Силабуси проходять валідацію у Kharkiv IT Cluster

KHARKIV IT CLUSTER

Прізвище, ім'я та по батькові розробника, вчене звання, науковий ступінь, посада: __ Ніщеменко Дмитро Олександрович, викладач_____
_____

Назва закладу освіти:_ Державний університет інформаційно-комунікаційних технологій_____

Шифр та назва спеціальності	F2 Інженерія програмного забезпечення
Назва освітньої програми	Технології цифрового розвитку
Назва дисципліни	**Конструювання програмного забезпечення Java**
Вид (основна, вибіркова)	Основна
Блок дисципліни (залишити блок, що відповідає дисципліні)	1. Алгоритмізація та програмування (зокрема, різні парадигми: структурне, ООП)
Кількість студентів (поточний рік)	85
Курс/Семестр	3/5,6
Навчальний рік валідації	2025/2026

ЗАГАЛЬНА ІНФОРМАЦІЯ ПРО ДИСЦИПЛІНУ

Анотація	Дисципліна «Конструювання програмного забезпечення Java» є фундаментальним курсом, що закладає основи професійного підходу до розробки програмних систем. Протягом першого семестру студенти переходять від вивчення базових синтаксичних конструкцій мови Java до глибокого занурення в парадигму об'єктно-орієнтованого програмування. Основний акцент робиться на засвоєнні ключових принципів проєктування SOLID та практичному застосуванні найпоширеніших патернів проєктування (GoF), що дозволяє створювати гнучкий, масштабований та підтримуваний код. Другий семестр курсу має виражену практичну спрямованість і побудований навколо розробки єдиного, цілісного проєкту. Студенти застосовують отримані в першому семестрі теоретичні знання для побудови сучасного REST API за допомогою Spring Framework. Послідовно, від лекції до лекції, проєкт збагачується новими шарами та технологіями, зокрема Spring MVC для веб-шару, Spring Data JPA для роботи з базами даних, та Spring Security для забезпечення безпеки. Такий підхід імітує реальний процес розробки в IT-компаніях.

Силабуси проходять валідацію у Kharkiv IT Cluster

KHARKIV IT CLUSTER

	У результаті вивчення дисципліни випускники набувають комплексних навичок, необхідних для позиції Junior Java Developer. Вони не лише володіють мовою Java та фреймворком Spring, але й розуміють принципи побудови якісної архітектури, вміють проєктувати системи з урахуванням майбутніх змін, писати тести, працювати з базами даних та контейнеризувати додатки за допомогою Docker. Курс формує міцний фундамент для подальшого професійного зростання в галузі інженерії програмного забезпечення.
Мета навчальної дисципліни	Сформувати у студентів фундаментальне розуміння принципів якісного проєктування ПЗ та навчити застосовувати базові патерни проєктування на мові Java.
Типи занять та контролю	Лекції, лабораторні роботи практичні роботи, самостійна робота. Підсумковий контроль – іспит

Загальний обсяг (кредитів)	**6**	**Лекції (занять)**	**18**	**Лабораторні (занять)**	**18**	**Практичні (занять)**	**18**	**Самостійна (годин)**	**72**

Попередні дисципліни	1. Аналіз вимог до програмного забезпечення 2. Основи інженерії програмного забезпечення

МАТЕРІАЛЬНО-ТЕХНІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДИСЦИПЛІНИ

- **Мультимедійний проектор.**
- **Комп’ютерний клас для проведення практичних занять.**
- **Програмне забезпечення для занять в аудиторії: Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse), Docker.**

СТРУКТУРА ТА ЗМІСТ ДИСЦИПЛІНИ

№	**Теоретична складова** *Назва, перелік питань або анотація лекції*	**Годин**	**Практична складова** *Опис та приклад завдання, а також посилання на методичні матеріали*	**Годин**	**Інструменти, засоби та технології**
Тема 1 – ВСТУП ДО МОВИ JAVA ТА ОСНОВИ ООП					
1	**Лекція 1.** Основи платформи Java та синтаксис мови. Що таке Java: JVM, JRE, JDK. Принцип "Write Once, Run Anywhere". Версіонування в Java: огляд ключових версій. Структура Java-програми. Компіляція та запуск.	2	**Практична робота №1.** Робота з базовими конструкціями мови. Завдання: 1. Налаштувати робоче середовище (JDK, IntelliJ IDEA). Рекомендована версія для лабораторних робіт: **Java 21 (LTS)**	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

	Базовий синтаксис: змінні, примітивні типи даних, оператори. Керуючі конструкції: if-else, switch-case. Цикли: for, while, do-while. Масиви.		2. Створити та запустити програму "Hello, World!". 3. Написати програму-калькулятор для простих арифметичних операцій (+, -, *, /) з двома числами. 4. Створити програму, яка знаходить максимальний елемент у заданому масиві чисел. **Лабораторна робота №1.** Алгоритмічні задачі з використанням циклів та масивів. Завдання: 1. Написати програму для сортування масиву чисел методом "бульбашки" (Bubble Sort). 2. Реалізувати програму, яка перевіряє, чи є введений рядок паліндромом. 3. Написати функцію, що обчислює факторіал числа з використанням циклу. 4. Вивести на консоль прості числа в діапазоні.		
2	**Лекція 2.** Об'єктно-орієнтоване програмування в Java. Основні поняття: клас, об'єкт, екземпляр. Інкапсуляція: модифікатори доступу (public, private, protected), гетери та сетери. Конструктори: їх призначення, перевантаження конструкторів. Ключове слово this. Поняття null. Статичні члени класу (static): поля та методи. Анотації та їх роль (Annotations). Функціональні інтерфейси (@FunctionalInterface). Лямбда-вирази, синтаксис та використання.		**Практична робота №2.** Створення та використання класів. Завдання: 1. Створити клас Student з полями: name, age, major. 2. Інкапсулювати поля класу Student, додавши приватні модифікатори та публічні гетери/сетери. 3. Додати в клас Student конструктор, що ініціалізує всі поля. 4. Створити декілька екземплярів класу Student у main методі та вивести інформацію про них. 5. Додати в клас статичне поле studentCount для підрахунку кількості створених об'єктів. **Лабораторна робота №2.** Взаємодія між об'єктами. Завдання: 1. Створити клас Book з полями title, author, year.	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

