Proiect Testarea Sistemelor Software

Proiectul a fost implementat sa simuleze comportamentul unei aplicatii de ridesharing, pe scurt sistemul ofera userilor functionalitati cum ar fi: creearea unor noi curse, rezervarea unui loc in masina, un sistem de recenzii pentru soferi si posibilitatea de a actualiza si a vedea in timp real starea curenta a unei curse.

Arhitectura aplicatiei este bazata pe Spring Boot, cu o structura tipica pe layere (controller, service, repository) si cu stocarea datelor in MongoDB.

Obiectivul acestui proiect este testarea sistematica a unei aplicatii de ridesharing, implementate cu Spring Boot. Sistemul propus utilizează testarea unitară și de integrare cu ajutorul JUnit, Mockito și Testcontainers, pentru a simula componente externe într-un mediu controlat. Validarea codului existent si testarea completa a aplicatiei reprezinta scopul final al acestui proiect ce va include:

- testarea unitara a logicii de business (ex: validarea recenziilor, reguli de creare cursa),
- testarea de integrare pentru fluxuri reale (ex: endpoint-uri REST, interactiunea cu baza de date MongoDB),
- simularea componentelor externe intr-un mediu controlat (cu ajutorul mock-urilor sau containerelor de test).

Testarea devine esentiala in contextual gestionarii fluxurilor complexe de date din mai multe motive:

- Siguranta datelor – sistemul trebuie sa asigure ca doar utilizatorii validati pot interactiona (ex: doar pasagerii pot lasa recenzii).
- Corectitudinea logicii aplicatiei – rezervarile, recenziile si statusul curselor trebuie sa respecte reguli clare.
- Prevenirea regresiilor – modificarile ulterioare in cod nu trebuie sa afecteze functionalitatile existente.
- Testabilitate in mediu real – folosirea MongoDB intr-un container real permite testarea intr-un mediu apropiat de productie.

Definitii esentiale:

- Testare unitara: Verificarea functionalitatii unei unitati individuale de cod
- Testare de integrare: Testarea modului in care mai multe componente colaboreaza
- Mocking: Tehnica de simulare a unor componente externe pentru a testa doar o bucata de logica
- Coverage: Procentul de cod care este atins de teste
- CI/CD: Practica de integrare continua, care ruleaza testele automat la fiecare commit[1]

Configuratia Hardware:

- Procesor (CPU): AMD Ryzen 7 8845HS
- Placă video (GPU): AMD Radeon Graphics (integrata)
- Memorie RAM: 32 GB DDR5
- Stocare: SSD NVMe PCIe 1 TB
- OS: Windows 11 Pro

Framework-uri folosite in proiect:

- JUnit (testare) versiune:5.11.4
- Mockito (mocking)
- Spring Boot (backend) versiune:3.4.0 versiune Java:21.06
- Testcontainers (testare cu MongoDB real in Docker) versiune 1.20.4

Comparatii Framework-uri/Tool-uri:

Framework/Tool	Avantaje	Dezavantaje
JUnit[3]	• Se integreaza nativ in Spring Boot • Usor de rulat automat (CI/CD) • Suporta testare structurata si modulara	• Nu acopera testare UI sau performanta • Pentru mocking avansat, necesita Mockito sau alte librarii
Mockito[4]	• Se pot testa bucati de cod fara interventia unor componente externe • Integrare buna cu JUnit si SpringTest • Un framework superior pentru testele unitare	• Folosirea excesiva a mocking poate pierde din vedere unele comportamente individuale • Codul nu mai este usor de vizualizat atunci cand folosesti multe mockuri
SpringBootTest	• Testarea se ruleaza pe o baza de date reala • Fiecare test porneste un container nou fara a afecta alte teste	• Testarea cu SpringBootTest dureaza o perioada semnificativ mai mare decat testarile unitare

