



# Ktor

## REST APIs mit Kotlin entwickeln

Werner Eberling

E-Mail: [werner.eberling@mathema.de](mailto:werner.eberling@mathema.de)

Mastodon: [@Wer\\_Eb@social.dev-wiki.de](https://@Wer_Eb@social.dev-wiki.de)

X / Twitter: [@Wer\\_Eb](https://@Wer_Eb)



Der Sprecher



# Werner Eberling

Principal Consultant / Autor

Email: [werner.eberling@mathema.de](mailto:werner.eberling@mathema.de)

Mastodon: [@Wer\\_Eb@social.dev-wiki.de](https://@Wer_Eb@social.dev-wiki.de)

X (fka. Twitter): [@Wer\\_Eb](https://@Wer_Eb)





# Was haben wir vor?

- Was ist Ktor?
- Ktor Projekte erzeugen
- Routing und Serialisierung
- Unit Testing
- Endpunkte Absichern
- Ktor erweitern
- Ausblick: Laufzeitszenarien



Und  
vor allem ...

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-hide-small">
    <a class="w3-bar-item w3-button w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars"></i></a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#band">BAND</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#tour">TOUR</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#contact">CONTACT</a>
    <div class="w3-dropdown-hover w3-hide-large w3-hide-small">
      <button class="w3-padding-large w3-button w3-block"><i class="fa fa-caret-down"></i></button>
      <div class="w3-dropdown-content w3-bar-block w3-padding-large" style="background-color: #f1f1f1; min-width: 100px; width: 100px; height: 100px; position: absolute; left: -10px; top: 100%; z-index: 1; border-radius: 5px; border: 1px solid #ccc; padding: 5px; font-size: 1em; font-weight: normal; margin: 0; list-style-type: none; background-color: white; color: black; text-decoration: none; text-align: center; transition: all 0.3s ease 0s; transform: rotateX(0deg); transform-origin: left top; opacity: 0; visibility: hidden; ">
        <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#">> Home </a>
        <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#">> About </a>
        <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#">> Contact </a>
      </div>
    </div>
  </div>
</div>
```

Folien und Musterlösung:  
<https://github.com/wern/ktor-sample>

Scan me:



Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>



Was ist Ktor?



„Ktor is a framework to easily build connected applications – web applications, *HTTP services*, mobile and browser applications.“

<https://ktor.io>



```
fun main() {  
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {  
        routing {  
            route("/hello") {  
                get {  
                    call.respondText("Hallo Ktor!")  
                }  
            }  
        }  
    }.start(wait = true)  
}
```

- Kotlin DSL zur Definition von HTTP Endpunkten  
Ok, eigentlich „nur“ Funktionen und Funktionsaufrufe in „normaler“ Kotlin Syntax ;)



Live Code Beispiel



*Sit back and watch :-)*

Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>



## Aufsetzen eines Ktor Projektes

The screenshot shows a web browser window titled "Generate Ktor project". The URL is "start.ktor.io/?\_ga=2.112485444.13...". The page is titled "Ktor Project Generator". It has a navigation bar with "Settings" (highlighted in blue) and "Plugins" (highlighted in grey). Below the navigation, there is a text area: "You can create a Ktor project in two simple steps: enter your project name and click **Add plugins** to select the plugins you need. You can also fine-tune the settings of your project." A "Project Name" input field contains "ktor-sample". At the bottom, there are two buttons: "Adjust project settings ▾" and "Add plugins".

- **Ktor Project Generator**  
erzeugt Projektrümpfe mit  
den gewünschten Plugins  
Ähnlich Spring Boot Starter

- **Alternativ:**  
IntelliJ IDEA Ultimate + Ktor  
Plugin



Übung

1. Campus Project konfigurieren
2. Projekt herunterladen, auspacken, öffnen und starten
3. <http://localhost:8080> aufrufen

Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>



## Definition der unterstützten Routen