			2. Створити клас Library, який містить масив об'єктів Book. 3. Реалізувати в класі Library метод для додавання нової книги. 4. Використати функціональний інтерфейс: Реалізувати в Library універсальний метод findBooks(Predicate<Book> filter), який приймає умову у вигляді лямбда-виразу і повертає список книг, що задовольняють цій умові.		
Тема 2 – ПРИНЦИПИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ДИЗАЙНУ (SOLID)					
6	**Лекція 3.** Успадкування та Поліморфізм. Принцип LSP. Успадкування, ключове слово extends, ієрархія класів. Перевизначення методів (@Override). Ключове слово super. Поліморфізм, один інтерфейс, багато реалізацій. Принцип заміщення Лісков, поведінка похідних класів. Успадкування vs. Композиція: коли і що обирати.	2	**Практична робота №3.** Реалізація ієрархії класів. Завдання: 1. Створити базовий клас Employee з полями name, salary та методом calculateBonus(). 2. Створити похідні класи Manager та Developer, що успадковуються від Employee. 3. Перевизначити метод calculateBonus() у похідних класах з власною логікою. 4. Створити масив типу Employee та заповнити його об'єктами Manager та Developer. 5. Пройтись по масиву та викликати для кожного об'єкта метод calculateBonus(), демонструючи поліморфізм. **Лабораторна робота №3.** Застосування поліморфізму та принципу LSP. Завдання: 1. Створити базовий клас Rectangle. Додайте private поля width та height. Публічні методи setWidth, setHeight, getArea який повертає width*height. 2. Створити похідний клас Square. Перевизначте методи setWidth, setHeight так, щоб вони завжди встановлювали однакові значення для ширини та висоти одночасно. 3. Написати код, що продемонструє проблему. Створіть екземпляр Rectangle і передайте його в	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

			тестовий метод, описаний нижче. Створіть екземпляр Square, приведіть його до типу Rectangle і також передайте в тестовий метод. Створіть статичний метод calculateAndCheck(Rectangle r), у якому встановлюється висота та ширина, а далі виводиться повідомлення про площу прямокутника. 4. Запустіть код і подивіться на результат. У звіті навести код та коротко пояснити, чому для Square результат виявився неочікуваним і як це порушує основне правило LSP. 5. Рефакторинг: Змініть архітектуру класів, щоб уникнути порушення. Створіть спільний інтерфейс або абстрактний клас Shape з методом getArea(). Rectangle та Square мають бути незалежними класами, що реалізують цей інтерфейс. Продемонструйте, що новий дизайн позбавлений цієї проблеми.		
7	**Лекція 4.** Абстрактні класи та Інтерфейси. Принципи OCP, ISP, DIP. Абстрактні класи та методи, створення часткових реалізацій. Інтерфейси як повна абстракція. Множинна реалізація інтерфейсів. Default-методи інтерфейсів: введення, застосування, обмеження та ризики. Принцип Відкритості/Закритості. Принципи Розділення Інтерфейсу (ISP) та Інверсії Залежностей (DIP). Введення в концепцію Dependency Injection як реалізацію DIP.	2	**Практична робота №4.** Робота з абстрактними класами та інтерфейсами. Завдання: 1. Створити інтерфейс Shape з методами getArea() та getPerimeter(). 2. Створити класи Circle та Rectangle, що реалізують інтерфейс Shape. 3. Створити клас-сервіс ShapeService, який приймає в конструктор об'єкт типу Shape. 4. Реалізувати в ShapeService метод, що виводить площу та периметр фігури. 5. Продемонструвати, що ShapeService залежить від абстракції (Shape), а не від конкретних класів. **Лабораторна робота №4.** Рефакторинг з використанням принципів дизайну. Завдання: 1. Створити клас PaymentProcessor, який	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

			жорстко залежить від класів CreditCardPayment та PayPalPayment. 2. Створити інтерфейс Payable з методом pay(). 3. Змусити класи CreditCardPayment та PayPalPayment реалізувати цей інтерфейс. 4. Провести рефакторинг класу PaymentProcessor, щоб він залежав від інтерфейсу Payable (реалізація DIP). 5. Додати новий спосіб оплати (напр., BitcoinPayment), не змінюючи код PaymentProcessor (демонстрація OCP).		
Тема 3 – КЛЮЧОВІ БІБЛІОТЕКИ ТА ПІДГОТОВКА ДО ENTERPRISE-РОЗРОБКИ					
11	**Лекція 5.** Java Collections Framework та обробка винятків. Огляд ієрархії колекцій: Collection, List, Set, Queue, Map. Вибір правильної колекції для задачі, ArrayList vs LinkedList, HashSet vs TreeSet. Реалізація методів equals() та hashCode(): контракти, типові помилки та вплив на роботу колекцій. Stream API, основні операції (filter, map, collect), робота з колекціями у функціональному стилі. Клас Optional, усунення проблеми null, приклади правильного використання. Основи вводу/виводу (I/O API). InputStream та OutputStream (потоки байтів). Reader та Writer (потоки символів). Буферизовані потоки (BufferedReader, BufferedWriter). RandomAccessFile. Обробка винятків try-catch-finally, throws (короткий огляд у контексті I/O)..	2	**Практична робота №5.** Робота з колекціями. Завдання: 1. Створити ArrayList рядків, додати кілька елементів, видалити один та вивести результат. 2. Створити HashSet та додати до нього кілька однакових елементів, переконатись, що дублікати не зберігаються. 3. Створити HashMap для зберігання пар "студент-оцінка" та вивести всіх студентів з оцінкою вище 80. 4. Написати функцію, яка приймає List<Integer> і повертає новий список без дублікатів. 5. Продемонструвати різницю у продуктивності додавання елементів в ArrayList та LinkedList. **Лабораторна робота №5.** Розробка системи з обробкою винятків. Завдання: 1. Створити клас UserRegistrationService. 2. Реалізувати в ньому метод registerUser(String email, String password). 3. Створити власні винятки: InvalidEmailException та WeakPasswordException. 4. Метод registerUser має перевіряти email та	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

