Компетентності рівня junior для Java	Загальна оцінка компетентності (max. 21)	Компанії						
		G5 Entertainment	SoftServe	Akvelon	Viseven	Knubisoft	CHI Software	Avenga
1. Основи Java. Базові знання з програмування								
1.1 Змінні та типи даних. Структури даних. Знає різні типи даних (примітивні та референсні) та структури у Java - числа, рядки, символи, булеві значення, масиви, класи та інтерфейси, переліки(enum), списки (ArrayList), стеки, черги та картки (Map), здатний використовувати їх для зберігання даних, усвідомлює правила оголошення та присвоєння значень змінним. Практикує використання різних колекційних типів, таких як ArrayList, LinkedList, HashMap тощо для зберігання та маніпулювання даними. Знає спеціальні операції та методи для створення та ініціалізації, роботи зі структурами даних. Розуміє принцип автоматичного перетворення примітивних типів в обгорткові (boxing) та навпаки (unboxing). Вмє розпізнавати моменти, коли відбувається автоматичне перетворення типів.	**21**	3	3	3	3	3	3	3
1.2 Оператори та цикли, інструкції мови, умовні конструкції Ідентифікує математичні оператори, логічні, побітові оператори та оператори порівняння. Знає умовні конструкції (if-else, switch) та вміє застосовувати їх для розробки розгалужень у програмі. Використовує цикли (for, while, do-while) для ітерації та повторення блоків коду. Вміє читати та розуміти код, який використовує ці оператори та інструкції.	**21**	3	3	3	3	3	3	3
1.3 Функції. Розуміє концепції функцій та методів. Вміє виділяти частини коду в окремі функції для забезпечення повторного використання та модульності. Знає поняття області видимості та простору імен у функціях. Використовує параметри та аргументи функції, параметри по замовчуванню, повернення значень з функції. Застосовує рекурсію для розв'язання завдань, що можуть бути представлені у вигляді повторюваних підзадач. Використовує механізм роботи стеку викликів для відстеження послідовності викликів функцій та методів під час виконання програми. Усвідомлює особливості виникнення помилки StackOverflowException при занадто глибокому рекурсивному виклику функцій, що призводить до переповнення стеку викликів. Має навички налагодження (debugging) коду з використанням breakpoints та переглядом стеку викликів (call stack) для зупинки виконання програми та аналізу значень змінних. Ідентифікує основні патерни проектування функцій, такі як функції вищого порядку, замикання (closures), рефакторинг коду функцій.	**19**	3	3	3	2	3	3	2
1.4 Date. Обробка дати та часу. Вміє створювати та маніпулювати об'єктами дати та часу. Розуміє форматування та розбір дати та часу за допомогою класу SimpleDateFormat. Використовує основні класи для роботи з датою та часом, таких як Date, Calendar, LocalDateTime, LocalDate, LocalTime. Здатний застосовувати спеціальні методи для виконання операцій з датою та часом, таких як порівняння, додавання, віднімання, форматування і розбір дати.	**10**	0	2	2	1	3	2	0
1.5 Помилки та виключення (Errors & Exceptions)								
1.5.1 Помилки, виключення Розуміє концепцію помилок та виключень, їх виклику та обробки в програмі. Використовує різні підходи до обробки помилок, такі як повернення статусу коду, використання спеціальних об'єктів, виходів (exit) з програми та використання виключень. Ідентифікує ситуації, коли виключення використовуються для сигналізування про помилкові стани, неочікувані умови та виключні ситуації. Вміє оголошувати та використовувати блоки try-catch-finally та механізм rethrow для обробки виключень в коді. Ідентифікує основні виключення, які можуть виникати в Java, і як їх обробляти.	**20**	3	3	3	2	3	3	3
1.5.2 Null-значення, генерація виключень Ідентифікує NullPointerException (NPE) та інші виключення, пов'язані з null-значеннями. Використовує анотацію Nullable для позначення того, що певний параметр або повернене значення може мати значення null. Практивує підходи для уникнення повернення null, такі як використання виключень, Optional, null object, empty list. Знає Null-Object Pattern як підхід для представлення відсутнього об'єкта замість null. Використовує властивості стеку викликів для відстеження походження виключень. Генерує виключення власного коду за допомогою throw. Розуміє відносини між класами виключень у ієрархії наслідування, які допомагають при правильному обробленні різних типів виключень. Використовує принцип "прокидання вгору" (rethrowing) виключень для обробки помилок на вищому рівні виклику.	**15**	3	1	3	2	1	2	3
1.6 Складні структури даних Розуміє структуру бінарного дерева та його використання для швидкого пошуку та сортування даних. Знає алгоритми обходу бінарного дерева, такі як інфіксний, префіксний, постфіксний обхід. Ідентифікує графи як абстрактну структуру, яка представляє сукупність вузлів та зв'язків між ними. Знає різні види графів, такі як орієнтовані та неорієнтовані, зважені та незважені, ациклічні та циклічні тощо. Розуміє особливості реалізації алгоритмів обходу графів, таких як пошук в ширину (BFS) та пошук у глибину (DFS). Вміє застосовувати ці алгоритми для пошуку шляхів, з'єднаності та інших операцій у графах.	**4**	0	1	1	0	1	0	1
2. Теоретичний блок. Розуміння ООП								