```
fun Route.greetingRouting(){
    route("/greetings") {
        get {
            call.respond(greetingStore.map { e -> e.value })
        }
    }
}
```

```
@Serializable
data class Greeting(val type:String, val greeting: String)

val greetingStore = mutableMapOf<String, Greeting>()
```

- Routendefinition über die Funktion `route`
- Nutzung von Lambda Ausdrücken für unterstützte Verben und deren hinterlegte Funktionen



```
fun Route.greetingRouting(){
    route("/greetings") {
        // [...]
        get("{type}") {
            val type = call.parameters["type"]
            call.respond(greetingStore[type] ?: HttpStatusCode.NotFound)
        }
    }
}
```

- Erweiterung des Pfades am HTTP Verb  
Angabe von Pfadvariablen möglich
- Zugriff auf Pfadvariablem über `call.parameters`



## Content (De-)Serialisierung

ktor-server-core

```
fun Application.configureSerialization() {  
    install(ContentNegotiation) {  
        json()  
    }  
}
```

ktor-server-content-negotiation

ktor-serialization-json

- Content Negotiation und Serialisierung werden über entsprechende Plugins realisiert

Funktion muss im Rahmen der Anwendungskonfiguration aufgerufen werden



## Zugriff auf den Request Body

```
fun Route.greetingRouting(){
    route("/greetings") {
        // [...]
        post {
            val greeting = call.receive<Greeting }()
            greetingStore[greeting.type] = greeting
            call.respond(HttpStatusCode.Created)
        }
    }
}
```

- Auslesen des Inhalts des Bodies über `call.receive`  
Angabe des Zieltyps via Typ-Parameter
- Deserialisierung erfolgt automatisch bei entsprechend konfiguriertem Plugin



## Start des embedded Servers

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        configureRouting()
        configureSerialization()
    }.start(wait = true)
}
```

```
fun Application.configureRouting() {
    routing {
        greetingRouting()
    }
}
fun Application.configureSerialization() {
    install(ContentNegotiation) {
        json()
    }
}
```

- Programmatischer Start und Konfiguration des Servers  
Verschiedene Laufzeitumgebungen möglich
- Konfiguration der benötigten Plugins via Funktionsaufruf  
Implementierung als Extension Functions



Übung



1. Greeting & Store implementieren
2. Endpunkte zur Anlage & Abfrage von Greetings implementieren
3. <http://localhost:8080> aufrufen & manuell testen

Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>



```
class ApplicationTest {  
    @Test  
    fun testStartWithEmptyGreetingStorage() = testApplication {  
        application {  
            configureRouting()  
            configureSerialization()  
        }  
        client.get("/greetings").apply {  
            assertEquals(HttpStatusCode.OK, status)  
            assertEquals("[]", bodyAsText())  
        }  
    }  
}
```

- Eigene Test Applikation  
ohne Serverinstanz  
ohne Netzwerkanbindung  
mit eigener Konfiguration
- Ktor HTTP Client zum  
Absetzen von Anfragen  
Verarbeiten der Antwort



Übung



1. Tests für gerade implementierte Endpunkte implementieren
2. Tests starten
3. Roten Test provozieren & fixen.

Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>



## Konfiguration via application.conf

```
ktor {  
    deployment {  
        port = 8080  
    }  
    application {  
        modules = [ de.mathema.ApplicationKt.module ]  
    }  
}
```

```
fun main (args: Array<String>) =  
    io.ktor.server.netty.EngineMain.main(args)  
  
fun Application.module() {  
    configureRouting()  
    configureSerialization()  
}
```

- Anwendung zieht sich ihre Konfiguration automatisch  
Mögliche Auswirkungen auf Tests!
- Konfiguration wird über Module geladen  
Wirkung über Modulgrenzen hinweg!

