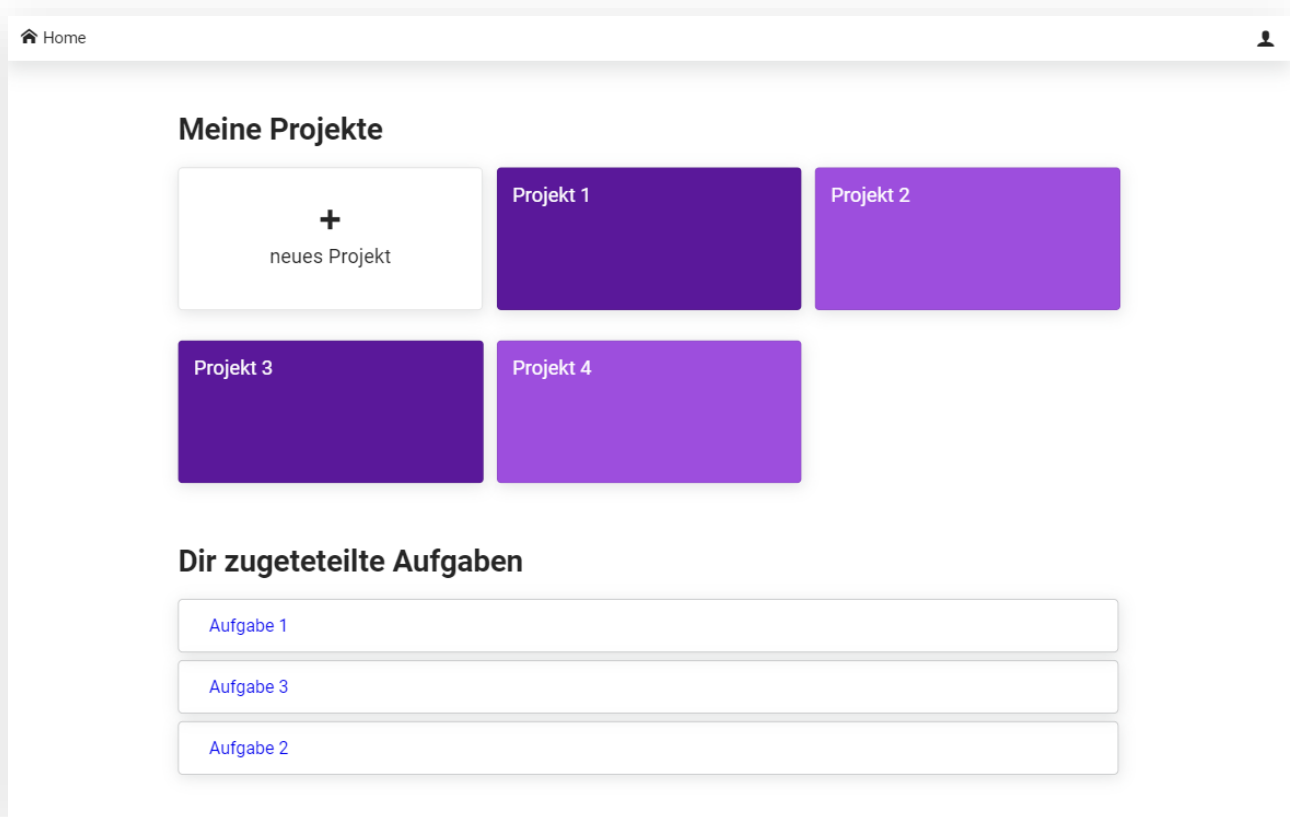


Entwicklung eines webbasierten Projektmanagers mit



Vorgelegt von: Philipp Oeschger

Matr.-Nr.: 257184

Studiengang: Medieninformatik, Bachelor

Modul: Aktuelle Entwicklungen im Bereich Online Medien

Professor: Dr. Dirk Eisenbiegler

Inhaltsverzeichnis

Abbildungsverzeichnis.....	5
Abkürzungsverzeichnis	6
1 Einleitung.....	1
2 Kotlin	2
2.1 Kotlin verglichen mit Java.....	3
3 Ktor	5
3.1 Serverseite in Ktor	5
3.1.1 Die Anwendung	5
3.1.2 Features.....	6
3.1.3 Calls und Pipelines.....	6
3.1.4 Application Call.....	7
3.1.5 Routing	7
3.1.6 Content Negotiation (Inhalts Aushandlung).....	8
3.2 Beispiel einer Ktor-Anwendung.....	9
3.3 Erstellung einer Ktor-Anwendung.....	10
3.4 Features.....	10
3.4.1 Authentifizierung.....	11
3.4.2 Templating.....	11
3.4.3 HTML-Templating über eine DSL.....	12
3.4.4 Templating mit Apache FreeMarker	12
3.4.5 Session Management	13
3.4.6 Statische Inhalte	14
3.4.7 Status Pages.....	14
3.4.8 WebSockets	14

3.5	Mircoservices in Ktor	14
3.6	Ktor vs. Spring.....	15
4	Anwendungsfall	16
4.1	Beschreibung.....	16
4.2	Technologien	16
4.3	Aufbau und Projektstruktur	17
4.4	Die Anwendung	17
4.5	Umsetzung der Datenbank.....	18
4.5.1	Exposed	18
4.5.2	Datenbankstruktur	19
4.5.3	Implementierung der Datenbank.....	19
4.6	Routing der Anwendung.....	21
4.7	Templating.....	21
4.8	Login Mechanismus.....	22
4.8.1	Registrierung	22
4.8.2	Login	24
4.8.3	Seitenauthentifizierung	24
4.9	File-Uploader	25
4.10	Endergebnis.....	25
5	Fazit	28
	Literaturverzeichnis.....	29
	Anhang	31
	Eidesstattliche Erklärung.....	34

Abbildungsverzeichnis

Abbildung 1: Visualisierung, Interpolation Kotlin und Java	3
Abbildung 2: Aufbau, Ktor Anwendung.....	6
Abbildung 3: Darstellung Calls und Pipelines	7
Abbildung 4: Content-Negotiation	8
Abbildung 5: Authentifizierung in Ktor.....	11
Abbildung 6: Registrierungs-Seite	23
Abbildung 7: Login-Seite.....	24
Abbildung 8: Hauptseite.....	26
Abbildung 9: Projektseite	27

Abkürzungsverzeichnis

JVM	Java Virtual Machine
CIO	Coroutine-based I/O
DSL	Domain Specific Language
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
REST	Representational State Transfer
MIME	Multipurpose Internet Mail Extensions
JSON	JavaScript Object Notation
IDE	Integrated Development Environment
HTML	Hypertext Markup Language
DOM	Document Object Model
TCP	Transmission Control Protocol
CSS	Cascading Style Sheets
ORM	Objekt Relational Mapping
MVC	Model-View-Controller
SQL	Structured Query Language
JDBC	Java Database Connectivity

1 Einleitung

Der Markt von Backend-Frameworks für Java-Systeme ist hart umkämpft. Neben bereits bekannten Libraries wie Spring stürzen sich neu Wettstreiter ins Rennen. Dass es nicht immer unbedingt Java sein muss, zeigt das Framework-Ktor.¹ Ein Framework komplett basierend auf der Programmiersprache Kotlin.² Eine moderne Programmiersprache mit vielen innovativen Neuerungen, um den Entwicklungsprozess zu verbessern.

Diese Arbeit soll sich genauer mit diesem Framework befassen. Im Laufe dieser Arbeit soll das Framework Ktor vorgestellt und dessen Möglichkeiten aufgezeigt werden. Es soll die Programmiersprache Kotlin präsentiert werden, auf welcher das Framework beruht und es soll gezeigt werden, inwiefern die Programmiersprache mit diesem Framework zusammenspielt.

Anschließend wird das entwickelte Anwendungsbeispiel vorgestellt. Im Falle dieser Seminararbeit wurde das Framework ausgetestet am Beispiel eines Projektmanagers umgesetzt als Web-Anwendung. Es soll eingegangen werden auf die Architektur und den Aufbau der umgesetzten Applikation. Hierbei soll aufgezeigt werden, welche Funktionalitäten des Frameworks bei der Umsetzung verwendet wurden und welche Funktionalitäten es zusätzlich gibt. Bei der zusätzlichen Verwendung weiter Tools oder Libraries werden diese ebenfalls aufgelistet und erklärt.

Zuletzt soll evaluiert werden, ob und wie gut sich das Framework für den gewählten Anwendungsfall eignet und aus welchen Gründen sich das Framework für den Anwendungsfall geeignet oder nicht geeignet hat.

¹ JetBrains.

² JetBrains.

2 Kotlin

Bevor auf Ktor selbst eingegangen werden kann, muss zunächst dessen tragende Programmiersprache Kotlin vorgestellt werden. Kotlin wurde 2011 veröffentlicht, 2016 die Version 1.0 und wurde vom Unternehmen JetBrains entwickelt. Kotlin ist eine streng typisierte Programmiersprache. Neben normalen Desktop Anwendungen kann Kotlin Code auch in JavaScript-Code umgewandelt werden und seit Android Studio Version 3.0 ist Kotlin neben Java einer der offiziellen Programmiersprachen für die Android-App Entwicklung.³

Kotlin-Code lässt sich in Byte Code übersetzen, welche so für die JVM nutzbar wird. Somit ist Kotlin so gut wie vollständig mit Java interpolierbar. Der Kotlin-Compiler kann Kotlin als auch Java-(Byte-) Code verarbeiten. Diese Interpolation ermöglicht Software-Unternehmen, ihre Java-Klassen Schrittweise in Kotlin-Klassen umzuwandeln, ohne dabei auf Komplikationen zu stoßen. Zuvor genutzte Java Libraries können weiterhin genutzt werden.⁴ Das Zusammenspiel zwischen Kotlin und Java wird in Abbildung 1 visualisiert.

³ alexbuerkle / Stephan Augsten 2019.

⁴ Schleinzner Thorsten 2018.

