# ARCHITECTURE

A[Vendor Portal] -->|Submit Data| B(Microservice)

B -->|REST/Kafka| C[Workflow Engine]

C --> D[PostgreSQL]

C --> E[Email Service]

C --> F[Approval Teams]

D --> C[State Recovery]

# DATABASE SCHEMA

-- Workflow Definitions

CREATE TABLE workflow\_definitions (

id TEXT PRIMARY KEY,

name TEXT NOT NULL,

version TEXT NOT NULL,

json\_config JSONB NOT NULL,

created\_at TIMESTAMPTZ DEFAULT NOW(),

is\_active BOOLEAN DEFAULT TRUE

);

-- Workflow Instances

CREATE TABLE workflow\_instances (

instance\_id TEXT PRIMARY KEY,

workflow\_id TEXT REFERENCES workflow\_definitions(id),

vendor\_id TEXT NOT NULL,

current\_status TEXT NOT NULL,

current\_phase TEXT,

context\_data JSONB,

created\_at TIMESTAMPTZ DEFAULT NOW(),

updated\_at TIMESTAMPTZ DEFAULT NOW()

);

-- Workflow Checkpoints

CREATE TABLE workflow\_checkpoints (

instance\_id TEXT REFERENCES workflow\_instances(instance\_id),

phase\_id TEXT NOT NULL,

step\_id TEXT NOT NULL,

status TEXT NOT NULL,

output\_data JSONB,

completed\_at TIMESTAMPTZ,

PRIMARY KEY (instance\_id, phase\_id, step\_id)

);

-- Pending Tasks

CREATE TABLE pending\_tasks (

task\_id TEXT PRIMARY KEY,

instance\_id TEXT REFERENCES workflow\_instances(instance\_id),

step\_id TEXT NOT NULL,

callback\_url TEXT NOT NULL,

status TEXT NOT NULL,

created\_at TIMESTAMPTZ DEFAULT NOW(),

expires\_at TIMESTAMPTZ

);

-- Failed Messages (For Recovery)

CREATE TABLE dead\_letters (

id SERIAL PRIMARY KEY,

message\_body JSONB NOT NULL,

source\_service TEXT NOT NULL,

error TEXT,

created\_at TIMESTAMPTZ DEFAULT NOW()

);

# WORK FLOW CORE MODELS

// WorkflowDefinition.cs

public class WorkflowDefinition

{

public string Id { get; set; }

public string Name { get; set; }

public string Version { get; set; }

public WorkflowConfig Config { get; set; }

}

public class WorkflowConfig

{

public List<WorkflowPhase> Phases { get; set; }

public Dictionary<string, object> Parameters { get; set; }

}

public class WorkflowPhase

{

public string PhaseId { get; set; }

public string Type { get; set; } // "sequential", "parallel"

public string Condition { get; set; }

public List<WorkflowStep> Steps { get; set; }

}

public class WorkflowStep

{

public string StepId { get; set; }

public string Action { get; set; } // "api\_call", "approval", "email"

public Dictionary<string, object> Config { get; set; }

public string Condition { get; set; }

}

# WORKF FLOW SERVICE

// WorkflowService.cs

public class WorkflowService

{

private readonly IWorkflowRepository \_repository;

private readonly IStepExecutor \_stepExecutor;

private readonly IConditionEvaluator \_conditionEvaluator;

public async Task<string> StartWorkflow(string workflowId, string vendorId, object initialData)

{

var instanceId = Guid.NewGuid().ToString();

await \_repository.CreateInstance(instanceId, workflowId, vendorId, initialData);

await ProcessNextSteps(instanceId);

return instanceId;

}

public async Task ProcessNextSteps(string instanceId)

{

var instance = await \_repository.GetInstance(instanceId);

var definition = await \_repository.GetDefinition(instance.WorkflowId);

foreach (var phase in definition.Config.Phases)

{

if (!\_conditionEvaluator.Evaluate(phase.Condition, instance.ContextData))

continue;

if (phase.Type == "sequential")

await ProcessSequentialPhase(instance, phase);

else

await ProcessParallelPhase(instance, phase);

}

}

private async Task ProcessSequentialPhase(WorkflowInstance instance, WorkflowPhase phase)

{

foreach (var step in phase.Steps)

