Capítulo VI: Product Verification & Validation

6.1. Testing Suites & Validation

6.1.1. Core Entities Unit Tests

En esta sección se detallan las pruebas unitarias realizadas sobre las entidades principales del sistema, verificando que sus atributos, métodos y reglas de negocio funcionen correctamente de forma aislada. Estas pruebas aseguran que la lógica interna de cada componente cumple con los requisitos definidos y no presenta errores, contribuyendo a la solidez y confiabilidad de la aplicación.

1. **testCreateAppointment_Success**(AppointmentServiceTest)

Esta prueba verifica que el servicio puede crear una cita médica exitosamente cuando se proporcionan todos los datos válidos. Simula la búsqueda de perfiles de doctor y paciente en la base de datos, y valida que el objeto AppointmentResponse retornado contenga la información correcta incluyendo los nombres completos del doctor y paciente, el estado SCHEDULED, y que el método save del repositorio se haya invocado exactamente una vez.

```
@Test
void testCreateAppointment_Success() {
    when(doctorRepository.findById(doctorId)).thenReturn(Optional.of(doctor));
    when(patientRepository.findById(patientId)).thenReturn(Optional.of(patient));

when(appointmentRepository.save(any(Appointment.class))).thenReturn(appointment);

    AppointmentResponse response = appointmentService.createAppointment(request);

    assertEquals("SCHEDULED", response.getStatus());
    assertEquals(doctor.getFullName(), response.getDoctorName());
    assertEquals(patient.getFullName(), response.getPatientName());
    verify(appointmentRepository, times(1)).save(any(Appointment.class));
}
```

2. **testCreateAppointment_DoctorNotFound**(AppointmentServiceTest)

Esta prueba valida el manejo de errores cuando se intenta crear una cita pero el perfil del doctor no existe en la base de datos. Verifica que el sistema lance una RuntimeException apropiada y, lo más importante, que NO se intente guardar ninguna cita en el repositorio cuando faltan datos críticos, garantizando la integridad de los datos.

```
@Test
void testCreateAppointment_DoctorNotFound() {
    when(doctorRepository.findById(doctorId)).thenReturn(Optional.empty());

    assertThrows(RuntimeException.class, () ->
appointmentService.createAppointment(request));
```

```
    verify(appointmentRepository, never()).save(any());
}
```

3. **testGetAppointmentsByDoctor_Success**(AppointmentServiceTest)

Esta prueba confirma que el servicio puede recuperar correctamente todas las citas asociadas a un doctor específico. Simula una lista de citas retornadas por el repositorio y valida que la respuesta contenga el número correcto de elementos y que los datos del doctor estén presentes en cada respuesta, verificando también que se haya llamado al método del repositorio con el ID correcto.

```
@Test
void testGetAppointmentsByDoctor_Success() {

when(appointmentRepository.findByDoctorId(doctorId)).thenReturn(List.of(appointmen
t1, appointment2));

    List<AppointmentResponse> responses =
appointmentService.getAppointmentsByDoctor(doctorId);

    assertEquals(2, responses.size());
    responses.forEach(r -> assertEquals(doctor.getFullName(), r.getDoctorName()));
    verify(appointmentRepository).findByDoctorId(doctorId);
}
```

4. **testUpdateAppointmentStatus_ToCancelled**(AppointmentServiceTest)

Esta prueba verifica la funcionalidad de cancelación de citas, asegurando que cuando se actualiza el estado a CANCELLED, el sistema también registre correctamente la razón de cancelación y la fecha/hora en que ocurrió. Valida que todos los campos relacionados con la cancelación se actualicen apropiadamente y que los cambios se persistan en la base de datos mediante el repositorio.

```
@Test
void testUpdateAppointmentStatus_ToCancelled() {

when(appointmentRepository.findById(appointmentId)).thenReturn(Optional.of(appoint
ment));

when(appointmentRepository.save(any(Appointment.class))).thenReturn(appointment);

    appointmentService.updateAppointmentStatus(appointmentId, "CANCELLED", "Motivo
de cancelación");

    assertEquals("CANCELLED", appointment.getStatus());
    assertNotNull(appointment.getCancelledAt());
    assertEquals("Motivo de cancelación", appointment.getCancellationReason());
    verify(appointmentRepository).save(appointment);
```

```
}
```

5. **testAddFollowUpNotes_Success**(AppointmentServiceTest)

Esta prueba valida que el sistema puede agregar notas de seguimiento a una cita existente después de que ha ocurrido. Verifica que las notas se almacenen correctamente en el objeto Appointment y que los cambios se guarden en la base de datos, lo cual es crucial para mantener el historial médico y las recomendaciones post-consulta.

```
@Test
void testAddFollowUpNotes_Success() {

when(appointmentRepository.findById(appointmentId)).thenReturn(Optional.of(appoint
ment));

when(appointmentRepository.save(any(Appointment.class))).thenReturn(appointment);

    appointmentService.addFollowUpNotes(appointmentId, "Notas de seguimiento");

    assertEquals("Notas de seguimiento", appointment.getFollowUpNotes());
    verify(appointmentRepository).save(appointment);
}
```

6. **testGenerateProfileId_Doctor**(ProfileTest)

Esta prueba verifica la generación automática de identificadores únicos para perfiles de doctores. Valida que cuando se crea un perfil de tipo DOCTOR, el sistema genere automáticamente un profileId que comience con el prefijo "DOC-", asegurando un sistema de identificación consistente y fácilmente reconocible en toda la aplicación.

```
@Test
void testGenerateProfileId_Doctor() {
    Profile doctor = new Profile(ProfileType.DOCTOR, "John", "Doe");
    assertTrue(doctor.getProfileId().startsWith("DOC-"));
}
```

7. **testGenerateProfileId_Patient**(ProfileTest)

Similar a la prueba anterior pero para pacientes, esta prueba confirma que los perfiles de tipo PATIENT reciben identificadores con el prefijo "PAT-". Esto garantiza que el sistema pueda diferenciar rápidamente entre tipos de perfiles mediante sus identificadores únicos, facilitando búsquedas y validaciones.

```
@Test
void testGenerateProfileId_Patient() {
    Profile patient = new Profile(ProfileType.PATIENT, "Jane", "Smith");
    assertTrue(patient.getProfileId().startsWith("PAT-"));
}
```

8. **testGetFullName**(ProfileTest)

Esta prueba valida un método de utilidad simple pero importante que concatena el nombre y apellido de un perfil. Aunque parece trivial, es fundamental para asegurar que la presentación de nombres en la interfaz de usuario sea consistente en toda la aplicación y que no haya problemas con espacios o formato.

```
@Test
void testGetFullName() {
    Profile profile = new Profile(ProfileType.PATIENT, "Ana", "García");
    assertEquals("Ana García", profile.getFullName());
}
```

9. **testGetAuthorities_Organization**(UserTest)

Esta prueba verifica la implementación de Spring Security en el modelo User, específicamente que los usuarios con rol ORGANIZATION reciban las autoridades correctas. Valida que el método getAuthorities retorne una colección que contenga "ROLE_ORGANIZATION", lo cual es esencial para el control de acceso basado en roles en toda la aplicación.

```
@Test
void testGetAuthorities_Organization() {
    User user = new User("org@example.com", "pass", Role.ORGANIZATION);
    Collection<? extends GrantedAuthority> authorities = user.getAuthorities();
    assertTrue(authorities.stream().anyMatch(a ->
a.getAuthority().equals("ROLE_ORGANIZATION")));
}
```

10. **testDefaultValues**(UserTest)

Esta prueba integral verifica que cuando se crea un nuevo usuario, todos los valores predeterminados se establezcan correctamente. Valida que el rol sea ORGANIZATION por defecto, que la cuenta esté activa, que el email no esté verificado inicialmente, y que todas las banderas de seguridad de Spring (cuenta no expirada, no bloqueada, credenciales no expiradas) estén configuradas apropiadamente para un nuevo usuario.

```
@Test
void testDefaultValues() {
```

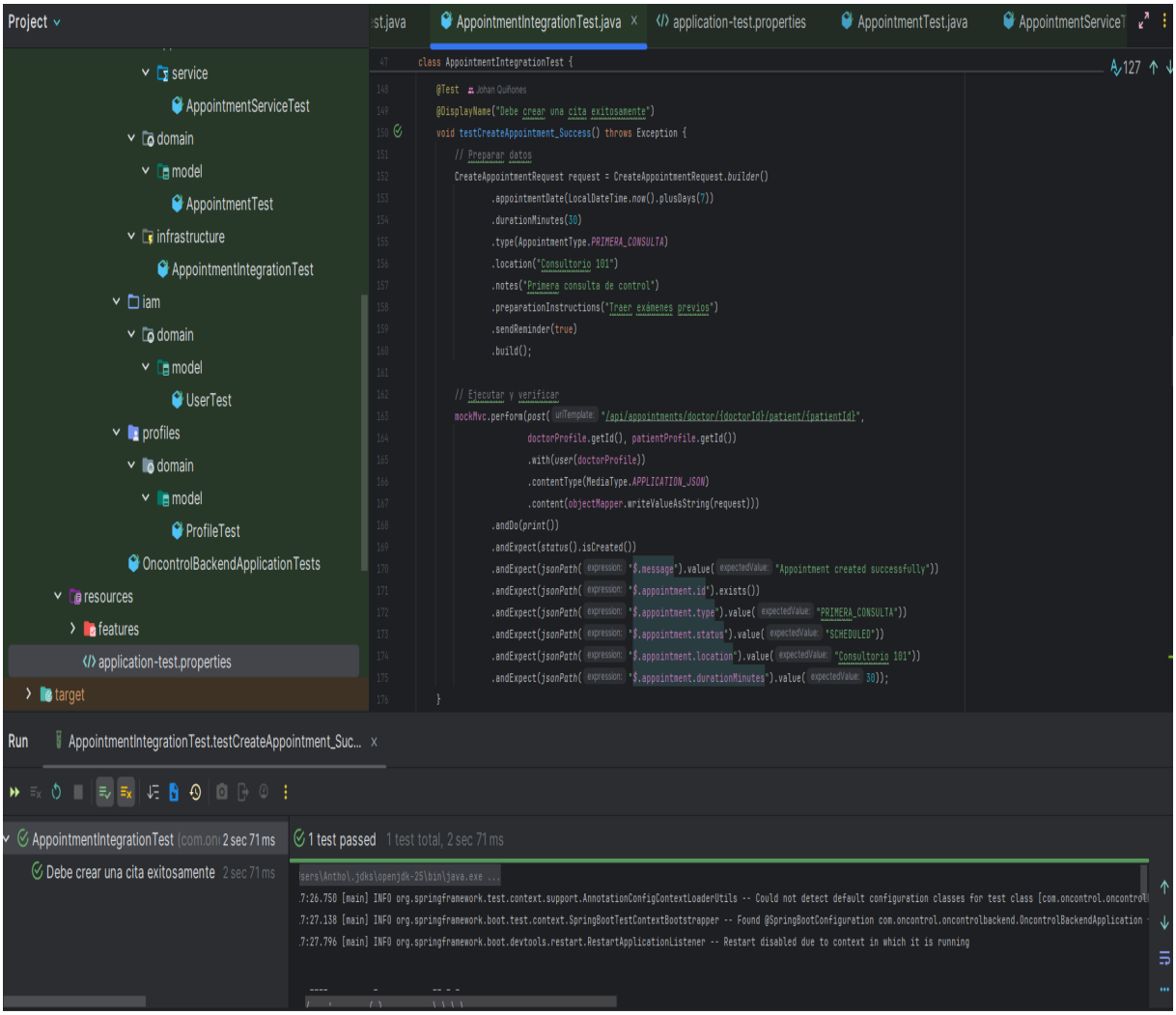
```
    User user = new User("org@example.com", "pass");
    assertEquals(Role.ORGANIZATION, user.getRole());
    assertTrue(user.isActive());
    assertFalse(user.isEmailVerified());
    assertTrue(user.isAccountNonExpired());
    assertTrue(user.isAccountNonLocked());
    assertTrue(user.isCredentialsNonExpired());
}
```

6.1.2. Core Integration Tests

1. **testCreateAppointment_Success**(AppointmentIntegrationTest)
   La prueba testCreateAppointment_Success valida el correcto funcionamiento del proceso de creación de citas médicas en el sistema. Mediante el uso de MockMvc, se simula una solicitud HTTP POST al endpoint correspondiente, verificando que la respuesta sea exitosa con el estado 201 (Created) y que los datos devueltos coincidan con los valores enviados. Esta prueba garantiza que el sistema registre adecuadamente las citas, manteniendo la coherencia y confiabilidad en la gestión de la información médica.

```
@Test
    @DisplayName("Debe crear una cita exitosamente")
    void testCreateAppointment_Success() throws Exception {
        // Preparar datos
        CreateAppointmentRequest request = CreateAppointmentRequest.builder()
                .appointmentDate(LocalDateTime.now().plusDays(7))
                .durationMinutes(30)
                .type(AppointmentType.PRIMERA_CONSULTA)
                .location("Consultorio 101")
                .notes("Primera consulta de control")
                .preparationInstructions("Traer exámenes previos")
                .sendReminder(true)
                .build();

        // Ejecutar y verificar

mockMvc.perform(post("/api/appointments/doctor/{doctorId}/patient/{patientId}",
                        doctorProfile.getId(), patientProfile.getId())
                        .with(user(doctorProfile))
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(objectMapper.writeValueAsString(request)))
                .andDo(print())
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.message").value("Appointment created
successfully"))
                .andExpect(jsonPath("$.appointment.id").exists())

.andExpect(jsonPath("$.appointment.type").value("PRIMERA_CONSULTA"))
                .andExpect(jsonPath("$.appointment.status").value("SCHEDULED"))
                .andExpect(jsonPath("$.appointment.location").value("Consultorio
```

