ZenWave 360º

Event-Driven Spring Modulith and AsyncAPI

Event-Driven Architectures

without the complexity

@ivangsa
@ZenWave360

Ivan Garcia Sainz-Aja
@ivangsa

Open Source Contributor

AsyncAPI
Ambassador

ZenWave 360

About Me

Programming with Springframework since 2004

spring®

HIBERNATE

AppFuse

Complex Systems from Simple Components

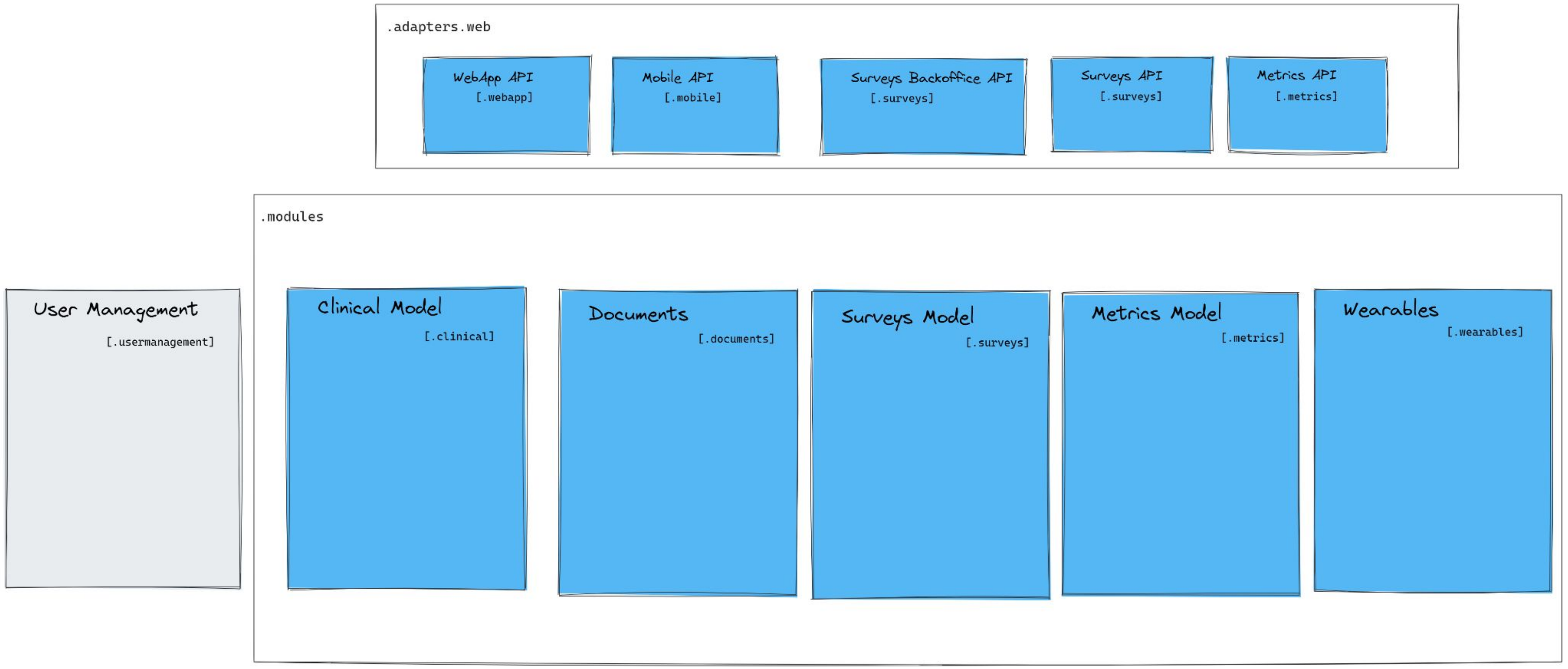
Complex Systems from Simple Components

- ✓ Software Design: Evolutionary, Separation of Concerns
- ✓ Performant, Non-Blocking, Eventual
- ✓ Reliable and Consistent

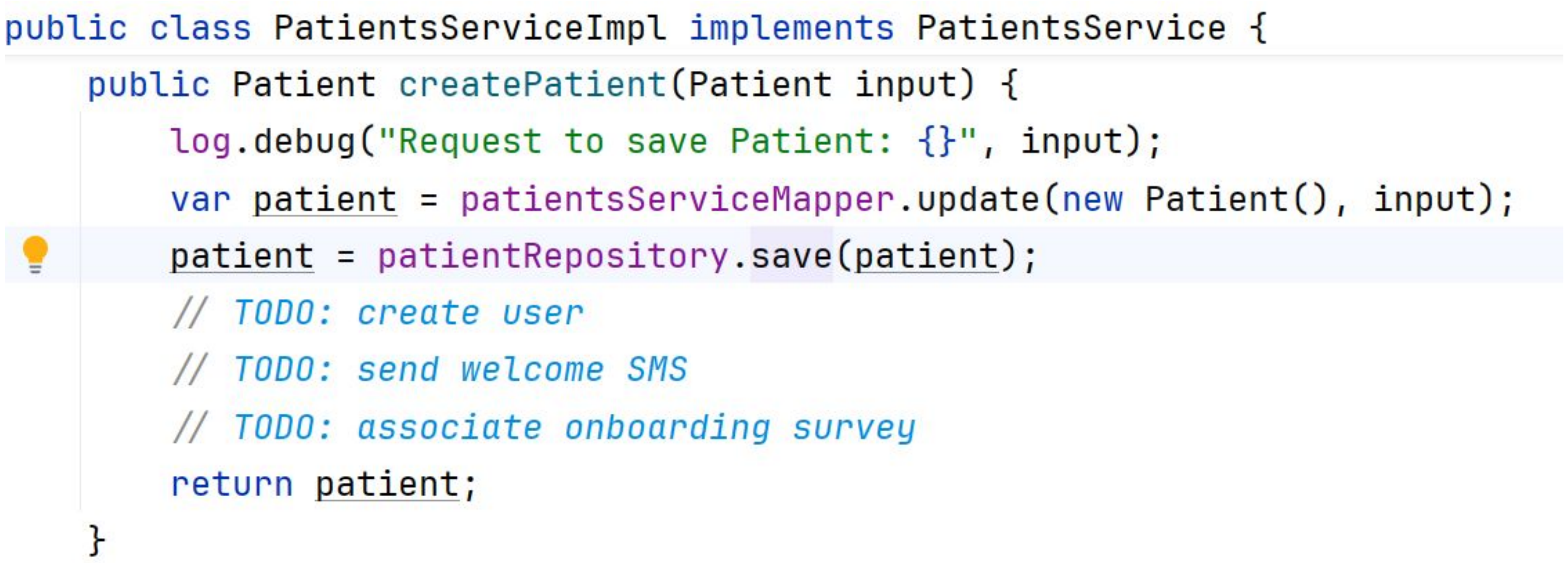
Complex Systems from Simple Components

- ✓ Software Design: Evolutionary, Separation of Concerns
- ✓ Performant, Non-Blocking, Eventual
- ✓ Reliable and Consistent

Clinical Tool - Use Case



Decoupling Modules with Events



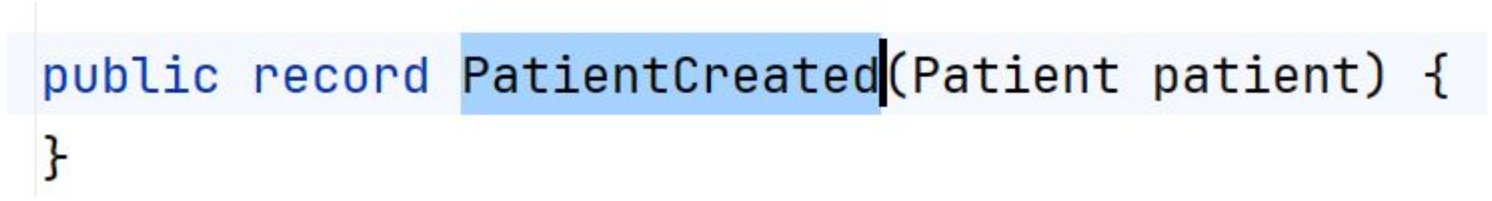
```java
public class PatientsServiceImpl implements PatientsService {
    public Patient createPatient(Patient input) {
        log.debug("Request to save Patient: {}", input);
        var patient = patientsServiceMapper.update(new Patient(), input);
        patient = patientRepository.save(patient);
        // TODO: create user
        // TODO: send welcome SMS
        // TODO: associate onboarding survey
        return patient;
    }
```

Complex Systems from Simple Components

- ☒ Software Design: Evolutionary, Separation of Concerns
- ☒ Performant, Non-Blocking, Eventual
- ✓ Reliable and Consistent

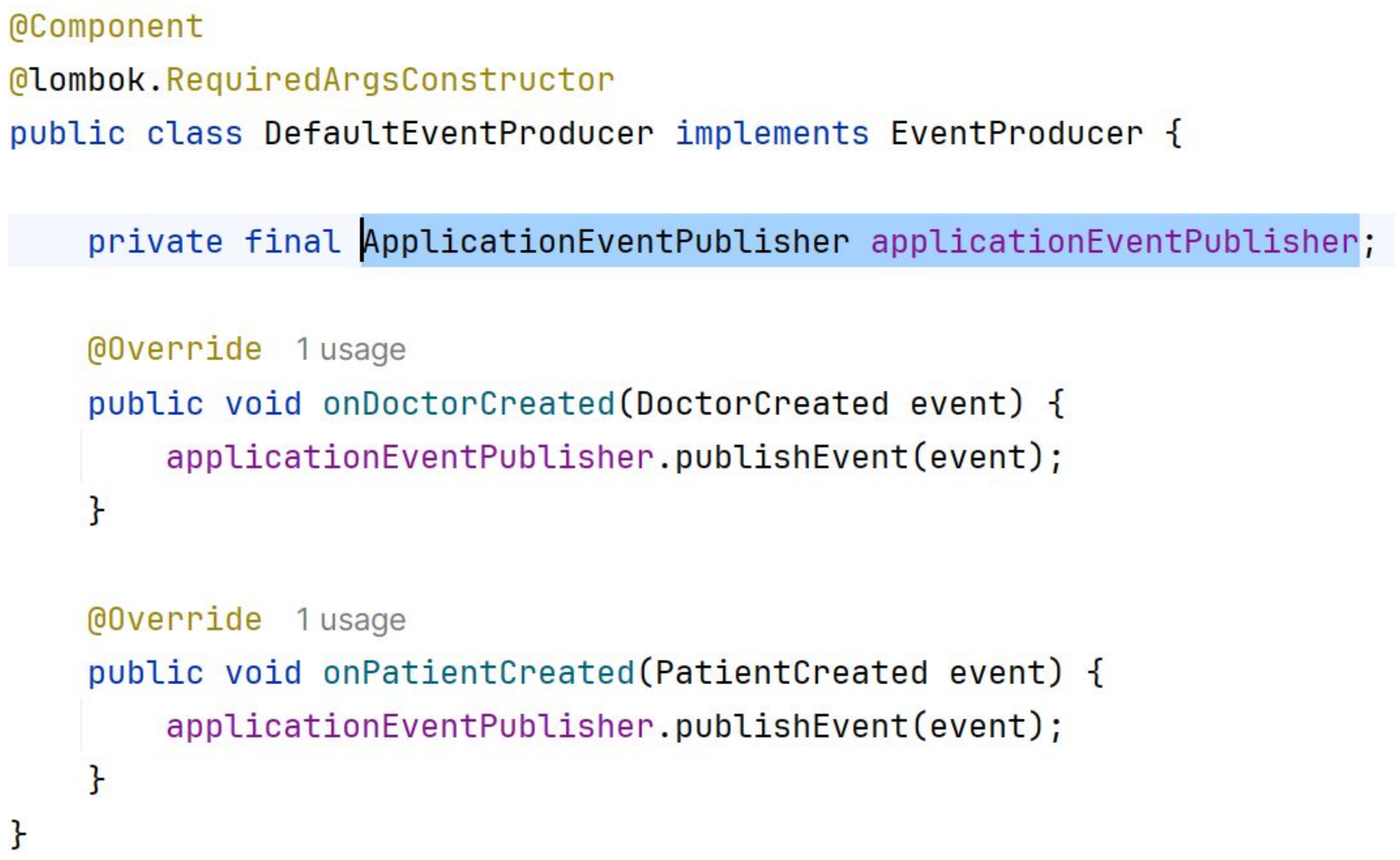
Creating a Domain Event

Creating a Domain Event



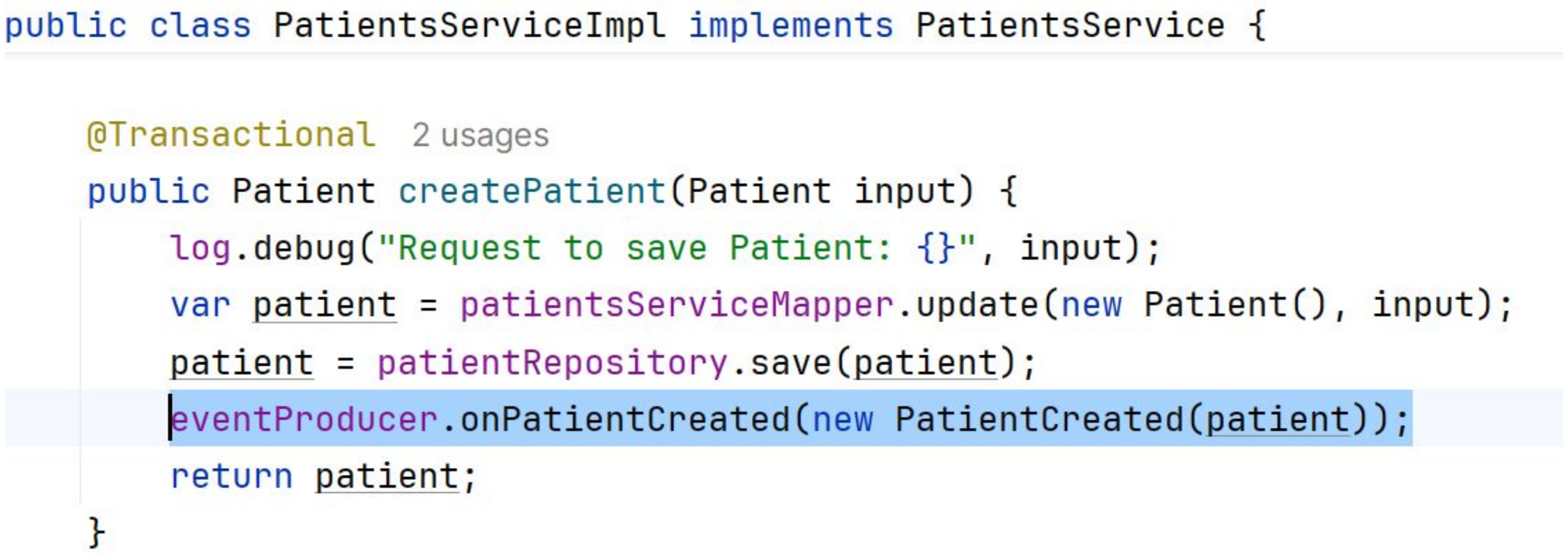
```java
public record PatientCreated(Patient patient) {
}
```

Event Producer using Spring Event-Bus



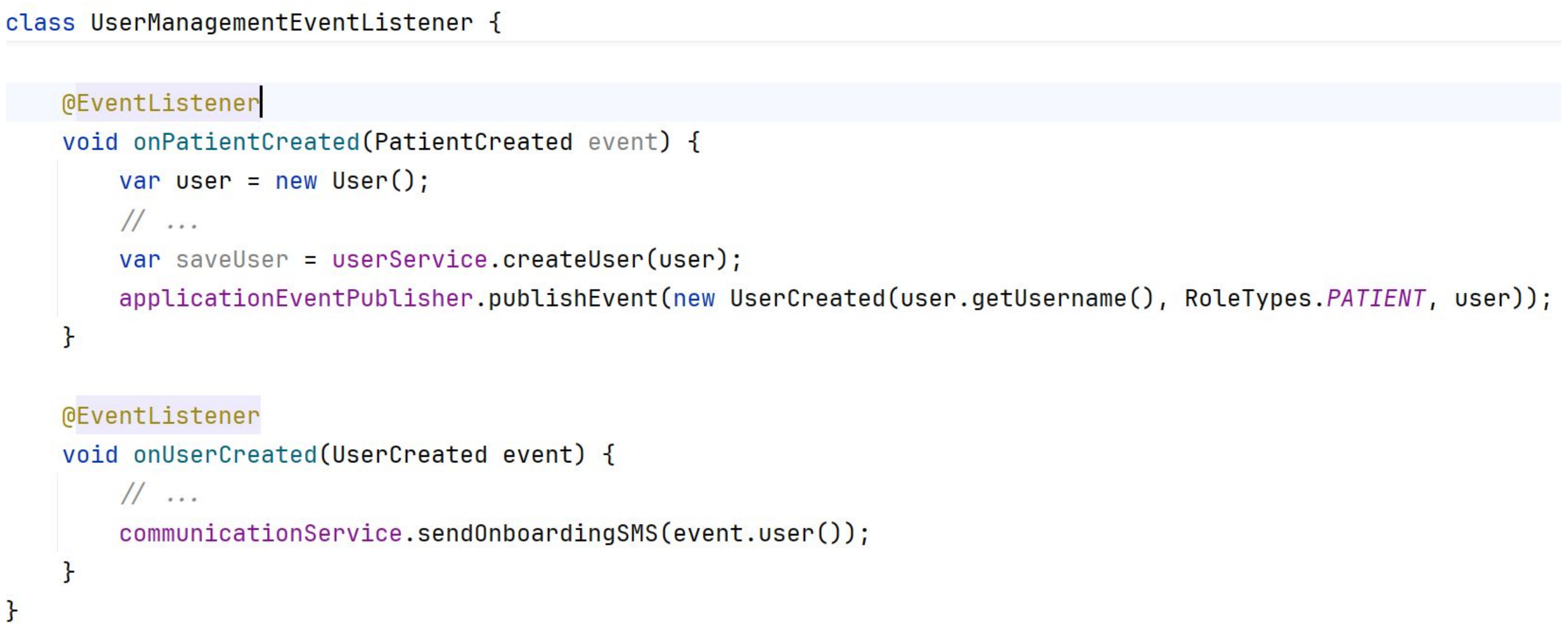
```
@Component
@lombok.RequiredArgsConstructor
public class DefaultEventProducer implements EventProducer {

    private final ApplicationEventPublisher applicationEventPublisher;

    @Override  1 usage
    public void onDoctorCreated(DoctorCreated event) {
        applicationEventPublisher.publishEvent(event);
    }

    @Override  1 usage
    public void onPatientCreated(PatientCreated event) {
        applicationEventPublisher.publishEvent(event);
    }
}
```

