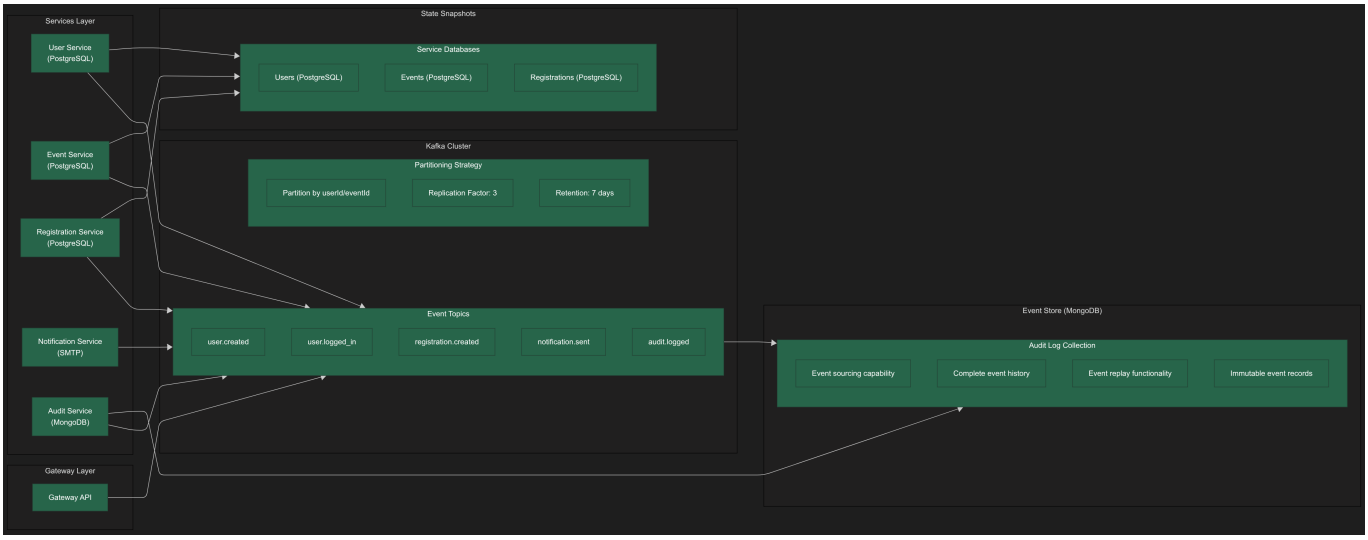
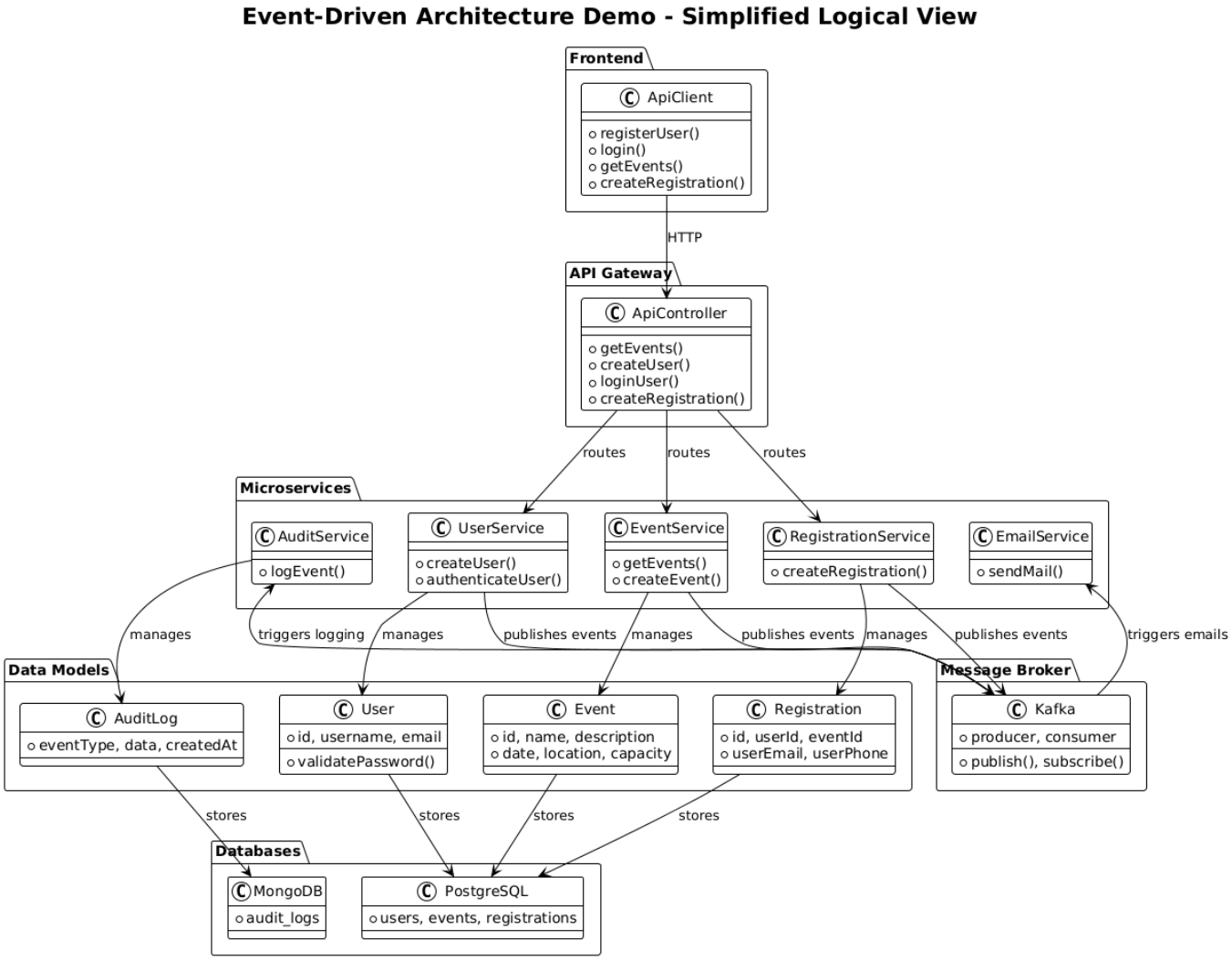
📋 Event-Driven Architecture Analysis - Lab 04

