
# Protobuf x Android: Meet the other way to structure your data

**Trung Le** - Engineer Lead @Nimble

Hi 👋

Sawadee krub 🇹🇭

Xin chào 🇻🇳


# Protobuf x Android: Meet the other way to structure your data

**Trung Le** - Engineer Lead @Nimble

# Agenda

- **Introduction to Protocol Buffer**
- Use cases
- Q&A

# Let’s get started 

# What is “structured data”?

```
<script type="application/ld+json">
  {
    "@context": "https://schema.org",
    "@type": "Organization",
    "name": "Example",
    "url": "http://www.example.com",
    "logo": "http://www.example.com/images/logo.png"
  }
</script>
```

# Structured data

Data that describes the content of digital documents.
An invisible layer of information that machines use to read the content.

# Common types

For Android developers:

- **XML:** structure UI layout; store Preferences.
- **JSON:** to structure data when communicate with the backend API.

# Common types (Cont.)

**Pros:**

👍 Human-readable and machine-readable.

👍 Tools

👍 Great community support community

# Common types (Cont.)

## Cons:

- 👎 Verbose (for XML).
- 👎 Deficient when parsing a large and complex file (XML).
- 👎 Not type-safe, or null-safe.

We need something more compact and efficient.

So Protocol Buffer? 🤔

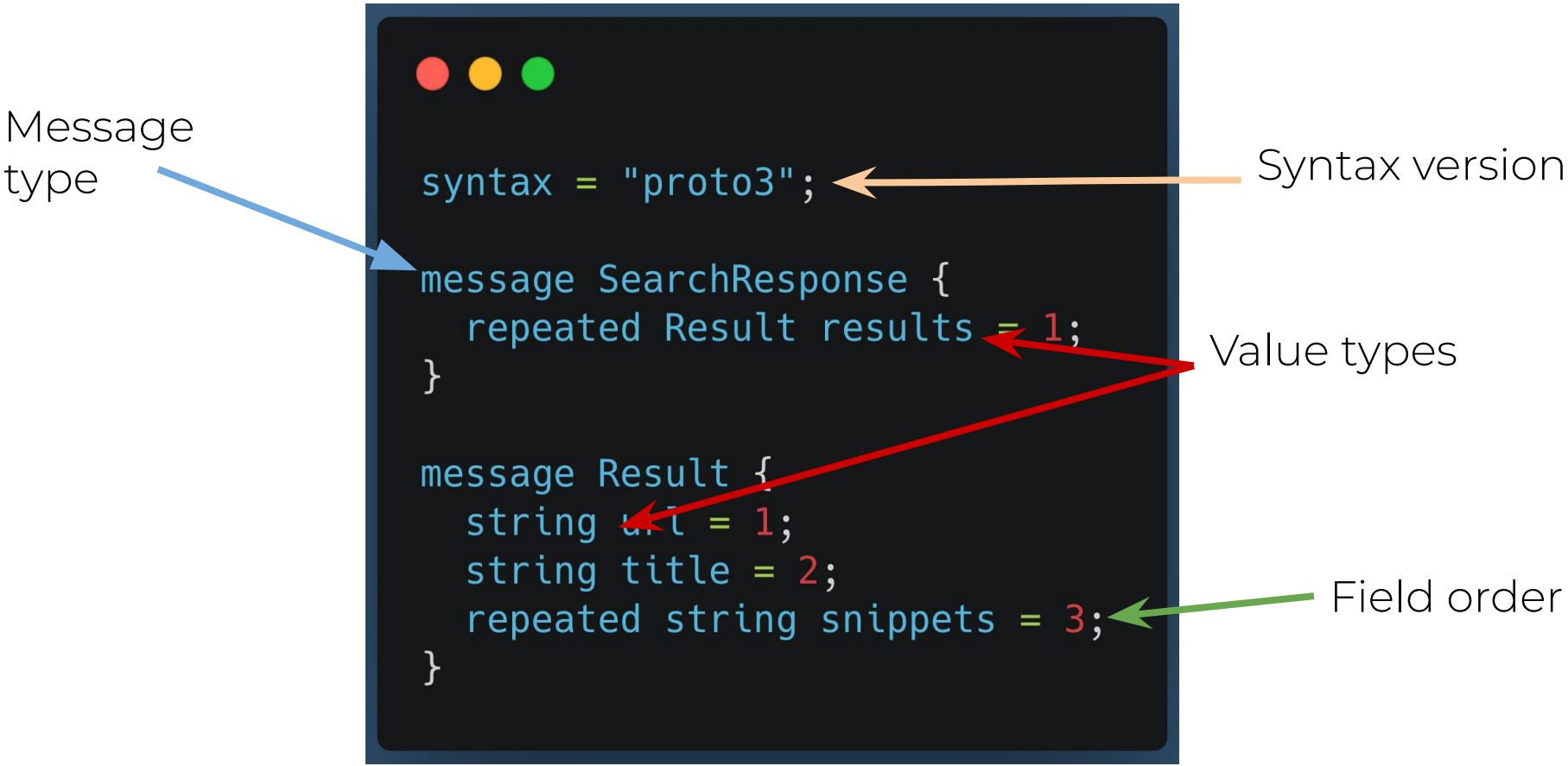
How many people have worked with protobuf files? 🙋‍♂️🙋‍♀️

How many of people know about it but yet to work with it in a real project? 🤔 🤓

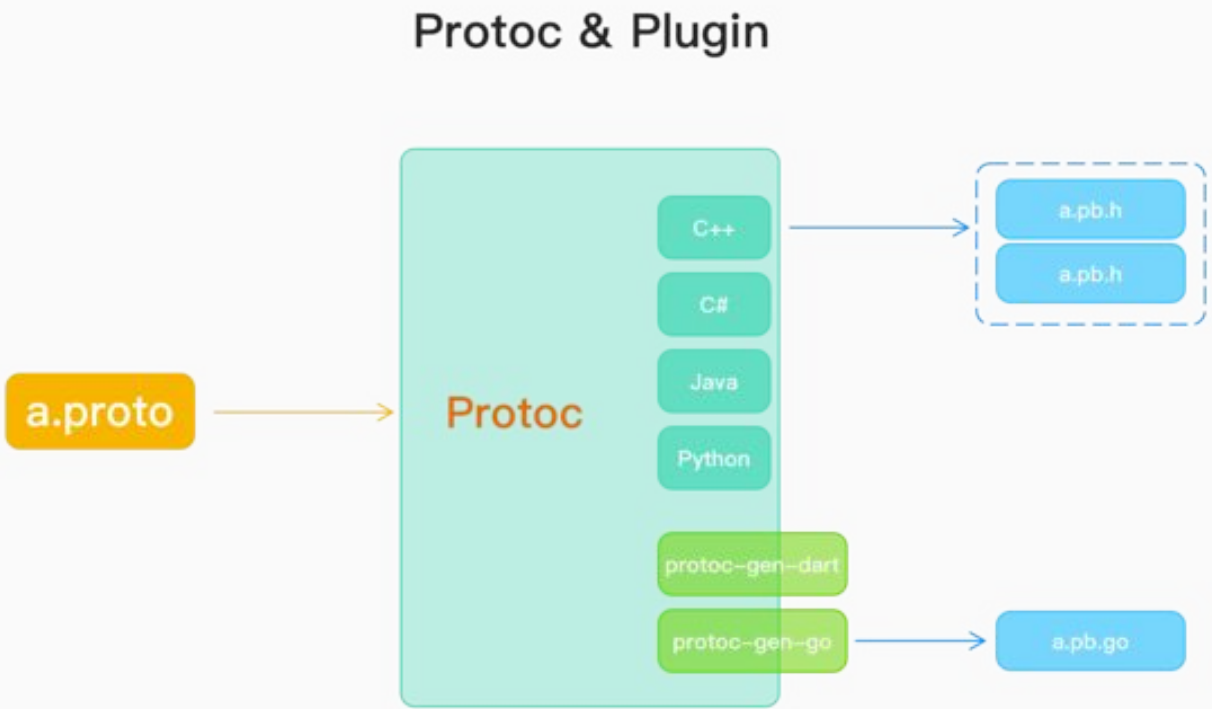
How many people: “what the heck is protobuf?” 😣 🤷‍♂️🤷‍♀️

# Protocol Buffer (protobuf)

# How it looks like



# Serialize the data



# With: id = 42

```
<id>42</id>
```

XML:  11 bytes if we assume ASCII or UTF-8 encoding and no whitespace.

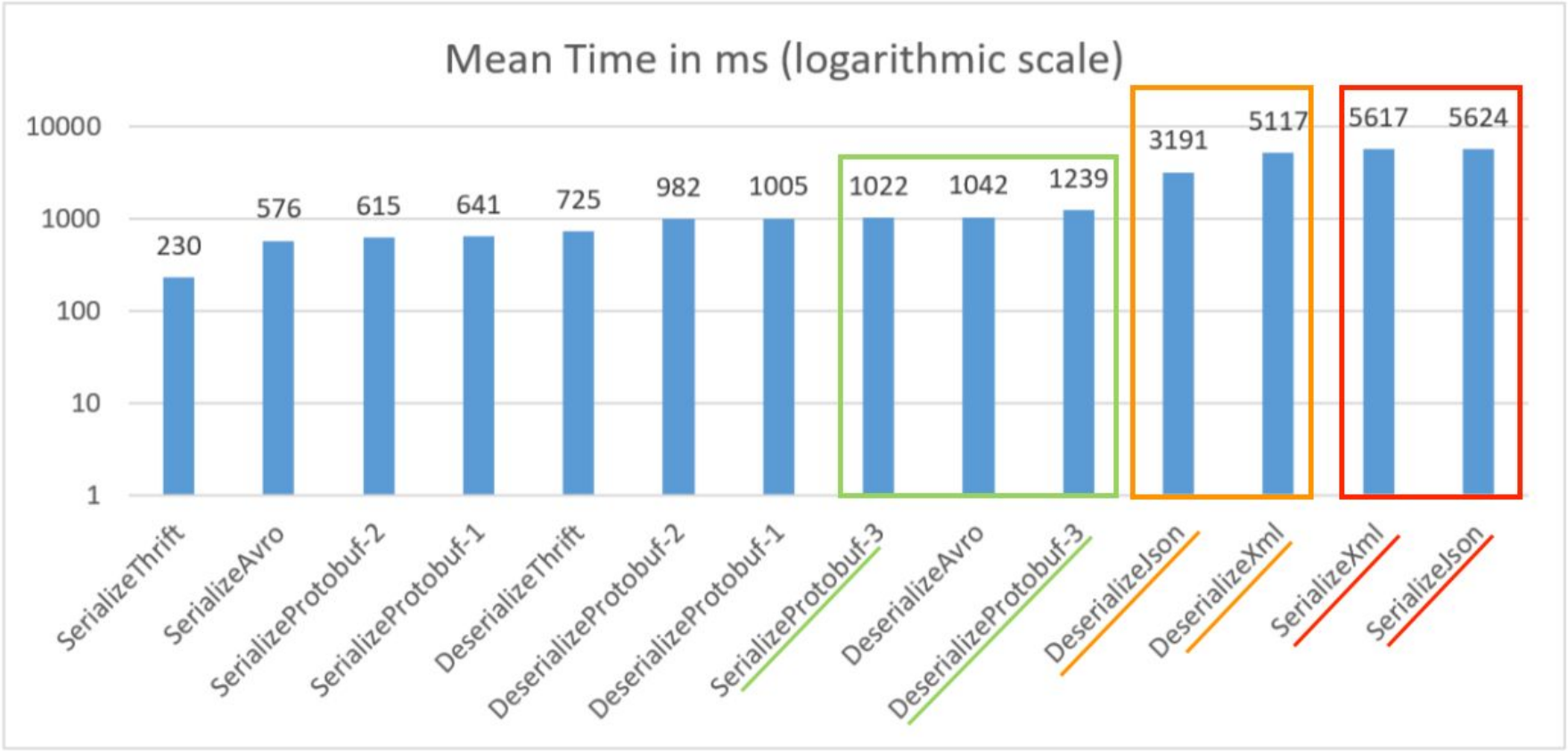
```
{"id":42}
```

JSON: 9 bytes if we assume ASCII or UTF-8 encoding and no whitespace.

```
08 2A
```

Protobuf binary: 2 bytes.

# Speed Comparison




*Source: https://labs.criteo.com/2017/05/serialization/*

# Comparing to JSON or XML

| JSON/XML | PROTOBUF |
| --- | --- |
| Transmit data with metadata details | Compresses the data and generates dense data |
| Is human-readable | Not human-readable 🤖 |
| Bigger size | Smaller size |

Let’s talk about how these things work in Android 🤔

# Agenda

- Introduction to Protocol Buffer
- **Use cases:**
  - Consuming data from API Service.
  - Local data storage with Proto DataStore
- Q&A

# Agenda

- Introduction to Protocol Buffer
- Use cases:
  - **Consuming data from backend API**
  - Local data storage with Proto DataStore
- Q&A

# The common

The API documentation - the source of truth, can be messed up sometimes 😣

# Protobuf: The source of truth

```
// Basically provide all the rpc calls to serve the Helpdesk purpose.
service HelpdeskService {
  /* Need help? Want to ask a question?
     Make an Inquire request along with the Inquiry detail,
     it returns Result */
  rpc Inquire(Inquiry) returns Answer{}
}

message Inquiry {
  // Describe what is your problem, mandatory field.
  string question = 1;
}

// The response of an Inquire request.
// Client can check the Inquiry message type for more detail. See [Type]
message Answer {
  // To distinguish the validation of the original inquiry/question.
  enum Type{
    VALID = 0;
    INVALID = 1;
    NONSENSE = 2; // You better ask something else.🤬
  }

  Type type = 1;
  string answer = 2; // Get your answer here
}
```

I have 2 protobuf files.