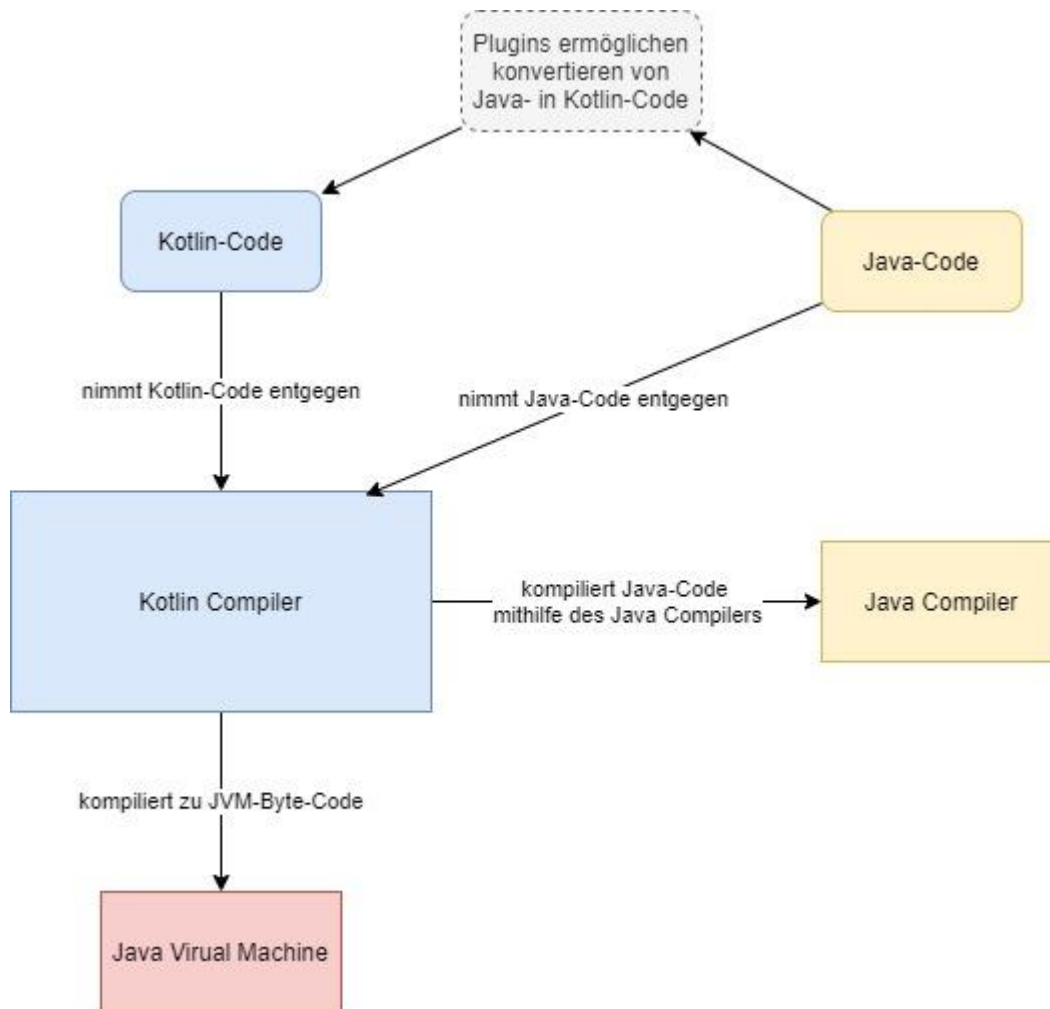


Abbildung 1: Visualisierung, Interpolation Kotlin und Java

2.1 Kotlin verglichen mit Java

In diesen Abschnitt wird genauer auf den Vergleich zwischen Kotlin und Java eingegangen. Dieses Unterkapitel wendet sich an Entwickler, welche bereits mit Java Erfahrung gesammelt haben. Da, wie im vorherigen Abschnitt beschrieben Kotlin mit Java interpolierbar ist, kommt der Vergleich zwischen den beiden Programmiersprachen schnell auf und es stellt sich die Frage, worin sich Kotlin von Java unterscheidet.

Hier sind einige der wichtigsten Kotlin-Features zusammengefasst:

- Variablen in Kotlin sind Nullsafe (können nur dann den Wert Null erhalten, wenn entsprechend deklariert).
- Semikolons am Ende einer Zeile können wegelassen werden.
- In Kotlin sind Datentypdeklarationen nicht zwingend (Datentypen weist der Kotlin-Compiler zur Laufzeit selbst zu).
- String Interpolation.

- Unterstützung von Lambda-Ausdrücken.
- Unterstützung von Datenklassen.
- Keine Primitive, Typen sind immer Objekte (In Java Unterscheidung: int/Int, float/Float).
- Default Werte für Funktionsparameter sind möglich.
- Code kann in JVM-Byte-Code als auch in JavaScript-Code umgewandelt werden.
- Unterstützung von Erweiterungsfunktionen.
- Möglichkeit Funktionen als Parameter zu übergeben.
- Funktionen benötigen keine Klasse und können auf oberster Ebene definiert werden.

Diese und noch viele weitere Features ermöglichen es Code wesentlich kompakter zu gestalten und damit einige überflüssige Zeilen an Code zu sparen. Alle zuvor genannten Features sind möglich ohne jegliche Performanceeinbußen bei der Kompilierzeit. In der Beschreibung des Anwendungsbeispiels wird in einigen Quellcode-Beispiele die genaue Syntax von Kotlin tiefer erklärt.

3 Ktor

Das Framework Ktor erlaubt es, Server als auch Client-Architektur zu verwalten. Außerdem unterstützt Ktor „Kotlin Coroutines“ und erlaubt somit Asynchronität. Das Framework wird vom Unternehmen JetBrains kontinuierlich weiterentwickelt und basiert deshalb auch auf deren Programmiersprache Kotlin.

Das Framework bietet eine Vielzahl an Features, wodurch verschiedene Umsetzungen von Server, als auch Client-Systemen möglich ist. Serverseitig können z. B. Webanwendungen, HTTP-APIs, REST-APIs oder auch Socket-Systeme umgesetzt werden. Weil Ktor Client-Architektur erlaubt, ist damit die Umsetzung von Microservices möglich.

3.1 Serverseite in Ktor

Ktor bietet einige Funktionen, welche die Serverseitige Entwicklung vieler unterschiedlicher Serversysteme ermöglichen. In diesem Abschnitt wird die Serverseite von Ktor vorgestellt und dessen Features. Außerdem werden die Grundkonzepte der Serverseite in Ktor erklärt.

3.1.1 Die Anwendung

Eine Kotlin-Anwendung besteht immer aus einer Application-Klasse. Eine Anwendung kann aus einem oder mehreren Modulen bestehen. Ein Modul ist ein Lambda (Anonyme Funktion) oder eine reguläre Funktion, welche meist eine Instanz der Applikation als Parameter übergeben bekommt. Ein Ktor-Server wird innerhalb einer Umgebung gestartet. Eine Applikation wird mithilfe der Umgebung initiiert oder beendet. Zum Ausführen einer Umgebung wird eine ApplicationEngine benötigt. Ktor unterstützt vier Engines, diese sind CIO, Jetty, Netty, Tomcat. Anwendungs-Module werden beim Starten dieser Anwendung gestartet. Durch das Installieren von Features lässt sich die Anwendung konfigurieren. Außerdem besitzt die Anwendung eine Konfigurationsdatei.

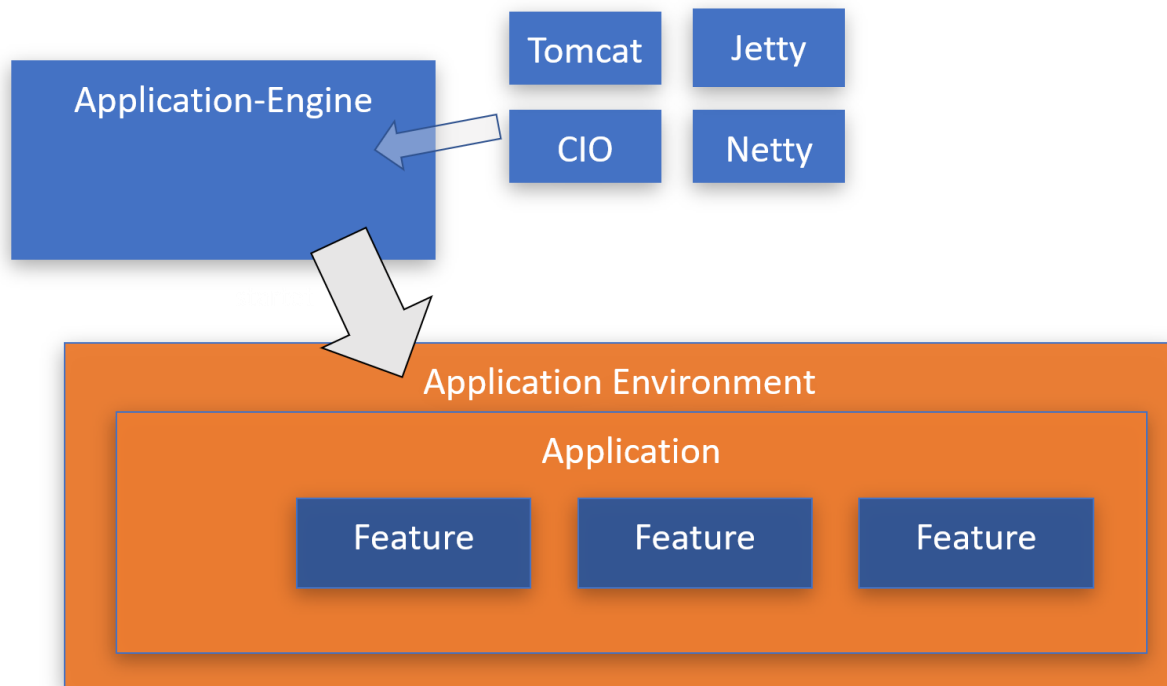


Abbildung 2: Aufbau, Ktor Anwendung

3.1.2 Features

Ein Feature ist eine Funktion, welche der Anwendung hinzugefügt werden kann. Oft fangen Features Anfragen oder Antworten ab und fügen der Anfrage ihre Funktionalitäten hinzu. Features können über die „Install()“-Funktion hinzugefügt werden.

3.1.3 Calls und Pipelines

Ein Paar aus Anfrage und Antwort wird in Ktor ein „ApplicationCall“ genannt. Diese „ApplicationCalls“ werden abgefangen durch die „ApplicationCallPipeline“, welche aus mehreren „Interceptors“ bestehen, die jeweils den ApplicationCall aufhalten, bearbeiten und an den nächsten Interceptor weitergeben. Ein Interceptor kann die Anfrage oder Antwort weiterleiten oder die gesamte Pipeline beenden.

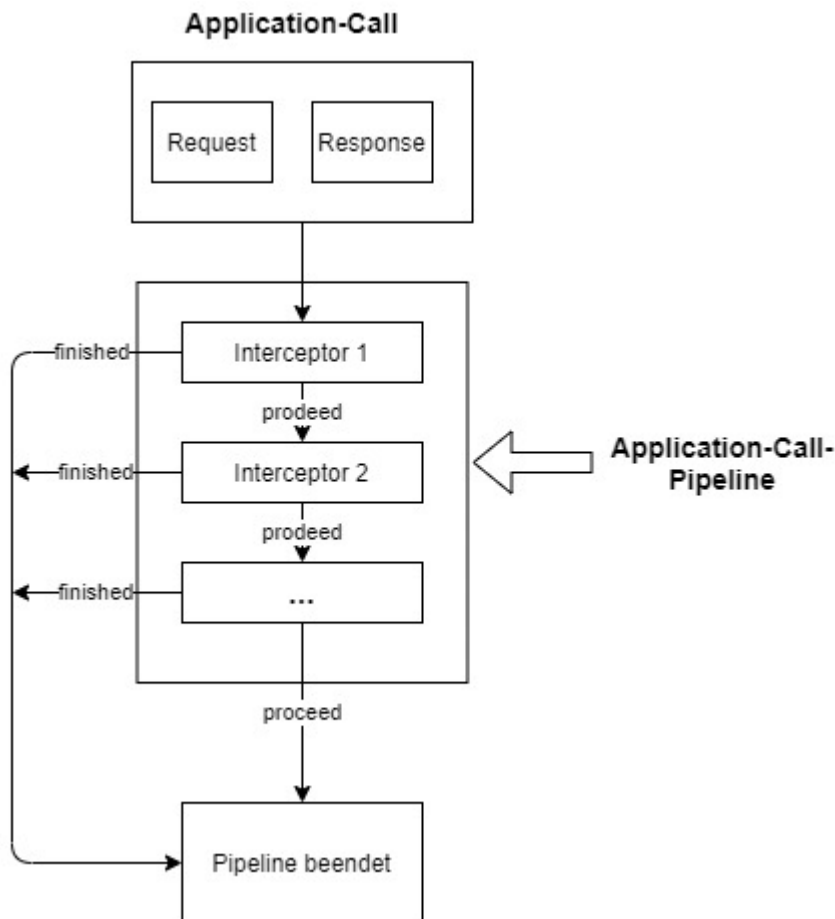


Abbildung 3: Darstellung Calls und Pipelines

3.1.4 Application Call

Wie bereits im vorherigen Abschnitt beschrieben besteht ein Application Call aus einer Anfrage und deren Antwort. Des Weiteren besitzt der ApplicationCall Parameter. Ein ApplicationCall hat also eine receive- (Empfange) und eine send- (Versende) Pipeline. Auf den Körper eines Request kann z. B. mit „call.receive<String>()“ zugegriffen werden. Diese Funktion gibt den Request (Anfrage)-Body als String-Objekt zurück. Hierbei wird der Request durch eine Receive-Pipeline abgefangen, welche den ursprünglichen Request Körper in einen String umwandelt. Ebenfalls kann die Response (Antwort) mit einer Respond-Pipeline abgefangen und bearbeitet werden.

