МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ЗАТВЕРДЖЕНО
на засіданні кафедри
інформаційних систем
Протокол № 1 від 22.08.2023 р.

ПОГОДЖЕНО
Проректор з навчально-методичної роботи

_____ Каріна НЕМАШКАЛО

СУЧАСНІ ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

робоча програма навчальної дисципліни (РПНД)

Галузь знань	**12 "Інформаційні технології"**
Спеціальність	**121 "Інженерія програмного забезпечення"**
Освітній рівень	**перший (бакалаврський)**
Освітня програма	**"Інженерія програмного забезпечення"**

Статус дисципліни	**вибіркова**
Мова викладання, навчання та оцінювання	**англійська**

Розробник:
к.т.н., доцент підписано КЕП Андрій ПОЛЯКОВ

Завідувач кафедри
інформаційних систем _____ Дмитро БОНДАРЕНКО

Гарант програми _____ Олег ФРОЛОВ

Харків
2024

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
SIMON KUZNETS KHARKIV NATIONAL UNIVERSITY OF ECONOMICS

APPROVED
at the meeting of the department
information systems
Protocol No. 1 of 22.08.2023

AGREED
Vice-rector for educational and methodological work

_____ Karina NEMASHKALO

MODERN TECHNOLOGY OF PROGRAMMING

Program of the course

Field of knowledge	**12 "Information technologies"**
Specialty	**121 "Software engineering"**
Study cycle	**first (undergraduate)**
Study programme	**"Software Engineering"**
Course status	**elective**
Language	**English**

Developer:
Ph.D. (Technical sciences),
associate professor digital signature Andrii POLIKOV

Head of Information systems
department:
Ph.D. (Technical sciences),
associate professor _____ Dmytro BONDARENKO

Head of Study Programme:
Ph.D. (Technical sciences),
associate professor _____ Oleg FROLOV

Kharkiv
2024

INTRODUCTION

Modern Programming Technologies is at the forefront of programming innovation, providing students with the latest technologies and trends. In today's world, programming is based on a continuous process of learning and adapting to change, and this discipline is designed to prepare students for such a dynamic environment. The course is based on working with RESTful APIs, peculiarities of virtualisation and containerisation technology using Docker, using AWS cloud services, the basics of working with CI/CD processes, and the practice of implementing the infrastructure-as-code concept through Terraform. Students acquire not only theoretical knowledge, but also apply it in practice, working in real environments. This allows students to understand how modern software systems function and effectively integrate into the professional field of programming. The discipline provides skills and knowledge that will help you navigate the dynamic world of modern programming technologies in the future.

Studying the discipline "Modern Programming Technologies" involves mastering the skills of development and design using modern technologies such as RESTful API, virtualisation and containerisation through Docker, using AWS cloud services, working with CI/CD processes and implementing the principles of "infrastructure as code" through Terraform. It allows to train specialists capable of working effectively in a modern software environment.

The purpose of teaching the course "Modern Programming Technologies" is to develop students' theoretical knowledge and practical skills in the field of modern programming technologies and methods. This includes the ability to develop and apply highly efficient programming technologies, as well as create, design, and manage systems based on microservice architecture. Higher education applicants must master the skills of using RESTful APIs, virtualisation, and containerisation technologies (Docker), cloud services (AWS), CI/CD technologies, working with databases and implementing the concept of "infrastructure as code". Working with the training material allows you to develop the skills of independent work, critical thinking, teamwork and presentation of your own ideas and software developments.

Tasks of the course are:

- developing a modern view of software development processes using the latest technologies and practices;
- to get acquainted with the principles of operation and specifics of RESTful API development;
- learn the basics of virtualisation and containerisation of applications using Docker;
- learn the CI/CD processes and their importance in modern software development;
- learn the principles of working with cloud services on the example of AWS and developing serverless applications;
- learn infrastructure as code on the example of Terraform;
- acquire teamwork skills and use of version control systems;

- acquire critical thinking and professional skills necessary for continuous self-improvement in the field of IT.

The subject of the discipline is the study of the basic principles, approaches, and methods of designing, developing and testing modern software systems, integrating various system components, using cloud technologies and serverless architectures, as well as implementing a continuous integration and delivery cycle.