Decoupling Modules with Events



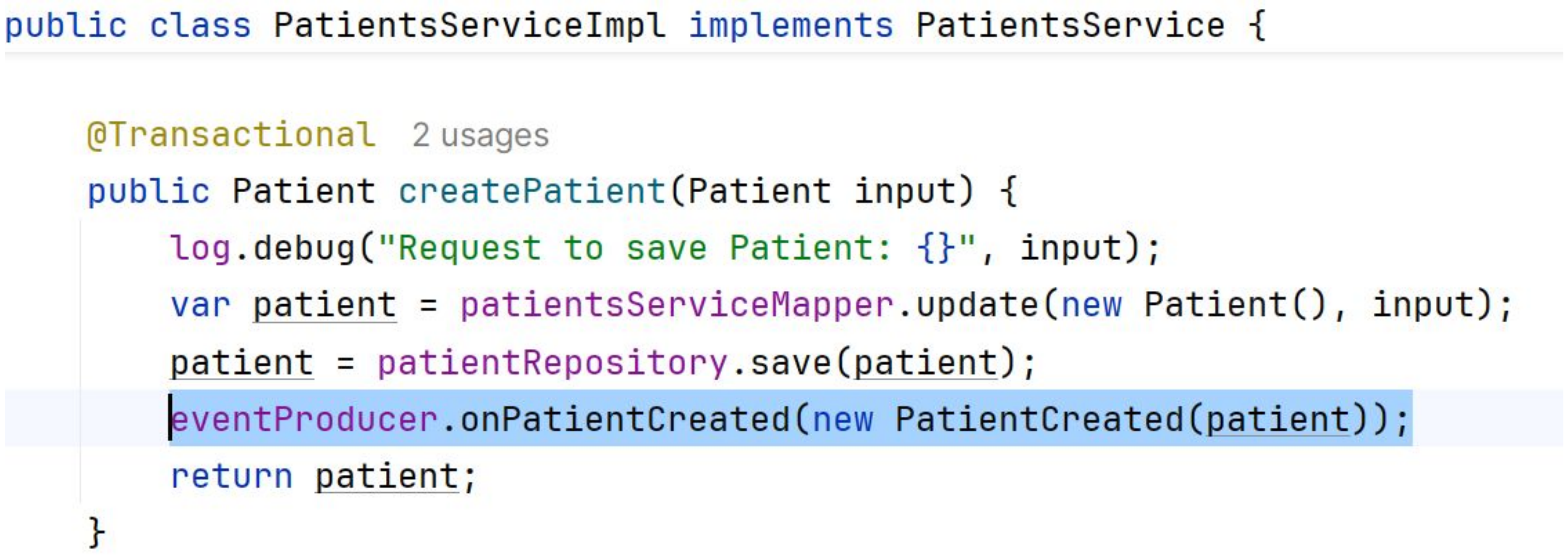
```java
public class PatientsServiceImpl implements PatientsService {

    @Transactional  2 usages
    public Patient createPatient(Patient input) {
        log.debug("Request to save Patient: {}", input);
        var patient = patientsServiceMapper.update(new Patient(), input);
        patient = patientRepository.save(patient);
        eventProducer.onPatientCreated(new PatientCreated(patient));
        return patient;
    }
```

Spring Event Listener



```java
class UserManagementEventListener {

    @EventListener
    void onPatientCreated(PatientCreated event) {
        var user = new User();
        // ...
        var saveUser = userService.createUser(user);
        applicationEventPublisher.publishEvent(new UserCreated(user.getUsername(), RoleTypes.PATIENT, user));
    }

    @EventListener
    void onUserCreated(UserCreated event) {
        // ...
        communicationService.sendOnboardingSMS(event.user());
    }
}
```

Decoupling Modules with Events



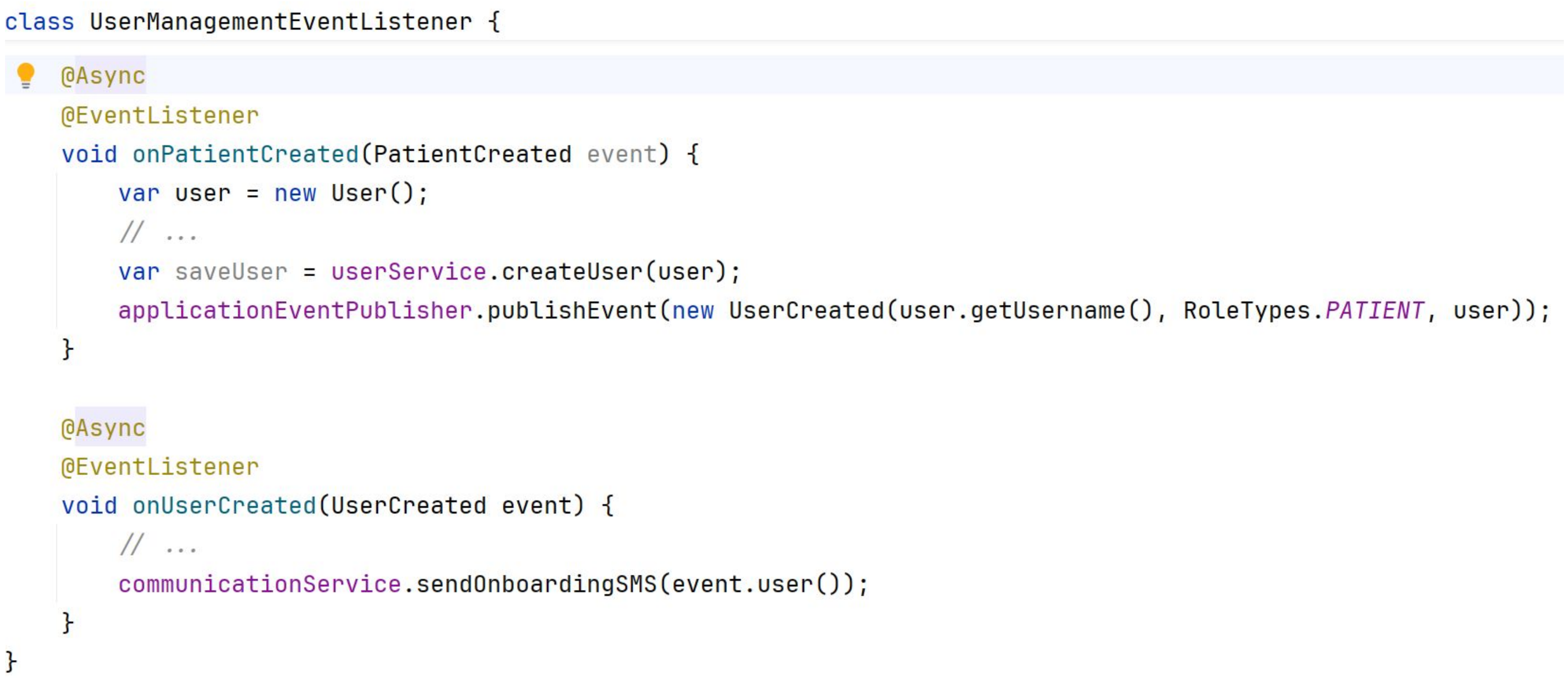
```
public class PatientsServiceImpl implements PatientsService {

    @Transactional  2 usages
    public Patient createPatient(Patient input) {
        log.debug("Request to save Patient: {}", input);
        var patient = patientsServiceMapper.update(new Patient(), input);
        patient = patientRepository.save(patient);
        eventProducer.onPatientCreated(new PatientCreated(patient));
        return patient;
    }
```

Complex Systems from Simple Components

- ✓ Software Design: Evolutionary, Separation of Concerns
- ✗ Performant, Non-Blocking, Eventual
- ✓ Reliable and Consistent

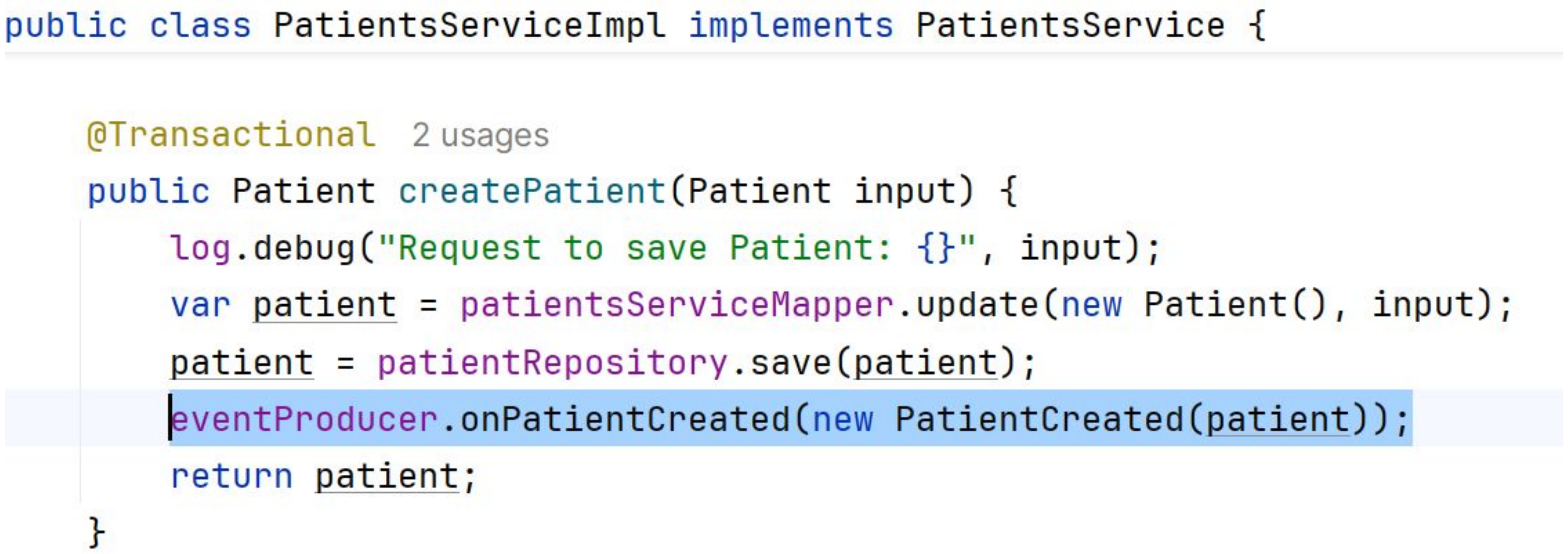
Async Event Handlers

Spring Event Listener



```java
class UserManagementEventListener {

    @Async
    @EventListener
    void onPatientCreated(PatientCreated event) {
        var user = new User();
        // ...
        var saveUser = userService.createUser(user);
        applicationEventPublisher.publishEvent(new UserCreated(user.getUsername(), RoleTypes.PATIENT, user));
    }

    @Async
    @EventListener
    void onUserCreated(UserCreated event) {
        // ...
        communicationService.sendOnboardingSMS(event.user());
    }
}
```

Decoupling Modules with Events



```java
public class PatientsServiceImpl implements PatientsService {

    @Transactional  2 usages
    public Patient createPatient(Patient input) {
        log.debug("Request to save Patient: {}", input);
        var patient = patientsServiceMapper.update(new Patient(), input);
        patient = patientRepository.save(patient);
        eventProducer.onPatientCreated(new PatientCreated(patient));
        return patient;
    }
```

Complex Systems from Simple Components

✓ Software Design: Evolutionary, Separation of Concerns

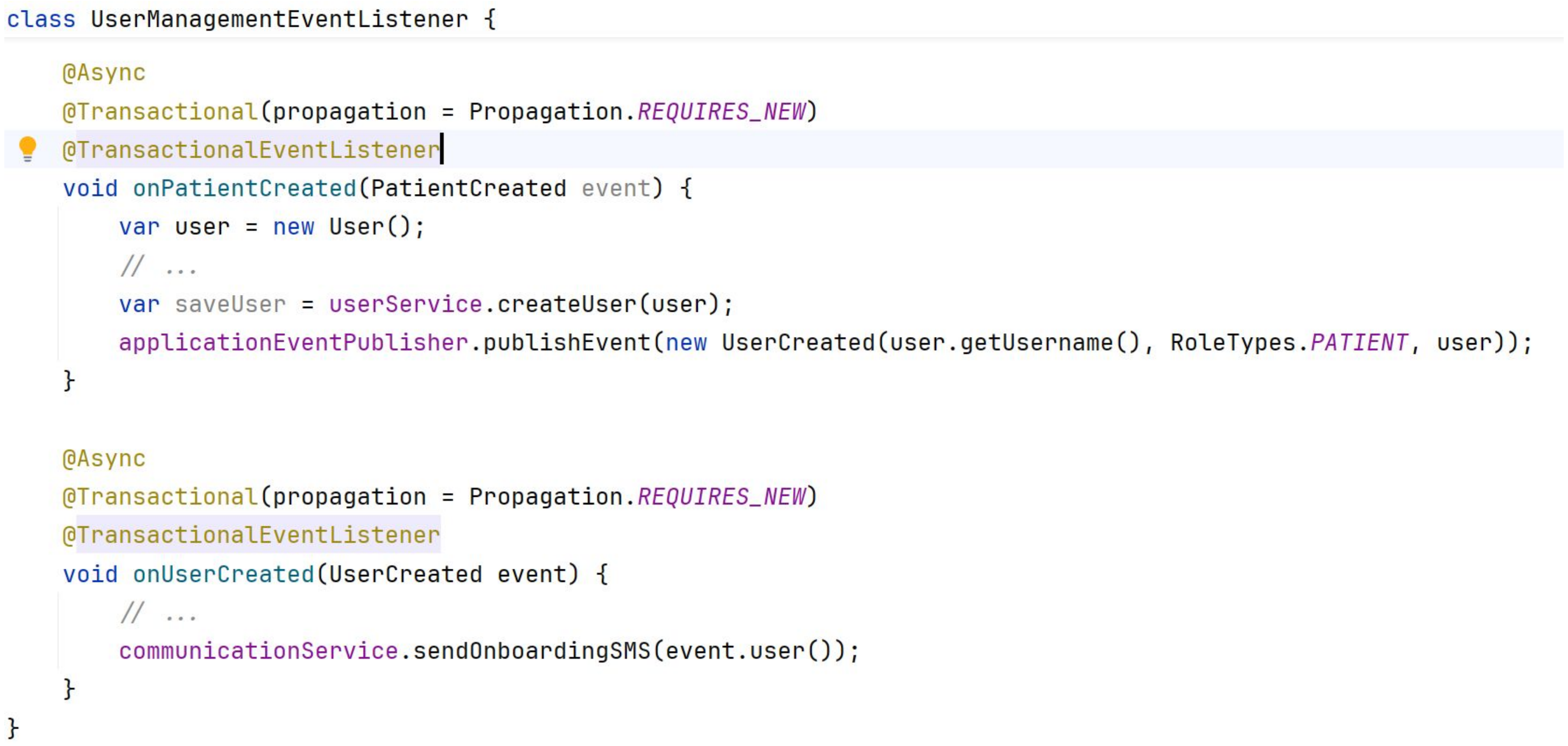
✓ Performant, Non-Blocking, Eventual

✗ Reliable and Consistent

- Handlers do not participate in parent transaction
- Handlers are triggered before parent Tx commit/rollback

Transactional Event Listeners

Spring Event Listener



```java
class UserManagementEventListener {

    @Async
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @TransactionalEventListener
    void onPatientCreated(PatientCreated event) {
        var user = new User();
        // ...
        var saveUser = userService.createUser(user);
        applicationEventPublisher.publishEvent(new UserCreated(user.getUsername(), RoleTypes.PATIENT, user));
    }

    @Async
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @TransactionalEventListener
    void onUserCreated(UserCreated event) {
        // ...
        communicationService.sendOnboardingSMS(event.user());
    }
}
```

