



View, add and edit your notes in the app

Spring Boot Microservices Project Example - Part 1 | Building Services

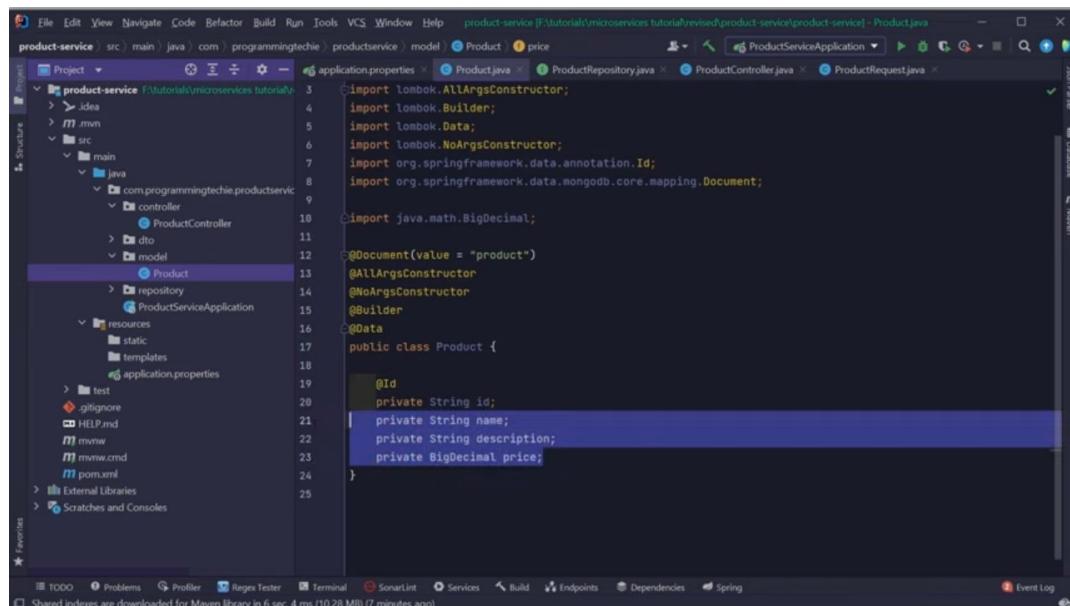
Generated on July 24, 2023

Summary

Notes

Screenshots

1



▶ 11:00

A screenshot of a software interface with a dark header bar. The header contains three tabs: "Dependencies", "Spring", and "Event Log". The "Event Log" tab is currently selected, indicated by a red arrow pointing to it. The main area of the window is heavily redacted with a large, hand-drawn style red mark.

The screenshot shows the IntelliJ IDEA interface with the project 'product-service' open. The code editor displays the `ProductService.java` file, which contains the implementation for creating a product. The code uses Lombok annotations like `@Service`, `@RequiredArgsConstructor`, and `@Slf4j`. It imports `Product` from the `com.programmingtechie.productservice.model` package and `ProductRepository` from the `com.programmingtechie.productservice.repository` package. The `createProduct` method uses a `Product.builder()` to create a new `Product` object with the provided name, description, and price, then saves it to the repository and logs the success message.

```
import com.programmingtechie.productservice.model.Product;
import com.programmingtechie.productservice.repository.ProductRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
@Slf4j
public class ProductService {

    private final ProductRepository productRepository;

    public void createProduct(ProductRequest productRequest) {
        Product product = Product.builder()
            .name(productRequest.getName())
            .description(productRequest.getDescription())
            .price(productRequest.getPrice())
            .build();

        productRepository.save(product);
        log.info("Product {} is saved", product.getId());
    }
}
```

▶ 15:18

The screenshot shows the IntelliJ IDEA interface with the project 'product-service' open. The code editor displays the `ProductController.java` file, which contains the `createProduct` endpoint. The controller uses `@RestController` and `@RequestMapping` annotations to map the `/api/product` endpoint. It injects the `ProductService` dependency and handles the incoming `ProductRequest` body to call the `createProduct` method on the service.

```
package com.programmingtechie.productservice.controller;

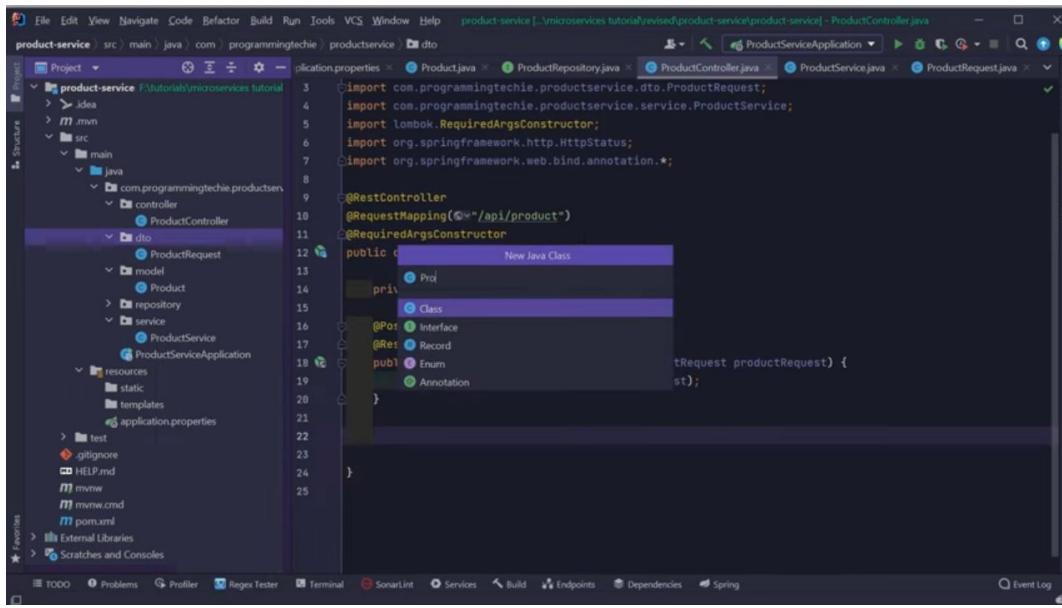
import com.programmingtechie.productservice.dto.ProductRequest;
import com.programmingtechie.productservice.service.ProductService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/product")
@RequiredArgsConstructor
public class ProductController {

    private final ProductService productService;

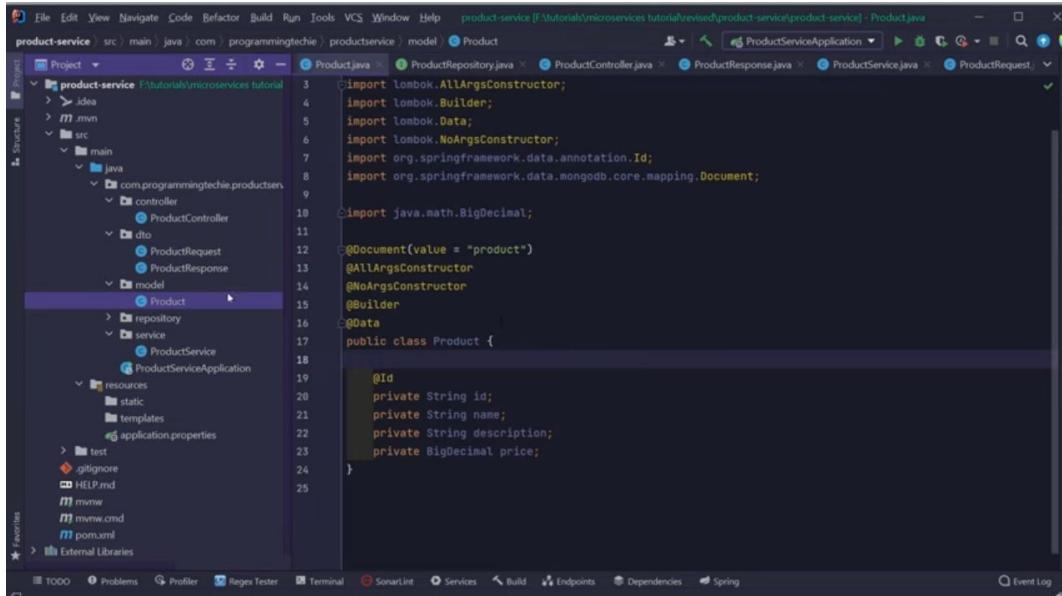
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public void createProduct(@RequestBody ProductRequest productRequest) {
        productService.createProduct(productRequest);
    }
}
```

▶ 16:08



```
File Edit View Navigate Code Behavior Build Run Tools VCS Window Help product-service [~/microservices tutorial/revised/product-service] - ProductController.java
product-service src main java com programmingtechie productservice dto
application.properties ProductRepository.java ProductServiceApplication ProductController.java ProductService.java ProductRequest.java
3 import com.programmingtechie.productservice.dto.ProductRequest;
4 import com.programmingtechie.productservice.service.ProductService;
5 import lombok.RequiredArgsConstructor;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.web.bind.annotation.*;
8
9 @RestController
10 @RequestMapping(value = "/api/product")
11 @RequiredArgsConstructor
12 public class ProductController {
13     private final ProductService productService;
14
15     @PostMapping
16     public ProductResponse createProduct(@RequestBody ProductRequest productRequest) {
17         return productService.createProduct(productRequest);
18     }
19
20 }
21
22 }
23
24 }
```

▶ 16:19



```
File Edit View Navigate Code Behavior Build Run Tools VCS Window Help product-service [~/microservices tutorial/revised/product-service] - Product.java
product-service src main java com programmingtechie productservice model
Product.java ProductRepository.java ProductController.java ProductResponse.java ProductService.java ProductRequest.java
3 import lombok.AllArgsConstructor;
4 import lombok.Builder;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7 import org.springframework.data.annotation.Id;
8 import org.springframework.data.mongodb.core.mapping.Document;
9
10 import java.math.BigDecimal;
11
12 @Document(value = "product")
13 @AllArgsConstructor
14 @NoArgsConstructor
15 @Builder
16 @Data
17 public class Product {
18
19     @Id
20     private String id;
21     private String name;
22     private String description;
23     private BigDecimal price;
24 }
```

▶ 17:07

The screenshot shows the IntelliJ IDEA interface with the code editor open to the `ProductController.java` file. The code defines a REST controller for products, using `@RestController`, `@PostMapping`, and `@GetMapping` annotations. It interacts with a `ProductService` and a `ProductRepository`. The code is well-formatted with color-coded syntax highlighting.

```
4 import com.intellij.psi.PsiMethod;
5 import com.intellij.psi.PsiType;
6 import com.intellij.psi.util.PsiUtil;
7 import com.intellij.psi.util.PsiUtilCore;
8 import com.intellij.psi.util.PsiUtilPsiUtil;
9 import com.intellij.psi.util.PsiUtilPsiUtil;
10 import com.intellij.psi.util.PsiUtilPsiUtil;
11 import com.intellij.psi.util.PsiUtilPsiUtil;
12 import com.intellij.psi.util.PsiUtilPsiUtil;
13 import com.intellij.psi.util.PsiUtilPsiUtil;
14 import com.intellij.psi.util.PsiUtilPsiUtil;
15 import com.intellij.psi.util.PsiUtilPsiUtil;
16 import com.intellij.psi.util.PsiUtilPsiUtil;
17 import com.intellij.psi.util.PsiUtilPsiUtil;
18 import com.intellij.psi.util.PsiUtilPsiUtil;
19 import com.intellij.psi.util.PsiUtilPsiUtil;
20 import com.intellij.psi.util.PsiUtilPsiUtil;
21 import com.intellij.psi.util.PsiUtilPsiUtil;
22 import com.intellij.psi.util.PsiUtilPsiUtil;
23 import com.intellij.psi.util.PsiUtilPsiUtil;
24 import com.intellij.psi.util.PsiUtilPsiUtil;
25 import com.intellij.psi.util.PsiUtilPsiUtil;
26 import com.intellij.psi.util.PsiUtilPsiUtil;
27 import com.intellij.psi.util.PsiUtilPsiUtil;
28 import com.intellij.psi.util.PsiUtilPsiUtil;
29 import com.intellij.psi.util.PsiUtilPsiUtil;
30 import com.intellij.psi.util.PsiUtilPsiUtil;
```

▶ 18:17

The screenshot shows the IntelliJ IDEA interface with the code editor open to the `ProductService.java` file. The code defines a service layer that interacts with a `ProductRepository` to create and get products. It uses `@Service`, `@RequiredArgsConstructor`, and `@Slf4j` annotations. The code includes logging for product creation.

```
4 import com.intellij.psi.PsiMethod;
5 import com.intellij.psi.PsiType;
6 import com.intellij.psi.util.PsiUtil;
7 import com.intellij.psi.util.PsiUtilCore;
8 import com.intellij.psi.util.PsiUtilPsiUtil;
9 import com.intellij.psi.util.PsiUtilPsiUtil;
10 import com.intellij.psi.util.PsiUtilPsiUtil;
11 import com.intellij.psi.util.PsiUtilPsiUtil;
12 import com.intellij.psi.util.PsiUtilPsiUtil;
13 import com.intellij.psi.util.PsiUtilPsiUtil;
14 import com.intellij.psi.util.PsiUtilPsiUtil;
15 import com.intellij.psi.util.PsiUtilPsiUtil;
16 import com.intellij.psi.util.PsiUtilPsiUtil;
17 import com.intellij.psi.util.PsiUtilPsiUtil;
18 import com.intellij.psi.util.PsiUtilPsiUtil;
19 import com.intellij.psi.util.PsiUtilPsiUtil;
20 import com.intellij.psi.util.PsiUtilPsiUtil;
21 import com.intellij.psi.util.PsiUtilPsiUtil;
22 import com.intellij.psi.util.PsiUtilPsiUtil;
23 import com.intellij.psi.util.PsiUtilPsiUtil;
24 import com.intellij.psi.util.PsiUtilPsiUtil;
25 import com.intellij.psi.util.PsiUtilPsiUtil;
26 import com.intellij.psi.util.PsiUtilPsiUtil;
27 import com.intellij.psi.util.PsiUtilPsiUtil;
28 import com.intellij.psi.util.PsiUtilPsiUtil;
29 import com.intellij.psi.util.PsiUtilPsiUtil;
30 import com.intellij.psi.util.PsiUtilPsiUtil;
```

▶ 18:40

The screenshot shows the IntelliJ IDEA interface with the project 'product-service' open. The code editor displays the `ProductService.java` file. The code implements a `ProductService` interface with two methods: `createProduct` and `getAllProducts`. The `createProduct` method uses a `Product.builder()` to create a `Product` object and then saves it to the `productRepository`. The `getAllProducts` method retrieves all products from the repository and maps them to `ProductResponse` objects.

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class ProductService {
    private final ProductRepository productRepository;

    public void createProduct(ProductRequest productRequest) {
        Product product = Product.builder()
            .name(productRequest.getName())
            .description(productRequest.getDescription())
            .price(productRequest.getPrice())
            .build();

        productRepository.save(product);
        log.info("Product {} is saved", product.getId());
    }

    public List<ProductResponse> getAllProducts() {
        List<Product> all = productRepository.findAll();
        return all.stream().map(product -> mapToProductResponse()).collect(Collectors.toList());
    }

    private ProductResponse mapToProductResponse() {
        return null;
    }
}
```

▶ 19:07

This screenshot shows the same `ProductService.java` file, but with a different implementation. The `createProduct` method now uses a `mapToProductResponse` private method to map the `Product` object to a `ProductResponse` object before saving it to the repository. The `getAllProducts` method remains the same, retrieving all products and mapping them to `ProductResponse` objects.

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class ProductService {
    private final ProductRepository productRepository;

    public void createProduct(ProductRequest productRequest) {
        Product product = Product.builder()
            .name(productRequest.getName())
            .description(productRequest.getDescription())
            .price(productRequest.getPrice())
            .build();

        productRepository.save(product);
        log.info("Product {} is saved", product.getId());
    }

    public List<ProductResponse> getAllProducts() {
        List<Product> products = productRepository.findAll();
        return products.stream().map(product -> mapToProductResponse()).collect(Collectors.toList());
    }

    private ProductResponse mapToProductResponse() {
        return null;
    }
}
```

▶ 19:52

The screenshot shows an IDE interface with a Java project named "product-service". The code editor displays the `ProductService.java` file. The code implements a `createProduct` method that uses a `Product.builder()` to create a `Product` object with name, description, and price, then saves it to a `productRepository`. It also logs the product ID. The `getAllProducts` method retrieves all products from the repository and maps them to `ProductResponse` objects. A `mapToProductResponse` private method is used to convert `Product` objects to `ProductResponse` objects using a builder pattern.

```
15  public class ProductService {  
16      private final ProductRepository productRepository;  
17  
18      @Override  
19      public void createProduct(ProductRequest productRequest) {  
20          Product product = Product.builder()  
21              .name(productRequest.getName())  
22              .description(productRequest.getDescription())  
23              .price(productRequest.getPrice())  
24              .build();  
25  
26          productRepository.save(product);  
27          log.info("Product {} is saved", product.getId());  
28      }  
29  
30      @Override  
31      public List<ProductResponse> getAllProducts() {  
32          List<Product> products = productRepository.findAll();  
33  
34          products.stream().map(product -> mapToProductResponse());  
35      }  
36  
37      private ProductResponse mapToProductResponse() {  
38          return ProductResponse.  
39              builder()  
40              .id(String id)  
41              .description(String description)  
42              .name(String name)  
43              .price(BigDecimal price)  
44          .build();  
45      }  
46  }
```

▶ 19:56

This screenshot shows the same Java project and code editor as the previous one, but the code editor now highlights the `ProductResponse.builder()` call in the `mapToProductResponse` method. A tooltip or code completion dropdown is visible, showing options like `.id(String id)`, `.description(String description)`, `.name(String name)`, and `.price(BigDecimal price)`.

```
15  public class ProductService {  
16      private final ProductRepository productRepository;  
17  
18      @Override  
19      public void createProduct(ProductRequest productRequest) {  
20          Product product = Product.builder()  
21              .name(productRequest.getName())  
22              .description(productRequest.getDescription())  
23              .price(productRequest.getPrice())  
24              .build();  
25  
26          productRepository.save(product);  
27          log.info("Product {} is saved", product.getId());  
28      }  
29  
30      @Override  
31      public List<ProductResponse> getAllProducts() {  
32          List<Product> products = productRepository.findAll();  
33  
34          products.stream().map(product -> mapToProductResponse());  
35      }  
36  
37      private ProductResponse mapToProductResponse() {  
38          return ProductResponse.  
39              builder()  
40              .id(String id)  
41              .description(String description)  
42              .name(String name)  
43              .price(BigDecimal price)  
44          .build();  
45      }  
46  }
```

▶ 20:04

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `ProductService.java` file. The code implements a `createProduct` method that takes a `ProductRequest` object, creates a `Product` object using a builder pattern, saves it to a repository, and logs the save operation. It also implements a `getAllProducts` method that retrieves all products from the repository and maps them to `ProductResponse` objects using a `mapToProductResponse` method.

```
15  ProductService.java
16  public class ProductService {
17
18      private final ProductRepository productRepository;
19
20      @Override
21      public void createProduct(ProductRequest productRequest) {
22          Product product = Product.builder()
23              .name(productRequest.getName())
24              .description(productRequest.getDescription())
25              .price(productRequest.getPrice())
26              .build();
27
28          productRepository.save(product);
29          log.info("Product {} is saved", product.getId());
30      }
31
32      @Override
33      public List<ProductResponse> getAllProducts() {
34          List<Product> products = productRepository.findAll();
35
36          products.stream().map(product -> mapToProductResponse(product));
37      }
38
39      private ProductResponse mapToProductResponse(Product product) {
40          return ProductResponse.builder()
41              .id(product.getId())
42              .name(product.getName())
43              .build();
44      }
45  }
```

▶ 20:21

The screenshot shows the same IntelliJ IDEA interface as the previous one, but with a cursor placed on the `.name(product.getName())` line within the `mapToProductResponse` method. This indicates that the developer is currently working on or reviewing this specific line of code.

```
15  ProductService.java
16  public class ProductService {
17
18      private final ProductRepository productRepository;
19
20      @Override
21      public void createProduct(ProductRequest productRequest) {
22          Product product = Product.builder()
23              .name(productRequest.getName())
24              .description(productRequest.getDescription())
25              .price(productRequest.getPrice())
26              .build();
27
28          productRepository.save(product);
29          log.info("Product {} is saved", product.getId());
30      }
31
32      @Override
33      public List<ProductResponse> getAllProducts() {
34          List<Product> products = productRepository.findAll();
35
36          products.stream().map(product -> mapToProductResponse(product));
37      }
38
39      private ProductResponse mapToProductResponse(Product product) {
40          return ProductResponse.builder()
41              .id(product.getId())
42              .name(product.getName())
43              .build();
44      }
45  }
```

▶ 20:36

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `ProductController.java` file. The cursor is positioned at line 37, where the `mapToProductResponse` method is being called. The code is as follows:

```
private ProductResponse mapToProductResponse(Producer product) {
    return ProductResponse.builder()
        .id(product.getId())
        .name(product.getName())
        .description(product.getDescription())
        .price(product.getPrice())
        .build();
}
```

▶ 20:54

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `ProductController.java` file. The cursor is positioned at line 37, where the `mapToProductResponse` method is being called. A tooltip is visible over the method name, showing its signature: `private ProductResponse mapToProductResponse(Producer product)`. The code is as follows:

```
private ProductResponse mapToProductResponse(Producer product) {
    return ProductResponse.builder()
        .id(product.getId())
        .name(product.getName())
        .description(product.getDescription())
        .price(product.getPrice())
        .build();
}
```

▶ 21:15

The screenshot shows the IntelliJ IDEA interface with the project 'product-service' open. The code editor displays the `ProductService.java` file, which contains the following code:

```
        .description(productRequest.getDescription())
        .price(productRequest.getPrice())
        .build();

    productRepository.save(product);
    log.info("Product {} is saved", product.getId());
}

public List<ProductResponse> getAllProducts() {
    List<Product> products = productRepository.findAll();

    return products.stream().map(this::mapToProductResponse).toList();
}

private ProductResponse mapToProductResponse(Product product) {
    return record(
        productRepository.findById(product.getId()),
        mapToProductResponse(product),
        createProduct(productRequest));
}

private void mapToProductResponse(Product product, ProductResponse productResponse) {
    productResponse
        .description(product.getDescription())
        .price(product.getPrice())
        .build();
}
```

▶ 21:27

The screenshot shows the Postman application interface. A new collection named 'Scratch Pad' is selected. A POST request is being prepared to the URL `http://localhost:8080/api/product`. The 'Body' tab is selected, showing the following JSON payload:

```
{
  "key": "Value"
}
```

The 'Send' button is highlighted in blue at the top right of the request configuration area.