The subject of the discipline is modern methods, technologies and tools used in the design, development, release and support of software products and systems. This includes components such as RESTful APIs, Docker, CI/CD processes, developing basic infrastructure objects through code using Terraform, using AWS cloud services and implementing the serverless concept.

The learning outcomes and competences formed by the course are defined in Table 1.

Table 1

Learning outcomes and competences formed by the course

Learning outcomes	Competences
LO07	GC05, SC13
LO12	SC03, SC14
LO17	GC02, SC12
LO18	GC06, SC12
LO21	GC02, SC08

where, LO07. To know and to apply in practice the fundamental concepts, paradigms and basic principles of functioning of language, tool and computing software engineering tools.

LO12. Apply effective software design approaches in practice.

LO17. Be able to apply component-based software development methods.

LO18. Know and be able to apply information technologies for data processing, storage and transmission.

LO21. To know, analyze, select, and competently apply information security (including cyber security) and data integrity tools in accordance with the applied tasks and software systems being developed.

GC02. Ability to apply knowledge in practical situations.

GC05. Ability to learn and master modern knowledge.

GC06. Ability to search, process and analyze information from various sources.

SC03. Ability to develop architectures, modules and components of software systems.

SC08. Ability to apply fundamental and inter disciplinary knowledge to successfully solve software engineering problems.

SC12. Ability to carry out the system integration process, apply change management standards and procedures to maintain the integrity, overall functionality and reliability of the software.

SC13. Ability to reasonably choose and master tools for software development and maintenance.

SC14. Ability to think algorithmically and logically.

COURSE CONTENT

Content module 1. **Designing, developing, and managing RESTful service APIs.**

Topic 1. **Development of RESTful services, their limitations and design.**

1.1. Introduction to the discipline. Control measures and learning outcomes.

1.2. What is an API? The evolution of the REST/JSON API.

1.3. Introduction to RESTful APIs, private, public and partner APIs, API value chain.

1.4. Introduction to the limitations of the REST architecture.

Topic 2. **REST API templates.**

2.1.REST API error handling templates.

2.2.REST API version control templates: change processing, API version control, implementation of ACME API version control.

2.3.REST API Cache Management Pattern: design and concept of the cache API, cache management directive, implementation of the cache API using cache management directives.

2.4.REST API response data processing template: partial responses and their implementation, pagination, and their implementation.

Topic 3. **REST API security and OpenAPI specification v3.1.0, Swagger, API management.**

3.1.API protection with basic authentication, tokens and JWT, API key and secret, API authorisation using OAuth2.0, functional attacks.

3.2. Requirements analysis process, introduction to REST and Swagger/OAI specifications.

3.3. Implementation of API governance, API lifecycle, API developer portal, API security/traffic management, API Analytics, API sales, and API monetisation.

Content module 2. **Fundamentals of virtualisation and containerisation of applications and services.**

Topic 4. Fundamentals of virtualisation and containerisation. Fundamentals of the Docker platform.

4.1. Containerisation and virtualisation, introduction to Docker, Docker architecture, containers, and images, docker cli, container management, lifecycle.

4.2. Basics of creating images (Dockerfile): commands FROM; WORKDIR, COPY, ADD; RUN; ENV, LABEL, USER; VOLUME and EXPOSE; VOLUME and EXPOSE; VOLUME and EXPOSE.

4.3. Images for production environments: parameterising Docker files with ARGs, creating and running reusable images, build time vs. runtime, creating smaller images, multi-stage image creation, creating a base image from scratch, application servers to run, hardware for production-level databases, implementing a proxy, the need for automation.

Topic 5. **Basics of working with data in a container and image repository.**

5.1. Docker volumes: storage issues, volume types.

5.2. Docker registries, setting up the local Docker registry, Docker HUB, searching for and running images.

Topic 6. **Orchestration of containers. docker composer.**

6.1. Running containers in Docker: Running production containers in Docker, Docker Compose basics.

6.2. Docker Compose File: version and volumes, networks, services, image management, and application lifecycle with Docker Compose.

6.3. Docker Swarm: Swarm architecture and services, preparing Swarm with Docker Machine, autonomous containers and services in Swarm, service modes and login routing grid, application stack in Swarm, environment and application lifecycle in Swar, brief description of Docker Runtime Environment.

