1. AnaliticheController

```
@Test
void getByIdTest() throws Exception {
    Analitiche a = new Analitiche("P1", "Parcheggio A", "op-1");
    ReflectionTestUtils.setField(a, "id", "123");

    when(analiticheService.getById("123")).thenReturn(a);

    mockMvc.perform(get("/api/analitiche/123"))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.id").value("123"))
            .andExpect(jsonPath("$.parcheggioId").value("P1"))
            .andExpect(jsonPath("$.nomeParcheggio").value("Parcheggio A"))
            .andExpect(jsonPath("$.operatoreId").value("op-1"));
}
```

```
@Test
void getByOperatoreIdTest() throws Exception {
    Analitiche a = new Analitiche("P2", "Parcheggio B", "op-2");
    ReflectionTestUtils.setField(a, "id", "A-1");

    when(analiticheService.getByOperatoreId("op-2")).thenReturn(a);

    mockMvc.perform(get("/api/analitiche/operatore/op-2"))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.id").value("A-1"))
            .andExpect(jsonPath("$.parcheggioId").value("P2"))
            .andExpect(jsonPath("$.nomeParcheggio").value("Parcheggio B"))
            .andExpect(jsonPath("$.operatoreId").value("op-2"));
}
```

```
@Test
void creaAnaliticheTest() throws Exception {
    Analitiche saved = new Analitiche("P3", "Parcheggio C", "op-3");
    ReflectionTestUtils.setField(saved, "id", "999");

    when(analiticheService.save(any(AnaliticheRequest.class))).thenReturn(saved);

    mockMvc.perform(post("/api/analitiche")
            .contentType(MediaType.APPLICATION_JSON)
            .content("""
                {
                  "parcheggioId": "P3",
                  "nomeParcheggio": "Parcheggio C",
                  "operatoreId": "op-3"
                }
            """))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.id").value("999"))
            .andExpect(jsonPath("$.parcheggioId").value("P3"))
            .andExpect(jsonPath("$.nomeParcheggio").value("Parcheggio C"))
            .andExpect(jsonPath("$.operatoreId").value("op-3"))
            .andExpect(jsonPath("$.log").isArray())
            .andExpect(jsonPath("$.log").isEmpty());
}
```

```java
@Test
void getLogByAnaliticaIdAndTipoTest() throws Exception {
    Log l = new Log();
    ReflectionTestUtils.setField(l, "id", "log2");
    ReflectionTestUtils.setField(l, "analiticaId", "123");
    l.setTipo(LogCategoria.ALLARME);
    l.setSeverita(LogSeverità.CRITICO);
    l.setTitolo("Allarme");
    l.setDescrizione("Dettagli");
    l.setData(LocalDateTime.of(2025, 2, 2, 10, 30));

    when(logService.getLogByAnaliticaIdAndTipo("123", "ALLARME")).thenReturn(List.of(l));

    mockMvc.perform(get("/api/analitiche/123/log/ALLARME"))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$[0].id").value("log2"))
            .andExpect(jsonPath("$[0].tipo").value("ALLARME"))
            .andExpect(jsonPath("$[0].severita").value("CRITICO"))
            .andExpect(jsonPath("$[0].data").value("2025-02-02T10:30:00"));

    verify(logService).getLogByAnaliticaIdAndTipo("123", "ALLARME");
}
```

2. AnaliticheServiceImpl

```java
@Test
void getByIdTest() {
    // given
    Analitiche a = new Analitiche("P1", "Parcheggio A", "op-1");
    when(repository.findById("123")).thenReturn(Optional.of(a));

    // when
    Analitiche result = service.getById("123");

    // then
    assertSame(a, result);
    verify(repository).findById("123");
    verifyNoMoreInteractions(repository);
}
```

```java
@Test
void getByIdEccezioneTest() {
    // given
    when(repository.findById("404")).thenReturn(Optional.empty());

    // when + then
    RuntimeException ex = assertThrows(RuntimeException.class, () -> service.getById("404"));
    assertEquals("Analitiche non trovata", ex.getMessage());

    verify(repository).findById("404");
    verifyNoMoreInteractions(repository);
}
```

```java
@Test
void getByOperatoreIdTest() {
    // given
    Analitiche a = new Analitiche("P2", "Parcheggio B", "op-2");
    when(repository.findByOperatoreId("op-2")).thenReturn(Optional.of(a));

    // when
    Analitiche result = service.getByOperatoreId("op-2");

    // then
    assertSame(a, result);
    verify(repository).findByOperatoreId("op-2");
    verifyNoMoreInteractions(repository);
}
```

```java
@Test
void getByOperatoreIdEccezioneTest() {
    // given
    when(repository.findByOperatoreId("op-x")).thenReturn(Optional.empty());

    // when + then
    RuntimeException ex = assertThrows(RuntimeException.class, () -> service.getByOperatoreId("op-x"));
    assertEquals("Analitiche non trovata", ex.getMessage());

    verify(repository).findByOperatoreId("op-x");
    verifyNoMoreInteractions(repository);
}
```

```java
@Test
void saveTest() {
    // given
    AnaliticheRequest req = new AnaliticheRequest("P3", "Parcheggio C", "op-3");

    // catturo l'entity passata a repository.save(...)