## Übung

1. Neue Applikation & passende Konfiguration anlegen
2. Modulfunktion implementieren & Anwendung starten (Tests?)
3. <http://localhost:8080> aufrufen & manuell testen

Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>



## Zugriff Absichern – Plugins

```
<dependency>
    <groupId>io.ktor</groupId>
    <artifactId>ktor-server-auth-jvm</artifactId>
    <version>${ktor_version}</version>
</dependency>
```

```
fun Application.configureAuthentication() {
    install(Authentication) {
        // ...
    }
}
```

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        configureAuthentication()
        // ...
    }.start(wait = true)
}
```

- Ktor bietet Plugins zur Authentifizierung
  - Basic
  - Bearer Token
  - JWT
  - ...
- Benötigt zusätzliche Bibliothek(en) und Plugin Konfiguration



```
fun Route.greetingRouting(){
    route("/greetings") {
        authenticate("auth-bearer") {
            get {
                // ...
            }
            post {
                // ...
            }
            // ...
        }
    }
}
```

- Zuordnung der Endpunkte zur Absicherung erfolgt im Rahmen der Routen-definition



## Basic Authentication

```
fun Application.configureBasicAuthentication() {  
    install(Authentication) {  
        basic("auth-basic") {  
            realm = "Ktor Sample realm"  
            validate { credentials ->  
                if (credentials.name == "Ktor" &&  
                    credentials.password == "KtorWorkshop") {  
                    UserIdPrincipal(credentials.name)  
                } else {  
                    null  
                }  
            }  
        }  
    }  
}
```

- Jeder Authentifizierungsmechanismus bekommt einen eindeutigen Namen  
Zur Referenzierung in Routendefinition
- Weitere Konfiguration vom Typ abhängig  
Hier: Realm & Definition einer Validierungsfunktion



## Bearer Authentication

```
fun Application.configureBasicAuthentication() {  
    install(Authentication) {  
        bearer("auth-bearer") {  
            realm = "Ktor Sample realm"  
            authenticate { credential ->  
                if (credential.token == "KtorWorkshop")  
                    UserIdPrincipal("Ktor")  
                } else {  
                    null  
                }  
            }  
        }  
    }  
}
```

- Authentifizierung erfolgt auf Basis des übergebenen Tokens  
Shared secret  
Username ergibt sich „im echten Leben“ implizit aus dem Tokenwert

## Übung

1. Neue Abhängigkeit definieren und herunterladen
2. Authentifizierungsmethode Bearer Token implementieren
3. <http://localhost:8080> aufrufen & manuell testen

Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>



## Authentication in Unit Tests

```
class ApplicationTest {  
    @Test  
    fun testStartWithEmptyGreetingStorage() = testApplication {  
        application {  
            configureBearerAuthentication()  
            configureRouting()  
            configureSerialization()  
        }  
        client.get("/greetings") {  
            headers {  
                header("Authorization", "Bearer KtorWorkshop")  
            }  
            }.apply {  
                assertEquals(HttpStatusCode.OK, status)  
                assertEquals("[]", bodyAsText())  
            }  
    }  
}
```

- Durch die Definition der entsprechenden Header Werte kann die Authentifizierung im Test genutzt / getestet werden



Übung

1. Unit Tests um Authentication  
erweitern

2. Test erfolgreich durchführen

Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>



```
<dependency>
    <groupId>io.ktor</groupId>
    <artifactId>ktor-server-auth-jwt-jvm</artifactId>
    <version>${ktor_version}</version>
</dependency>
```

```
fun Application.configureAuthentication() {
    install(Authentication) {
        // ...
    }
}
```

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        configureAuthentication()
        // ...
    }.start(wait = true)
}
```

- JWT Authentifizierung benötigt zusätzliche Bibliothek und ggf. weitere Logik zur Token Erstellung



## JWT Authentication

```
fun Application.configureBasicAuthentication() {  
    install(Authentication) {  
        bearer("auth-jwt") {  
            realm = "Ktor Sample realm"  
            verifier { ... }  
            validate { ... }  
            challenge { ... }  
        }  
    }  
}
```