Content module 3. **Designing and developing CI/CD processes.**

Topic 7. **The concept of CI. Jenkins.**

7.1. The concept of continuous integration, delivery and deployment (CI/CD).

7.2.7.2. Jenkins architecture: configuration and setup, plugins, security, backup, secret storage.

7.3. Jenkins Job: Job configuration, Jenkins scripts, Job parameterisation, logical input in Jenkins Job, continuous integration (CI).

7.4. Pipeline in Jenkins: declarative and scripted pipelines.

Topic 8. **Types of deployment. Jenkins Pipeline.**

8.1. Deployment: Blue/Green, Canare, A/B.

8.2. Continuous deployment (CD) with Jenkins. Infrastructure as code.

8.3.8.3. Jenkins DSL operation, Jenkins as a code pipeline, Distributed builds in Jenkins, Jenkins integration with Docker, Jenkins security aspects.

Content module 4. **Cloud technologies and Cloud-native applications. Infrastructure as Code (IaC).**

Topic 9. **Introduction to AWS, identity and access management, AWS Computing.**

9.1. Getting started with AWS Cloud: What is AWS? AWS global infrastructure. Interaction with AWS. Creating an AWS account

9.2. Security in AWS: Security and the AWS shared responsibility model. Protecting the AWS root user. Approaches to protecting your AWS account.

9.3. Identity and access management in AWS: An introduction to identity and access management in AWS. Role-based access in AWS. Demonstration of AWS IAM. Hosting an employee directory application on AWS. Default Amazon Machine Image (AMI) for Amazon EC2.

9.4.AWS Compute: Computing as a Service on AWS. Introduction to Amazon Elastic Computing Cloud. The life cycle of an Amazon EC2 instance. Container services on AWS.

Topic 10. **Topic 10. AWS networking, storage, and databases.**

10.1. Working in the AWS network: Networking on AWS. Introduction to Amazon VPC. Amazon VPC routing. Protecting your network with Amazon VPC Security. Hybrid connectivity with AWS.

10.2. 1Create a VPC and restart the enterprise directory application on Amazon EC2.

10.3. Types of storage on AWS: Amazon EC2 Instance Storage and Amazon Elastic Block Store. Object storage with Amazon S3. Choosing the right service for data storage. Create an Amazon S3 Bucket.

10.4. Databases on AWS. Study of databases on AWS. Amazon's relational database service. Specialised databases on AWS. Introduction to Amazon DynamoDB. Choosing the right AWS database service.

Topic 11. **Infrastructure as a Code (IaC). Terraform.**

11.1. What is Infrastructure as a Code (IaC)? The advantages of infrastructure as code, the basics of Terraform. Comparison of Terraform with other IaC tools.

11.2. Working with Terraform. Preparing your AWS account. Installing Terraform. Deploying a single server, a single web server, a configured web server, a cluster of web servers, a load balancer. Removing unnecessary resources.

11.3. Reusing infrastructure with Terraform modules: what is a module? Input parameters of the module, local and output variables of the module. Features of using modules. Version management.

11.4. Working with Terraform: loops, conditional expressions, deployment, and features of use: loops, conditional expressions, zero-downtime deployment.

Topic 12. **Serverless applications. Introduction to AWS Lambda.**

12.1. Cloud application factors.

12.2. AWS CLI and API. Introduction to the AWS API Management Console CLI SDK. Introduction to the AWS CLI. Cloud9, AWS API, AWS CLI. Exploring the AWS SDK (Java). Using temporary credentials in AWS Cloud9. Serverless application model. AWS toolkit for IntelliJ. Cloud9 Temporary Credentials, AWS SDK, AWS Toolkits, AWS SAM - Java. Setting up and exploring the SDK

12.3. Introduction to API-based design. API-based development. What is an API gateway? Dragon API: API gateway terminology. What is API Gateway Notes, API Driven Dev Notes. Models and mappings. Creating a GET API using integration simulation. Publishing an API. Using Postman to create requests. Stages of the API gateway, deployment, invocation, Postman.

12.4. API authentication. Introduction to authentication and API gateway. Access control to the API gateway. Authentication and authorisation of the API gateway. Introduction to Amazon Cognito. Using Amazon Cognito to sign in and invoke the API Gateway. Using Amazon Cognito to log in and invoke an API gateway using JavaScript.

12.5. Serverless computing and lambda. Introduction to AWS Lambda. Executing AWS Lambda. AWS Lambda permissions. Introduction to Lambda, Lambda execution, Lambda permissions. Triggers, Push, Pull model. Reusing the context of a lambda execution. AWS Lambda compliance. Asynchronous and synchronous responses.

Aliases and versions. Creating an AWS lambda function in Java. Creating and debugging a lambda using the AWS Toolkit for IntelliJ. Creating a lambda function, version and aliases.

The list of laboratory classes in the course is given in Table 2.

Table 2

The list of laboratory studies

Name of the topic and/or task	Content
Topic 1 – 3	Design and development of RESTful APIs using Swagger and modern frameworks
Topic 4 – 6	Exploring containerisation tools and creating an application image
Topic 7, 8	Building a CI/CD automation process
Topic 9 – 11	Infrastructure development in AWS with Terraform.
Topic 12	Cloud-native application development in AWS

The list of independent work in the course is given in Table 3.

Table 3

List of self-studies

Name of the topic and / or task	Content
Topic 1 – 12	Study of lecture material
Topic 1 – 12	Preparing for laboratory classes
Topic 1 – 12	Preparing for the current control work
Topic 1 – 12	Preparing for the exam

The number of hours of lectures and laboratory studies and hours of self-study is given in the technological card of the course.

TEACHING METHODS

In the process of teaching the course, the following teaching methods are used to achieve certain learning outcomes and intensify the educational process

Verbal (lecture (Topics 1, 2, 4, 7, 9, 12), problem lecture (Topics 1 - 3, 7, 8), lecture-visualisation (Topics 1 - 12)).

In person (demonstration (Topics 1 - 12)).

Laboratory work (Topics 1 - 12), case method (Topics 1 - 3, 6, 8, 10, 11).

FORMS AND METHODS OF ASSESSMENT

The University uses a 100-point cumulative system for assessing the learning outcomes of students.

Current control is carried out during lectures, laboratory classes and aims at checking the level of readiness of the student to perform a specific job and is evaluated by the amount of points scored:

– for courses with a form of semester control as an exam: maximum amount is 60 points; minimum amount required is 35 points.

The final control includes current control and an exam.

Semester control is carried out in the form of a semester exam.

The maximum number of points that a student of higher education can receive during the examination (examination) is 40 points. The minimum amount for which the exam is considered passed is 25 points.

The final grade in the discipline is determined by summing the points for the current and final control.

During the teaching of the discipline, the following control measures are used:

Current control: defence of laboratory works (32 points), current control works (10 points), presentations (18 points).

Semester control: Exam (40 points)

More detailed information on the assessment system is provided in technological card of the course.

An example of an exam card and assessment criteria.

Semyon Kuznets Kharkiv National University of Economics
First (bachelor) level of higher education
Specialty "Software Engineering"
Educational program "Software engineering"
Term II
Навчальна дисципліна «Modern technology of programming»

EXAM CARD № 1

Task 1. Test.

Question No. 1	
Which of the following is true about AWS	
Choose one of 4 answers:	
1)	EC3 is a provider of analytics as a service
2)	Amazon Elastic Cloud - a system for creating virtual disks
3)	SimpleDB works with Amazon EC2 and Amazon S3
4)	None of the above

Question No. 2	
Assume that a subnet is created and an EC2 instance is running on it with the default settings. Which of the following options will be ready to use in the EC2 instance as soon as it is launched?	
Choose one of 4 answers:	
1)	Elastic IP
2)	Public IP
3)	Private IP
4)	Internet Gateway

Question No. 3	
In the Swagger definition, what defines each endpoint in a section that contains all endpoints?	
Choose one of 4 answers:	
1)	absolute URL
2)	query string
3)	relative URL
4)	HTTP method

Question No. 4	
The Docker container is often described as an improvement over some other technology?	
Choose one of 4 answers:	
1)	virtual machines
2)	cloud computing
3)	DevOps
4)	microservices

Question No. 5	
Which sentence best describes Docker?	
Choose one of 4 answers:	
1)	None of the above
2)	Once built, run everywhere
3)	Easy to operate - much more enjoyable
4)	Build once, run twice

Question No. 6	
What HTTP verb is usually used to update or create a resource in the API?	
Choose one of 4 answers:	
1)	POST
2)	WRITE
3)	SUBMIT
4)	CREATE

