AI Service – Visual Handbook

Generated on 2025-10-02 13:12:02

This edition includes: a one■page endpoint reference, a table of contents, and for each endpoint an example use case, embedded image (if found), and the PUML source.

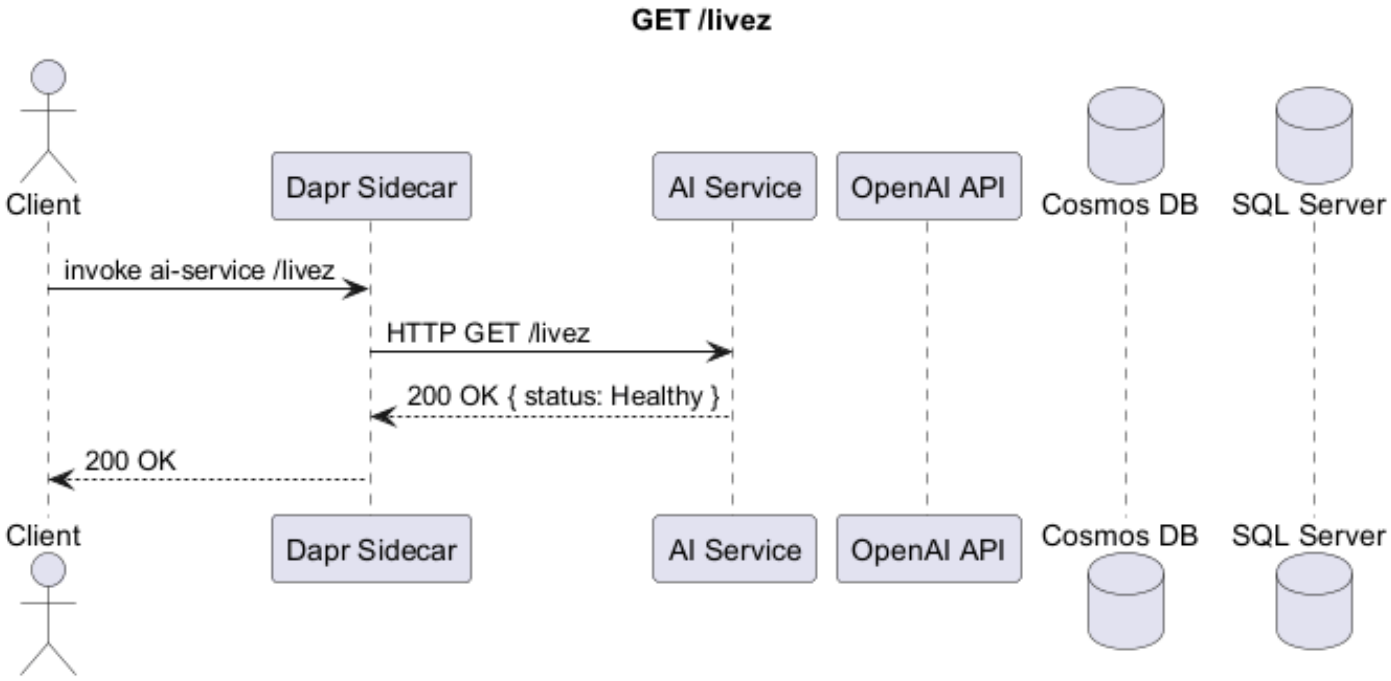
Endpoint → Primary Use Case

Endpoint	Primary Use Case	Requester	Response Shape
/livez	K8s/Compose liveness probe	Cluster/DevOps	{ status }
/readyz	Readiness check (OpenAI, Cosmos)	Cluster/DevOps	{ status, deps }
/version	Check build + models running	Ops/Admin	{ gitSha, buildTime }
/v1/jobs/:companyId/:jobId/enhance	Polish recruiter’s draft	Employer/Admin UI	{ enhanced fields }
/v1/jobs/:companyId/generate	Generate draft from brief	Employer/Admin UI	{ job draft }
/v1/jobs/:companyId/:jobId/rewrite	Targeted edits	Employer/Admin UI	{ rewritten sections }
/v1/embeddings	Embed arbitrary text	AI Service/Dev	{ vectors }
/v1/jobs/:companyId/:jobId/embed	Persist job embedding	Employer/Admin API	{ embeddingId }
/v1/resumes/:companyId/:resumeId/embed	Persist resume embedding	Employer/Admin API	{ embeddingId }
/v1/search/semantic	Semantic search jobs/resumes	Recruiter UI	{ hits[], tookMs }
/v1/resumes/:companyId/parse	Parse resume (PDF/DOC)	Candidate Upload / Admin API	{ resumeId, sections }
/v1/resumes/:companyId/:resumeId/summarize	Summarize resume	Recruiter UI	{ summary, keywords }
/v1/resumes/:companyId/:resumeId/redact	Remove PII for fair screening	Admin API	{ redactedText }
/v1/match/score	Score 1 job vs 1 resume	Recruiter UI	{ score, reasons }
/v1/match/batch	Batch match (shortlists)	Recruiter UI	{ ranked lists }
/v1/events/job.created	Enhance/embed after new job	Admin API (event)	{ tasks }
/v1/events/job.updated	Re-embed after edits	Admin API (event)	{ tasks }
/v1/events/resume.uploaded	Parse/embed/match new resume	Admin API (event)	{ tasks }
/v1/tasks	Submit async task	Admin/DevOps	{ taskId }
/v1/tasks/:taskId	Check async status	Admin/DevOps	{ status, result }
/v1/tasks/:taskId (DELETE)	Cancel task	Admin/DevOps	{ cancelled }
/v1/text/rewrite	Rewrite arbitrary text	Any UI	{ text }
/v1/text/extract	Extract entities	Admin/Recruiter	{ entities }
/v1/stats	Service stats/usage	Admin/Ops	{ counts, tokens }
/v1/models	List available models	Admin/Ops	{ models }
/v1/config	Safe runtime config	Admin/Ops	{ settings }
/v1/testdata/resumes	Generate synthetic resumes	Admin (ai.admin)	{ ids[], zipUrl }
/v1/content/check	Moderation for jobs/resumes	Employer/Admin API	{ allowed, categories }

Table of Contents

1. GET /livez (livez.puml)

Example use case: Kubernetes/Compose liveness probe to confirm the process is up. Use during deploys and restarts.

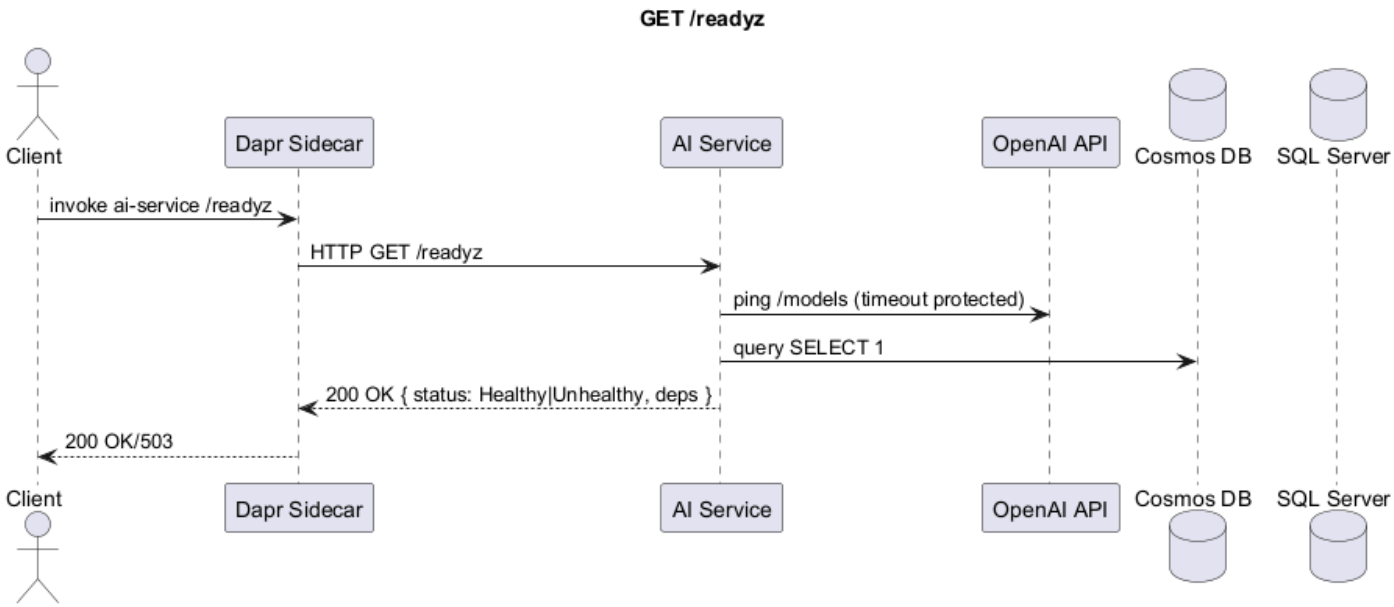



```
@startuml
title GET /livez
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: invoke ai-service /livez
Dapr -> AIS: HTTP GET /livez
AIS --> Dapr: 200 OK { status: Healthy }
Dapr --> Client: 200 OK
@enduml
```

2. GET /readyz (readyz.puml)

Example use case: Readiness probe that pings OpenAI and Cosmos. Use after changing keys, firewall, or throughput.

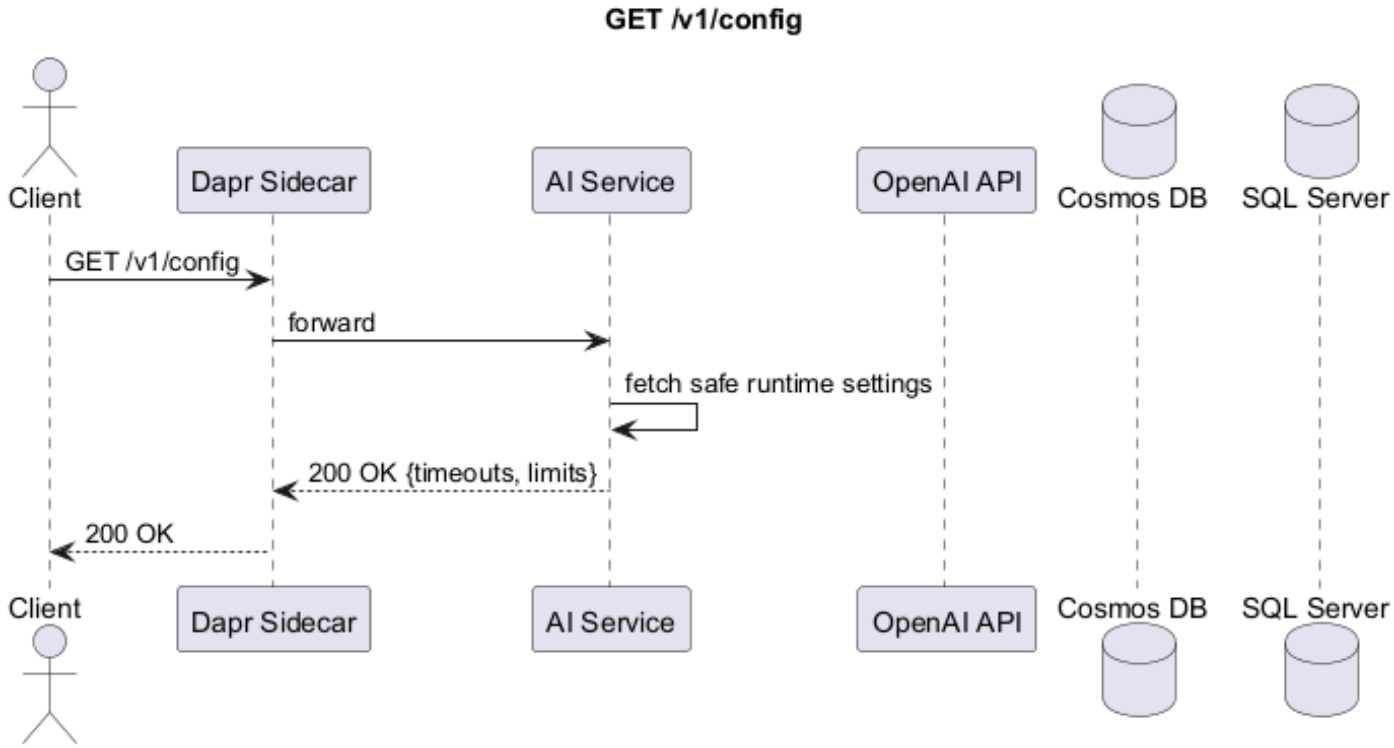



```
@startuml
title GET /readyz
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: invoke ai-service /readyz
Dapr -> AIS: HTTP GET /readyz
AIS -> OpenAI: ping /models (timeout protected)
AIS -> Cosmos: query SELECT 1
AIS --> Dapr: 200 OK { status: Healthy|Unhealthy, deps }
Dapr --> Client: 200 OK/503
@enduml
```

3. GET /v1/config (v1-config.puml)

Example use case: Expose safe runtime settings (timeouts, limits) to help UI adapt to env.