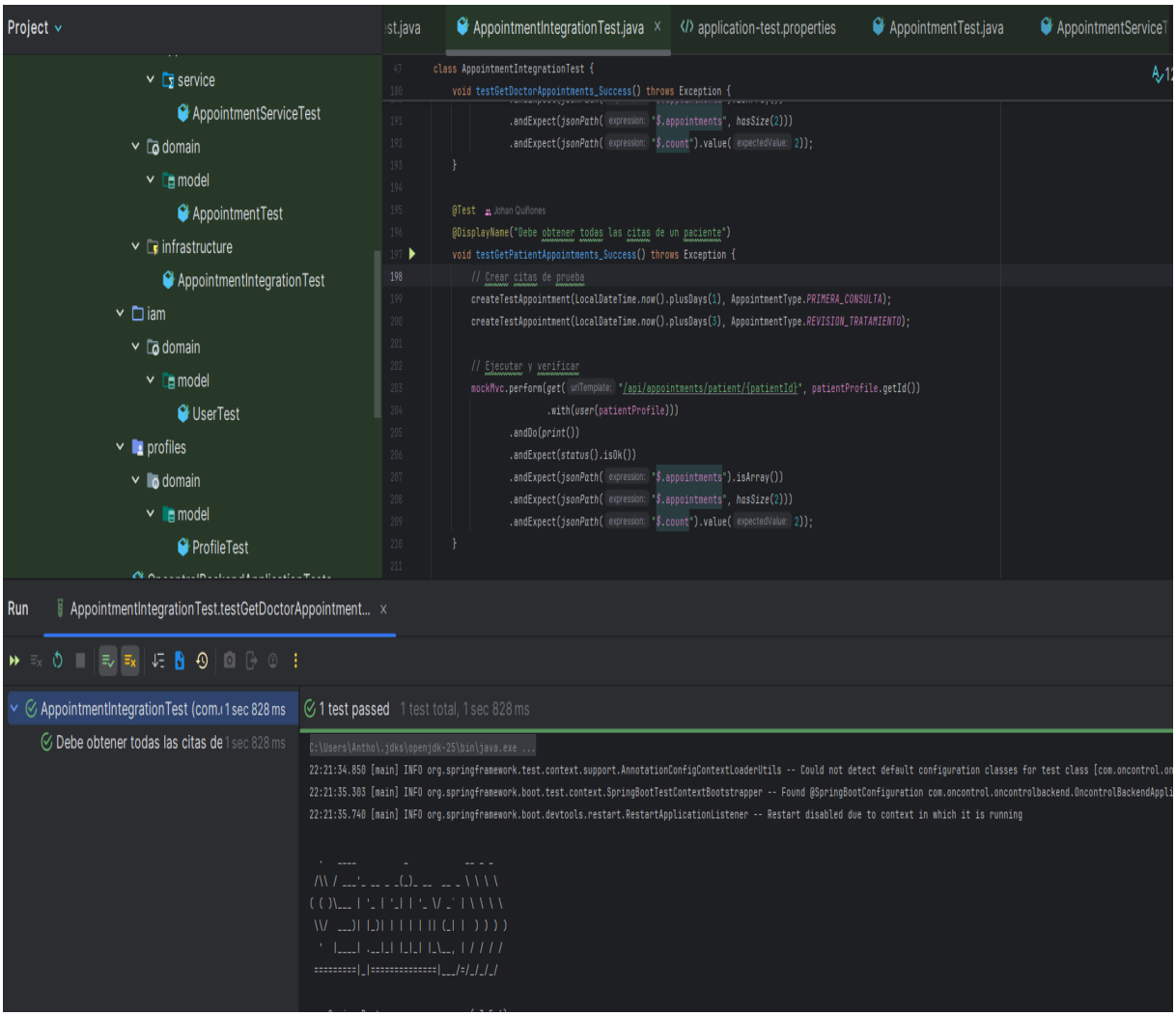
```
101"))
                .andExpect(jsonPath("$.appointment.durationMinutes").value(30));
    }
```



2. **testGetDoctorAppointments_Success**(AppointmentIntegrationTest)
   La prueba testGetDoctorAppointments_Success evalúa la capacidad del sistema para recuperar correctamente todas las citas asociadas a un médico específico. Utilizando MockMvc, se simula una solicitud HTTP GET al endpoint correspondiente y se verifica que la respuesta tenga el estado 200 (OK). Asimismo, se comprueba que el resultado contenga un arreglo con el número esperado de citas registradas, asegurando la correcta funcionalidad del módulo de consulta de citas médicas.

```
@Test
    @DisplayName("Debe obtener todas las citas de un doctor")
    void testGetDoctorAppointments_Success() throws Exception {
        // Crear citas de prueba
        createTestAppointment(LocalDateTime.now().plusDays(1),
AppointmentType.PRIMERA_CONSULTA);
        createTestAppointment(LocalDateTime.now().plusDays(2),
AppointmentType.CONSULTA_SEGUIMIENTO);

        // Ejecutar y verificar
```

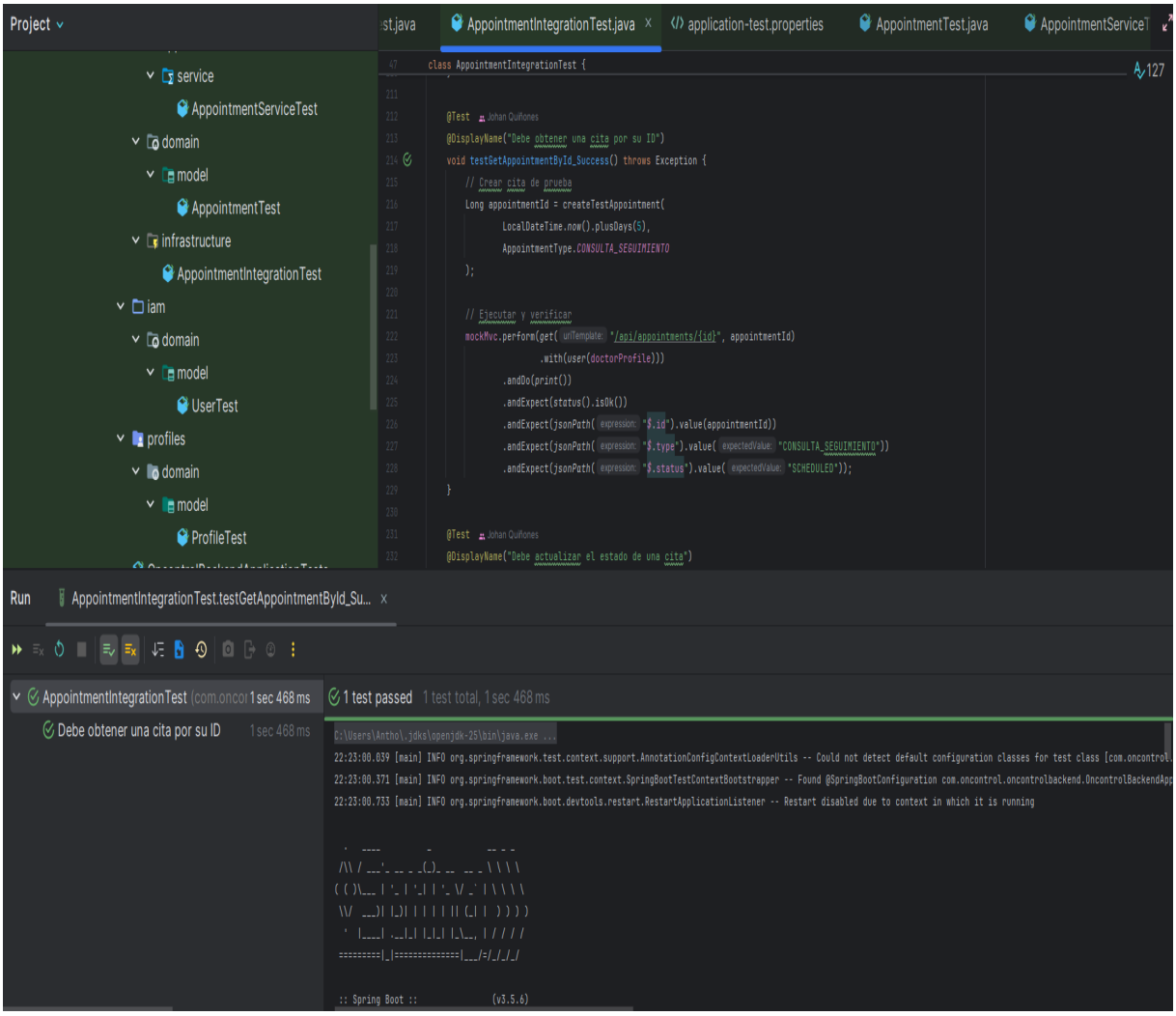
```
        mockMvc.perform(get("/api/appointments/doctor/{doctorId}",
doctorProfile.getId())
                        .with(user(doctorProfile)))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.appointments").isArray())
                .andExpect(jsonPath("$.appointments", hasSize(2)))
                .andExpect(jsonPath("$.count").value(2));
    }
```



3. **testGetPatientAppointments_Success**(AppointmentIntegrationTest)
   La prueba testGetPatientAppointments_Success verifica que el sistema pueda obtener correctamente todas las citas registradas para un paciente determinado. A través de MockMvc, se simula una solicitud HTTP GET al endpoint correspondiente, comprobando que la respuesta tenga el estado 200 (OK) y que contenga un arreglo con el número esperado de citas. Esta prueba garantiza la correcta recuperación y visualización de la información médica asociada a cada paciente dentro del sistema.

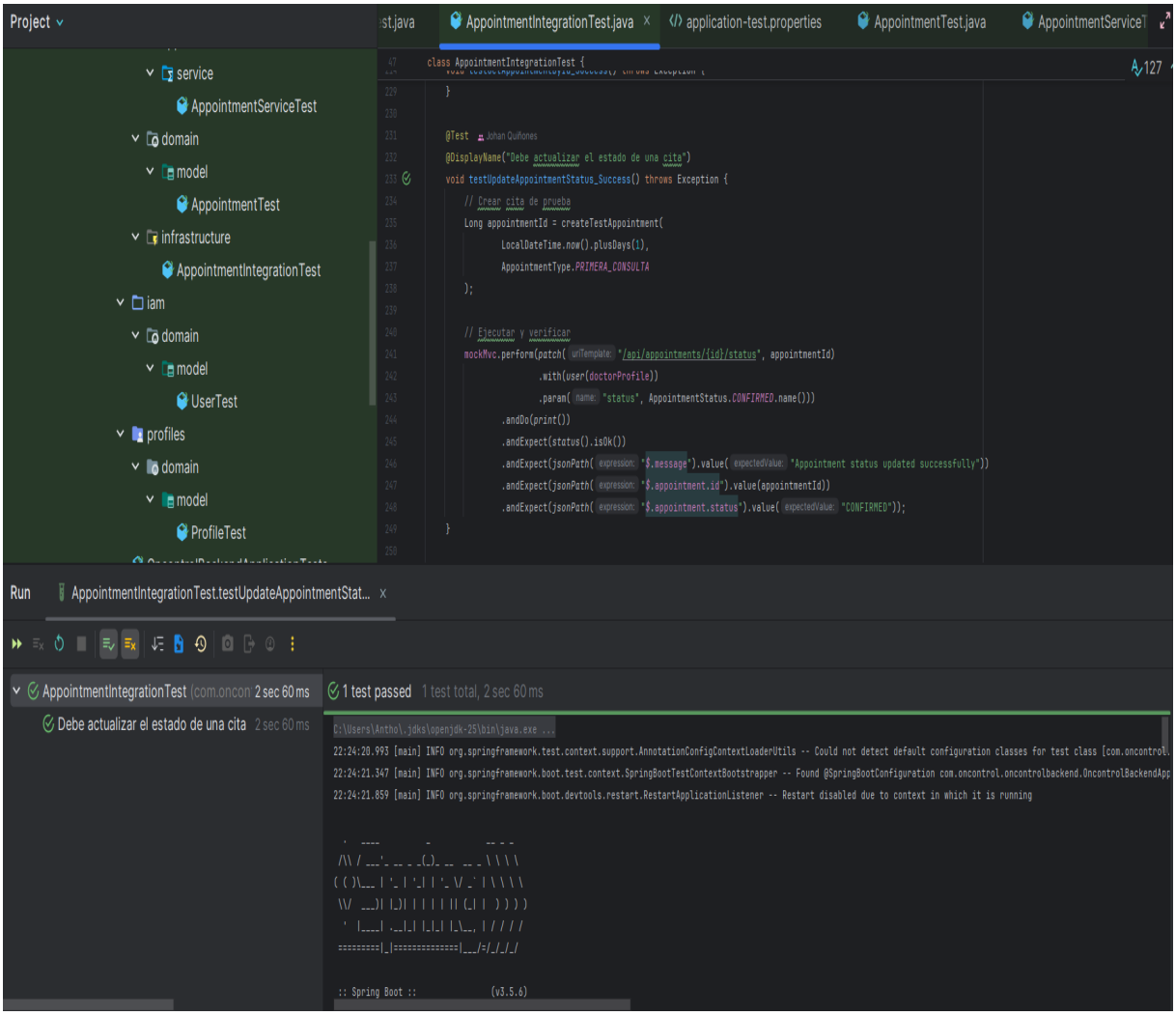
```
@Test
    @DisplayName("Debe obtener todas las citas de un paciente")
    void testGetPatientAppointments_Success() throws Exception {
        // Crear citas de prueba
```

```
        createTestAppointment(LocalDateTime.now().plusDays(1),
AppointmentType.PRIMERA_CONSULTA);
        createTestAppointment(LocalDateTime.now().plusDays(3),
AppointmentType.REVISION_TRATAMIENTO);

        // Ejecutar y verificar
        mockMvc.perform(get("/api/appointments/patient/{patientId}",
patientProfile.getId())
                        .with(user(patientProfile)))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.appointments").isArray())
                .andExpect(jsonPath("$.appointments", hasSize(2)))
                .andExpect(jsonPath("$.count").value(2));
    }
```



4. **testGetAppointmentById_Success**(AppointmentIntegrationTest)
   La prueba testGetAppointmentById_Success comprueba que el sistema pueda recuperar correctamente una cita específica utilizando su identificador único. Mediante MockMvc, se simula una solicitud HTTP GET al endpoint correspondiente y se valida que la respuesta tenga el estado 200 (OK). Además, se verifica que los datos devueltos, como el ID, el tipo y el estado de la cita, coincidan con los valores esperados, garantizando la fiabilidad del proceso de consulta individual de citas médicas.

