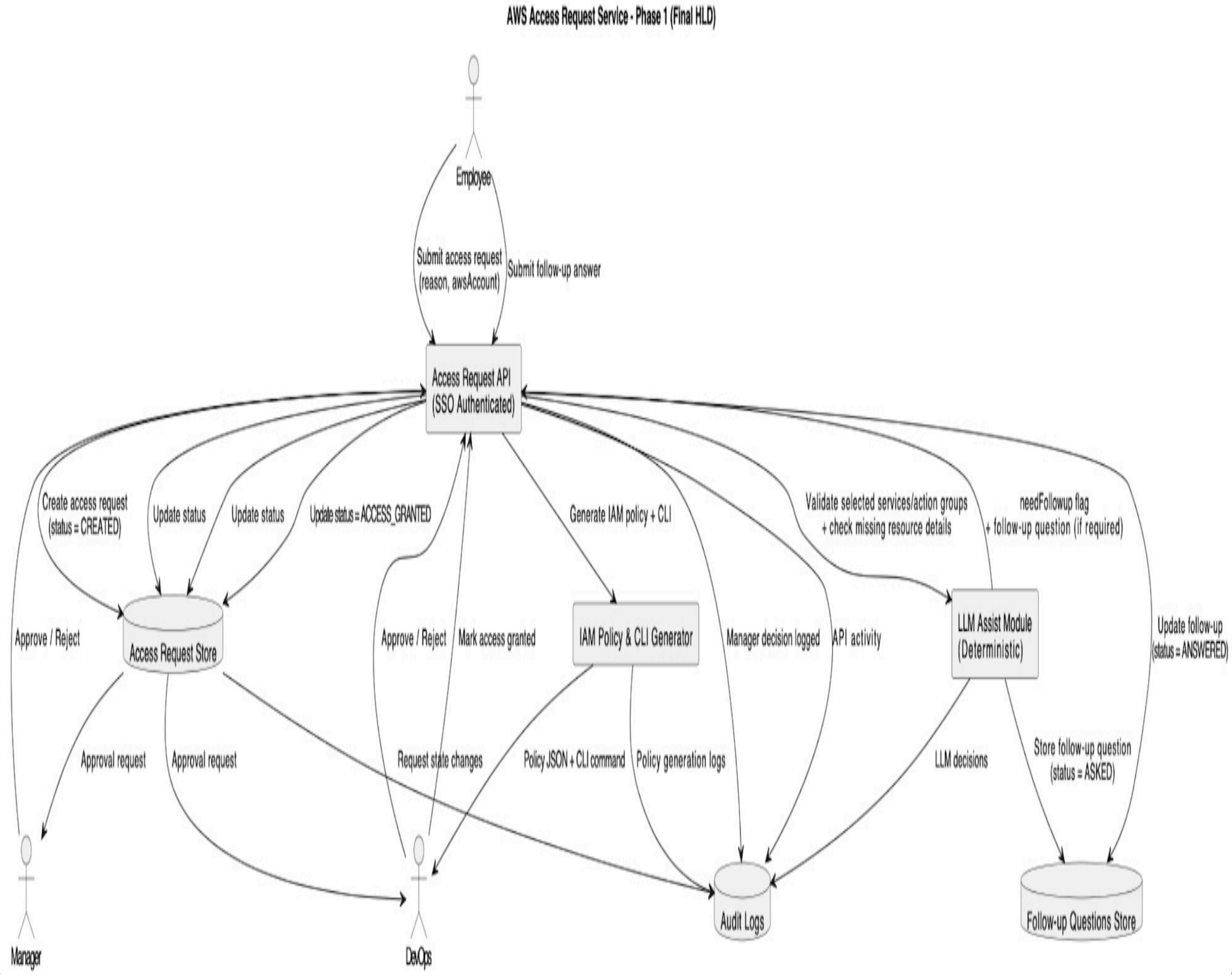
AWS Access Request Service - Phase 1 (Final HLD)
Employee
Submit access request (reason, awsAccount)
Submit follow-up answer
Access Request API (SSO Authenticated)
Create access request (status = CREATED)
Update status
Update status
Update status = ACCESS_GRANTED
Generate IAM policy + CLI
Validate selected services/action groups + check missing resource details
needFollowup flag + follow-up question (if required)
Approve / Reject
Access Request Store
Approve / Reject
Mark access granted
IAM Policy & CLI Generator
Manager decision logged
API activity
LLM Assist Module (Deterministic)
Update follow-up (status = ANSWERED)
Approval request
Approval request
Request state changes
Policy JSON + CLI command
Policy generation logs
LLM decisions
Store follow-up question (status = ASKED)
Manager
DevOps
Audit Logs
Follow-up Questions Store

