*(click)*="saveAndClose()"

*[background]*="

            'linear-gradient(89.75deg, #002b6e 0.23%, #2288cc 99.81%)'

          "

        >

        </ait-button>

        <ait-button

*\*ngIf*="!isChanged && mode === 'EDIT'"

*[id]*="'saveAndClose'"

*[width]*="'200px'"

*[title]*="'SAVE & CLOSE'"

*[style]*="'disabled'"

        ></ait-button>

      </div>

    </div>

  </div>

</div>

ァイル添付または');

  getReference = () => *this*.translateService.translate('参照');

  getMaxFileText = () => *this*.translateService.translate('ファイルまで添付できます。');

  getFileTypeText = () => *this*.translateService.translate('形式のファイルのみ添付できます。');

  check = () => {

    return *this*.getNote() || (*this*.getFileMaxUpload() || *this*.maxFiles) || (*this*.getFileTypeSup() || *this*.fileTypes)

  }

  sumSizeFiles = (*files*: *any*[]) => {

    return (*this*.fileRequest[0]?.size || 0) / (1024);

  }

  getSrc = (*file*) => {

    if (!*file*.isError) {

      return *this*.safelyURL(*file*.data\_base64, *file*.file\_type) ||

        '../../../../assets/images/file.svg'

    }

    return '../../../../assets/images/not-f.jpg'

  }

  handleErrorImage = (*file*) => {

*this*.errorImage = { ...*this*.errorImage, [*file*?.\_key]: true }

    // file.isError = true;

  }

<div *class*="input\_area">

          <ait-label *[label]*="'FILEs'"></ait-label>

          <label *class*="file-max"

            ><nb-icon *icon*="attach-outline"></nb-icon>Upload max 5 files and

            10MB/file</label

          >

          <ait-input-file

*[id]*="'file'"

*[maxSize]*="1000000"

*[maxFiles]*="5"

*[hasStatus]*="false"

*[placeholder]*="'Select file'"

*[isReset]*="isResetFile"

*[isClear]*="isClear"

*[fileKeys]*="userEducationInfo.controls['file'].value"

*(watchValue)*="

              takeFiles($event.value, 'userEducationInfo', 'file')

            "

*[isSubmit]*="isSubmit"

          ></ait-input-file>

        </div>

      </div>

    </div>

  </ait-card-content>

  <div *class*="sticky">

    <div *class*="action-row">

      <div *class*="block">

        <ait-button

*[id]*="'cancel'"

*[title]*="'CANCEL'"

*[background]*="

            'linear-gradient(89.75deg, #002b6e 0.23%, #2288cc 99.81%)'

          "

*(click)*="back()"

        ></ait-button>

        <ait-button

*\*ngIf*="mode === 'EDIT'"

*[id]*="'delete'"

*[width]*="'200px'"

*[title]*="'DELETE'"

*(click)*="onDelete()"

*[background]*="

            'linear-gradient(89.75deg, #002b6e 0.23%, #2288cc 99.81%)'

          "

        >

        </ait-button>

        <ait-button

*[id]*="'reset'"

*[width]*="'200px'"

*[title]*="'RESET'"

*(click)*="resetForm()"

*[background]*="

            'linear-gradient(89.75deg, #002b6e 0.23%, #2288cc 99.81%)'

          "

        >

        </ait-button>

        <ait-button

*\*ngIf*="mode === 'NEW'"

*[id]*="'saveAndContinue'"

*[title]*="'SAVE & CONTINUE'"

*(click)*="saveAndContinue()"

*[background]*="

            'linear-gradient(89.75deg, #002b6e 0.23%, #2288cc 99.81%)'

          "

        ></ait-button>

        <ait-button

*[id]*="'saveAndClose'"

*\*ngIf*="(isChanged && mode === 'EDIT') || mode === 'NEW'"

*[width]*="'200px'"

*[title]*="'SAVE & CLOSE'"

*(click)*="saveAndClose()"

*[background]*="

            'linear-gradient(89.75deg, #002b6e 0.23%, #2288cc 99.81%)'

          "

        >

        </ait-button>

        <ait-button

*\*ngIf*="!isChanged && mode === 'EDIT'"

*[id]*="'saveAndClose'"

*[width]*="'200px'"

*[title]*="'SAVE & CLOSE'"

*[style]*="'disabled'"

        ></ait-button>

      </div>

    </div>

  </div>

</div>

  getTime = (*time*: *number*) => {

    if (*time*) {

      return *this*.dayJSService.calculateDateTime(*time*);

