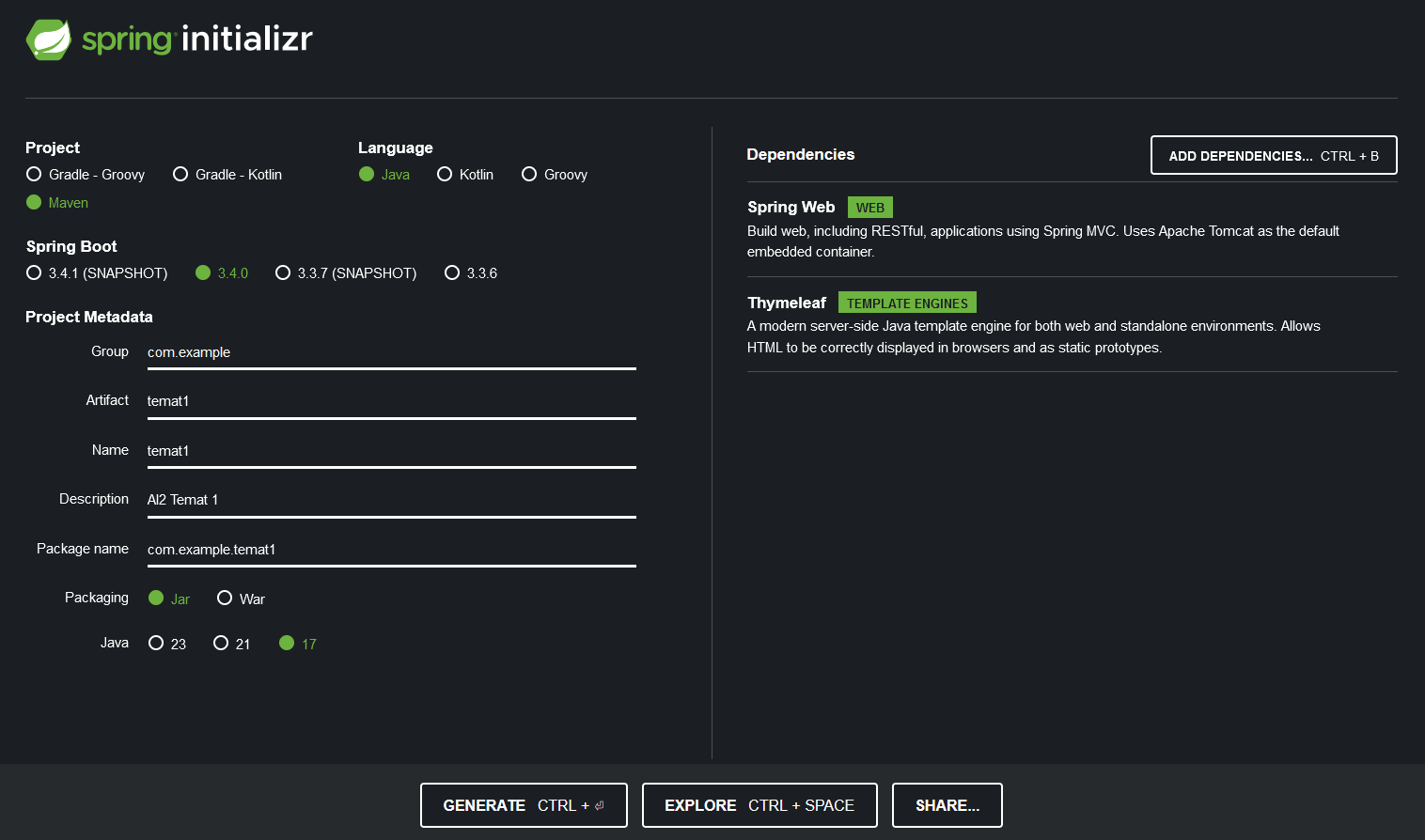
Sprawozdanie – AI2 Temat 1

Wprowadzenie do Spring Boot - prosta aplikacja webowa z webserwisem

Aleksander Łysoń, 46587

## Spring Initializr

W celu utworzenia nowej aplikacji wykorzystano generator Spring Initializr. Wybrano typ projektu Maven, język Java 17 i dodano zależności Spring Web i Thymeleaf.

  
Ustalone parametry w generatorze Spring Initializr

Projekt otworzono w IDE **Intellij**. W pliku application.properties dodano linijkę *server.port=46587* w celu ustawienia nie-defaultowego numeru portu HTTP.

