Jetpack/Jetbrains Compose

The new UI kid on the block

15.03.2022, JavaLand, Germany

by Dr. Michael Paus

Background

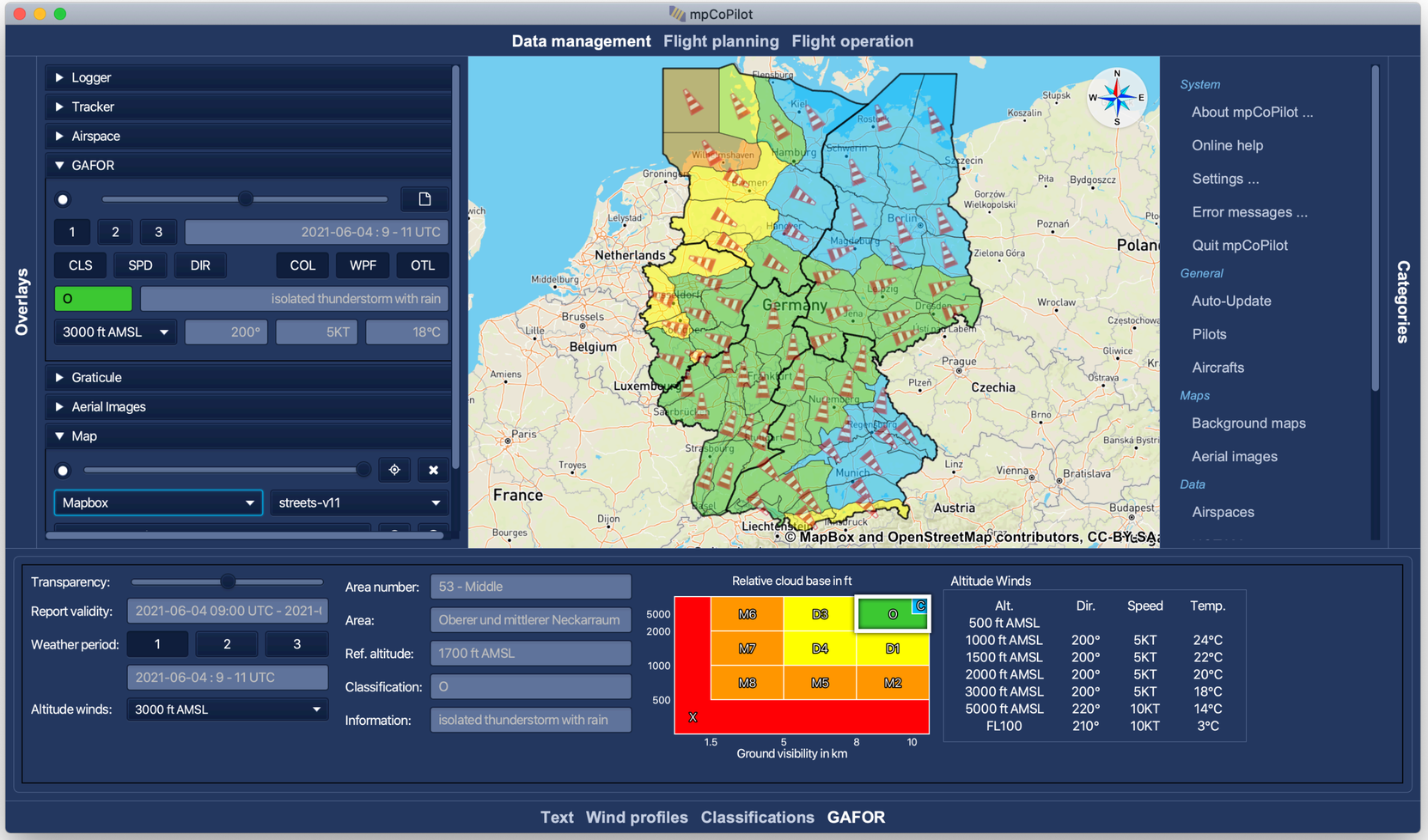
About me

- Dr. Michael Paus
- Aerospace engineer from Stuttgart
- Developer, Consultant, JavaFX-enthusiast
- OpenJFX author
- Chairman of the Java User Group Stuttgart e.V. (Organizers of the annual Java Forum Stuttgart)

