

CREATING WEB APPLICATIONS IN KOTLIN

THE YEAST GIST OF THIS TALK

(MORE ON THAT LATER...)

- Brief introduction to Ktor
- Creation of a Ktor project
- Creation of endpoints (CRUD-ish)
- Pros and cons

WHAT IS KTOR?

- A framework for building asynchronous servers and clients using Kotlin.
- It provides a Domain Specific Language (DSL) for server-side development.
- Can be used on web, iOS and Android.

(used by JetBrains as the backend for their
KotlinConf mobile and web app)

CREATING A KTOR WEB APPLICATION

Tools and requirements:

- IntelliJ IDEA
- Minimum version of Kotlin 1.3
- Application engine to handle requests and responses between client and server → Netty

STRUCTURE OF A KTOR WEB APPLICATION

FEATURES → in charge of providing functionality into the request/response cycle ==> Gradle dependencies

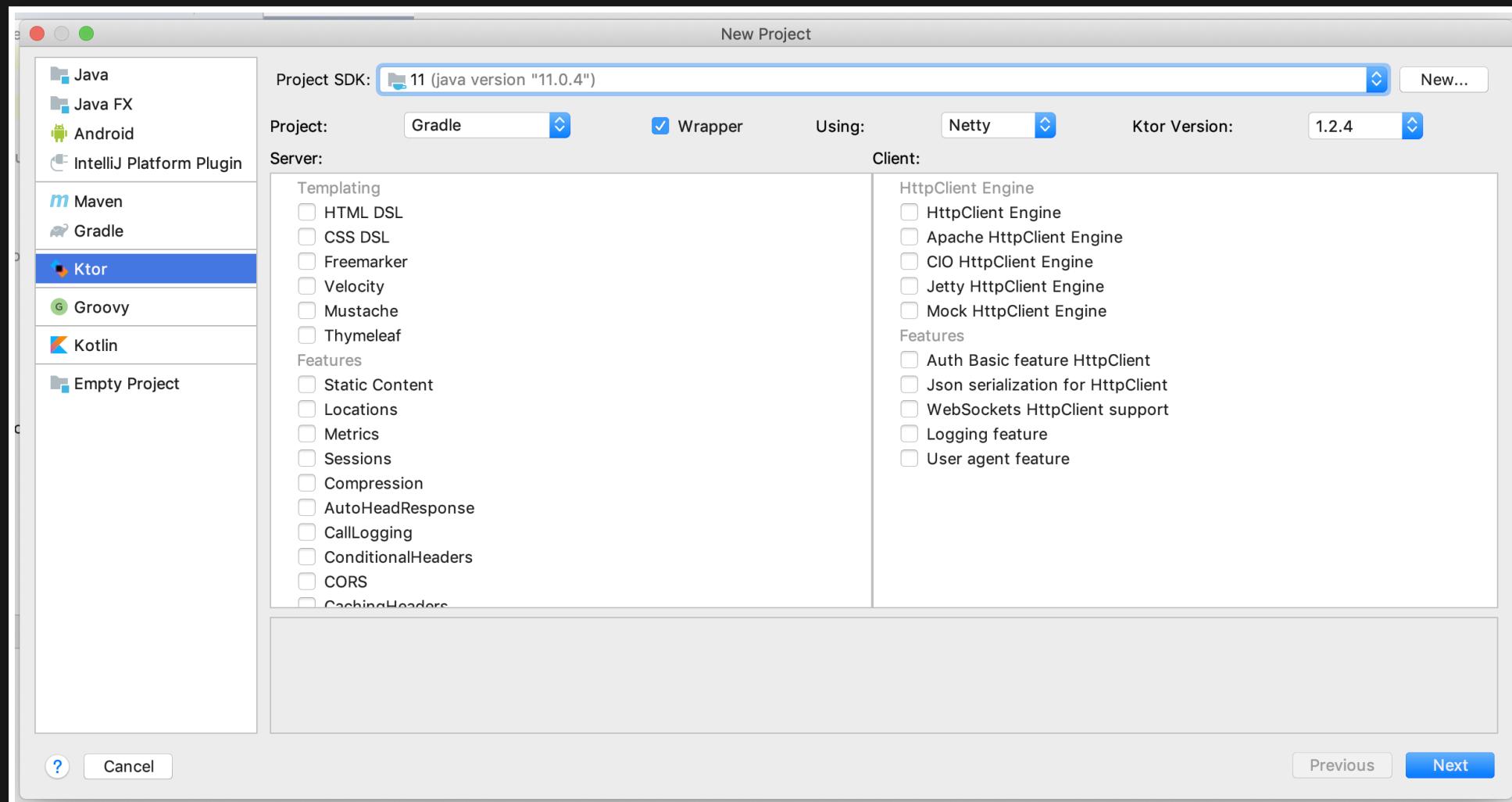
Examples:

Headers for network calls, processing JSON, authentication, routing...

