Gemeinsame Client-Server-Entwicklung mit Ktor und Kotlin Multiplatform

Daniel Bälz & Robert Zetzsche

GDG DevFest Karlsruhe 2024

Daniel Bälz

Freiberuflicher Android Entwickler

https://dbaelz.de

Robert Zetzsche

Mobile Dev Lead

https://movisens.com

WERBUNG: Mobile Development Karlsruhe

https://www.meetup.com/de-DE/karlsruhe-mobile-development-meetup/

Ktor?
Kotlin Multiplatform?

Kotlin Multiplatform

“Reuse Kotlin code across Android, iOS, web, desktop, and server-side while keeping native code if needed.”

Kotlin Multiplatform

- Ziel: Code plattformübergreifend teilen
- Entwickelt von JetBrains
- Unterstützt Android, iOS, JVM, Web und Native (macOS, Windows, Linux)

Compose Multiplatform

- Deklaratives UI Framework
- Ziel: UI plattformübergreifend teilen
- Aufbauend auf Compose Compiler und Compose Runtime
  - Ebenfalls genutzt in Jetpack Compose UI (Android)
- Entwickelt von JetBrains (und Google)

Ktor

“Create asynchronous client and server applications. Anything from microservices to multiplatform HTTP client apps in a simple way.”

Ktor

- Framework zur Client- und Serveranwendungen
- Entwickelt von JetBrains
- Ktor 3.0.0 veröffentlicht im Oktober 2024
- Unterteilung in Ktor Server und Ktor Client

Ktor Prinzipien

- Einfach verwendbar
  - Start mit minimalem Setup möglich
  - Client und Server einfach konfiguriert
- Leichtgewichtiger Ansatz
  - Wenig „Magie“ wie Annotations oder CoC
  - Asynchronität mit Kotlin Bordmittel
- Flexibel und erweiterbar
  - Erweitert durch installierbare und eigene Plugins
  - Viele Protokolle und Features unterstützt

Ktor Server

- Plattformen: JVM und Native*
  - Native: Linux, macOS, iOS, watchOS, tvOS
  - Kein Windows
- Konfiguration im Code oder mittels Konfigurationsdatei
- Strukturierung über Module und Extension Functions
- Erweitert um Funktionalität über Plugins

Ktor Server Engines

- Verschiedene Engines für verschiedene Plattformen
  - Netty, Jetty Tomcat, ServletApplicationEngine, CIO*
- Engines werden über Abhängigkeiten hinzugefügt
- Konfigurierbar und bei Serverstart auswählbar

Ktor Server Plugins

- Manche in Core, die meisten als getrenntes Artefakt
- Beispiele
  - Security: Authentication (Basic, JWT, LDAP), Firebase, ...
  - Routing: Type-Safe routing, SSE, Static Content, ...
  - Serialization: Content Negotiation, kotlinx.serialization, GSON, ...
  - HTTP: Caching Headers, Compression, CORS, ...
  - Templating: HTML DSL, Mustache, Pebble, Thymeleaf, Velocity, ...
  - Datenbankanbindung: Exposed, MongoDB, ...
  - Sockets: WebSockets, Raw Sockets, Secure Sockets

