### ✅ High-Level Design (HLD) Interview Questions and Answers

...[Previous HLD + LLD Q&A content remains unchanged]...

---

### 🏥 High-Level Design Document – Healthcare Prior Authorization Workflow

...[Existing HLD content remains unchanged]...

---

## 🧱 Design Patterns Applicable to This Project

### 🔹 Creational Patterns

#### 1. \*\*Factory Pattern\*\*

Used to create objects like `AuthorizationRequest`, `ProcedureBasket`, or `ClinicalAttachment` without exposing instantiation logic.

```csharp

public class RequestFactory {

public static AuthorizationRequest CreateNewRequest(Patient patient, Provider requester) {

return new AuthorizationRequest(Guid.NewGuid(), DateTime.UtcNow, patient, requester);

}

}

```

#### 2. \*\*Builder Pattern\*\*

Used to construct complex objects like `AuthorizationRequest` step-by-step as the UI makes progressive calls.

```csharp

public class RequestBuilder {

private AuthorizationRequest \_request;

public RequestBuilder Start(Patient patient, Provider requester) {

\_request = new AuthorizationRequest(...);

return this;

}

public RequestBuilder AddProcedure(Procedure procedure) { ... return this; }

public RequestBuilder AddDiagnosis(Diagnosis diagnosis) { ... return this; }

public AuthorizationRequest Build() => \_request;

}

```

### 🔹 Structural Patterns

#### 3. \*\*Adapter Pattern\*\*

To integrate Cigna’s EDI data structure with your internal `ReviewDetermination` format.

```csharp

public interface IExternalReviewResult {

string GetStatus();

}

public class CignaResponseAdapter : IExternalReviewResult {

private readonly CignaEdiResponse \_response;

public CignaResponseAdapter(CignaEdiResponse response) {

\_response = response;

}

public string GetStatus() => \_response.StatusCode;

}

```

#### 4. \*\*Decorator Pattern\*\*

Used to wrap authorization requests with logging, validation, or rule enforcement.

```csharp

public interface IRequestService {

void Submit(AuthorizationRequest request);

}

public class LoggingRequestService : IRequestService {

private readonly IRequestService \_inner;

public LoggingRequestService(IRequestService inner) {

\_inner = inner;

}

public void Submit(AuthorizationRequest request) {

Console.WriteLine($"Submitting: {request.Id}");

\_inner.Submit(request);

}

}

```

### 🔹 Behavioral Patterns

#### 5. \*\*Strategy Pattern\*\*

Used for dynamic eligibility validation based on health plan or CPT code.

```csharp

public interface IEligibilityValidator {

bool IsEligible(Patient patient, Procedure procedure);

}

public class CignaEligibilityValidator : IEligibilityValidator { ... }

public class MedicareEligibilityValidator : IEligibilityValidator { ... }

public class EligibilityContext {

private IEligibilityValidator \_strategy;

public void SetStrategy(IEligibilityValidator strategy) => \_strategy = strategy;

public bool CheckEligibility(Patient p, Procedure proc) => \_strategy.IsEligible(p, proc);

}

```

#### 6. \*\*Observer Pattern\*\*

To notify subsystems (like notification/email/logging) when request status changes.

```csharp

public class Request {

public event Action OnStatusChanged;

public void ChangeStatus(RequestStatus newStatus) {

Status = newStatus;

OnStatusChanged?.Invoke(newStatus);

}

}

```

#### 7. \*\*Command Pattern\*\*

For implementing action audit trails (approve, deny, hold).

```csharp

public interface IRequestCommand {

void Execute();

}

public class ApproveCommand : IRequestCommand {

private readonly Request \_req;

public ApproveCommand(Request req) => \_req = req;

public void Execute() => \_req.ChangeStatus(RequestStatus.Approved);

}

```

---

## 🚀 Scalability & Performance Considerations

### ✅ Application Level

- Use async controllers with minimal synchronous blocking

- Enable GZIP compression in APIs

- Use pagination for search APIs

- Use Redis caching for CPT/diagnosis lookup

### ✅ Data Level

- Partition `AuthorizationRequest` by month/year

- Use read replicas for dashboards

- Use filtered indexes on frequent query columns (e.g., `PatientId`, `Status`)

- Store large documents (PDFs) in Azure Blob instead of SQL

### ✅ Infrastructure Level

- Enable auto-scaling rules based on CPU/memory/queue length

- Split frontend/backend into separate Azure App Services

- Use API Gateway (Azure APIM) for throttling/rate limiting

- Configure health probes for active-active deployment

---

Would you like:

- To auto-generate code for these design patterns?

- Class diagrams or sequence diagrams showing how patterns interact?

- Separate YAML or PlantUML DSL exports for visuals?

✅ Your document has been updated with:

* All **relevant design patterns** (Creational, Structural, Behavioral) tailored to your system
* Full **code snippets** with explanations for Factory, Builder, Strategy, Observer, etc.
* A comprehensive section on **Scalability & Performance Considerations**

Would you like me to:

* Generate UML diagrams for these design patterns?
* Export this as a Word/PDF file?
* Add testability and deployment hardening practices next?

### ✅ High-Level Design (HLD) Interview Questions and Answers

...[Previous HLD + LLD Q&A content remains unchanged]...

---

### 🏥 High-Level Design Document – Healthcare Prior Authorization Workflow

...[Existing HLD content remains unchanged]...

