BOOKWISH

333-Team
Enescu Irina-Ștefania, Guță George, Titirigă Tiberiu-Nicolae, Voinea Ana-Maria
(Specializarea Informatică, Grupa 333)

WOW!

BookWish este o aplicație web creată pentru a facilita gestionarea cărților, autorilor și cititorilor într-o bibliotecă digitală. Sistemul permite utilizatorilor să efectueze diverse sarcini, cum ar fi adăugarea, actualizarea și ștergerea informațiilor legate de cărți, autori și cititori.

FUNCȚIONALITĂȚI

Gestionarea autorilor

- Utilizatorii pot adăuga un autor nou în sistemul de bibliotecă online, furnizând detalii necesare precum nume, naționalitate, dată de naștere sau dată de deces.
- Utilizatorii pot actualiza data decesului unui autor existent.
- Utilizatorii pot obține o listă de toți autorii înregistrați în sistemul de bibliotecă online.
- Utilizatorii pot șterge un autor din sistem, dar numai dacă autorul nu a scris nicio carte înregistrată în sistem.

Gestionarea cititorilor

- Utilizatorii pot adăuga un cititor nou în bibliotecă, furnizând detalii necesare precum nume, adresă de email, telefon, dată de naștere și dată de înregistrare.
- Utilizatorii pot actualiza informațiile de contact ale unui cititor existent.
- Utilizatorii pot șterge un cititor din sistem, dar numai dacă cititorul nu a citit nicio carte și nu a adăugat nici o recenzie unei cărți.
- Utilizatorii pot obține o listă de toți cititorii înregistrați în sistemul de bibliotecă online.

FUNCȚIONALITĂȚI

Gestionarea cărților

- Utilizatorii pot adăuga cărți noi, furnizând detalii precum titlu, anul publicării, autor, gen, editură.
- Cărțile pot fi șterse din sistem dacă este necesar.
- Utilizatorii pot obține o listă de cărți citite de un anumit cititor.
- Utilizatorii pot obține o listă de cărți scrise de un anumit autor.

Gestionarea înregistrărilor

- Utilizatorii pot marca citirea unei cărți noi adăugând-o ca înregistrare.
- Utilizatorii pot obține o listă cu toate înregistrările din sistemul bbibliotecii online.

Gestionarea recenziilor

- Utilizatorii pot adăuga o recenzie unei cărți pe care au citit-o, specificând detalii precum evaluarea și feedback-ul.
- Utilizatorii pot actualiza o recenzie adăugată unei cărți.
- Utilizatorii pot șterge o recenzie adăugată unei cărți dacă nu mai este relevantă.

DEFINIREA API-URILOR

- **POST/authors** - adaugă un autor nou în sistem
- **GET/authors** - returnează toți autorii din sistem
- **PATCH/authors/{author_id}** - actualizează informațiile unui autor existent
- **DELETE/authors** - șterge un autor după nume și data de naștere din sistem doar dacă acesta nu a scris nicio carte

- **POST/readers** - adaugă un cititor nou în sistem
- **PATCH/readers/{reader_id}** - actualizează informațiile unui cititor existent
- **DELETE/readers/records/{reader_id}** - șterge un cititor din sistem și înregistrările sale
- **GET/readers** - returnează toți cititorii din sistem

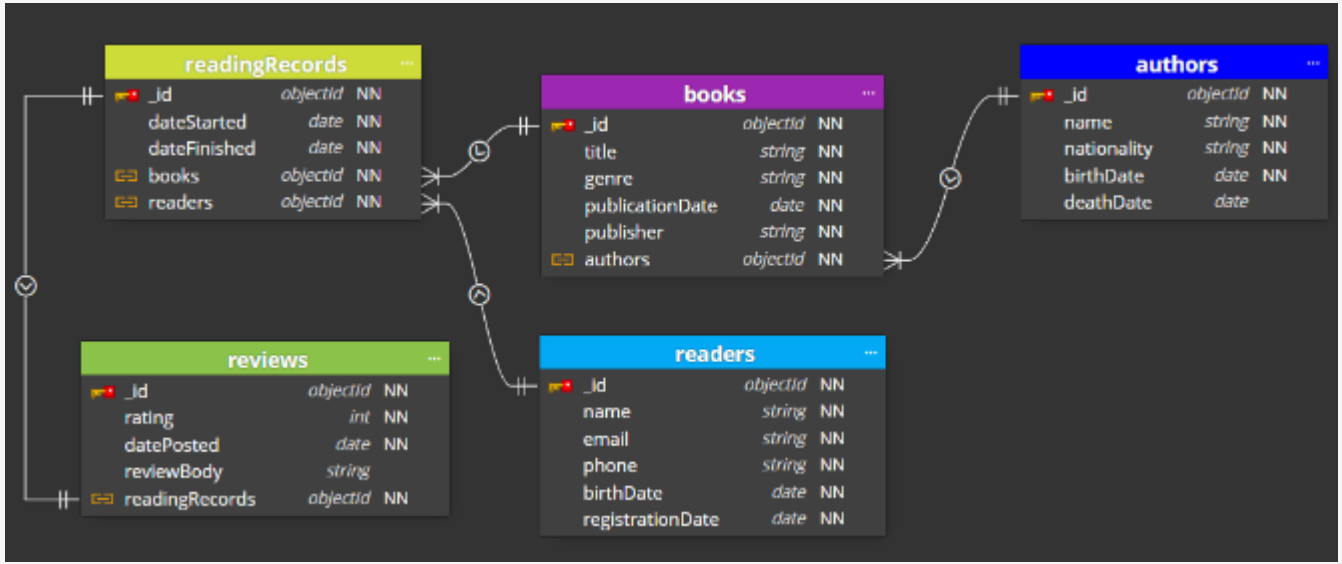
- **POST/records** – adaugă o nouă înregistrare de lectură, cititorul citește o carte nouă

