Bill Pay Application Design

We will use Microservice architecture for design of Bill Pay Application.

Below Microservices are identified at this stage:

1. Registration Service
2. Wallet Service
3. FileOperationsService
4. BillerService
5. PaymentService

All these services will be exposed to customer through API Gateway.

Inter service communication will be handled using Publisher Subscriber model. AWS Simple Notification Service will be used for this. Services will subscribe to topics of their interest and then can take actions once relevant messages are received.

# Registration Service:

This Service will be used for registering new customer. Here customer email Id will be accepted and stored.

Once customer is register then CustomerRegistered message is passed to the AWS SNS.

## Responsibilities:

1. Register User
2. Activate User
3. Forgot/ Reset Password
4. Add Biller

# Wallet Service:

This service will be responsible for creating wallet and maintaining all transactions of the wallet.

## Responsibilities:

1. Create Wallet
2. Add Money To Wallet
3. Debit From Wallet
4. Get Transactions
5. Get Available Balance

Biller Service:

This service will fetch bills from billers along with payment due amount. We can schedule this service to fetch data at regular frequency of the day. In ideal scenario once in a day should be sufficient to get bills information.

## Responsibilities:

1. Fetch Latest Bills for user
2. Fetch Third Party Billers

File Operations Service:

This service will be responsible for handling bulk operations.

For files received in standard format, it can read entries and pass it on the AWS SQS for further processing of payments.

In case of adhoc files, Data Formatter comes in picture which will fetch required information from non-standard file and pass it on to SQS.

## Responsibilities:

1. Process Standard File
2. Process AdHoc File
3. Format Unstructured Data
4. Queue In Billing Requests

Payment Service:

This service will be responsible for actual payment to the billers. This will read the contents from AWS SQS. Every bill will be passed through SQS and in coordination with wallet service actual payment will be processed

We can implement dead letter queue here to track failed bill payments.

## Responsibilities:

1. Process Bill
2. Pay Biller Using Wallet

# Proposed Tech Stack

|  |  |  |
| --- | --- | --- |
| **Service** | **Tech stack** | **Comments** |
| Registration Service  Wallet Service  Biller Service  Payments Service | .Net Core  SQL | As data for these services will be structured so choosing SQL DB  .Net Core will help in high performance APIs and light weight deployment |
| FileOperationsService | .Net Core  DynamoDB | Here DynamoDB or any NoSQL database can be chosen as we can expect unstructured data for files specifically non-standard files. |
| Logging | AWS CloudWatch  OR  NewRelic | We will need to keep a track of most of events and actions in all service areas. Hence AWS Cloudwatch or NewRelic can be selected |
| PCI Compliance | Encryption for transmitted data | PCI DSS standards can be followed. Not having much details around its implementation. |
| Authorization, Rate Limiting | AWS API Gateway | Use capabilities of AWS API Gateway for the authorization and rate limiting. |

# Testing Strategy:

Every service will have its own set of Unit test cases.

We will also have separate Integration tests which will be executed at certain defined interval or at time of code checkin.

We will also implement Health Checks and alerting based on same.

As we may have limited UI so we may not require A/B testing at this stage but we can have that in case required

# Deployments

We will mostly use Azure DevOps pipelines to configure and deploy resources. We can even have canary deployments configured. These deployments can be gated deployments and we can also configure Approval mechanism at certain stages.

# Scalability

Depending upon the AWS resource that we are using either it will be taken care by AWS itself or for some resources we have to do additional configuration for the scaling resources based on certain triggers likes number of requests or response time.