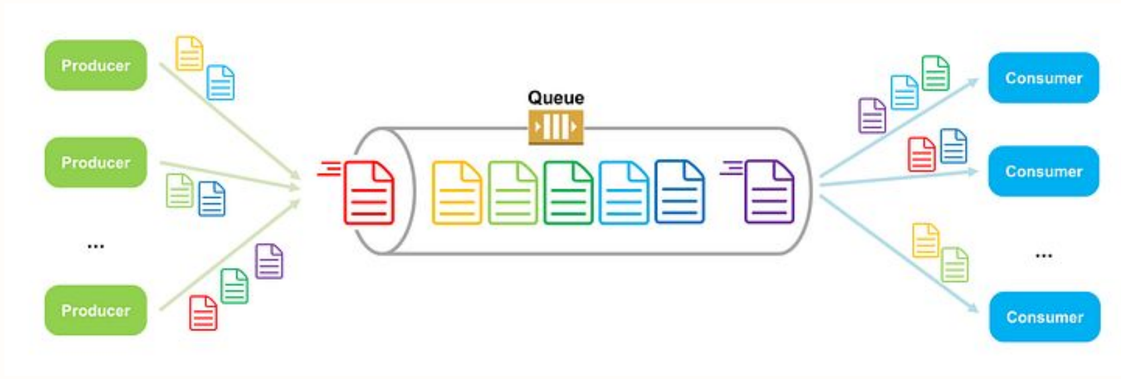
Simple Notification Service (SNS)

- **Pub/Sub messaging** for fan-out event delivery
- Good for broadcasting events to multiple consumers



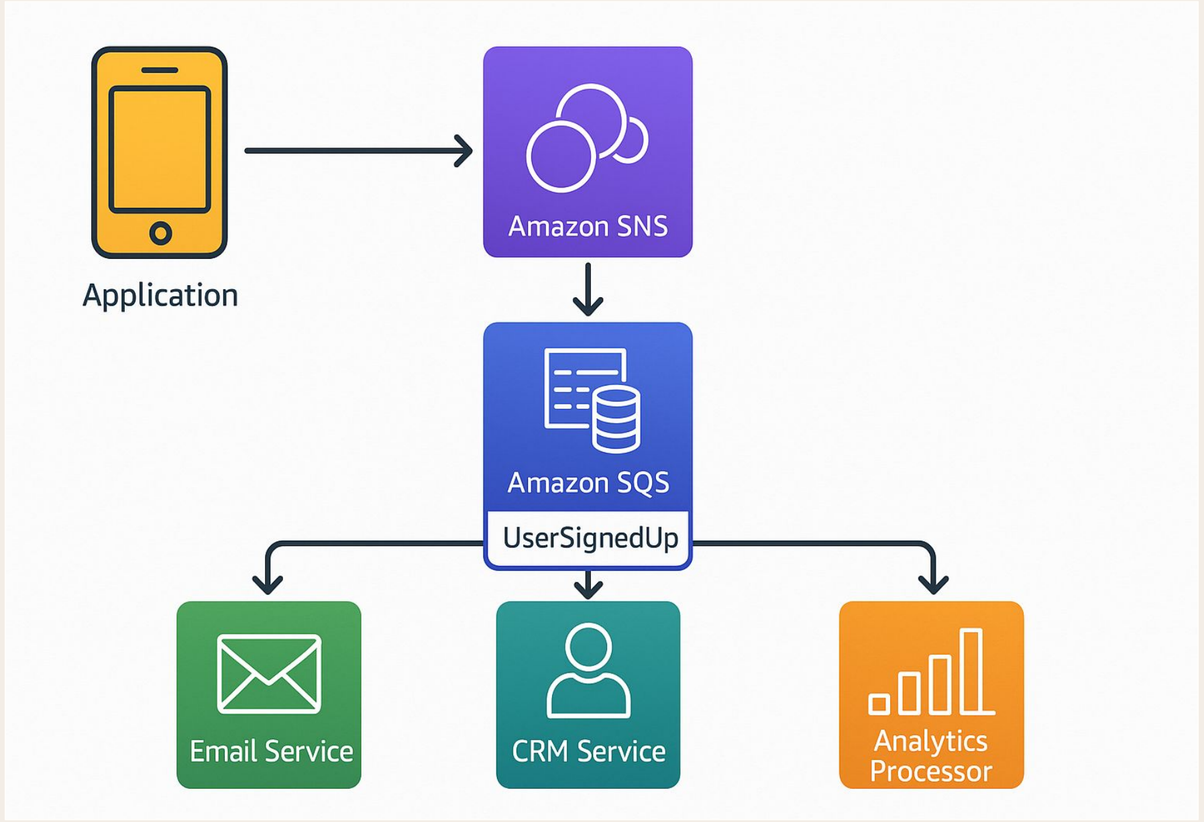
Simple Queue Service (SQS)

- **Message queue** for decoupling producers and consumers
- Ensures reliable delivery and retries



`UserSignedUp` Event Flow

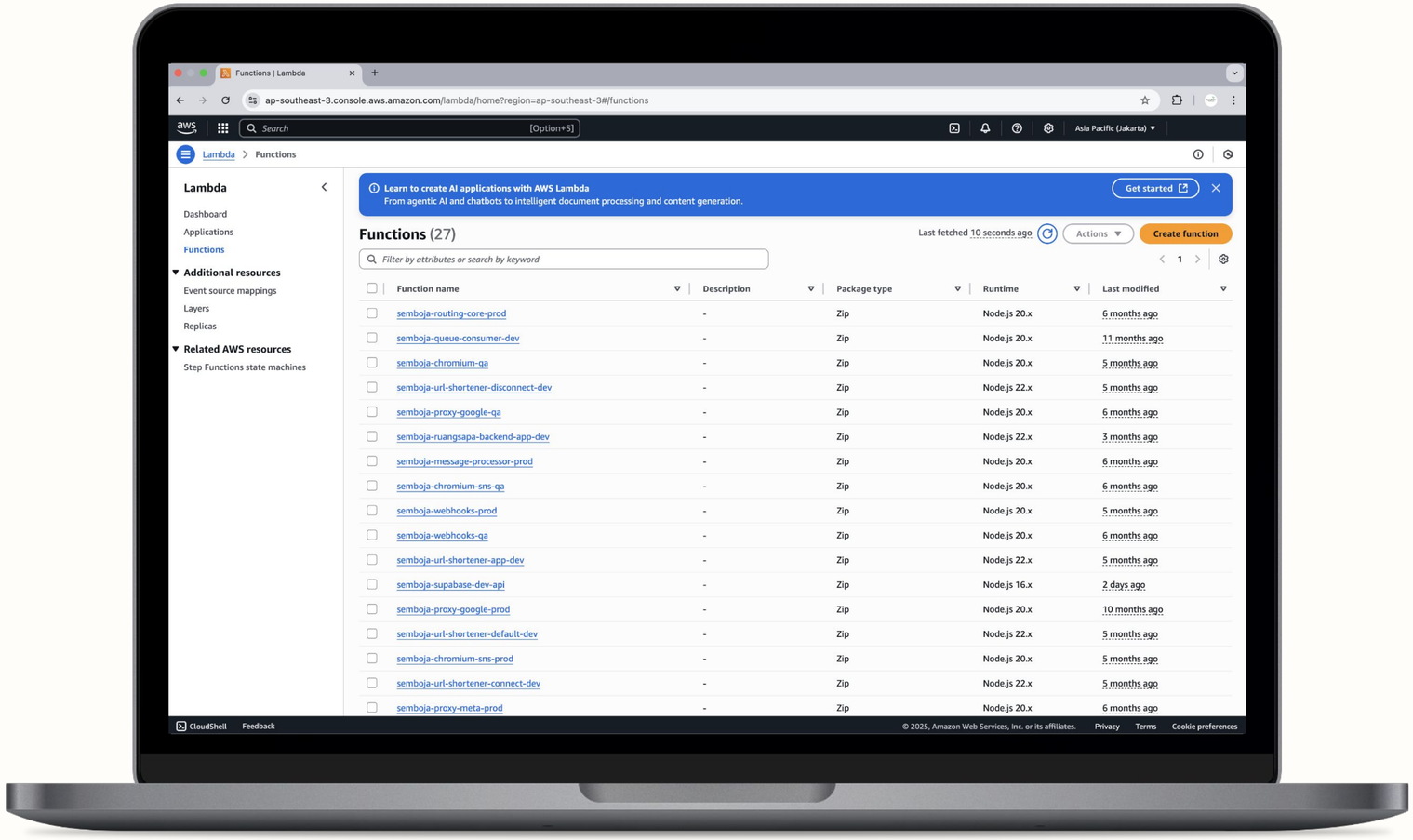
1. A user signs up
   - app publishes a `UserSignedUp` event
2. Several services react:
   - a Lambda function sends a welcome email
   - another Lambda function creates a profile in the CRM
   - an analytics service logs the event for BI



Demo

AWS Dashboard.

AWS API Gateway, SNS, SQS and Lambda.



API Gateway

“It replaces the traditional API middleware layer. It’s the front door to our services: **secured**, **scaled**, and **managed**.”

SNS

“Simple Notification Service is useful when one event needs to **notify multiple systems** at once.”

SQS

“Simple Queue Service helps **absorb traffic spikes** and ensures that even if a backend service fails, no data is lost.”

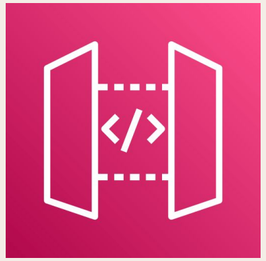
OSOME

*“So, whether we’re **integrating** components **synchronously** via REST or **asynchronously** via events and queues, AWS provides specialized **middleware** services for each.*

*In the next slide, we’ll compare their **trade-offs** and help we choose the right one for our use case.”*

Comparing Middleware Platforms

API Gateway



✅ Works well for synchronous REST requests

✅ Offers rate limiting, API keys, CORS, caching

❌ Not ideal for background or long-running tasks

“Use this when clients need immediate feedback. Like a web or mobile app calling a backend service.”

Simple Notification Service

✅ Great for real-time, broadcast-like scenarios

✅ Simple to integrate with Lambda, HTTP, SQS

❌ Limited filtering, cannot store messages

“SNS is excellent for sending alerts or events to multiple systems, but it's not suitable if we need durable message storage.”

Simple Queue Service

✅ Designed for reliability and decoupling

✅ FIFO and DLQ support add robustness

❌ Requires polling, and there’s no real-time push

“SQS is our go-to for background jobs or when a consumer needs to process messages at its own pace.”

OSOME

*“There’s **no one-size-fits-all** middleware. Our choice depends on factors like latency tolerance, reliability, fan-out needs, and system coupling.*

*Let’s now look at how we **choose the right platform** for our enterprise.”*

Selection Considerations

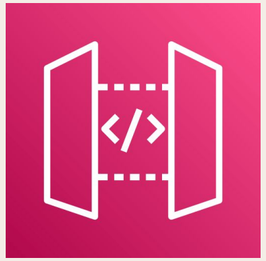
Scalability

SNS **auto-scale** without pre-provisioning

SQS also scales, *but* consumer scaling may be manual (unless paired with Lambda)

"For **unpredictable spikes** in demand, event-based services like SNS shine."

