📋 Event Schemas Documentation

📍 Tổng Quan Event Schemas trong Dự Án

Event-driven architecture trong dự án này sử dụng schema-driven approach để đảm bảo consistency và loose coupling giữa các services. Tất cả event schemas được centralized trong thư mục `/shared/` để tạo ra "single source of truth".

🏗️ Cấu Trúc Event Schemas

```
/shared/
├── events/
│   ├── user.events.js          # User event schemas
│   ├── order.events.js         # Order event schemas
│   ├── product.events.js       # Product event schemas
│   └── notification.events.js  # Notification event schemas
├── validation/
│   ├── event-validator.js      # Schema validation logic
│   └── schemas/
│       ├── base-event.schema.js
│       └── common.schemas.js
└── types/
    └── event.types.js          # TypeScript definitions
```

📝 Chi Tiết Event Schemas

1. User Events Schema (`/shared/events/user.events.js`)

```
const USER_EVENTS = {
  USER_REGISTERED: 'user.registered',
  USER_UPDATED: 'user.updated',
  USER_DELETED: 'user.deleted',
  USER_EMAIL_VERIFIED: 'user.email.verified'
};

const USER_REGISTERED_SCHEMA = {
  type: 'object',
  properties: {
    userId: {
      type: 'string',
      format: 'uuid',
      description: 'Unique identifier for the user'
    },
    email: {
      type: 'string',
      format: 'email',
      description: 'User email address'
```

```
    },
    name: {
      type: 'string',
      minLength: 1,
      maxLength: 100,
      description: 'User full name'
    },
    timestamp: {
      type: 'string',
      format: 'date-time',
      description: 'When the event occurred'
    },
    metadata: {
      type: 'object',
      properties: {
        source: { type: 'string', enum: ['web', 'mobile', 'api'] },
        version: { type: 'string', default: '1.0' }
      }
    }
  },
  required: ['userId', 'email', 'name', 'timestamp']
};

const USER_UPDATED_SCHEMA = {
  type: 'object',
  properties: {
    userId: { type: 'string', format: 'uuid' },
    changes: {
      type: 'object',
      properties: {
        email: { type: 'string', format: 'email' },
        name: { type: 'string', minLength: 1 },
        phone: { type: 'string' }
      }
    },
    timestamp: { type: 'string', format: 'date-time' }
  },
  required: ['userId', 'changes', 'timestamp']
};

module.exports = {
  USER_EVENTS,
  USER_REGISTERED_SCHEMA,
  USER_UPDATED_SCHEMA
};
```

2. Order Events Schema (`/shared/events/order.events.js`)

```
const ORDER_EVENTS = {
  ORDER_CREATED: 'order.created',
  ORDER_UPDATED: 'order.updated',
```

```
  ORDER_CANCELLED: 'order.cancelled',
  ORDER_COMPLETED: 'order.completed',
  ORDER_PAYMENT_PROCESSED: 'order.payment.processed'
};

const ORDER_CREATED_SCHEMA = {
  type: 'object',
  properties: {
    orderId: {
      type: 'string',
      format: 'uuid',
      description: 'Unique order identifier'
    },
    userId: {
      type: 'string',
      format: 'uuid',
      description: 'Customer who placed the order'
    },
    items: {
      type: 'array',
      minItems: 1,
      items: {
        type: 'object',
        properties: {
          productId: { type: 'string', format: 'uuid' },
          productName: { type: 'string' },
          quantity: { type: 'number', minimum: 1 },
          unitPrice: { type: 'number', minimum: 0 },
          totalPrice: { type: 'number', minimum: 0 }
        },
        required: ['productId', 'quantity', 'unitPrice', 'totalPrice']
      }
    },
    totalAmount: {
      type: 'number',
      minimum: 0,
      description: 'Total order amount'
    },
    currency: {
      type: 'string',
      enum: ['USD', 'EUR', 'VND'],
      default: 'USD'
    },
    shippingAddress: {
      type: 'object',
      properties: {
        street: { type: 'string' },
        city: { type: 'string' },
        country: { type: 'string' },
        postalCode: { type: 'string' }
      },
      required: ['street', 'city', 'country']
    },
    timestamp: { type: 'string', format: 'date-time' }
```

```
  },
  required: ['orderId', 'userId', 'items', 'totalAmount', 'timestamp']
};

module.exports = {
  ORDER_EVENTS,
  ORDER_CREATED_SCHEMA
};
```