▶ 22:19

Postman

File Edit View Help

Home Workspaces Reports Explore

Search Postman

Sign In Create Account

Working locally in Scratch Pad. Switch to a Workspace

No Environment

POST http://localhost:8080/api/product

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Beautify

Body (9)

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "name": "Iphone 13",
3   "description": "Iphone 13",
4   "price": 1200
5 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize Text

Status: 201 Created Time: 172 ms Size: 128 B Save Response

1

23:06

File Edit View Navigate Code Behavior Build Run Tools VCS Window Help product-service

product-service src test

Project Structure

product-service P:\tutorials\microservices\tutorial\revise

src

- idea
- mvn
- src
 - main
 - test
 - java
 - com.programmingtechie.productservice
- target
- .gitignore
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml
- product-service.iml
- External Libraries
- Scratches and Consoles

Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open them

Run TODO Problems Profiler Regex Tester Terminal SonarLint Services Endpoints Build Dependencies Spring Event Log

Tests passed: 2 (11 minutes ago)

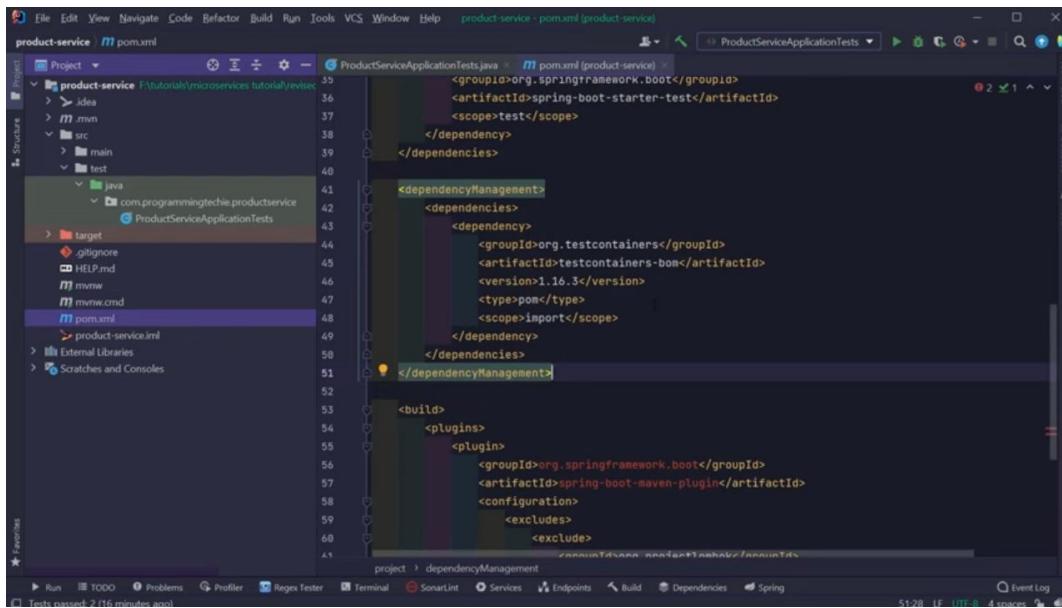
23:38

The screenshot shows the Testcontainers website at testcontainers.org. The page title is "About". The sidebar on the left lists "Testcontainers" sections: Home, Quickstart, Features, Modules, Test framework integration, System Requirements, Getting help, and Contributing. The main content area contains the Testcontainers logo, a search bar, and a table of contents on the right. Handwritten red annotations include a large arrow pointing to the "About" section with the word "open" written above it, and the words "Junit" and "Containers" written diagonally across the page.

▶ 24:28

The screenshot shows the "MongoDB Module" page on the Testcontainers website. The page title is "Usage example". The sidebar on the left lists "Testcontainers" sections: Home, Quickstart, Features, Modules (with "Databases" expanded), Database containers, JDBC support, R2DBC support, Cassandra Module, CockroachDB Module, Couchbase Module, Clickhouse Module, DB2 Module, Dynalite Module, InfluxDB Module, MariaDB Module, MongoDB Module, MS SQL Server Module, and MySQL Module. The main content area shows code examples for creating and starting a MongoDB container, a note about multi-node clusters, and sections for Motivation and General info. Handwritten red annotations include a large arrow pointing to the "Usage example" section with the word "open" written above it, and the words "Junit" and "Containers" written diagonally across the page.

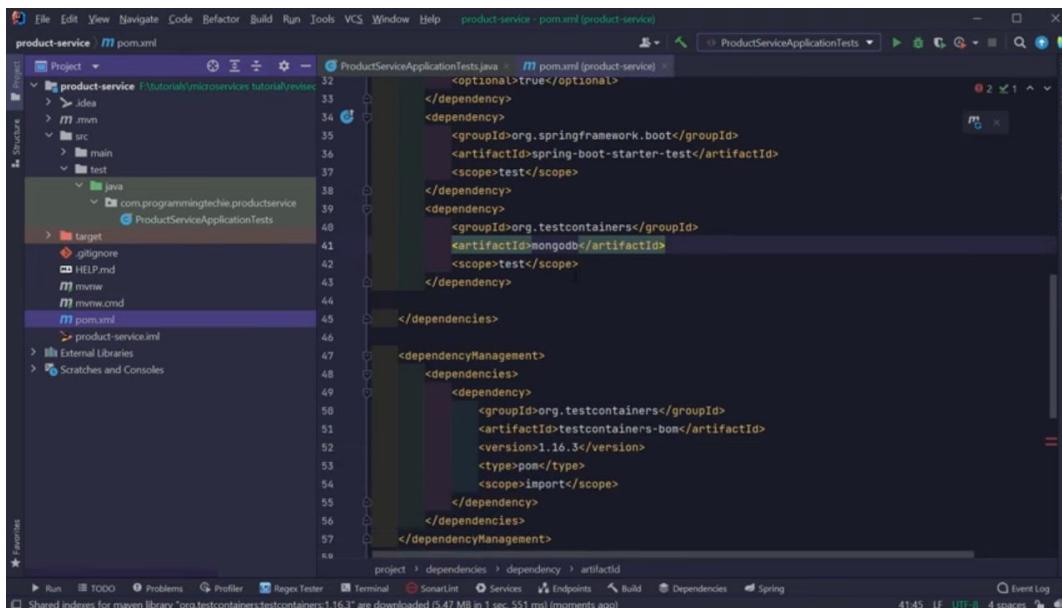
▶ 26:06



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

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

▶ 27:19



```
<dependency>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>mongo</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>testcontainers-bom</artifactId>
            <version>1.16.3</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

▶ 28:09

Jupiter / JUnit 5 - Testcontainers

testcontainers.org/test_framework_integration/junit_5/#adding-testcontainers-junit-5-support-to-your-project-dependencies

Jupiter / JUnit 5

Limitations

Testcontainers

- Home
- Quickstart
- Features
- Modules
- Test framework integration
 - JUnit 4
 - Jupiter / JUnit 5
 - Spock
 - Manual container lifecycle control
- System Requirements
- Getting help
- Contributing

Since this module has a dependency onto JUnit Jupiter and on Testcontainers core, which has a dependency onto JUnit 4.x, projects using this module will end up with both, JUnit Jupiter and JUnit 4.x in the test classpath.

This extension has only be tested with sequential test execution. Using it with parallel test execution is unsupported and may have unintended side effects.

Adding Testcontainers JUnit 5 support to your project dependencies

Add the following dependency to your pom.xml / build.gradle file:

```
testImplementation "org.testcontainers:junit-jupiter:1.16.3"
```

Table of contents

- Extension
- Examples
- Restarted containers
- Shared containers
- Singleton containers
- Limitations
- Adding Testcontainers JUnit 5 support to your project dependencies

Previous JUnit 4 Next Spock

Made with Material for MkDocs

▶ 29:01

Jupiter / JUnit 5 - Testcontainers

testcontainers.org/test_framework_integration/junit_5/#adding-testcontainers-junit-5-support-to-your-project-dependencies

Jupiter / JUnit 5

Limitations

Testcontainers

- Home
- Quickstart
- Features
- Modules
- Test framework integration
 - JUnit 4
 - Jupiter / JUnit 5
 - Spock
 - Manual container lifecycle control
- System Requirements
- Getting help
- Contributing

JUnit 4.x in the test classpath.

This extension has only be tested with sequential test execution. Using it with parallel test execution is unsupported and may have unintended side effects.

Adding Testcontainers JUnit 5 support to your project dependencies

Add the following dependency to your pom.xml / build.gradle file:

```
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>1.16.3</version>
    <scope>test</scope>
</dependency>
```

Table of contents

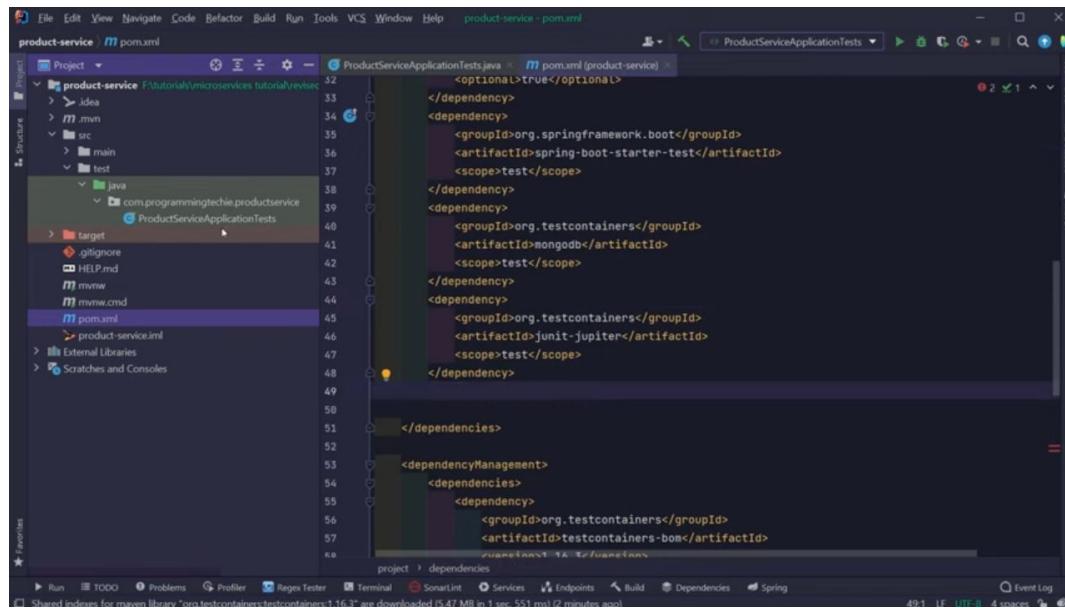
- Extension
- Examples
- Restarted containers
- Shared containers
- Singleton containers
- Limitations
- Adding Testcontainers JUnit 5 support to your project dependencies

Previous JUnit 4 Next Spock

Made with Material for MkDocs

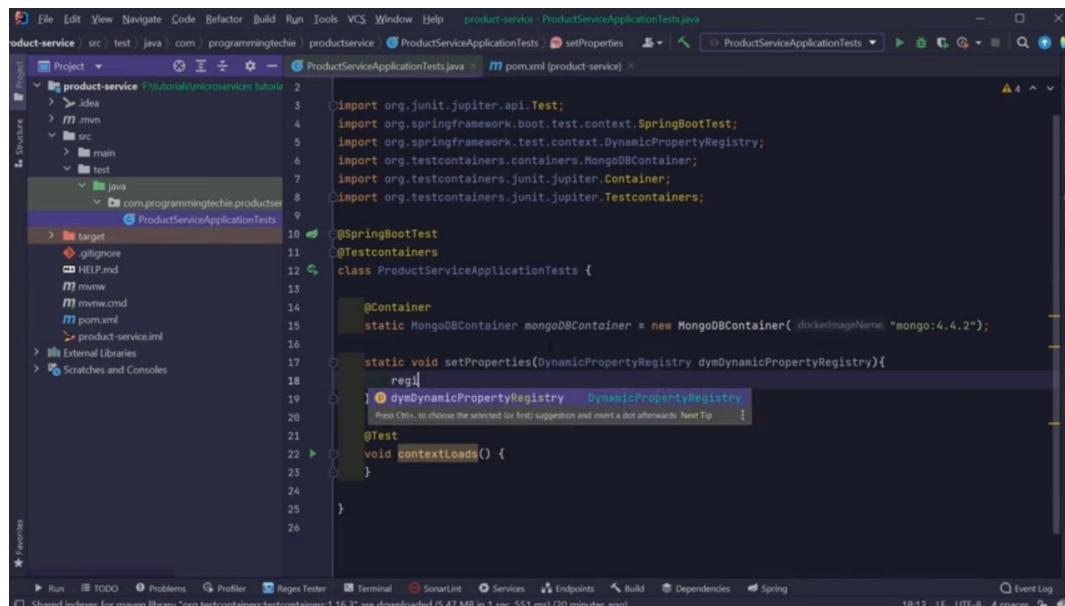
▶ 29:07

Copy
The
Maven
dependency



```
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>mongodb</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>junit-jupiter</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>testcontainers-bom</artifactId>
        </dependency>
    </dependencies>
</dependencyManagement>
```

▶ 29:28



```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.DynamicPropertyRegistry;
import org.testcontainers.containers.MongoDBContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.junit.jupiter.Testcontainers;

@SpringBootTest
@Testcontainers
class ProductServiceApplicationTests {

    @Container
    static MongoDBContainer mongoDBContainer = new MongoDBContainer("mongo:4.2");

    static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
        reg| dyDynamicPropertyRegistry DynamicPropertyRegistry
    }

    @Test
    void contextLoads() {
    }
}
```

▶ 31:44

The screenshot shows a Java IDE interface with a code editor displaying a test class named `ProductServiceApplicationTests`. The code uses `@Testcontainers` annotations and imports from `org.testcontainers` and `org.springframework.boot.test.context.SpringBootTest`. A code completion dropdown is open over the `mongoDBContainer` field, listing various methods such as `add`, `equals`, `hashCode`, `toString`, and `getClass`.

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.testcontainers.containers.MongoDBContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.junit.jupiter.Testcontainers;

@SpringBootTest
@Testcontainers
class ProductServiceApplicationTests {

    @Container
    static MongoDBContainer mongoDBContainer = new MongoDBContainer("dockerImageName: mongo:4.4.2");

    static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
        dynDynamicPropertyRegistry.
            add(String name, Supplier<Object> valueSupplier)
            equals(Object obj)
            hashCode()
            toString()
            getClass()
            arg
            cast
            functionCall(expr)
            notify()
            notifyAll()
            wait()
            wait(long timeoutMillis)
            wait(long timeoutMillis, int nanos)
            castVar
    }

    @Test
    void contextLoads() {
    }
}
```

▶ 31:47

This screenshot is identical to the one above, showing the same code editor and code completion dropdown for the `mongoDBContainer` field.

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.testcontainers.containers.MongoDBContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.junit.jupiter.Testcontainers;

@SpringBootTest
@Testcontainers
class ProductServiceApplicationTests {

    @Container
    static MongoDBContainer mongoDBContainer = new MongoDBContainer("dockerImageName: mongo:4.4.2");

    static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
        dynDynamicPropertyRegistry.add("name: spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl);
    }

    @Test
    void contextLoads() {
    }
}
```

▶ 32:18

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.DynamicPropertyRegistry;
import org.springframework.test.context.DynamicPropertySource;
import org.testcontainers.containers.MongoDBContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.junit.jupiter.Testcontainers;

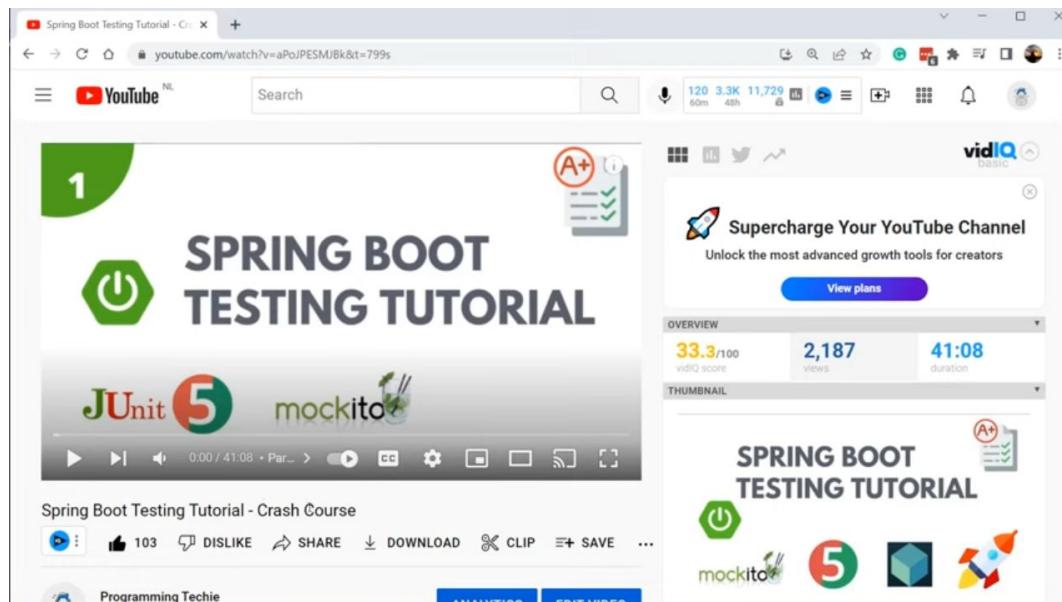
@SpringBootTest
@Testcontainers
class ProductServiceApplicationTests {

    @Container
    static MongoDBContainer mongoDBContainer = new MongoDBContainer("mongo:4.4.2");

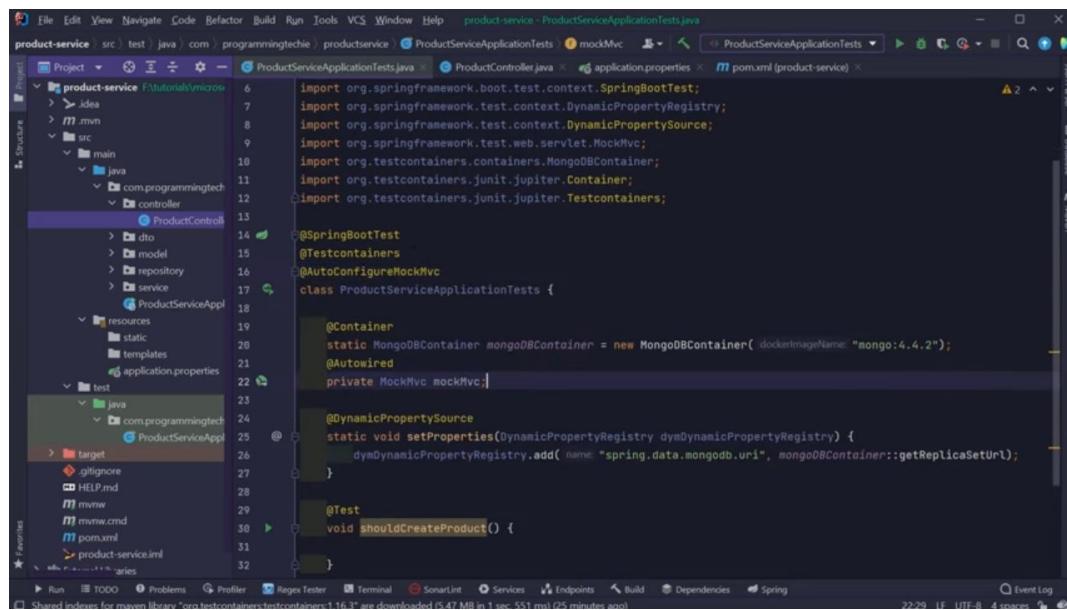
    @DynamicPropertySource
    static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
        dynDynamicPropertyRegistry.add("spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl);
    }

    @Test
    void contextLoads() {
    }
}
```