    }

    return null;

  }

  getImage = (*file*: *any*, *isError* = false) => {

    if (!*isError*) {

      return *this*.safelyURL(*file*.data\_base64, *file*.file\_type)

    }

    return 'https://d30y9cdsu7xlg0.cloudfront.net/png/47682-200.png';

  }

  editFileName = (*file*) => {

    return `ait\_${*Date*.now()}\_${*file*?.name}`;

  }

  getFileCount = () => {

    return [...*this*.fileDatas, ...*this*.displayedFiles].length;

  }

  safelyURL = (*data*, *type*) => *this*.santilizer.bypassSecurityTrustUrl(`data:${*type*};base64, ${*data*}`);

  checkMaxSize = (*file*: *any*[]) => {

    return *this*.fileRequest.length > 0 ?  *this*.fileRequest[0]?.size <= *this*.maxSize \* 1024 : true;

  }

  checkMaxFile = () => {

    return *this*.fileDatas.length + *this*.displayedFiles.length < *this*.getFileMaxUpload();

  }

  getTextDelete = () => *this*.translateService.translate('c\_2002');

  getKB = (*bytes*: *number*) => {

    return *bytes* ? *bytes* / 1024 : null;

  }

  ngOnInit() {

    const settingFiles = ['FILE\_TYPE\_SUPPORT', 'FILE\_MAX\_UPLOAD', 'FILE\_MAX\_SIZE\_MB'];

*this*.masterDataService.find({

      parent\_code: *CLASS*.SYSTEM\_SETTING,

    }, {

      \_key: true,

      code: true,

      parent\_code: true,

      class: true,

      name: true

    }).then(*r* => {

      const settings = *r*.data.filter(*d* => settingFiles.includes(*d*?.code));

*this*.settings = settings.map((*s*: *any*) => ({ ...*s*, value: *s*?.name }));

      if (settings.length !== 0) {

        if (!*this*.maxSize) {

*this*.maxSize = *Number*(*this*.settings.find(*f* => *f*.code === 'FILE\_MAX\_SIZE\_MB')?.value);

        }

        if (!*this*.maxFiles) {

*this*.maxFiles = *Number*(*this*.getValueByCode('FILE\_MAX\_UPLOAD')?.value);

        }

        if (!*this*.fileTypes) {

*this*.fileTypes = *this*.getValueByCode('FILE\_TYPE\_SUPPORT')?.value;

        }

      }

    })

    if (*this*.fileKeys && *this*.fileKeys.length !== 0) {

*this*.fileUploadService.getFilesByFileKeys(*this*.fileKeys || []).then((*r*: *any*) => {

        if (*r*?.status === *RESULT\_STATUS*.OK) {

*this*.dataDisplayDf = *r*.data;

*this*.displayedFiles = *r*.data;

        }

      })

    }

  }

  getFileMaxUpload = () => {

    const maxfile = *this*.settings.find(*f* => *f*.code === 'FILE\_MAX\_UPLOAD');

    return *this*.maxFiles ? *this*.maxFiles : maxfile ? maxfile?.value : null;

  }

  getFileTypeSup = () => {

    const supfile = *this*.settings.find(*f* => *f*.code === 'FILE\_TYPE\_SUPPORT');

    return *this*.fileTypes ? *this*.fileTypes : supfile ? supfile?.value : null;

  }

  getMaxSizeFile = () => {

    return *this*.maxSize ? *this*.maxSize \* 1024 : 5000000;

  }

  getValueByCode = (*code*) => {

    return *this*.settings.find(*f* => *f*.code === *code*)?.value;

  }

  checkFileExt = (*file*) => {

    const check = AitAppUtils.checkFileExt(*this*.fileTypes || FILE\_TYPE\_SUPPORT\_DEFAULT, *file*);

    if (check.status !== 1) {

*this*.messageErrorFileSp = *this*.translateService.getMsg('E0012')

        .replace('{0}', *this*.fileTypes || *this*.getValueByCode('FILE\_TYPE\_SUPPORT') || FILE\_TYPE\_SUPPORT\_DEFAULT);

      return false;

    }

    return check?.status === 1;

  }

  /\*\*

   \* on file drop handler

   \*/

  onFileDropped(*$event*) {

*this*.fileBrowseHandler(*$event*)

*this*.styleHover = {

      background: 'rgba(0,0,0,0.15)'

