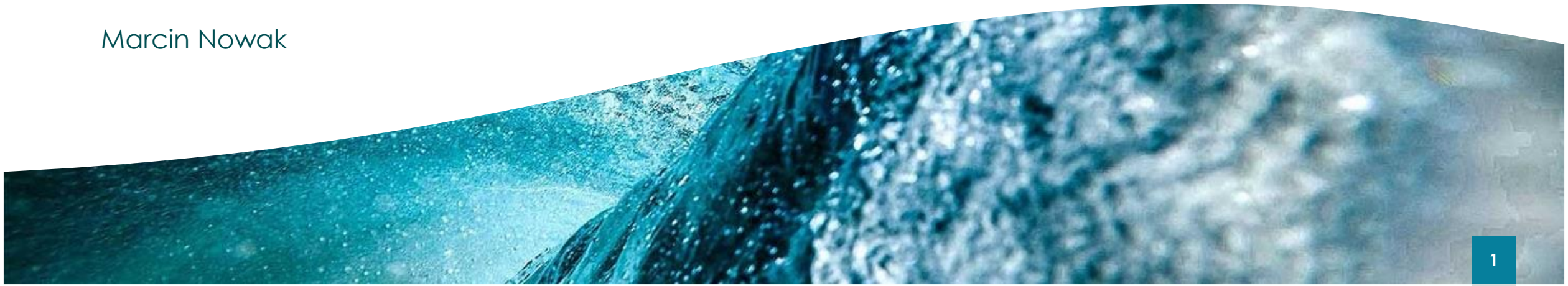
Backend

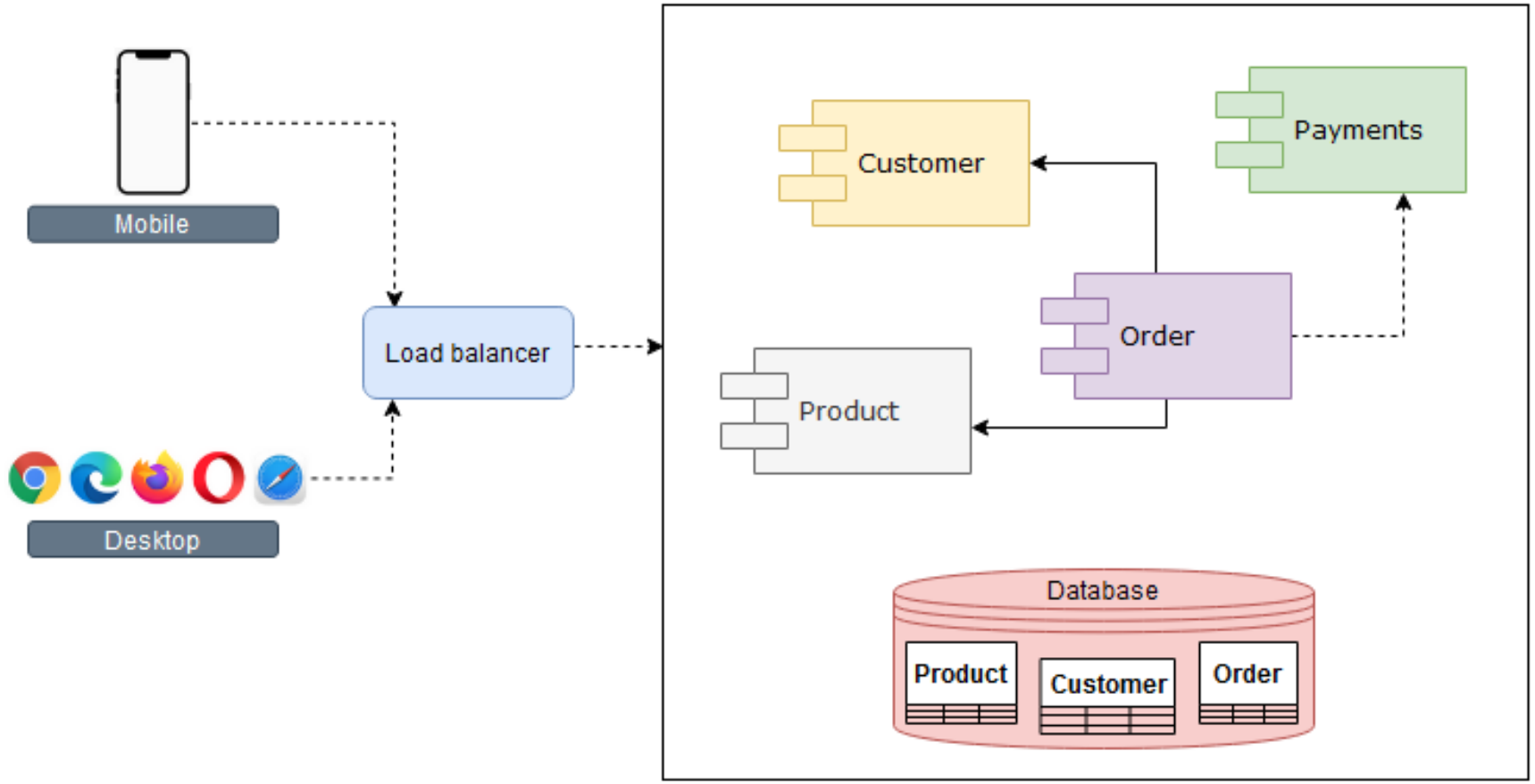
Marcin Nowak



Agenda

1. Architektura monolitowa
2. Architektura heksagonalna
3. Architektura mikroserwisowa
4. Spring – podstawowe koncepty
5. REST
6. AMQP (Advanced Message Query Protocol) na przykładzie RabbitMQ
7. Maven

Architektura monolitowa



Architektura monolitowa

+ Łatwy początek developmentu
+ Łatwy do deploymentu
+ Łatwy do skalowania (jako całość)
+ Testy end-to-end
+ Monitorowanie aplikacji

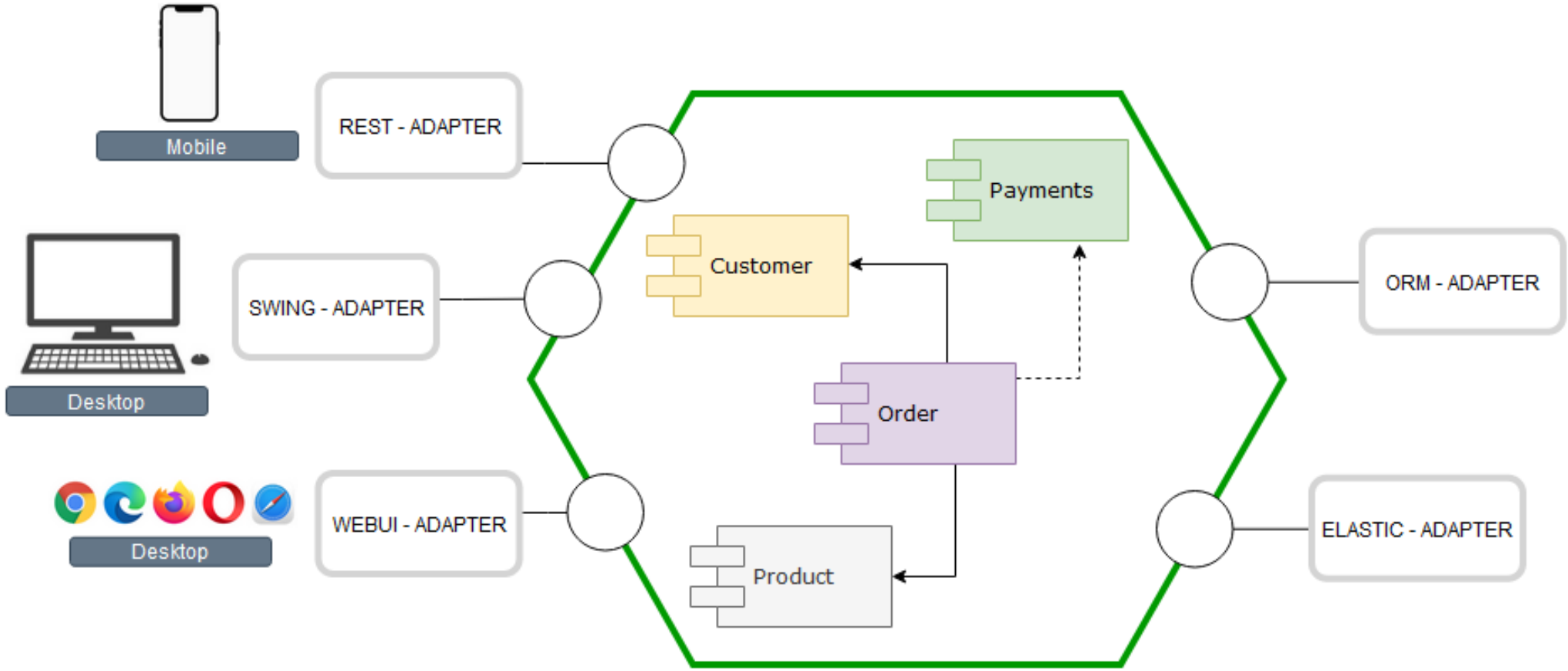
- Wysoki próg wejścia do zespołu (dużo zagmatwanego kodu)
- Przeciążone (wolne) działanie narzędzi (IDE)
- Niemożliwy do skalowania per komponent
- Skalowanie developmentu
- Długofalowe przywiązanie do technologii
- Naprawianie błędów i wprowadzanie nowych funkcjonalności jest czasochłonne (time to market)