2.1 Основні концепції ООП. Об'єкти Знає принципи використання об'єктів для організації та структурування даних та функціональності у програмі. Ідентифікує розрізнення між класом (шаблоном) та його екземплярами (об'єктами). Знає про розділення пам'яті Java на heap (куди зберігаються об'єкти) та stack (куди зберігаються локальні змінні та стек викликів). Розуміє основні концепції ООП (інкапсуляція, спадкування та поліморфізм).	**21**	3	3	3	3	3	3	3
2.2 Класи, конструктори, властивості, модифікатори доступу Ідентифікує поняття класу та його роль в ООП. Знає синтаксис класів у Java та їх використання для створення об'єктів. Знає метод визначення конструктора класу, оператор створення нових об'єктів класу, процедуру визначення методів класу для виконання певних операцій над об'єктами класу, виклик методів класу на об'єктах класу. Ідентифікує різницю між статичними (static) та нестатичними (non-static) полями та методами класу. Усвідомлює поняття приватних властивостей та їх використання для приховування даних від зовнішнього доступу. Застосовує модифікатори доступу (public, private та protected) для забезпечення безпеки та контролю доступу до полів та методів класу. Ідентифікує абстрактні класи та їх відмінність від інтерфейсів.	**21**	3	3	3	3	3	3	3
2.3 Наслідування Розуміє поняття наслідування та його ролі в ООП. Вміє створювати ієрархії класів за допомогою наслідування. Розуміє механізм успадкування властивостей та методів між класами. Знає про обмеження одночасного успадкування в Java (підтримка одиночного успадкування). Вміє перевизначати (override) методи у підкласах для зміни їх поведінки. Розуміє правила доступу до членів батьківського класу з підкласу в залежності від модифікаторів доступу (public, private, protected, без модифікатора). Усвідомлює особливості заборони наслідування в final-класах. Вміє використовувати абстрактні класи та інтерфейси для визначення контрактів та створення реалізацій у підкласах. Використовує композицію та агрегацію для створення складних об'єктів та взаємодії між ними. Ідентифікує діамантову проблему (diamond problem) і розуміє способи її вирішення в Java (інтерфейси та правило ранжування методів). Може використовувати суперклас для доступу до спеціалізованих функцій підкласу та спільних ресурсів.	**19**	2	3	2	3	3	3	3
2.4 Поліморфізм								
2.4.1 Ідентифікує поняття та концепцію поліморфізму та його роль в ООП. Вміє створювати та використовувати поліморфні методи, які можуть виконувати різні дії залежно від типу об'єкта. Використовує поліморфізм для заміни одного об'єкта іншим, який реалізує однаковий інтерфейс. Вміє використовувати поліморфізм для роботи з об'єктами різних класів через спільний інтерфейс або базовий клас. Розуміє, як поліморфізм використовується у контексті успадкування. Ідентифікує переваги поліморфізму в термінах розширюваності, гнучкості та повторного використання коду. Здатний використовувати інтерфейси для створення поліморфних об'єктів та їх використання через посилання на інтерфейс.	**17**	2	2	2	3	3	2	3
2.4.2 Розуміє принцип "програмування на інтерфейсах" (programming to interfaces) для забезпечення більш гнучкої та розширюваної кодової бази. Використовує приведення типів (Type Casting) для доступу до специфічних методів підкласу через посилання на базовий клас або інтерфейс, що дозволяє змінити тип об'єкта на інший тип в межах його успадкованості. Розуміє використання поліморфізму при роботі з колекціями, такими як списки, масиви, мапи тощо, для забезпечення універсального доступу до об'єктів різних класів. Ідентифікує поліморфізм, раннє (під час компіляції) та пізнє (під час виконання програми) зв'язування, вміє використовувати поліморфізм для забезпечення обробки об'єктів підкласів з використанням їх батьківського класу. Знає про віртуальну таблицю методів (vtable) та її використання для забезпечення пізнього зв'язування. Використовує інтерфейси для визначення контрактів та реалізації поліморфізму. Усвідомлює особливості відсутності методів у маркерного інтерфейсу,який служить для позначення класу спеціальною характеристикою.	**13**	2	2	2	2	1	1	3
2.5 Об'єктно-орієнтований дизайн Керується принципами SOLID при розробці програм. Розуміє значення модульності та повторного використання коду, принципи Unix philosophy. Застосовує при розробці програм концепцію tiny types, створюючи малий, спеціалізований тип даних замість використання примітивних типів. Знає переваги застосування незмінних (Immutability) об'єктів у програмуванні. Практикує створення глибоких копій об'єктів для запобігання небажаним змінам у даних. Розуміє патерн Factory Method, який дозволяє створювати об'єкти без прив'язки до конкретного класу, використовує приватний конструктор для контролю процесу створення об'єктів та створення екземплярів статичними методами.	**15**	2	2	2	2	3	1	3
3. Розширені знання Java Core								
3.1 Робота з потоками введення-виводу								
3.1.1 Розуміє основні поняття та принципи застосування потоків введення-виводу (I/O streams) в Java. Використовує класи InputStream, OutputStream, Reader та Writer для взаємодії з різними типами даних. Застосовує FileInputStream та FileOutputStream для роботи з байтовими потоками файлів. Знає класи BufferedReader та BufferedWriter для роботи з символьними потоками файлів.	**10**	1	2	2	1	3	1	
3.1.2 Ідентифікує концепції пакету NIO та його відмінності від стандартного I/O. Знає класи та компоненти пакету NIO, такі як ByteBuffer, CharBuffer, Selector, Channel, Buffer, FileChannel та інші. Вміє використовувати неблокуючі (non-blocking) операції I/O та селектори (selectors) для ефективної роботи з багатьма потоками одночасно. Вміє створювати та керувати буферами (buffers) для оптимального використання пам'яті при роботі з даними. Розуміє принципи роботи з каналами (channels) для введення-виведення даних. Використовує селектори (selectors) для моніторингу багатьох каналів та відстеження подій вводу-виводу. Вміє використовувати буфери для зменшення часу введення-виводу та забезпечення ефективності роботи з даними. Знає про кешування та його вплив на продуктивність роботи з файлами.	**5**	1	0	2	1	0	1	
3.2 Багатопоточність у Java								