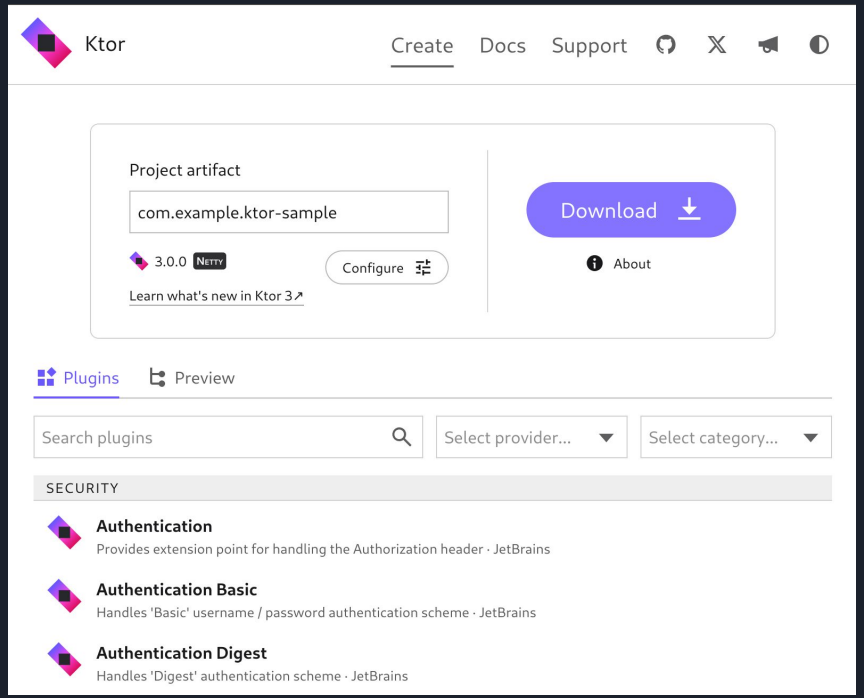
Ktor Client

- Plattformen: JVM, Android, JavaScript und Native
  - Native: macOS, iOS, watchOS, tvOS, Linux, Windows
- Unterschiedliche Engines für die Plattformen
  - Werden über eigene Abhängigkeiten hinzugefügt
  - Nicht alle Engines unterstützen HTTP/2 und WebSockets
- Client konfigurierbar im Code
- Erweiterbar durch Plugins

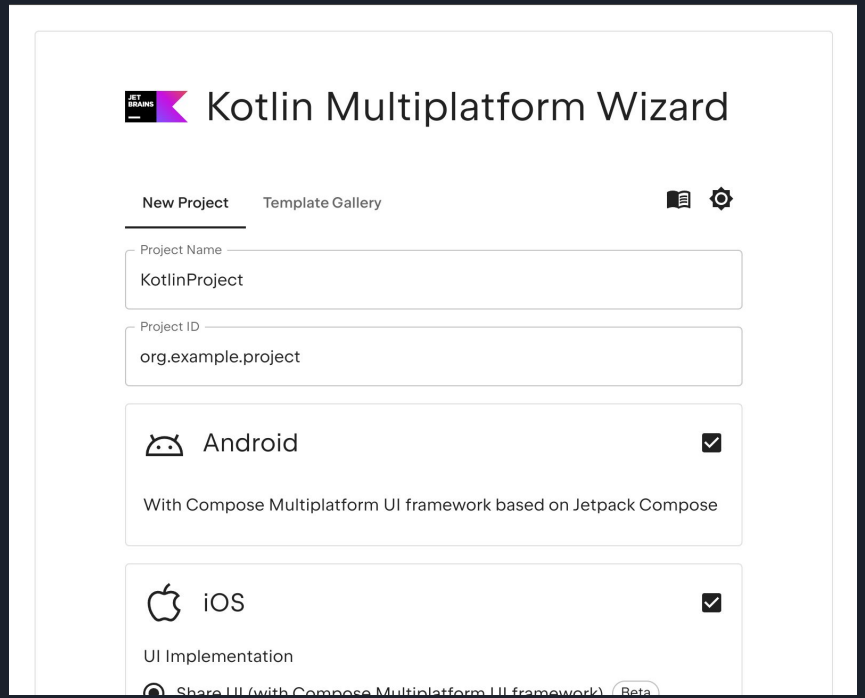
Ktor Client Plugins

- Analog zu Server über Plugins erweiterbar
- Manche in Core, die meisten als getrenntes Abhängigkeit
- Beispiele
  - Content Negotiation
  - Logging
  - Serialization
  - Authorization