{

if (!\_conditionEvaluator.Evaluate(step.Condition, instance.ContextData))

continue;

var result = await \_stepExecutor.Execute(step, instance);

await \_repository.SaveCheckpoint(

instance.InstanceId,

phase.PhaseId,

step.StepId,

result.Status,

result.Output);

if (!result.Success && step.Config["onFailure"] == "restartPhase")

break; // Will restart phase on next recovery

}

}

}

# STEPS EXECUTOR

// EmailStepExecutor.cs

public class EmailStepExecutor : IStepExecutor

{

public string ActionType => "email";

public async Task<StepResult> Execute(WorkflowStep step, WorkflowInstance instance)

{

var emailServiceUrl = step.Config["serviceUrl"].ToString();

var payload = new {

To = step.Config["recipient"],

Template = step.Config["template"],

Context = instance.ContextData

};

using var client = new HttpClient();

var response = await client.PostAsJsonAsync(emailServiceUrl, payload);

return new StepResult(

response.IsSuccessStatusCode,

await response.Content.ReadAsAsync<Dictionary<string, object>>());

}

}

// ApprovalStepExecutor.cs

public class ApprovalStepExecutor : IStepExecutor

{

public string ActionType => "approval";

public async Task<StepResult> Execute(WorkflowStep step, WorkflowInstance instance)

{

var taskId = Guid.NewGuid().ToString();

await \_repository.CreatePendingTask(

taskId,

instance.InstanceId,

step.StepId,

$"{\_config.CallbackBaseUrl}/approvals/{taskId}",

"pending");

return StepResult.Pending(); // Engine will wait for callback

}

}

# COMMUNICATION (OPTION A – REST API)

// WorkflowController.cs

[ApiController]

[Route("api/workflows")]

public class WorkflowController : ControllerBase

{

[HttpPost("start")]

public async Task<IActionResult> StartWorkflow(

[FromBody] StartWorkflowRequest request)

{

var instanceId = await \_workflowService.StartWorkflow(

request.WorkflowId,

request.VendorId,

request.InitialData);

return Ok(new { InstanceId = instanceId });

}

[HttpPost("callbacks/{taskId}")]

public async Task<IActionResult> HandleCallback(

string taskId,

[FromBody] CallbackData data)

{

await \_workflowService.ProcessCallback(taskId, data);

return Ok();

}

}

# ENGINE RECOVERY SYSTEM

// WorkflowRecoveryService.cs

public class WorkflowRecoveryService : BackgroundService

{

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

while (!stoppingToken.IsCancellationRequested)

{

var incompleteInstances = await \_repository.GetIncompleteInstances();

foreach (var instance in incompleteInstances)

{

await \_workflowService.ProcessNextSteps(instance.InstanceId);

}

await Task.Delay(TimeSpan.FromMinutes(5), stoppingToken);

}

}

}

# RECOVERY (INSIDE THE MICROSERVICE)

// In your microservice:

public async Task SubmitToWorkflowEngine(object data)

{

try

{

// Try REST first

var response = await \_httpClient.PostAsJsonAsync(

"http://wfe/api/workflows/start",

data);

if (!response.IsSuccessStatusCode)

throw new Exception("Workflow engine unavailable");

}

catch (Exception ex)

{

// Fallback to database

await \_db.ExecuteAsync(

@"INSERT INTO dead\_letters

(message\_body, source\_service, error)

VALUES (@Body, 'vendor-service', @Error)",

new { Body = JsonSerializer.Serialize(data), Error = ex.Message });

}

}

# WORK FLOW JSON (SAMPLE)

{

"id": "vendor-prequalification-v1",

"name": "Vendor Prequalification",

"phases": [

{

"phaseId": "initial-review",

"type": "sequential",

"condition": "true",

"steps": [

{

"stepId": "send-acknowledgement",

"action": "email",

"config": {

"serviceUrl": "http://email-service/send",

"template": "vendor-submission-ack",

"recipient": "{{vendor.email}}"

}

},

{

"stepId": "finance-approval",

"action": "approval",

"condition": "{{vendor.contractValue}} > 50000",

"config": {

"approvers": ["finance-team@company.com"],

"onFailure": "restartPhase"

}

}

]

},

{

"phaseId": "hse-review",

"type": "parallel",

"condition": "{{vendor.industry}} == 'construction'",

"steps": [

{

"stepId": "safety-check",

"action": "api\_call",

"config": {

"url": "http://hse-service/validate",

"method": "POST",

"successCondition": "response.status == 'approved'"