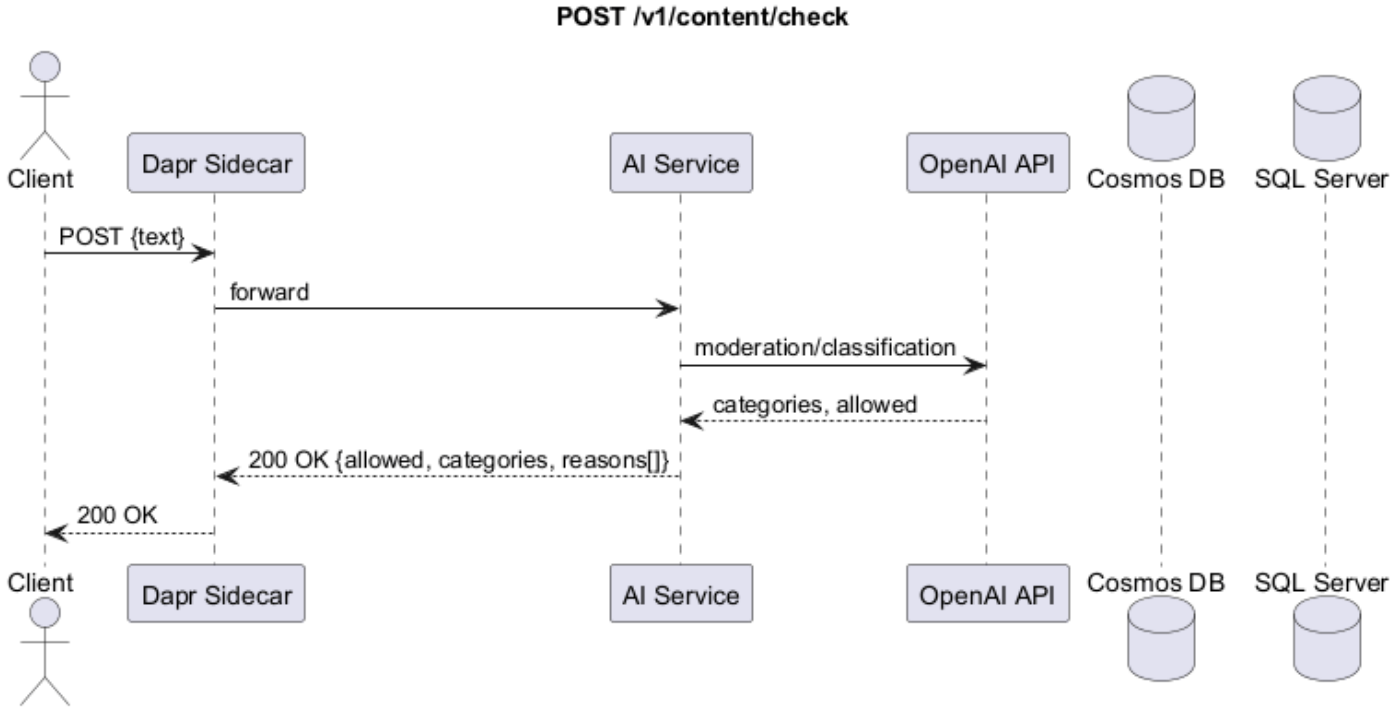



```
@startuml
title GET /v1/config
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: GET /v1/config
Dapr -> AIS: forward
AIS -> AIS: fetch safe runtime settings
AIS --> Dapr: 200 OK {timeouts, limits}
Dapr --> Client: 200 OK
@enduml
```

4. POST /v1/content/check (v1-content-check.puml)

Example use case: Moderation check to block unsafe/biased language before external publishing.

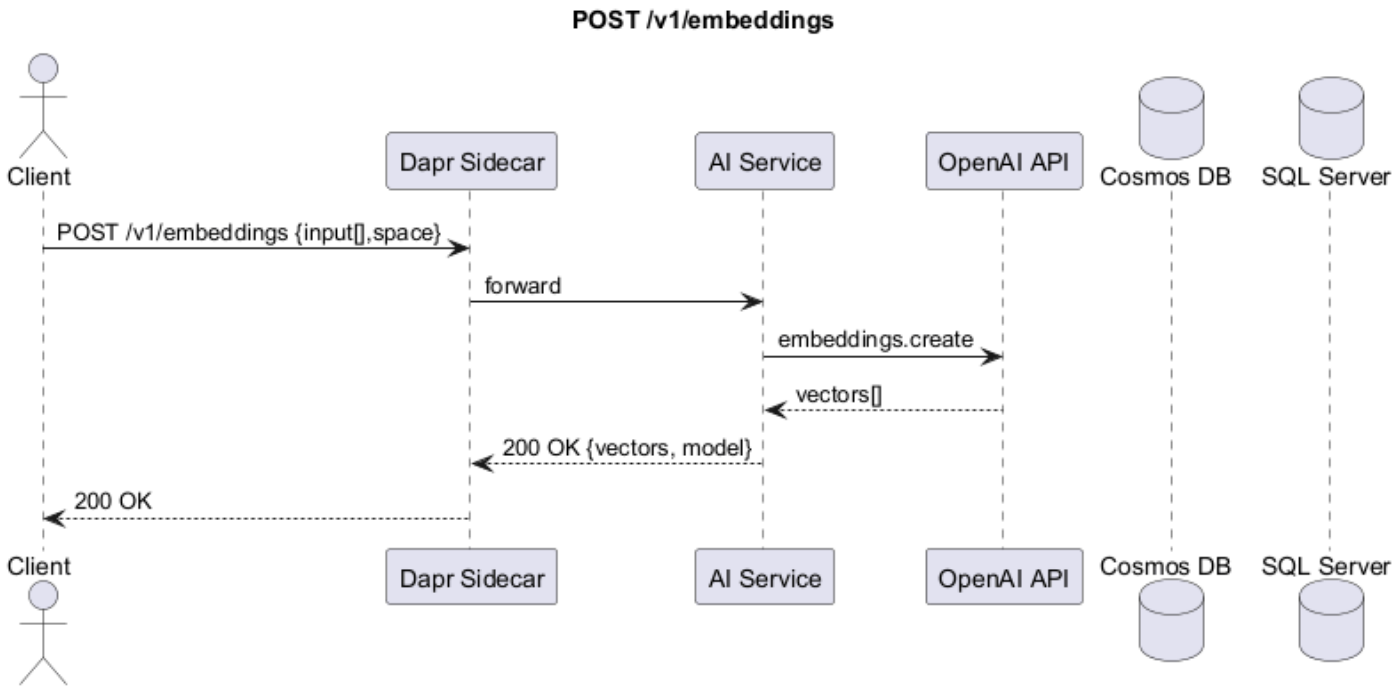



```
@startuml
title POST /v1/content/check
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST {text}
Dapr -> AIS: forward
AIS -> OpenAI: moderation/classification
OpenAI --> AIS: categories, allowed
AIS --> Dapr: 200 OK {allowed, categories, reasons[]}
Dapr --> Client: 200 OK
@enduml
```

5. POST /v1/embeddings (v1-embeddings.puml)

Example use case: Ad-hoc vectorize arbitrary text for experiments or precomputation on a new corpus.

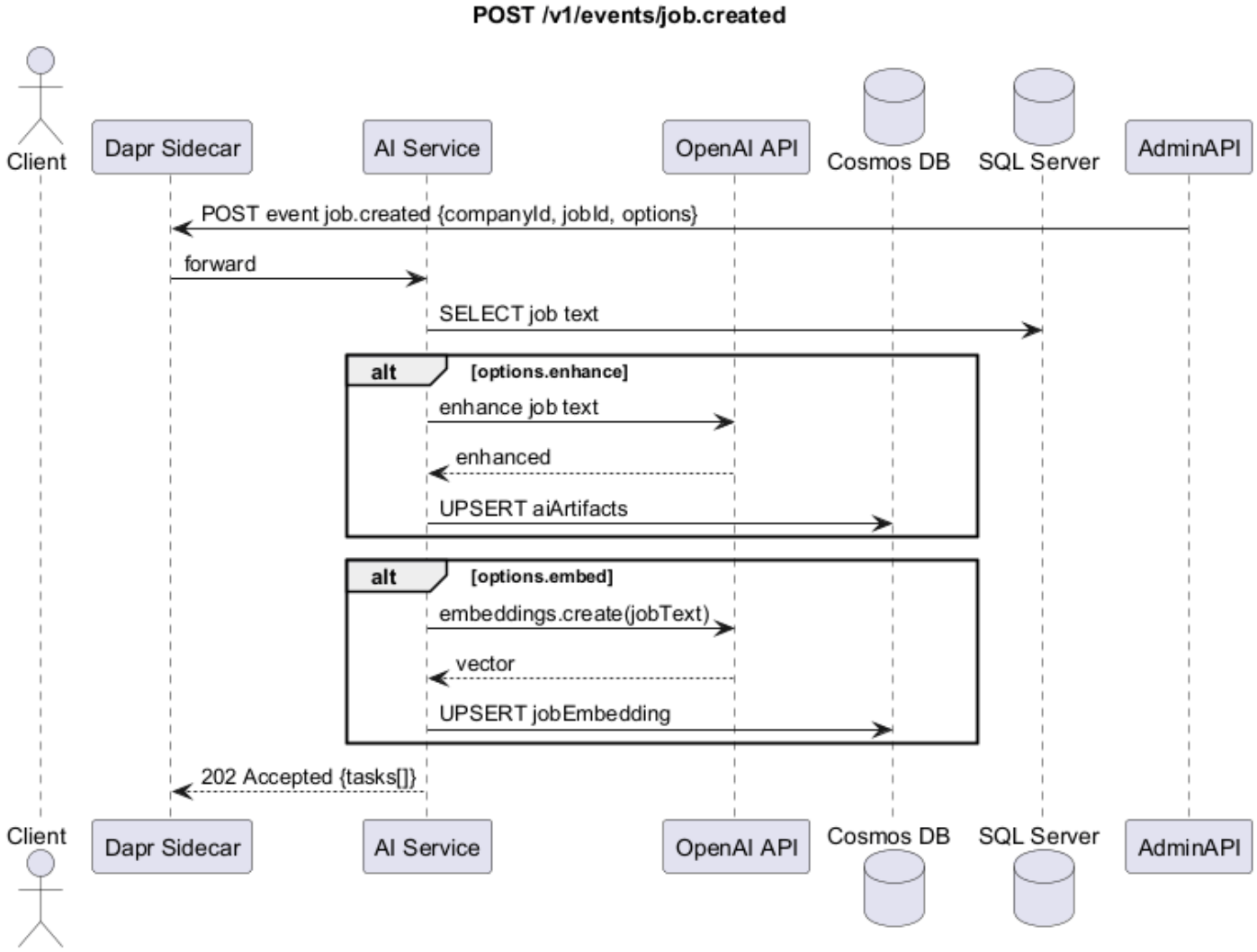



```
@startuml
title POST /v1/embeddings
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST /v1/embeddings {input[],space}
Dapr -> AIS: forward
AIS -> OpenAI: embeddings.create
OpenAI --> AIS: vectors[]
AIS --> Dapr: 200 OK {vectors, model}
Dapr --> Client: 200 OK
@enduml
```

6. POST /v1/events/job.created (v1-events-job-created.puml)

Example use case: On new job: enhance text and/or create embeddings automatically via event.

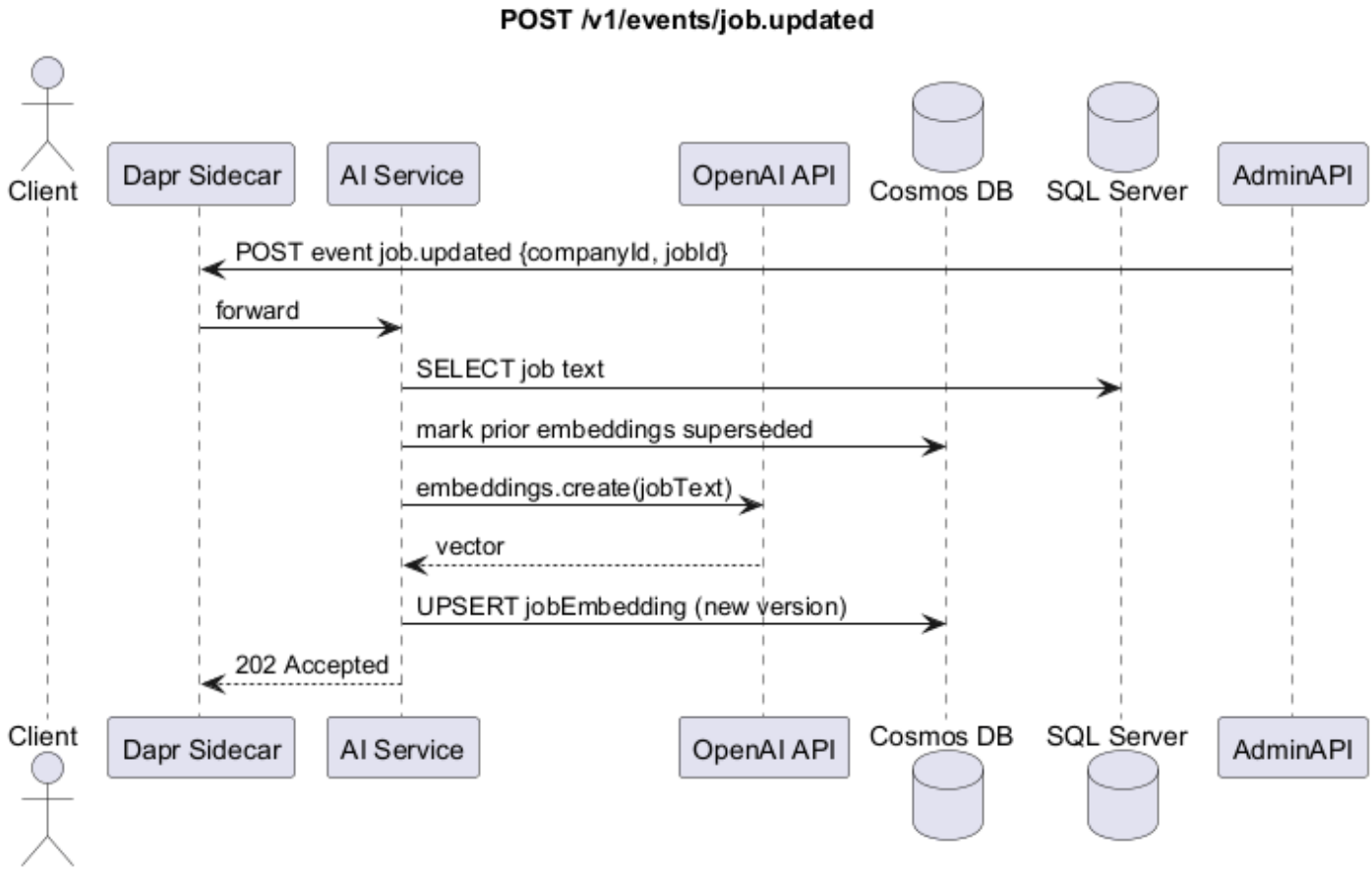



```
@startuml
title POST /v1/events/job.created
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

AdminAPI -> Dapr: POST event job.created {companyId, jobId, options}
Dapr -> AIS: forward
AIS -> SQL: SELECT job text
alt options.enhance
  AIS -> OpenAI: enhance job text
  OpenAI --> AIS: enhanced
  AIS -> Cosmos: UPSERT aiArtifacts
end
alt options.embed
  AIS -> OpenAI: embeddings.create(jobText)
  OpenAI --> AIS: vector
  AIS -> Cosmos: UPSERT jobEmbedding
end
AIS --> Dapr: 202 Accepted {tasks[]}
@enduml
```

7. POST /v1/events/job.updated (v1-events-job-updated.puml)

Example use case: When a job is updated, re-embed and invalidate cached matches if needed.

