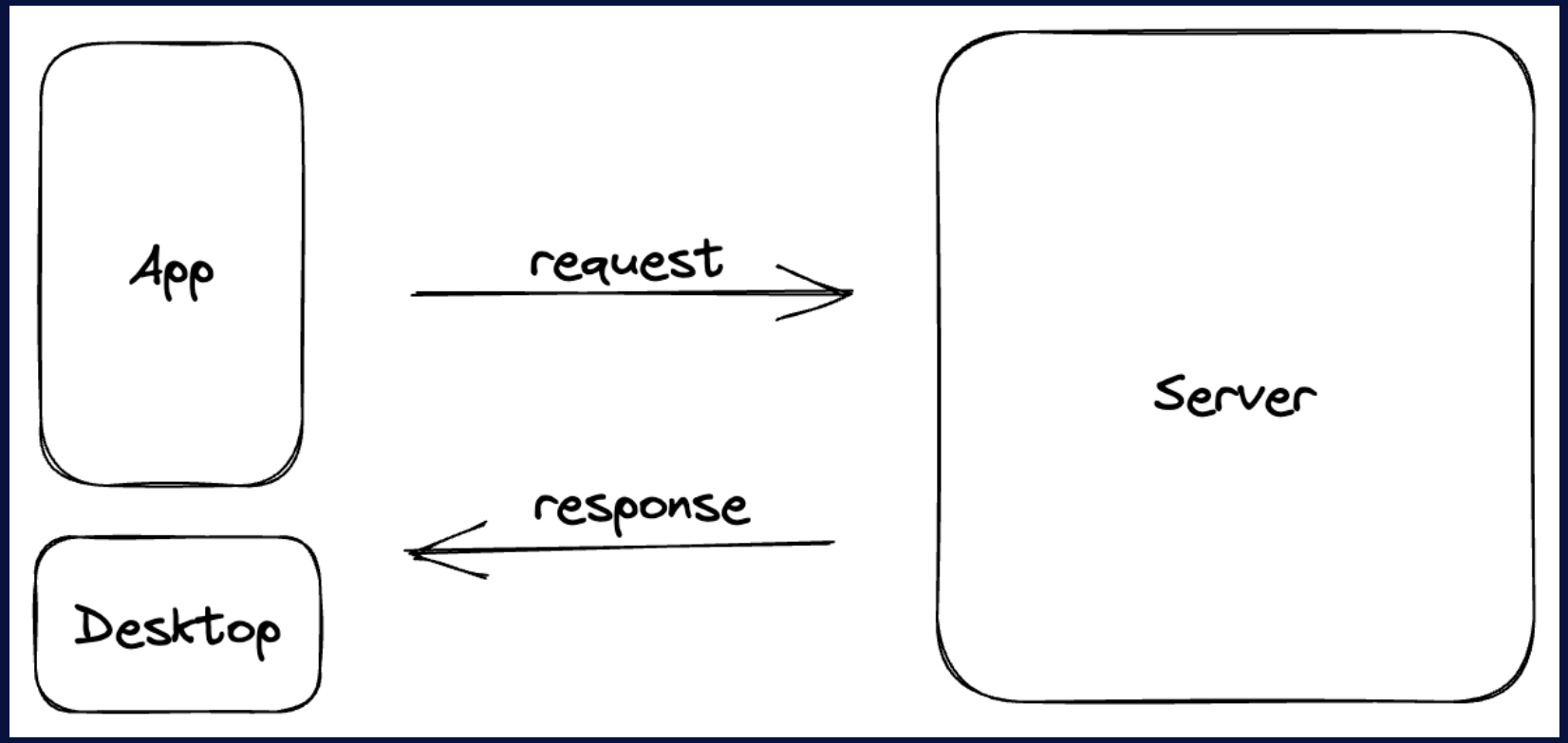
A FRIENDLY INTRODUCTION TO KTOR SERVER

What is Ktor?