		• Poate ingreuna pipeline-urile de build
Testcontainers[7]	• Testarea se ruleaza pe o baza de date reala • Fiecare test porneste un container nou fara a afecta alte teste	• Dureaza multa vreme sa porneasca containerele docker • Testele sunt greu de diagnosticat
Pit Test	• Verifică eficiența testelor unitare • Detectează cod netestat (mutanți supraviețuitori) • Se integrează cu JUnit și Gradle	• Timp de execuție mai mare decât testele clasice • Poate genera fals pozitive în unele cazuri
JMeter	• Măsoară performanța reală a aplicației REST • Suportă testare concurentă cu zeci/sute de requesturi • Interfață grafică + rapoarte detaliate	• Nu verifică logica internă a aplicației • Necesită configurare manual • Nu e potrivit pentru testare unitară

Servicii externe:

- MongoDB (containerizat)
- GitHub Codespaces (dev environment)
- Docker (mediu de test izolat)

Tool-uri suport:

- Gradle (build system)
- Postman (pentru testare API manuala, optional)
- Pit Test versiune:1.15.0
- JMeter(masurarea performantei)

Aplicatia se ruleaza cu:

- ./gradlew build && ./gradlew bootRun (local)
- make build && ./start.sh (in Docker)
- MongoDB ruleaza local prin Testcontainers, cu setup automat in testele de integrare
- Testele sunt scrise cu JUnit, organizate pe module:
- ReviewServiceTest.java – teste unitare cu Mockito
- ReviewControllerIntegrationTest.java – test de integrare cu context Spring si DB real

Functional Testing:

Partitionarea in clase de echivalenta:

Aceasta tehnica presupune impartirea valorilor de intrare in clase logice echivalente: o valoare din fiecare clasa este suficienta pentru a valida comportamentul.[5]

Exemplu 1: rating – valid si invalid:

```java
@Test
public void testCreateReview_ShouldReturnBadRequest_WhenRatingExceedsMaximum() throws Exception {
    ReviewRequestDTO invalidReviewRequest = new ReviewRequestDTO();
    invalidReviewRequest.setReviewerId("user3");
    invalidReviewRequest.setReviewedId("driver2");
    invalidReviewRequest.setRideId("ride3");
    invalidReviewRequest.setRating(10);
    invalidReviewRequest.setComment("Great ride!");

    mockMvc.perform(post("/reviews")
            .contentType(MediaType.APPLICATION_JSON)
            .content(new ObjectMapper().writeValueAsString(invalidReviewRequest)))
            .andExpect(status().isBadRequest())
            .andExpect(jsonPath("$").value("Rating must be between 1 and 5."));
}
```

Explicatie: Se testeaza clasa invalida in care ratingul depaseste valoarea maxima admisa (5). Validarea este deja implementata in serviciu si returneaza mesajul corespunzator.

Exemplu 2: seatsAvailable – clase valida, sub si peste limita

```java
@Test
void createRide_ShouldReturnBadRequest_WhenNoSeatsAvailable() throws Exception {
    setupValidDriverAndVehicle();
    RideRequestDTO request = createValidRideRequest();
    request.setSeatsAvailable(0);

    mockMvc.perform(post("/rides")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isBadRequest())
            .andExpect(jsonPath("$").value("Number of seats has to be greater than 0."));
}
```

```java
@Test
void createRide_ShouldReturnBadRequest_WhenSeatsAvailableExceedsLimit() throws Exception {
    RideRequestDTO request = createValidRideRequest();
    request.setSeatsAvailable(1000);

    mockMvc.perform(post("/rides")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isBadRequest())
            .andExpect(jsonPath("$").value("Number of seats exceeds the allowed limit."));
}
```

Explicatie: Testele acopera clasele invalide de input pentru numarul de locuri: sub minim (0) si peste maxim (1000). Este suficient un caz pentru fiecare, pentru ca orice alta valoare in acea clasa ar duce la acelasi rezultat.

Analiza valorilor de frontiera:

Aceasta tehnica testeaza exact valorile de la limita ale unui domeniu de intrare (minimul/maximul acceptabil + imediat sub/peste).

Exemplu: departureTime – valoare in trecut

```
@Test
void createRide_ShouldReturnBadRequest_WhenDepartureTimeInPast() throws Exception {
    setupValidDriverAndVehicle();
    RideRequestDTO request = createValidRideRequest();
    request.setDepartureTime(Instant.now().minusSeconds(3600));

    mockMvc.perform(post("/rides")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isBadRequest())
            .andExpect(jsonPath("$").value("Departure time must be in the future."));
}
```

Explicatie: Se testeaza limita inferioara a timpului de plecare – cu o valoare exact sub minimul acceptat . Comportamentul corect este respingerea cu eroare.

