Contents

[Step 1: Create a workspace. 2](#_Toc145926070)

[Create the directory structure: 2](#_Toc145926071)

[Create a Sports2000 database 2](#_Toc145926072)

[Serve the database on port 20000. 2](#_Toc145926073)

[Setup Developer Studio workspace 2](#_Toc145926074)

[Create Project 2](#_Toc145926075)

[Database connection 2](#_Toc145926076)

[Creating a PASOE 3](#_Toc145926077)

[Configuring a PASOE 3](#_Toc145926078)

[PROPATH 3](#_Toc145926079)

[Database Connection 4](#_Toc145926080)

[Webhandlers 4](#_Toc145926081)

[Testing the PASOE 4](#_Toc145926082)

[The goal of this workshop 8](#_Toc145926083)

[Rebuilding the webhandler 8](#_Toc145926084)

[Basic Cleanup 8](#_Toc145926085)

[Then empty the complete HandleGet method except for the RETURN 0. 8](#_Toc145926086)

[Communicating with the BusinessEntity 9](#_Toc145926087)

[Get a Single Record 10](#_Toc145926088)

[Getting a collection of records 12](#_Toc145926089)

[Set a filter, select the fields and set the batch size. 13](#_Toc145926090)

[Updating Data -HandlePost 15](#_Toc145926091)

# Step 1: Create a workspace.

## Create the directory structure:

mkdir C:\PUG\_2023

mkdir C:\PUG\_2023\WebhandlerWorkshop

mkdir C:\PUG\_2023\WebhandlerWorkshop\Eclipse

mkdir C:\PUG\_2023\WebhandlerWorkshop\DB

mkdir C:\PUG\_2023\WebhandlerWorkshop\src

## Create a Sports2000 database

Start a proenv and navigate to:

C:\PUG\_2023\WebhandlerWorkshop\DB

Execute a procopy to copy the sports2000 DB

procopy %DLC%\sports2000 Sports2000

## Serve the database on port 20000.

\_mprosrv Sports2000 -S 20000

# Setup Developer Studio workspace

Start OpenEdge Developer Studio

Choose the C:\PUG\_2023\WebhandlerWorkshop\Eclipse

as workspace directory.

Click on Workbench

## Create Project

In project explorer (on the left of the Developer Studio) click

Create new OpenEdge project.

Project Name: WebhandlerWorkshop

Project type: General OpenEdge Project

The location must be:

C:\PUG\_2023\WebhandlerWorkshop\src

## Database connection

Now configure the Developer Studio database connection

Window -> Preferences -> Progress OpenEdge -> Database connections

Connection Name: Sports2000

Physical Name: Sports2000

Port: 20000

Then we need to add the DB to the project.

In the project explorer right mouse click on the

WebHandler project -> Properties -> Progres OpenEdge -> Database connections

Select Sports2000 -> Apply and Close.

You could make your life a bit easier by

Windows -> Prefences -> Progress OpenEdge -> Editor

And set your preferred configuration for casing, auto complete etc.

# Creating a PASOE

Now go back to the proenv console, go to C:\PUG\_2023\WebhandlerWorkshop

and use:

%DLC%\servers\pasoe\bin\tcman create -p 30000 -P 30001 -s 30002 .\Pasoe

To create a PASOE in the folder: C:\PUG\_2023\WebhandlerWorkshop\Pasoe

## Configuring a PASOE

Go to: C:\PUG\_2023\WebhandlerWorkshop\Pasoe\conf

And open openedge.properties with an editor

To create a working PASOE with Webhandlers we need to:

* Set the PROPATH
* Connect the database
* Configure the WebHandlers

## PROPATH

Go to the section:

AppServer.Agent.Pasoe

Find the PROPATH entry and add: C:\PUG\_2023\WebhandlerWorkshop\src to the PROPATH while leaving the rest of the PROPATH intact.

## Database Connection

Then we need to configure the PASOE database connection. We go to the section:

AppServer.SessMgr.Pasoe

And we add: (case sensitive)

agentStartupParam=-db Sports2000 -H localhost -S 20000

This will enable the PASOE to connect to the sports2000 database.