- **POST/books** - adaugă o carte nouă în sistem
- **DELETE/books/{book_id}** - șterge o carte din sistem și înregistrările sale
- **GET/books/readers/{reader_id}** - returnează toate cărțile citite de un cititor
- **GET/books/authors/{author_id}** - returnează toate cărțile scrise de un autor

- **POST/reviews** – adaugă o recenzie nouă la o carte citită de cititor
- **PATCH/reviews/{review_id}** – editează o recenzie existentă la o carte citită
- **DELETE/reviews/{review_id}** – șterge o recenzie existentă la o carte citită

- **GET/records** - returnează toate înregistrările de lectură din biblioteca online

STRUCTURA BAZEI DE DATE A APLICAȚIEI WEB



CONTROLLER – CERERI HTTP

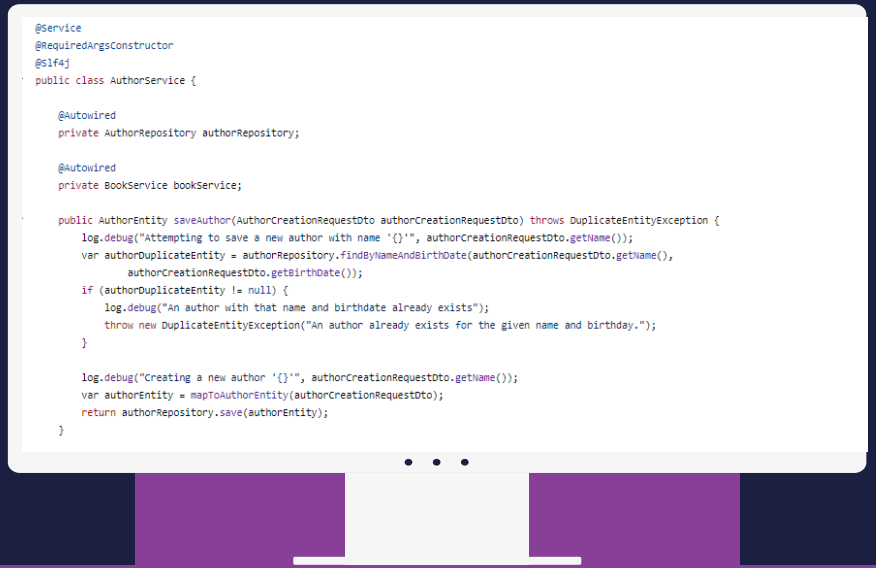
În controller sunt definite metode care corespund API-urilor definite în aplicație. În aceste metode, sunt procesate datele primite în cererea HTTP și se returnează un răspuns corespunzător către client. În exemplul din imagine sunt implementate cele doua API-uri corespunzătoare ReadingRecord:

- POST/records – adaugă o nouă înregistrare de lectură, cititorul citește o carte nouă
- GET/records - returnează toate înregistrările de lectură din biblioteca online

```java
@Controller
public class ReadingRecordController {
    @Autowired
    private ReadingRecordService readingRecordService;

    @GetMapping("/readingrecords")
    @ResponseBody
    private List<ReadingRecordEntity> getReadingRecord() {
        return readingRecordService.getAllReadingRecords();
    }

    @PostMapping("/readingrecords")
    @ResponseBody
    public ResponseEntity<ReadingRecordEntity> createReadingRecord(
            @RequestBody ReadingRecordCreationRequestDto readingRecordCreationRequestDto) {
        var newReadingRecord = readingRecordService.saveReadingRecord(readingRecordCreationRequestDto);
        return ResponseEntity.ok(newReadingRecord);
    }
```

SERVICE – LOGICA SISTEMULUI

În service sunt implementate operațiunile sistemului, mai exact logica aplicației. Acesta este folosit pentru a izola logica sistemului de controller și pentru a face codul mai ușor de testat. În exemplul din imagine este implementată metoda apelată în momentul salvării unui autor în baza de date. Se poate observa validarea datelor introduse, fiind verificat faptul că nu mai există în baza de date un autor cu același nume și aceeași dată de naștere. Metoda privată mapToAuthorEntity() pregătește datele pentru crearea unui AuthorEntity, mapându-le corespunzător.