3.1.5 Routing

Ktor unterstützt Routing, eine Funktion mithilfe der Request strukturiert gehandhabt werden können. Hierzu stellt Ktor eine DSL zur Verfügung. Mithilfe des Features können Application-Calls abgefangen werden und mithilfe von Handlern bearbeitet werden. Ktor bietet sinnvolle Extension-Funktionen (einer Funktion von Kotlin) um Routing-Handler zu konfigurieren. Routen lassen sich in einer Baumstruktur ordnen und Parameter innerhalb der Route können abgefangen werden.

3.1.6 Content Negotiation (Inhalts Aushandlung)

Content Negotiation bietet eine Möglichkeit die Inhalts Aushandlung zwischen Client und Server zu Verwalten. Die Content Negotiation ist ein grundlegendes Konzept des HTTP. Der Client sendet zunächst einen Accept-Anfrage-Header mit den möglichen Inhalts-Typen in Form von MIME-Types. Der Server bestimmt daraufhin einen Inhalts-Typ und antwortet daraufhin mit einem Content-Type-Header.⁵

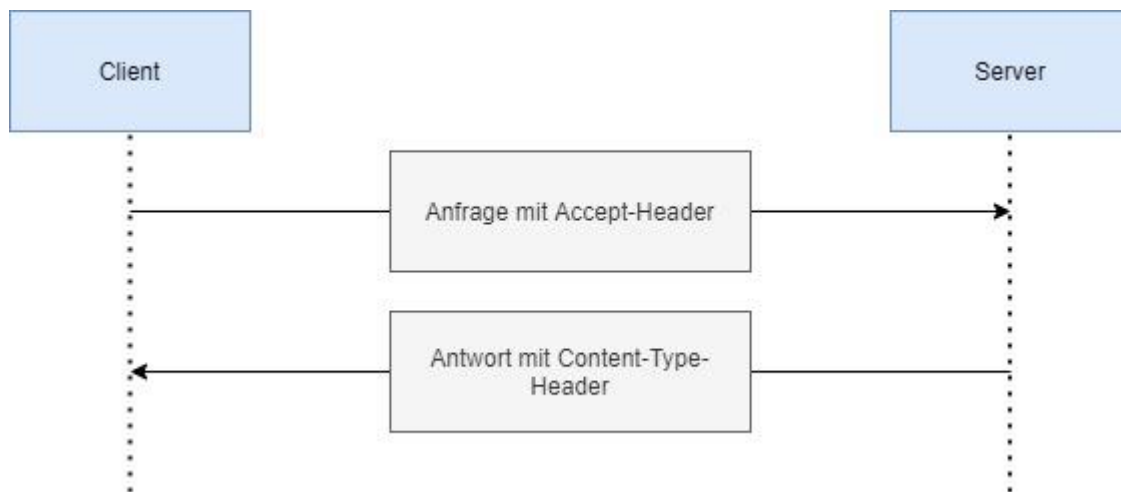


Abbildung 4: Content-Negotiation

Nun bietet Ktor zusätzlich noch Content-Konverter Libraries wie Jackson oder Gson, welche als Feature eingebunden werden können, um z. B. JSON-Objekte zu serialisieren.

⁵ MDN web doc.

3.2 Beispiel einer Ktor-Anwendung

In diesem Abschnitt wird anhand eines Beispiels erklärt, wie eine einfache Ktor-Server-Anwendung aufgebaut ist.

Die Anwendung (Applikation.kt):

```
package com.example

import io.ktor.application.*
import io.ktor.response.*
import io.ktor.routing.*
import io.ktor.http.*

fun main(args: Array<String>): Unit =
    io.ktor.server.netty.EngineMain.main(args)

fun Application.module(testing: Boolean = false) {
    routing {
        get("/") {
            call.respondText("HELLO WORLD!", contentType =
                ContentType.Text.Plain)
        }
    }
}
```

Zunächst wird auf die Anwendungsdatei eingegangen. (vgl. Anwendungs-Code) Wir sehen im Beispiel-Code zunächst eine Main-Funktion, in welcher die Server-Engine (in diesen Fall Netty) gestartet wird. Darunter wurde eine Erweiterungs-Funktion für die Ktor-Klasse Application definiert, welche „module“ heißt. Das ist möglich, weil Kotlin es erlaubt, Erweiterungs-Funktionen für Klassen zu definieren.

Innerhalb der „Application.module()“ Funktion wird die routing-Funktion aufgerufen, die Teil einer Routing Klasse ist. Hier sieht man nun wieder einige Eigenheiten der Programmiersprache Kotlin. Zum einen erlaubt Kotlin in mancher Situation es, die einfachen Klammern nach dem Funktionsnamen wegzulassen. Außerdem kann in Kotlin einer anderen Funktion z. B. eine Lambda-Funktion mitgegeben werden. Die Letzte als Parameter mitgegebene Funktion kann hinter dem Funktionsaufruf definiert werden, was hier auch gemacht wird. Diese Lambda-Funktion dient hier als Konfiguration des Routing-Features. Innerhalb der Konfiguration wird mithilfe eines Aufrufs der Get-Funktion eine neue Route definiert. Es soll ein HTTP-GET-Request abgefangen werden zum Root-Pfad „/“. Innerhalb der Funktion kann in der als Parameter mitgegebenen Anonymen-Funktion nun beschrieben werden, wie nun verfahren werden soll. In Ktor besteht die Möglichkeit, auf ein call-Objekt zuzugreifen, mit dem z. B. Parameter abgefangen werden können oder in diesem Fall eine http-Response zurückgegeben werden kann. Nach dem Aufruf des Pfades wird eine Response mit Inhalts-Typ Text.Plain mit dem Inhalt „Hallo Welt“ zurückgegeben.

Damit das hier definierte Modul beim Starten der Umgebung ausgeführt werden kann, sollte sie in der Konfigurations-Datei (vgl. Konfigurationsdatei-Code unten) angegeben werden. Nach Belieben können auch mehrere Module definiert werden. Es sollten aber alle Module eine Erweiterungs-Funktion der Klasse Application sein.

Konfigurationsdatei (application.conf):

```
ktor {
    deployment {
        port = 8080
        port = ${?PORT}
    }
    application {
        modules = [ com.example.ApplicationKt.module ]
    }
}
```

3.3 Erstellung einer Ktor-Anwendung

Das Erstellen einer Ktor-Anwendung geht bei der Verwendung der IDE IntelliJ sehr einfach. Zunächst muss das Ktor Plugin über den Plugin-Manager installiert werden. Danach erscheint, wenn man in der IDE ein neues Projekt erstellen will ein Reiter „Ktor“. Dort können Einstellungen getroffen werden und bestimmte Ktor-Features registriert werden.

Bei der Verwendung einer anderen IDE kann optional der Projekt-Generator verwendet werden. Das über den Generator erstellte Projekt kann schließlich einfach in der gewünschten IDE importiert werden.⁶

3.4 Features

Ktor bietet Möglichkeiten gewünschte Features innerhalb einer Anwendung zu registrieren. Das Registrieren passiert über das Aufrufen der „install()“-Funktion innerhalb eines Moduls. Das sieht dann folgendermaßen aus:

```
install(ContentNegotiation) {
    jackson {
        enable(SerializationFeature.INDENT_OUTPUT)
    }
}
```

Die „install()“-Funktion wird mit der Klasse des gewollten Features als Parameter aufgerufen. Im Beispiel wird hierbei der JSON-Konverter „Jackson“ konfiguriert.

⁶ JetBrains.

3.4.1 Authentifizierung

Ktor bietet ein Authentifizierungs-Feature, um Anmeldedaten auszulesen und den Anmeldeprozess zu verwalten.

Im Framework gibt es hierfür zwei wichtige Konzepte, Principals und Credentials. Ein Principal ist ein Objekt, welches authentifiziert werden kann, z. B. ein Nutzer oder ein Gerät. Ein Credential repräsentiert eine Gruppe von Eigenschaften, welche verwendet werden, um ein Principal zu authentifizieren. Durch das Verwenden des Authentifikations-Features können eigenen Authentifizierungs-Provider konfiguriert werden, denen dann Routen, die eine Authentifizierung dieses Providers voraussetzen, zugewiesen werden können. Dieser Vorgang wird in Abbildung 8 verdeutlicht. Es wurde ein Authentifikation-Provider konfiguriert was zum Beispiel eine Verknüpfung mit Google-Authentifikation sein könnte. Für Route 1 wird keine Authentifizierung vorausgesetzt. Für Routen 2 und 3 hingegen muss zunächst die Authentifizierung mit dem konfigurierten Provider gegeben sein.

