# Error Handling

## Exception Responses, Exception Handlers, Global Handlers


**SoftUni Team**

**Technical Trainers**

Software University

SoftUni

Software University

https://softuni.bg

# Questions

Software University

sli.do

# #java-web

# Table of Contents



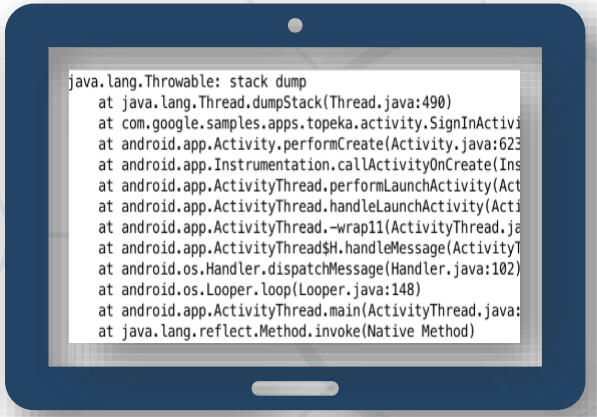
# Error Handling

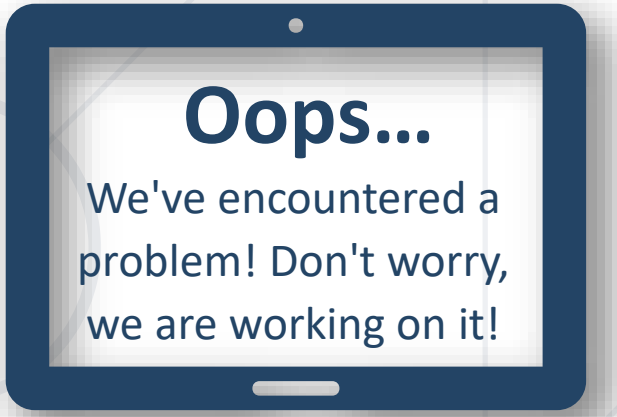
Anticipate! Detect! Resolve!

# Error Handling

- **Error handling** refers to:
  - The **anticipation**, **detection** and **resolution** of programming errors
  - The response & recovery procedures from error conditions
- Error handling is necessary!
  - Improves **user experience**
  - **Optimizes** debugging
  - Facilitates **code maintenance**
  - Ensures **product quality**

# Error Handling Example




# Error Handling in Spring

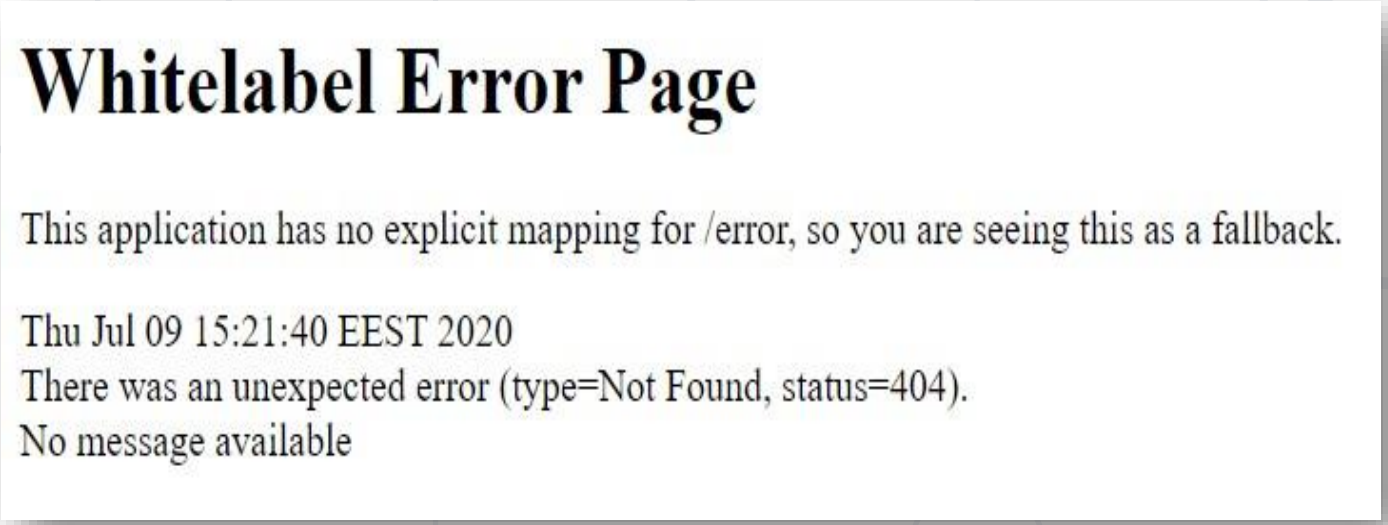
- Spring MVC offers **no default** (fall-back) **error page** out of the box, however Spring Boot does
- At start-up, Spring Boot **tries to find** a mapping for **/error**
- Spring MVC provides **several approaches** to error handling
  - Per exception
  - Per controller
  - Globally