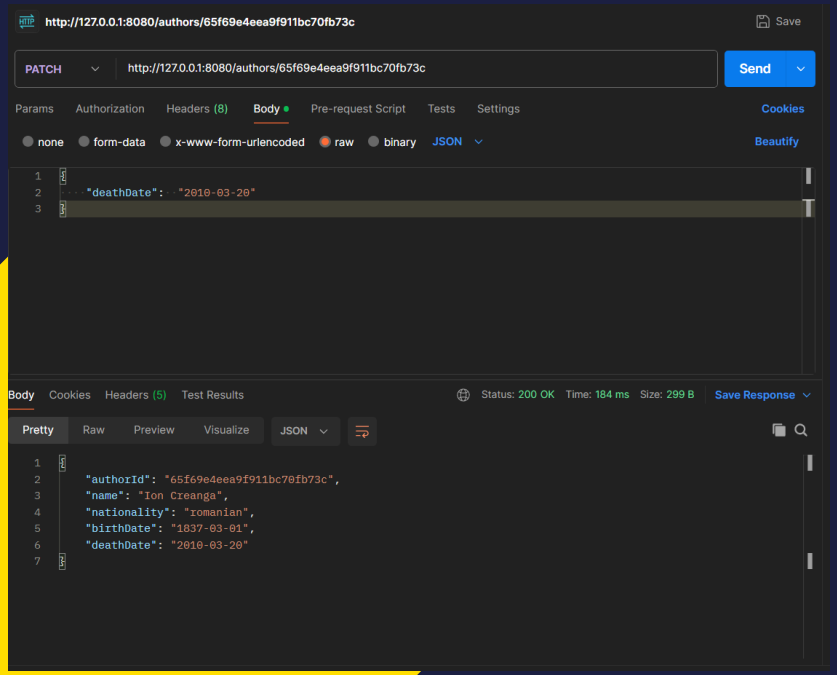
```java
@Service
@RequiredArgsConstructor
@Slf4j
public class AuthorService {

    @Autowired
    private AuthorRepository authorRepository;

    @Autowired
    private BookService bookService;

    public AuthorEntity saveAuthor(AuthorCreationRequestDto authorCreationRequestDto) throws DuplicateEntityException {
        log.debug("Attempting to save a new author with name '{}'", authorCreationRequestDto.getName());
        var authorDuplicateEntity = authorRepository.findByNameAndBirthDate(authorCreationRequestDto.getName(),
                authorCreationRequestDto.getBirthDate());
        if (authorDuplicateEntity != null) {
            log.debug("An author with that name and birthdate already exists");
            throw new DuplicateEntityException("An author already exists for the given name and birthday.");
        }

        log.debug("Creating a new author '{}'", authorCreationRequestDto.getName());
        var authorEntity = mapToAuthorEntity(authorCreationRequestDto);
        return authorRepository.save(authorEntity);
    }
```

REPOSITORY – BAZA DE DATE

În contexul aplicației web, layer-ul repository devine un intermediar între baza de date și logica aplicației. Acesta abstractizează operațiunile și facilitează conexiunea cu baza de date. În exemplul alăturat este prezentat AuthorRepository și metodele corespunzătoare de furnizare a datelor. Întrucât se extinde MongoRepository, corpul cererilor sql nu mai trebuie specificat, fiind interpretat automat prin intermediul numelui metodei (findByNameAndBirthDate) și a parametrilor sugestivi: String name și LocalDate birthdate.

```java
package ro.unibuc.hello.data;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

import java.time.LocalDate;
import java.util.List;

@Repository
public interface AuthorRepository extends MongoRepository<AuthorEntity, String> {
    AuthorEntity findByName(String name);
    AuthorEntity findByNameAndBirthDate(String name, LocalDate birthDate);
    List<AuthorEntity> findAll();
}
```

TESTAREA SISTEMULUI - POSTMAN



Pentru testarea API-urilor am utilizat atât Postman, cât și Thunder Client. Acestea oferă un mediu de trimitere a cererilor HTTP și de examinare a răspunsurilor pentru a verifica funcționarea corectă a API-ului. Colecțiile de teste au fost partajate în cadrul echipei pentru a facilita o colaborare mai bună.

Ca exemplu avem un request de tip PATCH ce face update la data decesului unui autor. Id-ul autorului se trimite ca PathVariable, în timp ce data actualizată a morții este trimisă ca RequestBody. Se observă în răspuns statusul 200 OK și entitatea returnată, cu data decesului actualizată.

TESTE UNITARE - JUNIT

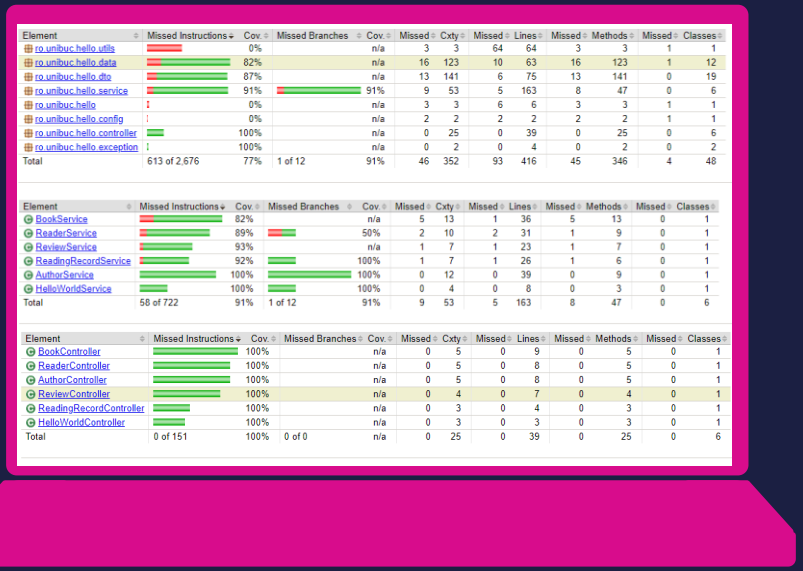
Pentru a testa funcționalitățile unei metode am implementat cu ajutorul JUnit teste unitare care sa acopere toate cazurile posibile, aspect ilustrat în exemplul de mai jos. În cazul în care există un autor cu id-ul specificat, verificăm returnarea unei liste de cărți, dar în cazul în care acesta nu există, verificăm aruncarea excepției custom EntityNotFoundException.