Latency Requirements

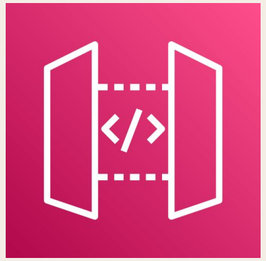


API Gateway provides **low-latency** REST responses

SQS introduce **some delay** due to async nature

“When the user needs an immediate response, stick with APIs.”

Coupling Strategy



Tightly coupled

API Gateway ties services directly together

Loosely coupled

Events via SNS don’t require upstream-downstream awareness

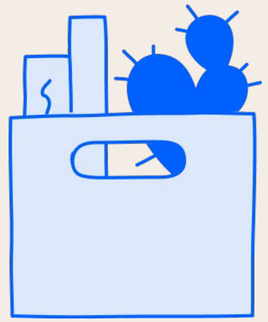
“The more decoupled our architecture, the more resilient and flexible it becomes.”

Resilience & Reliability

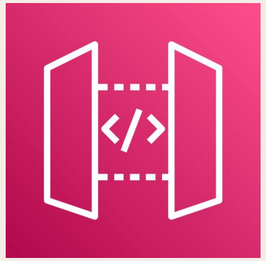
Use **SQS with DLQ** (Dead Letter Queue) to avoid message loss

Ensure **idempotency** in Lambda functions to handle retries gracefully

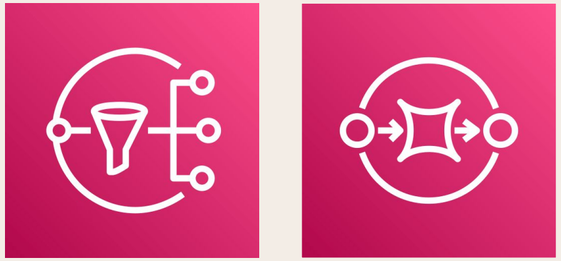
“Failures will happen. What matters is how gracefully our system recovers.”



Cost & Complexity



API Gateway is **cost-efficient** for low-volume APIs



SNS + SQS can become **complex** and **costly** if overused

"Start simple. Only add complexity when we really need the flexibility."

OSOME

*“These are the same criteria used in **enterprise architecture**, only now they’re applied using cloud-native AWS services.*

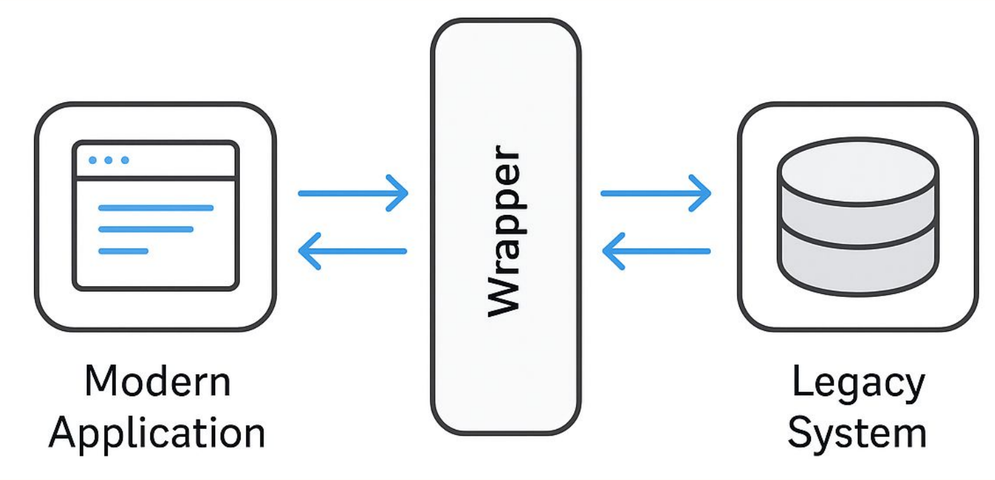
*Next, we’ll look at how integration can be done using the **wrapper** approach.”*

Integration Using the “**Wrapper**” Approach

📘 What is a Wrapper?

A wrapper is a **thin interface** placed around a legacy system or incompatible module.

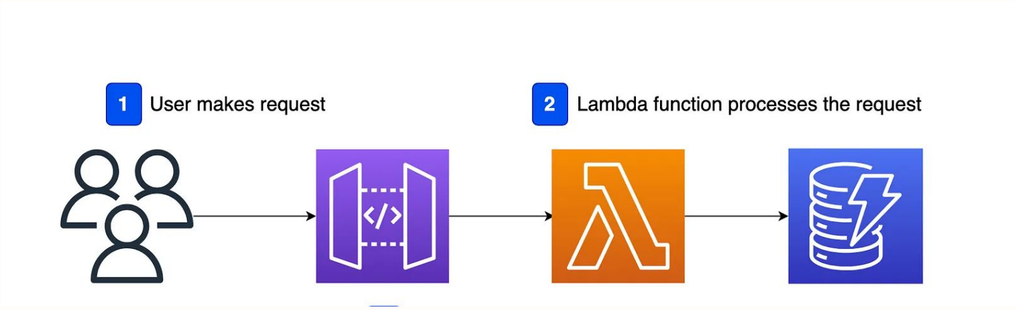
*“It doesn’t change the underlying system but translates **requests/responses**.”*



Common Wrapper Use Cases in AWS

API Gateway + Lambda as a Wrapper

- Wraps any backend, old or new, as a RESTful API
- Secure, scalable, and easy to expose to clients



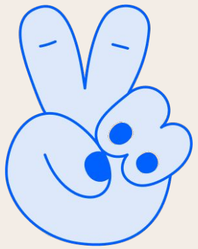
Transform SOAP/XML into REST/JSON

- Use Lambda functions to convert XML input/output into modern JSON APIs
- Useful when integrating old ERP, CRM, or banking systems

Benefits of the Wrapper Approach

Non-invasive

No need to rewrite legacy code



Compatibility

Makes old systems cloud- or API-compatible

Flexibility

Acts as a stepping stone to full modernization

OSOME

*“By using the **wrapper** approach on AWS, we bring legacy systems into **modern** cloud architectures. Paving the way for gradual digital transformation.*

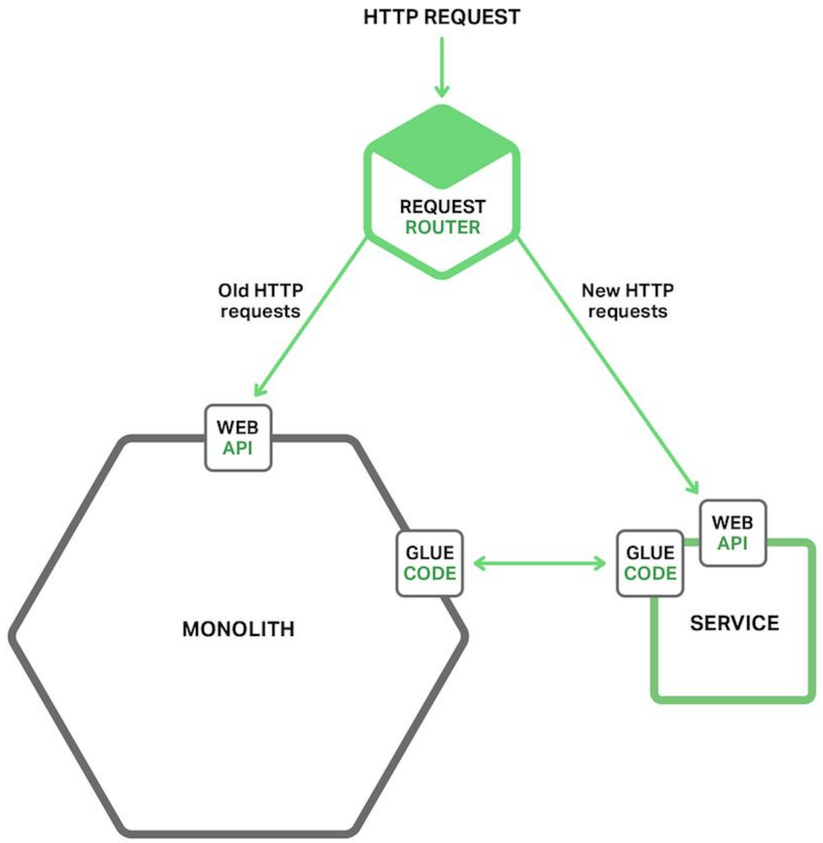
*Next, we’ll explore the **glue code** approach, which complements this strategy in different scenarios.”*

Integration Using the “**Glue Code**” Approach

📘 What is Glue Code?

It is **custom scripting** or **programming** that connects different services, formats, or APIs.

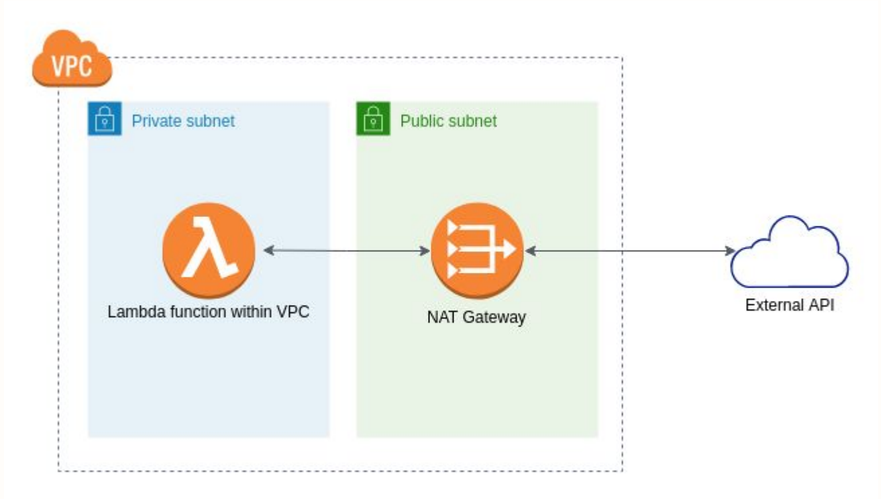
*“It often handles **transformations**, **coordination**, or **data reshaping**.”*

