№	Тема	Задание	Примерные вопросы
1	Basic REST service	1. Создать и запустить локально простейший веб/REST сервис, используя любой открытый пример с использованием Java stack: Spring (Spring Boot)/maven/gradle/Jersey/ Spring MVC. 2. Добавить GET ендпоинт, принимающий входные параметры в качестве queryParams в URL согласно варианту, и возвращающий любой hard-coded результат в виде JSON согласно варианту.	HTTP, HTTP status code Spring vs Spring boot @PathVariable, @RequestParam, @RequestBody, @ResponseBody @Controller vs @RestController JSON
2	JPA (Hibernate/Spring Data)	1. Подключить в проект БД (PostgreSQL/MySQL/и т.д.). (0 - 7 баллов) - Реализация связи один ко многим @OneToMany (8 - 10 баллов) - Реализация связи многие ко многим @ManyToMany 2. Реализовать CRUD-операции со всеми сущностями.	БД, pk, fk, виды связей, виды join JPA, ORM, Hibernate, SpringData application.properties(yml) @Repository, @Entity, @Table, @Column, @OneToMany, @ManyToOne, @ManyToMany, @JoinColumn CascadeType, FetchType Open Session in View @Transactional @EntityGraph
3	Data caching	1. Добавить в проект GET ендпоинт (он должен быть полезный) с параметром(-ами). Данные должны быть получены из БД с помощью "кастомного" запроса (@Query) с параметром(-ами) ко вложенной сущности. 2. Добавить простейший кэш в виде in-memory Map (в виде отдельного бина).	JPQL, nativeQuery Коллекции в Java (List, Set и т.д.) Collection vs Collections HashMap, equals() & hashCode(), коллизия Кэш Пагинация
4	Error logging/handling	1. Обработать 400 и 500 ошибки. 2. Добавить логирование действий и ошибок (аспекты). 3. Подключить Swagger & CheckStyle. Убрать стилистические ошибки.	Обработка ошибок, @ControllerAdvice, @ExceptionHandler, @Valid Логирование, уровни логирования, основные библиотеки Spring AOP, Aspect, Join point, Advice Паттерны проектирования Типы паттернов. Минимум по одному каждого типа + Singleton Рефакторинг
5	Batch data processing & Testing	1.Добавить POST метод для работы со списком параметров (передаются в теле запроса) для bulk операций, организовать работу сервиса используя Java 8 (Stream API, лямбда-выражения). 2. Покрытие Unit-тестами на >80% (бизнес-логика).	Bulk операции Stream API (операции), виды операций, лямбда-выражения Виды тестирования, пирамида тестирования, TDD Unit тестирование vs интеграционное тестирование JUnit & Mockito
6	Concurrency	1. Добавить сервис для подсчёта обращений к основному сервису. Счётчик должен быть реализован в виде отдельного класса, доступ к которому должен быть синхронизирован. 2. Используя jmeter/postman или любые другие средства сконфигурировать нагрузочный тест и убедиться, что счётчик обращений работает правильно при большой нагрузке.	Многопоточность в java, процессы & потоки, жизненный цикл потока Race condition & Deadlock, Monitor & Mutex & Semaphore, Atomic action Создание потока volatile, synchronized; join, wait, notify, notifyAll, sleep, yield и т.д. Нагрузочное тестирование, ELK стек Jmeter, Selenium
7	Client	1. Сделать клиентскую часть (UI) с использованием любых технологий и/или библиотек (Spring MVC, JS-фреймворки: React, Angular и т.д.) для GET запроса (OneToMany/ManyToMany). 2. Реализовать UI для добавления, удаления и обновления. P.S. Для вдохновения используйте примеры сайтов, которыми пользуетесь сами: Google, Yandex, GitHub и т.д.	"Толстый" vs "тонкий" клиент, адаптивная верстка, single-page приложения, SEO
8	Deploy	1. (0 - 7 баллов) Запустить приложение в докере. 2. (8 - 9 баллов) Разместить приложение на любой бесплатный хостинг (Render, Railway, Fly.io, Openshift и т.д.) 3. (10 баллов + 5 баллов к любой лабе) Настроить CI на GitHub (Написать скрипт в репозитории для сборки исходников в jar-файл и размещения на своем хостинге).	Докер, образ, контейнер Докер команды CI/CD