Cuvinte cheie: mock, assert

```
@Test
public void test_getAllBooksByAuthor_EntityNotFoundException() throws Exception {

    // Given
    var authorId = "1";
    when(authorRepository.findById(anyString()))
            .thenThrow(new EntityNotFoundException("Author not found with id: " + authorId));

    // When
    var exception = assertThrows(EntityNotFoundException.class,
            () -> bookService.getBooksByAuthor("1"));
    Assertions.assertEquals("Entity: Author not found with id: 1 was not found", exception.getMessage());

}
```

```
@Test
public void test_getAllBooksByAuthor_Ok() throws Exception {
    // Given
    var author = AuthorEntity.builder().authorId("1").name("Mircea Eliade").nationality("romanian")
            .birthDate(LocalDate.of(1837, 03, 1)).deathDate(LocalDate.of(1986, 4, 22))
            .build();

    var book = BookEntity.builder().bookId("1").title("Baltagul").genre("roman")
            .publicationDate(LocalDate.of(1930, 11, 1))
            .publisher("Cartea Romaneasca")
            .author(author)
            .build();

    var bookList = List.of(book);
    var authorId = "1";
    when(authorRepository.findById(anyString())).thenReturn(Optional.of(author));
    when(bookRepository.findByAuthor(any())).thenReturn(bookList);

    // When
    var response = bookService.getBooksByAuthor(authorId);

    // Then
    Assertions.assertEquals(bookList, response);
}
```

ACOPERIRE LA NIVEL DE INSTRUCȚIUNE



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ro.unibuc.hello.utils		0%		n/a	3	3	64	64	3	3	1	1
ro.unibuc.hello.data		82%		n/a	16	123	10	63	16	123	1	12
ro.unibuc.hello.dto		87%		n/a	13	141	6	75	13	141	0	19
ro.unibuc.hello.service		91%		91%	9	53	5	163	8	47	0	6
ro.unibuc.hello		0%		n/a	3	3	6	6	3	3	1	1
ro.unibuc.hello.config		0%		n/a	2	2	2	2	2	2	1	1
ro.unibuc.hello.controller		100%		n/a	0	25	0	39	0	25	0	6
ro.unibuc.hello.exception		100%		n/a	0	2	0	4	0	2	0	2
Total	613 of 2,676	77%	1 of 12	91%	46	352	93	416	45	346	4	48

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
BookService		82%		n/a	5	13	1	36	5	13	0	1
ReaderService		89%		50%	2	10	2	31	1	9	0	1
ReviewService		93%		n/a	1	7	1	23	1	7	0	1
ReadingRecordService		92%		100%	1	7	1	26	1	6	0	1
AuthorService		100%		100%	0	12	0	39	0	9	0	1
HelloWorldService		100%		100%	0	4	0	8	0	3	0	1
Total	58 of 722	91%	1 of 12	91%	9	53	5	163	8	47	0	6

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
BookController		100%		n/a	0	5	0	9	0	5	0	1
ReaderController		100%		n/a	0	5	0	8	0	5	0	1
AuthorController		100%		n/a	0	5	0	8	0	5	0	1
ReviewController		100%		n/a	0	4	0	7	0	4	0	1
ReadingRecordController		100%		n/a	0	3	0	4	0	3	0	1
HelloWorldController		100%		n/a	0	3	0	3	0	3	0	1
Total	0 of 151	100%	0 of 0	n/a	0	25	0	39	0	25	0	6

După cum se poate observa, am scris teste unitare atât pentru service, cât și pentru controller sau data, ajungând la un o acoperire la nivel de instrucțiune foarte mare - mai exact 77%. Aceasta reprezintă un aspect foarte important deoarece măsoară procentul de instrucțiuni care au fost executate cel puțin o dată în timpul unui set de teste.

Se remarcă procentajele privind acoperirea la nivel de instrucțiune a claselor, metodelor și intrucțiunilor. Acestea sunt furnizate sub forma unui HTML cu un raport clar în care observăm ce instrucțiuni sunt acoperite de testele scrise de noi.

TESTE DE INTEGRARE

Pentru a testa modul în care interacționează componentele între ele am ales să scriem teste de integrare. Acestea ajută la gestionarea detaliată a dependențelor, semnalându-se cazurile în care două componente funcționează corespunzător individual, dar defectuos împreună.