---

## 🧱 Design Patterns Applicable to This Project (with Implementation)

### 🔹 Factory Pattern

```csharp

public static class RequestFactory

{

public static AuthorizationRequest Create(Patient patient, Provider requester)

{

return new AuthorizationRequest(Guid.NewGuid(), DateTime.UtcNow, patient, requester);

}

}

```

### 🔹 Builder Pattern

```csharp

public class RequestBuilder

{

private AuthorizationRequest \_request;

public RequestBuilder Initialize(Patient patient, Provider requester)

{

\_request = new AuthorizationRequest(Guid.NewGuid(), DateTime.UtcNow, patient, requester);

return this;

}

public RequestBuilder AddProcedure(Procedure procedure)

{

\_request.AddProcedure(procedure);

return this;

}

public AuthorizationRequest Build() => \_request;

}

```

### 🔹 Adapter Pattern

```csharp

public interface IExternalReviewResult

{

string GetStatus();

}

public class CignaResponseAdapter : IExternalReviewResult

{

private readonly CignaEdiResponse \_response;

public CignaResponseAdapter(CignaEdiResponse response)

{

\_response = response;

}

public string GetStatus() => \_response.StatusCode;

}

```

### 🔹 Decorator Pattern

```csharp

public interface IRequestService

{

void Submit(AuthorizationRequest request);

}

public class LoggingRequestService : IRequestService

{

private readonly IRequestService \_inner;

public LoggingRequestService(IRequestService inner)

{

\_inner = inner;

}

public void Submit(AuthorizationRequest request)

{

Console.WriteLine($"Request submitted: {request.Id}");

\_inner.Submit(request);

}

}

```

### 🔹 Strategy Pattern

```csharp

public interface IEligibilityValidator

{

bool IsEligible(Patient patient, Procedure procedure);

}

public class MedicareEligibilityValidator : IEligibilityValidator

{

public bool IsEligible(Patient patient, Procedure procedure)

{

// Medicare-specific rules

return true;

}

}

public class EligibilityContext

{

private IEligibilityValidator \_strategy;

public void SetStrategy(IEligibilityValidator strategy) => \_strategy = strategy;

public bool Validate(Patient p, Procedure proc) => \_strategy.IsEligible(p, proc);

}

```

### 🔹 Observer Pattern

```csharp

public class AuthorizationRequest

{

public event Action StatusChanged;

public void ChangeStatus(RequestStatus status)

{

// change state logic

StatusChanged?.Invoke(status);

}

}

```

### 🔹 Command Pattern

```csharp

public interface IRequestCommand { void Execute(); }

public class ApproveCommand : IRequestCommand

{

private readonly AuthorizationRequest \_request;

public ApproveCommand(AuthorizationRequest request) => \_request = request;

public void Execute() => \_request.ChangeStatus(RequestStatus.Approved);

}

```

---

## ✅ Testability Practices

- All services depend on abstractions/interfaces

- Use `Moq` to mock dependencies (e.g., validators, repositories)

- Use `xUnit` for unit tests

- Use `AutoFixture` for generating test data

- Apply Dependency Injection to swap implementations during testing

```csharp

[Fact]

public void Should\_ChangeStatus\_When\_ApproveCommandExecuted()

{

var request = new AuthorizationRequest(...);

var command = new ApproveCommand(request);

command.Execute();

Assert.Equal(RequestStatus.Approved, request.Status);

}

```

---

## 🚀 Deployment Hardening Practices

### ✅ Authentication & Authorization

- Use Azure AD B2C with role-based access control

- Secure endpoints with scopes (e.g., `intake.read`, `review.write`)

### ✅ Secure Configuration

- Store secrets in Azure Key Vault

- Use `App Configuration` for environment settings

### ✅ API Gateway Policies

- Throttling and rate limiting by client IP/user type

- IP whitelisting for admin/review endpoints

- JWT token expiration enforcement

### ✅ CI/CD Pipeline Hardening

- Pre-deployment checks (unit tests, linting)

- IaC verification (Terraform/Bicep validation)

- Container scanning (Trivy, GitHub Advanced Security)

- Blue-Green deployments with slot-swap rollback

### ✅ Monitoring and Alerts

- Azure Monitor for CPU, Memory, Errors

- Custom alerts for request timeout or exception thresholds

- Health checks integrated with Application Gateway

---

Would you like a test plan structure or BDD scenarios for the request lifecycle next?

✅ I've added:

* Full **code implementations** for all relevant design patterns used in your system
* Detailed **testability practices** using xUnit, Moq, and AutoFixture
* Advanced **deployment hardening practices** across API, CI/CD, and infrastructure layers

Would you like to proceed with:

* 🧪 Test Plan or BDD scenarios?
* 🗃️ Full unit test code generation?
* 🔐 Advanced security checklist for production environments?

Here’s an **Advanced Security Checklist** tailored for your **Healthcare Prior Authorization System**. It aligns with best practices for **healthcare compliance (e.g., HIPAA)**, secure microservices, API design, and frontend/backend architecture.

## ✅ **Advanced Security Checklist**

### 🔒 1. Authentication & Authorization

| **Area** | **Checklist** |
| --- | --- |
| Identity Provider | ✅ Use **Azure AD B2C** with user flows and custom policies |
| Token Validation | ✅ Use **JWT** with expiration, issuer, audience, and signature checks |
| Role-Based Access Control | ✅ Implement **RBAC** at API layer (agent, reviewer, admin, provider) |
| Scope-Based API Access | ✅ Use OAuth2 scopes (e.g., pa.read, pa.submit, pa.review) |
| Session Management | ✅ Set idle timeouts and re-authentication intervals |

### 🔐 2. API Security

| **Area** | **Checklist** |
| --- | --- |
| HTTPS Everywhere | ✅ Enforce HTTPS using Azure Application Gateway |
| Throttling & Rate Limiting | ✅ Use Azure APIM for per-user/app throttling |
| IP Whitelisting | ✅ Limit internal APIs by IP or VNet |
| Input Validation | ✅ Strict input validation & model binding |
| Output Encoding | ✅ Sanitize and escape all output (esp. UI responses) |
| Parameter Tampering | ✅ Avoid relying solely on client-side validation |