Getting Started



https://start.ktor.io/

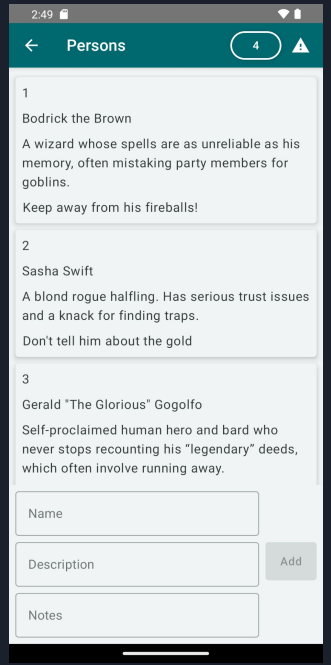


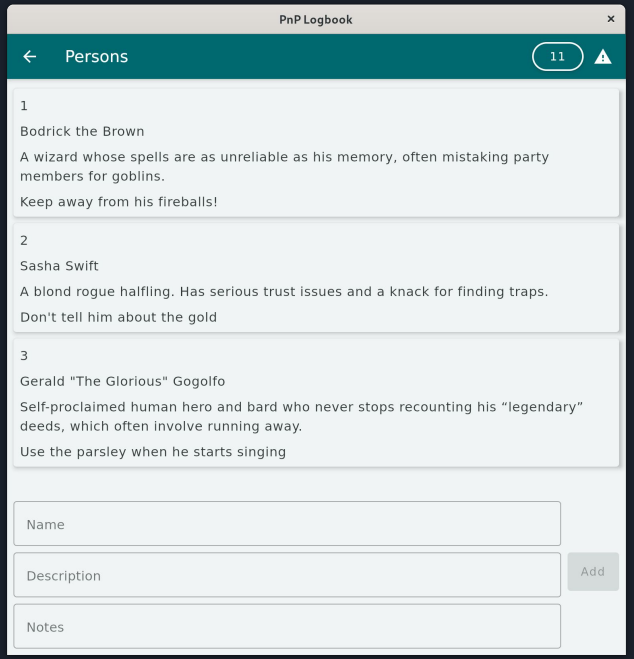
https://kmp.jetbrains.com/

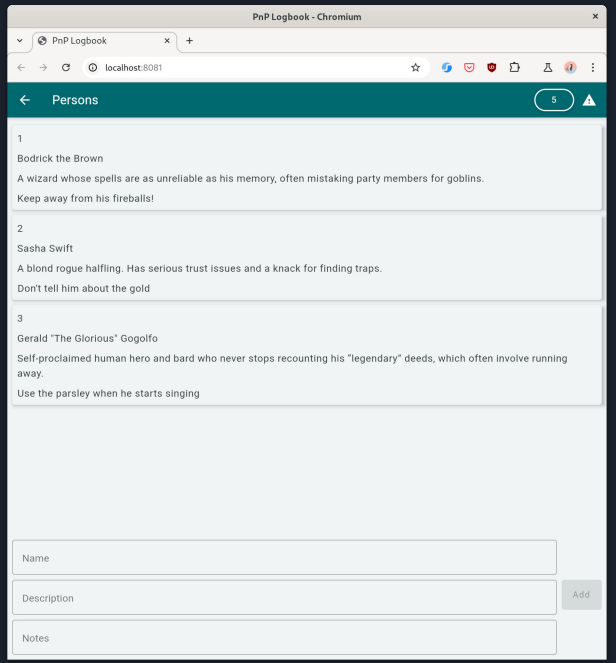
Start Coding

- JetBrains IDEs: Intellij IDEA oder Fleet
  - Plugin bzw. Android Studio für Android
  - Xcode für iOS
- Build System: Gradle oder Amper
- Nach Import loslegen

Demo Projekt







Demo Projekt

Code Sharing

- Models
- Repositories
- Networking
  - Client und Server Teil
- Business Logik
  - Validierung

Deployment

- Konfiguration im Code über *embeddedServer()*
- Konfigurationsdatei
  - HOCON oder YAML
  - Vordefinierte und eigene Properties nutzbar
  - Können über Env Variablen (mit defaults) gesetzt werden
  - Über Command Line überschreibbar
- Selfcontained App oder Servlet Container
  - Docker per Standard Deployment Plugin

Arbeiten im Team

- Mono Repo mit Shared, Client und Server
- Shared als externe Abhängigkeit
  - Jeweils ein Artefakte pro Zielplattform und Zielarchitektur
    - Android: AAR
    - JVM: Jar
    - iOS/Wasm/KN: klib
  - Metadata Artefakt mit common Code und Informationen über die spezifischen Artefakte
  - Das Metadata Artefakt wird im Projekt eingebunden und die Abhängigkeiten werden durch Gradle aufgelöst

Ktor vs. Competition

- Konkurrenz eher Java fokussiert, Kotlin als Addendum
- Spring Boot / Quarkus
  - Weniger leichtgewichtig und modular
  - Größere Community
- http4k / Javalin / Vert.X
  - Eher Libraries als Frameworks
  - Bessere Performance