    ArgumentCaptor<Analitiche> captor = ArgumentCaptor.forClass(Analitiche.class);

    // faccio ritornare al repository lo stesso oggetto ricevuto
    when(repository.save(any(Analitiche.class))).thenAnswer(inv -> inv.getArgument(0));

    // when
    Analitiche saved = service.save(req);

    // then: verifica che abbia chiamato save e che il mapping sia corretto
    verify(repository).save(captor.capture());
    Analitiche entityPassed = captor.getValue();

    assertEquals("P3", entityPassed.getParcheggioId());
    assertEquals("Parcheggio C", entityPassed.getNomeParcheggio());
    assertEquals("op-3", entityPassed.getOperatoreId());

    assertEquals("P3", saved.getParcheggioId());
    assertEquals("Parcheggio C", saved.getNomeParcheggio());
    assertEquals("op-3", saved.getOperatoreId());

    verifyNoMoreInteractions(repository);
}
```

3. LogController

```java
@Test
void creaLogTest() throws Exception {
    Log saved = new Log();
    ReflectionTestUtils.setField(saved, "id", "log1");
    ReflectionTestUtils.setField(saved, "analiticaId", "a1");
    saved.setTipo(LogCategoria.EVENTO);
    saved.setSeverita(LogSeverità.INFO);
    saved.setTitolo("Titolo");
    saved.setDescrizione("Descrizione");
    saved.setData(LocalDateTime.of(2025, 1, 1, 12, 0));

    when(service.salvaLog(any(LogRequest.class))).thenReturn(saved);

    mockMvc.perform(post("/api/log")
            .contentType(MediaType.APPLICATION_JSON)
            .content("""
                {
                  "analiticaId": "a1",
                  "tipo": "EVENTO",
                  "severita": "INFO",
                  "titolo": "Titolo",
                  "descrizione": "Descrizione",
                  "data": "2025-01-01T12:00:00"
                }
            """))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.id").value("log1"))
            .andExpect(jsonPath("$.tipo").value("EVENTO"))
            .andExpect(jsonPath("$.titolo").value("Titolo"))
            .andExpect(jsonPath("$.descrizione").value("Descrizione"))
            .andExpect(jsonPath("$.data").value("2025-01-01T12:00:00"))
            .andExpect(jsonPath("$.severita").value("INFO"));

    verify(service).salvaLog(any(LogRequest.class));
}
```

```java
@Test
void getLogByAnaliticaIdTest() throws Exception {
    Log l = new Log();
    ReflectionTestUtils.setField(l, "id", "log1");
    ReflectionTestUtils.setField(l, "analiticaId", "a1");
    l.setTipo(LogCategoria.ALLARME);
    l.setSeverita(LogSeverità.CRITICO);
    l.setTitolo("Allarme");
    l.setDescrizione("Dettagli");
    l.setData(LocalDateTime.of(2025, 2, 2, 10, 30));

    when(service.getLogByAnaliticaId("a1")).thenReturn(List.of(l));

    mockMvc.perform(get("/api/log/analitiche/a1/log"))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$[0].id").value("log1"))
            .andExpect(jsonPath("$[0].tipo").value("ALLARME"))
            .andExpect(jsonPath("$[0].titolo").value("Allarme"))
            .andExpect(jsonPath("$[0].descrizione").value("Dettagli"))
            .andExpect(jsonPath("$[0].data").value("2025-02-02T10:30:00"))
            .andExpect(jsonPath("$[0].severita").value("CRITICO"));

    verify(service).getLogByAnaliticaId("a1");
}
```

```java
@Test
void getLogByAnaliticaIdAndTipoTest() throws Exception {
    Log l = new Log();
    ReflectionTestUtils.setField(l, "id", "log2");
    ReflectionTestUtils.setField(l, "analiticaId", "a1");
    l.setTipo(LogCategoria.EVENTO);