- Framework for micro services, web applications, and HTTP services
- Open Source, by JetBrains
- Based on Kotlin Coroutines (asynchronous)


App
request
Server
response
Desktop

Self Contained Package

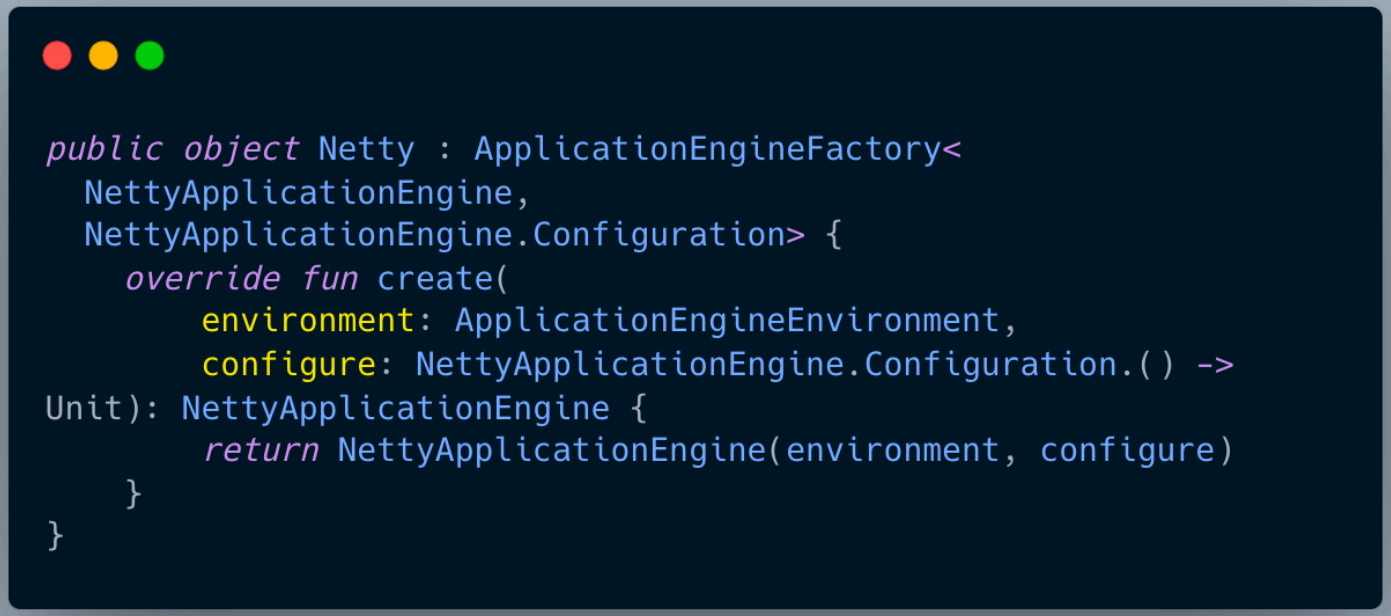
- Application controls the engine settings, connection, and SSL options

Servlet

- Servlet container controls the application lifecycle and connection settings

Your First Server

```
fun main() {
  embeddedServer(
    factory = Netty,
    port = 8080,
    host = "0.0.0.0",
    module = Application::module
  )
  .start(wait = true)
}
```


public object Netty : ApplicationEngineFactory<
NettyApplicationEngine,
NettyApplicationEngine.Configuration> {
override fun create(
environment: ApplicationEngineEnvironment,
configure: NettyApplicationEngine.Configuration.() ->
Unit): NettyApplicationEngine {
return NettyApplicationEngine(environment, configure)
}
}

Application Engine Environment

- Connectors that describe where and how server should listen.
- The running application
- start/stop functions

Application Engine Configuration

- Current parallelism level (e.g. the number of available processors)
- Threads used for new connections, processing connections, parsing messages

