Mini project #1 - Reflective web server with coroutines
Advanced Programming

Anders Kalhauge Tobias Grundtvig

Fall 2019

The project

The task is to create a web server which will use any class that implements a simple interface as content. The interface's only jobs are to mark a class as fit for web publishing and to provide a method, that tells the class to persist its memory.

Example of interface:

```
interface WebContent {
  fun save() // persist the content to file/database
  }
```

The content class should have functions corresponding to RESTful methods and urls:

- the url `/member` requested with the `GET` method corresponds to the method `getMember(): List<Member>`
- the url `/member/<integer>` also with `GET` corresponds to `getMember(id: Int): Member?`
- the url `/member` requested with `PUT` and a JSON member in the body, corresponds to `putMember(member: Member): Member`

Example of content:

```
class ChoirContent(/* filename, ... */) : WebContent {
  fun getMember(): List<Member> =
      TODO("Implement␣GET␣/member")
  fun getMember(id: Int): Member? =
      TODO("Implement␣GET␣/member/7")
  fun putMember(member: Member): Member =
      TODO("Implement␣PUT␣/member")
  // ...
  override fun save() {
    TODO("implement␣function␣save")
    }
  }
```

The data should be exchanged in JSON format, use a data class to describe the data, here is a **very** simple example:

```
data class Member(val id: Int, val name: String)
```

The web server interface can be as simple as:

```
class WebServer(val content: WebContent, val port: Int = 80) {
  fun start() { TODO("Implement␣start") }
  fun stop() { TODO("Implement␣stop") }
  }
```

You should be able to start the server with any class implementing the interface described above, not only the choir content!

The server should be as simple to start as:

```
fun main() {
  val content = ChoirContent(/* filename, ... */)
  val server = WebServer(content, 4711)
  server.start()
  }
```

...or even better:

```
ChoirContent(/* filename, ... */).publish(4711)
```

Requirements

The web server shall be based on raw socket calls, ie. no other middleware is allowed.

The server endpoints (accepted urls) shall be dynamically extracted from the content class using reflection.

Concurrency in the server should be handled using coroutines.

Data shall be communicated using JSON as protocol. You can use 3rd party software as gson for this task, or write your own parser as we did in class.

Hand in

A link to the github repository. In groups on Peergrade by Wednesday November 20th at 12:00