> Każda organizacja projektująca system (...) bez wątpienia opracuje projekt, którego struktura jest kopią struktury komunikacji w tej firmie.
>
> - Melvin Convay, *How do committees invent?* (Prawo Convaya)

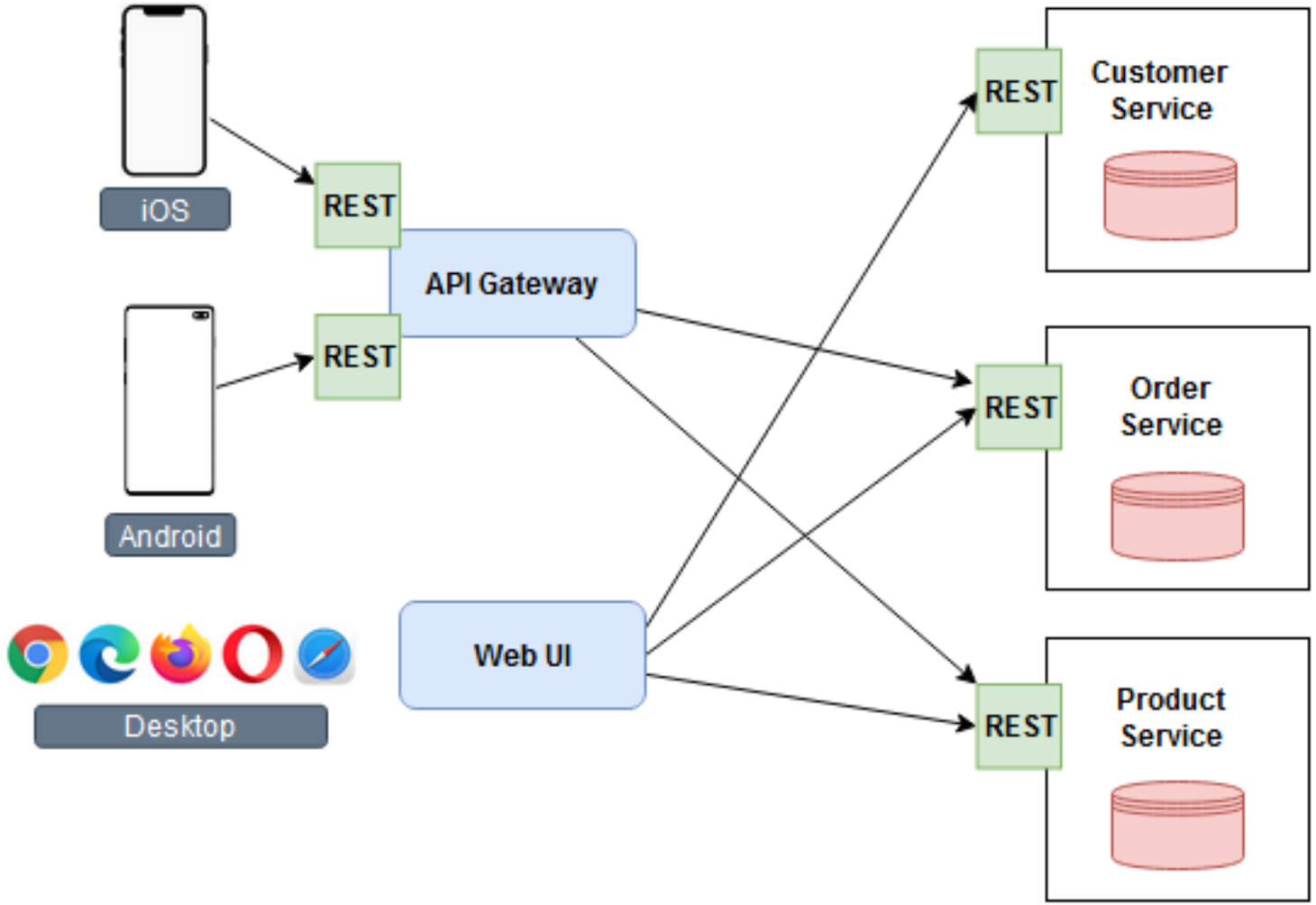
Architektura warstwowa

Prezentacyjna	• REST Controller • Wzywa serwis • Mapowania TS - Encja
Biznesowa	• Logika biznesowa • Serwisy • Orkiestracja • Encje
Integracyjna	• Połączenie z innymi aplikacjami • Via message broker (kolejki, JMS/AMQP) • Via WebService (SOAP) • Via REST
Danych	• Encje • Komunikacja z bazą danych

Architektura Heksagonalna



Architektura mikroserwisowa

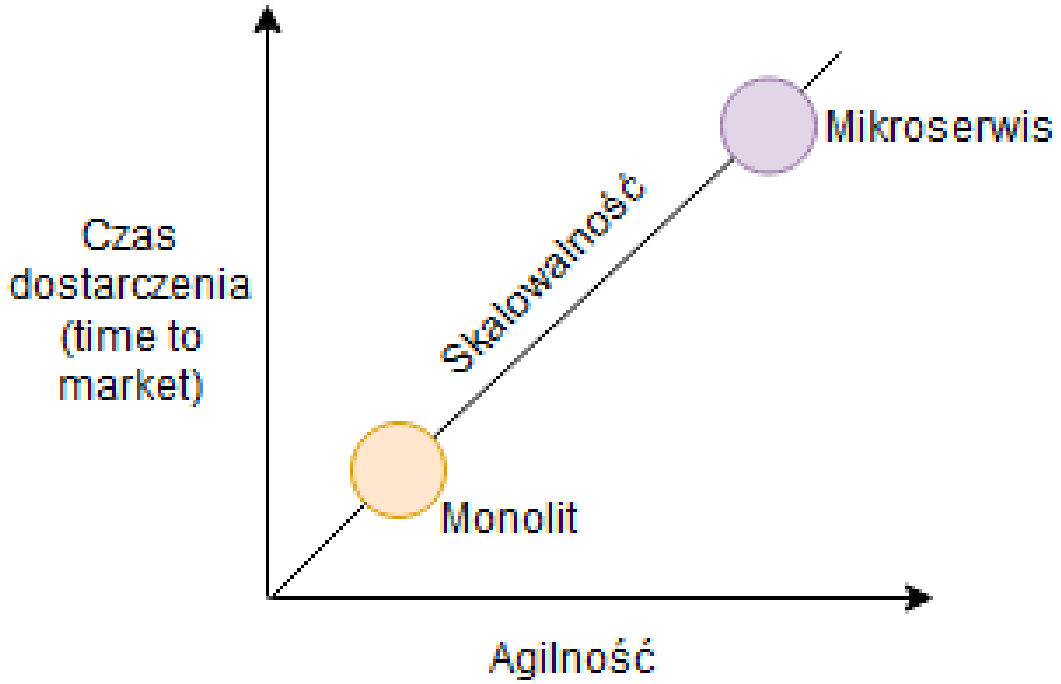


Architektura mikroserwisowa

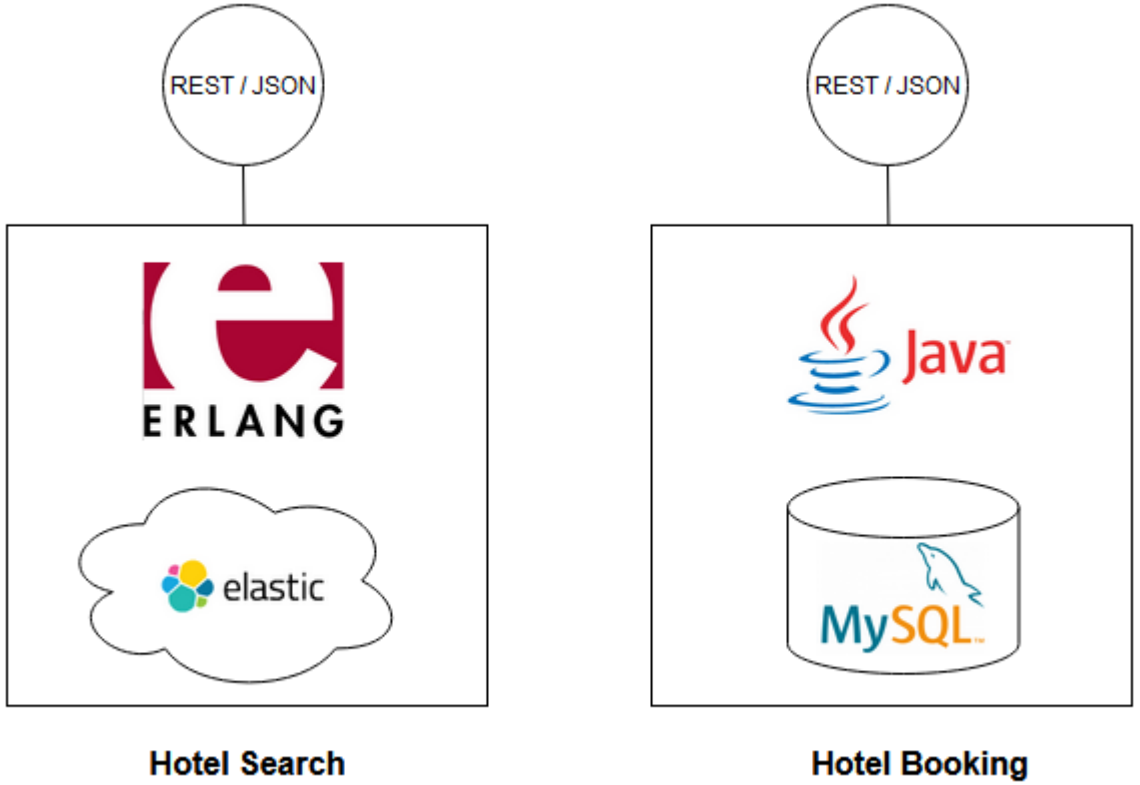
+ Złożony system podzielony na łatwiej zarządzalne serwisy
+ Większa elastyczność przy wyborze technologii
+ Łatwy deployment poszczególnych części
+ Skalowalność per mikroserwis
+ Niższy próg wejścia dla developera per mikroserwis