# Error Handling in Spring

- Each option has its own use cases and circumstances
- You can use:
  - **Response-annotated** custom exceptions
  - **Controller-based** handlers on specified actions
  - **@ControllerAdvise** annotated classes for global handlers

# Custom error page

- To disable the default **White label error page** for a Spring Boot application:
  - We must save **error.html** file in resources/templates directory, it'll automatically be picked up by the default Spring




# ErrorController Interface

- Spring Boot maps **/error** to **BasicErrorController** which populates model with error attributes and then returns 'error' as the view name
- To replace BasicErrorController with our own custom controller which can map to /error, we need to **implement ErrorController** interface

# ErrorController Interface

```
@Controller
public class MyErrorController implements ErrorController {
    @RequestMapping
    @ResponseBody
    public String handle(HttpServletRequest request){
        // Some code ...
    }
}
```

# HTTP Status Codes

## Annotated Custom Exceptions

# HTTP Status Codes

- Unhandled exceptions during a request produce HTTP 500 response
- Any custom exception can be annotated with **@ResponseStatus**
  - Supports all HTTP status codes
  - When thrown and unhandled – produces error page with appropriate response

```
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Product was not found.")
public class ProductNotFoundException extends RuntimeException {
   // Exception definition
}
```

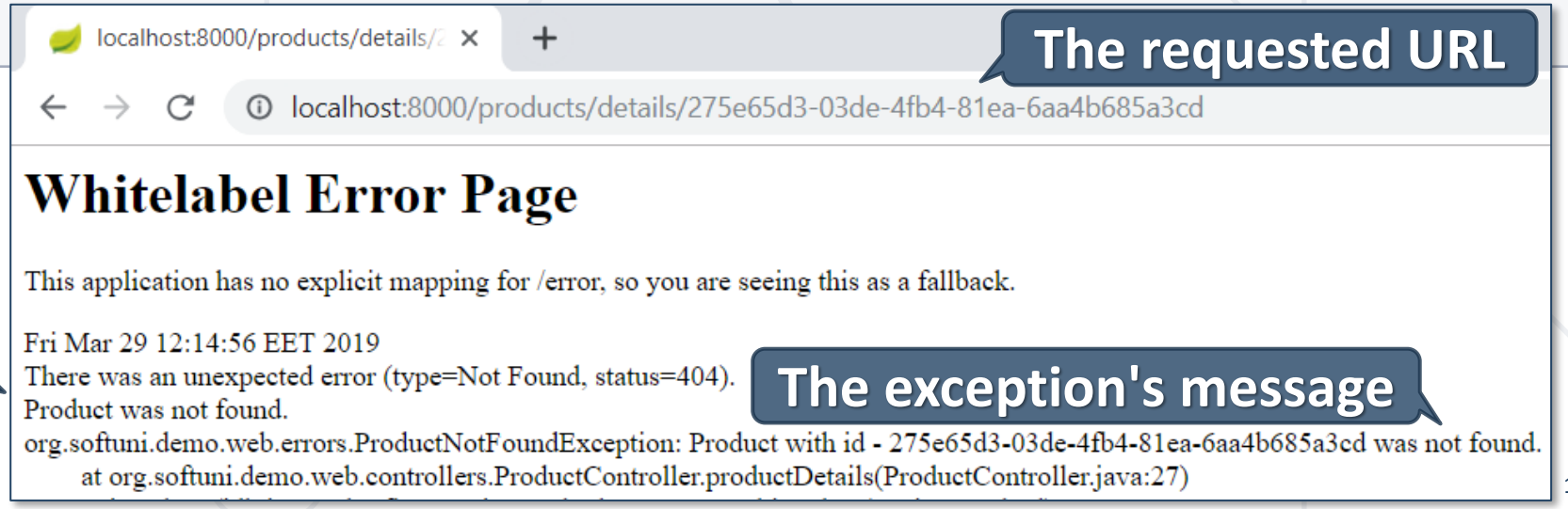
# HTTP Status Codes

- And the controller action, throwing the exception

```
@GetMapping("/products/details/{id}")
public ModelAndView productDetails(@PathVariable String id, ModelAndView modelAndView) {
   Product product = this.productRepository.findProductById(id);

   if(product == null) throw new ProductNotFoundException(id);

   modelAndView.addObject("product", product);
   return this.view("product/details", modelAndView);
}
```




The requested URL

The produced HTTP Status & Message

The exception's message

# Controller-Based Error Handling

## Exceptions & Views

# Controller-Based Error Handling

- You can define Controller-specific Exception Handlers
  - Annotated with **@ExceptionHandler** annotation
  - They work **only** for the **Controller** they are defined in
  - Can be annotated with **@ResponseStatus** to convert HTTP status
  - Can accept the **caught exception** as a **parameter**
  - Can return **ModelAndView** or **String** (view name)
  - Can catch **multiple** exception types