Complex Systems from Simple Components

✓ Software Design: Evolutionary, Separation of Concerns

✓ Performant, Non-Blocking, Eventual

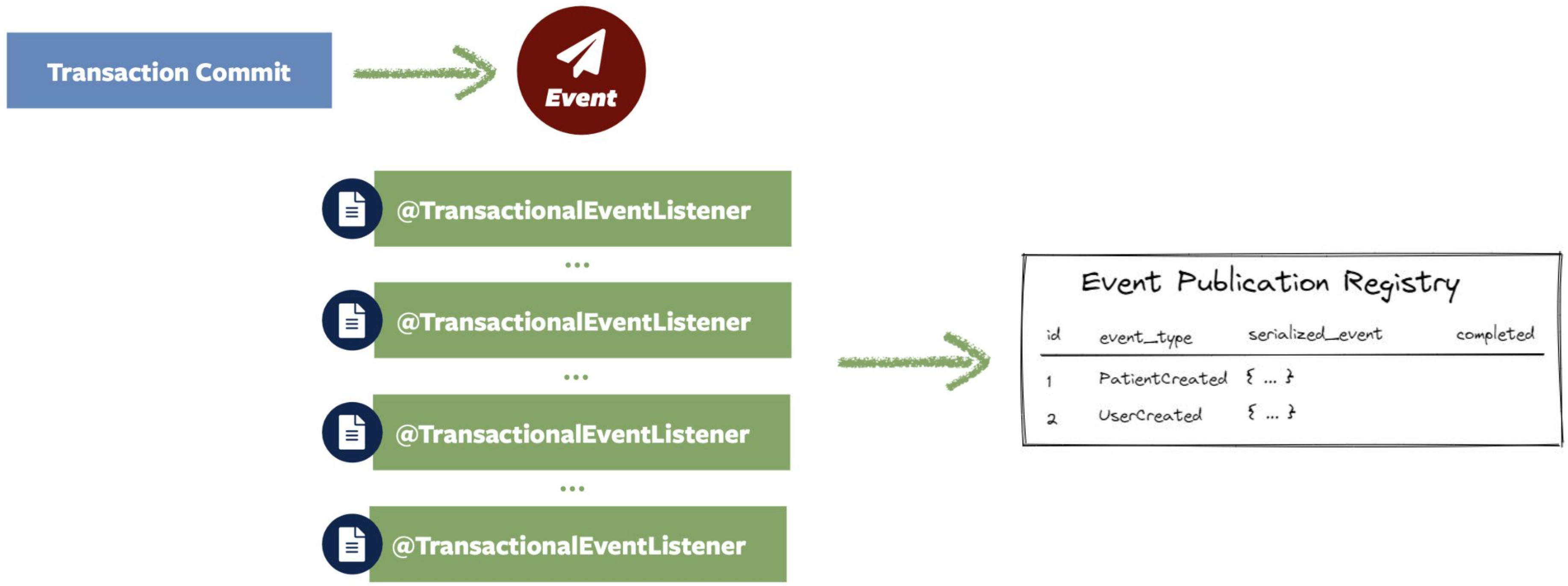
✗ Reliable and Consistent

- Handlers do not participate in parent transaction
- ~~*Handlers are triggered before parent Tx commit/rollback*~~

Spring Modulith:

Event Publication Registry

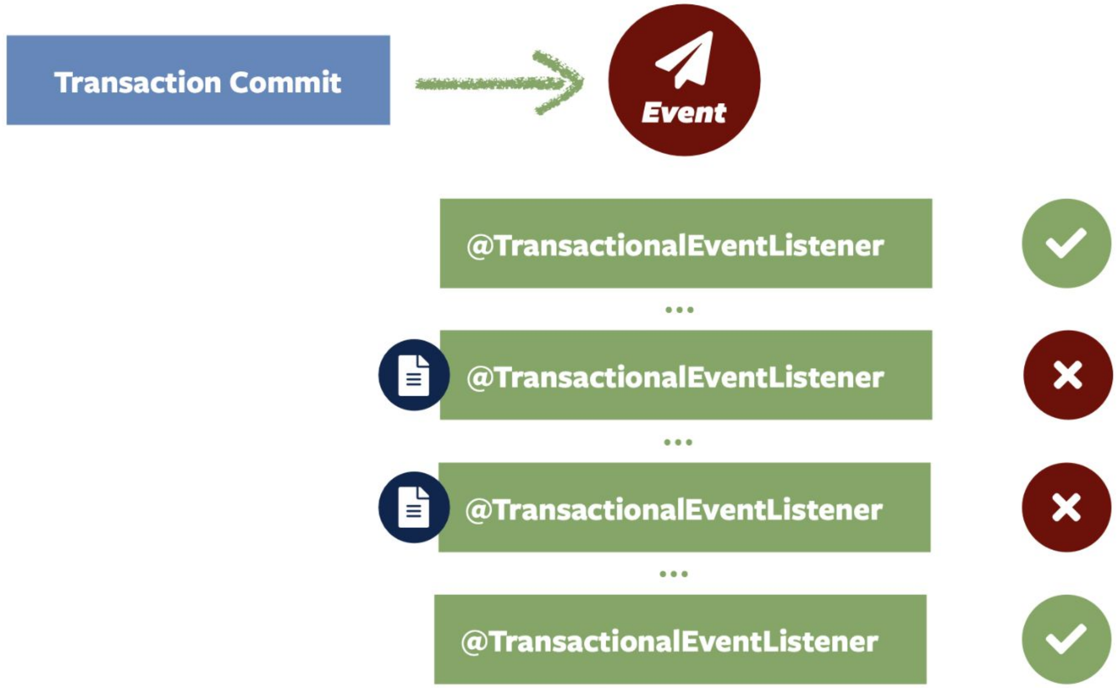
Event-Driven Spring Modulith

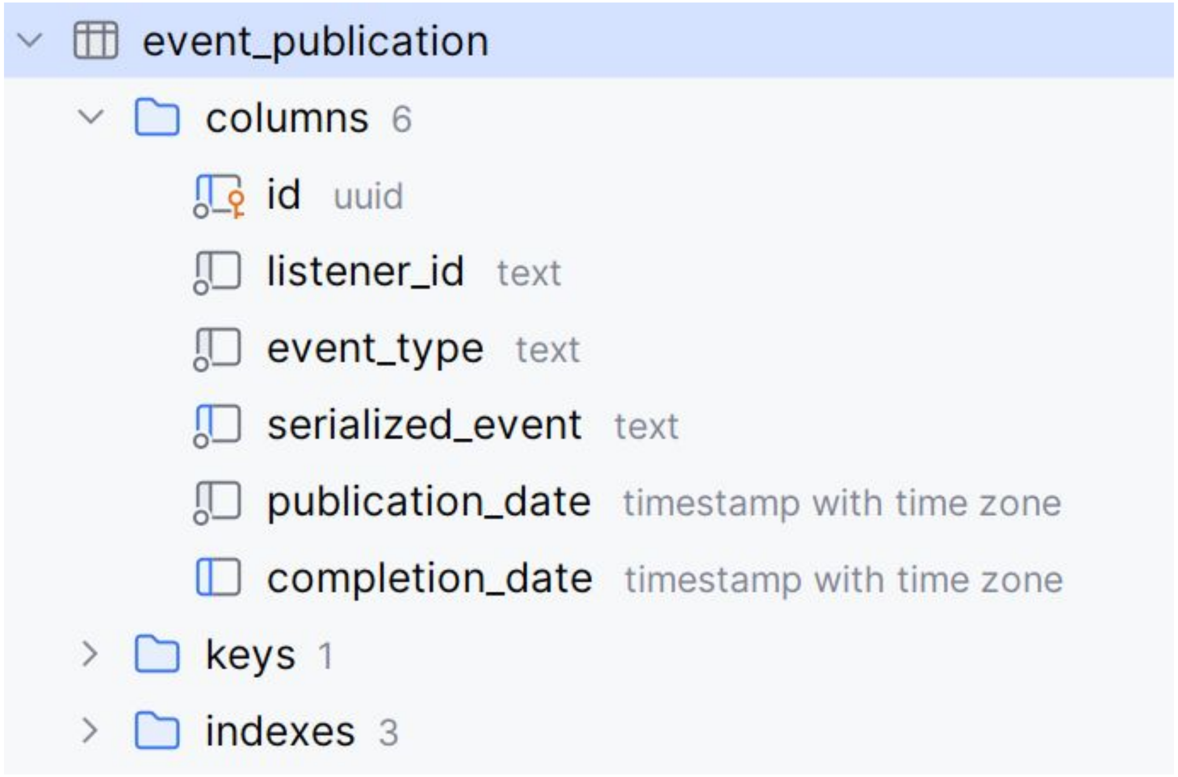

Transaction Commit
Event
@TransactionalEventListener
...
@TransactionalEventListener
...
@TransactionalEventListener
...
@TransactionalEventListener

Event Publication Registry

id	event_type	serialized_event	completed
1	PatientCreated	{ ... }	
2	UserCreated	{ ... }	

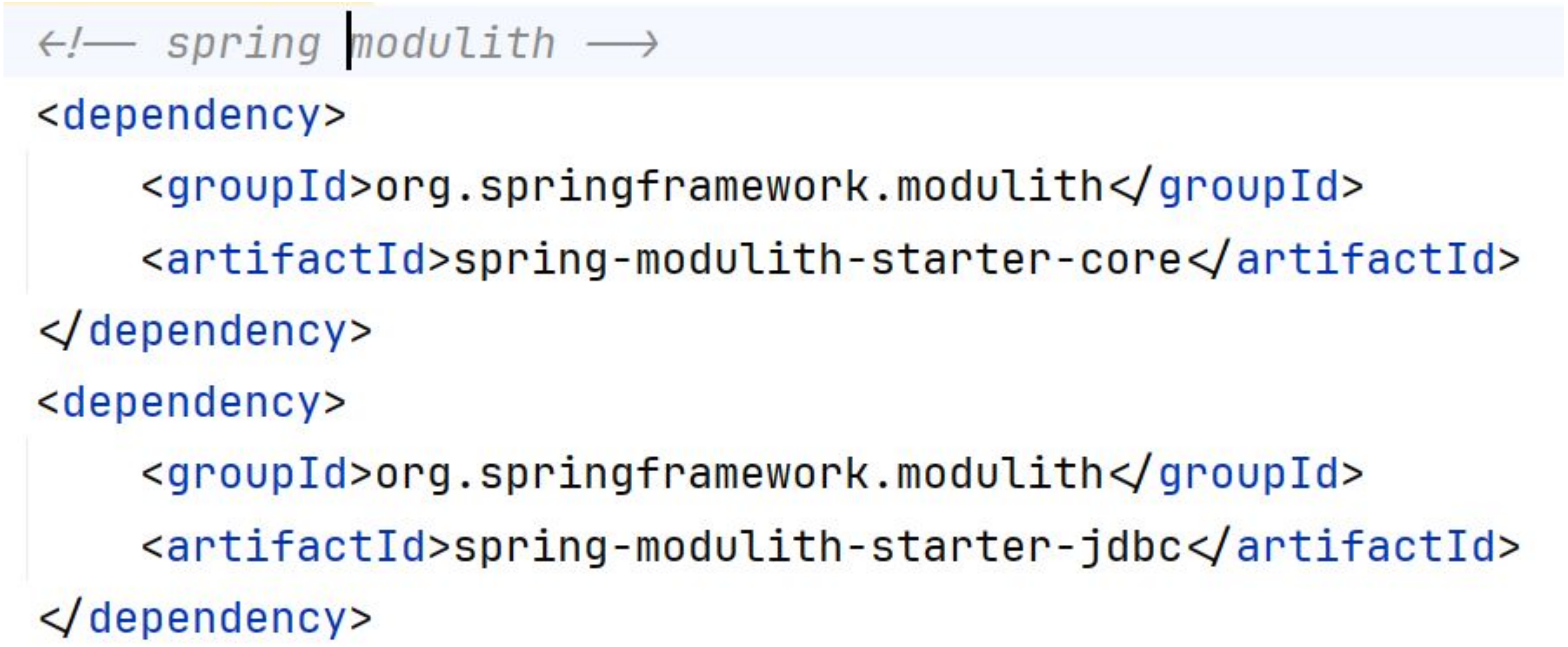
Event-Driven Spring Modulith


Transaction Commit
Event
@TransactionalEventListener
...
@TransactionalEventListener
...
@TransactionalEventListener
...
@TransactionalEventListener



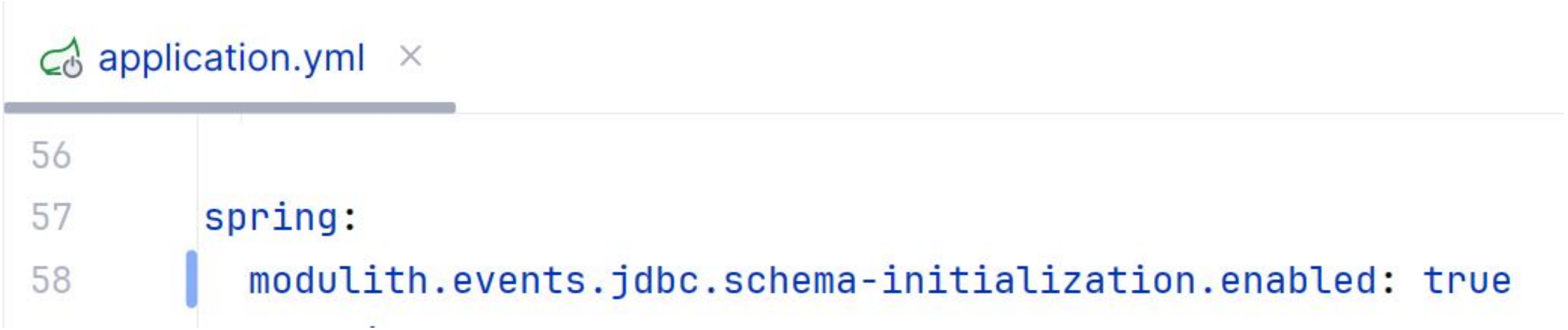
event_publication
columns 6
id uuid
listener_id text
event_type text
serialized_event text
publication_date timestamp with time zone
completion_date timestamp with time zone
keys 1
indexes 3

Adding Spring Modulith Events



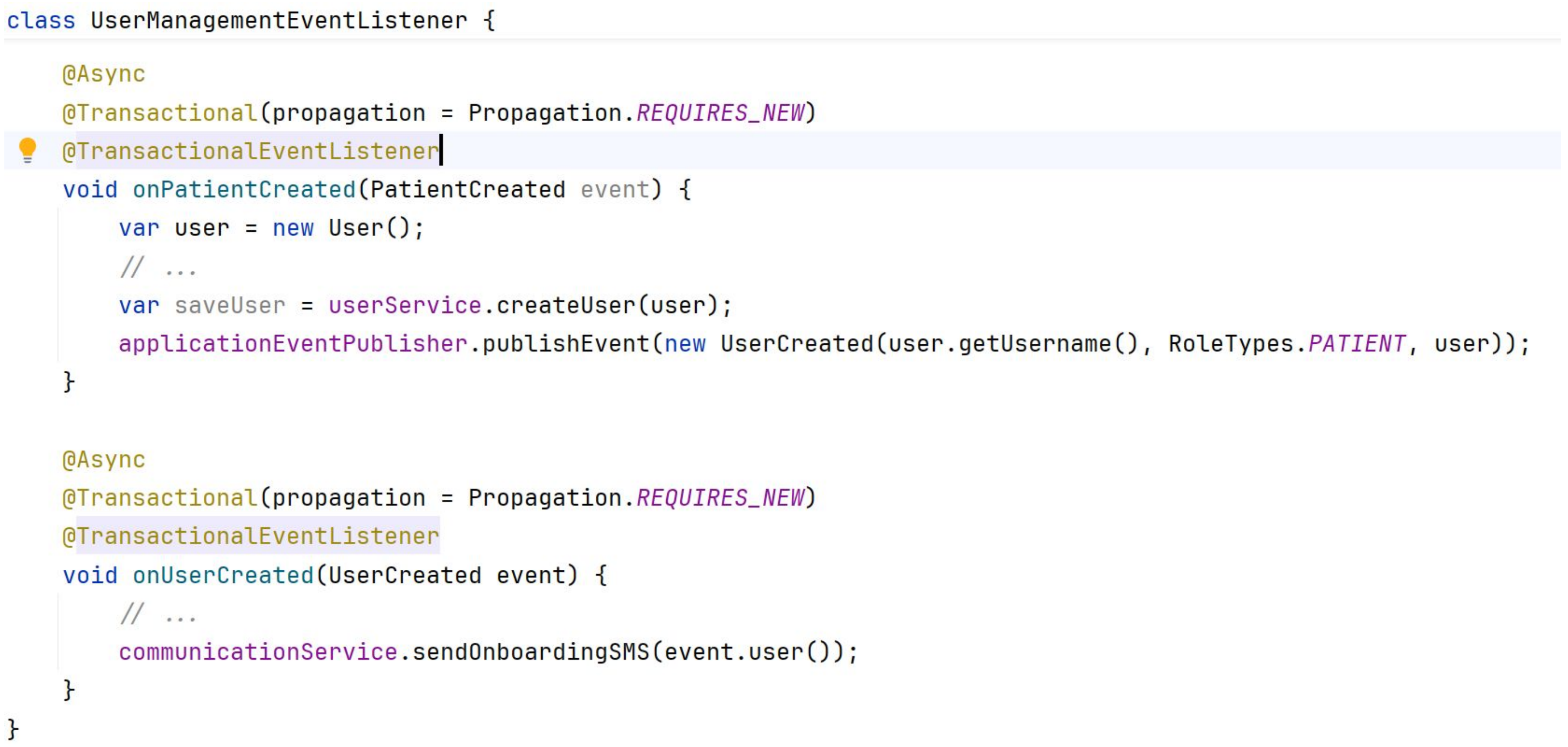
```xml
<!-- spring modulith -->
<dependency>
    <groupId>org.springframework.modulith</groupId>
    <artifactId>spring-modulith-starter-core</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.modulith</groupId>
    <artifactId>spring-modulith-starter-jdbc</artifactId>
</dependency>
```

