Instytut Informatyki
Kolegium Nauk Przyrodniczych
Uniwersytet Rzeszowski

Przedmiot:

Programowanie zespołowe

Sprawozdanie: Testy jednostkowe

Wykonali: Filip Cyboran, Jakub Bednarczyk, Szymon Pluta, Karol Bal, Damian Bernat

Prowadzący: mgr inż. Adam Szczur

Rzeszów 2023

1. Klasa ItmsApplicationTests (zdj. 1)

Ta klasa jest częścią frameworka testowego Spring Boot i służy do testowania kontekstu aplikacji oraz połączenia z bazą danych dla Systemu Zarządzania IT (ITMS).

Adnotacje:

@SpringBootTest: Ta adnotacja informuje Spring Boot, aby uruchomił kontekst aplikacji do celów testowych.

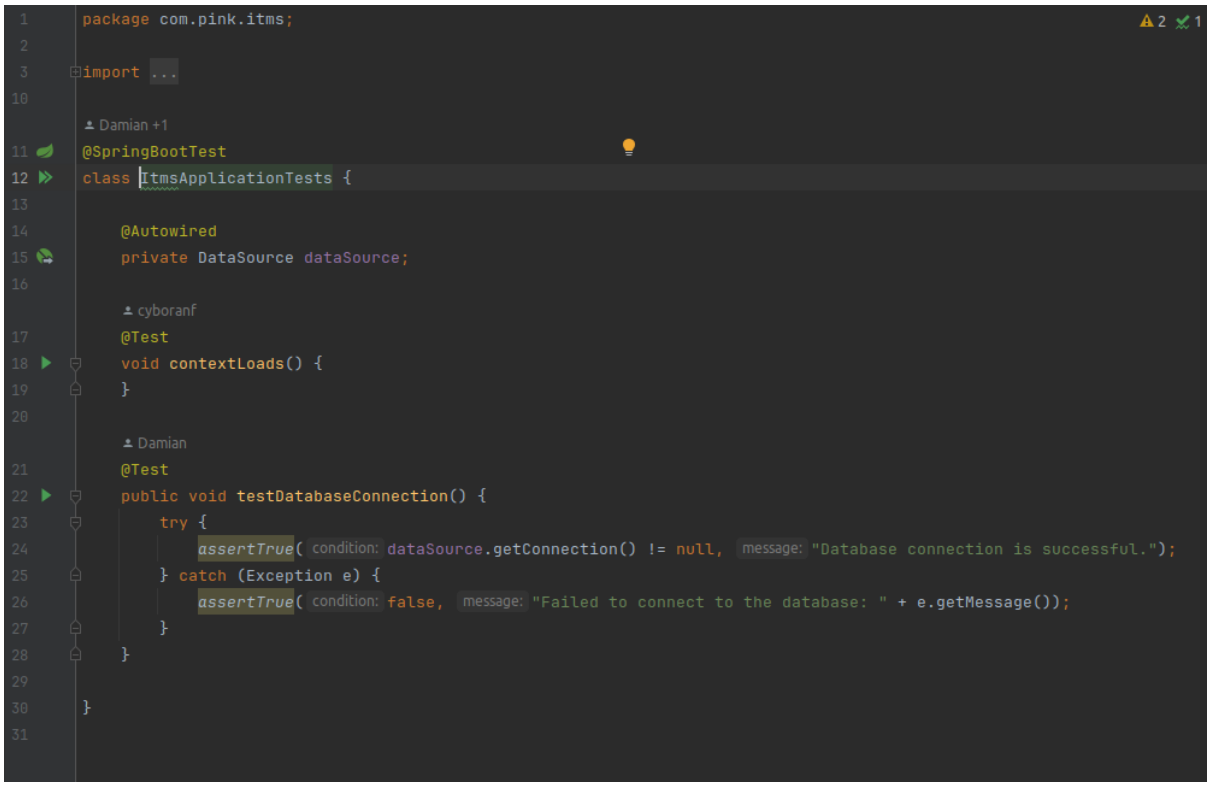
@Autowired: Używana do automatycznego wstrzyknięcia obiektu DataSource do klasy testowej, co jest niezbędne do operacji na bazie danych.