3.2.1 Ідентифікує поняття потоку (Thread) та його взаємодії зі стеком викликів (Call Stack) та пам'яттю. Знає інтерфейс Runnable, який представляє завдання, яке можна виконати в окремому потоці. Усвідомлює переваги використання багатопоточності, такі як покращення продуктивності, використання багатоядерних процесорів та покращення ресурсоємності. Використовує клас Thread та вміє створювати та запускати потоки виконання. Застосовує методи контролю над життєвим циклом потоків та взаємодії між ними (start, run, join), методи керування виконанням потоків (sleep, yield). Ідентифікує проблеми стандартної синхронізації Java та особливості використання пакету java.util.concurrent для роботи з багатопотоковістю. Розуміє проблеми багатопоточності, таких як гонки даних (race conditions), блокування та взаємоблокування (deadlocks), перегляди (data races) та інші. Вміє виявляти та уникати цих проблем шляхом використання відповідних механізмів синхронізації. Використовує ключове слово synchronized для стандартної синхронізації доступу до спільних ресурсів. Вміє використовувати монітори (monitors) та блокування (locks) для керування доступом до критичних розділів коду.	**14**	1	2	2	1	3	3	2
3.2.2 Використовує потокобезпечні алгоритми та механізми, таких як синхронізатори (synchronizers), семафори (semaphores), блокуючі черги (blocking queues), взаємні виключення (mutex) та інші. Практикує використання неблокуючих (non-blocking) алгоритмів та структур даних, такі як AtomicInteger, AtomicReference, ConcurrentHashMap. Застосовує потокові змінні (ThreadLocal variables) для забезпечення безпечного доступу до потокових даних. Використовує синхронізацію на базі стану (state-based synchronization) та примітиви замків (lock primitives) для оптимізації доступу до критичних розділів коду. Розуміє патерну Observer як спосіб реалізації асинхронної комунікації між об'єктами, вміє використовувати callback-функції для здійснення зворотного виклику та обробки результатів асинхронних операцій.	**12**	1	1	2	1	3	2	2
3.3 Колекції в Java Ідентифікує колекції, такі як Collection, List (впорядкована колекція, яка може містити дублікати елементів), Set (колекція без дублікатів елементів), Map (колекція пар "ключ-значення" з унікальними ключами), та їх особливості. Здатний обрати необхідну колекцію, враховуючи специфіку проекту та потреби програми, з погляду на поведінку колекції, складність операцій, спосіб зберігання даних та ефективність виконання певних операцій. Використовує додаткові колекції та інструменти, які надаються сторонніми бібліотеками, такими як Guava. Працює з колекціями Queue, що використовується для зберігання елементів у порядку "першим прийшов - першим вийшов" (FIFO), та Dequeue (Double-Ended Queue), що дозволяє виконувати операції вставки та вилучення елементів з обох кінців. Вміє правильно перевизначати методи equals та hashCode у власних класах для правильної роботи з HashSet, HashMap колекціями. Використовує патерн Ітератор для послідовного доступу до елементів колекції.	**20**	3	3	3	2	3	3	3
3.4 Узагальнення (Generics) Знає generics як механізм типізації, що дозволяє створювати класи та методи, що працюють з різними типами даних. Вміє створювати та використовувати прості generic-класи, такі як List<T>, Set<T>, Map<K, V> та інші. Використовує <?> (необмежений метасимвол) для роботи з необмеженими generics та ключове слово extends для вказання обмежень типу в generic-дефініціях. Здатний розробити метод generic, щоб він міг працювати з різними типами даних, використовує generic-методи для поліпшення перевикористовування коду. Розуміє синтаксис виклику статичного методу з використанням generics та вміє передати типовий параметр.	**16**	2	2	2	2	3	3	2