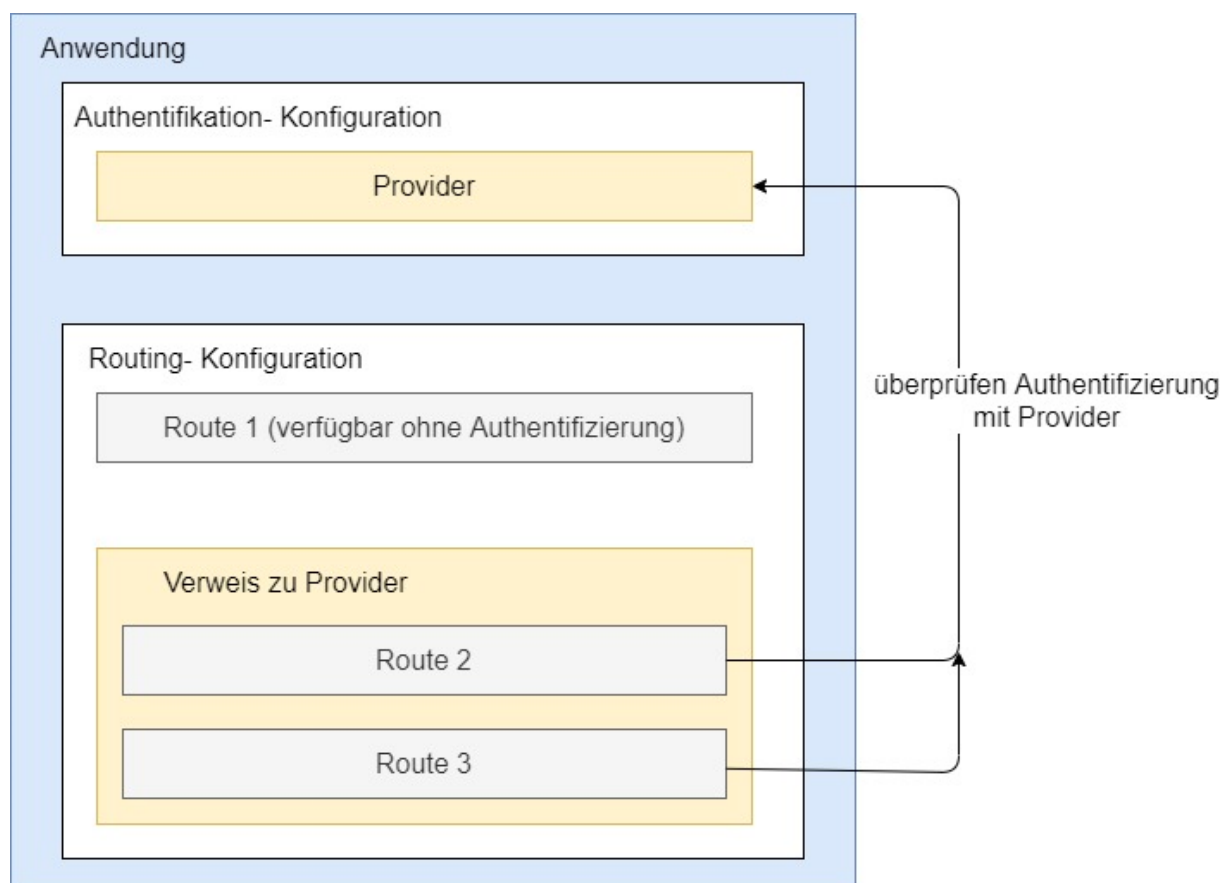


Abbildung 5: Authentifizierung in Ktor

3.4.2 Templating

Ktor unterstützt verschiedene HTML-Templating-Engines, welche es erlauben HTML-Dokumente dynamisch zu gestalten und anzupassen je nach Inhalt bzw. Vorhandensein der übergebenen Daten.

Die Templating Möglichkeiten, welche Kotlin unterstützt sind FreeMarker, HTML-DSL Mustache, Pebble, Thymeleaf und Velocity. Aus Zeitgründen und weil viele dieser Templating-Engines sich oft nur geringfügig syntaktisch unterscheiden, wird im Verlauf der Arbeit nur auf die beiden Variante FreeMarker und HTML-DSL eingegangen.

3.4.3 HTML-Templating über eine DSL

Das Feature HTML-DSL verwendet die Library `kotlinx.html`, um mithilfe einer DSL HTML-Elemente zu erzeugen. Eine DOM Struktur kann in Ktor erzeugt werden durch das Aufrufen der `respondHTML` Methode des `ApplicationCall`-Objekts.

HTML-DSL-Beispiel:

```
call.respondHtml {
    body {
        h1 { +"HTML" }
        p {
            + "Hier steht Text"
        }
        a ("/") {
            + "zum Root Pfad"
        }
    }
}
```

Innerhalb der „`call.respondHtml()`“ Methode lässt sich eine DOM-Struktur einer HTML-Rückgabe definieren (vgl. HTML-DSL-Beispiel). Ein HTML Element kann durch den Tag, danach falls nötig einfache Klammern mit Attributen des Elements, gefolgt von geschweiften Klammern und den Inhalt als String mit einem Plus davor, beschrieben werden. Es ist möglich, innerhalb der Blöcke If-Abfragen und/ oder Schleifen zu definieren. ⁷

3.4.4 Templating mit Apache FreeMarker

Ktor unterstützt die Templating-Engine Apache FreeMarker. Wie mit dem HTML-DSL-Feature ist es auch mit FreeMarker möglich, HTML-Output zu generieren. Im Vergleich zu HTML-DSL, worüber die HTML-Struktur direkt im Code definiert werden kann, werden in FreeMarker Template-Dateien mit der Endung „.ftl“ (FreeMarker Template Language) angelegt. Die Syntax dieser Datei erinnert an eine gewöhnliche HTML-Datei, mit dem Unterschied, dass Funktionen wie If-Abfragen, Iterationen arithmetische Operationen und weitere Möglichkeiten geboten werden.

Freemarker muss über die „`install()`“-Funktion registriert werden und über einen Template-Loader geladen werden. ⁸

⁷ (Jetbrains 2017).

⁸ Apache.

Initialisierung von FreeMarker mit Template-Loader:

```
install(FreeMarker) {  
    templateLoader = ClassTemplateLoader(this::class.java.classLoader,  
    "templates")  
}
```

index.ftl:

```
<html>  
    <body>  
        <h1>Hallo ${user.vorname} ${user.nachname}</h1>  
    </body>  
</html>
```

Aufrufen des Templates:

```
data class User(val vorname: String, val nachname: String)  
  
get("/") {  
    val user = User("Max", "Mustermann")  
    call.respond(FreeMarkerContent("index.ftl", mapOf("user" to user)))  
}
```

In den oberen Code-Schnipseln sieht man ein Beispiel einer Implementierung von FreeMarker in Ktor. Zuerst muss FreeMarker als Feature registriert werden mit der „install()“-Funktion und FreeMarker muss mit einem Template-Loader geladen werden. Im Template-Loader wurde ein Pfad angegeben, welcher vom Root-Verzeichnis ausgehend auf den „Resources/Templates“-Ordner verweist. Innerhalb des Ordners kann eine „index.ftl“ Datei angelegt werden. Innerhalb der Datei kann auf Daten zugegriffen werden. Im Beispiel wird ein „user“-Objekt mit den Attributen „vorname“ und „nachname“ ausgegeben. Das Template kann innerhalb einer Route als Antwort zurückgegeben werden. Im Beispiel wird nun die zuvor erstellte „index.ftl“ aufgerufen und ein Objekt der Daten-Klasse User übergeben. (vgl. Aufrufen des Templates)

Daten-Klassen sind ein sehr praktisches Feature der Programmiersprache Kotlin. Was genau Daten-Klassen sind, wird in der späteren Beschreibung des Anwendungsbeispiels genauer erklärt.

3.4.5 Session Management

Ktor bietet die Möglichkeit, vereinfacht Sessions zu verwalten. So können Daten eines Clients innerhalb einer Reihe von HTTP-Anfragen und Antworten gespeichert werden. Sessions können entweder innerhalb eines Cookies gespeichert oder über den Header übertragen werden. Sessions können entweder Client-Seitig gespeichert werden und das gesamte Session-Objekt wird zwischen

Server und Client hin und her geschickt oder Serverseitig und es wird nur die Session-ID ausgetauscht, während sich das Session-Objekt weiterhin auf dem Server befindet. Innerhalb einer Anwendung kann auf die Session über „call.sessions“ zugegriffen werden.

3.4.6 Statische Inhalte

Ktor unterstützt es, statische Inhalte zu übertragen. Dieses Feature eignet sich hervorragend, um Inhalte wie Stylesheets, Skripte oder Bilder innerhalb einer Seite einzubinden. Mit dem Feature kann registriert werden, in welchem Ordner sich statische Dateien befinden und welche Dateiendungen diese haben.

3.4.7 Status Pages

Das Status-Page-Feature ermöglicht es auf Fehler-Zustände innerhalb der Anwendung zu reagieren. Mithilfe des Features kann z. B. für Exceptions oder HTTP-Statuscodes eine passende Rückmeldung gegeben oder auf eine andere Seite innerhalb der Anwendung weitergeleitet werden.

3.4.8 WebSockets

WebSockets bietet eine API, welche es ermöglichen, Bidirektionale Verbindung zwischen Server und Client aufzubauen und Informationen in Echtzeit auszutauschen. Das WebSocket-Protokoll basiert auf dem TCP.⁹ Websockets werden in Ktor unterstützt durch dieses Feature.

3.5 Microservices in Ktor

Ktor ermöglicht die Umsetzung von Microservices. Im Grunde gibt es keine klare Definition von Microservices. Dennoch haben alle Microservices eins gemeinsam, und zwar, dass sie auf der Unix-Philosophie aufbauen „do one thing and do it well“, erledige eine Aufgabe und erledige diese gut.

In der Regel beschreiben Microservices mehrere kleine Services, die unabhängig voneinander arbeiten und auch auf unterschiedlichen Technologien basieren können. Diese Services werden bei einer Microservice-Architektur zu einem großen Service verbunden. Gegensätzlich zur Microservice-Architektur wird in gewöhnliche Systeme meist ein monolithisches System verfolgt, also befinden sich alle Services in einer Anwendung, während bei einem Microservice System Services individuell und unabhängig voneinander funktionieren.¹⁰

Damit eine Microservice Architektur für eine Server-Anwendung verwirklicht werden kann, muss dieser in der Lage sein, Anfragen an andere Systeme zu stellen. Nun ist die Aufgabe eines Servers in erster Linie nur, Anfragen entgegenzunehmen und zu bearbeiten. Um dazu noch zu ermöglichen, Anfragen an andere Systeme zu schicken, wird ein Client benötigt. Glücklicherweise ermöglicht Ktor

⁹ Digital Guide IONOS 2020b.

¹⁰ Digital Guide IONOS 2020a.

die Umsetzung von http-Clients. Somit wird es möglich, Anfragen über ein Netzwerk zu schicken. Eine andere Möglichkeit, ist es eine Library für das umsetzen eines Clients zu verwenden. Eine Mögliche Library für diesen Anwendungsfall wäre z. B. Retrofit.¹¹

Das Framework Ktor selbst stellt keine Vorgaben dazu, wie eine Microservice-Architektur umgesetzt werden muss. Ktor selbst bietet Tools und Denkanstöße für die Umsetzung setzt aber keine Grenzen und lässt dem Entwickler frei entscheiden ob er nun die Funktionalitäten von Ktor selbst nutzen, eine externe Library verwenden oder ein eigenes Modul entwickeln möchte.

3.6 Ktor vs. Spring

Auf dem Markt muss sich Ktor gegen andere etablierte Frameworks beweisen. Eine der wohl weitverbreitetsten Java-Backend Lösungen, ist das Framework-Spring.

Während Spring auf Java basiert und Kotlin nur unterstützt, wurde Ktor hingegen komplett in Kotlin geschrieben und bietet deshalb Features wie Coroutines oder Extension-Functions. Für Routing wird in Ktor eine DSL angeboten, welche das Erstellen von Controllern wesentlich vereinfacht und übersichtlicher gestaltet. Viele der aus Spring bekannten Features findet man in Ktor wieder.