			пароль і кидати відповідні винятки. 5. Написати клієнтський код, що викликає цей метод та коректно обробляє можливі винятки в блоці try-catch.		
12	**Лекція 6.** Багатопотоковість та введення в асинхронність. Поняття процесу та потоку. Проблеми паралельного доступу до даних. Створення потоків, успадкування від Thread та реалізація Runnable. Синхронізація, ключове слово synchronized, монітори. Lock API: коли використовувати замість synchronized, переваги та гнучкість. Потокобезпечні колекції (ConcurrentHashMap, CopyOnWriteArrayList та ін.): як вони вирішують проблему синхронізації в колективному доступі. ExecutorService, сучасний підхід до управління пулом потоків. Введення в асинхронну розробку, Future та CompletableFuture.	2	**Практична робота №6.** Створення та запуск потоків. Завдання: 1. Створити та запустити два потоки, один через успадкування від Thread, інший – через Runnable. 2. Кожен потік має виводити своє ім'я та лічильник від 1 до 10. 3. Написати програму з загальним лічильником, який інкрементують кілька потоків. 4. Продемонструвати проблему race condition. 5. Використати synchronized для вирішення проблеми з доступом до спільного лічильника. **Лабораторна робота №6.** Робота з пулом потоків. Завдання: 1. Створити ExecutorService з фіксованим пулом потоків (напр., 3 потоки). 2. Створити клас-завдання (Runnable), який імітує довгу операцію (напр., засинає на 2 секунди). 3. Відправити на виконання в пул 10 таких завдань. 4. Переконатися, що завдання виконуються паралельно, але не більше ніж 3 одночасно. 5. Коректно завершити роботу ExecutorService після виконання всіх завдань.	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)
Тема 4 – ЯКІСТЬ ТА АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ(SOFTWARE QUALITY AND ARCHITECTURE)					
13	**Лекція 7.**Основи патернів проєктування. Що таке патерни проєктування, їх користь, класифікація (твірні, структурні, поведінкові).	2	**Практична робота №7.** Реалізація патерну "Декоратор". Завдання:	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

	Factory Method: Делегування створення об'єктів підкласам. Вирішення проблеми створення об'єктів, коли точний тип невідомий заздалегідь. Builder: Покрокове створення складних, конфігурованих об'єктів. Decorator: Динамічне додавання нових обов'язків об'єкту без зміни його коду. Принцип "обгортання". Strategy: Інкапсуляція сімейства алгоритмів та забезпечення їх взаємозамінності..		1.Створити інтерфейс Report з методом generate(), що повертає String. 2. Створити базовий клас SimpleReport, що реалізує інтерфейс і повертає простий текстовий звіт (наприклад, "Base report data"). 3. Створити абстрактний клас-декоратор ReportDecorator, що також реалізує Report і зберігає посилання на "обгорнутий" об'єкт Report. 4. Створити конкретні декоратори: HeaderDecorator: додає до звіту заголовок (наприклад, "Report Header\n---\n"). FooterDecorator: додає до звіту підвал (наприклад, "\n---\nReport Footer"). 5. У main методі створити об'єкт SimpleReport і послідовно "обгорнути" його декораторами HeaderDecorator та FooterDecorator. Вивести результат на консоль. **Лабораторна робота №7.** Реалізація патерну "Будівельник". Завдання: 1. Створити клас HttpClient* з великою кількістю конфігураційних параметрів (url, method, headers, body, timeout тощо). 2. Реалізувати для цього класу внутрішній статичний клас Builder. 3. Зробити конструктор HttpClient приватним. 4. Реалізувати в Builder методи для налаштування кожного параметра та метод build(), що повертає готовий об'єкт HttpClient. 5. Створити кілька клієнтів з різними конфігураціями за допомогою будівельника. *Примітка: У цій лабораторній роботі HttpClient не виконує реальних HTTP-запитів. Його мета — лише демонстрація застосування патерну Builder для створення складних об’єктів.		

14	**Лекція 8.** Unit-тестування та керування залежностями (JUnit, Mockito, Maven). Поняття unit-тестів, піраміда тестування. JUnit 5: структура тесту (given-when-then), фікстури (@BeforeEach, @AfterEach). Mockito: створення моків, @Mock, @InjectMocks. Системи збірки Maven для управління залежностями та життєвим циклом проєкту. Підключення залежностей через Maven (pom.xml).	2	**Практична робота №8.** JUnit та структура тестів. Завдання: 1. Створити Maven-проєкт (типу maven-archetype-quickstart). 2. Додати залежність junit-jupiter у pom.xml. 3. Створити простий клас Calculator з методами add, subtract, multiply, divide. 4. Написати тести для цього класу з використанням JUnit 5. Використати структуру given-when-then у назвах методів. Продемонструвати використання @BeforeEach для підготовки об'єкта.. 1. **Лабораторна робота №8.**Unit-тестування з Mockito та Maven. Завдання: 1. Додати залежність mockito-core у pom.xml. 2. Створити інтерфейс UserRepository з методом findById(String id). 3. Створити клас UserService, який у конструкторі приймає UserRepository і має метод getUserName(String id). 4. Написати unit-тести для UserService, замокавши UserRepository за допомогою Mockito. Використати анотації @Mock, @InjectMocks. Продемонструвати поведінкове тестування: коли викликається getUserName, метод findById має бути викликаний рівно 1 раз. 5. Запустити тести через Maven (mvn test)".	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)
15	**Лекція 9.** Асинхронна взаємодія. Введення в асинхронну комунікацію. Синхронна vs. Асинхронна комунікація. Патерн "Видавець-Підписник" (Publisher-Subscriber) Ролі: Видавець (Publisher), Підписник (Subscriber),	2	**Практична робота №9.** Імплементація патерну "Видавець-Підписник" Завдання: 1. Створити інтерфейс EventListener з одним методом update(String message). 2. Створити кілька конкретних реалізацій-	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

	Канал/Тема (Topic). Брокери повідомлень (Message Brokers): Огляд RabbitMQ. Exchange, Queue, Binding. Огляд Apache Kafka, концепція "розподіленого логу подій"..		підписників: EmailNotificationListener: виводить у консоль "Sending email with message: [message]". LogListener: виводить у консоль "Logging message: [message]". 3. Створити клас EventManager (видавець), який буде керувати підписниками. Клас повинен мати методи subscribe(EventListener listener) та unsubscribe(EventListener listener). Клас повинен мати метод notify(String message), який проходить по всіх підписниках і викликає їх метод update. 4. У main методі: Створити екземпляр EventManager. Створити та підписати кілька слухачів. Викликати метод notify кілька разів, щоб продемонструвати, як усі підписники отримують сповіщення. **Лабораторна робота №9. Взаємодія сервісів через RabbitMQ.** Завдання: 1. Налаштування середовища: Запустити офіційний образ RabbitMQ в Docker-контейнері. 2. Створити проєкт "Видавець" (OrderService): Додати залежність для RabbitMQ клієнта. Написати код, який підключається до RabbitMQ, створює повідомлення (напр., JSON з інформацією про замовлення) і відправляє його в exchange з назвою orders. 3. Створити проєкт "Підписник" (NotificationService): Також додати залежність RabbitMQ клієнта. Написати код, який підключається до RabbitMQ. Створює queue (чергу), "прив'язує" її до exchange orders. Запускає слухача, який чекає на повідомлення з		