Let’s make a sample app 👨💻

# dinosaur.proto

```
syntax = "proto3";

message Dinosaur {
  /** Common name of this dinosaur, like "Stegosaurus". */
  string name = 1;

  /** URLs with images of this dinosaur. */
  repeated string picture_urls = 2;

  double length_meters = 3;
  double mass_kilograms = 4;
  com.squareup.Period period = 5;
}

message Dinosaurs {
  repeated Dinosaur dinosaurs = 1;
}
```

# periods.proto

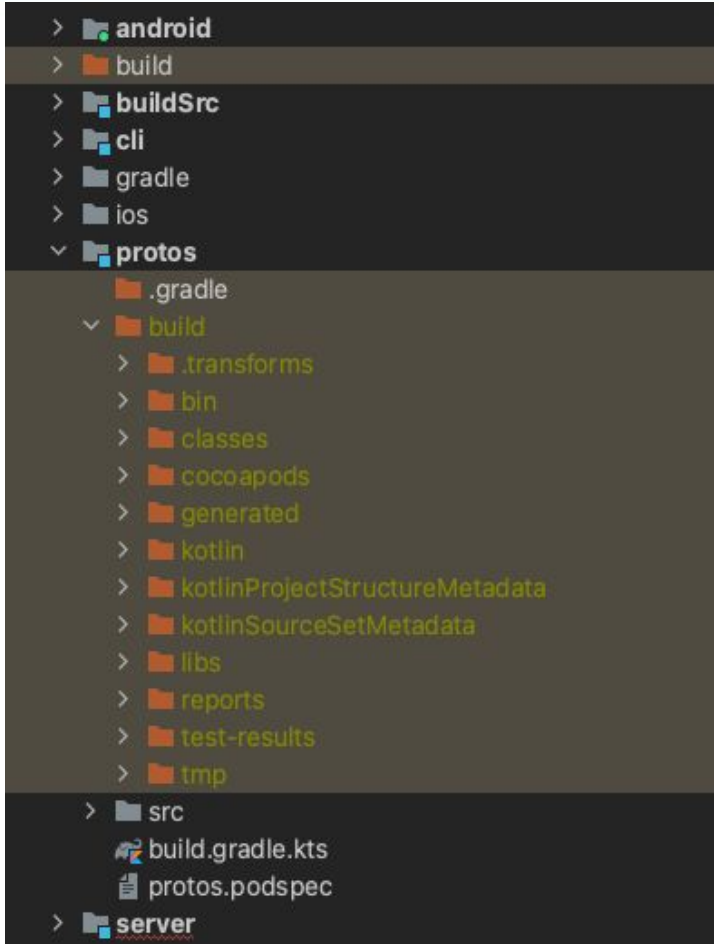
```proto
syntax = "proto3";

enum Period {
  /** Period undefined. */
  UNKNOWN = 0;

  /** 145.5 million years ago — 66.0 million years ago. */
  CRETACEOUS = 1;

  /** 201.3 million years ago — 145.0 million years ago. */
  JURASSIC = 2;

  /** 252.17 million years ago — 201.3 million years ago. */
  TRIASSIC = 3;
}
```

# Generate compiler choices

- Google: **protoc** https://github.com/protocolbuffers/protobuf
- Square: **Wire** - https://github.com/square/wire
- **Kroto-plus** - https://github.com/marcoferrer/kroto-plus

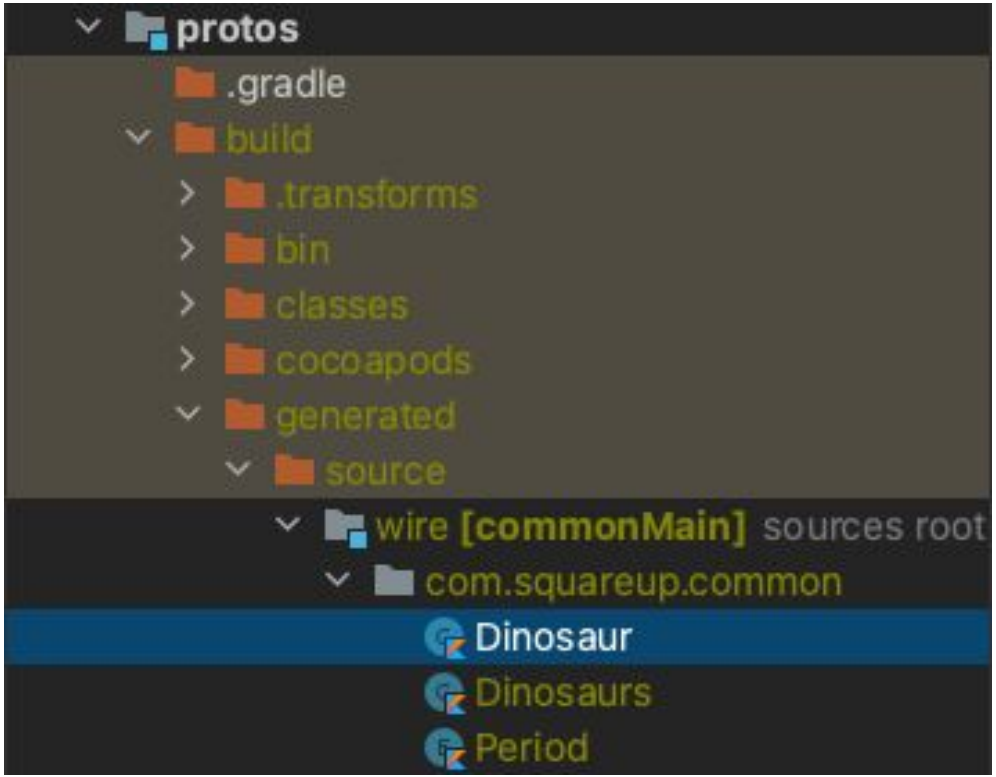
# Ingredients for the demo

- **Wire**: to compile the protobuf file.
- **Kotlin Multiplatform**
- **Ktor**: to be the server side application.




```
$ ./gradlew protos:build
```

# The generated stubs




```
import ...

class Dinosaur(
  /**
   * Common name of this dinosaur, like "Stegosaurus".
   */
  @field:WireField(
    tag = 1,
    adapter = "com.squareup.wire.ProtoAdapter#STRING",
    label = WireField.Label.OMIT_IDENTITY
  )
  val name: String = "",
  picture_urls: List<String> = emptyList(),
  @field:WireField(
    tag = 3,
    adapter = "com.squareup.wire.ProtoAdapter#DOUBLE",
    label = WireField.Label.OMIT_IDENTITY,
    jsonName = "lengthMeters"
  )
  val length_meters: Double = 0.0,
  @field:WireField(
    tag = 4,
    adapter = "com.squareup.wire.ProtoAdapter#DOUBLE",
    label = WireField.Label.OMIT_IDENTITY,
    jsonName = "massKilograms"
  )
  val mass_kilograms: Double = 0.0,
  @field:WireField(
    tag = 5,
    adapter = "com.squareup.common.Period#ADAPTER",
```