Enabling Event Registry Schema


application.yml

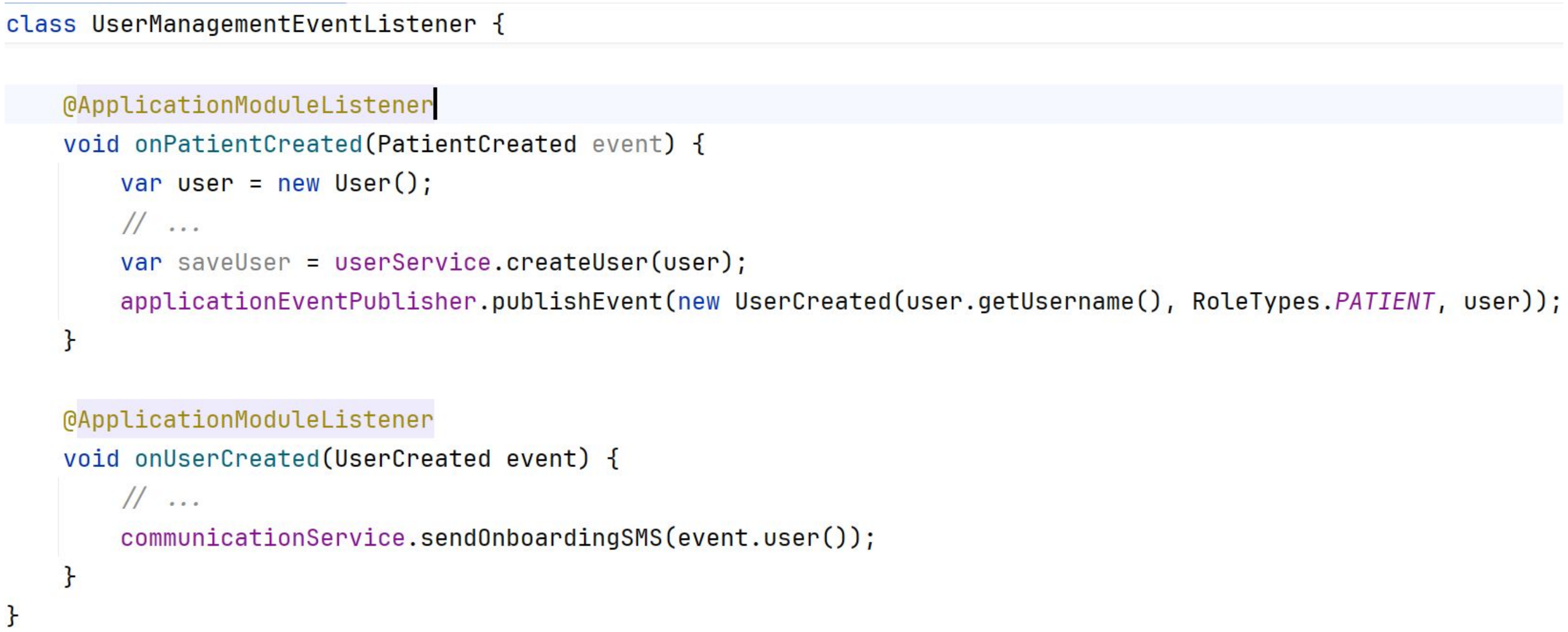
```yaml

spring:
  modulith.events.jdbc.schema-initialization.enabled: true
```

Spring Event Listener



```java
class UserManagementEventListener {

    @Async
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @TransactionalEventListener
    void onPatientCreated(PatientCreated event) {
        var user = new User();
        // ...
        var saveUser = userService.createUser(user);
        applicationEventPublisher.publishEvent(new UserCreated(user.getUsername(), RoleTypes.PATIENT, user));
    }

    @Async
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    @TransactionalEventListener
    void onUserCreated(UserCreated event) {
        // ...
        communicationService.sendOnboardingSMS(event.user());
    }
}
```

Spring Event Listener



```java
class UserManagementEventListener {

    @ApplicationModuleListener
    void onPatientCreated(PatientCreated event) {
        var user = new User();
        // ...
        var saveUser = userService.createUser(user);
        applicationEventPublisher.publishEvent(new UserCreated(user.getUsername(), RoleTypes.PATIENT, user));
    }

    @ApplicationModuleListener
    void onUserCreated(UserCreated event) {
        // ...
        communicationService.sendOnboardingSMS(event.user());
    }
}
```

Complex Systems from Simple Components

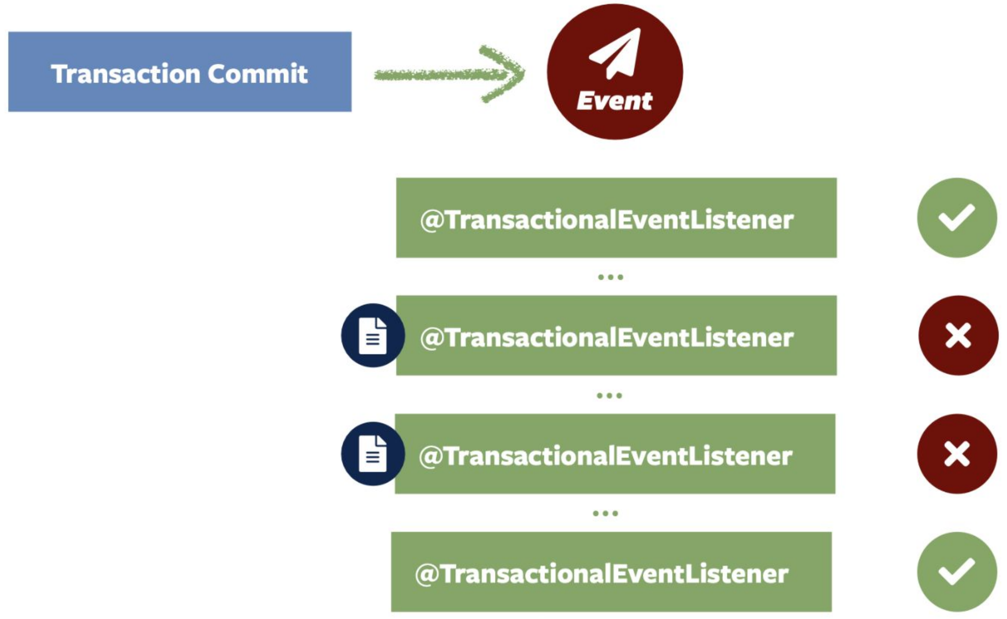
✓ Software Design: Evolutionary, Separation of Concerns

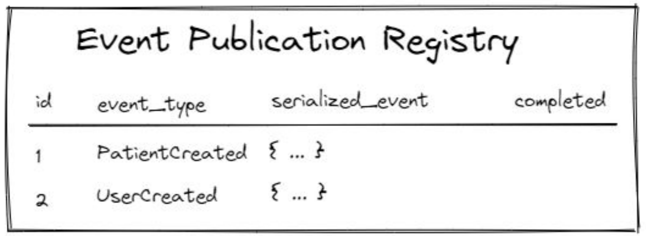
✓ Performant, Non-Blocking, Eventual

✗ Reliable and Consistent

- Handlers do not participate in parent transaction
- ~~*Handlers are triggered before parent Tx commit/rollback*~~

Event-Driven Spring Modulith

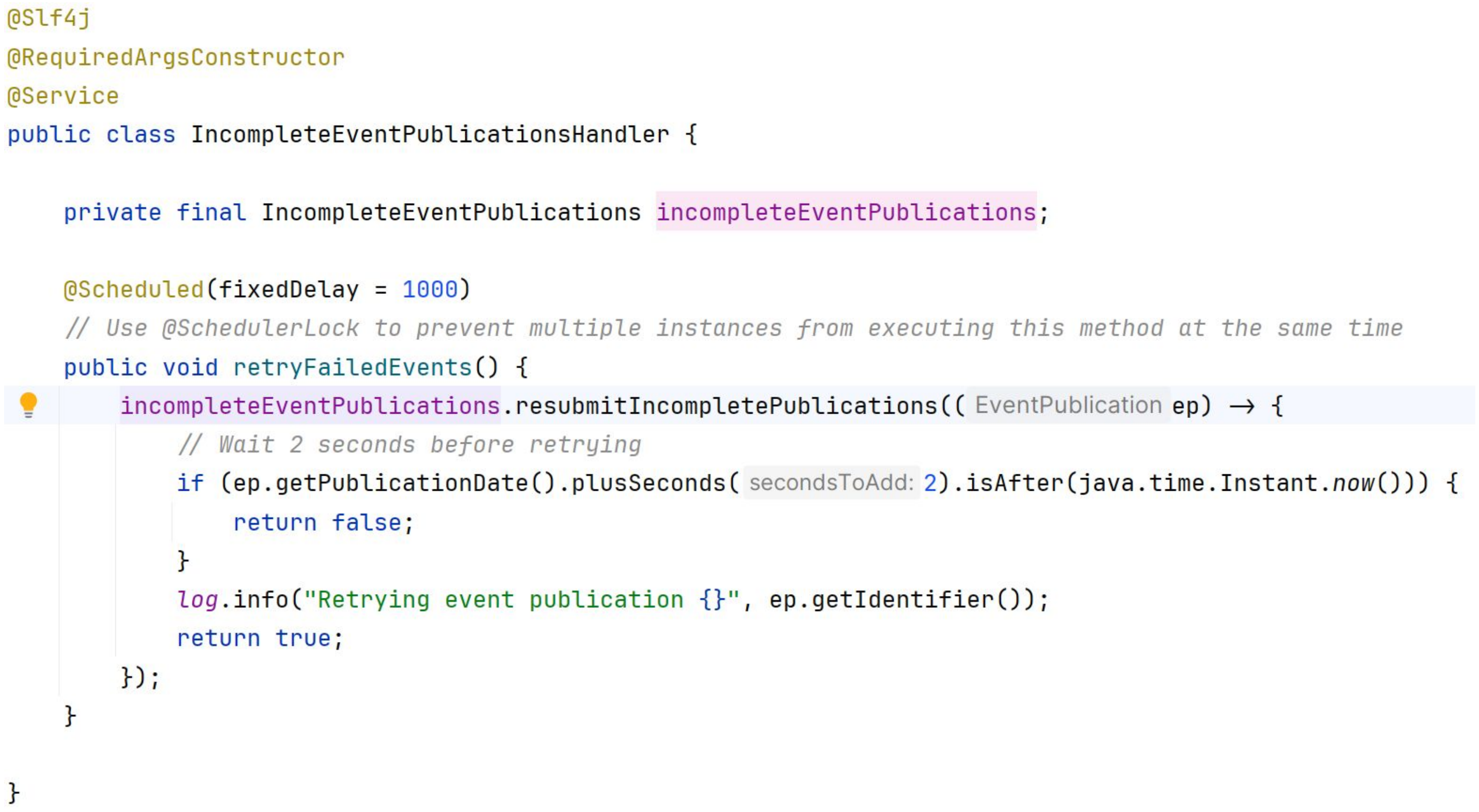

Transaction Commit
Event
@TransactionalEventListener ✓
...
@TransactionalEventListener ✗
...
@TransactionalEventListener ✗
...
@TransactionalEventListener ✓



Event Publication Registry

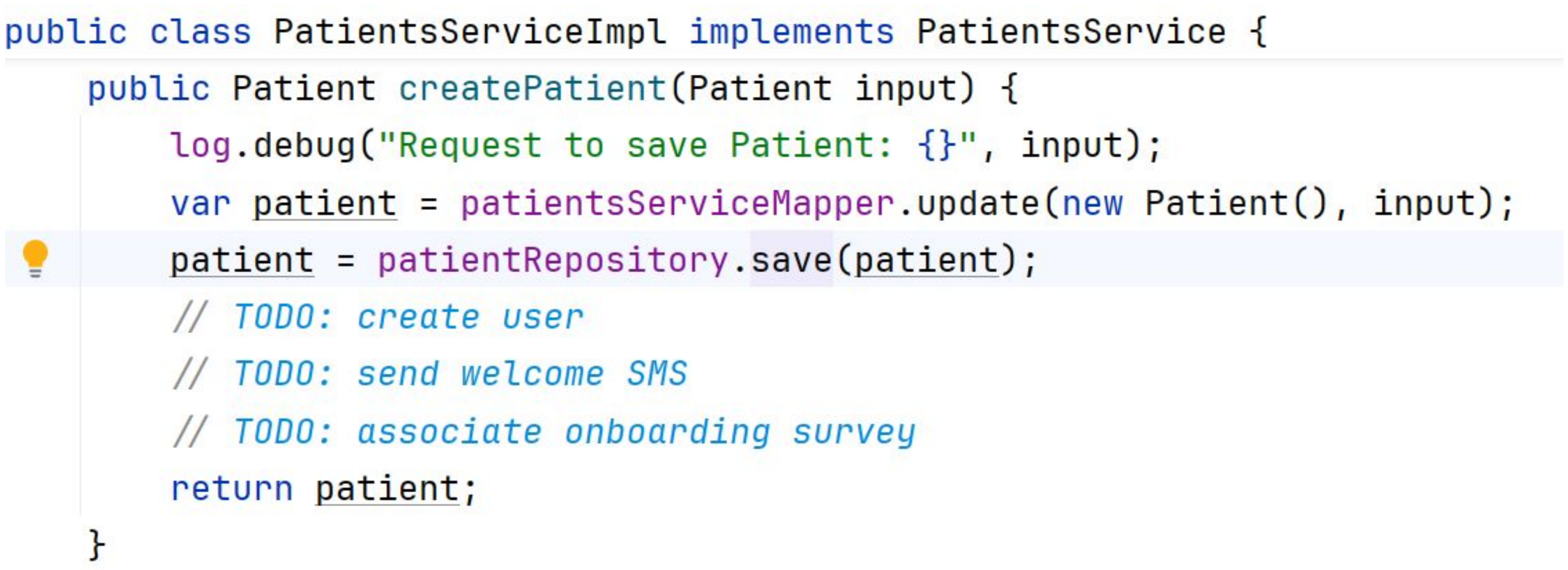
id	event_type	serialized_event	completed
1	PatientCreated	{ ... }	
2	UserCreated	{ ... }	

Event-Driven Spring Modulith



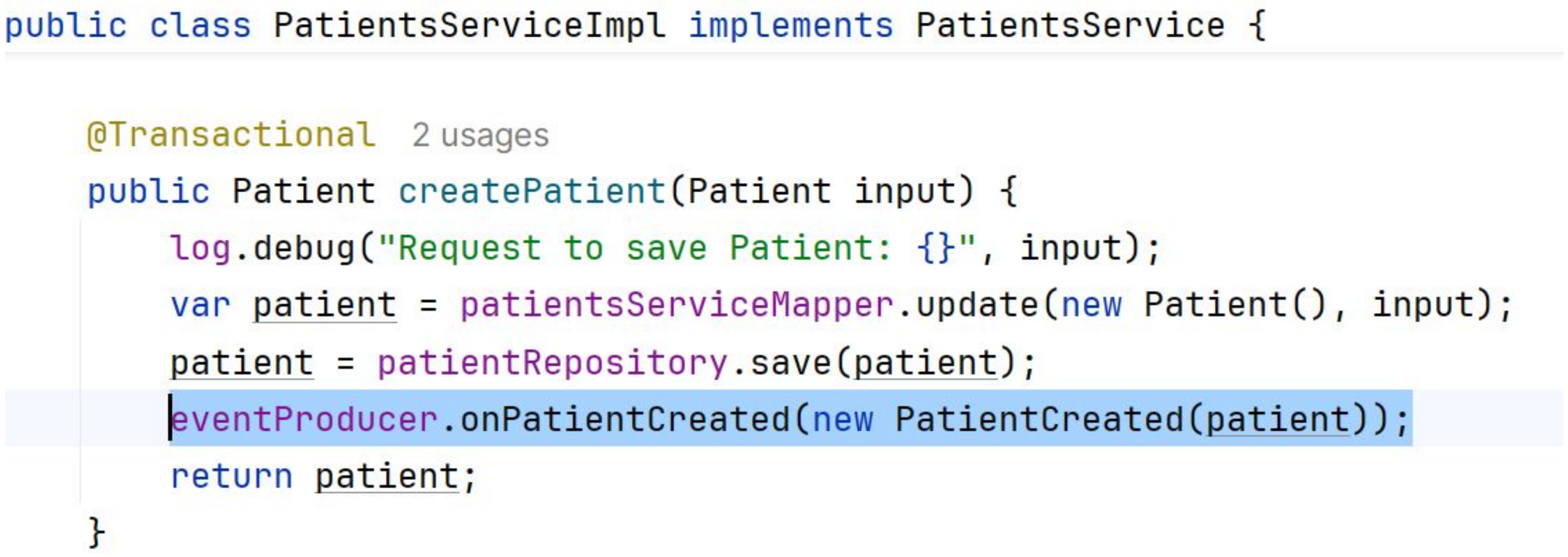
```java
@Slf4j
@RequiredArgsConstructor
@Service
public class IncompleteEventPublicationsHandler {

    private final IncompleteEventPublications incompleteEventPublications;

    @Scheduled(fixedDelay = 1000)
    // Use @SchedulerLock to prevent multiple instances from executing this method at the same time
    public void retryFailedEvents() {
        incompleteEventPublications.resubmitIncompletePublications(( EventPublication ep) -> {
            // Wait 2 seconds before retrying
            if (ep.getPublicationDate().plusSeconds( secondsToAdd: 2).isAfter(java.time.Instant.now())) {
                return false;
            }
            log.info("Retrying event publication {}", ep.getIdentifier());
            return true;
        });
    }

}
```