Metody:

contextLoads(): Pusty test, który po prostu weryfikuje, czy kontekst aplikacji Spring ładuje się bez rzucania błędów.

testDatabaseConnection(): Metoda ta sprawdza, czy aplikacja może pomyślnie połączyć się z konfigurowaną bazą danych. Potwierdza, że połączenie z bazą danych nie jest null i obsługuje wyjątki wskazujące na niepowodzenia połączenia, zapewniając niepowodzenie testu, jeśli zostanie przechwycony wyjątek.

Zdj. 1



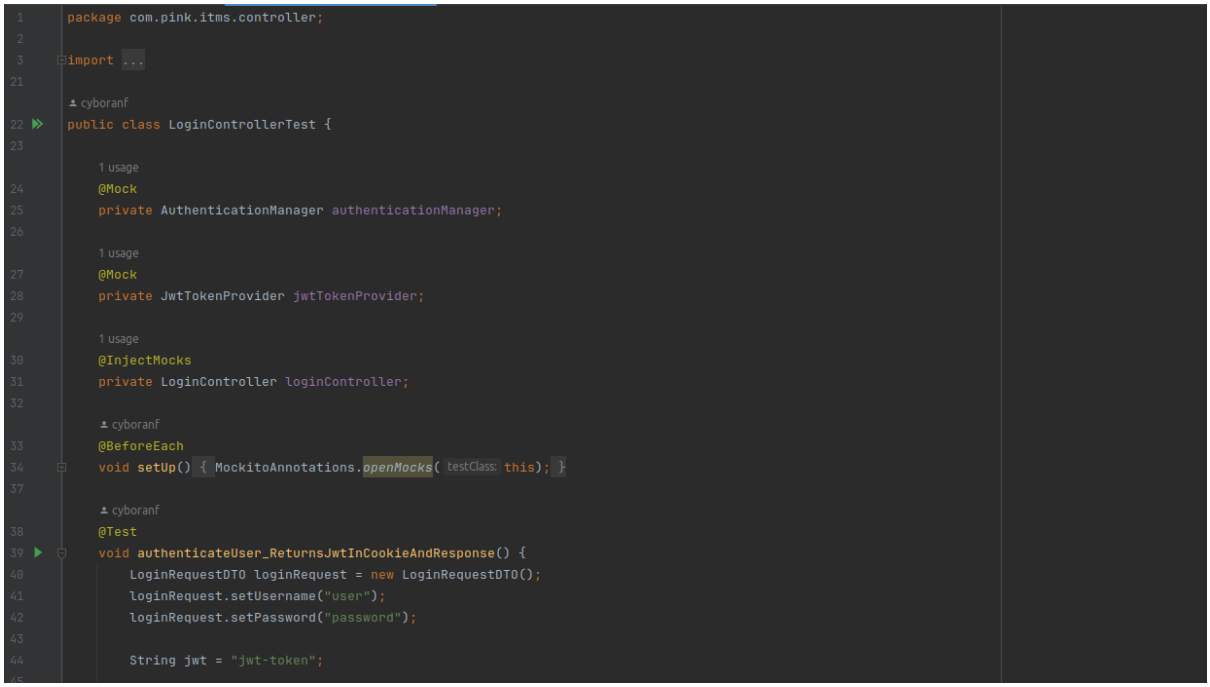
Metoda testowa - authenticateUser_ReturnsJwtInCookieAndResponse(): (zdjecia: 2.1 oraz 2.2)

Przygotowuje mock LoginRequestDTO z nazwą użytkownika i hasłem.

Mockuje proces uwierzytelniania, aby zwrócić predefiniowany token JWT.