Question No. 7	
What is one of the advantages of GraphQL over REST approaches??	
Choose one of 4 answers:	
1)	more secure by default
2)	flexibility of requests/responses
3)	compatibility with a large number of gateways
4)	more stable APIs

Question No. 8	
What are the benefits of auto-scaling?	
Choose one of 4 answers:	
1)	Fault tolerance
2)	Better availability
3)	Better cost management
4)	All of the above

Question No. 9	
_____ are instances of Docker images that can be run with the command Docker run	
Choose one of 4 answers:	
1)	Container
2)	File
3)	Cloud
4)	Hub

Question No. 10	
What types of query input parameters can be used in Swagger?	
Choose one of 4 answers:	
1)	Use 'host' for the host input parameter, 'port' for the port input parameter, 'method' for the method input parameter.
2)	There are no input query parameters in the Swagger definition.
3)	Use 'path' for the URL input parameter, 'query' for the query string input parameter, 'body' for the query body input parameter
4)	Use 'url' for the URL input parameter, 'querystring' for the query string input parameter, 'content' for the query body input parameter

Task 2.

Working with containerised applications

1. Deploying the container:

a. Upload the image httpd:2.4.57-alpine to the local repository. (Take a screenshot of the process and the result, give the command).

b. Display the list of available images on the docker server (Take a screenshot of the result)

c. Create a container from the image httpd:2.4.57-alpine, name it with the first letter of your full name and run it. (Run the command, take a screenshot of the result)

d. Print the list of executable containers. (Run the command, take a screenshot of the result)

e. Connect to the container and examine it (bash or sh. Run commands, take a screenshot of the result):

i. display information about the version and name of the operating system from the files (/etc/*release);

ii. get the name of the container (hostname);

iii. get information on the apach version (httpd -v).

f. Stop the container and display a list of all containers. (Run the command, take a screenshot of the result).

g. Restore the container. (Give the command, take a screenshot of the result)

2. Service deployment:

a. Create another container from the httpd:2.4.57-alpine image with the configuration to forward port 80 from the container to 8083 of the host OS. (Take a screenshot of the process and the result, run the command)

b. Check in your browser that the nginx service is available at http://localhost:8083 or from the host console with the command "curl http://localhost:8083"

Approved at the meeting of the Department of Information Systems
protocol No. ____ from «___»_____20___p.

Examiner _____ Ph.D., associate professor, Andrii POLIKOV

Head
of the Department _____ Ph.D., associate professor. Dmytro BONDARENKO

Evaluation criteria

The final score for the exam is the sum of the scores for all tasks, rounded to the nearest whole number according to the rules of mathematics.

The algorithm for solving each task includes separate stages that differ in complexity, labour intensity and importance for solving the task. Therefore, individual tasks and stages of their solution are evaluated separately from each other as follows:

Task 1 (test).

The first question is dedicated to testing theoretical knowledge of the discipline. The test includes 10 questions worth 1.5 points each. A total of 15 points. The test lasts 20 minutes.

Task 2 (heuristic).

The second question is devoted to the development of the application deployment process using the Docker containerised environment. Configuration files are written using the YAML language and bash commands. A set of commands is being developed that allows you to manage the life cycle of containers with an application and monitor the environment. The student must use the Visual Code, Docker, Linux and bash environments. In this case, the applicant is allowed to use existing reference literature. After checking the application, the applicant receives K1 points according to the following requirements (Table 4).

Table 4

Assessment criteria for the diagnostic task

The second task is scored from 0 to 25 points. The number of points for the second task is determined in accordance with the following scale:

Scores K_1	**Requirements**
25	The task has been completed in full.
24	The task is completed in full. There are small comments on the organisation of the code structure and interface.
22 – 23	The task is mostly completed. There are comments on the organisation of the code structure and the organisation of the deployment environment.
20 – 21	The task is complete, but not in full. The container project is working, but some of the requirements or capabilities for the deployment environment specified in the assignment have not been implemented.
18 – 19	The task is complete, but not in full. The container design is working, but two or three requirements or capabilities of the deployment environment specified in the assignment are not met.
16 – 17	The task is complete, but not in its entirety. The container design is working, but four or five of the requirements or capabilities for the deployment environment specified in the assignment are not met.
11 – 15	The task is complete, but not in its entirety. The container design is working, but more than five requirements or capabilities of the deployment environment specified in the task are not met.
7 – 10	The container design at least enables one deployment environment functionality to be performed correctly.
3 – 6	The container runs, but contains gross errors, and none of the deployment environment requirements specified in the assignment are fully met.
2	The container or environment does not meet the task statement.
1	The design of the container or environment does not contain any code developed by the student.
0	No programme available.