    }

  }

  onFileDragLeave() {

*this*.styleHover = {

    }

  }

  /\*\*

   \* handle file from browsing

   \*/

  fileBrowseHandler(*files*) {

    if (*files* && *files*[0]) {

*this*.loading = true;

      const FR = **new** *FileReader*();

      FR.onload = (*e*: *any*) => {

        const index = *e*.target.result.indexOf(',');

*this*.currentBase64 = *e*.target.result.slice(index + 1);

*this*.prepareFilesList(*files*);

      }

      FR.readAsDataURL(*files*[0]);

    }

  }

  /\*\*

   \* Delete file from files list

   \* @param *index* (File index)

   \*/

  deleteFile(*file*?: *any*, *index*?: *number*, *isSubmit* = false) {

    if (*isSubmit*) {

*this*.fileUploadService.removeFile(*this*.deletedKeys).then(*r* => {

        if (*r*.status === *RESULT\_STATUS*.OK) {

*this*.checkReq();

*this*.checkCommon()

        }

      })

    }

    else {

*this*.deletedKeys = [...*this*.deletedKeys, { \_key: *file*?.\_key }];

*this*.savedData = *this*.savedData.filter(*s* => *s*.\_key !== *file*?.\_key)

*this*.files.splice(*index*, 1);

*this*.fileDatas = *this*.fileDatas.filter(*f* => *f*.\_key !== *file*?.\_key);

*this*.displayedFiles = *this*.displayedFiles.filter(*f* => *f*.\_key !== *file*?.\_key);

*this*.watchValue.emit({ value: [...*this*.fileDatas, ...*this*.displayedFiles] });

*this*.checkReq();

*this*.checkCommon()

    }

  }

  /\*\*

   \* Simulate the upload process

   \*/

  uploadFilesSimulator(*\_key*: *string*) {

    const file = *this*.fileDatas.find(*f* => *f*.\_key === *\_key*);

    setTimeout(() => {

      const progressInterval = setInterval(() => {

        if (file.progress === 100) {

          clearInterval(progressInterval);

        } else {

          file.progress += 5;

        }

      }, 10);

    }, 500);

  }

  checkCommon = () => {

*this*.messageErrorFileSp = '';

    if (!*this*.checkMaxFile()) {

*this*.messageErrorFileSp =

*this*.translateService.getMsg('E0155').replace('{0}', *this*.getFileMaxUpload().toString());

      return false;

    }

    else if (!*this*.checkMaxSize(*this*.fileRequest)) {

*this*.messageErrorFileSp =

*this*.translateService.getMsg('E0157').replace('{0}', *this*.formatBytes(*this*.getMaxSizeFile(), 2).toString());

      return false;

    }

    else {

      return true;

    }

  }

  /\*\*

   \* Convert Files list to normal array list

   \* @param *files* (Files List)

   \*/

  prepareFilesList(*files*: *Array*<*any*>) {

*this*.messageErrorFileSp = '';

    const fileReq = [];

    for (const item of *files*) {

      fileReq.push({

        type: item.type,

        name: *this*.editFileName(item),

        size: item.size

      })

    }

*this*.fileRequest = fileReq;

    // this.fileDatas = [...this.fileDatas, files[files.length - 1]];

*this*.files = *this*.files = [*files*[*files*.length - 1]];

    if (*this*.checkCommon()) {

*this*.files = [*files*[*files*.length - 1]];

      if (*this*.checkFileExt(fileReq[0])) {

        setTimeout(() => {

          const { type, ...objKeys } = *this*.fileRequest[0];

          const data = [

            {

              ...objKeys,

              file\_type: type,

              company: *this*.company,

              user\_id: AitAppUtils.getUserId(),

              data\_base64: *this*.currentBase64,

              \_key: Guid.create().toString()

            }

          ]

*this*.savedData = [...*this*.savedData, ...data];

*this*.fileDatas = [...*this*.fileDatas, { ...data[0], progress: 0 }];

*this*.watchValue.emit({ value: [...*this*.fileDatas, ...*this*.displayedFiles] });

*this*.fileDatas.forEach((*file*, *index*) => {

*this*.uploadFilesSimulator(*file*.\_key);

          })

*this*.loading = false;

*this*.checkReq();

        }, 400)

      }

    }

    setTimeout(() => {

*this*.loading = false;

    }, 100)

  }

  checkReq = () => {

*this*.componentErrors = [];

    if (*this*.required) {

      if ([...*this*.displayedFiles, ...*this*.fileDatas].length === 0) {

        const err = *this*.translateService.getMsg('E0001').replace('{0}', *this*.getTitle());

