Rozgrzewka:

1. Utwórz nową aplikację wykorzystującą Spring Boot. Aby to osiągnąć utwórz w IntelliJ nowy projekt i wybierz Spring Initializr jako typ projektu. Wybierz wymagane zależności - będziemy potrzebowali zależności od **web** oraz **thymeleaf**. Po wygenerowaniu projektu spróbuj uruchomić funkcję main, aby upewnić się, że wszystko utworzyło się poprawnie.
2. Utwórz nowy kontroler o nazwie **GreetingController**, który będzie odpowiedzialny za wyświetlenie powitania dla użytkownika.
    a. Przygotuj endpoint **/greeting,** który przyjmie jako query param imię użytkownika
    b. Kontroler powinien zwrócić następującą wiadomość: Hello {{name}}! It's good to see you!
    c. Jeżeli parameter **name** nie zostanie wysłany to jego domyślna wartość powinna wynosić Stranger
    d. Postaraj się oddelegować logikę tworzenia wiadomości do osobnego serwisu **GreetingService.***
    e. np. http://localhost:8080**/greeting** -> Hello Stranger! It's good to see you!
    f. np. http://localhost:8080**/greeting?name=Maciek** -> Hello Maciek! It's good to see you!
3. Zamieńmy nasze powitanie wyświetlane w przeglądarce jako zwykły tekst na prosty html. W tym celu:
    a. Utwórz nowy endpoint **/greeting-html**, który będzie zachowywać się identycznie do endpointa utworzonego w punkcie drugim
    b. Utwórz plik **greeting.html** i korzystając z tagów html: <html>, <head>, <title>, <body> oraz <p>. Przygotuj odpowiednią strukturę html.
    c. Dodaj tekst do wyświetlenia do Model
    d. Wykorzystaj tekst, który należy wyświetlić w widoku html. Pamiętaj o użyciu th:text
    e. Z metody obsługującej endpoint zwróć nazwę widoku, który powinien zostać wyświetlony
    f. Wypisana wiadomość powitalna powinna mieć kolor zielony.
4. Przygotuj prosty formularz w którym podasz imię nazwisko i wiek.
    a. Przygotuj klasę, która przechowa te dane
    b. Przygotuj widok, który ten formularz wyświetli -> GET /person-details-editor
    c. Przygotuj endpoint POST /person-details, który przyjmie dane z formularza i wypisze je na ekran
    d. Przygotuj serwis który wykorzysta dane otrzymane w POST i zapisze je w pamięci
    e. Przygotuj widok GET /person-details, który wyświetli zapisane dane w ładnej formie
    f. Z poziomu GET /person-details dodaj przycisk umożliwiający przejście do person-details.

ToDo application - w ramach tego ćwiczenia napiszemy prostą aplikację, która pozwoli nam na zarządzanie listą zadań. W skład jej funkcjonalności będzie wchodzić:

1. Stworzenie nowego zadania z informacjami o nazwie zadania, dacie wykonania oraz szczegółowym opisie
2. Wyświetlenie najbliższego zadania
3. Oznaczenie zadania jako wykonane

Tworzenie nowego zadania:

1. Przygotuj klasę reprezentującą zadanie do wykonania. Powinna ona składać się z następujących informacji:
    a. id - najlepiej jako UUID
    b. Nazwa zadania
    c. Data wykonania
    d. Szczegóły zadania
    e. Czy zadanie zostało już wykonane
2. Przygotuj widok, który będzie formularzem do tworzenia nowego zadania i który pozwoli wypełnić dane podane powyżej. Przydatne tagi: <form>, <input>, <button>. Widok powinien wyświetlić się po wejściu na endpoint **/create-todo**
3. Przygotuj nowy endpoint typu **POST /create-todo**. Powinien on przyjąć todo item z wypełnionymi danymi oraz wypisać go na ekran.

Zapisywanie zadania w pamięci:
Korzystając z patternu Repository zapisz świeżo utworzone zadanie w List lub Map trzymanej w pamięci. Pamiętaj, że przed zapisem zadania powinieneś wygenerować dla niego nowe UUID - dobrym miejscem na to będzie serwis odpowiedzialny za zapis.

Wyświetlanie zadania:

1. Dodaj nowy widok **/recent-todo**, który pozwoli na wyświetlenie szczegółów zadania. Nie zapomnij wyświetlić też ID. Pamiętaj, że jeżeli chcesz coś wyświetlić musi to być dodane do interfejsu **Model**. Na początku wyświetlaj tylko pierwszy element z listy.
2. W serwisie odpowiedzialnym za pobieranie zadania dodaj logikę, która sprawi że użytkownik zobaczy najbliższe zadanie, które ma zrobić.
3. Zmodyfikuj endpoint **POST /create-todo** tak aby po poprawnym utworzeniu użytkownikowi wyświetlone zostały szczegóły świeżo utworzonego zadania.

Spięcie wszystkiego w całość:

1. Przygotuj plik index.html który będzie zawierał proste menu:
    a. Create todo item
    b. Show most recent todo
2. Do każdego utworzone widoku dodaj możliwość powrotu do strony głównej
3. Przydatne tagi: <ul>, <li>, <a>

Oznaczanie zadania jako ukończone:

1. W widoku **/recent-todo** dodaj przycisk który pozwoli na oznaczenie zadania jako ukończone.
2. Przygotuj **POST /finish-todo**, który obsłuży logikę kończenia zadania. Powinien on przyjąć dane aktualnie wyświetlanego zadania i na podstawie jego ID zaktualizować w repozytorium jego status.

3. Aby to osiągnąć, będzie należało przygotować ukryty formularz wykorzystując input type hidden. Formularz powinien wysłać tylko ID zadania.
4. Po poprawnym zaktualizowaniu przekieruj użytkownika do strony głównej lub pozostaw go na aktualnie wyświetlanej.
5. Zmodyfikuj logikę znajdowania zadania do wyświetlenia. Odfiltruj te które zostały już wykonane.
6. Zmodyfikuj widok **/recent-todo**. Jeżeli nie ma żadnego zadania do wyświetlenia wyświetl stosowny komunikat.

Walidacja nowego zadania:

1. Zwaliduj próbę utworzenia nowego zadania. Następujące dane powinny zostać sprawdzone:
    a. Nazwa nie może być pusta
    b. Data nie powinna być z przeszłości
    c. Szczegóły zadania nie powinny być dłuższe niż 160 znaków
2. Jeżeli walidacja się nie powiedzie użytkownik powinien zostać wyświetlony stosowny komunikat.