}

}

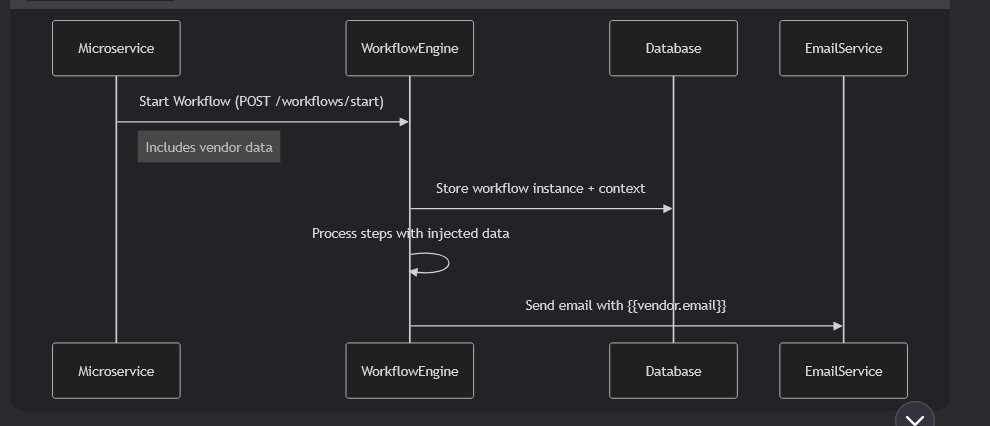
]

}

]

}

# EXAMPLE



Microservice initiating the work flow.

// In your Vendor Submission Microservice

public async Task SubmitVendor(VendorSubmissionRequest request)

{

// 1. Save vendor data to your database

var vendorId = await \_vendorRepository.SaveVendor(request);

// 2. Prepare workflow initial data

var workflowData = new {

vendor = new {

id = vendorId,

email = request.ContactEmail,

contractValue = request.EstimatedContractValue,

industry = request.BusinessCategory

},

documents = request.DocumentUrls

};

// 3. Start workflow (REST API call)

var response = await \_httpClient.PostAsJsonAsync(

"http://workflow-engine/api/workflows/start",

new {

WorkflowId = "vendor-prequalification-v1",

VendorId = vendorId,

InitialData = workflowData // <-- Critical payload

});

response.EnsureSuccessStatusCode();

}

Based on the initial values, prepare a json like this,



Save this information in the workflow engine as the initial data for the workflow instance. Take this Json and compare with the workflow instance json config and fill the required information as below.

public class TemplateEngine

{

public string Render(string template, object contextData)

{

return Regex.Replace(template, @"\{\{(.+?)\}\}", match =>

{

var path = match.Groups[1].Value;

return (string)JsonPathQuery(contextData, path);

});

}

private object JsonPathQuery(object data, string path)

{

// Simplified JSON path resolver

var parts = path.Split('.');

object current = data;

foreach (var part in parts)

{

if (current == null) return null;

var prop = current.GetType().GetProperty(part);

current = prop?.GetValue(current);

}

return current;

}

}

Email engine can use the data like the below.

public class EmailStepExecutor : IStepExecutor

{

public async Task<StepResult> Execute(WorkflowStep step, WorkflowInstance instance)

{

// Resolve templates

var resolvedRecipient = \_templateEngine.Render(

step.Config["recipient"].ToString(),

instance.ContextData);

var payload = new {

To = resolvedRecipient, // Now "vendor@example.com"

Template = step.Config["template"],

Context = instance.ContextData

};