Exemplu: password – sub limita de lungime

```
@Test
void testCreateUser_ShortPassword() {
    UserRequestDTO newUser = new UserRequestDTO("Andrei", "Popescu", "andrei@gmail.com", "0787828282", "123");

    assertThrows(InvalidUserException.class, () -> {
        userService.createUser(newUser);
    });

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

Explicatie**:** Se testeaza cazul in care parola este mai scurta decat limita impusa (de ex. < 6 caractere). Asta e o valoare de la limita inferioara invalida.

Exemplu: rating – peste limita maxima admisa

```java
@Test
public void testCreateReview_ShouldReturnBadRequest_WhenRatingExceedsMaximum() throws Exception {
    ReviewRequestDTO invalidReviewRequest = new ReviewRequestDTO();
    invalidReviewRequest.setReviewerId("user3");
    invalidReviewRequest.setReviewedId("driver2");
    invalidReviewRequest.setRideId("ride3");
    invalidReviewRequest.setRating(10);
    invalidReviewRequest.setComment("Great ride!");

    mockMvc.perform(post("/reviews")
            .contentType(MediaType.APPLICATION_JSON)
            .content(new ObjectMapper().writeValueAsString(invalidReviewRequest)))
            .andExpect(status().isBadRequest())
            .andExpect(jsonPath("$").value("Rating must be between 1 and 5."));
}
```

Explicatie:

- Se testeaza cazul in care valoarea ratingului trimis in review depaseste limita maxima permisa (5).
- Valoarea 10 este o valoare de la limita superioara invalida**,** adica max + 1, folosita pentru analiza valorilor de frontiera (Boundary Value Analysis)**.**

Category Partitioning

Aceasta abordare presupune clasificarea valorilor de intrare in categorii si combinarea acestor categorii in scenarii de test.

Exemplu: startLocation == endLocation

```java
@Test
void createRide_ShouldReturnBadRequest_WhenSameStartEndLocation() throws Exception {
    setupValidDriverAndVehicle();
    RideRequestDTO request = createValidRideRequest();
    request.setEndLocation(request.getStartLocation());

    mockMvc.perform(post("/rides")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isBadRequest())
            .andExpect(jsonPath("$").value("Start location has to be different from end location"));
}
```

Explicatie: Aceasta este o categorie logica (start si end nu trebuie sa fie egale). In test se invalideaza doar aceasta regula, restul valorilor raman valide.

Exemplu: seatPrice < 0

```
@Test
void createRide_ShouldReturnBadRequest_WhenNegativePrice() throws Exception {
    setupValidDriverAndVehicle();
    RideRequestDTO request = createValidRideRequest();
    request.setSeatPrice(-10);

