Bill Pay Application

High Level Design Document

XYZ Corporation would like to create a new bill payment application to act as hub between customers and billers. XYZ Corporation will maintain customer account with balances in it, will maintain biller list along with bill data and will store all transaction data.

Registration Service: This micro service will be responsible for registering the customer as well as wallet creation and funds movement into the wallet.

**API Specifications –**

1. ***https:/{host}/customer-registration/onboard*** **Type – Post**

Payload – email Id (mandatory), transaction amount with currency (optional)

Customer will be onboarded and wallet will be created and if amount is present then credit entry will be made.

1. ***https:/{host}/customer-registration/onboard*** **Type – Put**

Same payload as above but transaction amount is mandatory as we will be crediting the wallet.

|  |  |
| --- | --- |
| Data model | Tech Stack |
| Customer   * Customer\_ID * Email\_id   Wallet   * Wallet\_Id * Customer\_id * Current\_bal * Current\_bal\_currency   Wallet\_history   * Wallet\_Id * Customer\_id * Transaction\_type * Transaction\_amt * Current\_bal * Current\_bal\_currency | * Java with SpringBoot, docker, Kubernetese, Cucumber framework * Database – any RDBMS * Hosting environment - Any public cloud * Testing strategy – Unit testing, automation testing (BDD) * Logging – Splunk or Scalyr (Alerts and dashboards will be used for reporting) * CI/CD – Git, Jenkins * Security – At API gateway level (JWT token based) * Disaster recovery & Auto scaling – Deploy application in more than two regions with 3 self-healing pods. Configure the auto scaling if cpu consumed by more than 75%. * Make automation suite part of Jenkins pipeline and fail the build if automation fails * Quality gates – checkmarx, owsap, jacoco coverage (all should be integrated in pipeline) |

Biller Service: This micro service will be responsible for below actions –

1. On-board billers in system
2. Select a biller, fetch the bill from 3rd party external applications and pay that using the wallet.
3. Move funds from customer wallet to biller account.

**API Specifications –**

1. ***https:/{host}/biller-service/onboard*** **Type – Post**

Payload – basic details like name & description

Different billers will be the consumer of this service.

1. ***https:/{host}/biller-service/bills/{id}*** **Type – Get**

See if the biller is supported by application if yes then see if bill details is available in bill table then return the same details else call the external system and fetch the bill by ID. If biller is not supported then let the customer know about this.

1. ***https:/{host}/biller-service/bill/pay*** **Type - Post**

Payload – (Request Body) – It will contains customer id, bill id, transaction amount.

Check the wallet of the customer for mentioned amount. If the balance is not sufficient then reject the transaction with proper exception. If balance is there then consume the biller api for payment. Make the transaction entry in wallet history table and update the wallet balance.

|  |  |
| --- | --- |
| Data model | Tech Stack |
| Biller\_Details   * Biller\_id * Biller\_name * Biller\_Desc * Audit columns   Bill\_Details   * Bill\_id * Biller\_id * content (blob)   Transaction\_log   * Biller\_id * Bill\_id * Customer\_id * Transaction\_amt * Transaction\_date * And other audit related fields | * Java with SpringBoot, docker, Kubernetese, Cucumber framework * Database – any RDBMS * Hosting environment - Any public cloud * Testing strategy – Unit testing, automation testing (BDD) * Logging – Splunk or Scalyr (Alerts and dashboards will be used for reporting) * CI/CD – Git, Jenkins * Security – At API gateway level (JWT token based) * Disaster recovery & Auto scaling – Deploy application in more than two regions with 3 self-healing pods. Configure the auto scaling if cpu consumed by more than 75%. * Make automation suite part of Jenkins pipeline and fail the build if automation fails * Quality gates – checkmarx, owsap, jacoco coverage (all should be integrated in pipeline)   Note – If the traffic/transactions are huge then we can think of other no-sql databases like Cassandra for transaction log details. |

Bulk Payment Service: This interface will be responsible for bulk payment. Bill payment files will be directly sent by biller in specific format then data ingestion will happen. After that ‘pay’ end point will be consumed to make the payment one by one.

Entire flow can be done synchronously in a batch or asynchronously by introducing any messaging broker after ingestion and putting the info on broker topic then consumer doing the needful.

* File Standard Specification/Schema

CSV file should contain below mandatory attributes and biller should directly upload the file in ftp server maintained by XYZ Corporation. All the files will picked up by schema validator in specific time – EoD or BoD or both based on the agreement and data ingestion will happen in a batch.

