Identity Hungary Informatikai Korlátolt Felelősségű Társaság

Profilunk a **PHP fejlesztés** és web alapú termékek előállítása.

6722 Szeged, Attila utca 11. 1. em. 1.
http://identity-hungary.hu
+36-62 452-239

GarNet API

Az API elérési útja

Az API elérési útját a **.env** fájlban található `REACT_APP_API_URL` változó határozza meg.

```
const API_URL = process.env.REACT_APP_API_URL
              // REACT_APP_API_URL=//api.garnet.identity-hungary.hu/api
```

Általános HTTP headers

```
const headers = {
  "Accept": "application/json",
  "X-Requested-With": "XMLHttpRequest",
  "Content-Type": "application/json",
  "X-localization": browserLanguage
}
```

Authorization-val rendelkező HTTP headers

Az Authorization-nak tartalmaznia kell a felhasználó `TOKEN` típusát és az `ACCESSTOKEN` -t.

```
const headersWidthAuth = {
  "Authorization": `${tokenType} ${accessToken}`,
  "Accept": "application/json",
  "X-Requested-With": "XMLHttpRequest",
  "Content-Type": "application/json",
  "X-localization": userDefaultLanguage
}
```

SecureFetch

A `secureFetch` függvény a megadott végpontra a megfelelő paraméterekkel végrehajtja a lekérdezést. Sikeres lekérdezés esetén a kapott választ a státuszkódja alapján kiértékeli és visszatér a válasszal. Ha a lekérdezés sikertelen a függvény visszatérési értéke az error objektum.

```
export const secureFetch = (url, method, secureHeaders, data) => {
  new Promise((resolve, reject) => {
    return fetch(`${API_URL}${url}`, {
      method: method || 'GET',
      body: data,
      headers: secureHeaders
    }).then(response => {
      if (response.ok) {
        return resolve(response)
      } else {
        switch (response.status) {
        case 401:
          if (url !== '/user/login') {
            console.log(`UNAUTHORIZED ${response.status}`)
            localStorage.clear()
            window.location.pathname = '/login'
          }
          break
        case 404:
          console.log(`Object not found ${response.status}`)
          break
        case 500:
          console.log(`Internal server error ${response.status}`)
          break
        case 422:
          console.log(`Unprocessable Entity ${response.status}`)
          break
        default:
          console.log(`Some error occured ${response.status}`)
          break
```

```
31          }
32          return reject(response)
33        }
34      })
35        .catch(error => {
36          return reject(error)
37        })
38    })
39  )
```

Vásárló lekérdezések

Termékkategóriák lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`params`	`withPaginate, withRelation, withTrash`
`URL`	`/public/productCategories${params}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getProducts = (params) => (
  secureFetch(
    `/public/productCategories${params}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error => error)
)
```

Függvény használata

```
import { getProducts } from './api/customerQueries'

getProducts(params).then(response => response).catch(error => error)
```

Eladók lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`params`	`withPaginate, withRelation, withTrash`
`URL`	`/public/sellers${params}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getSellers = (params) => {
  secureFetch(
    `/public/sellers${params}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error =>  error)
}
```

Függvény használata

```
import { getSellers } from './api/customerQueries'

getSellers(params).then(response => response).catch(error => error)
```

Termékkategória lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`id`	`termékkategória id`
`URL`	`/public/productCategories/${id}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getCategoryById = (id) => {
  secureFetch(
    `/public/productCategories/${id}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error =>  error)
}
```

Függvény használata

```
import { getCategoryById } from './api/customerQueries'

getCategoryById(id).then(response => response).catch(error => error)
```

Számlák lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`params`	`withPaginate, withRelation, withTrash`
`URL`	`/public/customer/${user_id}/invoices${params}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getInvoices = (user_id, params) => {
  secureFetch(
    `/public/customer/${user_id}/invoices${params}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { getInvoices } from './api/customerQueries'

getInvoices(user_id, params).then(response => response).catch(error => error)
```

Számla lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`invoice_id`	`számla id`
`params`	`withPaginate, withRelation, withTrash`
`URL`	`/public/customer/${user_id}/invoices${params}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getInvoiceByID = (user_id, invoice_id, params) => (
  secureFetch(
    `/public/customer/${user_id}/invoices/${invoice_id}${params}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error => error)
)
```

Függvény használata

```
import { getInvoiceByID } from './api/customerQueries'

getInvoiceByID(user_id, invoice_id, params).then(response => response).catch(error => error)
```

Számla létrehozása

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`URL`	`/public/customer/${user_id}/invoices`
`method`	`POST`
`headers`	`unique header`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "seller_id"     : "required|integer",
  "invoice_no"    : "required",
  "user_id"       : "required|integer",
  "purchased"     : "required",
  "ap_no"         : "sometimes",
  "files"         : "required"
}
```

Függvény deklarációja

```
export const createInvoice = (data, user_id) => {
  let formData = new FormData()

  for (let key in data) {
    if (key === 'files') {
      data[key].forEach(file => formData.append('files[]', file, file.name))
    } else {
      formData.append(key, data[key])
    }
  }
  return secureFetch(
    `/public/customer/${user_id}/invoices`,
    'POST',
    {
      'Authorization': `${localStorage.getItem('token_type')} ${localStorage.getItem('access_token')}`,
      'X-Requested-With': 'XMLHttpRequest',
      'X-localization': localStorage.getItem('default_languages')
    },
    formData
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { createInvoice } from './api/customerQueries'

createInvoice(data, user_id).then(response => response).catch(error => error)
```

Számla módosítása

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`invoice_id`	`számla id`
`URL`	`/public/customer/${user_id}/invoices/${invoice_id}`
`method`	`PUT`
`headers`	`headersWidthAuth`
`body`	`data`

BODY paraméter értéke

```
const data = {
   "seller_id"     : "required|integer",
   "invoice_id"    : "required|integer",
   "invoice_no"    : "required",
   "user_id"       : "required|integer",
   "purchased"     : "required",
   "ap_no"         : "sometimes"
}
```

Függvény deklarációja

```
export const updateInvoice = (user_id, invoice_id, data) => {
  secureFetch(
    `/public/customer/${user_id}/invoices/${invoice_id}`,
    'PUT',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
    JSON.stringify(data)
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { updateInvoice } from './api/customerQueries'

updateInvoice(user_id, invoice_id, data).then(response => response).catch(error => error)
```

Számla törlése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`invoice_id`	`számla id`
`URL`	`/public/customer/${user_id}/invoices/${invoice_id}`
`method`	`DELETE`
`headers`	`headersWidthAuth`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "invoice_id"   : "required|integer",
  "user_id"      : "required|integer"
}
```

Függvény deklarációja

```
export const deleteInvoice = (user_id, invoice_id, data) => {
  secureFetch(
    `/public/customer/${user_id}/invoices/${invoice_id}`,
    'DELETE',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
    JSON.stringify(data)
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { deleteInvoice } from './api/customerQueries'

deleteInvoice(user_id, invoice_id, data).then(response => response).catch(error => error)
```

Fájl lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`file_id`	`fájl id`
`disposition`	`` `0 ``
`URL`	`/public/customer/${user_id}/invoices${params}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getFileDownload = (user_id, file_id, disposition) => (
  secureFetch(
    `/public/customer/${user_id}/uploads/${file_id}/download/${disposition}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error => error)
)
```

Függvény használata

```
import { getFileDownload } from './api/customerQueries'

getFileDownload(user_id, file_id, disposition).then(response => response).catch(error => error)
```

Fájl feltöltése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`URL`	`/public/customer/${user_id}/uploads`
`method`	`POST`
`headers`	`unique header`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "invoice_id"        : "required",
  "invoice_item_id"   : "sometimes",
  "user_id"           : "required|integer",
  "files"             : "required"
}
```

Függvény deklarációja

```
export const uploadFiles = (user_id, data) => {
  let formData = new FormData()

  for (let key in data) {
    if (key === 'files') {
      data[key].forEach(file => formData.append('files[]', file, file.name))
    } else {
      formData.append(key, data[key])
    }
  }
  return secureFetch(
    `/public/customer/${user_id}/uploads`,
    'POST',
    {
      'Authorization': `${localStorage.getItem('token_type')}
```

```
    ${localStorage.getItem('access_token')}`,
16        'X-Requested-With': 'XMLHttpRequest',
17        'X-localization': localStorage.getItem('default_languages')
18      },
19      formData
20    ).then(res => res)
21     .catch(error => error)
22  }
```

Függvény használata

```
1  import { uploadFiles } from './api/customerQueries'
2
3  uploadFiles(user_id, data).then(response => response).catch(error => error)
```

Fájl törlése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`file_id`	`fájl id`
`URL`	`/public/customer/${user_id}/uploads/${file_id}`
`method`	`DELETE`
`headers`	`headersWidthAuth`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "file_id"   : "required|integer",
  "user_id"   : "required|integer",
}
```

Függvény deklarációja

```
export const deleteFile = (user_id, file_id, data) => {
  secureFetch(
    `/public/customer/${user_id}/uploads/${file_id}`,
    'DELETE',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
    JSON.stringify(data)
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { deleteFile } from './api/customerQueries'

deleteFile(user_id, file_id, data).then(response => response).catch(error => error)
```

Tremékek lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`params`	`withPaginate, withRelation, withTrash`
`URL`	`/public/customer/${user_id}/items${params}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getItems = (user_id, params) => {
  secureFetch(
    `/public/customer/${user_id}/items${params}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { getItems } from './api/customerQueries'

getItems(user_id, params).then(response => response).catch(error => error)
```

Számlához tartozó tremékek lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`invoice_id`	`számla id`
`params`	`withPaginate, withRelation, withTrash`
`URL`	`/public/customer/${user_id}/invoices/${invoice_id}/items${params}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getInvoiceItems = (user_id, invoice_id, params) => {
  secureFetch(
    `/public/customer/${user_id}/invoices/${invoice_id}/items${params}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { getInvoiceItems } from './api/customerQueries'

getInvoiceItems(user_id, invoice_id, params).then(response => response).catch(error => error)
```

Új tremék létrehozása

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`invoice_id`	`számla id`
`URL`	`/public/customer/${user_id}/invoices/${invoice_id}/items`
`method`	`POST`
`headers`	`unique headers`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "invoice_id"              : "required|integer",
  "name"                    : "required|string",
  "product_category_id"     : "required|integer",
  "product_subcategory_id"  : "required|integer",
  "manufacture"             : "required|string",
  "type_no"                 : "required|string",
  "barcode"                 : "sometimes|string",
  "price"                   : "required|integer",
  "warranty_in_month"       : "required|integer",
  "warranty"                : "required",
  "has_extra_form_item"     : "required",
}
```

Függvény deklarációja

```
export const createInvoiceItem = (user_id, invoice_id, data) => {
  let formData = new FormData()

  for (let key in data) {
    if (key === 'files') {
      data[key].forEach(file => formData.append('files[]', file, file.name))
    } else {
      formData.append(key, data[key])
    }
  }
  return secureFetch(
    `/public/customer/${user_id}/invoices/${invoice_id}/items`,
    'POST',
    {
      'Authorization': `${localStorage.getItem('token_type')} ${localStorage.getItem('access_token')}`,
      'X-Requested-With': 'XMLHttpRequest',
      'X-localization': localStorage.getItem('default_languages')
    },
    formData
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { createInvoiceItem } from './api/customerQueries'

createInvoiceItem(user_id, invoice_id, data).then(response => response).catch(error => error)
```

Tremék módosítása

secureFetch bemeneti értékei

Paraméter	Érték
user_id	felhasználó id
invoice_id	számla id
item_id	termék id
URL	/public/customer/${user_id}/invoices/${invoice_id}/items/${item_id}
method	PUT
headers	headersWidthAuth
body	data

BODY paraméter értéke

```
const data = {
  "invoice_id"              : "required|integer",
  "name"                    : "required|string",
  "product_category_id"     : "required|integer",
  "product_subcategory_id"  : "required|integer",
  "manufacture"             : "required|string",
  "type_no"                 : "required|string",
  "barcode"                 : "nullable|string",
  "price"                   : "required|integer",
  "warranty_in_month"       : "required|integer",
  "warranty"                : "required|boolean",
  "has_extra_form_item"     : "required"
}
```

Függvény deklarációja

```
export const updateInvoiceItem = (user_id, invoice_id, item_id, data)=> {
  secureFetch(
    `/public/customer/${user_id}/invoices/${invoice_id}/items/${item_id}`,
    'PUT',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
    JSON.stringify(data)
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { updateInvoiceItem } from './api/customerQueries'

updateInvoiceItem(user_id, invoice_id, item_id, data).then(response => response).catch(error =>
error)
```

Tremék törlése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`invoice_id`	`számla id`
`item_id`	`termék id`
`URL`	`/public/customer/${user_id}/invoices/${invoice_id}/items/${item_id}`
`method`	`DELETE`
`headers`	`headersWidthAuth`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "user_id"       : "required|integer",
  "invoice_id"    : "required|integer",
  "item_id"       : "required|integer"
}
```

Függvény deklarációja

```
export const deleteInvoiceItem = (user_id, invoice_id, item_id, data) => {
  secureFetch(
    `/public/customer/${user_id}/invoices/${invoice_id}/items/${item_id}`,
    'DELETE',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
    JSON.stringify(data)
  ).then(res => res)
   .catch(error =>  error)
}
```

Függvény használata

```
import { deleteInvoiceItem } from './api/customerQueries'

deleteInvoiceItem(user_id, invoice_id, item_id, data).then(response => response).catch(error =>
error)
```

Tremék lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`invoice_id`	`számla id`
`item_id`	`termék id`
`params`	`withRelation`
`URL`	`/public/customer/${user_id}/invoices/${invoice_id}/items/${item_id}${params}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getItemById = (user_id, invoice_id, item_id, params) => (
  secureFetch(
    `/public/customer/${user_id}/invoices/${invoice_id}/items/${item_id}${params}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error => error)
)
```

Függvény használata

```
import { getItemById } from './api/customerQueries'

getItemById(user_id, invoice_id, item_id, params).then(response => response).catch(error => error)
```

Extra form elem lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`sub_cat_id`	`termék alkategória id`
`URL`	`/public/extraFormItems/${sub_cat_id}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getExtraFormItems = (sub_cat_id) => {
  secureFetch(
    `/public/extraFormItems/${sub_cat_id}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { getExtraFormItems } from './api/customerQueries'

getExtraFormItems(sub_cat_id).then(response => response).catch(error => error)
```

Vásárló adatainak lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`URL`	`/public/customer/${user_id}/profile`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const customerProfilData = (user_id) => (
  secureFetch(
    `/public/customer/${user_id}/profile`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    )
  ).then(res => res)
   .catch(error => error)
)
```

Függvény használata

```
import { customerProfilData } from './api/customerQueries'

customerProfilData(user_id).then(response => response).catch(error => error)
```

Vásárló adatainak módosítása

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`URL`	`/public/customer/${user_id}/profile`
`method`	`PUT`
`headers`	`headersWidthAuth`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "user_id"                     : "required|integer",
  "email"                       : "sometimes|email|confirmed",
  "password"                    : "sometimes|string|min:8|confirmed|regex:/^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{6,}$/",
  "default_language"            : "required|string",
  "first_name"                  : "required|string",
  "last_name"                   : "required|string",
  "wired_phone"                 : "sometimes|string",
  "mobile_phone"                : "required|string",
  "expire_alert_in_day"         : "required|integer",
  "home_address_postal_code"    : "required|integer",
  "home_address_city"           : "required|string",
  "home_address_street"         : "required|string",
  "home_address_street_number"  : "required|string",
  "home_address_door_number"    : "sometimes|string",
  "postal_address_postal_code"  : "required|integer",
  "postal_address_city"         : "required|string",
  "postal_address_street"       : "required|string",
  "postal_address_street_number": "required|string",
  "postal_address_door_number"  : "sometimes|string"
}
```