// Send to email service

var response = await \_httpClient.PostAsJsonAsync(

step.Config["serviceUrl"].ToString(),

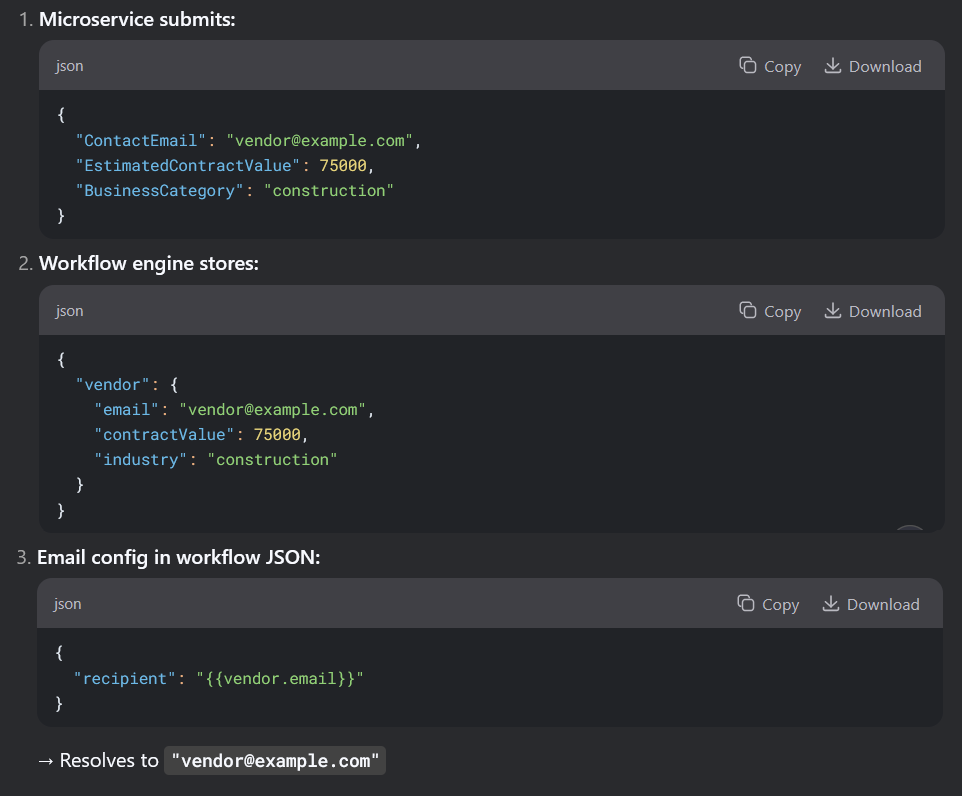
payload);

return new StepResult(response.IsSuccessStatusCode);

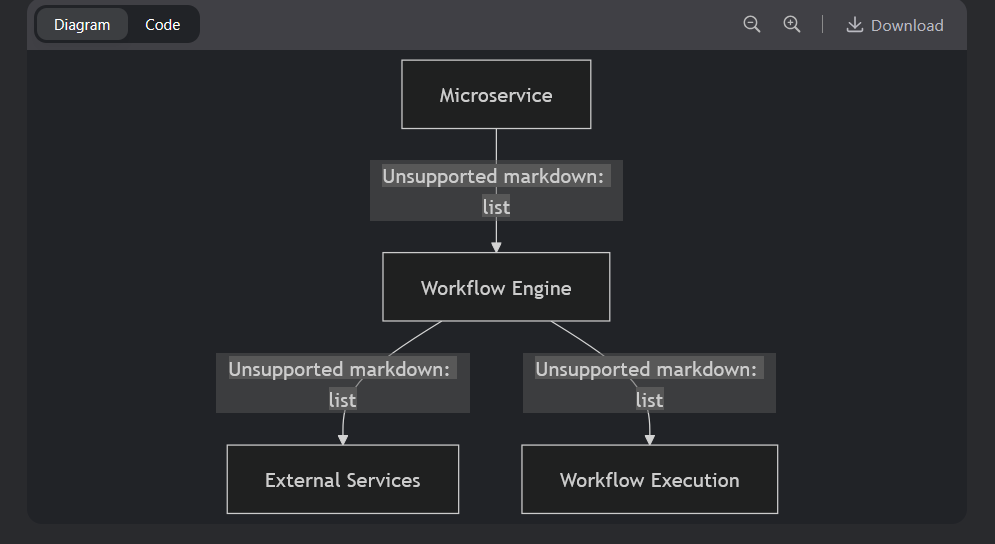
}

}

DATA FLOW:



# DECOUPLED DATA FLOW STRATEGY



Stragey

Microservice passes in only the minimal data required for initiating the workflow.

// Microservice initiates workflow with JUST the vendor ID

await \_workflowClient.StartWorkflow("vendor-prequal", new {

VendorId = vendor.Id // Only the essential ID

});

Workflow engine fetch additional data when needed.

{

"steps": [

{

"stepId": "enrich-vendor-data",

"action": "api\_call",

"config": {

"url": "http://vendor-service/vendors/{{vendorId}}",

"resultPath": "vendor" // Stores response here

}

},

{

"stepId": "send-email",

"action": "email",

"config": {

"recipient": "{{vendor.contactEmail}}", // Uses enriched data

"template": "welcome"

}

}

]

}

Allow for fall back values in the config definition.

{

"parameters": {

"approvalDeadlineDays": 7,

"requiredDocs": ["tax-id", "insurance-certificate"]

},

"steps": [

{

"config": {

"deadline": "{{parameters.approvalDeadlineDays}} days"

}

}

]

}

Enrichment concept:

