Creatio Academy

Custom request handler

Custom request handler implemented using remote module

Version 8.0

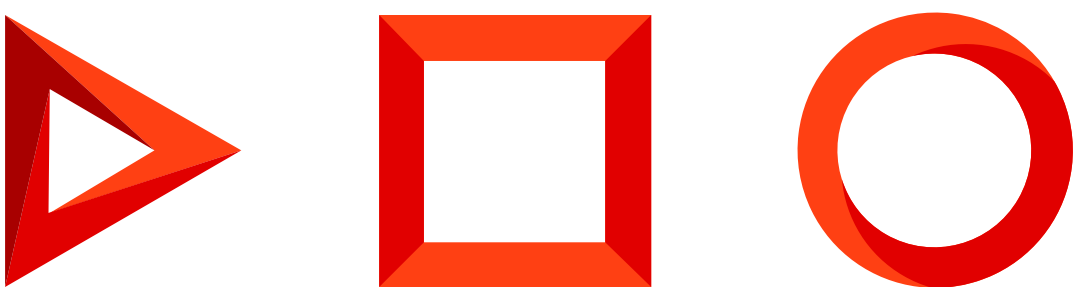


Table of Contents

Custom request handler implemented using remote module **4**
1. Create an Angular project to develop a custom request handler using remote module 4
2. Create a custom request using remote module 4
3. Create a custom service using remote module 5
4. Create custom request handlers using remote module 6
5. Add the request implemented using remote module to the Freedom UI page 10

Implement a custom request handler using remote module **11**
1. Create an Angular project to develop a custom request handler using remote module 11
2. Create a custom request using remote module 11
3. Create a custom service using remote module 12
4. Create custom request handlers using remote module 14
5. Add the request implemented using remote module to the Freedom UI page 18

Outcome of the example 19

Custom request handler implemented using remote module

Medium

You can implement a custom request handler using remote module in Creatio version 8.0.8 Atlas and later.

Creatio lets you implement the business logic of the Freedom UI page using **request handlers**. Creatio executes request handlers when an event is triggered on a visual element of a Freedom UI page. You can use base request handlers or implement a new one. Base handlers are executed at different life cycle stages. Learn more in a separate article: Creatio front-end architecture.

You can modify the business logic of a request handler based on the type of related request. View the request type in the component documentation or source code of the schema that implements the request logic.

You can implement a custom request handler in the following **ways**:

- Use the `handlers` schema section of the Freedom UI page. Supported in Creatio 8.0 Atlas and later. View examples that invoke handlers in separate articles: Page customization.
- Use a remote module. Supported in Creatio 8.0.8 Atlas and later.

To **implement a custom request handler using remote module**:

1. Create an Angular project to develop a custom request handler using remote module.
2. Create a custom request using remote module.
3. Create a custom service using remote module.
4. Create custom request handlers using remote module.
5. Add the request implemented using remote module to the Freedom UI page.

1. Create an Angular project to develop a custom request handler using remote module

Develop a custom request handler in a dedicated `npm` package using an external IDE. This example covers the request handler development in Microsoft Visual Studio Code.

You can create an Angular project to develop a custom request handler in multiple ways, similarly to creating an Angular project to develop a custom UI component using remote module. To do this, follow the instruction in a separate article: Custom UI component implemented using remote module.

2. Create a custom request using remote module

1. Create an Angular class in the project. To do this, run the `ng g class some-request-name.request` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the `SomeRequestNameRequest` class files to the `src/app/` project directory.

2. Implement the request.

   a. Open the `some-request-name.request.ts` file.

   b. Inherit the `BaseRequest` class from the `@creatio-devkit/common` library.

   c. Add the type and parameter of the request.

   d. Import the required functionality from the libraries into the class.

   e. Save the file.

`some-request-name.request.ts` file

```
/* Import the required functionality from the libraries. */
import { BaseRequest } from "@creatio-devkit/common";

export class SomeRequestNameRequest extends BaseRequest{
  /* The type and parameters of the request. */
  public someParameterName!: someParatemerType;
}
```

3. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `<%projectName%>` name specified on step 1.

3. Create a custom service using remote module

1. Create an Angular class in the project. To do this, run the `ng g class some-service-name.service` command at the command line terminal of Microsoft Visual Studio Code.

   As a result, Microsoft Visual Studio Code will add the `SomeServiceNameService` class files to the `src/app/` project directory.

2. Implement the service that receives data from external web service.

   a. Open the `some-service-name.service.ts` file.

   b. Flag the `SomeServiceNameService` class using the `Injectable` decorator.

   c. Implement the `someMethodName()` method that receives data from external web service.

   d. Import the required functionality from the libraries into the class.

   e. Save the file.

`some-service-name.service.ts` file

```
/* Import the required functionality from the libraries. */
```

```
import { Injectable } from "@angular/core";

@Injectable({
  providedIn: 'root',
})
export class SomeServiceNameService {
  /* Receive data from external web service. */
  public someMethodName(someParameterName: someParatemerType): Promise<{
    someParameterName: Record<string, number>
  }> {
    /* Implement the business logic. */
    ...

    /* Return data from external web service. */
    return Promise.resolve({
      someParameterName: {'someParameterName': /* Receive data from external web service. */
    });
  }
}
```

3. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `<%projectName%>` name specified on step 1.

4. Create custom request handlers using remote module

1. Create a custom handler that displays data received from the external web service on the Freedom UI page.
   a. Set up the Freedom UI page.
      a. Repeat steps 1-6 of the procedure to implement a custom UI component using remote module.
      b. Add a `SomeAttributeName` attribute that stores data received from the external web service to the `viewModelConfig` schema section.

`viewModelConfig` schema section

```
viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
    "attributes": {
        ...,
        /* The attribute that stores data received from the external web service. */
        "SomeAttributeName": {}
    }
}/**SCHEMA_VIEW_MODEL_CONFIG*/,
```

c. Bind the `caption` property of the corresponding component to the `$SomeAttributeName` model attribute in the `viewConfigDiff` schema section.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    ...,
    {
        "operation": "insert",
        "name": "SomeComponentName",
        "values": {
            ...,
            /* Bind the SomeAttributeName attribute to the caption property. */
            "caption": "$SomeAttributeName",
            ...
        },
        ...
    }
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

d. Click [*Save*] on the Client Module Designer's toolbar.

b. Create an Angular class in the project. To do this, run the `ng g class some-handler-name.handler` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the `SomeHandlerNameHandler` class files to the `src/app/` project directory.

c. Implement the handler.

a. Open the `some-handler-name.handler.ts` file.

b. Add the configuration object that declares the request handler.

- Set `type` property to `usr.SomeHandlerNameHandler`. The `type` property is the type of handler.
- Set `requestType` property to `crt.UpdateCurrentTimeRequest`. The `type` property is the type of request to execute the handler specified in the `type` property.

e. Flag the `SomeHandlerNameHandler` class using the `CrtRequestHandler` decorator.

f. Inherit the `BaseRequestHandler` class from the `@creatio-devkit/common` library.

g. Implement the handling of the request result.

h. Generate the value to display in the component of Freedom UI page.

i. Import the required functionality from the libraries into the class.

j. Save the file.

`some-handler-name.handler.ts` file

```
/* Import the required functionality from the libraries. */
import { BaseRequestHandler, CrtRequestHandler } from "@creatio-devkit/common";
import { SomeServiceNameService } from "./some-service-name.service";
import { SomeRequestNameRequest } from "./some-request-name.request";

/* Add the CrtRequestHandler decorator to the SomeHandlerNameHandler class. */
@CrtRequestHandler({
    type: 'usr.SomeHandlerNameHandler',
    requestType: 'crt.SomeRequestNameRequest',
  })

export class SomeHandlerNameHandler extends BaseRequestHandler{
  constructor(private _someServiceNameService: SomeServiceNameService) {
    super();
  }
  public async handle(request: SomeRequestNameRequest): Promise<unknown> {
    const result = await this._someServiceNameService.someMethodName(request.someParameterN
    const someParameter1Name = result.someParameterName[request.someParameterName] ?? 0
    /* Generate the value to display in the component of Freedom UI page. */
    request.$context['SomeAttributeName'] = someParameter1Name;
    return someParameter1Name;
  }
}
```

2. Create a custom handler that is executed when Creatio initializes a Freedom UI page.

   a. Create an Angular class in the project. To do this, run the `ng g class some-handler1-name.handler` command at the command line terminal of Microsoft Visual Studio Code.