Biller\_id, bill\_id, customer\_id, transaction\_amt, due\_date

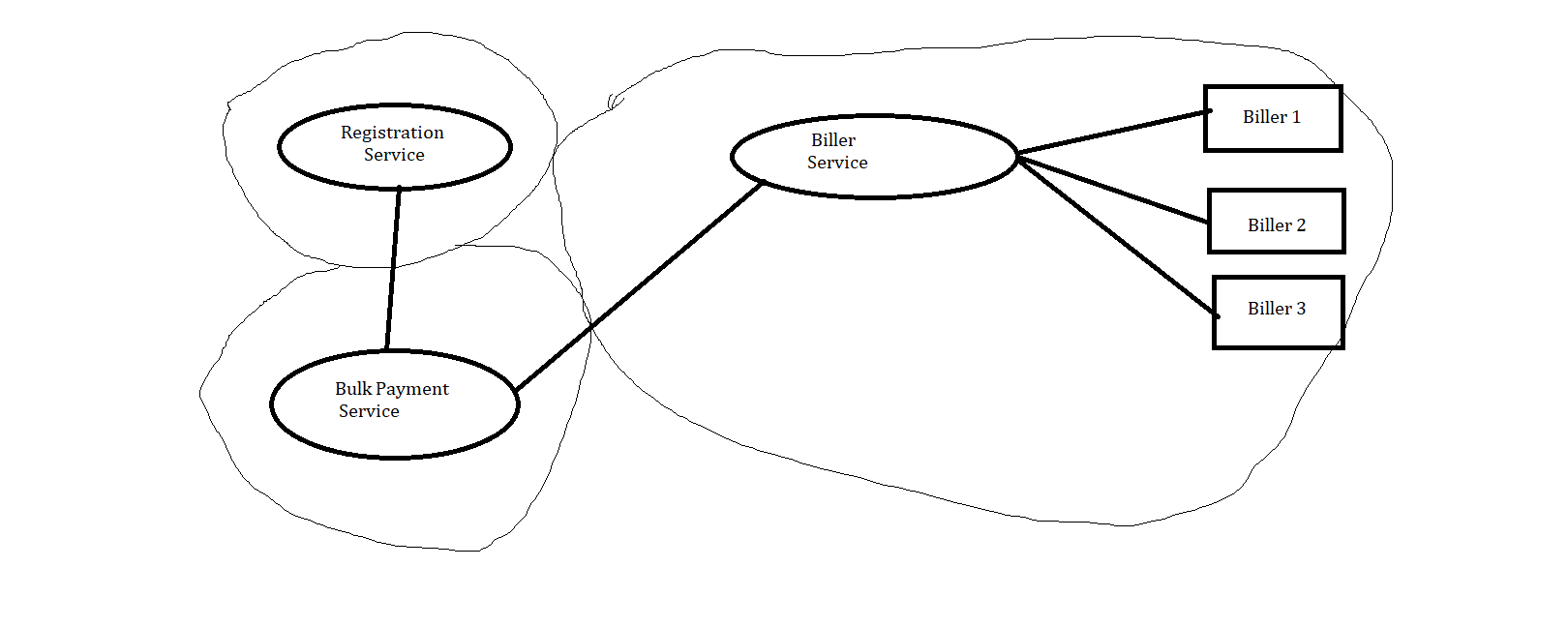
Schema validator will parse all these csv files and put in ‘bulk\_payment\_details’ table and then call the biller service to make the payment. If there are any failures then same can be logged with exception. We can also introduce retry mechanism if needed.

|  |  |
| --- | --- |
| Data model | Tech Stack |
| Bulk\_payment\_details   * Biller\_id * Bill\_id * Customer\_id * Transaction\_amt * Transaction\_date * Status i.e. success or failure * Reason * Retry - number | Tech stack mostly would be same here as mentioned above. Spring batch framework can be used with cron configuration for data ingestion. Nightly job can be set up for re try purpose if something fails. |

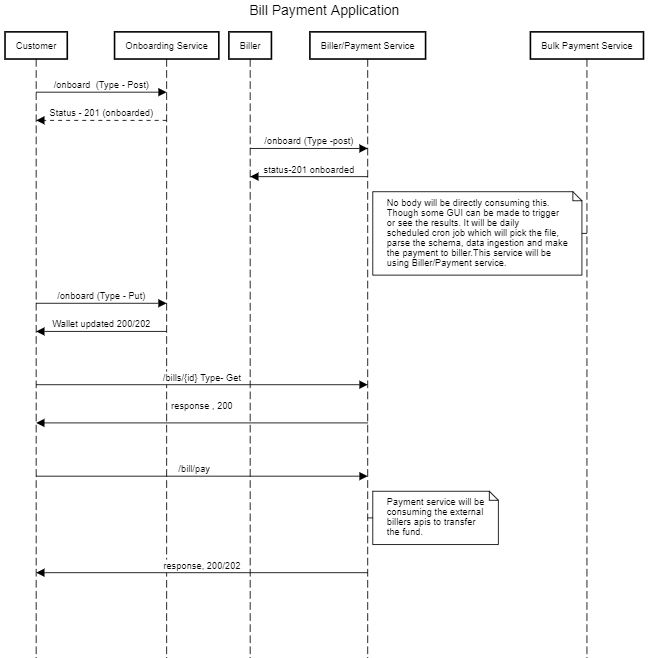
* Non Standard Specification/Schema

If any new format comes in future then only we have to extend schema validator to parse the new file and ingest the data

**Bounded Context Diagram**



**Sequence Diagram**

****