# Controller-Based Error Handling

```
@ExceptionHandler({PersistenceException.class,
TransactionException.class})
public ModelAndView handleDbExceptions(DatabaseException e) {
    ModelAndView modelAndView = new ModelAndView("error");
    modelAndView.addObject("message", e.getMessage());

    return modelAndView;
}
```

Parent Exception

```
<html>
<head>...</head>
<body>
    <h1>An error occurred while processing your request!</h1>
    <p th:text="|Error: ${message}|"></p>
</body>
</html>
```

**An error occurred while processing your request!**

Error: Database server is down.

# Controller-Based Error Handling

- Handler methods have **flexible signatures**
  - You can pass in servlet-related objects as parameters
    - **HttpServletRequest**
    - **HttpServletResponse**
    - **HttpSession**
    - **Principal**

# Controller-Based Error Handling

- The **Model** or **ModelAndView** cannot be a parameter though
  - Instead of passing it, you have to setup it inside the method
  - Nevertheless, this is not an issue because the **IoC container** would have done the same (pass an **empty instance**)

# Controller-Based Error Handling

- It is not a good practice for full error **stacktraces** to be exposed
  - Your users don't need to see ugly exception web-pages
  - You may even have security policies which **strictly forbid** any public exception info
  - Hide as **much information** as **possible** and present **User-friendly** error pages
  - For **testing** purposes you may view details
    - This may need an **environment** setup

# Global Application Exception Handling

@ControllerAdvice Classes

# Global Exception Handling

- There is a way to achieve Global exception handling in Spring
  - This is done through the **@ControllerAdvise** annotation
- Any class annotated with **@ControllerAdvise** turns into an **interceptor-like** controller:
  - Enables **global exception handling**
  - Enables **model enhancement** methods

# Global Exception Handling

- In **@ControllerAdvice** classes you still use **@ExceptionHandler**
  - However, this time it refers to the whole application
  - The error handling is not limited only to a specific controller

```
@ControllerAdvice
public class GlobalExceptionHandler {
   @ExceptionHandler({TransactionException.class, PersistenceException.class})
   public ModelAndView handleDatabaseErrors(DatabaseException e) {
      ModelAndView modelAndView = new ModelAndView("index");
      modelAndView.addObject("message", e.getMessage());
      modelAndView.addObject("stack", {...} /* Formatted Stack Trace */);

      return modelAndView;
  }}
```