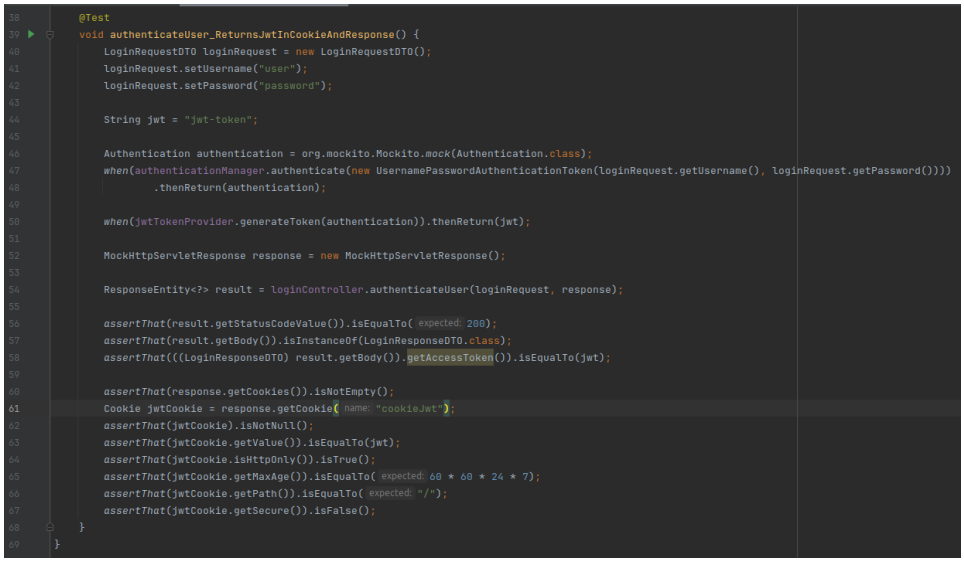
Wywołuje metodę authenticateUser w LoginController i sprawdza zwróconą odpowiedź HTTP oraz token JWT osadzony w ciasteczkach. Zapewnia to, że metoda zachowuje się zgodnie z oczekiwaniami, zwracając kod statusu 200, poprawny JWT w ciele odpowiedzi oraz bezpieczne ciasteczko.

Zdj. 2.1



```
package com.pink.itms.controller;

import ...

cyboranf
public class LoginControllerTest {

    1 usage
    @Mock
    private AuthenticationManager authenticationManager;

    1 usage
    @Mock
    private JwtTokenProvider jwtTokenProvider;

    1 usage
    @InjectMocks
    private LoginController loginController;

    cyboranf
    @BeforeEach
    void setUp() { MockitoAnnotations.openMocks( testClass: this); }

    cyboranf
    @Test
    void authenticateUser_ReturnsJwtInCookieAndResponse() {
        LoginRequestDTO loginRequest = new LoginRequestDTO();
        loginRequest.setUsername("user");
        loginRequest.setPassword("password");

        String jwt = "jwt-token";
```

Zdj. 2.2



```
    @Test
    void authenticateUser_ReturnsJwtInCookieAndResponse() {
        LoginRequestDTO loginRequest = new LoginRequestDTO();
        loginRequest.setUsername("user");
        loginRequest.setPassword("password");

        String jwt = "jwt-token";

        Authentication authentication = org.mockito.Mockito.mock(Authentication.class);
        when(authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword())))
                .thenReturn(authentication);

        when(jwtTokenProvider.generateToken(authentication)).thenReturn(jwt);

        MockHttpServletResponse response = new MockHttpServletResponse();

        ResponseEntity<?> result = loginController.authenticateUser(loginRequest, response);

        assertThat(result.getStatusCodeValue()).isEqualTo( expected: 200);
        assertThat(result.getBody()).isInstanceOf(LoginResponseDTO.class);
        assertThat(((LoginResponseDTO) result.getBody()).getAccessToken()).isEqualTo(jwt);

        assertThat(response.getCookies()).isNotEmpty();
        Cookie jwtCookie = response.getCookie( name: "cookieJwt");
        assertThat(jwtCookie).isNotNull();
        assertThat(jwtCookie.getValue()).isEqualTo(jwt);
        assertThat(jwtCookie.isHttpOnly()).isTrue();
        assertThat(jwtCookie.getMaxAge()).isEqualTo( expected: 60 * 60 * 24 * 7);
        assertThat(jwtCookie.getPath()).isEqualTo( expected: "/");
        assertThat(jwtCookie.getSecure()).isFalse();
    }
}
```