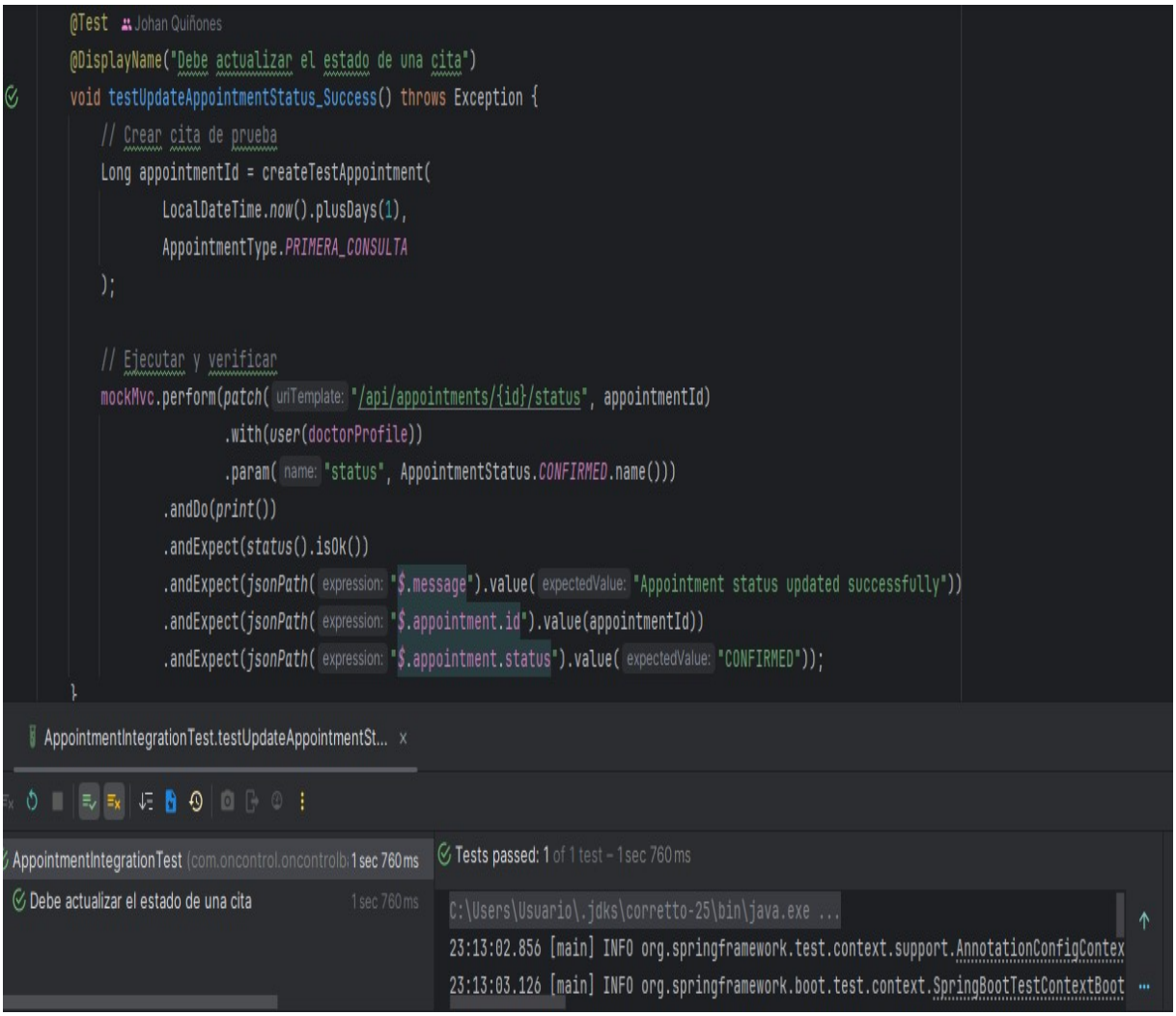
```java
@Test
    @DisplayName("Debe obtener una cita por su ID")
    void testGetAppointmentById_Success() throws Exception {
        // Crear cita de prueba
        Long appointmentId = createTestAppointment(
                LocalDateTime.now().plusDays(5),
                AppointmentType.CONSULTA_SEGUIMIENTO
        );

        // Ejecutar y verificar
        mockMvc.perform(get("/api/appointments/{id}", appointmentId)
                        .with(user(doctorProfile)))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(appointmentId))
                .andExpect(jsonPath("$.type").value("CONSULTA_SEGUIMIENTO"))
                .andExpect(jsonPath("$.status").value("SCHEDULED"));
    }
```



5. **testUpdateAppointmentStatus_Success**(AppointmentIntegrationTest)
   La prueba testUpdateAppointmentStatus_Success comprueba que el sistema pueda actualizar

correctamente el estado de una cita médica existente. Para ello, se crea una cita de prueba con una fecha futura y se ejecuta una solicitud HTTP PATCH al endpoint correspondiente, autenticando la acción con un perfil de doctor. Finalmente, se valida que la respuesta tenga el estado 200 (OK), contenga un mensaje de confirmación y refleje el nuevo estado CONFIRMED, garantizando el correcto funcionamiento del proceso de actualización de citas.

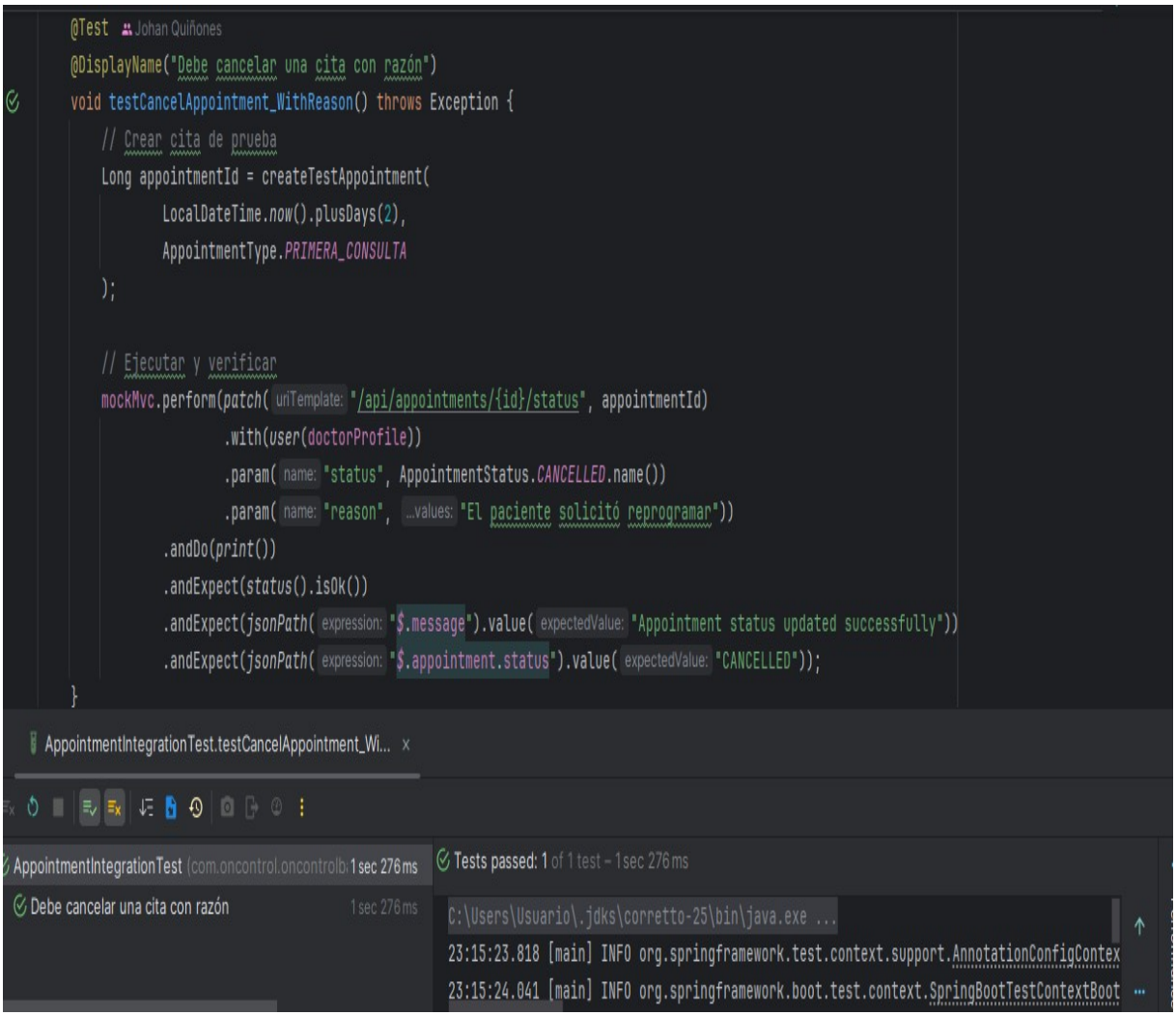
```
@Test
    @DisplayName("Debe actualizar el estado de una cita")
    void testUpdateAppointmentStatus_Success() throws Exception {
        // Crear cita de prueba
        Long appointmentId = createTestAppointment(
                LocalDateTime.now().plusDays(1),
                AppointmentType.PRIMERA_CONSULTA
        );

        // Ejecutar y verificar
        mockMvc.perform(patch("/api/appointments/{id}/status", appointmentId)
                        .with(user(doctorProfile))
                        .param("status", AppointmentStatus.CONFIRMED.name()))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.message").value("Appointment status updated
successfully"))
                .andExpect(jsonPath("$.appointment.id").value(appointmentId))
                .andExpect(jsonPath("$.appointment.status").value("CONFIRMED"));
    }
```



6. **testCancelAppointment_WithReason**(AppointmentIntegrationTest)
   La prueba testCancelAppointment_WithReason comprueba que el sistema permita cancelar correctamente una cita médica registrando una razón específica. Para ello, se crea una cita de prueba con una fecha futura y se ejecuta una solicitud HTTP PATCH al endpoint correspondiente, autenticando la acción con un perfil de doctor y enviando el nuevo estado CANCELLED junto con el motivo de cancelación. Finalmente, se valida que la respuesta tenga el estado 200 (OK), un mensaje de confirmación y el estado actualizado, asegurando el correcto funcionamiento del proceso de cancelación de citas.

```
@Test
    @DisplayName("Debe cancelar una cita con razón")
    void testCancelAppointment_WithReason() throws Exception {
        // Crear cita de prueba
        Long appointmentId = createTestAppointment(
                LocalDateTime.now().plusDays(2),
                AppointmentType.PRIMERA_CONSULTA
        );

        // Ejecutar y verificar
        mockMvc.perform(patch("/api/appointments/{id}/status", appointmentId)
                        .with(user(doctorProfile))
                        .param("status", AppointmentStatus.CANCELLED.name())
                        .param("reason", "El paciente solicitó reprogramar"))
```

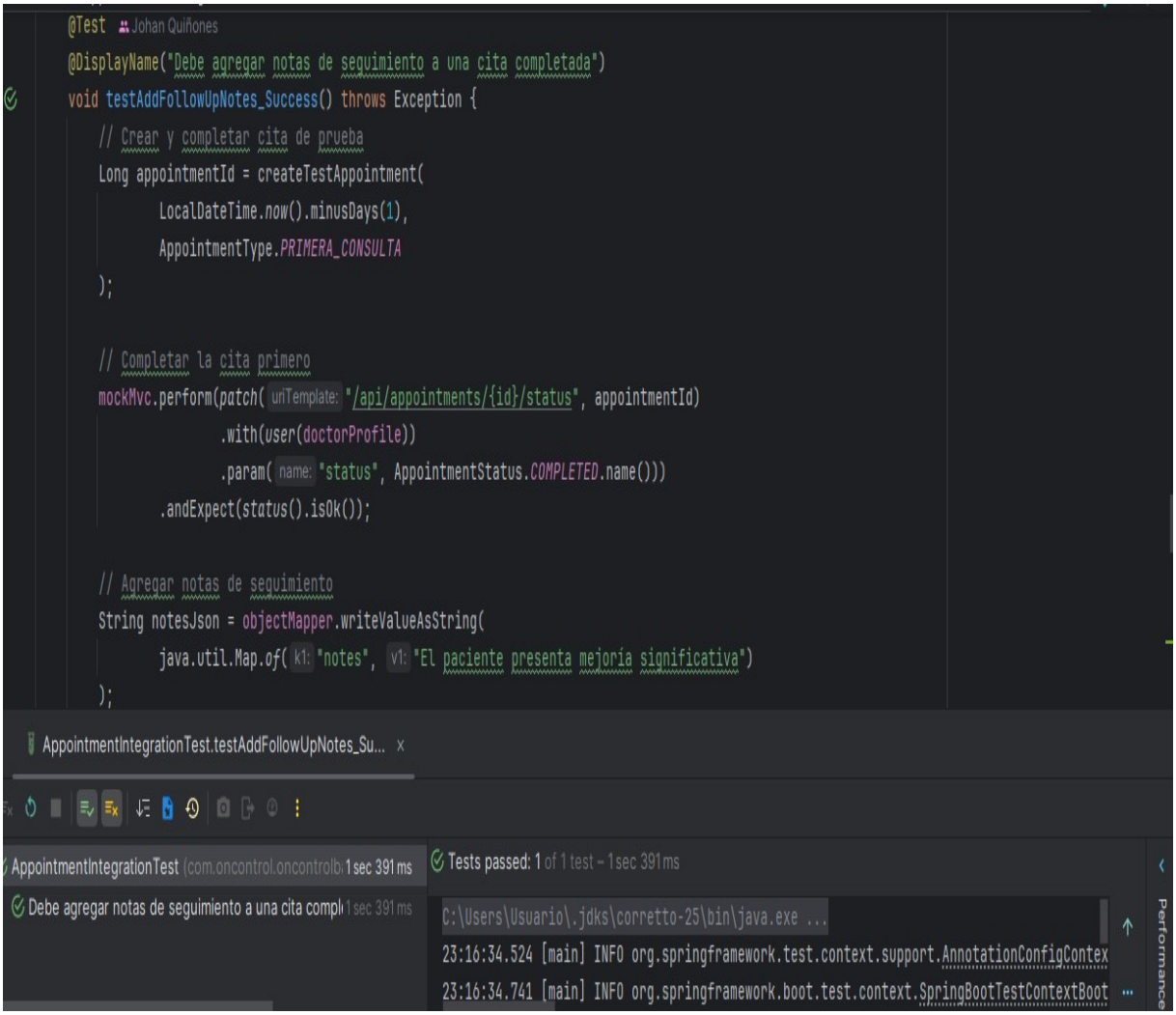
```
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.message").value("Appointment status updated
successfully"))
                .andExpect(jsonPath("$.appointment.status").value("CANCELLED"));
    }
```



7. **testAddFollowUpNotes_Success**(AppointmentIntegrationTest)
   La prueba testAddFollowUpNotes_Success comprueba que el sistema permita agregar correctamente notas de seguimiento a una cita previamente completada. Para ello, se crea una cita de prueba con fecha pasada, se actualiza su estado a COMPLETED y posteriormente se envía una solicitud HTTP PATCH al endpoint correspondiente con las notas de seguimiento en formato JSON. Finalmente, se valida que la respuesta tenga el estado 200 (OK), incluya un mensaje de confirmación y los datos de la cita actualizada, garantizando el correcto registro del seguimiento médico.

