Library WebAppCommon

Stand 02.10.2020

Dies ist eine Shared Library (DLL), die verschiedene gemeinsam genutzte Funktionen bereitstellt, die in FastCGI- und C++/Qt-basierten Webapplikationen benötigt werden

Typischer Ablauf von solchen WebApps

Browser

URL

DOCBOX WebApp

FastCGI connector (libfcgi, 3rd party)

Webserver (z.B. nginx oder Apache)

HTTP server

View (HTML etc.)

PageManager

PageConfig

Controller/Action

ViewData

HTTP server response

Rendering (HTML, JS, CSS, etc)

Klasse **RequestData**

Dient dazu, HTTP GET und POST Parameter serverseitig abzufragen. Ebenso stehen Funktionen zum handeln von Dateiuploads bereit

Beispiele:

* QString wert = requestData->getString("feldname"); // liefert den GET-Parameter "feldname" als QString
* int wert = requestData->getInt("feldname"); // liefert den GET-Parameter "feldname" als Integer und validiert gleichzeitig, dass der Parameter nach int konvertiert werden kann
* int wert = requestData->postInt("feldname"); // liefert den POST-Parameter "feldname" als Integer und validiert gleichzeitig, dass der Parameter nach int konvertiert werden kann
* UploadedFile uploadedFile = requestData->uploadedFile("feldname");  
  uploadedFile.move("D:\foo\bar\zieldatei.pdf");

Klasse **SessionData**

Verwaltet HTTP-Sessionbezogene Daten mit Hilfe eines Session-Cookies und stellt Methoden bereit, um Werte in der Session zu speichern und zu lesen

Beispiele:

* QString wert = sessionData->stringValue("bezeichner"); // liest den String, der in der Session auf dem Server unter dem Bezeichner "bezeichner" gespeichert wurde
* sessionData->setValue("bezeichner", 42); // speichert den Integerwert 42 in der Session. Solange die Session gültig ist, kann dieser Wert mit intValue beim nächsten Aufruf serverseitig wieder abgerufen werden

Klasse **Action**

Subklassen implementieren die run-Methode. Controller-Action, die die Benutzereingabe auswertet, abhängig davon die DB-Transaktion steuert, die Daten von Model und Repository anfordert (Vergleichbar mit Controller-Actions beispielsweise bei Typo3 Extbase Extensions)  
 und für die View in einer **ViewData**-Subklasse sammelt (entspricht bei Typo3 view->assign("foo","bar"); )

Klasse **View**

Wie in MVC-Muster. Subklassen implementieren update (std::unique\_ptr<ViewData> viewdata). Diese Methode ist vergleichbar bei Java mit Observer: update(Observable o, Object arg), viewdata entspricht dem „arg“.  
Entsprechend ist in der Update-Methode ein Downcast erforderlich. Für std::unique\_ptr gibt dazu es eine Hilfsfunktion QtCommon2::static\_cast\_unique\_ptr im Header core/cast.h in der Library QtCommon2