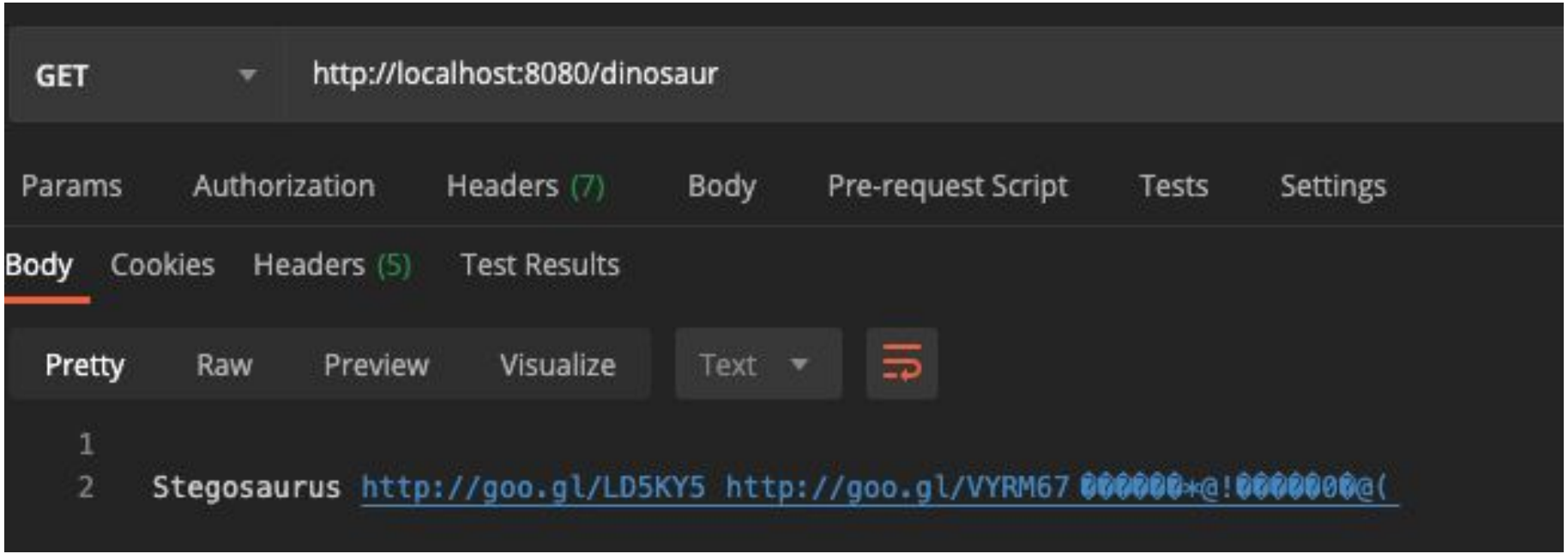
# A quick sneak into the server side

```
fun Application.main() {
  install(DefaultHeaders)
  routing {
    get("/hello") {
      call.respond("Hello there!")
    }

    get("/dinosaur") {
      call.respondBytes(
        bytes = Dinosaur.ADAPTER.encode(dinosaurs.first()),
        contentType = ContentType("application", "protobuf"),
        status = HttpStatusCode.OK
      )
    }
  }
}
```

# Data set

```
val dinosaurs = listOf(
  Dinosaur(
    name = "Stegosaurus",
    period = JURASSIC,
    length_meters = 13.0,
    mass_kilograms = 8800.0,
    picture_urls = listOf("http://goo.gl/LD5KY5", "http://goo.gl/VYRM67")
  ),
  Dinosaur(
    name = "T-Rex",
    period = CRETACEOUS,
    length_meters = 9.0,
    mass_kilograms = 5000.0,
    picture_urls = listOf("https://bit.ly/3my1mk0")
  )
)
```

# A look into the network call



# Decode

Choose File response.pb

Decode

## Results

Field #1: 0A String Length = 11, Hex = 0B, UTF8 = "Stegosaurus"
Field #2: 12 String Length = 20, Hex = 14, UTF8 = "http://goo.gl/LD ..." (total 20 chars)
Field #2: 12 String Length = 20, Hex = 14, UTF8 = "http://goo.gl/VY ..." (total 20 chars)
Field #3: 19 Fixed64 Value = 462350796744923545 6, Hex = 00-00-00-00-00-00-2A-40
Field #4: 21 Fixed64 Value = 4666063465490677760, Hex = 00-00-00-00-00-30-C1-40
Field #5: 28 Varint Value = 2, Hex = 02

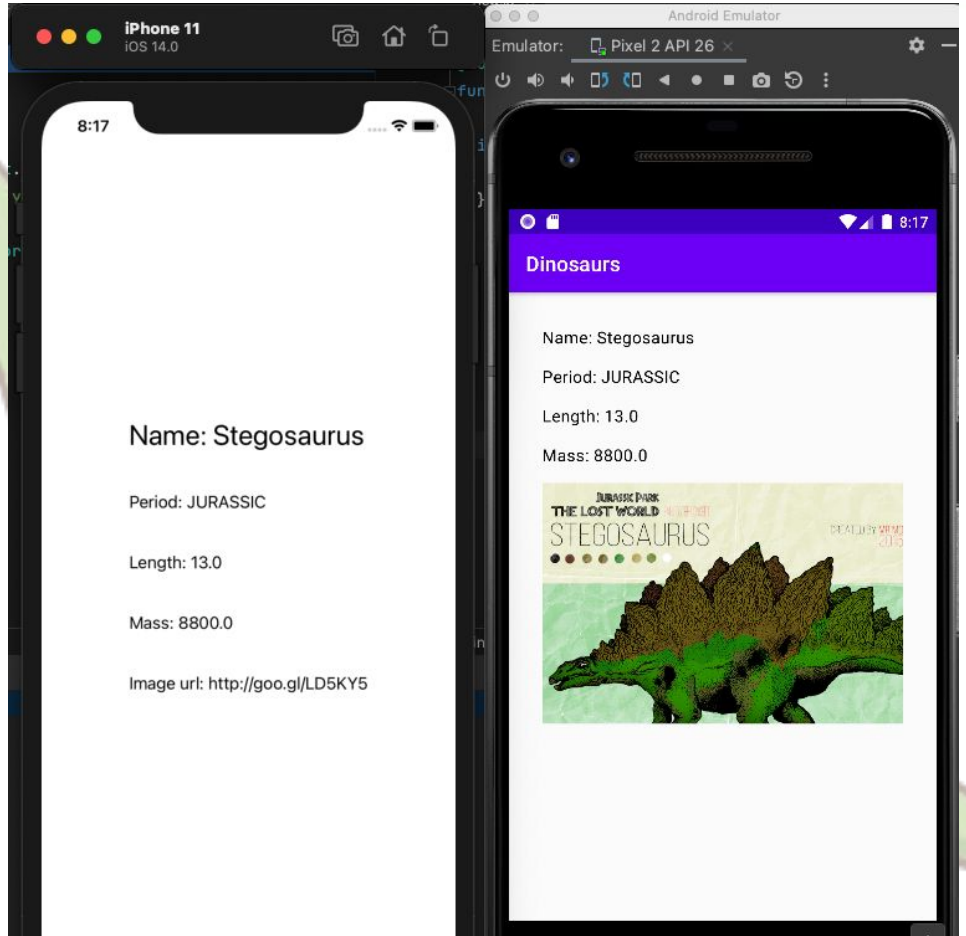
# On Android

```
interface DinosaursApi {
  @GET("/dinosaur")
  suspend fun getDinosaur(): Dinosaur
}

object DinosaurRetrofit {
  val dinosaursApi: DinosaursApi = Retrofit.Builder()
    .baseUrl("http://10.0.2.2:8080")
    .addConverterFactory(WireConverterFactory.create())
    .build()
    .create(DinosaursApi::class.java)
}
```

