instituto
superior de
engenharia
de lisboa

POLITÉCNICO
DE LISBOA
POLYTECHNIC
UNIVERSITY OF LISBON

Comunicação HTTP

PDM - Programação para Dispositivos Móveis

Paulo Pereira
paulo.pereira@isel.pt

PRO
DIGI

Motivação (1)

- Em geral os sistemas são distribuídos (i.e. cliente / servidor)
  - Os dados críticos estão no servidor
  - Permite regulação dos acessos
  - Lógica de negócio especificada no backend
- Dispositivos Android são uma das interfaces do sistema

Motivação (2)

- As APIs Web (a.k.a. APIs HTTP) viabilizam independência tecnológica
  - Os clientes são heterogéneos: Android, iOS, Web
  - Interfaces em diversos contextos (e.g. browser, apps, integrações)
- A presença de APIs Web é por isso quase ubíqua

Protocolo HTTP em 30s

- Modelo de interacção **pedido - resposta**
- Cada pedido é tratado de forma independente: **stateless**
- Segurança através de TLS. HTTP sobre TLS = **HTTPS**
- Suporte para **content negotiation**
  (o cliente indica o que pretende e o servidor tenta satisfazer)

Exemplo de interacção HTTP

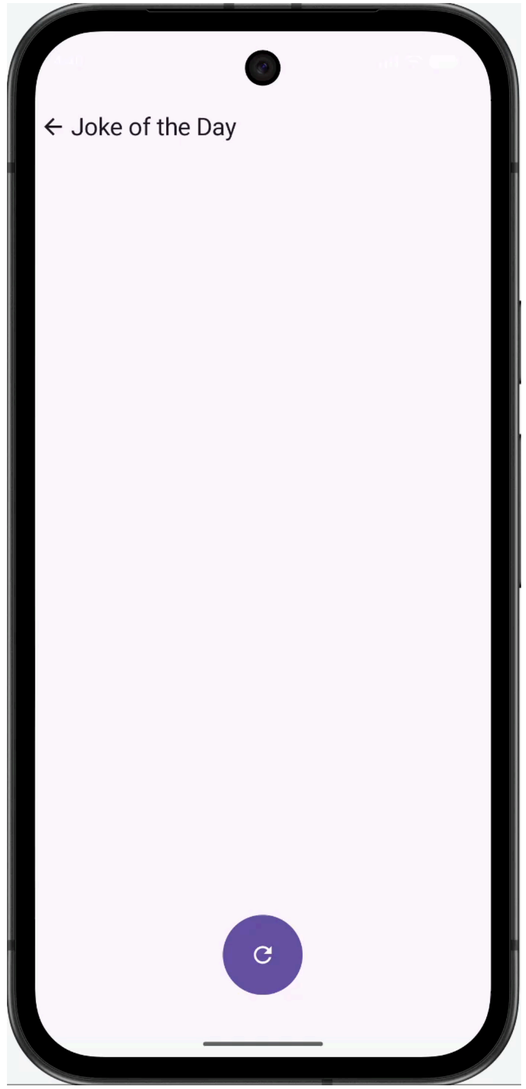
curl -H "Accept: application/json" https://icanhazdadjoke.com/

```
GET / HTTP/1.1
Host: icanhazdadjoke.com
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
...

{
  "id": "R7UfaahVfFd",
  "joke": "Why did the scarecrow win an award? Because he was outstanding in his field.",
  "status": 200
}
```

Clientes HTTP em Android

- As interacções HTTP são operações de I/O
  - Onde colocar o código correspondente?
- Como lidar com os detalhes da comunicação HTTP?
- E como lidar com a codificação/descodificação das representações externas? (i.e. JSON)

Exemplo



- Demo *Joke Of Day*
  - Apresenta piadas obtidas a partir de web APIs
  - Não considera a existência de ecrãs com tamanhos variáveis
- Implementação encontra-se aqui:
  - https://github.com/palbp/pdm.prodigi/tree/main/demos/JokeOfDay
  - Usa a API https://icanhazdadjoke.com

JokesService

```
interface JokesService {
    suspend fun fetchJoke(): Joke
}

data class Joke(val text: String, val source: URL) {
    init {
        require(text.isNotBlank()) { "The joke's text must not be blank" }
    }
}
```

- Requisito de acesso a web API modelado usando abstracção
- Operação de I/O representada através de função *suspend*

Pedidos HTTP com Ktor (1)

```
suspend fun HttpClient.request(
    url: URL,
    block: HttpRequestBuilder.() -> Unit = {}
): HttpResponse
```

- Existem variantes para os vários métodos HTTP

  Exemplo:

```
val url = URL("https://icanhazdadjoke.com/")
val response: HttpResponse = client.get(url) {
    header("accept", "application/json")
}
```

Respostas HTTP com Ktor

- Instâncias de HttpResponse viabilizam acesso às respostas HTTP
  - Incluindo *status code* e *headers HTTP*
- Suporte para descodificação do corpo da resposta
  - Via *content negotiation* (e.g. codificação JSON)
  - Transformação automática entre representações (através de bibliotecas como Gson, Jackson e Kotlinx serialization)

Implementação do serviço

```
class IcanhazDadJokes(private val client: HttpClient) : JokesService {
    private val source = URL("https://icanhazdadjoke.com/")

    override suspend fun fetchJoke(): Joke =
        client
            .get(url = source) { header("accept", "application/json") } 1
            .body<JokeDto>() 2
            .toJoke(source) 3

2   @Serializable
    private data class JokeDto(val id: String, val joke: String, val status: Int) {
        fun toJoke(source: URL) = Joke(text = joke, source = source)
    }
}
```

Configuração cliente Ktor

- A instância de HttpClient é configurável
  - Instalando extensões à biblioteca Ktor (i.e. *plugins*)
  - Indicando o *engine* HTTP a usar (dependente da plataforma alvo)
- A configuração é também determinada pelas dependências usadas

```
val httpClient = HttpClient {
    install(plugin = ContentNegotiation) {
        json(json = Json { isLenient = true; ignoreUnknownKeys = true })
    }
}
```

Dependências Ktor usadas

```
implementation("io.ktor:ktor-client-core:${ktor_version}")
implementation("io.ktor:ktor-client-content-negotiation:${ktor_version}")
implementation("io.ktor:ktor-serialization-kotlinx-json:${ktor_version}")
implementation("io.ktor:ktor-client-okhttp:${ktor_version}")
```

- Para mais informação acerca das dependências
  - https://ktor.io/docs/client-dependencies.html#add-ktor-dependencies

Documentação

- RFCs HTTP
  - HTTP/1.1 - https://www.rfc-editor.org/rfc/rfc9112.html
  - HTTP Semantics - https://www.rfc-editor.org/info/rfc9110
  - HTTP Caching - https://www.rfc-editor.org/rfc/rfc9111.html
- JSON - https://www.json.org/json-en.html
- Ktor
  - https://ktor.io/docs/client-create-and-configure.html
  - https://ktor.io/docs/client-requests.html
  - https://ktor.io/docs/client-responses.html

instituto
superior de
engenharia
de lisboa

POLITÉCNICO
DE LISBOA
POLYTECHNIC
UNIVERSITY OF LISBON

Comunicação HTTP

PDM - Programação para Dispositivos Móveis

Paulo Pereira
paulo.pereira@isel.pt

PRO
DIGI