   As a result, Microsoft Visual Studio Code will add the `SomeHandler1NameHandler` class files to the `src/app/` project directory.

   b. Implement the handler.

      a. Open the `some-handler1-name.handler.ts` file.

      b. Add the configuration object that declares the request handler.

         - Set `type` property to `usr.SomeHandler1NameHandler`.
         - Set `requestType` property to `crt.HandleViewModelInitRequest`.
         - Set `scopes` property to code of Freedom UI page. The request handler is executed when Creatio initializes the Freedom UI page that has the specified code. If the property is empty or missing, it is a global handler that Creatio executes on all Freedom UI pages.

      f. Flag the `SomeHandler1NameHandler` class using the `CrtRequestHandler` decorator.

      g. Inherit the `BaseRequestHandler` class from the `@creatio-devkit/common` library.

      h. Implement the update of data received from the external web service when Creatio initializes the Freedom UI page.

i. Import the required functionality from the libraries into the class.

j. Save the file.

`some-handler1-name.handler.ts` file

```
/* Import the required functionality from the libraries. */
import { BaseRequest, BaseRequestHandler, CrtRequestHandler, HandlerChainService } from "@
import { SomeRequestNameRequest } from "./some-request-name.request";

/* Add the CrtRequestHandler decorator to the SomeHandler1NameHandler class. */
@CrtRequestHandler({
  type: 'usr.SomeHandler1NameHandler',
  requestType: 'crt.HandleViewModelInitRequest',
  scopes: ['CodeOfSomeFreedomUIPage'],
})

export class SomeHandler1NameHandler extends BaseRequestHandler{
  public async handle(request: BaseRequest): Promise<unknown> {
    /* Update data received from the external web service when Creatio initializes the Free
    await HandlerChainService.instance.process({
      type: 'crt.SomeRequestNameRequest',
      someParameterName: 'someParameterValue',
      $context: request.$context
    } as SomeRequestNameRequest);

    return this.next?.handle(request);
  }
}
```

3. Register the handlers.

a. Open the `app.module.ts` file.

b. Add the `SomeHandlerNameHandler` and `SomeHandler1NameHandler` handlers to the `requestHandlers` section in the `CrtModule` decorator.

c. Import the required functionality from the libraries into the class.

d. Save the file.

`app.module.ts` file

```
/* Import the required functionality from the libraries. */
import { CrtModule } from '@creatio-devkit/common';
import { SomeHandlerNameHandler } from './some-handler-name.handler';
import { SomeHandler1NameHandler } from './some-handler1-name.handler';
...
```

```
@CrtModule({
  ...,
  /* Specify that SomeHandlerNameHandler and SomeHandler1NameHandler are request handlers. */
  requestHandlers: [
    SomeHandlerNameHandler,
    SomeHandler1NameHandler
  ],
})
...
```

4. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `<%projectName%>` name specified on step 1.

5. Add the request implemented using remote module to the Freedom UI page

1. Repeat steps 2-7 of the procedure to implement custom UI component using remote module.
2. Change the property value of the corresponding component in the `viewConfigDiff` schema section to request configuration object.
   - Enter the request name in the `request` property.
   - Enter the configuration object of parameters in the `params` property.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    {
        "operation": "insert",
        "name": "SomeButtonName",
        "values": {
            ...,
            "clicked": {
                /* Bind the sending of the custom request to the button click event. */
                "request": "crt.SomeRequestNameRequest",
                "params": {
                    "someParameterName": "someParameterValue"
                }
            }
        },
        ...
    },
    ...
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

3. Click [*Save*] on the Client Module Designer's toolbar.

As a result, Creatio will add the request to the Freedom UI page. When an event initiates a request, the related handler will be executed.