- Obsługa komunikacji między mikroserwisami
- Rozproszone transakcje
- Bardziej czasochłonne testy integracyjne
- Dostarczanie zmian afektujących zmiane zależne od kilku mikroserwisów
- Trudniejszy deployment
- Trudny podział na mikroserwisy

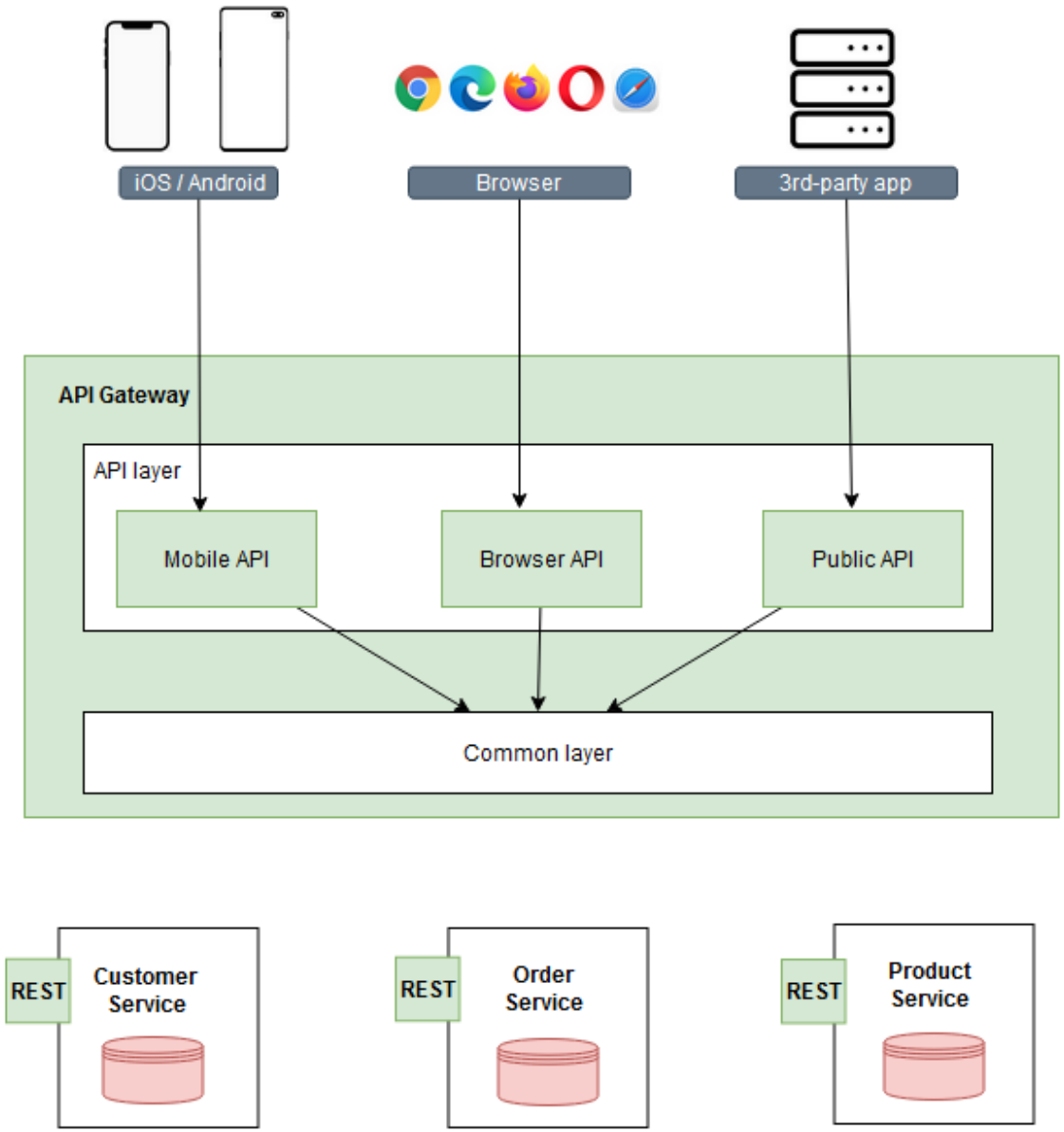
W kierunku mikroserwisów



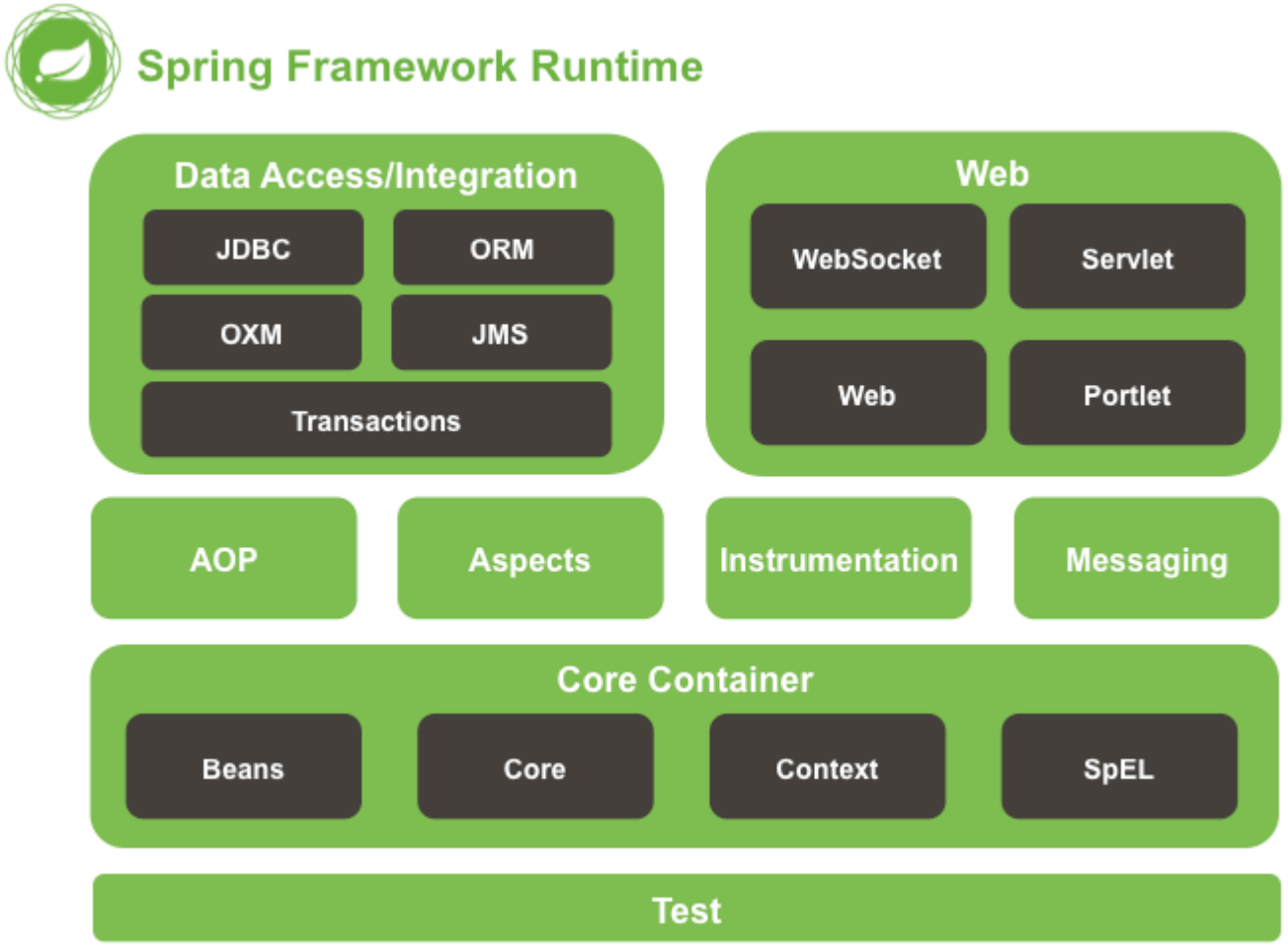
Mikroserwisy – większa techn. elastyczność



API Gateway


iOS / Android
Browser
3rd-party app
API Gateway
API layer
Mobile API
Browser API
Public API
Common layer
REST
Customer Service
REST
Order Service
REST
Product Service

Spring


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

Spring – skąd popularność?