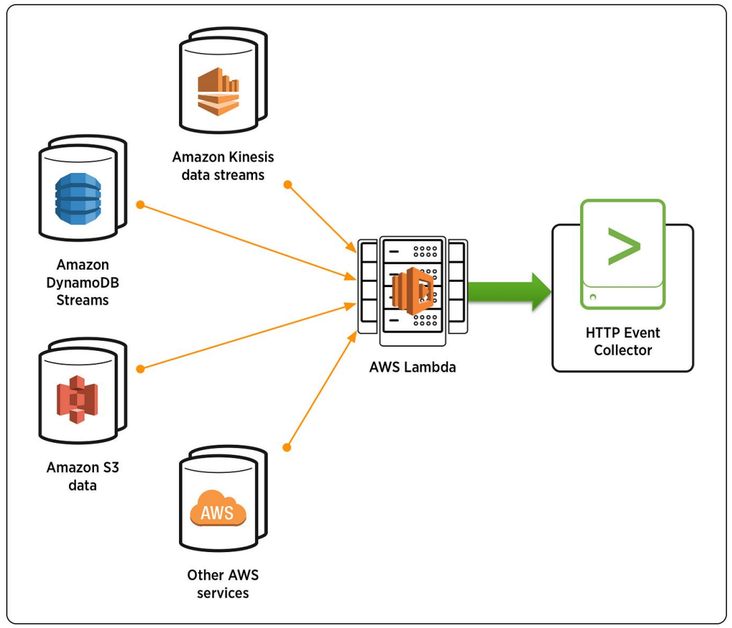


Example of Glue Code in AWS

Lambda as Transformers or Brokers

- E.g., receive an event, transform it, and call an external API
- Translate between systems with different data formats or contracts

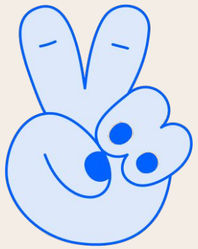




Benefits of the Glue Code

Tailored

to exact needs



Enables integration

where no standard option fits

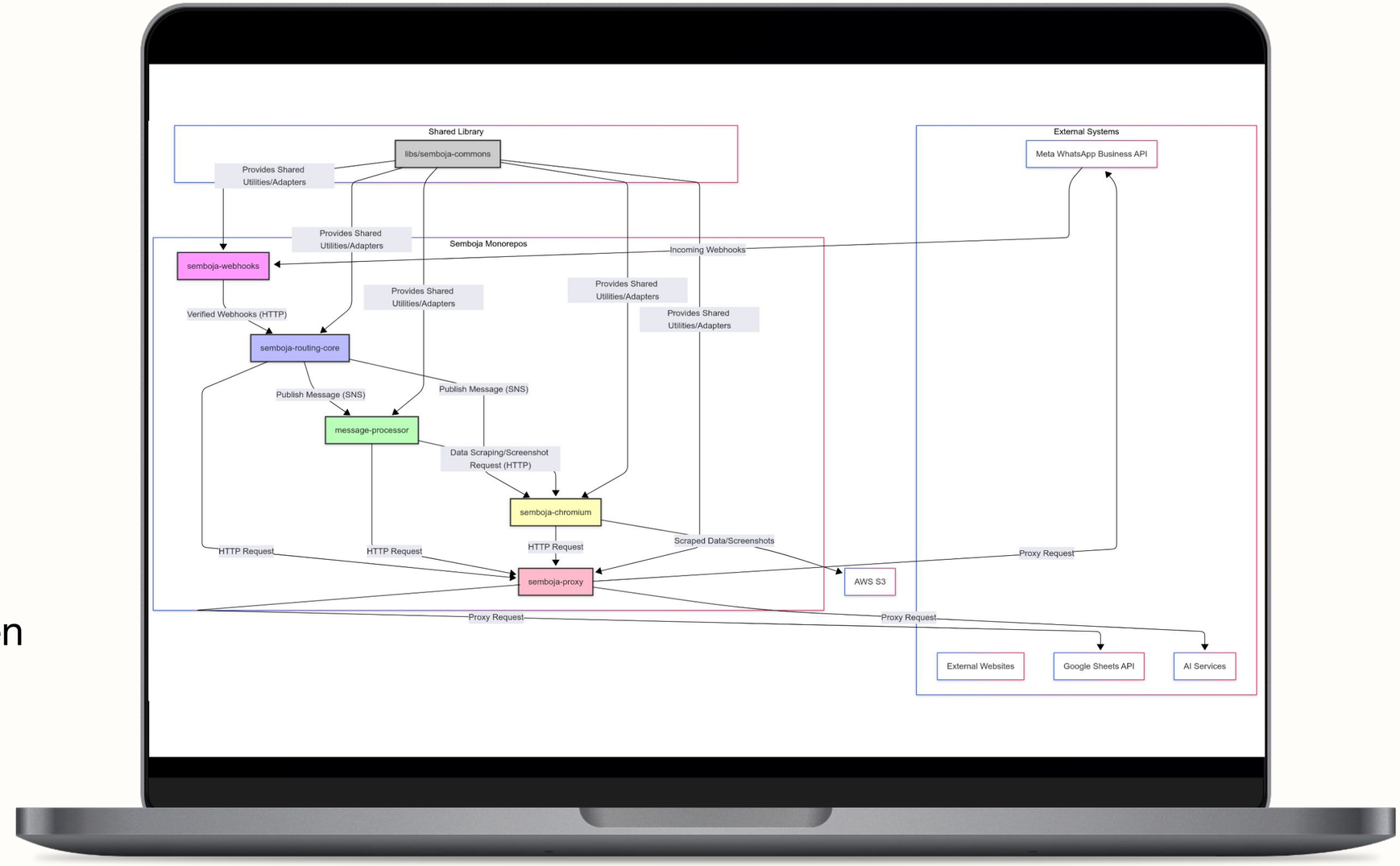
Useful

for migrating complex business logic

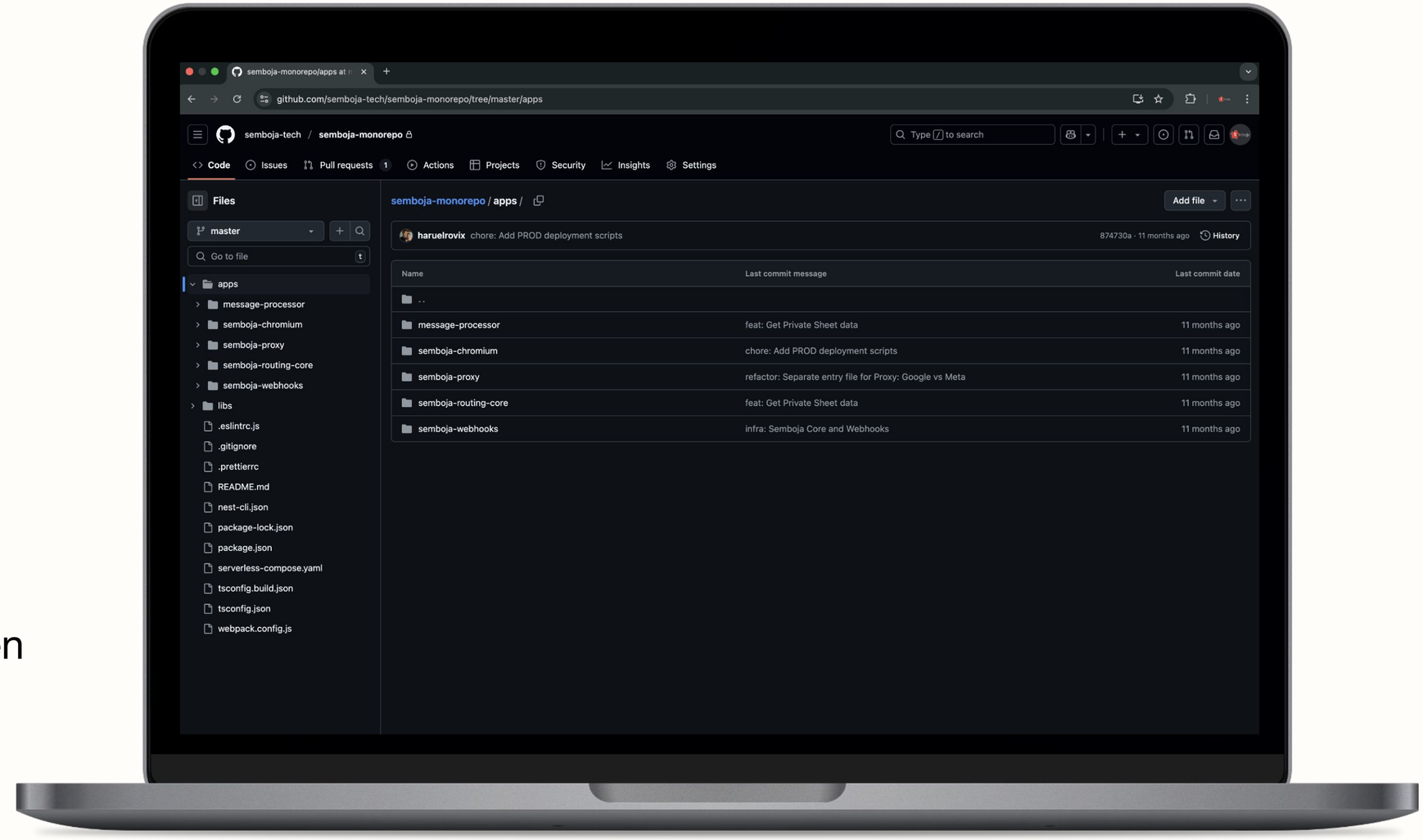
Demo

Semboja-monorepos.

An API-Based and Event-Driven integrations.