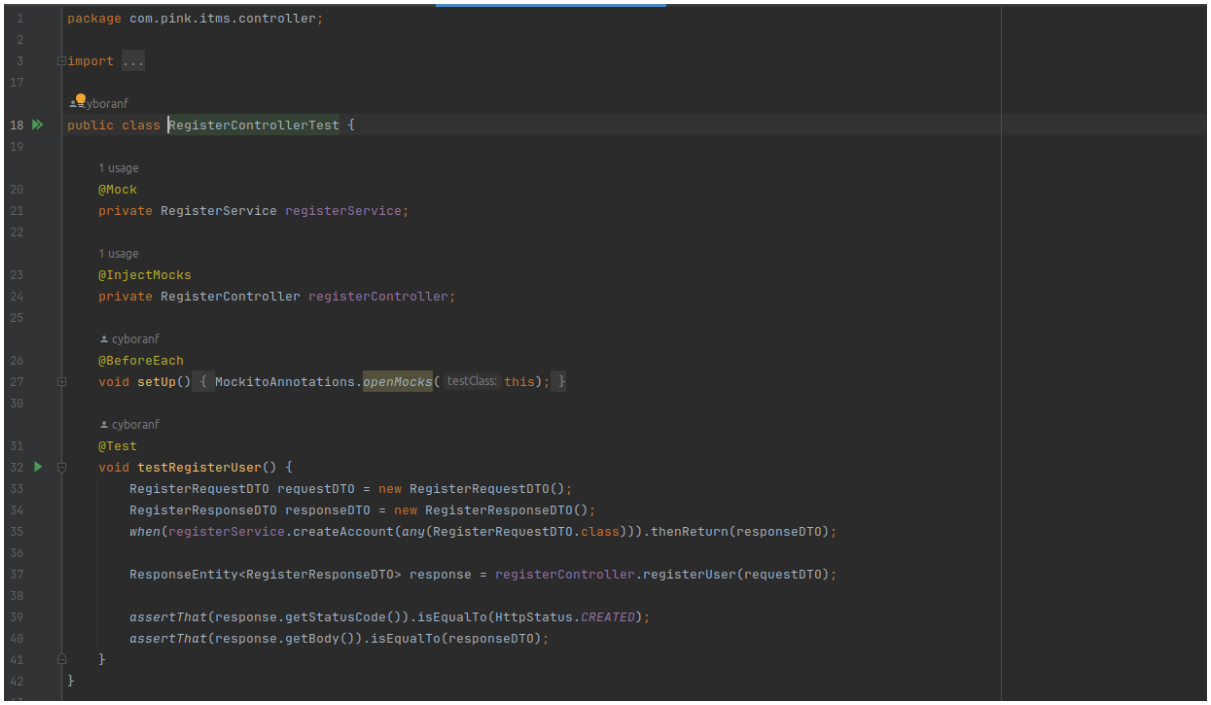
Metoda testowa - testRegisterUser(): (zdjecie 3)

Tworzy RegisterRequestDTO do testów.

Konfiguruje mock odpowiedzi, która ma być zwrócona, gdy metoda registerService.createAccount zostanie wywołana.

Testuje, czy metoda registerUser zwraca ResponseEntity ze statusem HTTP CREATED i czy ciało odpowiedzi zawiera oczekiwany RegisterResponseDTO.

Zdj. 3



```java
package com.pink.itms.controller;

import ...

cyboranf
public class RegisterControllerTest {

    1 usage
    @Mock
    private RegisterService registerService;

    1 usage
    @InjectMocks
    private RegisterController registerController;

    cyboranf
    @BeforeEach
    void setUp() { MockitoAnnotations.openMocks( testClass: this); }

    cyboranf
    @Test
    void testRegisterUser() {
        RegisterRequestDTO requestDTO = new RegisterRequestDTO();
        RegisterResponseDTO responseDTO = new RegisterResponseDTO();
        when(registerService.createAccount(any(RegisterRequestDTO.class))).thenReturn(responseDTO);

        ResponseEntity<RegisterResponseDTO> response = registerController.registerUser(requestDTO);

        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.CREATED);
        assertThat(response.getBody()).isEqualTo(responseDTO);
    }
}
```