			черги і, при отриманні, виводить його вміст у консоль. 4. Демонстрація: Спочатку запустити "Підписника". Потім запустити "Видавця". Переконатись, що повідомлення, відправлене видавцем, було отримане та оброблене підписником.		
Тема 5 – ЕКОСИСТЕМА SPRING ТА ПОБУДОВА ВЕБ-СЕРВІСІВ					
	Лекція 10. Вступ до Spring Framework та управління проєктом. Архітектура Spring Framework та ключові модулі. Spring Boot як засіб для швидкого старту та автоконфігурації. Принципи RESTful API для побудови архітектури клієнт-серверних додатків. Робота Spring Initializr для генерування структури проєкту.	2	**Практична робота №10.** Ініціалізація проєкту та знайомство зі структурою. Завдання: 1. Використовуючи Spring Initializr (start.spring.io), згенерувати скелет проєкту. 2. Додати залежності Spring Web та Lombok. 3. Відкрити проєкт в IntelliJ IDEA та розібрати його структуру (папки src/main/java, src/main/resources, файл pom.xml). 4. Проаналізувати вміст pom.xml, знайти підключені залежності Spring Boot. 5. Запустити порожній додаток та переконатися, що він стартує без помилок. **Лабораторна робота №10.** Створення першого REST-контролера. Завдання: 1. Створити новий пакет для контролерів (напр., com.example.project.controller). 2. У цьому пакеті створити клас, анотований як @RestController. 3. Реалізувати в контролері простий GET-ендпоінт, що повертає рядок "Hello, World!". 4. Запустити додаток і перевірити роботу ендпоінту через браузер або Postman. 5. Додати ще один ендпоінт, який приймає ім'я як параметр запиту (@RequestParam) і повертає персоналізоване привітання.	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

	Лекція 11. Розробка веб-шару з Spring MVC. Основні анотації Spring MVC для обробки HTTP-запитів (@GetMapping, @PostMapping тощо). Передача даних за допомогою @PathVariable, @RequestParam та @RequestBody. Концепція DTO (Data Transfer Object) для передачі даних між шарами. Налаштування валідації вхідних даних за допомогою анотацій Jakarta Bean Validation (@NotNull, @Size). Обробка помилок та повернення коректних HTTP-статусів.	2	**Практична робота №11.** Проєктування DTO та валідації. Завдання: 1. Створити пакет для DTO. 2. Розробити класи DTO для створення та відображення основної сутності проєкту (напр., CreateBookRequestDto, BookDto). 3. Додати до полів CreateBookRequestDto анотації для валідації (напр., назва не може бути порожньою). 4. Створити клас-виняток для обробки помилок валідації (@ControllerAdvice). 5. Переконатись, що при відправці некоректних даних API повертає помилку зі статусом 400 Bad Request. **Лабораторна робота №11.** Реалізація CRUD-операцій на веб-шарі. Завдання: 1. Реалізувати повний набір CRUD-ендпоінтів у контролері для вашої сутності. 2. Використовувати DTO, розроблені на практичній роботі, як тіло запитів та відповідей. 3. Для зберігання даних тимчасово використовувати HashMap або ArrayList всередині контролера. 4. Перевірити роботу всіх ендпоінтів (Create, Read, Update, Delete) за допомогою Postman. 5. Реалізувати отримання елемента за id та отримання повного списку елементів.	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)
Тема 6 – РОБОТА З ДАНИМИ ТА БІЗНЕС-ЛОГІКА					
	Лекція 12. Постійне зберігання даних з Spring Data JPA. Введення в ORM та JPA (Java Persistence API). Роль Hibernate. Анотації JPA для мапінгу класів на таблиці в БД (@Entity, @Table, @Id). Патерн "Репозиторій" та його реалізація в Spring	2	**Практична робота №12.** Підключення до БД та створення сутності. Завдання: 1. Додати залежності Spring Data JPA та драйвер для PostgreSQL в pom.xml. 2. Налаштувати підключення до бази даних у файлі application.properties.	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

	Data JPA. Інтерфейс JpaRepository та його стандартні методи для роботи з даними. Написання власних методів запитів за назвою методу (derived query methods).		3. Створити клас-сутність (напр., Book), анотований @Entity, з полями та первинним ключем @Id. 4. Створити інтерфейс-репозиторій, що успадковується від JpaRepository. 5. Запустити додаток та переконатися, що Hibernate автоматично створив таблицю в базі даних. **Лабораторна робота №12.** Інтеграція бази даних у проєкт. Завдання: 1. Підключити до проєкту базу даних PostgreSQL, запущену в Docker. 2. Видалити з контролера тимчасове сховище (HashMap). 3. Впровадити (inject) репозиторій у контролер. 4. Переписати всі методи CRUD-контролера для використання методів репозиторію (save(), findById(), findAll(), deleteById()). 5. Перевірити, що дані тепер зберігаються в базі даних після перезапуску додатку.		
	Лекція 13. Побудова сервісного шару та реалізація бізнес-логіки. Призначення сервісного шару як посередника між контролером та репозиторієм. Принципи SOLID у побудові сервісів (SRP, DIP). Анотації @Service та @Autowired для реалізації Dependency Injection. Обробка транзакцій за допомогою анотації @Transactional. Мапінг між DTO та сутностями (Entity).	2	**Практична робота №13.** Проєктування та реалізація сервісного шару. Завдання: 1. Створити інтерфейс для сервісу (напр., BookService). 2. Створити клас-імплементацію BookServiceImpl, анотований @Service. 3. Впровадити репозиторій у сервіс. 4. Реалізувати в сервісі всі CRUD-методи, які делегують виклики репозиторію. 5. Обгорнути методи, що змінюють дані, в анотацію @Transactional. **Лабораторна робота №13.** Рефакторинг проєкту з додаванням сервісного шару. Завдання: 1. Впровадити сервіс (BookService) у контролер	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