Modules

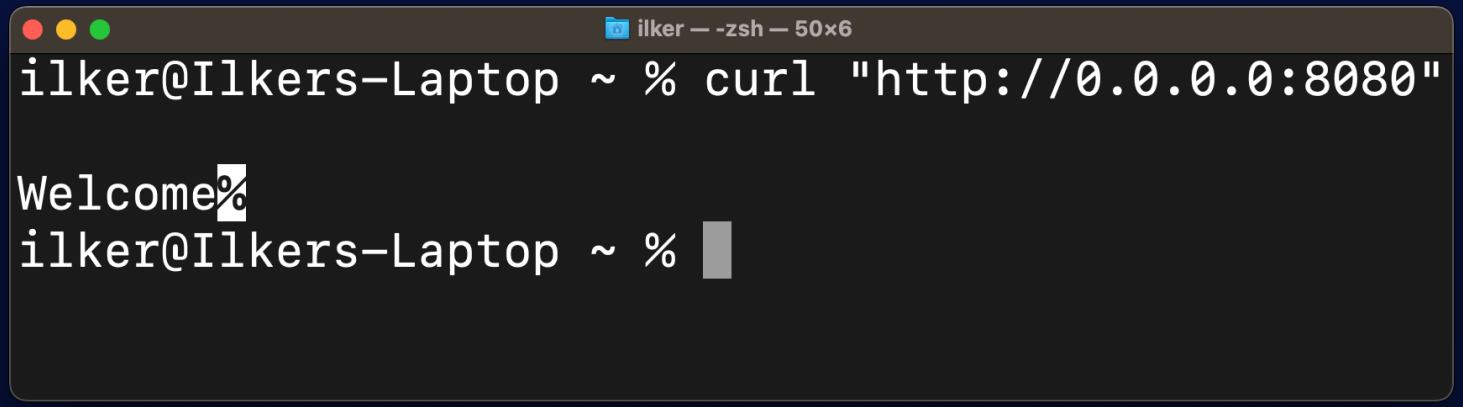
```
public actual val modules:
MutableList<Application.() -> Unit> =
mutableListOf()
```

Extension functions of an Application to structure the plugins

```
fun Application.module() {
    install(Resources)
    install(StatusPages)
    install(RequestValidation)
    configureRouting()
    configureSerialization()
}
```

```
fun Application.configureRouting() {
    routing {
        indexRoute()
        beerRoutes()
    }
}
```

```
fun Route.indexRoute() {
    route("/") {
        get {
            call.respondText(
                text = "Welcome",
                status =
HttpStatusCode.OK
            )
        }
    }
}
```


ilker — -zsh — 50×6
ilker@Ilkers-Laptop ~ % curl "http://0.0.0.0:8080"
Welcome%
ilker@Ilkers-Laptop ~ %

Routing

```
fun Route.beerRoutes() {
    route("/beer") {
        get {...}
        get("{name?}") {...}
        get("{id?}") {...}
        post {...}
        put("{id?}") {...}
        delete("{id?}") {...}
    }
}
```

Type Safe Routing

```
fun Route.beerRoutes() {
    get<resources.Beer> {...}
    get<resources.Beer.Id> {...}
    post<resources.Beer> {...}
    put<resources.Beer.Id> {...}
    delete<resources.Beer.Id> {...}
}
```

Resources

```
@Resource("/beer")
class Beer(
    val name: String? = null
) {
    @Resource("{id}")
    class Id(val parent: Beer =
Beer(), val id: Long)
}
```

Resources

- To define a route handler for a typed resource, pass a resource class to a verb function (get, post, put...).
- Serializable by default

Build Links from Resources

```
val link = href(Beer.Id(id = 1))
```

Get Beers

```
curl --header "Content-Type:
application/json" \
"http://0.0.0.0:8080/beer"
```


{
"data":null,
"errors":[
{"text":"No beers found!"}
]
}


get<resources.Beer> { resource ->
resource.name?.let {
beerStorage
.filter { it.name == resource.name }
.takeIf { it.isNotEmpty() }
?.let {
call.response.status(HttpStatusCode.OK)
call.respond(Response(data = it))
}
}
}