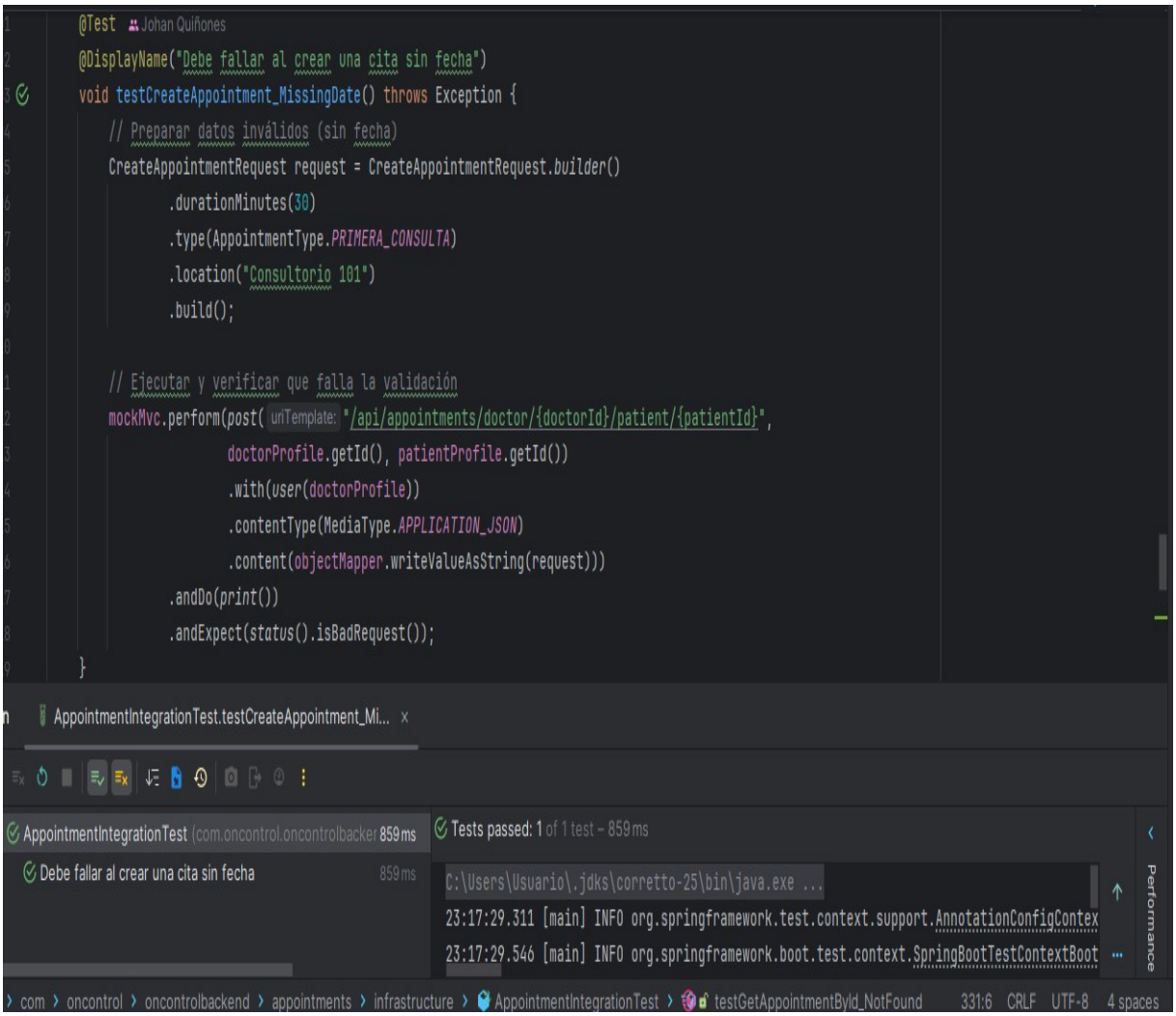
```
@Test
    @DisplayName("Debe agregar notas de seguimiento a una cita completada")
    void testAddFollowUpNotes_Success() throws Exception {
        // Crear y completar cita de prueba
        Long appointmentId = createTestAppointment(
                LocalDateTime.now().minusDays(1),
                AppointmentType.PRIMERA_CONSULTA
```

```
        );

        // Completar la cita primero
        mockMvc.perform(patch("/api/appointments/{id}/status", appointmentId)
                        .with(user(doctorProfile))
                        .param("status", AppointmentStatus.COMPLETED.name()))
                .andExpect(status().isOk());

        // Agregar notas de seguimiento
        String notesJson = objectMapper.writeValueAsString(
                java.util.Map.of("notes", "El paciente presenta mejoría
significativa")
        );

        mockMvc.perform(patch("/api/appointments/{id}/follow-up", appointmentId)
                        .with(user(doctorProfile))
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(notesJson))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.message").value("Follow-up notes added
successfully"))
                .andExpect(jsonPath("$.appointment.id").value(appointmentId));
    }
```



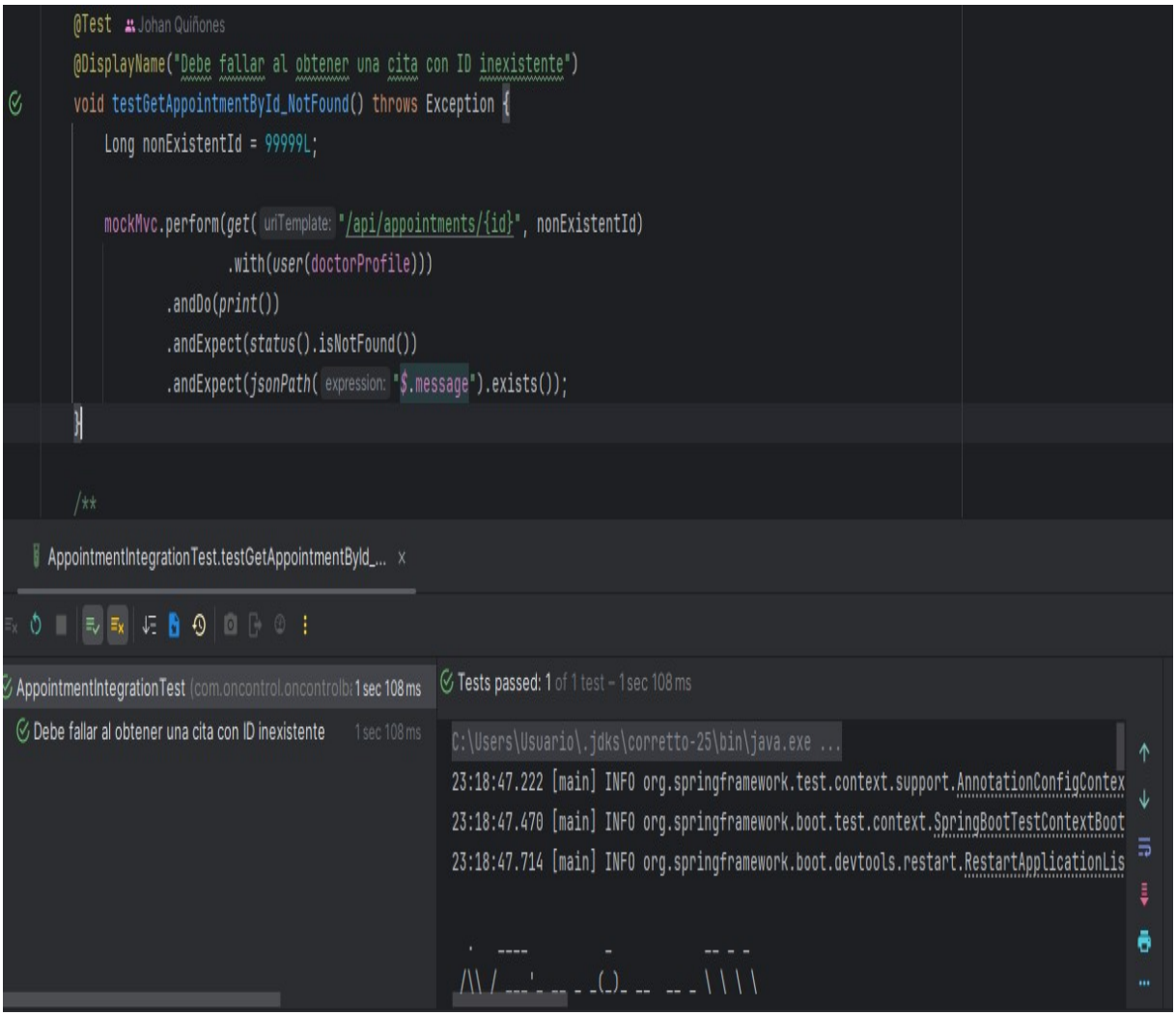
8. **testCreateAppointment_MissingDate**(AppointmentIntegrationTest)
   La prueba testCreateAppointment_MissingDate comprueba que el sistema valide correctamente los datos obligatorios al intentar crear una cita médica sin especificar una fecha. Para ello, se construye una solicitud con información incompleta y se envía mediante una petición HTTP POST al endpoint correspondiente. Finalmente, se verifica que la respuesta tenga el estado 400 (Bad Request), asegurando que el sistema rechace apropiadamente solicitudes con datos inválidos.

```
@Test
    @DisplayName("Debe fallar al crear una cita sin fecha")
    void testCreateAppointment_MissingDate() throws Exception {
        // Preparar datos inválidos (sin fecha)
        CreateAppointmentRequest request = CreateAppointmentRequest.builder()
                .durationMinutes(30)
                .type(AppointmentType.PRIMERA_CONSULTA)
                .location("Consultorio 101")
                .build();

        // Ejecutar y verificar que falla la validación

mockMvc.perform(post("/api/appointments/doctor/{doctorId}/patient/{patientId}",
                        doctorProfile.getId(), patientProfile.getId())
                        .with(user(doctorProfile))
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(objectMapper.writeValueAsString(request)))
                .andDo(print())
                .andExpect(status().isBadRequest());
    }
```



9. **testGetAppointmentById_NotFound**(AppointmentIntegrationTest)
   La prueba testGetAppointmentById_NotFound comprueba que el sistema maneje correctamente la búsqueda de una cita médica utilizando un identificador inexistente. Para ello, se realiza una solicitud HTTP GET al endpoint correspondiente con un ID no registrado en la base de datos. Finalmente, se valida que la respuesta tenga el estado 404 (Not Found) y contenga un mensaje descriptivo, garantizando el adecuado manejo de errores en consultas de citas inexistentes.

```
@Test
    @DisplayName("Debe fallar al obtener una cita con ID inexistente")
    void testGetAppointmentById_NotFound() throws Exception {
        Long nonExistentId = 99999L;

        mockMvc.perform(get("/api/appointments/{id}", nonExistentId)
                        .with(user(doctorProfile)))
                .andDo(print())
                .andExpect(status().isNotFound())
                .andExpect(jsonPath("$.message").exists());
    }
```



6.1.3. Core Behavior-Driven Development

En esta sección se describe cómo el equipo utiliza la metodología Behavior-Driven Development (BDD) para asegurar que el sistema cumpla con los requisitos funcionales desde la perspectiva del usuario final. Se emplean escenarios escritos en lenguaje natural, utilizando la herramienta Cucumber junto con Gherkin, para definir y automatizar pruebas que validan los flujos principales de la aplicación, como el agendamiento de citas médicas. Esto permite una mejor comunicación entre desarrolladores, testers y stakeholders, facilitando la detección temprana de errores y asegurando que el software entregue valor real al usuario.

Acceptance Tests:

- appointment-booking.feature:

```gherkin
Feature: Agendamiento de citas médicas
  Como paciente
  Quiero agendar citas con médicos especialistas
  Para recibir atención médica oncológica

  Scenario: Agendar cita médica exitosamente
    Given el paciente ha iniciado sesión
    And está en la sección "Agendar Cita"
    When busca especialidad "Oncología"
    And selecciona al doctor "Dr. Williams Gongora"
    And elige fecha "15 de octubre" y hora "10:00 AM"
    And selecciona tipo de cita "Consulta"
    And ingresa notas "Chequeo regular"
    And hace clic en "Confirmar Cita"
    Then ve el mensaje "Cita agendada exitosamente"
    And la cita aparece en "Mis Citas" con estado "Programada"
    And recibe notificación de confirmación

  Scenario: Intento de agendar sin doctor disponible
    Given el paciente está agendando una cita
    When intenta agendar con un doctor que no existe
    Then ve el mensaje "Doctor no encontrado"
    And la cita no se agenda
```

- appointment-list.feature:

```gherkin
Feature: Visualización de citas agendadas
  Como médico oncólogo
  Quiero ver todas mis citas programadas
  Para organizar mi agenda médica

  Scenario: Ver lista de citas del día
    Given el médico ha iniciado sesión
    When accede a "Mis Citas"
    Then ve todas sus citas programadas ordenadas por fecha
    And cada cita muestra: paciente, fecha, hora y tipo
    And puede hacer clic en una cita para ver detalles completos

  Scenario: Médico sin citas programadas
    Given el médico no tiene citas agendadas
    When accede a "Mis Citas"
    Then ve el mensaje "No tienes citas agendadas"
    And ve la opción "Ver solicitudes de citas"
```

- landing-contact.feature:

```gherkin
Feature: Formulario de contacto
  Como visitante interesado
  Quiero contactar al equipo de OnControl
  Para obtener más información o resolver dudas

  Scenario: Enviar consulta exitosamente
    Given estoy en la sección "Contacto"
    When completo el formulario con mis datos
      | Nombre | Email | Tipo Usuario | Mensaje |
      | Juan Pérez | juan@email.com | Paciente | ¿Cómo puedo registrarme? |
    And hago clic en "Enviar Mensaje"
    Then veo el mensaje "Mensaje enviado exitosamente"
    And veo "Te contactaremos pronto"
    And recibo email de confirmación

  Scenario: Intento de envío con datos incompletos
    Given estoy completando el formulario de contacto
    When dejo el campo email vacío
    And hago clic en "Enviar Mensaje"
    Then veo el mensaje "Por favor complete todos los campos"
    And el formulario no se envía
```

- landing-navigation.feature:

```gherkin
Feature: Navegación en landing page
  Como visitante de OnControl
  Quiero navegar fácilmente por la página
  Para conocer el producto y sus beneficios

  Scenario: Visualizar información principal
    Given estoy visitando la página principal de OnControl
    When la página carga completamente
    Then veo el título "Apoyo integral para pacientes oncológicos"
    And veo una descripción clara del propósito
    And veo una imagen de médico y paciente usando la plataforma
    And veo el menú principal con: Características, Beneficios, Problemática, Testimonios, Contacto
    And el menú permanece visible cuando hago scroll

  Scenario Outline: Navegar a diferentes secciones
    Given estoy en la página principal
    When hago clic en el menú "<seccion>"
    Then la página se desplaza automáticamente a "<seccion>"
    And veo el contenido relacionado a "<contenido_esperado>"

    Examples:
      | seccion         | contenido_esperado                        |
      | Características | Calendario, medicamentos, comunicación    |
      | Beneficios      | Ventajas para médicos, pacientes y familiares|
      | Problemática    | Estadísticas del cáncer en Perú           |
      | Testimonios     | Experiencias de usuarios reales           |
```

- login.feature:

```gherkin
Feature: Inicio de sesión
  Como usuario registrado
  Quiero iniciar sesión con mis credenciales
  Para acceder a mi cuenta personalizada

  Scenario: Inicio de sesión exitoso
    Given el usuario tiene una cuenta registrada como "org@test.com"
    And está en la página de inicio de sesión
    When ingresa email "org@test.com" y contraseña correcta
    And hace clic en "Iniciar Sesión"
    Then accede a su dashboard exitosamente
    And ve su nombre de usuario en la barra superior

  Scenario: Intento de acceso con cuenta inactiva
    Given el usuario "test@test.com" tiene su cuenta desactivada
    When intenta iniciar sesión con sus credenciales
    Then ve el mensaje "Cuenta inactiva. Contacte al administrador"
    And no puede acceder al sistema
```

- password-change.feature:

```gherkin
Feature: Cambio de contraseña
  Como usuario autenticado
  Quiero cambiar mi contraseña
  Para mantener mi cuenta segura

  Scenario: Cambio exitoso de contraseña
    Given el usuario está en su perfil
    And hace clic en "Cambiar contraseña"
    When ingresa su contraseña actual correctamente
    And ingresa una nueva contraseña válida
    And confirma la nueva contraseña
    Then ve el mensaje "Contraseña actualizada exitosamente"
    And su sesión se cierra automáticamente

  Scenario: Error al ingresar contraseña actual incorrecta
    Given el usuario está cambiando su contraseña
    When ingresa una contraseña actual incorrecta
    Then ve el mensaje "Contraseña actual incorrecta"
    And la contraseña no se actualiza
```