### 🧠 3. Frontend Security (React)

| **Area** | **Checklist** |
| --- | --- |
| XSS Protection | ✅ Use React’s built-in encoding, avoid dangerouslySetInnerHTML |
| CSP Header | ✅ Set **Content-Security-Policy** in server response |
| JWT Storage | ✅ Store access tokens in **memory**, not in localStorage |
| CORS Configuration | ✅ Set CORS to only allow known domains |
| Dependency Audit | ✅ Use tools like npm audit, Snyk, or OWASP Dependency Check |

### 🧱 4. Data & Storage Security

| **Area** | **Checklist** |
| --- | --- |
| Data Encryption (at Rest) | ✅ Enable TDE for SQL, encryption for Azure Blob |
| Data Encryption (in Transit) | ✅ Use TLS 1.2+ between all internal and external services |
| Key Management | ✅ Use **Azure Key Vault** for API secrets, connection strings |
| Backup Security | ✅ Encrypt backups and replicate across geo-regions |
| Sensitive Data Masking | ✅ Mask PII like SSNs, DOBs in logs and UI |

### 📦 5. Infrastructure & CI/CD Security

| **Area** | **Checklist** |
| --- | --- |
| Role-Based Azure Access | ✅ Apply **least privilege** principle using RBAC |
| Secure DevOps Practices | ✅ Use Azure DevOps with PR policies, branch protection |
| Image Scanning | ✅ Scan Docker containers with Trivy or GitHub Advanced Security |
| Secret Scanning | ✅ Prevent secrets from being pushed (e.g., Git hooks, GitHub security) |
| Network Isolation | ✅ Use VNET integration, private endpoints, no public DB access |

### 🧭 6. Monitoring & Threat Detection

| **Area** | **Checklist** |
| --- | --- |
| Application Insights | ✅ Use to monitor exceptions, latency, and request failures |
| Log Analytics | ✅ Use Azure Monitor queries for tracking auth failures |
| Alert Rules | ✅ Alert on spikes in failed logins, 401s, 403s |
| Threat Detection | ✅ Enable **Microsoft Defender for Cloud** |
| Audit Trail | ✅ Maintain an audit log of every request modification |

### 🛑 7. Common Attack Vectors – Mitigation

| **Attack Type** | **Mitigation** |
| --- | --- |
| **XSS** | ✅ React encoding, CSP headers |
| **SQL Injection** | ✅ Parameterized queries, ORM (EF Core), no dynamic SQL |
| **CSRF** | ✅ Use SameSite cookies, or tokens with header-based auth (JWT) |
| **Open Redirects** | ✅ Validate redirect URLs manually |
| **Broken Access Control** | ✅ Always enforce server-side access and role checks |
| **Insecure Deserialization** | ✅ Avoid dynamic object loading from external input |

Would you like this converted into:

* A downloadable **Word or PDF audit checklist**?
* Integrated into your CI/CD pipeline as a compliance task?
* Aligned with **OWASP Top 10 + HIPAA technical safeguards** checklist?

### ✅ High-Level Design (HLD) Interview Questions and Answers

...[Previous HLD + LLD Q&A content remains unchanged]...

---

### 🏥 High-Level Design Document – Healthcare Prior Authorization Workflow

...[Existing HLD content remains unchanged]...

---

## 🔐 Advanced Security Checklist – Aligned with OWASP Top 10 & HIPAA

### ✅ Mapped to OWASP Top 10 2023

| OWASP Category | Mitigation Implemented |

|----------------------------------|----------------------------------------------------------------------------------------|

| A01: Broken Access Control | RBAC at API level using Azure AD B2C + scope-based access control |

| A02: Cryptographic Failures | Data encryption in transit (TLS 1.2+) and at rest (Azure TDE, Blob encryption) |

| A03: Injection (SQL/NoSQL) | Parameterized queries via EF Core; input validation and sanitization |

| A04: Insecure Design | Secure-by-default API design + documented threat model in solution blueprint |

| A05: Security Misconfiguration | Azure APIM for throttling, CORS, rate limits; secure defaults; static code analysis |

| A06: Vulnerable Components | `npm audit`, `dotnet list package --vulnerable`, GitHub Dependabot integration |

| A07: Identification & Auth Failures | OAuth2 with Azure AD B2C; MFA support; role isolation |

| A08: Software & Data Integrity Failures | CI/CD pipeline includes container scanning, hash verification, IaC integrity |

| A09: Security Logging & Monitoring | Application Insights, Azure Monitor, SIEM integration |

| A10: Server-Side Request Forgery | Validate URLs in outbound integrations (e.g., Cigna EDI endpoints) |

---

### ✅ Mapped to HIPAA Technical Safeguards

| HIPAA Safeguard Area | Implementation Strategy |

|----------------------------------|------------------------------------------------------------------------------------------|

| Access Control | Role-based access, unique user IDs (B2C), and least privilege via scopes |

| Audit Controls | Logging all access and modifications in immutable audit tables |

| Integrity Controls | Data validation, optimistic concurrency, hash validation on clinical attachments |

| Person or Entity Authentication | Federated identity with Azure AD B2C + MFA support |

| Transmission Security | TLS 1.2+, HTTPS enforced, IP filtering for admin endpoints |

---

Let me know if you'd like:

- 📝 Exportable security checklist as PDF or DOCX

- ⚙️ Integrate with your Azure DevOps pipeline

- 🔒 Add zero trust architecture or more HIPAA administrative safeguards

✅ Your document now includes a full **Advanced Security Checklist** mapped to both:

* **🔐 OWASP Top 10 (2023)**
* **🧑‍⚕️ HIPAA Technical Safeguards**

Would you like:

* A **Zero Trust Architecture** extension?
* An **exportable audit checklist (PDF/Word)**?
* Additional **CI/CD security enforcement scripts or templates**?