🖼️ Sơ Đồ Kiến Trúc (Architecture Diagrams)

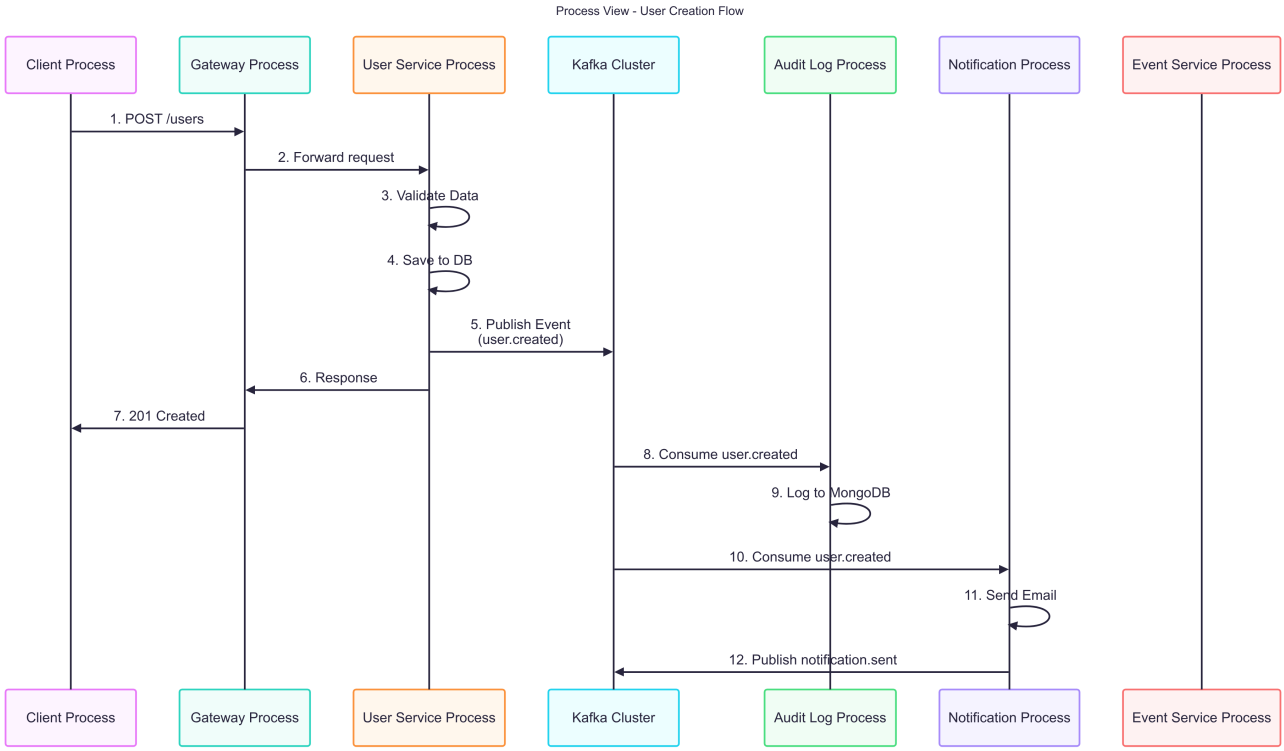
1. Sơ đồ lưu trữ sự kiện (Event Store Architecture)



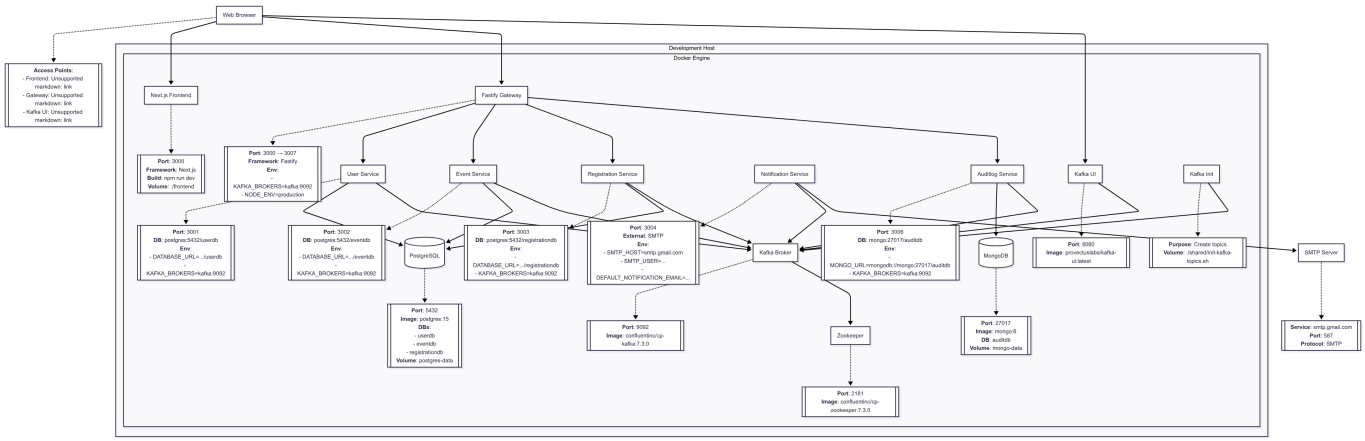
2. Sơ đồ logic tổng thể (Logical Architecture)