▶ 33:18



▶ 34:33



```
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.DynamicPropertyRegistry;
import org.springframework.test.context.DynamicPropertySource;
import org.springframework.test.web.servlet.MockMvc;
import org.testcontainers.containers.MongoDBContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.junit.jupiter.Testcontainers;

@AutoConfigureMockMvc
class ProductServiceApplicationTests {

    @Container
    static MongoDBContainer mongoDBContainer = new MongoDBContainer("mongo:4.4.2");
    @Autowired
    private MockMvc mockMvc;

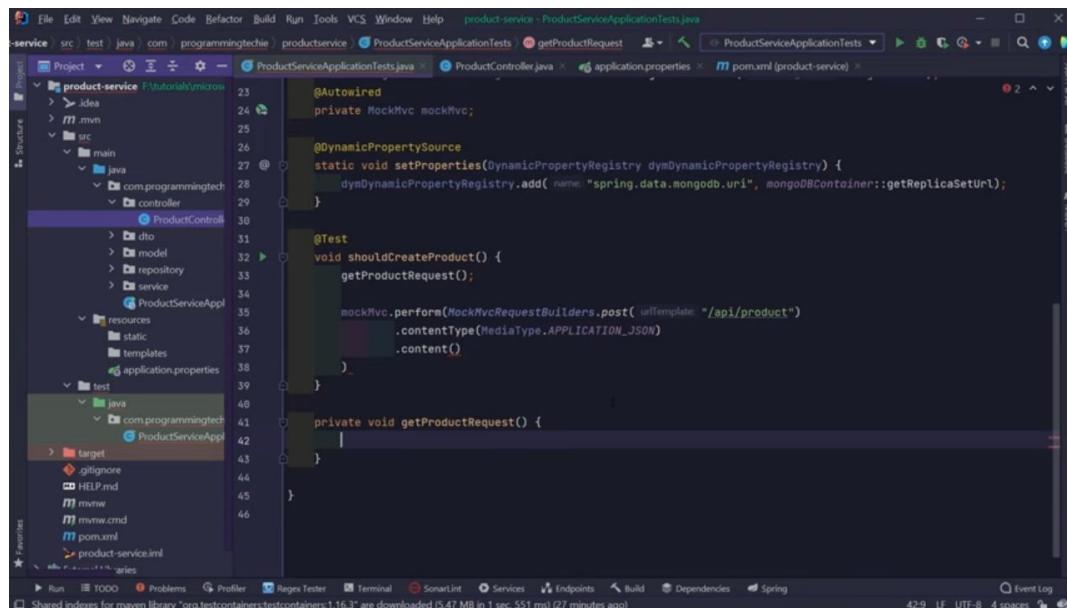
    @DynamicPropertySource
    static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
        dynDynamicPropertyRegistry.add("spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl);
    }

    @Test
    void shouldCreateProduct() {
        getProductRequest();

        mockMvc.perform(MockMvcRequestBuilders.post("/api/product")
                .contentType(MediaType.APPLICATION_JSON)
                .content(""));
    }

    private void getProductRequest() {
    }
}
```

▶ 35:32



```
@Autowired
private MockMvc mockMvc;

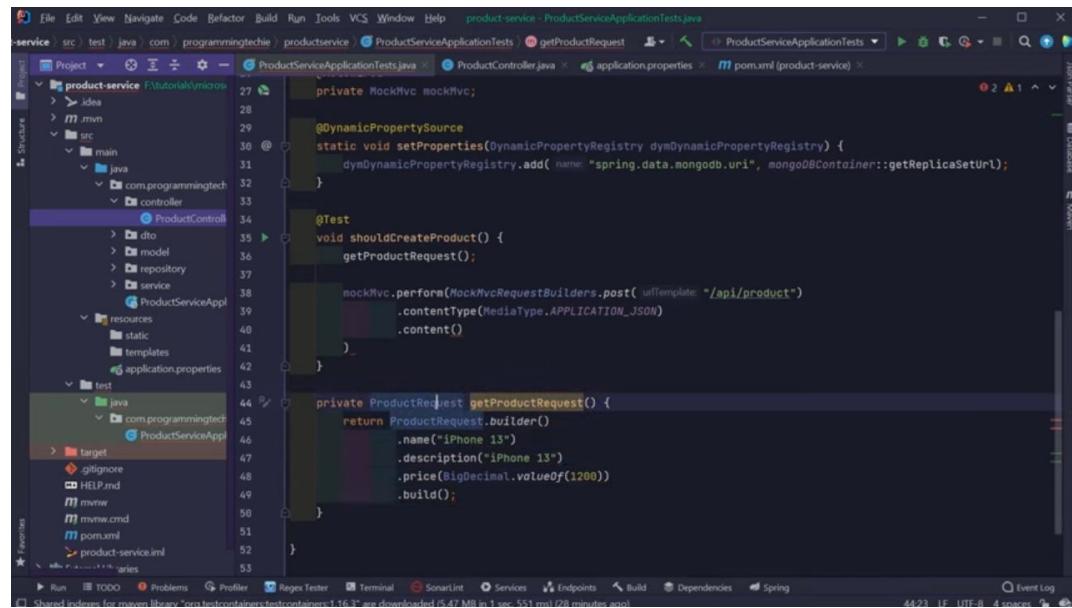
@DynamicPropertySource
static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
    dynDynamicPropertyRegistry.add("spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl);
}

@Test
void shouldCreateProduct() {
    getProductRequest();

    mockMvc.perform(MockMvcRequestBuilders.post("/api/product")
            .contentType(MediaType.APPLICATION_JSON)
            .content(""));
}

private void getProductRequest() {
}
```

▶ 37:15



```
private MockMvc mockMvc;

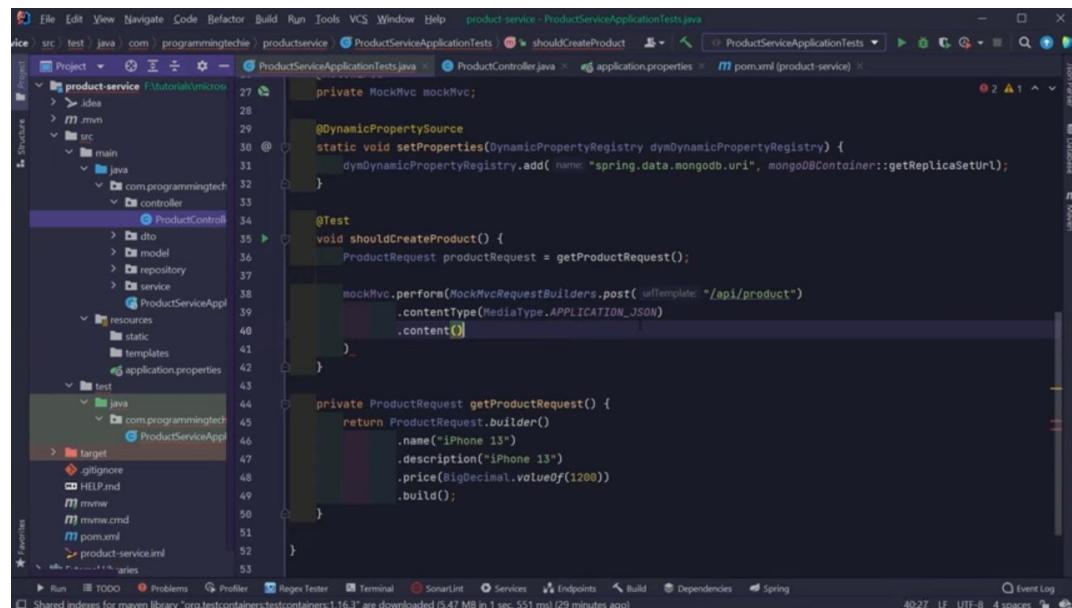
@DynamicPropertySource
static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
    dynDynamicPropertyRegistry.add( name: "spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl);
}

@Test
void shouldCreateProduct() {
    getProductRequest();

    mockMvc.perform(MockMvcRequestBuilders.post( urlTemplate: "/api/product")
        .contentType(MediaType.APPLICATION_JSON)
        .content()
    );
}

private ProductRequest getProductRequest() {
    return ProductRequest.builder()
        .name("iPhone 13")
        .description("iPhone 13")
        .price(BigDecimal.valueOf(1200))
        .build();
}
```

▶ 38:01



```
private MockMvc mockMvc;

@DynamicPropertySource
static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
    dynDynamicPropertyRegistry.add( name: "spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl);
}

@Test
void shouldCreateProduct() {
    ProductRequest productRequest = getProductRequest();

    mockMvc.perform(MockMvcRequestBuilders.post( urlTemplate: "/api/product")
        .contentType(MediaType.APPLICATION_JSON)
        .content()
    );
}

private ProductRequest getProductRequest() {
    return ProductRequest.builder()
        .name("iPhone 13")
        .description("iPhone 13")
        .price(BigDecimal.valueOf(1200))
        .build();
}
```

▶ 38:18

```
private MockMvc mockMvc;

@DynamicPropertySource
static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
    dynDynamicPropertyRegistry.add( name: "spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl);
}

@Test
void shouldCreateProduct() {
    ProductRequest productRequest = getProductRequest();

    mockMvc.perform(MockMvcRequestBuilders.post( urlTemplate: "/api/product")
        .contentType(MediaType.APPLICATION_JSON)
        .content()
    )
}

private ProductRequest getProductRequest() {
    return ProductRequest.builder()
        .name("iPhone 13")
        .description("iPhone 13")
        .price(BigDecimal.valueOf(1200))
        .build();
}
```

▶ 38:20

```
@AutoConfigureMockMvc
class ProductServiceApplicationTests {

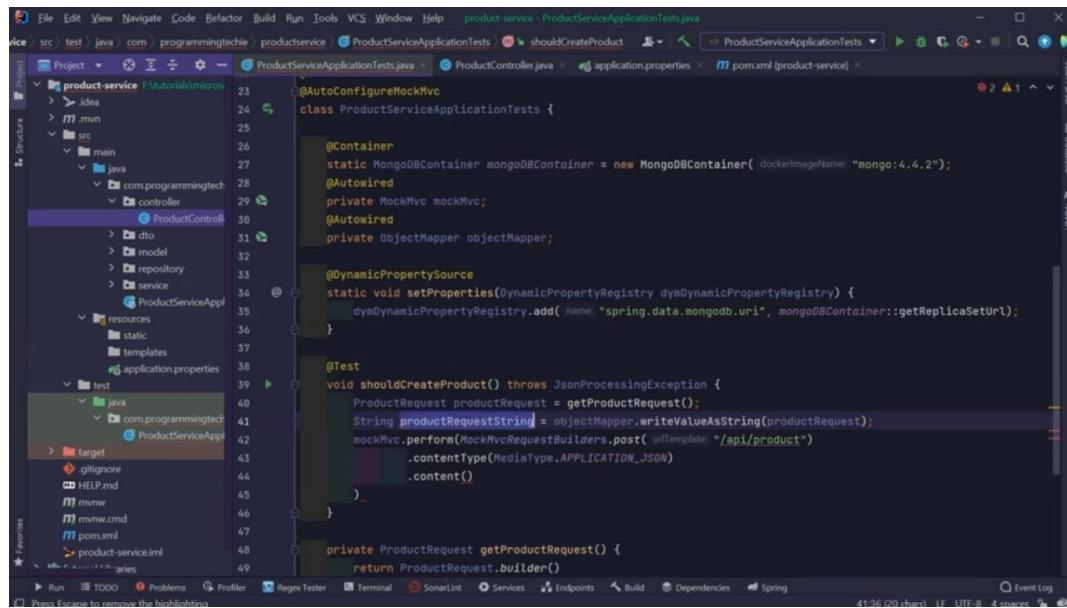
    @Container
    static MongoDbContainer mongoDBContainer = new MongoDbContainer( dockerImageName: "mongo:4.4.2");
    @Autowired
    private MockMvc mockMvc;
    @Autowired
    private ObjectMapper objectMapper;

    @DynamicPropertySource
    static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
        dynDynamicPropertyRegistry.add( name: "spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl);
    }

    @Test
    void shouldCreateProduct() throws JsonProcessingException {
        ProductRequest productRequest = getProductRequest();
        objectMapper.writeValueAsString(productRequest);
        mockMvc.perform(MockMvcRequestBuilders.post( urlTemplate: "/api/product")
            .contentType(MediaType.APPLICATION_JSON)
            .content()
        )
    }

    private ProductRequest getProductRequest() {
        return ProductRequest.builder()
    }
}
```

▶ 39:19



File Edit View Navigate Code refactor Build Run Tools VCS Window Help product-service - ProductServiceApplicationTests.java

product-service

src

test

java

com

programmingtech

productservice

ProductServiceApplicationTests

ProductController

application.properties

pom.xml

ProductServiceApp

resources

static

templates

ProductServiceApplicationTests.java

```
@AutoConfigureMockMvc
class ProductServiceApplicationTests {

    @Container
    static MongoDBContainer mongoDBContainer = new MongoDBContainer("dockerImageName: \"mongo:4.4.2\"");
    @Autowired
    private MockMvc mockMvc;
    @Autowired
    private ObjectMapper objectMapper;

    @DynamicPropertySource
    static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
        dynDynamicPropertyRegistry.add("name: \"spring.data.mongodb.uri\"", mongoDBContainer::getReplicaSetUrl);
    }

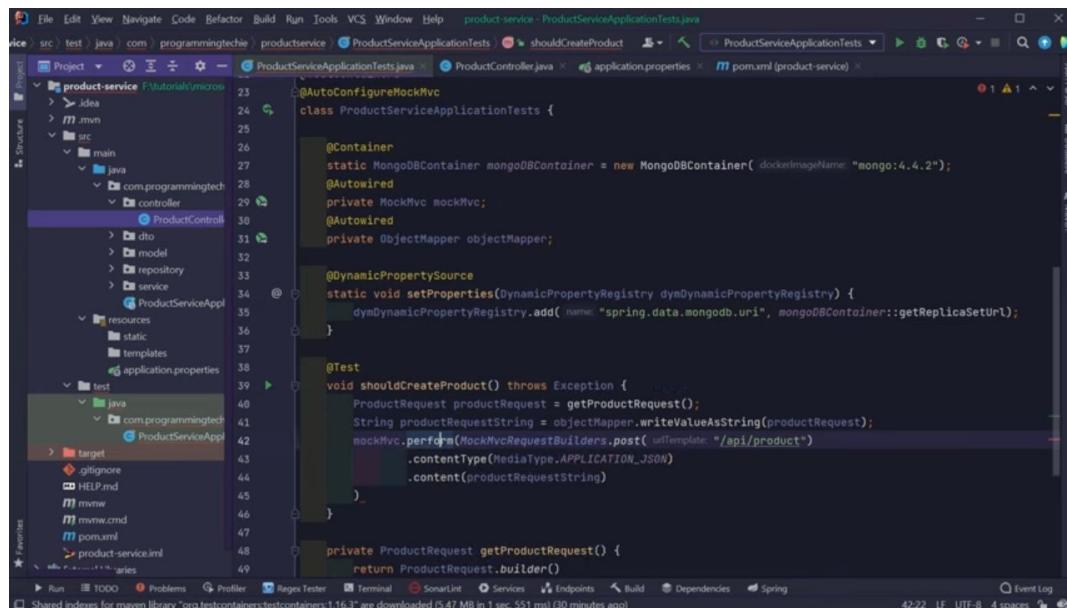
    @Test
    void shouldCreateProduct() throws JsonProcessingException {
        ProductRequest productRequest = getProductRequest();
        String productRequestString = objectMapper.writeValueAsString(productRequest);
        mockMvc.perform(MockMvcRequestBuilders.post("/api/product")
                .contentType(MediaType.APPLICATION_JSON)
                .content(productRequestString));
    }

    private ProductRequest getProductRequest() {
        return ProductRequest.builder()
    }
}
```

Run TODO Problems Profiler Regex Tester Terminal SonarLint Services Endpoints Build Dependencies Spring Event Log

41:36 (2024-01-16 10:36:41) 1E 1TE 0 4 sources 0

▶ 39:29



File Edit View Navigate Code refactor Build Run Tools VCS Window Help product-service - ProductServiceApplicationTests.java

product-service

src

test

java

com

programmingtech

productservice

ProductServiceApplicationTests

ProductController

application.properties

pom.xml

ProductServiceApp

resources

static

templates

ProductServiceApplicationTests.java

```
@AutoConfigureMockMvc
class ProductServiceApplicationTests {

    @Container
    static MongoDBContainer mongoDBContainer = new MongoDBContainer("dockerImageName: \"mongo:4.4.2\"");
    @Autowired
    private MockMvc mockMvc;
    @Autowired
    private ObjectMapper objectMapper;

    @DynamicPropertySource
    static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
        dynDynamicPropertyRegistry.add("name: \"spring.data.mongodb.uri\"", mongoDBContainer::getReplicaSetUrl);
    }

    @Test
    void shouldCreateProduct() throws Exception {
        ProductRequest productRequest = getProductRequest();
        String productRequestString = objectMapper.writeValueAsString(productRequest);
        mockMvc.perform(MockMvcRequestBuilders.post("/api/product")
                .contentType(MediaType.APPLICATION_JSON)
                .content(productRequestString));
    }

    private ProductRequest getProductRequest() {
        return ProductRequest.builder()
    }
}
```

Run TODO Problems Profiler Regex Tester Terminal SonarLint Services Endpoints Build Dependencies Spring Event Log

42:22 (2024-01-16 10:42:22) 1E 1TE 0 4 sources 0

▶ 39:39

```
File Edit View Navigate Code Befactor Build Run Tools VCS Window Help product-service - ProductServiceApplicationTests.java
src test java com programmingtechie productservice ProductServiceApplicationTests.java ProductController.java application.properties pom.xml (product-service)
Project
src test java com programmingtechie productservice ProductServiceApplicationTests.java ProductController.java application.properties pom.xml (product-service)
25 @AutoConfigureMockMvc
26 class ProductServiceApplicationTests {
27
28     @Container
29     static MongoDBContainer mongoDBContainer = new MongoDBContainer( dockerImageName: "mongo:4.4.2" );
30
31     @Autowired
32     private MockMvc mockMvc;
33
34     @Autowired
35     private ObjectMapper objectMapper;
36
37     @DynamicPropertySource
38     static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
39         dynDynamicPropertyRegistry.add( name: "spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl );
40     }
41
42     @Test
43     void shouldCreateProduct() throws Exception {
44         ProductRequest productRequest = getProductRequest();
45         String productRequestString = objectMapper.writeValueAsString(productRequest);
46         mockMvc.perform(MockMvcRequestBuilders.post( UriTemplate.of("/api/product") )
47                         .contentType(MediaType.APPLICATION_JSON)
48                         .content(productRequestString))
49                         .andExpect(status().isCreated());
50     }
51
52     private ProductRequest getProductRequest() {
53         return ProductRequest.builder()
54             .
```