    mockMvc.perform(post("/rides")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isBadRequest())
            .andExpect(jsonPath("$").value("Price has to be greater or equal to 0."));
}
```

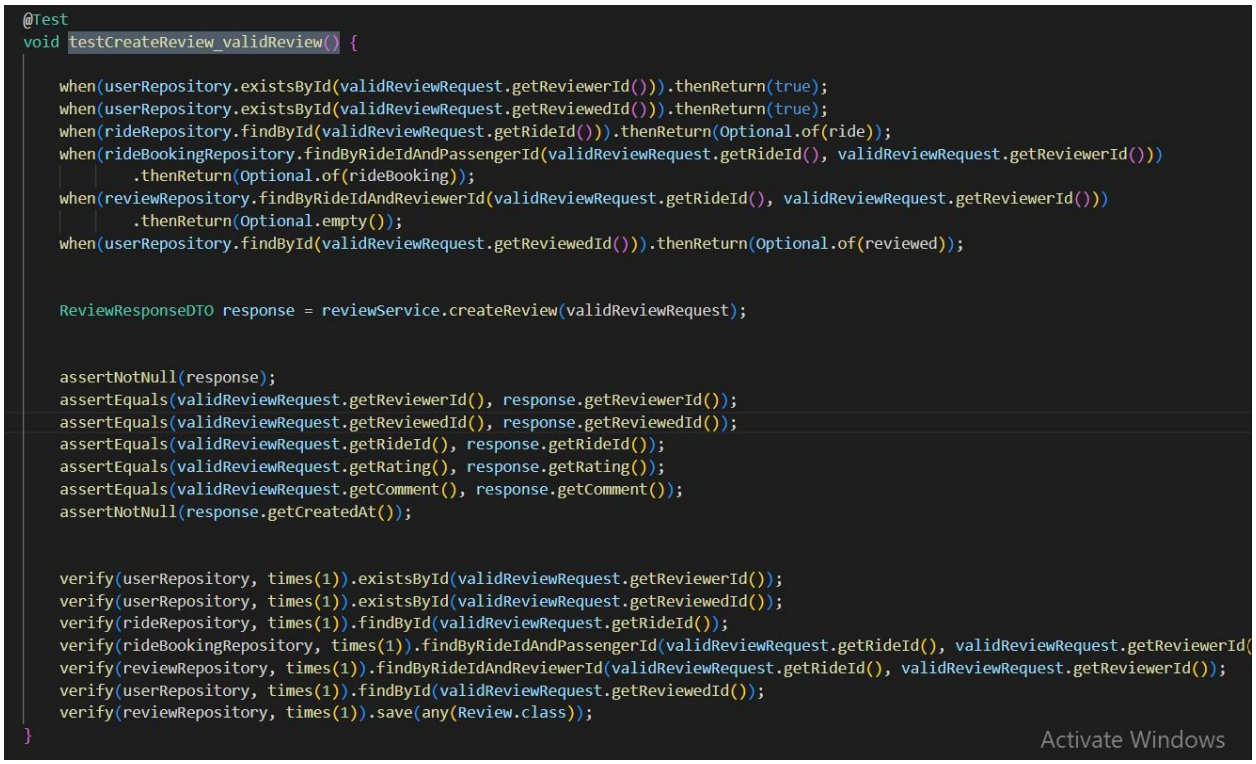
Explicatie: seatPrice este tratat ca o categorie numerica, iar testul acopera cazul in care acesta este invalid (negativ). Este singurul parametru invalid in test.

Structural Testing

Statement Testing

Definitie:Se asigura ca fiecare instructiune dintr-o metoda este executata cel putin o data de un test.

Exemplu: ReviewServiceTest.testCreateReview_validReview()



```
@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(), validReviewRequest.getReviewerId()))
            .thenReturn(Optional.of(rideBooking));
    when(reviewRepository.findByRideIdAndReviewerId(validReviewRequest.getRideId(), validReviewRequest.getReviewerId()))
            .thenReturn(Optional.empty());
    when(userRepository.findById(validReviewRequest.getReviewedId())).thenReturn(Optional.of(reviewed));


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


Explicatie:

Acest test apeleaza calea completa de succes din metoda createReview(), executand toate instructiunile**:** validari de ID-uri, cautari in DB, salvarea review-ului, construirea obiectului DTO etc.

Instructiunile liniare (if-uri fara else) sunt parcurse.

Acoperire: Statement coverage 100**%** pentru ramura „happy path”.

Decision Testing

Definitie:Asigura ca fiecare decizie (if, else) are ambele ramuri (true si false) acoperite de cel putin un test.

Exemplu: testCreateReview_reviewerDoesNotExist() + testCreateReview_validReview()

```java
@Test
void testCreateReview_reviewerDoesNotExist() {
    when(userRepository.existsById(validReviewRequest.getReviewerId())).thenReturn(false);


    InvalidReviewException exception = assertThrows(InvalidReviewException.class, () -> reviewService.createReview(validReviewRequest));
    assertEquals("Reviewer does not exist as user.", exception.getMessage());
}
```

Explicatie:

In createReview(), exista:

```java
if (!userRepository.existsById(reviewerId)) {
    throw new InvalidReviewException("Reviewer does not exist as user.");
}
```

Acest test asigura ca ramura „false” a fost testata.

Combinat cu testul de succes (validReview), ambele ramuri ale deciziei sunt acoperite.