    l.setSeverita(LogSeverità.INFO);
    l.setTitolo("Evento");
    l.setDescrizione("Desc");
    l.setData(LocalDateTime.of(2025, 3, 3, 9, 0));

    when(service.getLogByAnaliticaIdAndTipo("a1", "EVENTO")).thenReturn(List.of(l));

    mockMvc.perform(get("/api/log/analitiche/a1/tipo/EVENTO"))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$[0].id").value("log2"))
            .andExpect(jsonPath("$[0].tipo").value("EVENTO"))
            .andExpect(jsonPath("$[0].severita").value("INFO"));

    verify(service).getLogByAnaliticaIdAndTipo("a1", "EVENTO");
}
```

```java
@Test
void aggiornaSeverityTest() throws Exception {
    Log l = new Log();
    ReflectionTestUtils.setField(l, "id", "log9");
    l.setTipo(LogCategoria.HISTORY);
    l.setSeverita(LogSeverità.ATTENZIONE);
    l.setTitolo("T");
    l.setDescrizione("D");
    l.setData(LocalDateTime.of(2025, 4, 4, 8, 0));

    Log savedAfter = new Log();
    ReflectionTestUtils.setField(savedAfter, "id", "log9");
    savedAfter.setTipo(LogCategoria.HISTORY);
    savedAfter.setSeverita(LogSeverità.CRITICO); // aggiornato
    savedAfter.setTitolo("T");
    savedAfter.setDescrizione("D");
    savedAfter.setData(LocalDateTime.of(2025, 4, 4, 8, 0));

    when(service.getLogById("log9")).thenReturn(Optional.of(l));
    when(service.salvaLog1(any(Log.class))).thenReturn(savedAfter);

    mockMvc.perform(put("/api/log/log9/severity")
            .param("severity", "CRITICO"))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.id").value("log9"))
            .andExpect(jsonPath("$.severita").value("CRITICO"));

    verify(service).getLogById("log9");
    verify(service).salvaLog1(any(Log.class));
}
```

```java
@Test
void aggiornaCategoryTest() throws Exception {
    Log l = new Log();
    ReflectionTestUtils.setField(l, "id", "log7");
    l.setTipo(LogCategoria.EVENTO);
    l.setSeverita(LogSeverità.INFO);
    l.setTitolo("T");
    l.setDescrizione("D");
    l.setData(LocalDateTime.of(2025, 5, 5, 7, 0));

    Log savedAfter = new Log();
    ReflectionTestUtils.setField(savedAfter, "id", "log7");
    savedAfter.setTipo(LogCategoria.ALLARME);
    savedAfter.setSeverita(LogSeverità.INFO);
    savedAfter.setTitolo("T");
    savedAfter.setDescrizione("D");
    savedAfter.setData(LocalDateTime.of(2025, 5, 5, 7, 0));

    when(service.getLogById("log7")).thenReturn(Optional.of(l));
    when(service.salvaLog1(any(Log.class))).thenReturn(savedAfter);

    mockMvc.perform(put("/api/log/log7/category")
            .param("category", "ALLARME"))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.id").value("log7"))
            .andExpect(jsonPath("$.tipo").value("ALLARME"));

    verify(service).getLogById("log7");
    verify(service).salvaLog1(any(Log.class));
}
```

4. LogServiceImpl

```java
@Test
void getLogByIdTest() {
    Log log = new Log();
    ReflectionTestUtils.setField(log, "id", "log1");
    when(repository.findById("log1")).thenReturn(Optional.of(log));

    Optional<Log> result = service.getLogById("log1");

    assertTrue(result.isPresent());
    assertEquals("log1", result.get().getId());
    verify(repository).findById("log1");
    verifyNoMoreInteractions(repository);
}
```

```java
@Test
void getLogByIdVuotoTest() {
    when(repository.findById("missing")).thenReturn(Optional.empty());

    Optional<Log> result = service.getLogById("missing");

    assertTrue(result.isEmpty());
    verify(repository).findById("missing");