- profile-management.feature:

```gherkin
Feature: Gestión de perfil de usuario
  Como usuario del sistema
  Quiero gestionar mi información personal
  Para mantener mis datos actualizados

  Scenario: Ver información de perfil de doctor
    Given el doctor ha iniciado sesión
    When accede a "Mi Perfil"
    Then ve su nombre completo "Dr. Williams Gongora"
    And ve su identificador único que comienza con "DOC-"
    And ve su especialidad médica
    And ve su información de contacto

  Scenario: Ver información de perfil de paciente
    Given el paciente ha iniciado sesión
    When accede a "Mi Perfil"
    Then ve su nombre completo "Johan Perez"
    And ve su identificador único que comienza con "PAT-"
    And ve su historial médico
    And ve sus tratamientos activos

  Scenario: Perfil activo desde la creación
    Given un nuevo usuario completa su registro
    When accede por primera vez a su perfil
    Then su cuenta está activa automáticamente
    And puede usar todas las funcionalidades del sistema
```

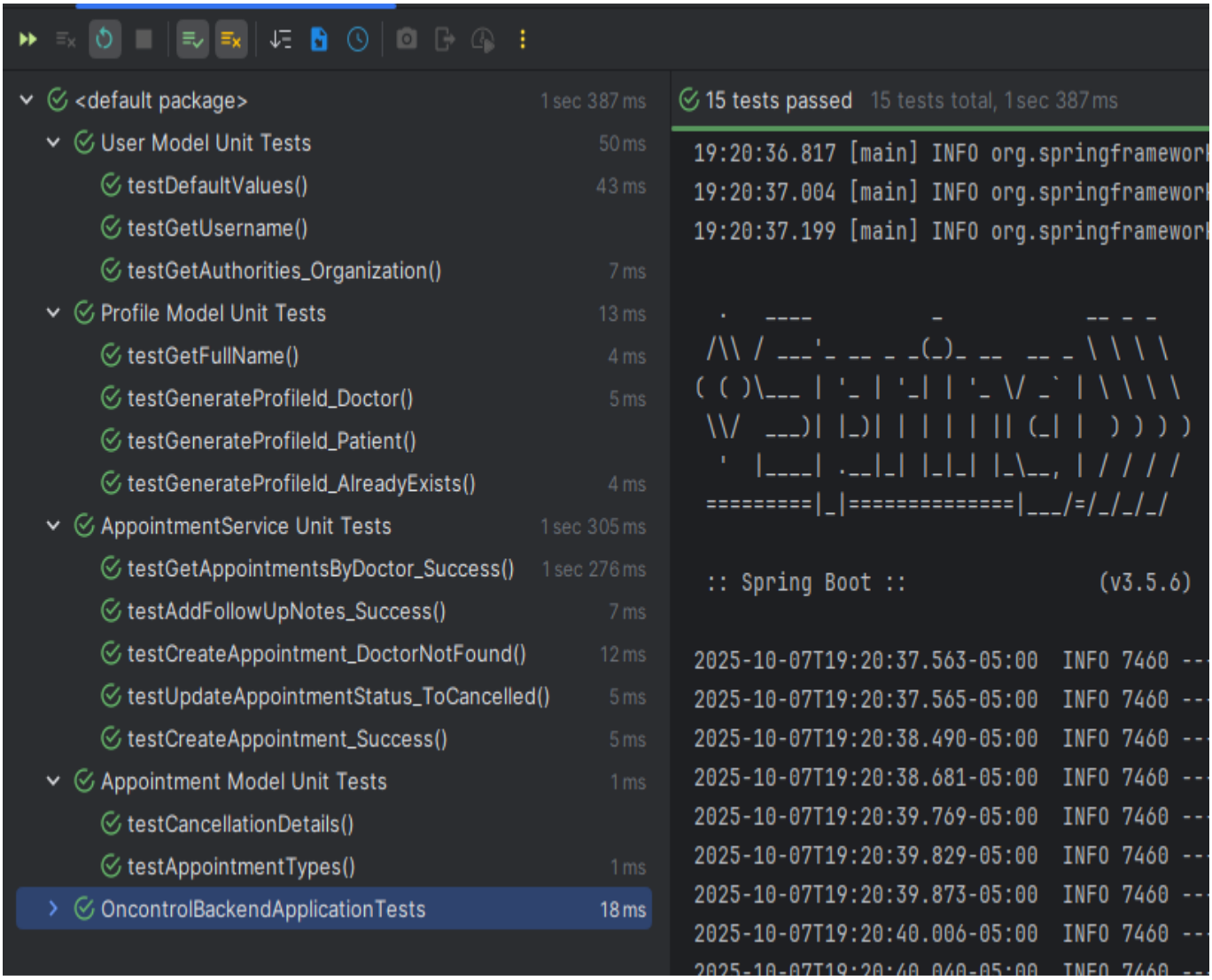
- user-registration.feature:

```
Feature: Registro de cuenta de usuario
  Como usuario general
  Quiero registrarme en la plataforma OnControl
  Para acceder a las funcionalidades del sistema

  Scenario: Registro exitoso de nueva cuenta
    Given el usuario está en la página de registro
    When ingresa sus datos personales
      | Email | Contraseña | Organización | País | Ciudad |
      | test@test.com | password | Test Org | PE | Lima |
    And hace clic en "Registrarse"
    Then ve el mensaje "Cuenta creada exitosamente"
    And es redirigido al dashboard
    And su cuenta está activa

  Scenario: Intento de registro con email existente
    Given existe una cuenta con email "test@test.com"
    When el usuario intenta registrarse con ese mismo email
    Then ve el mensaje "El email ya está registrado"
    And permanece en la página de registro
```

6.1.4. Core System Tests

En esta sección se presentan las pruebas desarrolladas para verificar el correcto funcionamiento de los componentes individuales de la aplicación. Estas pruebas aseguran que cada clase y método cumpla con su responsabilidad de manera aislada, contribuyendo a la calidad, mantenibilidad y robustez del sistema, y permitiendo detectar errores de forma temprana durante el ciclo de desarrollo.

Verificacion de todas las pruebas



- **Pruebas Unitarias:**

```java
    @Test  new *
    void testCreateAppointment_Success() {
        // Arrange
        CreateAppointmentRequest request = CreateAppointmentRequest.builder()
                .appointmentDate(LocalDateTime.now().plusDays(1))
                .durationMinutes(30)
                .type(AppointmentType.PRIMERA_CONSULTA)
                .location("Room 420")
                .notes("Regular checkup")
                .build();

        when(profileRepository.findById(1L)).thenReturn(Optional.of(doctorProfile));
        when(profileRepository.findById(2L)).thenReturn(Optional.of(patientProfile));
        when(appointmentRepository.save(any(Appointment.class))).thenReturn(appointment);

        AppointmentResponse response = appointmentService.createAppointment( doctorProfileId: 1L,  patientProfileId: 2L, request);

        assertNotNull(response);
        assertEquals( expected: 1L, response.getId());
        assertEquals( expected: "Williams Gongora", response.getDoctorName());
        assertEquals( expected: "Johan Quinonez", response.getPatientName());
        assertEquals(AppointmentStatus.SCHEDULED, response.getStatus());
        verify(appointmentRepository, times( wantedNumberOfInvocations: 1)).save(any(Appointment.class));
    }
```

```java
    @Test  new *
    void testCreateAppointment_DoctorNotFound() {
        CreateAppointmentRequest request = CreateAppointmentRequest.builder()
                .appointmentDate(LocalDateTime.now().plusDays(1))
                .type(AppointmentType.CONSULTA_GENERAL)
                .build();

        when(profileRepository.findById(1L)).thenReturn( t: Optional.empty());

        assertThrows(RuntimeException.class, () ->
                appointmentService.createAppointment( doctorProfileId: 1L,  patientProfileId: 2L, request)
        );
        verify(appointmentRepository, never()).save(any(Appointment.class));
    }
```

```
116
117     @Test  new *
118     void testGetAppointmentsByDoctor_Success() {
119         List<Appointment> appointments = Collections.singletonList(appointment);
120         when(appointmentRepository.findByDoctorId(1L)).thenReturn(appointments);
121
122         List<AppointmentResponse> responses = appointmentService.getAppointmentsByDoctor( doctorProfileId: 1L);
123
124         assertNotNull(responses);
125         assertEquals( expected: 1, responses.size());
126         assertEquals( expected: "Williams Gongora", responses.getFirst().getDoctorName());
127         verify(appointmentRepository, times( wantedNumberOfInvocations: 1)).findByDoctorId(1L);
128     }
129
```

```
130     @Test  new *
131     void testUpdateAppointmentStatus_ToCancelled() {
132         when(appointmentRepository.findById(1L)).thenReturn(Optional.of(appointment));
133         when(appointmentRepository.save(any(Appointment.class))).thenReturn(appointment);
134
135         AppointmentResponse response = appointmentService.updateAppointmentStatus(
136                 id: 1L,
137                 AppointmentStatus.CANCELLED,
138                 reason: "Patient requested cancellation"
139         );
140
141         assertNotNull(response);
142         assertEquals(AppointmentStatus.CANCELLED, appointment.getStatus());
143         assertEquals( expected: "Patient requested cancellation", appointment.getCancellationReason());
144         assertNotNull(appointment.getCancelledAt());
145         verify(appointmentRepository, times( wantedNumberOfInvocations: 1)).save(appointment);
146     }
147
```

```
148     @Test  new *
149     void testAddFollowUpNotes_Success() {
150         String followUpNotes = "Patient should return in 2 weeks for follow-up";
151         when(appointmentRepository.findById(1L)).thenReturn(Optional.of(appointment));
152         when(appointmentRepository.save(any(Appointment.class))).thenReturn(appointment);
153
154         AppointmentResponse response = appointmentService.addFollowUpNotes( id: 1L, followUpNotes);
155
156         assertNotNull(response);
157         assertEquals(followUpNotes, appointment.getFollowUpNotes());
158         verify(appointmentRepository, times( wantedNumberOfInvocations: 1)).save(appointment);
159     }
160 }
161
```

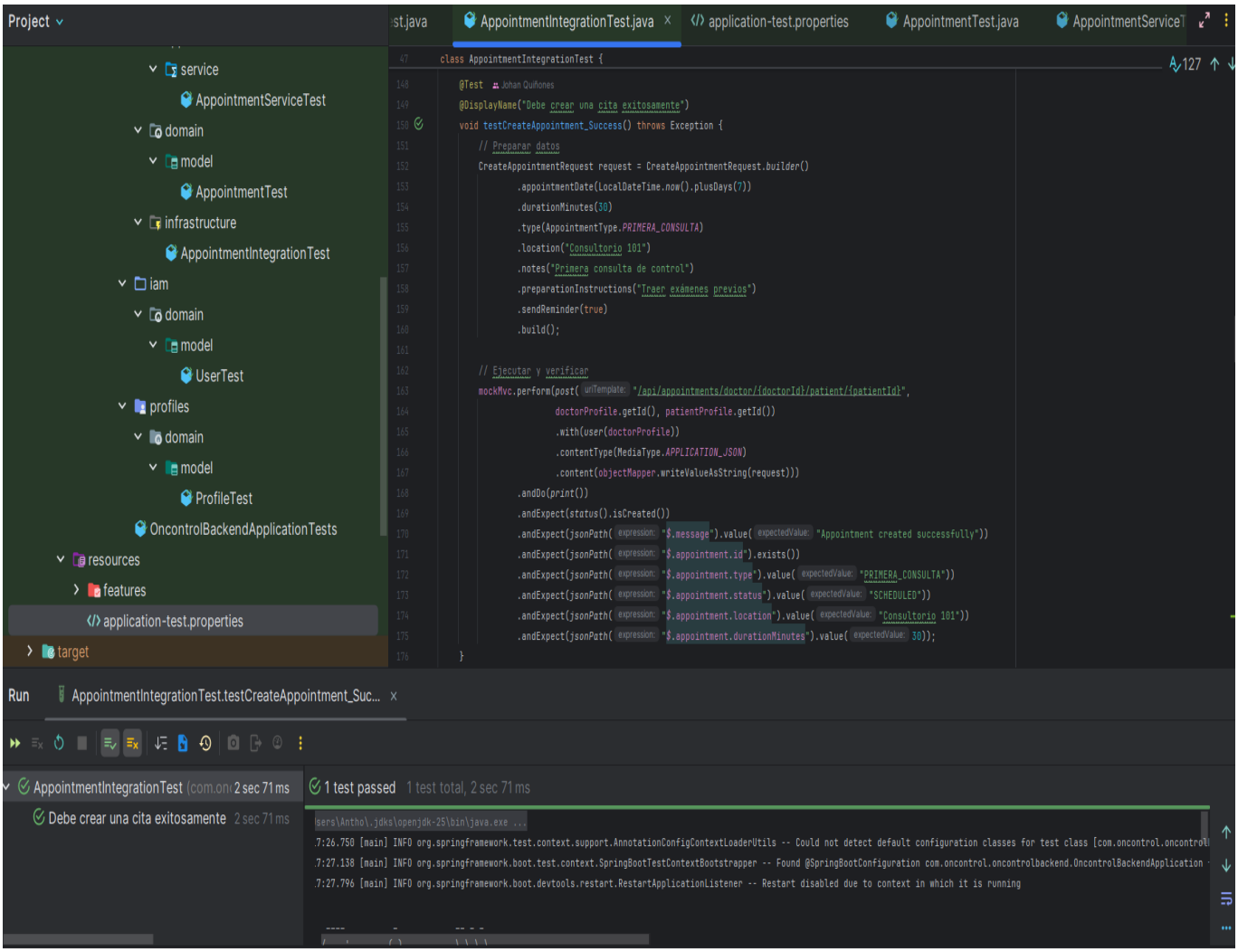
```java
    @Test  new *
    void testGenerateProfileId_Doctor() {
        Profile profile = Profile.builder()
                .profileType(ProfileType.DOCTOR)
                .firstName("Williams")
                .lastName("Gongora")
                .email("doctor@test.com")
                .password("password123")
                .build();

        profile.generateProfileId();

        assertNotNull(profile.getProfileId());
        assertTrue(profile.getProfileId().startsWith("DOC-"));
    }

    @Test  new *
    void testGenerateProfileId_Patient() {
        Profile profile = Profile.builder()
                .profileType(ProfileType.PATIENT)
                .firstName("Johan")
                .lastName("Quinonez")
                .email("patient@test.com")
                .password("password123")
                .build();

        profile.generateProfileId();

        assertNotNull(profile.getProfileId());
        assertTrue(profile.getProfileId().startsWith("PAT-"));
    }
```