Zum Thema Lesbarkeit macht Ktor einige nützliche Änderungen. Keine Annotationen werden benötigt und mit ein wenig Übung mit der von Ktor gebotenen DSL, wird es ziemlich einfach, Code zu refactor (Strukturverbessern des Quelltextes) und z. B. kleinere Codeschnipsel oder Funktionen in eigene Dateien auszulagern.

In Sachen Community und Funktionalitäten kann Ktor mit Spring nicht mithalten. Aber viele der Funktionen, welche in Ktor umgesetzt wurden, werden sehr gut gelöst. Für jeden Entwickler, dem die Kotlin-Syntax gefällt und der ein Kotlin-basiertes Backend-Framework ausprobieren möchte, für den lohnt es sich, Ktor auszutesten. Die Online-Dokumentation ist sehr gut durchdacht und übersichtlich gestaltet, kann also mit der Dokumentation von Spring gut mithalten. Auch was Skalierbarkeit angeht, sollte Ktor gegenüber Spring keine Nachteile haben.¹²

¹¹ Medium 2022.

¹² Möller 2018.

4 Anwendungsfall

4.1 Beschreibung

Ziel war es, eine Projektmanager-Applikation zu verwirklichen, welche auf Webtechnologien basiert. Hierbei sollten folgende Funktionalitäten mindestens gegeben sein:

- Anlegen von Nutzern.
- Login Funktion.
- Anlegen und Verwalten von Projekten.
- Hinzufügen von Nutzern zu Projekten.
- Hinzufügen von Aufgaben zu Projekten.
- Zuteilen von Nutzern zu Aufgaben.
- Veröffentlichen von Nachrichten innerhalb eines Projektes.

Falls zeitlich möglich sollten folgende Funktionen zusätzlich umgesetzt werden:

- File-Uploader.
- Einzelchat zwischen Personen.
- Kalenderfunktion zum zeitlichen Einplanen von Aufgaben.
- Implementieren und setzen von Meilensteinen.

Alle wichtigen Daten sollen in einer Datenbank persistent gespeichert werden.

In den weiteren Unterkapiteln werden die Rahmenbedingungen und die Umsetzung der Funktionalitäten genauer erklärt.

4.2 Technologien

Zur Verwirklichung des Anwendungsfalls bietet Ktor einige Features, die sich für die Anwendung eignen. Viele der Features wurden bereits im vorherigen Kapiteln vorgestellt. (vgl. 3.4)

Für die Umsetzung wurden folgenden Features verwendet. Das standardmäßig eingebundene Routing-Feature wird innerhalb des Projektes genutzt für die Verwaltung der http-Requests. Zur Umsetzung der Login Funktion und der Zugriffsverwaltung von Seiten wurde das Authentifikation-Feature verwendet. Nach einem Login wird der Nutzer in einer Session gespeichert, dafür wird das Session-Management Feature von Ktor verwendet. Für einzelne Unterseiten sollen Templates erstellt werden. Hierzu wurde die Templating-Engine FreeMarker verwendet. Stylesheets und JavaScript werden mithilfe der Verwaltung von statischen Inhalten für Ktor eingebunden. Falls Bestimmte Exceptions oder http-Zustände auftreten sollten, sollen diese entsprechend gehandhabt werden mit dem Status-Pages-Feature.

Zum Verwalten des Stylings wird Sass verwendet,¹³ eine Erweiterungs-Sprache welche gewöhnliches CSS mit weiteren Funktionalitäten wie Variablen oder Schleifen erweitert. Sass vereinfacht die Pflege großer Styling-Dateien. Um den Rahmen nicht zu sprengen und da Sass für die Funktionalität der Web-App keine Rolle spielt, wird in dieser Arbeit nicht weiter auf Sass eingegangen.

Für das Management der Datenbank wurde das ORM-Framework Exposed verwendet.¹⁴ Als Datenbank wird eine PostgreSQL-Datenbank benutzt.¹⁵ Durch das ORM-Framework sind aber auch andere Arten von Datenbanken verwendbar.

4.3 Aufbau und Projektstruktur

Die Ordnerstruktur setzt sich einerseits aus den bei der Erstellung eines Ktor-Projekts vorgenerierten Strukturen mit Verzeichnissen, Konfigurationsdateien und aus der intern selbst gewählten Struktur von Verzeichnissen und Dateien. Eine Beschreibung der wichtigsten Elemente der Projektstruktur ist im Anhang zu finden.

Es wurde sich an einer MVC-Struktur orientiert mit dem Unterschied, dass Views (Templates) nicht nach Änderung des Datenmodells (Model) aktualisiert werden, sondern erst bei einem laden der Seite die Steuerung(Controller) das Template und die erfordernten Daten aufruft.¹⁶ Die Vorteile einer klaren Trennung zwischen Steuerung, Darstellung, Geschäftslogik und das problemlose Ändern der einzelnen Komponenten, ohne dabei die anderen Elemente zu beeinflussen, bleibt dennoch erhalten. Wenn man die Struktur mit einem MVC-Modell vergleicht, enthält der „Resources/Template“-Ordner die Views, das „src/Routes“-Package die Controller und das „src/database“-Package enthält die Geschäftslogik bzw. die Daten-Modelle.

4.4 Die Anwendung

Die Anwendung befindet sich in „src/Applikation.kt“, genauer ist die Anwendung die Funktion „Application.module()“. Diese Funktion wird beim Start der Anwendung in einer Anwendungs-Umgebung ausgeführt. (vgl. 3.1.1, 3.2) Am Anfang werden zunächst die benötigten Feature-Registriert durch Aufrufen der „install()“-Funktion. Darunter zählt FreeMarker (Templating), Sessions (Session-Handhabung), Authentication (Authentifizierungs-Verwaltung) und Status-Pages (Exception- und Status-Handhabung). Für die Authentifizierung muss ein Authentifikation-Provider definiert werden. Der Provider wurden ausgelagert in „src/general/Authentifizierung.kt“. Nach der Registrierung wird, falls nötig ein Upload-Ordner erstellt. Danach wird das Datenbank-Objekt

¹³ Sass 2006.

¹⁴ JetBrains 2016.

¹⁵ PostgreSQL 1996.

¹⁶ D. 2018.

initialisiert. Zuletzt werden Routen (Controller) definiert. Alle Routen (Controller) wurden in „src/Routes“ ausgelagert.

4.5 Umsetzung der Datenbank

Wie bereits erwähnt ist die Datenbank eine PostgreSQL-Datenbank. PostgreSQL bedient sich, wie der Name andeuten lässt an den SQL-Standards. Um nicht zu weit vom eigentlichen Thema abzulenken und da das ORM-Framework eine größere Rolle bei der Umsetzung der Datenbank spielte, wird auf PostgreSQL in laufe der Arbeit nicht weiter eingegangen. Außerdem unterstützt das ORM-Framework Exposed viel verschiedene Arten von Datenbanken. Es wird lediglich der passende Treiber benötigt.

4.5.1 Exposed

Exposed ist ein ORM-Framework. Diese Art von Frameworks ermöglicht es, in einer Objektorientierten-Programmiersprache, ein Objekt in einer relationalen Datenbank abzubilden (mapping). ORM-Frameworks bilden die Tabellen einer Datenbank ab. Diese Objekte können dann verwendet werden, um mit der Datenbank bzw. Datenbanktabelle zu interagieren.

ORM-Frameworks bieten vielerlei Vorteile. Sie erlauben das übersetzten der Datentypen aus der objektorientierten Programmiersprachen in für die Datenbank verwendbare Datentypen und umgekehrt von Datenbank-Datentypen in Datentypen für die Programmiersprache. ORM-Frameworks schützen eine Datenbank vor SQL-Injections. Insgesamt fügen ORM-Framework ein weiteres Abstraktionslevel zur Datenbank hinzu, wodurch der Umgang mit Datenbanken erleichtert wird.¹⁷

Exposed ist ein Kotlin spezifisches ORM-Framework, das neben den bereits im Vorabschnitt genannten Aspekten noch weitere Funktionen mit sich bringt. Zunächst unterstützt das Framework wie bereits erwähnt viele verschiedene Datenbanken, darunter zählen MySQL, PostgreSQL, MariaDB, H2 und viele mehr. Zusätzlich bietet Exposed eine DSL welche sich mehr an SQL-Statements orientieren, ist also von Vorteil für Entwickler, welche schon etwas Erfahrung mit SQL haben. Ebenfalls ist es möglich, JDBC-Connection Pooling mit z. B. HikariCP zu implementieren.¹⁸

Für Entwickler, die ein Build-Tool verwenden (Ktor nutzt standardmäßig Gradle) sollte die Einbindung des Frameworks keine großen Schwierigkeiten bereiten.¹⁹ Es müssen nur die Abhängigkeiten des Frameworks in der Konfigurationsdatei des Build-Tools eingefügt werden, danach kann das Framework innerhalb des Projektes genutzt werden.

¹⁷ Killerphp 2009.

¹⁸ brettwooldridge 2013.

¹⁹ Gradle 2009.

4.5.2 Datenbankstruktur

Die Datenbank setzt sich aus folgenden Tabellen zusammen, welche falls sinnvoll mit Fremdschlüssel miteinander verbunden sind:

- Projects: Enthält alle Projekte.
- Users: Enthält alle Nutzer.
- Messages: Enthält Nachrichten der Projekte.
- Tasks: Enthält alle Aufgaben der Projekte.
- Files: Speichert Speicherort der in Projekten hochgeladenen Dateien.
- ProjectUsers: Verknüpft Projekte mit Nutzern.
- TasksUsers: Verknüpft Aufgaben mit Nutzern.

Im Anhand kann die Verbindung zwischen den einzelnen Tabellen in einem vereinfachten ER-Diagramm genauer betrachtet werden.

4.5.3 Implementierung der Datenbank

Unter „src/database“ finden sich alle Dokumente wieder, welche zum Verwalten und Initialisieren der Datenbank benötigt werden. In einer Konfigurationsdatei mit dem Namen „databaseConfig.kt“ werden Metadaten wie die jdbc-URL, die Treiberklasse, der Nutzernamen und das Passwort festgelegt. Außerdem kann die Anzahl der Pools für das Connection-Pooling bestimmt werden.

Datenbank Konfiguration („databaseConfig.kt“):

```
val config = HikariConfig().apply {  
    jdbcUrl = "jdbc:postgresql://localhost:5432/projektmanager"  
    driverClassName = "org.postgresql.Driver"  
    username = "postgres"  
    password = "123456"  
    maximumPoolSize = 10  
}
```

Es gibt eine „models.kt“-Datei, welche alle Datenklassen der Anwendung enthält. Datenklassen sind ein Konzept der Programmiersprache Kotlin welche es ermöglichen Daten zu bündeln. Das Nutzen von Datenklassen bietet vielerlei Vorteile, z. B. dass automatisch eine „equals()“- und eine „hashCode()“-Methode zur Verfügung gestellt werden. Ebenfalls wird eine „toString()“-Methode bereitgestellt.²⁰ In diesem Fall eignen sich Datenklasse hervorragend zum Zwischenspeichern der Datenbank Elemente. Im folgenden Code-Schnipsel sieht man als Beispiel die Datenklasse „Project.“

²⁰ Schleinzer Thorsten 2018.