- Uproszczone testowanie jednostkowe
- Redukcja nadmiarowego kodu
- Elastyczność architektury
- Aktualność

```
PreparedStatement st = null;
try {
    st = conn.prepareStatement(INSERT_BOOK_QUERY);
    st.setString(1, object.getTitle());
    st.setBoolean(2, object.isLongAndBoring());
    st.execute();
}
catch (SQLException e) {
    logger.error("Failed : " + INSERT_BOOK_QUERY, e);
}
finally {
    if (st != null) {
        try {
            st.close();
        }
        catch (SQLException e) {
            // nothing to be done
        }
    }
}
```

```
jdbcTemplate.update(INSERT_BOOK_QUERY,
    object.getTitle(),
    object.isLongAndBoring()
);
```

Spring Dependency Injection

```
public class QuoteServiceImpl {
  public Document retrieveBillableDocument(Quote quote) {
    DocumentServiceImpl documentService = new DocumentServiceImpl();
    List<Document> documents = documentService.findDocuments(quote.getQuoteNumber())

    for (Document document : documents) {
        if(document.isBillable()){
            return document;
        }
    }

    return null;
  }
}
```

Spring Dependency Injection

```
public interface DocumentService {
   List<Document> findDocuments(String quoteNumber);
}

public interface QuoteService {
  public Document retrieveBillableDocument(Quote quote);
}

public class QuoteServiceImpl implements QuoteService{
  private DocumentService documentService;

  @Override
  public Document retrieveBillableDocument(Quote quote) {
    List<Document> documents = documentService.findDocuments(quote.getQuoteNumber())

    for (Document document : documents) {
        if(document.isBillable()){
            return document;
        }
    }

    return null;
  }

  public void setDocumentService(DocumentService documentService) {
    this.documentService = documentService;
  }
}
```

Spring Dependency Injection

Skąd Spring IoC wie, żeby stworzyć beany dla QuoteServiceImpl i DocumentServiceImpl?

Skąd kontener Spring IoC wie, żeby wstrzyknąć DocumentServiceImpl do QuoteServiceImpl?

Skąd kontener Spring IoC wie, gdzie szukać beanów?

Spring Dependency Injection

Skąd Spring IoC wie, żeby stworzyć beany dla QuoteServiceImpl i DocumentServiceImpl?

```
@Repository
public class DocumentServiceImpl implements DocumentService
```

```
@Service
public class QuoteServiceImpl implements QuoteService
```

Spring Dependency Injection

Skąd kontener Spring IoC wie, żeby wstrzyknąć DocumentServiceImpl do QuoteServiceImpl?

```
@Service
public class QuoteServiceImpl implements QuoteService

  @Autowired
  private DocumentService documentService;
```

Spring Dependency Injection

Skąd kontener Spring IoC wie gdzie szukać beanów?

```java
package pl.wroc.pwr.data.impl;

@Repository
public class DocumentService implements DocumentService
```

```java
package pl.wroc.pwr.service.impl;

@Service
public class QuoteServiceImpl implements QuoteService
```

```java
@Configuration
@ComponentScan(basePackages = { "pl.wroc.pwr" })
class SpringContext {
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<beans>
    <context:component-scan base-package ="pl.wroc.pwr"/>
</beans>
```

Spring DI i Mockito

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"/applicationContext.xml" })
public class QuoteServiceTest {

    @Autowired
    private QuoteService quoteService;

    @Test
    public void testFindDocument() {
        //given
        String quoteNumber = "ABC1234";
        //data loaded in test schema

        //when
        List<Document> documents = quoteService.findDocuments(quoteNumber);

        //then
        assertThat(documents, hasSize(1));
    }
}
```

```java
@RunWith(MockitoJUnitRunner.class)
public class QuoteServiceTest {

    @InjectMocks
    private QuoteService quoteService = new QuoteServiceImpl();

    @Mock
    private DocumentService documentService;

    @Test
    public void testFindDocument() {
        //given
        String quoteNumber = "ABC1234";
        Document document = new Document();
        document.setBillable(true);

        //when
        Mockito.when(documentService.findDocuments(ArgumentMatchers.eq(quoteNumber)))
                .thenReturn(document);
        List<Document> documents = quoteService.findDocuments(quoteNumber);

        //then
        assertThat(documents, hasSize(1));
    }
}
```

Spring AOP

```
@Aspect
@Component
public class RestLoggerAspect {
  private static final Logger LOGGER = LoggerFactory.getLogger(RestLoggerAspect.class);

  @Pointcut("@annotation(org.springframework.web.bind.annotation.GetMapping)")
  private void getMappingPointcut() {}

  @Pointcut("@annotation(org.springframework.web.bind.annotation.PostMapping)")
  private void postMappingPointcut() {}

  @Pointcut("@annotation(org.springframework.web.bind.annotation.DeleteMapping)")
  private void deleteMappingPointcut() {}

  @Pointcut("@annotation(org.springframework.web.bind.annotation.PutMapping)")
  private void putMappingPointcut() {}

  @Around("getMappingPointcut() || (deleteMappingPointcut()) && !cacheRestServicePointcut() && !getUserRestServicePointcut()")
  public Object logMethodsWithArguments(ProceedingJoinPoint joinPoint) throws Throwable {
    ImmutablePair<Object, StopWatch> runResult = runTaskWithTimeMeasurement(joinPoint);
    if (LOGGER.isInfoEnabled()) {
      LOGGER.info("logs...");
    }
    return runResult.getKey();
  }
```

Spring AOP

```
xmlns:tx="http://www.springframework.org/schema/tx"


<tx:advice
    id="txDaoAdvice"
    transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method
            name="*"
            rollback-for="Throwable" />
    </tx:attributes>
</tx:advice>

<tx:advice id="serviceInOwnTransactionAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*InOwnTransaction" propagation="REQUIRES_NEW" read-only="false" rollback-for="Throwable" />
    </tx:attributes>
</tx:advice>
```

Spring - profile

application-oracle.properties

```
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle://localhost:1521:xe
spring.datasource.username=root
spring.datasource.password=root
```

```
@Profile({ORACLE})
public interface MessageLogOracleRepository
```

application-h2.properties

```
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:db
spring.datasource.username=sa
spring.datasource.password=sa
```

```
@Profile(H2)
@Repository
public class MockDateQueriesForH2
```

application-{profile}.properties

Spring - profile

```xml
<profiles>
    <profile>
        <id>dev</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <properties>
            <spring.profiles.active>dev</spring.profiles.active>
        </properties>
    </profile>
    <profile>
        <id>prod</id>
        <properties>
            <spring.profiles.active>prod</spring.profiles.active>
        </properties>
    </profile>
</profiles>
```

mvn clean package -Pprod

-Dspring.profiles.active=dev

```xml
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee                    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <context-param>
        <param-name>spring.profiles.active</param-name>
        <param-value>backend-server,backend-DEBUG-server</param-value>
    </context-param>
```

REST

1) *Architektura klient-serwer*
2) *Bezstanowość*
3) *Buforowalność* (cacheability)
4) *Warstwowość*
5) *Jednolitość interfejsu*

Annotacja	Starsza forma	Znaczenie
`@GetMapping`	`@RequestMapping(method = RequestMethod.GET)`	Pobiera dane
`@PostMapping`	`@RequestMapping(method = RequestMethod.POST)`	Tworzy nowe dane
`@PutMapping`	`@RequestMapping(method = RequestMethod.PUT)`	Aktualizuje istniejące dane
`@DeleteMapping`	`@RequestMapping(method = RequestMethod.DELETE)`	Usuwa dane

REST – prosty przykład

```
@RestController
public class CustomerController {

    @Autowired
    private CustomerService customerService;
    @Autowired
    private CustomerAssembler customerAssembler;

    @GetMapping(value = "/{id}")
    public Customer findCustomerById(@PathVariable Long id) {
        return customerService.findById(id);
    }

    @PostMapping(value="/")
    public Customer create(@RequestBody Customer customer) {
        return customerService.save(customer);
    }

    @PutMapping(value="/{id}")
    public void update(@RequestBody Customer updateData, @PathVariable Long id)
        throws CustomerNotFoundException {
        Customer customer = customerService.findById(id);
        if(customer!=null){
            Customer updateCustomer = customerAssembler.applyUpdate(customer, updateData);
            customerService.save(updateCustomer);
        }
        else{
            throw new CustomerNotFoundException(id);
        }
    }

    @DeleteMapping(value="/{id}")
    public void delete(@PathVariable Long id) {
        customerService.delete(customerService.findById(id));
    }
}
```

```
public class Customer {
    private final long id;
    private final String name;

    public Customer(long id, String name) {
        this.id = id;
        this.name = name;
    }

    public long getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```

REST @ControllerAdvice

```
@ControllerAdvice
public class RestExceptionHandler extends ResponseEntityExceptionHandler {

    private static final Logger LOG = LoggerFactory.getLogger(RestExceptionHandler.class);

    @ExceptionHandler(Exception.class)
    protected ResponseEntity<Object> handleAllExceptions(Exception ex) {
        LOG.errorIncludingThrowable(ex);
        return buildResponseEntity(HttpStatus.INTERNAL_SERVER_ERROR, "Internal server error.");
    }

    @ExceptionHandler(ValidationException.class)
    protected ResponseEntity<Object> handleValidationExceptions(ValidationException ex) {
        LOG.info(ex);
        return buildResponseEntity(HttpStatus.BAD_REQUEST, "Validation error");
    }

    private ResponseEntity<Object> buildResponseEntity(HttpStatus httpStatus, String error) {
        ApiError apiError = new ApiError(httpStatus, error);
        return new ResponseEntity<>(apiError, apiError.getStatus());
    }

    @ResponseBody
    @ExceptionHandler(CustomerNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    protected ResponseEntity<Object> handleCustomerNotFoundExceptions(CustomerNotFoundException ex) {
        LOG.info(ex);
        return "Customer not found";
    }
}
```

REST Hateoas

I am getting frustrated by the number of people calling any HTTP-based interface a REST API. That is RPC (Remote Procedure Call). (...) In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API.