The application

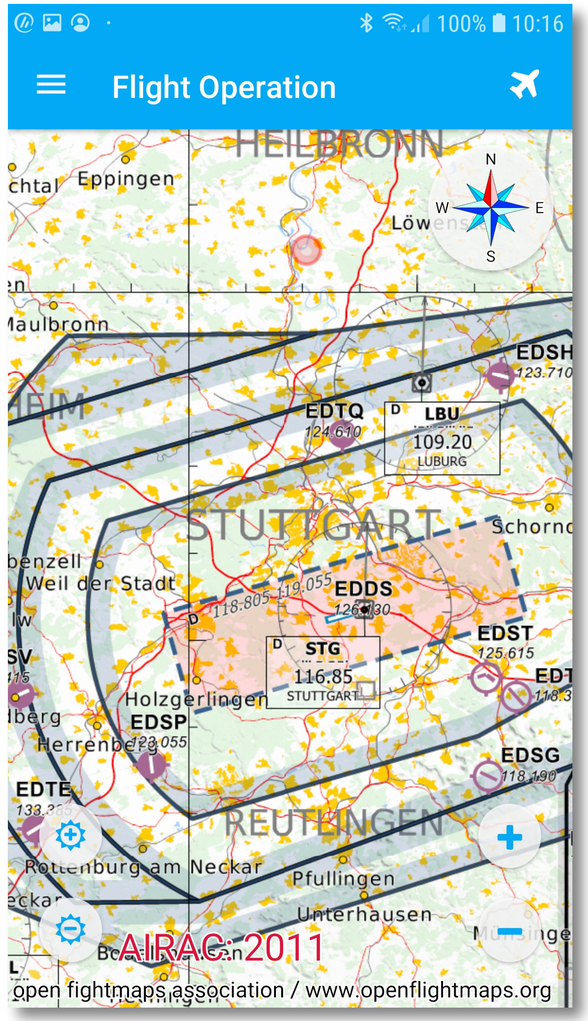
- Prototype of a fully JavaFX-based General-Aviation flight planning software.
- Testbed for new concepts in flight management, data management and flight planning.
- Special version used by „Deutscher Aeroclub“ DAeC to validate airspace data.
- Mobile version for flight operation.
- Web version for community interaction.

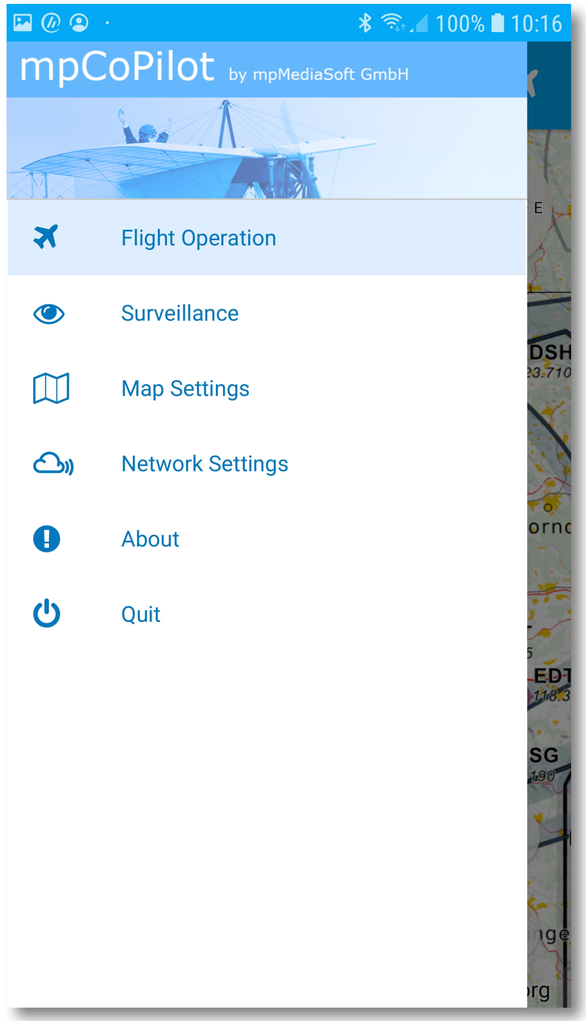
Desktop example view (JavaFX)

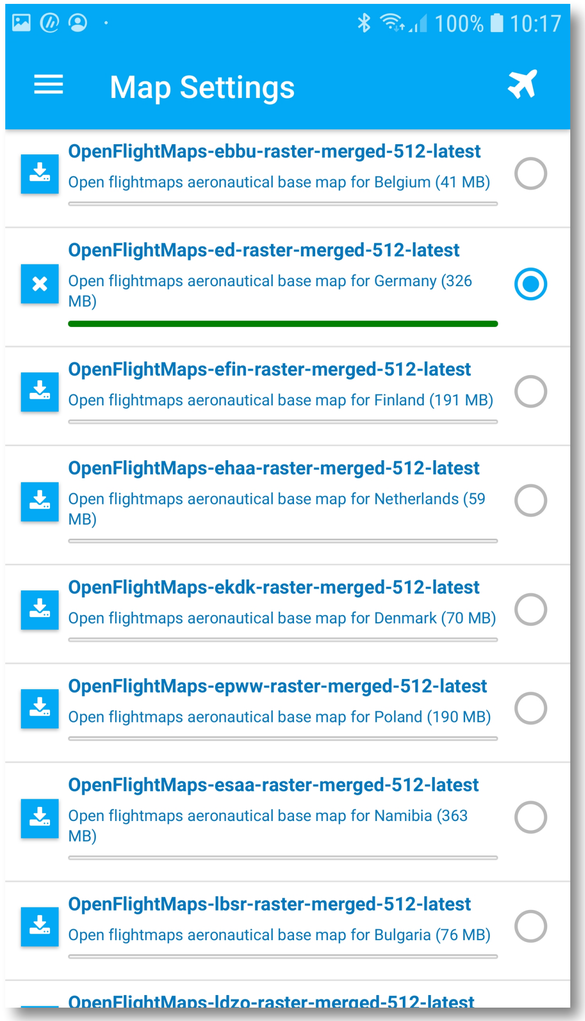


- Various modules
- GUI in JavaFX
- Multiplatform

Android example view (JavaFX with reuse of desktop modules)







Contents and take-aways

- Provide basic facts about Compose and JavaFX.
- Show fundamental concept of Compose via a small example.
- Discuss potential issues about going multi-platform.