## Klasa reprezentująca dane i klasa serwisu

Dodano klasę **Person** reprezentującą dane osobowe z prywatnymi polami *firstName* i *familyName*. Następnie utworzono dla nich settery i gettery.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie  
Person.java

Następnie utworzono klasę **PersonService** oznaczoną adnotacją *@Service* i prywatnym polem na dane *List<Person>.* Dodano metodę inicjalizującą z adnotacją *@PostConstruct* i przykładowe dwa obiekty klasy **Person**. Na koniec zaimplementowano podstawowe metody do manipulacji danymi: *getPeople*, *getPerson*, *addPerson*, *setPerson*, *removePerson*.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie  
PersonService.java

## Klasa kontrolera

Utworzono klasę **PersonController** z odpowiednią adnotacją *@Controller*. Utworzono metody do obsługi requestów GET i określono dla nich ścieżki wywołań.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie  
PersonController.java

Następnie dodano trzy pliki HTML widoków: **persons**.html, **person**.html i **error**.html.

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie  
person.html

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie  
persons.html

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie  
error.html

## Dodanie widoku dodawania danych

Utworzono plik HTML **addPerson** zawierający formularz edycji danych. Zmodyfikowano **PersonController**, by obsługiwał nowo utworzony plik (metody *showAddPersonForm* oraz *addPerson*). Do widoku **persons** dodano przycisk przekierowujący na widok **addPerson**.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie  
addPerson.html

## Testowanie aplikacji

W przeglądarce **Firefox** dodano nową osobę Sample Text.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie  
Firefox

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie  
Firefox

Następnie odświeżono listę osób w **trybie incognito** i w przeglądarce **Edge**.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie  
Firefox (incognito)

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie  
Edge

W obydwu przypadkach nowa osoba wyświetliła się na liście, implikując wykorzystanie zakresu Singleton. Jedna instancja serwisu jest współdzielona przez wszystkie żądania HTTP i przeglądarki, a wprowadzone zmiany są widoczne dla wszystkich użytkowników.

Następnie podjęto próbę zmiany zakresu na sesyjny przez dopisanie @Scope(„session”), jednak wystąpił błąd sugerujący zastosowanie **proxyMode**. Rozwiązaniem okazała się linijka

*@Scope(value=WebApplicationContext.SCOPE\_SESSION, proxyMode=ScopedProxyMode.TARGET\_CLASS),*

która rozwiązała problem, a aplikacja uruchomiła się. Tym razem po dodaniu nowej osoby w jednej przeglądarce, nie pojawiła się ona ani w trybie incognito, ani w drugiej przeglądarce.

## Kontroler REST

Do aplikacji dodano kontroler **REST** wraz z metodami *GET*, *POST*, *PUT* i *DELETE*. Do każdej metody ustalono ścieżki wywołań i zaimplementowano możliwość zwrócenia statusu 404 not found.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie  
PersonRestController.java (fragment)

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie  
PersonRestController.java (fragment)

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie  
JSON dla /api/persons

## Przetestowanie wywołań REST

Pobrano i zainstalowano **Postman Desktop Agent** w celu przetestowania API lokalnie. Przetestowano kolejno metody:

* **GET**

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie  
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

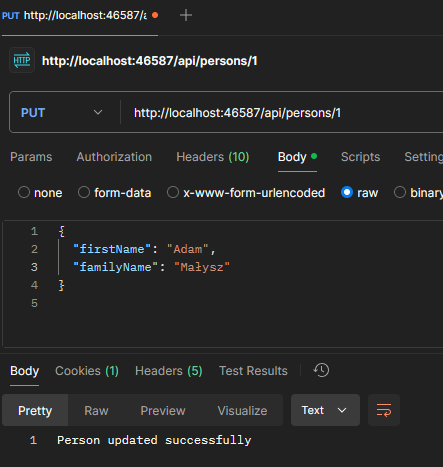
Opis wygenerowany automatycznie

* **POST**

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

* **PUT**



* **DELETE**

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Następnie użyto ponownie metody *GET* w celu wypisania całej listy osób i sprawdzenia, czy metody *POST*, *PUT* i *DELETE* faktycznie zadziałały.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Na koniec pozostało przetestowanie funkcjonalności mechanizmu zwracającego status 404 not found. W tym celu spróbowano wykorzystać metodę *DELETE* i usunąć nieistniejący rekord.

Obraz zawierający tekst, oprogramowanie, Oprogramowanie multimedialne, zrzut ekranu

Opis wygenerowany automatycznie

Wszystkie testy zakończyły się pomyślnie. Aplikacja webowa postawiona na **Spring Boot** pozwala na wyświetlanie i dodawanie osób, a dodatkowe **REST API** umożliwia dalsze manipulowanie danymi w aplikacji.