Condition Testing

Definitie:Testarea fiecarei conditii simple dintr-o expresie compusa. Chiar daca if (a && b) este testat ca „true”, trebuie sa verificam si cazuri cand a este true dar b false, si invers.

Exemplu aplicabil: RideService.createRide()

```java
@Test
void testCreateRide_DepartureTimeInPast() {

    RideRequestDTO request = createValidRideRequest();
    request.setDepartureTime(Instant.now().minusSeconds(3600)); // 1 hour in the past

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

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

Explicatie:

Aici avem un if (conditie) cu o singura conditie, deci testul ofera si condition coverage, deoarece:

- ride.getDepartureTime().isBefore(Instant.now()) == true → ramura testata
- Complementul este testat de scenariile valide (ex: testCreateRide_Success), cand conditia este false.

Analiza raportului creat de generatorul de mutanti:

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
6	78% 219/280	64% 86/135	84% 86/102

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
ro.unibuc.hello.service	6	78% 219/280	64% 86/135	84% 86/102

Report generated by PIT 1.15.0

Enhanced functionality available at arcmutate.com

Teste suplimentare pentru a omori doi dintre mutantii ramasi in viata

139	1.1 **Location :** updateRideStatusToInProgress **Killed by :** none replaced return value with null for ro/unibuc/hello/service/RideService::updateRideStatusToInProgress → SURVIVED
139	1.1 **Location :** updateRideStatusToInProgress **Killed by :** ro.unibuc.hello.service.RideServiceTest.[engine:junit-jupiter]/[class:ro.unibuc.hello.service.RideServiceTest]/[method:testUpdateRideStatusToInProgressReturnsCorrectDTO()] replaced return value with null for ro/unibuc/hello/service/RideService::updateRideStatusToInProgress → KILLED

```java
@Test
    void testAvgRatingIsCorrectAfterValidReview() {
        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(), validReviewRequest.getReviewerId()))
                .thenReturn(Optional.of(rideBooking));
        when(reviewRepository.findByRideIdAndReviewerId(validReviewRequest.getRideId(), validReviewRequest.getReviewerId()))
                .thenReturn(Optional.empty());
        when(userRepository.findById(validReviewRequest.getReviewedId())).thenReturn(Optional.of(reviewed));

        when(userRepository.existsById(validReviewRequest2.getReviewerId())).thenReturn(true);
        when(userRepository.existsById(validReviewRequest2.getReviewedId())).thenReturn(true);
        when(rideRepository.findById(validReviewRequest2.getRideId())).thenReturn(Optional.of(ride));
        when(rideBookingRepository.findByRideIdAndPassengerId(validReviewRequest2.getRideId(), validReviewRequest2.getReviewerId()))
                .thenReturn(Optional.of(rideBooking));
        when(reviewRepository.findByRideIdAndReviewerId(validReviewRequest2.getRideId(), validReviewRequest2.getReviewerId()))
                .thenReturn(Optional.empty());
        when(userRepository.findById(validReviewRequest2.getReviewedId())).thenReturn(Optional.of(reviewed));


        // Act: creează review-ul valid
        reviewService.createReview(validReviewRequest);
        reviewService.createReview(validReviewRequest2);