Implement a custom request handler using remote module

Medium

The example is relevant to Creatio version 8.0.8 and later.

Example. Add request handlers that execute the following actions:

- Display current date and time in the [*Label*] type component on the record page of the custom [*Requests*] section.
- Change the value in the [*Label*] type component after a user clicks the custom button.

Implement handlers using a remote module created in Angular framework.

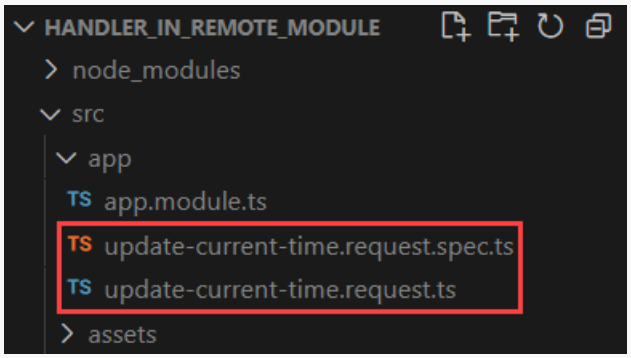
1. Create an Angular project to develop a custom request handler using remote module

To **create an Angular project to develop a custom request handler using remote module**, follow the instruction in a separate article: Implement custom UI component using remote module.

2. Create a custom request using remote module

1. Create an Angular class in the project. To do this, run the `ng g class update-current-time.request` command at the command line terminal of Microsoft Visual Studio Code.

   As a result, Microsoft Visual Studio Code will add the `UpdateCurrentTimeRequest` class files to the `src/app/` project directory.



2. Implement the request.

   a. Open the `update-current-time.request.ts` file.

   b. Inherit the `BaseRequest` class from the `@creatio-devkit/common` library.

   c. Add the type and parameter of the request.

   d. Import the required functionality from the libraries into the class.

   e. Save the file.

`update-current-time.request.ts` file

```
/* Import the required functionality from the libraries. */
import { BaseRequest } from "@creatio-devkit/common";

export class UpdateCurrentTimeRequest extends BaseRequest{
  /* The type and parameters of the request. */
  public dateTime!: string;
}
```

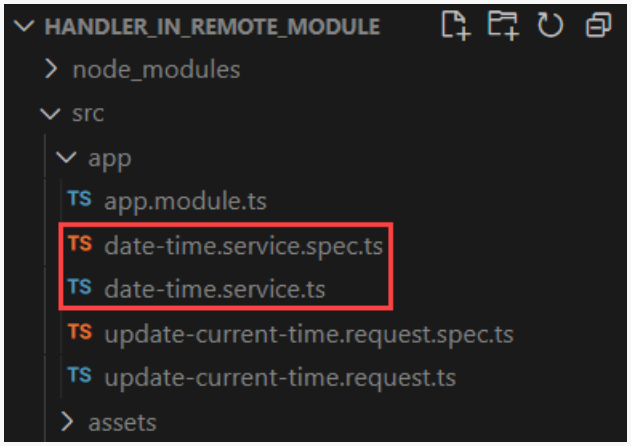
3. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `sdk_remote_module_package` name.

3. Create a custom service using remote module

1. Create an Angular class in the project. To do this, run the `ng g class date-time.service` command at the command line terminal of Microsoft Visual Studio Code.

   As a result, Microsoft Visual Studio Code will add the `DateTimeService` class files to the `src/app/` project directory.



2. Implement the service that generates current date and time.

   a. Open the `date-time.service.ts` file.

   b. Flag the `DateTimeService` class using the `Injectable` decorator.

   c. Implement the `getCurrentDateTime()` method that receives current date and time.

   d. Import the required functionality from the libraries into the class.

   e. Save the file.