*this*.isError = true;

*this*.componentErrors = [err];

*this*.onError.emit({ isValid: false });

      }

      else {

*this*.isError = false;

*this*.onError.emit({ isValid: true });

      }

    }

    else {

*this*.isError = false;

*this*.onError.emit({ isValid: true });

    }

  }

  /\*\*

   \* format bytes

   \* @param *bytes* (File size in bytes)

   \* @param *decimals* (Decimals point)

   \*/

  formatBytes(*bytes*, *decimals*?) {

    if (*bytes* === 0) {

      return '0 Bytes';

    }

    const k = 1024;

    const dm = *decimals* <= 0 ? 0 : *decimals* || 2;

    const sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'];

    const i = Math.floor(Math.log(*bytes*) / Math.log(k));

    return parseFloat((*bytes* / Math.pow(k, i)).toFixed(dm)) + ' ' + sizes[i];

  }

  // eslint-disable-next-line no-prototype-builtins

  hasProperty = (*prop*: *string*, *obj*: *any*) => (obj || {}).hasOwnProperty(prop || '');

  async submitMultipleForm() {

    const data = *this*.savedData.map(*m* => {

      const { type, ...objKeys } = m;

      return {

        ...objKeys,

        company: *this*.company,

        user\_id: AitAppUtils.getUserId(),

      }

    });

    try {

      data.forEach(async (*file*: *any*) => {

        delete file.progress;

        await *this*.fileUploadService.uploadFile(file);

      });

    } catch (err) {

      return {

        status: 0

      }

    }

  }

}

  /\*\*

   \* on file drop handler

   \*/

  onFileDropped(*$event*) {

*this*.fileBrowseHandler(*$event*)

*this*.styleHover = {

      background: 'rgba(0,0,0,0.15)'

    }

  }

  onFileDragLeave() {

*this*.styleHover = {

    }

  }

  /\*\*

   \* handle file from browsing

   \*/

  fileBrowseHandler(*files*) {

    if (*files* && *files*[0]) {

*this*.loading = true;

      const FR = **new** *FileReader*();

      FR.onload = (*e*: *any*) => {

        const index = *e*.target.result.indexOf(',');

*this*.currentBase64 = *e*.target.result.slice(index + 1);

*this*.prepareFilesList(*files*);

      }

      FR.readAsDataURL(*files*[0]);

    }

  }

  /\*\*

   \* Delete file from files list

   \* @param *index* (File index)

   \*/

  deleteFile(*file*?: *any*, *index*?: *number*, *isSubmit* = false) {

    if (*isSubmit*) {

*this*.fileUploadService.removeFile(*this*.deletedKeys).then(*r* => {

        if (*r*.status === *RESULT\_STATUS*.OK) {

*this*.checkReq();

*this*.checkCommon()

        }

      })

    }

    else {

*this*.deletedKeys = [...*this*.deletedKeys, { \_key: *file*?.\_key }];

*this*.savedData = *this*.savedData.filter(*s* => *s*.\_key !== *file*?.\_key)

*this*.files.splice(*index*, 1);

*this*.fileDatas = *this*.fileDatas.filter(*f* => *f*.\_key !== *file*?.\_key);

*this*.displayedFiles = *this*.displayedFiles.filter(*f* => *f*.\_key !== *file*?.\_key);

*this*.watchValue.emit({ value: [...*this*.fileDatas, ...*this*.displayedFiles] });

*this*.checkReq();

*this*.checkCommon()

    }

  }

  /\*\*

   \* Simulate the upload process

   \*/

  uploadFilesSimulator(*\_key*: *string*) {

    const file = *this*.fileDatas.find(*f* => *f*.\_key === *\_key*);

    setTimeout(() => {

      const progressInterval = setInterval(() => {

        if (file.progress === 100) {

          clearInterval(progressInterval);

        } else {

          file.progress += 5;

        }

      }, 10);

    }, 500);

  }

  checkCommon = () => {

*this*.messageErrorFileSp = '';

    if (!*this*.checkMaxFile()) {

*this*.messageErrorFileSp =

*this*.translateService.getMsg('E0155').replace('{0}', *this*.getFileMaxUpload().toString());

      return false;

    }

    else if (!*this*.checkMaxSize(*this*.fileRequest)) {

*this*.messageErrorFileSp =

*this*.translateService.getMsg('E0157').replace('{0}', *this*.formatBytes(*this*.getMaxSizeFile(), 2).toString());

      return false;