▶ 40:28

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure for "product-service".
- Code Editor:** The main editor window displays the file `ProductServiceApplicationTests.java`. A yellow dot icon is visible near the bottom of the code area.
- Code:** The code in the editor is as follows:

```
ProductRequest productRequest = getProductRequest();
String productRequestString = objectMapper.writeValueAsString(productRequest);
mockMvc.perform(MockMvcRequestBuilders.post("/api/product")
    .contentType(MediaType.APPLICATION_JSON)
    .content(productRequestString))
    .andExpect(status().isCreated());
```

```
private ProductRequest getProductRequest() {
    return ProductRequest.builder()
        .name("iPhone 13")
        .description("iPhone 13")
        .price(BigDecimal.valueOf(1200))
        .build();}
```

- Run Tab:** The bottom navigation bar shows the "Run" tab is active, with the test class `ProductServiceApplicationTests` selected.
- Output Tab:** The bottom right panel shows the "Output" tab, displaying the results of the test run:

- Tests passed: 1 of 1 Test - 687 ms
- Test Results section shows a single test named `shouldCreateProduct()` with a duration of 687 ms.
- Logs section shows the following output:
 - main] org.mongodb.driver.connection : Opened connection [connectionId{localValue:3, server'↑
 - main] c.p.p.service.ProductService : Product 625bc9fb7cc9d8593d175777 is saved

▶ 41:25

```
private ProductRepository productRepository;

@DynamicPropertySource
static void setProperties(DynamicPropertyRegistry dynDynamicPropertyRegistry) {
    dynDynamicPropertyRegistry.add( name: "spring.data.mongodb.uri", mongoDBContainer::getReplicaSetUrl);
}

@Test
void shouldCreateProduct() throws Exception {
    ProductRequest productRequest = getProductRequest();
    String productRequestString = objectMapper.writeValueAsString(productRequest);
    mockMvc.perform(MockMvcRequestBuilders.post( urlTemplate: "/api/product")
        .contentType(MediaType.APPLICATION_JSON)
        .content(productRequestString))
        .andExpect(status().isCreated());
    Assertions.assertTrue( condition: productRepository.findAll().size() == 1);
}

private ProductRequest getProductRequest() {
    return ProductRequest.builder()
        .name("iPhone 13")
        .description("iPhone 13")
        .price(BigDecimal.valueOf(1200))
        .build();
}
```

▶ 42:34

Project: Maven Project

Spring Boot: 3.0.0 (SNAPSHOT)

Spring Configuration Processor

WEB

Spring Reactive Web

Spring GraphQL

▶ 45:30

IntelliJ IDEA interface showing the `Order.java` file under the `microservices-parent/order-service/order-service/src/main/java/com/programmingtechie/orderservice/model` path. The code defines the `Order` entity with Lombok annotations and a database table mapping.

```
1 package com.programmingtechie.orderservice.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Getter;
5 import lombok.NoArgsConstructor;
6 import lombok.Setter;
7
8 import javax.persistence.*;
9
10 @Entity
11 @Table(name = "t_orders")
12 @Getter
13 @Setter
14 @NoArgsConstructor
15 @AllArgsConstructor
16 public class Order {
17     @Id
18     @GeneratedValue(strategy = GenerationType.AUTO)
19     private Long id;
20     private String orderNumber;
21     private List<OrderLineItems> orderLineItemList;
22 }
```

▶ 49:32

IntelliJ IDEA interface showing the same `Order.java` file. A context menu is open over the `orderLineItemList` reference, displaying options such as 'Create class', 'Create enum', and 'Add Maven dependency'.

```
1 package com.programmingtechie.orderservice.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Getter;
5 import lombok.NoArgsConstructor;
6 import lombok.Setter;
7
8 import javax.persistence.*;
9
10 @Entity
11 @Table(name = "t_orders")
12 @Getter
13 @Setter
14 @NoArgsConstructor
15 @AllArgsConstructor
16 public class Order {
17     @Id
18     @GeneratedValue(strategy = GenerationType.AUTO)
19     private Long id;
20     private String orderNumber;
21     private List<OrderLineItems> orderLineItemList;
22 }
```

▶ 49:49

The screenshot shows the IntelliJ IDEA interface with the project 'microservices-parent' open. The 'order-service' module is selected. The 'src/main/java/com/programmingtechie/orderservice/model' package is expanded, showing the 'Order.java' file. The code defines a class 'Order' with annotations for Lombok and JPA. The 'orderLineItemsList' field is annotated with @OneToMany(cascade = CascadeType.ALL). The code editor has syntax highlighting and error markers.

```
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.*;
import java.util.List;

@Entity
@Table(name = "t_orders")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String orderNumber;
    private List<OrderLineItems> orderLineItemsList;
}
```

▶ 51:24

The screenshot shows the IntelliJ IDEA interface with the project 'microservices-parent' open. The 'order-service' module is selected. The 'src/main/java/com/programmingtechie/orderservice/model' package is expanded, showing the 'Order.java' file. The code defines a class 'Order' with annotations for Lombok and JPA. The 'orderLineItemsList' field is annotated with @OneToMany(cascade = CascadeType.ALL). A yellow warning icon is present on the line containing the annotation. The code editor has syntax highlighting and error markers.

```
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.*;
import java.util.List;

@Entity
@Table(name = "t_orders")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String orderNumber;
    private List<OrderLineItems> orderLineItemsList;
}
```

▶ 53:22

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** The left sidebar displays the project structure. The main module is "microservices-parent" (F:\tutorials\microservices\tutorial\revise). It contains the "order-service" module, which has a "src" directory containing "main" and "java". The "java" directory contains the "com.programmingtechie.orderservice.controller" package, which includes the "OrderController" class.
- Code Editor:** The right pane shows the code for "OrderController.java". The code defines a REST controller for placing orders:

```
package com.programmingtechie.orderservice.controller;

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/order")
public class OrderController {

    @PostMapping
    public String placeOrder() {
        return "Order Placed Successfully";
    }
}
```

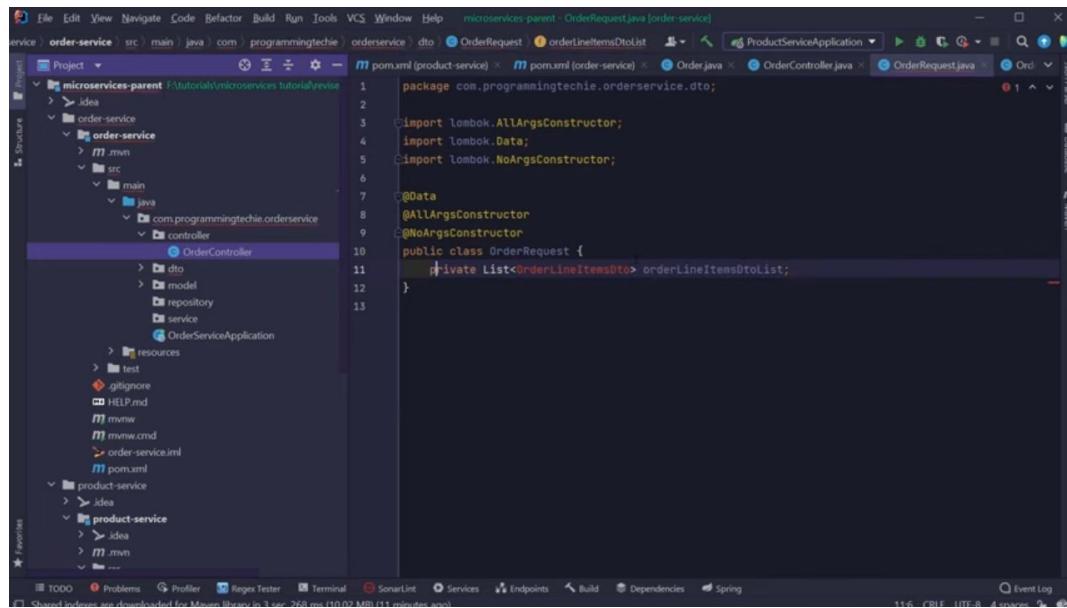
▶ 54:06

```
package com.programmingtechie.orderservice.dto;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import javax.persistence.Id;  
import java.util.List;  
  
@PostMapping  
public OrderResponse createOrder(@RequestBody OrderRequest orderRequest) {  
    OrderResponse response = new OrderResponse();  
    response.setOrderNumber(orderRequest.getOrderNumber());  
    response.setCustomerName(orderRequest.getCustomerName());  
    response.setOrderItems(orderRequest.getOrderItems());  
    return response;  
}
```

The screenshot shows the IntelliJ IDEA interface with the following details:

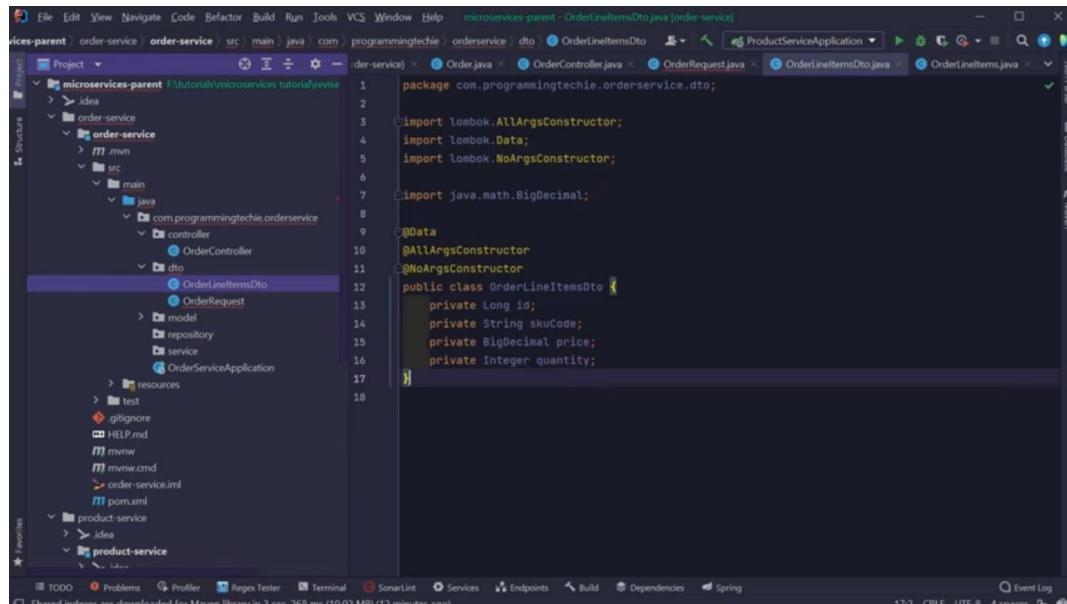
- Project Structure:** The left sidebar shows the project structure for "microservices-parent". It includes modules for "order-service" and "product-service", each with their own sub-modules like "src", "main", and "java".
- Editor:** The main editor window displays the `OrderController.java` file. The code defines a REST controller with a `@PostMapping` annotation and a `@RequestBody` parameter named `orderRequest`.
- Code Completion:** A tooltip is open over the `orderRequest` parameter, listing various Java and Spring annotations:
 - RequestMapping
 - RequestBody
 - OneToMany
 - Id
 - NoArgsConstructor
 - GeneratedValue
 - Generated
 - GeneratedLombok
 - GeneratedJavaAnnotationProcessing
 - GeneratedHibernateAnnotations
 - Generated
- Bottom Navigation:** The bottom navigation bar includes tabs for "TODO", "Problems", "Profiler", "RegEx Tester", "Terminal", "SonarLint", "Services", "Endpoints", "Build", "Dependencies", and "Spring".
- Event Log:** The bottom right corner shows the "Event Log" tab.

▶ 54:30



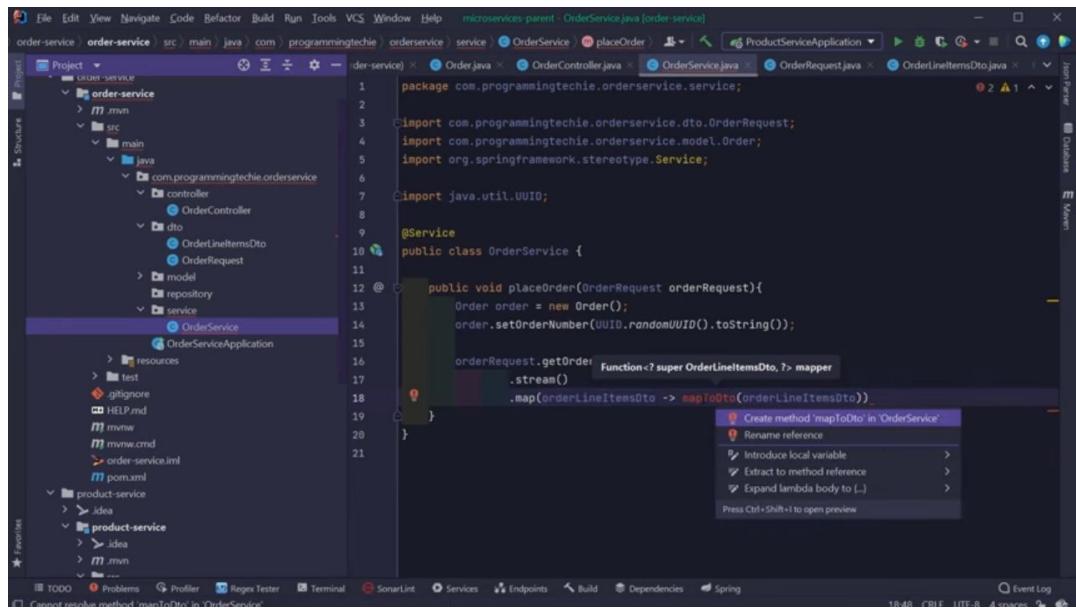
```
package com.programmingtechie.orderservice.dto;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@AllArgsConstructor
@NoArgsConstructor
public class OrderRequest {
    private List<OrderLineItemsDto> orderLineItemsDtoList;
}
```

▶ 54:59



```
package com.programmingtechie.orderservice.dto;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.math.BigDecimal;
@Data
@AllArgsConstructor
@NoArgsConstructor
public class OrderLineItemsDto {
    private Long id;
    private String skuCode;
    private BigDecimal price;
    private Integer quantity;
}
```

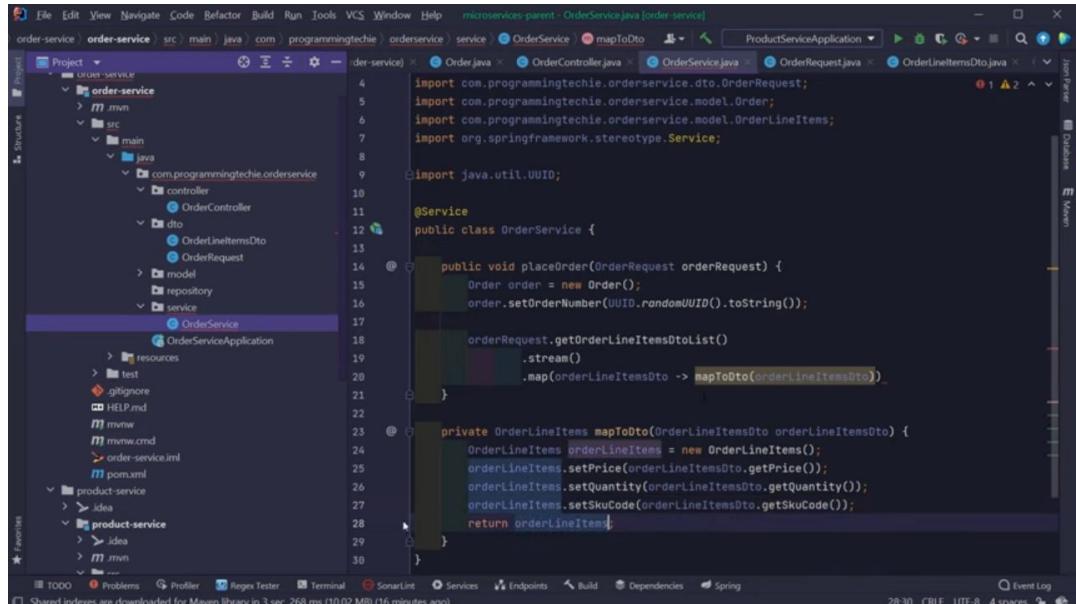
▶ 55:40



IntelliJ IDEA interface showing the OrderService.java file. The cursor is at line 18, column 19, where a lambda expression is being typed. A code completion dropdown is open, showing options like 'Create method 'mapToDto'' and 'Rename reference'.

```
1 package com.programmingtechie.orderservice.service;
2
3 import com.programmingtechie.orderservice.dto.OrderRequest;
4 import com.programmingtechie.orderservice.model.Order;
5 import org.springframework.stereotype.Service;
6
7 import java.util.UUID;
8
9 @Service
10 public class OrderService {
11
12     @Override
13     public void placeOrder(OrderRequest orderRequest) {
14         Order order = new Order();
15         order.setOrderNumber(UUID.randomUUID().toString());
16
17         orderRequest.getOrderLineItems()
18             .stream()
19             .map(orderLineItemsDto -> mapToDto(orderLineItemsDto))
20
21     }
22 }
```

▶ 57:35



IntelliJ IDEA interface showing the OrderService.java file. The cursor is at line 23, column 19, where the mapToDto method is being implemented. The code completion dropdown has been closed, and the implementation logic is visible.

```
1 import com.programmingtechie.orderservice.dto.OrderRequest;
2 import com.programmingtechie.orderservice.model.Order;
3 import com.programmingtechie.orderservice.model.OrderLineItems;
4 import org.springframework.stereotype.Service;
5
6 import java.util.UUID;
7
8 @Service
9 public class OrderService {
10
11     @Override
12     public void placeOrder(OrderRequest orderRequest) {
13         Order order = new Order();
14         order.setOrderNumber(UUID.randomUUID().toString());
15
16         orderRequest.getOrderLineItems()
17             .stream()
18             .map(orderLineItemsDto -> mapToDto(orderLineItemsDto))
19
20     }
21
22     private OrderLineItems mapToDto(OrderLineItemsDto orderLineItemsDto) {
23         OrderLineItems orderLineItem = new OrderLineItems();
24         orderLineItem.setPrice(orderLineItemsDto.getPrice());
25         orderLineItem.setQuantity(orderLineItemsDto.getQuantity());
26         orderLineItem.setSkuCode(orderLineItemsDto.getSkuCode());
27
28         return orderLineItem;
29     }
30 }
```

▶ 58:12

IntelliJ IDEA interface showing the `OrderService.java` file. The code implements the `OrderService` interface with a `placeOrder` method. A code completion tooltip is open over the `.toList()` method call, suggesting to introduce a local variable named `orderLineItems`.

```
import com.programmingtechie.orderservice.dto.OrderRequest;
import com.programmingtechie.orderservice.model.Order;
import com.programmingtechie.orderservice.model.OrderLineItems;
import org.springframework.stereotype.Service;
import java.util.UUID;

@Service
public class OrderService {

    public void placeOrder(OrderRequest orderRequest) {
        Order order = new Order();
        order.setOrderNumber(UUID.randomUUID().toString());

        orderRequest.getOrderLineItemsDtoList() (List<OrderLineItemsDto>)
            .stream()
            .map(this::mapToDto)
            .collect(Collectors.toList());
    }

    private OrderLineItems mapToDto(OrderLineItemsDto orderLineItemsDto) {
        OrderLineItems orderLineItems = new OrderLineItems();
        orderLineItems.setPrice(orderLineItemsDto.getPrice());
        orderLineItems.setQuantity(orderLineItemsDto.getQuantity());
        orderLineItems.setSkuCode(orderLineItemsDto.getSkuCode());
        return orderLineItems;
    }
}
```

▶ 58:29

IntelliJ IDEA interface showing the `OrderRepository.java` file. It defines a Java interface extending `JpaRepository` for the `Order` entity.

```
package com.programmingtechie.orderservice.repository;

import com.programmingtechie.orderservice.model.Order;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

public interface OrderRepository extends JpaRepository<Order, Long> {
```

▶ 59:21

IntelliJ IDEA interface showing the `OrderService.java` file. The code implements the `OrderService` interface, which includes a `placeOrder` method. This method creates a new `Order`, sets its `orderNumber` to a random UUID, and then processes a list of `OrderLineItems` from the `orderRequest`. The `mapToDto` method is used to convert `OrderLineItemsDto` objects into `OrderLineItems` objects.

```
public class OrderService {  
    private final OrderRepository orderRepository;  
  
    public void placeOrder(OrderRequest orderRequest) {  
        Order order = new Order();  
        order.setOrderNumber(UUID.randomUUID().toString());  
  
        List<OrderLineItems> orderLineItems = orderRequest.getOrderLineItemsDtoList() .stream()  
            .map(this::mapToDto) Stream<OrderLineItems>.  
            .toList();  
  
        order.setOrderLineItemsList(orderLineItems);  
  
        orderRepository.save(order);  
    }  
  
    private OrderLineItems mapToDto(OrderLineItemsDto orderLineItemsDto) {  
        OrderLineItems orderLineItems = new OrderLineItems();  
        orderLineItems.setPrice(orderLineItemsDto.getPrice());  
        orderLineItems.setQuantity(orderLineItemsDto.getQuantity());  
        orderLineItems.setSkuCode(orderLineItemsDto.getSkuCode());  
        return orderLineItems;  
    }  
}
```

▶ 1:00:08

IntelliJ IDEA interface showing the `OrderService.java` file. The code is identical to the previous screenshot, implementing the `OrderService` interface with a `placeOrder` method that saves an `Order` object to the repository after setting its `orderNumber` and processing `OrderLineItems`.

```
public class OrderService {  
    private final OrderRepository orderRepository;  
  
    public void placeOrder(OrderRequest orderRequest) {  
        Order order = new Order();  
        order.setOrderNumber(UUID.randomUUID().toString());  
  
        List<OrderLineItems> orderLineItems = orderRequest.getOrderLineItemsDtoList() .stream()  
            .map(this::mapToDto) Stream<OrderLineItems>.  
            .toList();  
  
        order.setOrderLineItemsList(orderLineItems);  
  
        orderRepository.save(order);  
    }  
  
    private OrderLineItems mapToDto(OrderLineItemsDto orderLineItemsDto) {  
        OrderLineItems orderLineItems = new OrderLineItems();  
        orderLineItems.setPrice(orderLineItemsDto.getPrice());  
        orderLineItems.setQuantity(orderLineItemsDto.getQuantity());  
        orderLineItems.setSkuCode(orderLineItemsDto.getSkuCode());  
        return orderLineItems;  
    }  
}
```

▶ 1:00:23

IntelliJ IDEA interface showing the `OrderController.java` file in the `order-service` project. The code defines a `placeOrder` method that returns a success message.

```
import com.programmingtechie.orderservice.dto.OrderRequest;
import com.programmingtechie.orderservice.service.OrderService;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

public class OrderController {

    private final OrderService orderService;

    @PostMapping("/api/order")
    public String placeOrder(@RequestBody OrderRequest orderRequest) {
        orderService.placeOrder(orderRequest);
        return "Order Placed Successfully";
    }
}
```

▶ 1:01:06

IntelliJ IDEA interface showing the `OrderController.java` file. An IDE error occurred at the `@ResponseStatus` annotation on line 18.