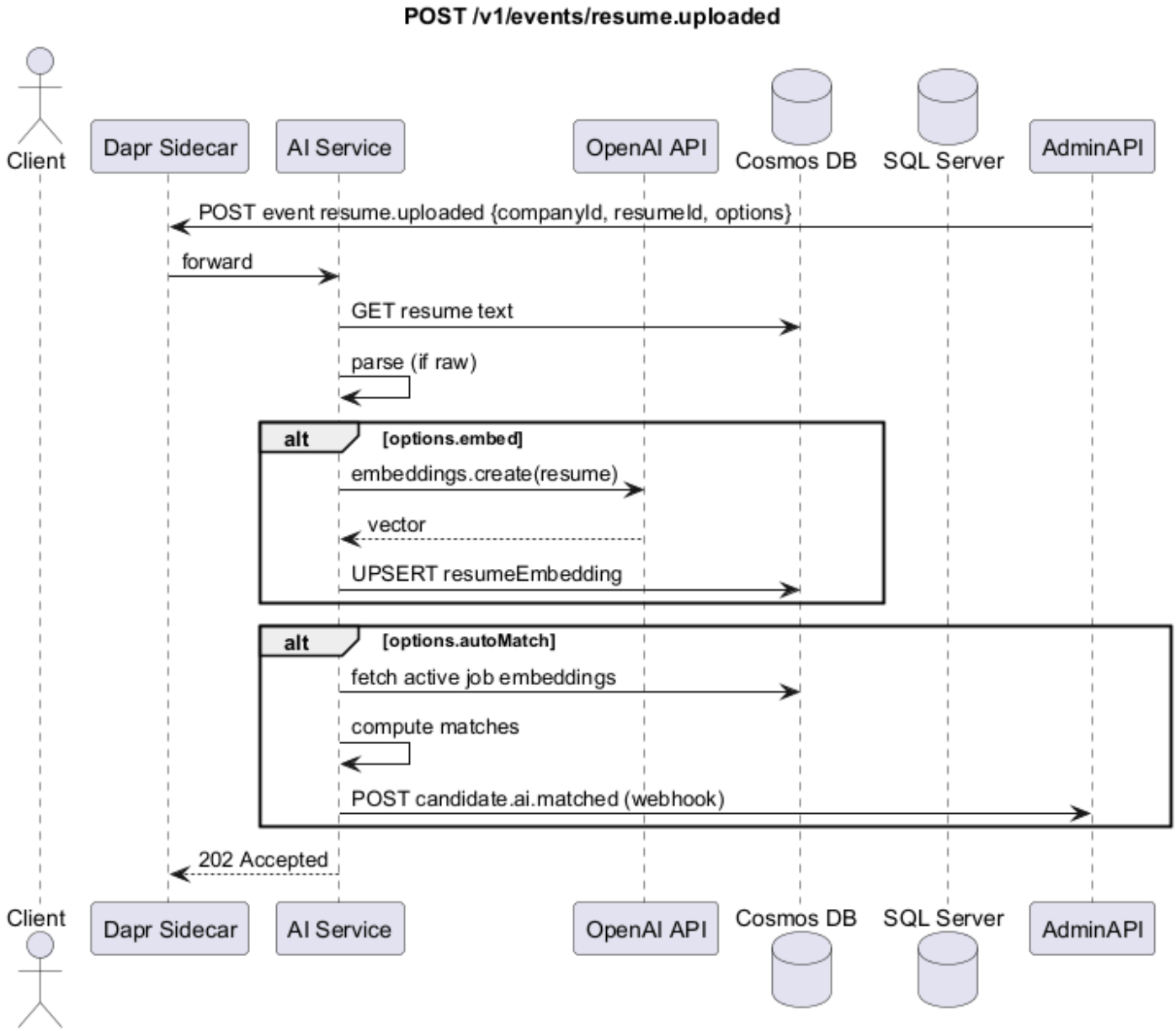



```
@startuml
title POST /v1/events/job.updated
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

AdminAPI -> Dapr: POST event job.updated {companyId, jobId}
Dapr -> AIS: forward
AIS -> SQL: SELECT job text
AIS -> Cosmos: mark prior embeddings superseded
AIS -> OpenAI: embeddings.create(jobText)
OpenAI --> AIS: vector
AIS -> Cosmos: UPSERT jobEmbedding (new version)
AIS --> Dapr: 202 Accepted
@enduml
```

8. POST /v1/events/resume.uploaded (v1-events-resume-uploaded.puml)

Example use case: On new resume: parse → embed → optional auto-match; notify Admin API.



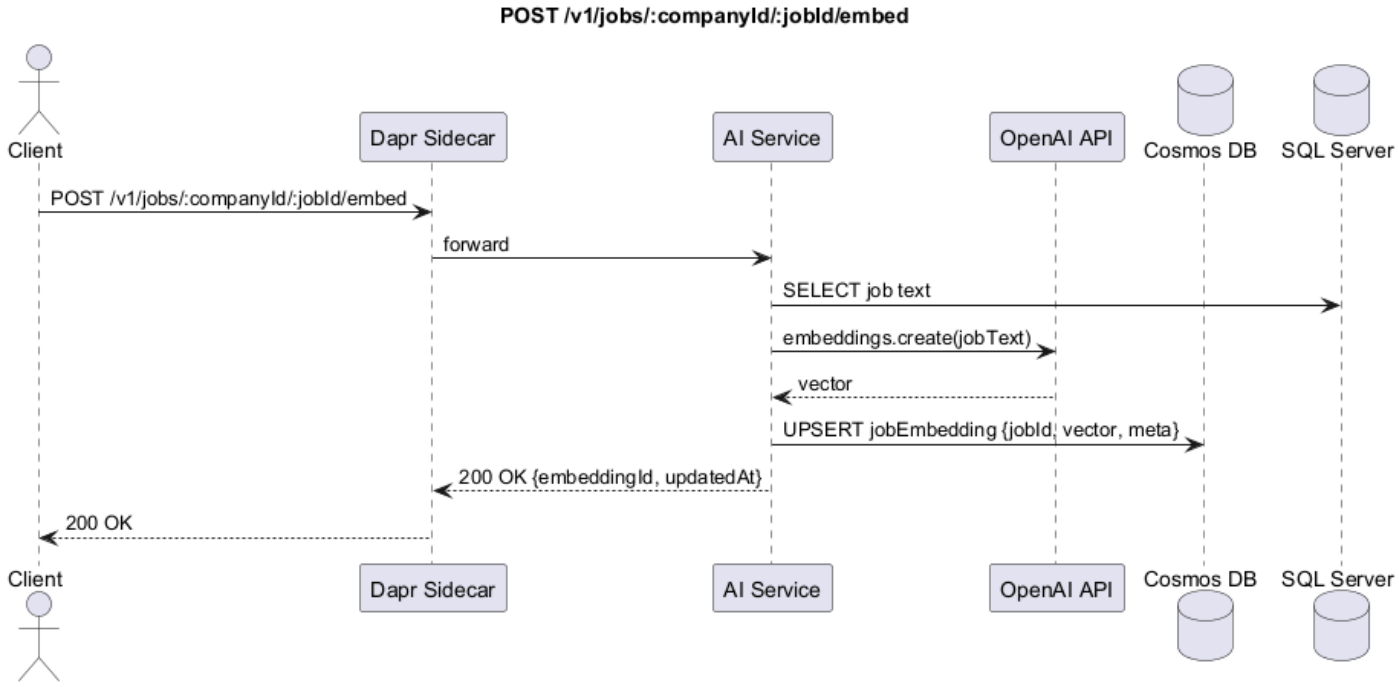

```
@startuml
title POST /v1/events/resume.uploaded
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

AdminAPI -> Dapr: POST event resume.uploaded {companyId, resumeId, options}
Dapr -> AIS: forward
AIS -> Cosmos: GET resume text
AIS -> AIS: parse (if raw)
alt options.embed
  AIS -> OpenAI: embeddings.create(resume)
  OpenAI --> AIS: vector
  AIS -> Cosmos: UPSERT resumeEmbedding
```

```
end
alt options.autoMatch
  AIS -> Cosmos: fetch active job embeddings
  AIS -> AIS: compute matches
  AIS -> AdminAPI: POST candidate.ai.matched (webhook)
end
AIS --> Dapr: 202 Accepted
@enduml
```

9. POST /v1/jobs/:companyId/:jobId/embed (v1-jobs-embed.puml)

Example use case: Refresh a job's embedding when the description changes or on job.created event.

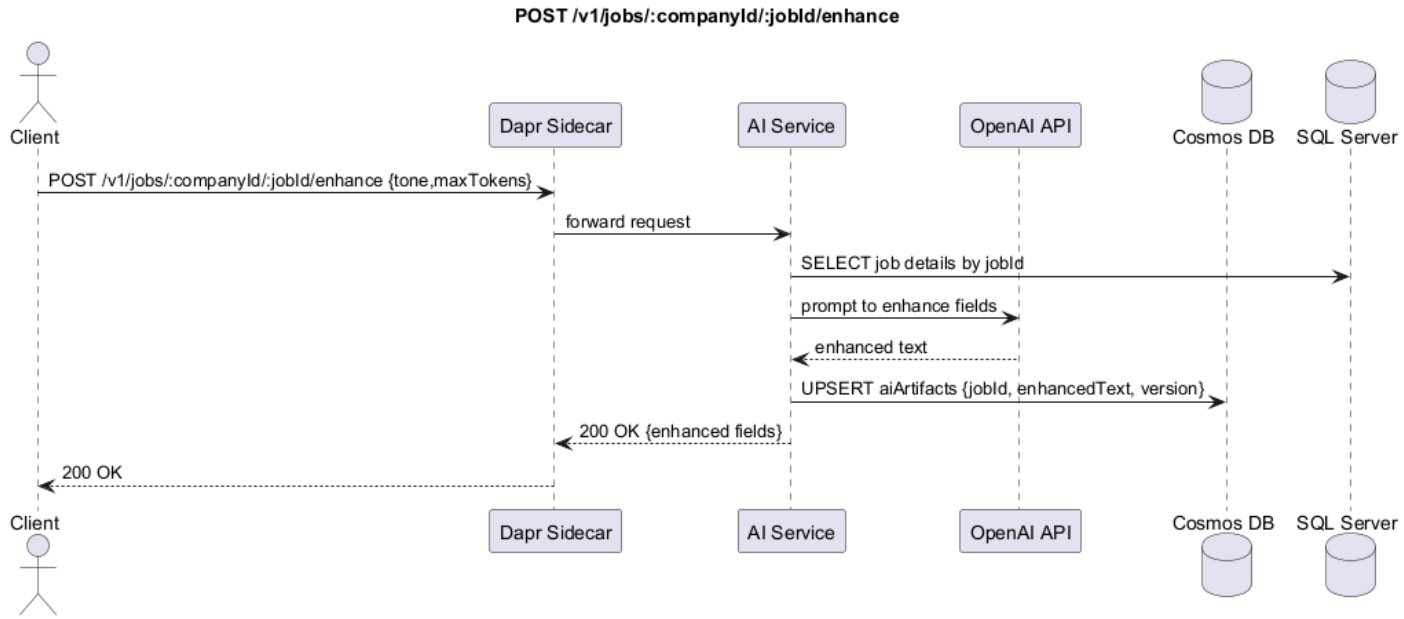



```
@startuml
title POST /v1/jobs/:companyId/:jobId/embed
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST /v1/jobs/:companyId/:jobId/embed
Dapr -> AIS: forward
AIS -> SQL: SELECT job text
AIS -> OpenAI: embeddings.create(jobText)
OpenAI --> AIS: vector
AIS -> Cosmos: UPSERT jobEmbedding {jobId, vector, meta}
AIS --> Dapr: 200 OK {embeddingId, updatedAt}
Dapr --> Client: 200 OK
@enduml
```

10. POST /v1/jobs/:companyId/:jobId/enhance (v1-jobs-enhance.puml)

Example use case: After a recruiter saves a rough draft, auto-polish tone and structure before publishing.

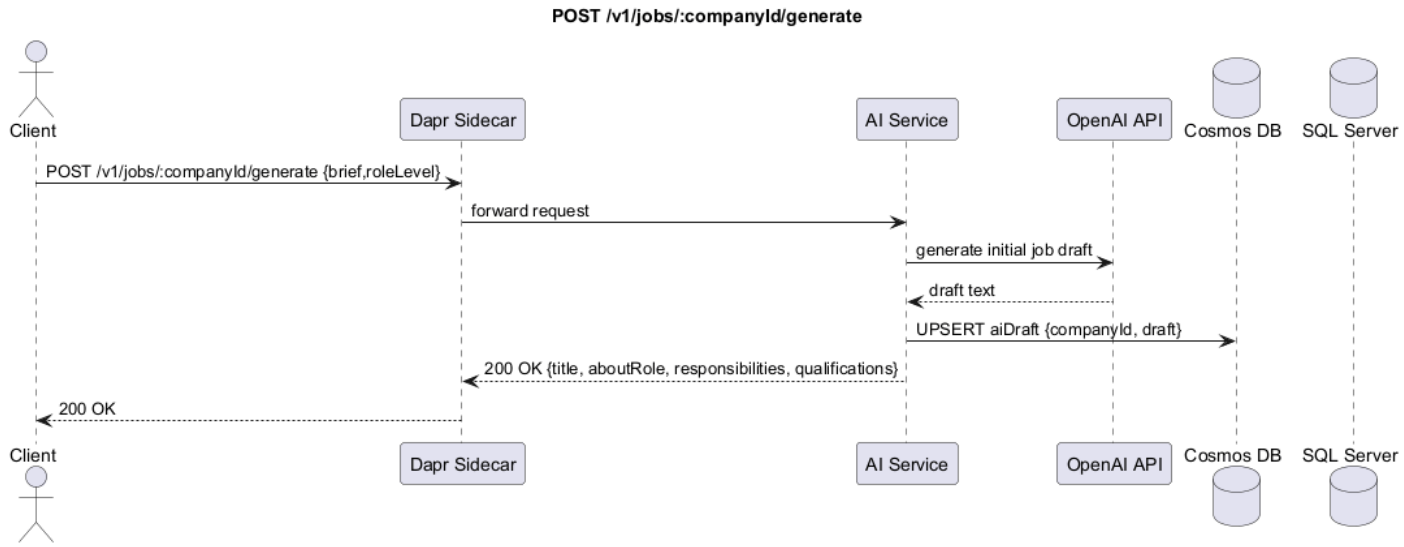



```
@startuml
title POST /v1/jobs/:companyId/:jobId/enhance
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST /v1/jobs/:companyId/:jobId/enhance {tone,maxTokens}
Dapr -> AIS: forward request
AIS -> SQL: SELECT job details by jobId
AIS -> OpenAI: prompt to enhance fields
OpenAI --> AIS: enhanced text
AIS -> Cosmos: UPSERT aiArtifacts {jobId, enhancedText, version}
AIS --> Dapr: 200 OK {enhanced fields}
Dapr --> Client: 200 OK
@enduml
```

11. POST /v1/jobs/:companyId/generate (v1-jobs-generate.puml)

Example use case: Turn a short brief from a hiring manager into a structured first-draft job posting.

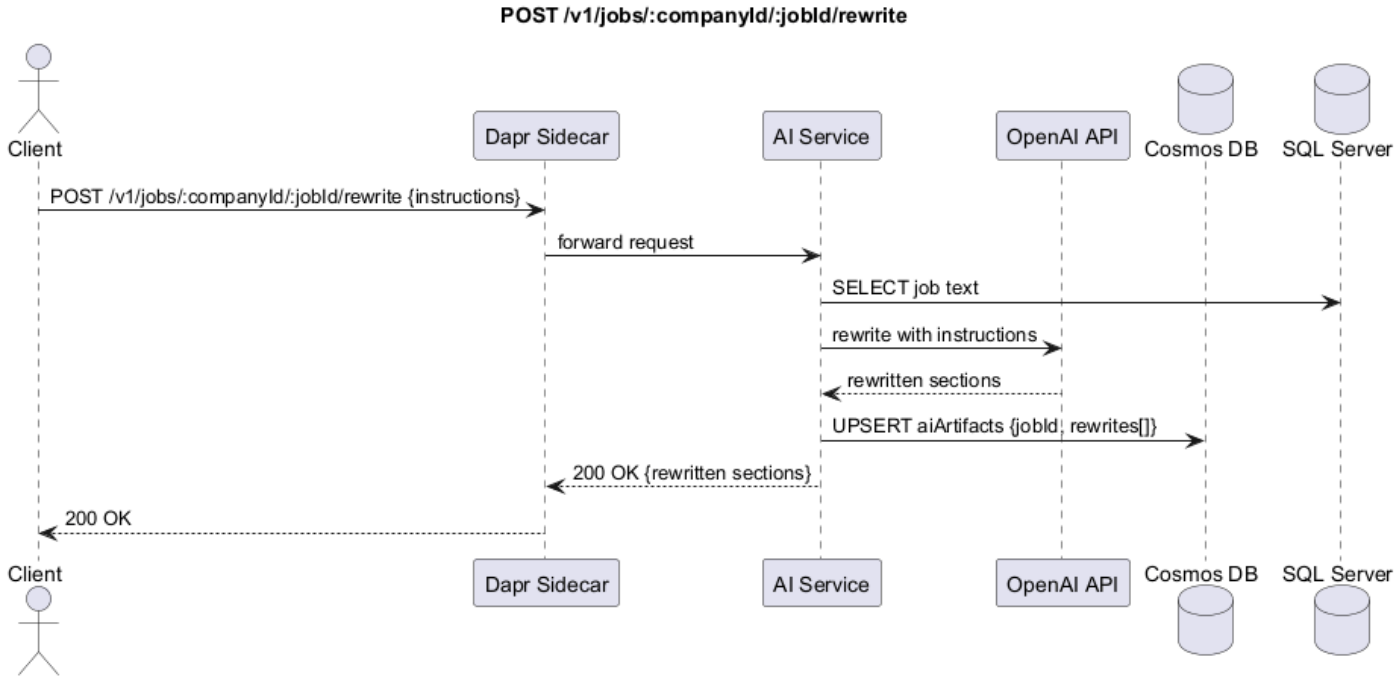



```
@startuml
title POST /v1/jobs/:companyId/generate
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST /v1/jobs/:companyId/generate {brief,roleLevel}
Dapr -> AIS: forward request
AIS -> OpenAI: generate initial job draft
OpenAI --> AIS: draft text
AIS -> Cosmos: UPSERT aiDraft {companyId, draft}
AIS --> Dapr: 200 OK {title, aboutRole, responsibilities, qualifications}
Dapr --> Client: 200 OK
@enduml
```

12. POST /v1/jobs/:companyId/:jobId/rewrite (v1-jobs-rewrite.puml)

Example use case: Targeted edits like 'remove buzzwords' or 'make concise' for a specific job section.

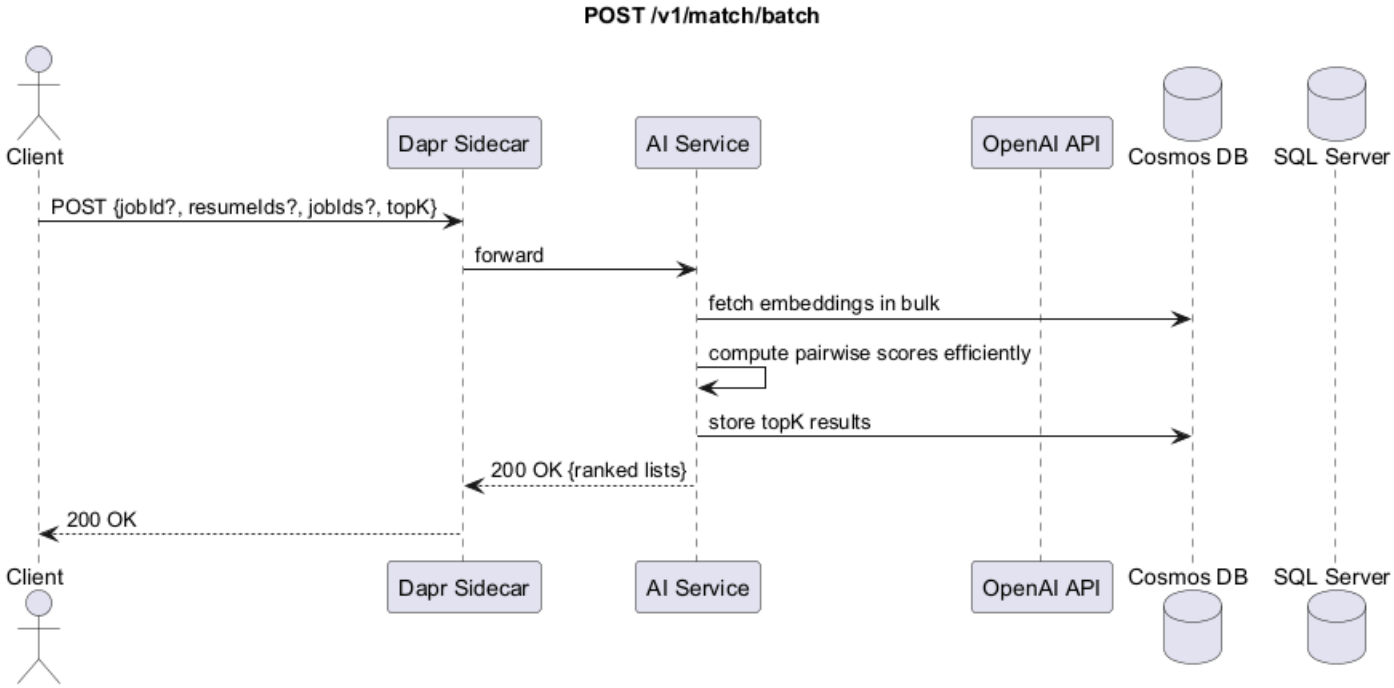



```
@startuml
title POST /v1/jobs/:companyId/:jobId/rewrite
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST /v1/jobs/:companyId/:jobId/rewrite {instructions}
Dapr -> AIS: forward request
AIS -> SQL: SELECT job text
AIS -> OpenAI: rewrite with instructions
OpenAI --> AIS: rewritten sections
AIS -> Cosmos: UPSERT aiArtifacts {jobId, rewrites[]}
AIS --> Dapr: 200 OK {rewritten sections}
Dapr --> Client: 200 OK
@enduml
```

13. POST /v1/match/batch (v1-match-batch.puml)

Example use case: Rank many candidates for a job (shortlist) or show a candidate's top-N matching jobs.

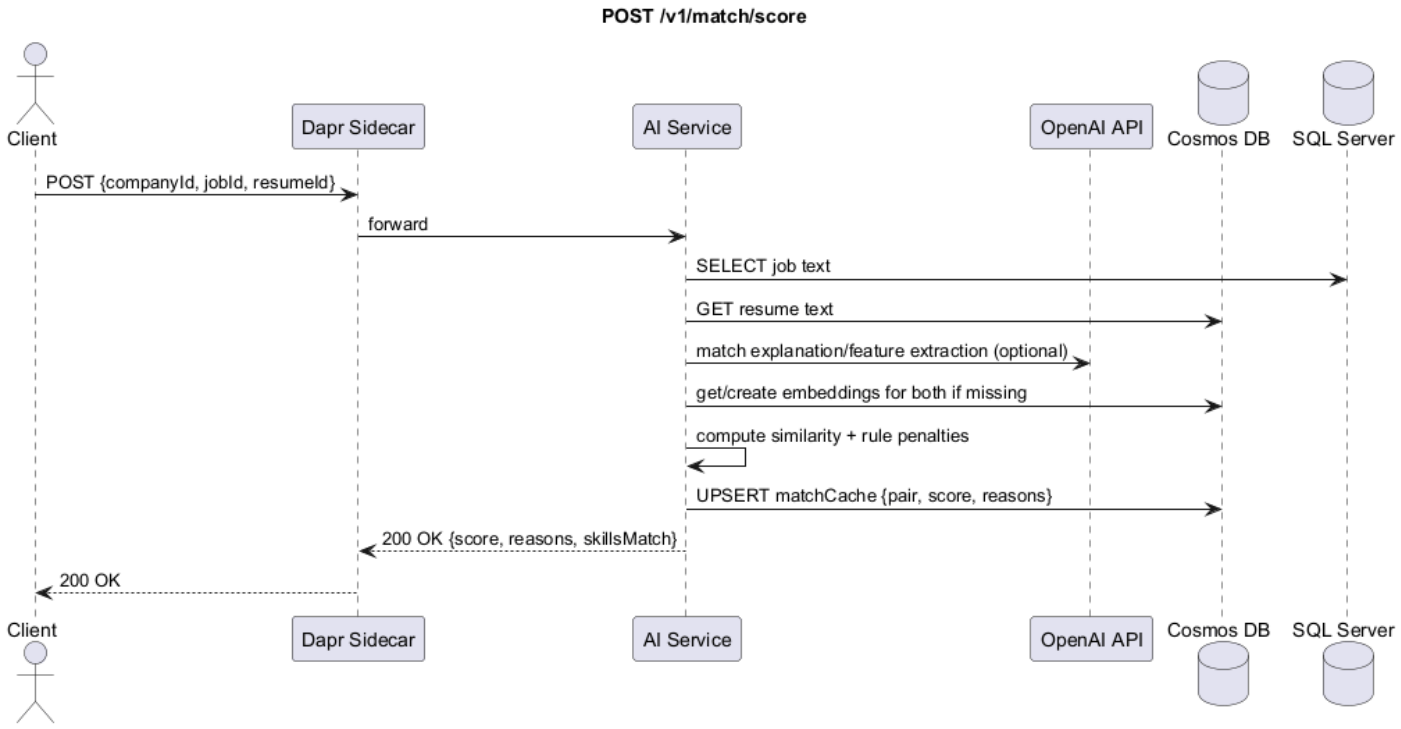



```
@startuml
title POST /v1/match/batch
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST {jobId?, resumeIds?, jobIds?, topK}
Dapr -> AIS: forward
AIS -> Cosmos: fetch embeddings in bulk
AIS -> AIS: compute pairwise scores efficiently
AIS -> Cosmos: store topK results
AIS --> Dapr: 200 OK {ranked lists}
Dapr --> Client: 200 OK
@enduml
```

14. POST /v1/match/score (v1-match-score.puml)

Example use case: Compute a fit score for one job vs one resume when viewing a candidate profile.

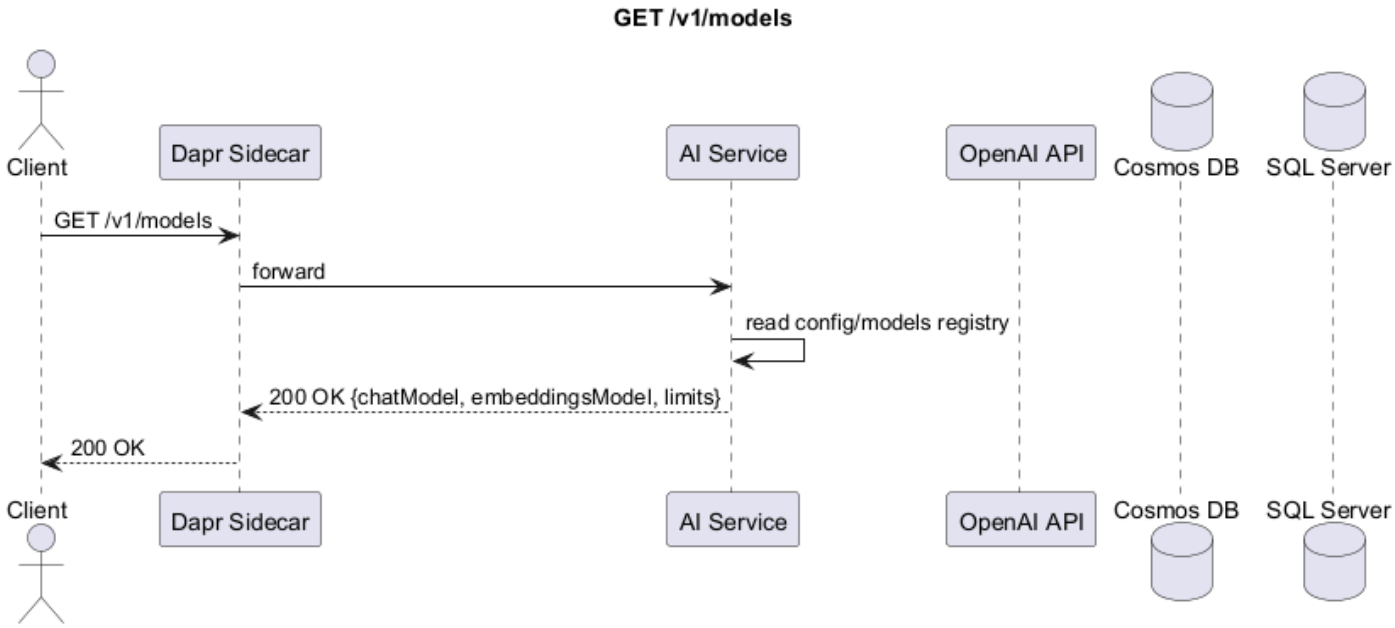



```
@startuml
title POST /v1/match/score
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST {companyId, jobId, resumeId}
Dapr -> AIS: forward
AIS -> SQL: SELECT job text
AIS -> Cosmos: GET resume text
AIS -> OpenAI: match explanation/feature extraction (optional)
AIS -> Cosmos: get/create embeddings for both if missing
AIS -> AIS: compute similarity + rule penalties
AIS -> Cosmos: UPSERT matchCache {pair, score, reasons}
AIS --> Dapr: 200 OK {score, reasons, skillsMatch}
Dapr --> Client: 200 OK
@enduml
```

15. GET /v1/models (v1-models.puml)

Example use case: List enabled chat/embedding models and limits; useful for feature flags.

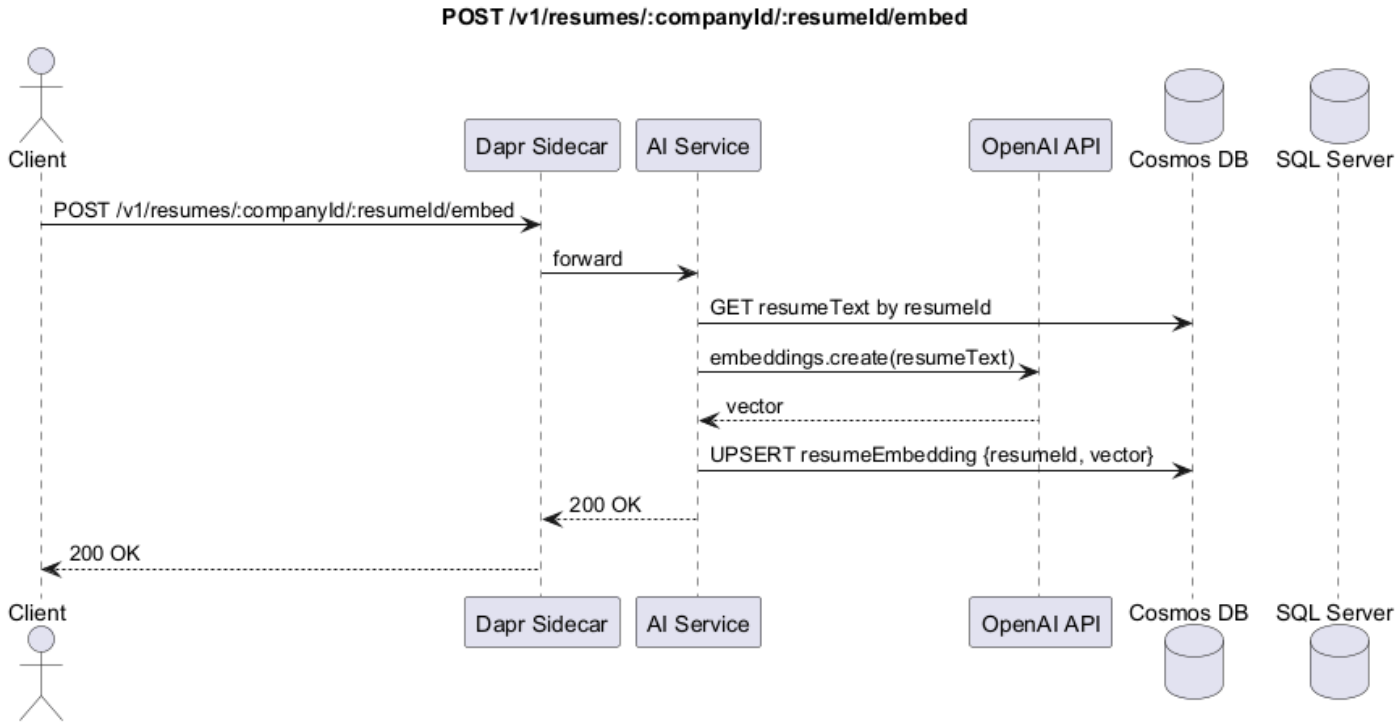



```
@startuml
title GET /v1/models
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: GET /v1/models
Dapr -> AIS: forward
AIS -> AIS: read config/models registry
AIS --> Dapr: 200 OK {chatModel, embeddingsModel, limits}
Dapr --> Client: 200 OK
@enduml
```

16. POST /v1/resumes/:companyId/:resumeId/embed (v1-resumes-embed.puml)

Example use case: Compute and persist a resume's vector immediately after parsing or profile updates.

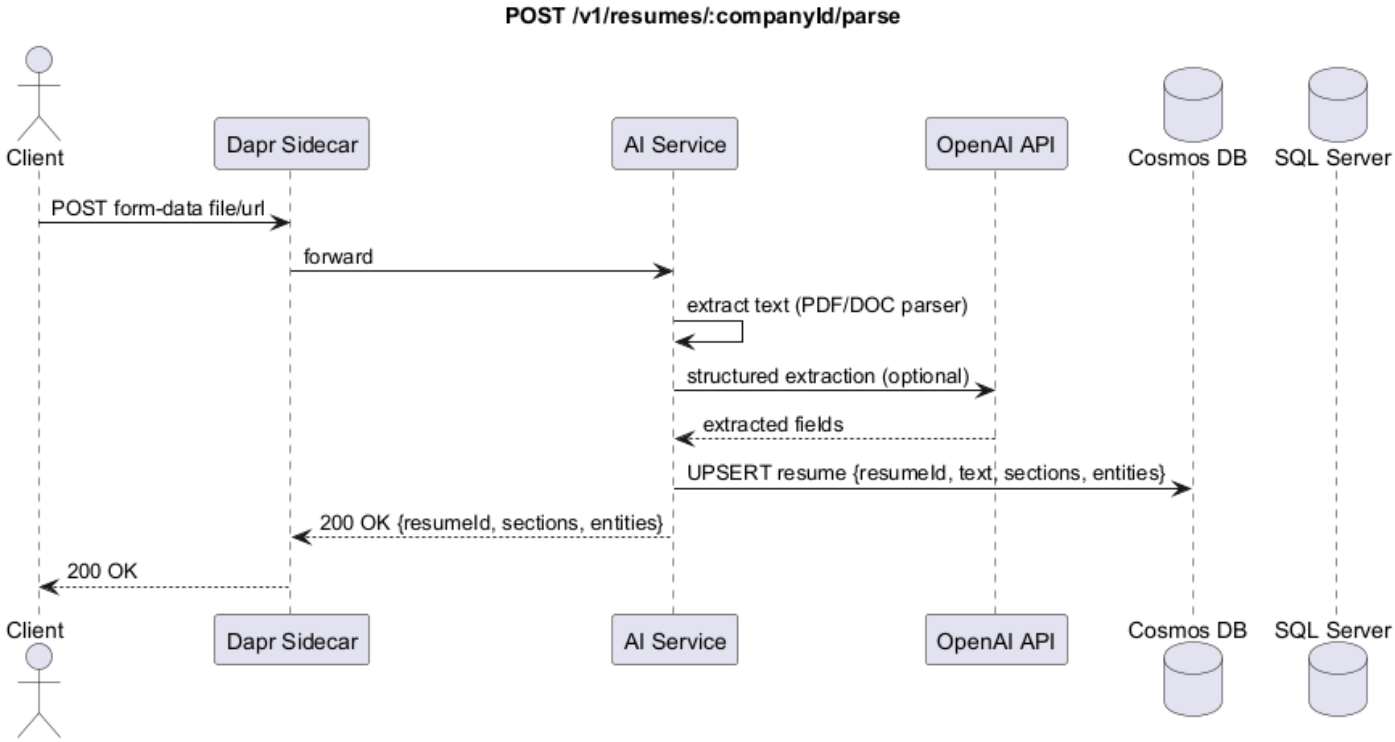



```
@startuml
title POST /v1/resumes/:companyId/:resumeId/embed
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST /v1/resumes/:companyId/:resumeId/embed
Dapr -> AIS: forward
AIS -> Cosmos: GET resumeText by resumeId
AIS -> OpenAI: embeddings.create(resumeText)
OpenAI --> AIS: vector
AIS -> Cosmos: UPSERT resumeEmbedding {resumeId, vector}
AIS --> Dapr: 200 OK
Dapr --> Client: 200 OK
@enduml
```

17. POST /v1/resumes/:companyId/parse (v1-resumes-parse.puml)

Example use case: Upload a PDF/DOC or URL and extract structured sections/entities for indexing.

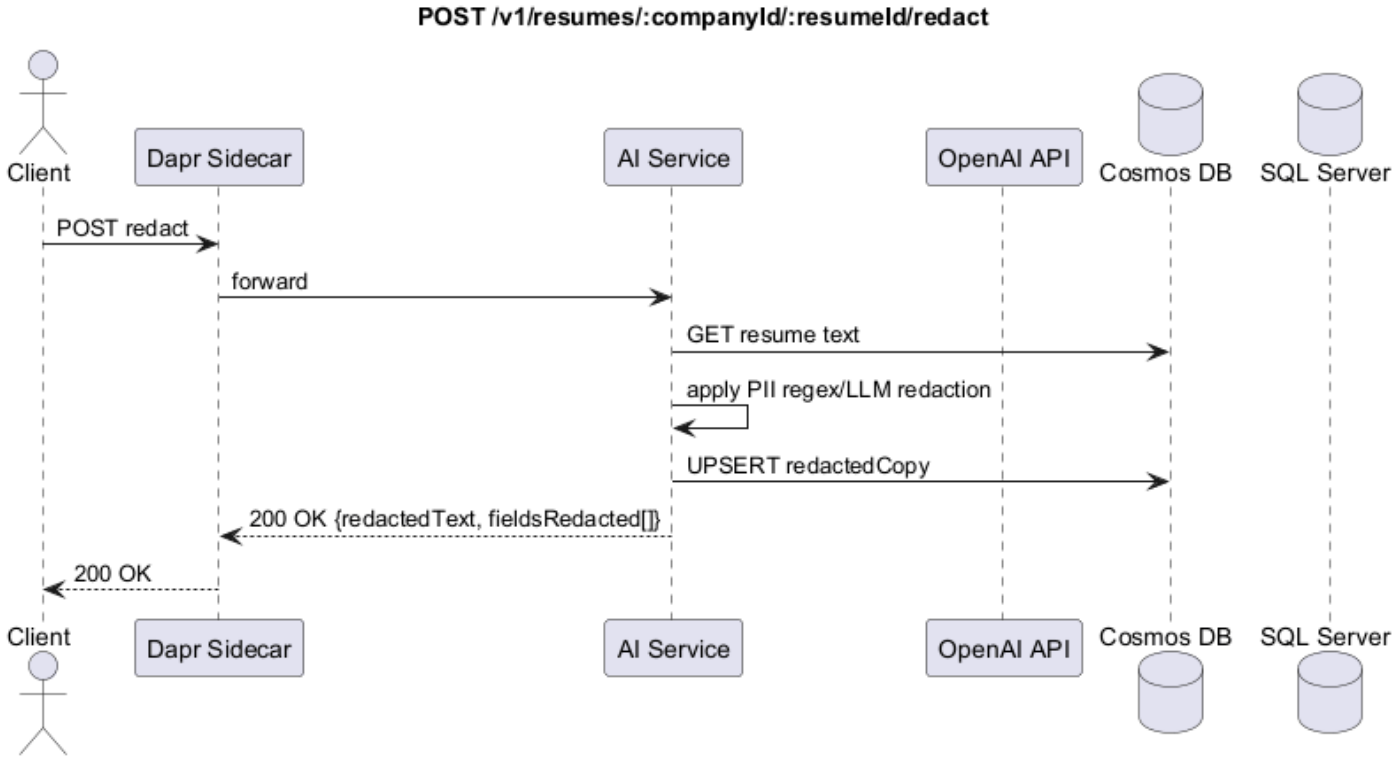



```
@startuml
title POST /v1/resumes/:companyId/parse
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST form-data file/url
Dapr -> AIS: forward
AIS -> AIS: extract text (PDF/DOC parser)
AIS -> OpenAI: structured extraction (optional)
OpenAI --> AIS: extracted fields
AIS -> Cosmos: UPSERT resume {resumeId, text, sections, entities}
AIS --> Dapr: 200 OK {resumeId, sections, entities}
Dapr --> Client: 200 OK
@enduml
```

18. POST /v1/resumes/:companyId/:resumeId/redact (v1-resumes-redact.puml)

Example use case: Produce a PII-blind version for fair screening or sharing with interviewers.

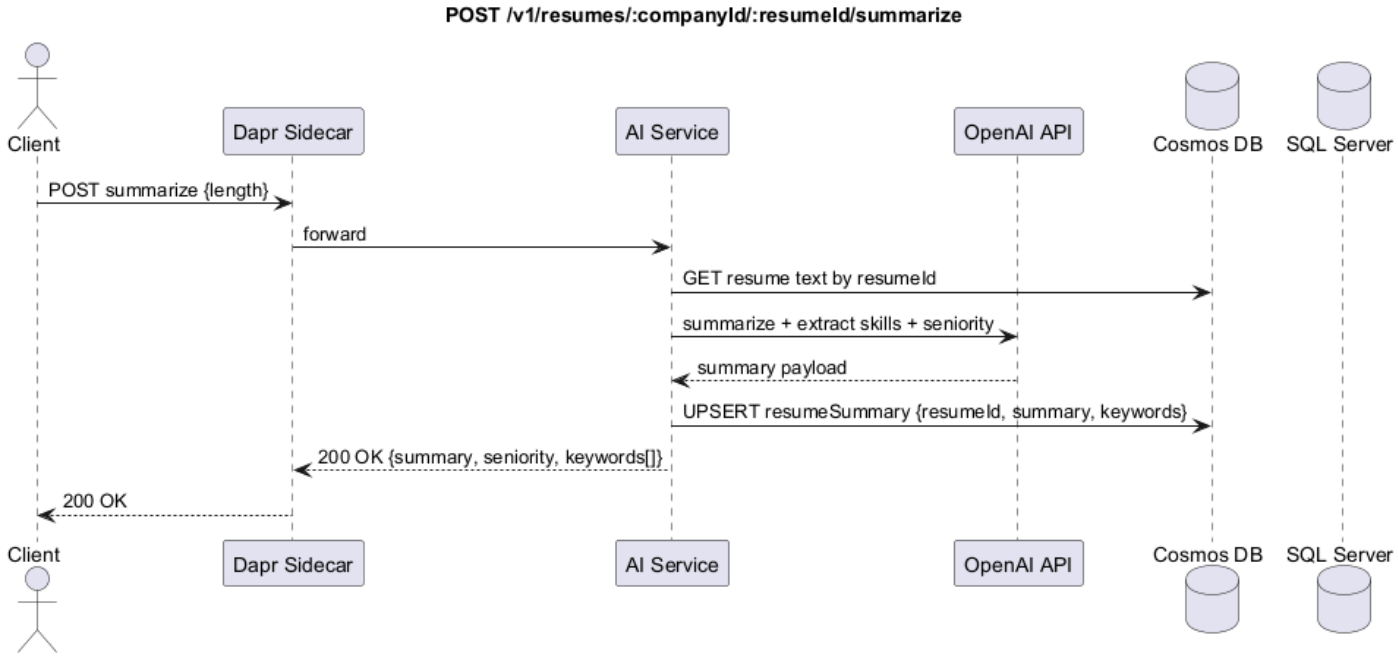



```
@startuml
title POST /v1/resumes/:companyId/:resumeId/redact
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST redact
Dapr -> AIS: forward
AIS -> Cosmos: GET resume text
AIS -> AIS: apply PII regex/LLM redaction
AIS -> Cosmos: UPSERT redactedCopy
AIS --> Dapr: 200 OK {redactedText, fieldsRedacted[]}
Dapr --> Client: 200 OK
@enduml
```

19. POST /v1/resumes/:companyId/:resumeId/summarize (v1-resumes-summarize.puml)

Example use case: Create a compact recruiter summary with seniority and key skills for list views.

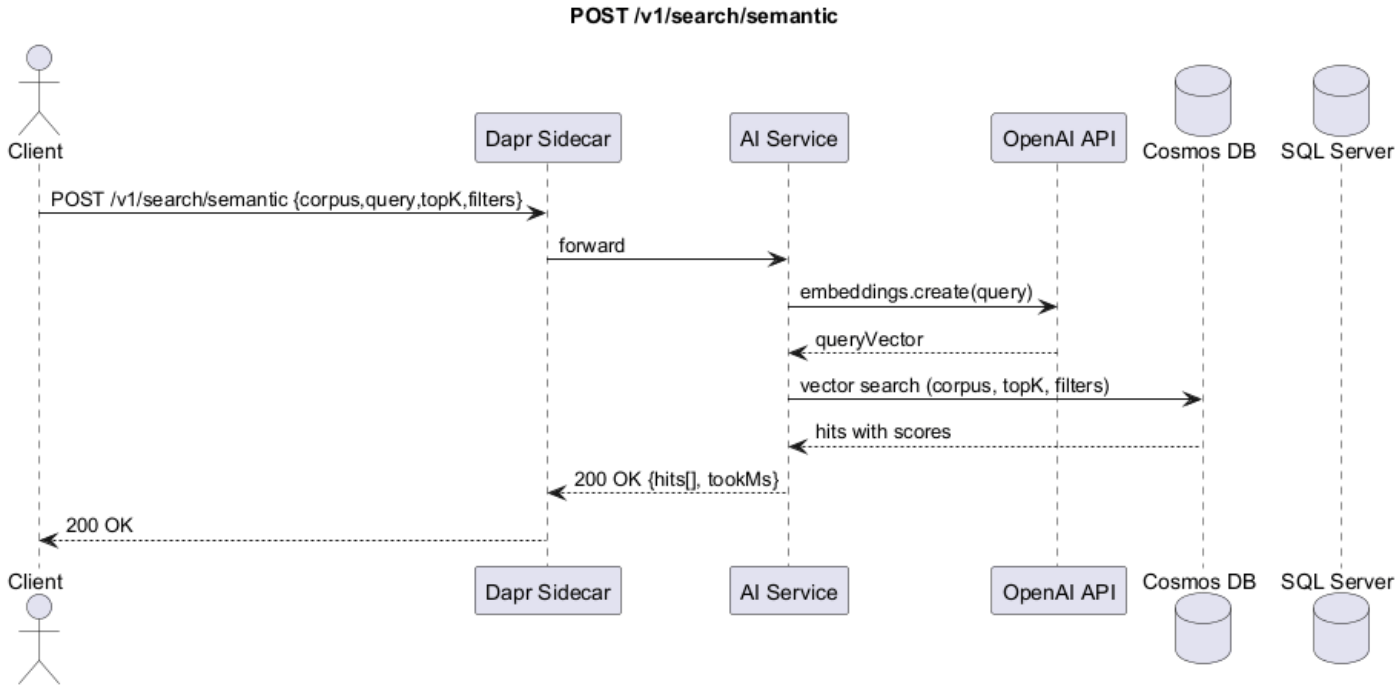



```
@startuml
title POST /v1/resumes/:companyId/:resumeId/summarize
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST summarize {length}
Dapr -> AIS: forward
AIS -> Cosmos: GET resume text by resumeId
AIS -> OpenAI: summarize + extract skills + seniority
OpenAI --> AIS: summary payload
AIS -> Cosmos: UPSERT resumeSummary {resumeId, summary, keywords}
AIS --> Dapr: 200 OK {summary, seniority, keywords[]}
Dapr --> Client: 200 OK
@enduml
```

20. POST /v1/search/semantic (v1-search-semantic.puml)

Example use case: Recruiter searches resumes/jobs with natural language plus filters (location, type).

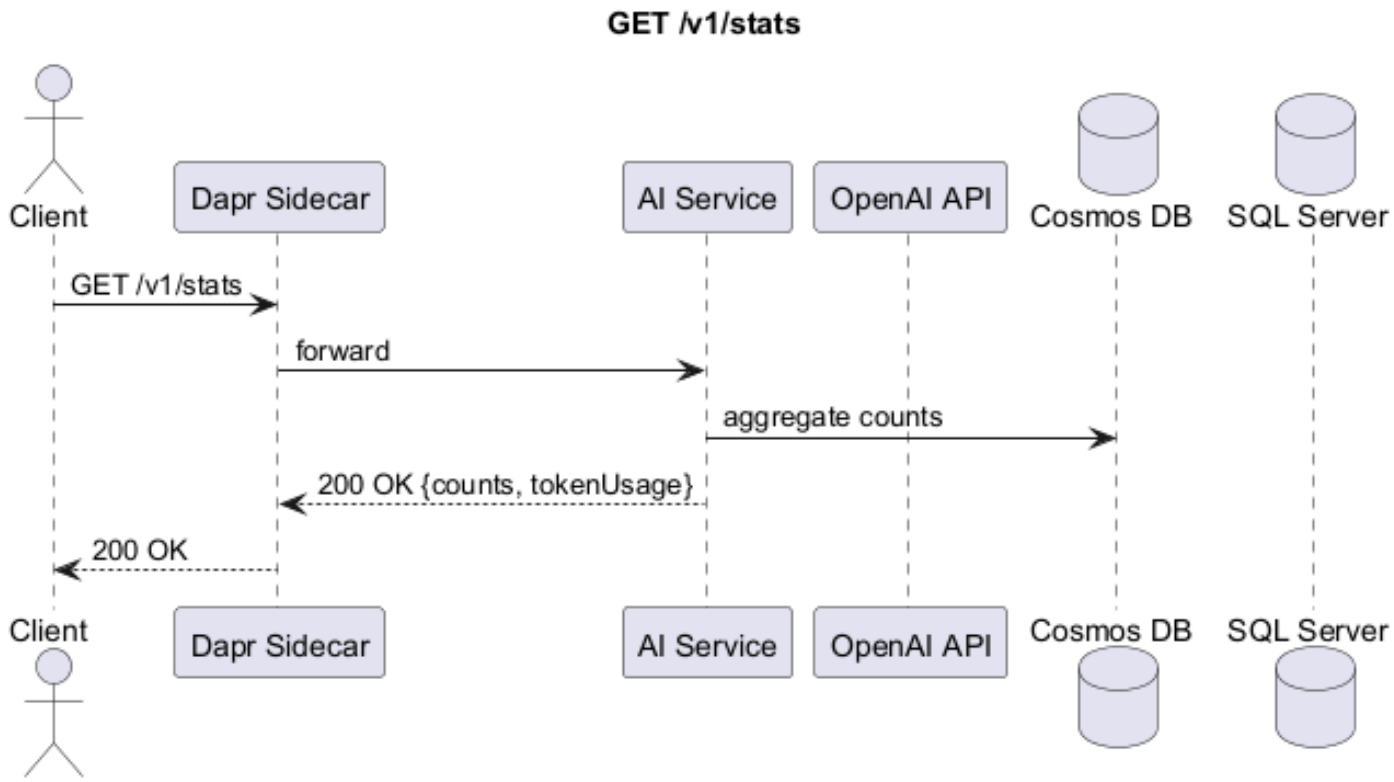



```
@startuml
title POST /v1/search/semantic
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST /v1/search/semantic {corpus,query,topK,filters}
Dapr -> AIS: forward
AIS -> OpenAI: embeddings.create(query)
OpenAI --> AIS: queryVector
AIS -> Cosmos: vector search (corpus, topK, filters)
Cosmos --> AIS: hits with scores
AIS --> Dapr: 200 OK {hits[], tookMs}
Dapr --> Client: 200 OK
@enduml
```

21. GET /v1/stats (v1-stats.puml)

Example use case: Ops dashboard—counts of embeddings, parse errors, token usage, recent errors.

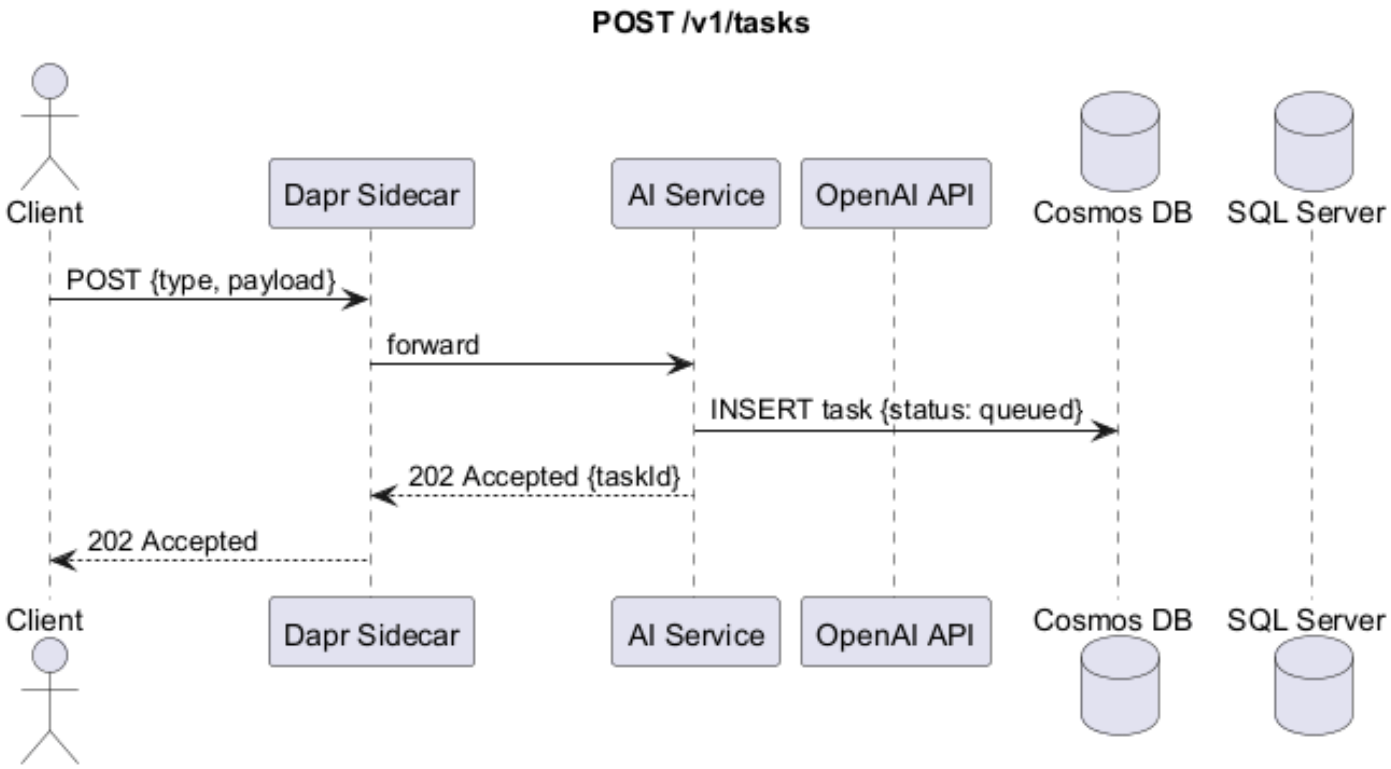



```
@startuml
title GET /v1/stats
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: GET /v1/stats
Dapr -> AIS: forward
AIS -> Cosmos: aggregate counts
AIS --> Dapr: 200 OK {counts, tokenUsage}
Dapr --> Client: 200 OK
@enduml
```

22. POST /v1/tasks (v1-tasks-create.puml)

Example use case: Queue long-running LLM/bulk jobs (e.g., re-embedding entire corpus overnight).

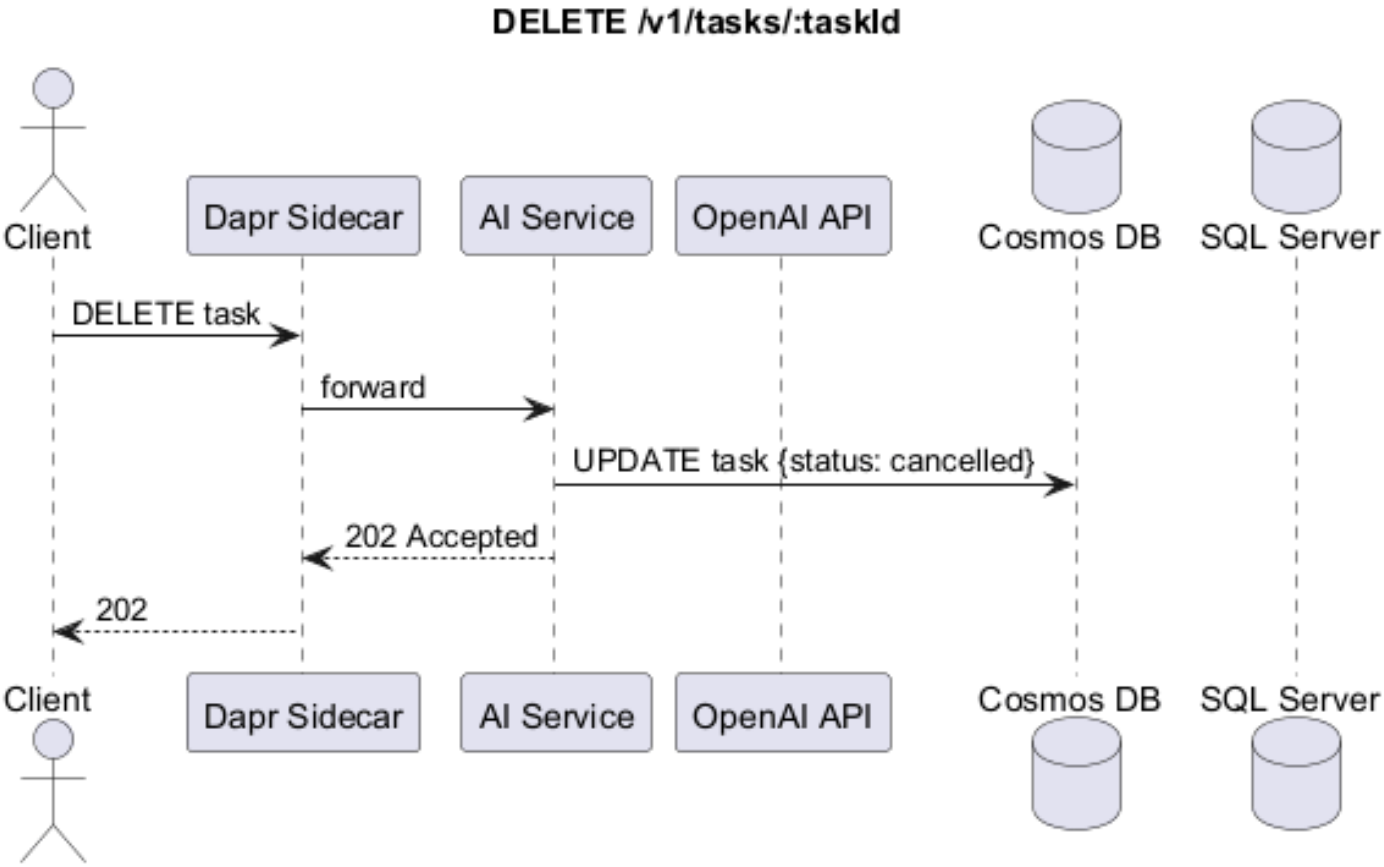



```
@startuml
title POST /v1/tasks
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST {type, payload}
Dapr -> AIS: forward
AIS -> Cosmos: INSERT task {status: queued}
AIS --> Dapr: 202 Accepted {taskId}
Dapr --> Client: 202 Accepted
@enduml
```

23. DELETE /v1/tasks/:taskId (v1-tasks-delete.puml)

Example use case: Cancel a long task if a user navigates away or the job was misconfigured.

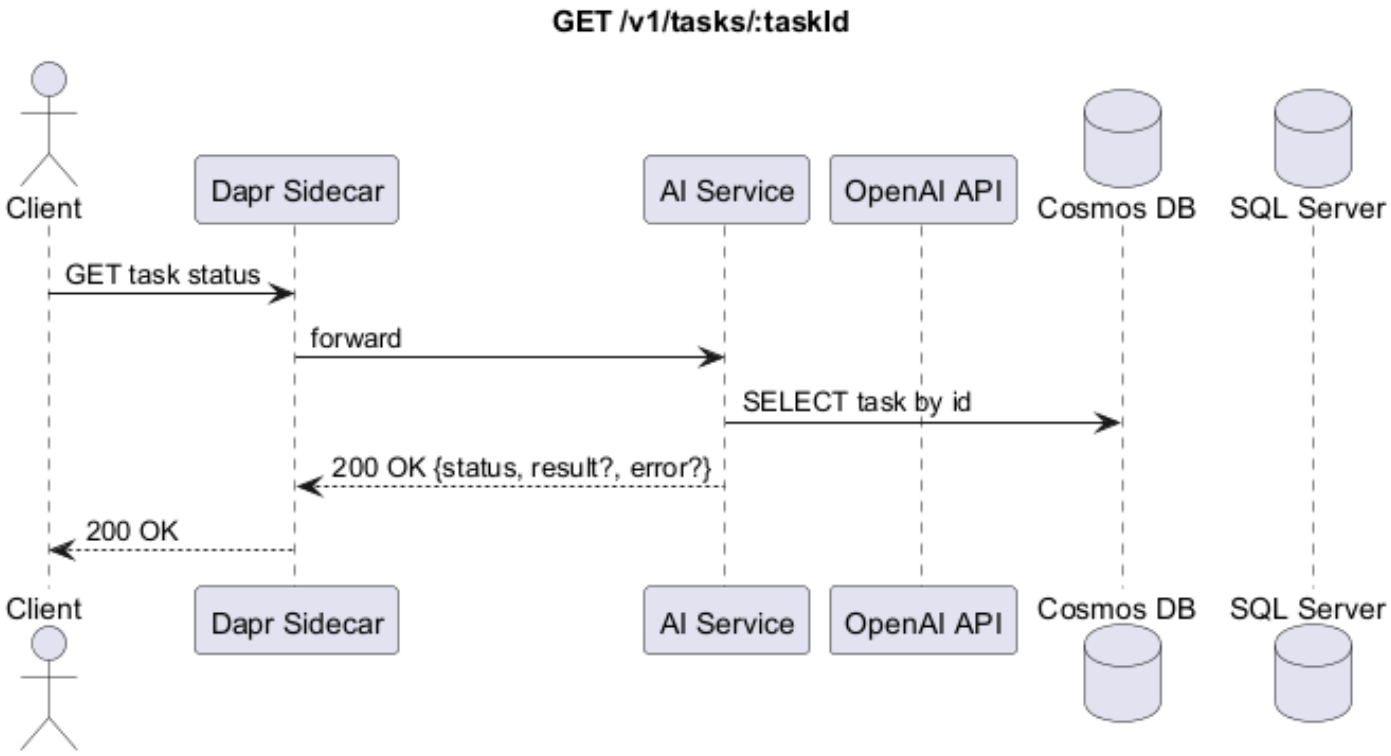



```
@startuml
title DELETE /v1/tasks/:taskId
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: DELETE task
Dapr -> AIS: forward
AIS -> Cosmos: UPDATE task {status: cancelled}
AIS --> Dapr: 202 Accepted
Dapr --> Client: 202
@enduml
```

24. GET /v1/tasks/:taskId (v1-tasks-get.puml)

Example use case: Poll task status from the UI progress bar or CI after bulk precompute jobs.