3. Product Events Schema (/shared/events/product.events.js)

```
const PRODUCT_EVENTS = {
  PRODUCT_CREATED: 'product.created',
  PRODUCT_UPDATED: 'product.updated',
  PRODUCT_DELETED: 'product.deleted',
  PRODUCT_STOCK_UPDATED: 'product.stock.updated',
  PRODUCT_PRICE_CHANGED: 'product.price.changed'
};

const PRODUCT_STOCK_UPDATED_SCHEMA = {
  type: 'object',
  properties: {
    productId: { type: 'string', format: 'uuid' },
    previousStock: { type: 'number', minimum: 0 },
    currentStock: { type: 'number', minimum: 0 },
    reason: {
      type: 'string',
      enum: ['sale', 'restock', 'adjustment', 'return']
    },
    timestamp: { type: 'string', format: 'date-time' }
  },
  required: ['productId', 'currentStock', 'reason', 'timestamp']
};

module.exports = {
  PRODUCT_EVENTS,
  PRODUCT_STOCK_UPDATED_SCHEMA
};
```

🔧 Event Validation System

Event Validator (/shared/validation/event-validator.js)

```
const Ajv = require('ajv');
const addFormats = require('ajv-formats');

// Import all schemas
const { USER_REGISTERED_SCHEMA, USER_UPDATED_SCHEMA } =
require('../events/user.events');
```

```
const { ORDER_CREATED_SCHEMA } = require('../events/order.events');
const { PRODUCT_STOCK_UPDATED_SCHEMA } =
require('../events/product.events');

class EventValidator {
  constructor() {
    this.ajv = new Ajv({ allErrors: true, verbose: true });
    addFormats(this.ajv);

    // Pre-compile all schemas for better performance
    this.compiledSchemas = new Map();
    this._initializeSchemas();
  }

  _initializeSchemas() {
    const schemas = {
      'user.registered': USER_REGISTERED_SCHEMA,
      'user.updated': USER_UPDATED_SCHEMA,
      'order.created': ORDER_CREATED_SCHEMA,
      'product.stock.updated': PRODUCT_STOCK_UPDATED_SCHEMA
    };

    Object.entries(schemas).forEach(([eventType, schema]) => {
      const validator = this.ajv.compile(schema);
      this.compiledSchemas.set(eventType, validator);
    });
  }

  validate(eventType, payload) {
    const validator = this.compiledSchemas.get(eventType);

    if (!validator) {
      throw new Error(`No schema found for event type: ${eventType}`);
    }

    const isValid = validator(payload);

    if (!isValid) {
      const errors = validator.errors.map(error => ({
        field: error.instancePath || error.schemaPath,
        message: error.message,
        rejectedValue: error.data
      }));

      throw new ValidationError(`Event validation failed for
${eventType}`, errors);
    }

    return true;
  }

  getAvailableEventTypes() {
    return Array.from(this.compiledSchemas.keys());
  }
```

```
}

class ValidationError extends Error {
  constructor(message, errors) {
    super(message);
    this.name = 'ValidationError';
    this.errors = errors;
  }
}

// Singleton instance
const eventValidator = new EventValidator();

module.exports = {
  EventValidator: eventValidator,
  ValidationError
};
```