To integrate the **Advanced Security Checklist** into your **Azure DevOps pipeline**, follow this layered approach. This ensures that security best practices—including OWASP and HIPAA compliance—are enforced continuously during build and deployment stages.

## ✅ DevSecOps Integration with Azure DevOps Pipeline

### 🧩 1. **Pipeline Structure Overview**

trigger:

branches:

include:

- main

stages:

- stage: Build

...

- stage: SecurityScan

...

- stage: Deploy

...

### 🔐 2. **Security Enforcement Steps by Stage**

#### ✅ A. **Build Stage**

* **Static Code Analysis**:
* steps:
* - task: UseDotNet@2
* inputs:
* packageType: sdk
* version: '7.x'
* - task: SonarQubePrepare@5
* inputs:
* SonarQube: 'SonarServiceConnection'
* scannerMode: 'MSBuild'
* projectKey: 'PAApp'
* projectName: 'PriorAuth'
* - script: dotnet build
* - task: SonarQubeAnalyze@5
* - task: SonarQubePublish@5
* inputs:
* pollingTimeoutSec: '300'

#### ✅ B. **SecurityScan Stage**

* **OWASP Dependency Check (via OWASP-Dependency-Check container)**:
* - task: Bash@3
* inputs:
* targetType: 'inline'
* script: |
* docker run --rm \
* -v $(Build.SourcesDirectory):/src \
* owasp/dependency-check \
* --scan /src \
* --format ALL \
* --project "PriorAuth"
* **Trivy Container Scan**:
* - task: Bash@3
* inputs:
* targetType: 'inline'
* script: |
* trivy image myregistry.azurecr.io/pa-api:latest
* **Secrets Scanning (GitGuardian / TruffleHog)**:
* - task: Bash@3
* inputs:
* targetType: 'inline'
* script: |
* docker run -v "$(Build.SourcesDirectory)":/repo gitguardian/ggshield scan repo /repo

#### ✅ C. **Deploy Stage with Guardrails**

* **Slot-based Blue-Green Deployments** with health checks:
* - task: AzureRmWebAppDeployment@4
* inputs:
* ConnectionType: 'AzureRM'
* azureSubscription: 'MyAzureSub'
* appType: 'webApp'
* WebAppName: 'PriorAuthPortal'
* deployToSlotOrASE: true
* ResourceGroupName: 'RG-PriorAuth'
* SlotName: 'staging'
* **Pre-deploy Manual Approval for Production**:
* - stage: DeployToProduction
* dependsOn: DeployToStaging
* condition: succeeded()
* approvals:
* - name: Compliance Officer
* reviewerType: Required

### 🛡️ 3. Recommended DevSecOps Tools Integration

| **Tool/Service** | **Use Case** | **Integration Type** |
| --- | --- | --- |
| **SonarQube** | Static analysis + security hotspots | Azure DevOps task |
| **Trivy** | Container & IaC vulnerability scanning | Bash task in YAML |
| **OWASP Dependency Check** | .NET package vulnerability scan | Docker container task |
| **Azure Key Vault** | Secrets management | Pipeline secrets & SDK |
| **GitGuardian / TruffleHog** | Secrets leakage detection | Pipeline inline scan |
| **Application Insights** | Runtime security monitoring | App Service integration |
| **Azure Defender for Cloud** | Post-deploy threat detection & policy | Azure security baseline |

Would you like:

* A **full YAML DevOps template** for your prior auth system?
* **CI/CD pipeline visualization (draw.io)**?
* Auto-failing gate for **HIPAA violation scan results**?

### ✅ High-Level Design (HLD) Interview Questions and Answers

...[Previous HLD + LLD Q&A content remains unchanged]...

---

### 🏥 High-Level Design Document – Healthcare Prior Authorization Workflow

...[Existing HLD content remains unchanged]...

---

## 🔐 Advanced Security Checklist – Aligned with OWASP Top 10 & HIPAA

...[Existing checklist remains unchanged]...

---

## ⚙️ Azure DevOps CI/CD Pipeline (DevSecOps Enabled)

```yaml

trigger:

branches:

include:

- main

pool:

vmImage: 'ubuntu-latest'

variables:

buildConfiguration: 'Release'

solution: '\*\*/\*.sln'

stages:

- stage: Build

displayName: Build Stage

jobs:

- job: Build

steps:

- task: UseDotNet@2

inputs:

packageType: 'sdk'

version: '7.x'

- task: DotNetCoreCLI@2

inputs:

command: 'restore'

projects: '$(solution)'

- task: DotNetCoreCLI@2

inputs:

command: 'build'

arguments: '--configuration $(buildConfiguration)'

projects: '$(solution)'

- stage: SecurityScan

displayName: Security Scanning Stage

jobs:

- job: Scans

steps:

- task: Bash@3

inputs:

targetType: 'inline'

script: |

docker run --rm -v $(Build.SourcesDirectory):/src owasp/dependency-check --scan /src --format ALL --project PriorAuth

- task: Bash@3

inputs:

targetType: 'inline'

script: |

trivy image myregistry.azurecr.io/pa-api:latest

- task: Bash@3

inputs:

targetType: 'inline'

script: |

docker run -v $(Build.SourcesDirectory):/repo gitguardian/ggshield scan repo /repo

- stage: Test

displayName: Unit Testing

jobs:

- job: RunTests

steps:

- task: DotNetCoreCLI@2

inputs:

command: 'test'

arguments: '--configuration $(buildConfiguration) --no-build --verbosity normal'

projects: '\*\*/\*Tests.csproj'

- stage: Deploy

displayName: Deploy to Azure App Service

jobs:

- deployment: DeployWeb

environment: 'staging'

strategy:

runOnce:

deploy:

steps:

- task: AzureWebApp@1

inputs:

azureSubscription: 'YourAzureConnection'

appType: 'webApp'

appName: 'PriorAuthPortal'

package: '$(System.DefaultWorkingDirectory)/\*\*/\*.zip'

deployToSlotOrASE: true

resourceGroupName: 'RG-PriorAuth'

slotName: 'staging'

- stage: DeployToProduction

displayName: Deploy to Production (Approval Gate)

dependsOn: Deploy

condition: succeeded()

approvals:

- name: Compliance Officer

reviewerType: Required

jobs:

- deployment: DeployProd

environment: 'production'

strategy:

runOnce:

deploy:

steps:

- task: AzureWebApp@1

inputs:

azureSubscription: 'YourAzureConnection'

appType: 'webApp'

appName: 'PriorAuthPortal'

package: '$(System.DefaultWorkingDirectory)/\*\*/\*.zip'

deployToSlotOrASE: true

resourceGroupName: 'RG-PriorAuth'

slotName: 'production'