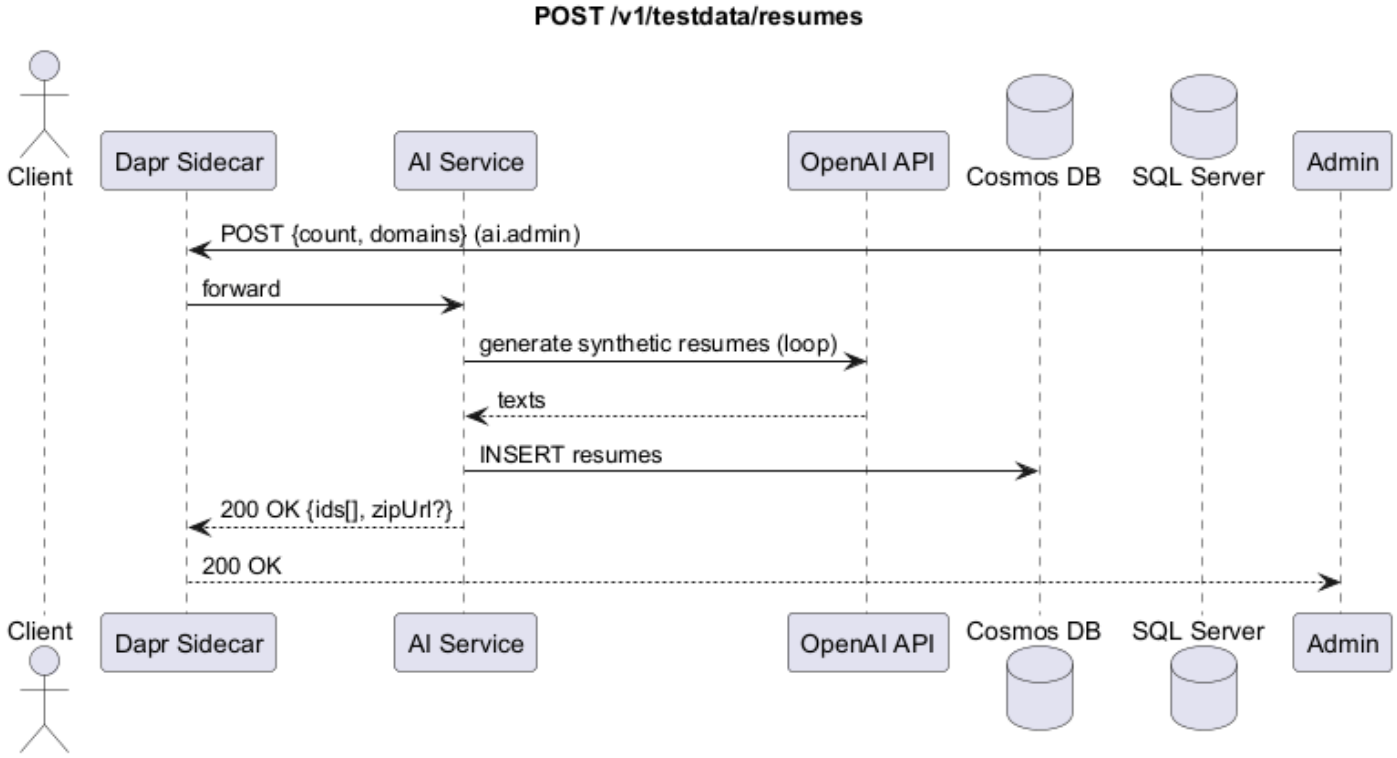



```
@startuml
title GET /v1/tasks/:taskId
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: GET task status
Dapr -> AIS: forward
AIS -> Cosmos: SELECT task by id
AIS --> Dapr: 200 OK {status, result?, error?}
Dapr --> Client: 200 OK
@enduml
```

25. POST /v1/testdata/resumes (v1-testdata-resumes.puml)

Example use case: Generate synthetic resumes for demos, QA fixtures, and load testing (admin-only).

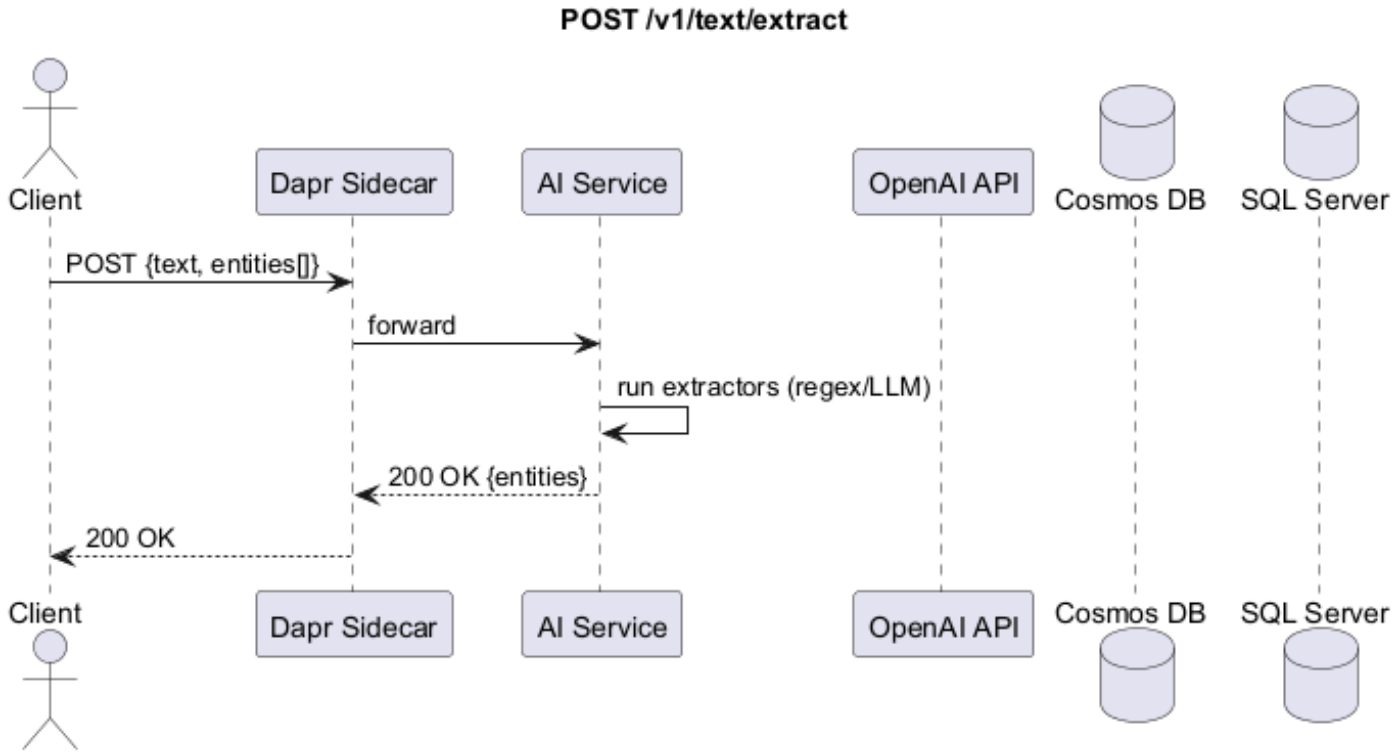



```
@startuml
title POST /v1/testdata/resumes
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Admin -> Dapr: POST {count, domains} (ai.admin)
Dapr -> AIS: forward
AIS -> OpenAI: generate synthetic resumes (loop)
OpenAI --> AIS: texts
AIS -> Cosmos: INSERT resumes
AIS --> Dapr: 200 OK {ids[], zipUrl?}
Dapr --> Admin: 200 OK
@enduml
```

26. POST /v1/text/extract (v1-text-extract.puml)

Example use case: Extract phones/emails/URLs/skills from free text like cover letters or notes.

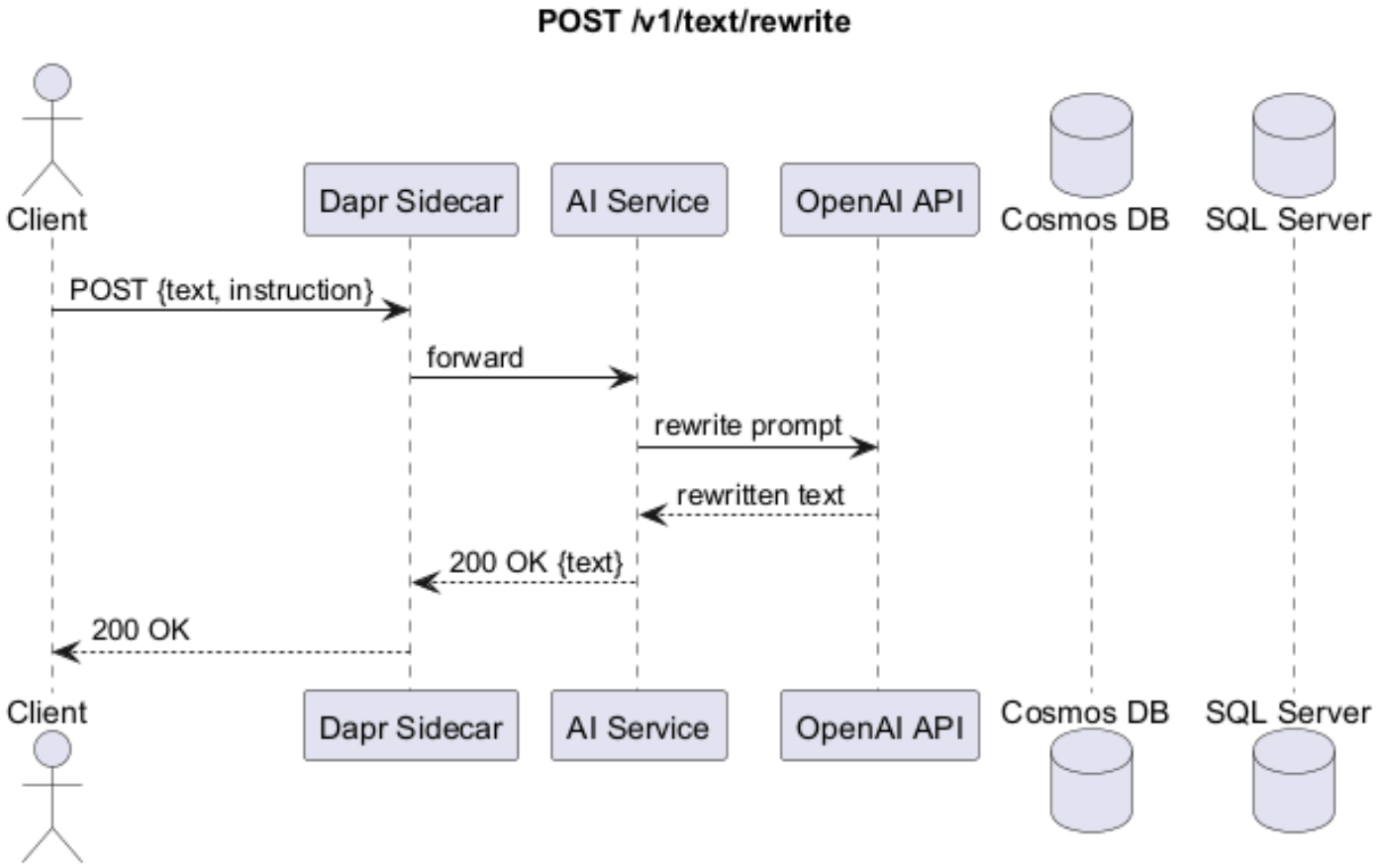



```
@startuml
title POST /v1/text/extract
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST {text, entities[]}
Dapr -> AIS: forward
AIS -> AIS: run extractors (regex/LLM)
AIS --> Dapr: 200 OK {entities}
Dapr --> Client: 200 OK
@enduml
```

27. POST /v1/text/rewrite (v1-text-rewrite.puml)

Example use case: Generic 'make shorter/clearer' used by any rich-text field in the UI.

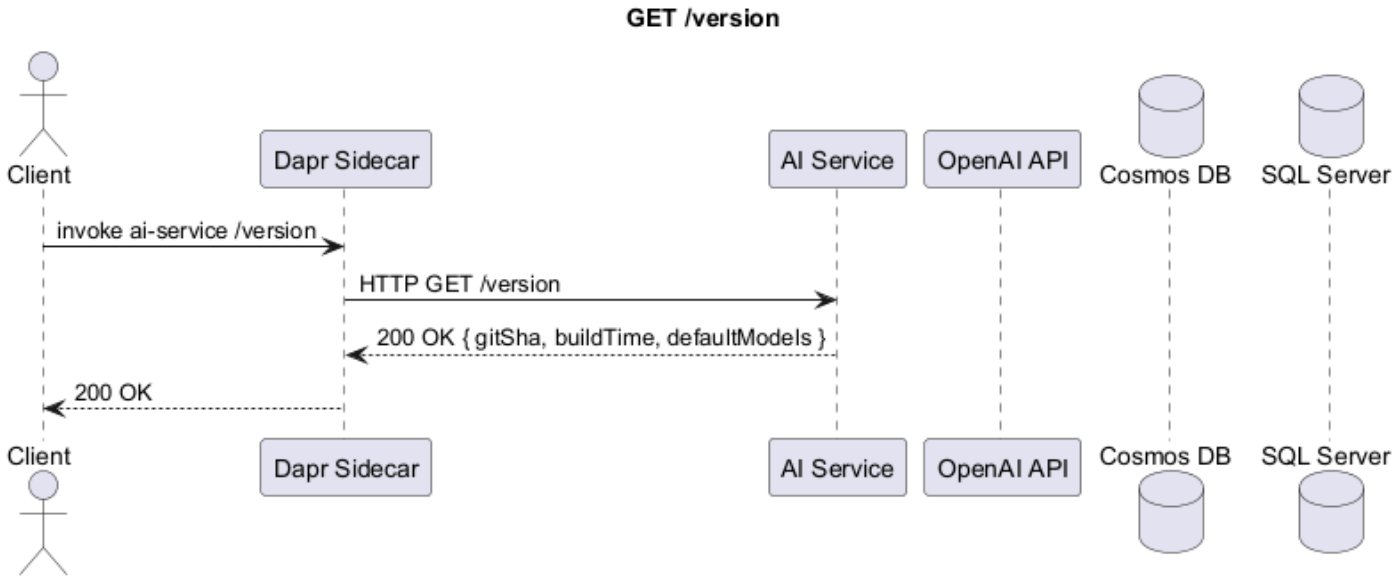



```
@startuml
title POST /v1/text/rewrite
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: POST {text, instruction}
Dapr -> AIS: forward
AIS -> OpenAI: rewrite prompt
OpenAI --> AIS: rewritten text
AIS --> Dapr: 200 OK {text}
Dapr --> Client: 200 OK
@enduml
```

28. GET /version (version.puml)

Example use case: Verify which build (git SHA) and default models are running during canary/rollback checks.




```
@startuml
title GET /version
actor Client
participant "Dapr Sidecar" as Dapr
participant "AI Service" as AIS
participant "OpenAI API" as OpenAI
database "Cosmos DB" as Cosmos
database "SQL Server" as SQL

Client -> Dapr: invoke ai-service /version
Dapr -> AIS: HTTP GET /version
AIS --> Dapr: 200 OK { gitSha, buildTime, defaultModels }
Dapr --> Client: 200 OK
@enduml
```