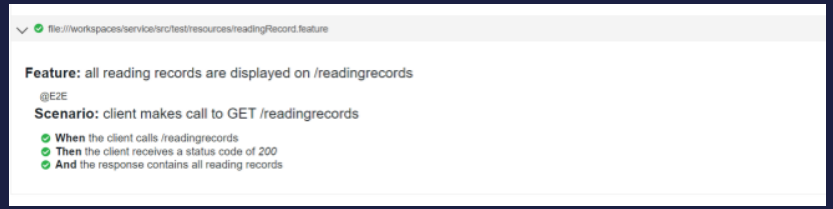
În exemplul alăturat se observă testarea conexiunii service – repository (AuthorRepository și AuthorService), AuthorRepository fiind adnotat cu @Autowired, și nu cu @Mock precum în cazul testelor unitare.

```java
@SpringBootTest
@Tag("IT")
class AuthorServiceTestIT {

    @Autowired
    AuthorRepository authorRepository;

    @Autowired
    AuthorService authorService;

    @Test
    void test_getAuthor_returnsAuthor() {
        // Given
        var authorCreationRequestDto = AuthorCreationRequestDto.builder().name("Mircea Eliadee").nationality("romanian")
                .birthDate(LocalDate.of(1837, 03, 1)).deathDate(LocalDate.of(1986, 4, 22))
                .build();

        // When
        var savedAuthor = authorService.saveAuthor(authorCreationRequestDto);

        // Then
        Assertions.assertEquals("Mircea Eliadee", savedAuthor.getName());
        Assertions.assertEquals(LocalDate.of(1837, 03, 1), savedAuthor.getBirthDate());
        Assertions.assertEquals(LocalDate.of(1986, 4, 22), savedAuthor.getDeathDate());
        Assertions.assertEquals("romanian", savedAuthor.getNationality());
    }
}
```

TESTE DE INTEGRARE - CUCUMBER

Pentru testarea E2E (End To End) a proiectului și a API-urilor dezvoltate am utilizat framework-ul Cucumber. Am specificat ca feature toți pașii pe care programul ar trebui să îi urmeze (scenariul) și am implementat fiecare pas definit anterior după cum urmează în imagini. Testele au trecut cu succes, fiind specificat statusul fiecărui pas executat.

```
Feature: all reading records are displayed on /readingrecords
    @E2E
    Scenario: client makes call to GET /readingrecords
        When the client calls /readingrecords
        Then the client receives a status code of 200
        And the response contains all reading records
```

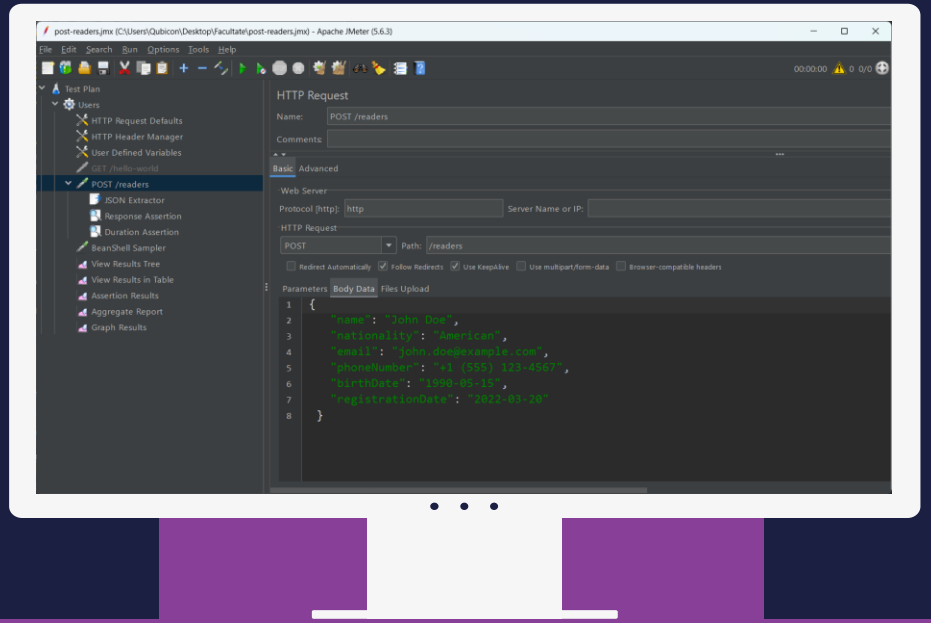



```java
@Given("^the client calls /authors")
public void the_client_issues_GET_authors() {
    executeGet("http://localhost:8080/authors");
}

@Then("^the client receives for /authors status code of (\\d+)$")
public void the_client_receives_status_code_of(int statusCode) throws Throwable {
    final HttpStatus currentStatusCode = latestResponse.getTheResponse().getStatusCode();
    assertThat("status code is incorrect : " + latestResponse.getBody(), currentStatusCode.value(), is(statusCode));
}

@And("^the client receives a response in JSON format with author \"(.+)\"$")
public void the_client_receives_json_response_with_author(String authorName) throws JSONException {
    String latestResponseBody = latestResponse.getBody();
    JSONArray jsonArray = new JSONArray(latestResponseBody);
    boolean foundAuthor = false;
    for (int i = 0; i < jsonArray.length(); i++) {
        JSONObject jsonObject = jsonArray.getJSONObject(i);
        if (jsonObject.getString("name").equals(authorName)) {
            foundAuthor = true;
            break;
        }
    }
    assertThat("Response JSON does not contain author: " + authorName, foundAuthor, is(true));
}

public void executeGet(String url) {
    final Map<String, String> headers = new HashMap<>();
    headers.put("Accept", "application/json");
    final HeaderSetup requestCallback = new HeaderSetup(headers);
    final ResponseErrorHandler errorHandler = new ResponseErrorHandler();

    restTemplate.setErrorHandler(errorHandler);
    latestResponse = restTemplate.execute(url, HttpMethod.GET, requestCallback, response -> {
        if (errorHandler.getHadError()) {
            return (errorHandler.getResults());
        } else {
            return (new ResponseResults(response));
        }
    });
}
```