(we could also create our own)

GENERATE A KTOR PROJECT

IntelliJ IDEA plugin



GENERATING A KTOR PROJECT (2ND OPTION)

Site start.ktor.io

Ktor Project Generator (1.3.0)

Configuration

Gradle project with Wrapper

Server Engine: Netty

Ktor 1.3.0

Group: com.example

Name: ktor-demo

Version: 0.0.1-SNAPSHOT

Swagger (Optional)

Build or [Install IntelliJ plugin](#)

Server

Filter Server Features

Templating

- HTML DSL** ([ktor-html-builder](#))
Generate HTML using Kotlin code like a pure-core template engine
[Documentation](#)
- CSS DSL** ([org.jetbrains:kotlin-css-jvm:1.0.0-pre.31-kotlin-1.2.41](#))
Generate CSS using Kotlin code
[Documentation](#)
- Freemarker** ([ktor-freemarker](#))
Serve HTML content using Apache's FreeMarker template engine
[Documentation](#)
- Velocity** ([ktor-velocity](#))
Serve HTML content using Apache's Velocity template engine
[Documentation](#)
- Mustache** ([ktor-mustache](#))
Serve HTML content using Mustache template engine
[Documentation](#)
- Thymeleaf** ([ktor-thymeleaf](#))
Serve HTML content using Thymeleaf template engine
[Documentation](#)

Show marked dependencies only

Client

Filter Client Features

HttpClient Engine

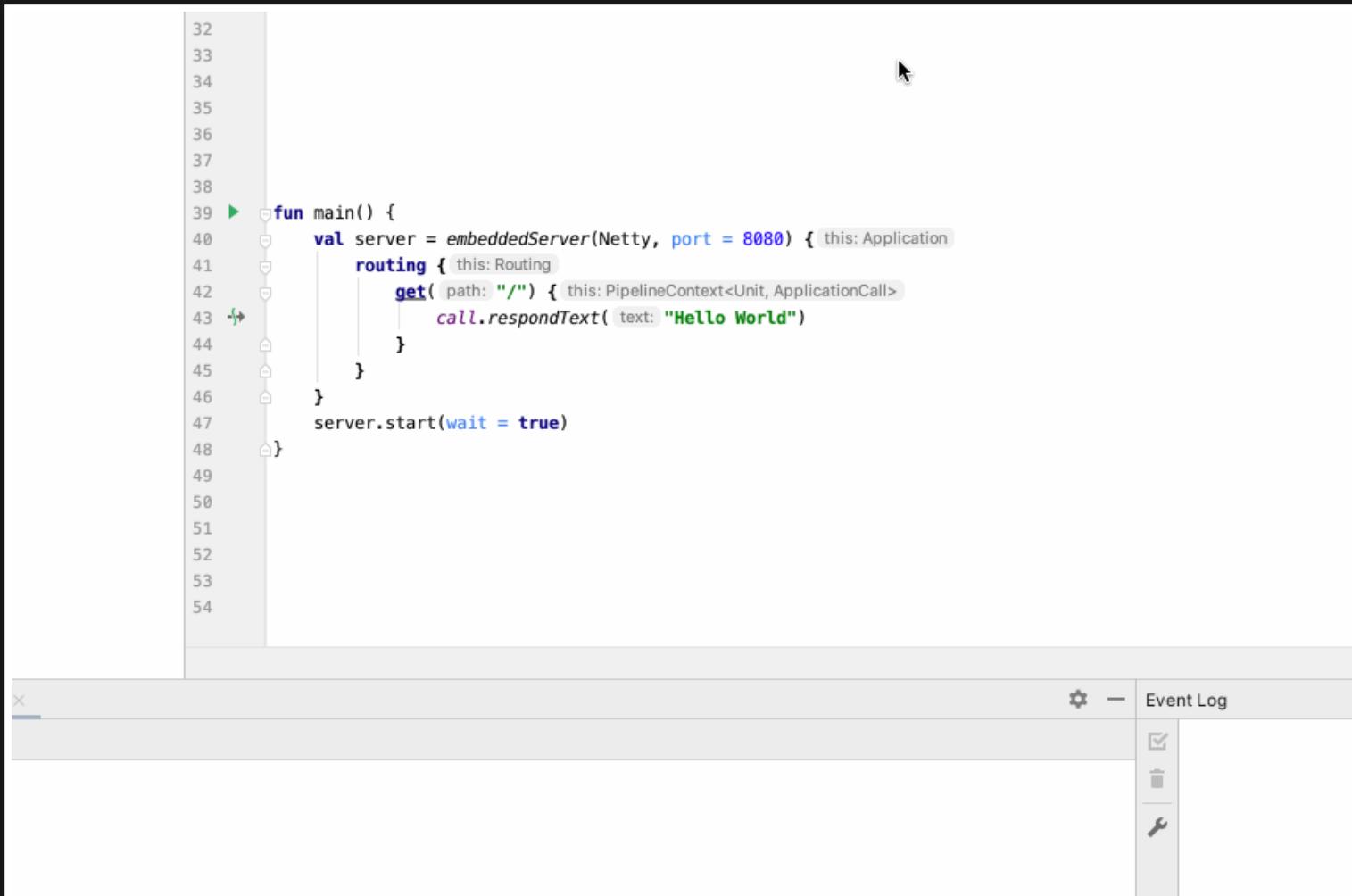
- HttpClient Engine** ([ktor-client-core, ktor-client-core-jvm](#))
Core of the HttpClient. Required for libraries.
[Documentation](#)
- Apache HttpClient Engine** ([ktor-client-apache](#))
Engine for the Ktor HttpClient using Apache. Supports HTTP 1.x and HTTP 2.0.
[Documentation](#)
- CIO HttpClient Engine** ([ktor-client-cio](#))
Engine for the Ktor HttpClient using CIO (Coroutine I/O). Only supports HTTP 1.x.
[Documentation](#)
- Jetty HttpClient Engine** ([ktor-client-jetty](#))
Engine for the Ktor HttpClient using Jetty. Only supports HTTP 2.x.
[Documentation](#)
- Mock HttpClient Engine** ([ktor-client-mock, ktor-client-mock-jvm](#))
Engine for using in tests to simulate HTTP responses programmatically.

Show marked dependencies only

HELLO WORLD

```
1 import io.ktor.application.*
2 import io.ktor.http.*
3 import io.ktor.response.*
4 import io.ktor.routing.*
5 import io.ktor.server.engine.*
6 import io.ktor.server.netty.*
7
8 ➤ fun main(args: Array<String>) {
9     val server = embeddedServer(Netty, port = 8080) { this: Application
10
11         routing { this: Routing
12             get(path: "/") { //DSL function
13                 call.respondText(text: "Welcome home (sanitarium)", ContentType.Text.Plain)
14             }
15         }
16     }
17     server.start(wait = true)
18 }
19 }
```

HOW TO RUN



A screenshot of an IDE interface showing Java code for a Netty server. The code is as follows:

```
32
33
34
35
36
37
38
39 ► fun main() {
40     val server = embeddedServer(Netty, port = 8080) { this: Application
41         routing { this: Routing
42             get( path: "/" ) { this: PipelineContext<Unit, ApplicationCall>
43                 call.respondText( text: "Hello World" )
44             }
45         }
46     }
47     server.start(wait = true)
48 }
```

The code defines a main function that creates an embedded Netty server on port 8080. The server's routing block handles a single endpoint at the root path ('/'). When a request is received at this path, it responds with the text "Hello World". The server is then started.

COMMON FEATURES

- `DefaultHeaders`: adds a default set of headers to every HTTP response.
- `StatusPages`: allows Ktor applications to respond appropriately to any failure state.
- `ContentNegotiation`: tells the server what to use for incoming and outbound requests. A.k.a. serialization/deserialization Kotlin objects <-> JSON.

DefaultHeaders

```
30
31 > fun main() {
32     val server = embeddedServer(Netty, port = 8080) { this: Application
33
34         install(DefaultHeaders) { this: DefaultHeaders.Configuration
35             header(name: "Ktor-Developer", value: "Alicia P") // will send this header with each response
36         }
37
38         routing { this: Routing
39             get(path: "/") { this: PipelineContext<Unit, ApplicationCall>
40                 call.respondText(text: "Hello World")
41             }
42         }
43     }
44     server.start(wait = true)
45 }
46 }
```

StatusPages

```
29
30 ► fun main() {
31     val server = embeddedServer(Netty, port = 8080) { this: Application
32
33         install(StatusPages) { this: StatusPages.Configuration
34             exception<Throwable> { e ->
35                 call.respond(HttpStatusCode.InternalServerError)
36                 // to log the exception we need to throw
37                 throw InternalServerErrorException("There was an error")
38             }
39         }
40
41         routing { this: Routing
42             get(path: "/") { this: PipelineContext<Unit, ApplicationCall>
43                 call.respondText(text: "Hello World")
44             }
45         }
46     }
47     server.start(wait = true)
48 }
49
50 class InternalServerErrorException(message: String): Throwable(message)
51
```

```
"/Applications/IntelliJ IDEA CE.app/Contents/jbr/Contents/Home/bin/java" ...
2019-10-27 16:35:36.805 [main] INFO  ktor.application - No ktor.deployment.watch patterns specified, automatic reload is not active
2019-10-27 16:35:36.850 [main] INFO  ktor.application - Responding at http://0.0.0.0:8080
2019-10-27 16:35:44.753 [nioEventLoopGroup-4-1] ERROR ktor.application - 500 Internal Server Error: GET - /
com.example.InternalServerErrorException: There was an error
    at com.example.ApplicationKt$main$server$1$2$1.invokeSuspend(Application.kt:46)
    at com.example.ApplicationKt$main$server$1$2$1.invoke(Application.kt)
    at io.ktor.util.pipeline.SuspendFunctionGun.loop(PipelineContext.kt:268)
    at io.ktor.util.pipeline.SuspendFunctionGun.access$loop(PipelineContext.kt:67)
```

ContentNegotiation

```
40
41 > fun main() {
42     val server = embeddedServer(Netty, port = 8080) { this: Application
43
44         install(ContentNegotiation) { this: ContentNegotiation.Configuration
45             gson { this: GsonBuilder
46                 // Configure Gson here
47                 setPrettyPrinting()
48             }
49         }
50
51         routing { this: Routing
52             get(path: "/") { this: PipelineContext<Unit, ApplicationCall>
53                 call.respondText(text: "Hello World")
54             }
55         }
56     }
57     server.start(wait = true)
58 }
59
60 }
```

NOW WE'RE READY TO CREATE AN IPA USING KTOR

(maybe a typo, but it really is a beer-related API)



GitHub repository

<https://github.com/aliciaphes/Kbeer>

Beer model

The screenshot shows a file tree on the left and a code editor on the right.

File Tree:

- .gradle
- .idea
- build
- gradle
- resources [main] resources root
- src [main] sources root
 - api
 - model

Code Editor:

```
2
3 package model
4
5 data class Beer(val name: String, val description: String) {
6     var id: Int = 0
7     var abv: Float = 0F // Alcohol by volume
8 }
```

The code editor shows a Java-like class definition for a `Beer` model. The class has three fields: `name`, `description`, and `id`. It also includes a `abv` field with a comment indicating it represents the alcohol by volume.

In-memory repository BeerRepository

```
1 package com.kbeer.repository
2
3 import com.kbeer.model.Beer
4 import java.util.concurrent.atomic.AtomicInteger
5
6
7 class BeerRepository {
8
9     private val idCounter: AtomicInteger = AtomicInteger()
10    private val beers = ArrayList<Beer>()
11
12    fun add(beer: Beer){
13        if (!beers.contains(beer)) {
14            beer.id = idCounter.incrementAndGet()
15            beers.add(beer)
16        }
17    }
18
19    fun beer(id: Int) = beers.find { it.id == id } ?: throw IllegalArgumentException("No beer found for $id.")
20
21    fun beers() = beers
22
23    fun remove(beer: Beer) {
24        if (!beers.contains(beer)) {
25            throw IllegalArgumentException("No beer found for ${beer.id}")
26        }
27        beers.remove(beer)
28    }
29
30    fun remove(id: Int) = beers.remove(beer(id))
```

Beer Route (1/2)

The screenshot shows a file tree on the left and a code editor on the right.

Project Structure:

- .gradle
- .idea
- build
- gradle
- resources [main] resources root
- src [main] sources root
 - api
 - model
 - Beer
 - repository
 - BeerRepository
 - webapp
 - Beers.kt
- Application.kt

test test sources root (highlighted in green)

- .gitignore
- build.gradle
- example2.iml
- gradle.properties
- gradlew
- gradlew.bat (highlighted in grey)
- settings.gradle
- External Libraries
- Scratches and Consoles

Code Editor (Kotlin):

```
2 package webapp
3
4 import io.ktor.application.call
5 import io.ktor.request.receive
6 import io.ktor.response.respond
7 import io.ktor.routing.Route
8 import io.ktor.routing.get
9 import io.ktor.routing.post
10 import model.Beer
11 import repository.BeerRepository
12
13 const val HOME = "/"
14 const val BEER = "/beer"
15
16 fun Route.beer(beerRepository: BeerRepository) {
17
18     get(HOME) { this: PipelineContext<Unit, ApplicationCall>
19         call.respond(beerRepository.beers())
20     }
21
22     post(BEER) { this: PipelineContext<Unit, ApplicationCall>
23         val beer = call.receive<Beer }()
24         beerRepository.add(beer)
25         call.respond(beer)
26     }
27
28 }
29
30 }
```

Beer Route (2/2)

Application.kt

```
▶ fun main(args: Array<String>): Unit = io.ktor.server.netty.EngineMain.main(args)
  fun Application.module() {
    install(ContentNegotiation) { this: ContentNegotiation.Configuration
      gson { this: GsonBuilder
        setPrettyPrinting()
      }
    }

    val beerRepository = BeerRepository()

    routing { this: Routing
      beer(beerRepository)
    }
  }
}
```

Adding a beer:

```
curl \  
--request POST \  
--header "Content-Type:  
application/json" \  
--data '{"name": "Franziskaner",  
"description": "German Hefeweizen",  
"abv": "5.0"}' \  
http://127.0.0.1:8080/beer
```

Adding a beer (in action)

The screenshot shows an IDE interface with a file tree on the left and a code editor on the right.

File Tree:

- repository
- BeerRepository
- webapp
- Beers.kt
- PrettyBeers.kt
- Application.kt
- Sucio.kt
- test test sources root
- .gitignore
- build.gradle
- example2.iml
- gradle.properties
- gradlew
- gradlew.bat
- settings.gradle
- External Libraries
- Scratches and Consoles

Code Editor (Route.kt):

```
12  const val HOME = "/"
13  const val BEER = "/beer"
14
15  fun Route.beer(beerRepository: BeerRepository) {
16      get(HOME) { this: PipelineContext<Unit, ApplicationCall>
17          call.respond(beerRepository.beers())
18      }
19      post(BEER) { this: PipelineContext<Unit, ApplicationCall>
20          val beer = call.receive<Beer }()
21          val added = beerRepository.add(beer)
22          if (added) {
23              call.respond(beer)
24          }
25      }
26  }
27
28  }
29 }
```

Run Tab:

Run: com.example2.ApplicationKt

```
"watch" : [
    # application.conf @ file:/Users/aliciaperez/Desktop/Ktor/example2/build/resources/main/application.conf: 6
    "example2"
]
},
# Content hidden
"security" : "***"
}
```

Output Log:

```
2019-11-03 19:30:01.829 [main] DEBUG Application - Java Home: /Applications/IntelliJ IDEA CE.app/Contents/jbr/Contents
2019-11-03 19:30:01.829 [main] DEBUG Application - Class Loader: jdk.internal.loader.ClassLoaders$AppClassLoader@2c13da15: []
2019-11-03 19:30:01.836 [main] INFO Application - No ktor.deployment.watch patterns match classpath entries, automatic reload is not active
2019-11-03 19:30:02.192 [main] INFO Application - Responding at http://0.0.0.0:8080
```

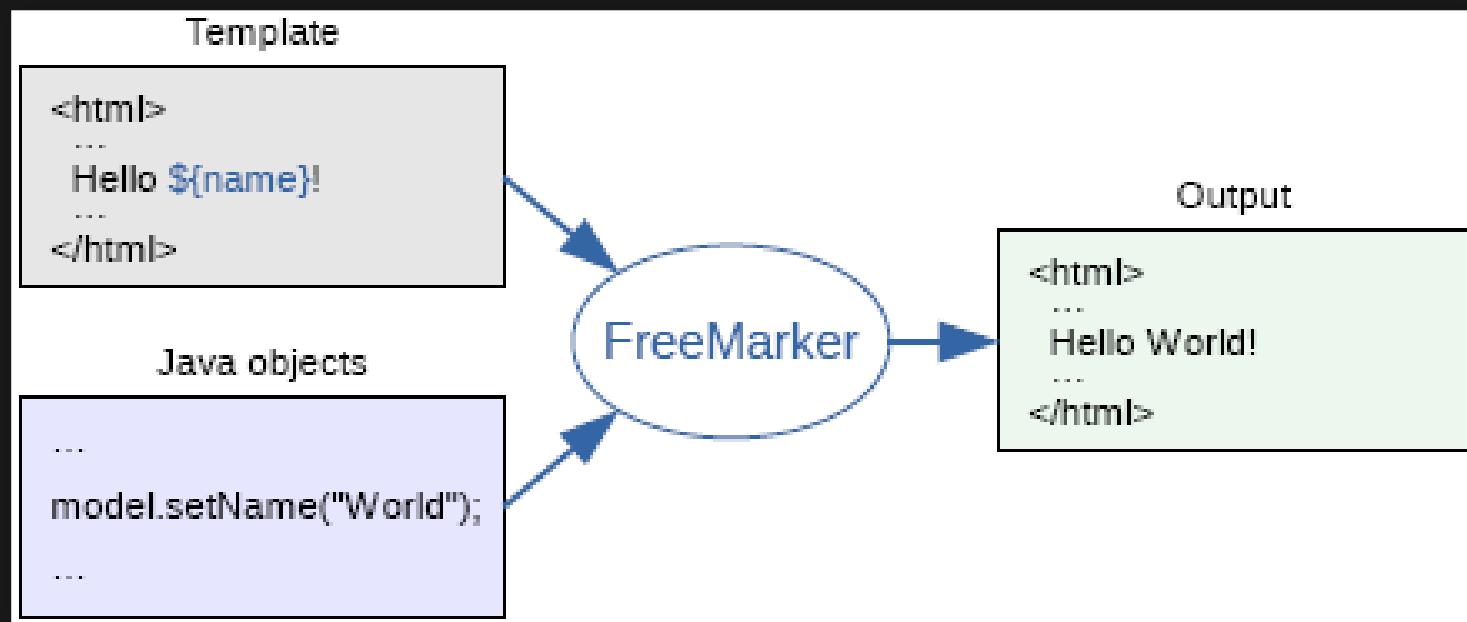
Toolbars and Buttons:

- Run (play button)
- Debug (stop button)
- TODO
- Build

It works, I guess...



Enter the template engine FreeMarker



Install feature and indicate the paths for FreeMarker

The screenshot shows a file browser interface with a tree view of project files on the left and a code editor on the right.

Project Structure:

- resources [main] resources root
 - styles
 - bootstrap.min.css
 - main.css
 - templates
 - beers.ftl
 - application.conf
 - logback.xml
- src [main] sources root
 - api
 - model
 - Beer
 - repository
 - BeerRepository
 - webapp
 - Beers.kt
- Application.kt

Code Editor (Application.kt):

```
15
16  fun main(args: Array<String>): Unit = io.ktor.server.netty.EngineMain.main(args)
17  fun Application.module() {
18
19      install(ContentNegotiation) { this: ContentNegotiation.Configuration
20          gson { this: GsonBuilder
21              setPrettyPrinting()
22          }
23      }
24      install(FreeMarker) { this: Configuration
25          templateLoader = ClassTemplateLoader(Application::class.java.classLoader, basePackagePath: "templates")
26      }
27
28      val beerRepository = BeerRepository()
29
30      routing { this: Routing
31
32          static { this: Route
33              // This serves files from the 'styles' folder in the application resources.
34              resources( resourcePackage: "styles" )
35          }
36
37          beer(beerRepository)
38
39      }
40
41  }
42 }
```

The code in the editor is a Ktor application configuration. It defines a main function and an application module. Inside the module, it installs ContentNegotiation (using Gson for JSON) and FreeMarker. For FreeMarker, it uses a ClassTemplateLoader with the base package path set to "templates". It also defines a static route to serve files from the 'styles' folder in the application resources. Finally, it defines a 'beer' route using the 'beerRepository'.

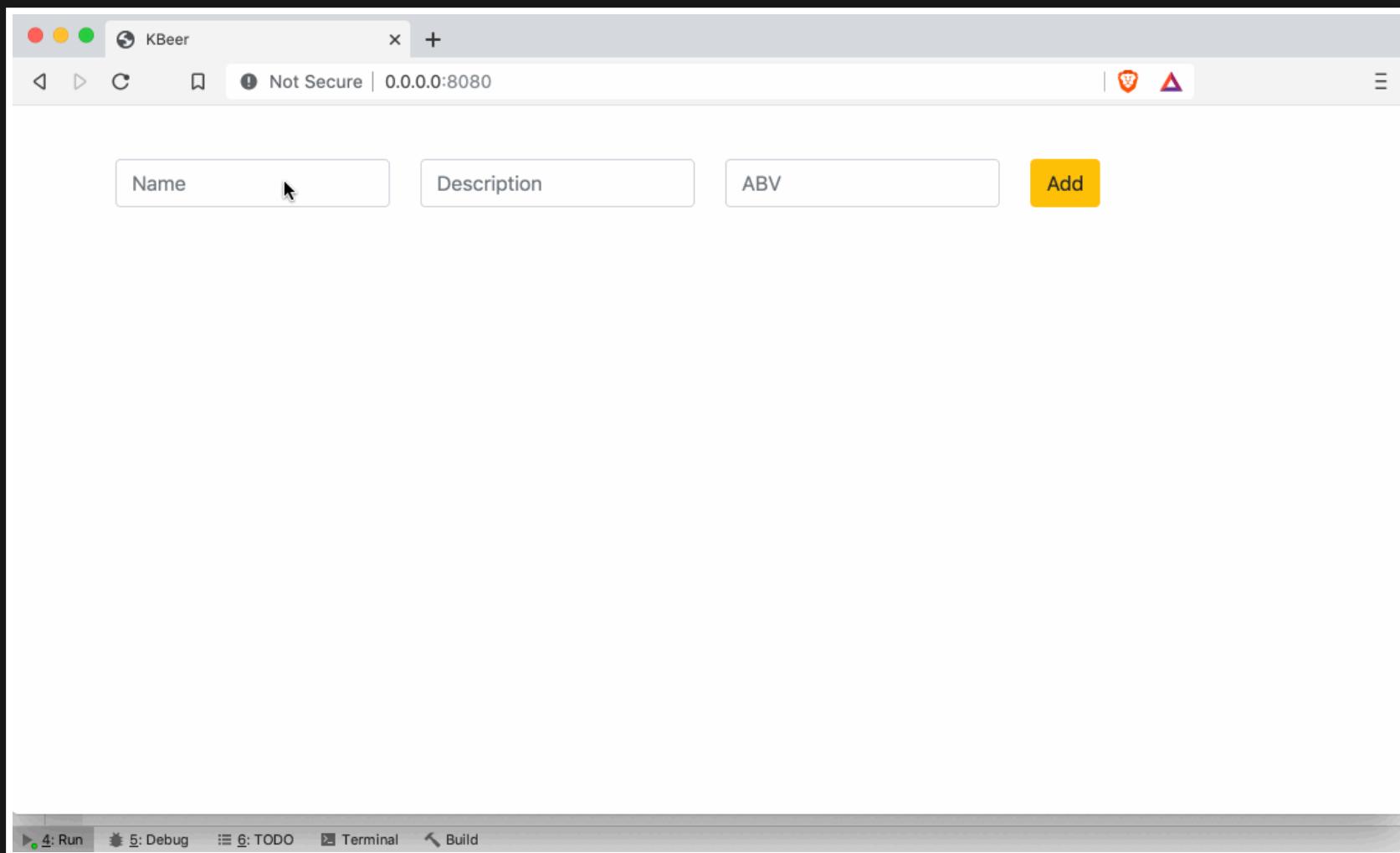
Template file beers.ftl

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5          <title>KBeer</title>
6          <link rel="stylesheet" type="text/css" href="bootstrap.min.css">
7          <link rel="stylesheet" type="text/css" href="main.css">
8      </head>
9      <body>
10         <div class="container with-margin-top">
11             <div class="row with-margin-top">
12                 <form action="/beer" method="post">
13                     <div class="col cell"><input type="text" name="name" class="form-control" placeholder="Name"/></div>
14                     <div class="col cell"><input type="text" name="description" class="form-control" placeholder="Description"/></div>
15                     <div class="col cell"><input type="text" name="abv" class="form-control" placeholder="ABV"/></div>
16                     <div class="col cell"><input type="hidden" name="action" value="add"></div>
17                     <div class="col cell"><button type="submit" class="btn btn-warning">Add</button></div>
18                 </form>
19             </div>
20         </div>
21         <#if beers?has_content>
22             <div class="container with-margin-top with-margin-bottom">
23                 <div class="row header">
24                     <div class="col-3 cell"><b>Name</b></div>
25                     <div class="col-7 cell"><b>Description</b></div>
26                     <div class="col-2 cell"><b>ABV</b></div>
27                 </div>
28                 <#list beers as beer>
29                     <div class="row with-border-bottom">
30                         <div class="col-3 cell">${beer.name}</div>
31                         <div class="col-7 cell">${beer.description}</div>
32                         <div class="col-2 cell">${beer.abv?string(",##0.0")}</div>
33                     </div>
34                 </#list>
35             </div>
36         </#if>
37     </body>
38 </html>
```

Adding the logic to add a beer - modify existing code

```
16  const val HOME = "/"
17  const val BEER = "/beer"
18
19
20  fun Route.beer(beerRepository: BeerRepository) {
21
22      get(HOME) { this: PipelineContext<Unit, ApplicationCall>
23          val beers = beerRepository.beers()
24          call.respond(FreeMarkerContent(template: "beers.ftl", mapOf("beers" to beers)))
25      }
26
27
28      post(BEER) { this: PipelineContext<Unit, ApplicationCall>
29          val parameters = call.receiveParameters()
30          val action = parameters["action"] ?: throw IllegalArgumentException("Missing parameter: action")
31          when (action) {
32              "add" -> {
33                  val name = parameters["name"] ?: throw IllegalArgumentException("Missing parameter: name")
34                  val description =
35                      parameters["description"] ?: throw IllegalArgumentException("Missing parameter: description")
36                  val abv =
37                      parameters["abv"] ?: throw IllegalArgumentException("Missing parameter: ABV")
38                  val beerToAdd = Beer(name, description, abv.toFloat())
39                  beerRepository.add(beerToAdd)
40              }
41          }
42          call.respondRedirect(HOME)
43      }
44  }
```

And now we run it...



Deleting a beer (template)

```
1  <!DOCTYPE html>
2  <html>
3 >    <head>=>
4  </head>
5  <body>
6      <div class="container with-margin-top">
7          <div class="row with-margin-top">
8              <form action="/beer" method="post">
9                  <div class="col cell"><input type="text" name="name" class="form-control" placeholder="Name"/></div>
10                 <div class="col cell"><input type="text" name="description" class="form-control" placeholder="Description"/></div>
11                 <div class="col cell"><input type="text" name="abv" class="form-control" placeholder="ABV"/></div>
12                 <div class="col cell"><input type="hidden" name="action" value="add"></div>
13                 <div class="col cell"><button type="submit" class="btn btn-warning">Add</button></div>
14             </form>
15         </div>
16     </div>
17     <#if beers?has_content>
18         <div class="container with-margin-top with-margin-bottom">
19             <div class="row header">
20                 <div class="col-2 cell"><b>Name</b></div>
21                 <div class="col-7 cell"><b>Description</b></div>
22                 <div class="col-1 cell"><b>ABV</b></div>
23                 <div class="col-2 cell">&nbsp;</div>
24             </div>
25             <#list beers as beer>
26                 <div class="row with-border-bottom">
27                     <div class="col-2 cell">${beer.name}</div>
28                     <div class="col-7 cell">${beer.description}</div>
29                     <div class="col-1 cell">${beer.abv?string(",##0.0")}</div>
30                     <div class="col-2 cell">
31                         <form action="/beer" method="post">
32                             <button type="submit" class="btn btn-danger">Delete</button>
33                             <input type="hidden" name="action" value="delete">
34                             <input class="btn btn-danger" type="hidden" name="id" value="${beer.id}" />
35                         </form>
36                     </div>
37                 </div>
38             </#list>
39         </div>
40     </div>
41     <#if>
42         <div>
43             <#if>
44                 <body>
45             </body>
46         </div>
```

Deleting a beer (code)

```
27
28 post(BEER) { this: PipelineContext<Unit, ApplicationCall>
29     val parameters = call.receiveParameters()
30     val action = parameters["action"] ?: throw IllegalArgumentException("Missing parameter: action")
31     when (action) {
32         "add" -> {
33             val name = parameters["name"] ?: throw IllegalArgumentException("Missing parameter: name")
34             val description =
35                 parameters["description"] ?: throw IllegalArgumentException("Missing parameter: description")
36             val abv =
37                 parameters["abv"] ?: throw IllegalArgumentException("Missing parameter: ABV")
38             val beerToAdd = Beer(name, description, abv.toFloat())
39             beerRepository.add(beerToAdd)
40         }
41         "delete" -> {
42             val id = parameters["id"] ?: throw IllegalArgumentException("Missing parameter: id")
43             beerRepository.remove(id.toInt())
44         }
45     }
46     call.respondRedirect(HOME)
47 }
48 }
49 }
```

Deleting a beer (in action)

Adding a random beer



PUNK API

V2 Documentation

Root Endpoint

The root endpoint should prefix all resources and is only accessible over HTTPS. CORS is also enabled.

<https://api.punkapi.com/v2/>

Creating our API client

The screenshot shows a file tree on the left and the code for a Ktor API client on the right.

File Tree:

- Kbeer (~Desktop/Ktor/Kbeer)
 - .gradle
 - .idea
 - build
 - gradle
 - resources [main] resources root
 - src [main] sources root
 - api
 - BeerApiClient
 - model
 - repository
 - webapp
 - Application.kt
 - test test sources root
 - build.gradle
 - gradle.properties
 - gradlew
 - gradlew.bat
 - settings.gradle
- External Libraries
- Scratches and Consoles

Code (BeerApiClient.kt):

```
1 package com.kbeer.api
2
3 import ...
4
5 class BeerApiClient {
6
7     companion object {
8         const val BASE_ENDPOINT =
9             "https://api.punkapi.com/v2/beers/random"
10    }
11
12    private val client = HttpClient(Apache) { this: HttpClientConfig<ApacheEngineConfig>
13        install(JsonFeature) { this: JsonFeature.Config
14            serializer = GsonSerializer()
15        }
16    }
17
18    suspend fun getBeerFromPunkIPA(): Beer {
19        try {
20            val content = client.get<ArrayList<Beer>>(BASE_ENDPOINT)
21            client.close()
22            return content[0]
23        } catch (e: Exception) {
24            throw Exception("Error retrieving IPA beer")
25        }
26    }
27
28
29
30
31
32
33
34
35
36 }
```

Creating the new endpoint and calling our API client

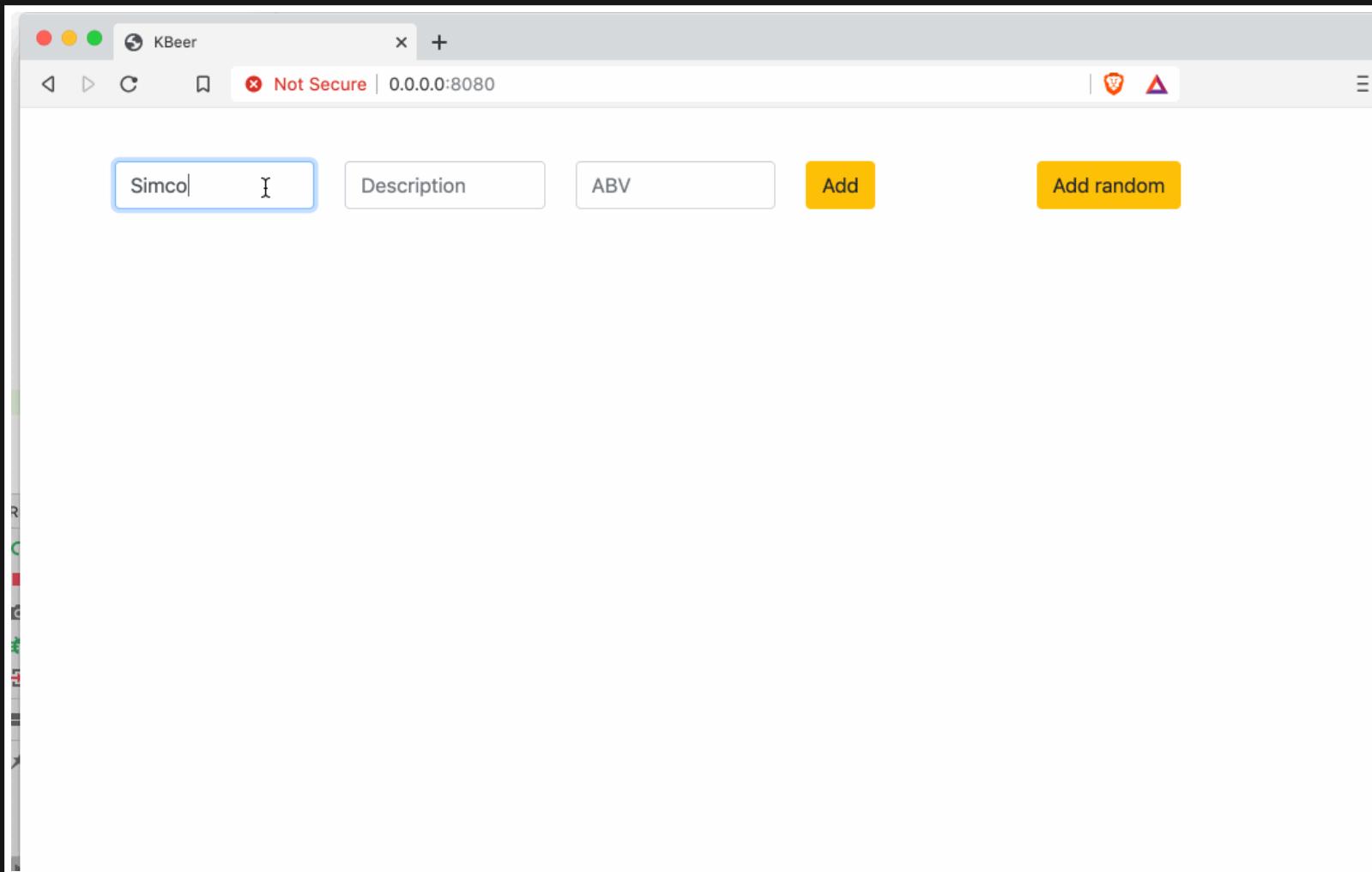
```
27
28     post(BEER) { this: PipelineContext<Unit, ApplicationCall>
29         val parameters = call.receiveParameters()
30         val action = parameters["action"] ?: throw IllegalArgumentException("Missing parameter: action")
31         when (action) {
32             "add" -> {
33                 val name = parameters["name"] ?: throw IllegalArgumentException("Missing parameter: name")
34                 val description =
35                     parameters["description"] ?: throw IllegalArgumentException("Missing parameter: description")
36                 val abv =
37                     parameters["abv"] ?: throw IllegalArgumentException("Missing parameter: ABV")
38                 val beerToAdd = Beer(name, description, abv.toFloat())
39                 beerRepository.add(beerToAdd)
40             }
41             "delete" -> {
42                 val id = parameters["id"] ?: throw IllegalArgumentException("Missing parameter: id")
43                 beerRepository.remove(id.toInt())
44             }
45             "random" -> {
46                 val beerApiClient = BeerApiClient()
47                 val beer = beerApiClient.getBeerFromPunkIPA()
48                 beerRepository.add(beer)
49             }
50         }
51         call.respondRedirect(HOME)
52     }
53 }
```

Modifying the template

