Instrukcja dla praktykantki Java

Celem tego zadania jest nauka korzystania z narzędzi i technologii powszechnie używanych w projektach Java.
Po zakończeniu tego ćwiczenia praktykantka powinna rozumieć i umieć zastosować w praktyce następujące technologie:
Git, Maven, Spring Boot, MapStruct, Swagger, REST API z Basic Auth oraz testowanie w JUnit 5.

1. Korzystanie z Gita

1. Zainstaluj Git:
   - Upewnij się, że Git jest zainstalowany na Twoim komputerze.
   - Sprawdź wersję Gita poleceniem: git --version

2. Skonfiguruj Git:
   - Ustaw swoje imię i email:
     git config --global user.name "Twoje Imię"
     git config --global user.email "twoj.email@example.com"

3. Zainicjuj repozytorium:
   - W terminalu przejdź do katalogu projektu i wpisz:
     git init

4. Dodawanie i zatwierdzanie zmian:
   - Dodaj wszystkie zmienione pliki do indeksu:
     git add .
   - Zatwierdź zmiany:
     git commit -m "Pierwszy commit"

5. Praca z repozytorium zdalnym:
   - Sklonuj repozytorium:
     git clone <url_repozytorium>
   - Wypchnij zmiany do zdalnego repozytorium:
     git push origin main

Zadanie: Zainicjuj nowe repozytorium Git, dodaj plik README.md z opisem projektu i wypchnij repozytorium na GitHub.

2. Korzystanie z Mavena

1. Utwórz projekt Maven:
   - Skorzystaj z polecenia:
     mvn archetype:generate -DgroupId=com.example -DartifactId=myproject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

2. Struktura projektu Maven:
   - Upewnij się, że katalogi src/main/java i src/test/java są poprawnie zorganizowane.

3. Dodawanie zależności:
   - Otwórz plik pom.xml i dodaj zależności, np.:
     ```
     <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter</artifactId>
       <version>3.1.3</version>
     </dependency>
     ```

4. Budowanie projektu:
   - Zbuduj projekt poleceniem:
     mvn clean install

Zadanie: Utwórz projekt Maven, dodaj zależność do Spring Boota i zbuduj projekt.

3. Korzystanie z Spring Boot

1. Stwórz nowy projekt Spring Boot:
   - Skorzystaj z Spring Initializr lub wykonaj polecenie:
     mvn spring-boot:generate

2. Podstawowa konfiguracja:
   - Dodaj klasę główną z adnotacją @SpringBootApplication.

3. Uruchom aplikację:
   - W terminalu:
     mvn spring-boot:run

Zadanie: Stwórz aplikację Spring Boot, która uruchamia serwer i loguje komunikat "Hello Spring Boot!" przy starcie.

4. Korzystanie z MapStructa

1. Dodaj zależność:
```
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>1.5.3.Final</version>
</dependency>
```

2. Utwórz prostego mappera:
- Klasa źródłowa:
```
public class User {
    private String name;
    private int age;
    // Gettery i settery
}
```
- Klasa docelowa:
```
public class UserDTO {
    private String name;
    private int age;
    // Gettery i settery
}
```
- Mapper:
```
@Mapper
public interface UserMapper {
    UserMapper INSTANCE = Mappers.getMapper(UserMapper.class);
    UserDTO userToUserDTO(User user);
}
```

Zadanie: Napisz mapper konwertujący obiekty klasy User na UserDTO.

5. Napisanie API w Swaggerze

1. Dodaj zależność:
```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.6.15</version>
</dependency>
```

2. Skonfiguruj Swaggera:
- Po uruchomieniu aplikacji Swagger UI będzie dostępne pod /swagger-ui.html.

3. Dodaj opis dla kontrolera:
- W klasie kontrolera użyj adnotacji:

```
@RestController
@RequestMapping("/api/users")
@Tag(name = "Users", description = "Operacje na użytkownikach")
public class UserController {
    @GetMapping("/{id}")
    public UserDTO getUser(@PathVariable Long id) {
        return new UserDTO("Jan", 25);
    }
}
```

Zadanie: Utwórz REST API z dokumentacją w Swagger UI.