3.5 Stream API. Stream pipelines, Collectors Знає можливості Stream API, який введений в Java 8, для роботи з потоками даних. Вміє створювати Stream, фільтрувати, трансформувати та об'єднувати дані за допомогою різних операцій Stream API. Розуміє концепцію Stream pipelines, яка складається з проміжних та термінальних операцій. Здатний використовувати різні проміжні операції, такі як map, filter, sorted, distinct та інші, для трансформації та фільтрації даних. Знає термінальні операції, такі як collect, forEach, reduce, які завершують Stream pipeline та повертають результат. Використовує Collectors, які є вбудованими збірниками результатів для Stream API, для збору даних у колекції, обчислення статистики та іншої операцій з результатами Stream.	**20**	3	3	3	2	3	3	3
3.6 Функціональний інтерфейс, lambda-expression Знає про функціональні інтерфейси, які мають один абстрактний метод та можуть мати декілька методів за замовчуванням або статичних методів. Здатний використовувати функціональні інтерфейси, такі як Predicate, Function, Consumer, Supplier та інші. Використовує лямбда-вирази для скорочення коду та покращення зрозумілості програми, як засіб створення анонімних функцій та передачі їх як параметрів. Ідентифікує різні сценарії використання функціональних інтерфейсів та лямбда-виразів у Java, вміє передавати функції як параметри, використовувати їх для фільтрації, мапування, збору даних та інших операцій над колекціями. Усвідомлює переваги та недоліки функціонального програмування у порівнянні з імперативним стилем.	**16**	3	3	2	1	1	3	3
3.7 Анотації та рефлексія Розуміє призначення та використання анотацій для позначення спеціальних властивостей, налаштувань або метаданих в програмі, надання додаткової інформації про програмний код. Знає вбудовані анотації у Java, такі як @Override, @Deprecated, @SuppressWarnings та інші. Використовує рефлексію для динамічної взаємодії з класами та об'єктами, виконання динамічних операцій та аналізу структури коду. Знає класи Class, Method, Field, Constructor та інші засоби для роботи з рефлексією. Вміє отримувати інформацію про класи, поля та методи, створювати об'єкти за допомогою рефлексії, викликати методи та доступатись до полів. Розуміє обмеження та потенційні негативні наслідки використання рефлексії, такі як зниження продуктивності та складноці при розробці та налагодженні коду. Усвідомлює переваги використання анотацій та рефлексії для забезпечення додаткової інформації та динамічної поведінки програми.	**16**	3	2	2	1	3	2	3
3.8 Java EE								
3.8.1 Знає специфікацію для платформи Java, призначену для розробки та розгортання підприємницьких застосунків. Вміє використовувати її функціональні можливості для розробки розподілених, масштабованих та надійних додатків. Вміє працювати з Servlets та JavaServer Pages (JSP) для створення динамічних веб-сторінок і обробки HTTP-запитів. Здатний створювати та використовувати Enterprise JavaBeans (EJB) для реалізації бізнес-логіки та управління транзакціями. Застосовує Java Persistence API (JPA) для зберігання та доступу до даних у базі даних.	**7**		2	1	2	1	1	0