Decoupling Modules with Events (before)



```java
public class PatientsServiceImpl implements PatientsService {
    public Patient createPatient(Patient input) {
        log.debug("Request to save Patient: {}", input);
        var patient = patientsServiceMapper.update(new Patient(), input);
        patient = patientRepository.save(patient);
        // TODO: create user
        // TODO: send welcome SMS
        // TODO: associate onboarding survey
        return patient;
    }
```

Decoupling Modules with Events (after)



```
public class PatientsServiceImpl implements PatientsService {

    @Transactional  2 usages
    public Patient createPatient(Patient input) {
        log.debug("Request to save Patient: {}", input);
        var patient = patientsServiceMapper.update(new Patient(), input);
        patient = patientRepository.save(patient);
        eventProducer.onPatientCreated(new PatientCreated(patient));
        return patient;
    }
```

Complex Systems from Simple Components

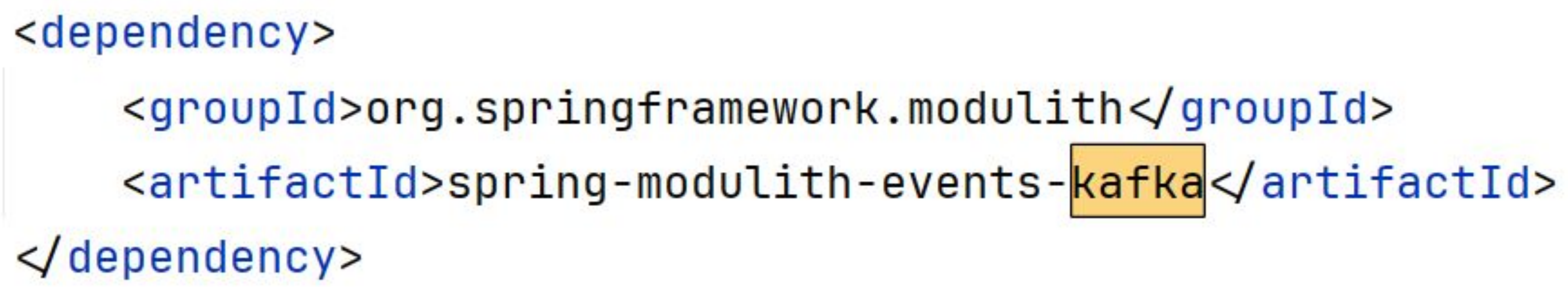
- ✓ Software Design: Evolutionary, Separation of Concerns
- ✓ Performant, Non-Blocking, Eventual
- ✓ Reliable and Consistent

Externalizing Events

Spring Modulith Events Externalizer

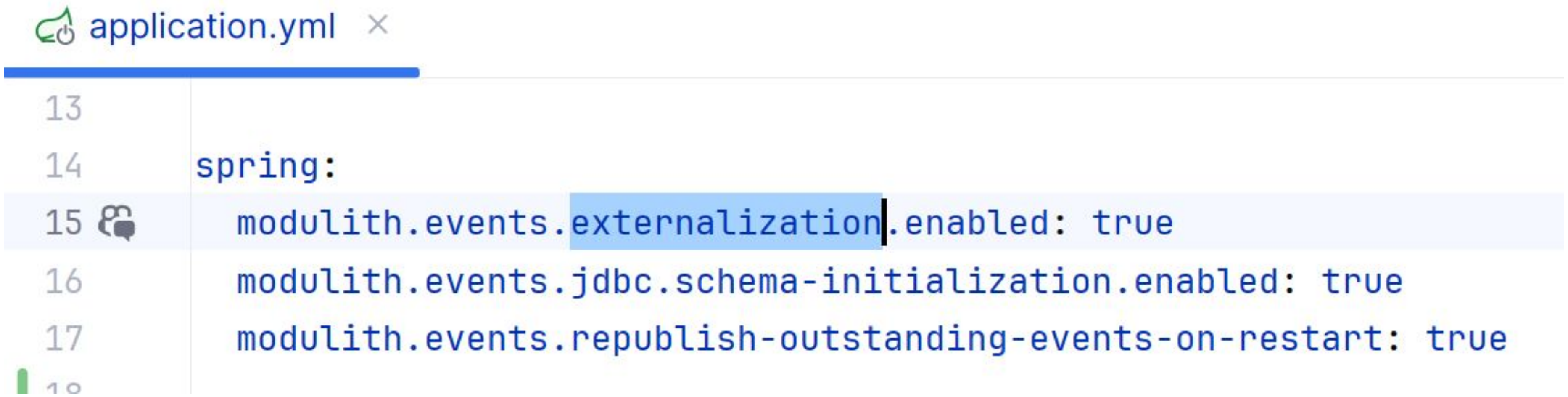
```java
// topic and routing key
@Externalized("patients-topic::#{#this.patient().getId()}")
public record PatientCreated(Patient patient) {
}
```

Spring Modulith Events Externalizer



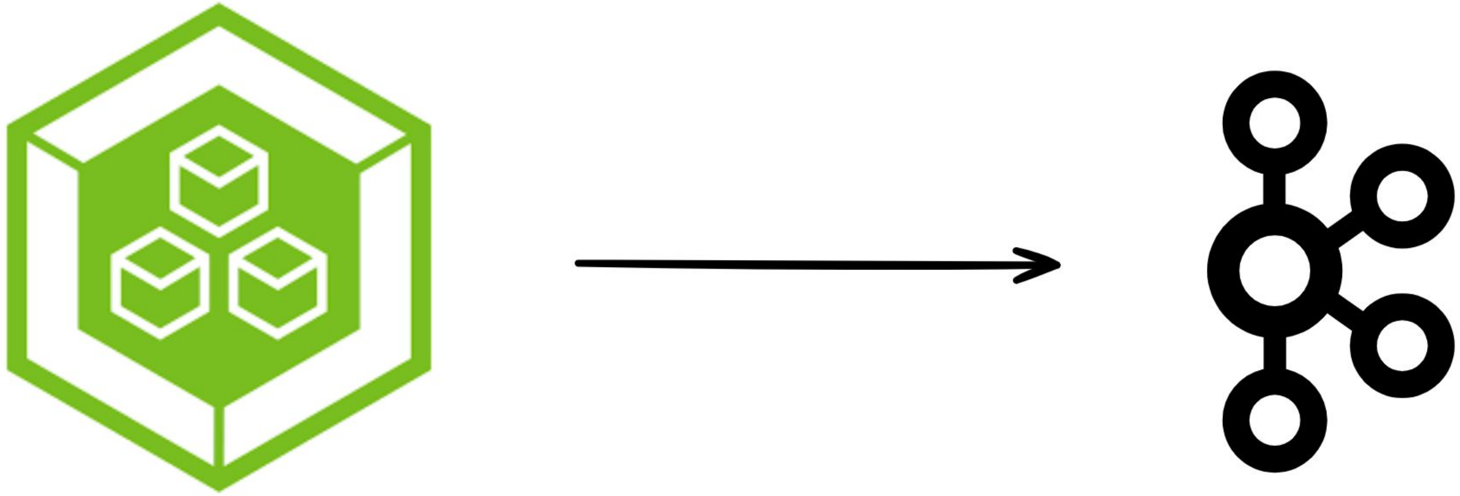
```xml
<dependency>
    <groupId>org.springframework.modulith</groupId>
    <artifactId>spring-modulith-events-kafka</artifactId>
</dependency>
```

Spring Modulith Events Externalizer

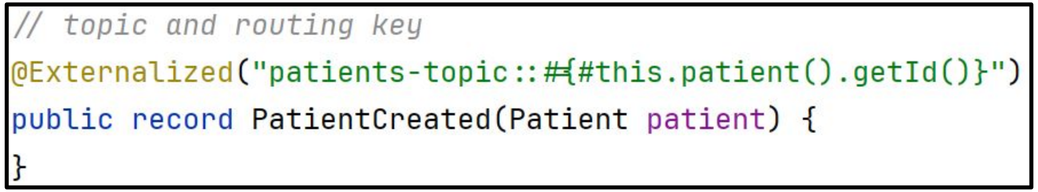

application.yml

```yaml

spring:
  modulith.events.externalization.enabled: true
  modulith.events.jdbc.schema-initialization.enabled: true
  modulith.events.republish-outstanding-events-on-restart: true
```

Spring Modulith Events Externalizer



Spring Modulith Events Externalizer



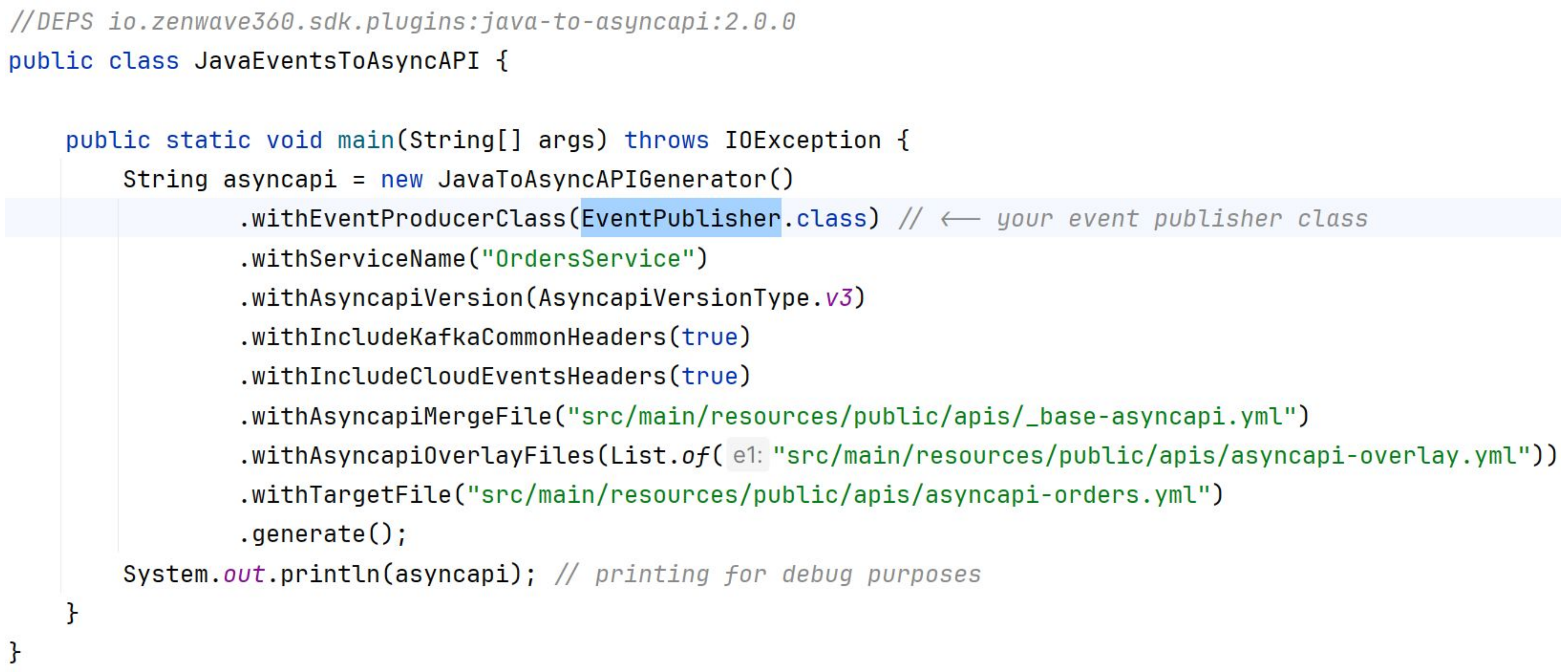
```java
// topic and routing key
@Externalized("patients-topic::#{#this.patient().getId()}")
public record PatientCreated(Patient patient) {
}
```

Convenience ✅	Quick and Convenient
API Documentation❌	No built-in support for formal API documentation
Schema Management❌	No friction to prevent breaking changes in event schemas that could impact consumers
API Governance❌	No standardized way to enforce API design standards: naming conventions, versioning, headers/metadata...

API Management with AsyncAPI

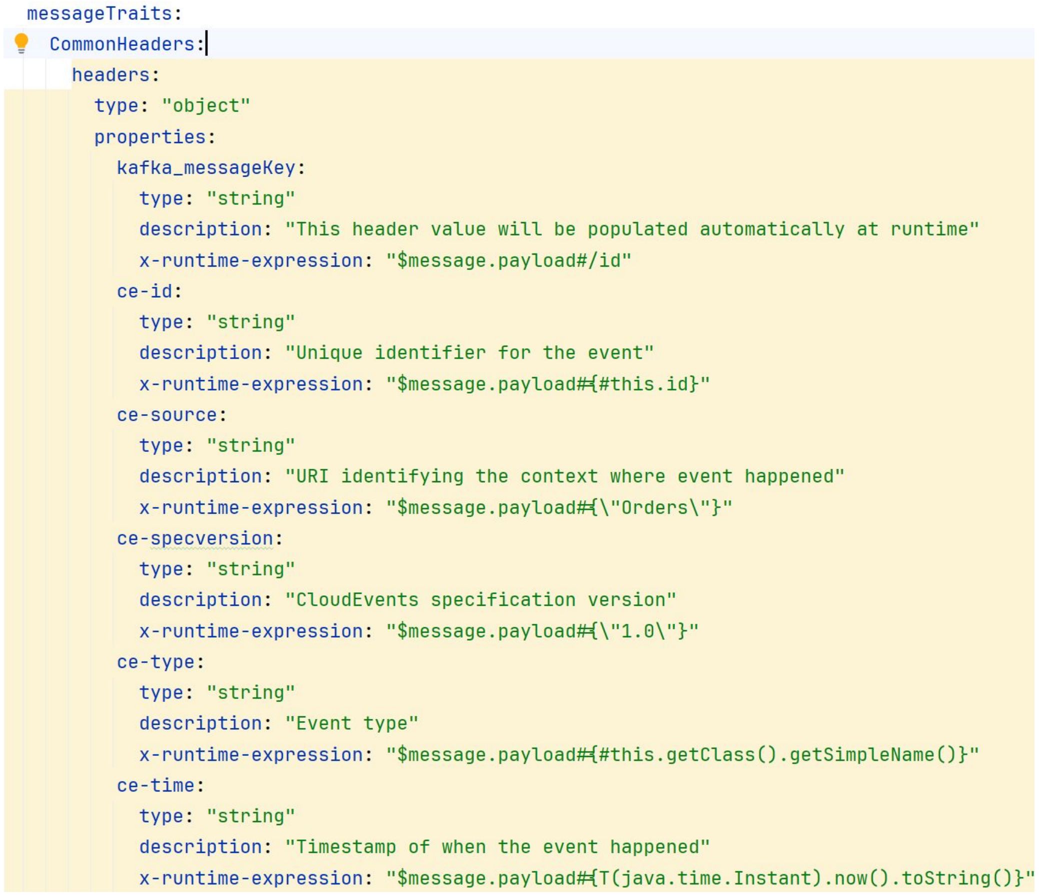
- ✅ **API-First Code Generation** from AsyncAPI
- ✅ **Reverse Engineering** AsyncAPI definition from your Event classes
- ✅ Spring Modulith Event **Externalizer**
  - ✅ **Spring Cloud Stream** - Message<?>
  - ✅ Support for **Avro and Json** payloads
  - ✅ Support for **Headers**/Metadata (CloudEvents)