## Webhandlers

Now we need to configure the webhandlers:

Go to the section: Pasoe.ROOT.WEB And add

handler1=Webhandler.CustomerWebhandler: /Customers/{CustNum}

handler2=Webhandler.CustomerWebhandler: /Customers

## Testing the PASOE

We can now start the PASOE to test the configuration:

C:\PUG\_2023\WebhandlerWorkshop\Pasoe\bin>tcman start

Wait some seconds (depending on your hardware specs) and check

C:\PUG\_2023\WebhandlerWorkshop\Pasoe\logs for the agent log. Open the agent log with your favorite tail program and check if the database is connected.

A close-up of a white page

Description automatically generated

Let's first test the principal of the Webhandler.

Create CustomerWebhandler.cls in the Webhandler folder:

A screenshot of a computer

Description automatically generated

/\*------------------------------------------------------------------------

File : CustomerWebhandler

Purpose :

Syntax :

Description :

Author(s) : Daniel

Created : Thu Sep 14 12:12:02 CEST 2023

Notes :

----------------------------------------------------------------------\*/

USING Progress.Lang.\*.

USING OpenEdge.Web.\* FROM PROPATH.

USING Progress.Json.ObjectModel.\* FROM PROPATH.

USING OpenEdge.Net.HTTP.\* FROM PROPATH.

BLOCK-LEVEL ON ERROR UNDO, THROW.

CLASS Webhandler.CustomerWebhandler INHERITS WebHandler:

/\*------------------------------------------------------------------------------

Purpose: Handler for unsupported methods. The request being serviced and

an optional status code is returned. A zero or null value means

this method will deal with all errors.

Notes:

------------------------------------------------------------------------------\*/

METHOD OVERRIDE PROTECTED INTEGER HandleNotAllowedMethod( INPUT poRequest AS OpenEdge.Web.IWebRequest ):

UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").

END METHOD.

/\*------------------------------------------------------------------------------

Purpose: Handler for unknown methods. The request being serviced and an

optional status code is returned. A zero or null value means

this method will deal with all errors.

Notes:

------------------------------------------------------------------------------\*/

METHOD OVERRIDE PROTECTED INTEGER HandleNotImplemented( INPUT poRequest AS OpenEdge.Web.IWebRequest ):

UNDO, THROW NEW Progress.Lang.AppError("METHOD NOT IMPLEMENTED").

END METHOD.

/\*------------------------------------------------------------------------------

Purpose: Default handler for the HTTP GET method. The request being

serviced and an optional status code is returned. A zero or

null value means this method will deal with all errors.

Notes:

------------------------------------------------------------------------------\*/

METHOD OVERRIDE PROTECTED INTEGER HandleGet( INPUT poRequest AS OpenEdge.Web.IWebRequest ):

DEFINE VARIABLE oJson AS JsonObject NO-UNDO .

DEFINE VARIABLE oResponse AS IHttpResponse NO-UNDO .

oJson = NEW JsonObject() .

oJson:Add ("Message", "Hello EMEA PUG 2023!").

ASSIGN

oResponse = NEW WebResponse ()

/\* HTTP messages require a content type \*/

oResponse:ContentType = 'application/json':U

oResponse:Entity = oJson

.

THIS-OBJECT:WriteResponse(oResponse).

RETURN 0.

END METHOD.

METHOD PROTECTED VOID WriteResponse (poResponse AS IHttpResponse):

DEFINE VARIABLE oWriter AS WebResponseWriter NO-UNDO.

oWriter = NEW WebResponseWriter (poResponse).

oWriter:Open ().

oWriter:Close ().

END METHOD.

END CLASS.

Save and compile the class.

restart the PASOE by killing the Java console (cross) and execute a tcman start in

C:\PUG\_2023\WebhandlerWorkshop\Pasoe\bin> tcman start

Then open the browser and go to:

http://localhost:3000/web/Customers

The result must be:

A screenshot of a computer

Description automatically generated

# The goal of this workshop

Write a webhandler which handles read and write access to the CustomerBusinessEntity for some basic functions for obtaining single records or collections.

The application is built on the Sports2000 database with the help of CCS /OERA compliant code.

In the OERA folder you will find base classes like BusinessEntity.cls.

In the BusinessLogic\Customer folder you will find the CustomerBusinessEntity which we will use today to fetch and save data. A BusinessEntity lives as a Service in the session and it can be addressed by using the ServiceManager.

Requesting data is done by a getDataTableRequest which is a JsonSerializable parameter class. This class has a number of properties to instruct the BusinessEntity which data and how much data to fetch. The code which handles the communication with the BusinessEntity is included.