— Roy Fielding

Sam URL typu /customers/1 nie jest RESTem

Samo używanie GET, POST, etc. nie jest RESTem

Posiadanie CRUDowych operacji w Controllerze nie jest RESTem

REST Hateoas – przykład

```
@GetMapping("/customers/{id}")
EntityModel<Customer> findById(@PathVariable Long id) {

  Customer customer = repository.findById(id)
      .orElseThrow(() -> new CustomerNotFoundException(id));

  return EntityModel.of(customer,
      linkTo(methodOn(CustomerController.class).findById(id)).withSelfRel(),
      linkTo(methodOn(DiscountController.class).findByCystomerId(id)).withRel("discounts"));
}
```

```
{
  "id": 1003,
  "name": "John Black",
  "status": "premium",
  "_links": {
    "self": {
      "href": "http://yourhost:8080/customers/1003"
    },
    "discounts": {
      "href": "http://yourhost:8080/discounts/1003"
    }
  }
}
```

REST – Hateoas i paging


```
{
   "rows":    [
   {
        "_links": {"self": {"href": "http://yourhost:8080/customers/1000200"}},
        "content":          {
           "id": "1000200",
           "name": "Jan Kowalski",
           ...
        },
        ...,
           {
        "_links": {"self": {"href": "http://yourhost:8080/customers/1000210"}},
        "content":          {
           "id": "1000210",
           "name": "Swiat Ksiazki",
           ...
        }
     }
   ],
   "_links":    {
     "self": {"href": "http://yourhost:8080/customers?pageSize=10"},
     "first": {"href": "http://yourhost:8080/customers?pageSize=10"},
     "previous": {"href": null},
     "next": {"href": "http://yourhost:8080/customers?pageSize=10&pageRecordFilter=customerId=1000210&pageAction=next"},
     "last": {"href": "http://yourhost:8080/customers?pageSize=10&pageAction=last"}
   }
}
```

REST + Swagger

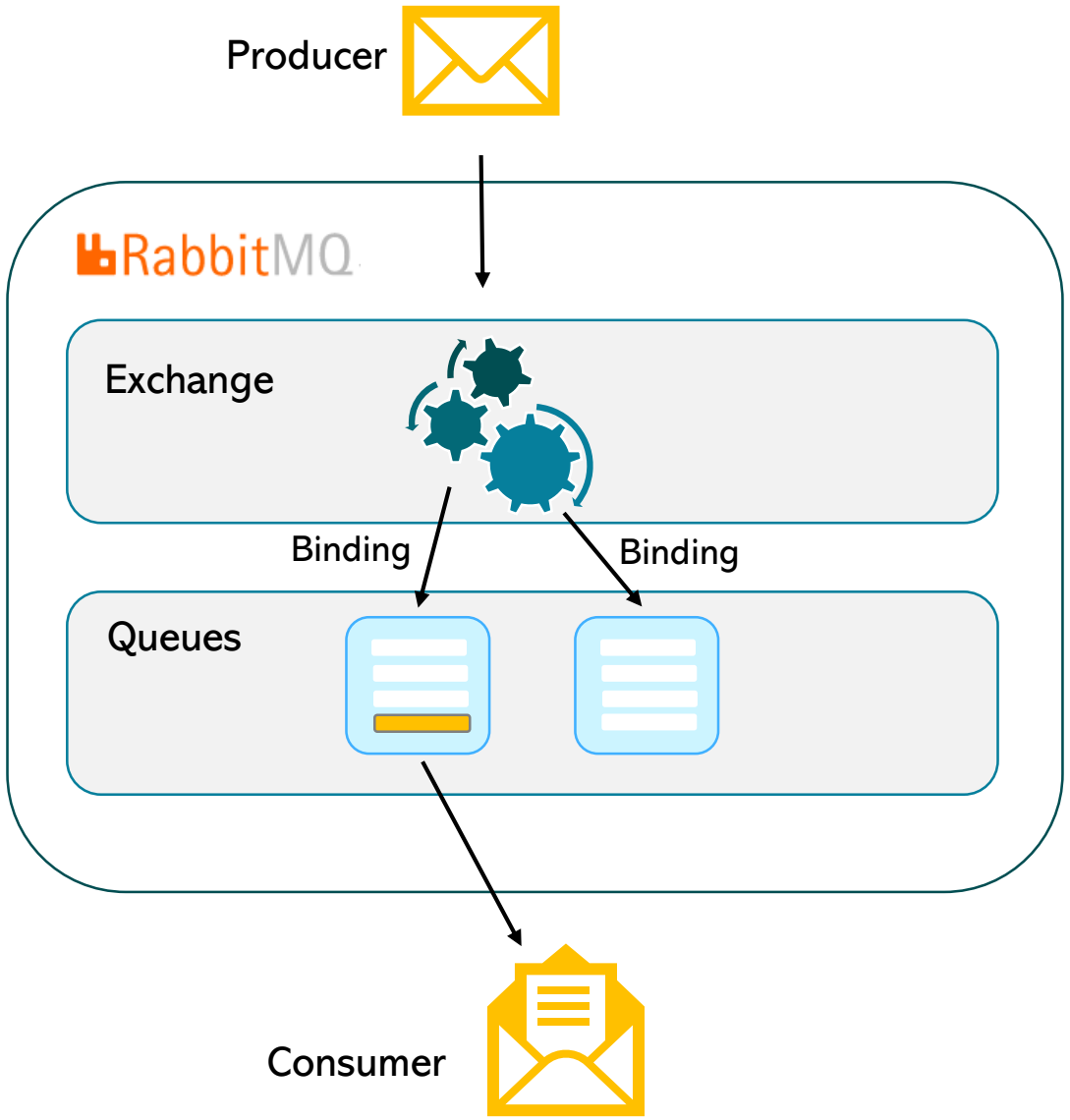
Supported by SMARTBEAR

```java
@RequestMapping(path = "/documents")
@Api(tags = { "documents" }, produces = "application/pdf", consumes = "application/json")
public interface DocumentRestController {
    @Nonnull
    @RequestMapping(path = "/document/{archiveReference}", method = RequestMethod.GET, produces = "application/pdf")
    @ApiOperation(value = "", notes = "Retrieves Quotation's Document content based on archiving reference passed as a param")
    public byte[] getDocument(
            @Nonnull @PathVariable(value = "archiveReference")
            @ApiParam(value = "Unique identifier set to Document") String archiveReference)
            throws DocumentNotFoundException, TechnicalErrorException;
```

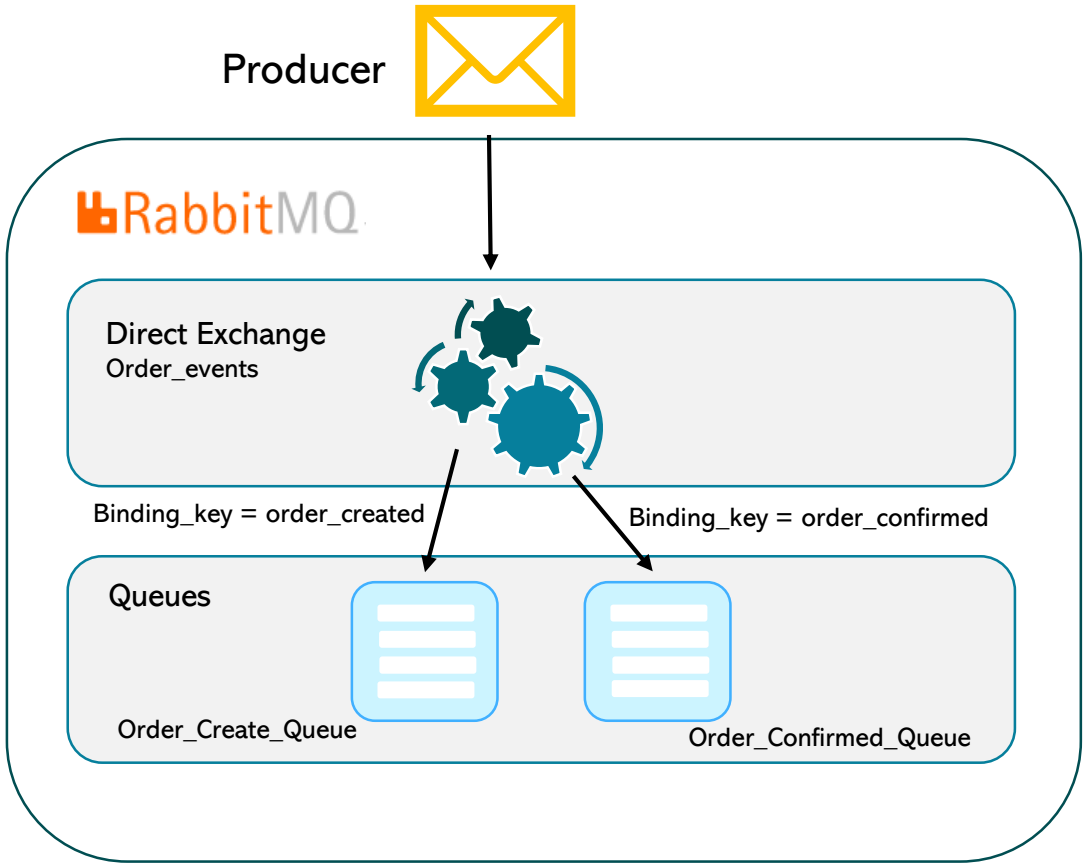
```json
"/documents/document/{archiveReference}" : {
  "get" : {
    "tags" : [ "documents" ],
    "summary" : "",
    "description" : "Retrieves Quotation's Document content based on archiving reference passed as a param",
    "operationId" : "getDocument",
    "produces" : [ "application/pdf" ],
    "parameters" : [ {
      "name" : "archiveReference",
      "in" : "path",
      "description" : "Unique identifier set to Document",
      "required" : true,
      "type" : "string"
    } ],
```

```json
"responses" : {
  "200" : {
    "description" : "successful operation",
    "schema" : {
      "type" : "array",
      "items" : {
        "type" : "string",
        "format" : "byte"
      }
    }
  },
  "404" : {
    "description" : "Document not found"
  }
}
```