```
1  <!DOCTYPE html>
2  <html>
3  <head>=-
4  </head>
5  <body>
6      <div class="container with-margin-top">
7          <div class="row with-margin-top">
8              <form action="/beer" method="post">
9                  <div class="col cell"><input type="text" name="name" class="form-control" placeholder="Name"/></div>
10                 <div class="col cell"><input type="text" name="description" class="form-control" placeholder="Description"/></div>
11                 <div class="col cell"><input type="text" name="abv" class="form-control" placeholder="ABV"/></div>
12                 <div class="col cell"><input type="hidden" name="action" value="add"/></div>
13                 <div class="col cell"><button type="submit" class="btn btn-warning">Add</button></div>
14             </form>
15             <div class="col cell">
16                 <form action="/beer" method="post">
17                     <input type="hidden" name="action" value="random">
18                     <button type="submit" class="btn btn-warning">Add random</button>
19                 </form>
20             </div>
21         </div>
22     </div>
23     <#if beers?has_content>
24         <div class="container with-margin-top with-margin-bottom">
25             <div class="row header">
26                 <div class="col-2 cell"><b>Name</b></div>
27                 <div class="col-7 cell"><b>Description</b></div>
28                 <div class="col-1 cell"><b>ABV</b></div>
29                 <div class="col-2 cell">&nbsp;</div>
30             </div>
31             <#list beers as beer>
32                 <div class="row with-border-bottom">
33                     <div class="col-2 cell">${beer.name}</div>
34                     <div class="col-7 cell">${beer.description}</div>
35                     <div class="col-1 cell">${beer.abv?string(",##0.0")}</div>
36                     <div class="col-2 cell">
37                         <form action="/beer" method="post">
```

```
42 <button type="submit" class="btn btn-danger">Delete</button>
43 <input type="hidden" name="action" value="delete">
44 <input class="btn btn-danger" type="hidden" name="id" value="${beer.id}" />
45 </form>
46 </div>
```

Adding a random beer (in action)



WHAT'S NEXT?

- Persistence and sessions
- Authentication
- Deployment

PROS AND CONS

- ✓ #KotlinEverywhere! (literally)
- ✓ Lightweight and customizable
- ✗ Documentation a bit confusing
- ✗ Choosing features when creating a new project

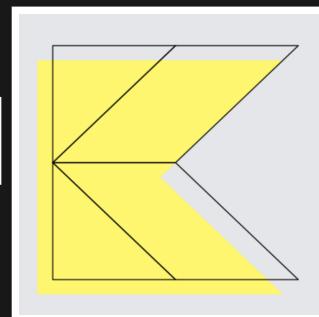
MATERIAL I FOUND HELPFUL

- Ryan Harter's video from KotlinConf 2018:
<https://www.youtube.com/watch?v=V4PS3ljlzlw>
- Dan Kim's video and slides from KotlinConf 2019:
<https://dankim.org/2019/12/16/kotlinconf-2019-ktor.html>
- Hadi Hariri's post on Ktor:
<https://www.infoq.com/articles/microservices-kotlin-ktor>
- Ray Wenderlich's "Server-Side Kotlin with Ktor":
<https://www.raywenderlich.com/2885892-server-side-kotlin-with-ktor>



(Have feedback?)

THAN



YOU!