3. Sơ đồ luồng xử lý (Process View)

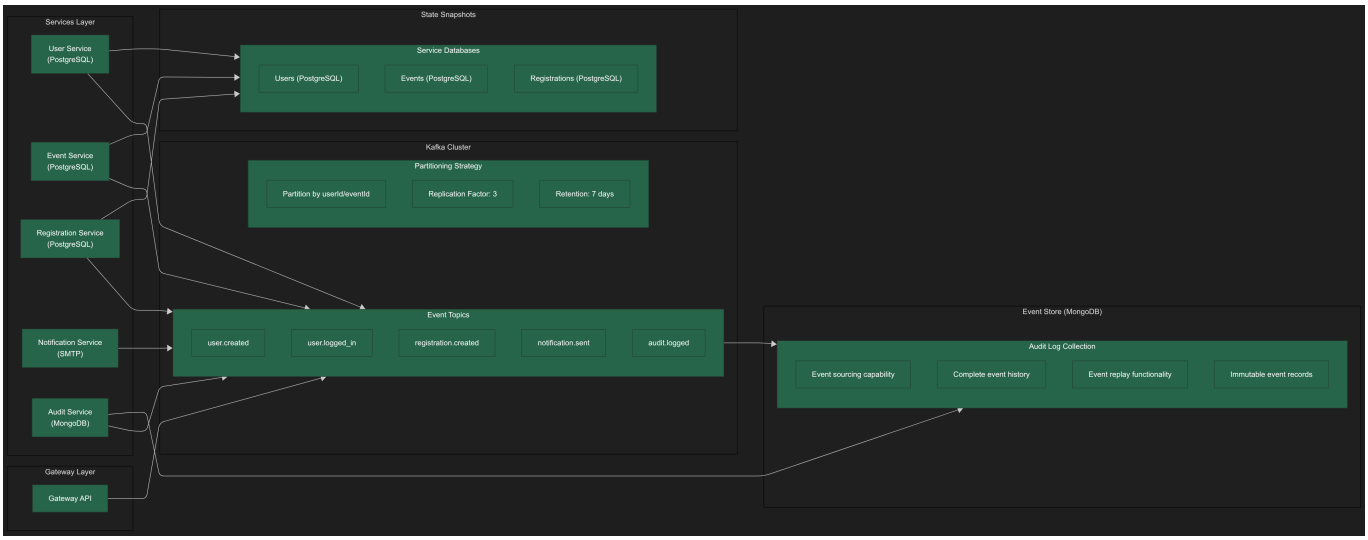


4. Sơ đồ triển khai (Deployment Architecture)



🎯 Câu 14: Đặc Tính Chất Lượng và Công Cụ Kiểm Tra

📊 Sơ Đồ Lưu Trữ Event-Driven Architecture



🔧 Công Cụ và Bước Triển Khai Event Storage

1. Kafka Setup và Configuration

```
# Tools: Apache Kafka, Zookeeper, Kafka UI
# Bước cài đặt:

1. Kafka Cluster Setup:
   # docker-compose.yml
   kafka:
     image: confluentinc/cp-kafka:7.3.0
     environment:
       KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
       KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
       KAFKA_LOG_RETENTION_HOURS: 168  # 7 days
       KAFKA_NUM_PARTITIONS: 3
       KAFKA_DEFAULT_REPLICATION_FACTOR: 1

2. Topic Configuration:
   # shared/init-kafka-topics.sh
   kafka-topics --create --topic user.created --partitions 3 --
replication-factor 1
   kafka-topics --create --topic registration.created --partitions 3 --
replication-factor 1

3. Schema Management:
   # event-schema.json
   {
     "type": "record",
     "name": "UserCreated",
     "fields": [
       {"name": "userId", "type": "string"},
       {"name": "timestamp", "type": "string"},
       {"name": "version", "type": "int", "default": 1}
     ]
   }
```

2. Event Store Implementation (MongoDB)

```
// Tools: MongoDB, Mongoose
// Bước cài đặt:

1. Event Store Model:
   // models/eventStore.js
   const eventStoreSchema = new mongoose.Schema({
     aggregateId: String,
     eventType: String,
     eventData: Object,
     eventVersion: Number,
     timestamp: { type: Date, default: Date.now },
     metadata: Object
   });

2. Event Writer:
   // services/eventWriter.js
   async function writeEvent(aggregateId, eventType, eventData, metadata =
{}) {
     const event = new EventStore({
       aggregateId,
       eventType,
       eventData,
       eventVersion: await getNextVersion(aggregateId),
       metadata
     });
     await event.save();

     // Publish to Kafka
     await publishToKafka(eventType, {
       aggregateId,
       eventData,
       timestamp: event.timestamp
     });
   }

3. Event Reader:
   // services/eventReader.js
   async function getEventsByAggregate(aggregateId, fromVersion = 0) {
     return await EventStore.find({
       aggregateId,
       eventVersion: { $gte: fromVersion }
     }).sort({ eventVersion: 1 });
   }

   async function replayEvents(aggregateId) {
     const events = await getEventsByAggregate(aggregateId);
     return events.reduce((state, event) => {
       return applyEvent(state, event);
     }, {});
   }
```