- A basic understanding of how Compose and JavaFX work and how they differ.
- Some guidance on getting started.
- A few resources for further reading and experimentation.

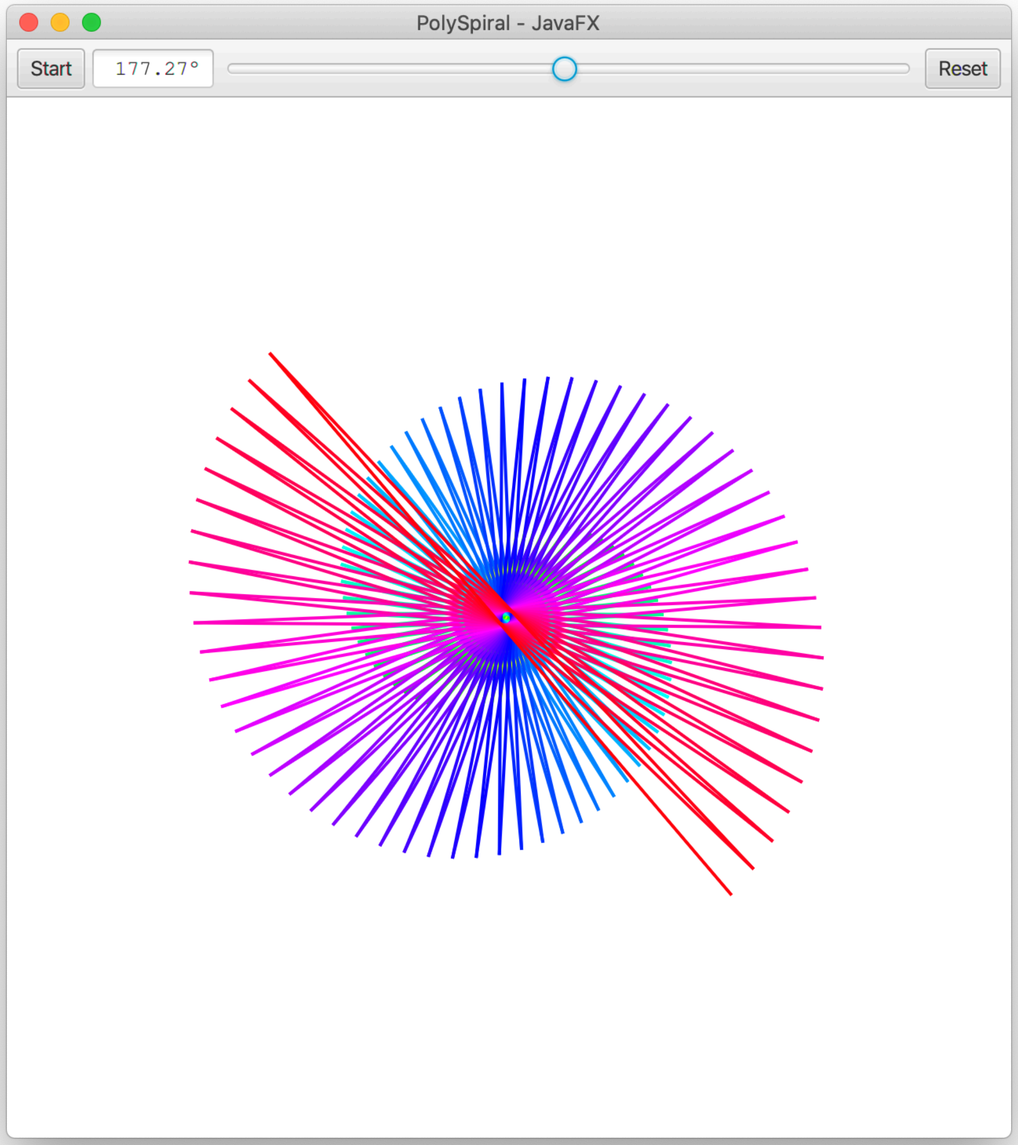
Basic facts 1/2

	JavaFX	**Compose**
Introduction	Version 1.0, December 2008 Version 2.0, Q3 2011	Version 1.0 (for Android), July 2021 Version 1.0 (Multiplatform), December 2021
Main design concept	Scene graph, Controls (objets), properties, bindings	Composables (functions), observable state, magic :-)
Languages	Any JVM language	Kotlin only (because of mandatory Kotlin compiler plugin)
Build tool	Any build tool (Maven, Gradle, ...)	Gradle only (because of Gradle plugin)
IDE support	Any IDE (Eclipse, IntelliJ, ...)	Practically only IntelliJ/Android-Studio
Maintainers	Oracle, Gluon, Community contributions	Google, Jetbrains, Community contributions
Big users	...	Google (Android ...), Jetbrains (Toolbox ...), Twitter, ...