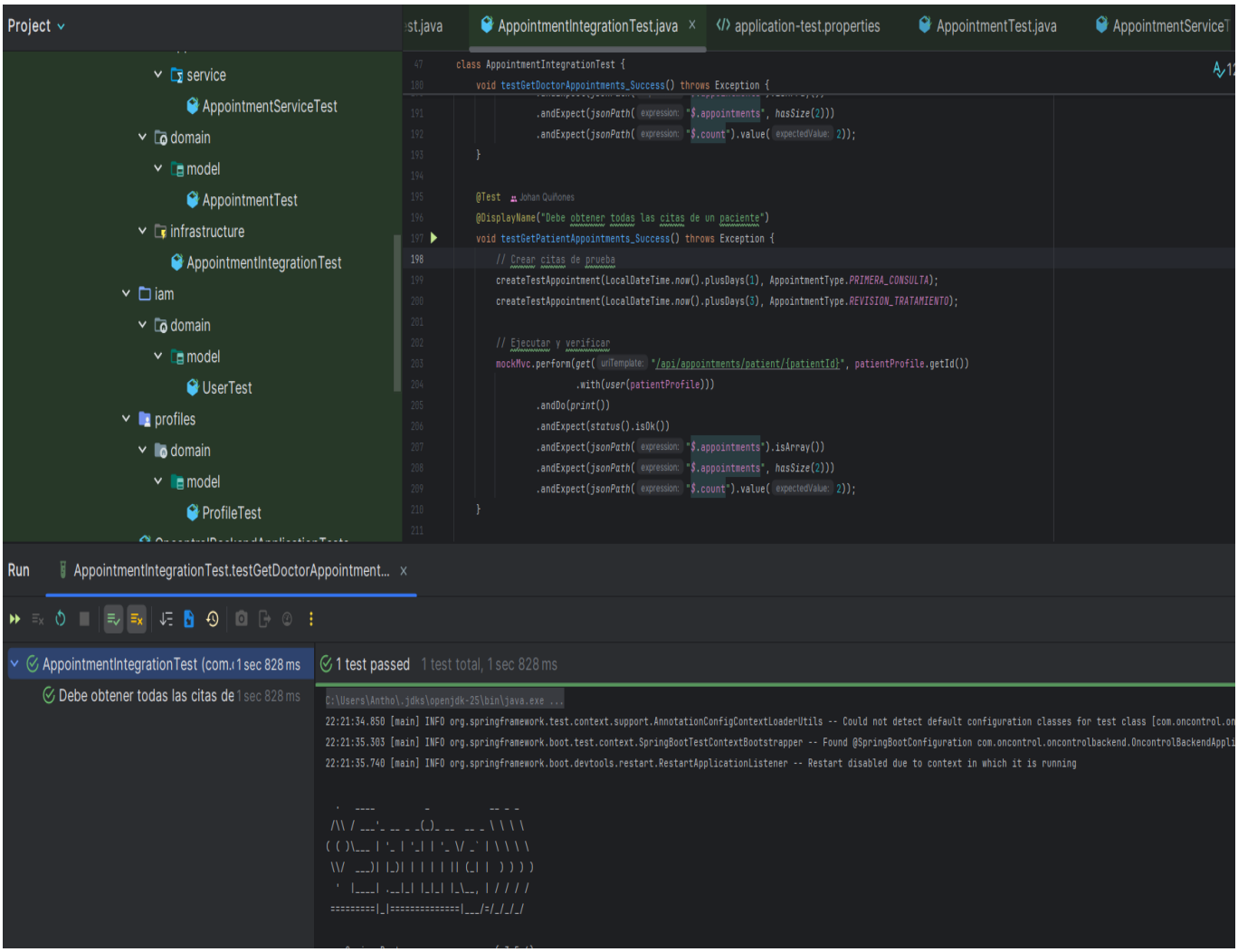
```java
@Test  new *
void testGetFullName() {
    Profile profile = Profile.builder()
            .firstName("Williams")
            .lastName("Gongora")
            .build();

    String fullName = profile.getFullName();

    assertEquals( expected: "Williams Gongora", fullName);
}
```

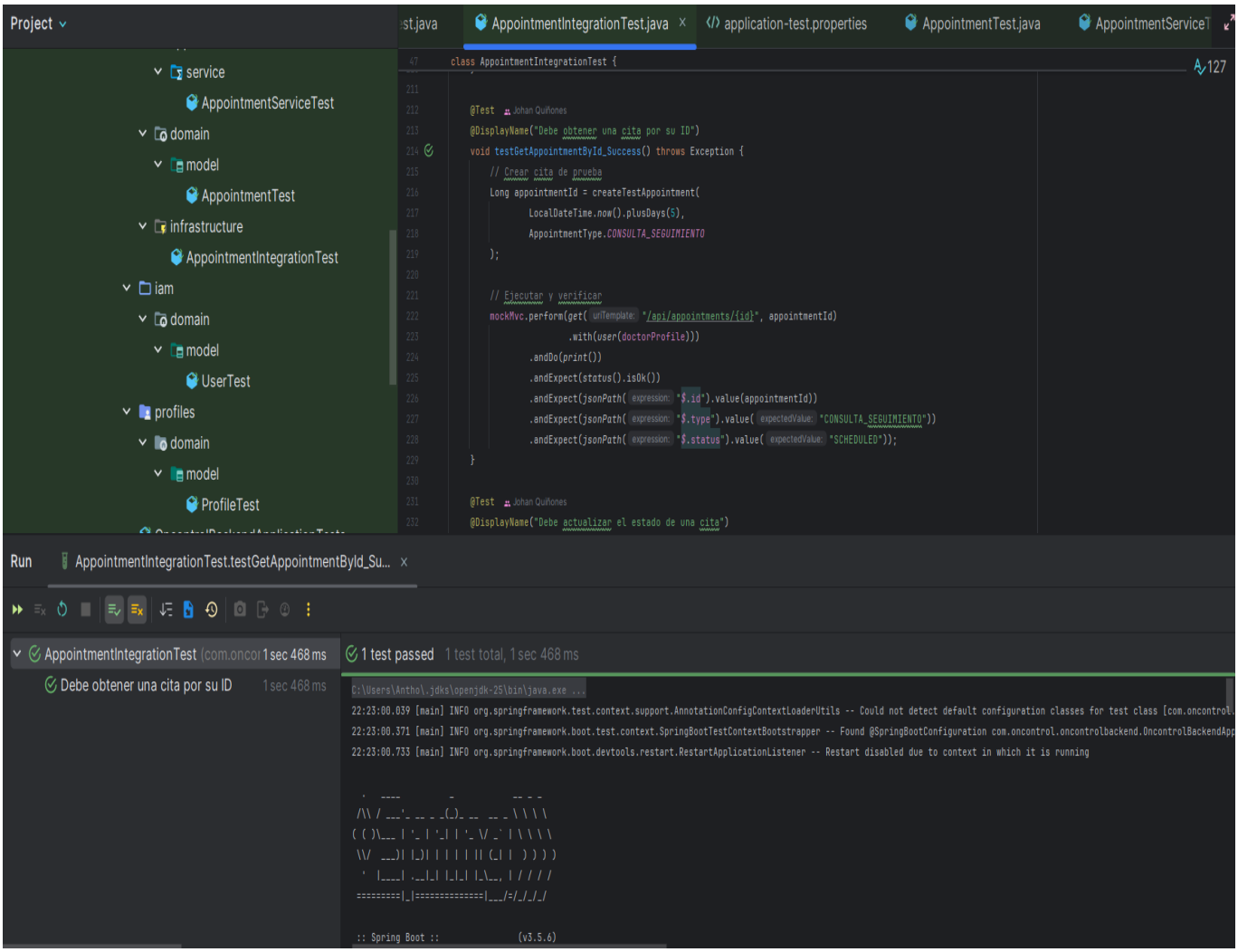
```java
@Test  new *
void testGetAuthorities_Organization() {

    User user = User.builder()
            .email("org@test.com")
            .password("password123")
            .organizationName("Test Hospital")
            .country("PE")
            .city("New York")
            .role(UserRole.ORGANIZATION)
            .build();

    Collection<? extends GrantedAuthority> authorities = user.getAuthorities();

    assertNotNull(authorities);
    assertEquals( expected: 1, authorities.size());
    assertTrue(authorities.stream()
            .anyMatch( capture of extends GrantedAuthority auth -> auth.getAuthority().equals("ROLE_ORGANIZATION")));
}
```

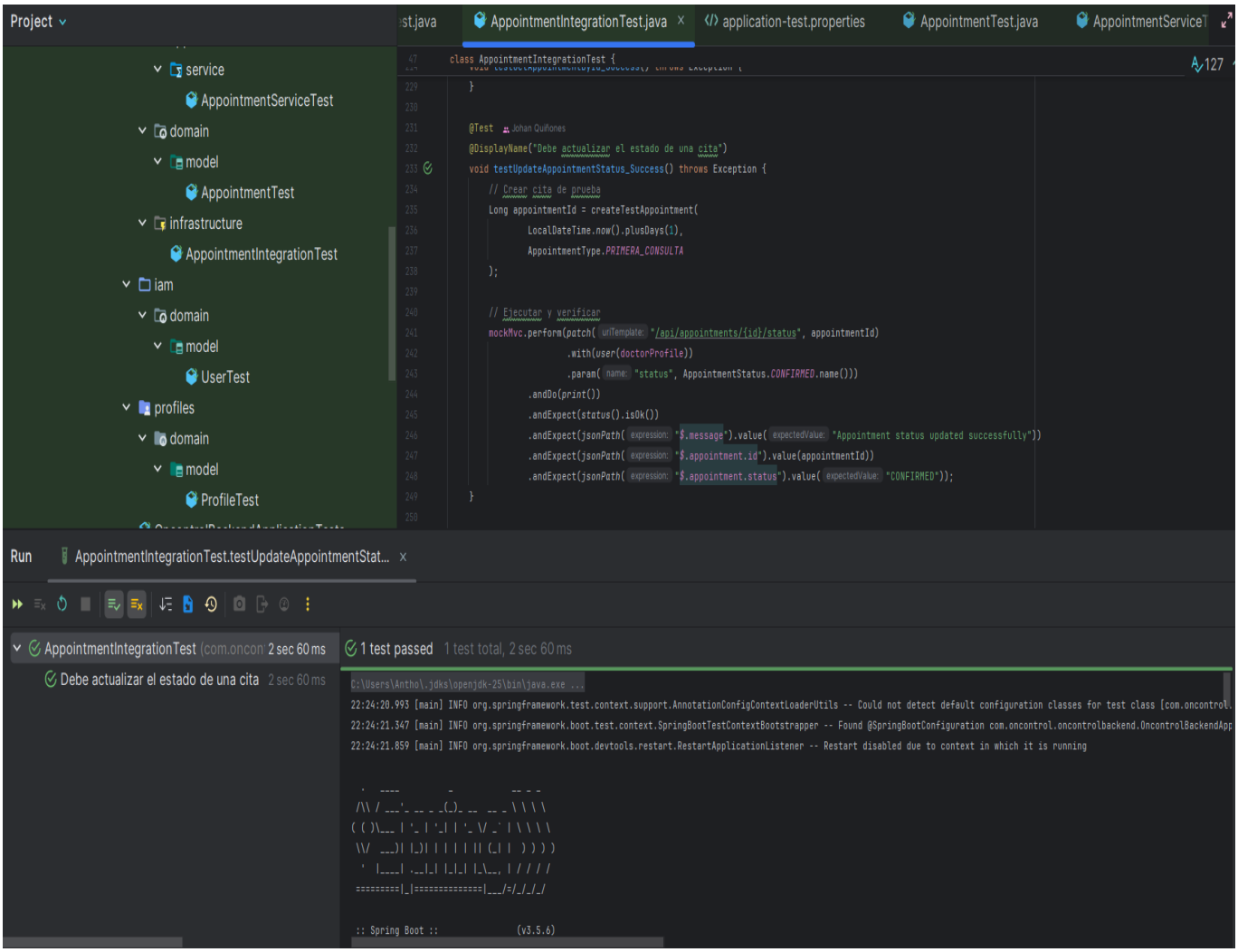
```java
    @Test  new *
    void testDefaultValues() {

        User user = User.builder()
                .email("test@test.com")
                .password("password")
                .organizationName("Test Org")
                .country("PE")
                .city("Lima")
                .build();

        assertEquals(UserRole.ORGANIZATION, user.getRole());
        assertTrue(user.getIsActive());
        assertFalse(user.getIsEmailVerified());
        assertTrue(user.isAccountNonExpired());
        assertTrue(user.isAccountNonLocked());
        assertTrue(user.isCredentialsNonExpired());
    }
}