3. Event Sourcing Implementation

```
// Tools: Event Store, Snapshot Store
// Bước cài đặt:

1. Aggregate Root:
   // domain/User.js
   class User {
     constructor(id) {
       this.id = id;
       this.events = [];
       this.version = 0;
     }

     static async fromHistory(events) {
       const user = new User(events[0].aggregateId);
       events.forEach(event => user.apply(event));
       return user;
     }

     createUser(userData) {
       const event = {
         type: 'UserCreated',
         data: userData,
         timestamp: new Date().toISOString()
       };
       this.apply(event);
       this.events.push(event);
     }

     apply(event) {
       switch(event.type) {
         case 'UserCreated':
           this.username = event.data.username;
           this.email = event.data.email;
           break;
       }
       this.version++;
     }
   }

2. Repository Pattern:
   // repositories/UserRepository.js
   class UserRepository {
     async save(user) {
       for (const event of user.events) {
         await writeEvent(user.id, event.type, event.data);
       }
       user.events = []; // Clear uncommitted events
     }
```

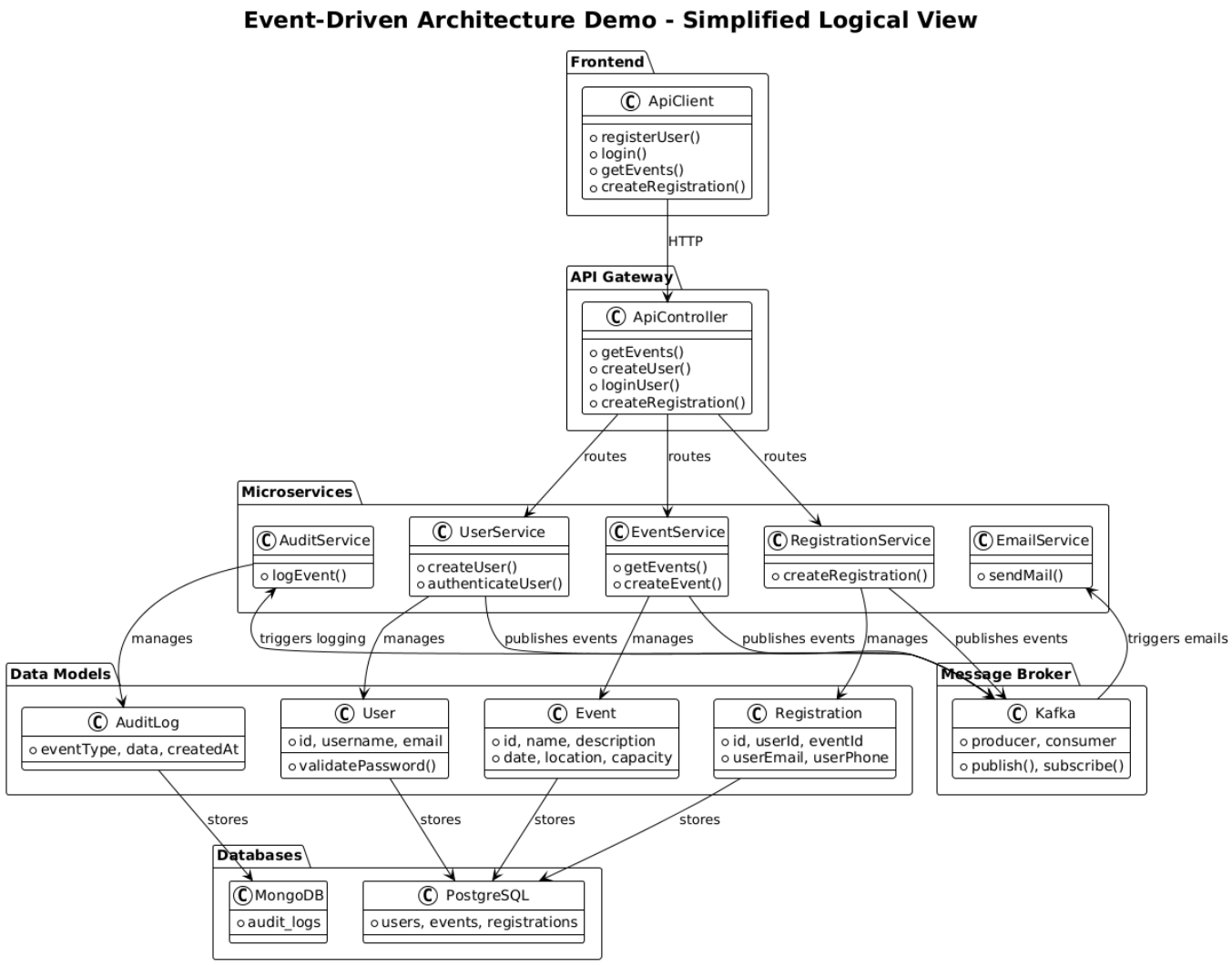
```
    async findById(userId) {
      const events = await getEventsByAggregate(userId);
      return events.length > 0 ? User.fromHistory(events) : null;
    }
  }
```