# Rebuilding the webhandler

## Basic Cleanup

Let’s replace the method WriteResponse by the code below. For the ease of the workshop we will always return Json.

METHOD PROTECTED VOID WriteResponse (poBody AS OpenEdge.Core.String,

piStatusCode AS INTEGER):

DEFINE VARIABLE oResponse AS IHttpResponse NO-UNDO.

DEFINE VARIABLE oWriter AS WebResponseWriter NO-UNDO.

ASSIGN

oResponse = NEW OpenEdge.Web.WebResponse()

oResponse:StatusCode = piStatusCode

oResponse:Entity = poBody

oResponse:ContentType = 'application/json':u

oResponse:ContentLength = poBody:Size

.

/\* dump to stream \*/

oWriter = NEW WebResponseWriter (oResponse).

oWriter:Open ().

oWriter:Close ().

FINALLY:

DELETE OBJECT oResponse NO-ERROR.

DELETE OBJECT oWriter NO-ERROR.

END FINALLY.

END METHOD.

## Then empty the complete HandleGet method except for the RETURN 0.

A computer screen shot of a computer

Description automatically generated

Remove all the test code.

## Communicating with the BusinessEntity

First of all for ease of developing we need to include the dataset include into the CustomerWebhandler class.

CLASS Webhandler.CustomerWebhandler INHERITS WebHandler:

{BusinessLogic\Customer\dsCustomer.i}

Next we must insert the following methods, which handle querying for a collection of records or a single record.

/\*------------------------------------------------------------------------------

Purpose: Get Data from the Business Entity

Notes:

------------------------------------------------------------------------------\*/

METHOD PUBLIC VOID GetData(pcQueryString AS CHARACTER,

pcPagingContext AS CHARACTER,

piNumRecords AS INTEGER,

OUTPUT pcNextBatch AS CHARACTER,

OUTPUT pcPrevBatch AS CHARACTER):

DEFINE VARIABLE oRequest AS GetDataRequest NO-UNDO.

DEFINE VARIABLE oTableRequests AS IGetDataTableRequest NO-UNDO EXTENT.

DEFINE VARIABLE oTableRequest AS GetDataTableRequest NO-UNDO.

DEFINE VARIABLE oResponse AS IGetDataResponse NO-UNDO.

DEFINE VARIABLE oService AS IBusinessEntity NO-UNDO.

oService = CAST (ServiceManager:GetService ("BusinessLogic.Customer.CustomerBusinessEntity"), IBusinessEntity).

oTableRequest = NEW GetDataTableRequest ().

oTableRequest:NumRecords = piNumRecords.

oTableRequest:TableName = "Customer".

oTableRequest:QueryString = pcQueryString.

oTableRequest:PagingContext = pcPagingContext.

EXTENT (oTableRequests) = 1.

oTableRequests[1] = oTableRequest.

oRequest = NEW GetDataRequest ().

oRequest:TableRequests = oTableRequests.

oResponse = oService:getData(oRequest, OUTPUT DATASET dsCustomer).

END METHOD.

/\*------------------------------------------------------------------------------

Purpose: Get a Single Record from the Business Entity by using the unique key

Notes:

------------------------------------------------------------------------------\*/

METHOD PUBLIC HANDLE GetUniqueRecord(piCustNum AS INT64):

DEFINE VARIABLE cDummy1 AS CHARACTER NO-UNDO.

DEFINE VARIABLE cDummy2 AS CHARACTER NO-UNDO.

DEFINE VARIABLE cQuery AS CHARACTER NO-UNDO.

cQuery = SUBSTITUTE ("WHERE Customer.CustNum = &1",

piCustNum).

THIS-OBJECT:GetData(cQuery,

?,

1,

OUTPUT cDummy1,

OUTPUT cDummy2).

FIND FIRST eCustomer WHERE CustNum = piCustNum NO-ERROR.

END METHOD.

## Get a Single Record

We first want to get this part working:

handler1=Webhandler.CustomerWebhandler: /Customers/{CustNum}

Which results in the URL: [http://localhost:30000/web/Customers/<number](http://localhost:30000/web/Customers/%3cnumber)>

To achieve that, let’s first start with formulating the reply, we will need 4 variables, a JsonObject, a Longchar,an OpenEdge.Core.String for the reply and an integer for the Status. Like this successful operations can be nicely 200 OK.

