Banking Simulation ReadyAPI Project Documentation

# 📘 Project Overview

\*\*Project Name\*\*: BankingSimulation

\*\*Type\*\*: ReadyAPI REST Virtualization Project

\*\*Description\*\*: A mock banking service with REST endpoints and scripts for validation and simulation.

# 🔗 Base Endpoint & Paths

Base URL: http://localhost:8088/

Available Paths: /customer, /accounts, /branch, /kyc (GET/POST for each)

# 📁 API Resources

1. /customer - Handles customer onboarding.

2. /accounts - Account creation and listing.

3. /branch - Bank branch registration.

4. /kyc - KYC document submission and retrieval.

# 🧪 Virtual Services

1. onboarding Virtual Service (Port 8088) - Handles all /customer, /accounts, /branch, /kyc logic.

2. test\_script (Port 8081) - Handles advanced validation via scripts for createCustomer, createaccount, etc.

# 🛠️ Dispatch Types

SCRIPT: Groovy-based payload and logic validation.

SEQUENCE: Rotates through predefined responses.

QUERY MATCH: Uses JSONPath to match response.

# 📦 Sample API Request Payloads

/customer POST:  
{  
 "name": "Shiva Shankar",  
 "email": "shiva@@gmail",  
 "dob": "1994-02-10",  
 "panNumber": "PQRSX9876Z"  
}

/accounts POST:  
{  
 "accountId": "ACC001",  
 "customerId": "CUST001",  
 "type": "savings",  
 "balance": 5000  
}

/branch POST:  
{  
 "branchId": "BR001",  
 "name": "Mumbai Central",  
 "ifsc": "BANK0001234"  
}

/kyc POST:  
{  
 "kycId": "KYC001",  
 "customerId": "CUST001",  
 "documentType": "Aadhar",  
 "documentNumber": "1234-5678-9012"  
}

# ✅ Validations & Configs

- Email and PAN validation using regex.

- DOB format checks (YYYY-MM-DD).

- Auto-generating customer numbers.

- Required field checks and error messaging.

- Project property: lastCustomerNumber = 100024

# 📜 Scripts Used

## Add Customer (onboarding Virtual Service)

import groovy.json.JsonSlurper  
import groovy.json.JsonOutput  
  
def request = mockRequest.getRequestContent()  
log.info "Incoming: " + request  
  
def slurper = new JsonSlurper()  
def data  
  
try {  
 data = slurper.parseText(request)  
} catch(Exception e) {  
 mockOperationResult.responseContent = JsonOutput.toJson([message: "Invalid JSON"])  
 return "Missing or Invalid Fields"  
}  
  
def requiredFields = ['name', 'email', 'dob', 'panNumber']  
if (!requiredFields.every { data.containsKey(it) }) {  
 mockOperationResult.responseContent = JsonOutput.toJson([message: "Missing required fields"])  
 return "Missing or Invalid Fields"  
}  
  
def emailRegex = /^[\w\.-]+@[\w\.-]+\.\w+$/  
if (!(data.email ==~ emailRegex)) {  
 mockOperationResult.responseContent = JsonOutput.toJson([message: "Invalid email format"])  
 return "Invalid Email"  
}  
  
mockOperationResult.responseContent = JsonOutput.prettyPrint(JsonOutput.toJson([  
 message: "Customer created successfully",  
 name: data.name,  
 panNumber: data.panNumber  
]))  
  
return "Customer Created"