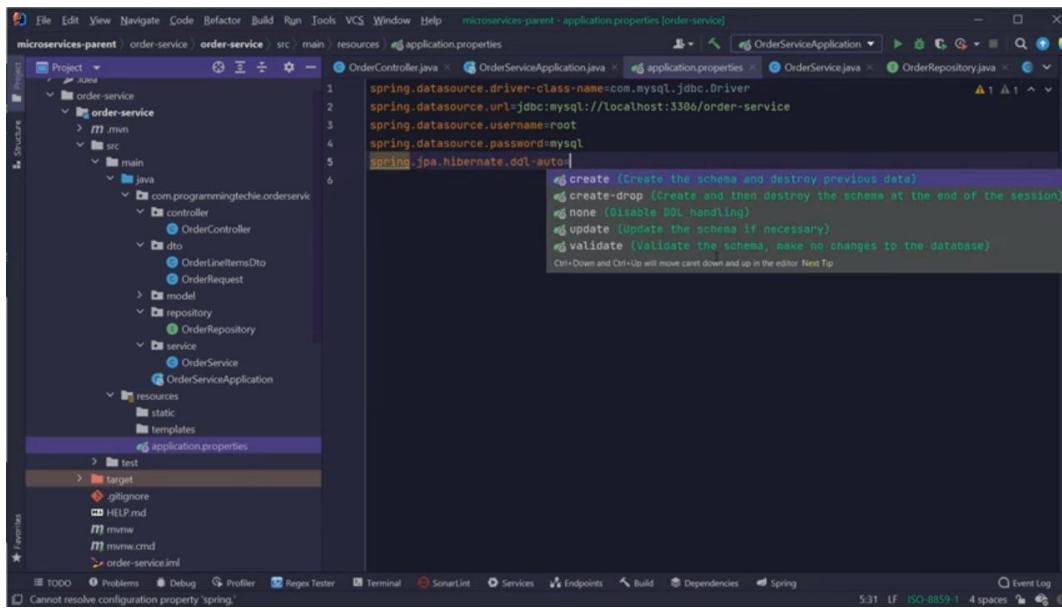
```
import com.programmingtechie.orderservice.dto.OrderRequest;
import com.programmingtechie.orderservice.service.OrderService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/order")
public class OrderController {

    private final OrderService orderService;

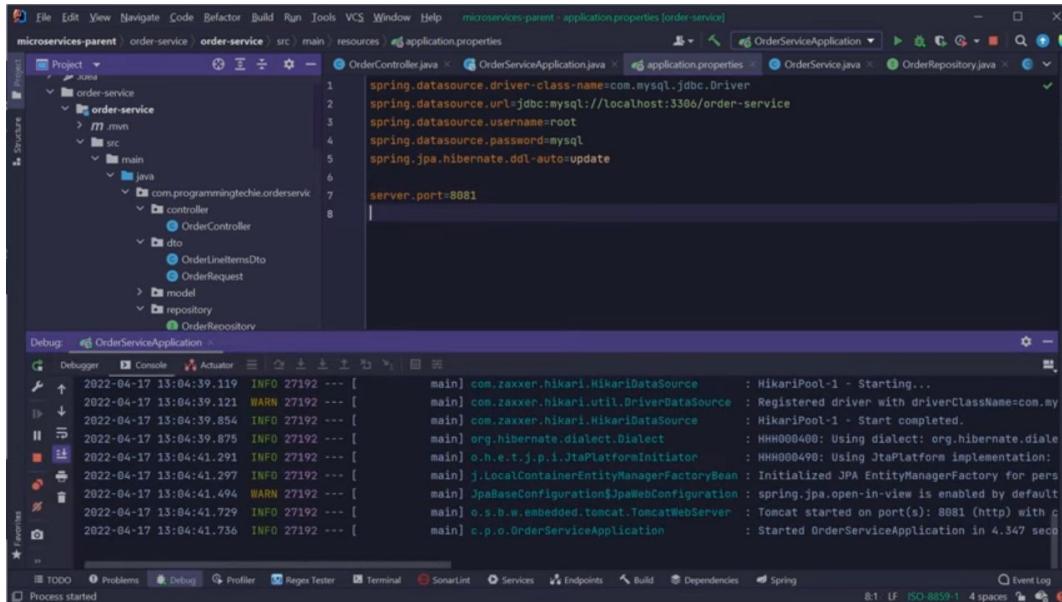
    @PostMapping()
    @ResponseStatus(HttpStatus.CREATED)
    public String placeOrder() {
        orderService.placeOrder();
        return "Order Placed Successfully";
    }
}
```

▶ 1:01:17



The screenshot shows the IntelliJ IDEA interface with the project 'microservices-parent' open. The 'order-service' module is selected. The code editor displays the 'application.properties' file. A tooltip is shown over the line 'spring.jpa.hibernate.ddl-auto'. The tooltip lists several options: 'create' (Create the schema and destroy previous data), 'create-drop' (Create and then destroy the schema at the end of the session), 'none' (Disable DDL handling), 'update' (Update the schema if necessary), and 'validate' (Validate the schema, make no changes to the database). The status bar at the bottom indicates 'Cannot resolve configuration property 'spring''.

▶ 1:02:44



The screenshot shows the IntelliJ IDEA interface with the project 'microservices-parent' open. The 'order-service' module is selected. The code editor displays the 'application.properties' file with the line 'server.port=8081' added. Below the editor is the 'Debug' tool window, which shows the 'OrderServiceApplication' process. The 'Console' tab in the debug window displays the application's startup logs, including the message 'Tomcat started on port(s): 8081 (http) with c'. The status bar at the bottom indicates 'Process started'.

▶ 1:03:04

Postman

File Edit View Help

Home Workspaces Reports Explore

Search Postman

No Environment

Save Send Sign Up for Free

GET http://localhost:8082/api/order

Params Authorization Headers (7) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1

Response

Find and Replace Console

Runner Trash

▶ 1:03:56

Postman

File Edit View Help

Home Workspaces Reports Explore

Search Postman

No Environment

Save Send Sign Up for Free

GET http://localhost:8081/api/order

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

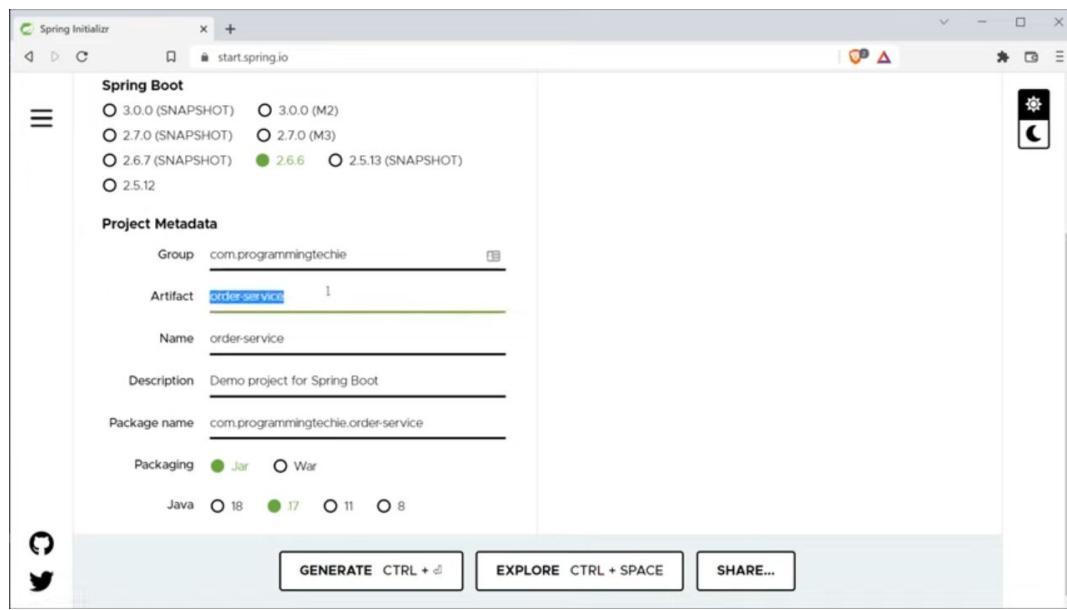
```
1 {
2   "orderLineItemsDtoList": [
3     {
4       "skuCode": "iphone_13",
5       "price": 1200,
6       "quantity": 1
7     }
8   ]
9 }
```

Response

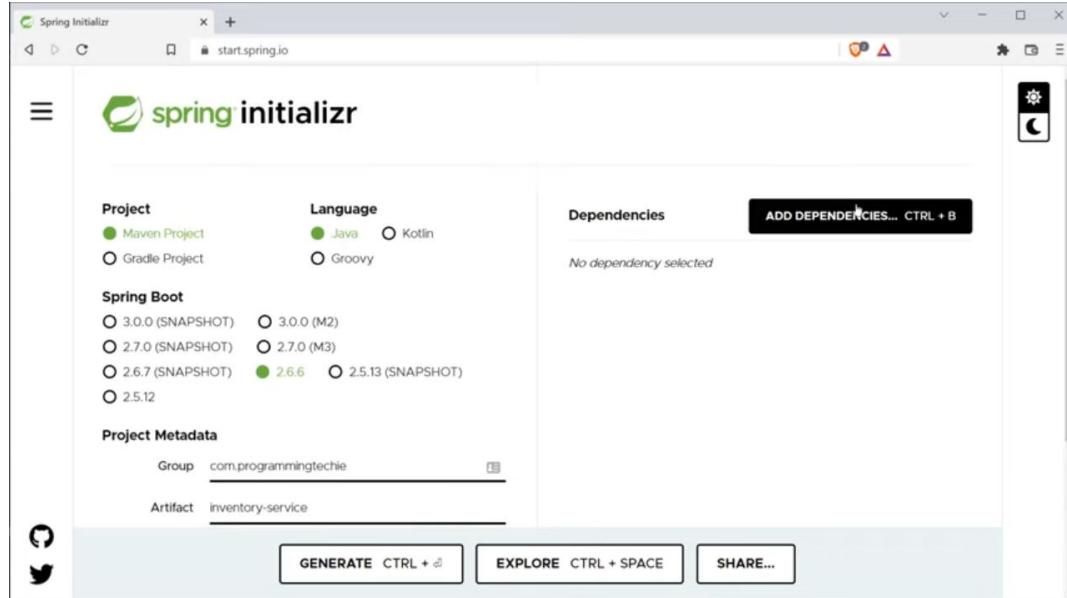
Find and Replace Console

Runner Trash

▶ 1:05:31



▶ 1:06:18



▶ 1:06:23

The screenshot shows the IntelliJ IDEA interface with the project 'inventory-service' open. The 'application.properties' file is selected in the left-hand navigation pane under 'src/main/resources'. The code editor displays the following configuration:

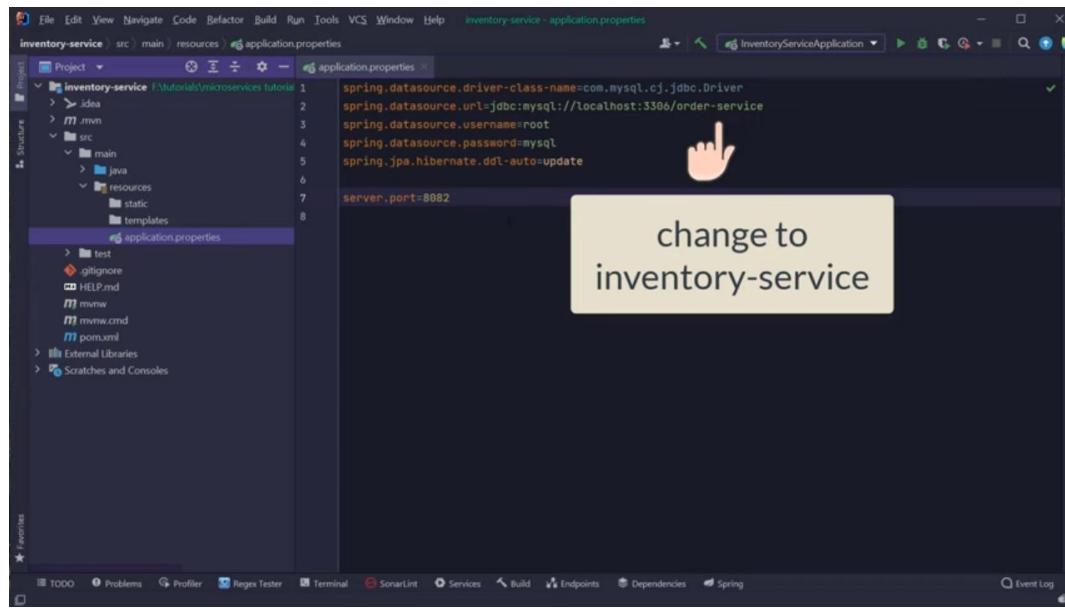
```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/order-service
spring.datasource.username=root
spring.datasource.password=mysql
spring.jpa.hibernate.ddl-auto=update
server.port=8081
```

▶ 1:07:00

The screenshot shows the IntelliJ IDEA interface with the project 'inventory-service' open. The 'application.properties' file is selected in the left-hand navigation pane under 'src/main/resources'. The code editor displays the following configuration:

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/order-service
spring.datasource.username=root
spring.datasource.password=mysql
spring.jpa.hibernate.ddl-auto=update
server.port=8081
```

▶ 1:07:30



File Edit View Navigate Code Refactor Build Run Tools VCS Window Help inventory-service - application.properties

Project inventory-service F:\Tutorials\microservices\tutorial

application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/order-service
spring.datasource.username=root
spring.datasource.password=mysql
spring.jpa.hibernate.ddl-auto=update
server.port=8082
```

change to
inventory-service

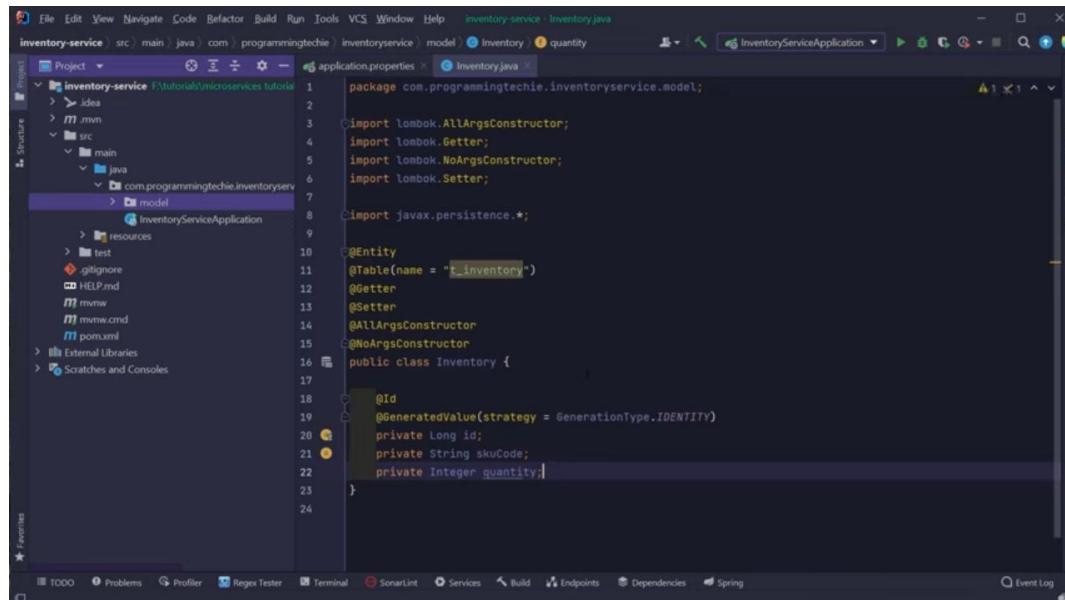
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help inventory-service - application.properties

Project inventory-service F:\Tutorials\microservices\tutorial

application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/order-service
spring.datasource.username=root
spring.datasource.password=mysql
spring.jpa.hibernate.ddl-auto=update
server.port=8082
```

▶ 1:08:14



File Edit View Navigate Code Refactor Build Run Tools VCS Window Help inventory-service - Inventory.java

Project inventory-service F:\Tutorials\microservices\tutorial

application.properties

```
package com.programmingtechie.inventoryservice.model;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.*;
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@Entity
@Table(name = "t_inventory")
public class Inventory {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String skuCode;
    private Integer quantity;
}
```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help inventory-service - Inventory.java

Project inventory-service F:\Tutorials\microservices\tutorial

application.properties

```
package com.programmingtechie.inventoryservice.model;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.*;
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@Entity
@Table(name = "t_inventory")
public class Inventory {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String skuCode;
    private Integer quantity;
}
```

▶ 1:09:36

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help inventory-service - InventoryRepository.java

Project inventory-service F:\Tutorial\microservices\tutorial

```
1 package com.programmingtechie.inventoryservice.repository;
2
3 import com.programmingtechie.inventoryservice.model.Inventory;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface InventoryRepository extends JpaRepository<Inventory, Long> {
```

com.programmingtechie.inventoryservice

- idea
- mvn
- src
- main
- java
- com.programmingtechie.inventoryservice
- model
- repository
- InventoryService

Copy Ctrl+C
Copy Path Reference... Ctrl+Shift+C
Cut Ctrl+X
Paste Ctrl+V
Find Usages Alt+F7
Find in Files... Ctrl+Shift+F
Replace in Files... Ctrl+Shift+R
Analyze >
Refactor >
Add to Favorites >
Format Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete
Override File Type
Build Module 'inventory-service'
Rebuild 'com.programmingtechie.inventoryservice'
Run Tests in 'com.programmingtechie.inventoryservice'
Debug Tests in 'com.programmingtechie.inventoryservice'
More Run/Debug >
Open In >

TODO Problems Profiler Local History Reload from Disk Event Log

▶ 1:10:41

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help inventory-service - InventoryService.java

Project inventory-service F:\Tutorial\microservices\tutorial