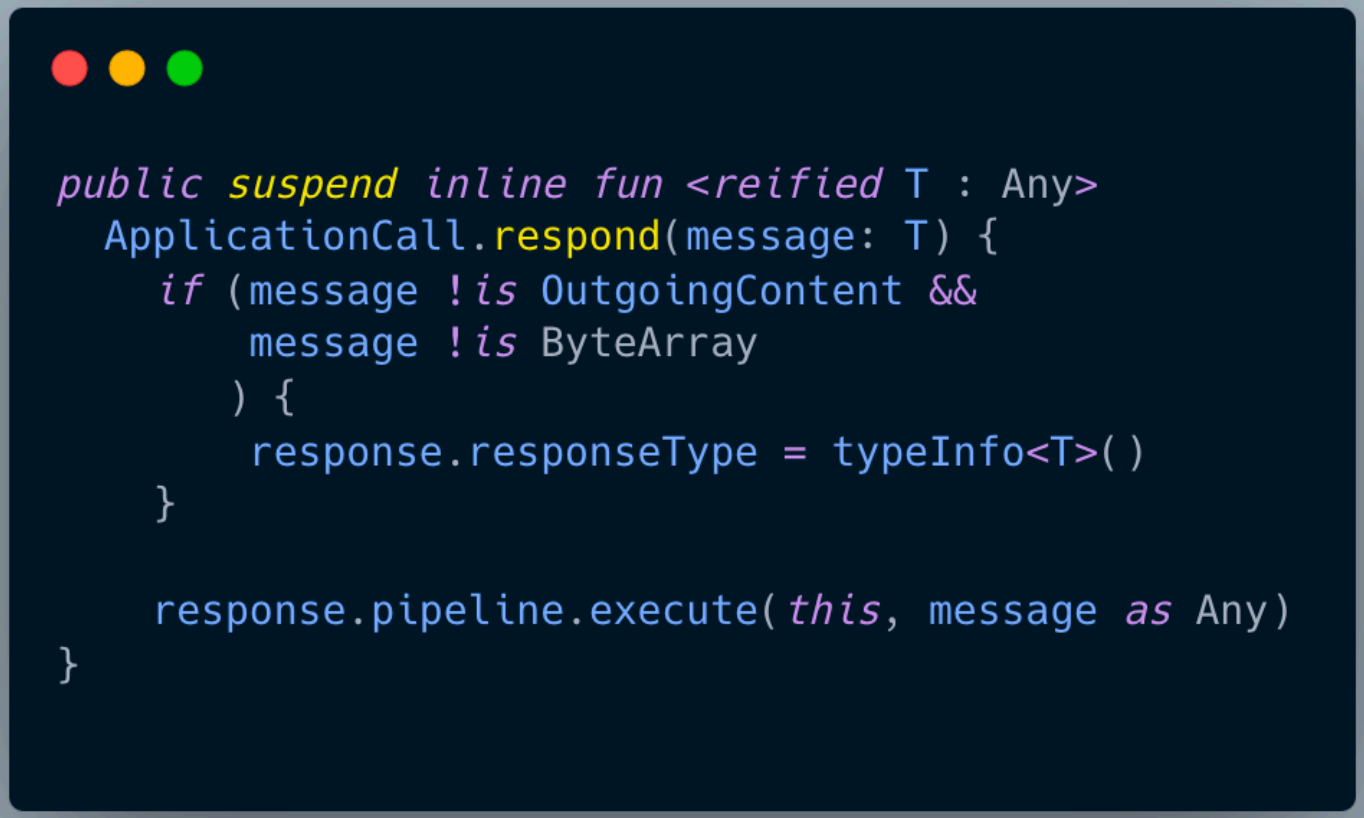

get<resources.Beer> { resource ->
resource.name?.let {
beerStorage
.filter { it.name == resource.name }
.takeIf { it.isNotEmpty() }
?.let {
call.response.status(HttpStatusCode.OK)
call.respond(Response(data = it))
}
}
}


