# Spring Boot: Efficient Development, Configuration, and Deployment

Adding PDF Generation Support to a Spring Boot App with Autoconfiguration



#### **Federico Mestrone**

Software Engineer and Training Consultant

@fedmest www.federicomestrone.com

## Target Audience

Have good knowledge of Spring Boot
Have previously developed with it

#### Would like to:

- Be more productive
- Learn advanced features
- Understand internal workings
- Deploy to scale on cloud

## Project Overview

### Learn through practice

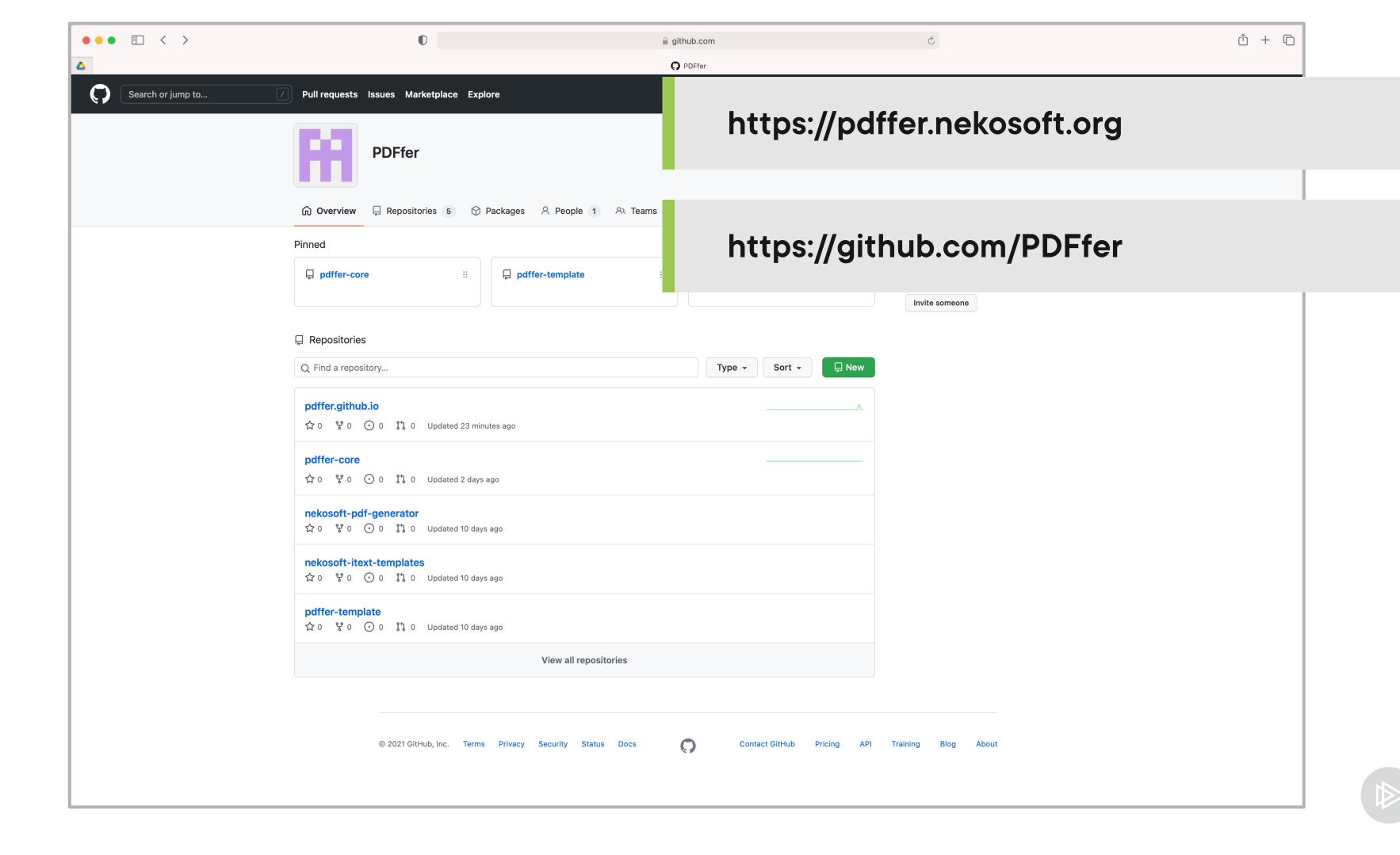
Developing PDFfer