METHOD OVERRIDE PROTECTED INTEGER HandleGet( INPUT poRequest AS OpenEdge.Web.IWebRequest ):

DEFINE VARIABLE oJsonObject AS JsonObject NO-UNDO.

DEFINE VARIABLE lcJson AS LONGCHAR NO-UNDO.

DEFINE VARIABLE oBody AS OpenEdge.Core.String NO-UNDO.

DEFINE VARIABLE iStatusCode AS INTEGER NO-UNDO.

oJsonObject = NEW JsonObject().

ASSIGN

lcJson = oJsonObject:GetJsonText()

oBody = NEW OpenEdge.Core.String(lcJSON)

.

WriteResponse(oBody, iStatusCode).

RETURN 0.

END METHOD.

We can use

iCustomerNumber = INT64 ( poRequest:GetPathParameter ("CustNum":U) ).

To obtain the CustomerNumber from the URL

When it’s not equal to zero or ? we can fetch the data from the BusinessEntity with 1 line of code:

IF iCustomerNumber > 0 THEN DO:

GetUniqueRecord(iCustomerNumber).

END.

The data is delivered tot the Webhandler in the form of a dataset and a temp-table. The temp-table we need to convert to a JsonObject.

IF AVAILABLE eCustomer THEN DO:

hTempTable = TEMP-TABLE eCustomer:HANDLE.

hTempTable:WRITE-JSON("JsonObject", oJsonObject).

iStatusCode = INTEGER( StatusCodeEnum:OK ).

END.

Then we can add some error handling too:

ELSE DO:

oJsonObject:Add ("Error", SUBSTITUTE ("Customer with CustNum &1 is not found.", iCustomerNumber)).

iStatusCode = INTEGER( StatusCodeEnum:NotFound ).

END.

A computer screen shot of a program

Description automatically generated

When the code is saved we can have a first try if we exposed our Single Customer record via REST by using a Webhandler.

http://localhost:30000/web/Customers/1

A screenshot of a computer screen

Description automatically generated

## Getting a collection of records

First of all we need to add some new variables to the method HandleGet.

DEFINE VARIABLE cQuery AS CHARACTER NO-UNDO.

DEFINE VARIABLE iBatchSize AS INTEGER NO-UNDO.

DEFINE VARIABLE cNextBatch AS CHARACTER NO-UNDO.

DEFINE VARIABLE cPrevBatch AS CHARACTER NO-UNDO.

cQuery will be the database query executed by the BusinessEntity. The iBatchSize determines the number of returned records. The cPrevBatch and cNextBatch will contain the row-id values of any records in a next or previous batch.

When INT64 ( poRequest:GetPathParameter ("CustNum":U) ) = 0

Then we do not seem to handle a single unique record, so we can conclude we handle a collection, therefor we can just use ELSE DO:.

ELSE DO:

/\*

\* a collection of records is requested

\*/

IF iBatchSize = ? OR iBatchSize = 0 THEN iBatchSize = 100.

cQuery = "".

GetData(cQuery,

"",

iBatchSize,

OUTPUT cNextBatch,

OUTPUT cPrevBatch).

END.

Afther the getData when the Temp-Table is filled, we can again, convert the Temp-Table to a JsonObject:

IF CAN-FIND (FIRST eCustomer) THEN DO:

hTempTable = TEMP-TABLE eCustomer:HANDLE.

hTempTable:WRITE-JSON( "JsonObject", oJsonObject ).

iStatusCode = INTEGER( StatusCodeEnum:OK ).

END.

And afterwards we do some error handling again:

ELSE DO:

oJsonObject:Add ("Error", "No Customers found").

iStatusCode = INTEGER( StatusCodeEnum:NotFound ).

END.

After we finished coding this we can test this:

http://localhost:30000/web/Customers

A computer screen shot of a computer program

Description automatically generated

## Set a filter, select the fields and set the batch size.

First of all the batch size. Most REST clients determine the size of the batches in which data is transferred.

The same counts for the fields which are returned. When a REST client does not need certain fields, then it would be a waste of bandwidth to send those.

Finally a REST client would like to filter on the data.

We will implement those three topics, so it can be used like this:

http://localhost:30000/web/Customers?Filter=Custnum%3E10&BatchSize=20&Fields=CustNum,Name,Country

To the method HandleGet we need to add two new variables:

DEFINE VARIABLE cFilterFields AS CHARACTER NO-UNDO.