- Authentifizierung erfolgt auf Basis des übergebenen Tokens  
*Verifier* prüft ob der Token gültig ist  
*Validator* prüft ob der Inhaber des Tokens die benötigten Rechte hat  
*Challenge* definiert die Antwort im Fehlerfall (optional)



## JWT Authentication – Token Verifizierung

```
fun Application.configureBasicAuthentication() {  
    install(Authentication) {  
        bearer("auth-jwt") {  
            realm = "Ktor Sample realm"  
            verifier {  
                JWT  
                    .require(Algorithm.HMAC256("KtorWorkshop"))  
                    .withAudience("http://localhost:8080/greetings")  
                    .withIssuer("http://localhost:8080")  
                    .build()  
            }  
            validate { ... }  
            challenge { ... }  
        }  
    }  
}
```

- Ist der Token grundsätzlich gültig?
- Verschiedene Algorithmen möglich
  - HMAC256 – Shared Secret
  - RSAC256 – Public/Private RSA KeyPair
  - ...



## JWT Authentication – Token Validierung

```
fun Application.configureBasicAuthentication() {  
    install(Authentication) {  
        bearer("auth-jwt") {  
            realm = "Ktor Sample realm"  
            verifier { ... }  
            validate { credential ->  
                if (credential.payload.getClaim("username")  
                    .asString() == "Ktor") {  
                    JWTPrincipal(credential.payload)  
                } else {  
                    null  
                }  
            }  
            challenge { ... }  
        }  
    }  
}
```

- Hat der Tokeninhaber die notwendigen Rechte?
- Genaue Logik ist anwendungsabhängig!



## JWT Authentication – Antwort im Fehlerfall

```
fun Application.configureBasicAuthentication() {  
    install(Authentication) {  
        bearer("auth-jwt") {  
            realm = "Ktor Sample realm"  
            verifier { ... }  
            validate { ... }  
            challenge { defaultScheme, realm ->  
                call.respond(HttpStatusCode.Unauthorized,  
                    "Token is not valid or has expired")  
            }  
        }  
    }  
}
```

- Optionale Antwort falls Verifizierung oder Validierung fehlgeschlagen.



## JWT Authentication – Tokenerstellung

```
fun Route.authenticationRouting(){
    route("/auth") {
        post("login") {
            val user = call.receive<User }()
            call.respond(
                if(user.password == authSecret) {
                    val token = JWT.create()
                        .withAudience("http://localhost:8080/greetings")
                        .withIssuer("http://localhost:8080")
                        .withClaim("username", user.username)
                        .withExpiresAt(
                            Date(System.currentTimeMillis() + 60000))
                        .sign(Algorithm.HMAC256("KtorWorkshop"))
                    hashMapOf("token" to token)
                } else {
                    HttpStatusCode.Unauthorized
                }
            )
        }
    }
}
```

```
@Serializable
data class User(val username:String, val password: String)
```

- Woher kommt der Token?
  - Zentraler IDP
  - Endpunkt innerhalb der Anwendung
- Konfiguration muss zwischen Erstellung und Prüfung des Tokens geteilt werden!

## Übung



1. Neue Abhängigkeit definieren und herunterladen
2. JWT Auth als Plugin konfigurieren & Endpunkt zur Token Erzeugung implementieren.
3. <http://localhost:8080> aufrufen & manuell testen

Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>



## Modulare Funktionalität durch Plugins

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        configureAuthentication()
        configureRouting()
        configureSerialization()
    }.start(wait = true)
}
```

```
fun Application.configureRouting() {
    routing {
        greetingRouting()
    }
}

fun Application.configureSerialization() {
    install(ContentNegotiation) {
        json()
    }
}

fun Application.configureAuthentication() {
    install(Authentication) {
        // ...
    }
}
```

- Ktor organisiert die meisten Funktionalitäten in Form von Plugins
  - Content Negotiation
  - Routing
  - Authentication
  - ...



## Ktor mit Custom Plugins erweitern

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        configureAuthentication()
        configureRouting()
        configureSerialization()
        configureCustomPlugin()
    }.start(wait = true)
}
```

```
val TypeAlwaysLowerCase = createApplicationPlugin(
    name = "TypeAlwaysLowerCase") {
    println("TypeAlwaysLowerCasePlugin is installed!")
}
```

```
fun Application.configureAuthentication() {
    install(TypeAlwaysLowerCase)
}
```