```

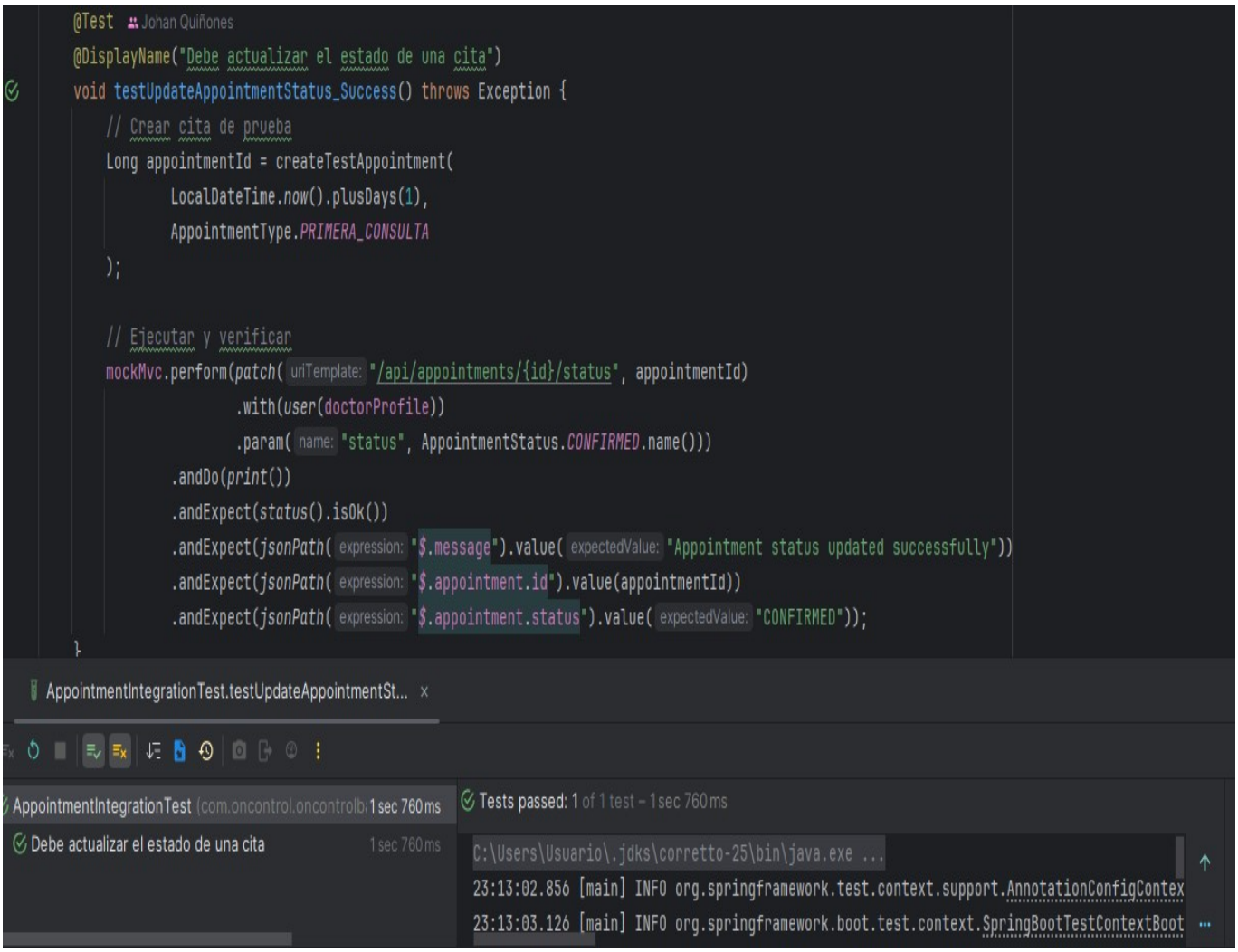
- **Pruebas de integración:**

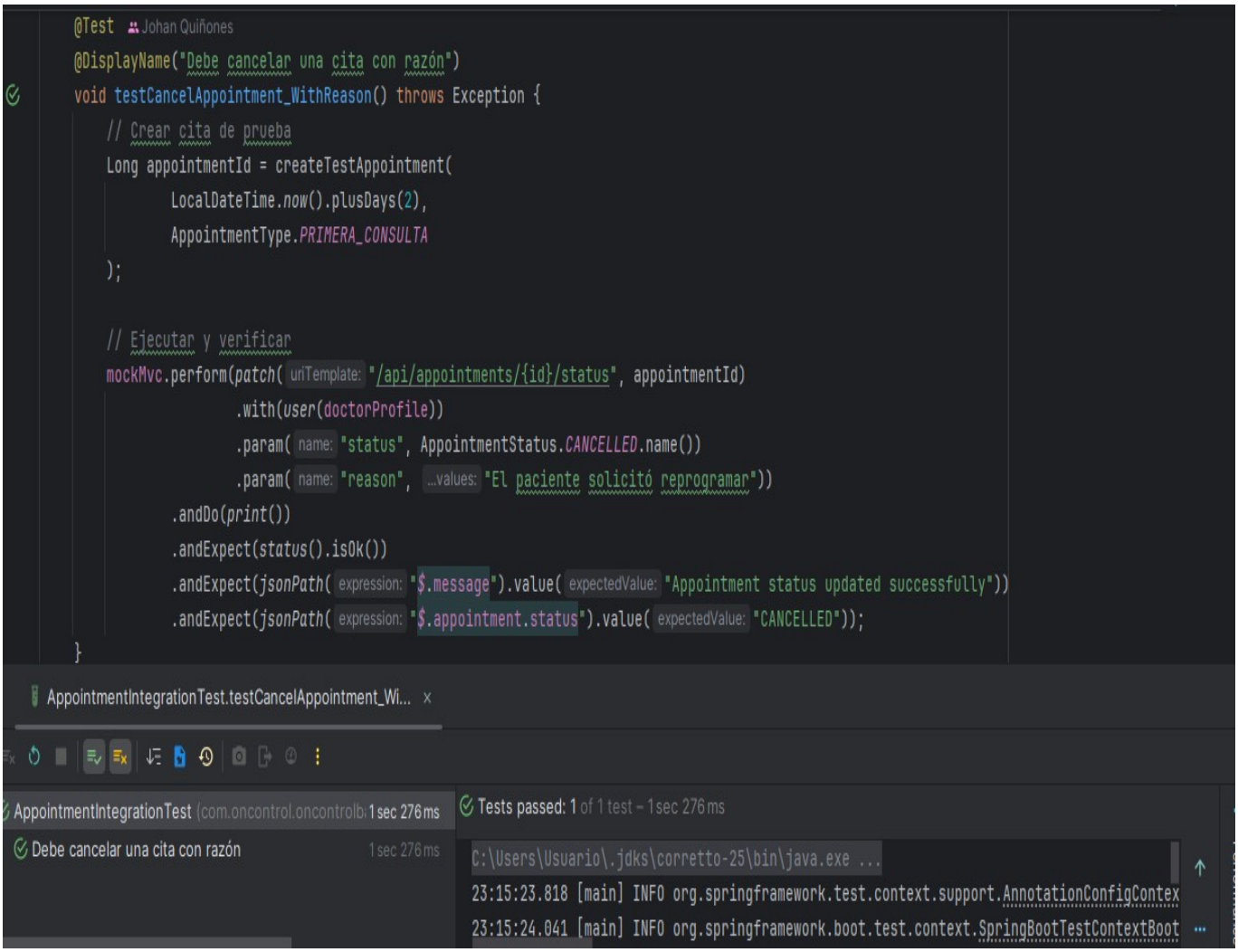

Project
AppointmentIntegrationTest.java
application-test.properties
AppointmentTest.java
service
AppointmentServiceTest
domain
model
AppointmentTest
infrastructure
AppointmentIntegrationTest
iam
UserTest
profiles
ProfileTest
OncontrolBackendApplicationTests
resources
features
application-test.properties
target
class AppointmentIntegrationTest {
@Test Johan Quiñones
@DisplayName("Debe crear una cita exitosamente")
void testCreateAppointment_Success() throws Exception {
// Preparar datos
CreateAppointmentRequest request = CreateAppointmentRequest.builder()
.appointmentDate(LocalDateTime.now().plusDays(7))
.durationMinutes(30)
.type(AppointmentType.PRIMERA_CONSULTA)
.location("Consultorio 101")
.notes("Primera consulta de control")
.preparationInstructions("Traer exámenes previos")
.sendReminder(true)
.build();
// Ejecutar y verificar
mockMvc.perform(post(uriTemplate: "/api/appointments/doctor/{doctorId}/patient/{patientId}",
doctorProfile.getId(), patientProfile.getId())
.with(user(doctorProfile))
.contentType(MediaType.APPLICATION_JSON)
.content(objectMapper.writeValueAsString(request)))
.andDo(print())
.andExpect(status().isCreated())
.andExpect(jsonPath(expression: "$.message").value(expectedValue: "Appointment created successfully"))
.andExpect(jsonPath(expression: "$.appointment.id").exists())
.andExpect(jsonPath(expression: "$.appointment.type").value(expectedValue: "PRIMERA_CONSULTA"))
.andExpect(jsonPath(expression: "$.appointment.status").value(expectedValue: "SCHEDULED"))
.andExpect(jsonPath(expression: "$.appointment.location").value(expectedValue: "Consultorio 101"))
.andExpect(jsonPath(expression: "$.appointment.durationMinutes").value(expectedValue: 30));
}
Run
AppointmentIntegrationTest.testCreateAppointment_Suc...
AppointmentIntegrationTest
2 sec 71 ms
Debe crear una cita exitosamente
2 sec 71 ms
1 test passed
1 test total, 2 sec 71 ms
sers\Antho\.jdks\openjdk-25\bin\java.exe ...
.7:26.750 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not detect default configuration classes for test class [com.oncontrol.oncontrol
.7:27.138 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper -- Found @SpringBootConfiguration com.oncontrol.oncontrolbackend.OncontrolBackendApplication
.7:27.796 [main] INFO org.springframework.boot.devtools.restart.RestartApplicationListener -- Restart disabled due to context in which it is running

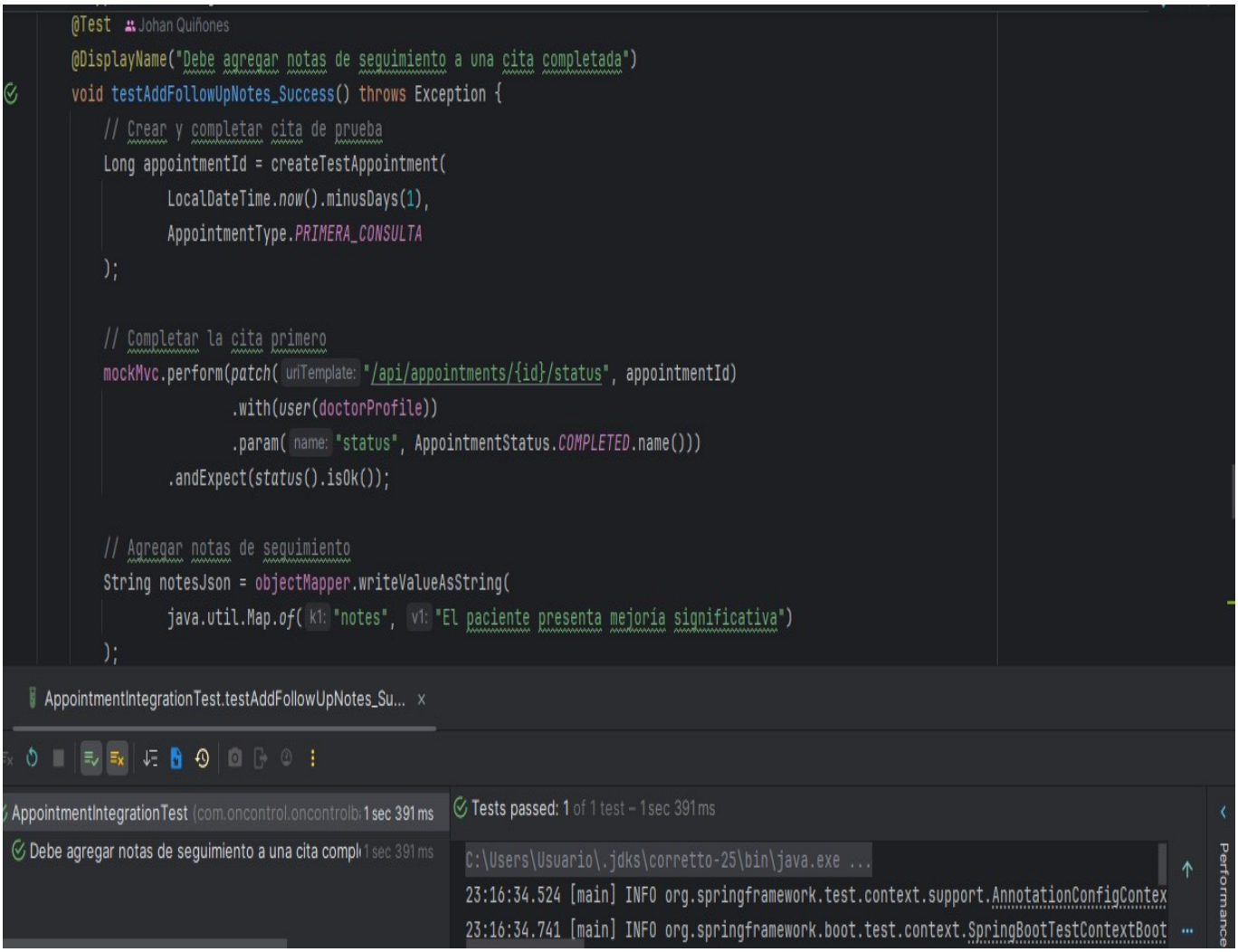

Project
service
AppointmentServiceTest
domain
model
AppointmentTest
infrastructure
AppointmentIntegrationTest
iam
domain
model
UserTest
profiles
domain
model
ProfileTest
AppointmentIntegrationTest.java
application-test.properties
AppointmentTest.java
class AppointmentIntegrationTest {
void testGetDoctorAppointments_Success() throws Exception {
.andExpect(jsonPath(expression: "$.appointments", hasSize(2)))
.andExpect(jsonPath(expression: "$.count").value(expectedValue: 2));
}
@Test Johan Quiñones
@DisplayName("Debe obtener todas las citas de un paciente")
void testGetPatientAppointments_Success() throws Exception {
// Crear citas de prueba
createTestAppointment(LocalDateTime.now().plusDays(1), AppointmentType.PRIMERA_CONSULTA);
createTestAppointment(LocalDateTime.now().plusDays(3), AppointmentType.REVISION_TRATAMIENTO);
// Ejecutar y verificar
mockMvc.perform(get(uriTemplate: "/api/appointments/patient/{patientId}", patientProfile.getId())
.with(user(patientProfile)))
.andDo(print())
.andExpect(status().isOk())
.andExpect(jsonPath(expression: "$.appointments").isArray())
.andExpect(jsonPath(expression: "$.appointments", hasSize(2)))
.andExpect(jsonPath(expression: "$.count").value(expectedValue: 2));
}
Run
AppointmentIntegrationTest.testGetDoctorAppointment...
AppointmentIntegrationTest (com. 1 sec 828 ms
Debe obtener todas las citas de 1 sec 828 ms
1 test passed 1 test total, 1 sec 828 ms
C:\Users\Antho\.jdks\openjdk-25\bin\java.exe ...
22:21:34.850 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not detect default configuration classes for test class [com.oncontrol.on
22:21:35.303 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper -- Found @SpringBootConfiguration com.oncontrol.oncontrolbackend.OncontrolBackendAppli
22:21:35.740 [main] INFO org.springframework.boot.devtools.restart.RestartApplicationListener -- Restart disabled due to context in which it is running