Project-Datenklasse

```
data class Project(val id: Int, val name: String, val description: String,
val deadline: LocalDate?)
```

Das Framework erlaubt es Abbildungen der Tabellen als Kotlin-Objects zu definieren. Die „objects“-Bezeichnung in Kotlin beschreibt singleton Klassen, also Klassen, von denen es nur eine Instanz geben kann. Im Fallbeispiel der Datenbank-Tabelle ergibt das einen Sinn, da es jedes Tabellen-Objekt in der Datenbank nur einmal geben kann. Im unteren Code-Schnipsel sieht man ein Beispiel einer Klasse mit der „object“-Bezeichnung.

Project-Tabellen-Objekt:

```
object Projects : Table() {
    val id: Column<Int> = integer("id").autoIncrement()
    val name: Column<String> = varchar("name", 50)
    val description: Column<String> = varchar("description", 500)
    val deadline: Column<String?> = varchar("deadline", 500).nullable()

    override val primaryKey = PrimaryKey(id, name = "PK_Project_ID");

    fun toProject(row: ResultRow): Project =
        Project(
            id = row[id],
            name = row[name],
            description = row[description],
            deadline = if (row[deadline] != null)
                LocalDate.parse(row[deadline]) else null
        )
}
```

Wie man feststellen kann, erbt diese Klasse von Table, eine Klasse, die durch das Framework Exposed bereitgestellt wird und eine Abbildung einer Datenbank-Tabelle verwirklicht. Innerhalb der Klasse wird eine Methode „toProject(...)“ definiert, welche einfach nur einen Datenbankeintrag zum passenden Datenklassenobjekt konvertiert.

Letztendlich wurde eine weitere Singleton-Klasse mit dem Namen „DatabaseObject“ erstellt. Diese Klasse soll als Abbild der Datenbank genutzt werden. Die Klasse dient zum Initialisieren der Datenbank und enthält alle Methoden, welche zum Zugriff auf die Datenbank benötigt werden. In der „Init()“-Methode. Wird zunächst mithilfe der Datenbank-Konfiguration ein DataSource-Objekt initialisiert. Außerdem wird überprüft, ob alle Tabellen in der Datenbank bereits existieren, falls nicht werden diese erstellt. Dafür bietet das Framework Exposed eine Funktion. (SchemaUtils.create([Tabellenklasse])).

4.6 Routing der Anwendung

Innerhalb des Projekts werden einige Routen zum Verwalten der http-Requests definiert. Viele der Routen sind http-Get-Requests, die meist von der Datenbank benötigte Daten zunächst abfragen und mit diesen dann ein Template aufrufen. Andere Routen sind http-Post-Request und übergeben Formular-Inhalte, welche dann die Daten in der Datenbank hinterlegen.

Controller für Get-Request zum Pfad “/”:

```
get("/") {
    val session = call.sessions.get<MySession>()

    if (session != null) {
        val projects = DatabaseObject.getUsersProjects(session.email);
        val tasks = DatabaseObject.getUserTasks(session.email);

        call.respond(
            FreeMarkerContent(
                "index.ftl", mapOf(
                    "data" to session,
                    "projects" to projects,
                    "tasks" to tasks
                )
            )
        )
    } else {
        throw AuthorizationException()
    }
}
```

Oberhalb sieht man als Beispiel einen Controller für den Get-Request an den Pfad „/“. Die Funktion lässt sich als Erweiterungsfunktion (Ein Funktionalität der Programmiersprache Kotlin) der Routing-Klasse definieren. Zunächst wird das Session-Objekt abgefragt. Falls das Session-Objekt vorhanden ist, wird mithilfe der darin enthaltenen Nutzer-E-Mail die Projekte und die Aufgaben des Nutzers aus der Datenbank abgefragt. Danach wird das passende Template abgerufen. Als Daten werden die Session selbst, die Projekt-Liste und die Aufgaben-Liste an das Template übergeben. Falls keine Session vorhanden ist, wird eine Exception geworfen.

Viele der http-Get-Request-Controller verfahren ähnlich nach diesem Schema. Die Session wird abgefragt. Für das Template werden die benötigten Daten aus der Datenbank abgefragt und letztendlich wird das Template aufgerufen und die Daten werden übergeben.

4.7 Templating

Wie bereits erwähnt bietet Ktor viele verschiedenen Templating-Engines zur Nutzung an. (vgl. 3.4.2) Für meinen Anwendungsfall habe ich mich für die Templating-Engine FreeMarker entschieden. Gründe dafür sind, dass FreeMarker alle Funktionalitäten bietet, die man bereits aus anderen Templating

Engines kennt (z. B. Thymeleaf, Velocity). Im Gegensatz zur vorgestellten Variante, HTML-DSL bietet FreeMarker durch die stark an HTML-orientierte Syntax eine bereits vertraute Umgebung. Außerdem ist es zwar möglich, aber nicht wirklich intuitiv in der HTML-DSL-Variante die Darstellung von der Programmlogik zu trennen, was wiederum der Aufteilung in Model, View und Controller schadet. (vgl. 3.4.3)

Templates werden im Unterordner „resources/templates“ gespeichert im Format „.ftl“. Einige Elemente welche mehrfach in mehreren Templates benötigt werden (Header, Navigation, ...), wurden als Teil-Templates im Unterordner „resources/templates/partials“ gespeichert. Diese Template-Teile werden dann in den Haupttemplates des Überordnens benutzt.

4.8 Login Mechanismus

Für die Umsetzung des Logins, waren verschiedene Komponenten relevant. Für die Zugriffsberechtigung auf einzelnen Seiten wurde das Authentifikation Feature benötigt. Zum Zwischenspeichern der Nutzer wurde das Ktor eigenen Session-Management benutzt. Der Nutzer wird in der Tabelle „User“ in der Datenbank hinterlegt. Es wurden hierfür 2 Templates erstellt, ein Registrierungsformular und ein Login-Formular.

4.8.1 Registrierung

Zunächst soll der Nutzer in der Lage sein sich zu registrieren. Beim nicht eingeloggten Seitenaufruf wird der Nutzer zunächst auf die Login-Seite weitergeleitet. Über die Login-Seite gelangt der Nutzer zur Registrierungsseite.

Registrierung

Email:

Full Name:

Password:

Confirm Password:

Registrieren

[Sie haben bereits einen Account? Zur Anmeldung](#)

Abbildung 6: Registrierungs-Seite

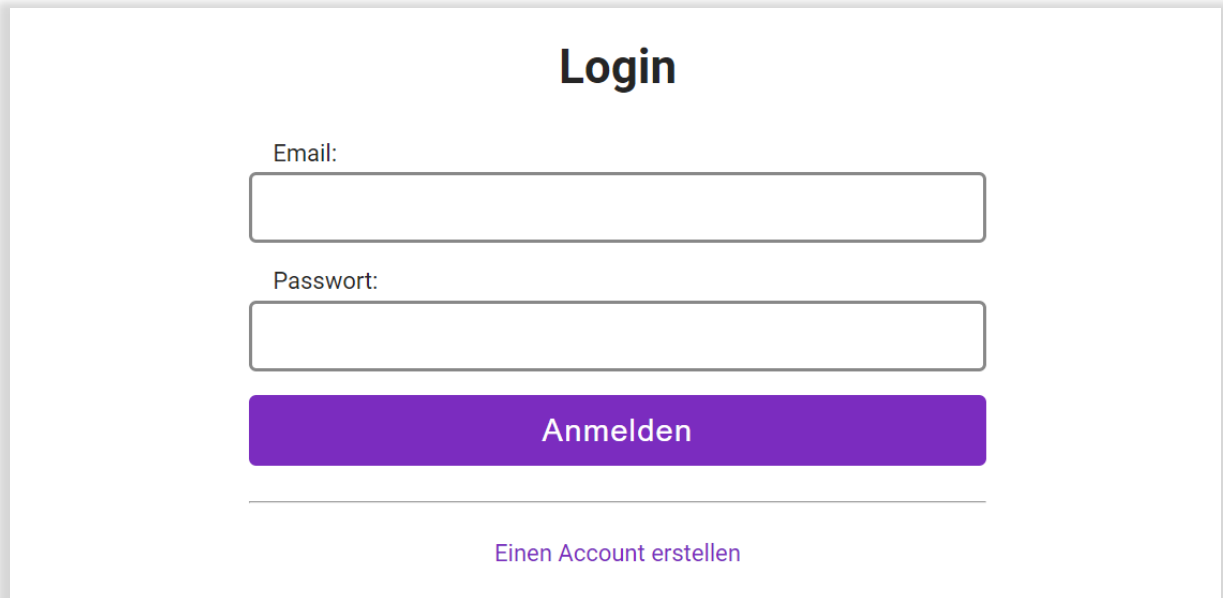
Über das Formular kann der Nutzer sich mit seinen Daten registrieren. Nach dem Abschicken des Formulars wird die passende Route mit einem http-Post-Request abgerufen. Der dazugehörige Pfad ist in diesem Fall „/signup“, übrigens der gleiche Pfad, über den mit einem http-Get-Request das Formular selbst abgerufen wird. Das ist in Ktor ohne Probleme möglich. Zunächst wird überprüft, ob die E-Mail des Nutzers bereits in der Datenbank vorhanden ist, falls ja schlägt der Registrierungsprozess fehl.

Als Nächstes stellt sich die Frage, wie sich am sichersten das Passwort in der Datenbank speichern lässt. Ich habe mich bei meiner Registrierung für eine SaltHash-Methode entschieden. Anstelle nur das Passwort in gehashter Form abzuspeichern wird beim hashen des Passwortes ein Salt (für jeden Nutzer zufällig generierter Wert) hinzugefügt. In der Datenbank wird dann für jeden Nutzer der gehashte Wert und der Salt gespeichert. Dieses Verfahren macht die Speicherung des Passwortes wesentlich sicherer, als wenn lediglich das Passwort ohne Salt gehasht und in der Datenbank abgespeichert wird.²¹

²¹ Arias 2018.

4.8.2 Login

Nachdem der Nutzer registriert wurde, ist er in der Lage, sich mit seiner E-Mail-Adresse und seinem Passwort auf der Login-Seite einzuloggen (vgl. Abbildung 16).



The image shows a login form titled "Login". It contains two input fields: "Email:" and "Passwort:". Below the "Passwort:" field is a purple button labeled "Anmelden". At the bottom of the form, there is a link that says "Einen Account erstellen".

Abbildung 7: Login-Seite

Beim Login wird aus der Datenbank der Nutzer mit der passenden E-Mail-Adresse geholt. Nun wird das der Salt des Nutzers mit dem im Login eingegebenen Passwort gehasht. Falls der entstandene Hash gleich mit dem Hash aus dem Datenbank-Eintrag ist, wird der Nutzer eingeloggt. Wenn der Nutzer sich erfolgreich einloggt, wird für ihn eine Session erstellt. Die Session wird als Cookie clientseitig gespeichert. Es wäre auch möglich, den Cookie serverseitig zu speichern oder die Session über den Header zu übertragen, aus Simplifizierung und Zeitgründen habe ich für einen clientseitigen Cookie entschieden. (vgl. 3.4.5). Wenn der Nutzer sich ausloggt, wird der Session-Cookie wieder gelöscht.