📡 Event Publishing với Schema Validation

User Event Publisher (`/services/user-service/src/events/user-event-publisher.js`)

```
const { EventValidator } = require('../../../shared/validation/event-validator');
const { USER_EVENTS } = require('../../../shared/events/user.events');
const kafkaProducer = require('../infrastructure/kafka-producer');

class UserEventPublisher {
  static async publishUserRegistered(userData) {
    const event = {
      type: USER_EVENTS.USER_REGISTERED,
      payload: {
        userId: userData.id,
        email: userData.email,
        name: userData.name,
        timestamp: new Date().toISOString(),
        metadata: {
          source: 'user-service',
          version: '1.0'
        }
      }
    };

    try {
      // Validate event trước khi publish
      EventValidator.validate(event.type, event.payload);

      // Publish event
      await kafkaProducer.send({
        topic: 'user-events',
```

```
          messages: [{
            key: userData.id,
            value: JSON.stringify(event),
            headers: {
              'event-type': event.type,
              'event-version': '1.0'
            }
          }]
        });

        console.log(`✅ Published event: ${event.type} for user
${userData.id}`);

      } catch (error) {
        console.error(`❌ Failed to publish event: ${error.message}`);

        // Log validation errors chi tiết
        if (error.name === 'ValidationError') {
          console.error('Validation errors:', error.errors);
        }

        throw error;
      }
    }

    static async publishUserUpdated(userId, changes) {
      const event = {
        type: USER_EVENTS.USER_UPDATED,
        payload: {
          userId,
          changes,
          timestamp: new Date().toISOString()
        }
      };

      // Validate và publish
      EventValidator.validate(event.type, event.payload);
      await kafkaProducer.send({
        topic: 'user-events',
        messages: [{
          key: userId,
          value: JSON.stringify(event)
        }]
      });
    }
  }

  module.exports = UserEventPublisher;
```

📨 Event Consumer với Schema Validation

Notification Event Consumer (/services/notification-service/src/consumers/user-event-consumer.js)

```
const { EventValidator } = require('../../../shared/validation/event-validator');
const { USER_EVENTS } = require('../../../shared/events/user.events');
const emailService = require('../services/email-service');

class UserEventConsumer {
  async handleMessage(message) {
    const { type: eventType, payload } = JSON.parse(message.value);

    try {
      // Validate incoming event
      EventValidator.validate(eventType, payload);

      // Route to appropriate handler
      switch (eventType) {
        case USER_EVENTS.USER_REGISTERED:
          await this.handleUserRegistered(payload);
          break;
        case USER_EVENTS.USER_UPDATED:
          await this.handleUserUpdated(payload);
          break;
        default:
          console.log(`Unhandled event type: ${eventType}`);
      }

    } catch (error) {
      if (error.name === 'ValidationError') {
        console.error(`Invalid event schema for ${eventType}:`, error.errors);

        // Send to dead letter queue for manual inspection
        await this.sendToDeadLetterQueue(message, error);
        return;
      }

      // Re-throw other errors để retry mechanism xử lý
      throw error;
    }
  }

  async handleUserRegistered(payload) {
    console.log(`👤 Processing user registration: ${payload.userId}`);

    // Send welcome email
    await emailService.sendWelcomeEmail({
      to: payload.email,
      name: payload.name,
      userId: payload.userId
    });
```

```
    // Create user profile in notification preferences
    await this.createNotificationProfile(payload.userId, payload.email);

    console.log(`✅ Welcome email sent to ${payload.email}`);
  }

  async handleUserUpdated(payload) {
    console.log(`👤 Processing user update: ${payload.userId}`);

    // Update notification preferences if email changed
    if (payload.changes.email) {
      await this.updateEmailPreferences(
        payload.userId,
        payload.changes.email
      );
    }
  }

  async sendToDeadLetterQueue(message, error) {
    // Implementation for dead letter queue
    console.error('Sending message to DLQ:', {
      topic: message.topic,
      partition: message.partition,
      offset: message.offset,
      error: error.message
    });
  }
}

module.exports = UserEventConsumer;
```

