Raport despre folosirea unui tool de AI

Introducere:

In cadrul proiectului am generat cateva teste cu ajutorul ChatGPT pentru a evalua cat de utile si corecte pot fi in comparative cu cele scrise manual. Am folosit AI-ul ca suport pentru a scrie 2 teste unitare si 2 teste de integrare ,iar in acest raport prezentam prompturile folosite ,codul generat si rezultatul testelor.

Prompt-uri si rezultatele obtinute:

Scrie un test unit folosind JUnit si Mockito pentru metoda createReview din ReviewService, care primeste un ReviewRequestDTO valid. Metoda ar trebui sa salveze review-ul doar daca utilizatorii si cursa exista si daca nu a mai fost trimis un review de catre acelasi utilizator pentru aceeasi cursa. Returneaza un ReviewResponseDTO. Foloseste when() si verify() acolo unde este necesar.

java Copy Edit

```java
@Test
void testCreateReview_validReview() {

    when(userRepository.existsById(validReviewRequest.getReviewerId())).thenReturn(true);
    when(userRepository.existsById(validReviewRequest.getReviewedId())).thenReturn(true);
    when(rideRepository.findById(validReviewRequest.getRideId())).thenReturn(Optional.of(ride));
    when(rideBookingRepository.findByRideIdAndPassengerId(validReviewRequest.getRideId(), validRe
            .thenReturn(Optional.of(rideBooking));
    when(reviewRepository.findByRideIdAndReviewerId(validReviewRequest.getRideId(), validReviewRe
            .thenReturn(Optional.empty());
```

```
    ReviewResponseDTO response = reviewService.createReview(validReviewRequest);


    assertNotNull(response);
    assertEquals(validReviewRequest.getReviewerId(), response.getReviewerId());
    assertEquals(validReviewRequest.getReviewedId(), response.getReviewedId());
    assertEquals(validReviewRequest.getRideId(), response.getRideId());
    assertEquals(validReviewRequest.getRating(), response.getRating());
    assertEquals(validReviewRequest.getComment(), response.getComment());
    assertNotNull(response.getCreatedAt());


    verify(userRepository, times(1)).existsById(validReviewRequest.getReviewerId());
    verify(userRepository, times(1)).existsById(validReviewRequest.getReviewedId());
    verify(rideRepository, times(1)).findById(validReviewRequest.getRideId());
    verify(rideBookingRepository, times(1)).findByRideIdAndPassengerId(validReviewRequest.getRide
    verify(reviewRepository, times(1)).findByRideIdAndReviewerId(validReviewRequest.getRideId(),
    verify(userRepository, times(1)).findById(validReviewRequest.getReviewedId());
    verify(reviewRepository, times(1)).save(any(Review.class));
}
```

Scrie un test de integrare cu Spring MockMvc pentru endpointul GET /reviews/by-driver/{driverId} care ar trebui sa returneze o lista de recenzii in format JSON. Testul trebuie sa verifice ca raspunsul HTTP este 200 OK, content-type-ul este application/json si ca rezultatul contine doua recenzii cu reviewerId diferit.

java

```java
@Test
public void testGetReviewsByDriver() throws Exception {
    mockMvc.perform(get("/reviews/by-driver/driver1"))
            .andExpect(status().isOk())
            .andExpect(content().contentType(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.length()").value(2))
            .andExpect(jsonPath("$[0].reviewerId").value("user1"))
            .andExpect(jsonPath("$[1].reviewerId").value("user2"));
}
```

Vreau un test de integrare cu Spring MockMvc pentru un endpoint POST /bookings, care primeste un RideBookingRequestDTO ce contine rideId, passengerId si createdAt. Daca totul e corect, trebuie sa returneze 201 Created si body-ul sa fie gol. Codifica payload-ul cu objectMapper si trimite-l ca JSON.