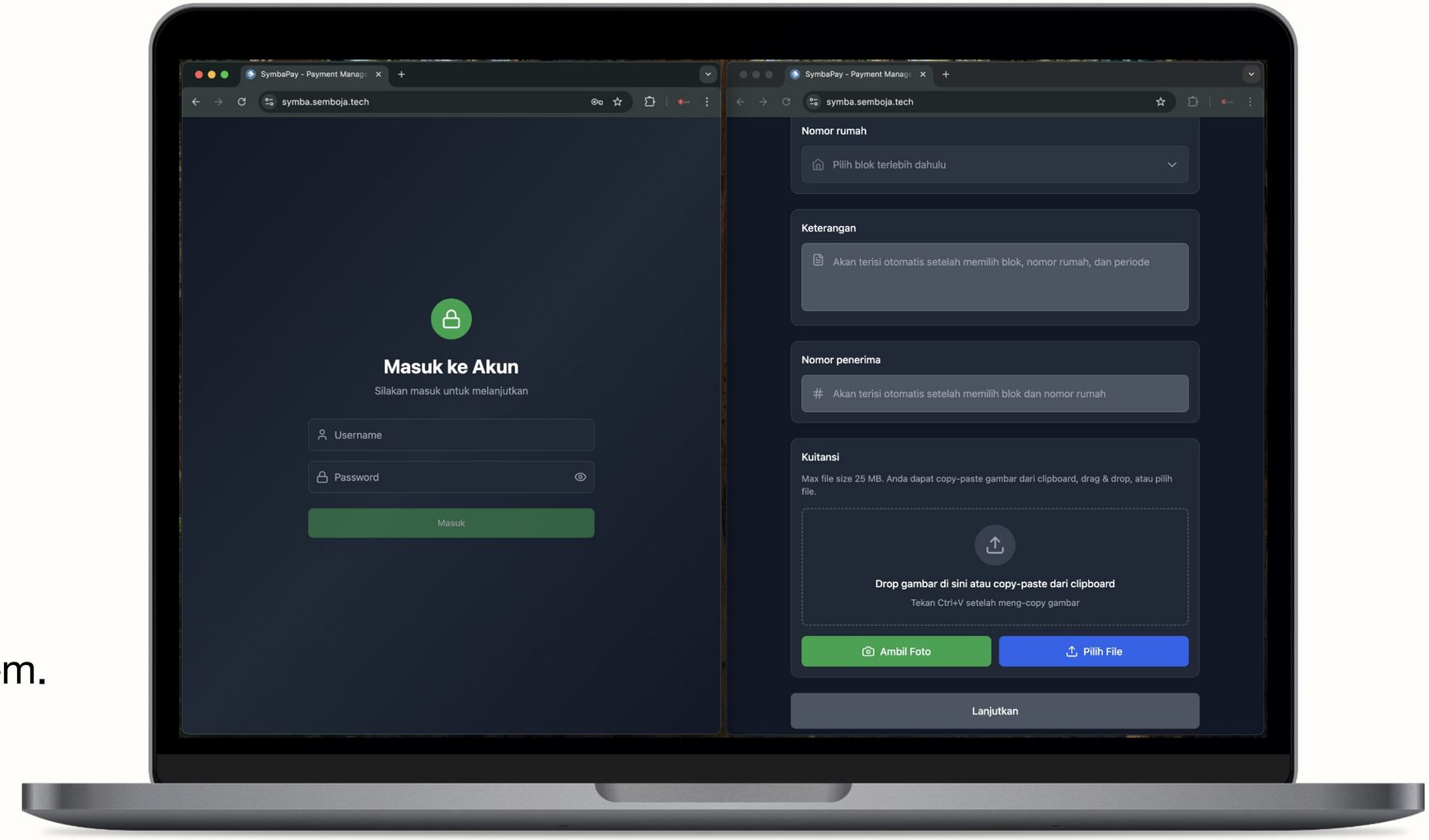
Demo



Semboja-monorepos.

An API-Based and Event-Driven integrations.

Demo



Semboja SymbaPay.

A payment management system.

OSOME

*"**Glue code** is powerful, but should be used wisely. In AWS, **serverless** tools make glue code easier to manage and scale, but architectural discipline is key.*

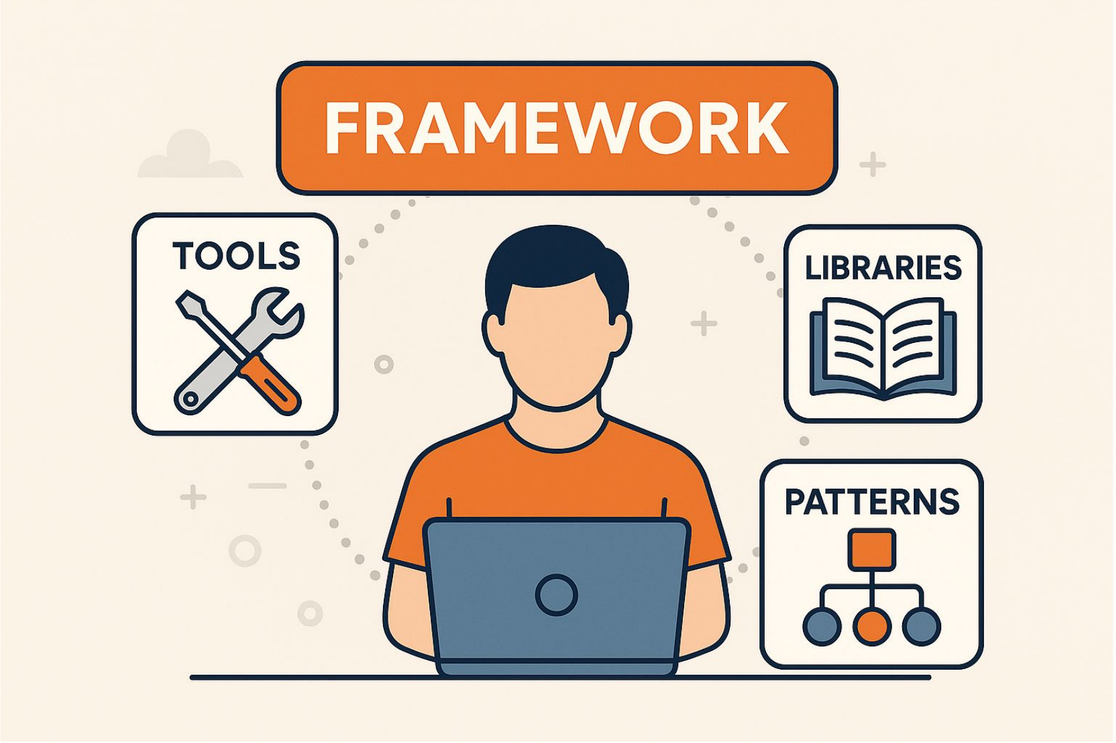
*Next, we'll see how **frameworks** help reduce the amount of glue we need."*

Frameworks that Support Integration

📘 What is a Framework?

A framework gives developers a **structured environment** for building applications.

*“It often comes with **tools**, **libraries**, and **patterns** for managing complexity.”*