3. Command Handler:

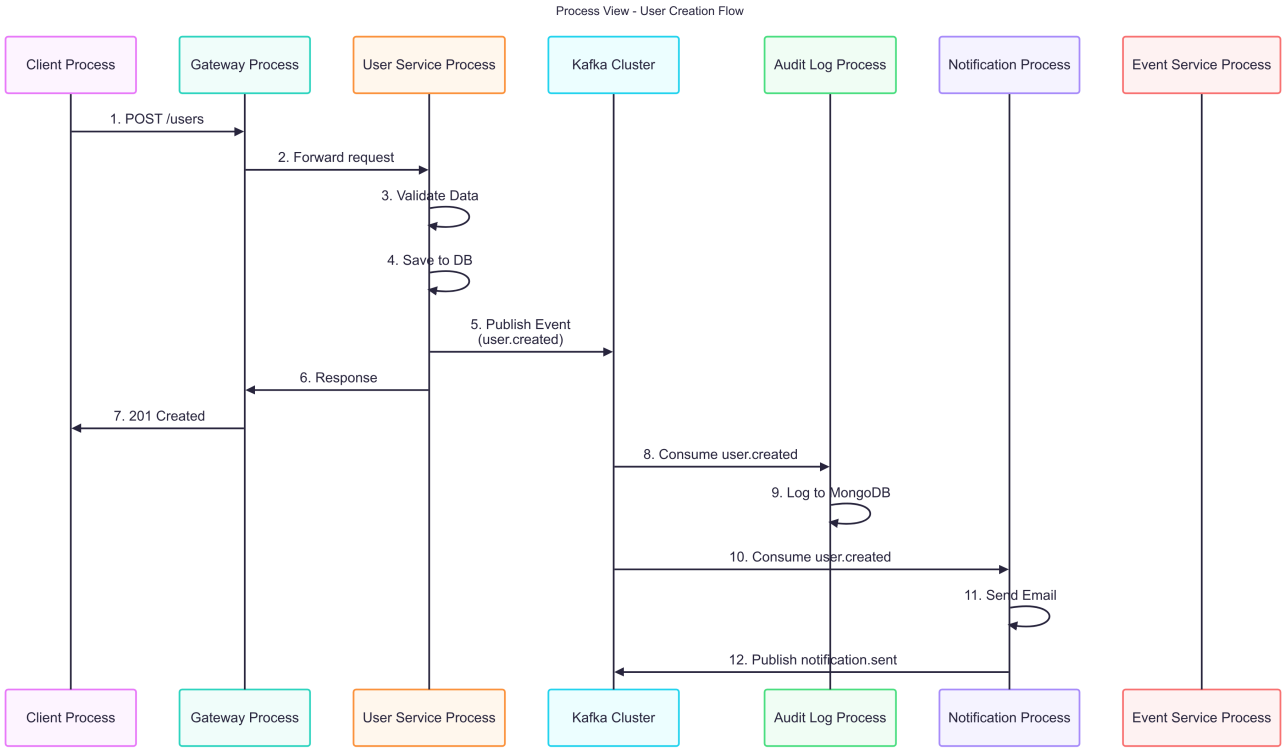
```
// handlers/CreateUserHandler.js
async function handle(command) {
  const user = new User(command.userId);
  user.createUser(command.userData);
  await userRepository.save(user);

  return { success: true, userId: user.id };
}
```

🎯 Câu 15: Góc Nhìn Logic và Process View

📐 Logical View - Kiến Trúc Logic



🔄 Process View - Luồng Xử Lý



🔧 Công Cụ và Implementation

1. Input Validation Service

```
// Tools: Joi, Express-validator, Fastify schema validation
// Bước implementation:

1. Schema Definition:
   // schemas/userSchema.js
   const Joi = require('joi');

   const createUserSchema = Joi.object({
     username: Joi.string().alphanum().min(3).max(30).required(),
     email: Joi.string().email().required(),
     password: Joi.string().min(8).pattern(new RegExp('^(?=.*[a-z])(?=.*
[A-Z])(?=.*[0-9])(?=.*[!@#\$%\^&\*])')).required(),
     fullName: Joi.string().min(2).max(100).required()
   });

2. Validation Middleware:
   // middleware/validation.js
   const validateInput = (schema) => {
     return (req, res, next) => {
       const { error, value } = schema.validate(req.body);
       if (error) {
         return res.status(400).json({
           success: false,
           message: 'Validation failed',
           errors: error.details.map(detail => ({
             field: detail.path[0],
             message: detail.message
```

```
            }))
          });
        }
        req.validatedData = value;
        next();
      };
    };

3. Business Logic Validation:
    // services/userValidationService.js
    class UserValidationService {
      async validateUniqueConstraints(userData) {
        const existingUser = await User.findOne({
          $or: [
            { email: userData.email },
            { username: userData.username }
          ]
        });

        if (existingUser) {
          throw new ValidationError('User with this email or username
already exists');
        }
      }

      async validateBusinessRules(userData) {
        // Check age requirements
        if (userData.dateOfBirth) {
          const age = this.calculateAge(userData.dateOfBirth);
          if (age < 13) {
            throw new ValidationError('User must be at least 13 years
old');
          }
        }

        // Check domain whitelist for email
        const allowedDomains = ['gmail.com', 'company.com'];
        const emailDomain = userData.email.split('@')[1];
        if (!allowedDomains.includes(emailDomain)) {
          throw new ValidationError('Email domain not allowed');
        }
      }
    }
```

2. Event-Driven Data Persistence

```
// Tools: Domain Events, Event Store, CQRS
// Bước implementation:

1. Command Handler với Validation:
    // handlers/CreateUserCommandHandler.js
```

```
    class CreateUserCommandHandler {
      constructor(userRepository, validationService, eventBus) {
        this.userRepository = userRepository;
        this.validationService = validationService;
        this.eventBus = eventBus;
      }

      async handle(command) {
        try {
          // 1. Input Validation
          await this.validationService.validateInput(command.userData);

          // 2. Business Rule Validation
          await
this.validationService.validateUniqueConstraints(command.userData);
          await
this.validationService.validateBusinessRules(command.userData);

          // 3. Create Domain Object
          const user = new User(command.userData);

          // 4. Generate Domain Events
          user.recordEvent(new UserCreatedEvent({
            userId: user.id,
            username: user.username,
            email: user.email,
            timestamp: new Date().toISOString()
          }));

          // 5. Persist to Database
          await this.userRepository.save(user);

          // 6. Publish Events
          await this.eventBus.publishAll(user.getUncommittedEvents());
          user.markEventsAsCommitted();

          return {
            success: true,
            userId: user.id,
            message: 'User created successfully'
          };

        } catch (error) {
          // 7. Error Handling & Compensation
          await this.handleError(error, command);
          throw error;
        }
      }

      async handleError(error, command) {
        // Log error event
        await this.eventBus.publish(new UserCreationFailedEvent({
          commandId: command.id,
          userData: command.userData,
```

```
          error: error.message,
          timestamp: new Date().toISOString()
        }));
      }
    }
```

2. Event-Driven Repository:

```
// repositories/EventDrivenUserRepository.js
class EventDrivenUserRepository {
  constructor(eventStore, snapshotStore) {
    this.eventStore = eventStore;
    this.snapshotStore = snapshotStore;
  }

  async save(user) {
    const events = user.getUncommittedEvents();

    // Start transaction
    const session = await this.eventStore.startSession();
    session.startTransaction();

    try {
      // Save events atomically
      for (const event of events) {
        await this.eventStore.append({
          aggregateId: user.id,
          eventType: event.constructor.name,
          eventData: event.getData(),
          expectedVersion: user.version,
          timestamp: event.timestamp
        }, { session });
      }

      // Update snapshot if needed
      if (events.length > 10) {
        await this.snapshotStore.save({
          aggregateId: user.id,
          data: user.getSnapshot(),
          version: user.version
        }, { session });
      }

      await session.commitTransaction();

    } catch (error) {
      await session.abortTransaction();
      throw error;
    } finally {
      session.endSession();
    }
  }

  async findById(userId) {
    // Try to load from snapshot first
```

```
        const snapshot = await this.snapshotStore.findLatest(userId);
        let user;
        let fromVersion = 0;

        if (snapshot) {
          user = User.fromSnapshot(snapshot.data);
          fromVersion = snapshot.version + 1;
        } else {
          user = new User({ id: userId });
        }

        // Load events since snapshot
        const events = await this.eventStore.getEvents(userId,
fromVersion);
        user.replayEvents(events);

        return user;
      }
    }
```

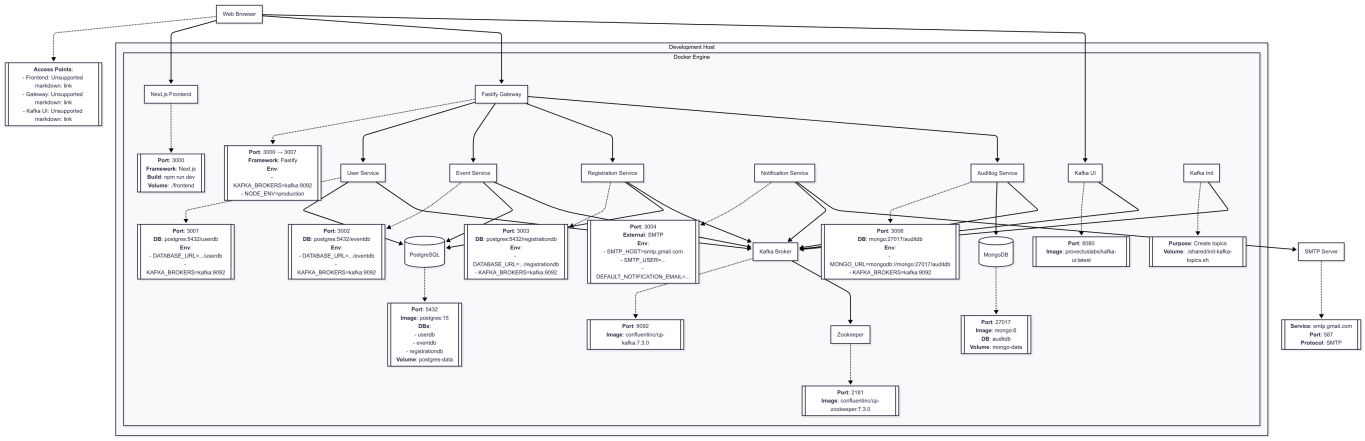
3. Complete Feature Implementation:

```
    // controllers/UserController.js
    class UserController {
      constructor(commandBus, queryBus) {
        this.commandBus = commandBus;
        this.queryBus = queryBus;
      }

      async createUser(req, res) {
        try {
          const command = new CreateUserCommand({
            id: generateId(),
            userData: req.validatedData,
            requestId: req.headers['x-request-id'],
            userId: req.user?.id // For audit trail
          });

          const result = await this.commandBus.send(command);

          res.status(201).json({
            success: true,
            data: {
              userId: result.userId,
              message: result.message
            },
            meta: {
              requestId: command.requestId,
              timestamp: new Date().toISOString()
            }
          });

        } catch (error) {
          if (error instanceof ValidationError) {
            res.status(400).json({
```

```
              success: false,
              error: 'Validation failed',
              details: error.message,
              requestId: req.headers['x-request-id']
            });
          } else {
            res.status(500).json({
              success: false,
              error: 'Internal server error',
              requestId: req.headers['x-request-id']
            });
          }
        }
      }
    }
```

🎯 Câu 16: Góc Nhìn Triển Khai (Deployment View)

🏗️ Deployment Architecture



🛠️ Công Cụ Triển Khai

1. Container Orchestration

```
# Tools: Docker, Kubernetes, Docker Compose
# Bước triển khai:

1. Containerization:
   # Dockerfile cho từng service
   FROM node:18-alpine
   WORKDIR /app
   COPY package*.json ./
   RUN npm ci --only=production
   COPY . .
   EXPOSE 3001
   HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
     CMD curl -f http://localhost:3001/health || exit 1
   CMD ["node", "src/index.js"]
```

```
2. Docker Compose (Development):
   # docker-compose.yml
   version: '3.8'
   services:
     user-service:
       build: ./user-service
       ports:
         - "3001:3001"
       environment:
         - DATABASE_URL=postgres://postgres:admin@postgres:5432/userdb
         - KAFKA_BROKERS=kafka:9092
       depends_on:
         - postgres
         - kafka
       deploy:
         replicas: 2
         resources:
           limits:
             memory: 512M
             cpus: "0.5"

3. Kubernetes Deployment:
   # k8s/user-service-deployment.yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: user-service
   spec:
     replicas: 3
     selector:
       matchLabels:
         app: user-service
     template:
       metadata:
         labels:
           app: user-service
       spec:
         containers:
         - name: user-service
           image: your-registry/user-service:latest
           ports:
           - containerPort: 3001
           env:
           - name: DATABASE_URL
             valueFrom:
               secretKeyRef:
                 name: db-secret
                 key: database-url
           resources:
             requests:
               memory: "256Mi"
               cpu: "250m"
             limits:
```

```
              memory: "512Mi"
              cpu: "500m"
          livenessProbe:
            httpGet:
              path: /health
              port: 3001
            initialDelaySeconds: 30
            periodSeconds: 10
          readinessProbe:
            httpGet:
              path: /ready
              port: 3001
            initialDelaySeconds: 5
            periodSeconds: 5
```

2. Infrastructure as Code

```
# Tools: Terraform, AWS CloudFormation, Ansible
# Bước triển khai:

1. Terraform Infrastructure:
   # infrastructure/main.tf
   provider "aws" {
     region = "us-west-2"
   }

   # EKS Cluster
   module "eks" {
     source = "terraform-aws-modules/eks/aws"

     cluster_name    = "eda-demo-cluster"
     cluster_version = "1.27"

     vpc_id     = module.vpc.vpc_id
     subnet_ids = module.vpc.private_subnets

     eks_managed_node_groups = {
       main = {
         desired_size = 3
         max_size     = 10
         min_size     = 3

         instance_types = ["t3.medium"]

         k8s_labels = {
           Environment = "production"
           Application = "eda-demo"
         }
       }
     }
   }
```

```
    # RDS PostgreSQL
    resource "aws_db_instance" "postgres" {
      identifier = "eda-demo-postgres"

      engine         = "postgres"
      engine_version = "14.7"
      instance_class = "db.t3.micro"

      allocated_storage     = 20
      max_allocated_storage = 100

      db_name  = "userdb"
      username = "postgres"
      password = var.db_password

      backup_retention_period = 7
      backup_window          = "07:00-09:00"
      maintenance_window     = "Sun:09:00-Sun:11:00"

      skip_final_snapshot = true
    }

    # MSK Kafka Cluster
    resource "aws_msk_cluster" "kafka" {
      cluster_name           = "eda-demo-kafka"
      kafka_version          = "2.8.1"
      number_of_broker_nodes = 3

      broker_node_group_info {
        instance_type   = "kafka.t3.small"
        ebs_volume_size = 20
        client_subnets  = module.vpc.private_subnets

        security_groups = [aws_security_group.kafka.id]
      }
    }
```

2. Ansible Playbook:

```
    # playbooks/deploy.yml
    ---
    - name: Deploy EDA Demo Application
      hosts: kubernetes
      tasks:
        - name: Apply Kubernetes manifests
          k8s:
            state: present
            definition: "{{ item }}"
          with_fileglob:
            - "../k8s/*.yaml"

        - name: Wait for deployment rollout
          k8s_info:
            api_version: apps/v1
```

```
            kind: Deployment
            name: "{{ item }}"
            namespace: default
            wait_condition:
              type: Progressing
              status: "True"
              reason: NewReplicaSetAvailable
            wait_timeout: 600
          loop:
            - user-service
            - event-service
            - registration-service
            - notification-service
            - audit-service
            - gateway-service
```