get<resources.Beer> { resource ->
resource.name?.let {
beerStorage
.filter { it.name == resource.name }
.takeIf { it.isNotEmpty() }
?.let {
call.response.status(HttpStatusCode.OK)
call.respond(Response(data = it))
}
}
}


get<resources.Beer> { resource ->
resource.name?.let {...}
?: kotlin.run {
call.response.status(
HttpStatusCode.NotFound
)
call.respond(
Response<Beer>(
errors = listOf(
Error(text = "No beers found!")
)
)
)
}
}


get<resources.Beer> { resource ->
resource.name?.let {...}
?: kotlin.run {
call.response.status(
HttpStatusCode.NotFound
)
call.respond(
Response<Beer>(
errors = listOf(
Error(text = "No beers found!")
)
)
)
}
}


get<resources.Beer> { resource ->
resource.name?.let {...}
?: kotlin.run {
call.response.status(
HttpStatusCode.NotFound
)
call.respond(
Response<Beer>(
errors = listOf(
Error(text = "No beers found!")
)
)
)
}
}



```
public suspend inline fun <reified T : Any>
  ApplicationCall.respond(message: T) {
    if (message !is OutgoingContent &&
        message !is ByteArray
       ) {
        response.responseType = typeInfo<T>()
    }

    response.pipeline.execute(this, message as Any)
}
```


public suspend inline fun <reified T : Any>
ApplicationCall.respond(message: T) {
if (message !is OutgoingContent &&
message !is ByteArray
) {
response.responseType = typeInfo<T>()
}
response.pipeline.execute(this, message as Any)
}

Response Model

```
@Serializable
data class Response<T>(
    val data: T? = null,
    val errors: List<Error>? = null
)
```

Create a Beer

```
curl --header "Content-Type:
application/json" \
--data "{"name": "Bock"}" \
"http://0.0.0.0:8080/beer"
```


{
"data":{
"id":5324519225709165061,
"name":"Bock"
},
"errors":null
}


post<resources.Beer> {
val beer = call.receive<BeerToCreate>()
val id = Random.nextLong()
beerStorage.add(
Beer(id = id, name = beer.name)
)
beerStorage.find { it.id == id }?.let {
call.response.status(HttpStatusCode.Created)
call.respond(Response(data = it))
}
}


post<resources.Beer> {
val beer = call.receive<BeerToCreate>()
val id = Random.nextLong()
beerStorage.add(
Beer(id = id, name = beer.name)
)
beerStorage.find { it.id == id }?.let {
call.response.status(HttpStatusCode.Created)
call.respond(Response(data = it))
}
}