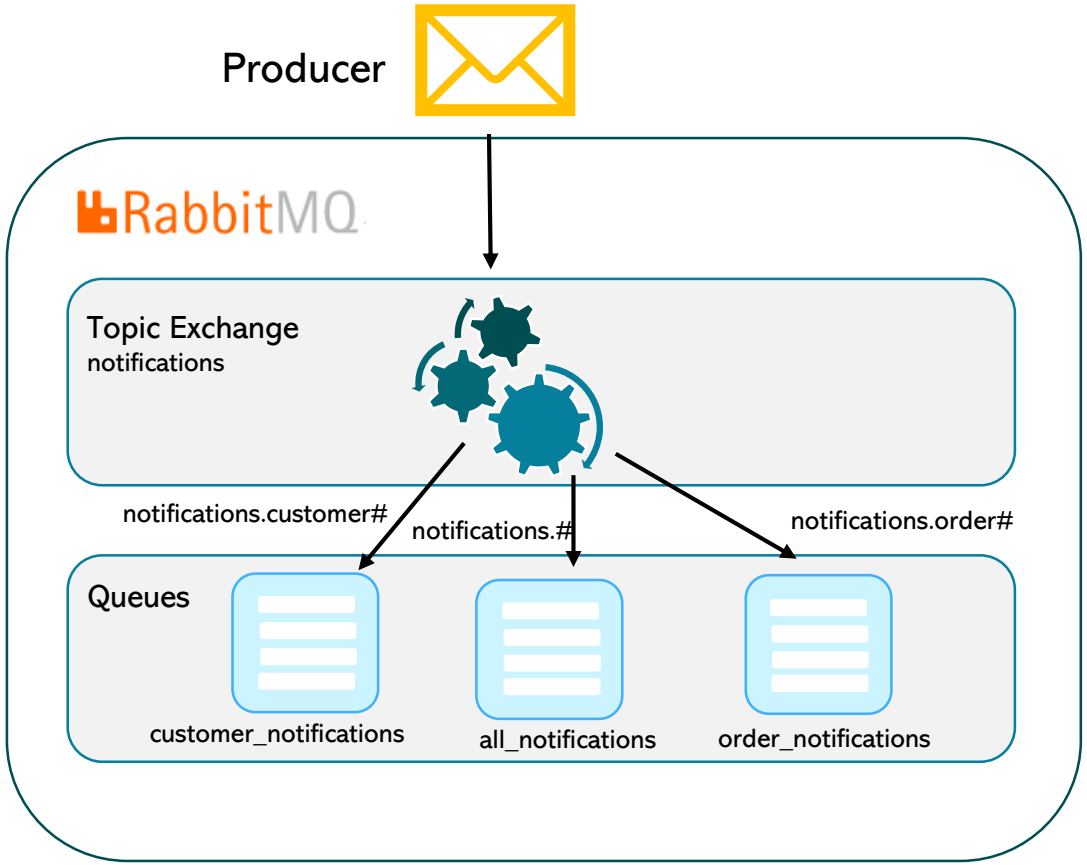
Rabbit MQ


Producer
RabbitMQ
Exchange
Binding
Binding
Queues
Consumer

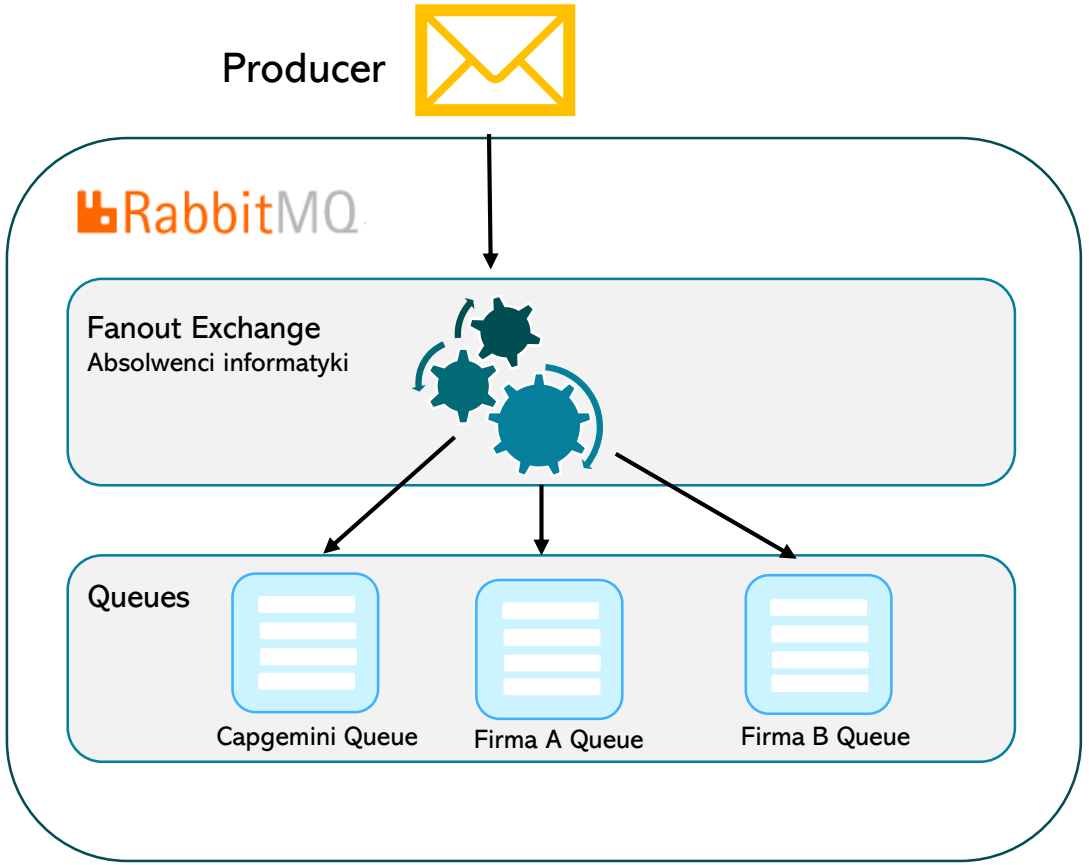
Rabbit MQ – Direct Exchange



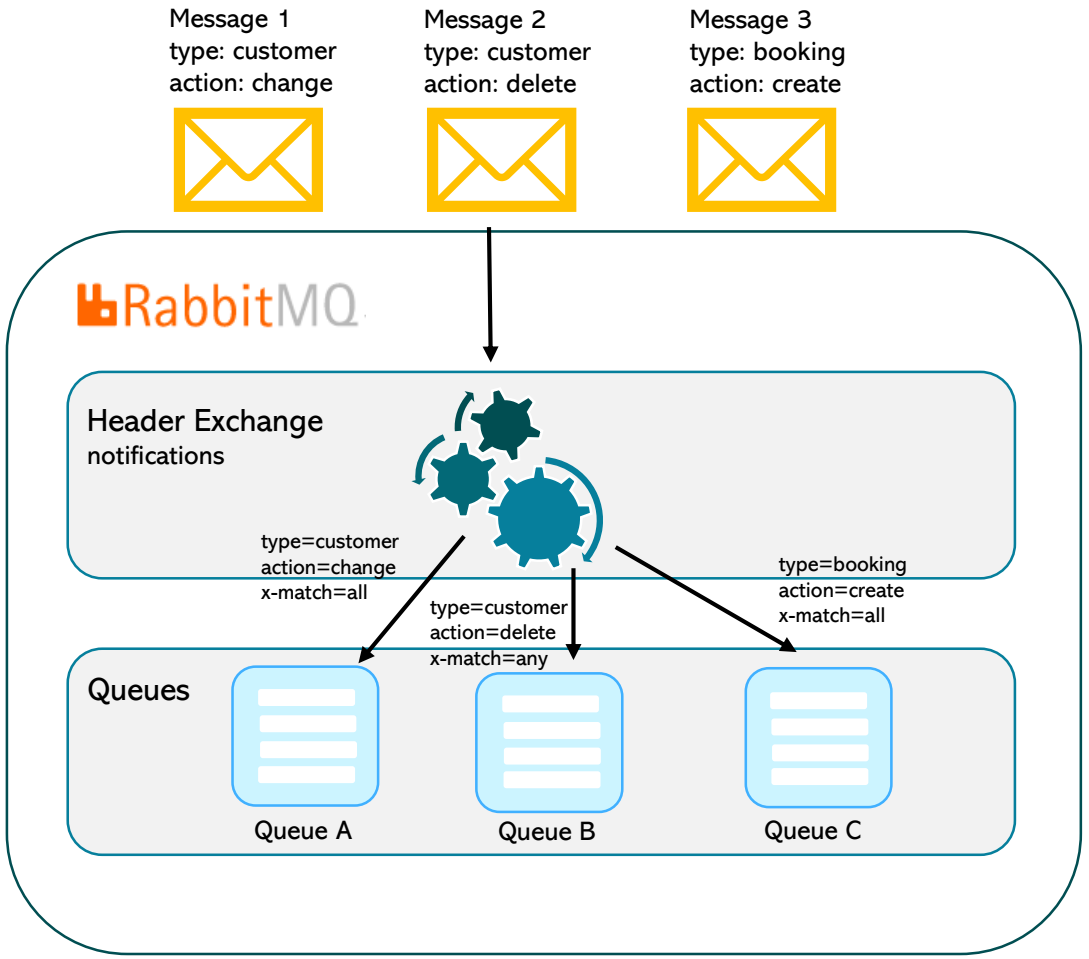
Rabbit MQ – Topic Exchange



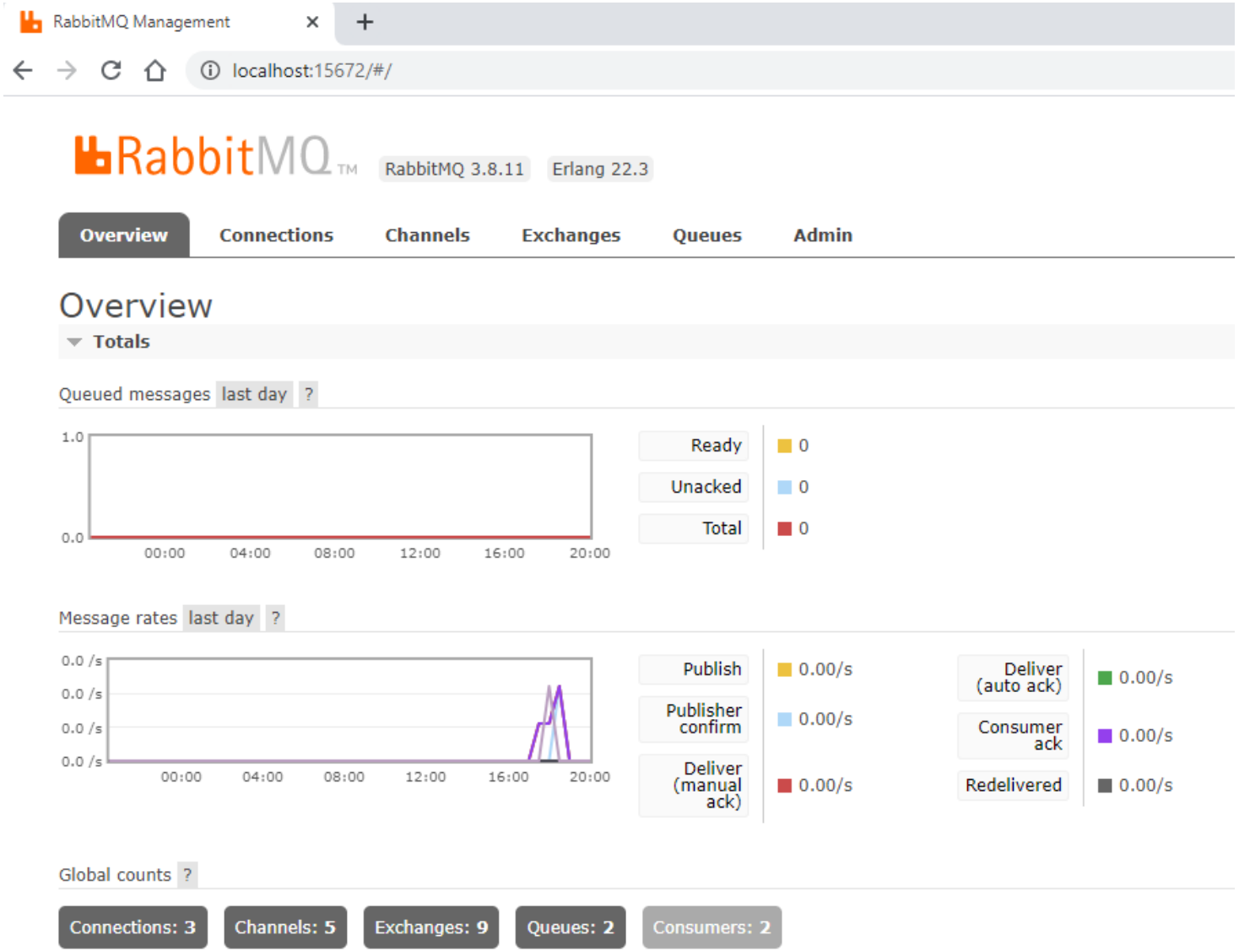
Rabbit MQ – Fanout Exchange



Rabbit MQ – Headers Exchange



Rabbit MQ

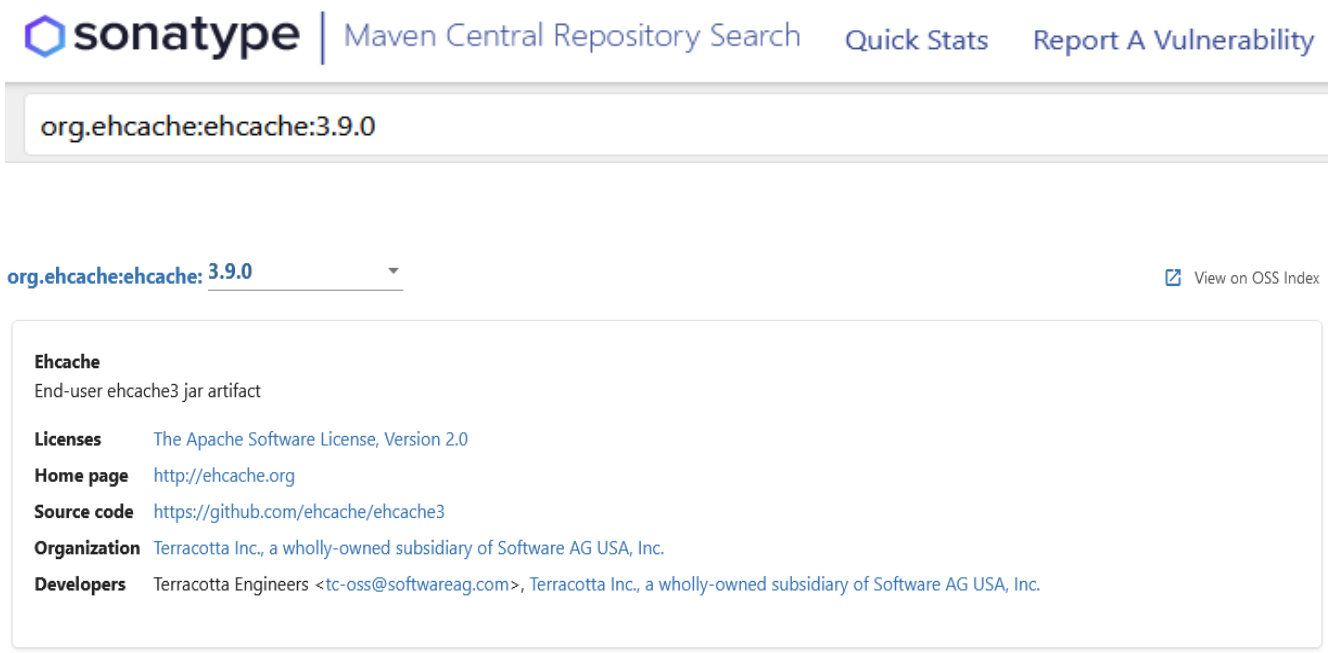

RabbitMQ Management
localhost:15672/#/
RabbitMQ
RabbitMQ 3.8.11
Erlang 22.3
Overview
Connections
Channels
Exchanges
Queues
Admin