# Launch it

```
scope.launch {
      val dinosaur = withContext(Dispatchers.IO) {
        try {
          dinosaursApi.getDinosaur()
        } catch (ex: Exception) {
          null
        }
      }
      setContent {
        DinosaursTheme {
          DinosaurView(dinosaur)
        }
      }
    }
```

# Sample apps on Android and iOS



Repository: https://github.com/nimblehq/wire-multiplatform-sample

README.md

# Protobuf meets Kotlin Multiplatform

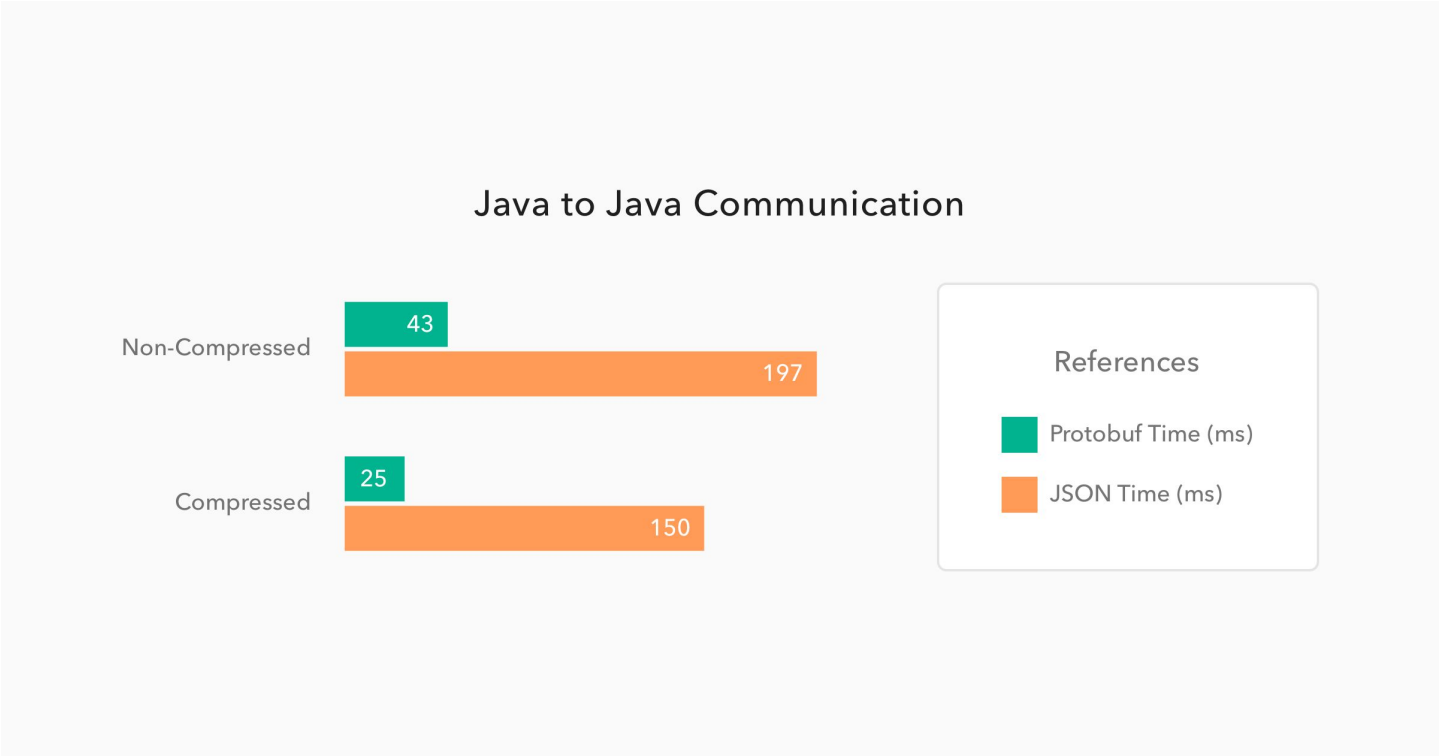
An example of building apps powered by Wire and Kotlin Multiplatform, serving protobuf response.

## Documentations

- Protocol Buffers: https://developers.google.com/protocol-buffers
- Wire: https://square.github.io/wire/
- Kotlin Multiplatform: https://kotlinlang.org/docs/reference/multiplatform.html
- Ktor: https://ktor.io/docs/quickstart-index.html
- Jetpack Compose (Android): https://developer.android.com/jetpack/compose/

## Getting started

# Benchmark




*Source: Auth0*

Where do I go after this? 🤔

⇒ gRPC

What else can we do with protobuf? 🤔

# Agenda

- Introduction to Protocol Buffer
- Use cases:
  - Consuming data from API
  - **Local data storage with Proto DataStore**
- Q&A

# Jetpack DataStore

# What is DataStore? 🤔

- A storage solution aimed at replacing **SharedPreference**.
- Store key-value pairs or typed object.
- It uses Kotlin, Coroutines and Flow to store data asynchronously.

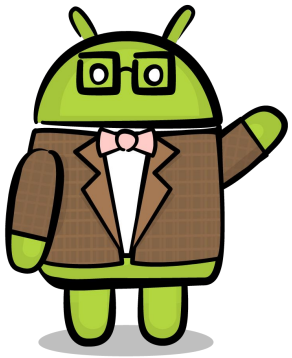
# SharedPreference - checkmate!

SharedPreferences has a synchronous API that's not safe to call on UI Thread.

The apply() blocks the UI thread on fsync()*.

* Apparently, even calling fsync() on a different thread might be useless, Redis co-creator was also doubtful about this behavior from the Linux kernel.

# SharedPreference vs DataStore



| Feature | SharedPreferences | Preferences DataStore | Proto DataStore |
|---|---|---|---|
| Async API | ✅ (only for reading changed values, via listener) | ✅ (via `Flow`) | ✅ (via `Flow`) |
| Synchronous API | ✅ (but not safe to call on UI thread) | ❌ | ❌ |
| Safe to call on UI thread | ❌* | ✅ (work is moved to `Dispatchers.IO` under the hood) | ✅ (work is moved to `Dispatchers.IO` under the hood) |
| Can signal errors | ❌ | ✅ | ✅ |
| Safe from runtime exceptions | ❌** | ✅ | ✅ |
| Has a transactional API with strong consistency guarantees | ❌ | ✅ | ✅ |
| Handles data migration | ❌ | ✅ (from SharedPreferences) | ✅ (from SharedPreferences) |
| Type safety | ❌ | ❌ | ✅ with Protocol Buffers |