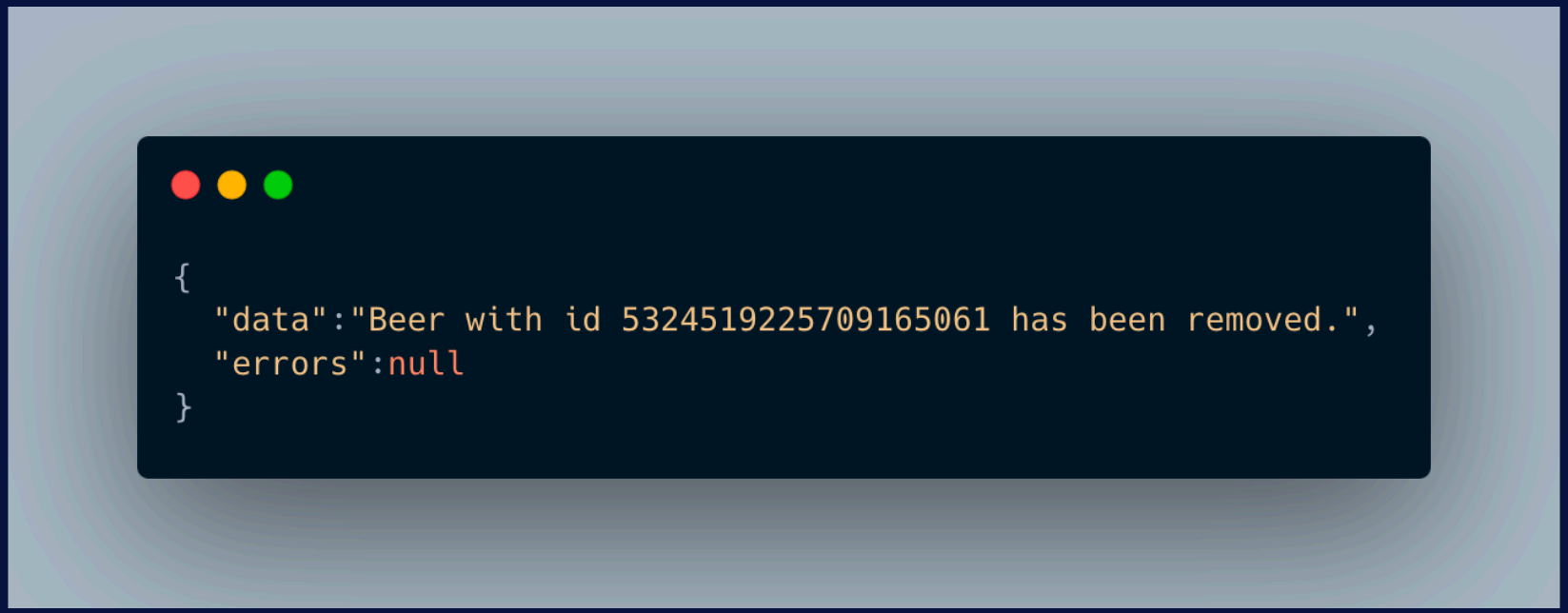

post<resources.Beer> {
val beer = call.receive<BeerToCreate>()
val id = Random.nextLong()
beerStorage.add(
Beer(id = id, name = beer.name)
)
beerStorage.find { it.id == id }?.let {
call.response.status(HttpStatusCode.Created)
call.respond(Response(data = it))
}
}


post<resources.Beer> {
val beer = call.receive<BeerToCreate>()
val id = Random.nextLong()
beerStorage.add(
Beer(id = id, name = beer.name)
)
beerStorage.find { it.id == id }?.let {
call.response.status(HttpStatusCode.Created)
call.respond(Response(data = it))
}
}


public suspend inline fun <reified T : Any>
ApplicationCall.receive(): T =
receiveNullable(typeInfo<T>())
?: throw
CannotTransformContentToTypeException(
typeInfo<T>().kotlinType!!
)

Delete a Beer

```
curl --header \
"Content-Type: application/json" \
--request DELETE \
"http://0.0.0.0:8080/beer/532..."
```


{
"data":"Beer with id 5324519225709165061 has been removed.",
"errors":null
}


delete<resources.Beer.Id> { resource ->
val id = resource.id
if (beerStorage.removeIf { it.id == id }) {
call.response.status(HttpStatusCode.Accepted)
call.respond(
Response(
data = "Beer with id $id has been removed."
)
)
} else {
// Error
}
}


delete<resources.Beer.Id> { resource ->
val id = resource.id
if (beerStorage.removeIf { it.id == id }) {
call.response.status(HttpStatusCode.Accepted)
call.respond(
Response(
data = "Beer with id $id has been removed."
)
)
} else {
// Error
}
}


delete<resources.Beer.Id> { resource ->
val id = resource.id
if (beerStorage.removeIf { it.id == id }) {
call.response.status(HttpStatusCode.Accepted)
call.respond(
Response(
data = "Beer with id $id has been removed."
)
)
} else {
// Error
}
}