```
1 package com.programmingtechie.inventoryservice.service;
2
3 import com.programmingtechie.inventoryservice.repository.InventoryRepository;
4 import lombok.RequiredArgsConstructor;
5 import org.springframework.stereotype.Service;
6
7 @Service
8 @RequiredArgsConstructor
9 public class InventoryService {
10
11     private final InventoryRepository inventoryRepository;
12
13     public boolean isInStock(String skuCode) {
14         return inventoryRepository.findBySkuCode();
15     }
16 }
```

com.programmingtechie.inventoryservice

- idea
- mvn
- src
- main
- java
- com.programmingtechie.inventoryservice
- controller
- InventoryController
- model
- repository
- service
- InventoryService

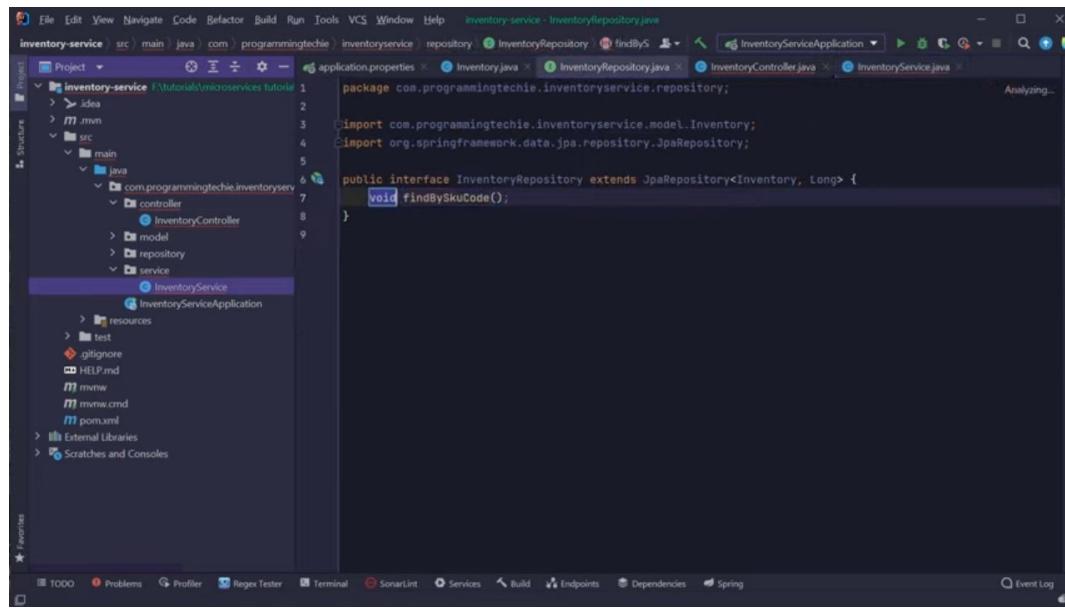
InventoryServiceApplication

application.properties Inventory.java InventoryRepository.java InventoryController.java InventoryService.java

Cannot resolve method 'findBySkuCode' in 'InventoryRepository'

Event Log

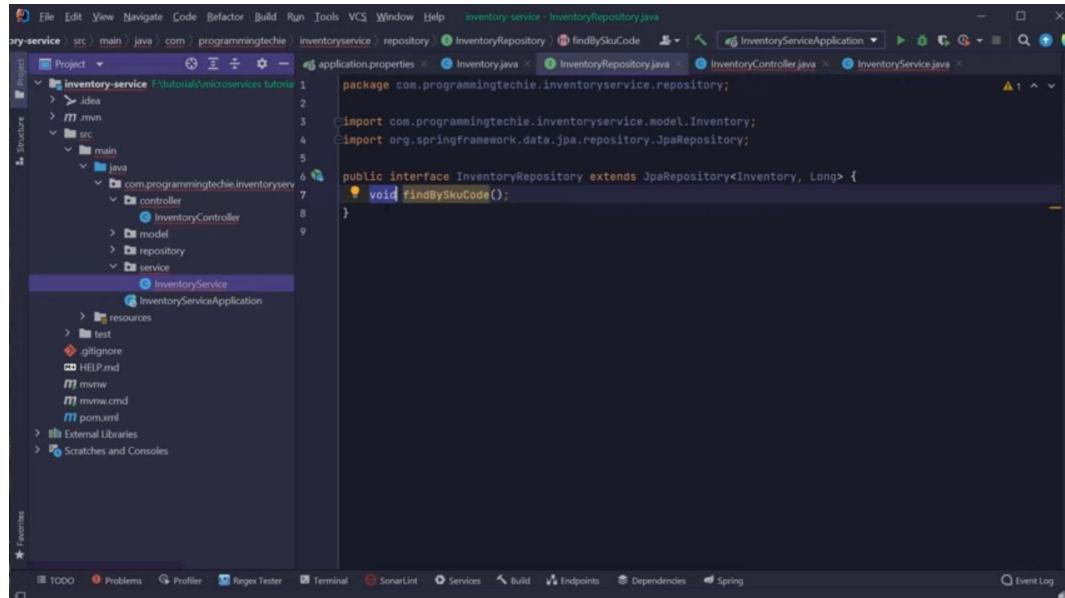
▶ 1:14:02



The screenshot shows the IntelliJ IDEA interface with the project 'inventory-service' open. The code editor displays the file 'InventoryRepository.java' which contains the following code:

```
1 package com.programmingtechie.inventoryservice.repository;
2
3 import com.programmingtechie.inventoryservice.model.Inventory;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface InventoryRepository extends JpaRepository<Inventory, Long> {
7     void findBySkuCode();
8 }
```

▶ 1:14:07

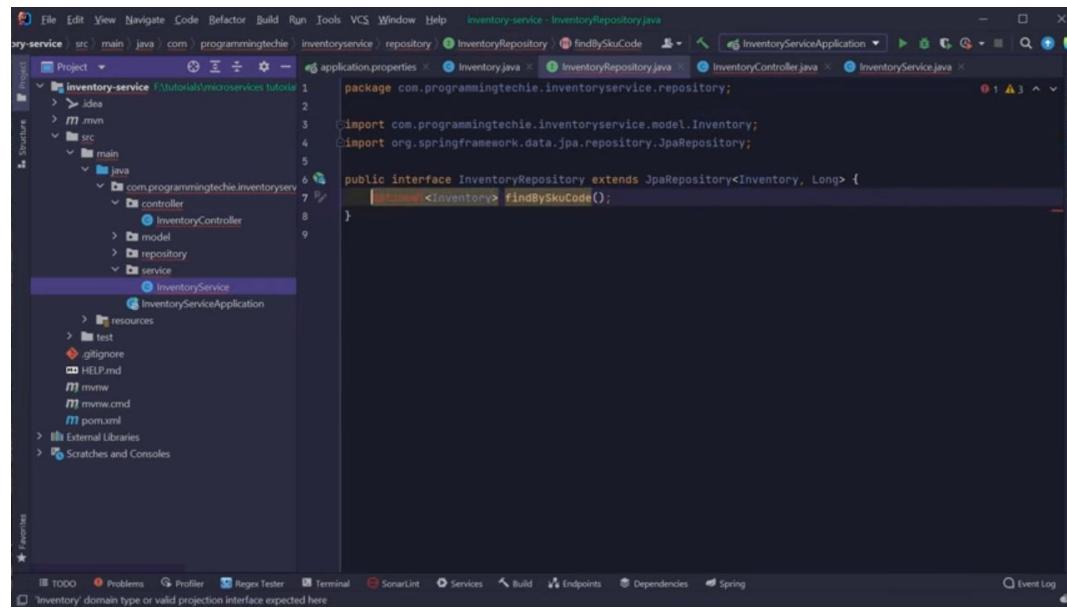


The screenshot shows the IntelliJ IDEA interface with the project 'inventory-service' open. The code editor displays the file 'InventoryRepository.java' which contains the following code:

```
1 package com.programmingtechie.inventoryservice.repository;
2
3 import com.programmingtechie.inventoryservice.model.Inventory;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface InventoryRepository extends JpaRepository<Inventory, Long> {
7     void findBySkuCode();
8 }
```

A yellow warning icon is visible in the top right corner of the code editor.

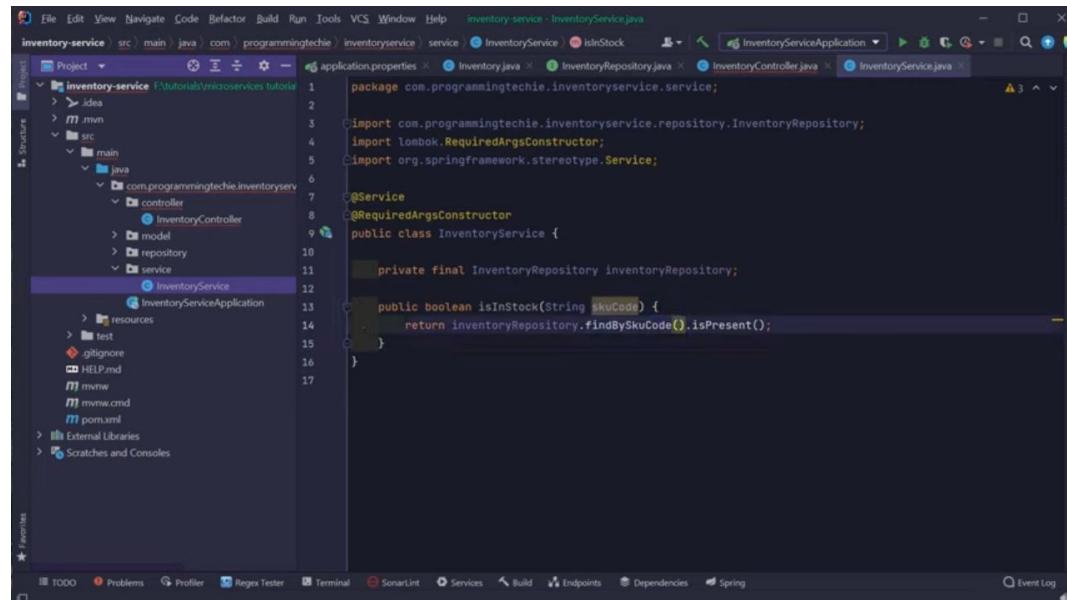
▶ 1:14:09



Screenshot of IntelliJ IDEA showing the implementation of the `InventoryRepository` interface. The code defines a method `findBySkuCode()` that returns a `Long`. The interface extends `JpaRepository<Inventory, Long>`.

```
package com.programmingtechie.inventoryservice.repository;
import com.programmingtechie.inventoryservice.model.Inventory;
import org.springframework.data.jpa.repository.JpaRepository;
public interface InventoryRepository extends JpaRepository<Inventory, Long> {
    @Query("SELECT i FROM Inventory i WHERE i.skuCode = :skuCode")
    Long findBySkuCode(@Param("skuCode") String skuCode);
}
```

▶ 1:14:33



Screenshot of IntelliJ IDEA showing the implementation of the `InventoryService` class. The class has a private field `InventoryRepository inventoryRepository` and a public method `isInStock(String skuCode)` that returns a boolean. The method uses the `findBySkuCode()` method from the repository to check if the item is present.

```
package com.programmingtechie.inventoryservice.service;
import com.programmingtechie.inventoryservice.repository.InventoryRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
@Service
@RequiredArgsConstructor
public class InventoryService {
    private final InventoryRepository inventoryRepository;
    public boolean isInStock(String skuCode) {
        return inventoryRepository.findBySkuCode(skuCode).isPresent();
    }
}
```

▶ 1:14:58

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help

inventory-service src main java com programmingtechie inventoryservice service

application.properties InventoryJava InventoryRepositoryJava InventoryControllerJava InventoryServiceJava

Project Inventory-service F:\Tutorials\microservices\tutorial

idea mvn main src main java com.programmingtechie.inventoryservice controller model repository service

InventoryService InventoryController InventoryJava InventoryRepositoryJava InventoryControllerJava InventoryServiceJava

1 package com.programmingtechie.inventoryservice.service;

2

3 import com.programmingtechie.inventoryservice.repository.InventoryRepository;

4 import lombok.RequiredArgsConstructor;

5 import org.springframework.stereotype.Service;

6 import org.springframework.transaction.annotation.Transactional;

7

8 @Service

9 @RequiredArgsConstructor

10 public class InventoryService {

11

12 private final InventoryRepository inventoryRepository;

13

14 @Transactional(readOnly = true)

15 public boolean isInStock(String skuCode) {

16 return inventoryRepository.findBySkuCode().isPresent();

17 }

18 }

19

resources .gitignore HELP.md mvnw mvnw.cmd pom.xml

External Libraries

Scratches and Consoles

1:15:29

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help microservices-parent - OrderService.java [order-service]

order-service order-service src main java com programmingtechie orderservice service OrderService placeOrder

application.properties OrderService OrderRepository OrderRequest OrderLineItemsDto

Project microservices-parent F:\Tutorials\microservices\tutorial\revised

idea order-service src main java com.programmingtechie.orderservice controller dto model repository service

OrderService OrderServiceApplication

11 import java.util.List;

12 import java.util.UUID;

13

14 @Service

15 @RequiredArgsConstructor

16 @Transactional

17 public class OrderService {

18

19 private final OrderRepository orderRepository;

20

21 @Override

22 public void placeOrder(OrderRequest orderRequest) {

23 Order order = new Order();

24 order.setOrderNumber(UUID.randomUUID().toString());

25

26 List<OrderLineItems> orderLineItems = orderRequest.getOrderLineItemsDtoList();

27 Stream<OrderLineItems> stream = orderLineItems.stream();

28 stream.map(this::mapToDto).toList();

29

30 order.setOrderLineItemsList(orderLineItems);

31

32 orderRepository.save(order);

33

34 }

35

36 private OrderLineItems mapToDto(OrderLineItemsDto orderLineItemsDto) {

37 OrderLineItems orderLineItems = new OrderLineItems();

38 return orderLineItems;

39 }

40 }

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1266

1267

1268

1269

1270

1271

1272

1273

1274

1275

1276

1277

1278

1279

1280

1281

1282

1283

1284

1285

1286

1287

1288

1289

1290

1291

1292

1293

1294

1295

1296

1297

1298

1299

1300

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314</

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `InventoryController.java` file:

```
1 package com.programmingtechie.inventoryservice.controller;
2
3 import com.programmingtechie.inventoryservice.service.InventoryService;
4 import lombok.RequiredArgsConstructor;
5 import org.springframework.http.HttpStatus;
6 import org.springframework.web.bind.annotation.*;
7
8 @RestController
9 @RequestMapping("/api/inventory")
10 @RequiredArgsConstructor
11 public class InventoryController {
12
13     private final InventoryService inventoryService;
14
15     @GetMapping("/{sku-code}")
16     @ResponseStatus(HttpStatus.OK)
17     public boolean isInStock(@PathVariable("sku-code") String skuCode) {
18         return inventoryService.isInStock(skuCode);
19     }
20 }
```

▶ 1:16:33

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `InventoryServiceApplication.java` file:

```
1 package com.programmingtechie.inventoryservice;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7 import org.springframework.context.annotation.Primary;
8 import org.springframework.core.io.Resource;
9 import org.springframework.core.io.FileSystemResource;
10 import org.springframework.core.io.InputStreamResource;
11 import org.springframework.core.io.ClassPathResource;
12 import org.springframework.core.io.ResourceLoader;
13 import org.springframework.core.io.Resource;
14 import org.springframework.core.io.InputStreamResource;
15 import org.springframework.core.io.FileSystemResource;
16 import org.springframework.core.io.Resource;
17 import org.springframework.core.io.InputStreamResource;
18 import org.springframework.core.io.FileSystemResource;
19 import org.springframework.core.io.Resource;
20 import org.springframework.core.io.InputStreamResource;
21 import org.springframework.core.io.FileSystemResource;
```

▶ 1:17:31

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure for "inventory-service". It includes a "src" folder containing "main" and "test" packages, and a "resources" folder with "static" and "templates" subfolders. A file named "application.properties" is also listed.
- Code Editor:** The main window displays the "InventoryServiceApplication.java" file. The code is as follows:

```
1 package com.programmingtechie.inventoryservice;
2
3 import ...
4
5 @SpringBootApplication
6 public class InventoryServiceApplication {
7
8     public static void main(String[] args) { SpringApplication.run(InventoryServiceApplication.class, args); }
9
10    @Bean
11    public CommandLineRunner loadData(InventoryRepository inventoryRepository) {
12        return args -> {
13            Inventory inventory = new Inventory();
14            inventory.setSkuCode("iphone_13");
15            inventory.setQuantity(100);
16        };
17    }
18
19 }
20
21
22
23
24
25
26 }
```

- Toolbars and Status Bar:** The bottom of the screen features toolbars for "TODO", "Problems", "Debug", "Profiler", "Regex Tester", "Terminal", "SonarLint", "Services", "Build", "Endpoints", "Dependencies", "Spring", and "Event Log". The status bar at the bottom indicates: "Build completed successfully in 2 sec, 15 ms (3 minutes ago)".

▶ 1:17:55

The screenshot shows the IntelliJ IDEA interface with the following details:

- File Structure:** The left sidebar shows the project structure for "inventory-service". It includes a "src" folder containing "main" and "test" packages. "main" contains "java" (with "com.programmingtechie.inventory" package), "resources" (with "static" and "templates" folders), and "application.properties". "test" contains "target", ".gitignore", "HELP.md", "mvnw", "mvnw.cmd", "pom.xml", "External Libraries", and "Scratches and Consoles".
- Code Editor:** The main editor window displays the `InventoryServiceApplication.java` file. The code is as follows:

```
1 package com.programmingtechie.inventoryservice;
2
3 import ...
4
5 @SpringBootApplication
6 public class InventoryServiceApplication {
7
8     public static void main(String[] args) { SpringApplication.run(InventoryServiceApplication.class, args); }
9
10    @Bean
11    public CommandLineRunner loadData(InventoryRepository inventoryRepository) {
12        return args -> {
13            Inventory inventory = new Inventory();
14            inventory.setSkuCode("iphone_13");
15            inventory.setQuantity(100);
16        };
17    }
18
19 }
20
21
22
23
24
25
26 }
```

- Toolbars and Status Bar:** The bottom of the screen features a toolbar with icons for TODO, Problems, Debug, Profiler, Terminal, SonarLint, Services, Build, Endpoints, Dependencies, Spring, and Event Log. The status bar at the bottom indicates "Build completed successfully in 2 sec. 15 ms (3 minutes ago)".

▶ 1:18:01

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** The project is named "inventory-service". It contains a "main" package with "com.programmingtechie.inventoryservice" and sub-packages "controller", "model", "repository", and "service".
- Code Editor:** The main file is "InventoryServiceApplication.java". The code is as follows:

```
package com.programmingtechie.inventoryservice;
import ...
@SpringBootApplication
public class InventoryServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(InventoryServiceApplication.class, args);
    }
    @Bean
    public CommandLineRunner loadData(InventoryRepository inventoryRepository) {
        return args -> {
            Inventory inventory = new Inventory();
            inventory.setSkuCode("iphone_13");
            inventory.setQuantity(100);