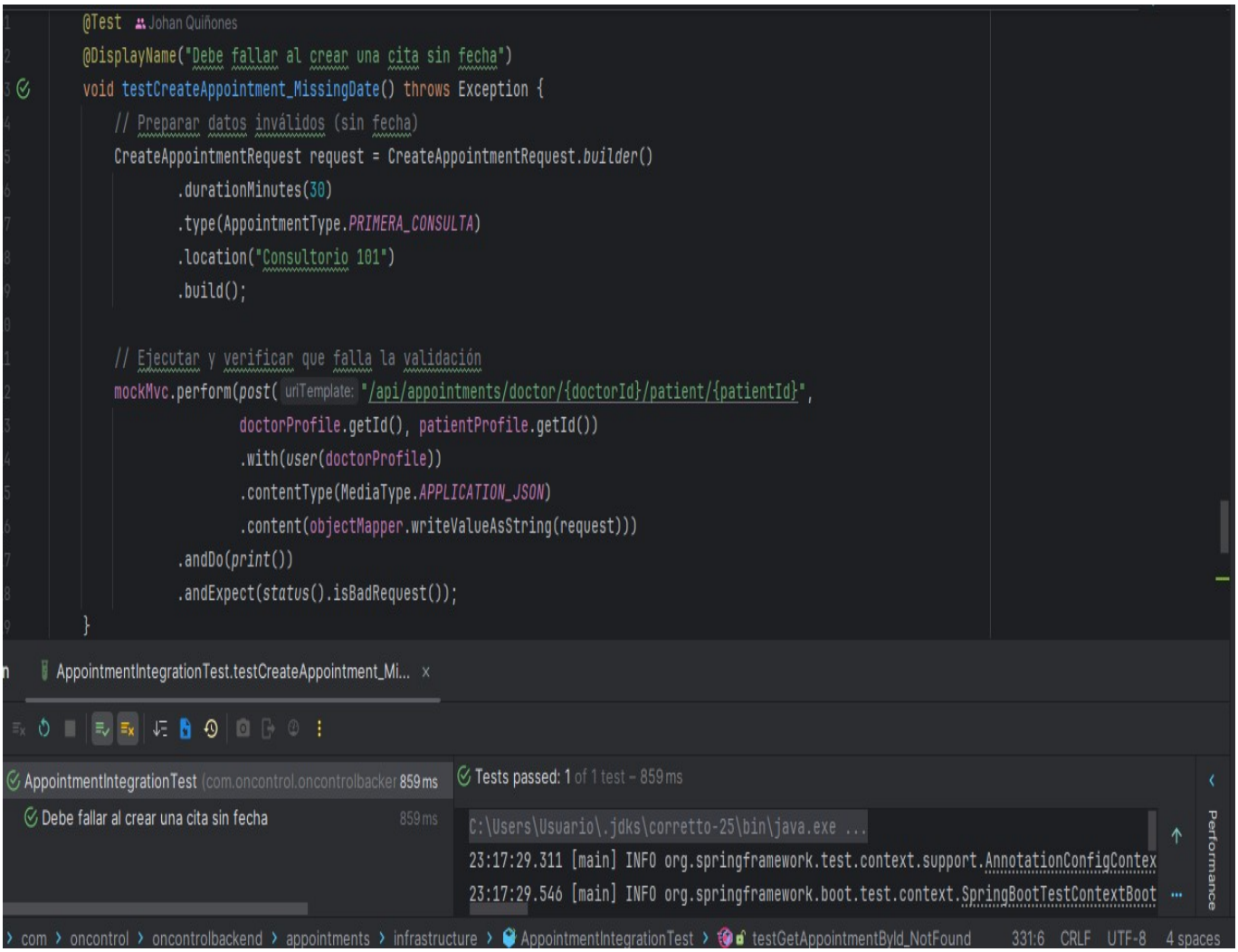

Project
AppointmentIntegrationTest.java
application-test.properties
AppointmentTest.java
service
AppointmentServiceTest
domain
model
AppointmentTest
infrastructure
AppointmentIntegrationTest
iam
domain
model
UserTest
profiles
domain
model
ProfileTest
class AppointmentIntegrationTest {
@Test
@DisplayName("Debe obtener una cita por su ID")
void testGetAppointmentById_Success() throws Exception {
// Crear cita de prueba
Long appointmentId = createTestAppointment(
LocalDateTime.now().plusDays(5),
AppointmentType.CONSULTA_SEGUIMIENTO
);
// Ejecutar y verificar
mockMvc.perform(get(uriTemplate: "/api/appointments/{id}", appointmentId)
.with(user(doctorProfile)))
.andDo(print())
.andExpect(status().isOk())
.andExpect(jsonPath(expression: "$.id").value(appointmentId))
.andExpect(jsonPath(expression: "$.type").value(expectedValue: "CONSULTA_SEGUIMIENTO"))
.andExpect(jsonPath(expression: "$.status").value(expectedValue: "SCHEDULED"));
}
@Test
@DisplayName("Debe actualizar el estado de una cita")
Run
AppointmentIntegrationTest.testGetAppointmentById_Su...
AppointmentIntegrationTest
Debe obtener una cita por su ID
1 sec 468 ms
1 test passed
1 test total, 1 sec 468 ms
C:\Users\Antho\.jdks\openjdk-25\bin\java.exe ...
22:23:00.039 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not detect default configuration classes for test class
22:23:00.371 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper -- Found @SpringBootConfiguration
22:23:00.733 [main] INFO org.springframework.boot.devtools.restart.RestartApplicationListener -- Restart disabled due to context in which it is running
:: Spring Boot ::
(v3.5.6)

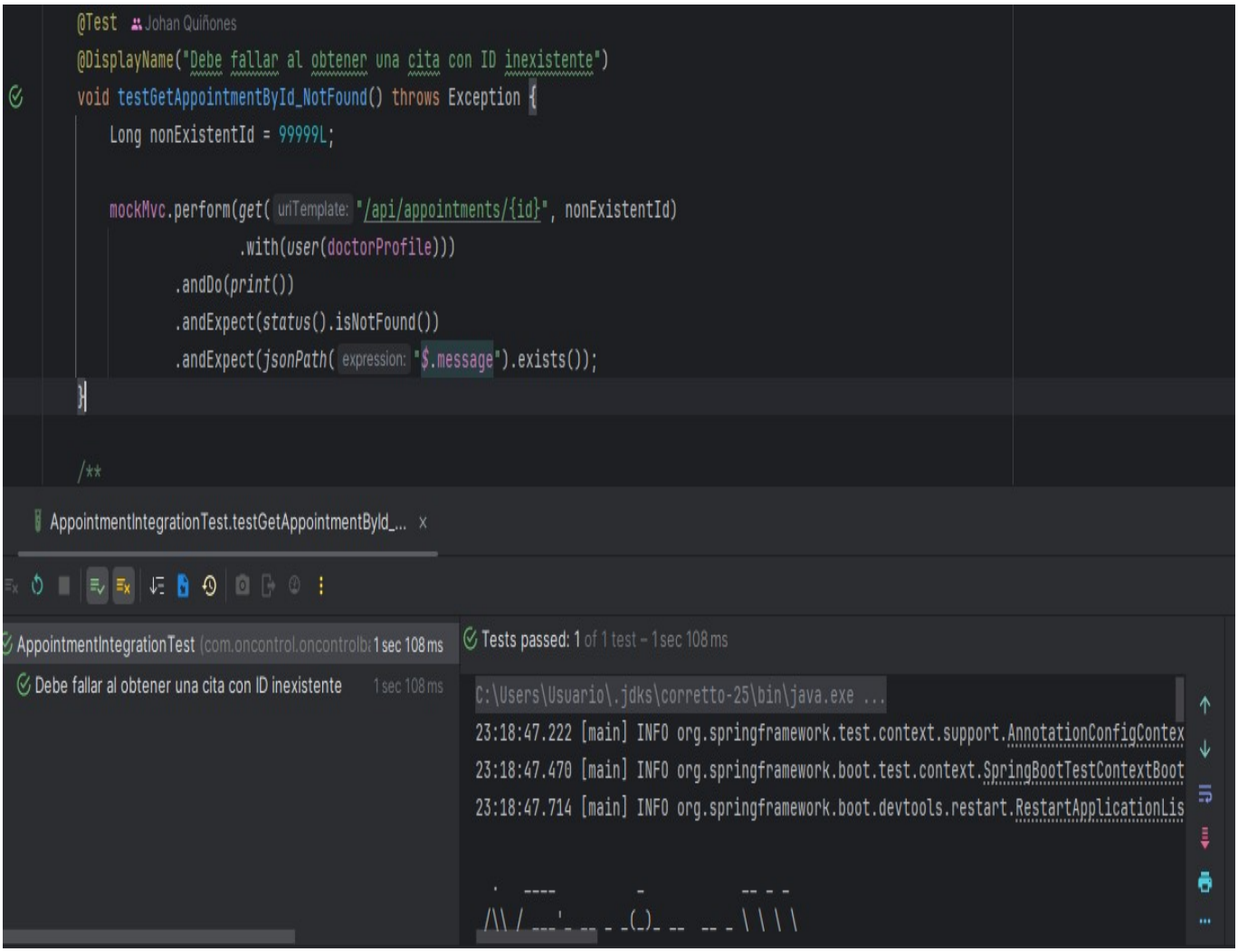

Project
service
AppointmentServiceTest
domain
model
AppointmentTest
infrastructure
AppointmentIntegrationTest
iam
domain
model
UserTest
profiles
domain
model
ProfileTest
AppointmentIntegrationTest.java
application-test.properties
AppointmentTest.java
class AppointmentIntegrationTest {
}
@Test Johan Quiñones
@DisplayName("Debe actualizar el estado de una cita")
void testUpdateAppointmentStatus_Success() throws Exception {
// Crear cita de prueba
Long appointmentId = createTestAppointment(
LocalDateTime.now().plusDays(1),
AppointmentType.PRIMERA_CONSULTA
);
// Ejecutar y verificar
mockMvc.perform(patch(uriTemplate: "/api/appointments/{id}/status", appointmentId)
.with(user(doctorProfile))
.param(name: "status", AppointmentStatus.CONFIRMED.name()))
.andDo(print())
.andExpect(status().isOk())
.andExpect(jsonPath(expression: "$.message").value(expectedValue: "Appointment status updated successfully"))
.andExpect(jsonPath(expression: "$.appointment.id").value(appointmentId))
.andExpect(jsonPath(expression: "$.appointment.status").value(expectedValue: "CONFIRMED"));
}
Run
AppointmentIntegrationTest.testUpdateAppointmentStat...
AppointmentIntegrationTest (com.oncon 2 sec 60 ms
Debe actualizar el estado de una cita 2 sec 60 ms
1 test passed 1 test total, 2 sec 60 ms
C:\Users\Antho\.jdks\openjdk-25\bin\java.exe ...
22:24:20.993 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not detect default configuration classes for test class [com.oncontrol
22:24:21.347 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper -- Found @SpringBootConfiguration com.oncontrol.oncontrolbackend.OncontrolBackendApp
22:24:21.859 [main] INFO org.springframework.boot.devtools.restart.RestartApplicationListener -- Restart disabled due to context in which it is running
:: Spring Boot :: (v3.5.6)


@Test Johan Quiñones
@DisplayName("Debe actualizar el estado de una cita")
void testUpdateAppointmentStatus_Success() throws Exception {
// Crear cita de prueba
Long appointmentId = createTestAppointment(
LocalDateTime.now().plusDays(1),
AppointmentType.PRIMERA_CONSULTA
);
// Ejecutar y verificar
mockMvc.perform(patch(uriTemplate: "/api/appointments/{id}/status", appointmentId)
.with(user(doctorProfile))
.param(name: "status", AppointmentStatus.CONFIRMED.name()))
.andDo(print())
.andExpect(status().isOk())
.andExpect(jsonPath(expression: "$.message").value(expectedValue: "Appointment status updated successfully"))
.andExpect(jsonPath(expression: "$.appointment.id").value(appointmentId))
.andExpect(jsonPath(expression: "$.appointment.status").value(expectedValue: "CONFIRMED"));
}
AppointmentIntegrationTest.testUpdateAppointmentSt...
AppointmentIntegrationTest (com.oncontrol.oncontrolb 1 sec 760 ms
Debe actualizar el estado de una cita 1 sec 760 ms
Tests passed: 1 of 1 test – 1 sec 760 ms
C:\Users\Usuario\.jdks\corretto-25\bin\java.exe ...
23:13:02.856 [main] INFO org.springframework.test.context.support.AnnotationConfigContex
23:13:03.126 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBoot


@Test Johan Quiñones
@DisplayName("Debe cancelar una cita con razón")
void testCancelAppointment_WithReason() throws Exception {
// Crear cita de prueba
Long appointmentId = createTestAppointment(
LocalDateTime.now().plusDays(2),
AppointmentType.PRIMERA_CONSULTA
);
// Ejecutar y verificar
mockMvc.perform(patch(uriTemplate: "/api/appointments/{id}/status", appointmentId)
.with(user(doctorProfile))
.param(name: "status", AppointmentStatus.CANCELLED.name())
.param(name: "reason", ...values: "El paciente solicitó reprogramar"))
.andDo(print())
.andExpect(status().isOk())
.andExpect(jsonPath(expression: "$.message").value(expectedValue: "Appointment status updated successfully"))
.andExpect(jsonPath(expression: "$.appointment.status").value(expectedValue: "CANCELLED"));
}
AppointmentIntegrationTest.testCancelAppointment_Wi...
AppointmentIntegrationTest (com.oncontrol.oncontrolb 1 sec 276 ms
Debe cancelar una cita con razón 1 sec 276 ms
Tests passed: 1 of 1 test – 1 sec 276 ms
C:\Users\Usuario\.jdks\corretto-25\bin\java.exe ...
23:15:23.818 [main] INFO org.springframework.test.context.support.AnnotationConfigContex
23:15:24.041 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBoot


@Test Johan Quiñones
@DisplayName("Debe agregar notas de seguimiento a una cita completada")
void testAddFollowUpNotes_Success() throws Exception {
// Crear y completar cita de prueba
Long appointmentId = createTestAppointment(
LocalDateTime.now().minusDays(1),
AppointmentType.PRIMERA_CONSULTA
);
// Completar la cita primero
mockMvc.perform(patch(uriTemplate: "/api/appointments/{id}/status", appointmentId)
.with(user(doctorProfile))
.param(name: "status", AppointmentStatus.COMPLETED.name()))
.andExpect(status().isOk());
// Agregar notas de seguimiento
String notesJson = objectMapper.writeValueAsString(
java.util.Map.of(k1: "notes", v1: "El paciente presenta mejoría significativa")
);
AppointmentIntegrationTest.testAddFollowUpNotes_Su...
AppointmentIntegrationTest (com.oncontrol.oncontrolb 1 sec 391 ms
Debe agregar notas de seguimiento a una cita compl 1 sec 391 ms
Tests passed: 1 of 1 test – 1 sec 391 ms
C:\Users\Usuario\.jdks\corretto-25\bin\java.exe ...
23:16:34.524 [main] INFO org.springframework.test.context.support.AnnotationConfigContex
23:16:34.741 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBoot
Performance


@Test Johan Quiñones
@DisplayName("Debe fallar al crear una cita sin fecha")
void testCreateAppointment_MissingDate() throws Exception {
// Preparar datos inválidos (sin fecha)
CreateAppointmentRequest request = CreateAppointmentRequest.builder()
.durationMinutes(30)
.type(AppointmentType.PRIMERA_CONSULTA)
.location("Consultorio 101")
.build();
// Ejecutar y verificar que falla la validación
mockMvc.perform(post(uriTemplate: "/api/appointments/doctor/{doctorId}/patient/{patientId}",
doctorProfile.getId(), patientProfile.getId())
.with(user(doctorProfile))
.contentType(MediaType.APPLICATION_JSON)
.content(objectMapper.writeValueAsString(request)))
.andDo(print())
.andExpect(status().isBadRequest());
}
AppointmentIntegrationTest.testCreateAppointment_Mi...
AppointmentIntegrationTest (com.oncontrol.oncontrolbacken 859 ms
Debe fallar al crear una cita sin fecha 859 ms
Tests passed: 1 of 1 test – 859 ms
C:\Users\Usuario\.jdks\corretto-25\bin\java.exe ...
23:17:29.311 [main] INFO org.springframework.test.context.support.AnnotationConfigContex
23:17:29.546 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBoot
Performance
com › oncontrol › oncontrolbackend › appointments › infrastructure › AppointmentIntegrationTest › testGetAppointmentById_NotFound
331:6 CRLF UTF-8 4 spaces


@Test Johan Quiñones
@DisplayName("Debe fallar al obtener una cita con ID inexistente")
void testGetAppointmentById_NotFound() throws Exception {
Long nonExistentId = 99999L;
mockMvc.perform(get(uriTemplate: "/api/appointments/{id}", nonExistentId)
.with(user(doctorProfile)))
.andDo(print())
.andExpect(status().isNotFound())
.andExpect(jsonPath(expression: "$.message").exists());
}
/**
AppointmentIntegrationTest.testGetAppointmentById_...
AppointmentIntegrationTest (com.oncontrol.oncontrolb: 1 sec 108 ms
Debe fallar al obtener una cita con ID inexistente 1 sec 108 ms
Tests passed: 1 of 1 test – 1 sec 108 ms
C:\Users\Usuario\.jdks\corretto-25\bin\java.exe ...
23:18:47.222 [main] INFO org.springframework.test.context.support.AnnotationConfigContex
23:18:47.470 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBoot
23:18:47.714 [main] INFO org.springframework.boot.devtools.restart.RestartApplicationLis