International Journal of Latest Engineering and Management Research (IJLEMR)
ISSN: 2455-4847
www.ijlemr.com || Volume 8– Issue 8 || August 2023 || PP. 50-60

Tools for Effective Java Backend Development

Zdzitavets Aliaksandr Leontevich
Capital.com, Poland, Software Engineer
Bachelor's Degree, Polotsk, Belarus

Abstract: This article provides a comprehensive overview of the key tools and frameworks for effective Java backend development. The entire development lifecycle is covered, discussing the roles and benefits of various types of tools, such as Integrated Development Environments (IDEs), build automation tools, testing frameworks, version control systems, database integration tools, containerization and orchestration tools, API documentation tools, and more. Furthermore, the article explores the prominent Java backend frameworks, including Spring Boot, JavaServer Faces (JSF), and Hibernate, examining their unique strengths and use cases in application development. Emphasizing the significance of these tools in optimizing workflows, leveraging best practices, enhancing collaboration, ensuring code quality, and building reliable and scalable applications, the article underscores the importance of selecting the right tools based on project needs, team expertise, and tool merits. Overall, the article serves as a comprehensive guide for Java backend developers to understand and utilize these tools and frameworks to their fullest extent, thereby enhancing their productivity and the quality of their software development efforts.

Keywords: Java, Backend Development, Integrated Development Environments (IDEs), Eclipse, IntelliJ IDEA, Build Automation Tools, Maven, Gradle, Testing Frameworks, JUnit, Mockito, Version Control Systems, Git, Subversion, Database Integration Tools, Hibernate, Containerization, Docker, Orchestration, Kubernetes, RESTful API Documentation, Swagger, Postman, Spring Boot, JavaServer Faces (JSF), Software Development Tools.

Introduction

In an era marked by an ever-increasing dependence on digital technologies, the role of software development in driving innovation across all sectors cannot be denied. Among the various programming languages, Java remains one of the most popular and versatile choices, especially when it comes to backend development. Renowned for its object-oriented structure, reliability, and scalability, Java is well-suited for various software applications - from the development of complex enterprise systems to simpler web and mobile applications [1].

The power and versatility of Java are the results not only of its inherent capabilities but also of the rich ecosystem of tools and resources it offers. These tools help simplify complex processes, enhance code quality, facilitate collaboration, and boost overall productivity. Developers who can adeptly navigate this tool landscape and effectively leverage their functionalities are better equipped to address the multifaceted challenges posed by contemporary software development.

This article embarks on a comprehensive exploration of tools essential for the efficient development of Java's server-side. We'll examine Integrated Development Environments (IDEs), build and testing tools, continuous integration/continuous deployment (CI/CD) tools, version control systems, and key frameworks.

By shedding light on the tools necessary for efficient server-side Java development, this article aims to assist both novice developers seeking foundational understanding and seasoned ones in need of refreshing knowledge or updates on the latest tools.

Integrated Development Environments (IDE)

At the heart of any effective development process is a competent Integrated Development Environment (IDE). An IDE is more than just a text editor - it's a comprehensive software package that offers tools for writing, running, testing, and debugging code. IDEs are designed to enhance productivity by providing all the tools a developer might need within a single application, reducing the need to switch between different applications and disrupting the workflow.

In Java backend development, choosing the right IDE is paramount for efficient code-writing. A good IDE offers features like intelligent code completion, built-in tools for compiling and running programs, a debugger for detecting and rectifying errors, and integration with build automation and version control tools. Some IDEs also support multiple programming languages, making them a universal tool in a multilingual development environment.

We'll look at some of the most widely-used IDEs in Java server-side development:

1. Eclipse

Eclipse is a highly popular open-source Integrated Development Environment (IDE) that has been in the industry for over a decade. It provides a reliable platform for Java development, offering comprehensive features and an extensive ecosystem of plugins for added functionalities.

Beyond Java, Eclipse supports a variety of programming languages such as C, C++, and Python, with the appropriate plug-ins. It also offers integration with popular version control systems like Git, and build tools such as Maven and Gradle. The IDE comes with a visual interface builder for developing JavaFX and Swing applications, a feature cherished by developers working on GUI-based applications.