RECOMMENDED LITERATURE

Main

1. Varanasi B. and others Spring REST: Building Java Microservices and Cloud Applications / B. Varanasi, M. Bartkov. — Berkeley, CA : Apress, 2022. — 251 p.

2. Gkatziouras E. A Developer's Essential Guide to Docker Compose Simplify the Development and Orchestration of Multi-Container Applications / E. Gkatziouras. — Birmingham : Packt Publishing, Limited, 2022. — 264 p.

3. Nickoloff J. and others Docker in action / J. Nickoloff, S. Kuenzli, B. Fisher. — Shelter Island, NY : Manning Publications Co, 2019. — 310 p.

4. Laster B. Jenkins 2: Up and Running: Evolve Your Deployment Pipeline for Next Generation Automation / B. Laster. — Sebastopol, CA : O'Reilly Media, 2018. — 577 p.

5. Culkin J. and other. AWS cookbook: recipes for success on AWS / J. Culkin, M. Zazon. — Beijing Boston Farnham Sebastopol Tokyo : O'Reilly, 2021. — 330 p.

6. Winkler S. and others Terraform in action / S. Winkler, A. Dadgar. — Shelter Island, NY : Manning, 2021. — 380 p.

Additional

7. Walls C. Spring in action / C. Walls. — Shelter Island, NY : Manning Publications Co, 2022. — 492 p.

8. Leszko R. Continuous delivery with Docker and Jenkins: delivering software at scale / R. Leszko. — Birmingham Mumbai : Packt Publishing, 2017. — 309 p.

9. Wittig M. and others Amazon Web Services in action / M. Wittig, A. Wittig, B. Whaley. — Shelter Island, NY : Manning, 2019. — 497 p.

10. Wadia Y. and others Mastering AWS Lambda. Learn how to build and deploy serverless applications / Y. Wadia, U. Gupta. — Birmingham : Packt Publishing, Limited, 2017. — 296 p.

11. Brikman Y. Terraform: up & running: writing infrastructure as code / Y. Brikman. — Beijing Boston : O'Reilly, 2021. — 341 p.

Information resources

12. API Documentation & Design Tools for Teams | Swagger [Електроний ресурс]. — Режим доступу: https://swagger.io/.

13. About Swagger Specification | Documentation | Swagger [Електроний ресурс] — Режим доступу: https://swagger.io/docs/specification/about/.

14. Docker: Accelerated Container Application Development [Електроний ресурс] — Режим доступу: https://www.docker.com/.

15. Overview of get started [Електроний ресурс] // Docker Documentation. — Електрон. дані. — Режим доступу: https://docs.docker.com/guides/get-started/.

16. Jenkins [Електроний ресурс] // Jenkins. — Електрон. дані. — Режим доступу: https://www.jenkins.io/.
17. User Handbook Overview [Електроний ресурс] // User Handbook Overview. — Електрон. дані. — Режим доступу: https://www.jenkins.io/doc/book/getting-started/.
18. Terraform by HashiCorp [Електроний ресурс] // Terraform by HashiCorp. — Електрон. дані. — Режим доступу: https://www.terraform.io/.
19. Build a Serverless Web Application with AWS Lambda, Amazon API Gateway, AWS Amplify, Amazon DynamoDB, and Amazon Cognito [Електронний ресурс] // Amazon Web Services, Inc. — URL: https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/.
20. Cloud Computing Services - Amazon Web Services (AWS) [Електронний ресурс]. – Режим доступу: — URL: https://aws.amazon.com/.
21. Welcome to AWS Documentation [Electronic resource] [Електронний ресурс]. – Режим доступу: — URL: https://docs.aws.amazon.com
22. Сучасні технології програмування (6.04.121) [Електронний ресурс] / Розробники Андрій Поляков, Олег Фролов // Персональні навчальні системи ХНЕУ ім. С. Кузнеця — Електрон. дані. — Режим доступу: https://pns.hneu.edu.ua/course/view.php?id=8632.