Popular Frameworks for AWS Integration

Deploy APIs and event-driven apps to AWS Lambda easily

Building efficient, reliable and scalable server-side applications

Case Study: Banking Integrations

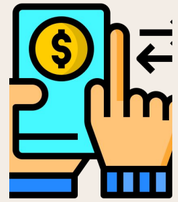
Semboja

WhatsApp and OCBC integrations

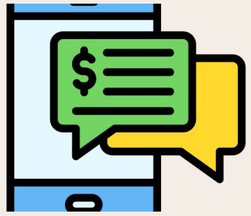
OCBC Chat Assistant features

Onboarding

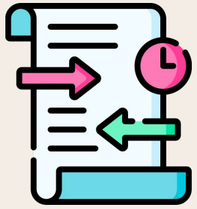
Transfer



Payment

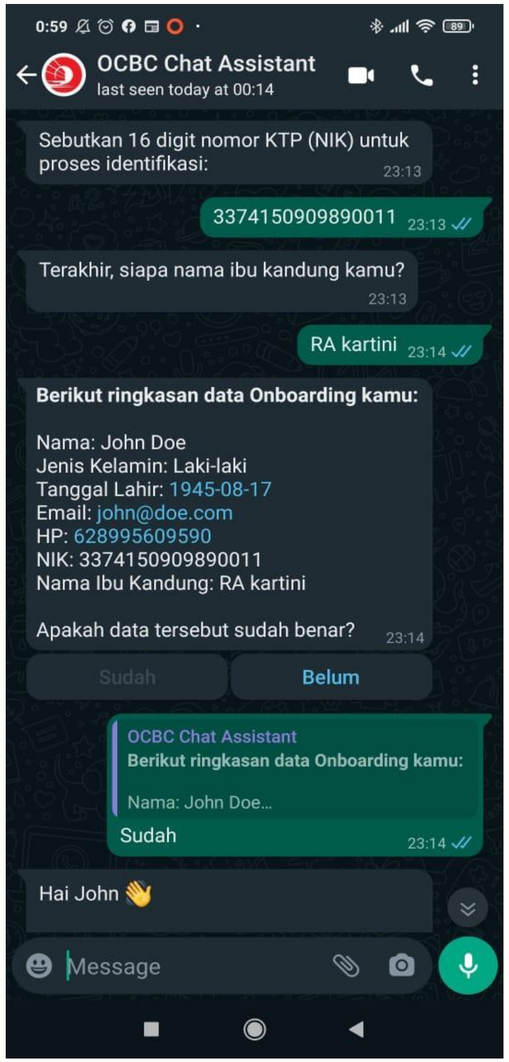


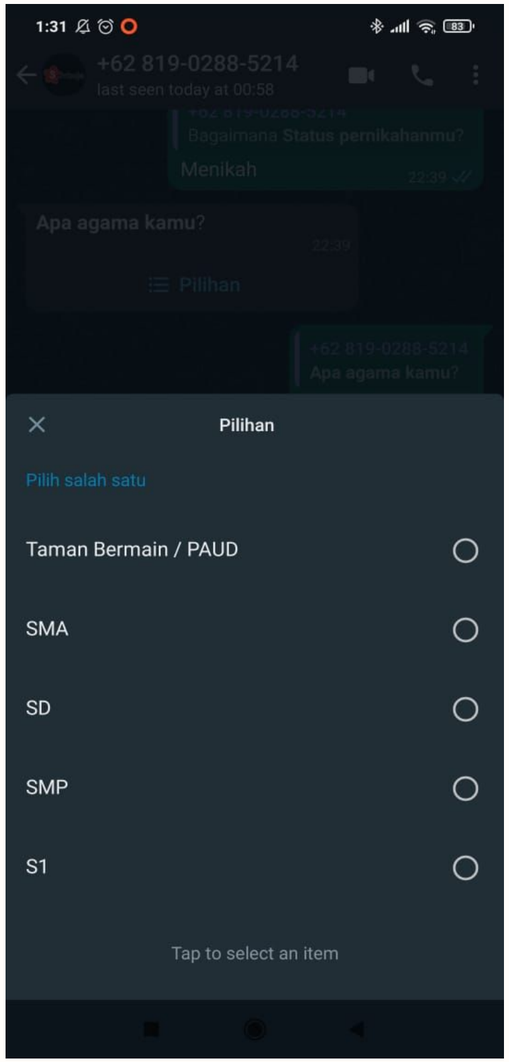
Account Balance

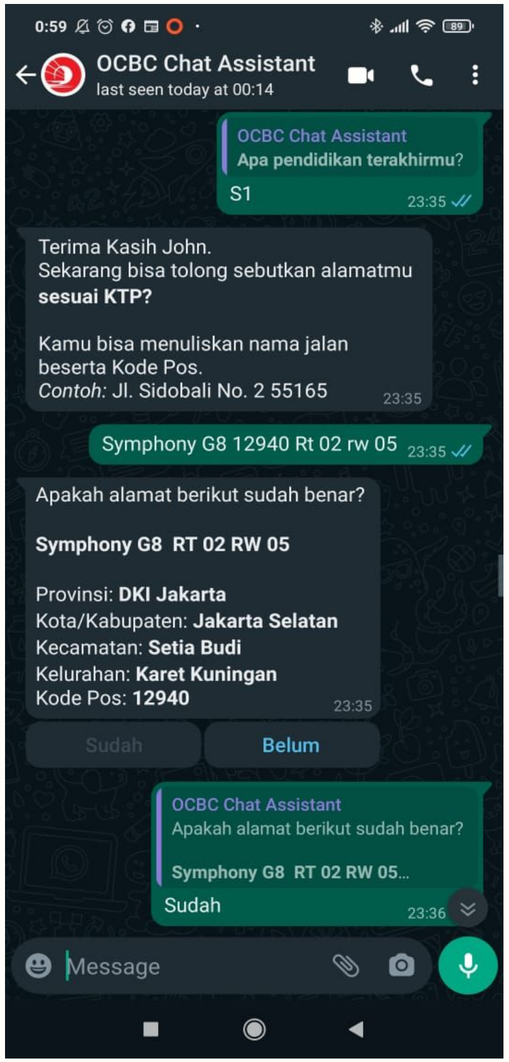


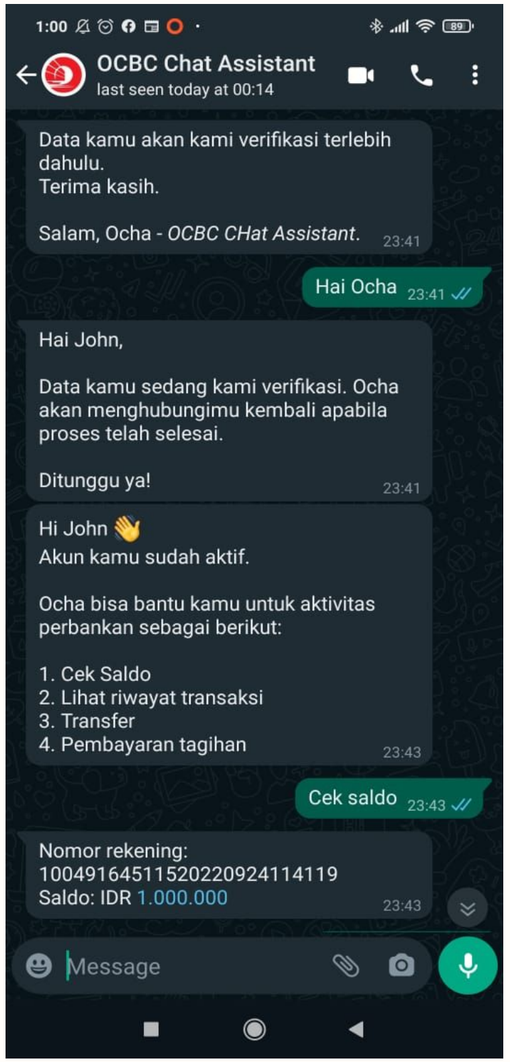
Transaction History

Onboarding


OCBC Chat Assistant
last seen today at 00:14
Sebutkan 16 digit nomor KTP (NIK) untuk proses identifikasi: 23:13
3374150909890011 23:13
Terakhir, siapa nama ibu kandung kamu? 23:13
RA kartini 23:14
Berikut ringkasan data Onboarding kamu:
Nama: John Doe
Jenis Kelamin: Laki-laki
Tanggal Lahir: 1945-08-17
Email: john@doe.com
HP: 628995609590
NIK: 3374150909890011
Nama Ibu Kandung: RA kartini
Apakah data tersebut sudah benar? 23:14
Sudah
Belum
OCBC Chat Assistant
Berikut ringkasan data Onboarding kamu:
Nama: John Doe...
Sudah 23:14
Hai John 👋
Message

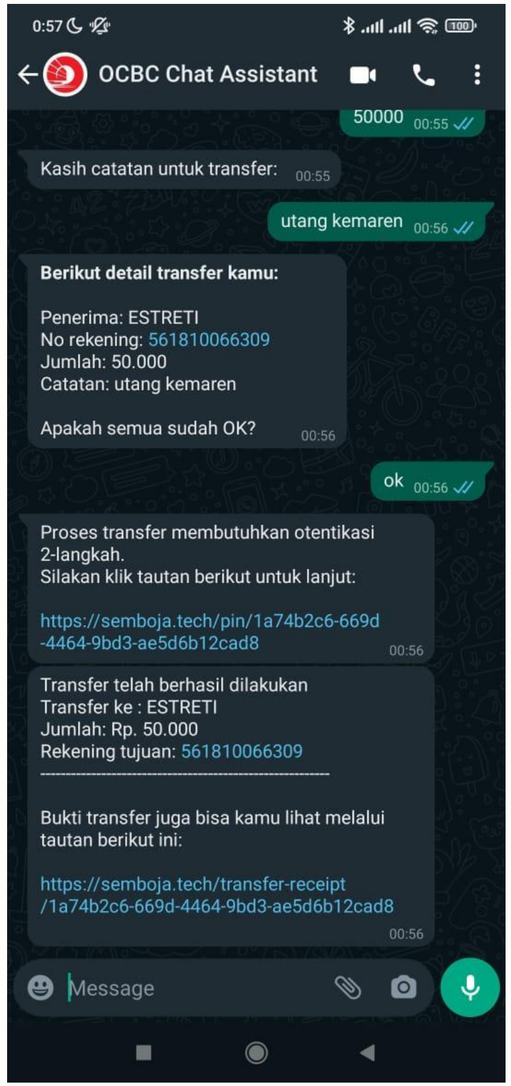

+62 819-0288-5214
last seen today at 00:58
Bagaimana Status pernikahanmu?
Menikah
Apa agama kamu?
Pilihan
Pilihan
Pilih salah satu
Taman Bermain / PAUD
SMA
SD
SMP
S1
Tap to select an item


OCBC Chat Assistant
last seen today at 00:14
OCBC Chat Assistant
Apa pendidikan terakhirmu?
S1 23:35
Terima Kasih John.
Sekarang bisa tolong sebutkan alamatmu sesuai KTP?
Kamu bisa menuliskan nama jalan beserta Kode Pos.
Contoh: Jl. Sidobali No. 2 55165 23:35
Symphony G8 12940 Rt 02 rw 05 23:35
Apakah alamat berikut sudah benar?
Symphony G8 RT 02 RW 05
Provinsi: DKI Jakarta
Kota/Kabupaten: Jakarta Selatan
Kecamatan: Setia Budi
Kelurahan: Karet Kuningan
Kode Pos: 12940 23:35
Sudah
Belum
OCBC Chat Assistant
Apakah alamat berikut sudah benar?
Symphony G8 RT 02 RW 05...
Sudah 23:36
Message

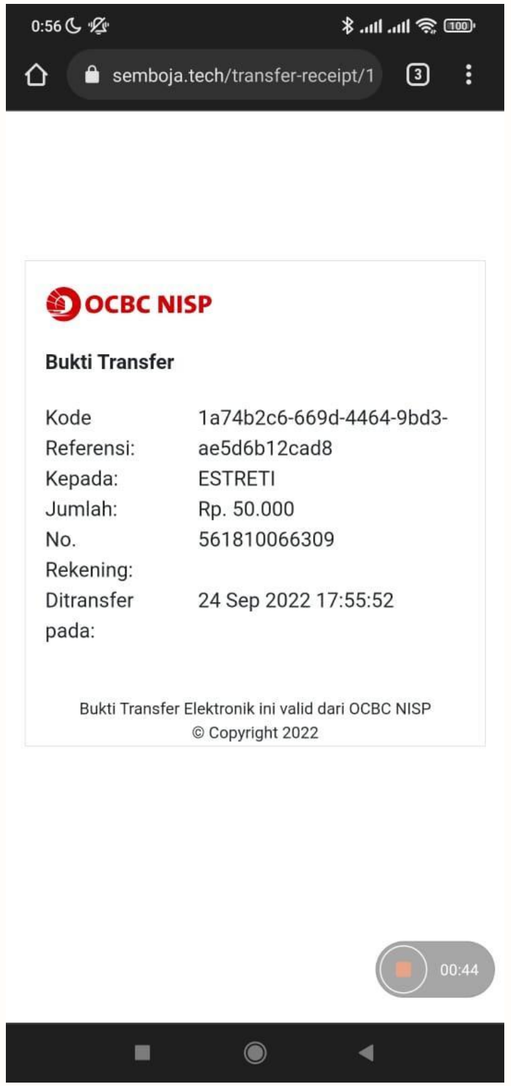

OCBC Chat Assistant
last seen today at 00:14
Data kamu akan kami verifikasi terlebih dahulu.
Terima kasih.
Salam, Ocha - OCBC CHat Assistant. 23:41
Hai Ocha 23:41
Hai John,
Data kamu sedang kami verifikasi. Ocha akan menghubungimu kembali apabila proses telah selesai.
Ditunggu ya! 23:41
Hi John 👋
Akun kamu sudah aktif.
Ocha bisa bantu kamu untuk aktivitas perbankan sebagai berikut:
1. Cek Saldo
2. Lihat riwayat transaksi
3. Transfer
4. Pembayaran tagihan 23:43
Cek saldo 23:43
Nomor rekening:
1004916451152022092411419
Saldo: IDR 1.000.000 23:43
Message

Transfer


0:57
OCBC Chat Assistant
transfer 00:55
Silakan masukkan nomor rekening tujuan transfer: 00:55
561810066309 00:55
Mau transfer berapa? 00:55
50000 00:55
Kasih catatan untuk transfer: 00:55
utang kemaren 00:56
Berikut detail transfer kamu:
Penerima: ESTRETI
No rekening: 561810066309
Jumlah: 50.000
Catatan: utang kemaren
Apakah semua sudah OK? 00:56
ok 00:56
Proses transfer membutuhkan otentikasi 2-langkah.
Silakan klik tautan berikut untuk lanjut:
https://semboja.tech/pin/1a74b2c6-669d-4464-9bd3-ae5d6b12cad8 00:56
Transfer telah berhasil dilakukan
Message

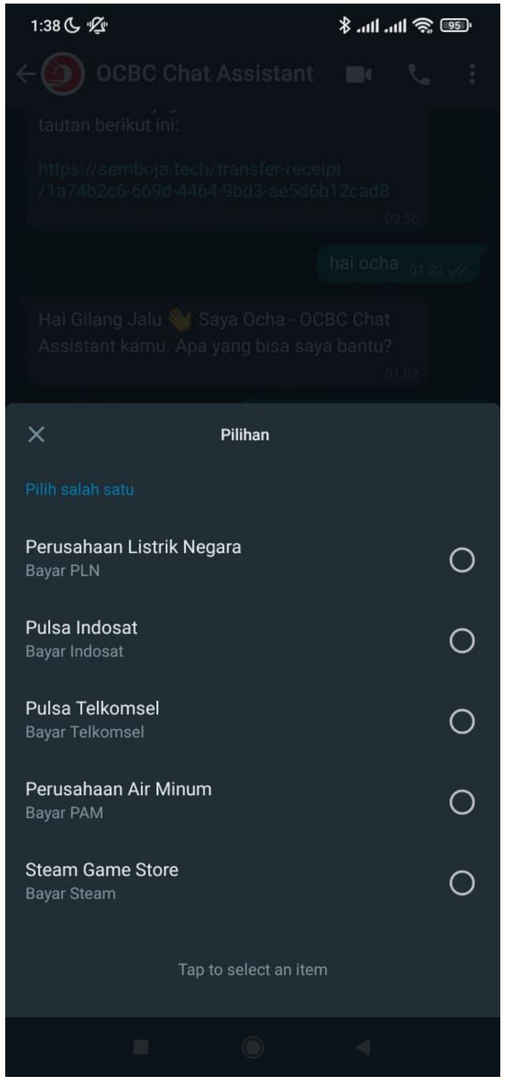

0:57
OCBC Chat Assistant
50000 00:55
Kasih catatan untuk transfer: 00:55
utang kemaren 00:56
Berikut detail transfer kamu:
Penerima: ESTRETI
No rekening: 561810066309
Jumlah: 50.000
Catatan: utang kemaren
Apakah semua sudah OK? 00:56
ok 00:56
Proses transfer membutuhkan otentikasi 2-langkah.
Silakan klik tautan berikut untuk lanjut:
https://semboja.tech/pin/1a74b2c6-669d-4464-9bd3-ae5d6b12cad8 00:56
Transfer telah berhasil dilakukan
Transfer ke : ESTRETI
Jumlah: Rp. 50.000
Rekening tujuan: 561810066309
Bukti transfer juga bisa kamu lihat melalui tautan berikut ini:
https://semboja.tech/transfer-receipt/1a74b2c6-669d-4464-9bd3-ae5d6b12cad8 00:56
Message


