Examen formatif 5 (40 minutes, 14%)

Étudiant	

Analyse d’échange HTTP FORMATIF

```
POST http://10.0.2.2:8080/suite/mult
Content-Type: application/json; charset=UTF-8
Content-Length: 16
{"mult":3,"n":4}
--> END POST (16-byte body)
<-- 200 http://10.0.2.2:8080/suite/mult (334ms)
Content-Type: application/json
Transfer-Encoding: chunked
Date: Mon, 11 May 2026 15:32:20 GMT
Keep-Alive: timeout=60
Connection: keep-alive
[0,3,6,9]
<-- END HTTP (9-byte body)
```

2 points Fournissez la code de la ou les classe(s) de transfert nécessaire(s) (kotlin ou java)

Réponse :

2 points Donnez la signature de la fonction Retrofit qui a été appelée pour envoyer la requête. (annotations, types des paramètres, type de retour …)

Réponse :

1 point Décomposez l’URL de la requête en ses éléments en mentionnant leurs noms et valeurs.

Réponse :

Trace client serveur

5 points Produisez la trace d'exécution quand l'utilisateur appuie sur le bouton.

- ne faites pas la trace de l'initialisation de l'application
- vous n'avez pas à détailler les objets call et response dans la fonction onResponse
- si une ligne envoie une requête HTTP, précisez l'URL la méthode et le corps
- si une ligne envoie une réponse HTTP, précisez le code et le corps

Signature Retrofit

```
interface MessagesApi {
  @POST("/messages/{n}")
  fun envoieMessage(@Path("n") n: Int, @Body message: String) : Call<List<String>>
}
```

Instance Retrofit

```
object RetrofitInstance {
    private const val BASE_URL = "http://10.0.2.2:8080/"
    private val retrofit = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(ScalarsConverterFactory.create())
        .addConverterFactory(GsonConverterFactory.create())
        .build()
    val api: MessagesApi = retrofit.create(MessagesApi::class.java)
}
```

```
@Composable
fun MainScreen(modifier: Modifier = Modifier) {
    val n : Int = 2
    val message : String = "Grave !"
    Button(
        modifier = modifier,
        onClick = {
            println("Requête va être lancée")
            RetrofitInstance.api.envoieMessage(n,message).enqueue(object : Callback<List<String>> {
                 override fun onResponse(call: Call<List<String>>, response: Response<List<String>>)
{
                     if (response.isSuccessful) {
                         println("Liste : ${response.body()}")
                     } else {
                         println("Erreur de réponse : ${response.code()}")
                     }
                 }
                 override fun onFailure(call: Call<List<String>>, t: Throwable) {
                     println("Erreur de requête")
                 }
             })
             println("Requête envoyée")
         }
    ){ Text(text = "Envoye !") }
}
```

Code serveur

```java
@Controller
public class MessagesController {

  public List<String> messages = new ArrayList("Salut !", "Ça farte ?");

  @PostMapping(value="messages/{n}")
  public ResponseEntity assembleMessages(@PathVariable Integer n, @RequestBody String message){
    messages.add(message);
    List<String> res = new ArrayList<>();
     for(int i = 0; i < n && i < messages.size(); i++){
       res.add(messages.get(messages.size() - 1 - i));
     }
     return ResponseEntity.ok(res);
   }
}
```

No ligne	Effets (variable(s), appel, retour, affichage(s))	Pile d'appel

No ligne	Effets (variable(s), appel, retour, affichage(s))	Pile d’appel