    }

    else {

      return true;

    }

  }

  /\*\*

   \* Convert Files list to normal array list

   \* @param *files* (Files List)

   \*/

  prepareFilesList(*files*: *Array*<*any*>) {

*this*.messageErrorFileSp = '';

    const fileReq = [];

    for (const item of *files*) {

      fileReq.push({

        type: item.type,

        name: *this*.editFileName(item),

        size: item.size

      })

    }

*this*.fileRequest = fileReq;

    // this.fileDatas = [...this.fileDatas, files[files.length - 1]];

*this*.files = *this*.files = [*files*[*files*.length - 1]];

    if (*this*.checkCommon()) {

*this*.files = [*files*[*files*.length - 1]];

      if (*this*.checkFileExt(fileReq[0])) {

        setTimeout(() => {

          const { type, ...objKeys } = *this*.fileRequest[0];

          const data = [

            {

              ...objKeys,

              file\_type: type,

              company: *this*.company,

              user\_id: AitAppUtils.getUserId(),

              data\_base64: *this*.currentBase64,

              \_key: Guid.create().toString()

            }

          ]

*this*.savedData = [...*this*.savedData, ...data];

*this*.fileDatas = [...*this*.fileDatas, { ...data[0], progress: 0 }];

*this*.watchValue.emit({ value: [...*this*.fileDatas, ...*this*.displayedFiles] });

*this*.fileDatas.forEach((*file*, *index*) => {

*this*.uploadFilesSimulator(*file*.\_key);

          })

*this*.loading = false;

*this*.checkReq();

        }, 400)

      }

    }

    setTimeout(() => {

*this*.loading = false;

    }, 100)

  }

  checkReq = () => {

*this*.componentErrors = [];

    if (*this*.required) {

      if ([...*this*.displayedFiles, ...*this*.fileDatas].length === 0) {

        const err = *this*.translateService.getMsg('E0001').replace('{0}', *this*.getTitle());

*this*.isError = true;

*this*.componentErrors = [err];

*this*.onError.emit({ isValid: false });

      }

      else {

*this*.isError = false;

*this*.onError.emit({ isValid: true });

      }

    }

    else {

*this*.isError = false;

*this*.onError.emit({ isValid: true });

    }

  }

  /\*\*

   \* format bytes

   \* @param *bytes* (File size in bytes)

   \* @param *decimals* (Decimals point)

   \*/

  formatBytes(*bytes*, *decimals*?) {

    if (*bytes* === 0) {

      return '0 Bytes';

    }

    const k = 1024;

    const dm = *decimals* <= 0 ? 0 : *decimals* || 2;

    const sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'];

    const i = Math.floor(Math.log(*bytes*) / Math.log(k));

    return parseFloat((*bytes* / Math.pow(k, i)).toFixed(dm)) + ' ' + sizes[i];

  }

  // eslint-disable-next-line no-prototype-builtins

  hasProperty = (*prop*: *string*, *obj*: *any*) => (obj || {}).hasOwnProperty(prop || '');

  async submitMultipleForm() {

    const data = *this*.savedData.map(*m* => {

      const { type, ...objKeys } = m;

      return {

        ...objKeys,

        company: *this*.company,

        user\_id: AitAppUtils.getUserId(),

      }

    });

    try {

      data.forEach(async (*file*: *any*) => {

        delete file.progress;

        await *this*.fileUploadService.uploadFile(file);

      });

    } catch (err) {

      return {

        status: 0

      }

    }

  }

}

/\* eslint-disable @angular-eslint/no-output-on-prefix \*/

/\* eslint-disable @typescript-eslint/no-unused-vars \*/

/\* eslint-disable @typescript-eslint/no-explicit-any \*/

/\* eslint-disable @typescript-eslint/no-inferrable-types \*/

// eslint-disable-next-line @nrwl/nx/enforce-module-boundaries

import { objKeys, RESULT\_STATUS } from '@ait/shared';

import { Component, EventEmitter, Input, OnChanges, OnInit, Output, SimpleChanges, ViewChild } from '@angular/core';

import { DomSanitizer } from '@angular/platform-browser';

import { select, Store } from '@ngrx/store';

import { FILE\_TYPE\_SUPPORT\_DEFAULT, MAX\_FILE\_DEFAULT } from '../../@constant';

import { AitBinaryDataService } from '../../services/ait-binary-data.service';

import { AitDayJSService } from '../../services/ait-dayjs.service';