Basic facts 2/2

	JavaFX	**Compose**
Cross-Platform techn.	JVM, GraalVM/Native-image	JVM, KMP bzw. KMM
Platforms	Windows, macOS, Linux (desktop) Android, iOS (Gluon mobile) embedded (Gluon mobile) Web (JPro on Server, Gluon on Client ???)	Windows, macOS, Linux (desktop) Android (native), iOS (only KMM + SwiftUI, direct Skiko Canvas rendering in the future) embedded (Raspberry Pi) Web (DOM based on Client, +Canvas rendering in 1.2 dev builds already)
License	GPL+Classpath for JavaFX GPL for Attach Proprietary for Charm-Glisten Various others for Gluon tooling	Apache License 2.0
Sources	GitHub + proprietary Gluon SW (charm-glisten)	GitHub

JavaFX PolySpiral Demo Desktop



Common PolySpiral Model Code

```
data class PolySpiralManagerState (
    val isRendering        : Boolean = false,
    val delayMillis        : Long    = 40,
    val length             : Double  = 5.0,
    val lengthIncrement    : Double  = 3.0,
    val strokeWidth        : Double  = 2.0,
    val angleIncrementDeg : Double  = 0.0
)
```

Common PolySpiral Model Code

```
class PolySpiralManager(val coroutineScope: CoroutineScope) {
    private var timerJob: Job? = null

    private val _polySpiralManagerState = MutableStateFlow(PolySpiralManagerState())
    val polySpiralManagerState = _polySpiralManagerState.asStateFlow()

    […]

    fun startRendering() {
        if (isNotRendering) {
            timerJob = coroutineScope.launch {
                _polySpiralManagerState.update { s -> s.copy(isRendering = isRendering) }
                while (true) {
                    with (polySpiralManagerState.value) {
                        _polySpiralManagerState.update {
                            s -> s.copy(angleIncrementDeg = (angleIncrementDeg + 0.05) % 360.0) }
                        delay(delayMillis)
                    }
                }
            }
        }
    }

    […]

}
```

Compose GUI Code 1/2

```
@Composable
fun PolySpiralApp() {
    val coroutineScope = rememberCoroutineScope()
    val polySpiralManager = remember { PolySpiralManager(coroutineScope) }
    val polySpiralManagerState by polySpiralManager.polySpiralManagerState.collectAsState()
    val uiScale = LocalDensity.current.density

    Surface {
        Column(modifier = Modifier.fillMaxSize()) {
            Row(
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.spacedBy(10.dp),
                modifier = Modifier.
                    background(Color(210, 230, 255)).padding(start = 10.dp, top = 5.dp, end = 10.dp, bottom = 5.dp).fillMaxWidth()
            ) {
                Button(
                    onClick = { if (polySpiralManagerState.isRendering) polySpiralManager.stopRendering()
                                else polySpiralManager.startRendering() },
                    modifier = Modifier.width(70.dp)
                ) {
                    Text(text = if (polySpiralManagerState.isRendering) "Stop" else "Start")
                }

                Box(
                    contentAlignment = Alignment.Center,
                    modifier = Modifier
                        .size(70.dp, 36.dp)
                        .clip(RoundedCornerShape(4.dp))
                        .background(Color.White)
                ) {
                    Text(
                        text = "%.2fº".format(polySpiralManagerState.angleIncrementDeg)
                    )
                }
```

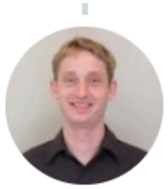
Compose GUI Code 2/2

```
            Slider(
                value = polySpiralManagerState.angleIncrementDeg.toFloat(),
                valueRange = 0f..360f,
                onValueChange = { polySpiralManager.angleIncrementDeg = it.toDouble() },
                modifier = Modifier.weight(1f)
            )

            Button(
                onClick = { polySpiralManager.reset() },
                modifier = Modifier
            ) {
                Text(text = "Reset")
            }
        }

        Canvas(modifier = Modifier.fillMaxSize().clipToBounds().background(Color.White)) {
            with (polySpiralManagerState) {
                drawSpiral(ComposeDrawScope(this@Canvas), length * uiScale, lengthIncrement * uiScale,
                    angleIncrementDeg, strokeWidth * uiScale)
            }
        }
    }
}
}
```

Compose GUI Code



Jim Sproch
@JimSproch

Replying to @a_key_bako

Declarative programming (including Compose) usually takes 3-6 months of usage before people have the " 💡⚡ holly-shit this is good" moment. Until then, you will struggle with it, you will fight it, it will frustrate you, and then it will all click in your mind six months later.