*Source: Android Dev Blog (@FMuntenescu)*

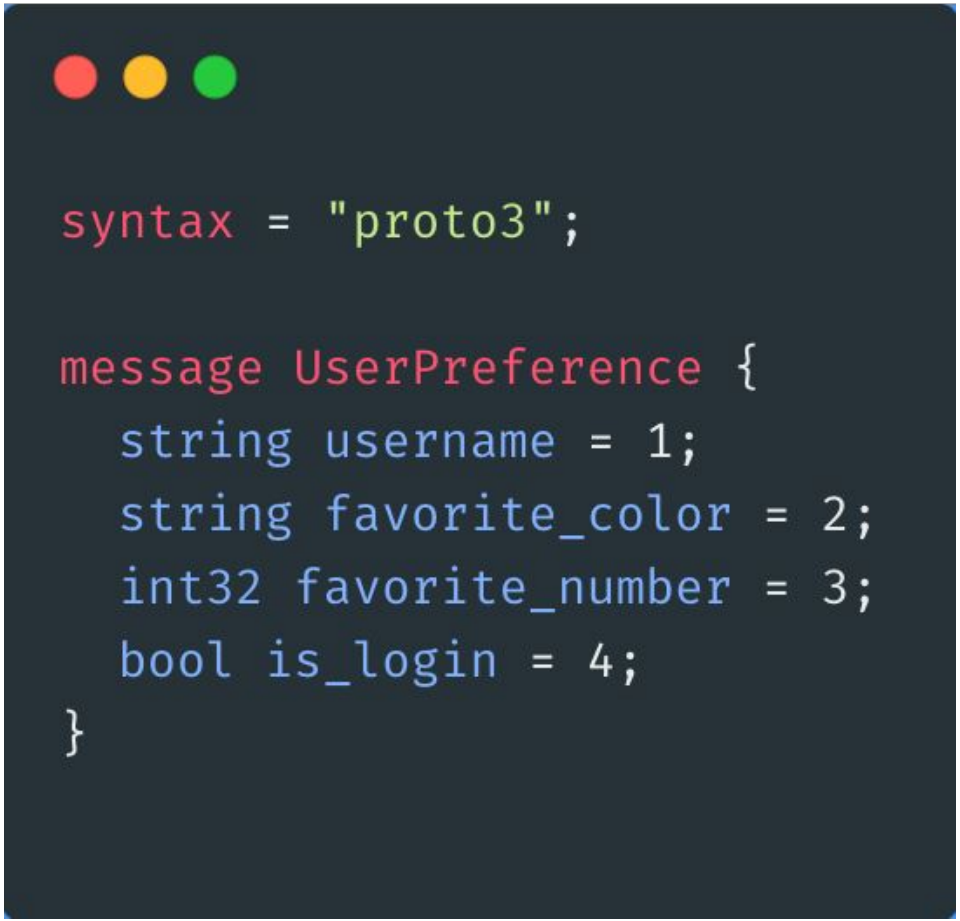
## Preferences DataStore

⇒ The upgraded `SharedPreferences`

## Proto DataStore

⇒ *Protocol Buffers*

# Demo



```
syntax = "proto3";

message UserPreference {
  string username = 1;
  string favorite_color = 2;
  int32 favorite_number = 3;
  bool is_login = 4;
}
```

# Ingredients

- **Protoc**: to compile the protobuf file.
- **Proto DataStore**: to read/write data.

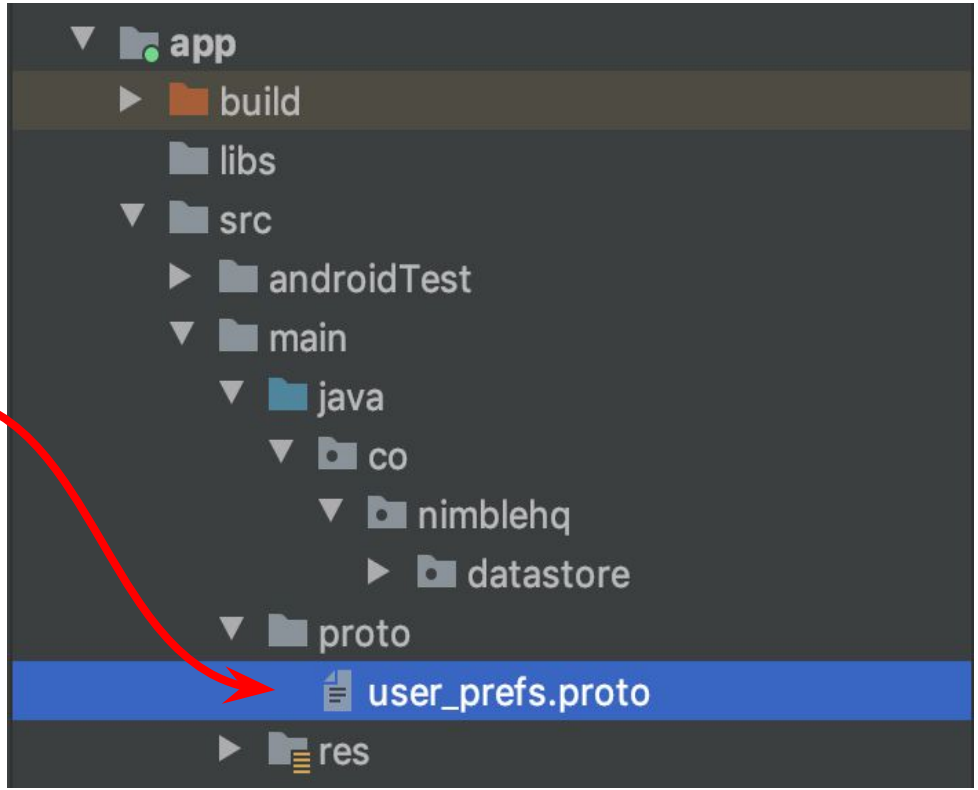
# Add dependencies

Configure ‘protoc’
At your module
build.gradle

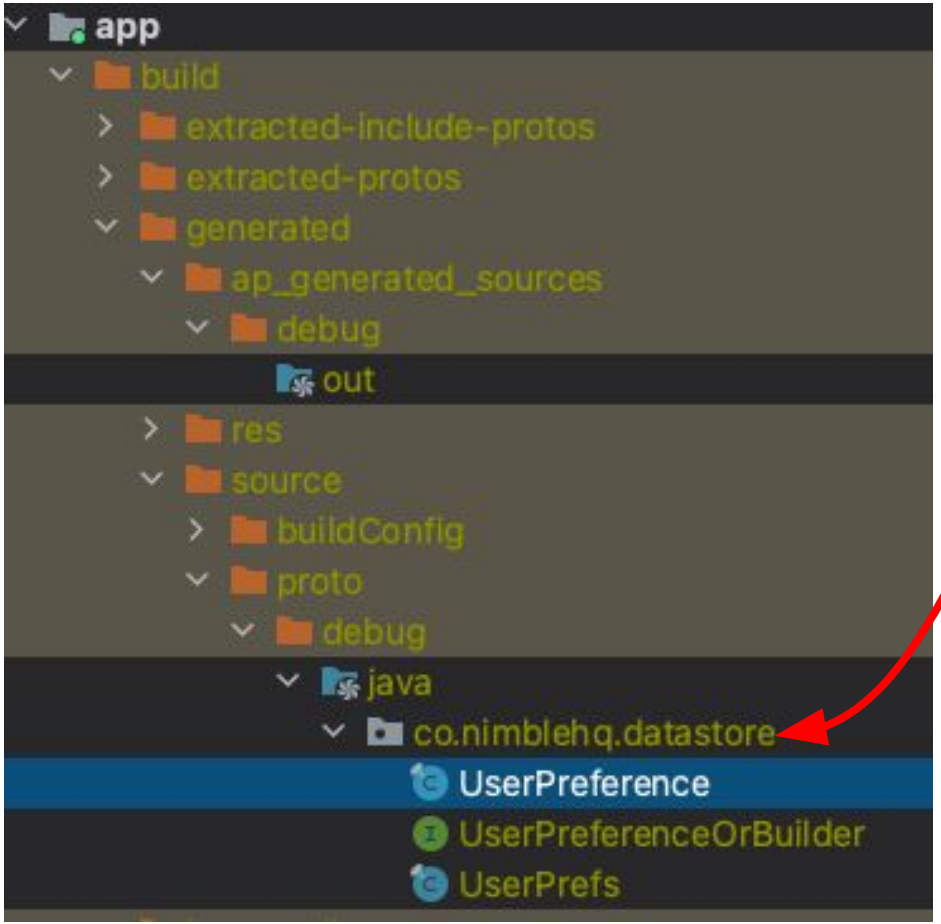
```
plugins {
    ...
    id "com.google.protobuf" version "0.8.14"
}

dependencies {
    implementation  "androidx.datastore:datastore-core:1.0.0-alpha05"
    implementation  "com.google.protobuf:protobuf-javalite:3.11.0"
    ...
}

protobuf {
    protoc {
        artifact = "com.google.protobuf:protoc:3.11.0"
    }

    generateProtoTasks {
        all().each { task ->
            task.builtins {
                java {
                    option 'lite'
                }
            }
        }
    }
}
```