        User updatedDriver = userRepository.findById(reviewed.getId()).orElseThrow();
        assertEquals(4.0, updatedDriver.getAvgRating());
        assertEquals(8, updatedDriver.getRatingsSum());
        assertEquals(2, updatedDriver.getReviewsNumber());
    }
```

128

2.2
Location : createReview
Killed by : none Replaced double division with multiplication → SURVIVED

128

2.2
Location : createReview
Killed by : ro.unibuc.hello.service.ReviewServiceTest.[engine:junit-jupiter]/[class:ro.unibuc.hello.service.ReviewServiceTest]/[method:testAvgRatingIsCorrectAfterValidReview()] Replaced double division with multiplication → KILLED

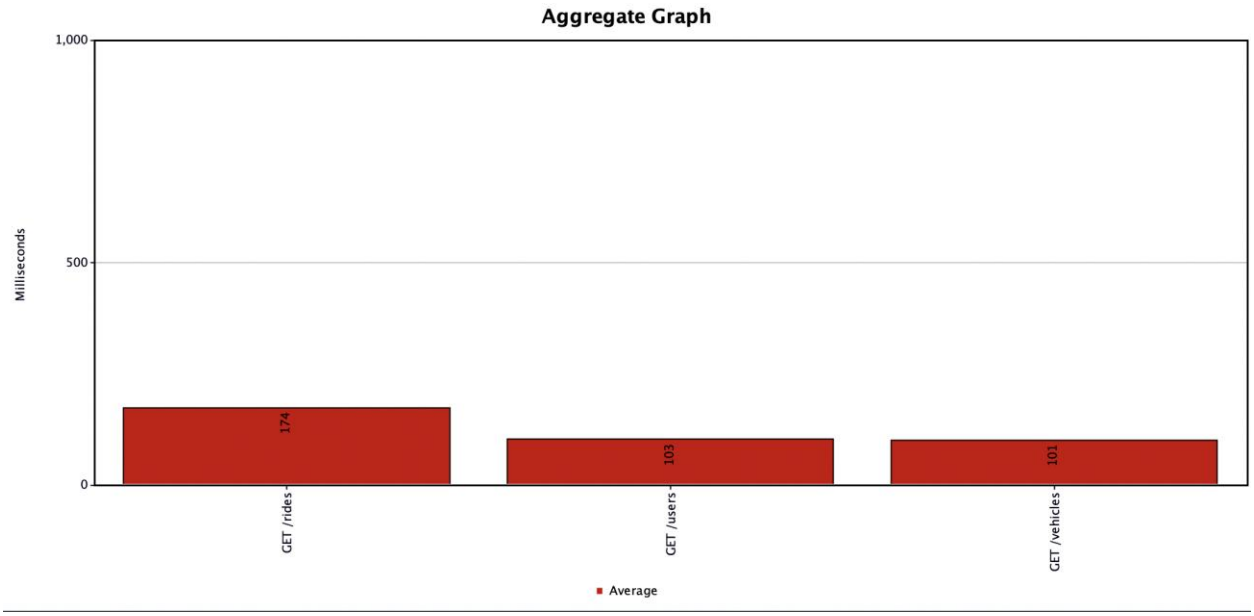
```java
@Test
    void testUpdateRideStatusToInProgressReturnsCorrectDTO() {
        String rideId = "ride123";
        Ride ride = new Ride();
        ride.setId(rideId);
        ride.setStatus(RideStatus.SCHEDULED);
        ride.setDepartureTime(Instant.now().minusSeconds(60));

        when(rideRepository.findById(rideId)).thenReturn(Optional.of(ride));
        when(rideRepository.save(any(Ride.class))).thenAnswer(invocation -> invocation.getArgument(0));

        RideResponseDTO result = rideService.updateRideStatusToInProgress(rideId);