DEFINE VARIABLE cWebQuery AS CHARACTER NO-UNDO.

IF iBatchSize = ? OR iBatchSize = 0 THEN iBatchSize = 100.

Just above this line we can now insert the following lines of code:

ASSIGN

cWebQuery = poRequest:URI:GetQueryValue('Filter')

cFilterFields = poRequest:URI:GetQueryValue('Fields')

iBatchSize = INTEGER ( poRequest:URI:GetQueryValue('BatchSize') )

NO-ERROR.

http://localhost:30000/web/Customers?Fields=CustNum,Name,Country

To make sure that this URL only selects the chosen fields as intended we have to walk all the fields in the Temp-Table and set the property SERIALIZE-HIDDEN to TRUE or FALSE.

We can create a method FilterFields:

METHOD PUBLIC VOID FilterFields( phTempTable AS HANDLE,

pcFields AS CHARACTER ):

END METHOD.

Then add the code to walk the Temp-Table fields and set SERIALIZE-HIDDEN.

DEFINE VARIABLE i AS INTEGER NO-UNDO.

DO i = 1 TO phTempTable:DEFAULT-BUFFER-HANDLE:NUM-FIELDS:

phTempTable:DEFAULT-BUFFER-HANDLE:BUFFER-FIELD(i):SERIALIZE-HIDDEN = NOT LOOKUP(phTempTable:DEFAULT-BUFFER-HANDLE:BUFFER-FIELD(i):NAME,pcFields) > 0.

END.

Just before we convert the Temp-Table into a Json, we can execute the method to show or hide the fields, we have to pass the TEMP-TABLE HANDLE to the method.

A screenshot of a computer code

Description automatically generated

Filtering the Query using the operators: =,<>,<,>,<=,=>:

Since developing those filters would take a bit long for the workshop, we can use the querybuilder object. Replace cQuery = “”. By:

cQuery = QueryBuilder:SimpleConvertWebQueryToOpenEdgeQuery(TEMP-TABLE eCustomer:HANDLE, cWebQuery).

This will convert the REST Query into an OpenEdge query.

<http://localhost:30000/web/Customers?Filter=Custnum%3E10,City=Austin&BatchSize=20&Fields=CustNum,Name,Country,City>

A screenshot of a computer

Description automatically generated

## Updating Data -HandlePost

Download a postman plugin for your webbrowser.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The plan is to get a unique record with a http GET.

Then update the Json data and use a http POST to update the data in the database.

First of all let’s create a HandlePost method:

METHOD OVERRIDE PUBLIC INTEGER HandlePost( INPUT poRequest AS OpenEdge.Web.IWebRequest ):

END.

We will need a number of variables in this method:

DEFINE VARIABLE oService AS IBusinessEntity NO-UNDO.

DEFINE VARIABLE oUpdateRequest AS UpdateDataRequest NO-UNDO.

DEFINE VARIABLE oJson AS JsonObject NO-UNDO.

DEFINE VARIABLE oBody AS Progress.Lang.Object NO-UNDO.

DEFINE VARIABLE oResponseBody AS OpenEdge.Core.String NO-UNDO.

DEFINE VARIABLE oEntityWriter AS OpenEdge.Net.HTTP.Filter.Payload.MessageWriter NO-UNDO.

DEFINE VARIABLE lcData AS LONGCHAR NO-UNDO.

DEFINE VARIABLE iStatusCode AS INTEGER NO-UNDO.

DEFINE VARIABLE lcJson AS LONGCHAR NO-UNDO.

Then we need to get our payload out of poRequest:Entity, that can be a bit tricky so just insert this code:

fix-codepage(lcData)= 'utf-8'.

oBody = poRequest:Entity.

IF TYPE-OF(oBody, Progress.Json.ObjectModel.JsonConstruct) THEN DO:

CAST(oBody, Progress.Json.ObjectModel.JsonConstruct):Write(OUTPUT lcData).

END.

ELSE IF TYPE-OF(oBody, Ccs.Common.Support.ILongcharHolder) THEN DO:

ASSIGN lcData = CAST(oBody, Ccs.Common.Support.ILongcharHolder):Value.

END.

ELSE IF TYPE-OF(oBody, OpenEdge.Core.Memptr)

OR TYPE-OF (oBody, OpenEdge.Core.ByteBucket)

THEN DO:

lcData = DYNAMIC-INVOKE(oBody, 'GetString', 1).