# Define a schema in proto file

```
syntax = "proto3";

option java_package = "co.nimblehq.datastore";
option java_multiple_files = true;

message UserPreference {
  string username = 1;
  string favorite_color = 2;
  int32 favorite_number = 3;
  bool is_login = 4;
}
```



# Build




Generated stubs

# Create Serializer for Proto class

```
object UserPreferenceSerializer : Serializer<UserPreference> {
    override val defaultValue: UserPreference
        get() = UserPreference.getDefaultInstance()

    override fun readFrom(input: InputStream): UserPreference {
        try {
            return UserPreference.parseFrom(input)
        } catch (exception: InvalidProtocolBufferException) {
            throw CorruptionException("Cannot read proto.", exception)
        }
    }

    override fun writeTo(t: UserPreference, output: OutputStream) {
        t.writeTo(output)
    }
}
```

# Create a Proto DataStore

- filename: where store the data.
- serializer : which serializer to use.

```
private val dataStore: DataStore<UserPreference> =
    context.createDataStore(
        fileName = "user_prefs.pb",
        serializer = UserPreferenceSerializer
    )
```

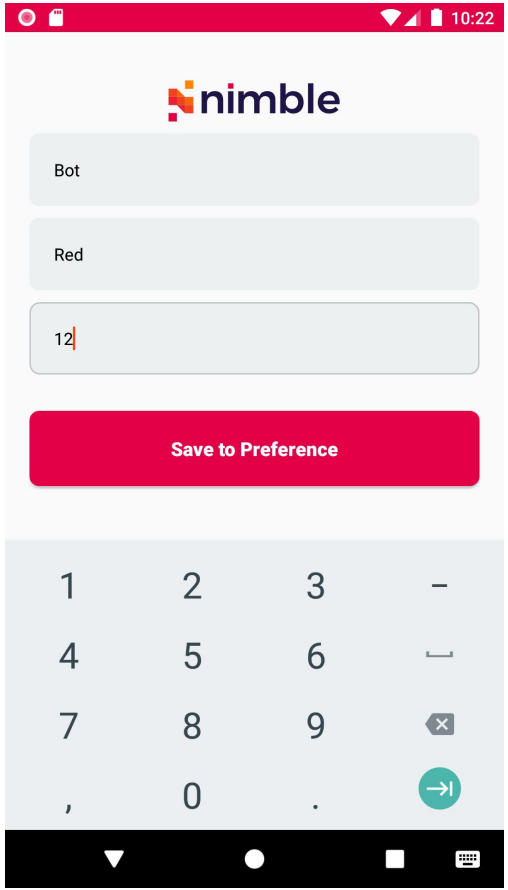
# Write data

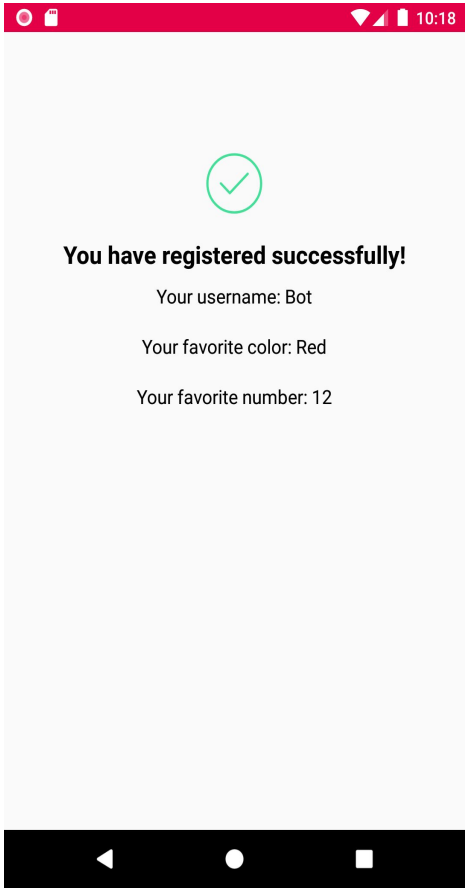
```
dataStore.updateData { userPreferences →
    userPreferences.toBuilder()
        .setUsername(username)
        .setFavoriteColor(favoriteColor)
        .setFavoriteNumber(favoriteNumber)
        .setIsLogin(isLogin)
        .build()
}
```