# Global Exception Handling (REST)

- **RESTful requests** may also generate unexpected exceptions
  - HTTP Error response codes are a good choice
  - However sometimes you might need more than just a status
    - **Customized Error Object**, which can be presented on the Client
    - **Limited Information** returned to the Client

# Global Exception Handling (REST)

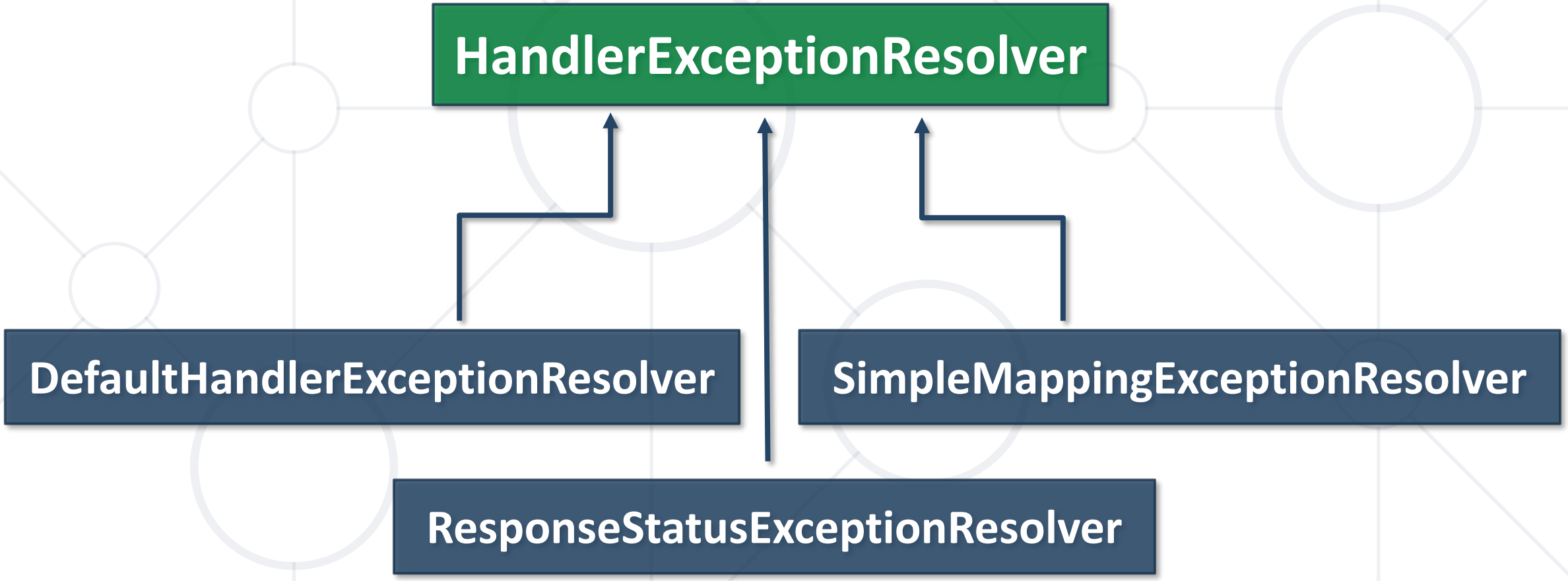
- You can customize the **Error Response** by introducing a class
  - The **Error Handler** itself remains the same as in casual web apps

```
public class ErrorInfo {
    public final String url;
    public final String ex;
    public ErrorInfo(String url, Exception ex) {
        this.url = url;
        this.ex = ex.getLocalizedMessage();
    }
}
```

# Global Exception Handling (REST)

```
@ControllerAdvice
public class GlobalRESTExceptionHandler {
   @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
   @ExceptionHandler({TransactionException.class,
                      PersistenceException.class})
   public @ResponseBody ErrorInfo handleRESTErrors(HttpServletRequest req,
                                                   DbException e) {
      return new ErrorInfo(req.getRequestURL(), ex);
   }
}
```

# HandlerExceptionResolver Interface



# Exception Techniques Use Cases

## What to Use When?

# What to Use When?

- Spring offers **many** choices, when it comes to **error** handling
- Be **careful mixing** too **many** of these
  - You may not get the behavior you wanted
- There are some semantics, that should be followed, though