3.8.2 Розуміє особливості роботи з Java API для XML Web Services (JAX-WS) та Java API for RESTful Web Services (JAX-RS) для створення та споживання веб-сервісів. Практикує управління транзакціями у JEE за допомогою Java Transaction API (JTA) та роботу з розподіленими транзакціями. Використовує Java Naming and Directory Interface (JNDI) для роботи з іменами та каталогами у розподіленому середовищі. Знає про контейнери JEE, які надають середовище для виконання компонентів. Вміє розгортати та налаштовувати JEE-додатки на серверах додатків, таких як Apache Tomcat або WildFly.	**6**	3	0	1	1	1	0	0
3.9 Java-сервлети, Chain Of Responsibility								
3.9.1 Розуміє призначення та функціональність Java-сервлетів у веб-розробці. Знає життєвий цикл сервлету (ініціалізація, обробка запиту, знищення) та можливість переозначення методів сервлету. Здатний налаштовувати конфігурації сервлетів за допомогою файлу web.xml або анотацій (залежно від версії Java EE). Вміє створювати та реалізовувати Java-сервлети для обробки HTTP-запитів та надсилання HTTP-відповідей. Розуміє які методи (GET, POST, PUT, DELETE тощо) можна обробляти в сервлеті і як виконати відповідні дії для кожного методу. Здатний застосувувати фільтри (filtrs) для обробки запитів до сервлетів (HttpServletRequest) або відповідей від сервлетів (HttpServletResponse). Використовує шаблони для генерування відповідей сервлету.	**11**	3	1	2	2	1	0	2
3.9.2 Розуміє процес об'єднання сервлетів з JavaServer Pages (JSP) для відображення динамічного контенту на веб-сторінках. Вміння обробляти та виводити помилки в сервлетах. Знає Chain Of Responsibility - поведінковий паттерн проектування, який дозволяє передавати запити через ланцюжок обробників. Використовує цей патерн для послідовної обробки запитів або подій різних об'єктів без прив'язки до конкретних обробників. Усвідомлює переваги розподілу обов'язків між різними об'єктами для обробки запитів та можливості динамічної зміни ланцюжка обробників.	**5**	1	1	1	1	1	0	0
3.10 Spring Framework. MVC								
3.10.1 Вміє структурувати додаток за концепцією MVC / MVT та встановлювати взаємозв'язки між компонентами. Усвідомлює процес передачі даних між моделлю, представленням та контролером (шаблоном). Розуміє принцип одностороннього потоку даних, де модель оновлюється контролером, а представлення отримує оновлені дані з моделі. Розуміє основні концепції Spring, такі як інверсія керування (IoC) та впровадження залежностей (Dependency Injection). Здатний створювати RESTful API за допомогою Spring Boot, що дозволяє взаємодіяти з додатком за допомогою HTTP-запитів. Практикує розробку самостійних додатків, що мають вбудовані веб-сервери (Tomcat або Jetty). Вміє зазначити конфігураційні параметри додатка, такі як порт, базовий шлях, з'єднання з базою даних тощо.	**17**	2	3	3	2	1	3	3
3.10.2 Має навички роботи з базами даних за допомогою Spring Data JPA для створення і взаємодії з репозиторіями, виконання CRUD-операцій з реляційними базами даних. Використовує Spring Boot Actuator для отримання інформації про стан додатка та моніторингу його роботи. Розуміє особливості тестування додатків Spring Boot, включаючи юніт-тести, інтеграційні тести та тести контролерів. Ідентифікує Spring Security як інструмент для забезпечення безпеки веб-додатків. Вміє налаштовувати автентифікацію та авторизацію.	**15**	3	2	3	1	1	2	3
3.11 Конфігурація та складання проєкту. Maven та Gradle Усвідомлює роль Maven у проектах Java. Знає структуру POM (Project Object Model) файлу та його використання для конфігурації проекту. Ідентифікує Gradle як альтернативний складач проектів для Java. Розуміє роль Gradle build скрипту і його синтаксис. Має навички управління залежностями та підключення зовнішніх бібліотек до проекту за допомогою Maven або Gradle. Здатний керувати залежностями проекту, додавати нові бібліотеки та оновлювати існуючі. Знає життєвий цикл збирання проекту з використанням Maven або Gradle (фази, задачі тощо).	**14**		3	2	2	1	3	3
4. Знання Web-технологій. Основи HTML та CSS, JavaScript								
4.1 Загальні питання веб технологій Розуміє клієнт-серверну архітектуру. Знає основні мережеві протоколи, розуміє принцип роботи протоколу HTTP, його основні методи, статусні коди та їх значення. Вміє працювати з заголовками HTTP, формувати HTTP-запити, вказуючи метод, шлях, заголовки та дані. Може обробляти HTTP-відповіді, читати статусні коди, заголовки та вміст відповіді. Розуміє принципи взаємодії із зовнішніми API з використанням HTTP-запитів Має навички використання інструментів розробника браузера для відстеження запитів мережі та аналізу заголовків та вмісту відповідей. Вміє використовувати cookie, LocalStorage для зберігання даних користувача	**16**	2	3	2	2	3	3	1
4.2 Загальна структура HTML-документа. Теги та атрибути. Семантика Знає основні елементи HTML-документу та вміє створювати з них правильну структуру. Знає основні HTML-теги, розуміє їх призначення та особливості, вміє обирати та використовувати найбільш доречні HTML-теги для розмітки різних типів контенту. Використовує семантичні елементи для більш чіткого та структурованого опису вмісту вебсторінки. Розуміє роль та значення атрибутів та їх вплив на поведінку та відображення елементів, вміє додавати та налаштовувати атрибути для елементів HTML. Розуміє важливість валідного HTML-коду та його ролі для сумісності, доступності та оптимізації вебсторінки. Здатний перевіряти та виправляти помилки валідації HTML за допомогою валідаторів та інструментів розробника.	**9**	1	2	1	1	3	1	0
4.3 Основні CSS-властивості та селектори. CSS-анімації. Flexbox. Grid Розуміє принципи каскадності та спадкування стилів. Вміє працювати з css-властивостями, css-селекторами, та їх ієрархією. Ідентифікує поняття блокової моделі, потоку, позиціонування. Має досвід використання CSS-технік flexbox та grid, вміє налаштовувати гнучку сітку для різних екранів та пристроїв, використовує медіазапити для створення адаптивних вебсторінок. Працює з різними одиницями вимірювання	**7**	1	1	1	1	3	0	0