			(BookController). 2. Переписати всі методи контролера так, щоб вони викликали методи сервісу, а не репозиторію. 3. Реалізувати мапінг між DTO та сутностями (напр., за допомогою бібліотеки MapStruct або вручну). 4. Додати в сервіс складнішу бізнес-логіку (напр., перевірку, що книга з такою назвою ще не існує, перед збереженням). 5. Переконатись, що контролер тепер відповідає лише за обробку HTTP-запитів, а вся логіка знаходиться в сервісі.		
	Лекція 14. Зв'язки між сутностями та розширені запити. Типи зв'язків у JPA: @OneToOne, @OneToMany, @ManyToOne, @ManyToMany. Налаштування каскадних операцій та типів завантаження (LAZY/EAGER). Реалізація пагінації та сортування за допомогою Pageable в Spring Data. Створення складних запитів за допомогою анотації @Query (JPQL).	2	**Практична робота №14.** Розширення моделі даних. Завдання: 1. Додати до проєкту нову сутність (напр., Category або Author). 2. Встановити зв'язок @ManyToOne або @ManyToMany між новою сутністю та існуючою. 3. Створити для нової сутності репозиторій, DTO та сервіс. 4. Додати в сервіс метод, що дозволяє знайти всі книги певної категорії. 5. Протестувати новий функціонал. **Лабораторна робота №14.** Реалізація пагінації та сортування. Завдання: 1. Модифікувати метод контролера для отримання списку всіх книг. 2. Додати до сигнатури методу параметр Pageable. 3. Передати цей параметр у відповідний метод сервісу та репозиторію. 4. Перевірити роботу пагінації та сортування через параметри запиту в Postman	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

			(напр., ?page=0&size=5&sort=title,asc). 5. Реалізувати фільтрацію книг за параметрами за допомогою JPQL-запиту в репозиторії.		
Тема 7 – ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ТА БЕЗПЕКИ					
	Лекція 15. Безпека додатків з Spring Security. Основи Spring Security: автентифікація та авторизація. Архітектура на основі фільтрів. SecurityFilterChain. Зберігання паролів у зашифрованому вигляді (PasswordEncoder). Захист REST API за допомогою JWT (JSON Web Token). Налаштування доступу до ендпоінтів на основі ролей користувачів (RBAC).	2	**Практична робота №15.** Базова конфігурація Spring Security. Завдання: 1. Додати залежність Spring Security в pom.xml. 2. Створити конфігураційний клас для налаштування SecurityFilterChain. 3. Налаштувати публічний доступ до ендпоінтів для реєстрації та входу. 4. Захистити всі інші ендпоінти, вимагаючи автентифікації. 5. Створити сутність User з полями email, password, role та відповідний репозиторій. **Лабораторна робота №15.** Реалізація автентифікації та авторизації. Завдання: 1. Реалізувати сервіс та контролер для реєстрації нових користувачів. 2. Реалізувати ендпоінт для входу, який у разі успіху повертає JWT. 3. Створити фільтр, який перевіряє JWT у кожному запиті та встановлює автентифікацію. 4. Налаштувати доступ до методів (напр., видалення книги) лише для користувачів з роллю ADMIN. 5. Перевірити роботу системи безпеки в Postman.	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)
	Лекція 16. Інтеграційне тестування у Spring Boot та використання Testcontainers. Поняття інтеграційного тестування. Різниця між unit- та інтеграційними тестами. Анотації Spring Boot для інтеграційного тестування: @SpringBootTest, @AutoConfigureMockMvc, @TestConfiguration.	2	**Практична робота №16.** Інтеграційне тестування REST-контролера з використанням MockMvc та Testcontainers. Завдання: 1.Створити тестовий клас для контролера (наприклад, BookControllerTest), анотований @SpringBootTest.	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse)

	Тестування REST-контролерів з використанням MockMvc. Тестування сервісного шару з підключенням до повного контексту Spring. Особливості налаштування середовища тестування (application-test.yml, профілі). Інструмент Testcontainers для створення ізольованого середовища БД у Docker-контейнерах. Запуск СУБД у контейнері під час тестування. Конфігурація Testcontainers у проєкті (додаткові залежності, запуск контейнерів у тестах). Ізольоване тестування транзакційної поведінки, взаємодії з базою та поведінки CRUD-операцій. Автоматичне підняття/завершення контейнерів перед і після тестів.		2. Налаштувати MockMvc для виконання HTTP-запитів до ендпоінтів. 3. Написати інтеграційний тест для ендпоінту створення книги: відправити POST-запит із JSON-тілом, перевірити HTTP-статус (201 Created), перевірити, що об'єкт збережено в базі. 4. Написати тест для ендпоінту отримання книги за ID: відправити GET-запит, перевірити HTTP-статус (200 OK), перевірити вміст JSON-відповіді. 5. Налаштувати Testcontainers для запуску реальної БД (PostgreSQL або MySQL) в Docker-контейнері. 6. Переконатись, що база даних створюється автоматично при запуску тестів і не впливає на основну БД розробки. **Лабораторна робота №16.** Інтеграційне тестування сервісного шару з використанням Testcontainers. Завдання: 1. Створити окремий тестовий клас для BookService (наприклад, BookServiceIntegrationTest), анотований @SpringBootTest. 2. Налаштувати Testcontainers для запуску БД PostgreSQL або MySQL у контейнері: додати відповідні залежності до pom.xml / build.gradle, створити контейнер через @Container і @Testcontainers. 3. Написати тести для методів сервісу: створення нової книги, отримання книги за ID, перевірка валідації (наприклад, що книга з однаковою назвою не створюється двічі). 4. Перевірити транзакційність: при помилці збереження дані не мають з'явитись у базі.		

