Certainly! Since the outline above is agent-agnostic, I will demonstrate how to turn this into a **template-style Agent Design Specification** by filling each section with instructive example text, placeholders, and “how-to-complete” tips. This structure will serve as a model for any agent (such as *ContentGen* or *Review*). For a real agent, simply replace the example text and placeholders with your agent-specific details.

**Agent Design Specification**

*VirtualAgentics Phase 1 — [Agent Name]*

**1. Document Control**

| **Version** | **Date** | **Author(s)** | **Reviewer(s)** | **Description** |
| --- | --- | --- | --- | --- |
| 0.1 | yyyy-mm-dd | <Name(s)> | <Name(s)> | Initial draft |

* **Related documents:**
  + System Architecture Overview
  + [Agent Communication and Events Spec]
  + [Security & Compliance Policies]

**2. Overview**

**2.1 Agent Purpose and Goals**

*Describe the agent’s function and business value in one or two sentences.*  
E.g.: “The ContentGen agent autonomously generates SEO-optimized articles in response to content requests, driving organic site traffic and revenue in Phase 1.”

**2.2 Context within VirtualAgentics**

* Sits under the **Marketing/Content** department.
* Works in tandem with the Review agent and Publish agent.
* Supports the company objective of autonomous content pipeline execution.

**2.3 Scope & Assumptions**

* **In scope:**
  + Generation of text content (articles, blog posts)
  + Handling event-based requests
* **Out of scope:**
  + Image/video content
  + Human approval workflows

**2.4 Dependencies**

* Upstream: Content requests from CMO agent
* Downstream: Events consumed by Review agent, content pushed to S3
* External: OpenAI API (content generation)

**3. Architecture & System Context**

**3.1 High-Level Context Diagram**

*(Insert or reference a diagram illustrating agent’s relation to other agents, services, databases, etc.)*

**3.2 Deployment Target**

* **Platform:** AWS Lambda (Python 3.11 runtime)
* **Environment:** Deployed to both dev and prod VPCs

**3.3 Runtime Environment & Resource Profile**

* Memory: 256 MB
* Timeout: 30s
* Concurrency: 2 (adjustable per workload)

**4. Interfaces**

**4.1 Event-Driven Interfaces**

**4.1.1 Subscribed Events**

| **Event Name** | **Topic** | **Schema Reference** | **Source** |
| --- | --- | --- | --- |
| ContentRequest | /content/request | content-request-v1.json | CMO agent |

**4.1.2 Published Events**

| **Event Name** | **Topic** | **Payload Schema** | **Destination(s)** |
| --- | --- | --- | --- |
| ContentReady | /content/ready | content-ready-v1.json | Review agent |

**4.2 Synchronous APIs / Webhooks**

*List any direct API endpoints (if none, state “N/A”).*

**4.3 Data-Store Interfaces**

* **DynamoDB:**
  + Table: va-phase1-content
  + Access: Write (new content metadata), Read (prior content checks)
* **S3:**
  + Bucket: va-phase1-content-objects
  + Path: /drafts/{id}.md (stores draft articles)

**4.4 External Service Calls**

* **OpenAI API:**
  + Endpoint: https://api.openai.com/v1/completions
  + Auth: Bearer token (from AWS Secrets Manager)
  + Throttling: 60 requests/minute per API key

**5. Inputs & Outputs**

**5.1 Input Catalogue**

| **Name** | **Format** | **Source** | **Required** | **Validation** |
| --- | --- | --- | --- | --- |
| content\_topic | string | event payload | Yes | Not empty, valid category |
| word\_count | integer | event payload | Yes | 300–2000 |
| prompt\_context | string | event payload | Optional | < 2k chars |

**5.2 Output Catalogue**

| **Name** | **Format** | **Destination** | **Consumer** |
| --- | --- | --- | --- |
| article\_text | Markdown | S3 bucket | Review agent |
| content\_id | UUID | DynamoDB | Audit, Analytics |
| event:ContentReady | event | SNS | Review agent |

**5.3 Error/Exception Outputs**

| **Error Name** | **Format** | **Output** | **Notes** |
| --- | --- | --- | --- |
| GenFailure | event | SNS /content/errors | Includes error code & message |
| ValidationError | event | SNS /content/errors | For invalid input parameters |

**6. Internal Processing Logic**

**6.1 Processing Flow Diagram / Pseudocode**

On ContentRequest Event:

- Validate input parameters

- Compose LLM prompt using template and event data

- Call OpenAI API

- On success: store article in S3, write metadata to DynamoDB

- Emit ContentReady event

- On error: emit GenFailure event

**6.2 Key Algorithms or Prompt Templates**

* LLM prompt: “Write a {word\_count}-word article on {content\_topic}. Context: {prompt\_context}.”
* Post-processing: Markdown formatting, length check

**6.3 Configuration Parameters**

* MAX\_WORDS: 2000
* OPENAI\_MODEL: gpt-3.5-turbo
* TIMEOUT\_SECONDS: 25

**6.4 Resource Utilization Expectations**

* CPU: negligible
* Memory: <128 MB typical
* OpenAI API tokens: ~2,000/request

**7. Lifecycle Management**

**7.1 Initialisation / Startup**

* Fetch secrets from AWS Secrets Manager
* Warm up OpenAI connection (optional)
* Verify S3/DynamoDB connectivity

**7.2 Runtime Behaviour Loop**

* Wait for ContentRequest event
* Handle single event per invocation (stateless)

**7.3 Error Handling & Recovery**

* Retries: 3 attempts on OpenAI errors (exponential backoff)
* Dead-letter: On repeated failure, send error event

**7.4 State Transitions & Persistence**

* State: Stateless (event-driven)
* Persistence: Article metadata in DynamoDB, content in S3

**7.5 Shutdown / Termination**

* Lambda handles shutdown natively (no special cleanup)

**7.6 Upgrade & Blue/Green Deployment Considerations**

* Use Lambda versioning & aliases for safe rollout

**8. Security & Compliance**

**8.1 IAM Role and Least-Privilege Policy Summary**

* Lambda execution role:
  + Allow: dynamodb:PutItem, s3:PutObject, secretsmanager:GetSecretValue
  + Restrict: resource ARNs to specific table, bucket, secret

**8.2 Secrets Management**

* Secret: /va/prod/openai-api-key
* Rotated manually every 90 days

**8.3 Data Classification & Encryption**

* S3 & DynamoDB: encrypted at rest (AES-256)
* All traffic TLS 1.2+

**8.4 Audit Logging Requirements**

* All content events and errors logged to CloudWatch
* API call logs via CloudTrail

**9. Observability**

**9.1 Logging**

* Structured JSON logs: timestamp, event type, request ID, error info
* No PII written to logs

**9.2 Metrics**

* ContentGen.SuccessCount
* ContentGen.FailureCount
* ContentGen.LatencyMs

**9.3 Distributed Tracing**

* [Optional/Planned] X-Ray enabled

**9.4 Alert Thresholds & Destinations**

* Alert if failure count > 5 in 10min (SNS to on-call)

**10. Performance & Scaling**

**10.1 Expected Workload Profile**

* 10–50 requests/day in Phase 1
* Burst: up to 10 in 1 minute

**10.2 Latency & Throughput Targets**

* <3 seconds avg. per request

**10.3 Auto-scaling Triggers and Limits**

* Lambda concurrency defaults to 2 (adjust if request volume increases)

**11. Testing Strategy**

**11.1 Unit Tests**

* Validate input parser (edge cases, invalid params)
* Mock OpenAI calls

**11.2 Integration Tests**

* Trigger end-to-end run in dev using test event
* Verify S3 & DynamoDB writes

**11.3 Contract Tests**

* Ensure event payload matches content-request-v1.json

**11.4 Load/Soak Tests**

* Simulate 10 requests/minute for 10 minutes in dev

**12. Operational Runbooks**

**12.1 Standard Deployment Steps**

* CI/CD pipeline: code push → PR review → merge to main → auto-deploy via GitHub Actions

**12.2 Rollback Procedure**

* Revert to previous Lambda version alias in AWS Console

**12.3 Common Incident Diagnostics**

* CloudWatch query:
  + filter @message like /ERROR/ | fields @timestamp, @message

**12.4 Known Limitations**

* Rate-limited by OpenAI API key
* Only supports English text

**13. Future Enhancements**

**13.1 Planned Phase 2+ Capabilities**

* Support images and multi-language generation
* Integrate retrieval-augmented generation (RAG)

**13.2 Technical Debt & Refactor Candidates**

* Refactor prompt construction for maintainability
* Add automated prompt quality validation

**End of Agent Design Specification**

**How to use:**

* For each new Phase 1 agent, duplicate this template and fill out each section using agent-specific details.
* Remove comments/instructions and placeholders as you go.
* For more complex agents, add sections or expand as needed to match functionality.
* Keep the format consistent for all agents in the system.