### It's a Spring Boot library that:

- Adds PDF generation capabilities
- By simply adding a dependency

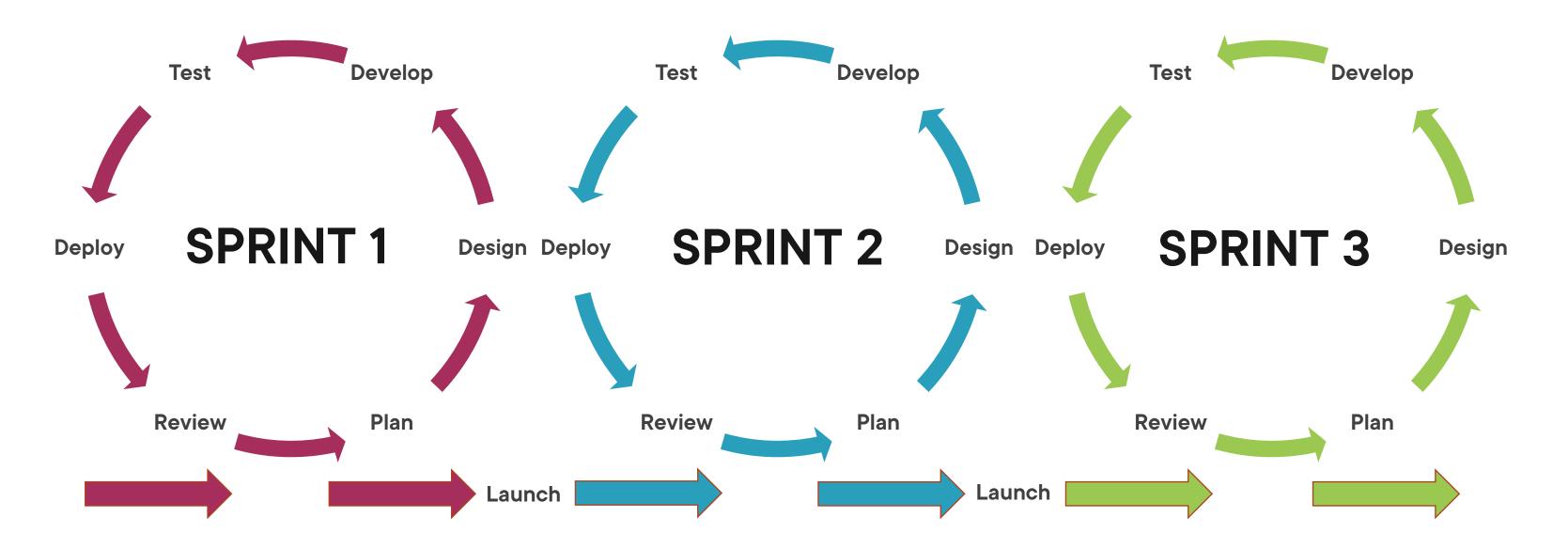
#### The library includes:

- Injectable beans to generate PDFs
- A flexible extensible template system
- Mail senders to attach PDFs
- HTTP endpoints to download PDFs

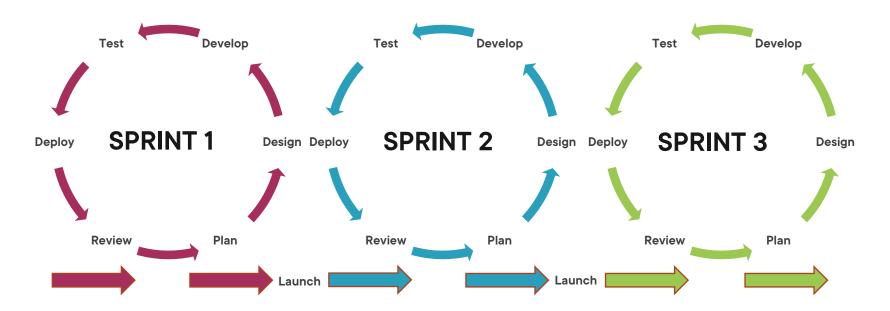




# **AGILE**



#### **AGILE**



### Adding PDF generation support to a Spring Boot app with autoconfiguration

Implementing a PDF template registry with subcontexts and custom scanners

Adding an HTTP API and email capabilities with conditional configuration and beans