            Inventory inventory1 = new Inventory();
            inventory1.setSkuCode("iphone_13_red");
            inventory1.setQuantity(0);
        };
    }
}
```

- Toolbars and Menus:** Standard IntelliJ IDEA menus like File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help are visible at the top.
- Bottom Status Bar:** Shows "Build completed successfully in 2 sec, 15 ms (3 minutes ago)".
- Event Log:** A small icon in the bottom right corner indicates an event log is present.

▶ 1:18:12

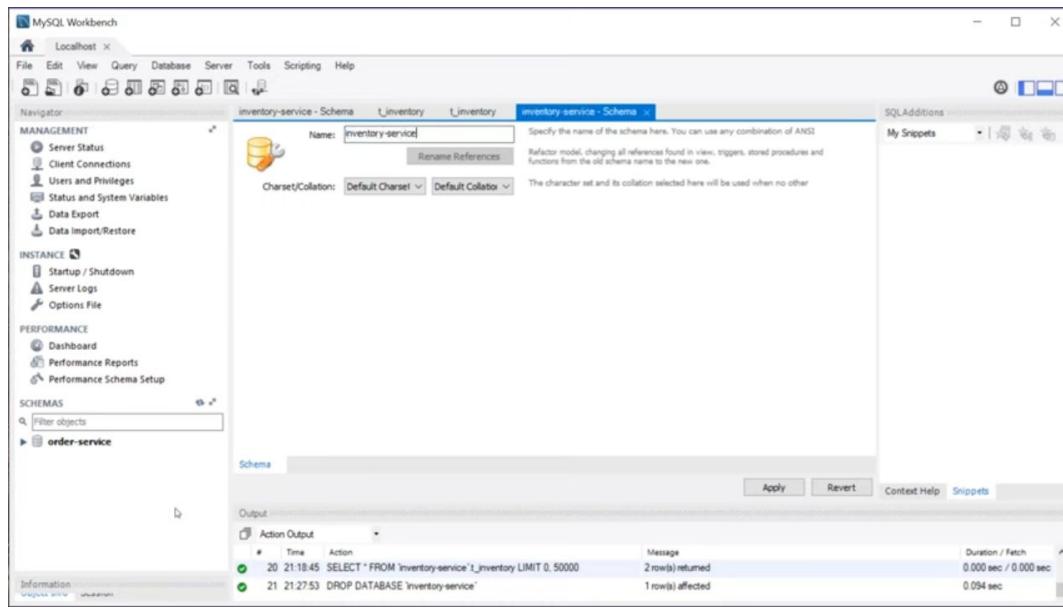
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** Shows the project structure under "inventory-service".
- Editor:** Displays the `InventoryServiceApplication.java` file content.
- Toolbars:** Includes "File", "Edit", "View", "Navigate", "Code", "Bafactor", "Build", "Run", "Tools", "VCS", "Window", "Help".
- Bottom Bar:** Includes "TODO", "Problems", "Debug", "Profiler", "Regex Tester", "Terminal", "SonarLint", "Services", "Build", "Endpoints", "Dependencies", "Spring", and "Event Log".

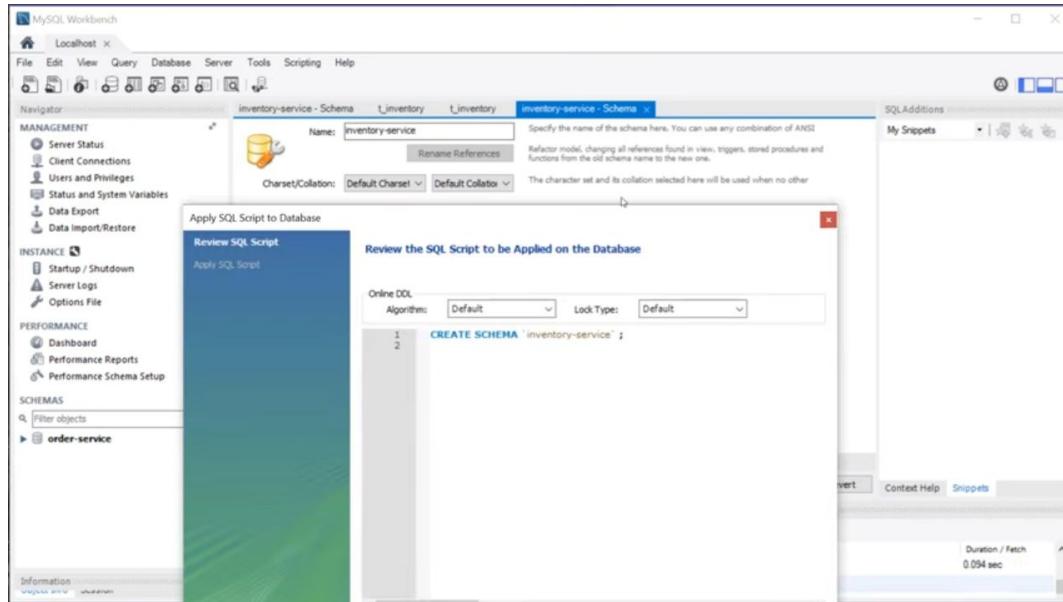
```
1 package com.programmingtechie.inventoryservice;
2
3 import ...
4
5 @SpringBootApplication
6 public class InventoryServiceApplication {
7
8     public static void main(String[] args) { SpringApplication.run(InventoryServiceApplication.class, args); }
9
10    @Bean
11    public CommandLineRunner loadData(InventoryRepository inventoryRepository) {
12        return args -> {
13            Inventory inventory = new Inventory();
14            inventory.setSkuCode("iphone_13");
15            inventory.setQuantity(100);
16
17            Inventory inventory1 = new Inventory();
18            inventory1.setSkuCode("iphone_13_red");
19            inventory1.setQuantity(0);
20
21            inventoryRepository.save(inventory);
22            inventoryRepository.save(inventory1);
23        };
24    }
25
26    @Override
27    public void configure(WebServerFactoryBuilder builder) {
28        builder.addPort(8081);
29    }
30
31 }
```

A yellow warning icon is present near the bottom of the code editor, around line 28.

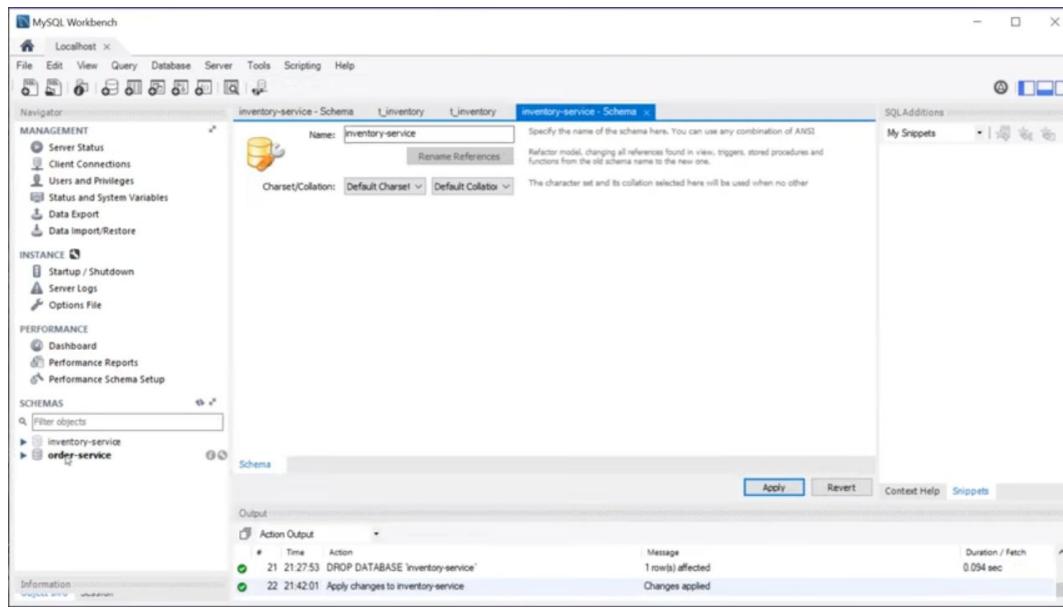
▶ 1:18:31



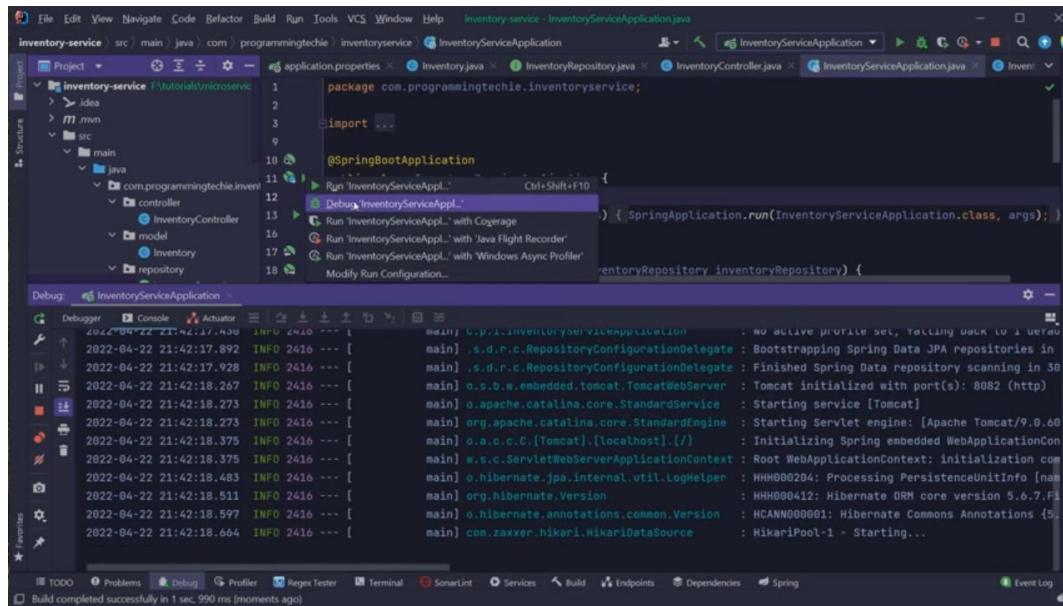
▶ 1:18:59



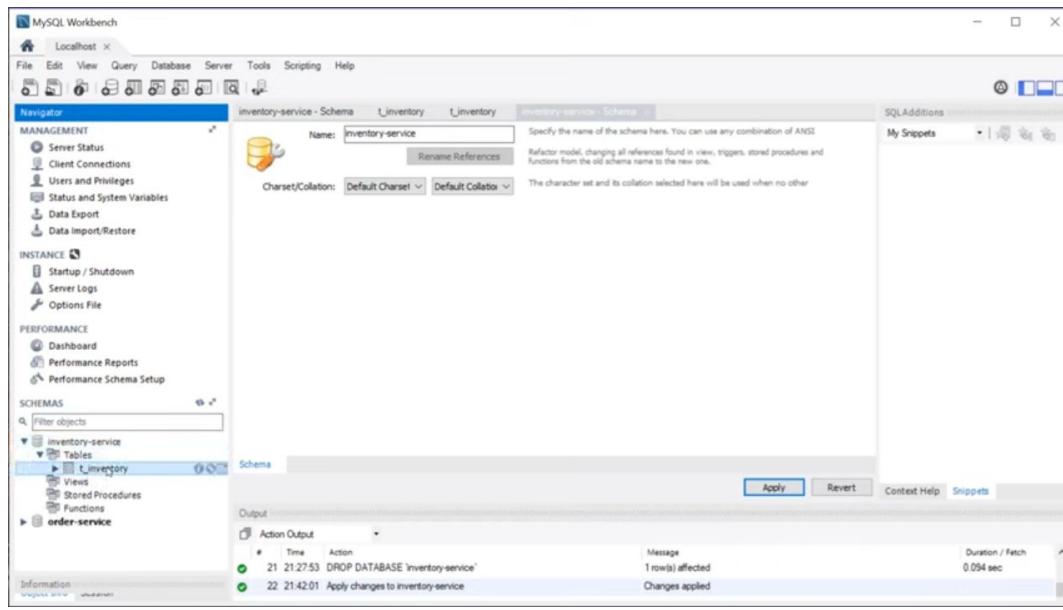
▶ 1:19:02



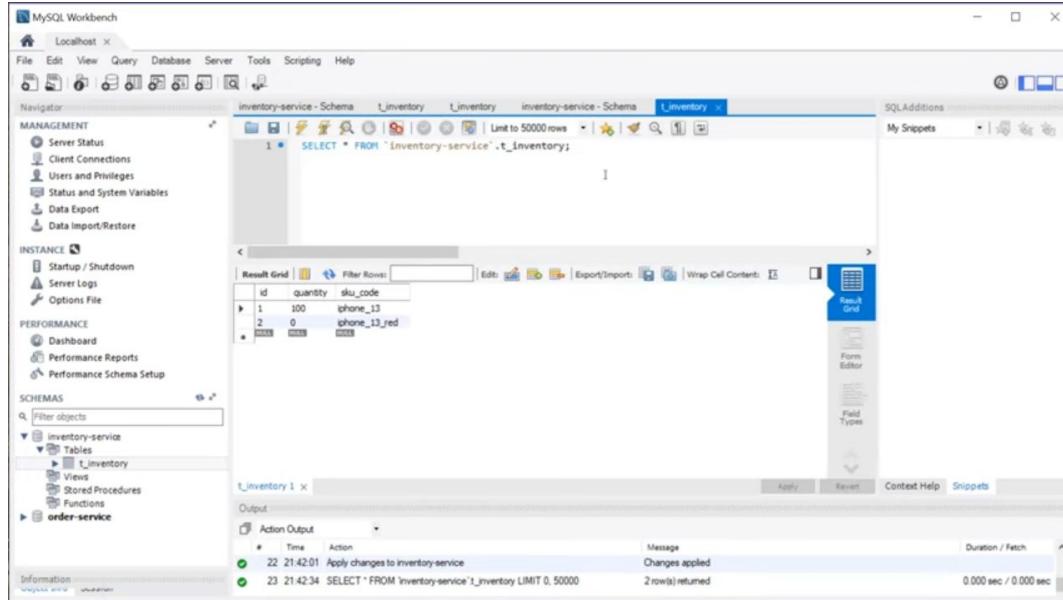
▶ 1:19:06



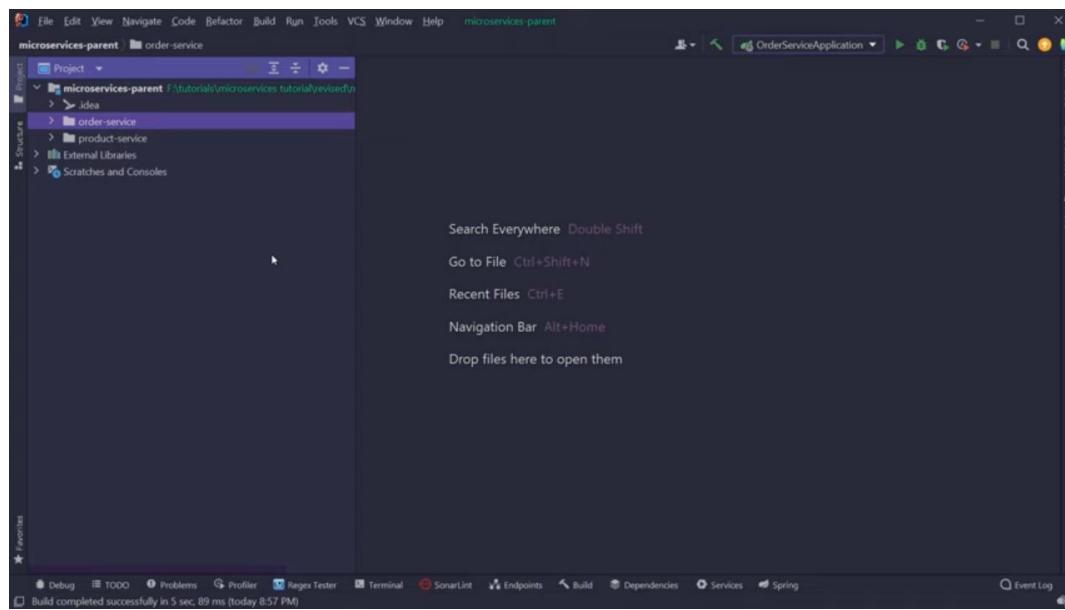
▶ 1:19:19



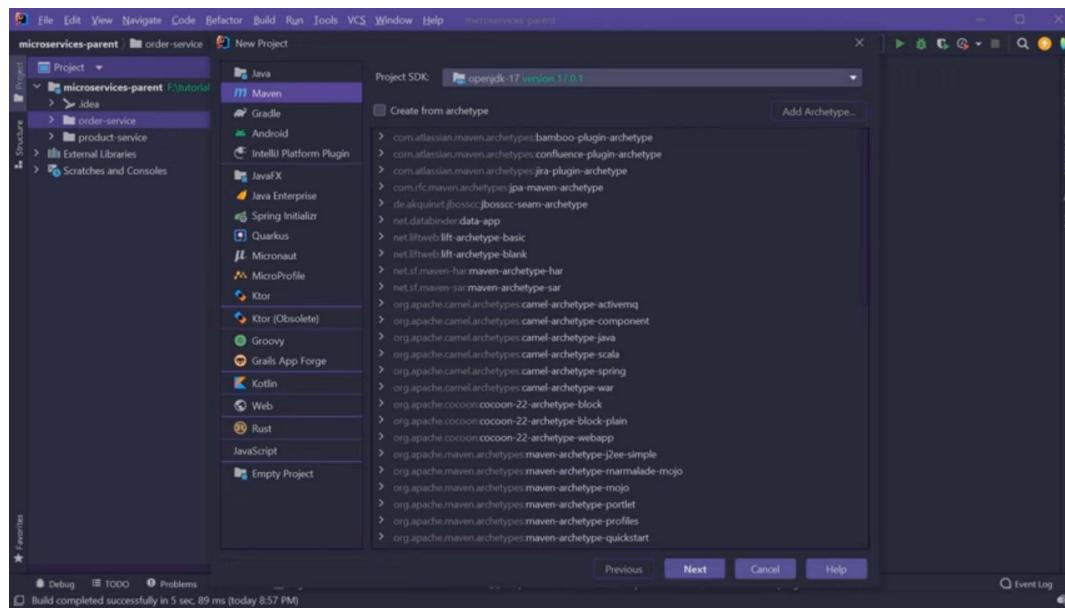
▶ 1:19:31



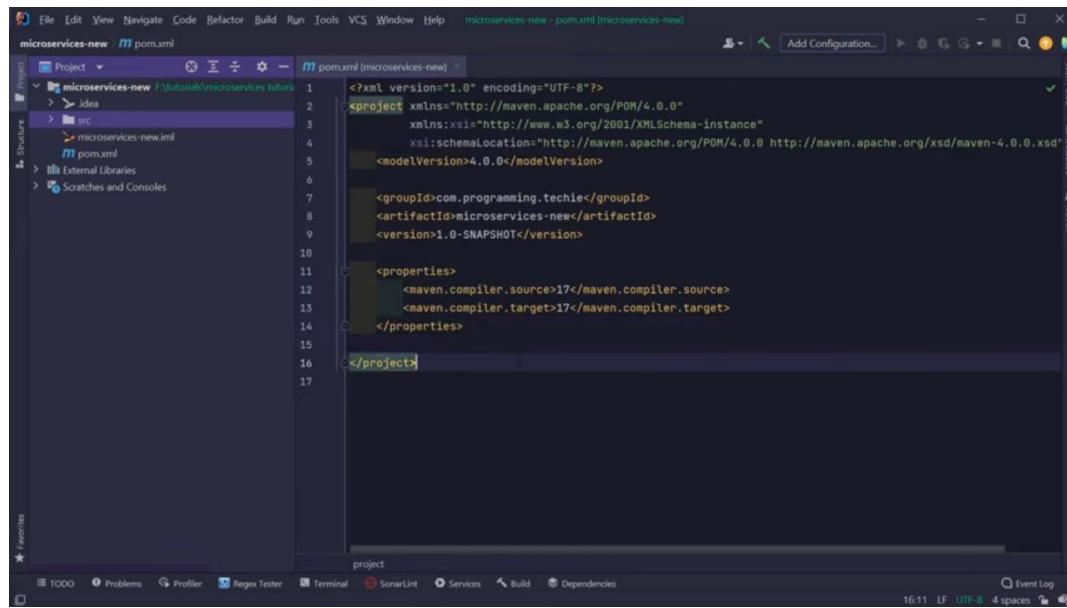
▶ 1:19:36



▶ 1:19:52



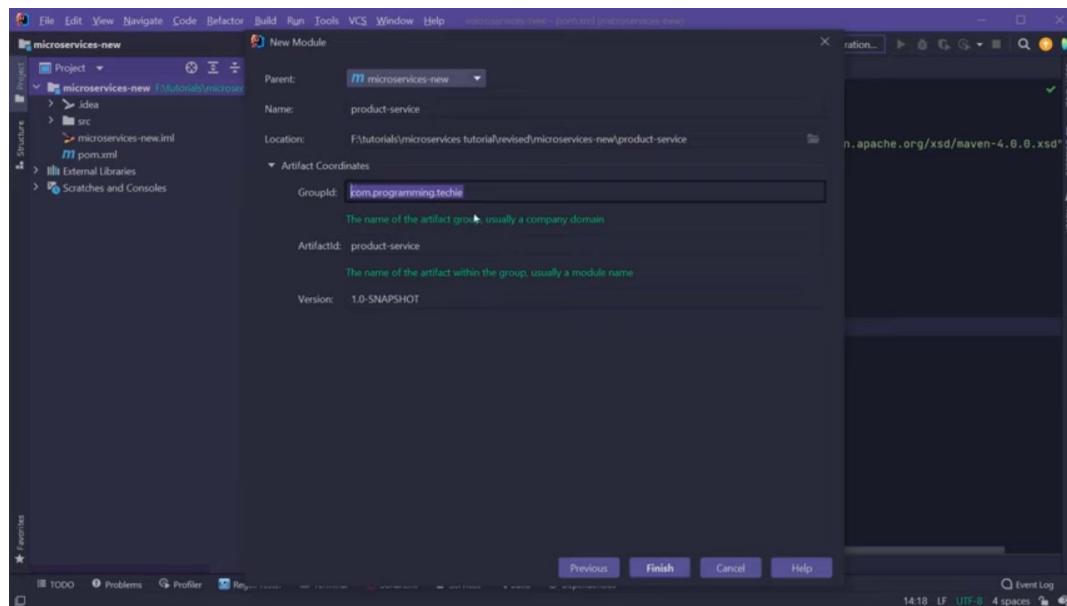
▶ 1:20:20



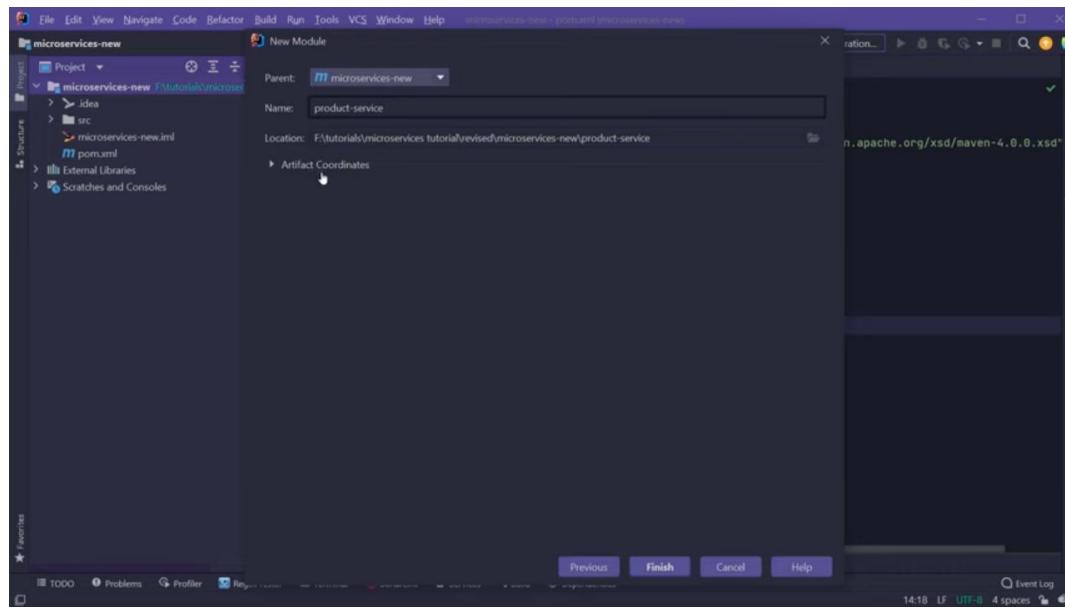
The screenshot shows the IntelliJ IDEA interface with the project 'microservices-new' open. The 'pom.xml' file is the active editor. The code displays the Maven POM configuration for the project.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.programming.techie</groupId>
    <artifactId>microservices-new</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
</project>
```