Event Externalization



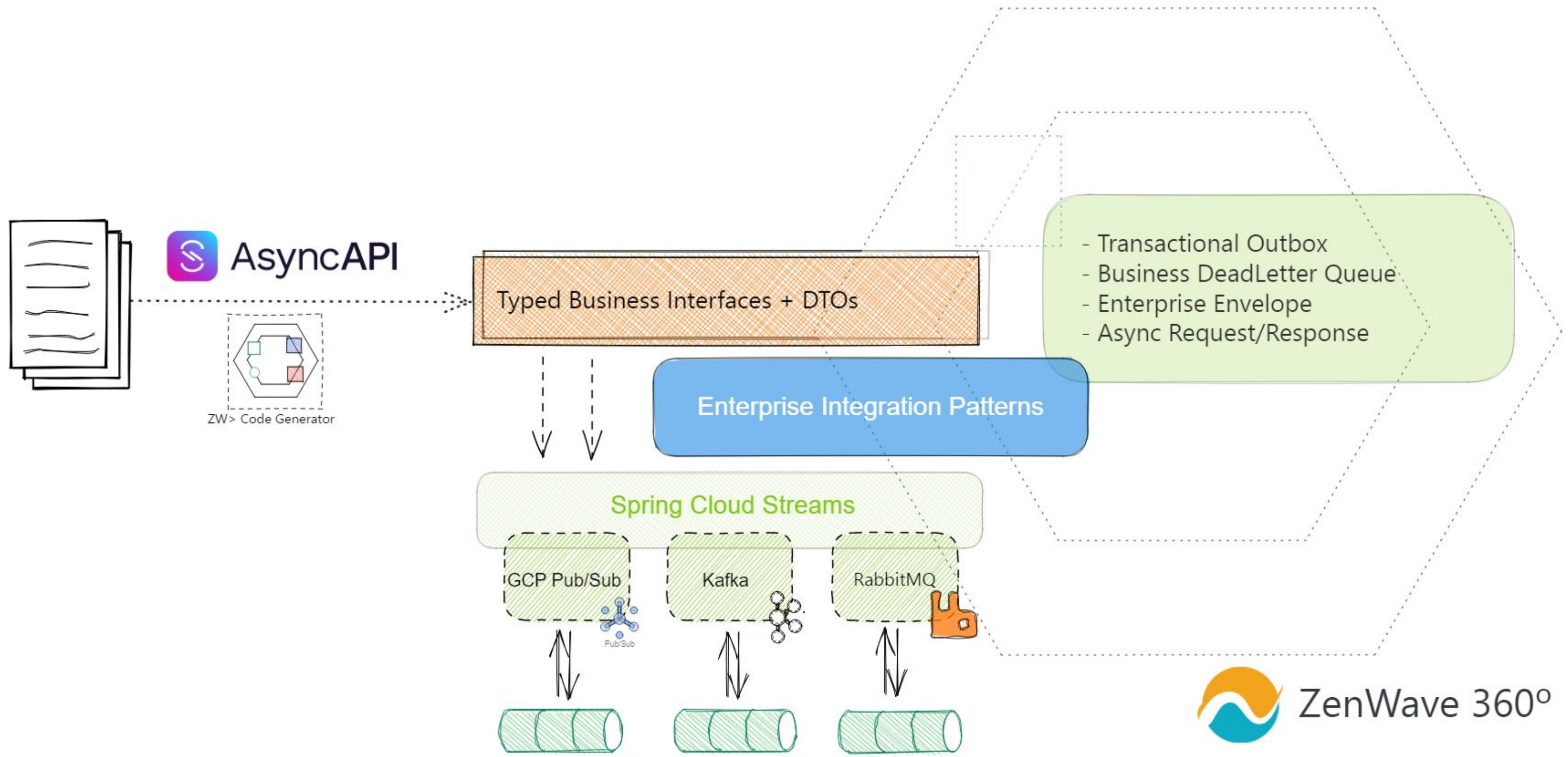
```java
//DEPS io.zenwave360.sdk.plugins:java-to-asyncapi:2.0.0
public class JavaEventsToAsyncAPI {

    public static void main(String[] args) throws IOException {
        String asyncapi = new JavaToAsyncAPIGenerator()
                .withEventProducerClass(EventPublisher.class) // ⟵ your event publisher class
                .withServiceName("OrdersService")
                .withAsyncapiVersion(AsyncapiVersionType.v3)
                .withIncludeKafkaCommonHeaders(true)
                .withIncludeCloudEventsHeaders(true)
                .withAsyncapiMergeFile("src/main/resources/public/apis/_base-asyncapi.yml")
                .withAsyncapiOverlayFiles(List.of( e1: "src/main/resources/public/apis/asyncapi-overlay.yml"))
                .withTargetFile("src/main/resources/public/apis/asyncapi-orders.yml")
                .generate();
        System.out.println(asyncapi); // printing for debug purposes
    }
}
```

Event Externalization



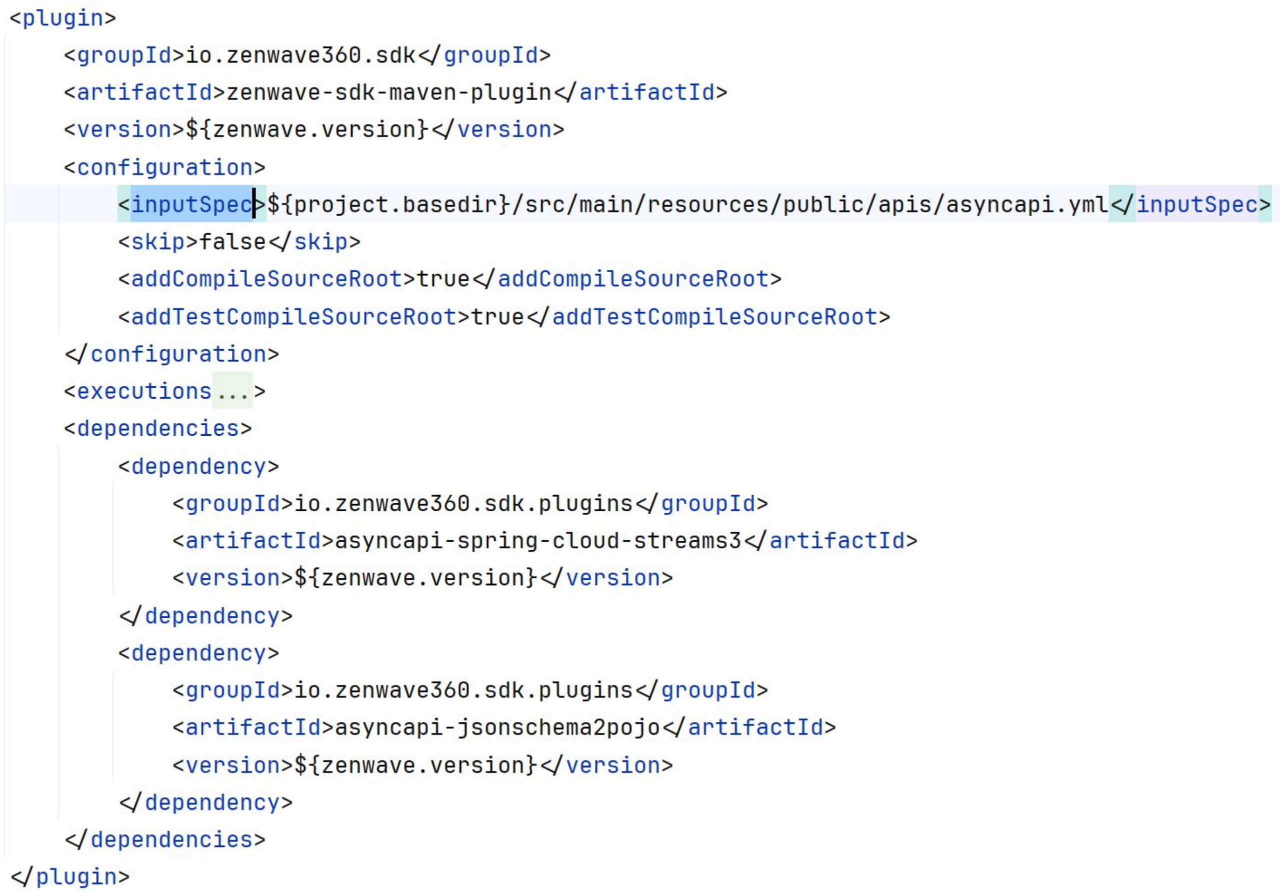
```yaml
messageTraits:
  CommonHeaders:
    headers:
      type: "object"
      properties:
        kafka_messageKey:
          type: "string"
          description: "This header value will be populated automatically at runtime"
          x-runtime-expression: "$message.payload#/id"
        ce-id:
          type: "string"
          description: "Unique identifier for the event"
          x-runtime-expression: "$message.payload#{#this.id}"
        ce-source:
          type: "string"
          description: "URI identifying the context where event happened"
          x-runtime-expression: "$message.payload#{\"Orders\"}"
        ce-specversion:
          type: "string"
          description: "CloudEvents specification version"
          x-runtime-expression: "$message.payload#{\"1.0\"}"
        ce-type:
          type: "string"
          description: "Event type"
          x-runtime-expression: "$message.payload#{#this.getClass().getSimpleName()}"
        ce-time:
          type: "string"
          description: "Timestamp of when the event happened"
          x-runtime-expression: "$message.payload#{T(java.time.Instant).now().toString()}"
```

ZenWave API-First Code Generator



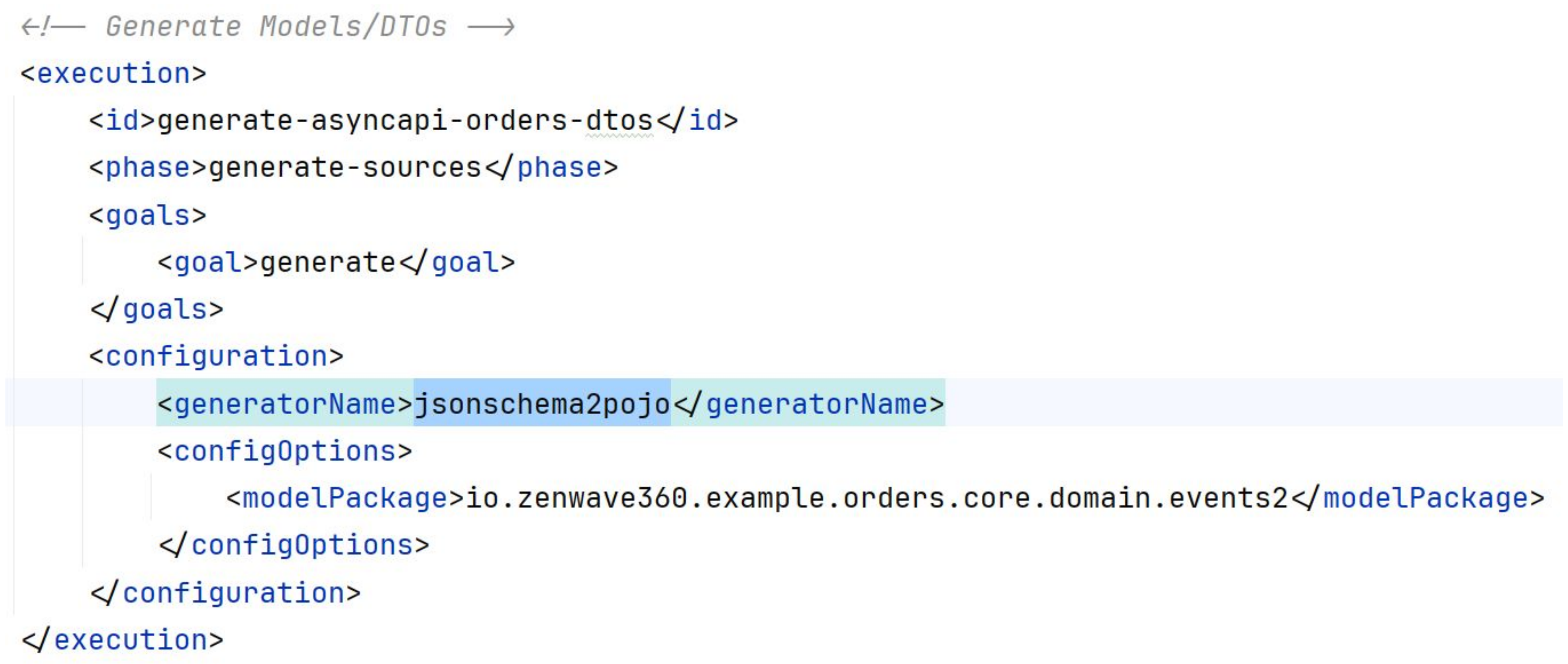
ZenWave 360º

API-First Code Generator for AsyncAPI



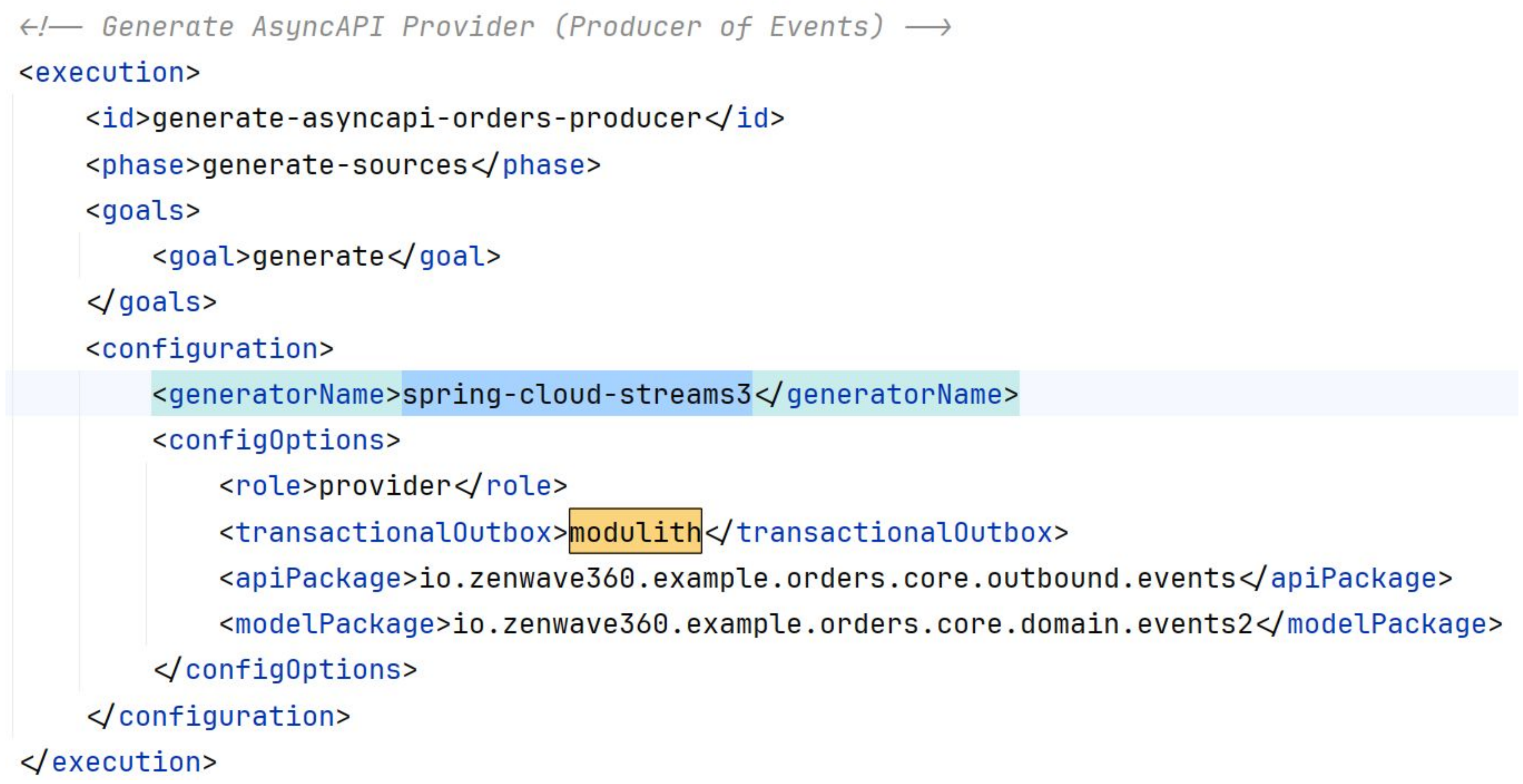
```xml
<plugin>
    <groupId>io.zenwave360.sdk</groupId>
    <artifactId>zenwave-sdk-maven-plugin</artifactId>
    <version>${zenwave.version}</version>
    <configuration>
        <inputSpec>${project.basedir}/src/main/resources/public/apis/asyncapi.yml</inputSpec>
        <skip>false</skip>
        <addCompileSourceRoot>true</addCompileSourceRoot>
        <addTestCompileSourceRoot>true</addTestCompileSourceRoot>
    </configuration>
    <executions...>
    <dependencies>
        <dependency>
            <groupId>io.zenwave360.sdk.plugins</groupId>
            <artifactId>asyncapi-spring-cloud-streams3</artifactId>
            <version>${zenwave.version}</version>
        </dependency>
        <dependency>
            <groupId>io.zenwave360.sdk.plugins</groupId>
            <artifactId>asyncapi-jsonschema2pojo</artifactId>
            <version>${zenwave.version}</version>
        </dependency>
    </dependencies>
</plugin>
```