			5. Забезпечити ізольованість тестів: використовувати окремий тестовий профіль, очищати БД між тестами або створювати новий контейнер для кожного класу. 6. Перевірити, що тести можна запускати без встановленої БД на машині — лише за наявності Docker.		
Тема 8 – РОЗГОРТАННЯ ТА ЕКСПЛУАТАЦІЯ ДОДАТКІВ					
	Лекція 17. Контейнеризація додатків з Docker. Основи Docker: образи та контейнери. Написання Dockerfile для пакування Spring Boot-додатку. Багатоетапні збірки для оптимізації розміру образу. Оркестрація багатоконтейнерних додатків за допомогою Docker Compose. Запуск додатку та бази даних в одній мережі.	2	**Практична робота №17.** Створення Docker-образу. Завдання: 1. Встановити Docker Desktop. 2. Створити в корені проєкту файл Dockerfile. 3. Написати інструкції для копіювання jar-файлу додатку в образ. 4. Вказати команду для запуску додатку всередині контейнера. 5. Зібрати Docker-образ за допомогою команди docker build. **Лабораторна робота №17.** Оркестрація з Docker Compose. Завдання: 1. Створити в корені проєкту файл docker-compose.yml. 2. Описати в файлі два сервіси: app (для Spring-додатку) та db (для PostgreSQL). 3. Налаштувати залежність app від db, щоб додаток стартував після бази даних. 4. Налаштувати змінні середовища для підключення додатку до БД всередині Docker-мережі. 5. Запустити всю систему однією командою docker-compose up та перевірити її роботу.	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse), Docker
	Лекція 18. Основи CI/CD та асинхронної комунікації. Концепції CI (Continuous Integration) та CD (Continuous Deployment).	2	**Практична робота №18.** Побудова CI-пайплайну. Завдання: 1. Створити в проєкті	4	Java Development Kit, IntelliJ IDEA (або Visual Studio Code, Eclipse), Docker

	Побудова автоматизованих пайплайнів за допомогою GitHub Actions. Етапи пайплайну: збірка, тестування, пакування в Docker-образ.		директорію .github/workflows. 2. Створити YAML-файл для опису пайплайну GitHub Actions. 3. Налаштувати пайплайн, щоб він спрацьовував на push до гілки main. 4. Додати кроки для checkout коду, налаштування Java та кешування Maven. 5. Додати крок для запуску тестів за допомогою команди mvn test. **Лабораторна робота №18.** Впровадження асинхронної задачі. Завдання: 1. Додати до docker-compose.yml сервіс для RabbitMQ. 2. Додати залежність spring-boot-starter-amqp в pom.xml. 3. Створити компонент (@Component), який відправляє повідомлення в чергу при створенні нової книги. 4. Створити інший компонент-слухач (@RabbitListener), який отримує це повідомлення. 5. При отриманні повідомлення слухач має імітувати якусь дію (напр., логувати "Надсилаємо email-сповіщення про нову книгу...").		

ТЕМИ ТА ЗАВДАННЯ ДЛЯ САМОСТІЙНОЇ РОБОТИ

№	Назва та опис завдання	Методи контролю та критерії оцінювання	Годин
1	**Реалізація патерну "Команда" (Command).** Створити простий пульт керування, який може вмикати/вимикати різні пристрої (напр., світло, телевізор). Кожна дія має бути інкапсульована в окремому класі-команді.	Перевірка коду в GitHub-репозиторії студента. Оцінюється коректність реалізації патерну, чітке розділення відповідальності між класами.	7
2	**Рефакторинг коду з порушенням SOLID.** Надати студентам готовий клас з очевидними порушеннями 2-3 принципів SOLID (напр., SRP та OCP). Завдання полягає у проведенні рефакторингу коду з детальним описом внесених змін та їх обґрунтуванням у	Перевірка коду ("до" та "після") у репозиторії. Оцінюється здатність ідентифікувати "запахи коду" та застосувати відповідні принципи для їх виправлення.	7

	README-файлі.		
3	**Застосування патерну "Прототип" (Prototype).** Розробити систему для створення ігрових персонажів, де нові персонажі створюються шляхом клонування існуючого прототипу з подальшою зміною деяких характеристик.	Перевірка коду в GitHub-репозиторії. Оцінюється правильне використання інтерфейсу Cloneable та розуміння поверхневого і глибокого копіювання.	7
4	**Впровадження глобальної обробки винятків.** Додати до семестрового проєкту централізований обробник винятків за допомогою @ControllerAdvice. Він має перехоплювати специфічні для проєкту помилки (напр., ResourceNotFoundException) та повертати клієнту стандартизовану відповідь у форматі JSON.	Pull Request до основного проєкту в GitHub. Оцінюється чистота реалізації, коректність HTTP-статусів, що повертаються, та формат JSON-відповіді.	7
5	**Автоматична генерація API-документації.** Інтегрувати в проєкт бібліотеку Springdoc OpenAPI для автоматичного створення інтерактивної Swagger UI документації для розробленого REST API. Додати описові анотації до контролерів та DTO.	Pull Request до основного проєкту. Оцінюється успішна інтеграція, доступність та повнота згенерованої документації для всіх ендпоінтів.	7
6	**Впровадження механізму міграцій бази даних.** Інтегрувати в проєкт інструмент Flyway або Liquibase. Створити початкову міграцію, яка генерує схему бази даних, та одну додаткову міграцію, що додає нову колонку до існуючої таблиці.	Pull Request до основного проєкту. Оцінюється коректність налаштування інструменту та правильна структура файлів міграцій.	7

ПРОЄКТ (за наявністю)

№	Назва та опис завдання	Метод контролю та захисту	Строки виконання
1	**Курсовий проєкт.** Виконання проєкту за індивідуальною темою з розробки програмного забезпечення на основі технологій Java	Презентація проєкту з демонстрацією роботи та відповіддю на запитання комісії. Підготовка вихідного коду, документації, відео демонстрації та презентації.	До кінця семестру 6

РЕКОМЕНДОВАНІ ДЖЕРЕЛА ІНФОРМАЦІЇ ТА НАВЧАЛЬНІ МАТЕРІАЛИ

Основні

№	Назва	До теми (вказати номер)
1	Schildt H. Java: The Complete Reference : 13th ed. / H. Schildt. — New York : McGraw-Hill Education, 2023. — 1376 p.	1, 2, 3, 4, 8, 9
2	Walls C. Spring in Action : 6th ed. / C. Walls. — Shelter Island, NY : Manning Publications Co. LLC, 2022. — 520 p.	10-15, 18

3	Gupta R. Java Design Patterns: A Hands-On Experience with Real-World Examples / R. Gupta. — Berkeley, CA : Apress L. P., 2022. — 450 p.	5, 6, 7
4	Spring Framework Documentation. Spring. URL: https://docs.spring.io/spring-framework/reference/index.html	10-18
5	Java Documentation. Oracle Help Center. URL: https://docs.oracle.com/en/java/	1-18
6	Java-програмування: комп'ютерний практикум [Електронний ресурс] : навч. посіб. / КПІ ім. Ігоря Сікорського; уклад.: Ю. А. Тарнавський. – Київ, 2021. – 95 с.	1, 2, 8