Benchmarks

1	vertx-postgres	453,406	100.0% (77.5)
2	quarkus	214,275	47.3% (36.6)
3	http4k-apache4	210,940	46.5% (36.1)
4	http4k-netty	193,803	42.7% (33.1)
5	http4k-apache	187,355	41.3% (32.0)
6	quarkus resteasy reactive + hibernate	185,825	41.0% (31.8)
7	http4k-ratpack	183,693	40.5% (31.4)
8	micronaut jdbc	179,741	39.6% (30.7)
9	http4k-undertow	165,603	36.5% (28.3)
10	http4k-jetty	162,957	35.9% (27.9)
11	javalin-postgres	161,275	35.6% (27.6)
12	micronaut-graalvm	149,900	33.1% (25.6)
13	http4k-jettyloom-pgclient	130,094	28.7% (22.2)
14	micronaut	126,807	28.0% (21.7)
15	http4k-helidon-pgclient	116,906	25.8% (20.0)
16	micronaut data mongodb	110,495	24.4% (18.9)
17	http4k-jettyloom-jdbc	95,389	21.0% (16.3)
18	http4k-helidon-jdbc	88,791	19.6% (15.2)
19	micronaut jdbc graalvm	86,986	19.2% (14.9)

19	micronaut jdbc graalvm	86,986	19.2% (14.9)
20	ktor-netty-exposed-dsl	80,906	17.8% (13.8)
21	ktor-netty-exposed-dao	79,409	17.5% (13.6)
22	micronaut r2dbc	75,595	16.7% (12.9)
23	ktor-cio	67,555	14.9% (11.5)
24	ktor-pgclient	65,090	14.4% (11.1)
25	http4k-ktornetty	62,514	13.8% (10.7)
26	micronaut vertx pg client graalvm	62,132	13.7% (10.6)
27	micronaut data jdbc graalvm	58,335	12.9% (10.0)
28	ktor	56,630	12.5% (9.7)
29	micronaut data mongodb graalvm	46,847	10.3% (8.0)
30	micronaut r2dbc graalvm	38,340	8.5% (6.6)
31	ktor-reactivepg	31,616	7.0% (5.4)
32	ktor-jetty	26,618	5.9% (4.5)
33	ktor-jasync	26,393	5.8% (4.5)
34	spring	24,082	5.3% (4.1)
35	spring-mongo	22,736	5.0% (3.9)
36	spring-jpa	22,368	4.9% (3.8)
37	http4k-sunhttploom	6,906	1.5% (1.2)
38	http4k	6,674	1.5% (1.1)

Benchmarks

Rnk	Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext	Weighted score
7	vert.x	1,194,427	581,542	32,505	453,406	17,122	5,624,745	**6,847** 84.7%
38	quarkus	903,185	318,897	17,610	214,275	6,697	2,861,479	**3,637** 45.0%
40	micronaut	568,955	221,741	28,171	179,741	15,209	1,327,013	**3,555** 44.0%
48	http4k	466,128	254,369	26,322	210,940	8,038	544,208	**2,978** 36.8%
52	ktor	426,795	185,734	26,927	80,906	14,735	803,388	**2,803** 34.7%
55	javalin	512,495	211,243	16,582	161,275	10,405	897,788	**2,755** 34.1%
88	spring	236,259	147,907	15,932	24,082	7,131	506,087	**1,507** 18.6%

Ktor Vor- und Nachteile

Vorteile

- Einfacher Einstieg
- 1st Party Kotlin Support
  - KMP and K/N
  - Coroutines
- Pick’ n Choose Plugins

Nachteile

- Vendor Lock In
- Performance
- (Noch) geringes Ökosystem
- Kein Serverless Support

Ressourcen

- Learn Ktor
- Awesome Ktor
- Benchmarks Kotlin - Fortunes
- Benchmarks Kotlin - Composite
- Have your Serverless Kotlin Functions and Eat Them Too
- Ktor Roadmap 2024
- Kotlin Multiplatform Roadmap 2025

Ressourcen

- GDG DevFest Karlsruhe 2023
  - Kotlin Multiplatform
  - Compose Multiplatform
- Demo Projekt (PnP Logbook)

Fragen?!