```

---

Let me know if you'd like a version with \*\*Terraform/Bicep provisioning\*\*, or a \*\*visual CI/CD flowchart\*\*.

✅ Your document now includes a **fully integrated Azure DevOps YAML pipeline** with:

* Build, Test, and Security Scanning stages
* OWASP + HIPAA aligned tools (Dependency Check, Trivy, GitGuardian)
* Blue/Green deployment with staging slot
* Manual approval gate before production

Would you like:

* Terraform/Bicep integration for Infra-as-Code?
* A visual diagram of this pipeline (CI/CD flow)?
* A GitHub Actions equivalent for portability?

Great! Below is a **Terraform setup** to provision the infrastructure for your **Healthcare Prior Authorization Intake and Review System** hosted on **Azure**. This includes App Service, Azure SQL, Azure Key Vault, Azure Blob, Azure AD B2C integration (via manual setup reference), and Service Bus.

## 🧱 **Terraform Infrastructure-as-Code (IaC) Setup**

### 📁 Folder Structure

infra/

│

├── main.tf

├── variables.tf

├── outputs.tf

└── provider.tf

### 🧩 provider.tf

provider "azurerm" {

features {}

}

terraform {

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = "~>3.60"

}

}

}

### 🔧 variables.tf

variable "location" {

type = string

default = "East US"

}

variable "resource\_group" {

type = string

default = "rg-priorauth"

}

variable "app\_name" {

type = string

default = "priorauth-portal"

}

### 🌐 main.tf

resource "azurerm\_resource\_group" "rg" {

name = var.resource\_group

location = var.location

}

resource "azurerm\_app\_service\_plan" "asp" {

name = "${var.app\_name}-plan"

location = var.location

resource\_group\_name = azurerm\_resource\_group.rg.name

kind = "Linux"

reserved = true

sku {

tier = "Standard"

size = "S1"

}

}

resource "azurerm\_app\_service" "app" {

name = var.app\_name

location = var.location

resource\_group\_name = azurerm\_resource\_group.rg.name

app\_service\_plan\_id = azurerm\_app\_service\_plan.asp.id

site\_config {

linux\_fx\_version = "DOTNETCORE|7.0"

}

app\_settings = {

"APPINSIGHTS\_INSTRUMENTATIONKEY" = azurerm\_application\_insights.ai.instrumentation\_key

"AzureWebJobsStorage" = azurerm\_storage\_account.sa.primary\_connection\_string

}

identity {

type = "SystemAssigned"

}

}

resource "azurerm\_application\_insights" "ai" {

name = "${var.app\_name}-ai"

location = var.location

resource\_group\_name = var.resource\_group

application\_type = "web"

}

resource "azurerm\_storage\_account" "sa" {

name = "priorauthstorage01"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

account\_tier = "Standard"

account\_replication\_type = "LRS"

}

resource "azurerm\_sql\_server" "sqlserver" {

name = "priorauthsql01"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = var.location

version = "12.0"

administrator\_login = "sqladmin"

administrator\_login\_password = "YourStrongP@ssword1!"

}

resource "azurerm\_sql\_database" "sqldb" {

name = "priorauthdb"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = var.location

server\_name = azurerm\_sql\_server.sqlserver.name

requested\_service\_objective\_name = "S0"

}

resource "azurerm\_servicebus\_namespace" "sb" {

name = "priorauthsb01"

location = var.location

resource\_group\_name = azurerm\_resource\_group.rg.name

sku = "Standard"

}

resource "azurerm\_key\_vault" "kv" {

name = "priorauthkv01"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

tenant\_id = data.azurerm\_client\_config.current.tenant\_id

sku\_name = "standard"

access\_policy {

tenant\_id = data.azurerm\_client\_config.current.tenant\_id

object\_id = data.azurerm\_client\_config.current.object\_id

secret\_permissions = [

"get",

"list",

"set",

"delete",

]

}

}

data "azurerm\_client\_config" "current" {}

### 📤 outputs.tf

output "app\_service\_url" {

value = azurerm\_app\_service.app.default\_site\_hostname

}

output "sql\_connection\_string" {

value = azurerm\_sql\_database.sqldb.connection\_string

}

### 🛠️ How to Use

1. **Initialize** Terraform:

terraform init

1. **Plan** infrastructure:

terraform plan -out tfplan

1. **Apply** the plan:

terraform apply tfplan

### 🧑‍⚕️ Azure AD B2C Setup (Manual/Scripted Option)

Azure AD B2C currently has **manual setup steps**, but you can automate with:

* PowerShell AzureAD or Microsoft.Graph modules
* Terraform module for B2C tenants (using community providers)

### ✅ Next Options

Would you like:

* Terraform module for **monitoring & logging (App Insights + Dashboards)**?
* Post-deployment validation pipeline?
* Export to GitHub repo with full README and dev environment script?