`date-time.service.ts` file

```
/* Import the required functionality from the libraries. */
import { formatDate } from "@angular/common";
import { Injectable } from "@angular/core";

@Injectable({
  providedIn: 'root',
})
export class DateTimeService {
  /* Receive current date and time. */
  public getCurrentDateTime(dateTime: string): Promise<{
    dateTime: Record<string, unknown>
  }> {
    /* Return current date and time. */
    return Promise.resolve({
      dateTime: {'dateTime': formatDate(new Date(), 'medium', 'en')}
    });
  }
}
```

3. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `sdk_remote_module_package` name.

4. Create custom request handlers using remote module

1. Create a custom handler that displays current date and time on the Freedom UI page.

   a. Set up the Freedom UI page.

      a. Use the [*Records & business processes*] template to create a custom `Requests` app. To do this, follow the instruction in the user documentation: Manage apps.

      b. Open the [*Requests form page*] page in the working area of the `Requests` app page.

      c. Add a [*Label*] type component to the working area of the Freedom UI Designer.

      d. Click the  button in the action panel of the Freedom UI Designer. After you save the page settings, Creatio opens the source code of the Freedom UI page.

      e. Add a `CurrentDateTime` attribute that stores data about the current date and time to the `viewModelConfig` schema section.

      `viewModelConfig` schema section

      ```
      viewModelConfig: /**SCHEMA_VIEW_MODEL_CONFIG*/{
          "attributes": {
              ...,
              /* The attribute that stores the current date and time. */
              "CurrentDateTime": {}
          }
      }/**SCHEMA_VIEW_MODEL_CONFIG*/,
      ```

      f. Bind the `caption` property of the `Label` element to the `$CurrentDateTime` model attribute in the `viewConfigDiff` schema section.

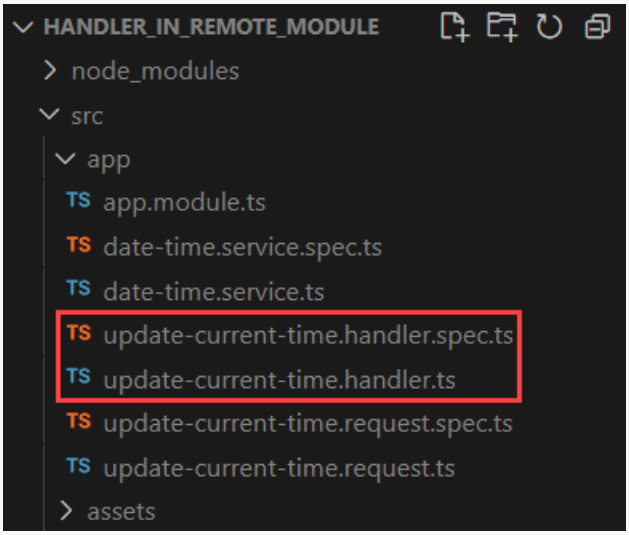
      `viewConfigDiff` schema section

      ```
      viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
          ...,
          {
              "operation": "insert",
              "name": "Label",
              "values": {
                  ...,
                  /* Bind the CurrentDateTime attribute to the caption property. */
                  "caption": "$CurrentDateTime",
                  ...
              },
              ...
          }
      ]/**SCHEMA_VIEW_CONFIG_DIFF*/,
      ```

g. Click [*Save*] on the Client Module Designer's toolbar.

b. Create an Angular class in the project. To do this, run the `ng g class update-current-time.handler` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the `UpdateCurrentTimeHandler` class files to the `src/app/` project directory.



c. Implement the handler.

a. Open the `update-current-time.handler.ts` file.

b. Add the configuration object that declares the request handler.

- Set the `type` property to `usr.UpdateCurrentTimeHandler`.
- Set the `requestType` property to `crt.UpdateCurrentTimeRequest`.

e. Flag the `UpdateCurrentTimeHandler` class using the `CrtRequestHandler` decorator.

f. Inherit the `BaseRequestHandler` class from the `@creatio-devkit/common` library.

g. Implement the handling of the request result.

h. Generate the value to display in the component of Freedom UI page.

i. Import the required functionality from the libraries into the class.

j. Save the file.