Додаткові

№	Назва	До теми (вказати номер)
1	Bloch J. Effective Java : 3rd ed. / J. Bloch. — Boston : Addison-Wesley Professional, 2018. — 416 p.	2, 3, 4, 8, 9
2	Мартін Р. С. Clean Code: A Handbook of Agile Software Craftsmanship : 2-ге вид. / R. C. Martin. — Hoboken, NJ : Addison-Wesley Professional, 2025. — 672 с. : ISBN 978-0-13-539857-9.	2, 3, 4, 13
3	Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design / R. C. Martin. — Boston : Prentice Hall, 2017. — 432 p.	4, 12, 13
4	The Destination for Java Developers. Dev.java. URL: https://dev.java/	1-18

КОНТРОЛЬНІ ЗАХОДИ

№	Назва та опис	Методи контролю та критерії оцінювання
1	Поточний контроль. Виконання практичних робіт та завдань на самостійну роботу впродовж семестру	Перевірка правильності виконання завдання, захист виконаної роботи шляхом відповіді на запитання викладача за темою завдання, оформлення звіту з виконаної роботи. Максимальна оцінка — до 60 балів у кожному семестрі.
2	Підсумковий контроль (семестровий екзамен). Екзамен у формі тестування.	Умовою допуску до підсумкового контролю є набрання студентом 40 балів у сукупності за всіма темами дисципліни у семестрі. Якщо за іспит студент набирає менше 20 балів, то ці бали не сумуються з балами за поточний контроль, таким чином студент отримує час на перескладання іспиту. Якщо студента не допущено до складання іспиту, як такого, що не виконав індивідуальний план, йому надається час до перескладання для виконання всіх вимог допуску. Студент має право на два перескладання. При повторному перескладанні іспиту його у студента може приймати комісія, яка створюється директором навчально-наукового інституту. Оцінка комісії є остаточною.

Силабуси проходять валідацію у Kharkiv IT Cluster

KHARKIV IT CLUSTER

Оцінювання студентів в семестрі здійснюється за накопичувальною 100-бальною системою, складається із двох основних блоків і розподіляється в пропорції 60 (бали, напрацьовані під час вивчення дисципліни при виконанні практичних і лабораторних робіт та завдань самостійної роботи) на 40 (підсумковий контроль).
Розрахунок рейтингових балів за видами робіт за семестр здійснюється за формулою:
Рейтинговий бал= П + Т +*Д*,
Поточний контроль студента (П) — сума балів за виконання практичних і лабораторних робіт та завдань на самостійну роботу протягом семестру;
Т – підсумковий контроль ;
Додаткові бали (Д) — визначаються залежно від виду виконаної роботи.

Студент може отримати ***додаткові бали*** впродовж семестру. Кількість балів за додаткові види робіт визначається диференційовано відповідно до виду та складності виконаного завдання.
Максимальна кількість додаткових балів, що можуть бути зараховані за дисципліну — 10.
Види робіт для отримання додаткових балів:

1) Участь у профільних олімпіадах чи конкурсах - оцінюється до 10 балів в залежності від результатів участі.
2) Отримання сертифікату в рамках інформальної освіти (не більше 2 курсів) - оцінюється до 5 балів за курс. Якщо теми курсів перетинаються між собою, то зарахування балів відбувається лише один раз. Враховуються лише ті теми курсів, що вказані в силабусі дисципліни.
3) Виконання додаткових творчих завдань (наприклад, робота над індивідуальним чи груповим проєктом, участь в стартапі тощо) (кількість не обмежується) - кількість балів визначається диференційовано відповідно до складності виконаного завдання. Додаткові творчі завдання у форматі індивідуального чи групового проєкту, стартапу тощо можуть замінювати основні роботи (практичні, самостійні), якщо вони покривають практичну складову відповідної теми освітнього компоненту.
4) Участь у наукових конференціях, підготовка наукових публікацій за тематикою освітньої компоненти:
    a) Тези доповіді на фаховій конференції — 3 бали
    b) Стаття у фаховому виданні — 5 балів
    c) Стаття в іноземному рецензованому виданні — 10 балів

Якщо сумарна кількість балів з урахуванням додаткових балів перевищує 100, вона округлюється до 100.

Підсумковий контроль проводиться у вигляді підсумкового тесту у комп’ютерному форматі з перевіркою викладачем відкритих питань тесту. Тест містить питання, що охоплюють теоретичну та практичну складову курсу. Під час тестування заборонено використання будь-яких джерел.

		Бали за кожне тестове питання визначаються відповідно до складності питання. Сумарна кількість балів за тест складає 40 балів.
3	Курсовий проєкт (6 семестр)	Курсова робота оцінюється згідно критеріїв якості реалізованого програмного забезпечення та наявності виконаного проєкту і оформлення документації. Максимальна оцінка — 100 балів.

РЕЗУЛЬТАТИ НАВЧАННЯ

ПРН 3. Знати основні процеси, фази та ітерації життєвого циклу програмного забезпечення.
ПРН 6. Уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення.
ПРН 12. Застосовувати на практиці ефективні підходи щодо проектування програмного забезпечення.

ЗВ'ЯЗОК ІЗ РИНКОМ ПРАЦІ

Спеціальність/професія, підготовці до діяльності в якій читається курс	Java розробник (Backend, Fullstack)
Посилання на вакансії (понад 3), та їх скриншоти, в яких вказані компетенції, що формуються на курсі, як вимоги до кандидатів	https://www.work.ua/jobs/5747767/ Middle Java Developer

Силабуси проходять валідацію у Kharkiv IT Cluster

KHARKIV IT CLUSTER

Вимоги до кандидатів:

- досвід роботи від 2 років;
- участь в обговоренні та прийнятті стратегічних рішень в подальшій розробці продукту згідно ТЗ;
- досвід побудови архітектури БД;
- написання якісного коду в рамках проєкту;
- написання тестів;
- написання документації.