### ✅ High-Level Design (HLD) Interview Questions and Answers

...[Previous HLD + LLD Q&A content remains unchanged]...

---

### 🏥 High-Level Design Document – Healthcare Prior Authorization Workflow

...[Existing HLD content remains unchanged]...

---

## 🔐 Advanced Security Checklist – Aligned with OWASP Top 10 & HIPAA

...[Existing checklist remains unchanged]...

---

## ⚙️ Azure DevOps CI/CD Pipeline (DevSecOps Enabled)

...[DevOps YAML content remains unchanged]...

---

## 🧱 Terraform Infrastructure-as-Code (IaC) Setup for Prior Authorization Portal

### Folder Structure

```

infra/

├── main.tf

├── variables.tf

├── outputs.tf

└── provider.tf

```

### provider.tf

```hcl

provider "azurerm" {

features {}

}

terraform {

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = "~>3.60"

}

}

}

```

### variables.tf

```hcl

variable "location" {

type = string

default = "East US"

}

variable "resource\_group" {

type = string

default = "rg-priorauth"

}

variable "app\_name" {

type = string

default = "priorauth-portal"

}

```

### main.tf

```hcl

resource "azurerm\_resource\_group" "rg" {

name = var.resource\_group

location = var.location

}

resource "azurerm\_app\_service\_plan" "asp" {

name = "${var.app\_name}-plan"

location = var.location

resource\_group\_name = azurerm\_resource\_group.rg.name

kind = "Linux"

reserved = true

sku {

tier = "Standard"

size = "S1"

}

}

resource "azurerm\_app\_service" "app" {

name = var.app\_name

location = var.location

resource\_group\_name = azurerm\_resource\_group.rg.name

app\_service\_plan\_id = azurerm\_app\_service\_plan.asp.id

site\_config {

linux\_fx\_version = "DOTNETCORE|7.0"

}

app\_settings = {

"APPINSIGHTS\_INSTRUMENTATIONKEY" = azurerm\_application\_insights.ai.instrumentation\_key

"AzureWebJobsStorage" = azurerm\_storage\_account.sa.primary\_connection\_string

}

identity {

type = "SystemAssigned"

}

}

resource "azurerm\_application\_insights" "ai" {

name = "${var.app\_name}-ai"

location = var.location

resource\_group\_name = var.resource\_group

application\_type = "web"

}

resource "azurerm\_storage\_account" "sa" {

name = "priorauthstorage01"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

account\_tier = "Standard"

account\_replication\_type = "LRS"

}

resource "azurerm\_sql\_server" "sqlserver" {

name = "priorauthsql01"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = var.location

version = "12.0"

administrator\_login = "sqladmin"

administrator\_login\_password = "YourStrongP@ssword1!"

}

resource "azurerm\_sql\_database" "sqldb" {

name = "priorauthdb"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = var.location

server\_name = azurerm\_sql\_server.sqlserver.name

requested\_service\_objective\_name = "S0"

}

resource "azurerm\_servicebus\_namespace" "sb" {

name = "priorauthsb01"

location = var.location

resource\_group\_name = azurerm\_resource\_group.rg.name

sku = "Standard"

}

resource "azurerm\_key\_vault" "kv" {

name = "priorauthkv01"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

tenant\_id = data.azurerm\_client\_config.current.tenant\_id

sku\_name = "standard"

access\_policy {

tenant\_id = data.azurerm\_client\_config.current.tenant\_id

object\_id = data.azurerm\_client\_config.current.object\_id

secret\_permissions = [

"get",

"list",

"set",

"delete",

]

}

}

data "azurerm\_client\_config" "current" {}

```

### outputs.tf

```hcl

output "app\_service\_url" {

value = azurerm\_app\_service.app.default\_site\_hostname

}

output "sql\_connection\_string" {

value = azurerm\_sql\_database.sqldb.connection\_string

}

```

### How to Use

```bash

terraform init

terraform plan -out tfplan

terraform apply tfplan

```

### Next Steps:

- Would you like to add \*\*Azure AD B2C automation\*\*?

- Generate \*\*Visual CI/CD + Infra architecture diagram\*\*?

- Integrate monitoring dashboards with alerts?

✅ The **Terraform infrastructure** section is now fully integrated into your document. It includes:

* App Service Plan & App
* Azure SQL, Blob, Key Vault, Service Bus
* Application Insights
* Reusable variable and output blocks

Would you like me to:

* Add **monitoring and alerting** with preconfigured metrics?
* Generate **Visual Diagram** (Terraform + App Architecture)?
* Include **Secrets rotation + access policies** in Key Vault?

### ✅ High-Level Design (HLD) Interview Questions and Answers

...[Previous HLD + LLD Q&A content remains unchanged]...

---

### 🏥 High-Level Design Document – Healthcare Prior Authorization Workflow

...[Existing HLD content remains unchanged]...

---

## 🔐 Advanced Security Checklist – Aligned with OWASP Top 10 & HIPAA

...[Existing checklist remains unchanged]...

---

## ⚙️ Azure DevOps CI/CD Pipeline (DevSecOps Enabled)

...[DevOps YAML content remains unchanged]...

---

## 🧱 Terraform Infrastructure-as-Code (IaC) Setup for Prior Authorization Portal

...[Terraform main.tf, variables.tf, outputs.tf, provider.tf remain unchanged]...