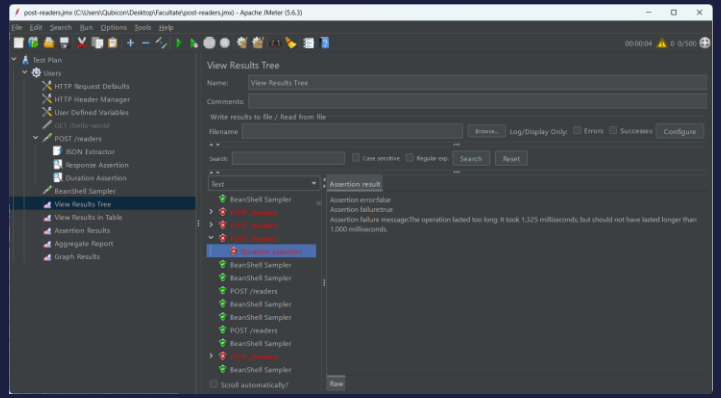
TESTARE DE PERFORMANTA

Pentru testarea performanței programului am utilizat tool-ul JMeter. Testele au fost configurate după modelul din imagine, fiind definite:

- X-Github-Token
- HTTP Header Manager
- HTTP Request Defaults
- JSON Extractor
- Response Assertions
- Duration Assertions
- View Results Tree or in Table
- Assertions Results



TESTARE DE PERFORMANTA



BeanShell Sampler-ul extrage id-ul din raspunsul json primiit: 2024-05-15 18:59:31,343 INFO o.a.j.u.BeanShellTestElement: Extracted id from JSON response: 6644dbe4c9e58f646b99f5f7

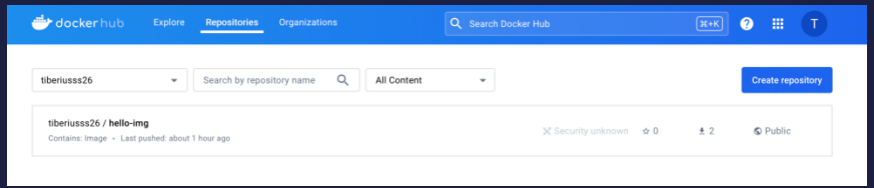
În imaginea alăturată sunt prezentate rezultatele apelării endpoint-ului postAuthors ce introduce mai mulți autori în baza de date. Se poate observa că unele teste au trecut cu succes, pe când alte nu s-au încadrat în așteptările noastre privind timpul de răspuns.

The operations lasted too long: It took 1235 milliseconds, but should not have lasted longer than 1000 milliseconds.

CI & CD

Am folosit Jenkins pentru a rula un job ce face build unei noi imagini de fiecare dată când apar modificări pe GitHub sau când job-ul este declanșat de trigger.

După cum se poate observa și în imagine, precizăm comenzile ce vor fi executate în cadrul rulării pipeline-ului – clean build, fetch git tag, build image with specified version, docker login și docker push. Din motive de securitate, credențialele pentru Docker le introducem prin intermediul DOCKER_PASSWORD (Jenkins Credentials). Imaginea este urcată în Docker:

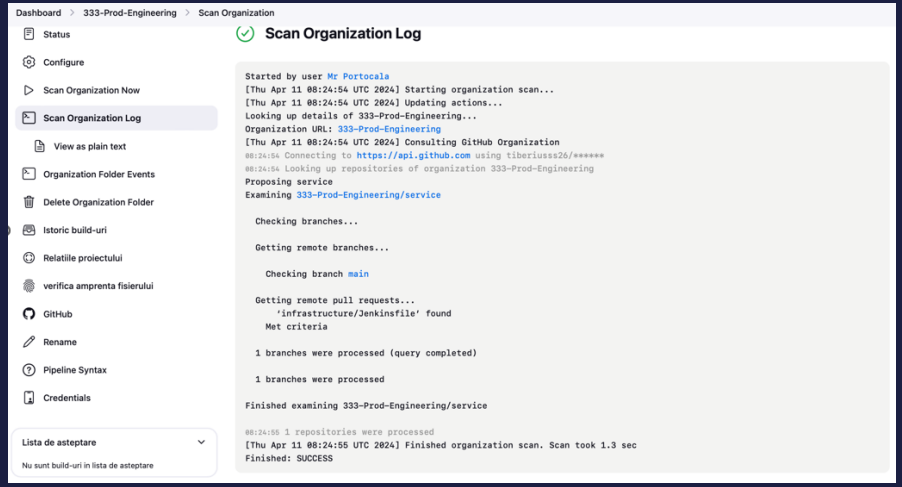


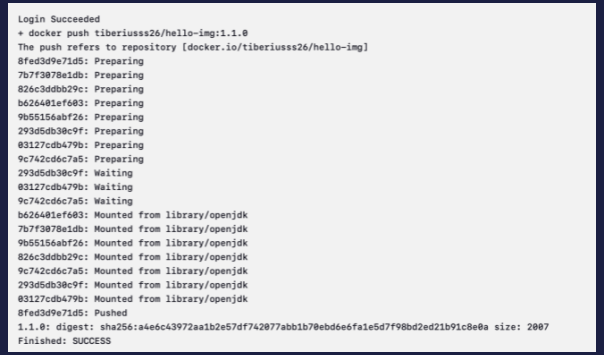

```
pipeline {
    agent any
    environment {
        DOCKER_PASSWORD = credentials("docker_password")

    }

    stages {
        stage('Build & Test') {
            steps {
                sh './gradlew clean build'
            }
        }

        stage('Tag image') {
            steps {
                script {
                    GIT_TAG = sh([script: 'git fetch --tag && git tag', returnStdout: true]).trim()
                    MAJOR_VERSION = sh([script: 'git tag | cut -d . -f 1', returnStdout: true]).trim()
                    MINOR_VERSION = sh([script: 'git tag | cut -d . -f 2', returnStdout: true]).trim()
                    PATCH_VERSION = sh([script: 'git tag | cut -d . -f 3', returnStdout: true]).trim()
            }
            sh "docker build -t tiberiusss26/hello-img:${MAJOR_VERSION}.\$((${MINOR_VERSION} + 1)).${PATCH_VERSION} ."
          }
        }

        stage('Push') {
            steps {
                sh "docker login docker.io -u $DOCKER_PASSWORD_USR -p $DOCKER_PASSWORD_PSW"
                sh "docker push $DOCKER_PASSWORD_USR/hello-img:${MAJOR_VERSION}.\$((${MINOR_VERSION} + 1)).${PATCH_VERSION}"
                }
        }

    }
}
```

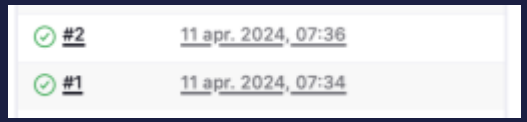
Jenkinsfile pentru definirea fiecărui pipeline stage

SCANAREA ORGANIZATIEI

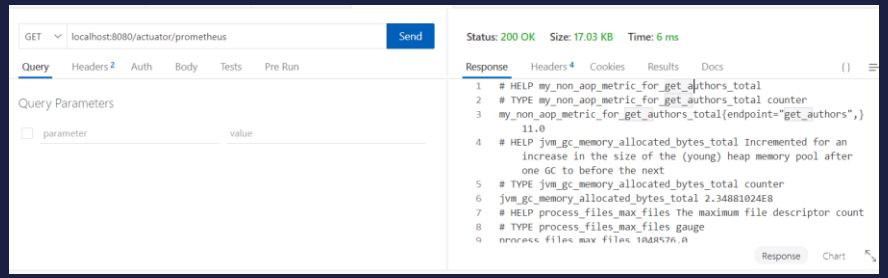
Pentru Organization Scan ne-am folosit de credențialele Docker definite anterior și de regex, precizând ca vrem să scanăm doar .*service. Se observă în imagini că pipeline-ul rulează cu succes.





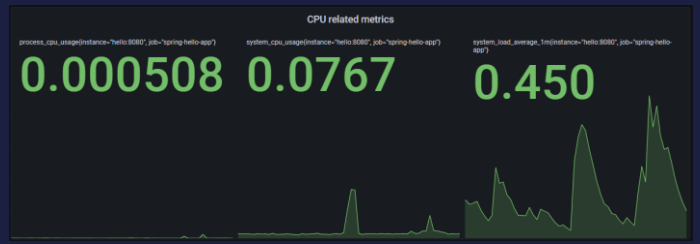


MONITORIZARE SI ALERTARE



Pentru monitorizare și alertare am utilizat Prometheus, un sistem open-source ce colectează diverse metrici și expune performanța și starea sistemului. Sistemul include Alert Manager. Pentru a vizualiza și interpreta metricile definite am utilizat Grafana.