`update-current-time.handler.ts` file

```
/* Import the required functionality from the libraries. */
import { BaseRequestHandler, CrtRequestHandler } from "@creatio-devkit/common";
import { DateTimeService } from "./date-time.service";
import { UpdateCurrentTimeRequest } from "./update-current-time.request";

/* Add the CrtRequestHandler decorator to the UpdateCurrentTimeHandler class. */
@CrtRequestHandler({
```

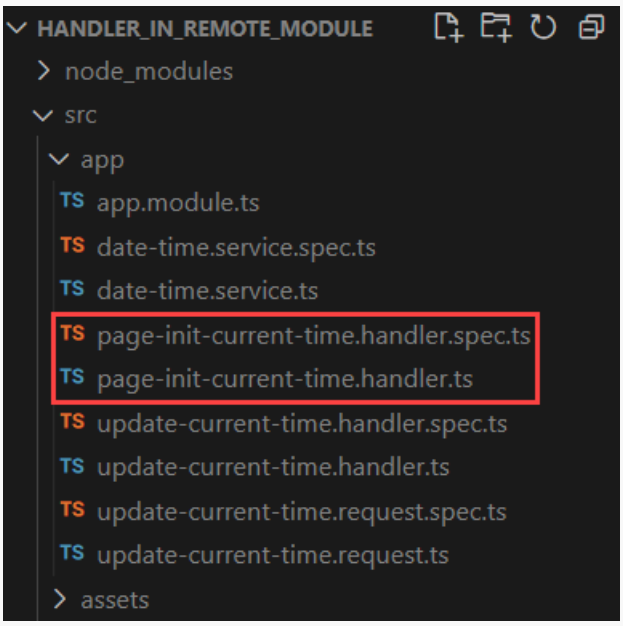
```
    type: 'usr.UpdateCurrentTimeHandler',
    requestType: 'crt.UpdateCurrentTimeRequest',
  })

export class UpdateCurrentTimeHandler extends BaseRequestHandler{
  constructor(private _dateTimeService: DateTimeService) {
    super();
  }
  public async handle(request: UpdateCurrentTimeRequest): Promise<unknown> {
    const result = await this._dateTimeService.getCurrentDateTime(request.dateTime);
    const dateTime = result.dateTime[request.dateTime] ?? 0
    /* Generate the value to display in the component of Freedom UI page. */
    request.$context['CurrentDateTime'] = dateTime;
    return dateTime;
  }
}
```

2. Create a custom handler that is executed when Creatio initializes a Freedom UI page.

   a. Create an Angular class in the project. To do this, run the `ng g class page-init-current-time.handler` command at the command line terminal of Microsoft Visual Studio Code.

   As a result, Microsoft Visual Studio Code will add the `PageInitCurrentTimeHandler` class files to the `src/app/` project directory.



   b. Implement the handler.

      a. Open the `page-init-current-time.handler.ts` file.

      b. Add the configuration object that declares the request handler.

         - Set the `type` property to `usr.PageInitCurrentTimeHandler`.

- Set the `requestType` property to `crt.HandleViewModelInitRequest`.
- Set the `scopes` property to `UsrRequests_FormPage`. The request handler is executed when Creatio initializes the Freedom UI page that has the `UsrRequests_FormPage` code.

f. Flag the `PageInitCurrentTimeHandler` class using the `CrtRequestHandler` decorator.

g. Inherit the `BaseRequestHandler` class from the `@creatio-devkit/common` library.

h. Implement the update of the current date and time when Creatio initializes the Freedom UI page.

i. Import the required functionality from the libraries into the class.

j. Save the file.