4.8.3 Seitenauthentifizierung

Bis auf die Login- und die Registrierungsseite, sind alle anderen Seiten durch das Authentifizierungs-Feature abgesichert.

Definieren von durch Authentifizierung abgesicherten Request-Pfaden:

```
authenticate("login") {  
  get("/") {  
    // Code  
  }  
}
```

Um eine Abfrage abzusichern, muss diese Route innerhalb einer „authentication(...)“-Funktion definiert werden. Der Funktion wird als Parameter der Name eines definierten Authentication-Providers übergeben, sowie eine Lambda-Funktion, welche die abgesicherte Route enthält. Nun wird in diesem Beispiel bei jedem Get-Request an den Pfad „/“ zuerst der Authentication-Provider mit dem Namen „login“ abgerufen.

Der „login“-Authentication-Provider ist folgendermaßen aufgebaut. Zuerst wird nach einer Nutzer-Session gesucht. Falls eine Session vorhanden ist, wurde der Nutzer bereits eingeloggt und der Authentifizierungsprozess kann übersprungen werden (Nutzer bekommt Zugriff auf die Seite). Falls keine Nutzer-Session besteht, wird der Nutzer zur Login-Seite weitergeleitet und muss sich zuerst einloggen. Danach erst hat der Nutzer Zugriff auf die abgesicherten Seiten.

4.9 File-Uploader

Für einzelne angelegte Projekte sollte es möglich sein, Dateien hochzuladen, um zu ermöglichen, Dateien innerhalb eines Projekts auszutauschen.

Ktor ermöglicht das Handhaben von Uploads. Über das bei jeder Anfrage übergebene call-Objekt kann durch das Aufrufen der Methode „call.receiveMultipart()“ ein http-multipart-request abgefangen werden. So lassen sich übertragene Dateien abfangen. Die Hochzuladende Datei bekommt einen neuen Dateinamen, welcher einen Zeitstempel und die Projektnummer enthält. Die Datei wird in einem bei der Initialisierung des Projekts angelegten „Upload“-Ordner hochgeladen. In der Datenbank wird die URL, als auch der ursprüngliche Dateiname abgespeichert. Beim Herunterladen der Datei bekommt die Datei wieder den ursprünglichen Dateinamen zugewiesen.

4.10 Endergebnis

Abschließend zu Seminar-Arbeit wurden folgende Funktionalitäten umgesetzt.

Der Nutzer ist dazu in der Lage sich zu registrieren. Danach kann er sich mit seinen persönlichen Login-Daten ein und ausloggen. (vgl. 4.7) Außerdem ist er dazu in der Lage, seine Nutzerdaten zu ändern.

Nachdem der Nutzer sich eingeloggt hat, landet er auf der Hauptseite. Beim Anlegen des Projektes wird dieses Projekt in der Datenbank gespeichert und der Ersteller wird dem Projekt hinzugefügt. Zusätzlich werden auf der Hauptseite alle Projekte angezeigt, in welchen der Nutzer ein Mitglied ist und alle Aufgaben, die ihm zugeteilt wurden. (vgl. Abbildung 18)

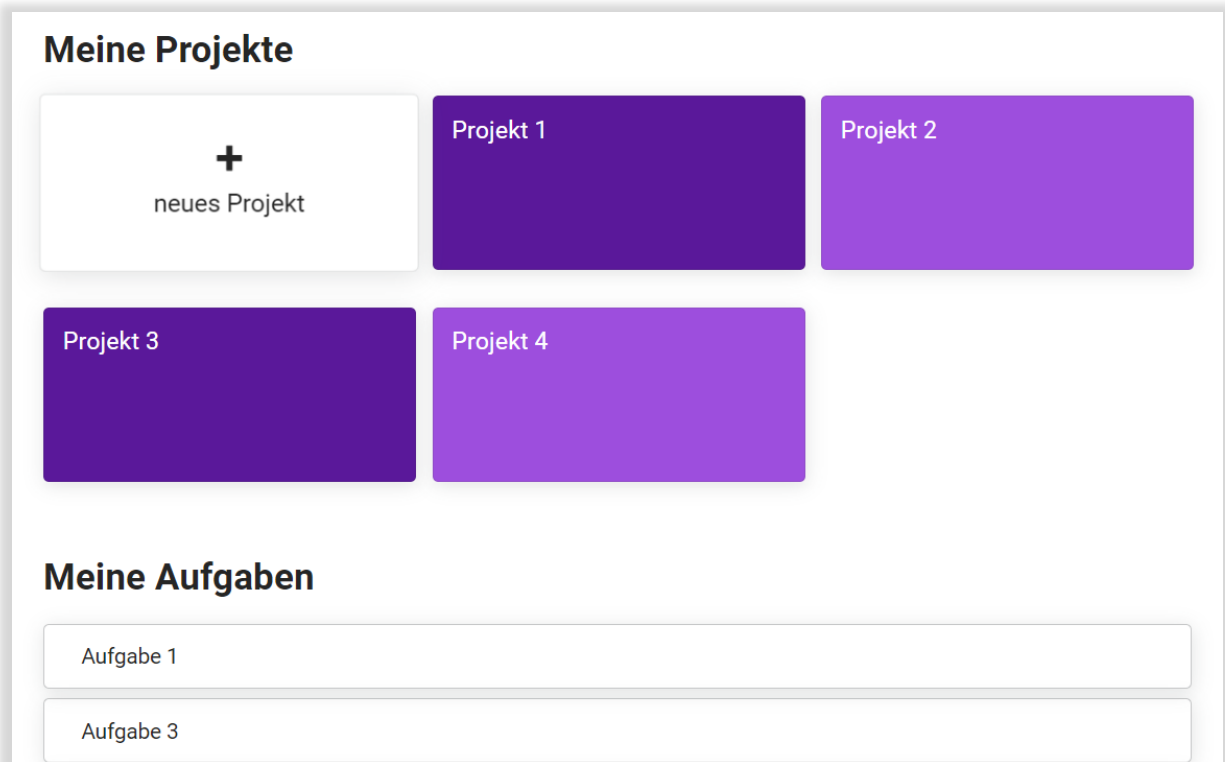


Abbildung 8: Hauptseite

Wenn auf der Hauptseite ein Projekt ausgewählt wird, landet der Nutzer auf der entsprechenden Projektseite. (vgl. Abbildung 19) Die Projektseite ist unterteilt in die Reiter Aufgaben, Einstellungen, Dateien und Nachrichten.

Im Reiter Aufgaben können Aufgaben erstellt werden. Den Aufgaben kann eine Beschreibung hinzugefügt werden und Aufgaben können Projektmitglieder zugewiesen werden. Ein Projekt kann sich im Zustand offen („OPEN“), in Bearbeitung („INWORK“) oder abgeschlossen („COMPLETED“) befinden.

Im Reiter „Einstellungen“ können über das Eintragen einer registrierten Nutzer-E-Mail, Mitglieder zum Projekt hinzugefügt werden.

Dateien können unter dem Reiter „Dateien“ hochgeladen werden. Nach dem Hochladen der Datei steht dies im Reiter zum Download verfügbar.

Im Nachrichten-Reiter können Nutzer Nachrichten erstellen. Die Nachrichten werden in der Datenbank-Tabelle „Messages“ gespeichert. Bei der Auflistung der Nachrichten im Reiter werden die Nachrichten nach Aktualität geordnet. Die neuste Nachricht wird an oberste Stelle aufgelistet.

Projekt 1

AufgabenEinstellungenDateienNachrichten

Aufgabe erstellen

Offenstehende Aufgaben

Aufgabe 1

Aufgabe 4

Aufgaben in Bearbeitung

Aufgabe 3

Abgeschlossene Aufgaben

Aufgabe 2

Abbildung 9: Projektseite

Letztendlich wurden alle Mindestfunktionalitäten umgesetzt. Von den zusätzlichen Funktionalitäten konnte der File-Uploader verwirklicht werden. (vgl. 4.1)

5 Fazit

Ktor erwies sich für den Anwendungsfall als durchaus gut geeignet. Zur Umsetzung einer Webanwendung bietet Ktor alle benötigten Funktionen wie Templating, Seiten-Authentifizierung und Stylesheet bzw. JavaScript Einbindung an. Eine Datenbankverbindung ist ohne Probleme möglich über ein gewöhnliches Datasource-Objekt oder in diesem Fall mit einem ORM-Framework.

Die Struktur ist, nach dem man mit der Kotlin-Syntax vertraut ist, sehr logisch gestaltet. Ebenfalls positiv fällt die Performanz der Anwendung auf. Selbst bei einem standardmäßigen Rechner wird die Webumgebung innerhalb von wenigen Sekunden vollständig gestartet.

Leider ist die Community sehr durchwachsen und es ist schwer auf externen Seiten (Youtube, Stackoverflow, etc.) Anleitungen für das Framework zu finden. Die kleine Community wird jedoch durch eine sehr hervorragende Online-Dokumentation der Entwickler selbst ausgeglichen. Die Dokumentation bietet, von grundsätzlichen Erklärungen des Frameworks, über Code-Schnipsel für bestimmte Anwendungsfälle bis hin zu vollständigen Beispiel-Anwendungen alles, was für die Erstellung eines eigenen Backend-Systems benötigt wird. In den wenigsten Fällen ist es tatsächlich nötig außerhalb der Dokumentation nach Informationen zu suchen und falls doch weitere Fragen auftreten sollten, gibt es die Möglichkeit, diese in der Ktor-Slack-Gruppe zu stellen oder direkt im GitHub-Repository ein GitHub-Issue zu erstellen.

Ktor eignet sich hervorragend für Backends, welche flexibel und skalierbar sein sollen denn dort liegen die Stärken des Frameworks. Ebenfalls glänzt Ktor mit einer in meinem Fall sehr positiv wirkenden Entwicklungsatmosphäre. Selten stieß ich auf Sackgassen bei der Entwicklung meines Anwendungsbeispiels.

Hier ist eine Liste für Situationen und Personen, für welche sich das Framework meiner Meinung nach eignet. Das Framework empfiehlt sich für Entwickler welche bereits erste Erfahrungen mit Kotlin gesammelt haben und mit der Programmiersprache ein Backend-System verwirklichen wollen. Es empfiehlt sich für Backends, dessen Anforderungen noch nicht feststehen und ein Backend welches performant und erweiterbar sein soll. Es empfiehlt sich für Microservices bzw. größere Verbünde aus verschiedenen Systemen. Das Framework empfiehlt sich für Entwickler, die sich von anderen Frameworks, wie z. B. Spring distanzieren wollen aber trotzdem ein Framework wollen, das von einem renovierten Entwicklungs-Unternehmen (JetBrains) unterstützt und abgesichert ist.

Insgesamt bin ich mit dem Framework sehr zufrieden und empfehle es gerne weiter an andere Entwickler.

Literaturverzeichnis

alexebuerkle / Stephan Augsten (2019): Was ist Kotlin? Definition „Kotlin (Programmiersprache)“. Dev-Insider. Online verfügbar unter <https://www.dev-insider.de/was-ist-kotlin-a-843723/>, zuletzt geprüft am 09.07.2020.