Досвід роботи з технологіями:

- Java 21, Spring MVC, Spring Boot, REST, API;
- JavaScript, CSS, HTML, Bootstrap, jQuery, Google Maps;
- Hibernate, JDBC;
- MySQL, Oracle, PostgreSQL
- Maven, GIT;
- Eclipse, IntelliJ IDEA;

https://www.work.ua/jobs/6772903/ Java, Back-End Developer

Силабуси проходять валідацію у Kharkiv IT Cluster

KHARKIV IT CLUSTER

Що ми очікуємо:

- Впевнене володіння Java або іншими JVM-мовами — **must**
- Досвід з **Spring Core** — **must**
- Практичні навички роботи з **SQL та Hibernate**
- Бажано знання **Spring Boot, Spring MVC, Spring Data**
- Знання **Maven або Gradle** — буде плюсом
- Розуміння принципів якісного дизайну ПЗ
- Досвід з **Kotlin** — буде плюсом
- Профільна освіта (Computer Science, математика або суміжні спеціальності) — **перевага**
- Добрі **комунікативні навички** — обов'язково

Робоче середовище:

- IDE: IntelliJ IDEA, WebStorm

https://www.work.ua/jobs/6033759/ Java-розробник

Ключові компетенції:

- Вища освіта в IT / інженерії / математиці. Готові розглянути студентів з досвідом роботи.
- Комерційний досвід розробки від 1 року.
- Впевнені знання Java та Java Core, як основної платформи розробки.
- Написання чистого, підтримуваного та ефективного коду.
- Відмінне знання ООП, алгоритмів та структур даних.
- Багатопоточне програмування.
- Знання GIT або інших систем керування версіями вихідного коду.

Силабуси проходять валідацію у Kharkiv IT Cluster

KHARKIV IT CLUSTER

https://jobs.dou.ua/companies/nerdysoft/vacancies/310091/ Intern Java Full Stack Developer

Вимоги:

- Базові знання Java та розуміння принципів SOLID;
- Розуміння базових концепцій ООП;
- Знання JS, HTML, CSS, React **(nice to have)**;
- Ступінь бакалавра в галузі прикладної математики, комп'ютерних наук, інформаційної безпеки або споріднених (може бути неповним);
- Проактивна, орієнтована на результат людина з готовністю брати на себе відповідальність та бажанням вчитися;
- Розвинені комунікативні навички, вміння чітко й зрозуміло доносити інформацію і брати участь в обговореннях;
- Рівень англійської — не нижче upper-intermediate (B2);
- **Розглядаємо кандидатів, які перебувають у Львові/Львівській області.**

Буде перевагою:

- Робота з базами даних (SQL, JDBC, JPA), розуміння REST API та навички написання юніт-тестів;
- Знання Node.JsTypescript, WebSockets, Tailwind, NgRx store, Nx monorepo, Unit tests, Storybook;
- Будь-який комерційний досвід роботи або пет-проекти.

Силабуси проходять валідацію у Kharkiv IT Cluster

KHARKIV IT CLUSTER

	https://jobs.dou.ua/companies/1plus1/vacancies/313836/ Back-end Developer (middle+) **Required skills:** – 4+ years of commercial strong Java experience, Java 8+ knowledge; – experience with Spring framework not SpringBoot, also (SpringBoot, Data, Security); – experience with MongoDB, Redis; – experience with search engines (Elasticsearch, Solr); – experience with Docker (mandatory) and Kubernetes (advantage); – experience with message brokers (RabbitMQ, SQS/SNS); – knowledge of SOLID/GRASP OO design principles; – understanding of tomcat servlet container; – experience in clean code techniques, refactoring and testing. **Nice to have:** – Experience in Cloud Computing Platforms (AWS/aws-sdk); – Java or any cloud Certification; – experience in working with distributed and high-availability systems; – ability to adopt new technologies fast; – excellent communication, attitude and teamwork skills; – ability to function both independently and in a large team; – passion towards making great products.
Перелік компетентностей із вказаних як вимоги до	Впевнене володіння Java Core. Розуміння та практичне застосування принципів ООП та SOLID. Досвід роботи з Spring Framework (Spring Boot, Spring MVC, Spring Data).

вакансії, які набувають студенти, в процесі проходження дисципліни.	Досвід розробки RESTful API. Досвід роботи з ORM (Hibernate/JPA). Практичні навички в написанні SQL-запитів. Досвід роботи з інструментами збірки проєктів (Maven). Досвід написання Unit та інтеграційних тестів (JUnit, Mockito). Досвід контейнеризації додатків за допомогою Docker. Розуміння принципів CI/CD. Досвід роботи з системами контролю версій (Git). Знання основ роботи клієнт-серверної архітектури.

Інструменти оцінювання результатів навчання за дисципліною*

№	**Об'єкт оцінювання** (знання методів та принципів, практичні навички, командна робота тощо)	**Методи контролю** (тести, виконання поточних практичних завдань та їх форма: написання коду, створення діаграми Гантта, створення прототипу тощо)	**Інструмент оцінювання** (доступ до результатів тесту, гостьова лекція, посилання на виконані завдання, посилання на проєкт, присутність на захисті проєктів, доступ до запису захисту тощо)
1	Практичні навички	Виконання практичних робіт, захист робіт та оформлення звітів з робіт	Посилання на звіти з виконаних завдань та журнал поточного оцінювання
2	Теоретичні знання	Комп’ютерні тести	Доступ до результатів тесту
3	Знання та навички, отримані в рамках інформальної освіти	Сертифікати, що підтверджують засвоєння знань за темами курсу	Доступ до файлів/посилань сертифікатів
4	Виконання курсового проєкту	Публічна презентація результатів виконання проєкту	Присутність на захисті курсового проєкту/ доступ до файлів вихідного коду, відео демонстрації, презентації, документації

*__Блок. Інструменти оцінювання результатів навчання за дисципліною —__ цей блок має на меті показати можливі варіанти проведення валідації за участі експертів від IT-компаній. Основна умова проведення валідації це попереднє погодження всіх аспектів з експертом та викладачем. Інструменти, завдяки яким це можна зробити:

- тести за темою (аналогічно технічному інтерв'ю);
- поточні завдання впродовж курсу (посилання на код на GitHub-репозитарію студента тощо) ;

Силабуси проходять валідацію у Kharkiv IT Cluster

KHARKIV
IT CLUSTER

- підсумковий проєкт (код, презентація, скринкаст презентації тощо);
- гостьові активності (гостьові лекції, відвідування контрольних активностей тощо).