3. CI/CD Pipeline

```
# Tools: GitHub Actions, Jenkins, GitLab CI
# Bước triển khai:

1. GitHub Actions Workflow:
   # .github/workflows/deploy.yml
   name: Deploy to Production

   on:
     push:
       branches: [main]

   jobs:
     test:
       runs-on: ubuntu-latest
       steps:
         - uses: actions/checkout@v3
         - uses: actions/setup-node@v3
           with:
             node-version: '18'
         - run: npm ci
         - run: npm run test
         - run: npm run lint

     build:
       needs: test
       runs-on: ubuntu-latest
       strategy:
         matrix:
           service: [user-service, event-service, registration-service,
notification-service, audit-service, gateway]
       steps:
         - uses: actions/checkout@v3
```

```
          - name: Build Docker Image
            run: |
              docker build -t ${{ secrets.REGISTRY }}/${{ matrix.service
}}:${{ github.sha }} ./${{ matrix.service }}
              docker build -t ${{ secrets.REGISTRY }}/${{ matrix.service
}}:latest ./${{ matrix.service }}

          - name: Push to Registry
            run: |
              echo ${{ secrets.REGISTRY_PASSWORD }} | docker login ${{
secrets.REGISTRY }} -u ${{ secrets.REGISTRY_USERNAME }} --password-stdin
              docker push ${{ secrets.REGISTRY }}/${{ matrix.service }}:${{
github.sha }}
              docker push ${{ secrets.REGISTRY }}/${{ matrix.service
}}:latest

      deploy:
        needs: build
        runs-on: ubuntu-latest
        environment: production
        steps:
          - uses: actions/checkout@v3

          - name: Setup kubectl
            uses: azure/setup-kubectl@v3
            with:
              version: 'latest'

          - name: Configure AWS credentials
            uses: aws-actions/configure-aws-credentials@v2
            with:
              aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
              aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
              aws-region: us-west-2

          - name: Update kubeconfig
            run: aws eks update-kubeconfig --name eda-demo-cluster

          - name: Deploy to Kubernetes
            run: |
              # Update image tags in manifests
              sed -i 's|image: .*|image: ${{ secrets.REGISTRY }}/user-
service:${{ github.sha }}|' k8s/user-service-deployment.yaml

              # Apply manifests
              kubectl apply -f k8s/

              # Wait for rollout
              kubectl rollout status deployment/user-service
              kubectl rollout status deployment/event-service
              kubectl rollout status deployment/registration-service
              kubectl rollout status deployment/notification-service
              kubectl rollout status deployment/audit-service
              kubectl rollout status deployment/gateway-service
```

```
2. Monitoring và Logging:
   # monitoring/prometheus-config.yml
   global:
     scrape_interval: 15s

   scrape_configs:
     - job_name: 'kubernetes-services'
       kubernetes_sd_configs:
         - role: service
       relabel_configs:
         - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_scrape]
           action: keep
           regex: true
         - source_labels:
[__meta_kubernetes_service_annotation_prometheus_io_path]
           action: replace
           target_label: __metrics_path__
           regex: (.+)

   # logging/fluentd-config.yml
   <source>
     @type tail
     path /var/log/containers/*.log
     pos_file /var/log/fluentd-containers.log.pos
     tag kubernetes.*
     format json
   </source>

   <match kubernetes.**>
     @type elasticsearch
     host elasticsearch.logging.svc.cluster.local
     port 9200
     index_name kubernetes
   </match>

3. Production Deployment Script:
   #!/bin/bash
   # scripts/deploy-production.sh

   set -e

   echo "🚀 Starting production deployment..."

   # Pre-deployment checks
   echo "📋 Running pre-deployment checks..."
   kubectl get nodes
   kubectl get pods -n default

   # Database migrations
   echo "🗄️ Running database migrations..."
   kubectl apply -f k8s/migrations-job.yaml
   kubectl wait --for=condition=complete job/db-migration --timeout=300s
```

```
    # Deploy services in order
    echo "🔄 Deploying services..."

    # 1. Deploy data tier
    kubectl apply -f k8s/postgres-deployment.yaml
    kubectl apply -f k8s/mongodb-deployment.yaml
    kubectl wait --for=condition=ready pod -l app=postgres --timeout=300s
    kubectl wait --for=condition=ready pod -l app=mongodb --timeout=300s

    # 2. Deploy message broker
    kubectl apply -f k8s/kafka-deployment.yaml
    kubectl wait --for=condition=ready pod -l app=kafka --timeout=300s

    # 3. Deploy microservices
    for service in user-service event-service registration-service
notification-service audit-service; do
      echo "Deploying $service..."
      kubectl apply -f k8s/$service-deployment.yaml
      kubectl rollout status deployment/$service --timeout=300s
    done

    # 4. Deploy gateway last
    kubectl apply -f k8s/gateway-deployment.yaml
    kubectl rollout status deployment/gateway-service --timeout=300s

    # Post-deployment verification
    echo "✅ Running post-deployment verification..."

    # Health checks
    for service in user-service event-service registration-service
notification-service audit-service gateway-service; do
      kubectl exec deployment/$service -- curl -f
http://localhost:3001/health
    done

    # Smoke tests
    kubectl apply -f k8s/smoke-tests-job.yaml
    kubectl wait --for=condition=complete job/smoke-tests --timeout=300s

    echo "🎉 Production deployment completed successfully!"

    # Display service URLs
    echo "📍 Service endpoints:"
    kubectl get ingress
```

📊 Monitoring và Observability

```
# Tools: Prometheus, Grafana, ELK Stack, Jaeger
# Bước setup:
```

```
1. Monitoring Stack:
   # monitoring/prometheus-operator.yaml
   apiVersion: monitoring.coreos.com/v1
   kind: ServiceMonitor
   metadata:
     name: eda-demo-services
   spec:
     selector:
       matchLabels:
         monitoring: enabled
     endpoints:
     - port: metrics
       path: /metrics
       interval: 30s

2. Grafana Dashboards:
   # monitoring/grafana-dashboard.json
   {
     "dashboard": {
       "title": "EDA Demo - System Overview",
       "panels": [
         {
           "title": "Event Processing Rate",
           "type": "graph",
           "targets": [
             {
               "expr": "rate(kafka_consumer_records_consumed_total[5m])",
               "legendFormat": "{{ service }}"
             }
           ]
         },
         {
           "title": "Service Response Times",
           "type": "graph",
           "targets": [
             {
               "expr": "histogram_quantile(0.95,
rate(http_request_duration_seconds_bucket[5m]))",
               "legendFormat": "95th percentile - {{ service }}"
             }
           ]
         }
       ]
     }
   }

3. Distributed Tracing:
   # tracing/jaeger-deployment.yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: jaeger
   spec:
     template:
```

```
      spec:
        containers:
        - name: jaeger
          image: jaegertracing/all-in-one:1.35
          ports:
          - containerPort: 16686
          - containerPort: 14268
          env:
          - name: COLLECTOR_ZIPKIN_HTTP_PORT
            value: "9411"
```

Hệ thống này đảm bảo:

- ✅ **High Availability** với multiple replicas
- ✅ **Scalability** với horizontal pod autoscaling
- ✅ **Monitoring** với Prometheus/Grafana
- ✅ **Logging** với ELK stack
- ✅ **Tracing** với Jaeger
- ✅ **Security** với network policies và secrets management
- ✅ **Automation** với CI/CD pipelines