4:55 PM · May 13, 2021 · Twitter Web App

Following

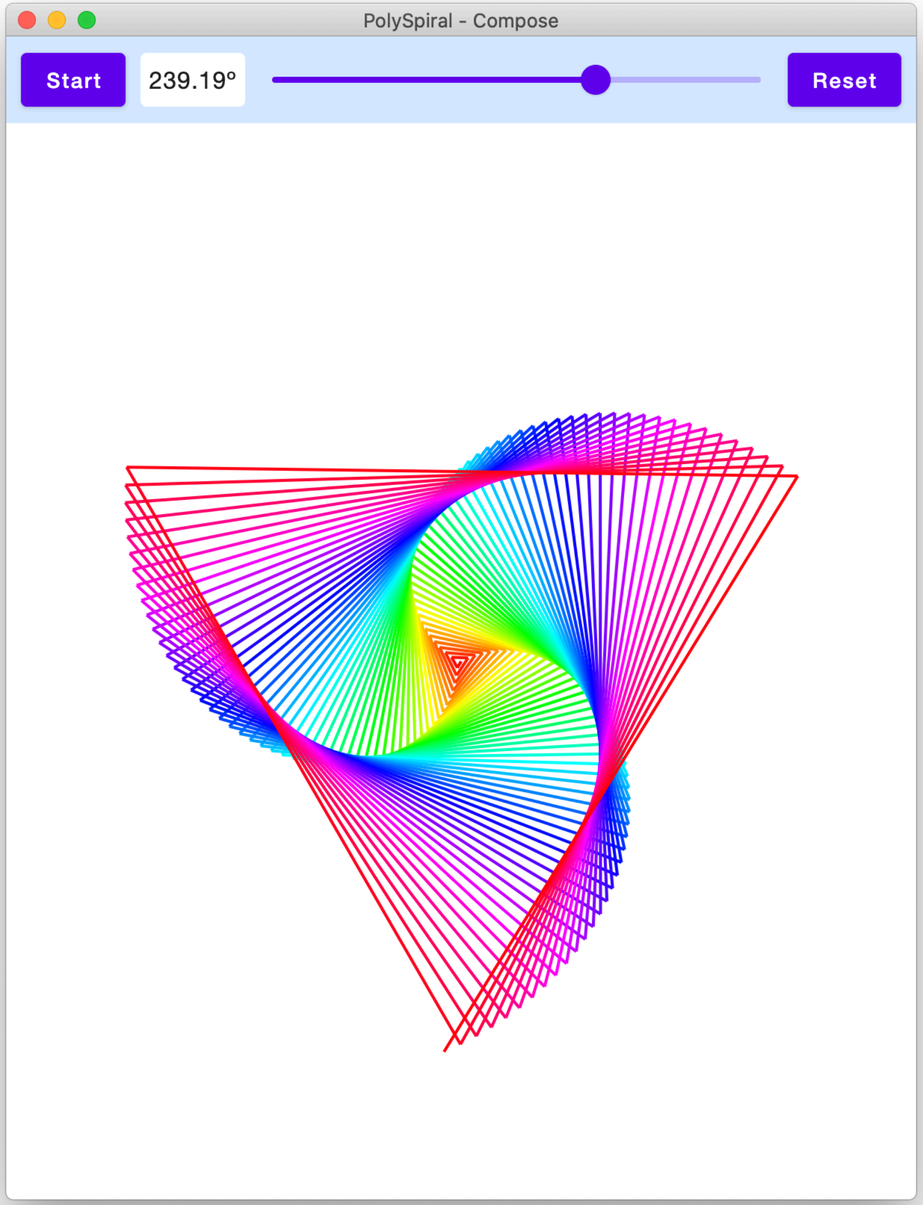
Jim Sproch
@JimSproch

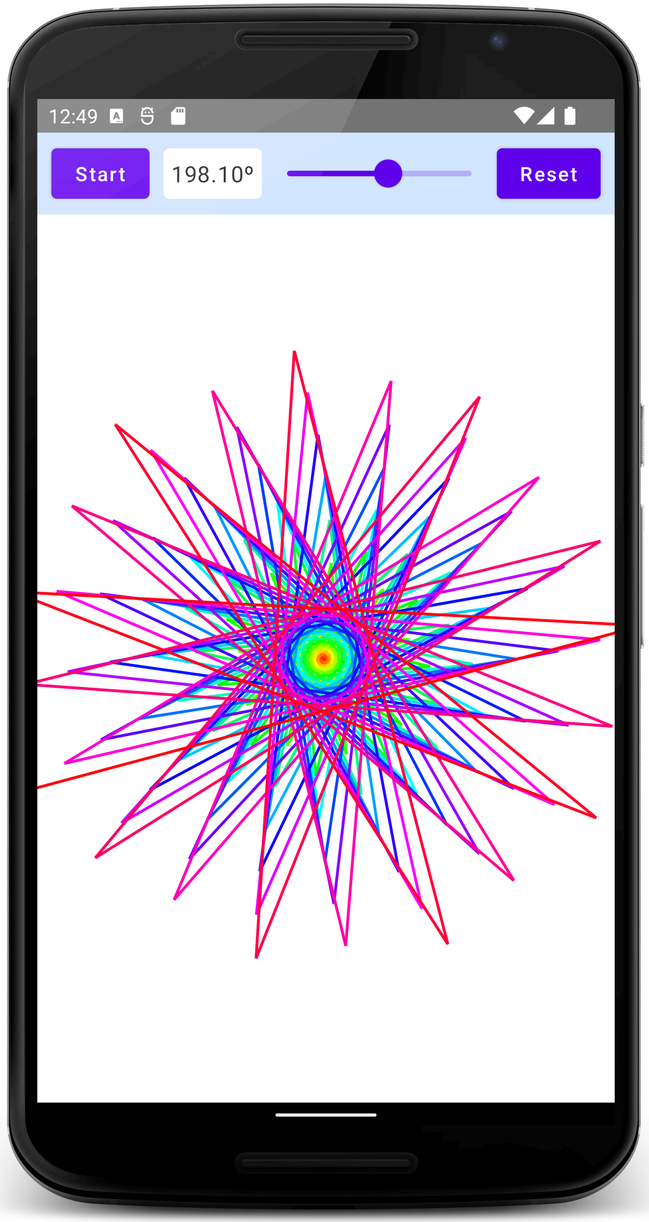
Senior software engineer at Google. Progenitor of Jetpack Compose (May 2017). Now working on giving Compose its next-generation super-power.