Overview
Totals
Queued messages last day ?
1.0
0.0
00:00
04:00
08:00
12:00
16:00
20:00
Ready 0
Unacked 0
Total 0
Message rates last day ?
0.0 /s
0.0 /s
0.0 /s
0.0 /s
00:00
04:00
08:00
12:00
16:00
20:00
Publish 0.00/s
Publisher confirm 0.00/s
Deliver (manual ack) 0.00/s
Deliver (auto ack) 0.00/s
Consumer ack 0.00/s
Redelivered 0.00/s
Global counts ?
Connections: 3
Channels: 5
Exchanges: 9
Queues: 2
Consumers: 2

Czym jest Maven

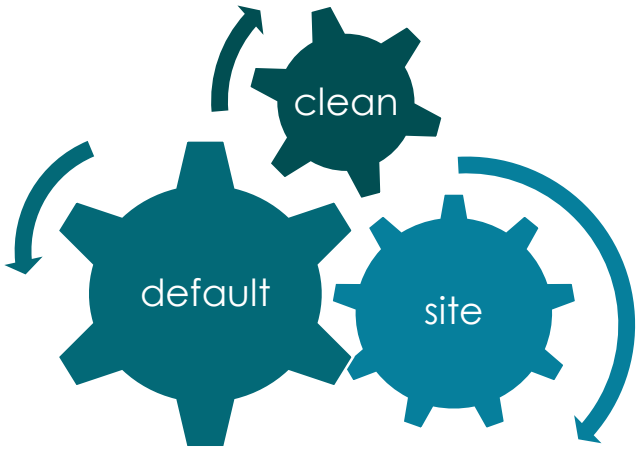
- Budowanie projektu
- Dokumentacja
- Zależności
- Wydania
- Dystrybucja