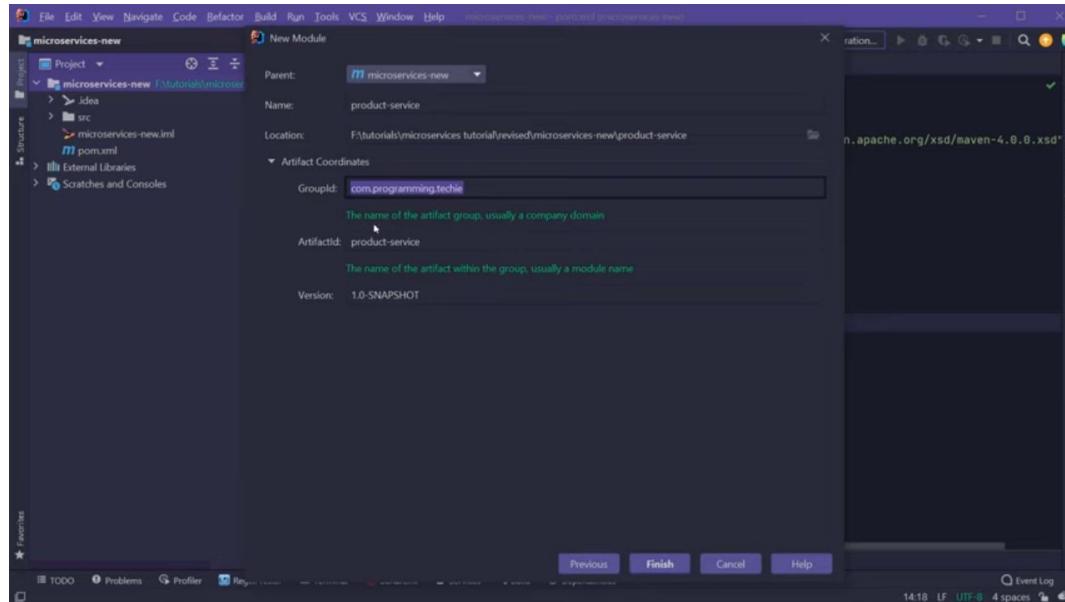
▶ 1:21:22



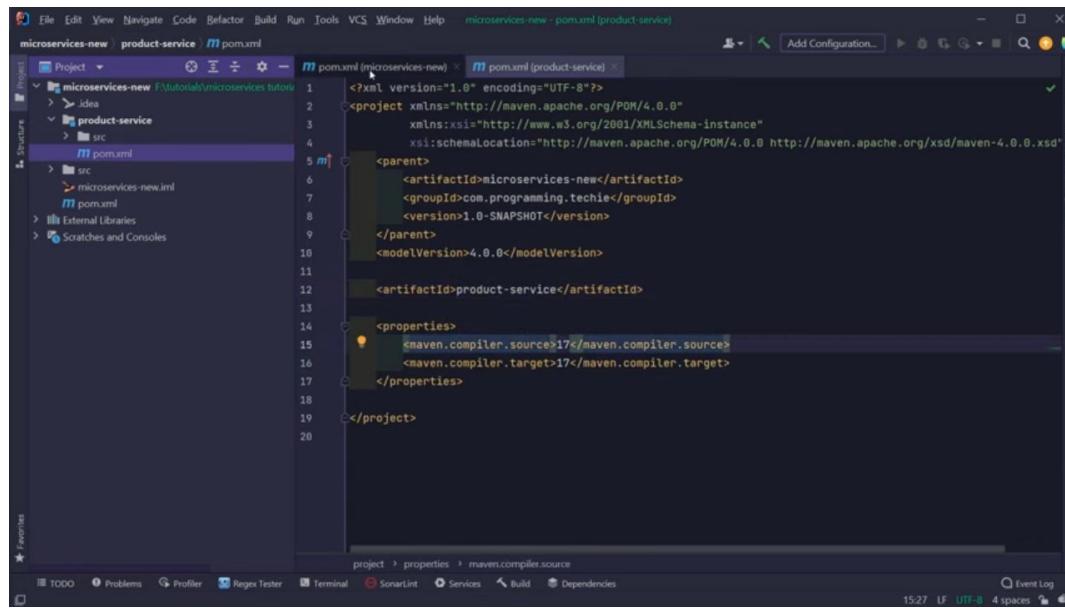
▶ 1:21:50



▶ 1:21:53

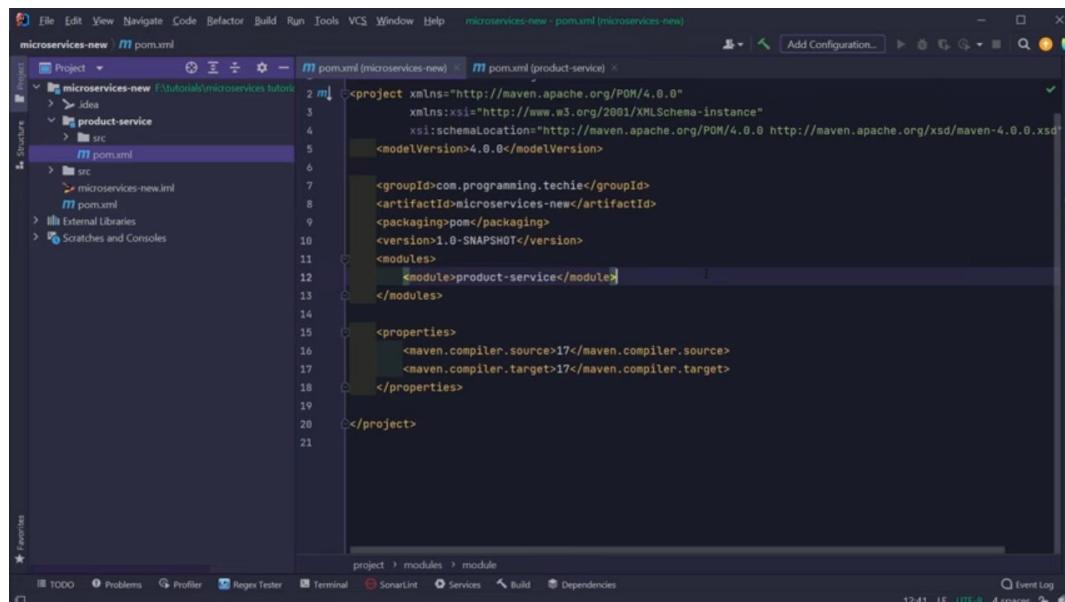


▶ 1:21:59



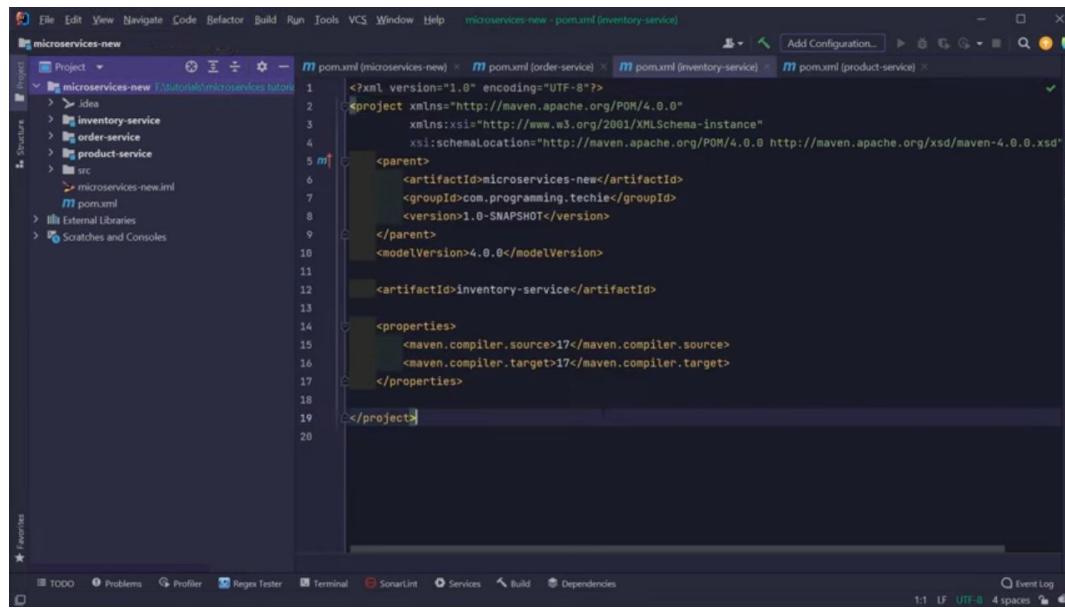
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>microservices-new</artifactId>
        <groupId>com.programming.techie</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>product-service</artifactId>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
</project>
```

▶ 1:22:16



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.programming.techie</groupId>
    <artifactId>microservices-new</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>product-service</module>
    </modules>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
</project>
```

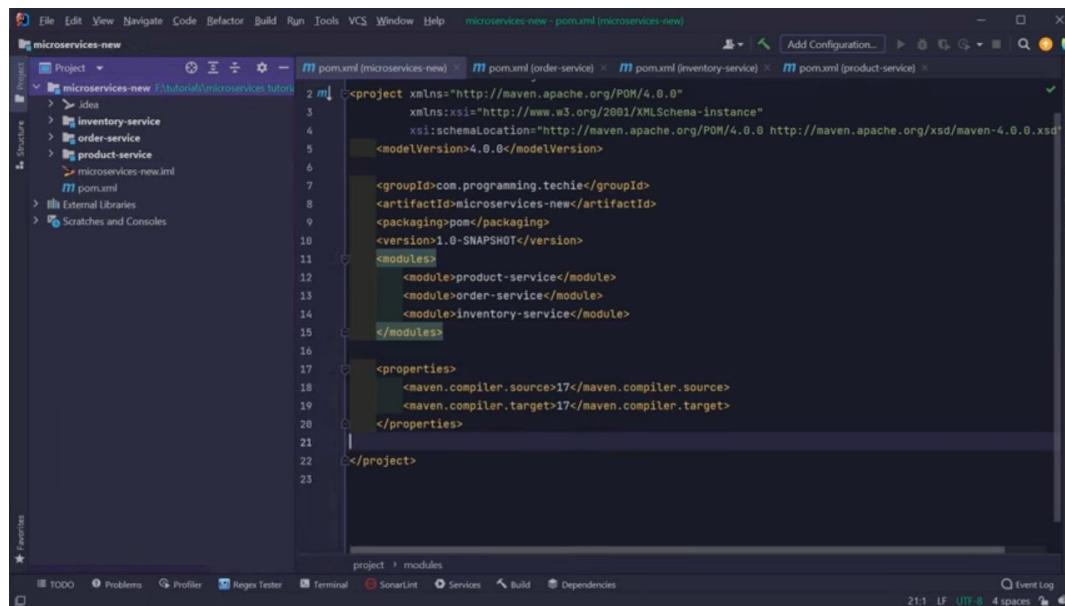
▶ 1:22:24



The screenshot shows the IntelliJ IDEA interface with the project 'microservices-new' open. The pom.xml file for the 'inventory-service' module is being edited. The code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>microservices-new</artifactId>
        <groupId>com.programming.techie</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>inventory-service</artifactId>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
</project>
```

▶ 1:22:42



The screenshot shows the IntelliJ IDEA interface with the project 'microservices-new' open. The pom.xml file for the root project is being edited. The code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.programming.techie</groupId>
    <artifactId>microservices-new</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>product-service</module>
        <module>order-service</module>
        <module>inventory-service</module>
    </modules>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
</project>
```

▶ 1:22:49

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.programming.techie</groupId>
    <artifactId>microservices-new</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
</project>
```

▶ 1:22:57

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.programming.techie</groupId>
    <artifactId>microservices-new</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <modules>
        <module>product-service</module>
        <module>order-service</module>
        <module>inventory-service</module>
    </modules>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
</project>
```

▶ 1:22:58

delete src folder

▶ 1:23:03

```
<dependency>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>mongodb</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>junit-jupiter</artifactId>
    <scope>test</scope>
</dependency>
```

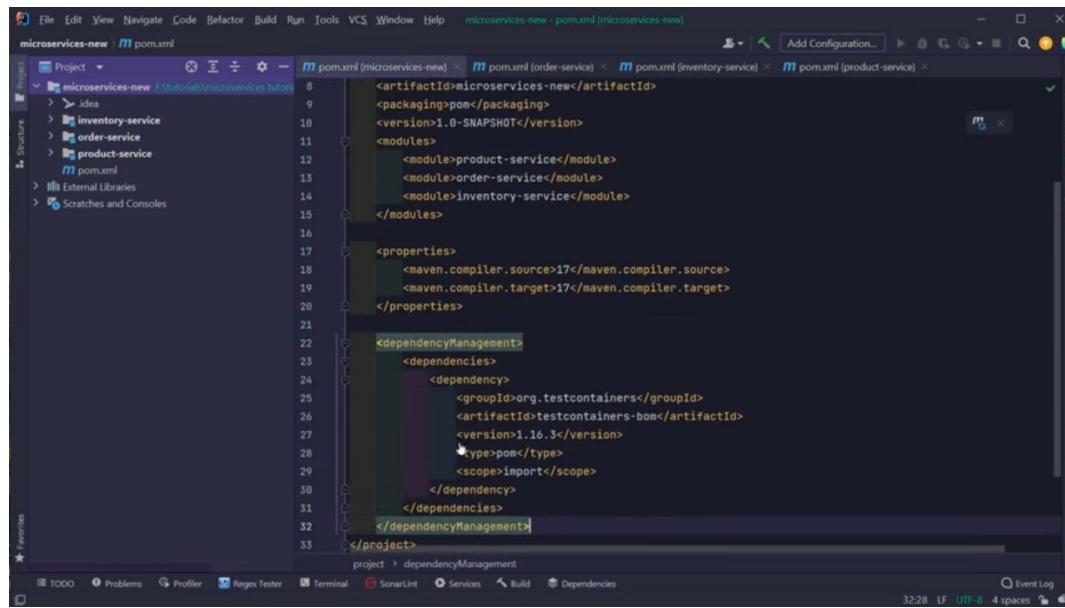
Build completed successfully in 5 sec, 89 ms (today 8:57 PM)

▶ 1:23:43

```
</dependencies>
```

Build completed successfully in 5 sec, 89 ms (today 8:57 PM)

▶ 1:23:54

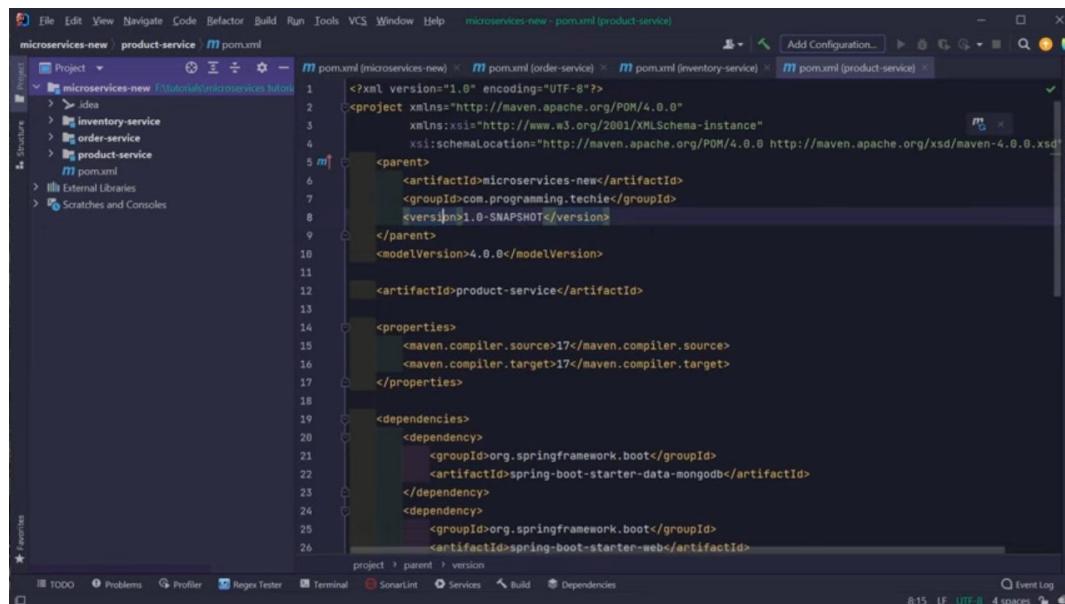


```
<artifactId>microservices-new</artifactId>
<packaging>pom</packaging>
<version>1.0-SNAPSHOT</version>
<modules>
    <module>product-service</module>
    <module>order-service</module>
    <module>inventory-service</module>
</modules>

<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
</properties>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>testcontainers-bom</artifactId>
            <version>1.16.3</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
</project>
```

▶ 1:25:50



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>microservices-new</artifactId>
        <groupId>com.programming.techie</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>product-service</artifactId>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
</project>
```

▶ 1:27:05

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** Shows "microservices-new" as the active project.
- Pom.xml Content:** The code editor displays the `pom.xml` file for the `microservices-new` project. The XML code includes definitions for a parent dependency on `spring-boot-starter-parent`, three child modules (`product-service`, `order-service`, and `inventory-service`), and compiler properties.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.6</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.programming.techie</groupId>
  <artifactId>microservices-new</artifactId>
  <packaging>pom</packaging>
  <version>1.0-SNAPSHOT</version>
  <modules>
    <module>product-service</module>
    <module>order-service</module>
    <module>inventory-service</module>
  </modules>
  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

```

- Toolbars and Status Bar:** Includes standard IntelliJ toolbars and a status bar at the bottom showing "13:5 LF 177E-B 4 spaces".

▶ 1:27:26