Mountain View, California Joined May 2019

310 Following **3,725** Followers

Compose Demo Desktop/Android

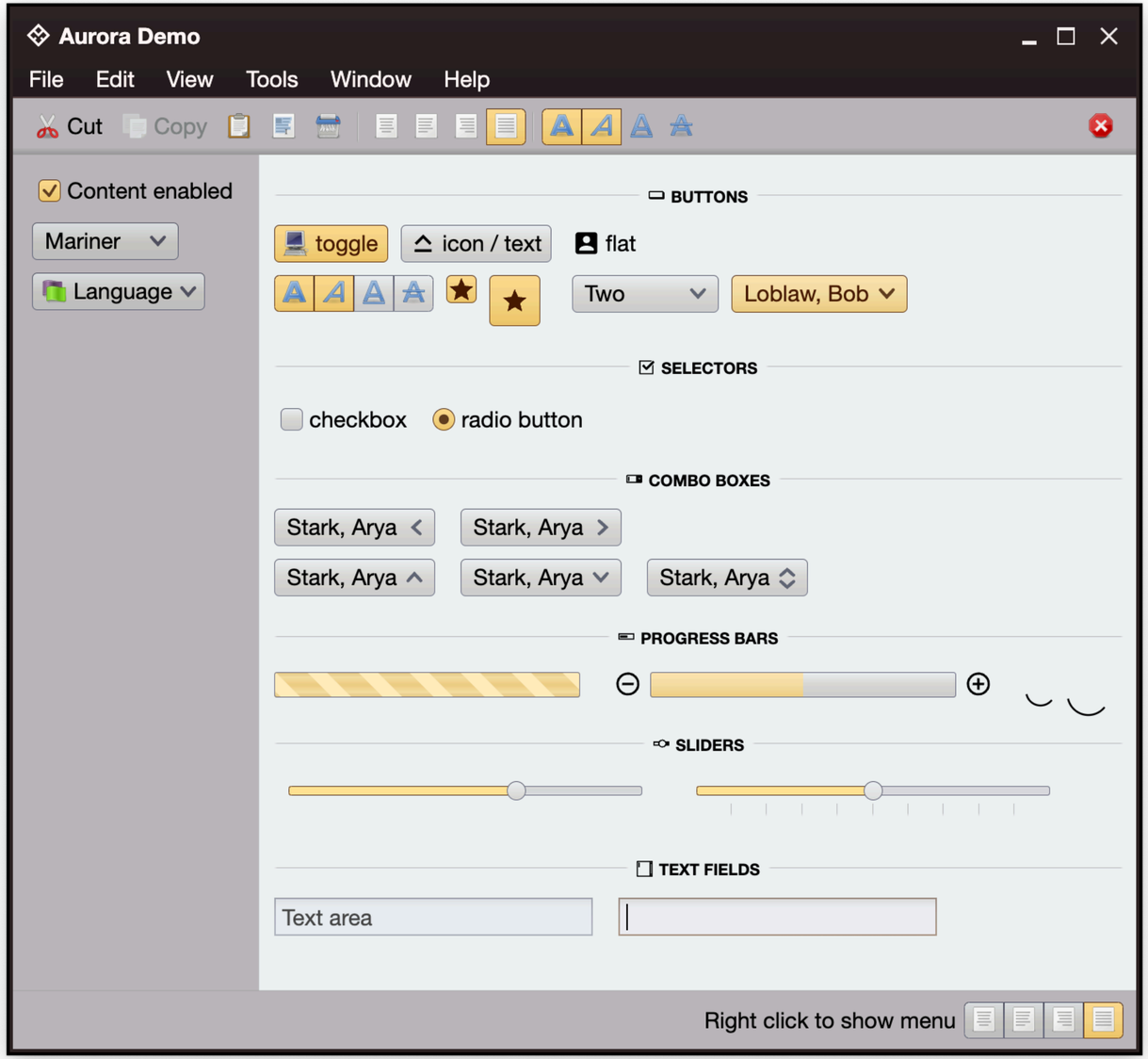




Compose Aurora Demo Desktop

Completely restyled

by Kirill Grouchnikov, https://github.com/kirill-grouchnikov/aurora



- Googles MaterialTheme is default theme
  - Adjustable (colors, typography, shapes)
- Custom themes
  - Completely restylable (see Aurora)
- Comparable to JavaFX CSS vs Skin

https://developer.android.com/jetpack/compose/themes

How to start

- Single-platform or multi-platform (KMP/KMM)?
  - Single-platform easier to handle than multi-platform.
  - How to integrate Java code?
    - Automatic/manual conversion to Kotlin.
    - Direct use on JVM/Android. (Android mostly compatible with Java 11!)
    - Substitution via expect/actual in multi-platform.
    - Maybe compilation via GraalVM/native-image and re-import via expect/actual

- Recommendation:
  - Start with single-platform or multi-platform limited to JVM/Android.