Externalizing configuration with properties and YAML files

Offering a stand-alone mode for our Spring Boot library

**Deploying Spring Boot applications** 



# Spring Initializr



A web-based tool that allows you to generate a skeleton Spring Boot application easily and effectively



Maven

Possibly first proper Java build tool

**Build files written in XML** 

Very popular

Well supported by all development tools



Gradle

More recent build tool

**Build files written in Groovy or Kotlin** 

More flexible and synthetic than Maven

- Also known for being faster

Possibly a steeper learning curve



# Spring Boot IDEs

### **Spring Tools 4**

- Eclipse
- Visual Studio Code
- Theia

**IntelliJ Ultimate Edition** 

**NetBeans** 



### Library Overview



#### In this module

- We will start development of PDFfer

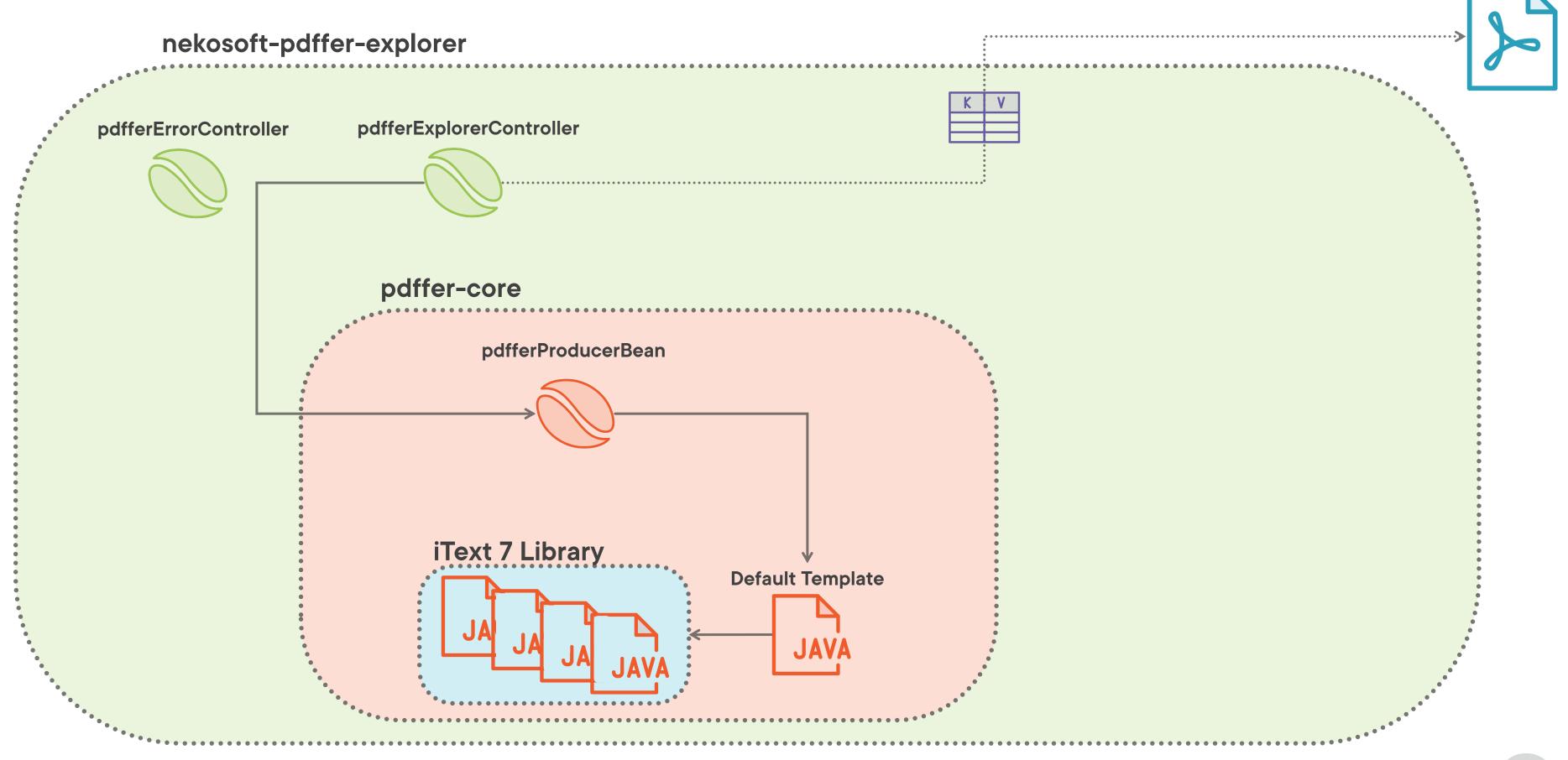
### **Basic library**

- No frills
- Not extensible
- Not configurable
- But pluggable



# The Initial Project





### Spring Configuration Class

```
package org.nekosoft.pdffer;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
@Configuration
@ComponentScan
public class PdfferCoreConfiguration {
```



### PDF Producer Bean

```
@Component
public class PdfferProducerBean {
    public byte[] generatePdfDocument(String templateName, Map<String, Object> data) {
        PdfTemplate template = findTemplate(templateName);
        template.setPdfData(data);
        if (!template.validate()) {
            throw new IllegalArgumentException("PDF Template payload is not valid");
        template.generate();
        return template.getPdfContent();
    PdfTemplate findTemplate(String templateName) {
        return new DefaultPdfTemplate();
```

### The PDF Template Inteface

```
package org.nekosoft.pdffer.template;
import java.util.Map;
public interface PdfTemplate {
    Map<String, Object> getPdfData();
    void setPdfData(Map<String, Object> data);
    boolean validate();
    void generate();
    byte[] getPdfContent();
}
```

# The iText 7 Library

A library to programmatically create and edit PDF documents, from very simple to very complex ones, available in both Java and .NET



### iText Licensing

### **Dual licensing**

- Affero General Public License (AGPL)
  - Highlight any changes you make to iText original source code