`page-init-current-time.handler.ts` file

```
/* Import the required functionality from the libraries. */
import { BaseRequest, BaseRequestHandler, CrtRequestHandler, HandlerChainService } from "@
import { UpdateCurrentTimeRequest } from "./update-current-time.request";

/* Add the CrtRequestHandler decorator to the PageInitCurrentTimeHandler class. */
@CrtRequestHandler({
  type: 'usr.PageInitCurrentTimeHandler',
  requestType: 'crt.HandleViewModelInitRequest',
  scopes: ['UsrRequests_FormPage'],
})

export class PageInitCurrentTimeHandler extends BaseRequestHandler{
  public async handle(request: BaseRequest): Promise<unknown> {
    /* Update the current date and time when Creatio initializes the Freedom UI page. */
    await HandlerChainService.instance.process({
      type: 'crt.UpdateCurrentTimeRequest',
      dateTime: 'dateTime',
      $context: request.$context
    } as UpdateCurrentTimeRequest);

    return this.next?.handle(request);
  }
}
```

3. Register the handlers.

a. Open the `app.module.ts` file.

b. Add the `UpdateCurrentTimeHandler` and `PageInitCurrentTimeHandler` handlers to the `requestHandlers` section in the `CrtModule` decorator.

c. Import the required functionality from the libraries into the class.

d. Save the file.

`app.module.ts` file

```
/* Import the required functionality from the libraries. */
import { CrtModule } from '@creatio-devkit/common';
import { PageInitCurrentTimeHandler } from './page-init-current-time.handler';
import { UpdateCurrentTimeHandler } from './update-current-time.handler';
...

@CrtModule({
  ...,
  /* Specify that UpdateCurrentTimeHandler and PageInitCurrentTimeHandler are request handler
  requestHandlers: [
    UpdateCurrentTimeHandler,
    PageInitCurrentTimeHandler
  ],
})
...
```

Complete source code of the `app.module.ts` file

4. Build the project. To do this, run the `npm run build` command at the command line terminal of Microsoft Visual Studio Code.

As a result, Microsoft Visual Studio Code will add the build to the `dist` directory of the Angular project. The build will have the `sdk_remote_module_package` name.

5. Add the request implemented using remote module to the Freedom UI page

1. Repeat steps 2-7 of the procedure to implement custom UI component using remote module.
2. Add a [*Button*] type component to the working area of the Freedom UI Designer.
3. Click the </> button in the action panel of the Freedom UI Designer. After you save the page settings, Creatio opens the source code of the Freedom UI page.
4. Change the `clicked` property value for the `RefreshCurrentDateTimeButton` element in the `viewConfigDiff` schema section to request configuration object.
   - Enter the `crt.UpdateCurrentTimeRequest` request name in the `request` property.
   - Enter the configuration object of parameters in the `params` property.

`viewConfigDiff` schema section

```
viewConfigDiff: /**SCHEMA_VIEW_CONFIG_DIFF*/[
    {
        "operation": "insert",
```

```
        "name": "RefreshCurrentDateTimeButton",
        "values": {
            ...,
            "clicked": {
                /* Bind the sending of the custom crt.UpdateCurrentTimeRequest request to the
                "request": "crt.UpdateCurrentTimeRequest",
                "params": {
                    "dateTime": "dateTime"
                }
            }
        },
        ...
    },
    ...
]/**SCHEMA_VIEW_CONFIG_DIFF*/,
```