One of Eclipse's strengths is its broad customizability. Developers can tailor the IDE to their specific needs, choosing from a plethora of plugins and configuration options. This functionality makes Eclipse a versatile tool that can be adjusted to handle any tasks, from simple Java applications to complex multilingual projects [2].

The code snippet provided below illustrates a simple "Hello World" program written in Java using the Eclipse IDE. Eclipse is widely adopted by Java developers due to its expansive features and plugin ecosystem. Being an open-source platform, it offers a robust suite of tools that facilitate software development.

```java
// Hello World in Eclipse
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

The code starts with declaring a public class named "HelloWorld." Within this class is the main method, which serves as the entry point for the Java application. The main method is declared as static, enabling it to be called without the need to instantiate an object of the class.

Inside the main method is a single line of code: System.out.println("Hello, World!"); This line prints the string "Hello, World!" to the console. The System.out.println operator is used to output text to the standard output stream.

This code serves as a foundational example to showcase the basic structure of a Java program in the Eclipse IDE. It demonstrates the simplicity of displaying a message to the console, which is often used as a starting point for learning Java programming.

Overall, Eclipse provides a conducive environment for Java development, offering features like code editing, debugging, project management, and integration with version control systems. It is widely embraced within the Java community, enabling developers to efficiently create and maintain Java applications.

2. IntelliJ IDEA

IntelliJ IDEA, developed by JetBrains, is another leading development environment for Java. Available in both a free version (Community Edition) and a paid version (Ultimate Edition), IntelliJ IDEA is known for its intelligent features that boost productivity.

IntelliJ IDEA stands out in the area of code completion. The IDE can predict what the developer intends to input, generating accurate suggestions that save time and reduce errors. It also comes with a range of built-in tools, including a high-level debugger, test runner, and a powerful static code analysis tool that can detect potential errors before they cause problems.

In addition to Java, IntelliJ IDEA supports other programming languages, such as Kotlin, Groovy, and Scala. The Ultimate Edition also offers tools for web development and enterprise development, including support for Spring, Hibernate, and various web technologies.

The following code snippets demonstrate the implementation of the "Hello World" program using two popular Java IDEs: IntelliJ IDEA and NetBeans. These IDEs provide powerful features and an intuitive development environment for Java programmers:

```
// Hello World in IntelliJ IDEA
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

The code begins with the declaration of a public class named "Main". Inside the class is the main method, which serves as the entry point to the program. As in the previous example, the main method is declared as static.

The main method contains a single line of code: System.out.println("Hello, World!"); This line prints the string "Hello, World!" to the console. The System.out.println operator is responsible for printing text.

IntelliJ IDEA stands out for its extensive auto-completion and code analysis capabilities. It offers intelligent coding suggestions, helping developers write efficient and error-free code. Its user-friendly interface and comprehensive toolkit make it a preferred choice among Java developers.

3. NetBeans

NetBeans is an open-source IDE that's well-known for its simple and user-friendly interface. Although NetBeans supports several languages like JavaScript, PHP, HTML, and others, it is primarily used for Java development.

NetBeans offers a clean and intuitive workspace, making it an excellent choice for beginners. It comes with a robust code editor, a visual debugger, and built-in support for Maven and Ant for build automation. It also supports Git, Subversion, and Mercurial for version control.

Another advantage of NetBeans is its full support for developing Java desktop applications using Swing. It includes a visual editor for designing graphical interfaces, making the development of desktop applications simpler.