4.4 Основи JavaScript Знає синтаксис JavaScript, змінні, оператори та умовні конструкції. Ідентифікує основні поняття, такі як функції, масиви та об'єкти. Вміє маніпулювати DOM (Document Object Model) за допомогою JavaScript, змінювати вміст та стилі елементів, додавати та видаляти елементи. Може здійснювати обробку подій та взаємодію з користувачем. Розуміє процес здістнення асинхронних запитів до сервера за допомогою AJAX (Asynchronous JavaScript and XML). Використовує вбудовані функції JavaScript, такі як fetch(), для отримання та відправлення даних на сервер.	**8**	1	1	1	1	3	1	0
4.5 Основи Bootstrap (шаблони, CSS, компоненти, JavaScript) Знає основні можливості Bootstrap, такі як використання готових шаблонів та компонентів для швидкої розробки веб-інтерфейсу. Розуміє базові CSS-класи та структуру стилів у Bootstrap. Здатний використовувати JavaScript-компоненти Bootstrap для створення інтерактивності та покращення функціональності на веб-сторінці.	**3**	0	1	1	1	0	0	0
4.6 Розуміння роботи API (типи, логіка роботи REST-застосунків) Знає основні принципи REST, такі як ресурси, URL-шляхи, HTTP-методи та параметри запиту. Розуміє структуру URL-шляхів та їх використання для ідентифікації ресурсів і підресурсів. Володіє навичками читання та створення даних у форматах JSON і XML, ідентифікує переваги та недоліки кожного формату і вміє обирати відповідний формат для конкретного випадку використання. Здатний використовувати різні методи автентифікації, такі як токени, сесії та OAuth. Вміє налаштовувати та використовувати механізми автентифікації та авторизації у REST-застосунках. Розуміє рівні доступу та особливості використання ролей користувачів для обмеження доступу до ресурсів.	**17**	3	3	2	2	2	2	3
5. Data storing								
5.1 Relational Database design and modelling Розуміє як працюють реляційні бази даних, їх основні компоненти та принципи організації даних. Ідентифікує процес нормалізації даних для забезпечення ефективного та оптимізованого зберігання даних у реляційних таблицях. Усвідомлює особливості ідентифікації сутностей, атрибути та їх типи у контексті проектування баз даних. Володіє навичками роботи з зв'язками між таблицями: знає зв'язки один до одного, один до багатьох та багато до багатьох між таблицями у реляційних базах даних.	**20**	3	3	3	2	3	3	3
5.2 Робота з базами даних SQL Має базові знання SQL syntax, вміє будувати запити (queries) для отримання інформації з бази даних, модифікації даних та виконання інших операцій. Вміє працювати з таблицями та схемами, розуміє зв'язки між таблицями. Здатний працювати з ключами та індексами для забезпечення ефективного доступу до даних. Розуміє типи з'єднань (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN) та може використовувати їх для об'єднання даних з різних таблиць та отримання з'єднаних даних Може будувати та виконувати SELECT-запити для отримання певних даних з таблиць та фільтрацію даних на основі певних умов, здійснювати їх агрегацію та групування Вміє працювати зі схемами баз даних, встановлювати права доступу користувачів тощо	**19**	3	3	2	2	3	3	3
5.3 Робота з базами даних NoSQL Знає різні інструменти та бібліотеки для роботи з NoSQL базами даних, такі як MongoDB, Cassandra, Redis, Couchbase Розуміє різницю між ACID (Atomicity, Consistency, Isolation, Durability) транзакціями, що характерні для реляційних баз даних, та BASE (Basically Available, Soft state, Eventually consistent) підходом, що використовується в NoSQL базах Вміє працювати зі специфічними структурами даних, які забезпечуються NoSQL базами, такими як дерева, графи, документи тощо Має навички тестування взаємодії NoSQL баз даних з іншими компонентами, такими як бекенд, фронтенд, API	**11**	1	1	2	1	2	2	2
5.4 JDBC Використовує Java Database Connectivity (JDBC) для підключення та взаємодії з реляційними базами даних з додатків, написаних на Java. Вміє створювати та виконувати SQL-запити до бази даних за допомогою JDBC Statement або PreparedStatement. Практикує управління транзакціями в контексті JDBC для забезпечення цілісності та надійності бази даних. Розуміє як виконувати Batch-операції - пакетні операції для вставки, оновлення або видалення даних у базі даних, що дозволяє покращити продуктивність. Вміння обробляти винятки та помилки, які можуть виникати під час взаємодії з базою даних.	**13**	3	2	3	1	0	1	3
5.5 Робота з ORM (Hibernate, MyBatis тощо) Розуміє концепції та переваги Object-Relational Mapping (ORM), які дозволяють спростити роботу з реляційними базами даних і забезпечити зручний доступ до даних у вигляді об'єктів. Використовує анотації для визначення сутностей (entity), зв'язків між сутностями (relationship), а також конфігурації доступу до бази даних. Розуміє, як забезпечити персистентність даних, тобто збереження об'єктів Java в базі даних і їх завантаження з бази даних. Знає популярні персистентні фреймворки Hibernate або MyBatis, їх призначення, використання та можливості. Вміє налаштовувати ORM фреймворки (наприклад, Hibernate) для взаємодії з реляційною базою даних. Здатний створювати запити до бази даних за допомогою ORM-запитів (наприклад, HQL або SQL-запитів в Hibernate). Працює з сесіями, транзакціями та управління станом об'єктів в контексті ORM. Усвідомлює можливості кешування даних у ORM фреймворках для поліпшення продуктивності додатків.	**13**	2	3	3	1	0	3	1
6. Testing								
6.1 Основи тестування Знає основні принципи тестування, його типи (Unit testing, Integration testing, End to End testing) та важливість для розробки програмного забезпечення, здатний виконувати кожен з перелічених типів. Ідентифікує відмінності різних типів тестування в рівні складності та обсязі тестових сценаріїв. Вміння писати невеликі, швидкі та автоматичні тестові сценарії Unit testing для перевірки правильності роботи окремих модулів або методів. Розуміє процес інтеграційного тестування та використання фреймворків, які допомагають тестувати взаємодію між різними компонентами програмного забезпечення. Вміє проводити тестування програмного забезпечення в реальному середовищі з урахуванням всіх компонентів системи. Практикує автоматизацію процесу тестування для забезпечення ефективності і повторюваності тестових сценаріїв. Здатний аналізувати результати тестування, виявляти помилки та робити відповідні корективи, співпрацювати з розробниками для забезпечення якості та правильності коду.	**17**	3	3	3	1	3	3	1