---

## 📈 Monitoring and Alerting Integration (Terraform)

### Application Insights Dashboard with Alert Rules

```hcl

resource "azurerm\_monitor\_action\_group" "alerts" {

name = "pa-action-group"

resource\_group\_name = azurerm\_resource\_group.rg.name

short\_name = "pa-alerts"

email\_receiver {

name = "devops-alert"

email\_address = "alerts@priorauth.com"

use\_common\_alert\_schema = true

}

}

resource "azurerm\_monitor\_metric\_alert" "high\_cpu" {

name = "HighCPUAlert"

resource\_group\_name = azurerm\_resource\_group.rg.name

scopes = [azurerm\_app\_service.app.id]

description = "Alert for high CPU utilization"

criteria {

metric\_namespace = "Microsoft.Web/sites"

metric\_name = "Percentage CPU"

aggregation = "Average"

operator = "GreaterThan"

threshold = 80

dimension {

name = "Instance"

operator = "Include"

values = ["\*"]

}

}

evaluation\_frequency = "PT1M"

window\_size = "PT5M"

severity = 2

action {

action\_group\_id = azurerm\_monitor\_action\_group.alerts.id

}

}

resource "azurerm\_monitor\_metric\_alert" "http\_errors" {

name = "HTTP5xxAlert"

resource\_group\_name = azurerm\_resource\_group.rg.name

scopes = [azurerm\_app\_service.app.id]

criteria {

metric\_namespace = "Microsoft.Web/sites"

metric\_name = "Http5xx"

aggregation = "Total"

operator = "GreaterThan"

threshold = 5

}

evaluation\_frequency = "PT5M"

window\_size = "PT5M"

severity = 3

action {

action\_group\_id = azurerm\_monitor\_action\_group.alerts.id

}

}

```

### Dashboard Recommendations (Manually Created)

- CPU & Memory usage (App Service)

- Request count & latency (Application Insights)

- SQL DTU consumption (Azure SQL)

- Storage capacity (Blob)

- Dead-letter queues and failures (Service Bus)

---

Would you like to:

- Export a full \*\*Terraform deployment pack\*\*?

- Generate \*\*App Insights KQL queries\*\* for deeper monitoring?

- Auto-export dashboard to PDF or image for reporting?

✅ Monitoring and alerting are now integrated into your Terraform configuration with:

* Action Group for alerts
* CPU usage and HTTP 5xx error alerts
* Dashboard recommendations for observability

Would you like to proceed with:

1. **Prebuilt Application Insights KQL queries** for telemetry dashboards?
2. **Log Analytics Workspace** setup for centralized logging?
3. **Export Terraform module** with README for full infra deployment?

### ✅ **Purpose of Alerting in Cloud Architecture**

Alerting ensures **proactive monitoring** and **quick response** in critical business systems like **Healthcare Prior Authorization**, where performance, availability, and data integrity are essential.

#### 🔔 **Why is Alerting Needed?**

| **Purpose** | **Description** |
| --- | --- |
| **Reliability & Uptime** | Detect when system/service goes down or behaves abnormally |
| **Compliance** | Meet SLA/availability and legal obligations (e.g., HIPAA monitoring) |
| **Early Warning System** | Identify issues (e.g., CPU spike, HTTP 5xx, queue backlogs) before failure |
| **Incident Management** | Notify SRE/DevOps via email/SMS/Teams/Slack |
| **Auditing & Forensics** | Historical view of system anomalies, root cause analysis |

## 🧱 Applicable **Microservices Design Patterns** for Healthcare Prior Authorization System

Since this is a **high-volume**, **multi-role**, **event-driven**, and **modular system**, these are the most relevant design patterns:

### 1. **API Gateway Pattern**

* Acts as a single entry point for frontend & provider portals.
* Performs routing, rate-limiting, authentication (via Azure AD B2C).

✅ **Use in Project**: All APIs are exposed through Azure API Management.

// Sample routing config (abstracted)

[Route("api/patient")]

[Authorize]

public class PatientController : ControllerBase

{

[HttpGet("{id}")]

public IActionResult GetPatient(Guid id) => Ok(...);

}

### 2. **Strangler Fig Pattern**

* For migrating old intake system to new microservices module-by-module without downtime.

✅ **Use in Project**: Gradually replace legacy monolith with new modular services for request intake, clinical review, and notifications.

### 3. **Saga Pattern (Orchestration)**

* Handles long-running, distributed transactions across microservices.

✅ **Use in Project**: When creating a request → validating providers → CPT codes → clinical docs → eligibility → notify → review decision.

**Orchestration-Based Saga (e.g., Request Orchestrator Service)**

public class RequestOrchestrator

{

private readonly IEventBus \_eventBus;

public async Task StartSaga(PriorAuthRequest request)

{

await \_eventBus.Publish(new ProviderValidationStarted(request.Id));

// Wait for ProviderValidatedEvent...

await \_eventBus.Publish(new EligibilityCheckStarted(request.Id));

// ...

}

}

### 4. **CQRS (Command Query Responsibility Segregation)**

* Separate read and write models for better performance and scalability.

✅ **Use in Project**: Intake service can POST a request while dashboard uses optimized read model to query historical data.

// Command

public record CreateRequestCommand(PatientId PatientId, List<ProcedureCode> Codes) : IRequest<Guid>;

// Query

public record GetRequestByIdQuery(Guid Id) : IRequest<RequestDto>;

### 5. **Event Sourcing**

* Persist events instead of current state.
* Good for full audit trail, rollback, and async workflows.