The code snippet below demonstrates a basic "Hello World" program written in Java using NetBeans:

```
// Hello World in NetBeans
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

The code structure in NetBeans is similar to the previous examples. It begins with the declaration of a public class named "Main". Inside this class is the main method, which serves as the program's entry point.

The code System.out.println("Hello, World!"); prints the "Hello, World!" message to the console. This line of code handles the output display.

NetBeans provides a clean and intuitive workspace, making it an ideal choice for beginners. It offers convenient debugging features and seamless integration with popular version control systems. NetBeans offers developers a comfortable development interface.

Overall, both IntelliJ IDEA and NetBeans offer reliable Java development environments equipped with features that facilitate effective coding, debugging, and project management. These IDEs optimize Java development workflows, catering to developers of varying skill levels.

In conclusion, the choice of an IDE often depends on personal preferences, the complexity of the project, and specific development process requirements. All three aforementioned IDEs - Eclipse, IntelliJ IDEA, and NetBeans - are trustworthy tools capable of facilitating efficient Java server-side development.

Build Tools

Following the IDE, the next set of tools critically important for Java server-side development are Build Tools. These are designed to automate the process of compiling, testing, and packaging code into executable formats. While these tasks can be carried out manually, it can be very time-consuming and prone to human errors for larger projects.

Build Tools use scripts, often written in XML or the tool's DSL (Domain-Specific Language), to specify how a project should be built. They handle dependencies, meaning they can automatically download and link libraries that the code depends on. This not only speeds up project setup but ensures that everyone working on the project is using the same version of libraries.

Let's delve into some of the most commonly used Build Tools in Java server-side development:

1. Apache Maven

More commonly known as Maven, Apache Maven is a powerful build automation tool predominantly used in Java projects. It uses an XML file to describe the software project being built, its dependencies on other external modules and components, and the build order.

Maven simplifies the build process by providing a unified build system. It offers several built-in goals for compiling code, packaging it into JAR or WAR, compiling tests, running tests, generating Javadocs, and more. Maven also manages the project's dependencies, automatically downloading the required libraries from a central repository.

One of Maven's best features is its project management capabilities. It can produce a website or PDF, complete with full documentation based on the information in the POM file and source code comments (JavaDoc). It also provides predefined goals for source code management operations and distribution management [3].

Below is an example of a simple Maven configuration file, pom.xml:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

2. Gradle:

While Maven has been the industry standard for many years, Gradle has emerged as a newer and more flexible alternative. Gradle was designed to overcome several of Maven's limitations, such as slower build times and a less flexible XML configuration system.

Gradle uses a Groovy-based DSL for writing build scripts, which is more flexible and easier to understand than Maven's XML configuration files. Gradle scripts, in addition to being more concise, also provide developers with a powerful toolkit for defining custom build logic.

Gradle also offers performance improvements over Maven. It uses a daemon to keep build information in memory, reducing the time needed for subsequent builds. Gradle also supports incremental builds, meaning it only rebuilds parts of the project that have changed since the last build, significantly reducing build times.

Below is an example of a simple build.gradle file:

```
apply plugin: 'java'

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'junit:junit:4.12'
}
```

The choice between Maven and Gradle often depends on the specific needs of a project. While Maven's convention-over-configuration approach might simplify project setup, Gradle's flexibility and performance might be more suitable for larger and more complex projects.

Testing Tools

Testing is an integral part of any software development lifecycle. A reliable set of tests can help identify errors at an early stage, facilitate code changes, and ensure the expected behavior of the software. The Java ecosystem offers a plethora of tools and libraries for testing, from unit testing frameworks to integration testing tools and mock object libraries.

Let's look at some of the most commonly used testing systems:

1. JUnit

JUnit is the de-facto standard for unit testing in Java. It provides a set of annotations and assertions for writing tests, making it easier for developers to write and execute tests. JUnit tests are typically small, test a single method or class, and run quickly, providing immediate feedback when the code is changed [4].

Below is an example of a simple JUnit test:

```
import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculatorTest {
    private Calculator calculator;

    @BeforeEach
    public void setUp() {
        calculator = new Calculator();
    }

    @Test
    @DisplayName("Test addition operation")
    public void testAdd() {
        assertEquals(5, calculator.add(2, 3));
    }
}
```

2. Mockito

While JUnit is excellent for unit testing, there are times when you need to isolate the class being tested from its dependencies. This is where mock object libraries, like Mockito, come into play.

Mockito allows you to create and configure mock objects. Using Mockito, you can create mock objects that replace the real ones, check the number of method calls, and set up stub methods to return specific values.

Below is an example of how Mockito can be used in conjunction with JUnit:

```java
import org.junit.jupiter.api.*;
import org.mockito.Mockito;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class SomeServiceTest {
    private SomeService someService;
    private SomeDependency someDependency;

    @BeforeEach
    public void setUp() {
        someDependency = Mockito.mock(SomeDependency.class);
        someService = new SomeService(someDependency);
    }

    @Test
    @DisplayName("Test someService operation")
    public void testSomeOperation() {
        Mockito.when(someDependency.someMethod()).thenReturn(5);
        assertEquals(5, someService.someOperation());
    }
}
```

3. Testcontainers

For integration tests, when it's essential to check how an application interacts with other systems (such as databases or message brokers), a tool like Testcontainers might be needed. Testcontainers is a Java library that supports JUnit tests by providing lightweight, throwaway instances of common databases, Selenium web browsers, and anything that can run in a Docker container [5].

Testcontainers ensure that tests are not dependent on the environment in which they are executed. They provide a consistent and controlled environment for integration tests, making them more reliable

Below is an example of simple usage of Testcontainers:

```java
import org.junit.jupiter.api.*;
import org.testcontainers.containers.PostgreSQLContainer;
import org.testcontainers.junit.jupiter.Container;
import org.testcontainers.junit.jupiter.Testcontainers;

@Testcontainers
public class SomeRepositoryTest {
    @Container
    public PostgreSQLContainer postgresqlContainer = new PostgreSQLContainer()
            .withDatabaseName("test")
            .withUsername("test")
            .withPassword("test");

    private SomeRepository someRepository;

    @BeforeEach
    public void setUp() {
        someRepository = new SomeRepository(postgresqlContainer.getJdbcUrl(), "test", "test");
    }

    // Your test methods here...
}
```

Choosing the right testing tools and environments for a project can significantly improve the software quality and developers' productivity. The tools mentioned above — JUnit, Mockito, and Testcontainers — have been well-received in the Java community and are an excellent choice for most server-side projects. However, it's important to remember that the "best" tools for any project will depend on the specific needs and constraints of that project.

Version Control Systems:

Version Control Systems (VCS) are another essential set of tools for developers. VCS allows tracking changes made to a project over time.

Let's look at some of the primary control systems:

1. Git:

Currently, Git is the most popular version control system. Developed by Linus Torvalds (the creator of Linux), Git is a distributed version control system. This means that every developer has a complete copy of the project's history on their local computer, making operations like branching and merging extremely fast.

Git is known for its flexibility and power, but it also has a reputation for being challenging to learn. However, once mastered, it offers a reliable set of features suitable for projects of any size and complexity.

2. Subversion:

Before the advent of Git, Subversion (abbreviated as SVN) was the preferred version management system for many projects. Unlike Git, Subversion is a centralized version control system, meaning there is a single central repository containing the entire project's history.

While Subversion is not as powerful and flexible as Git, it's easier to learn and use. It follows a simple model where the latest code is always on the trunk, and features or bug fixes are developed in branches.

Although Git has largely surpassed Subversion in popularity, Subversion remains an excellent choice for version control, especially for teams that prefer a simpler, linear development model.

Continuous Integration/Continuous Deployment (CI/CD) Tools:

Continuous Integration/Continuous Deployment (CI/CD) is a modern software development practice that involves regularly integrating code changes into the central repository (Continuous Integration) and then automatically building and deploying the application in various environments (Continuous Deployment).

The CI/CD practice can help identify errors at an early stage, reduce integration issues, and provide rapid feedback to developers.

Let's look at the most popular tools in this area:

1. Jenkins:

Jenkins is a popular open-source tool that facilitates the CI/CD practice. It offers a plethora of plugins to support the build, deployment, and automation of any project. Jenkins allows continuous building and testing of software projects, making it easier for developers to integrate changes into the project, and for users to get a new build. It also ensures continuous software delivery by providing powerful ways to define build pipelines and integrates with a multitude of testing and deployment technologies.

2. Travis CI:

Travis CI is a hosted distributed continuous integration service used to build and test software projects hosted on GitHub. It is configured by adding a file named .travis.yml to the repository's root directory, which instructs Travis CI on what to do. It's free for public repositories and offers various paid plans for private repositories.

In conclusion, having a robust set of tools is essential for effective Java server-side development. From IDEs that provide a comprehensive environment for writing, running, and debugging code, to build tools that automate the process of compiling, testing, and packaging code, testing tools that ensure the code behaves properly, version control systems that track code changes, to CI/CD tools that automate the process of integrating changes and deploying applications — each category of tools plays a vital role in the software development process. Choosing the right tools can significantly impact the efficiency and productivity of developer teams and, ultimately, the project's success.

Containerization and Orchestration Tools:

As software architecture has shifted towards microservices and with the advent of DevOps practices, containerization and orchestration tools have become an integral part of Java server-side development. They help in creating reproducible development environments and effectively manage and scale applications.

Let's delve into the primary tools for containerization and orchestration:

1. **Docker**:

Docker is a platform that utilizes OS-level virtualization to deliver software in packages called containers. A Docker container encompasses everything required to run an application: the code, runtime, libraries, environment variables, and config files. Containers are isolated from each other and have their own software, libraries, and configuration files; they can interact through strictly defined channels.

Containers ensure a consistent environment across all stages of development, testing, and production, addressing the "it works on my machine" dilemma. Docker is also integrated with many popular CI/CD tools, making it easy to incorporate into your build and deployment pipelines [5].

Below is an example of a Dockerfile for a Java application:

```
FROM openjdk:11
EXPOSE 8080
ADD /target/my-app-1.0.0.jar my-app.jar
ENTRYPOINT ["java","-jar","/my-app.jar"]
```

2. **Kubernetes**:

Kubernetes (often abbreviated as K8s) is an open-source platform for automating the deployment, scaling, and management of containerized applications. It clusters containers that constitute an application into logical units for easy management and discovery.

Kubernetes offers a platform for fault-tolerant deployment of distributed systems. It handles the scaling and failover of applications, provides deployment templates, and much more [6].

RESTful API Documentation Tools:

When developing server APIs, proper documentation methods can save a lot of time during interface integration and later stages of the development lifecycle.

In Java, there are several powerful tools for maintaining API documentation:

1. **Swagger**:

Swagger (also known as OpenAPI) is a machine-readable interface file specification for describing, producing, consuming, and visualizing RESTful web services. Swagger allows developers to interact with the API in a user interface of an isolated environment, which provides an understanding of how the API responds to parameters and options.

A Swagger document is a RESTful API contract that defines endpoints, request/response types, etc. Swagger documents are represented in a structured format, and they can be written in YAML or JSON.

Java server-side developers often use a library called SpringFox, which automates the generation of machine-readable JSON/YAML Swagger definitions for REST APIs written using Spring Boot [9].

2. **Postman**:

Postman is a scalable API testing tool that quickly integrates into the CI/CD pipeline. It allows users to send requests to web servers and receive responses back. Beyond that, Postman is a tool that deals with API development from start to finish. It enables the creation, testing, and modification of API interfaces while also effectively documenting them.

Each of these tools and frameworks brings something unique, all serving to make Java server-side development more efficient, more manageable, and ultimately more successful. Whether it's the organization and automation provided by build tools, code safety systems created by testing tools, the advantages of Git version management, optimized pipelines created by Jenkins, the flexibility and isolation of Docker, or the power and scalability of Kubernetes – all play a significant role [10].

Frameworks:

Frameworks are essential tools for Java server-side developers. They provide a predefined structure and a set of features that simplify the development process and help avoid writing boilerplate code. They can be incredibly useful in terms of increasing productivity, applying best practices, ensuring scalability, and ensuring security.

Let's look at some of the key frameworks used in Java server-side development:

1. Spring Boot:

Spring Boot is one of the most popular tools for backend development in Java. It is a project built on the Spring Framework. It simplifies the bootstrapping and development of new Spring applications by providing default behavior and automatically configuring the application.

Spring Boot assists in creating stand-alone, production-grade applications that "just run", typically on an embedded instance of Tomcat, Jetty, or Undertow. It adopts a confident view of the Spring platform, which can quickly launch an application and also offers flexibility for customization if needed [8].

Below is a basic example of a RESTful web service in Spring Boot:

```
import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.web.bind.annotation.*;

@RestController
@EnableAutoConfiguration
public class Example {

    @RequestMapping("/")
    String home() {
        return "Hello, World!";
    }

    public static void main(String[] args) throws Exception {
        SpringApplication.run(Example.class, args);
    }
}
```

2. JavaServer Faces (JSF)

JavaServer Faces (JSF) is a Java specification for building component-based user interfaces for web applications. It also includes a set of APIs for representing user interface components and managing their state, handling events and input validation, defining page navigation, as well as supporting internationalization and accessibility.

JSF simplifies the creation of user interfaces by assembling reusable components on the page. Developers can also create their own components for specific use cases. JSF provides a robust architecture for managing component states, processing component data, validating user input, and handling events.

3. Hibernate ORM

Hibernate ORM is a Java framework that simplifies the development of Java applications to interact with databases. It provides a foundation for mapping an object-oriented domain model to a relational database, allowing applications to interact with databases using standard SQL without writing a lot of repetitive standard JDBC code.

Hibernate implements the Java Persistence API (JPA), the standard Java API for object-relational mapping. In addition to its own "native" API, Hibernate is also an implementation of the Java Persistence API (JPA) specification. This makes Hibernate potentially compatible with any other ORM tool that implements JPA [7].

Below is an example of using Hibernate to perform an operation with a database:

```
import org.hibernate.Session;
import org.hibernate.Transaction;

public class App {
    public static void main(String[] args) {

        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            // start a transaction
            transaction = session.beginTransaction();

            // save the student object
            Student student = new Student("John", "Doe", "john.doe@example.com");
            session.save(student);

            // commit transaction
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }
}
```

The choice of a framework should depend on the specific needs of the project. Some projects might benefit from the simplicity and convention-over-configuration philosophy of Spring Boot, while others might need the user interface component model provided by JSF or the object-relational mapping of Hibernate. Each of these frameworks offers something unique, and understanding their strengths and weaknesses can help choose the right tool for the job.

Conclusion

In this article, a range of tools was reviewed that can enhance the efficiency and effectiveness of a Java backend developer. These tools span the entire development lifecycle, and each tool serves a specific purpose: from integrated environments that serve as the primary workspace to build automation tools, testing environments, version control systems, database integration tools, containerization and orchestration tools, and API documentation tools.

In closing, it's worth noting that the tools and technologies chosen by a Java backend developer or a development team can significantly influence the quality and efficiency of software engineering. It's crucial to make an informed choice based on the project's needs, the team's experience, and the pros and cons of each tool. A well-equipped Java backend developer not only writes good code but also knows how to use the right tools to optimize the workflow, collaborate with other team members, ensure code quality, and optimize software application delivery.

References

[1]. Malikova Z. T., Kaarov Y. A. Review of the best designers for creating websites // Izvestiya OshTU. 2022. No. 1.
[2]. Oracle Java documentation, 2023 [Electronic resource], Access mode: https://docs.oracle.com/javase/
[3]. Apache Maven project. Apache Maven registration, 2023 [Electronic resource], Access mode: https://maven.apache.org/guides/
[4]. JUnit User Manual 5. JUnit documentation, 2023 [Electronic resource], Access mode: https://junit.org/junit5/docs/current/user-guide/

[5]. Docker documents. Docker documentation, 2023 [Electronic resource], Access mode: https://docs.docker.com/
[6]. Kubernetes. Kubernetes Documentation, 2023 [Electronic resource], Access mode: https://kubernetes.io/docs/
[7]. Switch to sleep mode. Hibernate ORM documentation, 2023 [Electronic resource], Access mode: https://hibernate.org/orm/documentation/
[8]. Dinesh Rajput. Spring Boot 2.0 Update: Create improved production and distributed systems with Spring Boot. // Packt Publishing Ltd, 2018. 390 p.
[9]. Swagger. Swagger documentation, 2023 [Electronic resource], Access mode: https://swagger.io/docs/
[10]. Postman Training Center. Postman's documentation, 2023 [Electronic resource], Access mode: https://learning.postman.com/docs/