java Copy Edit

```java
@Test
public void testCreateRideBooking() throws Exception {
    Ride ride = rideRepository.findAll().get(0);
    User freeUser = userRepository.findAll().get(2);  // Get the free user)

    Instant createdAt = Instant.parse("2025-03-31T15:52:00Z");
    RideBookingRequestDTO rideBookingRequestDTO = new RideBookingRequestDTO();
    rideBookingRequestDTO.setRideId(ride.getId());
    rideBookingRequestDTO.setPassengerId(freeUser.getId());
    rideBookingRequestDTO.setCreatedAt(createdAt);
```

```java
    Instant createdAt = Instant.parse("2025-03-31T15:52:00Z");
    RideBookingRequestDTO rideBookingRequestDTO = new RideBookingRequestDTO();
    rideBookingRequestDTO.setRideId(ride.getId());
    rideBookingRequestDTO.setPassengerId(freeUser.getId());
    rideBookingRequestDTO.setCreatedAt(createdAt);

    mockMvc.perform(post("/bookings")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(rideBookingRequestDTO)))
        .andExpect(status().isCreated())
        .andExpect(content().string(""));
}
```

Scrie un test unit JUnit pentru metoda createRide din RideService, care arunca o exceptie RideConflictException daca soferul are deja o alta cursa ce se suprapune ca orar. Simuleaza existenta unei alte curse folosind rideRepository.findByDriverIdAndTimeOverlap() care returneaza o lista nevida. Verifica folosind assertThrows.

java Copy Edit

```java
@Test
void testCreateRide_DriverOverlapAsDriver() {

    RideRequestDTO request = createValidRideRequest();

    when(userRepository.existsById(request.getDriverId())).thenReturn(true);

    when(vehicleRepository.existsByLicensePlate(request.getCarLicensePlate())).thenReturn(true);
```

```java
    when(vehicleRepository.existsByLicensePlate(request.getCarLicensePlate())).thenReturn(true);

    when(rideRepository.findByDriverIdAndTimeOverlap(
        request.getDriverId(),
        request.getDepartureTime(),
        request.getArrivalTime()
    )).thenReturn(List.of(new Ride()));

    assertThrows(RideConflictException.class, () -> {
        rideService.createRide(request);
    });
}
```

Testele generate de AI:

Nume test	Functionalitate testata	Tip test	Status
testCreateReview_validReview()	Creare review valid	Unitar (JUnit + Mockito)	Executat cu succes, acoperire completa pentru caz pozitiv
testGetReviewsByDriver()	Obtinere review-uri prin endpoint REST	Integrare (Spring MockMvc)	Rulat corect, returneaza datele asteptate in format JSON
testCreateRideBooking()	Booking ride prin REST API	Integrare (MockMvc)	Endpoint funcțional, testul valideaza raspunsul HTTP correct
testCreateRide_DriverOverlapAsDriver()	Suprapunere orar curse	Unitar	Simuleaza corect scenariul de exceptie

Comparatie intre testele generate de AI si testele proprii:

Testele generate cu ajutorul AI-ului s-au integrat foarte bine in codul proiectului. Din punct de vedere al structurii, logicii si stilului de scriere, ele au fost foarte apropiate de testele pe care le-as fi scris manual. AI-ul a reusit sa inteleaga corect arhitectura aplicatiei si a folosit fara probleme repository-urile, serviciile si modelele de date, respectand conventiile deja existente (precum Mockito, MockMvc, assertEquals, verify etc.).

Nu a fost nevoie de modificari majore – codul generat a fost curat, logic si usor de adaptat. Practic, diferentele dintre testele scrise manual si cele generate au fost minime sau chiar inexistente in unele cazuri. Ele se comporta la fel, sunt organizate la fel si respecta aceleasi principii.

Per total, experienta a fost una pozitiva. AI-ul a oferit un sprijin real, iar codul propus s-a „integrat” atat de bine incat, pentru cineva din afara, ar fi greu de spus care test a fost scris de o persoana si care de un AI.

Concluzie:

Pe parcursul testarii, au fost utilizate si unele solutii generate automat, care s-au integrat natural in cadrul existent al proiectului. Acestea au oferit un plus de flexibilitate si au completat in mod echilibrat abordarea generala de testare, fara a modifica structura sau stilul aplicatiei.