Logging für deinem Spring Boot Webservice

Schritt 1: Logging-Abhängigkeit prüfen

Spring Boot bringt bereits eine Standard-Logging-Konfiguration mit, die auf SLF4J und Logback basiert. Die Spring Boot Starter (z.B. `spring-boot-starter-web`) enthalten SLF4J und Logback, sodass keine zusätzlichen Abhängigkeiten benötigt werden.

Schritt 2: Logger konfigurieren

Du kannst in jeder Klasse einen `Logger` hinzufügen. Dieser wird über `LoggerFactory` erstellt und bietet alle notwendigen Log-Level (`INFO`, `DEBUG`, `WARN`, `ERROR`).

Beispiel für deine Controller-Klasse `ZinsRechnerController`:

1. Importiere die Logger-Klassen und füge den Logger hinzu:

```
java
Code kopieren
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class ZinsRechnerController {
    private static final Logger logger =
LoggerFactory.getLogger(ZinsRechnerController.class);

    // ... bestehende Methoden ...
}
```

2. Ersetze die `System.out.println`-Ausgabe in der `berechne`-Methode durch ein Logging:

```
java
Code kopieren
@PostMapping("/berechne")
public String berechne(@ModelAttribute ZinsRechner zinsRechner, Model
model) {
    logger.info("Berechnung gestartet: {}", zinsRechner.toString());

    double endkapital = zinsRechner.berechneEndkapital();
    model.addAttribute("endkapital", endkapital);

    List<Double> jahresentwicklung = new ArrayList<>();
    List<Integer> jahre = new ArrayList<>();
    double kapital = zinsRechner.getStartkapital();

    for (int jahr = 1; jahr <= zinsRechner.getLaufzeit(); jahr++) {
        kapital *= (1 + zinsRechner.getZinssatz() / 100);
        jahresentwicklung.add(kapital);
        jahre.add(jahr);
        logger.debug("Jahr {}: Kapital = {}", jahr, kapital);  //
Detailliertes Debug-Logging
    }

    model.addAttribute("jahresentwicklung", jahresentwicklung);
    model.addAttribute("jahre", jahre);

    logger.info("Berechnung abgeschlossen, Endkapital: {}", endkapital);
    return "ergebnis";
```

```
    }
```

- Verwende `logger.info` für allgemeine Informationen und `logger.debug` für detaillierte Informationen, die nur bei aktiviertem Debug-Level sichtbar sind.

Schritt 3: Logging-Konfiguration in der `application.properties`

Falls du die Log-Level für bestimmte Pakete anpassen willst, füge dies in der Datei `src/main/resources/application.properties` hinzu:

```
properties
Code kopieren
# Standard-Log-Level für alle Pakete
logging.level.root=INFO

# Detailliertes Debug-Log-Level für den Controller
logging.level.com.example.Way2Million.controller=DEBUG
```

Dies stellt sicher, dass nur der Controller auf Debug-Level geloggt wird, während andere Teile der Anwendung auf dem höheren INFO-Level bleiben.

Schritt 4: Anpassen der Log-Ausgabeformatierung (Optional)

Standardmäßig ist Logback der Logging-Mechanismus in Spring Boot. Falls du das Ausgabeformat ändern willst, kannst du dies in einer `logback-spring.xml` konfigurieren. Für einfache Anforderungen reicht das Standardformat in den meisten Fällen aus.

Zusammenfassung

- **Logger einfügen:** `Logger logger = LoggerFactory.getLogger(...)`
- **Log-Level nutzen:** `info` für allgemeine Logs, `debug` für Details.
- **Log-Level in `application.properties` steuern:** `logging.level.[package]=[LEVEL]`

Mit diesen Schritten hast du schnell ein einfaches, aber wirksames Logging in deinem Spring Boot Projekt implementiert!

Minütliches Logging Seitenaufruf

Schritt 1: Erstellen eines `PageViewService`-Service

1. Erstelle eine neue Klasse `PageViewService` im Paket `com.example.Way2Million.service`.
2. In dieser Klasse kannst du die Zählvariablen und die geplante Methode `logPageViews()` platzieren.

```
java
Code kopieren
package com.example.Way2Million.service;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;

@Service
public class PageViewService {

    private static final Logger logger =
LoggerFactory.getLogger(PageViewService.class);

    private int indexPageViews = 0;
    private int ergebnisPageViews = 0;

    // Methode zum Erhöhen des Zählers für index.html
    public void incrementIndexPageViews() {
        indexPageViews++;
    }

    // Methode zum Erhöhen des Zählers für ergebnis.html
    public void incrementErgebnisPageViews() {
        ergebnisPageViews++;
    }

    // Geplante Methode, die alle 60 Sekunden die Seitenaufrufe protokolliert
    @Scheduled(fixedRate = 60000) // 60 Sekunden in Millisekunden
    public void logPageViews() {
        logger.info("Seitenaufrufe - index.html: {}, ergebnis.html: {}",
indexPageViews, ergebnisPageViews);
    }
}
```

Schritt 2: Verwenden des `PageViewService` im `ZinsRechnerController`

In deinem Controller kannst du dann den `PageViewService` über Dependency Injection einbinden und die Zähler erhöhen, wenn die jeweiligen Seiten aufgerufen werden.

```
java
Code kopieren
package com.example.Way2Million.controller;

import com.example.Way2Million.model.ZinsRechner;
import com.example.Way2Million.service.PageViewService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class ZinsRechnerController {

    private final PageViewService pageViewService;

    @Autowired
    public ZinsRechnerController(PageViewService pageViewService) {
        this.pageViewService = pageViewService;
    }

    @GetMapping("/")
    public String showForm(Model model) {
        pageViewService.incrementIndexPageViews(); // Zähler für index.html
erhöhen
```

```
        model.addAttribute("zinsRechner", new ZinsRechner());
        return "index";
    }

    @PostMapping("/berechne")
    public String berechne(@ModelAttribute ZinsRechner zinsRechner, Model model)
{
        pageViewService.incrementErgebnisPageViews(); // Zähler für
ergebnis.html erhöhen
        double endkapital = zinsRechner.berechneEndkapital();
        model.addAttribute("endkapital", endkapital);

        // ... Berechnung und Model-Attribute ...
        return "ergebnis";
    }
}
```

Schritt 3: `Scheduler einschalten`

In der Main Klasse kannst du den Scheduler einschalten

```
@SpringBootApplication
@EnableScheduling  // Aktiviert die Spring Scheduling-Funktion
public class Way2MillionApplication {
      public static void main(String[] args) {
            SpringApplication.run(Way2MillionApplication.class, args);
      }

}
```