🔄 Event Versioning Strategy

Base Event Schema (`/shared/schemas/base-event.schema.js`)

```
const BASE_EVENT_SCHEMA = {
  type: 'object',
  properties: {
    eventId: {
      type: 'string',
      format: 'uuid',
      description: 'Unique event identifier'
    },
    eventType: {
      type: 'string',
      description: 'Type of the event (e.g., user.registered)'
    },
    eventVersion: {
      type: 'string',
      pattern: '^\\d+\\.\\d+$',
```

```
      default: '1.0',
      description: 'Event schema version'
    },
    timestamp: {
      type: 'string',
      format: 'date-time',
      description: 'When the event occurred'
    },
    source: {
      type: 'string',
      description: 'Service that produced the event'
    },
    correlationId: {
      type: 'string',
      format: 'uuid',
      description: 'ID to track related events'
    },
    causationId: {
      type: 'string',
      format: 'uuid',
      description: 'ID of the event that caused this event'
    },
    payload: {
      type: 'object',
      description: 'Actual event data'
    }
  },
  required: ['eventId', 'eventType', 'timestamp', 'source', 'payload']
};

module.exports = { BASE_EVENT_SCHEMA };
```

📊 Schema Evolution và Backward Compatibility

Version Migration Handler

```
// filepath: /shared/validation/schema-migration.js
class SchemaMigration {
  static migrateEvent(eventType, payload, fromVersion, toVersion) {
    const migrationKey = `${eventType}:${fromVersion}->${toVersion}`;

    const migrations = {
      'user.registered:1.0->1.1': (payload) => ({
        ...payload,
        // Add new field với default value
        phoneNumber: payload.phoneNumber || null,
        // Rename field
        userName: payload.name,
        name: undefined
      }),
```

```
        'order.created:1.0->2.0': (payload) => ({
          ...payload,
          // Restructure nested objects
          customer: {
            userId: payload.userId,
            email: payload.customerEmail
          },
          userId: undefined,
          customerEmail: undefined
        })
      };

      const migrationFn = migrations[migrationKey];
      if (!migrationFn) {
        throw new Error(`No migration found for ${migrationKey}`);
      }

      return migrationFn(payload);
    }
  }

  module.exports = SchemaMigration;
```

✅ Lợi Ích của Schema-Driven Events

1. **Type Safety & Validation**

- Đảm bảo dữ liệu đúng format trước khi publish/consume
- Catch lỗi sớm trong development cycle
- Prevent runtime errors do invalid data

2. **Documentation tự động**

- Schema = Living Documentation
- IDE autocomplete và type hints
- API documentation generation

3. **Loose Coupling**

- Services chỉ cần biết event schema, không cần biết implementation
- Schema evolution không break existing consumers
- Contract-based development

4. **Quality Assurance**

- Automated contract testing
- Schema compatibility testing
- Event replay với validation

5. **Monitoring & Debugging**

- Schema violation alerts
- Event payload inspection
- Data quality metrics

🧪 Testing Event Schemas

Schema Contract Tests

```
// filepath: /tests/schemas/event-schema.test.js
const { EventValidator } = require('../../shared/validation/event-
validator');

describe('Event Schema Validation', () => {
  describe('User Events', () => {
    test('should validate valid user.registered event', () => {
      const validPayload = {
        userId: '123e4567-e89b-12d3-a456-426614174000',
        email: 'test@example.com',
        name: 'John Doe',
        timestamp: '2023-08-18T10:30:00Z'
      };

      expect(() => {
        EventValidator.validate('user.registered', validPayload);
      }).not.toThrow();
    });

    test('should reject user.registered with invalid email', () => {
      const invalidPayload = {
        userId: '123e4567-e89b-12d3-a456-426614174000',
        email: 'invalid-email',
        name: 'John Doe',
        timestamp: '2023-08-18T10:30:00Z'
      };

      expect(() => {
        EventValidator.validate('user.registered', invalidPayload);
      }).toThrow('Event validation failed');
    });
  });
});
```

Event schemas trong dự án này đảm bảo tính nhất quán, type safety và loose coupling - là nền tảng quan trọng cho Event-Driven Architecture thành công! 🚀