✅ **Use in Project**: Track all actions like request submitted, status updated, reviewed, approved/denied.

public record RequestCreatedEvent(Guid RequestId, DateTime CreatedOn, Guid CreatedBy) : IDomainEvent;

public class Request : AggregateRoot

{

public void Apply(RequestCreatedEvent @event)

{

this.Id = @event.RequestId;

this.CreatedOn = @event.CreatedOn;

this.CreatedBy = @event.CreatedBy;

}

}

### 6. **Outbox Pattern (for Reliability)**

* Guarantees delivery of events to Kafka even if DB transaction succeeds but event publishing fails.

✅ **Use in Project**: When sending events to Kafka after request creation or review status update.

public class OutboxMessage

{

public Guid Id { get; set; }

public string Payload { get; set; }

public DateTime OccurredOn { get; set; }

public string Type { get; set; }

}

Use a background service to poll and publish messages to Kafka.

### 7. **Bulkhead & Circuit Breaker Patterns (Resilience)**

* Bulkhead: Isolate service failures (e.g., eligibility check)
* Circuit Breaker: Avoid retry storms on downstream failures

✅ **Use in Project**: External eligibility service calls, Cigna EDI service integration

services.AddHttpClient("Eligibility")

.AddTransientHttpErrorPolicy(policy =>

policy.CircuitBreakerAsync(5, TimeSpan.FromMinutes(2)));

## 🛠️ What's Next?

Would you like me to:

* Implement these patterns fully using DDD in C#?
* Generate unit tests for Saga and Outbox?
* Visualize the **Microservice Interaction Diagram**?
* Add these to the **Deployment Hardening Plan**?

Let me know which pattern you'd like expanded first.

### ✅ High-Level Design (HLD) Interview Questions and Answers

...[Previous HLD + LLD Q&A content remains unchanged]...

---

### 🏥 High-Level Design Document – Healthcare Prior Authorization Workflow

...[Existing HLD content remains unchanged]...

---

## 🔐 Advanced Security Checklist – Aligned with OWASP Top 10 & HIPAA

...[Existing checklist remains unchanged]...

---

## ⚙️ Azure DevOps CI/CD Pipeline (DevSecOps Enabled)

...[DevOps YAML content remains unchanged]...

---

## 🧱 Terraform Infrastructure-as-Code (IaC) Setup for Prior Authorization Portal

...[Terraform config remains unchanged]...

---

## 📈 Monitoring and Alerting Integration (Terraform)

...[Monitoring rules remain unchanged]...

---

## 🎯 Purpose of Alerting in Cloud Architecture

...[Purpose of alerting remains unchanged]...

---

## 🧰 Applicable Microservices Design Patterns (with Implementation)

...[Patterns remain unchanged]...

---

## 🔄 Complete Saga Orchestration (DDD-Based)

### Purpose

Manages distributed workflow for Prior Auth Request creation using Domain Events and Sagas to ensure consistency across microservices.

### Key Services

- `RequestOrchestratorService`: Starts and tracks saga

- `SagaStateStore`: Stores current state/progress

- `EventBus`: Publishes/subscribes domain events

- `Handlers`: React to events and trigger next step

### Domain Events

```csharp

public record RequestCreatedEvent(Guid RequestId) : IDomainEvent;

public record ProviderValidatedEvent(Guid RequestId, bool IsValid) : IDomainEvent;

public record EligibilityCheckedEvent(Guid RequestId, bool Eligible) : IDomainEvent;

public record CPTValidatedEvent(Guid RequestId, List ValidCodes) : IDomainEvent;

public record ReviewCompletedEvent(Guid RequestId, string FinalStatus) : IDomainEvent;

```

### Orchestrator Service

```csharp

public class RequestOrchestratorService

{

private readonly IEventBus \_eventBus;

private readonly ISagaStateStore \_stateStore;

public RequestOrchestratorService(IEventBus eventBus, ISagaStateStore stateStore)

{

\_eventBus = eventBus;

\_stateStore = stateStore;

}

public async Task StartSaga(Guid requestId)

{

await \_stateStore.CreateAsync(requestId);

await \_eventBus.Publish(new RequestCreatedEvent(requestId));

}

}

```

### Sample Event Handler

```csharp

public class RequestCreatedHandler : IDomainEventHandler

{

private readonly IEventBus \_eventBus;

public async Task Handle(RequestCreatedEvent evt)

{

// Step 1: Validate Provider

await \_eventBus.Publish(new StartProviderValidation(evt.RequestId));

}

}

```

### Saga State Store (In-Memory/Redis/SQL)

```csharp

public interface ISagaStateStore

{

Task CreateAsync(Guid requestId);

Task UpdateStepAsync(Guid requestId, string step);

Task GetCurrentStepAsync(Guid requestId);

}

```

### Handler Chaining with Domain Events

```csharp

public class ProviderValidatedHandler : IDomainEventHandler

{

private readonly IEventBus \_eventBus;

public async Task Handle(ProviderValidatedEvent evt)

{

if (evt.IsValid)

await \_eventBus.Publish(new EligibilityCheckStarted(evt.RequestId));

}

}

```

### Final Review Handler

```csharp

public class ReviewCompletedHandler : IDomainEventHandler

{

private readonly ISagaStateStore \_stateStore;

public async Task Handle(ReviewCompletedEvent evt)

{

await \_stateStore.UpdateStepAsync(evt.RequestId, "Completed");

// Notify external systems / close saga

}

}

```

This fully decouples orchestration using DDD events.

Would you like to:

- Generate integration tests for the saga flow?

- Add compensating actions for rollback?

- Export as diagram or markdown?

✅ Saga orchestration using DDD has been added, including:

* Domain events for each step
* Orchestrator service
* Event handlers
* Saga state persistence interface
* Final review and saga completion

Would you like to proceed with:

1. **Adding integration tests** for this saga flow?
2. Implementing **compensating actions** for failure recovery?
3. Generating a **visual orchestration flow diagram**?