6. Przygotowanie REST Endpointów

1. Stwórz kontroler:

```
@RestController
@RequestMapping("/api")
public class ApiController {
    @GetMapping("/hello")
    public String hello() {
        return "Hello, world!";
    }
}
```

2. Uruchom aplikację i przetestuj endpoint:
- Przejdź do http://localhost:8080/api/hello.

Zadanie: Dodaj endpoint zwracający listę użytkowników.

7. Basic Auth

1. Dodaj zależność:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

2. Konfiguracja:
- Stwórz klasę:

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
      .authorizeRequests().anyRequest().authenticated()
      .and()
      .httpBasic();
  }
}
```

Zadanie: Zabezpiecz endpoint /api/hello za pomocą Basic Auth.

8. Testowanie z JUnit 5

1. Dodaj zależność:

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter</artifactId>
  <version>5.9.3</version>
  <scope>test</scope>
</dependency>
```

2. Przykład testu:

```
@Test
void shouldReturnHelloMessage() {
  ApiController controller = new ApiController();
  assertEquals("Hello, world!", controller.hello());
}
```

Zadanie: Napisz testy jednostkowe dla endpointu /api/users.

9. Przygotowanie bazy danych H2

1. Dodaj zależność do bazy H2:

```xml
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

2. Konfiguracja bazy H2 w pliku `application.properties`:
  ```properties
  spring.datasource.url=jdbc:h2:mem:testdb
  spring.datasource.driverClassName=org.h2.Driver
  spring.datasource.username=sa
  spring.datasource.password=
  spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
  ```

3. Dostęp do konsoli H2:
  - Po uruchomieniu aplikacji konsola H2 będzie dostępna pod adresem: `http://localhost:8080/h2-console`.
  - Użyj domyślnych danych logowania:
    - JDBC URL: `jdbc:h2:mem:testdb`
    - User: `sa`
    - Password: (puste pole).

Zadanie: Skonfiguruj bazę H2 w aplikacji Spring Boot i przetestuj połączenie poprzez konsolę H2.

10. Korzystanie z JPA

1. Dodaj zależność do JPA:
  ```xml
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  ```

2. Utwórz encję:
  ```java
  @Entity
  public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Double price;

    // Gettery i settery
  ```

```
}
```

3. Utwórz repozytorium:
```java
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

4. Dodaj metodę w kontrolerze do zapisu i odczytu danych:
```java
@RestController
@RequestMapping("/api/products")
public class ProductController {

    @Autowired
    private ProductRepository productRepository;

    @PostMapping
    public Product addProduct(@RequestBody Product product) {
        return productRepository.save(product);
    }

    @GetMapping
    public List<Product> getProducts() {
        return productRepository.findAll();
    }
}
```

Zadanie: Utwórz encję `Product`, skonfiguruj repozytorium JPA oraz stwórz REST API do zapisu i odczytu produktów w bazie danych.

Opis zadania funkcjonalnego

Twoim zadaniem jest przygotowanie aplikacji w Spring Boot, która będzie działać jako backend sklepu internetowego.
Aplikacja powinna umożliwiać:
1. Dodawanie produktów do bazy danych.
2. Pobieranie listy produktów.
3. Zabezpieczenie dostępu do API za pomocą Basic Auth.

4. Dokumentację API w Swagger UI.
5. Testowanie funkcjonalności aplikacji za pomocą JUnit 5.

Funkcjonalność sklepu:
- Produkt powinien mieć nazwę, cenę oraz unikalny identyfikator.
- Aplikacja powinna przechowywać dane w pamięci, wykorzystując bazę danych H2.
- Użytkownicy powinni mieć możliwość logowania się za pomocą Basic Auth, aby korzystać z API.

Powodzenia w realizacji zadania!