1:34
semboja.tech/pin/1a74b2c6-66
OCBC NISP
Masukkan PIN untuk melanjutkan proses transfer
Referensi: 1a74b2c6-6
Penerima: ESTRETI
Rekening: 561810066309
Jumlah: 50.000
Tanggal: 24-09-2022
Catatan: utang kemaren
PIN:
Submit
© Copyright 2022


0:56
semboja.tech/transfer-receipt/1
OCBC NISP
Bukti Transfer
Kode Referensi: 1a74b2c6-669d-4464-9bd3-ae5d6b12cad8
Kepada: ESTRETI
Jumlah: Rp. 50.000
No. Rekening: 561810066309
Ditransfer pada: 24 Sep 2022 17:55:52
Bukti Transfer Elektronik ini valid dari OCBC NISP
© Copyright 2022
00:44

Payment

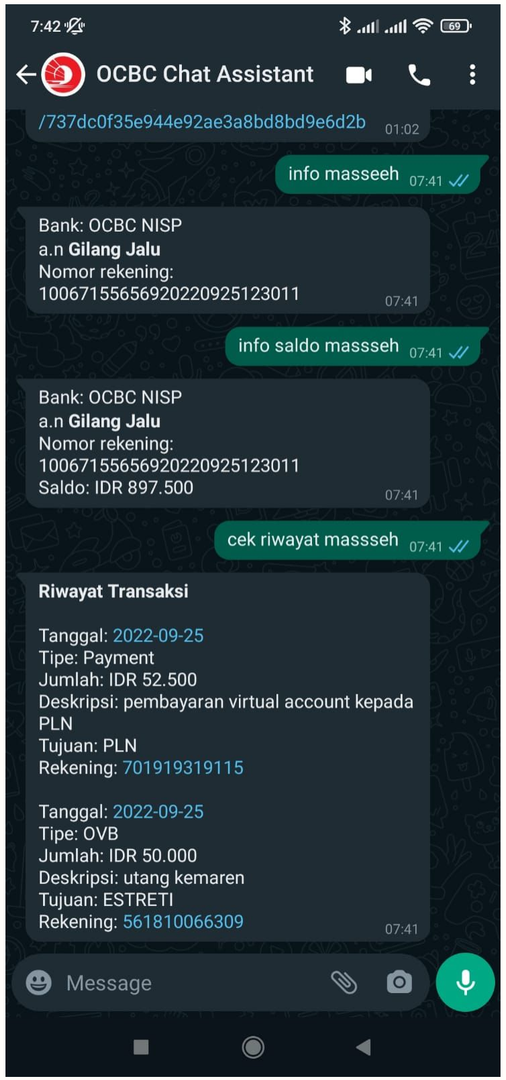

1:03
OCBC Chat Assistant
hai ocha 01:02
Hai Gilang Jalu 👋 Saya Ocha - OCBC Chat Assistant kamu. Apa yang bisa saya bantu? 01:02
mau bayar tagihan 01:02
Daftar Tagihan 01:02
Pilihan
OCBC Chat Assistant
Daftar Tagihan
Perusahaan Listrik Negara
Bayar PLN 01:02
Bayar Tagihan PLN 01:02
Pilihan
OCBC Chat Assistant
Bayar Tagihan PLN
Rp. 52.500
VA: 701919319115 01:02
Menunggu Pembayaran:
Tagihan: PLN
Jumlah: Rp. 52.500
Nomor VA: 701919319115
Message


1:38
OCBC Chat Assistant
tautan berikut ini:
https://semboja.tech/transfer-receipt/1a74b2c6-669d-4464-9bd3-ae5d6b12cad8
hai ocha 01:02
Hai Gilang Jalu 👋 Saya Ocha - OCBC Chat Assistant kamu. Apa yang bisa saya bantu? 01:02
Pilihan
Pilih salah satu
Perusahaan Listrik Negara
Bayar PLN
Pulsa Indosat
Bayar Indosat
Pulsa Telkomsel
Bayar Telkomsel
Perusahaan Air Minum
Bayar PAM
Steam Game Store
Bayar Steam
Tap to select an item

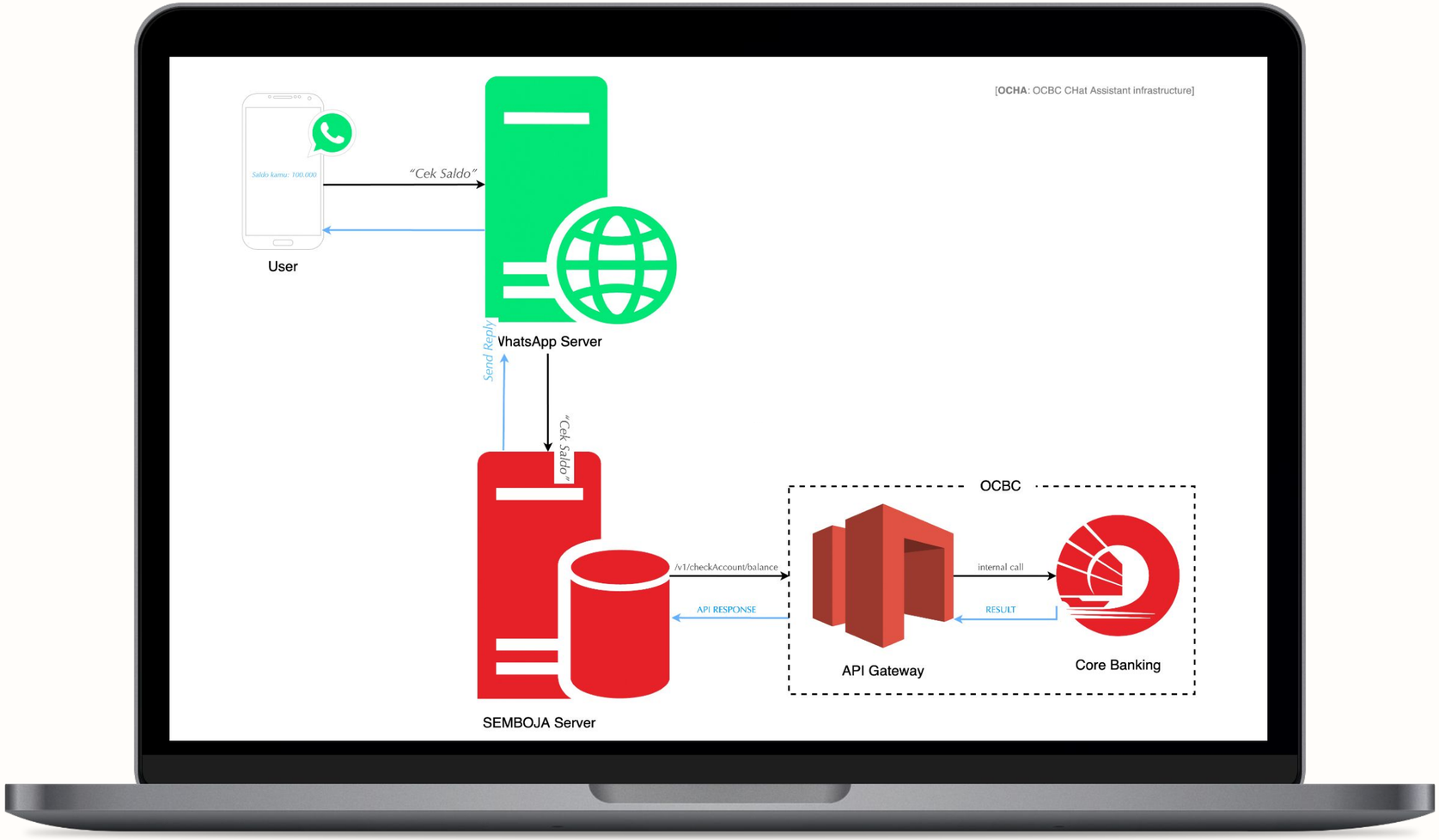

1:35
semboja.tech/payment/PLN/9/
OCBC NISP
Masukkan PIN untuk melanjutkan pembayaran
Tagihan: PLN
Jumlah: Rp. 52.500
Nomor VA: 701919319115
PIN:
Submit


1:42
semboja.tech/receipt/737dc0f3
OCBC NISP
Bukti Pembayaran
Referensi: 737dc0f35e944e92ae3a8bd8bd9e6d2b
Tagihan: PLN
Jumlah: Rp. 52.500
Nomor VA: 701919319115
Dibayar pada: 24 Sep 2022 18:02:46
Bukti Pembayaran Elektronik ini valid dari OCBC NISP
© Copyright 2022

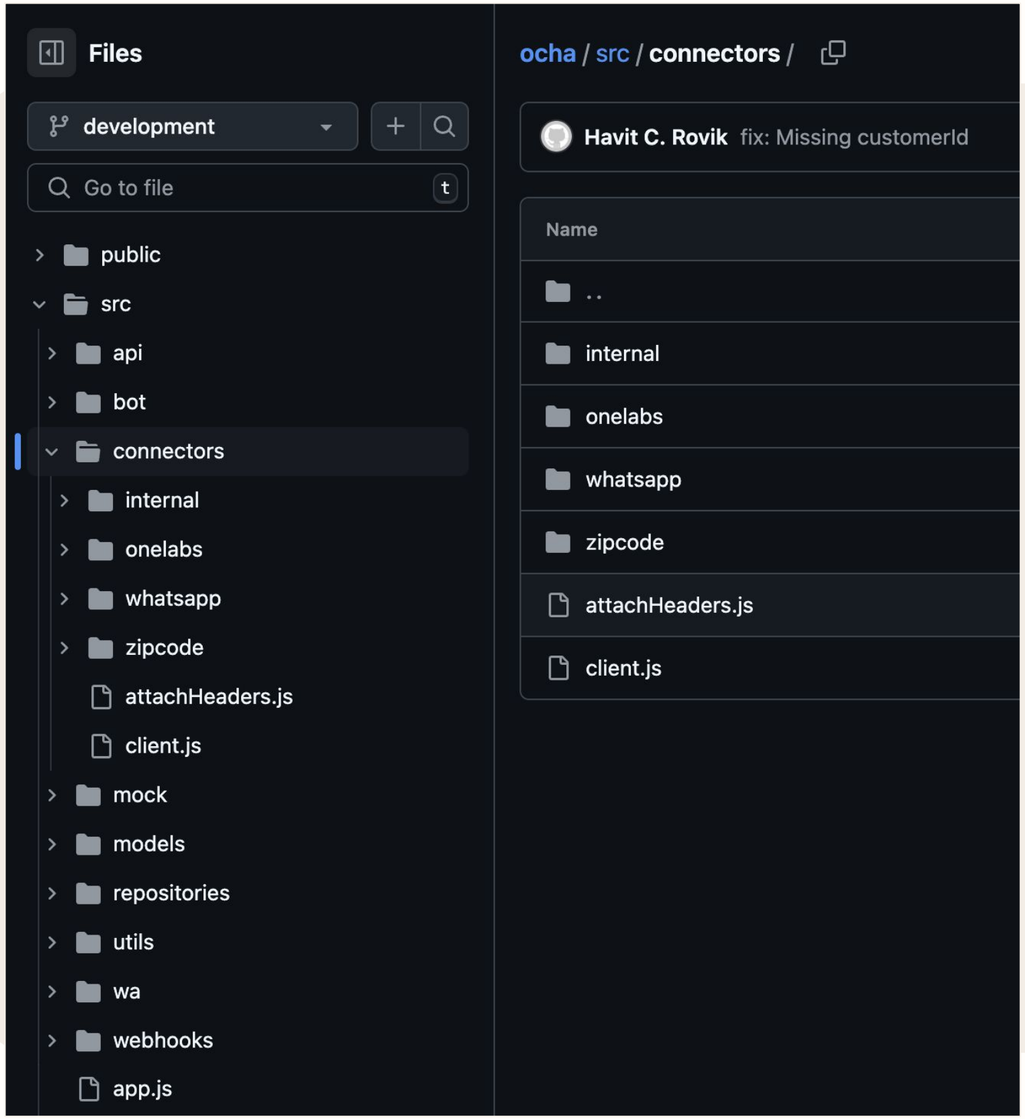
Check Balance & Transaction History

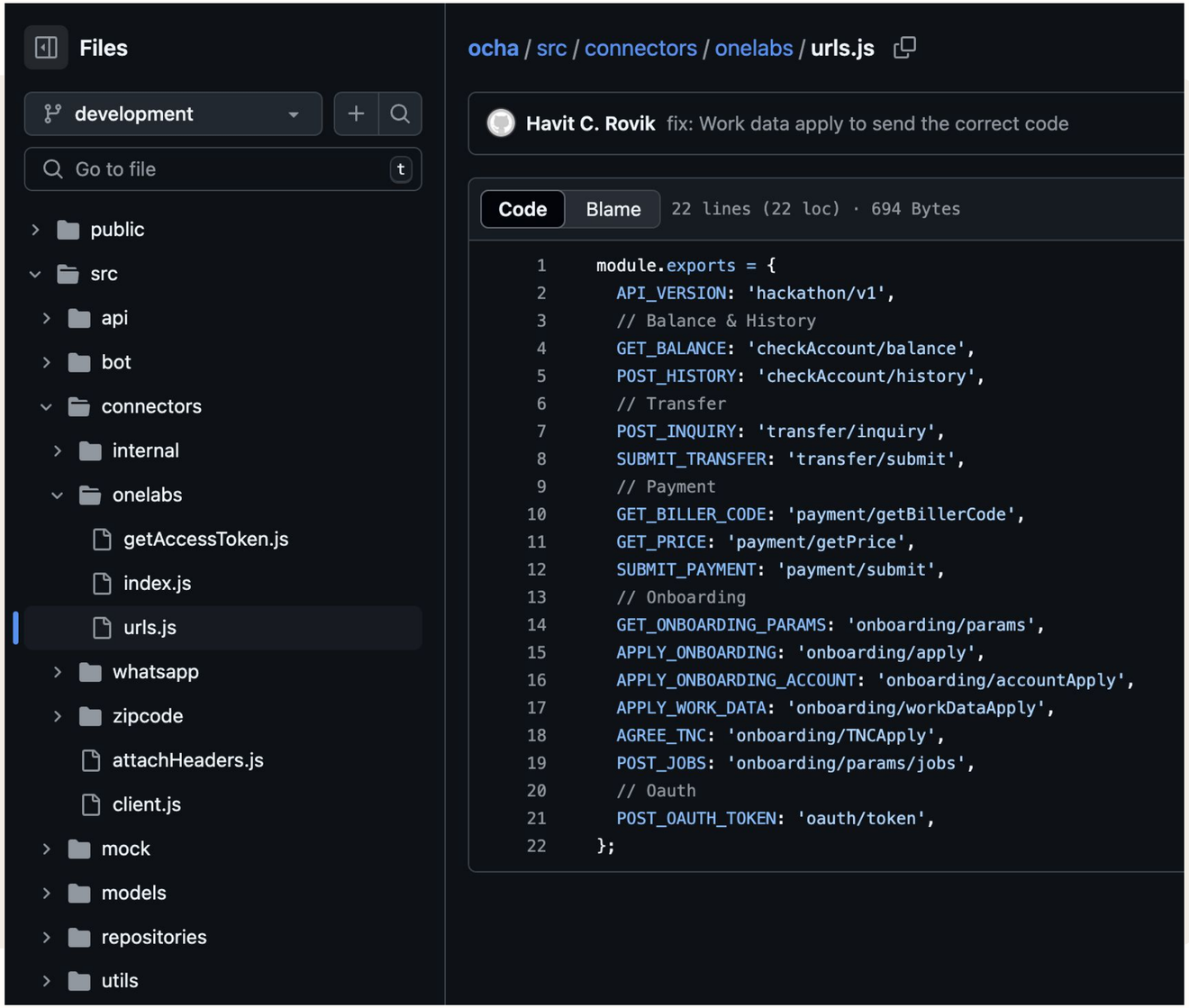


OCHA Infrastructure



Under the Hood (1)





Under the Hood (2)

Files

development

Go to file t

- public
- src
  - api
  - bot
  - connectors
    - internal
    - onelabs
      - getAccessToken.js
      - index.js
      - urls.js
    - whatsapp
    - zipcode
    - attachHeaders.js
    - client.js
  - mock
  - models
  - repositories
  - utils
  - wa
  - webhooks
  - app.js
  - index.js
  - init.js

ocha / src / connectors / onelabs / **getAccessToken.js**

Havit C. Rovik fix: Wrong payload for Grant Type

Code | Blame | 35 lines (30 loc) · 737 Bytes

```
1  const axios = require('axios');
2  const urls = require('./urls');
3
4  const instance = axios.create({
5    baseURL: process.env.OCBC_API_URL,
6    auth: {
7      username: process.env.OCBC_CLIENT_ID,
8      password: process.env.OCBC_CLIENT_SECRET,
9    },
10 });
11
12 module.exports = {
13   postAuthToken: async () => {
14     const data = {};
15     const payload = {
16       grant_type: 'client_credentials',
17     };
18
19     try {
20       const response = await instance({
21         url: urls.POST_OAUTH_TOKEN,
22         data: Object.keys(payload)
23           .map((key) => `${key}=${encodeURIComponent(payload[key])}`)
24           .join('&'),
25         method: 'POST',
26       });
27
28       return response.data;
29     } catch (err) {
30       console.error(err);
31     }
32
33     return data;
34   },
35 };
```

ocha / src / connectors / onelabs / **index.js**

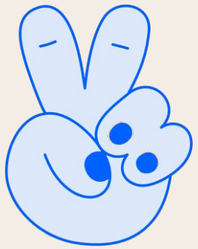
Code | Blame | 89 lines (74 loc) · 2.94 KB

```
14 instance.interceptors.request.use(async (config) => {
15   if (!memoryCache.get('OAUTH_TOKEN')) {
16     const oauth = await postAuthToken();
17     if (oauth) {
18       memoryCache.put('OAUTH_TOKEN', oauth.access_token, oauth.expires_in * 1000);
19     }
20   }
21
22   let authorizationHeaders = {};
23   const token = memoryCache.get('OAUTH_TOKEN');
24   if (token) {
25     const timestamp = Signature.getTimestamp();
26
27     let data;
28     if (config.data && config.data.customerId) {
29       const { customerId, ...rest } = config.data;
30
31       const session = await sessionOps.findOne({ where: { customerId } });
32
33       authorizationHeaders = {
34         'X-OCBC-ONBOARDING-TOKEN': session.token,
35       };
36
37       data = rest;
38     } else {
39       data = config.data || '';
40     }
41
42     const method = config.method.toUpperCase();
43     const url = `/hackathon/v1/${config.url}`;
44
45     authorizationHeaders = {
46       ...authorizationHeaders,
47       'X-OCBC-APIKey': Signature.apiKey,
48       'X-OCBC-Timestamp': timestamp,
49       'X-OCBC-Signature': new Signature().buildSignature(method, url, token, data, timestamp),
50       Authorization: `Bearer ${token}`,
51     };
52
```

Benefits of Using Frameworks

Less Glue Code

Reusable patterns reduce the need for custom integrations



Best Practices Built-In

Security, error handling, and retries are often pre-configured

Scalability & Maintenance

Easier to onboard new developers and scale teams

OSOME

*“**Frameworks** allow us to focus on business logic rather than wiring components manually.*

*Next, we’ll shift gears and look at how data integration, specifically the **data warehouse**, fits into enterprise integration.”*

Data Warehousing in Enterprise Integration

📘 What is a Data Warehouse?

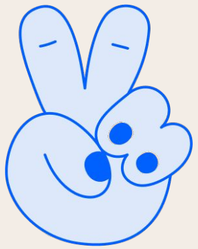
Centralized repository for structured, historical business data from **multiple sources**.

Optimized for OLAP (Online Analytical Processing), not for transactions.

“This is where we go to get the big picture: long-term trends, patterns, and reports.”

Role in Integration

Aggregates data from multiple operational systems



Supports analytics, reporting, and Business Intelligence

Enables a “single source of truth”

Business Intelligence

Reporting

Providing regular summary data to key decision-makers to support their ability to set business strategy and direction.

Data visualization

Presenting information visually in a way that aids the rapid understanding of complex information.

Predictive analytics

Analyzing historical data to predict future patterns using statistical techniques like data mining, machine learning, and predictive modeling.

Business Intelligence (2)

Data mining

Searching through large datasets to find useful patterns or trends.

Complex event processing

Analyzing streaming, real-time data from sources such as stock-market feeds, traffic reports, or electrical grids with sensors.

Business performance management

Analyzing and measuring performance goals, such as operational excellence goals defined by online shopping and customer satisfaction.

Amazon Redshift

“Powers modern data analytics at scale”

OSOME

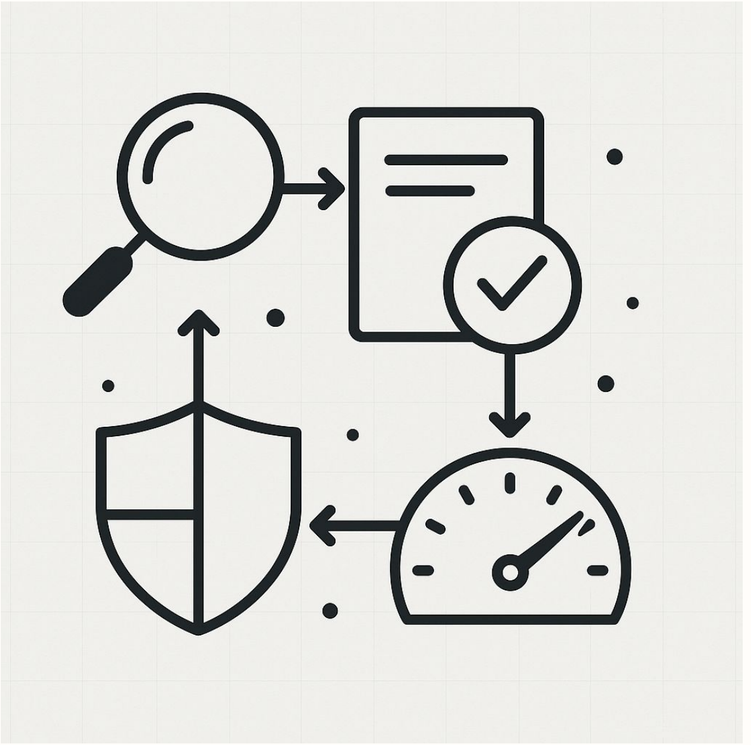
*"**Data warehouses** are essential to enterprise information integration. They unify scattered data into a coherent, reliable foundation for decision-making.*

*Finally, let's examine how integration choices affect **testing** and **evaluation**."*

Testing & Evaluation Impact

What is it?

Testing & Evaluation Impact refers to the influence that a system's integration approach, architecture, or changes have on the way it must be **tested** and **validated**.



How Integration type affects Testing

API-based Integration

Easier to unit test and mock using tools like **Postman**, Jest, or Supertest

Response-based → simple assertion-driven tests

"We can simulate most things with mocks or local testing."

Event-Driven Integration

Requires testing full event flow, often with real components

Harder to simulate timing and order issues

Test side effects, not just return values

"We test not what comes *back*, but what *happens* as a result."

Postman

"Execute, test, and interact with APIs in seconds"

Evaluation Impact

Latency and Performance

- Measure end-to-end time across services
- Use CloudWatch or X-Ray metrics

Failure Handling

- Test retry strategies, dead-letter queues, and fallback logic
- Use chaos testing in non-prod

Data Integrity

Ensure no data loss or duplication across retries/events

Observability

Ensure logs, traces, and metrics are unified and queryable

“What we can’t observe, we can’t trust.”

OSOME

*“Good **integration** is invisible to the user, but that only happens when **testing** is visible to the developer.*

With the right architecture and tooling, we can makes this level of visibility achievable.”

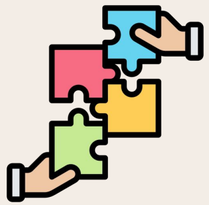
Deployment

What is Deployment?

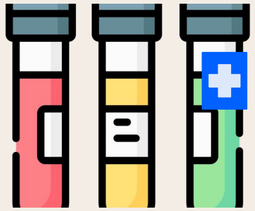
A process of releasing and running an application or system in a live environment, making it available for users or systems to access and use.

*“It is taking our application from **development** and putting it into **production**.”*

Typical Deployment Process



Build

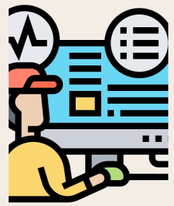


Test

Package

Deploy

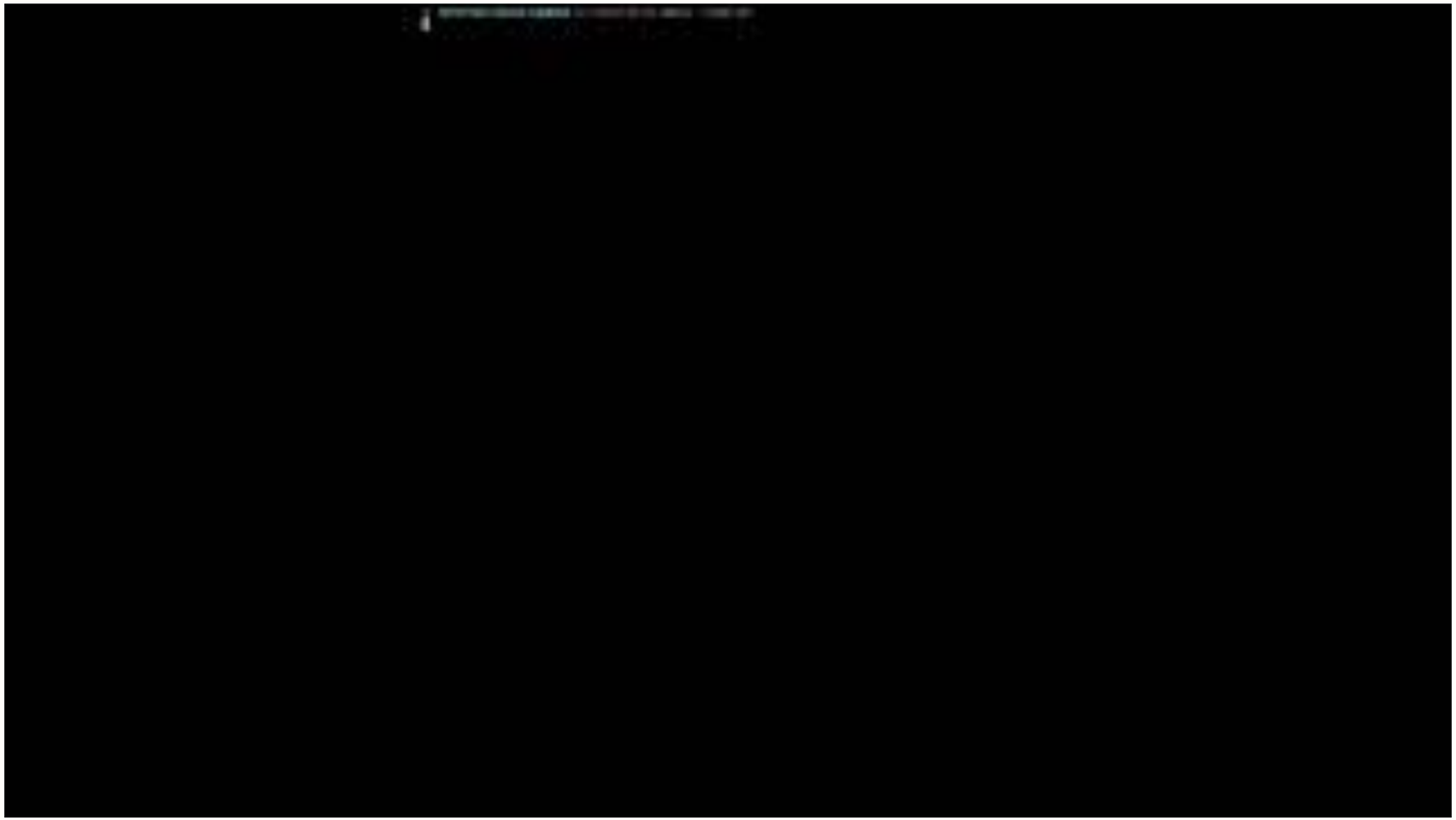
Release



Monitor

Serverless

“Deploying to AWS”

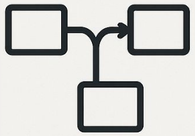


OSOME

*“Deployment in the **cloud** has evolved.*

*With serverless architecture and AWS services, we can now **deploy** faster, scale effortlessly, and focus more on building value rather than managing infrastructure.”*

What we've learned today



Express different ways
for middleware platforms.

Demonstrate the advantages
and disadvantages of some middleware platforms.

Justify major considerations
for the selection of an enterprise integration platform.

Express different ways of
integration using the "wrapper" approach.

Express different ways of
integration using the "glue code" approach.

Describe how a framework
facilitates integration of components.

Describe how the data warehouse
concept relates to enterprise information integration.

Describe how integration
choices affect testing and evaluation.

unisa Universitas 'Aisyiyah Yogyakarta | UNGGUL | Program Studi Teknologi Informasi | BAIK SEKALI | IABEE

Havit Choirul Rovik

- Senior Fullstack Engineer Osome
- Technical Writer Semboja Techno Solutions

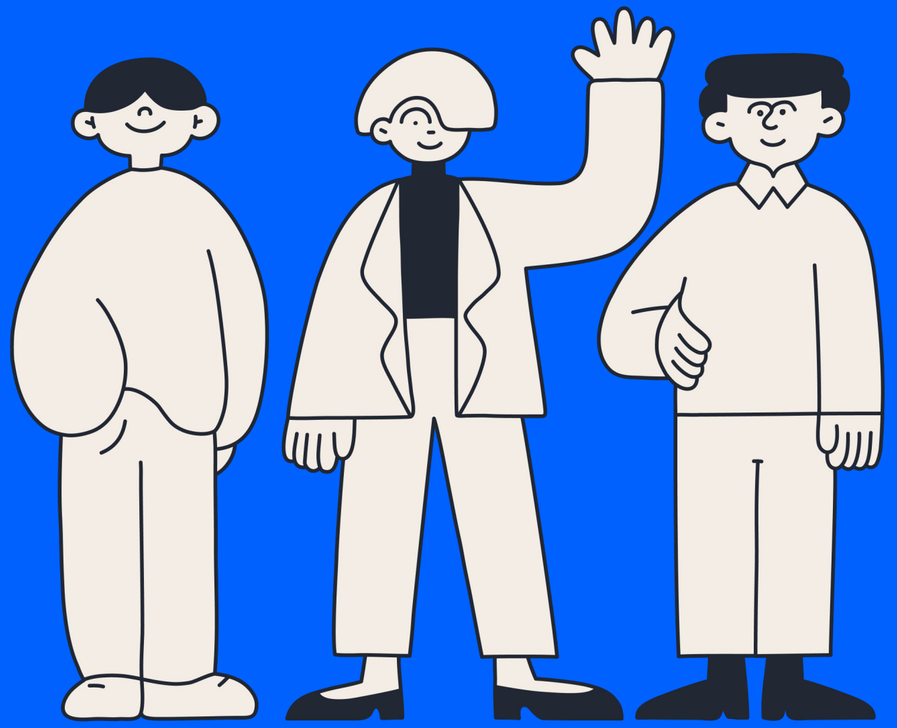
KODE#10

Tema :
Integration and Deployment

Selasa,
24 Juni 2025
08.00 WIB - Selesai
Ruang SM 4.8.15

Wajib bagi seluruh Mahasiswa PSTI UNISA Yogyakarta

teknologiinformasiunisa | TI Unisa | psti.unisayogya.ac.id



OSOME

linkedin.com/in/havit