    verifyNoMoreInteractions(repository);
}
```

```java
@Test
void salvaLog1Test() {
    Log log = new Log();
    log.setTitolo("Titolo");
    log.setDescrizione("Descrizione");
    log.setSeverita(LogSeverità.INFO);
    log.setTipo(LogCategoria.EVENTO);

    when(repository.save(any(Log.class))).thenAnswer(inv -> inv.getArgument(0));

    Log saved = service.salvaLog1(log);

    // data impostata "ora"
    assertNotNull(saved.getData());
    assertEquals("Titolo", saved.getTitolo());
    assertEquals("Descrizione", saved.getDescrizione());
    assertEquals(LogSeverità.INFO, saved.getSeverita());
    assertEquals(LogCategoria.EVENTO, saved.getTipo());

    verify(repository).save(any(Log.class));
    verifyNoMoreInteractions(repository);
}
```

```java
@Test
void salvaLogTest() {
    LocalDateTime time = LocalDateTime.of(2025, 1, 1, 12, 0);
    LogRequest req = new LogRequest("a1", LogCategoria.ALLARME, LogSeverità.CRITICO, "T", "D", time);

    ArgumentCaptor<Log> captor = ArgumentCaptor.forClass(Log.class);

    when(repository.save(any(Log.class))).thenAnswer(inv -> inv.getArgument(0));

    Log saved = service.salvaLog(req);

    verify(repository).save(captor.capture());
    Log passed = captor.getValue();

    assertEquals(LogCategoria.ALLARME, passed.getTipo());
    assertEquals(LogSeverità.CRITICO, passed.getSeverita());
    assertEquals("T", passed.getTitolo());
    assertEquals("D", passed.getDescrizione());
    assertEquals(time, passed.getData());

    assertEquals(passed.getTitolo(), saved.getTitolo());
    verifyNoMoreInteractions(repository);
}
```

```java
@Test
void getLogByAnaliticaIdTest() {
    Log l1 = new Log();
    ReflectionTestUtils.setField(l1, "id", "L1");
    Log l2 = new Log();
    ReflectionTestUtils.setField(l2, "id", "L2");

    when(repository.findByAnaliticaIdOrderByDataDesc("A1")).thenReturn(List.of(l1, l2));

    List<Log> result = service.getLogByAnaliticaId("A1");

    assertEquals(2, result.size());
    assertEquals("L1", result.get(0).getId());
    verify(repository).findByAnaliticaIdOrderByDataDesc("A1");
    verifyNoMoreInteractions(repository);
}
```

```java
@Test
void getLogByAnaliticaIdAndTipoTest() {
    Log l = new Log();
    ReflectionTestUtils.setField(l, "id", "L3");

    when(repository.findByAnaliticaIdAndTipo("A2", "EVENTO")).thenReturn(List.of(l));

    List<Log> result = service.getLogByAnaliticaIdAndTipo("A2", "EVENTO");

    assertEquals(1, result.size());
    assertEquals("L3", result.get(0).getId());
    verify(repository).findByAnaliticaIdAndTipo("A2", "EVENTO");
    verifyNoMoreInteractions(repository);
}
```

5. ParcheggioController

```java
@Test
void cercaTest() throws Exception {
    ParcheggioResponse p1 = new ParcheggioResponse("1", "Parcheggio A", "Centro", 100, 20, 45.5, 9.1);
    ParcheggioResponse p2 = new ParcheggioResponse("2", "Parcheggio B", "Centro", 50, 10, 45.51, 9.11);

    when(parcheggioService.cercaPerArea("Centro")).thenReturn(List.of(p1, p2));

    mockMvc.perform(get("/api/parcheggi/cerca").param("area", "Centro"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].id").value("1"))
            .andExpect(jsonPath("$[1].nome").value("Parcheggio B"));

    verify(parcheggioService).cercaPerArea("Centro");
}
```

```java
@Test
void prenotaTest() throws Exception {
    LocalDateTime orario = LocalDateTime.of(2025, 1, 1, 10, 0);

    PrenotazioneResponse resp = new PrenotazioneResponse("1", "utente1", "parcheggio1", orario, "QR123");