## Create Customer (test\_script)

import groovy.json.JsonSlurper  
  
def requestBody = mockRequest.getRequestContent()  
if (requestBody == null || requestBody.trim().isEmpty()) {  
 context.responseContent = '{"status": "validationfailed", "errors": ["Empty request body"]}'  
 return "validationfailed"  
}  
  
def json  
try {  
 json = new JsonSlurper().parseText(requestBody)  
} catch (Exception e) {  
 context.responseContent = '{"status": "validationfailed", "errors": ["Invalid JSON format"]}'  
 return "validationfailed"  
}  
  
def name = json.name  
def email = json.email  
def dob = json.dob  
def pan = json.panNumber  
  
def errors = []  
  
if (!name || !(name ==~ /^[A-Za-z ]{2,}$/)) errors << "Invalid or missing name"  
if (!email || !(email ==~ /^[\w.-]+@[\w.-]+\.\w{2,}$/)) errors << "Invalid or missing email"  
if (!dob || !(dob ==~ /^\d{4}-\d{2}-\d{2}$/)) errors << "Invalid or missing DOB"  
if (!pan || !(pan ==~ /^[A-Z]{5}[0-9]{4}[A-Z]{1}$/)) errors << "Invalid or missing PAN"  
  
if (errors) {  
 context.responseContent = new groovy.json.JsonBuilder([  
 status: "validationfailed",  
 errors: errors  
 ]).toPrettyString()  
 return "validationfailed"  
}  
  
def project = mockOperation.mockService.project  
def lastNumberProp = "lastCustomerNumber"  
def lastNumber = project.hasProperty(lastNumberProp) ? project.getPropertyValue(lastNumberProp)?.toInteger() : 100000  
def nextNumber = lastNumber + 1  
def customerNumber = "CUST${nextNumber}"  
project.setPropertyValue(lastNumberProp, nextNumber.toString())  
  
context.customerNumber = customerNumber  
return "success"

## Create Account (test\_script)

import groovy.json.JsonSlurper  
import groovy.json.JsonOutput  
  
def reqBody = mockRequest?.getRequestContent()  
  
if (!reqBody || reqBody.trim().isEmpty()) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Empty request body"])  
 return "ValidationError"  
}  
  
def json  
try {  
 json = new JsonSlurper().parseText(reqBody)  
} catch (Exception e) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Invalid JSON format"])  
 return "ValidationError"  
}  
  
if (!json.accountId || !json.accountType || !json.customer || !json.branch || json.initialDeposit == null) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Missing required fields"])  
 return "ValidationError"  
}  
  
if (!(json.accountId ==~ /^ACC\d+$/)) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Invalid accountId"])  
 return "ValidationError"  
}  
  
if (!(json.accountType in ["Savings", "Current"])) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Invalid accountType"])  
 return "ValidationError"  
}  
  
def customer = json.customer  
if (!customer.name || !customer.email || !customer.dob || !customer.pan) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Missing customer details"])  
 return "ValidationError"  
}  
  
def emailRegex = /^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$/  
def panRegex = /^[A-Za-z0-9]{10}$/  
def dobRegex = /^\d{4}-\d{2}-\d{2}$/  
  
if (!(customer.email ==~ emailRegex)) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Invalid customer email"])  
 return "ValidationError"  
}  
  
if (!(customer.pan ==~ panRegex)) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Invalid customer PAN"])  
 return "ValidationError"  
}  
  
if (!(customer.dob ==~ dobRegex)) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Invalid DOB format"])  
 return "ValidationError"  
}  
  
def branch = json.branch  
if (!branch.branchId || !branch.ifsc) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Branch details required"])  
 return "ValidationError"  
}  
  
def branchRegex = /^BR\d+$/  
def ifscRegex = /^[A-Z]{4}0\d{6}$/  
  
if (!(branch.branchId ==~ branchRegex)) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Invalid branchId"])  
 return "ValidationError"  
}  
  
if (!(branch.ifsc ==~ ifscRegex)) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Invalid IFSC format"])  
 return "ValidationError"  
}  
  
if (json.initialDeposit < 1000) {  
 mockResponse.responseContent = JsonOutput.toJson([status: "fail", message: "Minimum deposit is ₹1000"])  
 return "ValidationError"  
}  
  
mockResponse.responseContent = JsonOutput.toJson([status: "success", message: "Account created successfully", data: json])  
return "Success"