The AWS Access Request Service (Phase-1) enables employees to request temporary AWS access using natural language. The system uses an LLM to reduce incomplete requests by suggesting services/actions and asking a single follow-up question when mandatory details (such as resource ARNs) are missing. All access requests follow a strict approval workflow and generate IAM policies and AWS CLI commands without automatic execution.

2.2 Core Components

2.2.1 Access Request API

- Entry point for all access requests
- Authenticated via existing SSO
- Accepts request reason and AWS account
- Orchestrates LLM calls and approval flow

2.2.2 LLM Assist Module

- Interprets request reason
- Suggests AWS services and action groups
- Determines if follow-up information is required
- Asks only one follow-up question at a time
- Produces deterministic JSON output

2.2.3 Access Request Store

- Persists access request data
- Maintains request lifecycle status
- Serves as the system's source of truth

2.2.4 Approval Workflow

- Manager approval for business validation
- DevOps approval for technical validation
- Manual execution of generated CLI commands

2.2.5 IAM Policy & CLI Generator

- Generates IAM policy JSON
- Generates AWS CLI command text
- Does not execute commands (Phase-1 constraint)

2.2.6 Audit Logging

- Captures all system events
- Immutable record for governance and compliance

---

3. Low Level Design (LLD)

3.1 Request Creation Flow

1. Employee submits an access request with a reason.
2. Access Request API invokes LLM Assist Module.
3. LLM suggests AWS services and action groups.
4. If required information is missing:
    - LLM returns needFollowup = true
    - System asks a single follow-up question.
5. User responds to follow-up.
6. LLM re-evaluates and finalizes suggestions.
7. Access request is stored with status CREATED.
8. Audit log entry is recorded.

3.2 Approval Flow

Manager Approval

- Reviews request intent
- Approves or rejects with reason
- Status updated accordingly

DevOps Approval

- Reviews access scope and generated CLI
- Approves or rejects
- On approval, policy and CLI command are generated

3.3 IAM Policy Generation Flow

- Inputs:
    - AWS services
    - IAM actions
    - Resource ARNs
- Outputs:
    - IAM policy JSON
    - AWS CLI command (text only)

4.

Database Design (Phase-1)

4.1 Entity Overview

Table Name	Purpose
users	Employee, Manager, DevOps identities
services	Allowed AWS services
action_groups	Logical permission groups
aws_actions	IAM actions per group
access_requests	Core request data
approvals	Manager & DevOps decisions
Audit_logs	Governance and audit trail
Suggestion follow up	To give the follow up question

---

5. DBMS Tables & SQL DDL

5.1 user: -NOT REQ(INSTEAD OF USER ID USE EMAIL)

```
CREATE TABLE access_requests (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,

  requester_email VARCHAR(255) NOT NULL,
  aws_account ENUM('ZINKA','DIVUM') NOT NULL,

  reason TEXT NOT NULL,
  services JSON NOT NULL,
  resource_arns JSON,

  status ENUM(
    'CREATED',
    'MANAGER_APPROVED',
    'DEVOPS_APPROVED',
    'ACCESS_GRANTED',
    'REJECTED'
  ) NOT NULL,

  duration_hours INT DEFAULT 24,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

5.2 services

```
CREATE TABLE services (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  service_name VARCHAR(100) NOT NULL
);
```

5.3 action_groups

```
CREATE TABLE action_groups (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  service_id BIGINT NOT NULL,
 Description VARCHAR(100) NOT NULL,
  group_name VARCHAR(100) NOT NULL,
  FOREIGN KEY (service_id) REFERENCES services(id)
);
```

5.4 aws_actions

Changes made :
Action_groups

```
CREATE TABLE action_groups (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  service_id BIGINT NOT NULL,
  group_name VARCHAR(100) NOT NULL
);
```

aws_actions:

```
CREATE TABLE aws_actions (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  aws_action VARCHAR(100) NOT NULL UNIQUE
);
```

junction table

```
CREATE TABLE action_group_actions (
  action_group_id BIGINT NOT NULL,
  aws_action_id BIGINT NOT NULL,
  PRIMARY KEY (action_group_id, aws_action_id),
  FOREIGN KEY (action_group_id) REFERENCES action_groups(id),
  FOREIGN KEY (aws_action_id) REFERENCES aws_actions(id)
);
```

5.5 access_requests

```
CREATE TABLE access_requests (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,

  requester_id BIGINT NOT NULL,
  aws_account ENUM('ZINKA','DIVUM') NOT NULL,
  reason TEXT NOT NULL,

  services JSON NOT NULL,
  resource_arns JSON,

  status ENUM(
    'CREATED',
    'MANAGER_REJECTD’,
    'DEVOPS_REJECTED',
    'ACCESS_GRANTED',

  ) NOT NULL,

  -- MANAGER APPROVAL
  manager_approver_id BIGINT,

  -- DEVOPS APPROVAL
  devops_approver_id BIGINT,

  duration_hours INT DEFAULT 24,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

  FOREIGN KEY (requester_id) REFERENCES users(id),
  FOREIGN KEY (manager_approver_id) REFERENCES users(id),
  FOREIGN KEY (devops_approver_id) REFERENCES users(id)
);
```

5.7 audit_logs

```
CREATE TABLE audit_logs (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  access_request_id BIGINT NOT NULL,
  event_type VARCHAR(100) NOT NULL,
  event_data JSON,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (access_request_id) REFERENCES access_requests(id)
);
```

5.8 follow up

```
CREATE TABLE followup_questions (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  access_request_id BIGINT NOT NULL,

  question TEXT NOT NULL,
  answer TEXT,

  status ENUM(
    'ASKED',
    'ANSWERED',
    'INVALID'
  ) NOT NULL DEFAULT 'ASKED',

  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  answered_at TIMESTAMP,

  FOREIGN KEY (access_request_id)
    REFERENCES access_requests(id)
);
```

---

6. LLM Design

6.1 LLM Input JSON

```
{
  "reason": "Need access to upload, download, and delete logs in S3",
  "allowedServices": ["S3"],
  "allowedActionGroups": {
    "S3": [
      "READ_OBJECTS",
      "UPLOAD_OBJECTS",
      "DELETE_OBJECTS",
      "LIST_BUCKET",
      "MANAGE_BUCKET"
    ]
  }
```

}

6.2 Standard Action Groups (S3)

```
{
 "S3": {
  "READ_OBJECTS": [
   "s3:GetObject",
   "s3:GetObjectVersion"
  ],
  "UPLOAD_OBJECTS": [
   "s3:PutObject",
   "s3:CreateMultipartUpload",
   "s3:UploadPart",
   "s3:CompleteMultipartUpload",
   "s3:AbortMultipartUpload"
  ],
  "DELETE_OBJECTS": [
   "s3:DeleteObject",
   "s3:DeleteObjectVersion"
  ],
  "LIST_BUCKET": [
   "s3:ListBucket"
  ],
  "MANAGE_BUCKET": [
   "s3:CreateBucket",
   "s3:DeleteBucket",
   "s3:GetBucketPolicy",
   "s3:PutBucketPolicy",
   "s3:GetBucketVersioning",
   "s3:PutBucketVersioning"
  ]
 }}
```

6.3 LLM Output JSON (Follow-up Required)

```
{
  "needFollowup": true,
  "followupQuestion": "Please provide the S3 bucket ARN(s) and optional object prefix.",
  "services": ["S3"],
  "actionGroups": [
    "READ_OBJECTS",
    "UPLOAD_OBJECTS",
    "DELETE_OBJECTS",
    "LIST_BUCKET"
  ]
}
```

6.4 LLM Output JSON (Complete)

```
{
  "needFollowup": false,
  "services": ["S3"],
  "actionGroups": [
    "READ_OBJECTS",
    "UPLOAD_OBJECTS",
    "DELETE_OBJECTS",
    "LIST_BUCKET"
  ],
  "resolvedResources": {
    "S3": {
      "bucketArn": "arn:aws:s3:::logs-bucket",
      "objectArn": "arn:aws:s3:::logs-bucket/logs/*"
    }
  }
}
```

Changes made