    when(parcheggioService.effettuaPrenotazione(any(PrenotazioneRequest.class))).thenReturn(resp);

    mockMvc.perform(post("/api/parcheggi/prenota")
            .contentType(MediaType.APPLICATION_JSON)
            .content("""
                {
                  "utenteId": "utente1",
                  "parcheggioId": "parcheggio1",
                  "orario": "2025-01-01T10:00:00"
                }
            """))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.id").value("1"))
            .andExpect(jsonPath("$.codiceQr").value("QR123"))
            .andExpect(jsonPath("$.orario").value("2025-01-01T10:00:00"));

    verify(parcheggioService).effettuaPrenotazione(any(PrenotazioneRequest.class));
}
```

```java
@Test
void getNearbyTest() throws Exception {
    ParcheggioResponse vicino = new ParcheggioResponse("3", "Vicino", "Nord", 10, 2, 45.50, 9.20);
    when(parcheggioService.cercaVicini(45.50, 9.20, 500.0)).thenReturn(List.of(vicino));

    mockMvc.perform(get("/api/parcheggi/nearby")
            .param("lat", "45.50")
            .param("lng", "9.20")
            .param("radius", "500"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].nome").value("Vicino"))
            .andExpect(jsonPath("$[0].area").value("Nord"));

    verify(parcheggioService).cercaVicini(45.50, 9.20, 500.0);
}
```

6. ParcheggioServiceImpl

```java
@Test
void cercaPerAreaTest() {
    Parcheggio p1 = new Parcheggio("A", "Centro", 100, 50, 45.5, 9.1);
    p1.setId("1");
    Parcheggio p2 = new Parcheggio("B", "Centro", 80, 20, 45.6, 9.2);
    p2.setId("2");

    when(parcheggioRepository.findByAreaContainingIgnoreCase("Centro")).thenReturn(List.of(p1, p2));

    List<ParcheggioResponse> result = service.cercaPerArea("Centro");

    assertEquals(2, result.size());
    assertEquals("A", result.get(0).nome());
    verify(parcheggioRepository).findByAreaContainingIgnoreCase("Centro");
}
```

```java
@Test
void effettuaPrenotazioneTest() {
    LocalDateTime orario = LocalDateTime.of(2025, 1, 1, 10, 0);

    Parcheggio parcheggio = new Parcheggio("A", "Centro", 100, 10, 45.5, 9.1);
    parcheggio.setId("p1");
    when(parcheggioRepository.findById("p1")).thenReturn(Optional.of(parcheggio));
    when(parcheggioRepository.save(any(Parcheggio.class))).thenAnswer(inv -> inv.getArgument(0));

    Prenotazione prenotazione = new Prenotazione("u1", "p1", orario, UUID.randomUUID().toString());
    prenotazione.setId("pr1");
    when(prenotazioneRepository.save(any(Prenotazione.class))).thenReturn(prenotazione);

    PrenotazioneRequest req = new PrenotazioneRequest("u1", "p1", orario);
    PrenotazioneResponse resp = service.effettuaPrenotazione(req);

    assertEquals("pr1", resp.id());
    assertEquals("u1", resp.utenteId());
    assertEquals("p1", resp.parcheggioId());
    assertEquals(orario, resp.orario());

    verify(parcheggioRepository).findById("p1");
    verify(parcheggioRepository).save(any(Parcheggio.class));
    verify(prenotazioneRepository).save(any(Prenotazione.class));
}
```

```java
@Test
void effettuaPrenotazionePostiEsauritiTest() {
    LocalDateTime orario = LocalDateTime.of(2025, 1, 1, 11, 0);

    Parcheggio parcheggio = new Parcheggio("A", "Centro", 100, 0, 45.5, 9.1);
    parcheggio.setId("p2");
    when(parcheggioRepository.findById("p2")).thenReturn(Optional.of(parcheggio));

    PrenotazioneRequest req = new PrenotazioneRequest("u1", "p2", orario);