        assertNotNull(result, "Returned RideResponseDTO should not be null");
    }
```

Apache JMeter:

Apache JMeter este un instrument open-source folosit pentru testarea performantei aplicatiilor, in special aplicatii web si servicii REST. Permite simularea unui numar mare de utilizatori simultani care trimit cereri catre server, pentru a evalua timpii de raspuns, stabilitatea si scalabilitatea sistemului.

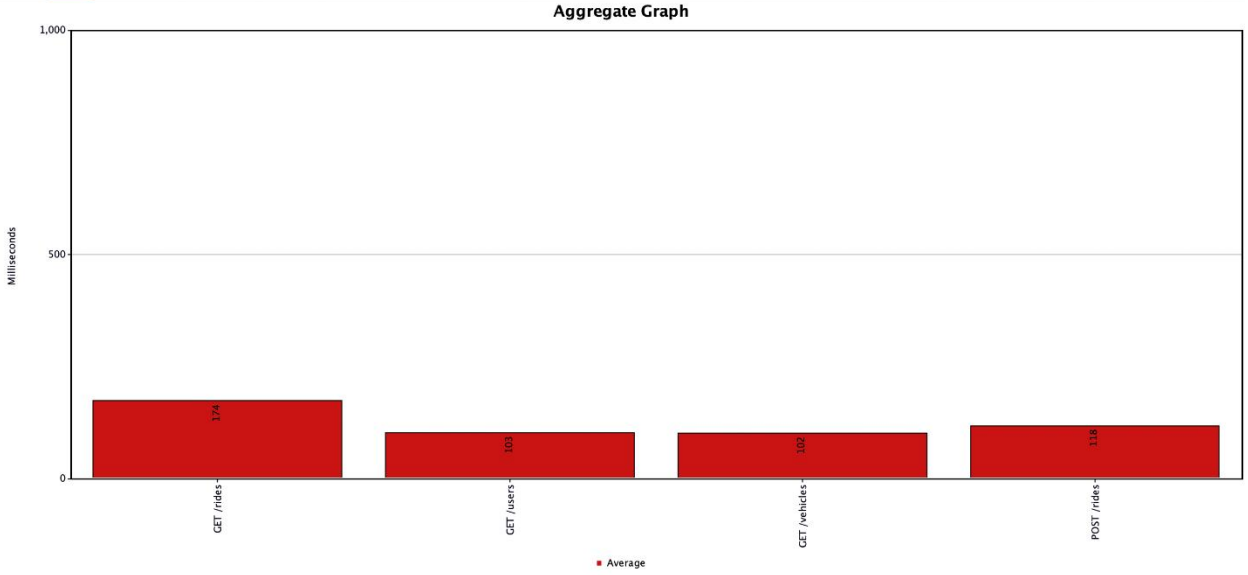
Analiza raportului JMeter:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
GET /rides	750	174	117	232	433	1504	64	1807	0.00%	40.9/sec	87.08	8.64
GET /users	375	103	94	123	153	233	63	1112	0.00%	87.6/sec	186.48	18.47
GET /vehicles	375	102	78	116	173	758	62	1182	0.00%	95.3/sec	202.92	20.10
TOTAL	1500	138	100	186	271	1182	62	1807	0.00%	19.0/sec	40.39	4.00



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
GET /rides	750	174	117	232	433	1504	64	1807	0.00%	40.9/sec	87.08	8.64
GET /users	375	103	94	123	153	233	63	1112	0.00%	87.6/sec	186.48	18.47
GET /vehicles	375	102	78	116	173	758	62	1182	0.00%	95.3/sec	202.92	20.10
POST /rides	1116	118	87	156	198	894	62	1522	100.00%	23.5/sec	54.02	13.28
TOTAL	2616	129	95	172	236	1037	62	1807	42.66%	4.3/sec	9.51	1.56

Settings Graph



Articole stiintifice si resurse relevante:

[1] I. Mirzayev, “Integrating Tests into CI/CD Pipelines,” *Medium*, 16-Oct-2024. [Online]. Available: https://medium.com/@idrakmirzoyev/integrating-tests-into-ci-cd-pipelines-4ca1c3e0592b. [Accessed: 15-May-2025].

[2] Lotus QA, “Top Java Testing Frameworks for 2023,” *Lotus Quality Assurance*. [Online]. Available: https://www.lotus-qa.com/blog/java-testing-frameworks/. [Accessed: 15-May-2025].

[3] GeeksforGeeks, “Introduction of JUnit,” *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/introduction-of-junit/. [Accessed: 15-May-2025].

[4] Waldo, “How to Mock a Generic Class with Mockito,” *Waldo Blog*. [Online]. Available: https://www.waldo.com/blog/mockito-mock-generic-class. [Accessed: 15-May-2025].

[5] GeeksforGeeks, “Equivalence Partitioning Method,” *GeeksforGeeks*. [Online]. Available: https://www.geeksforgeeks.org/equivalence-partitioning-method/. [Accessed: 15-May-2025].

[6] That API Company, “A Comprehensive Guide to API Testing Strategies and Tools,” *That API Company*. [Online]. Available: https://thatapicompany.com/a-comprehensive-guide-to-api-testing-strategies-and-tools/. [Accessed: 15-May-2025].

[7] Testcontainers, “Introducing Testcontainers,” *Testcontainers Guides*. [Online]. Available: https://testcontainers.com/guides/introducing-testcontainers/. [Accessed: 15-May-2025].