API-First Code Generator for AsyncAPI



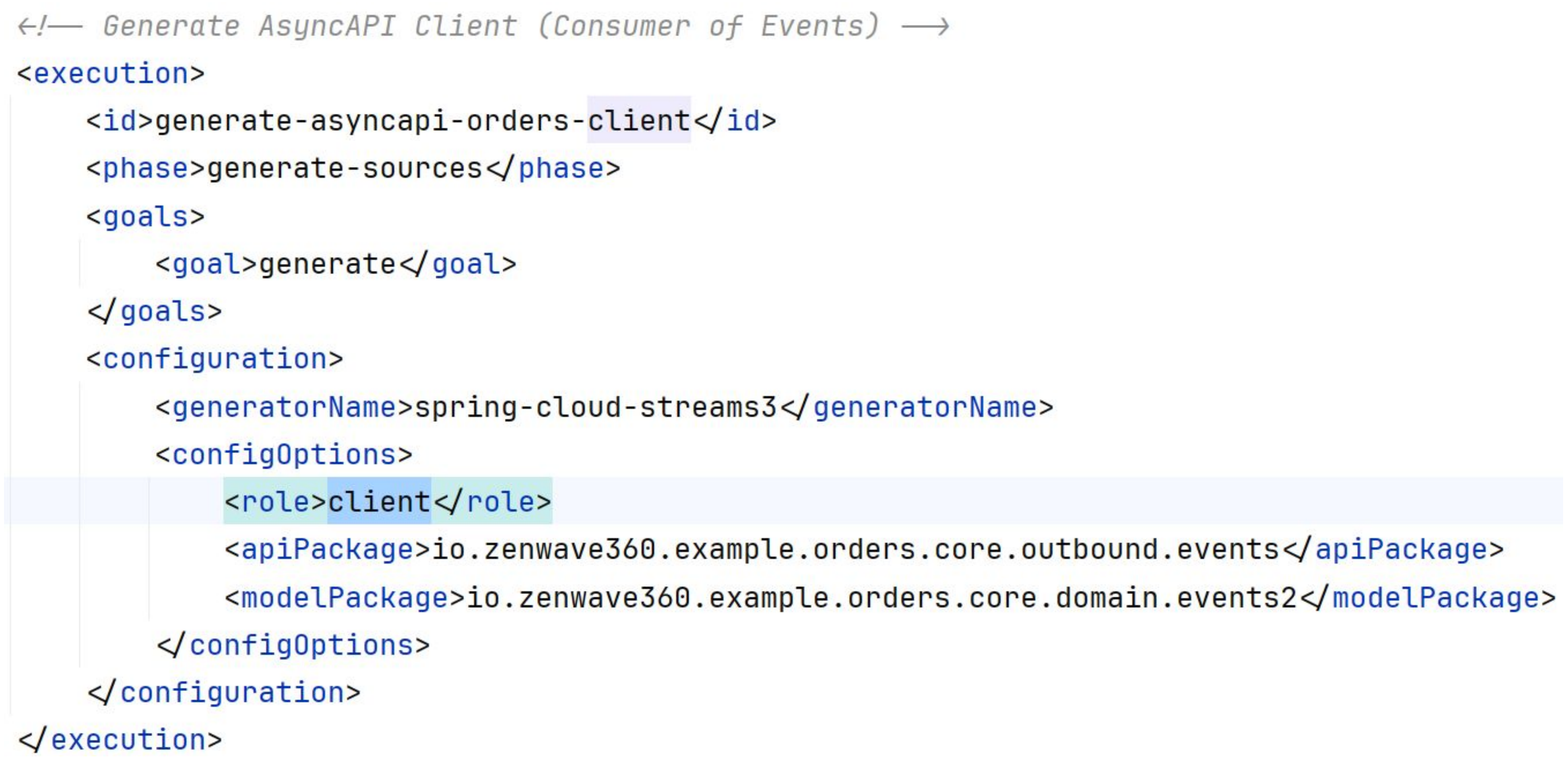
```xml
<!-- Generate Models/DTOs -->
<execution>
    <id>generate-asyncapi-orders-dtos</id>
    <phase>generate-sources</phase>
    <goals>
        <goal>generate</goal>
    </goals>
    <configuration>
        <generatorName>jsonschema2pojo</generatorName>
        <configOptions>
            <modelPackage>io.zenwave360.example.orders.core.domain.events2</modelPackage>
        </configOptions>
    </configuration>
</execution>
```

API-First Code Generator for AsyncAPI



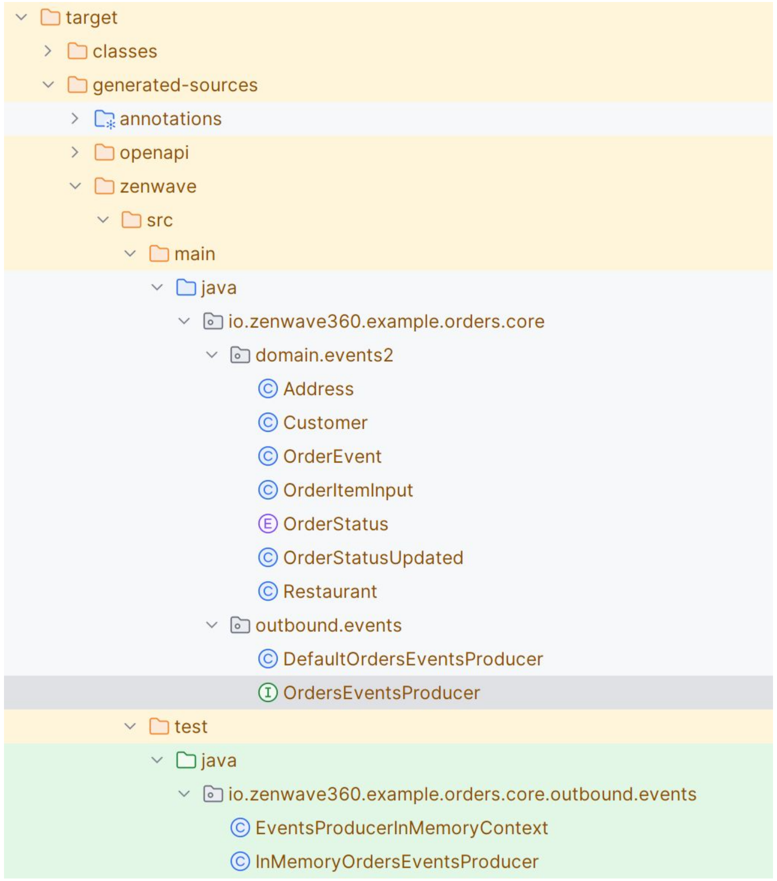
```xml
<!-- Generate AsyncAPI Provider (Producer of Events) -->
<execution>
    <id>generate-asyncapi-orders-producer</id>
    <phase>generate-sources</phase>
    <goals>
        <goal>generate</goal>
    </goals>
    <configuration>
        <generatorName>spring-cloud-streams3</generatorName>
        <configOptions>
            <role>provider</role>
            <transactionalOutbox>modulith</transactionalOutbox>
            <apiPackage>io.zenwave360.example.orders.core.outbound.events</apiPackage>
            <modelPackage>io.zenwave360.example.orders.core.domain.events2</modelPackage>
        </configOptions>
    </configuration>
</execution>
```

API-First Code Generator for AsyncAPI



```xml
<!-- Generate AsyncAPI Client (Consumer of Events) -->
<execution>
    <id>generate-asyncapi-orders-client</id>
    <phase>generate-sources</phase>
    <goals>
        <goal>generate</goal>
    </goals>
    <configuration>
        <generatorName>spring-cloud-streams3</generatorName>
        <configOptions>
            <role>client</role>
            <apiPackage>io.zenwave360.example.orders.core.outbound.events</apiPackage>
            <modelPackage>io.zenwave360.example.orders.core.domain.events2</modelPackage>
        </configOptions>
    </configuration>
</execution>
```

API-First Code Generator for AsyncAPI

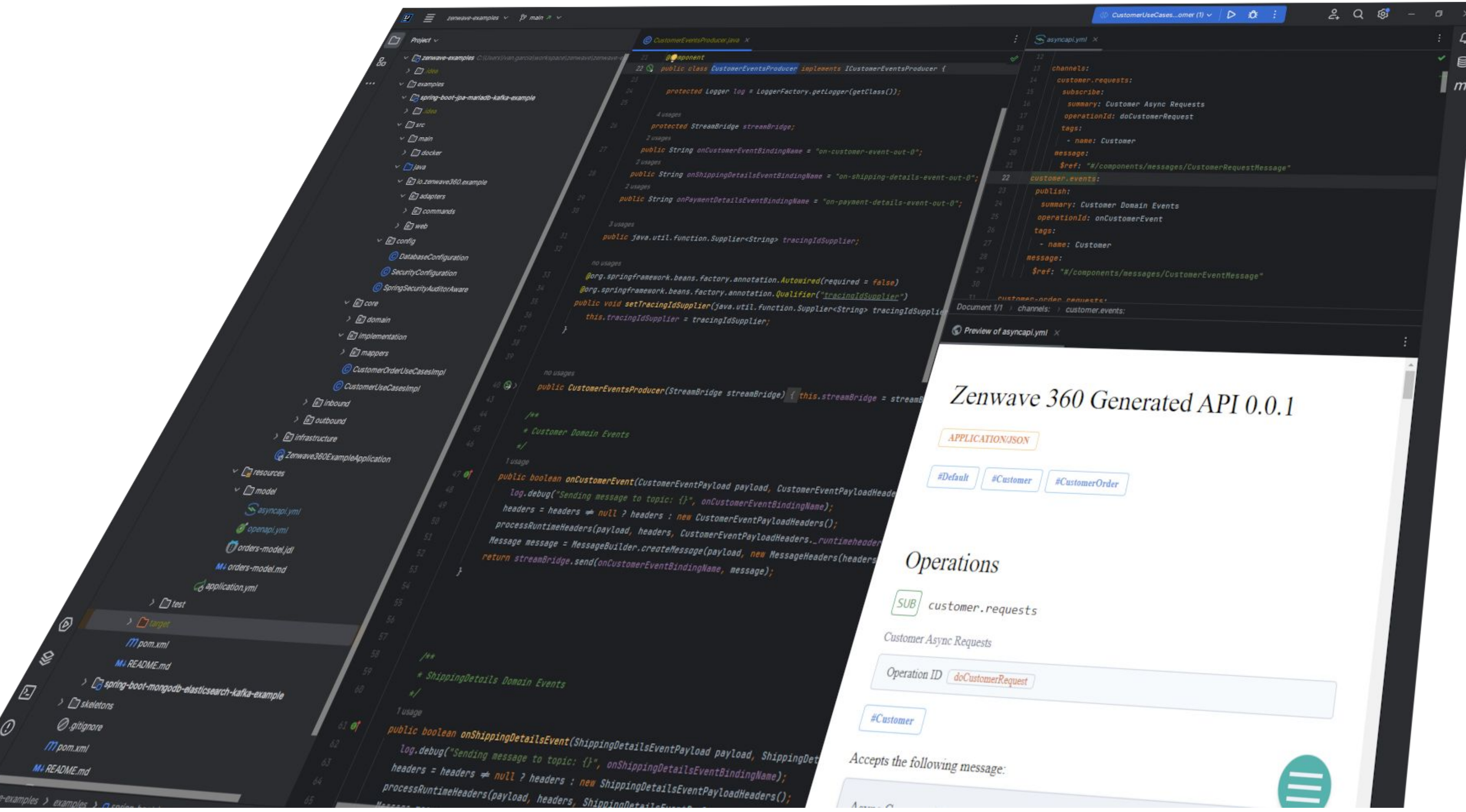


```
target
  classes
  generated-sources
    annotations
    openapi
    zenwave
      src
        main
          java
            io.zenwave360.example.orders.core
              domain.events2
                Address
                Customer
                OrderEvent
                OrderItemInput
                OrderStatus
                OrderStatusUpdated
                Restaurant
              outbound.events
                DefaultOrdersEventsProducer
                OrdersEventsProducer
        test
          java
            io.zenwave360.example.orders.core.outbound.events
              EventsProducerInMemoryContext
              InMemoryOrdersEventsProducer
```

Models / DTOs

Events Producer (Spring Cloud Stream)

Events Producer (In Memory)



```
Project
zenwave-examples  C:\Users\ivan.garcia\workspace\zenwave\zenwave-e
  .idea
  examples
    spring-boot-jpa-mariadb-kafka-example
      .idea
      src
        main
          docker
          java
            io.zenwave360.example
              adapters
                commands
                web
              config
                DatabaseConfiguration
                SecurityConfiguration
                SpringSecurityAuditorAware
              core
                domain
                implementation
                  mappers
                  CustomerOrderUseCasesImpl
                  CustomerUseCasesImpl
                inbound
                outbound
              infrastructure
              Zenwave360ExampleApplication
          resources
            model
              asyncapi.yml
              openapi.yml
              orders-model.jdl
              orders-model.md
            application.yml
        test
      target
      pom.xml
      README.md
    spring-boot-mongodb-elasticsearch-kafka-example
  skeletons
  .gitignore
  pom.xml
  README.md
```

CustomerEventsProducer.java

```java
@Component
public class CustomerEventsProducer implements ICustomerEventsProducer {

    protected Logger log = LoggerFactory.getLogger(getClass());

    4 usages
    protected StreamBridge streamBridge;
    2 usages
    public String onCustomerEventBindingName = "on-customer-event-out-0";
    2 usages
    public String onShippingDetailsEventBindingName = "on-shipping-details-event-out-0";
    2 usages
    public String onPaymentDetailsEventBindingName = "on-payment-details-event-out-0";

    3 usages
    public java.util.function.Supplier<String> tracingIdSupplier;

    no usages
    @org.springframework.beans.factory.annotation.Autowired(required = false)
    @org.springframework.beans.factory.annotation.Qualifier("tracingIdSupplier")
    public void setTracingIdSupplier(java.util.function.Supplier<String> tracingIdSupplie
        this.tracingIdSupplier = tracingIdSupplier;
    }

    no usages
    public CustomerEventsProducer(StreamBridge streamBridge) { this.streamBridge = streamB

    /**
     * Customer Domain Events
     */
    1 usage
    public boolean onCustomerEvent(CustomerEventPayload payload, CustomerEventPayloadHeade
        log.debug("Sending message to topic: {}", onCustomerEventBindingName);
        headers = headers != null ? headers : new CustomerEventPayloadHeaders();
        processRuntimeHeaders(payload, headers, CustomerEventPayloadHeaders._runtimeheader
        Message message = MessageBuilder.createMessage(payload, new MessageHeaders(headers
        return streamBridge.send(onCustomerEventBindingName, message);
    }



    /**
     * ShippingDetails Domain Events
     */
    1 usage
    public boolean onShippingDetailsEvent(ShippingDetailsEventPayload payload, ShippingDet
        log.debug("Sending message to topic: {}", onShippingDetailsEventBindingName);
        headers = headers != null ? headers : new ShippingDetailsEventPayloadHeaders();
        processRuntimeHeaders(payload, headers, ShippingDetailsEvent
```

asyncapi.yml

```yaml
channels:
  customer.requests:
    subscribe:
      summary: Customer Async Requests
      operationId: doCustomerRequest
      tags:
        - name: Customer
      message:
        $ref: "#/components/messages/CustomerRequestMessage"
  customer.events:
    publish:
      summary: Customer Domain Events
      operationId: onCustomerEvent
      tags:
        - name: Customer
      message:
        $ref: "#/components/messages/CustomerEventMessage"

  customer-order.requests:
```

Document 1/1 › channels: › customer.events:

Preview of asyncapi.yml

Zenwave 360 Generated API 0.0.1

APPLICATION/JSON

#Default #Customer #CustomerOrder

Operations

SUB customer.requests

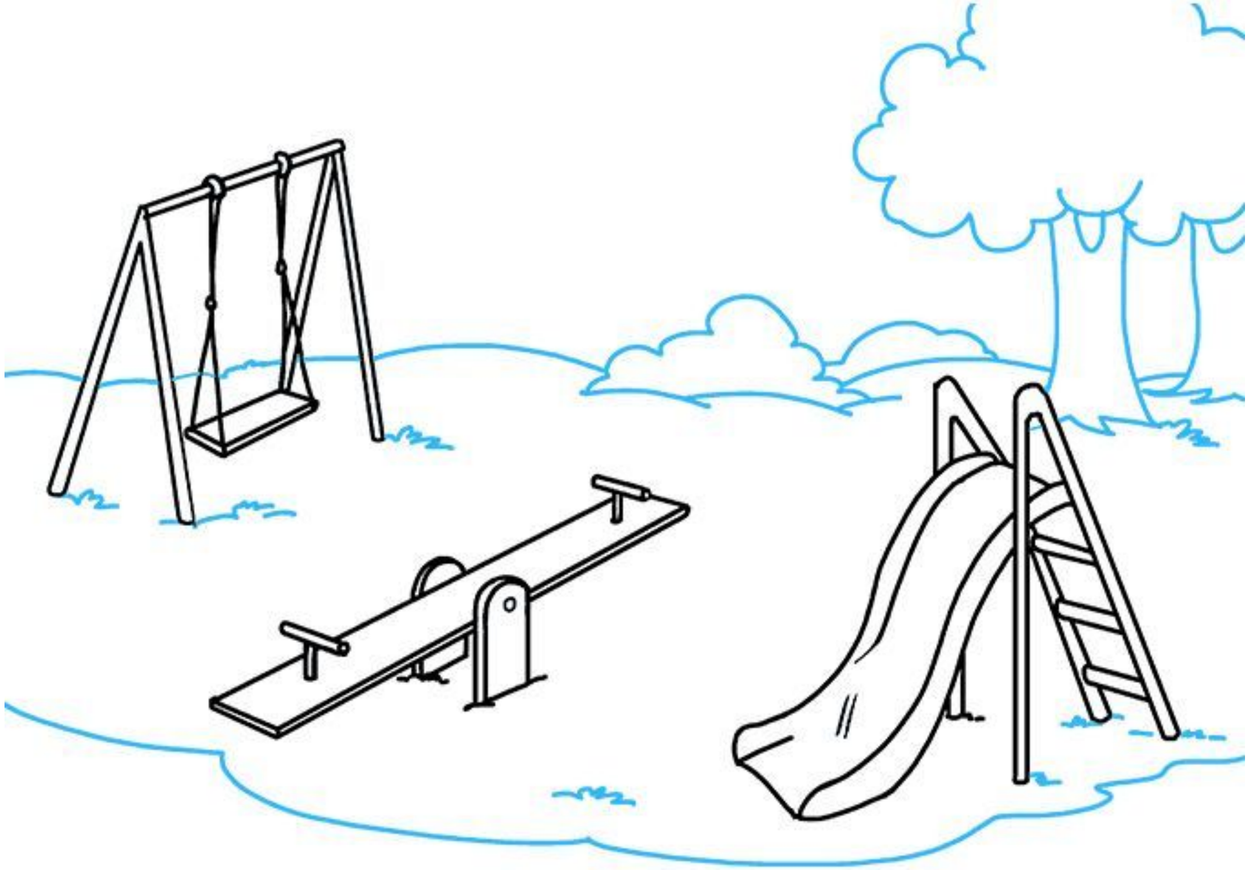
Customer Async Requests

Operation ID doCustomerRequest

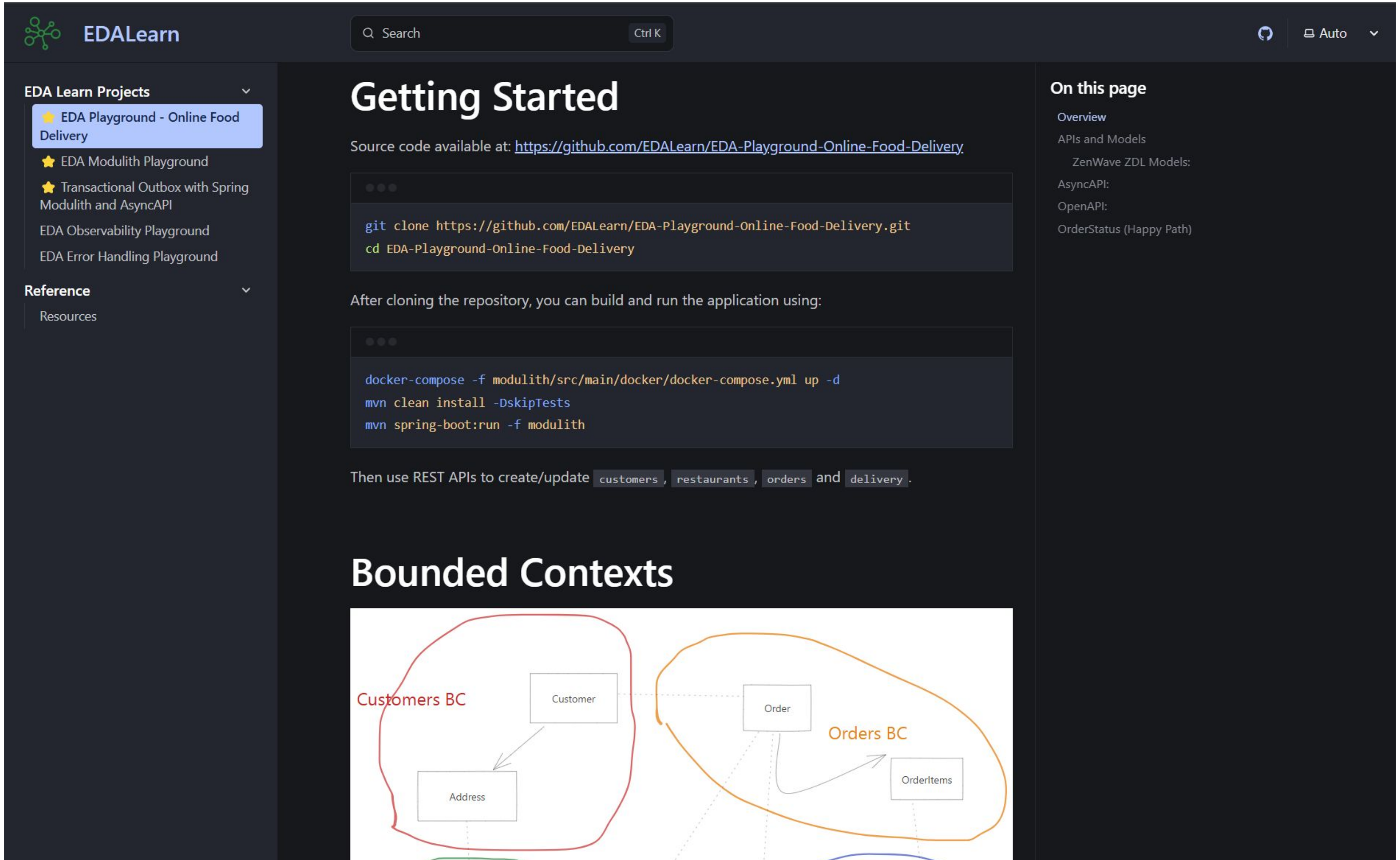
#Customer

Accepts the following message:

EDA Playgrounds



Event-Driven Architectures



EDALearn

Search Ctrl K

Auto

EDA Learn Projects

- EDA Playground - Online Food Delivery
- EDA Modulith Playground
- Transactional Outbox with Spring Modulith and AsyncAPI
- EDA Observability Playground
- EDA Error Handling Playground

Reference

- Resources

Getting Started

Source code available at: https://github.com/EDALearn/EDA-Playground-Online-Food-Delivery

```
git clone https://github.com/EDALearn/EDA-Playground-Online-Food-Delivery.git
cd EDA-Playground-Online-Food-Delivery
```

After cloning the repository, you can build and run the application using:

```
docker-compose -f modulith/src/main/docker/docker-compose.yml up -d
mvn clean install -DskipTests
mvn spring-boot:run -f modulith
```

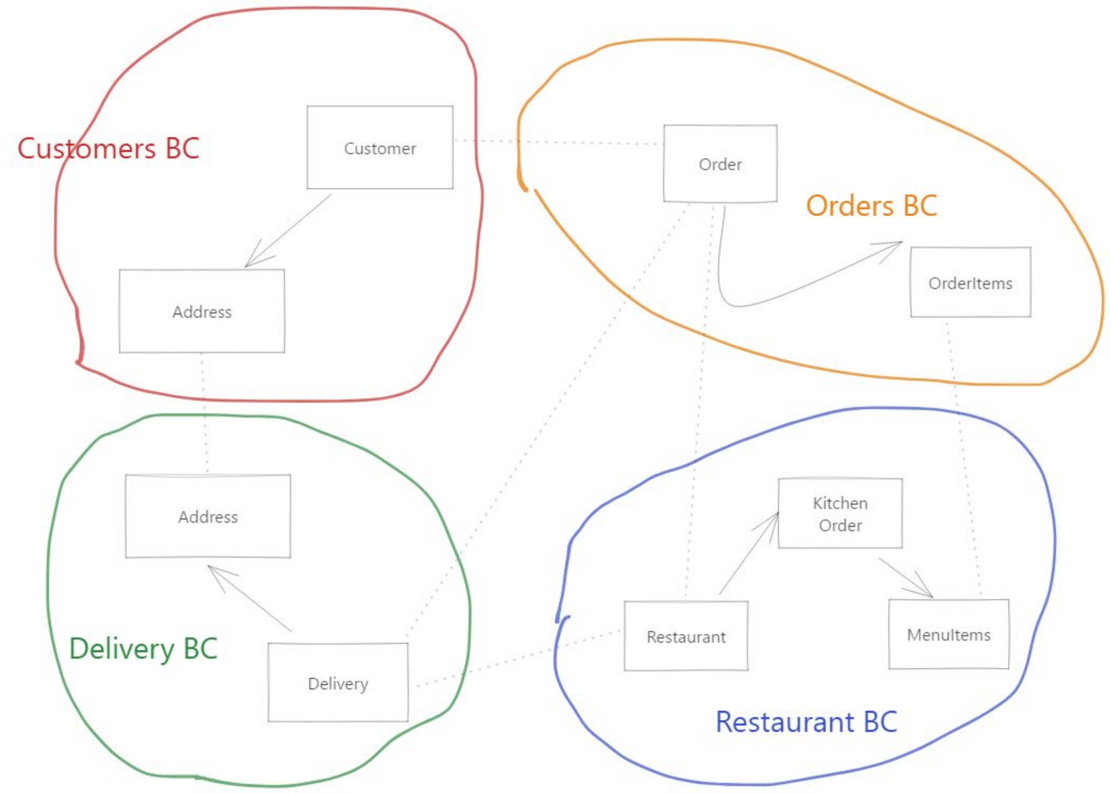
Then use REST APIs to create/update `customers`, `restaurants`, `orders` and `delivery`.

Bounded Contexts

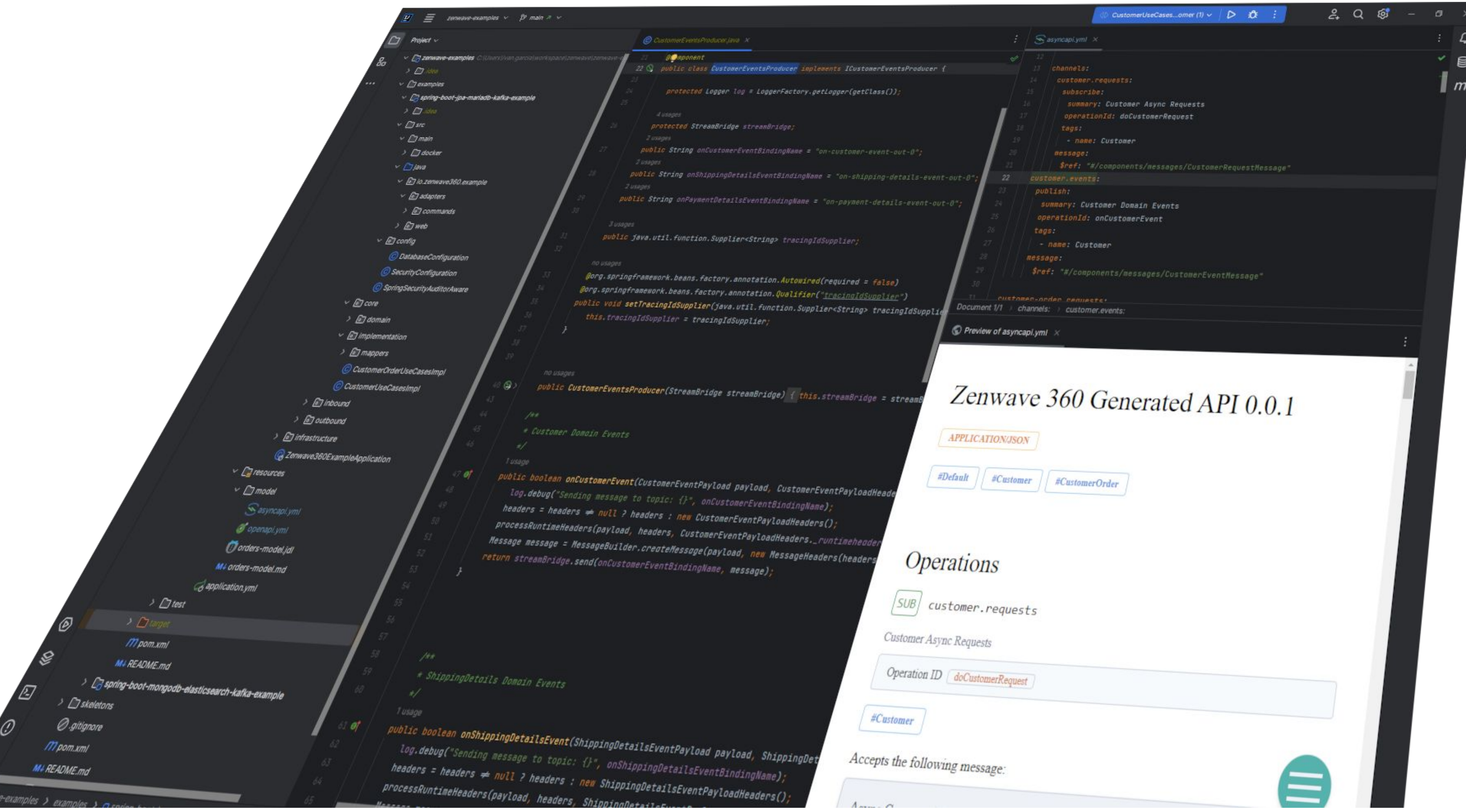
On this page

- Overview
- APIs and Models
  - ZenWave ZDL Models:
- AsyncAPI:
- OpenAPI:
- OrderStatus (Happy Path)

Event-Driven Architectures



BOUNDED CONTEXTS



```java
@Component
public class CustomerEventsProducer implements ICustomerEventsProducer {

    protected Logger log = LoggerFactory.getLogger(getClass());

    4 usages
    protected StreamBridge streamBridge;
    2 usages
    public String onCustomerEventBindingName = "on-customer-event-out-0";
    2 usages
    public String onShippingDetailsEventBindingName = "on-shipping-details-event-out-0";
    2 usages
    public String onPaymentDetailsEventBindingName = "on-payment-details-event-out-0";

    3 usages
    public java.util.function.Supplier<String> tracingIdSupplier;

    no usages
    @org.springframework.beans.factory.annotation.Autowired(required = false)
    @org.springframework.beans.factory.annotation.Qualifier("tracingIdSupplier")
    public void setTracingIdSupplier(java.util.function.Supplier<String> tracingIdSupplie
        this.tracingIdSupplier = tracingIdSupplier;
    }

    no usages
    public CustomerEventsProducer(StreamBridge streamBridge) { this.streamBridge = streamB

    /**
     * Customer Domain Events
     */
    1 usage
    public boolean onCustomerEvent(CustomerEventPayload payload, CustomerEventPayloadHeade
        log.debug("Sending message to topic: {}", onCustomerEventBindingName);
        headers = headers != null ? headers : new CustomerEventPayloadHeaders();
        processRuntimeHeaders(payload, headers, CustomerEventPayloadHeaders._runtimeheader
        Message message = MessageBuilder.createMessage(payload, new MessageHeaders(headers
        return streamBridge.send(onCustomerEventBindingName, message);
    }

    /**
     * ShippingDetails Domain Events
     */
    1 usage
    public boolean onShippingDetailsEvent(ShippingDetailsEventPayload payload, ShippingDet
        log.debug("Sending message to topic: {}", onShippingDetailsEventBindingName);
        headers = headers != null ? headers : new ShippingDetailsEventPayloadHeaders();
        processRuntimeHeaders(payload, headers, ShippingDetailsEventPayloadHeaders();
```

```yaml
channels:
  customer.requests:
    subscribe:
      summary: Customer Async Requests
      operationId: doCustomerRequest
      tags:
        - name: Customer
      message:
        $ref: "#/components/messages/CustomerRequestMessage"
  customer.events:
    publish:
      summary: Customer Domain Events
      operationId: onCustomerEvent
      tags:
        - name: Customer
      message:
        $ref: "#/components/messages/CustomerEventMessage"
  customer-order.requests:
```

Zenwave 360 Generated API 0.0.1

APPLICATION/JSON

#Default #Customer #CustomerOrder

Operations

SUB customer.requests

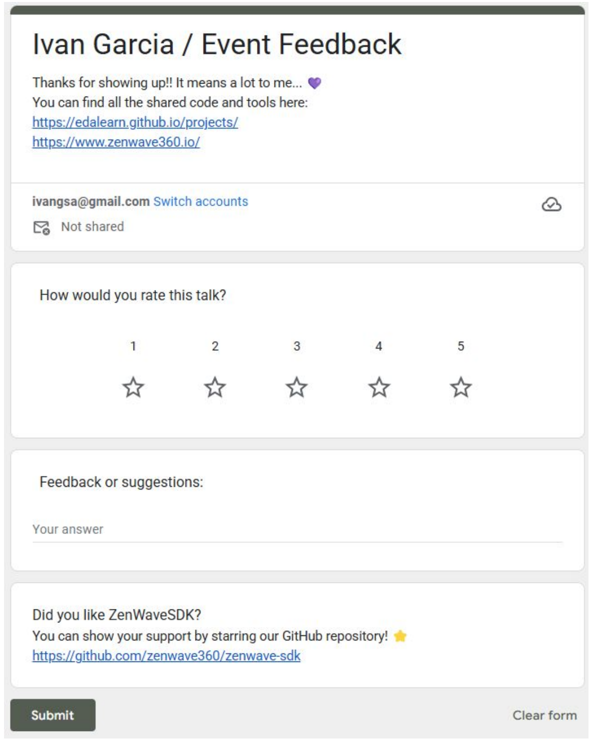
Customer Async Requests

Operation ID doCustomerRequest

#Customer

Accepts the following message:

Thanks for showing up!! 💜



Ivan Garcia / Event Feedback

Thanks for showing up!! It means a lot to me... 💜
You can find all the shared code and tools here:
https://edalearn.github.io/projects/
https://www.zenwave360.io/

ivangsa@gmail.com Switch accounts
Not shared

How would you rate this talk?

1 2 3 4 5

☆ ☆ ☆ ☆ ☆

Feedback or suggestions:

Your answer

Did you like ZenWaveSDK?
You can show your support by starring our GitHub repository! ⭐
https://github.com/zenwave360/zenwave-sdk

Submit Clear form

https://forms.gle/8TBxwN9chckSYMss8