SSL and Certificates

- Ktor uses Java KeyStore (JKS) as a storage facility for certificates.
- Self-signed certificates for testing purposes by calling `buildKeyStore()`.


fun main() {
val keyStoreFile = File("build/keystore.jks")
val keyStore = buildKeyStore {
certificate("alias") {
password = "password"
domains = listOf(
"127.0.0.1",
"0.0.0.0",
"localhost"
)
}
}
keyStore.saveToFile(keyStoreFile, "password")
}


fun main() {
val keyStoreFile = File("build/keystore.jks")
val keyStore = buildKeyStore {
certificate("alias") {
password = "password"
domains = listOf(
"127.0.0.1",
"0.0.0.0",
"localhost"
)
}
}
keyStore.saveToFile(keyStoreFile, "password")
}


fun main() {
val keyStoreFile = File("build/keystore.jks")
val keyStore = buildKeyStore {
certificate("alias") {
password = "password"
domains = listOf(
"127.0.0.1",
"0.0.0.0",
"localhost"
)
}
}
keyStore.saveToFile(keyStoreFile, "password")
}


fun main() {
...
val environment = applicationEngineEnvironment {
sslConnector(
keyStore = keyStore,
keyAlias = "alias",
keyStorePassword = { "password".toCharArray() },
privateKeyPassword = { "password".toCharArray() }
) {
port = 8443
keyStorePath = keyStoreFile
}
module(Application::module)
}
}


fun main() {
...
val environment = applicationEngineEnvironment {
sslConnector(
keyStore = keyStore,
keyAlias = "alias",
keyStorePassword = { "password".toCharArray() },
privateKeyPassword = { "password".toCharArray() }
) {
port = 8443
keyStorePath = keyStoreFile
}
module(Application::module)
}
}

```
fun main() {
    ...

    embeddedServer(
        factory = Netty,
        environment = environment
    ).start(wait = true)
}
```

Send HTML in Response


fun Application.module() {
routing {
get("/beer") {
val beers = listOf("Bock", "Glitch")
call.respondHtml(HttpStatusCode.OK) {
head {
title { +"Beers" }
}
body {
h1 { +${beers.joinToString(",")} }
}
}
}
}
}


fun Application.module() {
routing {
get("/beer") {
val beers = listOf("Bock", "Glitch")
call.respondHtml(HttpStatusCode.OK) {
head {
title { +"Beers" }
}
body {
h1 { +${beers.joinToString(",")} }
}
}
}
}
}


fun Application.module() {
routing {
get("/beer") {
val beers = listOf("Bock", "Glitch")
call.respondHtml(HttpStatusCode.OK) {
head {
title { +"Beers" }
}
body {
h1 { +${beers.joinToString(",")} }
}
}
}
}
}

Respond with HTML Form


get("/beer") {
call.respondHtml {
body {
form(
action = "/beer",
encType = FormEncType.applicationXWwwFormUrlEncoded,
method = FormMethod.post
) {
p {
+"Beer Name:"
textInput(name = "beer name")
}
p {
submitInput() { value = "Create Beer" }
}
}
}
}

Templates

- To respond, call the `respondHtmlTemplate()`.
- To create a template, implement the `Template` interface.
- Inside templates, use `Placeholder` or `TemplatePlaceholder`.

Conclusions


ALWAYS HAS BEEN
KOTLIN IS CAPABLE
OF EVERYTHING
imgflip.com


USE KOTLIN
ONLY FOR BACKEND
USE KOTLIN
BOTH FOR
FRONTEND AND BACKEND
USE KOTLIN
FOR ANDROID,
IOS, FRONTEND,
BACKEND, DESKTOP
imgflip.com

The Repository

Where To Stalk Me

https://androiddev.social/@ilker
https://github.com/ilkeraslan
https://www.polywork.com/ilkeraslan
https://www.linkedin.com/in/aslanilker/
https://blog.ilker.it/