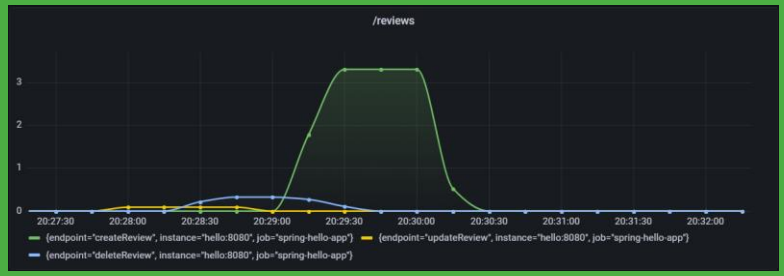
- Metrici custom (ex. metrică ce măsoară numărul de request-uri pentru getAuthors)
- Metrici de performanță



MONITORIZARE ȘI ALERTARE

În exemplul alăturat sunt ilustrate metrici definite pe funcționalitățile corespunzătoare reviews (delete, create si update).

```
# HELP controller_delete_review_counter_total
# TYPE controller_delete_review_counter_total counter
controller_delete_review_counter_total{endpoint="deleteReview",} 3.0
# HELP controller_create_review_counter_total
# TYPE controller_create_review_counter_total counter
controller_create_review_counter_total{endpoint="createReview",} 6.0
# HELP controller_update_review_counter_total
# TYPE controller_update_review_counter_total counter
controller_update_review_counter_total{endpoint="updateReview",} 1.0
```



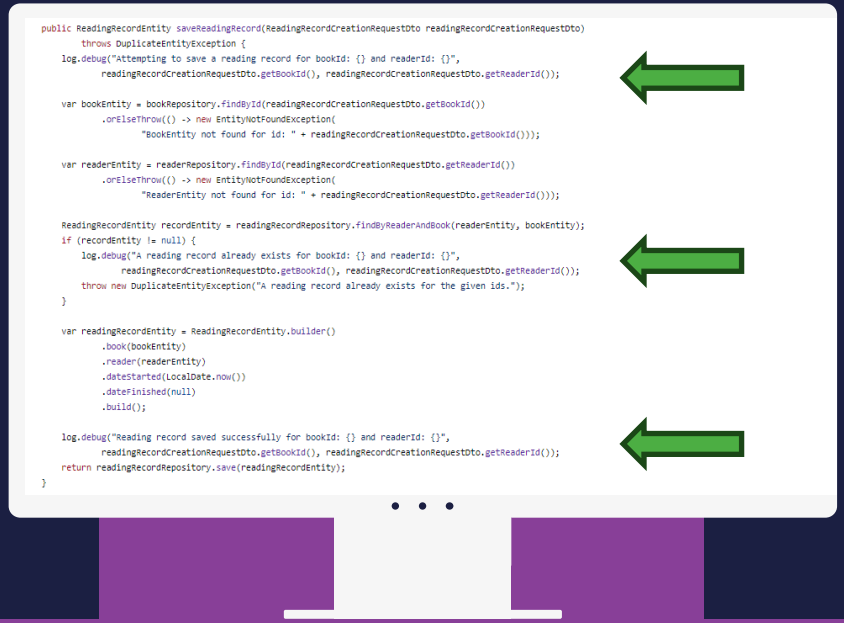
Am decis să configurăm alerte pentru detectarea unor anomalii precum primirea unui număr mult prea mare de request-uri în același timp. Folosind JMeter am generat trafic artifical pentru a declansa alerta definita anterior.

LOG-URI PERSONALIZATE

Am ales sa utilizăm log-uri întrucât sunt foarte utile pentru diagnosticarea si depanarea problemelor într-un sistem informatic.

După cum se observă în exemplul alăturat, marcăm fiecare informație importantă cu un log specific cu nivel de DEBUG:

- Initiere salvare inregistrare
- Inregistrarea deja exista in sistem
- Inregistrarea a fost salvata cu succes



```
public ReadingRecordEntity saveReadingRecord(ReadingRecordCreationRequestDto readingRecordCreationRequestDto)
        throws DuplicateEntityException {
    log.debug("Attempting to save a reading record for bookId: {} and readerId: {}",
            readingRecordCreationRequestDto.getBookId(), readingRecordCreationRequestDto.getReaderId());

    var bookEntity = bookRepository.findById(readingRecordCreationRequestDto.getBookId())
            .orElseThrow(() -> new EntityNotFoundException(
                    "BookEntity not found for id: " + readingRecordCreationRequestDto.getBookId()));

    var readerEntity = readerRepository.findById(readingRecordCreationRequestDto.getReaderId())
            .orElseThrow(() -> new EntityNotFoundException(
                    "ReaderEntity not found for id: " + readingRecordCreationRequestDto.getReaderId()));

    ReadingRecordEntity recordEntity = readingRecordRepository.findByReaderAndBook(readerEntity, bookEntity);
    if (recordEntity != null) {
        log.debug("A reading record already exists for bookId: {} and readerId: {}",
                readingRecordCreationRequestDto.getBookId(), readingRecordCreationRequestDto.getReaderId());
        throw new DuplicateEntityException("A reading record already exists for the given ids.");
    }

    var readingRecordEntity = ReadingRecordEntity.builder()
            .book(bookEntity)
            .reader(readerEntity)
            .dateStarted(LocalDate.now())
            .dateFinished(null)
            .build();

    log.debug("Reading record saved successfully for bookId: {} and readerId: {}",
            readingRecordCreationRequestDto.getBookId(), readingRecordCreationRequestDto.getReaderId());
    return readingRecordRepository.save(readingRecordEntity);
}
```

MULTUMIM!

333-Team
Enescu Irina-Ștefania, Guță George, Titirigă Tiberiu-Nicolae, Voinea Ana-Maria
(Specializarea Informatică, Grupa 333)