6.2 JUnit Вміє працювати з JUnit, усвідомлює його роль у тестуванні та його відмінності від інших фреймворків. Знає основні аннотації JUnit, такі як @Test, @Before, @After, @BeforeClass, @AfterClass та їх застосування. Використовує основні методи асертів JUnit, такі як assertEquals, assertTrue, assertFalse, assertNull, assertNotNull для перевірки різних умов. Вміє створювати параметризовані тести, які дозволяють виконувати один і той же тестовий метод з різними параметрами. Розуміє життєвий цикл тестових методів та порядок їх виконання. Розробляє ефективні та репрезентативні тестові сценарії, які покривають різні аспекти програмного забезпечення.	**17**	2	3	2	1	3	3	3
6.3 Mockito Застосовує мокінг і Mockito для ефективного тестування та виявлення помилок.Розуміє що таке mock-об'єкти, як вони використовуються в тестуванні та як створити моки для заміни реальних об'єктів. Використовує стабілізацію (stubbing) методів mock-об'єктів для повернення певних значень або поведінки під час виконання тестових сценаріїв. Вміє проводити верифікацію виклику певного методу на mock-об'єкті з певними параметрами під час виконання тесту. Здатний мокувати залежності для ізоляції тестованого об'єкта від залежних компонентів. Вміє аналізувати код та визначати, де потрібно мокування, а де - реальні об'єкти.	**16**	2	2	2	1	3	3	3
6.4 Spring Testing Розуміє загальну структуру та функціональність Spring Testing Framework. Практикує тестування контролерів (класів з анотацією @Controller або @RestController) з використанням Spring MVC Test Framework. Здатний проводити тестування різних компонентів Spring, таких як сервіси (класи з анотацією @Service), репозиторії (класи з анотацією @Repository) тощо. Вміє мокувати залежності за допомогою анотацій @MockBean та ін'єктувати компоненти за допомогою анотації @Autowired. Усвідомлює особливості тестування застосувань, що використовують Spring Security, та знає як налаштовувати імітацію аутентифікації та авторизації. Розуміє як мокувати залежності Spring-компонентів для ізоляції тестових сценаріїв. Вміє перевіряти що певні методи були викликані з правильними параметрами під час виконання тесту.	**12**	1	2	2	1	1	2	3
7. Інструментарій розробника								
7.1 Системи контролю версій - GIT Розуміє git flow - методологію роботи з гілками для ефективного управління процесом розробки. Здатний створювати репозиторії та ініціалізувати проєкти з використанням Git. Розуміє та використовує команди Git (commit, push, pull, fetch, checkout, add, status, revert та інші). Вміє працювати з гілками (branches) для розробки функціональності відокремлено від основної гілки (наприклад, розробка нових функцій у власних гілках та їх об'єднання в основну гілку). Знайомий з процедурою створення пулл-реквестів для обговорення та злиття змін з різних гілок.	**18**	2	3	2	2	3	3	3
7.2 Навички використання IDE Вміє працювати з популярними IDE, такими як Visual Studio Code, IntelliJ IDEA, Eclipse, або WebStorm. Використовує редактор коду, налаштування розширень та плагінів для поліпшення продуктивності розробки. Застосовує вбудовані інструменти IDE для налагодження коду, аналізу помилок, рефакторингу та контролю якості коду.	**18**	2	3	2	2	3	3	3
7.3 Навички використання командного рядка (Terminal) Має базове розуміння навігації по файловій системі через командний рядок. Використовує команди для перегляду, створення, зміни, видалення та архівації файлів та тек через командний рядок. Застосовує через командний рядок команди для роботи з Git, таких як клонування репозиторію, створення гілок, комітів та інші.	**13**	2	3	2	2	1	0	3
7.4 Знання Linux-середовища, Docker Розуміє основні концепції та принципи роботи з операційною системою Linux. Вміє працювати з командним рядком Linux, виконувати команди, керувати файловою системою, управляти користувачами та правами доступу. Працює зі скриптовими мовами, такими як Bash, для автоматизації рутинних завдань та конфігурації середовища. Вміє створювати, налаштовувати та управляти Docker контейнерами для окремих мікросервісів. Знає Docker-команди для роботи з образами, контейнерами, мережами, томами та іншими компонентами Docker.	**11**	1	2	2	1	1	1	3
8. Розмовна англійська								
8.1 Правильна побудова фрази з узгодженням часу Вміє використовувати відповідні часові форми дієслів і структури речень для точного вираження часу в комунікації. Здатний правильно використовувати часові форми, такі як Present Simple, Present Continuous, Past Simple, Past Continuous, Future Simple, для передачі правильного значення в повідомленні.	**15**	1	3	2	2	3	2	2
8.2 Професійна іт термінологія в розмові з замовником Знає і використовує професійні IT-терміни та скорочення, які використовуються в IT-індустрії, для чіткого та зрозумілого спілкування зі замовником. Вміє перекладати складні технічні концепції на доступну мову для замовника, не знайомого з IT-галуззю.	**13**	2	2	3	1	2	1	2
8.3 Професійна іт термінологія у діловому звіті Знає і використовує спеціалізовану IT-термінологію та фразеологію при написанні ділових звітів. Вміє передати інформацію про технічні питання та проєкти зрозуміло та конкретно, використовуючи адекватні IT-терміни та абревіатури.	**14**	2	2	3	2	2	1	2
8.4 Професійна термінологія для Java Знає та розуміє специфічну термінологію, що використовується в контексті Java-розробки Розуміє та використовує специфічні терміни, патерни, алгоритми, фреймворки, бібліотеки та інші інструменти, які використовуються в Java-розробці.	**19**	3	1	3	3	3	3	3