Függvény deklarációja

```
export const updateCustomerProfilData = ( user_id, data) => {
  secureFetch(
    `/public/customer/${user_id}/profile`,
    'PUT',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
    JSON.stringify(data)
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { updateCustomerProfilData } from './api/customerQueries'

updateCustomerProfilData(user_id, data).then(response => response).catch(error => error)
```

Lejáró termékek lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`params`	`withPaginate, withRelation`
`URL`	`/public/customer/${user_id}/expiredItems${params}`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const getExpiredItems = (user_id, params) => {
  secureFetch(
    `/public/customer/${user_id}/expiredItems${params}`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { getExpiredItems } from './api/customerQueries'

getExpiredItems(user_id, params).then(response => response).catch(error => error)
```

Figyelmeztetések lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`URL`	`/public/customer/${user_id}/notifications/expire`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const notifications = (user_id) => (
  secureFetch(
    `/public/customer/${user_id}/notifications/expire`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
  ).then(res => res)
   .catch(error => error)
)
```

Függvény használata

```
import { notifications } from './api/customerQueries'

notifications(user_id).then(response => response).catch(error => error)
```

Figyelmeztetések státuszának módosítása

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`notification_id`	`figyelmeztetés id`
`URL`	`/public/customer/${user_id}/notifications/${notification_id}/status`
`method`	`POST`
`headers`	`headersWidthAuth`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "read_at" => "required"
}
```

Függvény deklarációja

```
export const updateNotification = (user_id, notification_id, data) => {
  secureFetch(
    `/public/customer/${user_id}/notifications/${notification_id}/status`,
    'POST',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
    JSON.stringify(data)
  ).then(res => res)
   .catch(error => error)
)
```

Függvény használata

```
import { notifications } from './api/customerQueries'

notifications(user_id).then(response => response).catch(error => error)
```

Figyelmeztető emailek lekérése

secureFetch bemeneti értékei

Paraméter	Érték
`user_id`	`felhasználó id`
`URL`	`/public/customer/${user_id}/notifications/email`
`method`	`GET`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const email = (user_id) => (
  secureFetch(
    `/public/customer/${user_id}/notifications/email`,
    'GET',
    headersWidthAuth(
      localStorage.getItem('token_type'),
      localStorage.getItem('access_token'),
      localStorage.getItem('default_languages')
    ),
  ).then(res => res)
   .catch(error => error)
)
```

Függvény használata

```
import { email } from './api/customerQueries'

email(user_id).then(response => response).catch(error => error)
```

Általános lekérdezések

Bejelentkezés

secureFetch bemeneti értékei

Paraméter	Érték
`URL`	`/user/login`
`method`	`POST`
`headers`	`headers`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "email"         : "required|string|email",
  "password"      : "required|string",
  "remember_me"   : "boolean"
}

```

Függvény deklarációja

```
export const userLogin = (data) => {
  secureFetch(
    '/user/login',                  // url
    'POST',                         // methond
    headers,                        // headers
    JSON.stringify(data)            // data
  ).then(res => res)
   .catch(error =>  error)
}
```

Függvény használata

```
import { userLogin } from './api/queries'

userLogin(data).then(response => response).catch(error => error)
```

Kijelentkezés

secureFetch bemeneti értékei

Paraméter	Érték
`URL`	`/user/logout`
`method`	`POST`
`headers`	`headersWidthAuth`

Függvény deklarációja

```
export const userLogout = () => {
  secureFetch(
    '/user/logout',                          // url
    'POST',                                  // method
    headersWidthAuth(
      localStorage.getItem('token_type'),    // token type
      localStorage.getItem('access_token'),  // access token
      localStorage.getItem('default_languages')  // felhasználó nyelvi beállítása
    )
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { userLogout } from './api/queries'

userLogout().then(response => response).catch(error => error)
```

Új jelszó igénylése

secureFetch bemeneti értékei

Paraméter	Érték
URL	/user/password/reset/create
method	POST
headers	headers
body	data

BODY paraméter értéke

```
const data = {
  "email": "required|string"
}
```

Függvény deklarációja

```
export const resetPasswordRequest = (data) => {
  secureFetch(
    '/user/password/reset/create',
    'POST',
    headers,
    JSON.stringify(data)
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { resetPasswordRequest } from './api/queries'

resetPasswordRequest(data).then(response => response).catch(error => error)
```

Felhasználó aktiváció

secureFetch bemeneti értékei

Paraméter	Érték
`URL`	`/user/signup/activate/${token}`
`method`	`GET`
`headers`	`headers`

Függvény deklarációja

```
export const userActivate = (token) => {
  secureFetch(
    `/user/signup/activate/${token}`,
    'GET',
    headers
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { userActivate } from './api/queries'

userActivate(token).then(response => response).catch(error => error)
```

Felhasználó jelszavának törlése token alapján

secureFetch bemeneti értékei

Paraméter	Érték
URL	/user/password/reset/find/${token}
method	GET
headers	headers

Függvény deklarációja

```
export const findByToken = (token) => {
  secureFetch(
    `/user/password/reset/find/${token}`,
    'GET',
    headers
    ).then(res => res)
     .catch(error => error)
  }
```

Függvény használata

```
import { findByToken } from './api/queries'

findByToken(token).then(response => response).catch(error => error)
```

Új jelszó

secureFetch bemeneti értékei

Paraméter	Érték
`URL`	`/user/password/reset`
`method`	`POST`
`headers`	`headers`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "token": "String",
  "password": "String",
  "password_confirmation": "String:same:password",
  "email": "String"
}
```

Függvény deklarációja

```
export const resetPassword = (data) => {
  secureFetch(
    '/user/password/reset',
    'POST',
    headers,
    JSON.stringify(data)
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { resetPassword } from './api/queries'

resetPassword(data).then(response => response).catch(error => error)
```

Regisztráció

secureFetch bemeneti értékei

Paraméter	Érték
`URL`	`/user/signup`
`method`	`POST`
`headers`	`headers`
`body`	`data`

BODY paraméter értéke

```
const data = {
  "email" : "String",
  "password" : "String",
  "password_confirmation" : "String:same",
  "role_id" : Number
}
```

Függvény deklarációja

```
export const userSignup = (data) => {
  secureFetch(
    '/user/signup',
    'POST',
    headers,
    JSON.stringify(data)
  ).then(res => res)
   .catch(error => error)
}
```

Függvény használata

```
import { userSignup } from './api/queries'

userSignup(data).then(response => response).catch(error => error)
```

- **ADMIN FORMS**

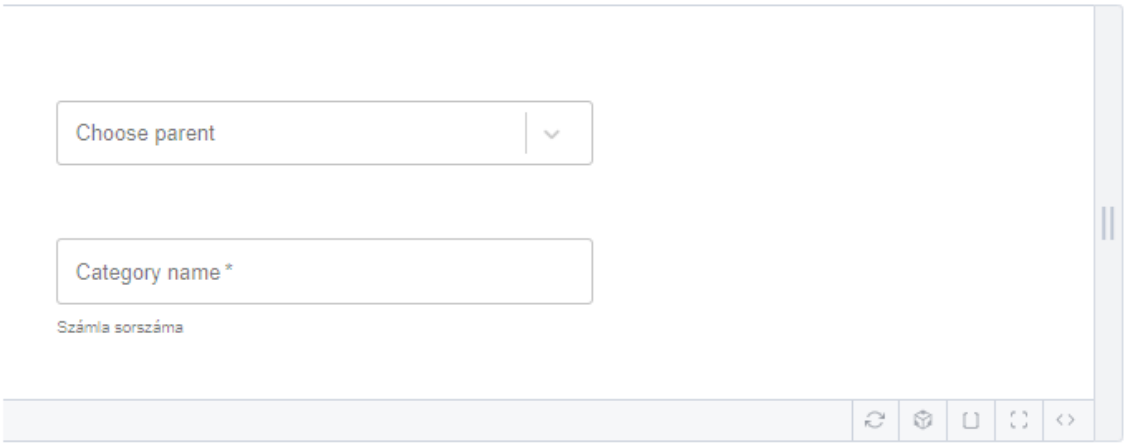
Új termékkategória létrehozása

Termékkategória létrehozásához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import CreateCategoryForm from './components/Froms/CreateCategoryForm'
```



Properties

name	String	= [Empty String]
submitHandler	Func	required
onChangeHandler	Func	required
parents	Array	required
setParentId	Func	required
selectedParent	Object	= null
errors	Object	= {}

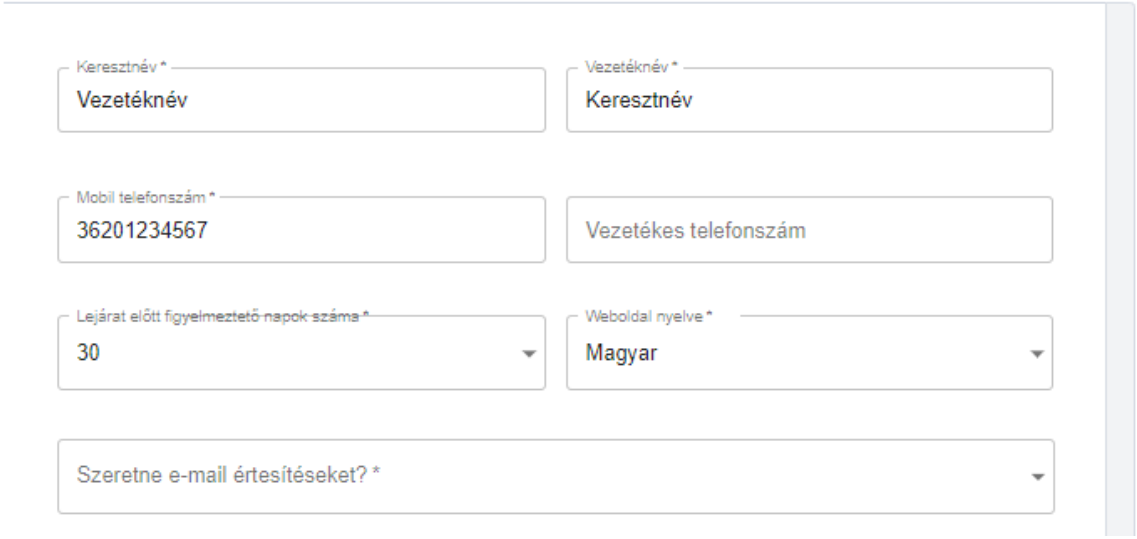
Vásárló adatainak módosítása

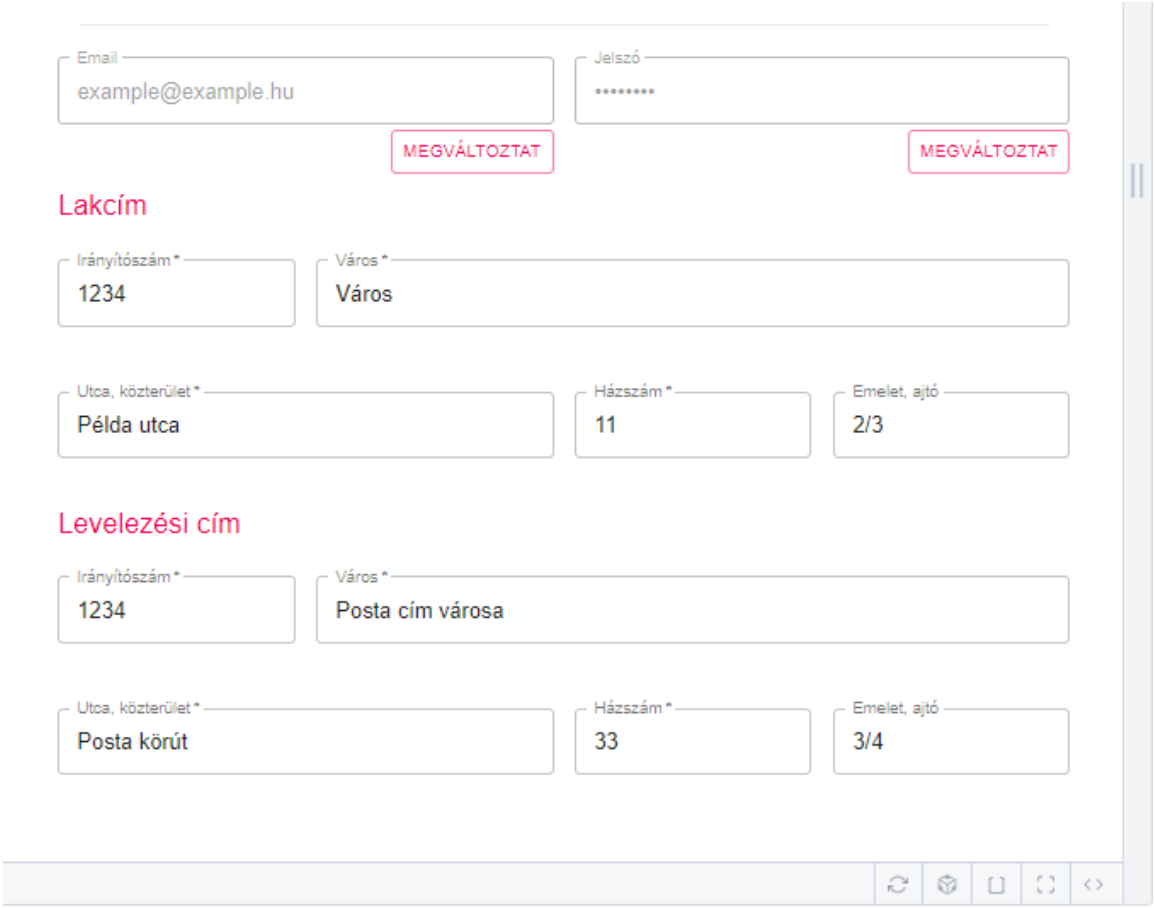
Vásárló adatainak módosításához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import CustomerUpdateForm from './components/Froms/CustomerUpdateForm'
```





Properties

onChange	Func	required
id	Number	required
user_email	String	required
first_name	String	required
mobile_phone	String	required
wired_phone	String	*= [Empty String]*
last_name	String	required
home_address_city	String	required
home_address_postal_code	Number	required

home_address_street	String	required
home_address_street_number	String	required
home_address_door_number	String	= [Empty String]
postal_address_door_number	String	= [Empty String]
postal_address_city	String	required
postal_address_postal_code	Number	required
postal_address_street	String	required
postal_address_street_number	String	required
expire_alert_in_day	Number	required
user_default_language	String	= "hu"
errors	Object	required

Extra form elem módosítása

Extra form elem mósodításához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import ExtraFormItemsForm from './components/Froms/ExtraFormItemsForm'
2
3  <ExtraFormItemsForm {...data} onChange={onChange} errors={errors} />
```

Properties

product_category_id	Number	= null
errors	Object	required
onChange	Func	required
name	String	required
description	String	required
form_item_0_type	String	= "text"
form_item_0_label	String	required
form_item_0_name	String	required
id	Number	required
form_item_0_value	String	= [Empty String]

Statek

State	Type	Default
subCategories	Array	[]
selected	Object	{}

Függvények

categories

A `getProductCategories()` függvény lekérdezi az összes kategóriát a visszakapott sikeres válaszból kiszűri az alkategóriákat, majd feltölti a **subCategories** nevű state tömböt az alkategóriák id-jával és nevével.

```
1  categories = () => getProductCategories('')
2    .then(res => res.json())
3    .then(json => json.success)
4    .then(subCategories => {
5      const suggestions = subCategories
6        .filter(suggestion => suggestion.parent_id !== null)
7        .map(suggestion => ({ value: suggestion.id, label: suggestion.name }))
8      this.setState({ subCategories: suggestions })
9    })
10   .then(() => this.defaultSelectedCategory())
11   .catch(error => error)
```

defaultSelectedCategory

A függvény beállítja a **product_category_id** property alapján az extra form elemhez tartozó kategóriát.

```
1  defaultSelectedCategory = () => {
2    this.setState(prevState => ({
3      selected: prevState.subCategories.filter(category => category.value ===
   this.props.product_category_id)
4    }))
5  }
```

setCategoryId

A függvény kategória módosítás esetén a kiválasztott kategóriát állítja be a **selected** state változó értékének.

```
setCategoryId = (selected) => {
  const notSelected = selected === null
  let categoryID = { product_category_id: null }

  if (notSelected) {
    selected = {}
  } else {
    categoryID = { product_category_id: selected.value }
  }

  this.setState({
    selected: selected
  })
  this.props.onChange(null, this.props.id, categoryID)
}
```

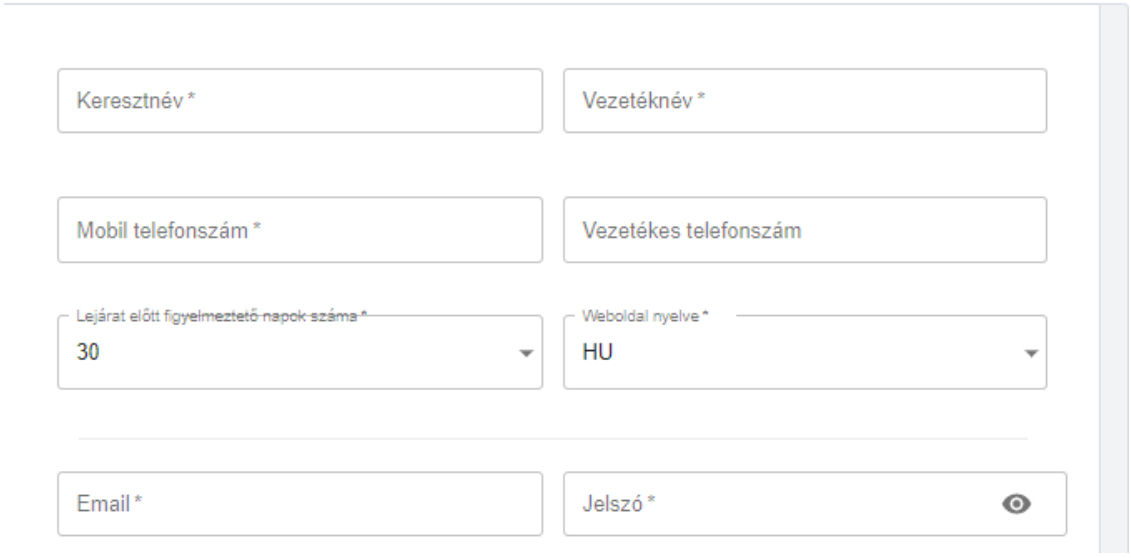
Új vásárló létrehozása

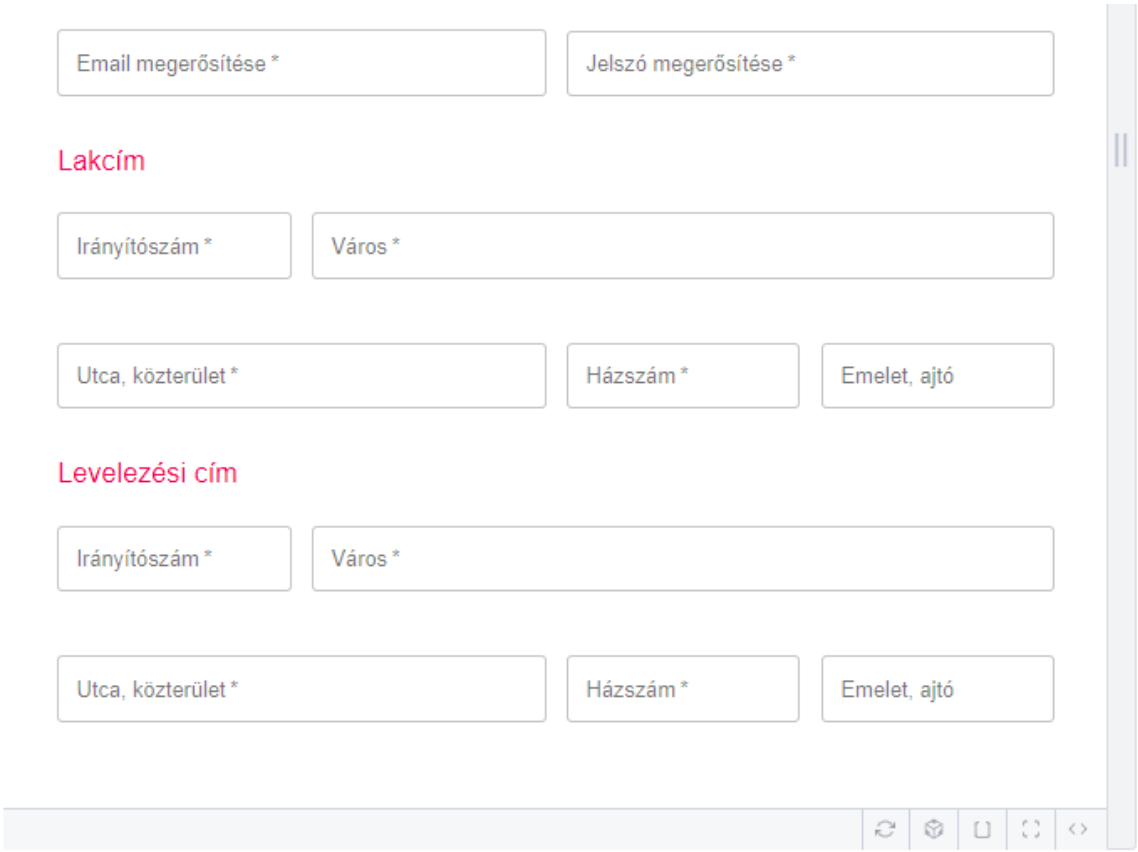
Vásárló létrehozásához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
import NewCustomerForm from './components/Froms/NewCustomerForm'
```





Properties

onChange	Func	required
id	Number	required
first_name	String	= [Empty String]
last_name	String	= [Empty String]
expire_alert_in_day	Number	= 30
home_address_city	String	= [Empty String]
home_address_postal_code	Number	= 0
home_address_street	String	= [Empty String]
home_address_street_number	String	= [Empty String]
home_address_door_number	String	= [Empty String]

postal_address_door_number	String	= [Empty String]
postal_address_city	String	= [Empty String]
postal_address_postal_code	Number	= 0
postal_address_street	String	= [Empty String]
postal_address_street_number	String	= [Empty String]
user_default_language	String	= "hu"
errors	Object	= {}

Extra form elem létrehozása

Extra form elem létrehozásához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import NewExtraFormItemsForm from './components/Froms/NewExtraFormItemsForm'
2
3  <NewExtraFormItemsForm {...data} onChange={onChange} errors={errors} />
```

Properties

product_category_id	Number	= null
errors	Object	required
onChange	Func	required
name	String	= [Empty String]
description	String	= [Empty String]
form_item_type	String	= "text"
form_item_label	String	= [Empty String]
form_item_name	String	= [Empty String]
id	Number	= null
form_item_value	String	= [Empty String]

Statek

State	Type	Default
subCategories	Array	[]
selected	Object	{}

Függvények

categories

A `getProductCategories()` függvény lekérdezi az összes kategóriát a visszakapott sikeres válaszból kiszűri az alkategóriákat, majd feltölti a **subCategories** nevű state tömböt az alkategóriák id-jával és nevével.

```
categories = () => getProductCategories('')
  .then(res => res.json())
  .then(json => json.success)
  .then(subCategories => {
    const suggestions = subCategories
      .filter(suggestion => suggestion.parent_id !== null)
      .map(suggestion => ({ value: suggestion.id, label: suggestion.name }))
    this.setState({ subCategories: suggestions })
  })
  .then(() => this.defaultSelectedCategory())
  .catch(error => error)
```

defaultSelectedCategory

A függvény beállítja a **product_category_id** property alapján az extra form elemhez tartozó kategóriát.

```
defaultSelectedCategory = () => {
  this.setState(prevState => ({
    selected: prevState.subCategories.filter(category => category.value ===
this.props.product_category_id)
  }))
}
```

setCategoryId

A függvény kategória módosítás esetén a kiválasztott kategóriát állítja be a **selected** state változó értékének.

```
setCategoryId = (selected) => {
  const notSelected = selected === null
  let categoryID = { product_category_id: null }
  if (notSelected) {
    selected = {}
  } else {
    categoryID = { product_category_id: selected.value }
  }

  this.setState({
    selected: selected
  })
  this.props.onChange(null, categoryID)
  }
```

Új jogosultság létrehozása

Jogosultság létrehozásához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
import NewPermissionForm from './components/Froms/NewPermissionForm'
```

Properties

onChange	Func	required
id	Number	= 0
name	String	= [Empty String]
guard_name	String	= "api"
errors	Object	required

Új szerepkör létrehozása

Szerepkör létrehozásához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import NewRoleForm from './components/Froms/NewRoleForm'
2
3  <NewRoleForm {...data} onChange={onChange} errors={errors} />
```

Properties

errors	Object	= {}
name	String	= [Empty String]
guard_name	String	= "api"
onChange	Func	required

Statek

State	Type	Default
permissions	Array	''
permissionsStatus	Object	''
filtered	Array	''

Függvények

getPermissions

Lekérdezi az összes jogosultságot, majd a szerepkörhöz hozzárendelt jogosultságok értékeit (**permissionStatus**) `true` -ra állítja.

```
const withPaginate = false
const withRelation = false
const withTrashed = false
const param = `?
withPaginate=${withPaginate}&withRelation=${withRelation}&withTrashed=${withTrashed}`
return getPermissions(param)
  .then(res => res.json())
  .then(json => json.success)
  .then(permissions => {
    this.setState({
      permissions
    })
    return permissions
  })
  .then(permissionData => {
```

```
    let permissionStatus = {}
    permissionData.map(allowedPermission => {
      return permissionStatus = { ...permissionStatus, [allowedPermission.id]: false }
    })
    this.setState({
      permissionStatus
    })
  })
  .catch(error => Error('getAllPermissions: ', error))
```

handleChange

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
handleChange = (e) => {
  const target = e.currentTarget.name
  const filteredPermissions = permissionFilter(this.state.permissionStatus, target)

  this.setState(() => {
    return ({
      permissionStatus: {
        ...this.state.permissionStatus,
        [target]: !this.state.permissionStatus[target]
      },
      filtered: filteredPermissions
    })
  })
  this.props.onChange(e, { permissionStatus: filteredPermissions })
}
```

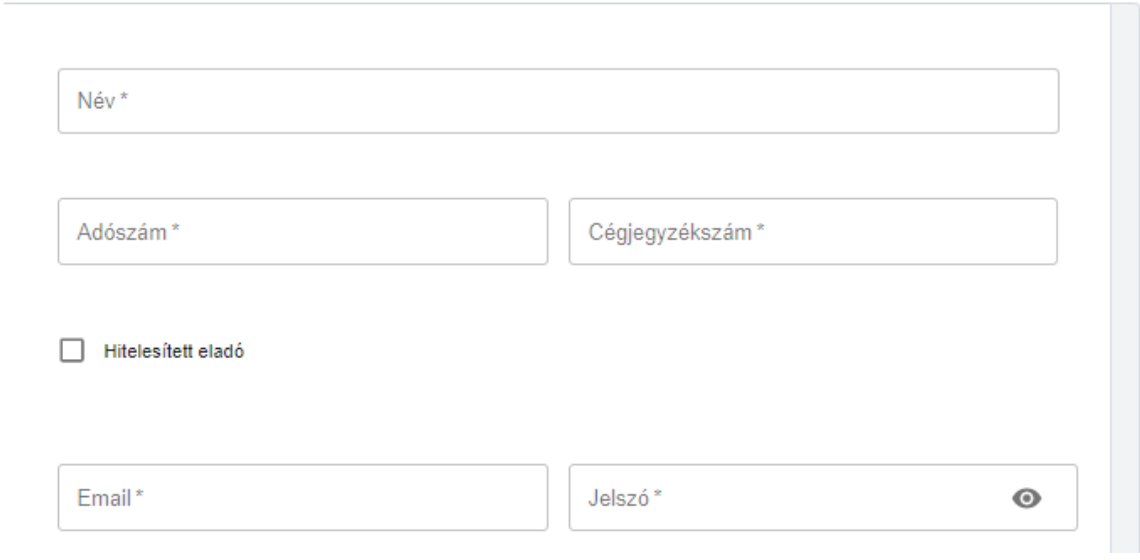
Új eladó létrehozása

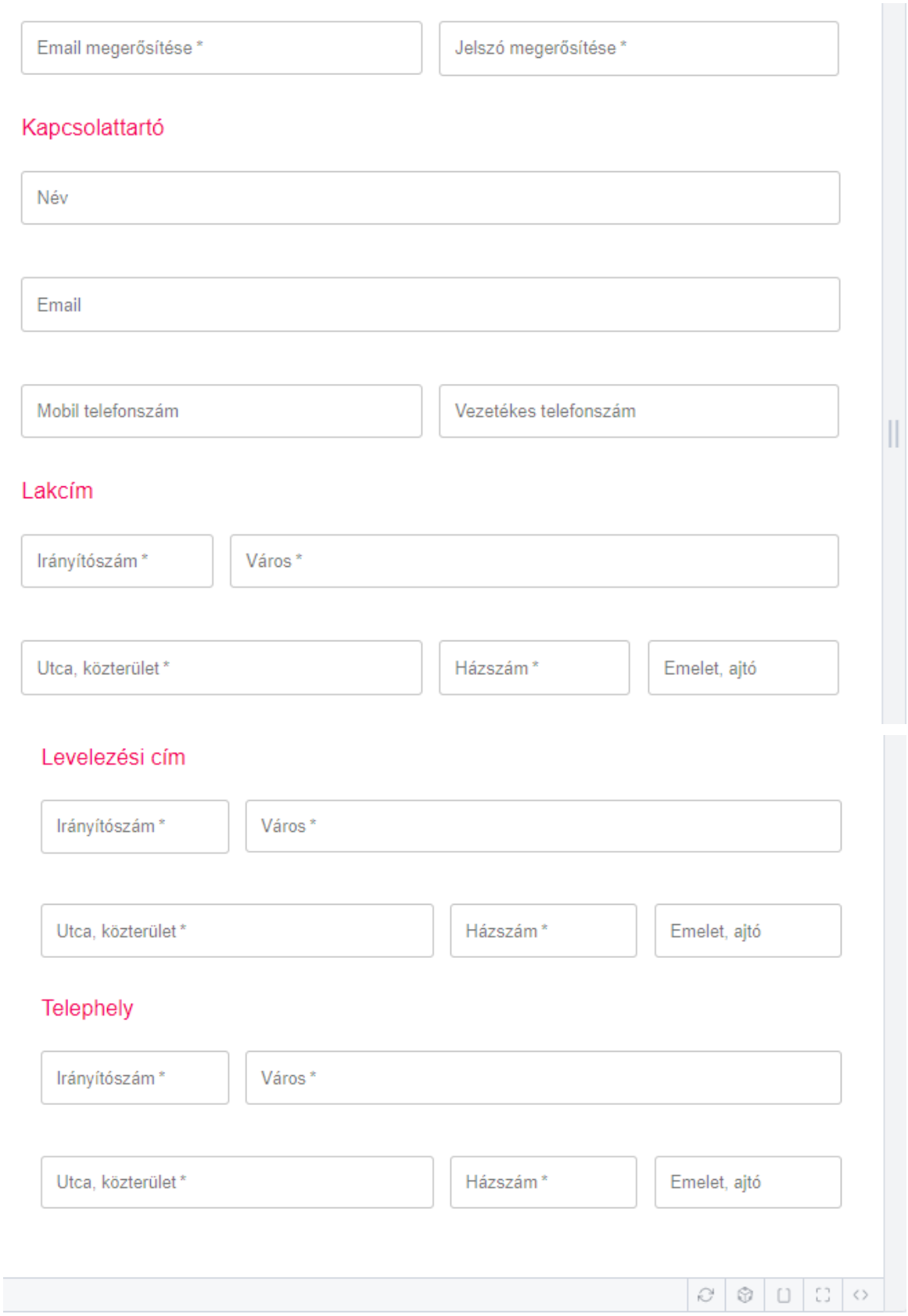
Eladó létrehozásához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
import NewSellerForm from './components/Froms/NewSellerForm'
```




Email megerősítése *
Jelszó megerősítése *
Kapcsolattartó
Név
Email
Mobil telefonszám
Vezetékes telefonszám
Lakcím
Irányítószám *
Város *
Utca, közterület *
Házszám *
Emelet, ajtó
Levelezési cím
Irányítószám *
Város *
Utca, közterület *
Házszám *
Emelet, ajtó
Telephely
Irányítószám *
Város *
Utca, közterület *
Házszám *
Emelet, ajtó

Properties

onChange	Func	required
id	Number	required
name	String	= [Empty String]
tax_number	String	= [Empty String]
registration_number	String	= [Empty String]
home_address_city	String	= [Empty String]
home_address_postal_code	String	= 0
home_address_street	String	= [Empty String]
home_address_street_number	String	= [Empty String]
home_address_door_number	String	= [Empty String]
postal_address_door_number	String	= [Empty String]

postal_address_city	String	= [Empty String]
postal_address_postal_code	Number	= 0
postal_address_street	String	= [Empty String]
postal_address_street_number	String	= [Empty String]
site_address_door_number	String	= [Empty String]
site_address_city	String	= [Empty String]
site_address_postal_code	String	= 0
site_address_street	String	= [Empty String]
site_address_street_number	String	= [Empty String]
contact_person_name	String	= [Empty String]
contact_person_email	String	= [Empty String]

contact_person_email	String	= [Empty String]
contact_person_mobile_phone	String	= [Empty String]
contact_person_wired_phone	String	= [Empty String]
authenticated_seller	Number	= 0
errors	Object	= {}

Szerepkör módosítása

Szerepkör módosításához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import RolesForm from './components/Froms/RolesForm'
2
3  <RolesForm {...data} onChange={onChange} errors={errors} />
```

Properties

id	Number	required
errors	Object	required
onChange	Func	required
name	String	required
guard_name	String	required
permissions	Array	required

Statek

State	Type	Default
roleData	Array	[]
permissions	Array	[]
permissionsStatus	Object	{}
filtered	Array	[]
name	String	''
guard_name	String	''

Függvények

getRoleById

Lekérdezi id alapján a szerepkör adatait, majd a hozzárendelt jogosultságok értékeit (**permissionStatus**) `true` -ra állítja.

```
const { id, name, guard_name } = this.props
getRoleById(id)
  .then(res => res.json())
  .then(json => json.success)
  .then(roleData => {
    const permissionStatus = permissionsStatus(roleData)
    this.setState({
      permissionStatus,
      permissions: roleData.allowedPermissions
    })
    return roleData
  })
  .then(roleData => {
    this.setState({ roleData })
    return roleData
  })
  .then(() => {
    const { permissionStatus } = this.state
    const filteredPermissions = permissionFilter(permissionStatus)
    this.setState({
      filtered: filteredPermissions
    })
  })
  .catch(error => Error('getRoleById: ', error))
this.setState({
  name: name,
  guard_name: guard_name
})
```

changeText

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
changeText = (e) => {
  const target = e.currentTarget.name
  const value = e.currentTarget.value
  this.setState({
    [target]: value
  })
  this.props.onChange(e, this.props.id, { permissionStatus: this.state.filtered })
}
```

handleChange

A jogosultságok jelölőmezőinek változásakor az aktuális mező nevével tárolt értéket megváltoztatja a jelölőmező értékére.

```
handleChange = (e) => {
  const target = e.currentTarget.name
  const filteredPermissions = permissionFilter(this.state.permissionStatus, target)

  this.setState(() => {
    return ({
      permissionStatus: {
        ...this.state.permissionStatus,
        [target]: !this.state.permissionStatus[target]
      },
      filtered: filteredPermissions
    })
  })

  this.props.onChange(e, this.props.id, { permissionStatus: filteredPermissions })
}
```

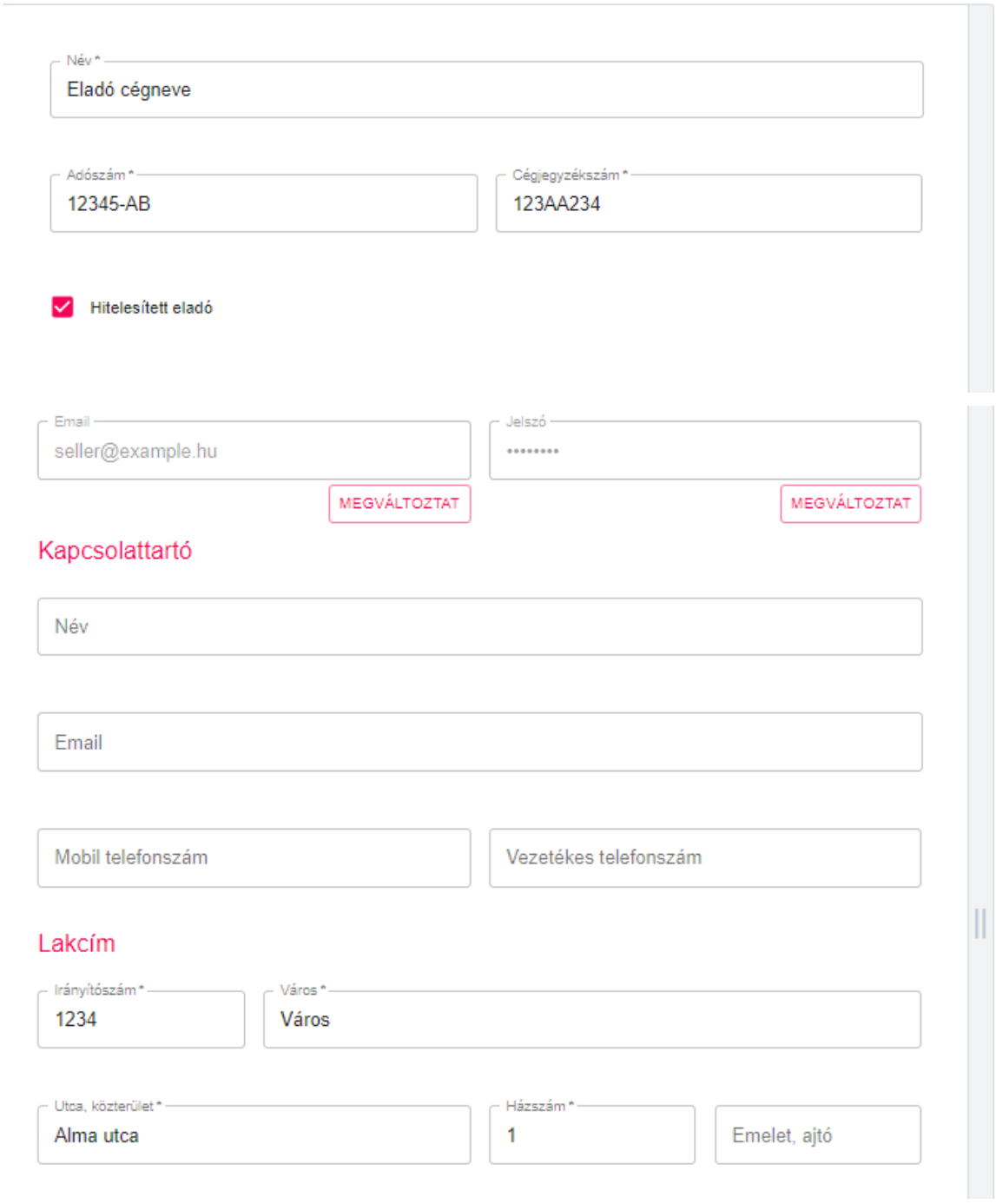
Eladó módosítása

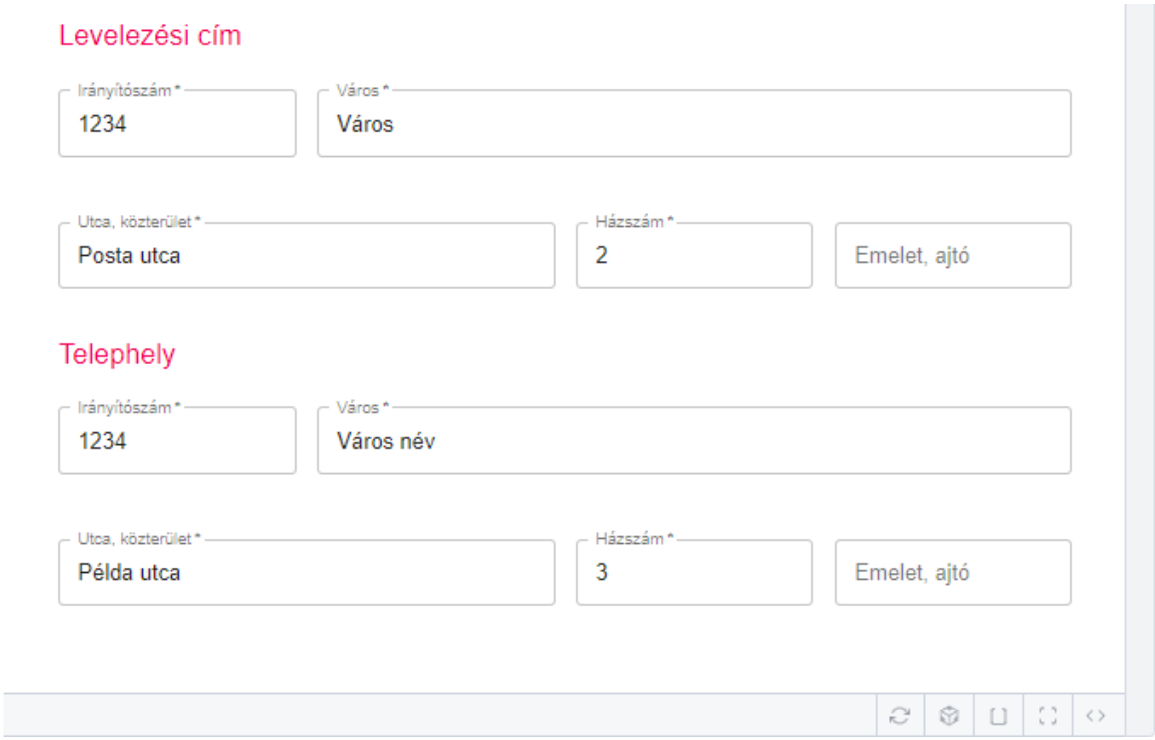
Eladó mósodításához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import SellerUpdateForm from './components/Froms/SellerUpdateForm'
```





Properties

onChange	Func	required
id	Number	required
user_email	String	required
name	String	required
tax_number	String	required
registration_number	String	required
home_address_city	String	required
home_address_postal_code	Number	required
home_address_street	String	required
home_address_street_number	String	required
home_address_door_number	String	*= [Empty String]*

postal_address_door_number	String	= [Empty String]
postal_address_city	String	required
postal_address_postal_code	Number	required
postal_address_street	String	required
postal_address_street_number	String	required
site_address_city	String	required
site_address_postal_code	Number	required
site_address_street	String	required
site_address_street_number	String	required
site_address_door_number	String	= [Empty String]

contact_person_name	String	= [Empty String]
contact_person_email	String	= [Empty String]
contact_person_mobile_phone	String	= [Empty String]
contact_person_wired_phone	String	= [Empty String]
authenticated_seller	Number	required
errors	Object	required

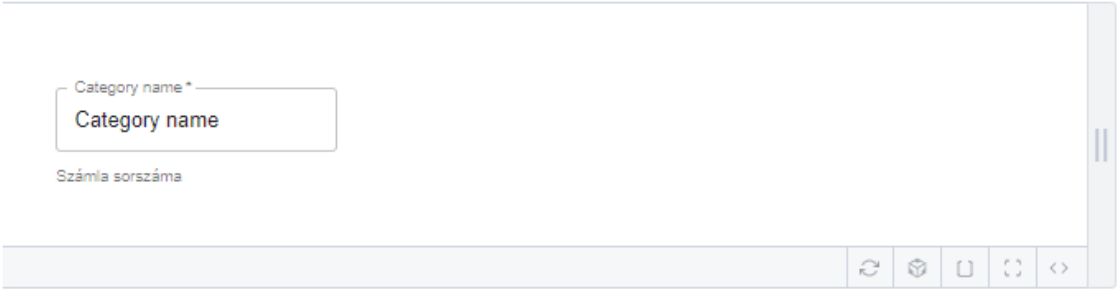
Termékkategória módosítása

Termékkategória módosításához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import UpdateCategoryForm from './components/Froms/UpdateCategoryForm'
```



Properties

name	String	required
parentId	Number	= null
submitHandler	Func	required
onChangeHandler	Func	required
selectedParent	Any	= null
errors	Object	required

- **ADMIN DIALOGS**

Kategória létrehozása

Admin felülethez tartozó dialógus komponens, amely a termékkategóriák létrehozásáért felel.

Általános használat

```
import CreateCategoryDialog from './components/Dialog/CreateCategoryDialog'

<CreateCategoryDialog
  maxWidth="lg"
  {...this.props}
  open={addNewCategoryModal}
  handleClose={this.handleClose}
  updateData={this.productCategoryData}
/>
```

Properties

open	Bool	required
handleClose	Func	required
maxWidth	String	= "lg"

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását ill. a létrehozáshoz szükséges ürlapot.

```
<Dialog maxWidth={maxWidth} dialogTitle="Új kategória létrehozása" open={open} onClose={handleClose} className="categoryDialog" >
  <DialogContent>
    <Typography variant="body1" component="p">
    </Typography>
    <CreateCategoryForm
      submitHandler={this.submitHandler}
      onChangeHandler={this.onChangeHandler}
      errors={errors}
      parents={parents}
      setParentId={this.setParentId}
      parentId={parentId}
    />
  </DialogContent>
  <DialogActions>
```

```
15      <Button onClick={handleClose} variant="outlined" color="default" size="medium" >
    {common.buttons.cancel}</Button>
16      <Button variant="outlined" form="CreateCategoryForm" type="submit" color="secondary"
    size="medium">{common.buttons.save}</Button>
17    </DialogActions>
18  </Dialog >
```

Statek

State	Type	Default
parents	Array	[]
errors	Object	{}
parentId	Number	null
name	String	[Empty string]

Függvények

productCategoryParents

A productCategoryParents meghívja a `getProductCategories()` függvényt. A visszakapott sikeres válaszból kiszűri az főkategóriákat, majd feltölti a **parents** nevű state tömböt az főkategóriák id-jával és nevével.

```
1  productCategoryParents = () => {
2    getProductCategories('')
3      .then(res => res.json())
4      .then(json => json.success)
5      .then(categories => categories.filter(category => category.parent_id === null))
6      .then(parents => {
7        const suggestions = parents.map(suggestion => ({ value: suggestion.id, label:
    suggestion.name }))
8        this.setState({ parents: suggestions })
9      })
10     .catch(error => error)
11 }
```

setParentId

A setParentId beállítja a kiválasztott főkategória id-ját a **parentId** értékének. Ha nincs kiválasztva főkategória akkor a **parentId** értéke `null` .

```
setParentId = (selected) => {
  const isSelected = selected !== null
  let parentId = null
  let selectedParent = null

  if (isSelected) {
    parentId = selected.value,
    selectedParent = selected
  }
  this.setState({
    parentId,
    selectedParent
  })
}
```

onChangeHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
onChangeHandler = (e) => {
    const target = e.target.name
    const value = e.target.value
    this.setState({
      [target]: value
    })
  }
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `createProductCategories()` függvénynek átadja paraméterül. Sikeres létrehozás esetén az oldalon lévő adatokat frissíti, megjelenik egy sikeres létrehozás üzenet és a dialógusablak bezáródik. Ha nem sikerült létrehozni a kategóriát, akkor az errors state objektum megkapja az aktuális hibaüzeneteket, amiket a dialógusablakban megjelenítünk.

```
submitHandler = (e) => {
    const { lang } = this.context
    e.preventDefault()
    const {
      name,
      parentId,
    } = this.state
    const data = {
      name,
      parent_id: parentId
    }
    createProductCategories(data)
      .then(res => res.json())
      .then(json => {
        if (json.success) {
          this.props.updateData()
          this.props.alertHandler('success', lang.common.messages.success.update,
lang.common.messages.success.title)
          this.props.handleClose()
        } else {
          this.setState({
            errors: json.errors
          })
        }
      })
      .catch(error => Error(error))
  }
```

Kategória módosítása

Admin felülethez tartozó dialógus komponens, amely a termékkategóriák módosításáért felel.

Általános használat

```
import UpdateCategoryDialog from './components/Dialog/UpdateCategoryDialog'

<UpdateCategoryDialog
  maxWidth="lg"
  {...this.props}
  open={updateCategoryModal}
  updateRowId={updateRowId}
  handleClose={this.handleClose}
  updateData={this.productCategoryData}
/>
```

Properties

open	Bool	required
updateRowId	String	required
updateData	Func	required
handleClose	Func	required
maxWidth	String	= "lg"

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását ill. a létrehozáshoz szükséges ürlapot.

```
<Dialog maxWidth={maxWidth} dialogTitle="Kategória módosítása" open={open} onClose={handleClose}
>
  <DialogContent>
    <Typography variant="body1" component="p"></Typography>
    <UpdateCategoryForm
      {...this.state}
      submitHandler={this.submitHandler}
      onChangeHandler={this.onChangeHandler}
    />
  </DialogContent>
  <DialogActions>
    <Button onClick={handleClose} variant="outlined" color="default" size="medium" >
      {common.buttons.cancel}
    </Button>
    <Button variant="outlined" form="UpdateCategoryForm" type="submit" color="secondary"
      {common.buttons.save}
    </Button>
  </DialogActions>
</Dialog >
```

Statek

State	Type	Default
parentId	Number	null
selectedParent	Object	{}
name	String	''
errors	Object	{}

Függvények

getMainCategory

A `getProductCategoryById()` függvény meghívásával a paraméterül kapott id segítségével lekérdezi a kategória adatait a visszakapott sikeres válaszból a **name** nevű statenek megadja a válaszban kapott name értéket, ill. ha kategóriának van szülő kategóriája, akkor annak az adatait is lekérdezi.

```
getMainCategory = () => {
  const { updateRowId } = this.props
  getProductCategoryById(updateRowId)
    .then(res => res.json())
    .then(json => json.success[0])
    .then(success => {
      this.setState({ name: success.name })
      if (success.parent_id !== null) {
        getProductCategoryById(success.parent_id)
          .then(res => res.json())
          .then(json => json.success[0])
          .then(success => this.setState({
            parentId: success.id,
            selectedParent: { value: success.id, label: success.name }
          })).catch(error => error)
      }
    }).catch(error => error)
}
```

onChangeHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
onChangeHandler = (e) => {
  const target = e.target.name
  const value = e.target.value
  this.setState({
    [target]: value
  })
}
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `updateProductCategories()` függvénynek átadja paraméterül. Sikeres módosítás esetén bezárja a dialógus ablakot, újratölti az adatokat és megjelenik egy sikeres módosítás üzenet. Ha nem sikerült módosítani a kategóriát, akkor az **errors** state objektum megkapja az aktuális hibaüzeneteket, amiket a dialógusablakban megjelenítünk.

```
submitHandler = (e) => {
  e.preventDefault()
  const { name, parentId } = this.state
  const { updateRowId } = this.props
  const { lang } = this.context
  const data = {
    name,
    id: updateRowId,
    parent_id: parentId
  }
  updateProductCategories(updateRowId, data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.props.updateData()
        this.props.alertHandler('success', lang.common.messages.success.update, lang.common.messages.success.title)
        this.props.handleClose()
      } else {
        this.setState({
          errors: json.errors
        })
      }
    })
    .catch(error => Error(error))
}
```

- **ADMIN TABLES**

Vásárló táblázat

Általános használat

```
import CustomersTable from './components/Tables/CustomersTable';

<CustomersTable editable={true} />

```

A vásárló táblázat komponensben történik a vásárlóval kapcsolatos összes lekérdezés úgy mint: összes listázása, új létrehozása, szerkesztés, státuszállítás.

Összes vásárló listázása

A `getCustomers()` függvény segítségével lekérdezzük az aktuális oldalon lévő összes vásárlót (oldalanként max 15). Majd előkészítjük a megjelenítendő adatokat a táblázatnak, összefűzzük a címadatokat, a vezeték és keresztnevet ill. hozzáadunk egy státusz mezőt az objektumba.

```
customersData = (currentPage) => getCustomers(currentPage)
  .then(res => res.json())
  .then(json => json.success)
  .then(customers => {
    this.setState({ customers })
    return customers.data
  })
  .then(customersData => addressConcatHelper(customersData))
  .then(customersDataWithAddress => nameConcatHelper(customersDataWithAddress))
  .then(customersDataWithName => addStatus(customersDataWithName))
  .then(customersData => this.setState({ customersData }))
  .catch(error => console.error(error))

```

Új vásárló létrehozása

Új vásárlót a `createCustomer(data)` függvény segítségével hozunk létre. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres létrehozás esetén bezárni a dialógus ablakot.

```
createCustomerPromise = (data) => (new Promise((resolve, rejects) => {
    createCustomer(data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.customersData(this.state.currentPage)
        this.setState({
          errors: {},
          alertType: 'success'
        })
        resolve('success')
      } else {
        this.setState({
          errors: json.errors,
          alertType: 'error'
        })
        rejects('fail')
      }
    })
  }))
```

Vásárló adatainak módosítása

A vásárló adatait `updateCustomer(data.user_id, updatedData)` függvény segítségével módosítjuk. A nem kötelező mezőket csak módosítás esetén fűzzük az objektumhoz. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres módosítás esetén bezárni a dialógus ablakot.

```
updateCustomerPromise = (data) => (new Promise((resolve, rejects) => {
    const changeEmail = (data.user_email !== '' && data.user_email_confirmation !== '') &&
(data.user_email && data.user_email_confirmation)

    const changePass = (data.password !== '' && data.password_confirmation !== '') &&
(data.password && data.password_confirmation)

    let updatedData = {
      id: data.user_id || '',
      first_name: data.first_name || '',
      last_name: data.last_name || '',
      mobile_phone: data.mobile_phone || '',
      home_address_postal_code: data.home_address_postal_code || '',
      home_address_city: data.home_address_city || '',
      home_address_street: data.home_address_street || '',
      home_address_street_number: data.home_address_street_number || '',
      postal_address_postal_code: data.postal_address_postal_code || '',
      postal_address_city: data.postal_address_city || '',
      postal_address_street: data.postal_address_street || '',
      postal_address_street_number: data.postal_address_street_number || ''
    }
    if (changeEmail) {
      const email = {
        email: data.email,
        email_confirmation: data.email_confirmation,
      }
      updatedData = { ...updatedData, ...email }
    }
    if (changePass) {
      const password = {
```

```
        password: data.password,
        password_confirmation: data.password_confirmation,
      }
      updatedData = { ...updatedData, ...password }
    }
    if (data.home_address_door_number !== '' && data.home_address_door_number !== null) {
      const doorNumber = {
        home_address_door_number: data.home_address_door_number
      }
      updatedData = { ...updatedData, ...doorNumber }
    }
    if (data.postal_address_door_number !== '' && data.postal_address_door_number !== null) {
      const doorNumber = {
        postal_address_door_number: data.postal_address_door_number
      }
      updatedData = { ...updatedData, ...doorNumber }
    }
    if (data.wired_phone !== '' && data.wired_phone !== null) {
      const wiredPhone = {
        wired_phone: data.wired_phone
      }
      updatedData = { ...updatedData, ...wiredPhone }
    }
    updateCustomer(data.user_id, updatedData)
      .then(res => res.json()).then(json => {
        if (json.success) {
          this.customersData(this.state.currentPage)
          this.setState({
            errors: {},
            alertType: 'success'
          })
          resolve('success')
        } else {
          this.setState({
            errors: json.errors,
            alertType: 'error'
          })
          rejects('fail')
        }
      })
      .catch(error => console.log(error))
  }))
```

Vásárló státuszának beállítása

A vásárló státuszát `toggleStatusCustomer(id, data)` függvény segítségével módosítjuk. Egy vásárló státusza akkor aktív ha a `customersData` objektumban a `user` objektum nem `null` . A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres módosítás esetén bezárni a dialógus ablakot.

```
toggleStatusCustomerPromise = (id) => (new Promise((resolve, rejects) => {
    const isActiveCustomer = this.state.customersData.filter(customer => customer.user_id === id)
[0].user !== null
    let active = true
    if (isActiveCustomer) {
      active = false
    }
    const data = {
      id,
      active
    }
    toggleStatusCustomer(id, data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.customersData(this.state.currentPage)
        this.setState({
          errors: {},
          alertType: 'success'
        })
        resolve('success')
      } else {
        this.setState({
          errors: json.errors,
          alertType: 'error'
        })
        rejects('fail')
      }
    })
  }))
```

Extra form elem táblázat

Általános használat

```
1  import ExtraFormItemsTable from './components/Tables/ExtraFormItemsTable';
2
3  <ExtraFormItemsTable editable={true} />
4
```

Az extra form elem táblázat komponensben történik az extra form elemekkel kapcsolatos összes lekérdezés úgy mint: összes listázása, új létrehozása, szerkesztés, státuszállítás.

Függvények

extraFormItemsData

A `getExtraFormItems()` függvény segítségével lekérdezzük az összes extra form elemet (oldalanként max 15).

```
1  extraFormItemsData = (currentPage) => {
2    currentPage = this.state.currentPage
3    const withPaginate = true
4    const withRelation = true
5    const withTrashed = true
6    const param = `?
   page=${currentPage}&withPaginate=${withPaginate}&withRelation=${withRelation}&withTrashed=${withTr
   ashed}`
7
8    return getExtraFormItems(param)
9      .then(res => res.json())
10     .then(json => {
11       const {lang} = this.context
12       if (json.success) {
13         return json.success
```

```
      } else {
        this.props.alertHandler('error', json.errors.id[0], lang.common.messages.errors.fail)
        this.setState({
          status: 200
        })
        return {}
      }
    })
    .then(formItems => {
      let formattedData = formItems.data.map(category => {
        if (category.parent_id === null) {
          return { ...category, 'name': `<b>${category.name}</b>` }
        } else {
          return category
        }
      })
      this.setState({
        extraFormItems: formItems,
        status: 200
      })
      return formattedData
    })
    .then(res => formDataCreator(res))
    .then(res => formDataCreator(res))
    .then(formattedData => addStatus(formattedData))
    .then(formattedData => this.setState({
      extraFormItemsData: formattedData,
    }))
    .catch(error => error)
}
```

createExtraFormItemsPromise

Új extra form elemet a `createExtraFormItems(data)` függvény segítségével hozunk létre. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres létrehozás esetén bezárni a dialógus ablakot.

```
createExtraFormItemsPromise = (data) => {
  new Promise((resolve, rejects) => {
    const newExtraFormItemData = {
      form_group_id: data.id,
      form_group_product_subcategory_id: data.product_category_id,
      form_group_name: data.name,
      form_group_description: data.description,
      form_item_type: 'text',
      form_item_name: data.form_item_name,
      form_item_label: data.form_item_label,
      form_item_value: data.form_item_value
    }
    const { lang } = this.context
    return (
      createExtraFormItems(newExtraFormItemData)
        .then(res => res.json())
        .then(json => {
          if (json.success) {
            this.extraFormItemsData(this.state.currentPage)
            this.setState({
              errors: {},
              alertType: 'success'
            })
            this.props.alertHandler('success', lang.common.messages.success.create, lang.common.messages.success.title)
            resolve('success')
          } else {
            this.setState({
              errors: json.errors,
              alertType: 'error'
            })
            this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
            rejects('fail')
          }
        }).catch(error => error)
    )
  })
}
```

updateExtraFormItemsPromise

Az extra form elem adatait `updateExtraFormItems(data.id, newData)` függvény segítségével módosítjuk. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres módosítás esetén bezárni a dialógus ablakot.

```
updateExtraFormItemsPromise = (data) => (new Promise((resolve, rejects) => {
  const newData = {
    form_group_id: data.id,
    form_group_product_subcategory_id: data.product_category_id,
    form_group_name: data.name,
    form_group_description: data.description,
    form_item_type: data.form_item_0_type,
    form_item_name: data.form_item_0_name,
    form_item_label: data.form_item_0_label,
    form_item_value: data.form_item_0_value
  }
  const { lang } = this.context
  updateExtraFormItems(data.id, newData)
    .then(res => res.json()).then(json => {
      if (json.success) {
```

```
        this.extraFormItemsData(this.state.currentPage)
        this.setState({
          errors: {},
          alertType: 'success'
        })
        this.props.alertHandler('success', lang.common.messages.success.update,
lang.common.messages.success.title)
        resolve('success')
      } else {
        this.setState({
          errors: json.errors,
          alertType: 'error'
        })
        this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
        rejects('fail')
```

```
      }
    })
    .catch(error => error)
}))
```

toggleStatusExtraFormItemsPromise

Az extra form elem státuszát `toggleStatusFormItems(id, data)` függvény segítségével módosítjuk. Egy elem státusza akkor aktív ha a `extraFormItemsData` objektumban a `deleted_at` objektum nem `null` . A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnak a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres módosítás esetén bezárni a dialógus ablakot.

```
toggleStatusExtraFormItemsPromise = (id) => (new Promise((resolve, rejects) => {
  const isActiveFormGroup = this.state.extraFormItemsData.filter(item => item.id === id)
[0].deleted_at === null
  let active = true
  if (isActiveFormGroup) {
    active = false
  }
  const data = {
    form_group_id: id,
    active
  }
  const { lang } = this.context
  toggleStatusFormItems(id, data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.forceUpdate()
        this.extraFormItemsData(this.state.currentPage)
        this.setState({
          errors: {},
          alertType: 'success'
        })
        this.props.alertHandler('success', lang.common.messages.success.update,
lang.common.messages.success.title)
        resolve('success')
      } else {
        this.setState({
          errors: json.errors,
          alertType: 'error'
        })
        this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
        rejects('fail')
      }
    }).catch(error => new Error(error))
}))
```

Függvények

permissionsData

A `getPermissions()` függvény segítségével lekérdezzük az összes jogosultságot (oldalanként max 15).

```
permissionsData = (currentPage) => {
  currentPage = currentPage === undefined && this.state.currentPage
  const withPaginate = true
  const withRelation = true
  const withTrashed = false
  const param = `?page=${currentPage}&withPaginate=${withPaginate}&withRelation=${withRelation}&withTrashed=${withTrashed}`

  return getPermissions(param)
    .then(res => {
      if (res.status === 200) {
        return res.json()
      }
    })
    .then(json => {
      const {lang} = this.context
      if (json.success) {
        return json.success
      } else {
        this.props.alertHandler('error', json.errors.id[0], lang.common.messages.errors.fail)
        this.setState({
          status: 200
        })
        return {}
      }
    })
    .then(permissions => {
      this.setState({ permissions })
      return permissions.hasOwnProperty('data') ? permissions.data : []
```

```
29      })
30      .then(permissionsData => {
31        this.setState({
32          formattedData: formDataCreator(permissionsData),
33          status: 200
34        })
35        return permissionsData
36      })
37      .then(permissionsData => this.setState({ permissionsData }))
38      .catch(error => Error('getPermissions:', error))
39  }
```

createPermissionPromise

Új jogosultságot a `createPermission(data)` függvény segítségével hozunk létre. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres létrehozás esetén bezárni a dialógus ablakot.

```
1   createPermissionPromise = (data) => (new Promise((resolve, rejects) => {
2     if (!data.hasOwnProperty('guard_name')) {
3       const guardName = {
4         guard_name: 'api'
5       }
6       data = { ...data, ...guardName }
7     }
8     const { lang } = this.context
9     createPermission(data)
10      .then(res => res.json())
11      .then(json => {
12        if (json.success) {
13          this.permissionsData(this.state.currentPage)
14          this.setState({
15            errors: {},
```

```
15            errors: {},
16            alertType: 'success'
17          })
18          this.props.alertHandler('success', lang.common.messages.success.create,
    lang.common.messages.success.title)
19          resolve('success')
20        } else {
21          this.setState({
22            errors: json.errors,
23            alertType: 'error'
24          })
25          this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
26
27          rejects('fail')
28        }
```

```
29      }).catch(error => Error('createPermission:', error))
30  }))
```

updatePermissionsPromise

A jogosultság adatait `updatePermission(data.id, updatedData)` függvény segítségével módosítjuk. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnunk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres módosítás esetén bezárni a dialógus ablakot.

```
1  updatePermissionsPromise = (data) => (new Promise((resolve, rejects) => {
2    let updatedData = {
3      id: data.id || '',
4      name: data.name || '',
5      guard_name: data.guard_name || 'api',
6    }
7    const { lang } = this.context
8    updatePermission(data.id, updatedData)
9      .then(res => res.json()).then(json => {
10       if (json.success) {
11         this.permissionsData(this.state.currentPage)
12         this.setState({
13           errors: {},
14           alertType: 'success'
15         })
16         this.props.alertHandler('success', lang.common.messages.success.update,
   lang.common.messages.success.title)
17         resolve('success')
18       } else {
19         this.setState({
20           errors: json.errors,
21           alertType: 'error'
22         })
23         this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
24
25         rejects('fail')
26       }
27     })
28     .catch(error => Error('updatePermission: ', error))
29 }))
```

deletePermissionPromise

A jogosultságok a `deletePermission(id, data)` függvény segítségével töröljük. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres törlés esetén bezárni a dialógus ablakot.

```
deletePermissionPromise = (id) => (new Promise((resolve, rejects) => {
  const data = {
    id: id,
  }
  const { lang } = this.context
  deletePermission(id, data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.permissionsData(this.state.currentPage)
        this.setState({
          errors: {},
          alertType: 'success'
        })
        this.props.alertHandler('success', lang.common.messages.success.delete,
        resolve('success')
      } else {
        this.setState({
          errors: json.errors,
          alertType: 'error'
        })
        this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)

        rejects('fail')
      }
    }).catch(error => Error('deletePermission: ', error))
}))
```

Termékkategória táblázat

Általános használat

```
1  import ProductCategoryTable from './components/Tables/ProductCategoryTable';
2
3  <ProductCategoryTable editable={true} />
4
```

Megjeleníti a termékkategóriákat főkategóriák szerint csoportosítva egy táblázatban.

Függvények

productCategoryData

A `getProductCategories()` függvény segítségével lekérdezzük az összes kategóriát.

```
1  productCategoryData = (currentPage) => {
2    currentPage = currentPage === undefined && this.state.currentPage
3    const withPaginate = false
4    const withRelation = false
5    const withTrashed = true
6    const param = `?
   page=${currentPage}&withPaginate=${withPaginate}&withRelation=$withRelation}&withTrashed=${withTra
   shed}`
7    const {lang} = this.context
8    return getProductCategories(param)
9      .then(res => res.json())
10     .then(json => {
11       if (json.success) {
12         return json.success
13       } else {
```

```
14          this.props.alertHandler('error', json.errors.id[0], lang.common.messages.errors.fail)
15          this.setState({
16            status: 200
17          })
18          return {}
19        }
20      })
21      .then(productCategories => {
22        this.setState({
23          productCategoryData: productCategories,
24          productCategory: productCategories,
25          status: 200
26        })
27      })
28      .catch(error => error)
29  }
```

Szerepkörök táblázata

Általános használat

```
import RolesTable from './components/Tables/RolesTable';

<RolesTable editable={true} />

```

A jogosultság táblázat komponensben történik a jogosultságokkal kapcsolatos összes lekérdezés úgy mint: összes listázása, új létrehozása, szerkesztés, törlés.

Függvények

rolesData

A `getRoles()` függvény segítségével lekérdezzük az összes szerepkört (oldalanként max 15).

```
rolesData = (currentPage) => {
  currentPage = currentPage === undefined && this.state.currentPage
  const withPaginate = true
  const withRelation = true
  const param = `?page=${currentPage}&withPaginate=${withPaginate}  withRelation=${withRelation}`
  const { lang } = this.context
  return getRoles(param)
    .then(res => {
      if (res.status === 200) {
        return res.json()
      }
    })
    .then(json => {
      if (json.success) {
        return json.success
```

```
      } else {
        this.props.alertHandler('error', json.errors.id[0], lang.common.messages.errors.fail)
        this.setState({
          status: 200
        })
        return {}
      }
    })
    .then(roles => {
      this.setState({ roles })
      return roles
    })
    .then(roles => {
      let data = []
      data = roles.data.map(role => {
        return ({ ...role, permissions: roles.permissions })
      })
      this.setState({ data })
      return roles.data
    })
    .then(rolesData => {
      this.setState({
        formattedData: formDataCreator(rolesData),
        status: 200
      })
      return rolesData
    })
    .then(rolesData => this.setState({ rolesData }))
    .catch(error => Error('getRoles: ', error))
}
```

createRolesPromise

Új szerepkört a `createRole(data)` függvény segítségével hozunk létre. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnak a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres létrehozás esetén bezárni a dialógus ablakot.

```
createRolesPromise = (data) => (new Promise((resolve, rejects) => {
  data = {
    name: data.name || '',
    guard_name: data.guard_name || 'api',
    permissions: data.permissionStatus || []
  }
  const { lang } = this.context
  createRole(data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.rolesData(this.state.currentPage)
        this.setState({
          errors: {},
          alertType: 'success'
        })
        this.props.alertHandler('success', lang.common.messages.success.create, lang.common.messages.success.title)
        resolve('success')
      } else {
        this.setState({
          errors: json.errors,
          alertType: 'error'
        })
        this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
        rejects('fail')
      }
    }).catch(error => Error('createRole: ', error))
}))
```

updateRolesPromise

A szerepkör adatait `updateRole(data.id, updatedData)` függvény segítségével módosítjuk. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres módosítás esetén bezárni a dialógus ablakot.

```
updateRolesPromise = (data) => (new Promise((resolve, rejects) => {
  let updatedData = {
    id: data.id,
    name: data.name || '',
    guard_name: data.guard_name || 'api',
    permissions: data.permissionStatus || []
  }
  const { lang } = this.context

  updateRole(data.id, updatedData)
    .then(res => res.json()).then(json => {
      if (json.success) {
        this.rolesData(this.state.currentPage)
        this.setState({
          errors: {},
          alertType: 'success'
        })
        this.props.alertHandler('success', lang.common.messages.success.update,
lang.common.messages.success.title)
        resolve('success')
      } else {
        this.setState({
          errors: json.errors,
          alertType: 'error'
        })
        this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
        rejects('fail')
      }
    })
    .catch(error => Error('updateRole: ', error))
}))
```

deleteRolePromise

A szerepköröket a `deleteRole(id, data)` függvény segítségével töröljük. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnak a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres törlés esetén bezárni a dialógus ablakot.

```
deleteRolePromise = (id) => (new Promise((resolve, rejects) => {
  const data = {
    id: id
  }
  const { lang } = this.context
  deleteRole(id, data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.rolesData(this.state.currentPage)
        this.setState({
          errors: {},
          alertType: 'success'
        })
        this.props.alertHandler('success', lang.common.messages.success.delete,
         resolve('success')
       } else {
         this.setState({
           errors: json.errors,
           alertType: 'error'
         })
         this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
         rejects('fail')
       }
    }).catch(error => Error('deleteRole: ', error))
}))
```

Eladó táblázat

Általános használat

```
1  import SellersTable from './components/Tables/SellersTable';
2
3  <SellersTable editable={true} />
4
```

Az eladó táblázat komponensben történik a eladóval kapcsolatos összes lekérdezés úgy mint: összes listázása, új létrehozása, szerkesztés, státuszállítás.

Függvények

sellersData

A `getSellers()` függvény segítségével lekérdezzük az aktuális oldalon lévő összes eladót (oldalanként max 15). Majd előkészítjük a megjelenítendő adatokat a táblázatnak, összefűzzük a címadatokat és hozzáadunk egy státusz mezőt az objektumba.

```
1  sellersData = (currentPage) => getSellers(currentPage)
2    .then(res => res.json())
3    .then(json => json.success)
4    .then(sellers => {
5      this.setState({ sellers })
6      return sellers.data
7    })
8    .then(sellersData => addressConcatHelper(sellersData))
9    .then(sellersData => addStatus(sellersData))
10   .then(sellersData => this.setState({ sellersData }))
11   .catch(error => console.error(error))
```

createSellerPromise

Új eladót a `createSeller(data)` függvény segítségével hozunk létre. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres létrehozás esetén bezárni a dialógus ablakot.

```
createSellerPromise = (data) => (new Promise((resolve, rejects) => {
  const isAuthSeller = data.hasOwnProperty('authenticated_seller')
  if (!isAuthSeller) {
    const authenticatedSeller = {
      authenticated_seller: false
    }
    data = { ...data, ...authenticatedSeller }
  } else {
    const authenticatedSeller = {
      authenticated_seller: true
    }
    data = { ...data, ...authenticatedSeller }
  }
  createSeller(data)
  .then(res => res.json())
```

```
   .then(json => {
     if (json.success) {
       this.sellersData(this.state.currentPage)
       this.setState({
         errors: {},
         alertType: 'success'
       })
       resolve('success')
     } else {
       this.setState({
         errors: json.errors,
         alertType: 'error'
       })
       rejects('fail')
     }
   })
}))
```

updateSellerPromise

Az eladó adatait `updateSeller(data.user_id, updatedData)` függvény segítségével módosítjuk. A nem kötelező mezőket csak módosítás esetén fűzzük az objektumhoz. A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres módosítás esetén bezárni a dialógus ablakot.

```
updateSellerPromise = (data) => {new Promise((resolve, rejects) => {
    const changeEmail = (data.user_email !== '' && data.user_email_confirmation !== '') &&
(data.user_email && data.user_email_confirmation)

    const changePass = (data.password !== '' && data.password_confirmation !== '') &&
(data.password && data.password_confirmation)

    let updatedData = {
      id: data.user_id || '',
      name: data.name || '',
      tax_number: data.tax_number || '',
      registration_number: data.registration_number || '',
      home_address_postal_code: data.home_address_postal_code || '',
      home_address_city: data.home_address_city || '',
      home_address_street: data.home_address_street || '',
      home_address_street_number: data.home_address_street_number || '',
      postal_address_postal_code: data.postal_address_postal_code || '',
      postal_address_city: data.postal_address_city || '',
      postal_address_street: data.postal_address_street || '',
      postal_address_street_number: data.postal_address_street_number || '',
      site_address_postal_code: data.site_address_postal_code || '',
      site_address_city: data.site_address_city || '',
      site_address_street: data.site_address_street || '',
      site_address_street_number: data.site_address_street_number || '',
      authenticated_seller: data.authenticated_seller || false
    }
    if (changeEmail) {
      const email = {
        email: data.email,
        email_confirmation: data.email_confirmation,
```

```
    }
    updatedData = { ...updatedData, ...email }
  }
  if (changePass) {
    const password = {
      password: data.password,
      password_confirmation: data.password_confirmation,
    }
    updatedData = { ...updatedData, ...password }
  }
  if (data.home_address_door_number !== '' && data.home_address_door_number !== null) {
    const doorNumber = {
      home_address_door_number: data.home_address_door_number
    }
    updatedData = { ...updatedData, ...doorNumber }
  }
  if (data.postal_address_door_number !== '' && data.postal_address_door_number !== null) {
    const doorNumber = {
      postal_address_door_number: data.postal_address_door_number
    }
    updatedData = { ...updatedData, ...doorNumber }
  }
  if (data.site_address_door_number !== '' && data.site_address_door_number !== null) {
    const doorNumber = {
      site_address_door_number: data.site_address_door_number
    }
    updatedData = { ...updatedData, ...doorNumber }
  }
  if (data.wired_phone !== '' && data.wired_phone !== null) {
    const wiredPhone = {
      wired_phone: data.wired_phone
    }
    updatedData = { ...updatedData, ...wiredPhone }
  }

  if (data.contact_person_name !== '' && data.contact_person_name !== null) {
    const contactPersonName = {
      contact_person_name: data.contact_person_name
    }
    updatedData = { ...updatedData, ...contactPersonName }
  }

  if (data.contact_person_email !== '' && data.contact_person_email !== null) {
    const contactPersonEmail = {
      contact_person_email: data.contact_person_email
```

```
      }
      updatedData = { ...updatedData, ...contactPersonEmail }
    }

    if (data.contact_person_mobile_phone !== '' && data.contact_person_mobile_phone !== null) {
      const contactPersonMobilePhone = {
        contact_person_mobile_phone: data.contact_person_mobile_phone
      }
      updatedData = { ...updatedData, ...contactPersonMobilePhone }
    }
    if (data.contact_person_wired_phone !== '' && data.contact_person_wired_phone !== null) {
      const contactPersonWiredPhone = {
        contact_person_wired_phone: data.contact_person_wired_phone
      }
      updatedData = { ...updatedData, ...contactPersonWiredPhone }
    }
    updateSeller(data.user_id, updatedData)
      .then(res => res.json()).then(json => {
        if (json.success) {
          this.sellersData(this.state.currentPage)
          this.setState({
            errors: {},
            alertType: 'success'
          })
          resolve('success')
        } else {
          this.setState({
            errors: json.errors,
            alertType: 'error'
          })
          rejects('fail')
        }
      })
      .catch(error => console.log(error))
  }))
```

toggleStatusSellerPromise

Az eladó státuszát `toggleStatusSeller(id, data)` függvény segítségével módosítjuk. Egy eladó státusza akkor aktív ha a `sellersData` objektumban a `user` objektum nem `null` . A Promise használatát a backend-en történő validálás indokolja. Amikor elküldjük a form adatait meg kell várnuk a backend válaszát, annak megfelelően vagy kiírni a hibaüzeneteket vagy sikeres módosítás esetén bezárni a dialógus ablakot.

```
toggleStatusSellerPromise = (id) => (new Promise((resolve, rejects) => {
    const isActiveCustomer = this.state.sellersData.filter(customer => customer.user_id === id)
[0].user !== null
    let active = true
    if (isActiveCustomer) {
      active = false
    }
    const data = {
      id,
      active
    }

    toggleStatusSeller(id, data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.sellersData(this.state.currentPage)
        this.setState({
          errors: {},
          alertType: 'success'
        })
        resolve('success')
      } else {
        this.setState({
          errors: json.errors,
          alertType: 'error'
        })
        rejects('fail')
      }
    })

  }))
```

- **AUTHORIZATION**

Authorization

Általános használat

Az Authorization függvénynek átadjuk egy tömbben, hogy mely szerepköröknek van joga a burkolt komponenst megjeleníteni.

```
1  import Authorization from './Authorization'
2
3  const AdminAuth = Authorization(['Admin', '...', ..., '...'])
4
```

Ha nem egyezik meg a belépett felhasználó jogköre az előre meghatározottakkal, akkor egy 401-es hiba oldalt töltünk be a megjeleníteni kívánt helyett.

```
1  import Admin from './pages/Admin'
2
3  AdminAuth(Admin)
4
```

COMPONENTS

Breadcrumb

A morzsamenü az URL alapján a kiindulóponttól a felhasználó jelenlegi tartózkodási helyéig vezető utat mutatja.

Properties

match	Object	required

Általános használat

```
1  import Breadcrumb from './components/Breadcrumb'
2
3  <Breadcrumb {...this.props} />
```

```
Invariant Violation: You should not use <Link> outside a <Router>
```

Dialog

Általában táblázat adatainak szerkesztéséhez használjuk formok megjelenítéséhez.

Általános használat

```
import Dialog from './components/Dialog'

<Dialog
  dialogTitle="Dialógus címe"
  onClose={
      () => {
      this.setState({
        openDialog: false
      })
    }
   }
  open={true}
>
  <Component />
</Dialog>
```

Properties

dialogTitle	String	= [Empty String]
onClose	Func	required
open	Bool	required
maxWidth	String	= "lg"

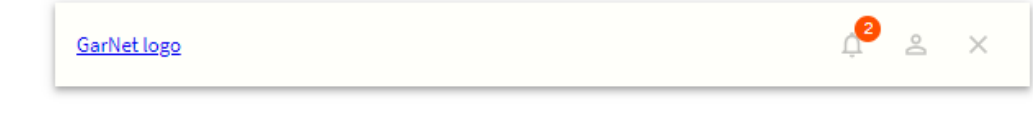
Headers

Customer header

Customer header használata

```
import Header from './components/Header'

<Header {...this.props} >
  <Navigation {...this.props} />
</Header>
```

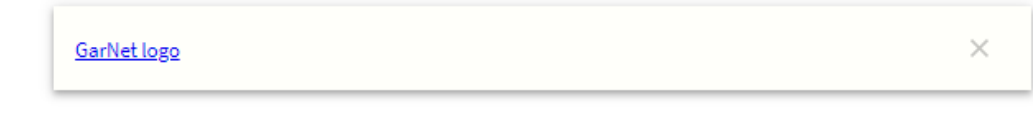
Customer header felépítése



Admin header

```
import Header from './components/Header'

<Header {...this.props} >
  <Navigation
    {...this.props}
    lang={lang}
    openMenulist={openMenulist}
    handleClose={this.handleClose}
    handleToggle={this.handleToggle}
  />
</Header>
```

Admin header felépítése



Notifications

Jobboldalró beúszó oldalsáv amely tartalmazza az értesítéseket.

Általános használat

```
import Notifications from '../componentes/Notifications'

<Notifications
  {...props}
  open={open}
  onClose={this.toggleDrawer(false)}
/>
```

Felépítése

Az oldalsávban két féle lista található, amelyek között tabokkal tudunk navigálni. Az első tabon a lejáró termékek listája, a második tabon pedig az ezekhez kapcsolódó kiküldött figyelmeztető emailek listája található.

```
<Drawer anchor="right" open={open} onClose={onClose}>
  <AppBar position="sticky">
    <Tabs value={value} onChange={this.handleChange}>
      <Tab label={lang.common.expiredProducts} />
      <Tab label={lang.common.activities} />
    </Tabs>
  </AppBar>
  {value === 0 && (
    <NotificationTabContainer>
      <NotificationList onClose={onClose} {...this.props} />
    </NotificationTabContainer>
  )}
  {value === 1 && (
    <NotificationTabContainer>
      <ActivityList onClose={onClose} />
    </NotificationTabContainer>
  )}
</Drawer>
```

NotificationList

A NotificationList komponens listaelemei kattinthatóak. Kattintásra elnavigálnak a lejáró termék oldalára.

ActivityList

Az ActivityList komponens a kiküldött figyelmeztető emailek listáját tartalmazza.

ProgressBar

A progressbar kiszámolja, a vásárlás dátumából és a hónapban megadott garancia időtartamából a garanciából eltelt időt százalékban. Ha az eltelt idő több mint 50%, akkor a kitöltés színe megváltozik zöldről narancssárgára ill. ha nagyobb mint 75% akkor pirosra.

Properties

month	Number	*= 0*
date	String	*= [Empty String]*

ProgressBar használata

```
1  import ProgressBar from './components/ProgressBar'
2
3  <ProgressBar month={item.warranty_in_month} date={invoice.purchased} />
```

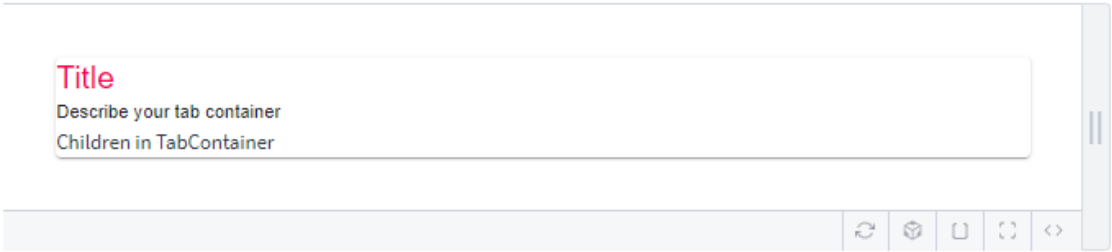
TabContainer

Properties

children	Object	=
title	String	= [Empty String]
describe	String	= [Empty String]
elevation	Number	= 0

ProgressBar használata

```
1  import TabContainer from './components/Tabs/TabContainer'
2
3  <TabContainer
4    title='Title'
5    describe='Describe your container'
6    elevation={0}>
7      <ExampleComponents />
8  </TabContainer>
```



- **CONTEXT**

NotificationContext

A context megosztja azokat az adatokat, amelyek „globálisnak” tekinthetők a React komponensek számára, mint például a hitelesített felhasználó, téma vagy a preferált nyelv. Ez a context az értesítések számát osztja meg a beburkolt komponensekkel.

Az értesítések számát csak hitelesített felhasználó érheti el. Ez a szám az olvasatlan értesítések darabszámát mutatja.

Általános használat

```
import NotificationContext from '../context/NotificationContext'
import { NotificationContext as Notification } from '../context/NotificationContext'

<NotificationContext>
  <Notification.Consumer>
    {notificatonNumber => (
      <Page {...notificatonNumber} />
    )}
  </Notification.Consumer>
</NotificationContext>
```

TranslationContext

A context megosztja azokat az adatokat, amelyek „globálisnak” tekinthetők a React komponensek számára, mint például a hitelesített felhasználó, téma vagy a preferált nyelv. Ez a context a beállított nyelvet osztja meg a beburkolt komponensekkel.

Az alkalmazás nyelvét alap esetben a böngésző nyelve határozza meg. Hitelesített felhasználó esetén viszont a felhasználó által beállított nyelv lesz az alap.

Általános használat

```
import TranslationContext from '../context/TranslationContext'
import { TranslationContext as Translate } from '../context/TranslationContext'

<TranslationContext>
  <Translate.Consumer>
    {lang => (
      <Page {...lang} />
    )}
  </Translate.Consumer>
</TranslationContext>
```

- **CUSTOMER FORMS**

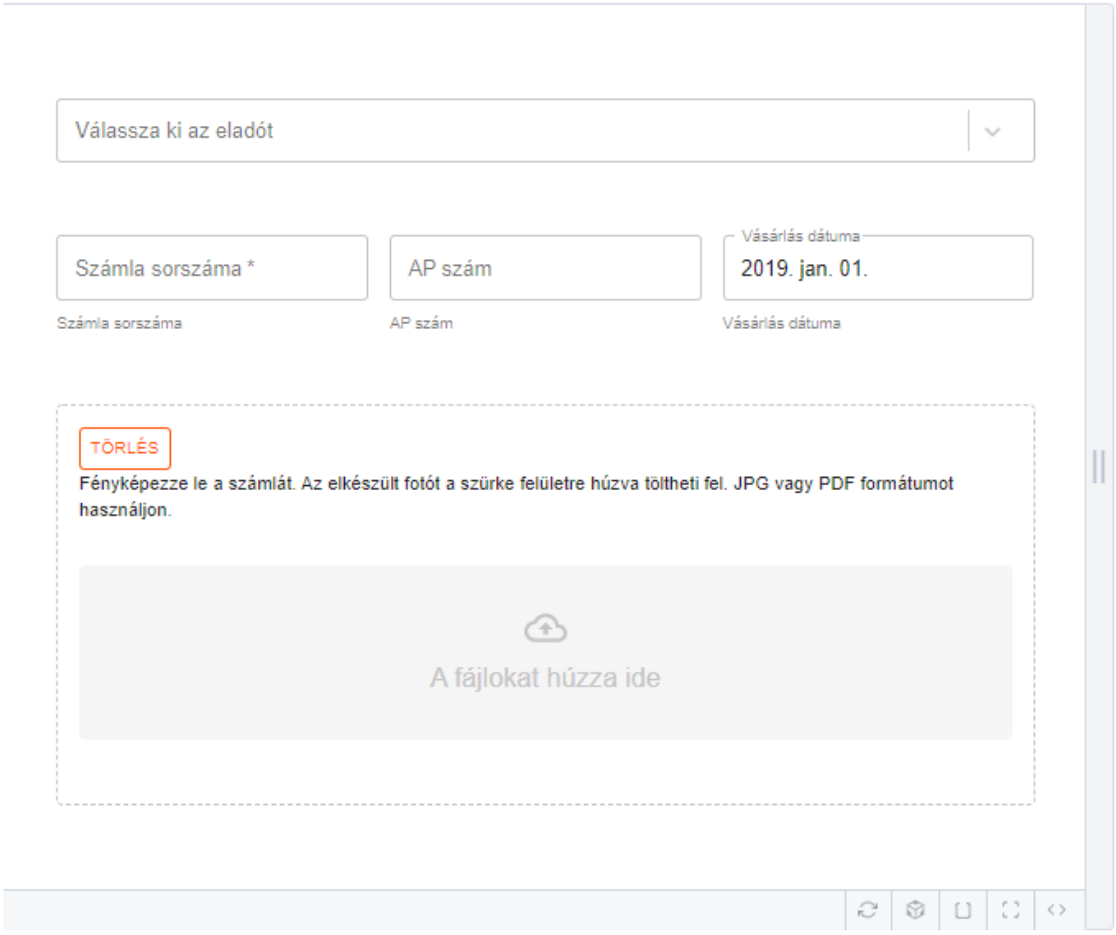
Új számla létrehozása

Számla létrehozásához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import CreateInvoiceForm from './components/Froms/CreateInvoiceForm'
```



Properties

sellers	Array	required
files	Array	= []
invoice_no	String	= [Empty String]
purchased	Any	required
selectedSeller	Object	= null
ap_no	String	= [Empty String]
submitHandler	Func	required
setSellerId	Func	= null
onChangeHandler	Func	required
handleDateChange	Func	required
invoiceDropzoneChange	Func	required
errors	Object	required