  - Retain iText copyright and producer information in output metadata
  - Your application and source code must be distributed under AGPL too
- Commercial license
  - Get a quote from iText to remove all free license restrictions



# Apache OpenPDF

Less restrictive LGPL license

Based on older version of iText



# The Client Project

# What's Going On?

# Project finds dependency, compiles correctly, and it runs

But it cannot find the PDFProducerBean

#### Beans must be

- Explicitly created
- Indirectly loaded
  - E.g. component scanner

### Nothing creates the PDFProducerBean here

#### **Create it manually?**

- No! That's not cool!
- We need Autoconfiguration!!!



# Spring Autoconfiguration



"... attempts to automatically configure your Spring application based on the jar dependencies that you have added"



# Enabling Autoconfiguration

# Auto-configuration is defined in the source project

- With the spring.factories file
- In the META-INF folder of a JAR

# Auto-configuration must be enabled in the target application

- Either with @EnableAutoconfiguration
- Or with @SpringBootApplication

### The spring.factories File

```
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.nekosoft.pdffer.PdfferCoreConfiguration
```



### Spring Configuration Class

```
package org.nekosoft.pdffer;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
@Configuration
@ComponentScan
public class PdfferCoreConfiguration {
```



### Spring Boot DevTools

### Property defaults and global properties

- Defines some defaults useful in dev
  - E.g. disabling caching
- Global properties across projects
  - In .spring-boot-devtools.properties under user's home directory

#### **Automatic restart**

- Whenever changes are made

#### Live reload

Of browser pages when plugin installed

### Remote debugging and updates

With appropriate server, package and IDE configurations



### Summary



### Effective development

- Spring Initialzr
  - From the web, console, or IDE

### **Effective configuration**

- Spring Boot autoconfiguration
  - The spring.factories file

### **Effective deployment**

- Spring Dev Tools
  - Auto-reload of source code changes



## Up Next:

Implementing a PDF Template Registry with Subcontexts and Custom Scanners