```
call LLM
IF follow-up needed:
    store question in followup_questions
```

return followupQuestion to FE

Pseudo code

```
llmResponse = callLLM(reason)

if llmResponse.needFollowup == true:
    insert into followup_questions (
        access_request_id,
        question,
        status
    ) values (
        requestId,
        llmResponse.followupQuestion,
        'ASKED'
    )
```

7. IAM Policy & CLI Output (From Action Groups)

7.1 Generated IAM Policy JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListLogsBucket",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": "arn:aws:s3:::logs-bucket"
    },
    {
      "Sid": "ReadWriteDeleteLogs",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:PutObject",
        "s3:CreateMultipartUpload",
        "s3:UploadPart",
        "s3:CompleteMultipartUpload",
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion"
      ],
      "Resource": "arn:aws:s3:::logs-bucket/logs/*"
    }
  ]
}
```

7.2 AWS CLI Command (Generated)

```
aws iam create-policy \
  --policy-name TempS3LogsFullAccess_001 \
  --policy-document file://policy.json
```

---

8. Phase-1 Constraints

- No automatic AWS execution
- No policy attachment
- No revoke workflow
- Manual DevOps execution only

---

9. Phase-1 Completion Criteria

Phase-1 is considered complete when:

- Requests are fully structured using LLM assistance
- Follow-up questions resolve missing data
- Approvals are enforced
- IAM policy & CLI are generated
- All actions are auditable

---

10. Conclusion

This design satisfies all Phase-1 PRD requirements while maintaining strong governance, deterministic LLM behavior, and extensibility for Phase-2 enhancements.

API : Changes asked -

1. Authentication (Common for all APIs)

- All APIs are authenticated via **SSO**
- User identity is derived from request headers (example):

X-User-Email: user@company.com

X-User-Role: EMPLOYEE | MANAGER | DEVOPS

---

2. Create Access Request API

Endpoint

POST /api/v1/access-requests

Purpose

- Accept initial access request

- Invoke LLM to suggest services/actions
- take answer
- update followup_questions
- continue flow

Request Body

```
{
  "reason": "Need access to upload logs to S3",
  "awsAccount": "ZINKA",
  "services": ["S3"],
  "actionGroups": [
    "READ_OBJECTS"
  ],
  "resources": {
    "bucketArn": "arn:aws:s3:::logs-bucket",
    "objectArn": "arn:aws:s3:::logs-bucket/*"
  }
}
```

Response – Follow-up Required

```
{
  "requestId": 101,
  "needFollowup": true,
  "followupQuestion": "Please provide the S3 bucket ARN(s).",
  "services": ["S3"],
  "actionGroups": [
    "READ_OBJECTS",
    "UPLOAD_OBJECTS"
  ]
}
```

When needFollowup = true, the system persists the follow-up question in the followup_questions entity for tracking and audit purposes. The question is returned in the API response for frontend display.

Response – No Follow-up Required

```
{

  "requestId": 101,

  "needFollowup": false,

  "services": ["S3"],

  "actionGroups": [

    "READ_OBJECTS",
    "UPLOAD_OBJECTS"

  ],

  "status": "CREATED"

}
```

---

3. Submit Follow-up Answer API

Endpoint

POST /api/v1/access-requests/{requestId}/followup/{followup_id}

Purpose

- Accept user’s response to LLM follow-up question
- Re-evaluate request completeness

Request Body

```
{
  "answer": {
    "S3": {
      "bucketArn": "arn:aws:s3:::logs-bucket",
      "objectArn": "arn:aws:s3:::logs-bucket/*"
    }
  }
}
```

Response

```
{
  "requestId": 101,
  "status": "CREATED",
  "services": ["S3"],
  "actionGroups": [
    "READ_OBJECTS",
    "UPLOAD_OBJECTS"
  ]
}
```

Backend behaviour :

1. User submits follow-up answer
2. Backend updates followup_questions.answer
3. Backend calls LLM again with:
   - original reason
   - previous LLM context
   - structured follow-up answer
4. LLM returns final structured output
5. Backend updates:
   - access_requests.resource_arns
   - access_requests.services
   - access_requests.status = CREATED