Complete source code of the page schema

5. Click [*Save*] on the Client Module Designer's toolbar.

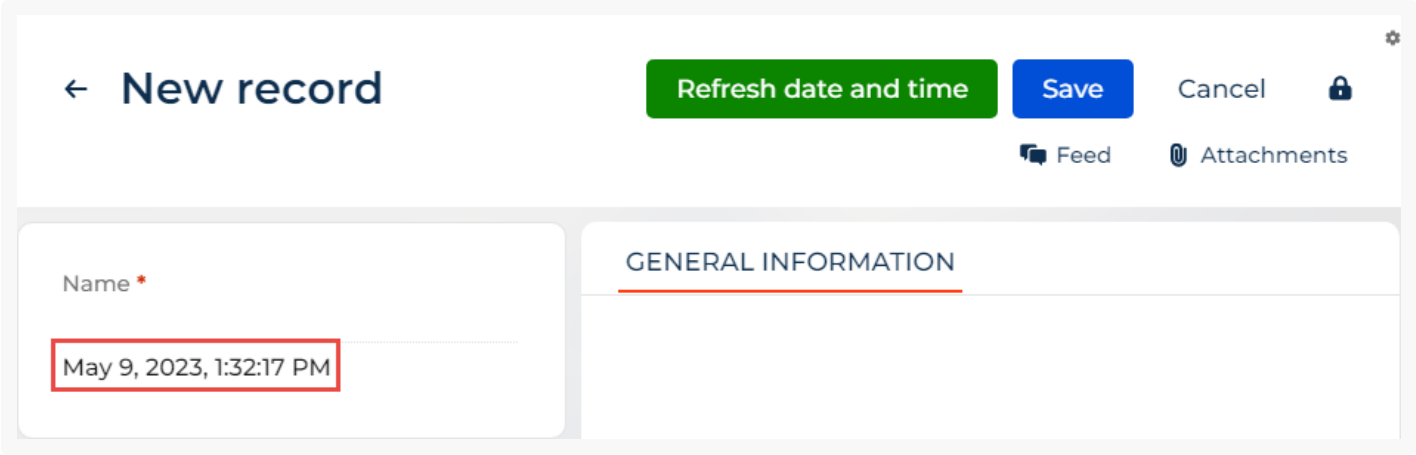
As a result, Creatio will add the request to the Freedom UI page.

Outcome of the example

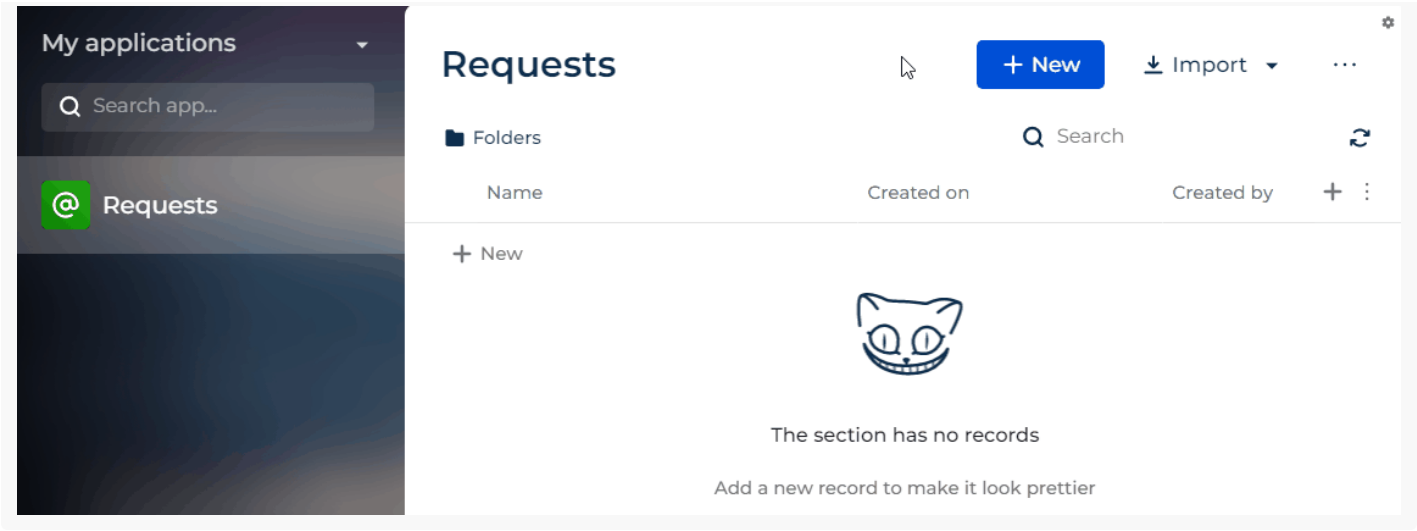
To **view the outcome of the example**:

1. Open the `Requests` app page and click [*Run app*].
2. Click [*New*] on the `Requests` app toolbar.
3. Click [*Refresh date and time*] on the request page toolbar.

As a result, Creatio will display current date and time on the request page.



Creatio will update the current date and time when you click the [*Refresh date and time*].



The handlers are implemented using the remote module created in Angular framework.