**Чек-лист для проверки ДЗ**

**Java интеграция с backend**

1. Общая проверка приложения:

1.1. Приложение доступно через предоставленную ссылку на GitHub репозиторий.

- Переходим по ссылке и проверяем доступность репозитория.

1.2. Репозиторий содержит README файл с описанием установки и запуска приложения.

- В README должны быть описаны шаги для установки зависимостей, компиляции и запуска приложения.

1.3. Студент указал, какую интеграцию он реализовал (REST, gRPC, RMI).

- Проверяем README на наличие этой информации.

2. Структура проекта:

2.1. Проект структурирован в соответствии со стандартами Spring Boot.

- Проверяем наличие директорий `src/main/java`, `src/main/resources` и `src/test`.

2.2. В проекте используются стандартные Spring Boot зависимости (например, `spring-boot-starter-web`).

- Проверяем файл `pom.xml` или `build.gradle`.

Опциональные интеграции:

Студент должен реализовать одну из следующих интеграций: REST, gRPC или RMI. Чеклист учитывает все варианты интеграций; нужно проверять только ту, которую выбрал студент.

3. Реализация REST:

3.1. Присутствует REST контроллер в проекте.

- Ищем в проекте классы с аннотацией `@RestController`.

3.2. Реализован хотя бы один эндпоинт, который отвечает на GET запросы.

- Проверяем наличие методов с аннотацией `@GetMapping` или `@RequestMapping`.

3.3. Эндпоинт возвращает корректный JSON ответ.

- Запускаем приложение и делаем запрос к эндпоинту с помощью Postman или curl. Проверяем, что возвращенный ответ соответствует ожидаемому JSON формату.

3.4. Присутствует документация Swagger для REST API.

- Проверяем, что в проекте настроен Swagger (например, наличие зависимости `springfox-swagger2` или `springdoc-openapi` в `pom.xml` или `build.gradle`, а также конфигурационные классы).

4. Реализация gRPC:

4.1. В проекте интегрирован gRPC сервер.

- Проверяем наличие зависимостей для gRPC в `pom.xml` или `build.gradle`.

4.2. Присутствуют `.proto` файлы для определения gRPC сервисов и сообщений.

- Проверяем наличие файлов с расширением `.proto`.

4.3. Реализован хотя бы один gRPC метод.

- Проверяем наличие скомпилированных классов из `.proto` файлов и их реализацию в Java коде.

4.4. Приложение корректно обрабатывает gRPC запросы.

- Запускаем gRPC клиент и проверяем взаимодействие с сервером.

4.5. Проверяем наличие классификации репозиториев: три отдельных репозитория (клиент, сервер и зависимости).

5. Реализация RPC (RMI):

5.1. Присутствует реализация RMI сервера в проекте.

- Проверяем наличие необходимого кода для запуска RMI сервера.

5.2. Реализован хотя бы один RMI метод.

- Проверяем наличие интерфейсов и классов с методами, аннотированными для RMI взаимодействия.

5.3. Приложение корректно обрабатывает RMI запросы.

- Запускаем RMI клиент и проверяем взаимодействие с сервером.

5.4. Проверяем наличие классификации репозиториев: три отдельных репозитория (клиент, сервер и зависимости).

6. Запуск и работа приложения:

6.1. Убедиться, что приложение компилируется и запускается без ошибок.

- Следуем шагам, описанным в README для запуска приложения.

6.2. Проверить, что интеграция (REST, gRPC, RMI) работает корректно.

- Проверяем работу выбранной интеграции с помощью соответствующих клиентов (Postman для REST, gRPC client, RMI client).