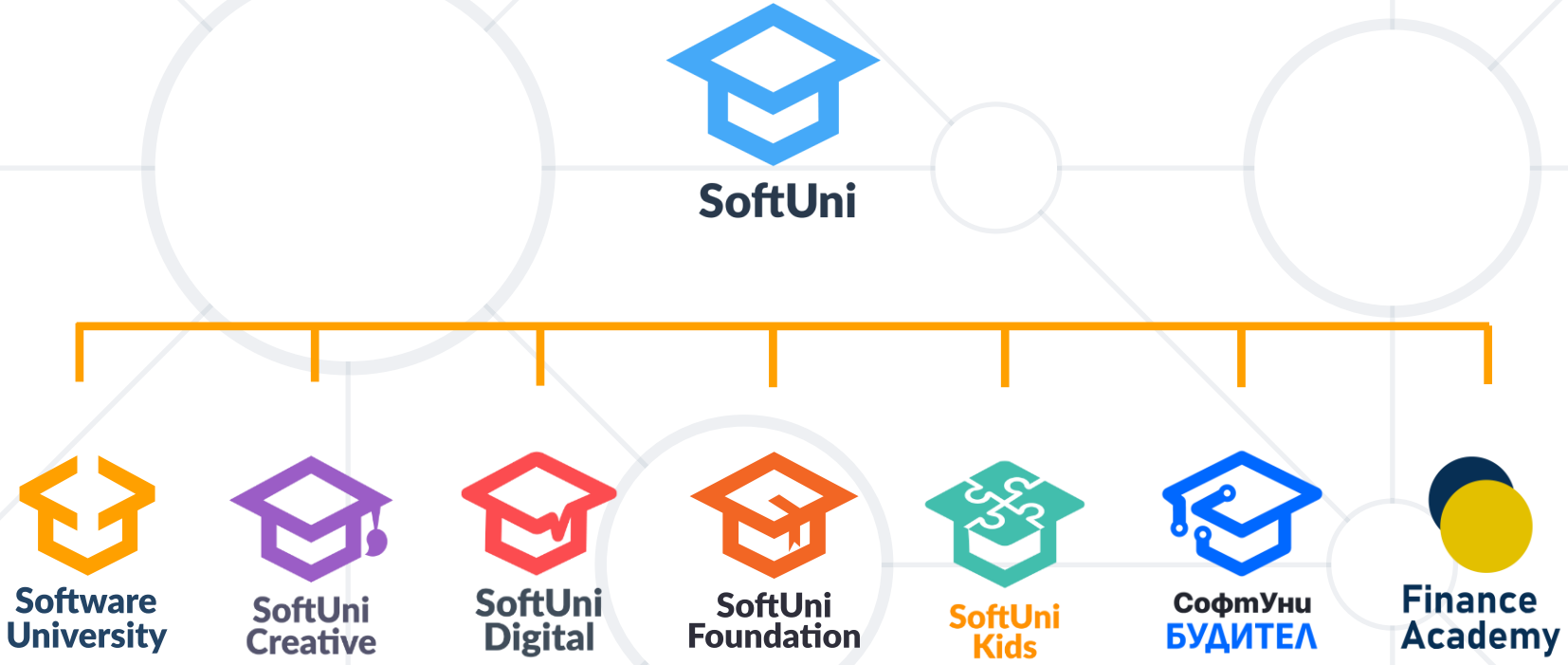
# Exception techniques use cases

- For custom exceptions, consider adding **@ResponseStatus** to then
- For Controller-specific exceptions, **@ExceptionHandler** methods should be added alongside the actions
- For all other exceptions, **@ExceptionHandler** methods in **@ControllerAdvice** classes should be implemented

# Summary

- Error Handling is essential
  - Improves User Experience
  - Improves Application maintenance
- Exception Responses
- Controller-based Exception
- Global Application Exception Handling
- Exception techniques use cases

# Questions?



# SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank
Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR
Serving the high achievers

SOFTWARE GROUP

PHAR VISION

# Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

# License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg