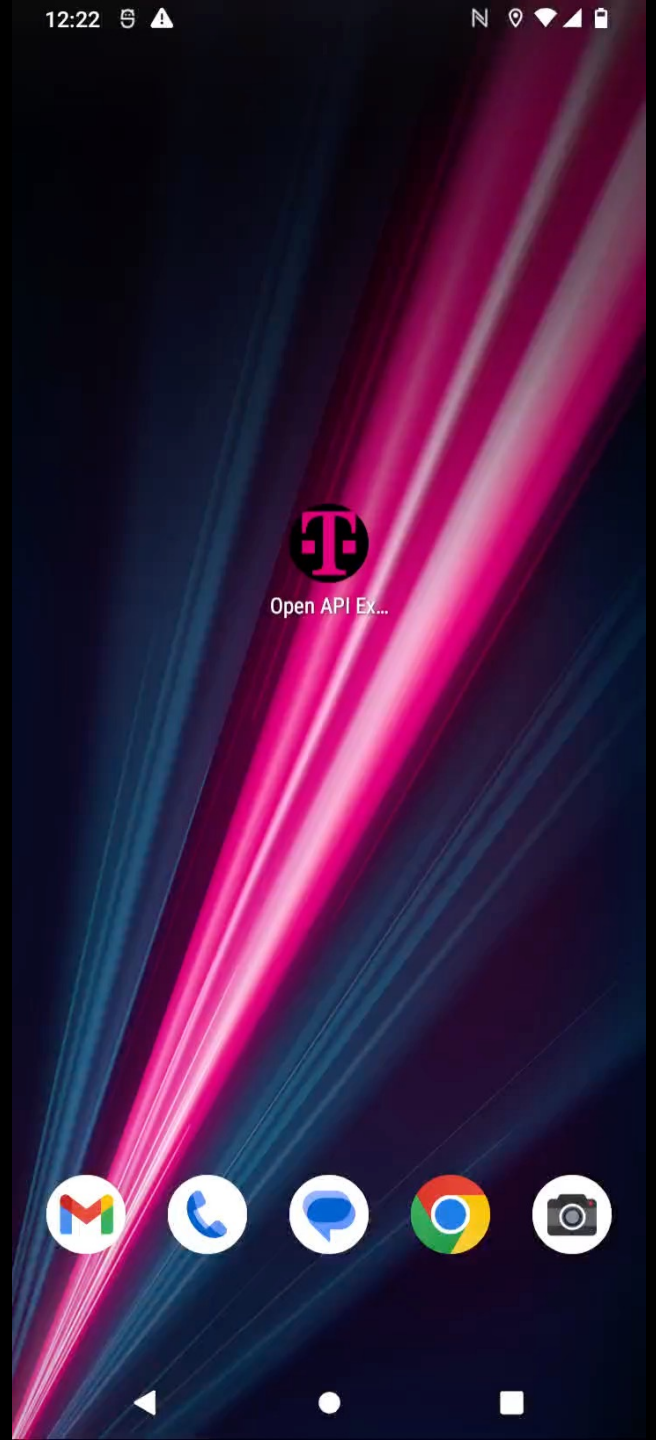
Developers

OpenAPI: Building an Android Parser

Mario Bodemann
Developer Evangelist

Telco made easy

The App



Open API Specification 

```yaml
openapi: 3.0.0
info:
 title: SMS API
 version: 1.1.7
paths:
 /messages:
  post:
   summary: Send SMS message
   responses:
    '200':
     $ref: '#/components/responses/send-sms-response'
   requestBody:
    $ref: '#/components/requestBodies/send-sms-request'
components:
 securitySchemes:
  auth:
   name: X-API-Key
   type: apiKey
   in: header
 requestBodies:
  send-sms-request:
   content:
    application/x-www-form-urlencoded:
     schema:
      type: object
      properties:
       From:
        type: string
        description: Telephone number in E.164 format, Sender ID, or short code.
       To:
        type: string
        description: Telephone number in E.164 format.
       Body:
        type: string
        description: Text body of the SMS message.
      required:
       - From
       - To
       - Body
 responses:
  send-sms-response:
   content:
    application/json:
     schema:
      type: object
      properties:
       sid:
        type: string
        description: The SMS message identifier.
      required:
       - sid
servers:
 - url: https://api.telekom.com/service/sms/v1
```

The Spec - Meta

```
openapi: 3.0.0
info:
  title: SMS API
  version: 1.1.7
```

The Spec - Meat

```
paths:
  /messages:
    post:
      summary: Send SMS message
      responses:
        '200':
          $ref: '#/components/responses/send-sms-response'
      requestBody:
        $ref: '#/components/requestBodies/send-sms-request'
```

The Spec - Components - Auth

```yaml
components:
  securitySchemes:
    auth:
      name: X-API-Key
      type: apiKey
      in: header
```

The Spec - References

```
    requestBody:
      $ref: '#/components/requestBodies/send-sms-request’
...
components:
  requestBodies:
    send-sms-request:
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:
              From:
                type: string
```

The Spec - Variables

```
send-sms-request:
  content:
    application/x-www-form-urlencoded:
      schema:
        type: object
        properties:
          From:
            type: string
            description: Telephone number in E.164 format, Sender ID, or short code.
        required:
          - From
          - To
          - Body
```

Developers

The App

Telco made easy

APIs

```
POST /verify
Execute location verification for a
user equipment

body
application/json

object
  accuracy* : number
  latitude* : number as double
  longitude* : number as double
  ueId* : object
      externalId : string
      ipv4Addr : string as ipv4
      ipv6Addr : string as ipv6
      msisdn : string
  uePort : integer
```

Execute

App Details

Basics

- Jetpack Compose
- KotlinX Serialization
- Retrofit
- OkHttp

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)

  viewmodel.loadLastAPI()

  setContent {
    val api by remember { viewmodel.api }
    val apiCalls by remember { viewmodel.apiCalls }
    val toggled by remember { viewmodel.toggled }
    val dialog by remember { viewmodel.dialog }
    val error by remember { viewmodel.error }

    OpenApiAppView(api, apiCalls, toggled, dialog, error)
  }
}
```

App Details New

com.charleskorn.kaml
https://github.com/charleskorn/kaml

```
class OpenApiParser {
    companion object {
        fun parse(ymlDescription: String): ApiSpecification {
            val yaml = Yaml( )

            val raw = yaml.parseToYamlNode(ymlDescription)

            val specification = yaml.decodeFromString<ApiSpecification>(ymlDescription)

            return specification.resolveReferences(raw.yamlMap)
        }
    }
}
```

References Parsed

Method

1. parse once with KAML:

```
@Serializable
data class RequestBody(
    @SerialName("\$ref")
    val reference: String? = null,
    val description: String? = null,
    val content: Map<String, Content>? = null,
    val required: Boolean = false,
)
```

References Parsed 2.0

2. traverse all fields and check its **reference** field

```
private fun RequestBody.resolveReferences(rawMap: YamlMap)
    return if (reference ≠ null) {
        rawMap.resolveReferences(reference).toRequestBody(
    } else {
```

3. Find it in references **components** section of specification
4. Convert found component into **RequestBody** (or whatever contains the **reference** field)
5. repeat

User Input Generation

- user clicked on execute
- find all parameter and variables
- traverse specification (server, and selected operation, ...)
- remember user input by path
- build user dialog

```
for ((name, content) in
operation.requestBody?.content ?: emptyMap()) {
    val schema = content.schema
    if (schema != null) {
        val fromBody =
parseSchemaForParameters(name, schema)
        result.putAll(fromBody)
    }
}
```

```
for (parameter in operation.parameters.orEmpty()) {
    val key = parameter.name
    result[key] =
        UserInput(... )
}
```

```
for ((index, server) in api.value?.servers.orEmpty().withIndex()) {
    for (variableName in server.variables?.keys.orEmpty()) {
        val variable = server.variables?.get(variableName)!!
        val key = "baseUrl.$index.$variableName"
        result[key] = UserInput( ... )
    }
```

```
for ((key, auth) in api.value?.components?.securitySchemes.orEmpty()) {
    if (key == "auth") {
        result[key] = UserInput(. ...    )
    }
}
```

Dialog Building

- build dialog based on found variables and parameters
- for each parameter: use its path and a default or saved value as input
- once confirmed
  - save user input to shared preferences
  - iterate through input to build api call

```
items(entries) {
    val key = it.key
    val input = it.value
    val required = input.required

    var text by remember { mutableStateOf(input.previous) }
    parameters = parameters.mute(key, text)

    Column {
      Text("$key${if (required) " *" else ""}")
      TextField(
        value = text,
        maxLines = 1,
        onValueChange = { changedValue ->
          text = changedValue
          parameters = parameters.mute(key,
changedValue)
        })
    }
```

Filing Data & Calling

- traverse spec again
- find the operation
- get user input from dialog
- fill in any parameters and variables
- execute call using okhttp
- show result


```
var builder = Request.Builder().url(url)
builder.post(
  operation.toRequestBody(userParameters)
)

val request = builder.build()

withContext(Dispatchers.IO) {
    val call = client.newCall(request)
    apiCalls.value = apiCalls.value +
call.execute().toApiCall()
}
```


Success
!WE HAVE TRAVERSED ALL!

What was that?

- Traversed the file for creating the initial structure
- Traversed the structure to fill in all references
- Traversed to display all operations
- Traversed to find all parameters and variables
- Traversed to fill in all user data
- Finally called the operation

Next steps

- MORE TRAVERSALS!
  - Or maybe less: Room for optimization?
- Better authorization handling
  - Oauth2?
- Markdown support
  - Images
- Repeation of calls
  - PLAY IT SAM
- UI / UX
- Export to curl?

Developers
Telco made easy

Thankeschön

stay tuned at https://github.com/dt-developers.

Developers
Telco made easy

Q’n’A