END.

ELSE

lcData = oBody:ToString().

MESSAGE STRING(lcData)

VIEW-AS ALERT-BOX.

A screenshot of a computer

Description automatically generated

And we can already see the Json in our logfile viewer when tailing the agent log in the Pasoe/logs directory:

A screenshot of a computer code

Description automatically generated

On top of the class define a new temp-table:

DEFINE TEMP-TABLE ttCustomer LIKE eCustomer.

Now in the HandlePost method, after the message lcData, place the following code:

/\*

\* Replace eCustomer by ttCustomer as a hack

\*/

lcData = REPLACE(lcData,"eCustomer","ttCustomer").

TEMP-TABLE ttCustomer:READ-JSON ("longchar", lcData, "MERGE").

We read the records into a separate Temp-Table, to be able to use the TRACKING-CHANGES and Before and After buffer options of the Temp-Table. Now we have the data in ttCustomer, we should obtain the data from the database in the eCustomer Temp-Table.

FOR EACH ttCustomer:

GetUniqueRecord(ttCustomer.CustNum).

END.

That fills the eCustomer table with the original values of the records.

Now we can use TRACKING-CHANGES on the eCustomer Temp-Table:

TEMP-TABLE eCustomer:TRACKING-CHANGES = TRUE.

And now we can update the table with the new or modified records:

FOR EACH ttCustomer:

FIND eCustomer WHERE eCustomer.CustNum = ttCustomer.CustNum NO-ERROR.

IF AVAILABLE eCustomer THEN DO:

BUFFER-COPY ttCustomer TO eCustomer.

iStatusCode = INTEGER (StatusCodeEnum:Accepted).

END.

ELSE DO:

CREATE eCustomer.

BUFFER-COPY ttCustomer TO eCustomer.

iStatusCode = INTEGER (StatusCodeEnum:Created).

END.

END.

After updating the data the BusinessEntity must save it, we request the ServiceManager for the BusinessEntity. The updateData method is not part of the IBusinessEntity interface, therefore we will need to cast oService as BusinessLogic.Customer.CustomerBusinessEntity to be able to access updateData.

oService = CAST (ServiceManager:GetService ("BusinessLogic.Customer.CustomerBusinessEntity"), IBusinessEntity).

oUpdateRequest = NEW UpdateDataRequest().

CAST (oService,BusinessLogic.Customer.CustomerBusinessEntity):updateData(DATASET dsCustomer, oUpdateRequest).

Finally, we can respond to the request with success:

oJson = NEW JsonObject().

oJson:Add ("Success", SUBSTITUTE ("Customer table updated")).

ASSIGN

lcJson = oJson:GetJsonText()

oResponseBody = NEW OpenEdge.Core.String(lcJSON)

.

WriteResponse(oResponseBody, iStatusCode).

RETURN 0.

It should look like this:

A screenshot of a computer program

Description automatically generated

## First use Postman with a GET to obtain a single Customer record:

A screenshot of a computer

Description automatically generated

Copy the Json,

Create a new Request of the POST type:

Select raw, and after raw, JSON.

Paste the Json.

Use the URL http://localhost:30000/web/Customers

A screenshot of a computer

Description automatically generated

Notice the status in the response:

A screenshot of a computer

Description automatically generated

Repeat the GET and confirm that the record really has been updated.

A screenshot of a computer

Description automatically generated

Check the PASOE agent logfile too:

A screen shot of a computer screen

Description automatically generated

## SalesRepWebhandler

When you are really fast, you could create a WebHandler for SalesRep. Which uses the SalesRep BusinessEntity. Adapt the code above to make it work.

In your SalesRepWebHandler, facilitate the uploading of a SalesRep picture, the picture can be saved into the SESSION:TEMP-DIR.

# Workshop Fin

We hope you enjoyed the workshop!

Please keep in mind that for the sake of the workshop we have taken some shortcuts.

All re occurring code in the WebHandler could be moved to a different (base) class. The literal strings like “Customer” and “BusinesLogic.Customer.CustomerBusinessEntity” could have been passed as properties. Only that would have taken away the fun of coding / copying and trying the code yourself.

Of course the SmartComponent Library offers the exposure of BusinessEntity data via the REST API out of the box. Next to that the SmartComponent Library is more