Documentation

- Most comprehensive documentation for Androids Jetpack Compose
  - https://developer.android.com/jetpack/compose/documentation
  - Difficult to distinguish Android specific parts from common Compose parts.
- Jetbrains Compose (multi-platform)
  - https://github.com/JetBrains/compose-jb
  - Important tutorials and examples for desktop specific concepts.
- Others
  - Stackoverflow (https://stackoverflow.com/questions/tagged/android-jetpack-compose+compose-desktop)
  - Slack (https://kotlinlang.slack.com, channels: compose, compose-desktop, compose-web, multiplatform)
  - Google (watch the time stamp!)

Staying platform independent 1/3

- Logging
  - Various alternatives, e.g., "io.github.microutils:kotlin-logging:2.1.21"
  - `private val log = KotlinLogging.logger {}`

- Navigation
  - Hot topic in Android world. Not really needed for desktop.
  - Avoid Compose Navigation because it is Android specific.
  - https://arkivanov.github.io/Decompose/ (interesting but complex)
  - https://github.com/adrielcafe/voyager (lightweight)

Staying platform independent 2/3

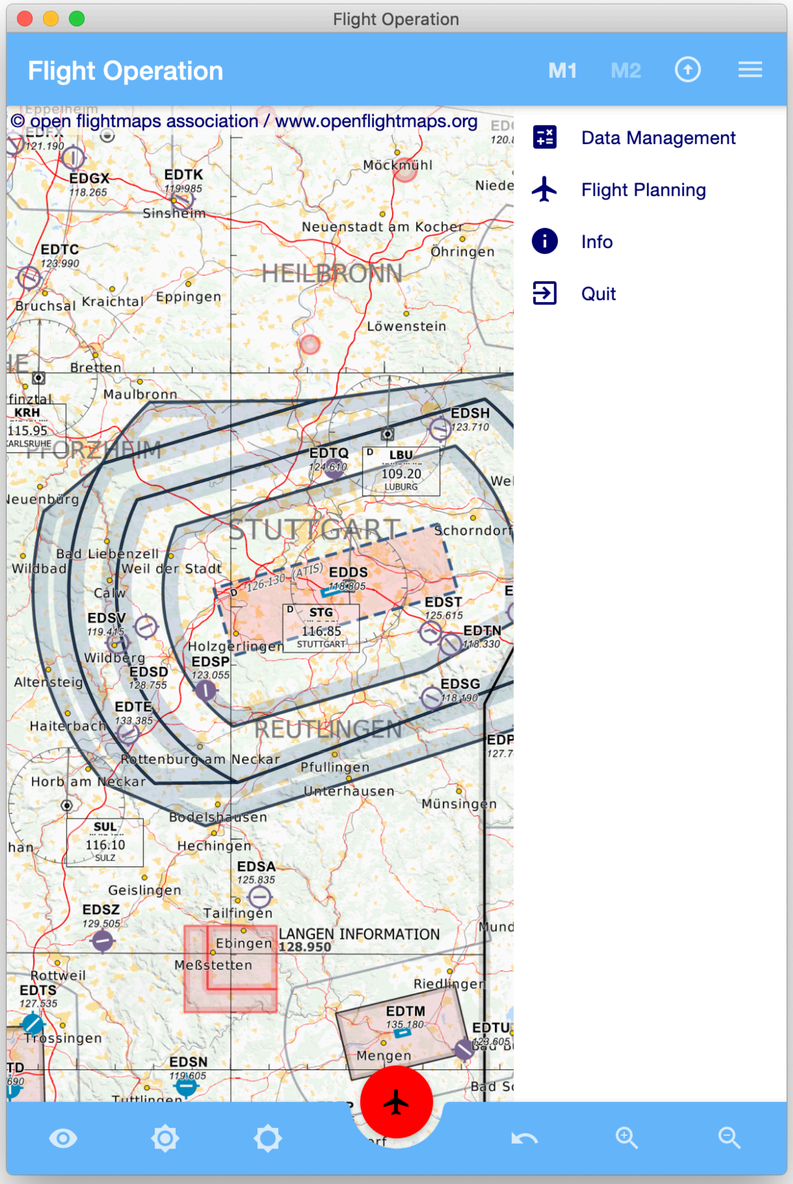
- Model - Runtime
  - Avoid ViewModel and LiveData because they are Android specific.
  - https://twitter.com/search?q=%20(from%3AJimSproch)%20ViewModel&src=typed_query&f=top
  - https://developer.android.com/guide/topics/resources/runtime-changes
  - Use coroutines and StateFlow as in example.
  - https://developer.android.com/kotlin/flow
  - https://medium.com/androiddevelopers/migrating-from-livedata-to-kotlins-flow-379292f419fb