Apache: What is Apache FreeMarker™? Online verfügbar unter <https://freemarker.apache.org/>, zuletzt geprüft am 14.07.2020.

Arias, Dan (2018): Adding Salt to Hashing. A Better Way to Store Passwords. Online verfügbar unter <https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/>, zuletzt geprüft am 02.08.2020.

brettwooldridge (2013): HikariCP. Online verfügbar unter <https://github.com/brettwooldridge/HikariCP>, zuletzt geprüft am 02.08.2020.

D., Steffen (2018): Model-View-Controller (MVC). Online verfügbar unter <https://www.methodpark.de/blog/model-view-controller-mvc/>, zuletzt geprüft am 02.08.2020.

Digital Guide IONOS (2020a): Microservice-Architectures: Mehr als die Summe ihrer Teile? Online verfügbar unter <https://www.ionos.de/digitalguide/websites/web-entwicklung/microservice-architecture-so-funktionieren-microservices/>, zuletzt geprüft am 23.07.20.

Digital Guide IONOS (2020b): Was ist WebSocket? Online verfügbar unter <https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-websocket/>, zuletzt geprüft am 15.07.2020.

Gradle (2009): Gradle Build Tool. Online verfügbar unter <https://gradle.org/>, zuletzt geprüft am 02.08.2020.

Jetbrains: Generate a Ktor project. Online verfügbar unter <https://ktor.io/quickstart/generator.html#>, zuletzt geprüft am 02.08.2020.

Jetbrains: JetBrains. Online verfügbar unter <https://www.jetbrains.com/>, zuletzt geprüft am 09.07.2020.

Jetbrains: Ktor. Easy to use, fun and asynchronous. Online verfügbar unter <https://ktor.io/>, zuletzt geprüft am 09.07.20.

Jetbrains (2016): Exposed. Online verfügbar unter <https://github.com/JetBrains/Exposed>, zuletzt geprüft am 01.08.2020.

Jetbrains (2017): kotlinx.html. Online verfügbar unter <https://github.com/Kotlin/kotlinx.html>, zuletzt geprüft am 14.07.2020.

JetBrainsTV (2018): KotlinConf 2018 - Building Server Backends with Ktor by Ryan Harter. JetBrains. Online verfügbar unter <https://www.youtube.com/watch?v=V4PS3IjIzlw>, zuletzt geprüft am 14.07.2020.

JetBrainsTV (2019a): KotlinConf 2019: Ktor for Mobile Developers: Fear the server no more! by Dan Kim. JetBrains. Online verfügbar unter <https://www.youtube.com/watch?v=SOPEc8JnFI4&t=1268s>, zuletzt geprüft am 14.07.2020.

JetBrainsTV (2019b): Server-Side Development with Ktor by Hadi Hariri - Bengaluru, June 22, 2019. JetBrains. Online verfügbar unter https://www.youtube.com/watch?v=Y4kyTpi_qO4.

Killerphp (2009): What are ORM Frameworks? Online verfügbar unter <https://www.killerphp.com/articles/what-are-orm-frameworks/>, zuletzt geprüft am 02.08.2020.

MDN web doc: Accept. Online verfügbar unter <https://developer.mozilla.org/de/docs/Web/HTTP/Headers/Accept>, zuletzt geprüft am 14.07.2020.

Medium (2022): Creating microservice using Kotlin Ktor. Online verfügbar unter <https://medium.com/@abangkis/creating-microservice-using-kotlin-ktor-682923b09d6>, zuletzt geprüft am 28.07.2020.

Möller, Lovis (2018): Ktor. Es muss nicht immer Spring sein. Online verfügbar unter [https://blog.codecentric.de/2018/11/ktor-es-muss-nicht-immer-spring-sein/#:~:text=Ktor%20ist%20ein%20Kotlin%20Framework,Webservern%20\(und%20%Dclients\).&text=Anders%20als%20Spring%20ist%20Ktor,viel%20effektiver%20nutzen%20als%20Spring.](https://blog.codecentric.de/2018/11/ktor-es-muss-nicht-immer-spring-sein/#:~:text=Ktor%20ist%20ein%20Kotlin%20Framework,Webservern%20(und%20%Dclients).&text=Anders%20als%20Spring%20ist%20Ktor,viel%20effektiver%20nutzen%20als%20Spring.), zuletzt geprüft am 03.08.2020.

PostgreSQL (1996): PostgreSQL. The World's Most Advanced Open Source Relational Database. Online verfügbar unter <https://www.postgresql.org/>, zuletzt geprüft am 01.08.2020.

Sass (2006): Sass. Online verfügbar unter <https://sass-lang.com/>, zuletzt geprüft am 01.08.20.

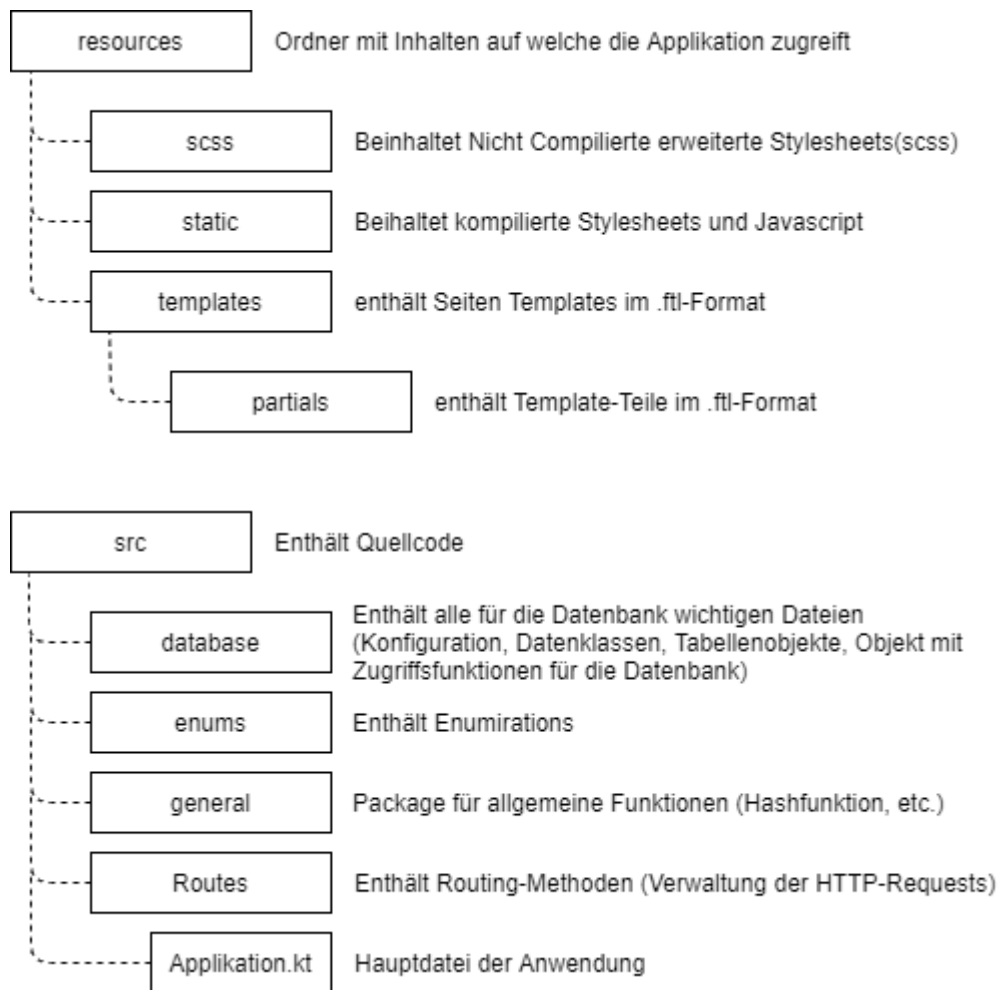
Schleinker Thorsten (2018): Kotlin Kompakt. Frankfurt am Main.

Anhang

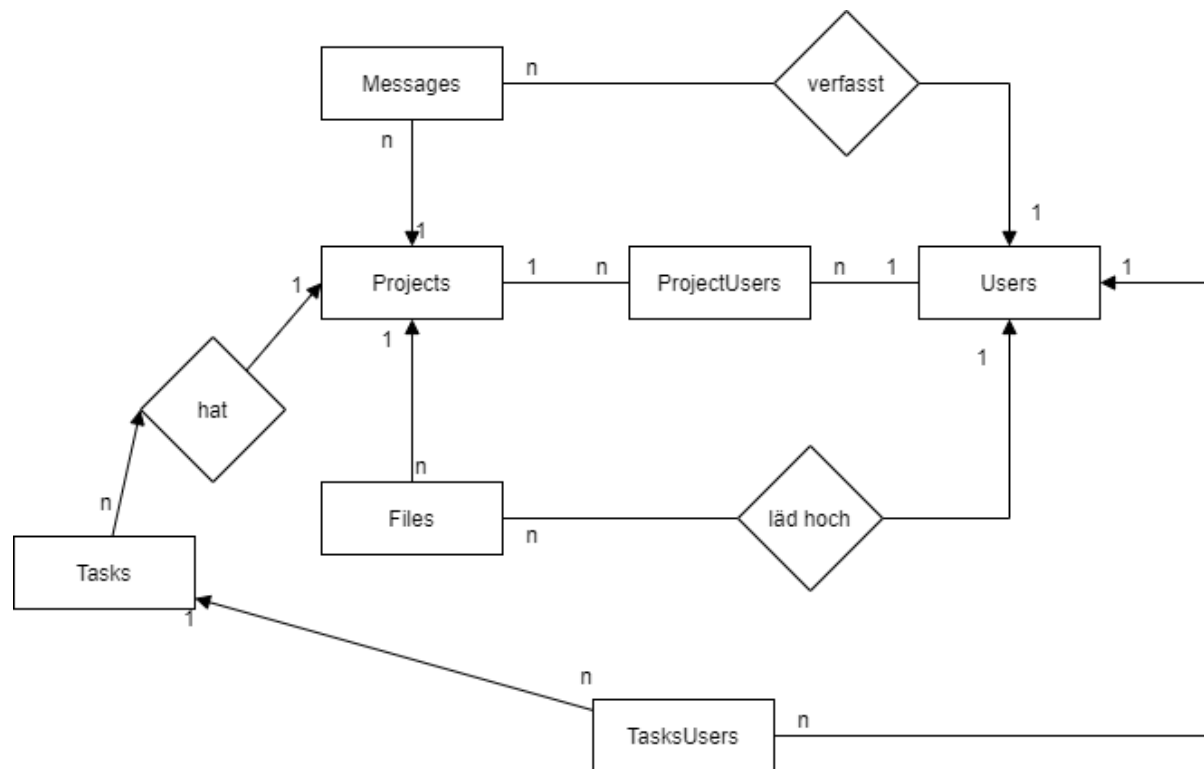
Anhang 1: Abbildung der Projektstruktur29

Anhang 2: vereinfachtes ER-Diagramm der Datenbank30

Anhang 1: Abbildung der Projektstruktur



Anhang 2: vereinfachtes ER-Diagramm der Datenbank



Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.



Furtwangen, der 03.08.2020

Unterschrift