8.5 Неформальне спілкування (small talks) Вміє вести неформальну розмову, розпочинати діалоги та підтримувати невимушену атмосферу. Знає загальновживану термінологію та фрази, які використовуються у неформальних розмовах, таких як загальні питання про погоду, цікаві події, особисті захоплення, спорт, фільми тощо.	**8**	1	1	2	1	1	1	1

Коментарі до таблиці

Оцінки компетентностей вказані станом на 2023/2024 рр., по шкалі від 0 до 3.

«3» - компетентність вкрай важлива, якщо кандидат нею не володіє, його подальше просування на вакансію неможливе.

«2» - компетентність, знання якої обов'язково знадобиться кандидату при подальшому професійному зростанні.

«1» - некритична компетентність, якою можна знехтувати.

«0» - неактуальна компетентність.

Зауваження від компаній

G5 Entertainment

3.7 - Аннотації - використовуються постійно, рефлексія - це більш досвідчений рівень

SoftServe

1.5.1 - Може описати різницю між checked та unchecked exceptions, може описати верхні рівні ієрархії exceptions, навести приклади errors, розуміє різницю між помилками виконання та помилками компіляції

1.5.2 - може бути об’єднано з 1.5.1

1.6 - В більшості випадків обізнаності щодо банального бінарного пошуку буває достатньо як індикатору знайомства кандидата з тематикою алгоритмів

2.3 - Кандидат повинен вміти описати різницю між інтерфейсами та абстрактними класами, знає про дефолтні методи інтерфейсів.

2.4.2 - Подробиці знання внтурішніх деталей імплементації віртуальної машини щодо роботи з пізнім зв’язуванням жодним чином не вимагаються, більш ніж достатньо якщо кандидат може описати, що підлягає ранньому зв’язуванню (приватні та статичні методи та доступ до полів).

3.1.2 - Для джунів знання про канали та буфери з NIO не вимагається

3.8.1 - JPA може бути перенесено звідси в 5.5.

3.8.2 - Здебільшого вимагається робта зі Spring а не з JEE

4.2 - Від Java-розробників не вимагається знання семантичної верстки, оптимізації, сумісності та доступності.

5.2 - Адміністрування (виставляти права доступу користувачів тощо) не вимагається

5.5. - Здебільшого очікується розуміння роботи з JPA, анотування entity-класів, використання репозиторіїв в Spring Data JPA, розуміння роботи декларативних механізмів транзакційності

6.2 - Не очікується обізнаності щодо особливостей старішої версії (JUnit4)

Загальний коментар - Варто додати також блок, що стосуватиметься питань архітектури та методологій розробки ПЗ.

Viseven

1.1 - Повинен дуже добре знати все описане, допускаються пробіли з складнішими колекціями

1.2 - Основні конструкції повинен знати файно. З читанням/розумінням коду допускаються складнощі, оскільки Junior

1.3 - Методи/функції повинен знати обов’язково.

1.4 - Бажано але не критично якщо не знає

1.6 - Не вимагається, буде плюсом хоча б теоретичні знання

2.3 - Мають бути гарні теоретичні знання, допускається малий практичний досвід

2.4 - Мають бути гарні теоретичні знання, допускається малий практичний досвід

3.6 - Бажано але не критично якщо не знає

7.4 - Достатньо теоретичних знань

CHI Software

2.4.2 - Рекомендуємо об'єднати з пунктом 2.4.1 як більш глибоке знання теми.

2.5 - Це важливі знання, але більш для Middle-рівня

3.1.1, 3.1.2 - опціональні знання

3.8.1 - Усі перераховані технології не є сучасними та актуальними, окрім JPA. JPA рекомендується об'єднати з пунктом 5.5

4.4. - Цей пункт та пункт 4.2 є опціональними знаннями, що більше необхідні спеціалістам Middle/Senior

7.4 - Це є плюс, але для Junior точно не must

8.2 - Ця компетенція скоріш не важлива, бо як правило розмовами з замовником займаються більш досвідчені спеціалісти. Але це, звісно, буде бонусом.

8.3 - Ця компетенція скоріш не важлива, бо звіти як правило пишуть більш досвідчені спеціалісти. Але це, звісно, буде бонусом.

Avenga

Основні вимоги: Java Core, Core Spring, Spring Boot, Developers Associate on AWS

1.3 - Методи слід розглядати як частину класу

3.3 - Guava не найкраща бібліотека колекцій. Сюди ж варто включити Streams

3.8.1 - некоректно. JSO і EJB це штуки з минулого тисячоліття. JPA це актуальний стандарт ,але без Spring Data сенсу не бачу джуну вчити це

3.9.1 - тільки в контексті 3.10

5.5 - тільки Гібернейт і тільки на основі Spring Data