- Eigene Plugins können über die Funktion *createApplicationPlugin()* erzeugt werden



## Eingriff in verschiedenen Aufruf Phasen

```
val TypeAlwaysLowerCase = createApplicationPlugin(  
    name = "TypeAlwaysLowerCase") {  
    println("TypeAlwaysLowerCasePlugin is installed!")  
  
    onCall { call -> println("onCall() received") }  
  
    onCallReceive { ...}  
  
    onCallRespond() { call -> println("onCallRespond() received") }  
}
```

- Ein Plugin kann auf verschiedene Ereignisse reagieren
  - Request Verarbeitung
  - (De)Serialisierung
  - Anwendungsübergänge



## Ergebnis der Deserialisierung ändern

```
val TypeAlwaysLowerCase = createApplicationPlugin(  
    name = "TypeAlwaysLowerCase") {  
    //...  
    onCallReceive { call ->  
        transformBody { data ->  
            if (call.request.origin.uri == "/greetings") {  
                val body = data.readUTF8Line()?:""  
                try {  
                    val type = Json { ignoreUnknownKeys = true }  
                        .parseToJsonElement(body)  
                        .jsonObject.get("type")  
                        ?.jsonPrimitive?.content ?: ""  
                    ByteReadChannel(body.replace(type,  
                        type.lowercase()))  
                } catch (ex : Exception){ ByteReadChannel(body) }  
                } else { data }  
            }.also { println("onCallReceive() received") }  
    }  
}
```

- Der Typ des übergebenen Greetings soll automatisch immer klein gespeichert werden

Reihenfolge beachten!  
Beispiel ändert nur den Request Body bei Greeting Erzeugung!



Übung

1. Neues Plugin erzeuge & installieren
2. Greeting.type in lowercase() ändern
3. <http://localhost:8080> aufrufen & manuell testen

Post It PNGs by Vecteezy  
<https://www.vecteezy.com/free-png/post-it>

A scenic landscape photograph featuring a red wooden bench in the lower-left foreground, positioned on a grassy hillside. The background shows a lush green valley with rolling hills and mountains under a cloudy sky. A bright lens flare is visible in the upper right corner.

Ausblick



- Neben der Nutzung eines Embedded Servers (z.B. Netty, Jetty, oder Tomcat) auch andere Betriebsarten möglich
  - WAR Deployment
  - Docker Container
  - Cloud Deployment
- Als besonderes embedded „Schmankerl“: CIO (Coroutine-based I/O)



- Keine Annotationswüste  
(Fast) alles ist eine Funktion
- Leichtgewichtige Testmöglichkeiten
- Leicht erweiterbar
- Strukturierte Nebenläufigkeit durch Koroutinen  
Bodys der Verb-Funktionen sind Suspending Functions
- Leichtgewichtige Alternative zu Spring MVC
  - Falls keine weiteren Spring Features benötigt werden!
  - Keine magische „Dependency triggered“ Funktionalität ;)
  - Bei Bedarf auch WAR Deployment möglich

*Einfach mal ausprobieren ;)*



MATHEMA  
CAMPUS

# Vielen Dank! Fragen?

Werner Eberling

E-Mail: [werner.eberling@mathema.de](mailto:werner.eberling@mathema.de)

Mastodon: [@Wer\\_Eb@social.dev-wiki.de](https://@Wer_Eb@social.dev-wiki.de)

X / Twitter: [@Wer\\_Eb](https://@Wer_Eb)

Beispiele: <https://github.com/wern/ktor-sample>

[www.mathema.de](http://www.mathema.de)



Beispiele?  
Scan me ;)