    assertThrows(IllegalStateException.class, () -> service.effettuaPrenotazione(req));
    verify(parcheggioRepository).findById("p2");
    verify(parcheggioRepository, never()).save(any());
    verify(prenotazioneRepository, never()).save(any());
}
```

```java
@Test
void effettuaPrenotazioneParcheggioNonTrovatoTest() {
    LocalDateTime orario = LocalDateTime.of(2025, 1, 1, 12, 0);

    when(parcheggioRepository.findById("missing")).thenReturn(Optional.empty());

    PrenotazioneRequest req = new PrenotazioneRequest("u1", "missing", orario);

    assertThrows(IllegalArgumentException.class, () -> service.effettuaPrenotazione(req));
    verify(parcheggioRepository).findById("missing");
    verify(parcheggioRepository, never()).save(any());
    verify(prenotazioneRepository, never()).save(any());
}
```

```java
@Test
void cercaViciniTest() {
    Parcheggio vicino = new Parcheggio("Vicino", "Centro", 50, 10, 45.50, 9.20);
    Parcheggio lontano = new Parcheggio("Lontano", "Centro", 50, 10, 46.00, 10.00);
    when(parcheggioRepository.findAll()).thenReturn(List.of(vicino, lontano));

    List<ParcheggioResponse> result = service.cercaVicini(45.50, 9.20, 500.0);

    assertEquals(1, result.size());
    assertEquals("Vicino", result.get(0).nome());
    verify(parcheggioRepository).findAll();
}
```

7. PrenotazioneController

```
@Test
void getStoricoTest() throws Exception {
    LocalDateTime orario = LocalDateTime.of(2025, 1, 1, 10, 0);

    PrenotazioneResponse r1 = new PrenotazioneResponse("p1", "u1", "park1", orario, "QR1");
    PrenotazioneResponse r2 = new PrenotazioneResponse("p2", "u1", "park2", orario.plusHours(1), "QR2");

    when(prenotazioneService.getStoricoUtente("u1")).thenReturn(List.of(r1, r2));

    mockMvc.perform(get("/api/prenotazioni/utente/u1"))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$[0].id").value("p1"))
            .andExpect(jsonPath("$[0].utenteId").value("u1"))
            .andExpect(jsonPath("$[0].parcheggioId").value("park1"))
            .andExpect(jsonPath("$[0].codiceQr").value("QR1"))
            .andExpect(jsonPath("$[1].id").value("p2"))
            .andExpect(jsonPath("$[1].codiceQr").value("QR2"));

    verify(prenotazioneService).getStoricoUtente("u1");
}
```

8. PrenotazioneServiceImpl

```
@Test
void getStoricoUtenteTest() {
    LocalDateTime orario1 = LocalDateTime.of(2025, 1, 1, 10, 0);
    LocalDateTime orario2 = LocalDateTime.of(2025, 1, 1, 11, 0);

    Prenotazione p1 = new Prenotazione("u1", "park1", orario1, "QR1");
    p1.setId("p1");

    Prenotazione p2 = new Prenotazione("u1", "park2", orario2, "QR2");
    p2.setId("p2");

    when(prenotazioneRepository.findByUtenteId("u1")).thenReturn(List.of(p1, p2));

    List<PrenotazioneResponse> result = service.getStoricoUtente("u1");

    assertEquals(2, result.size());

    assertEquals("p1", result.get(0).id());
    assertEquals("u1", result.get(0).utenteId());
    assertEquals("park1", result.get(0).parcheggioId());
    assertEquals(orario1, result.get(0).orario());
    assertEquals("QR1", result.get(0).codiceQr());

    assertEquals("p2", result.get(1).id());
    assertEquals("u1", result.get(1).utenteId());
    assertEquals("park2", result.get(1).parcheggioId());
    assertEquals(orario2, result.get(1).orario());
    assertEquals("QR2", result.get(1).codiceQr());

    verify(prenotazioneRepository).findByUtenteId("u1");
    verifyNoMoreInteractions(prenotazioneRepository);
}
```