# Read data with Flow

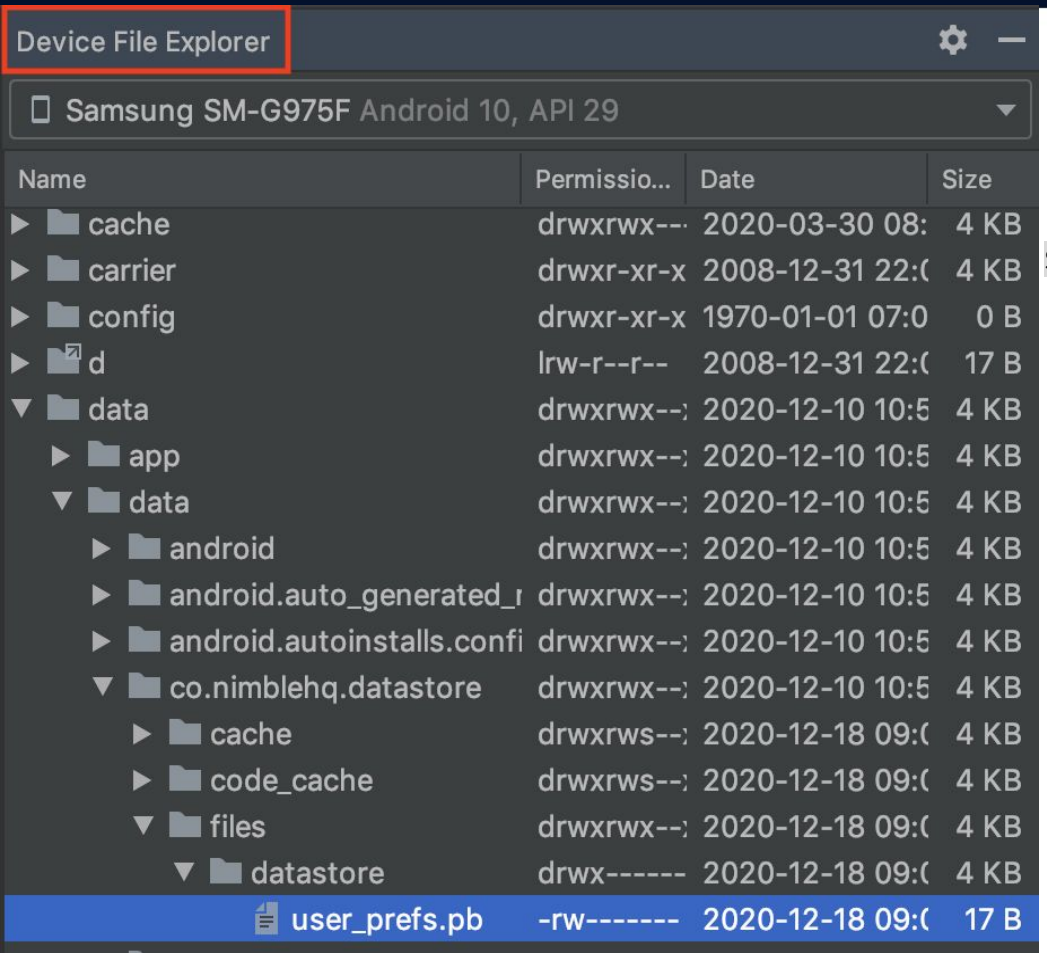
```
val userPreferenceFlow: Flow<UserEntity> = dataStore.data.catch {
        if (it is IOException) {
            Log.e(TAG, "Error reading sort order preferences.", it)
            emit(UserPreference.getDefaultInstance())
        } else {
            throw it
        }
    }.map { UserEntity(it.username, it.favoriteColor, it.favoriteNumber, it.isLogin) }
```

# Result


10:22
nimble
Bot
Red
12
Save to Preference
1 2 3 –
4 5 6
7 8 9
, 0 .


10:18
You have registered successfully!
Your username: Bot
Your favorite color: Red
Your favorite number: 12

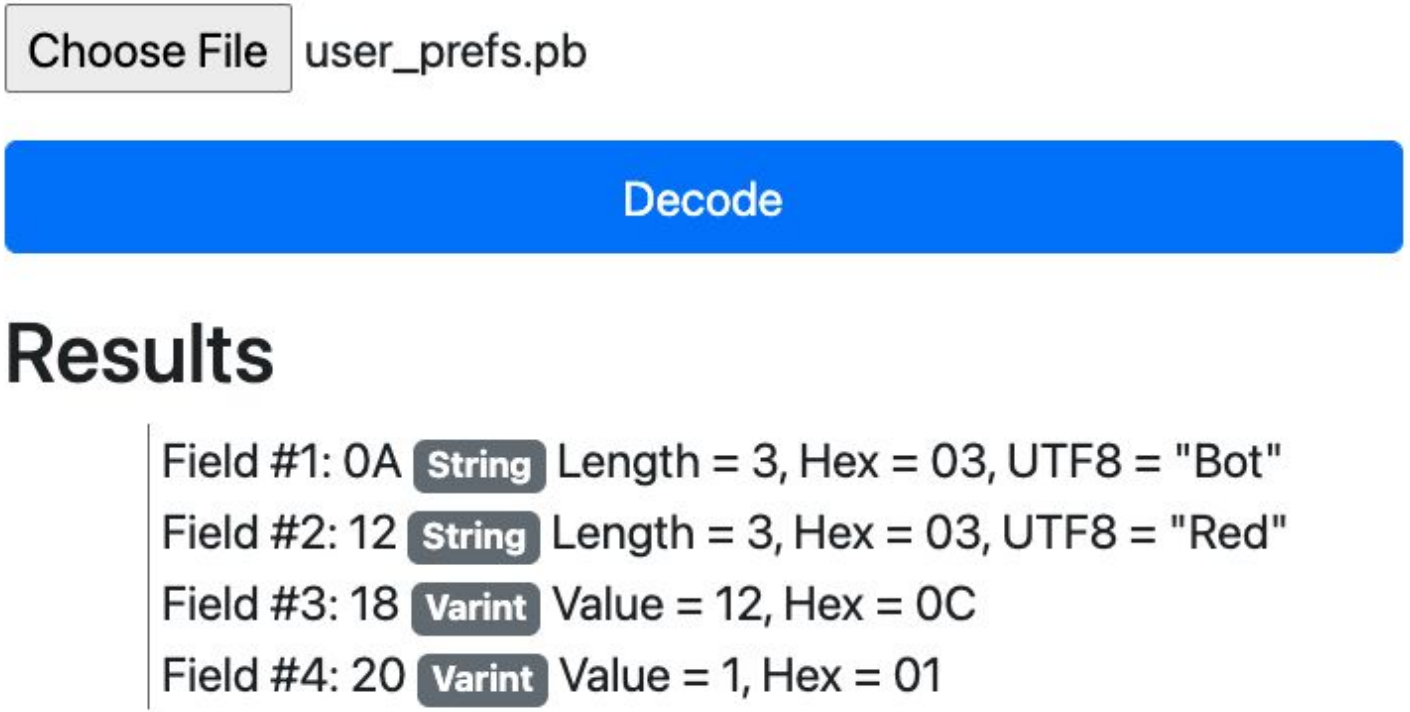
# Where are the files .pb stored? 🤔




## Use Device File Explorer (emulator)

$ data/data/{{package_name}}/files/datastore/user_prefs.pb

# What's inside the pb file



# Repository

https://github.com/nimblehq/proto-datastore-example-impl

README.md

## Proto Datastore example

### Usage

Clone the repository and build.

### Validate pb file

- Use File Explorer or adb shell with root access to the path:
  `data/data/co.nimblehq.datastore/files/datastore/user_prefs.pb`
- Now you can parse the protobuf (.pb) file to reveal the content, or you can check with this:
  https://protogen.marcgravell.com/decode

### License

- DataStore version: v1.0.0-alpha05 (Dec 2020)

# Q & A

👉 Link to download the presentation

Find out more at our booth! Or come talk to us!

Look for the Nimble logo up high 👋

# Thanks!


## Contact Nimble

nimblehq.co
hello@nimblehq.co

## Bangkok

399 Interchange 21 Sukhumvit Road, Unit #2402-03, Klong Toei, Wattana, Bangkok 10110, Thailand

## Singapore

160 Robinson Road, #14-04 Singapore Business Federation Centre. Singapore 068914

## Hong Kong

20th Floor, Central Tower
28 Queen's Road, Central, Hong Kong

## Ho Chi Minh

Lim Tower 3, 03-109, 29A Nguyen Dinh Chieu, Da Kao Ward, District 1, Ho Chi Minh City, Vietnam

## Da Nang

5th floor, 29 Yen Bai, Hai Chau 1, Hai Chau District, Da Nang 550000, Vietnam


nimble