---

4. Get Access Request Details API

Endpoint

GET /api/v1/access-requests/{requestId}

Purpose

- Fetch request details for UI display
- Used by Employee, Manager, and DevOps views

Response

```
{
  "requestId": 101,
  "requesterEmail": "user@company.com",
  "awsAccount": "ZINKA",
  "reason": "Need access to upload logs to S3",
  "services": ["S3"],
  "actionGroups": ["Upload Objects"],
  "resourceArns": ["arn:aws:s3:::logs-bucket/*"],
  "status": "CREATED",
  "createdAt": "2026-01-19T10:30:00Z"
}
```

---

5. Manager Approval API

Endpoint

POST /api/v1/access-requests/{requestId}/manager-approval

Purpose

- Manager approves or rejects request

Request Body

```
{
  "decision": "APPROVED",
  "reason": "Required for log processing"
}
```

Response

```
{
  "requestId": 101,
  "status": "MANAGER_APPROVED"
}
```

---

6. DevOps Approval API

Endpoint

POST /api/v1/access-requests/{requestId}/devops-approval

Purpose

- DevOps reviews and approves request
- Triggers IAM policy + CLI generation

Request Body

```
{
  "decision": "APPROVED",
  "reason": "Scope looks correct"
}
```

Response

```
{
  "requestId": 101,
  "status": "DEVOPS_APPROVED",
  "iamPolicy": {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": ["s3:PutObject"],

        "Resource": ["arn:aws:s3:::logs-bucket/*"]

      }

    ]

  },

  "awsCliCommand": "aws iam create-policy --policy-name TempS3Access_101
--policy-document file://policy.json"

}
```

---

7. Mark Access Granted API

Endpoint

POST /api/v1/access-requests/{requestId}/mark-granted

Purpose

- DevOps confirms manual execution of CLI command

Request Body

{}

Response

```
{
  "requestId": 101,
  "status": "ACCESS_GRANTED"
}
```

---

8. Error Response (Common)

```
{
  "errorCode": "INVALID_REQUEST",
  "message": "Missing required field: awsAccount"
}
```

Spring Boot Layered Architecture

```
com.company.awsaccess
├── AwsAccessApplication
│
├── controller
│   ├── AccessRequestController
│   ├── FollowupController
│   ├── ApprovalController
│
├── service
│   ├── AccessRequestService
│   ├── LlmAssistService
│   ├── FollowupService
│   ├── ApprovalService
│   ├── PolicyGenerationService
│   ├── AuditLogService
│
├── repository
│   ├── UserRepository
│   ├── ServiceRepository
│   ├── ActionGroupRepository
│   ├── AwsActionRepository
│   ├── AccessRequestRepository
│   ├── ApprovalRepository
│   ├── FollowupQuestionRepository
│   ├── AuditLogRepository
│
├── model (Entity)
│   ├── User
│   ├── ServiceEntity
│   ├── ActionGroup
│   ├── AwsAction
│   ├── ActionGroupAction
│   ├── AccessRequest
│   ├── Approval
│   ├── FollowupQuestion
│   ├── AuditLog
│
├── dto
│   ├── request
│   │   ├── CreateAccessRequestDto
│   │   ├── FollowupAnswerDto
│   │   ├── ApprovalDecisionDto
│   │
│   ├── response
│   │   ├── AccessRequestResponseDto
│   │   ├── FollowupRequiredResponseDto
│   │   ├── IamPolicyResponseDto
│
├── llm
│   ├── LlmClient
│   ├── LlmRequestBuilder
│   ├── LlmResponseParser
```

```
│
├── security
│   ├── SsoAuthFilter
│   ├── UserContext
│
├── exception
│   ├── BadRequestException
│   ├── UnauthorizedException
│   ├── GlobalExceptionHandler
│
└── util
    ├── JsonUtils
    ├── TimeUtils
```

2. Controller Layer (API Layer)

2.1 AccessRequestController

Handles **request creation** and **fetching request details**

```
@RestController
@RequestMapping("/api/v1/access-requests")
public class AccessRequestController {
```

APIs handled:

- POST /api/v1/access-requests
- GET /api/v1/access-requests/{requestId}
- POST /api/v1/access-requests/{requestId}/mark-granted

2.2 FollowupController

Handles **LLM follow-up lifecycle**

```
@RestController
@RequestMapping("/api/v1/access-requests")
public class FollowupController {
```

APIs handled:

- POST /{requestId}/followup/{followupId}

This separation satisfies:

> "Create a separate entity and flow for follow-up questions"

2.3 ApprovalController

```
@RestController
@RequestMapping("/api/v1/access-requests")
public class ApprovalController {
```

APIs handled:

- POST /{requestId}/manager-approval
- POST /{requestId}/devops-approval

---

3. Service Layer (Business Logic)

This is **where ALL PRD rules live**.