- Czytelność/widoczność
- Reużywalność
- Łatwe utrzymanie (maintenance)
- Zrozumiałość

Maven – repozytorium i zależności



Maven – cykle życia, fazy i cele

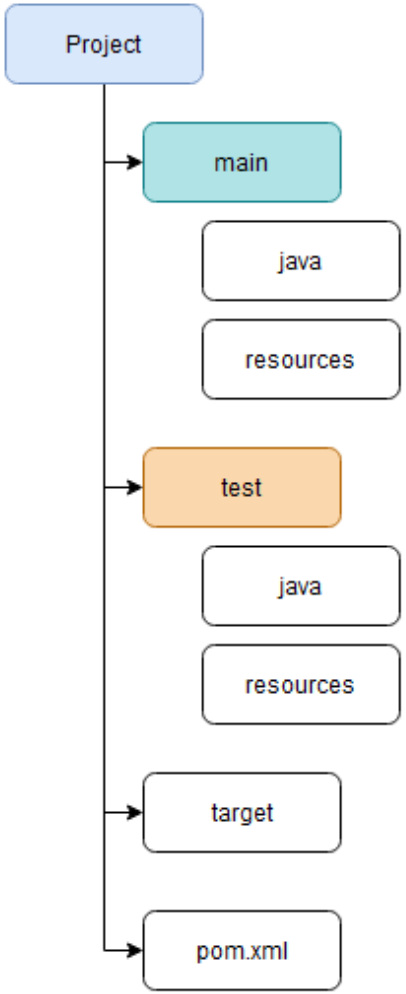


default – fazy (phases)	plugin:goal
validate	
compile	compiler:compile
test-compile	compiler:testCompile
test	surefire:test
package	jar:jar, rar:rar, war:war
verify	
install	install:install
deploy	deploy:deploy

```
<plugin>
    <artifactId>maven-release-plugin</artifactId>
    <version>${maven.release.version}</version>
    <executions>
        <execution>
            <goals>
                <goal>clean</goal>
                <goal>prepare</goal>
                <goal>prepare-with-pom</goal>
                <goal>rollback</goal>
                <goal>perform</goal>
                <goal>stage</goal>
                <goal>branch</goal>
                <goal>update-versions</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

mvn clean dependency:copy-dependencies package

Maven. Struktura projektu



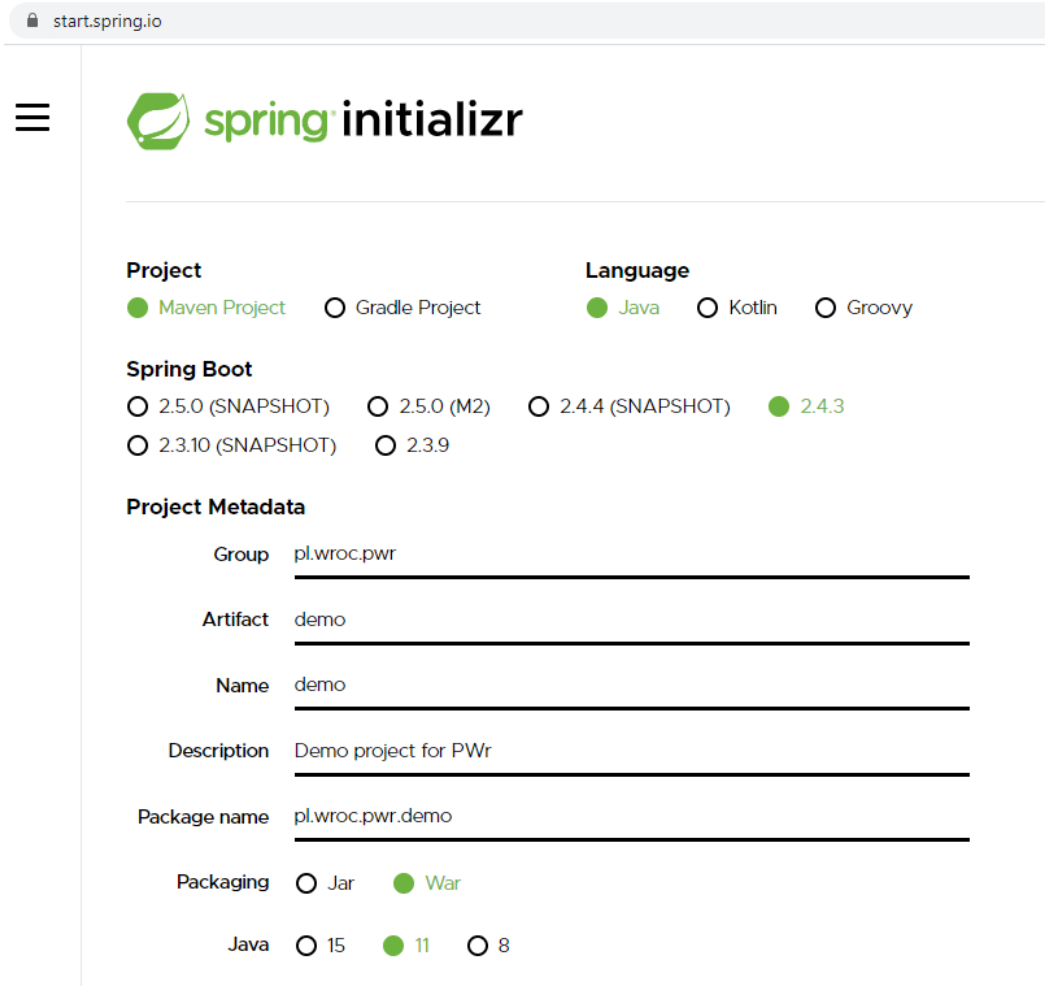

```
mvn archetype:generate
-DgroupId=pl.wroc.pwr
-DartifactId=demo
-DarchetypeArtifactId=maven-archetype-quickstart
-DarchetypeVersion=1.4
-DinteractiveMode=false
```

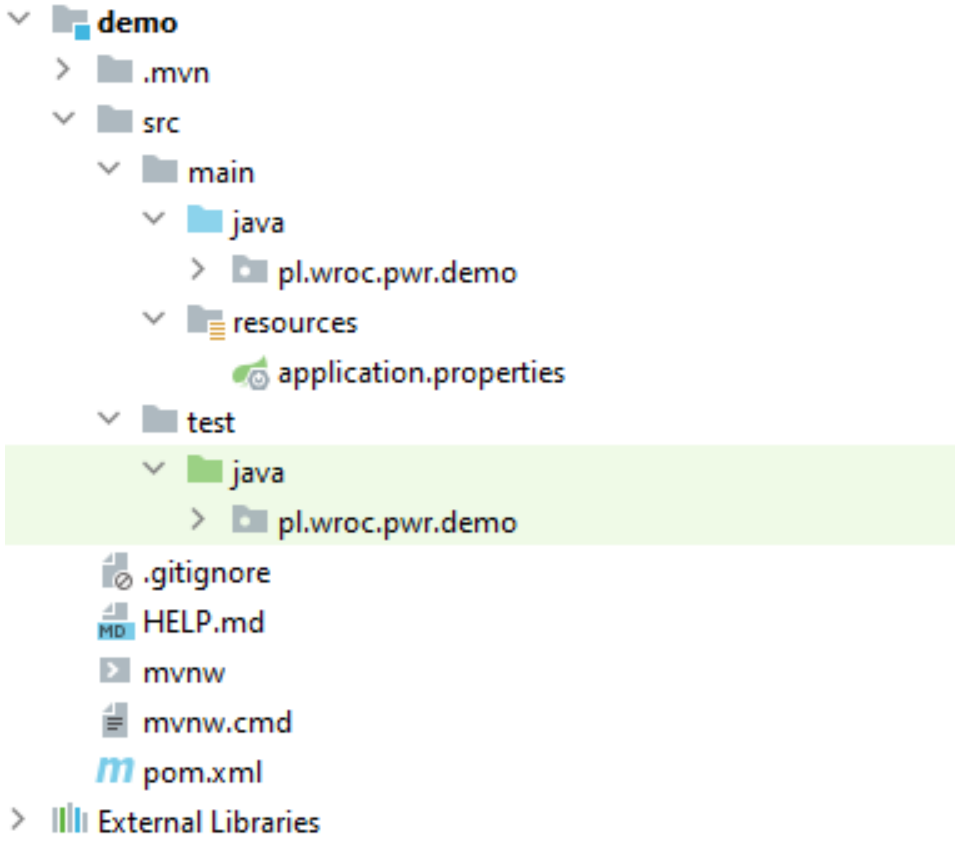
pom.xml

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.2</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>

<groupId>com.marcin</groupId>
<artifactId>demo</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>demo</name>
<description>Demo project for Spring Boot</description>
<properties>
    <java.version>11</java.version>
</properties>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Spring boot


start.spring.io
spring initializr
Project
Maven Project
Gradle Project
Language
Java
Kotlin
Groovy
Spring Boot
2.5.0 (SNAPSHOT)
2.5.0 (M2)
2.4.4 (SNAPSHOT)
2.4.3
2.3.10 (SNAPSHOT)
2.3.9
Project Metadata
Group
pl.wroc.pwr
Artifact
demo
Name
demo
Description
Demo project for PWr
Package name
pl.wroc.pwr.demo
Packaging
Jar
War
Java
15
11
8


demo
.mvn
src
main
java
pl.wroc.pwr.demo
resources
application.properties
test
java
pl.wroc.pwr.demo
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries

Dziękuję za uwagę