Backend Integration for the Employee Portal

In Chapter 7, you were shown the development of an employee portal. For complete implementation, you'll need the following ABAP source code for backend integration.

1.1 Backend Integration

Because the main focus of this book is SAPUI5, you are already very familiar with ABAP, therefore we will omit any explanations of the ABAP source code at this point. If you do not want to implement the ABAP part yourself, you will find the necessary ABAP objects as a transport request for download.

As already mentioned, this portal is based on your own database tables for data protection reasons. For this reason, you have to create the following dictionary objects.

1.1.1 Data Dictionary Objects

Domains

Domain ZUI5_STATUS (CHAR 2) with the following fixed values:

- 01—Created
- 02—Approved
- 03—Rejected

Domain ZUI5_ABS_TYPE (CHAR 2) with the following fixed values:

- 01—Leave
- 02—Special Leave

- 03—Sick Leave
- 04—Time in Lieu

Data Elements

- Data element ZUI5_STATUS with the elementary type domain ZUI5_STATUS
- Data element ZUI5_ABS_TYPE with the elementary type domain ZUI5_ABS_TYPE

Structures

Structure ZUI5_ABS (absence types)

Component	Component Type
ABS_KEY	DOMVALUE_L
ABS_TEXT	DDTEXT

Structure ZUI5_STAT (status types)

Component	Component Type
STATUS_KEY	DOMVALUE_L
STATUS_TEXT	DDTEXT

Structure ZUI5_ABS_TYPES (absences: input helps)

Component	Component Type
ABSENCE	ZUI5_ABS_T
STATUS	ZUI5_STAT_T

Structure ZUI5_TIME (time recording)

Component	Component Type
CUSTOMER	ZUI5_CUSTOMER_T
PROJECTS	ZUI5_PROJECTS_T
TIME_REPORT	ZUI5_TIME_SHEET_T

Table Types

- Table type ZUI5_ABS_T with the row type ZUI5_ABS
- Table type ZUI5_CUSTOMER_T with the row type ZUI5_CUSTOMER
- Table type ZUI5_PROJECTS_T with the row type ZUI5_PROJECTS
- Table type ZUI5_STAT_T with the row type ZUI5_STAT
- Table type ZUI5_TIME_SHEET_T with the row type ZUI5_TIME_SHEET

Database Tables

Application table ZUI5_ABSENCE (leave days); see Figure 1.1.

Field	Key	Ini...	Data element	Data Type	Length
MANDT	☑	☑	MANDT	CLNT	3
USERNAME	☑	☑	XUBNAME	CHAR	12
REQ_ID	☑	☑	GUID	RAW	16
START_DATE	☐	☐	SYDATUM	DATS	8
END_DATE	☐	☐	SYDATUM	DATS	8
TYPE	☐	☐	ZUI5_ABS_TYPE	CHAR	10
DESCR	☐	☐		STRING	0
STATUS	☐	☐	ZUI5_STATUS	CHAR	2

Figure 1.1 Table ZUI5_ABSENCE

Customizing table ZUI5_CUSTOMER (customers); see Figure 1.2.

Field	Key	Ini...	Data element	Data Type	Length
MANDT	☑	☑	MANDT	CLNT	3
CUSTOMER_ID	☑	☑	BU_PARTNER	CHAR	10
CUSTOMER_TEXT	☐	☐	BU_NAME0R1	CHAR	40

Figure 1.2 Table ZUI5_CUSTOMER

Generate a maintenance view for this Customizing table, and create a fictitious customer master, as shown, for example, in Figure 1.3.

	Partner	Name 1
	9000000001	TechCom
	9000000002	AB Company
	9000000003	VIP Customer
	9000000004	Pinapple Holding
	9000000005	Banana AG
	9000000006	Fruit Company

Figure 1.3 Customer Master for the Input Help for Time Recording

Customizing table ZUI5_EMPLOYEES (employee database); see Figure 1.4.

Field	Key	Ini...	Data element	Data Type	Length
MANDT	☑	☑	MANDT	CLNT	3
PERS_NO	☑	☑	AD_PERSNUM	CHAR	10
TITLE_P	☐	☐	AD_TITLETX	CHAR	30
FIRSTNAME	☐	☐	AD_NAMEFIR	CHAR	40
LASTNAME	☐	☐	AD_NAMELAS	CHAR	40
DEPARTMENT	☐	☐	AD_DPRTMNT	CHAR	40
FUNCTION	☐	☐	AD_FNCTN	CHAR	40
TEL_NUMBR	☐	☐	AD_TLNMBR1	CHAR	30
MOB_NUMBR	☐	☐	AD_TLNMBR1	CHAR	30
E_MAIL	☐	☐	AD_SMTPADR	CHAR	241

Figure 1.4 Employee List

Generate a maintenance view for this table also, and maintain as many fictional employees as possible. Here's a small tip: The maintenance of so many entries is very time-consuming, so if you have the option, copy approximately 100 users from the SAP user master record into this table.

Customizing table ZUI5_PROJECTS (projects); see Figure 1.5.

Field	Key	Ini...	Data element	Data Type	Length
MANDT	☑	☑	MANDT	CLNT	3
CUSTOMER_ID	☑	☑	BU_PARTNER	CHAR	10
PROJECT_ID	☑	☑		NUMC	8
PROJECT_DESC	☐	☐	BU_NAMEOR1	CHAR	40

Figure 1.5 Projects

Link the field CUSTOMER_ID via a foreign key relationship with the Customizing table ZUI5_CUSTOMER as a check table, and generate a maintenance view for this table also. Enter some projects for each customer from table ZUI5_CUSTOMER, as shown in Figure 1.6.

Partner	+	Name 1
9000000001	1	Application support
9000000001	2	SAP Development
9000000001	3	PoC Solution Manager
9000000001	4	SAP Consulting
9000000002	1	Application support
9000000002	2	Development
9000000002	3	Miscellaneous
9000000002	4	SAP Consulting
9000000003	1	Application support
9000000003	2	Development
9000000003	3	Upgrade
9000000003	4	SAP Consulting
9000000004	1	Application support
9000000004	2	Development
9000000004	3	SAP Consulting
9000000005	1	Application support
9000000005	2	SAPUI5 Development
9000000005	3	SAP Consulting
9000000006	1	Application support
9000000006	2	SAPUI5 Development
9000000006	3	SAP Consulting

Figure 1.6 Customer Projects

Application table ZUI5_TASK (task list); see Figure 1.7.

Field	Key	Ini...	Data element	Data Type	Length
MANDT	☑	☑	MANDT	CLNT	3
USERNAME	☑	☑	XUBNAME	CHAR	12
TASKID	☑	☑	GUID	RAW	16
CREA_DATE	☐	☐	CREATEDATE	DATS	8
DUE_DATE	☐	☐	DATS	DATS	8
DONE	☐	☐	XFELD	CHAR	1
PRIORITY	☐	☐		INT1	3
TASK	☐	☐		STRING	0

Figure 1.7 Task List

Application table ZUI5_TIME_SHEET (time recording); see Figure 1.8.

Field	Key	Ini...	Data element	Data Type	Length
MANDT	☑	☑	MANDT	CLNT	3
USERNAME	☑	☑	XUBNAME	CHAR	12
ACTIVITY_DATE	☑	☑	SYDATUM	DATS	8
TIME_FROM	☑	☑	TIMS	TIMS	6
TIME_TO	☑	☑	TIMS	TIMS	6
INTERNAL	☐	☐	XFELD	CHAR	1
CUSTOMER_ID	☐	☐	BU_PARTNER	CHAR	10
PROJECT_ID	☐	☐		NUMC	8
DESCRIPTION	☐	☐		STRING	0

Figure 1.8 Time Sheet

All necessary dictionary objects are now created, and we can turn to the processing class.

1.1.2 ABAP Class ZUI5_EMPLOYEE_PORTAL

HTTP Handler

In this application, we will again use the familiar HTTP handler `ZUI5_HTTP_HANDLER` and outsource only the business logic to a separate class. For this example, processing can take place in a stateless manner; thus, you have to create only a class with static methods. This new class, which we'll name `ZUI5_EMPLOYEE_PORTAL`, is called by the HTTP handler. The

examples and the transferred data are chosen deliberately so that each method is possible with the same interface. Create the final class ZUI5_EMPLOYEE_PORTAL with the methods shown in Figure 1.9.

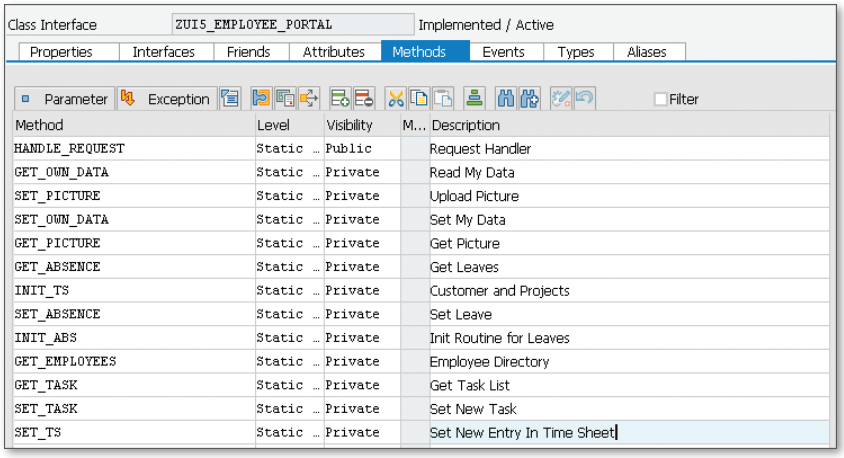

Figure 1.9 Methods of the Class ZUI5_EMPLOYEE_PORTAL

Dispatcher Method

Only the method HANDLE_REQUEST has the visibility PUBLIC, because it is called by the HTTP handler ZUI5_HTTP_HANDLER. This method acts as a kind of dispatcher and calls the respective business method depending on the context. For this reason, you can implement all other methods with the visibility PRIVATE. Each method has the same interface (see Figure 1.10).

Parameter	Type	P...	O...	Typing ...	Associated Type
IO_REQUEST	Importi...	☐	☐	Type Re...	IF_HTTP_REQUEST
ES_FILE	Exporti...	☑	☐	Type	TY_FILE
EV_JSON	Exporti...	☑	☐	Type	STRING

Figure 1.10 Interface of All Methods

Type Definitions

The class also contains two type definitions—TY_FILE and S_MESSAGE—with the definitions from Listing 1.1.

```
types:
    begin of ty_file,
           file_name    type  string,
           file_type    type  string,
           content_name type  string,
           file_content type  xstring,
         end of ty_file .
  types:
    begin of s_message,
                severity(1) type c,
                text        type string,
      end of s_message .
```

Listing 1.1 Type Definitions of the Class ZUI5_EMPLOYEE_PORTAL

Method HANDLE_REQUEST

AJAX Parameter Definition

In addition to the already known parameter `application`, we transfer a second parameter, `area`, in the SAPUI5 application, which means that the AJAX call will always have the form `/ui5?application=portal&area=<AREA>`. We specify the ABAP method name in this URL parameter; we can call the methods dynamically without much effort as a result. Thus, for example, the URL `/ui5?application=portal&area=get_task` will call the method `GET_TASK`.

As mentioned earlier, the method `HANDLE_REQUEST` assumes this task; for this method, this results in the content of Listing 1.2.

```
  data: lv_form            type string,
        exception          type ref to cx_root,
        lo_json            type ref to zui5_json_serializer,
        ls_message         type s_message,
        lv_error(1).
*Read URL parameter
lv_form = to_upper( io_request->get_form_field( 'area' ) ).

  if sy-subrc <> 0 or lv_form is initial.
    ls_message-severity = 'E'.
    ls_message-text     = 'Form field AREA is missing'.
    lv_error            = abap_true.
  endif.
* URL parameter corresponds to the method name
```

```
  try.
      call method (lv_form)
        exporting
          io_request = io_request
        importing
          ev_json    = ev_json
          es_file    = es_file.

    catch cx_sy_dyn_call_error into exception.
      ls_message-severity = 'E'.
      ls_message-text     = exception->get_text( ).
      lv_error            = abap_true.
  endtry.
* Create JSON
  if lv_error = abap_true.
     CREATE OBJECT lo_json
       EXPORTING
            DATA = ls_message.
* Serialize
   lo_json->serialize( ).
* JSON data string
   ev_json = lo_json->get_data( ).
endif.
```

Listing 1.2 Method HANDLE_REQUEST

Digression: JSON [▪]

As mentioned in Chapter 6, there is currently no standardized way in which to create a JSON file in ABAP. Therefore, you adjust the call for creating a JSON file depending on the tool you are using. At this point, it is important only that you create a valid file.

All we are missing now are the `getter` and `setter` methods of the individual portal areas.

Method GET_OWN_DATA

Reading Personal Data

The `GET_OWN_DATA` method is called by the portal in the My Data section, because the personal data is determined in this service (see Listing 1.3).

```
  data: address         type bapiaddr3.
        return          type standard table of bapiret2.
        alias           type bapialias.
        lo_json            type ref to zui5_json_serializer,
        lr_mime_rep     type ref to if_mr_api,
        content         type xstring,
        ls_loio         type skwf_io,
        lv_url          type string,
        ls_message         type s_message,
        lt_message      type message.

  call function 'BAPI_USER_GET_DETAIL'
    exporting
      username = sy-uname
    importing
      address  = address
    tables
      return   = return.

  if not return is initial.
    ls_message-severity = 'E'.
    ls_message-text     = 'BAPI_USER_GET_DETAIL failed'.
        create object lo_json
           exporting
              data = ls_message.
  else.
        create object lo_json
           exporting
              data = ls_message.
  endif.
* serialize data
   lo_json->serialize( ).
* get serialized json data string
   ev_json = lo_json->get_data( ).
```

Listing 1.3 Method GET_OWN_DATA

Method GET_PICTURE

Reading Image

The method `GET_PICTURE` loads the image of the employee from the MIME Repository. Note that the image is stored in the ALIAS field in the user master. If you already use this field for other purposes, you can store

the information in a separate database table. Then, adjust the methods SET_PICTURE and GET_PICTURE accordingly (see Listing 1.4).

```
  data: address            type bapiaddr3,
        return             type standard table of bapiret2,
        alias              type bapialias,
        lr_mime_rep     type ref to if_mr_api,
        content         type xstring,
        path               type string,
        lo_json            type ref to zui5_json_serializer,
        ls_message         type s_message,
        lv_error(1).

  call function 'BAPI_USER_GET_DETAIL'
    exporting
      username = sy-uname
    importing
      alias    = alias
    tables
      return   = return.

  if not alias-useralias is initial.
* Please adjust if necessary !
 es_file-file_name = alias-useralias.
 path = |SAP/EMPLOYEE/{ alias-useralias }|.
 lr_mime_rep = cl_mime_repository_api=>if_mr_api~get_api( ).
    lr_mime_rep->get(
      exporting
        i_url                 = path
        i_check_authority     = abap_false
      importing
         e_content             = es_file-file_content
         e_mime_type           = es_file-file_type
      exceptions
        parameter_missing     = 1
        error_occured         = 2
        not_found             = 3
        permission_failure    = 4
        others                = 5
           ).

  if sy-subrc <> 0.
    ls_message-severity = 'E'.
```

```
    ls_message-text     = 'Read from MIME failed'.
    lv_error            = abap_true.
    endif.
  else.
    ls_message-severity = 'W'.
    ls_message-text     = 'Picture not found'.
    lv_error            = abap_true.
  endif.

  if lv_error = abap_false.
    ls_message-severity = 'S'.
    ls_message-text     = 'Everything OK'.
  endif.

        create object lo_json
           exporting
              data = ls_message.

  lo_json->serialize( ).
  ev_json = lo_json->get_data( ).
```

Listing 1.4 Method GET_PICTURE

Method SET_PICTURE

Uploading Image

This method accepts the uploaded image and stores it in the MIME Repository under the path *SAP/EMPLOYEES/<file name>*. The file name is entered in the ALIAS field in the user master; adjust the location if necessary. As shown in Chapter 6, you must read the multipart segment in the first step (see Listing 1.5) and store it in an internal table.

```
data: lt_files          type table of ty_file,
      lv_content_type   type string,
      lv_content_name   type string,
      lv_filename       type string,
      lv_path           type string,
      lv_change         type abap_bool,
      lr_mime_rep       type ref to if_mr_api,
      alias             type bapialias,
      aliasx            type bapialiasx,
      return            type table of bapiret2,
      lo_json           type ref to zui5_json_serializer,
```

```
      ls_message         type s_message,
      ls_file            type ty_file,
      lo_multipart       type ref to if_http_entity,
      lv_error(1).

field-symbols: <ls_query> type ihttpnvp,
               <file>     type ty_file.

"File upload in the multipart segment
do io_request->num_multiparts( ) times.
  clear ls_file.
  "Read multipart
   lo_multipart =
           io_request->get_multipart( index = sy-index ).
  "Read content information
  lv_content_type = lo_multipart->get_header_field(
     name = if_http_header_fields=>content_type ).
  lv_content_name = lo_multipart->get_header_field(
     name = if_http_header_fields_sap=>content_name ).
  lv_filename = lo_multipart->get_header_field(
     name = if_http_header_fields_sap=>content_filename ).

  if not lv_content_type is initial and
     not lv_content_name is initial and
     not lv_filename     is initial.
    ls_file-file_content = lo_multipart->get_data( ).
    ls_file-content_name = lv_content_name.
    ls_file-file_type    = lv_content_type.
    ls_file-file_name    = lv_filename.
    append ls_file to lt_files.
  endif.
enddo.
```

Listing 1.5 Reading Multipart Segment

Writing Image into the MIME Repository

After you have read the image information from the multipart segment, you can then write this into the MIME Repository. In this example, write the image under the path *SAP/EMPLOYEE/<FILE NAME>* into the MIME Repository (see Listing 1.6).

```
if not lt_files[] is initial.
  "Load image into MIME
```

```
  read table lt_files assigning <file> index 1.

  lv_path = |SAP/EMPLOYEE/{ to_upper( <file>-file_name ) }|.
  condense lv_path no-gaps.

 lr_mime_rep = cl_mime_repository_api=>if_mr_api~get_api( ).

    lr_mime_rep->put(
      exporting
        i_url                    = lv_path
        i_content                = <file>-file_content
        i_suppress_package_dialog = 'X'
        i_dev_package            = '$TMP'
        i_suppress_dialogs       = 'X'
      exceptions
        parameter_missing      = 1
        error_occured          = 2
        cancelled                 = 3
        permission_failure     = 4
        data_inconsistency        = 5
        new_loio_already_exists   = 6
        is_folder                 = 7
        others                    = 8 ).

    if sy-subrc <> 0.
      ls_message-severity = 'E'.
      ls_message-text     = 'Upload to MIME failed'.
      lv_error            = abap_true.
    else.
      "Remember changes for update
      lv_change = abap_true.
      alias     = <file>-file_name.
      aliasx    = abap_true.
    endif.
  endif.
```

Listing 1.6 Saving Image in the MIME Repository

Writing Image Information in User Master

As described previously, the image path is stored in the ALIAS field in the user master. If you already used this field for other purposes, you can

also store the information in a separate table. Write the messages into a JSON file that you will display later on the UI (see Listing 1.7).

```
        create object lo_json
    exporting
      data = ls_message.
  lo_json->serialize( ).
  ev_json = lo_json->get_data( ).

  if lv_error = abap_false.
* Adjust if necessary !
    call function 'BAPI_USER_CHANGE'
      exporting
        username = sy-uname
        alias    = alias
        aliasx   = aliasx
      tables
        return   = return.

    if not return[] is initial.
      ls_message-severity = 'E'.
      ls_message-text     = 'BAPI_USER_CHANGE failed'.
      lv_error            = abap_true.
    endif.
  endif.
  if lv_error = abap_false.
    ls_message-severity = 'S'.
    ls_message-text     = 'Everything OK'.
  endif.

        create object lo_json
    exporting
      data = ls_message.
  lo_json->serialize( ).
  ev_json = lo_json->get_data( ).
```

Listing 1.7 Updating User Master

Method INIT_ABS

Absence Types

The method `INIT_ABS` reads the possible fixed values for the leave request (domains ZUI5_ABS_TYPE and ZUI5_STATUS). You need these fixed values in the portal for binding the corresponding `dropdown` controls. First,

read the fixed values of the domain ZUI5_ABS_TYPE to determine the absence types (see Listing 1.8).

```
data: abs_types          type zui5_abs_types,
      lt_dd07v           type standard table of dd07v,
      lo_json            type ref to zui5_json_serializer,
      ls_message         type s_message,
      lv_error(1).
field-symbols:   <abs_type>  type zui5_abs,
                 <abs_types> type zui5_abs_t,
                 <stat>      type zui5_stat,
                 <stats>     type zui5_stat_t,
                 <dd07v>     type dd07v.

assign component 'ABSENCE' of structure abs_types
                                    to <abs_types>.
assign component 'STATUS' of structure abs_types
                                    to <stats>.

call function 'DDIF_DOMA_GET'
  exporting
    name          = 'ZUI5_ABS_TYPE'
    langu         = sy-langu
  tables
    dd07v_tab     = lt_dd07v
  exceptions
    illegal_input = 1
    others        = 2.

if sy-subrc <> 0.
  ls_message-severity = 'E'.
  ls_message-text     = 'Read ZUI5_ABS_TYPE failed'.
  lv_error            = abap_true.
else.
  loop at lt_dd07v assigning <dd07v>.
    append initial line to <abs_types>
                       assigning <abs_type>.
    <abs_type>-abs_key  = <dd07v>-domvalue_l.
    <abs_type>-abs_text = <dd07v>-ddtext.
  endloop.
endif.
```

Listing 1.8 Fixed Values of the Absence Types

Status of the Leave Request

Next, you determine the fixed values of the domain ZUI5_STATUS for determining the possible status of a leave request (see Listing 1.9).

```
clear: lt_dd07v.
call function 'DDIF_DOMA_GET'
  exporting
    name          = 'ZUI5_STATUS'
    langu         = sy-langu
  tables
    dd07v_tab     = lt_dd07v
  exceptions
    illegal_input = 1
    others        = 2.

if sy-subrc <> 0.
  ls_message-severity = 'E'.
  ls_message-text     = 'Read ZUI5_STATUS failed'.
  lv_error            = abap_true.
else.
  loop at lt_dd07v assigning <dd07v>.
    append initial line to <stats>
                          assigning <stat>.
    <stat>-status_key = <dd07v>-domvalue_l.
    <stat>-status_text = <dd07v>-ddtext.
  endloop.
endif.

if lv_error = abap_true.
   CREATE OBJECT lo_json
     EXPORTING
          DATA = ls_message.
else.
    CREATE OBJECT lo_json
     EXPORTING
          DATA = abs_types.
endif.
 lo_json->serialize( ).
 ev_json = lo_json->get_data( ).
```

Listing 1.9 Status Values of the Leave Request

Method GET_ABSENCE

Leave Requests

The method `GET_ABSENCE` reads the stored absences from the database table and returns the information as JSON (see Listing 1.10).

```
data: lt_abs          type standard table of zui5_absence,
      ls_message         type s_message,
      lo_json         type ref to zui5_json_serializer.

select * from zui5_absence into table lt_abs
                            where username = sy-uname.

if sy-subrc = 0.
  create object lo_json
    exporting
      data = lt_abs.
else.
  ls_message-severity = 'I'.
  ls_message-text     = 'No absences found'.
  create object lo_json
    exporting
      data = ls_message.
endif.

lo_json->serialize( ).
ev_json = lo_json->get_data( ).
```

Listing 1.10 Method GET_ABSENCE

Method SET_ABSENCE

Leave Request

The leave request is transferred in the form fields STARTDATE, ENDDATE, REASON, and COMMENT. These form fields are read in the method `SET_ABSENCE` and are written to the database table ZUI5_ABSENCE (see Listing 1.11).

```
data: lt_query_string    type tihttpnvp,
      ls_abs             type zui5_absence,
      lo_json            type ref to zui5_json_serializer,
      ls_message         type s_message.

field-symbols: <ls_query> type ihttpnvp.
```

```
io_request->get_form_fields(
               CHANGING fields = lt_query_string ).
ls_abs-username = sy-uname.
ls_abs-req_id   =cl_system_uuid=>create_uuid_x16_static( ).

 loop at lt_query_string assigning <ls_query>.
   <ls_query>-name = to_upper( val = <ls_query>-name ).
   case <ls_query>-name.
     when 'STARTDATE' .
       ls_abs-start_date = <ls_query>-value.
     when 'ENDDATE'.
       ls_abs-end_date  = <ls_query>-value.
     when 'REASON'.
       ls_abs-type      = <ls_query>-value.
     when 'COMMENT'.
       ls_abs-descr     = <ls_query>-value.
   endcase.
 endloop.
 ls_abs-status = '01'. "Created

 insert into zui5_absence values ls_abs.

 case sy-subrc.
   when 0.
     ls_message-severity = 'S'.
     ls_message-text     = 'Absence created'.
   when others.
     ls_message-severity = 'E'.
      ls_message-text  = 'Absence
                           could not be created'.
 endcase.

       create object lo_json
   exporting
     data = ls_message.
* serialize data
 lo_json->serialize( ).
* get serialized json data string
 ev_json = lo_json->get_data( ).
```

Listing 1.11 Method SET_ABSENCE

Method SET_OWN_DATA

Reading Form Fields

You can maintain the selected master data yourself in the MY DATA area of the portal. This is transferred in the form fields DEPARTMENT, FUNCTION, TELEPHONE, and TELEPHONE_EXT. The form fields are read in the first part of the method `SET_OWN_DATA` (see Listing 1.12).

```
data: lt_query_string    type tihttpnvp,
      lv_change          type abap_bool,
      return             type table of bapiret2,
      lo_json            type ref to zui5_json_serializer,
      ls_message         type s_message,
      address            type bapiaddr3,
      addressx           type bapiaddr3x,
      lv_error(1).

field-symbols: <ls_query> type ihttpnvp,
               <return>   type bapiret2,
               <file>     type ty_file.

io_request->get_form_fields(
            CHANGING fields = lt_query_string ).

loop at lt_query_string assigning <ls_query>.
  <ls_query>-name = to_upper( val = <ls_query>-name ).
  case <ls_query>-name.
    when 'DEPARTMENT' .
      if not <ls_query>-value is initial.
        address-department  = <ls_query>-value.
        addressx-department = abap_true.
        lv_change = abap_true.
      endif.
    when 'FUNCTION'.
      if not <ls_query>-value is initial.
        address-function  = <ls_query>-value.
        addressx-function = abap_true.
        lv_change = abap_true.
      endif.
    when 'TELEPHONE'.
      if not <ls_query>-value is initial.
        address-tel1_numbr  = <ls_query>-value.
        addressx-tel1_numbr = abap_true.
```

```
        lv_change = abap_true.
      endif.
    when 'TELEPHONE_EXT'.
      if not <ls_query>-value is initial.
        address-tel1_ext  = <ls_query>-value.
        addressx-tel1_ext = abap_true.
        lv_change = abap_true.
      endif.
  endcase.
endloop.
```

Listing 1.12 Reading Form Fields

Changing User Master

In the second part of the method, the changed data is changed in the user master (see Listing 1.13).

```
if lv_change = abap_true.
  call function 'BAPI_USER_CHANGE'
    exporting
      username = sy-uname
      address  = address
      addressx = addressx
    tables
      return   = return.

  if not return[] is initial.
    loop at return assigning <return>
                     where type = 'E'.
      exit.
    endloop.
    if sy-subrc = 0.
      ls_message-severity = <return>-type.
      ls_message-text     = <return>-message.
    else.
      read table return assigning <return> index 1.
      ls_message-severity = <return>-type.
      ls_message-text     = <return>-message.
    endif.
  endif.
else.
  ls_message-severity = 'I'.
  ls_message-text     = 'Data was not changed'.
```

```
endif.

if ls_message-text is initial.
  ls_message-severity = 'S'.
  ls_message-text     = 'Data successfully changed'.
endif.

   CREATE OBJECT lo_json
     EXPORTING
          DATA = ls_message.
 lo_json->serialize( ).
 ev_json = lo_json->get_data( ).
```

Listing 1.13 Changing User Master

Method GET_EMPLOYEES

Employee Database

This method reads the employee database and provides the result as JSON (see Listing 1.14).

```
data: employees     type table of zui5_employees,
      ls_message         type s_message,
      lo_json         type ref to zui5_json_serializer.

select * from zui5_employees into table employees.

if sy-subrc <> 0.
  ls_message-severity = 'E'.
  ls_message-text     = 'No employees found'.
  create object lo_json
    exporting
      data = ls_message.
else.
  create object lo_json
    exporting
      data = employees.
endif.

lo_json->serialize( ).
ev_json = lo_json->get_data( ).
```

Listing 1.14 GET_EMPLOYEES

Method GET_TASK

Task List

This method reads the stored tasks and provides the result as JSON (see Listing 1.15).

```
data: lt_task            type standard table of zui5_task,
      ls_message         type s_message,
      lo_json         type ref to zui5_json_serializer.

select * from zui5_task into table lt_task
                           where username = sy-uname.

if sy-subrc = 0.
  create object lo_json
    exporting
      data = lt_task.
else.
  ls_message-severity = 'I'.
  ls_message-text     = 'No tasks found'.
  create object lo_json
    exporting
      data = ls_message.
endif.

lo_json->serialize( ).
ev_json = lo_json->get_data( ).
```

Listing 1.15 Method GET_TASK

Method SET_TASK

Creating Task

This method creates a new task or changes an existing task if, for example, this is set to *Done*. In this method, you first read the transferred fields from the HTTP response (see Listing 1.16).

```
data: lt_query_string    type tihttpnvp,
      ls_task            type zui5_task,
      lv_update          type abap_bool,
      lo_json            type ref to zui5_json_serializer,
      ls_message         type s_message.

field-symbols: <ls_query> type ihttpnvp.
```

```
io_request->get_form_fields(
            CHANGING fields = lt_query_string ).

ls_task-username  = sy-uname.
ls_task-taskid    =
               cl_system_uuid=>create_uuid_x16_static( ).
ls_task-crea_date = sy-datum.

loop at lt_query_string assigning <ls_query>.
  <ls_query>-name = to_upper( val = <ls_query>-name ).
  case <ls_query>-name.
    when 'TASKID'.
      if not <ls_query>-value is initial.
        ls_task-taskid  = <ls_query>-value.
        lv_update = abap_true.
      endif.
    when 'DUE_DATE'.
      ls_task-due_date = <ls_query>-value.
    when 'DONE'.
      if <ls_query>-value = 'X'.
        ls_task-done     = abap_true.
      else.
        ls_task-done     = abap_false.
      endif.
    when 'PRIORITY'.
      ls_task-priority = <ls_query>-value.
    when 'TASK'.
      ls_task-task    = <ls_query>-value.
  endcase.
endloop.
```

Listing 1.16 Reading Fields

Database Change

If a TaskID has been transferred from the UI, the status of the task is set; in all other cases, a new entry is written to the database (see Listing 1.17).

```
if lv_update = abap_true.
  update zui5_task set done = ls_task-done
               where taskid = ls_task-taskid.
else.
  insert into zui5_task values ls_task.
endif.
```

```
if sy-subrc = 0.
  ls_message-severity = 'I'.
  ls_message-text     = 'Task created / changed'.
else.
  ls_message-severity = 'E'.
  ls_message-text     = 'Data was not changed'.
endif.

      create object lo_json
  exporting
    data = ls_message.
lo_json->serialize( ).
ev_json = lo_json->get_data( ).
```

Listing 1.17 Updating Task List

Method INIT_TS

Reading Time Sheet

This method reads the time sheet entries of the last six months and provides them as JSON. The customers and the available projects are also read and transferred in this method (see Listing 1.18). By merging multiple transfers, you can very clearly see the difference to individual transfers; it is better for practicing the JSON binding to the SAPUI5 page.

```
data: customer      type zui5_customer,
      projects      type zui5_projects,
      begin         type sy-datum,
      ls_message          type s_message,
      lo_json             type ref to zui5_json_serializer,
      time          type zui5_time.
field-symbols:   <customer> type zui5_customer_t,
                 <projects> type zui5_projects_t,
                 <report>   type zui5_time_sheet_t.

assign component 'CUSTOMER' of structure time
                                to <customer>.
assign component 'PROJECTS' of structure time
                                to <projects>.
assign component 'TIME_REPORT' of structure time
                                to <report>.
```

```
select * from zui5_customer into table <customer>.
select * from zui5_projects into table <projects>.

"Half-year reporting
begin = sy-datum - 180.

select * from zui5_time_sheet into table <report>
                      where username = sy-uname
                        and activity_date >= begin.

if not time is initial.
  create object lo_json
    exporting
      data = time.
else.
  ls_message-severity = 'I'.
  ls_message-text     = 'Time sheet not found'.
  create object lo_json
    exporting
      data = ls_message.
endif.
lo_json->serialize( ).
ev_json = lo_json->get_data( ).
```

Listing 1.18 Method INIT_TS

Method SET_TS

Recording Times

This method reads the transferred entry for the time sheet from the request and writes it to the database. In this method, the relevant fields are first read from the request (see Listing 1.19).

```
data: lt_query_string    type tihttpnvp,
      ls_ts              type zui5_time_sheet,
      lv_update          type abap_bool,
      lv_delete          type abap_bool,
      lo_json            type ref to zui5_json_serializer,
      ls_message         type s_message.
field-symbols: <ls_query> type ihttpnvp.

io_request->get_form_fields(
            CHANGING fields = lt_query_string ).
```

```
ls_ts-username        = sy-uname.

loop at lt_query_string assigning <ls_query>.
  <ls_query>-name = to_upper( val = <ls_query>-name ).
  case <ls_query>-name.
    when 'DATE'.
      ls_ts-activity_date    = <ls_query>-value.
    when 'TIMEFROM'.
      ls_ts-time_from        = <ls_query>-value.
    when 'TIMETO'.
      ls_ts-time_to          = <ls_query>-value.
    when 'CUSTOMER'.
      ls_ts-customer_id      = <ls_query>-value.
    when 'PROJECT'.
      ls_ts-project_id       = <ls_query>-value.
    when 'INTERNAL'.
      if <ls_query>-value = 'true'.
        ls_ts-internal       = abap_true.
      endif.
    when 'COMMENT'.
      ls_ts-description     = <ls_query>-value.
    when 'DELETE'.
      if <ls_query>-value = abap_true.
        lv_delete = 'X'.
      endif.
    when 'KEY'.
      split <ls_query>-value at '&' into
      ls_ts-username ls_ts-activity_date
      ls_ts-time_from ls_ts-time_to
      in character mode.
  endcase.
endloop.
```

Listing 1.19 Reading Request Fields

Database Updating

Next, the values are written to the database and a corresponding message is returned to the UI (see Listing 1.20).

```
if lv_delete = 'X'.
  delete from zui5_time_sheet
              where username      = ls_ts-username
                and activity_date = ls_ts-activity_date
```

```
                and time_from     = ls_ts-time_from
                and time_to       = ls_ts-time_to.
else.
  insert into zui5_time_sheet values ls_ts.
endif.

if sy-subrc = 0.
  ls_message-severity = 'I'.
  if lv_delete = 'X'.
    ls_message-text     = 'Entry deleted'.
  else.
    ls_message-text     = 'Time recorded'.
  endif.
else.
  ls_message-severity = 'E'.
  ls_message-text     = 'Data was not changed'.
endif.

create object lo_json
  exporting
    data = ls_message.
lo_json->serialize( ).
ev_json = lo_json->get_data( ).
```

Listing 1.20 Writing Recorded Times to the Database

You have now implemented all services required by the portal. Finally, you have to add the queries from the portal to the HTTP handler.

1.1.3 HTTP Handler

In order to communicate with the portal, three things have to be adjusted in the HTTP handler:

- The queries from the portal application have to be forwarded to the processing class `ZUI5_EMPLOYEE_PORTAL`.
- As you learned in Chapter 6, image files are sent and received in binary format. For this reason, you also have to add the return of data in binary format to the HTTP return response.

- Because you sometimes send larger volumes of data, it is advisable to activate compression for the return response at this point.

This results in the adjustment of the class ZUI5_HTTP_HANDLER, as shown in Listing 1.21.

```
case <ls_query>-value.
* Employee portal
when 'portal'.
      zui5_employee_portal=>handle_request(
        exporting
          io_request = server->request
        importing
           es_file    = ls_file
           ev_json    = lv_json
             ).
*…
endcase.
* Activate compression
  server->response->set_compression(
                disable_extended_checks = abap_true
                options  = if_http_server=>co_compress_in_all_
cases ).

* Add data in binary format to response
  if not ls_file-file_content is initial.
  server->response->set_data( data = ls_file-file_content ).
  server->response->set_content_type(
                     content_type = ls_file-file_type ).
  else.
  server->response->set_cdata( data  = lv_json ).
  server->response->set_content_type(
                     content_type = 'application/json' ).
  endif.
```

Listing 1.21 Adjustment of the HTTP Handler

With this adjustment, all necessary steps are now completed in the backend.