3.1 AccessRequestService

Responsibilities

- Create access request

- Persist initial request
- Call LLM
- Control request state transitions

```
public class AccessRequestService {
```

Key methods

```
createRequest(CreateAccessRequestDto dto)
getRequestById(Long requestId)
markAccessGranted(Long requestId)
```

3.2 LlmAssistService

Single responsibility: LLM interaction

```
public class LlmAssistService {
```

Responsibilities

- Build LLM input JSON
- Call LLM
- Validate deterministic output
- Parse response

```
LlmResponse callLlm(LlmInput input)
```

This service is used in:

- Create Access Request
- Submit Follow-up Answer

3.3 FollowupService

public class FollowupService {

Responsibilities

- Persist follow-up question
- Persist follow-up answer
- Trigger LLM re-evaluation

createFollowupQuestion(requestId, question)
answerFollowup(followupId, structuredAnswer)

Directly maps to:

> followup_questions table
> status = ASKED → ANSWERED

3.4 ApprovalService

public class ApprovalService {

Responsibilities

- Validate approver role
- Record approval decision

- Move request state

managerApproval(requestId, decision)
devopsApproval(requestId, decision)

3.5 PolicyGenerationService

public class PolicyGenerationService {

Responsibilities

- Convert action groups → IAM actions
- Generate IAM policy JSON
- Generate AWS CLI command (text only)

generatePolicy(accessRequest)
generateCliCommand(policy)

No AWS execution (Phase-1 constraint enforced here).

3.6 AuditLogService

public class AuditLogService {

Responsibilities

- Write immutable audit logs for:
    - request creation

    - LLM decisions
    - follow-up asked
    - follow-up answered
    - approvals
    - policy generation
    - access granted

---

4. Repository Layer (Persistence)

Each table has **one repository**, no shortcuts.

```
public interface AccessRequestRepository extends JpaRepository<AccessRequest, Long>
```

Same pattern for:

- UserRepository
- ApprovalRepository
- FollowupQuestionRepository
- AuditLogRepository
- ServiceRepository
- ActionGroupRepository
- AwsActionRepository

---

5. Entity Layer (Exact DB Mapping)

Entities map **1-to-1** with your DDL.

Examples:

AccessRequest Entity

```
@Entity
@Table(name = "access_requests")
public class AccessRequest {
```

Fields:

- id
- requesterId
- awsAccount
- reason
- services (JSON)
- resourceArns (JSON)
- status
- durationHours
- createdAt

FollowupQuestion Entity

```
@Entity
```

```
@Table(name = "followup_questions")
public class FollowupQuestion {
```

Fields:

- id
- accessRequestId
- question
- answer
- status
- createdAt
- answeredAt

---

6. DTO Layer (Frontend Contract)

This is **critical for FE sharing**.

CreateAccessRequestDto

```
reason
awsAccount
services[]
actionGroups[]
resources{}
```

FollowupAnswerDto

```
answer {
  S3 {
    bucketArn
    objectArn
  }
}
```

AccessRequestResponseDto

requestId
status
services
actionGroups
followupQuestion

---

7. Security Layer (SSO)

public class SsoAuthFilter extends OncePerRequestFilter

Reads headers:

X-User-Email
X-User-Role

Populates:

UserContext.set(email, role)

Used across services for:

- authorization
- approvals
- audit logs

---

8. Exception Handling

```
@ControllerAdvice
public class GlobalExceptionHandler {
```

Handles:

- invalid request
- missing follow-up
- unauthorized approval
- invalid state transitions

Maps to:

```
{
  "errorCode": "INVALID_REQUEST",
  "message": "..."
}
```