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="de.mpmediasoft.mpcopilot.android">
    …
    <application
            …
            <activity
                 …
             android:configChanges="colorMode|density|fontScale|keyboard|keyboardHidden|layoutDirection|locale|mcc|mnc|navigation|orientation|screenLayout|
screenSize|smallestScreenSize|touchscreen|uiMode">
           …
       </activity>
    </application>
</manifest>
```

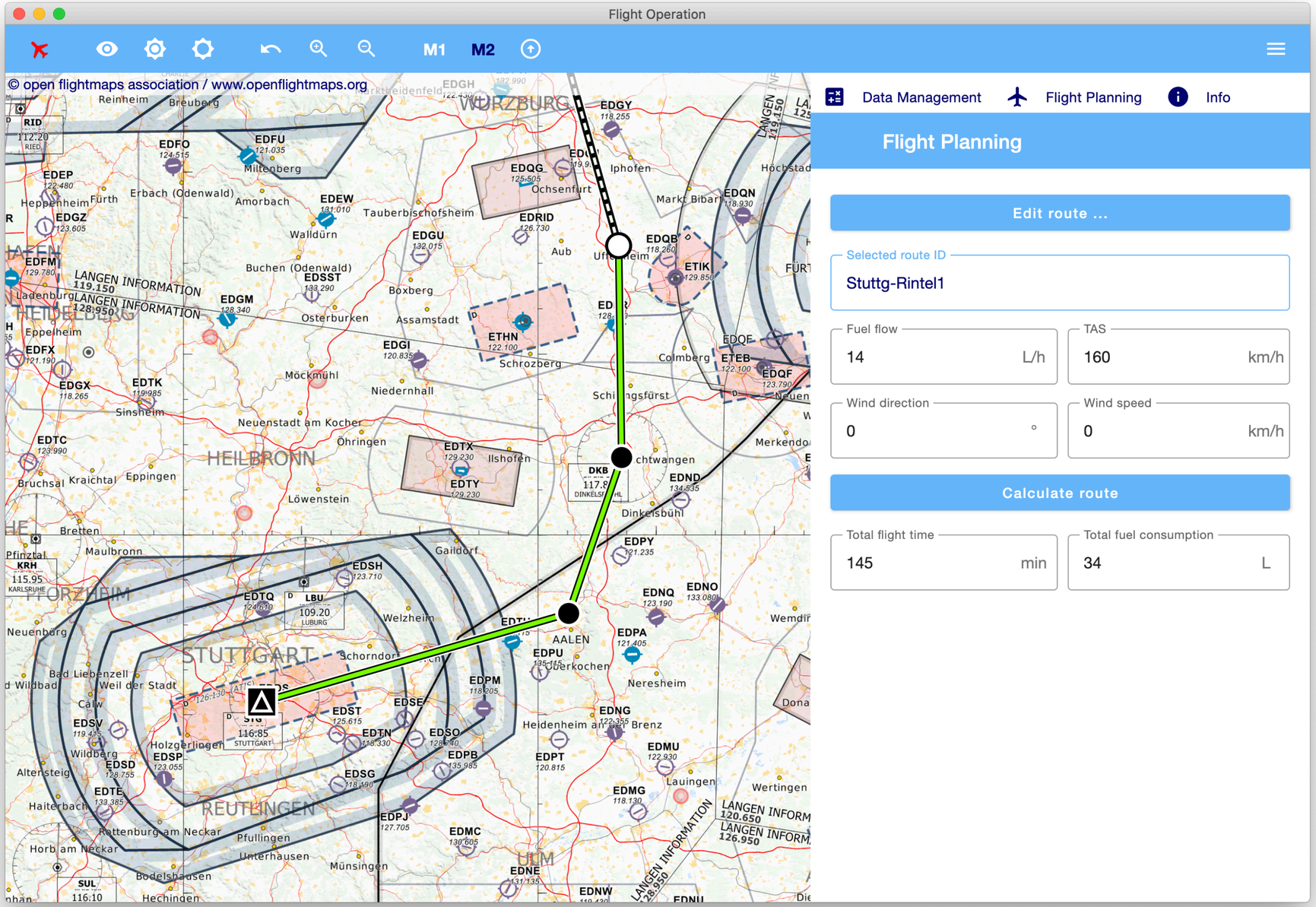
Staying platform independent 3/3

- Model - Persistence
  - Avoid Room because it is Android specific.
  - Use SQLite directly via JDBC or more elegantly via SQLDelight
  - https://github.com/xerial/sqlite-jdbc (Desktop), https://github.com/SQLDroid/SQLDroid (Android)
  - https://cashapp.github.io/sqldelight/ (Multiplatform)

Compose mpCoPilot Desktop/Android 1/2



Compose mpCoPilot Desktop/Android 2/2



Conclusions

- Compose is a powerful new GUI framework.
- Very mature for its age.
- Needs better Java integration in the multi-platform context.
- Tooling more consistent and complete as JavaFX but has to catch up with the fast development.
- It is fun to use and it is also very productive.
- Don’t get frustrated and give up too early :-)

Links

- https://blog.jetbrains.com/kotlin/2021/08/compose-multiplatform-goes-alpha/
- https://blog.jetbrains.com/kotlin/2021/12/compose-multiplatform-1-0-is-going-live/
- https://www.jetbrains.com/lp/compose-desktop/
- https://github.com/mipastgt/JavaLandTalk2022

Questions