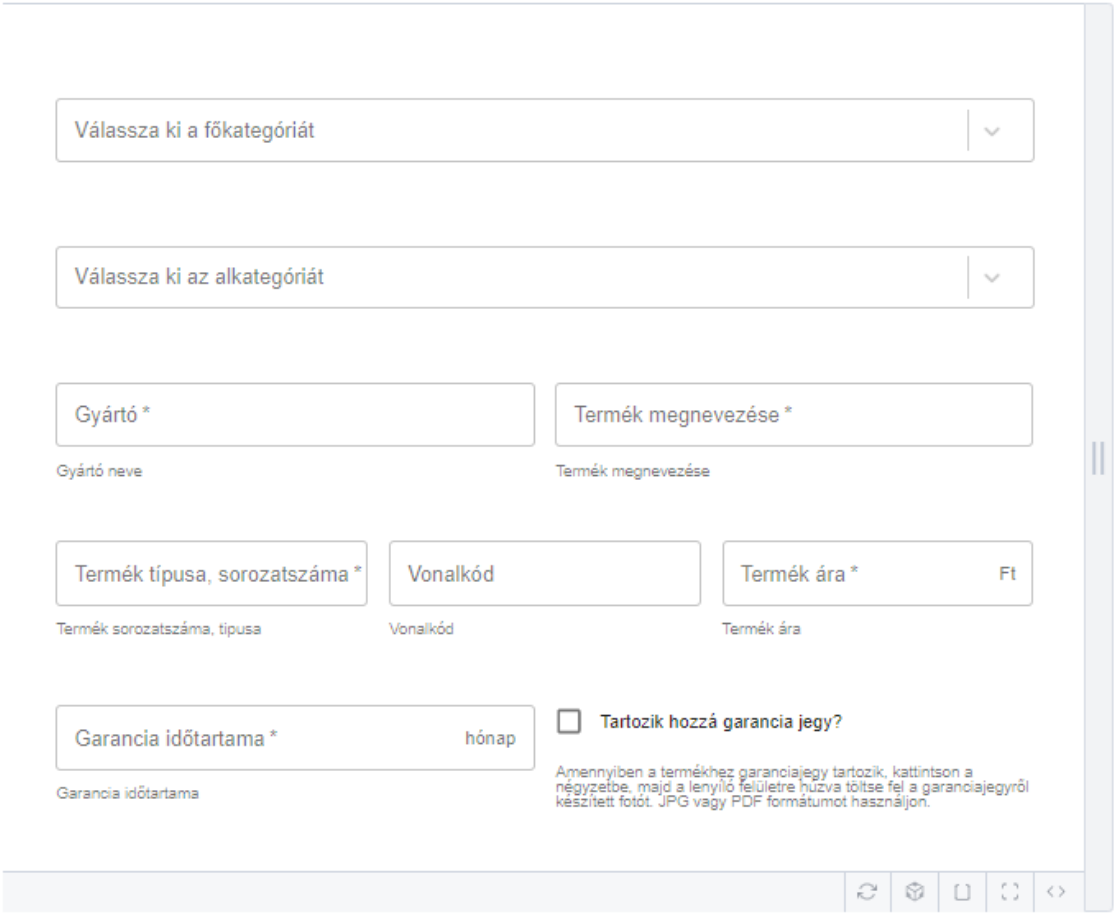
Új termék létrehozása

Termék létrehozásához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import CreateItemForm from './components/Froms/CreateItemForm'
```



Properties

name	String	*= [Empty String]*
mainCategories	Array	required
subCategories	Array	required
files	Array	*= []*
invoiceDropzoneChange	Func	required
manufacture	String	*= [Empty String]*
type_no	String	*= [Empty String]*
barcode	String	*= [Empty String]*
warranty_in_month	String	*= [Empty String]*
price	String	*= [Empty String]*
warranty	Bool	*= false*

setMainCategoryId	Func	required
selectedMainCategory	Any	*= null*
selectedSubCategory	Any	*= null*
setSubCategoryId	Func	required
onChangeHandler	Func	required
submitHandler	Func	required
extraItems	Array	required
errors	Object	required

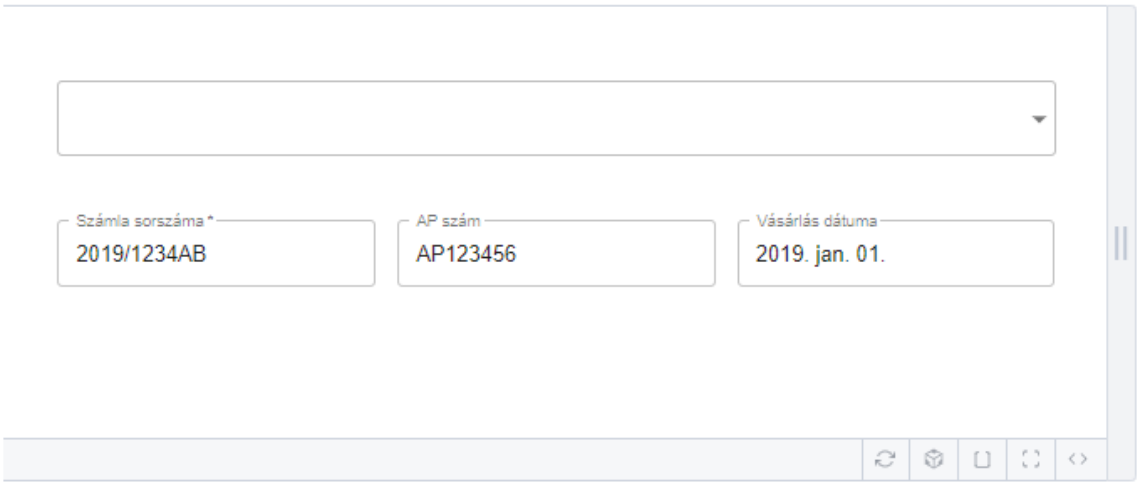
Számla módosítása

Számla módosításához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import UpdateInvoiceForm from './components/Froms/UpdateInvoiceForm'
```



Properties

sellers	Array	required
invoice_no	String	= [Empty String]
purchased	Any	required
selectedSeller	Object	= null
ap_no	String	= [Empty String]
submitHandler	Func	required
onChangeHandler	Func	required
handleDateChange	Func	required
errors	Object	required

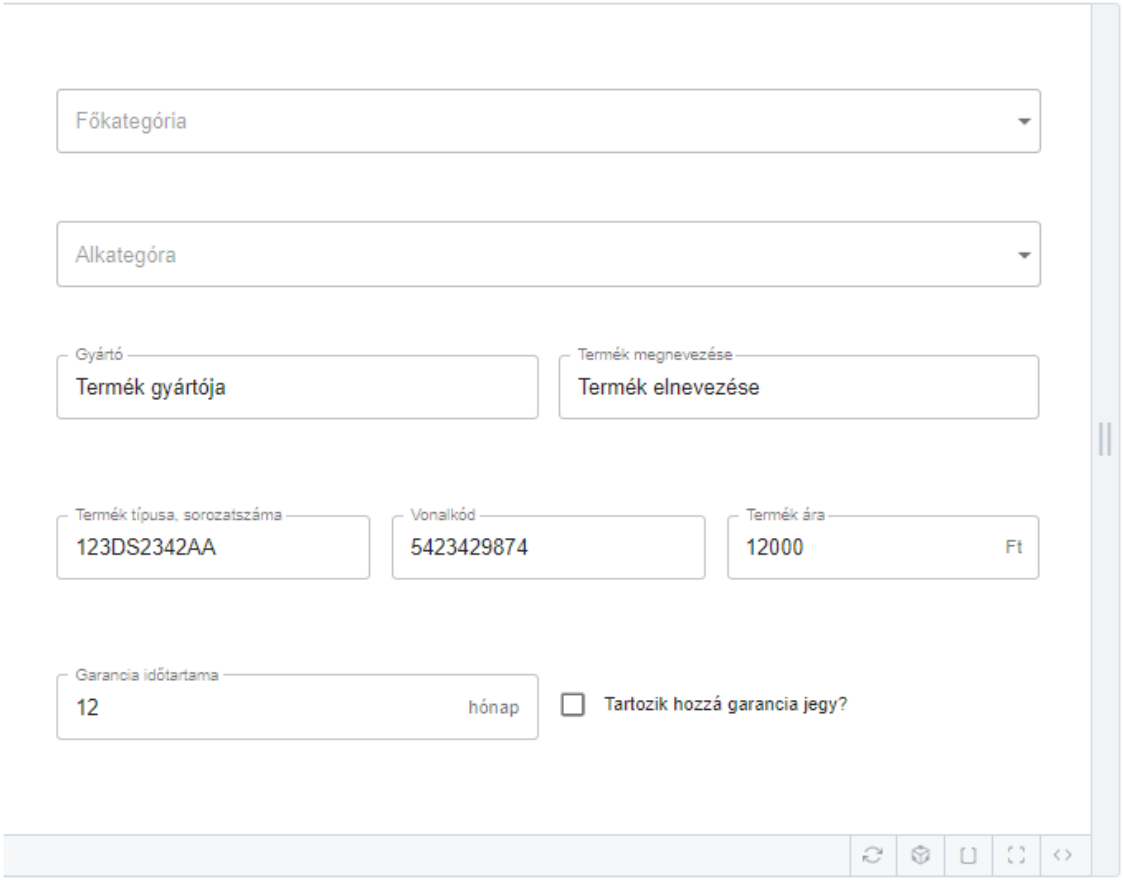
Termék módosítása

Termék módosításához szükséges űrlap.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import UpdateItemForm from './components/Froms/UpdateItemForm'
```



Properties

name	String	= [Empty String]
manufacture	String	= [Empty String]
type_no	String	= [Empty String]
barcode	String	= [Empty String]
warranty_in_month	String	= [Empty String]
price	String	= [Empty String]
warranty	Bool	= false
selectedMainCategory	Any	= null
selectedSubCategory	Any	= null
onChangeHandler	Func	required
submitHandler	Func	required
extraItems	Array	required
errors	Object	required

- **CUSTOMER DIALOGS**

Számla létrehozása

Vásárló felületéhez tartozó dialógus komponens, amely a számlák létrehozásáért felel.

Általános használat

```
import CreateInvoiceDialog from './components/Dialog/CreateInvoiceDialog'

<CreateInvoiceDialog
  maxWidth="lg"
  {...this.props}
  open={openInvoiceDialog}
  handleClose={this.handleClose}
/>
```

Properties

open	Bool	required
handleClose	Func	required
maxWidth	String	= "lg"

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását ill. a létrehozáshoz szükséges ürlapot.

```
<Dialog maxWidth={maxWidth} dialogTitle={dialogs.createInvoice.title} open={open} onClose={handleClose} >
  <DialogContent>
    <Typography variant="body1" component="p">
      {dialogs.createInvoice.text}
    </Typography>
    <CreateInvoiceForm {...this.state}
      setSellerId={this.setSellerId}
      submitHandler={this.submitHandler}
      onChangeHandler={this.onChangeHandler}
      handleDateChange={this.handleDateChange}
      invoiceDropzoneChange={this.invoiceDropzoneChange}
    />
  </DialogContent>
  <DialogActions>
```

```
15      <Button onClick={handleClose} variant="outlined" color="default" size="medium" >
    {common.buttons.cancel}</Button>
16      <Button variant="outlined" form="CreateInvoiceForm" type="submit" color="secondary"
    size="medium">{common.buttons.save}</Button>
17    </DialogActions>
18  </Dialog >
```

Statek

State	Type	Default
sellers	Array	[]
files	Array	[]
invoice_no	String	''
ap_no	String	''
sellerId	Number	null
purchased	Date	new Date()
errors	Object	{}

Függvények

getSellers

`getSellers()` függvény meghívásával a visszakapott sikeres válaszból feltölti a **sellers** nevű state tömböt az eladók id-jával és nevével.

```
getSellers(params)
  .then(res => res.json())
  .then(json => json.success)
  .then(sellers => {
    const suggestions = sellers.map(suggestion => ({ value: suggestion.id, label: suggestion.name
}))
    this.setState({ sellers: suggestions })
  }).catch(error => error)
```

onChangeHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
onChangeHandler = (e) => {
  const target = e.target.name
  const value = e.target.value
  this.setState({
    [target]: value
  })
}
```

invoiceDropzoneChange

Feltölti a **files** tömböt a feltölteni kívánt fájlokkal.

```
invoiceDropzoneChange = (files) => {
  this.setState((prevState) => ({
    files: [...prevState.files, ...files]
  }))
}
```

handleDateChange

Beállítja a dátumválasztó által kiválasztott értéket.

```
handleDateChange = (date) => {
  this.setState({ purchased: date })
}
```

setSellerId

Az eladók listából kiválasztott listaelem objektumot állítja be a **selectedSeller** értékének, valamint a **sellerId** a kiválasztott listalem értékét kapja meg. Ha nincs kiválasztott elem, akkor a **sellerId** értéke `null` .

```
setSellerId = (selected) => {
  const isSelected = selected !== null
  let sellerId = null
  let selectedSeller = {}
  if (isSelected) {
    sellerId = selected.value
    selectedSeller = selected
  }
  this.setState({
    sellerId,
    selectedSeller
  })
}
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `createInvoice()` függvénynek átadja paraméterül. Sikeres létrehozás esetén megjelenik egy sikeres létrehozás üzenet és átirányítja a számla részletes oldalára a felhasználót. Ha nem sikerült létrehozni a kategóriát, akkor az **errors** state objektum megkapja az aktuális hibaüzeneteket, amik a dialógusablakban megjelennek.

```
submitHandler = (e) => {
  e.preventDefault()
  const { sellerId, purchased, files, invoice_no, ap_no } = this.state
  const user_id = localStorage.getItem('user_id')
  const formattedPurchased = this.formatDate(purchased)
  const { lang } = this.context
  const data = {
    seller_id: sellerId,
    user_id,
    purchased: formattedPurchased,
    ap_no,
    files,
    invoice_no
  }
  createInvoice(data, user_id)
```

```
16      .then(res => res.json())
17      .then(json => {
18        if (json.success === 'true') {
19          this.props.alertHandler('success', lang.common.messages.success.update,
    lang.common.messages.success.title)
20          return json
21        } else {
22          this.setState({
23            errors: json.errors
24          })
25          this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
26        }
27      })
28      .then(json => {
29        if (json.success === 'true') {
30          return this.props.history.push(`/customer/invoices/${json.invoice_id}`)
31        }
32      }).catch(error => Error(error))
33  }
```

Termék létrehozása

Vásárló felületéhez tartozó dialógus komponens, amely a termékek létrehozásáért felel.

Általános használat

```
1  import CreateItemDialog from './components/Dialog/CreateItemDialog'
2
3  <CreateItemDialog
4    maxWidth="lg"
5    {...this.props}
6    updateData={this.toItemDetail}
7    invoice_id={invoice_id}
8    open={openNewDialog}
9    handleClose={this.handleClickClose}
10 />
```

Properties

open	Bool	required
handleClose	Func	required
maxWidth	String	= "lg"

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását ill. a létrehozáshoz szükséges ürlapot.

```
<Dialog
  maxWidth={maxWidth}
  dialogTitle={dialogs.createItem.title}
  open={open}
  onClose={handleClose}>
  <DialogContent>
  <Typography variant="body1" component="p">{dialogs.createItem.text}</Typography>
    <CreateItemForm
      {...this.state}
      close={handleClose}
      selectedMainCategory={selectedMainCategory}
      mainCategories={mainCategories}
      setMainCategoryId={this.setMainCategoryId}
      selectedSubCategory={selectedSubCategory}
      subCategories={subCategories}
       setSubCategoryId={this.setSubCategoryId}
       submitHandler={this.submitHandler}
       onChangeHandler={this.onChangeHandler}
       invoiceDropzoneChange={this.invoiceDropzoneChange}
       extraItems={extraItems} />
  </DialogContent>
  <DialogActions>
    <Button onClick={handleClose} variant="outlined" color="default" size="medium" >
{common.buttons.cancel}</Button>
    <Button variant="outlined" form="CreateItemForm" type="submit" color="secondary"
size="medium">{common.buttons.save}</Button>
  </DialogActions>
</Dialog >
```

Statek

State	Type	Default
product_category_id	Number	0
product_subcategory_id	Number	0
manufacture	String	''
files	Array	[]
mainCategories	Array	[]
subCategories	Array	[]
warranty	Boolean	false
name	String	''
type_no	String	''
barcode	String	''
price	String	''
warranty_in_month	String	''
extraItems	Array	[]
errors	Object	{}

Függvények

products

A `getProducts()` függvény meghívásával a visszakapott sikeres válaszból feltölti a **mainCategories** nevű state tömböt a főkategüriák id-jával és nevével.

```
products = () => {
  const params = ''
  return (
    getProducts(params)
      .then(res => res.json())
      .then(json => json.success)
      .then(mainCategories => {
        const suggestions = mainCategories.map(suggestion => ({ value: suggestion.id, label: suggestion.name }))
        this.setState({ mainCategories: suggestions })
      })
      .catch(error => error)
  )
}
```

onChangeHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
onChangeHandler = (e) => {
  const target = e.target.name
  const value = e.target.value
  const checked = e.target.checked
  this.setState({
    [target]: value !== 'warranty'
      ? value
      : checked
  })
}
```

invoiceDropzoneChange

Feltölti a **files** tömböt a feltölteni kívánt fájlokkal.

```
invoiceDropzoneChange = (files) => {
  this.setState((prevState) => ({
    files: [...prevState.files, ...files]
  }))
}
```

extraFormItems

A `getExtraFormItems()` függvény meghívásával lekérdezi a kiválaszott alkategória id alapján az összes extra form elemet.

```
extraFormItems = (selectedId) => {
  getExtraFormItems(selectedId)
    .then(res => res.json())
    .then(json => json.success)
    .then(extraItems => {
      let itemDefaultState
      extraItems.forEach(itemGroup => (
        itemGroup.form_item.forEach(formitem => (
          itemDefaultState = {
          ...itemDefaultState,
          [formitem.name]: ''
          }
        ))
      ))
      this.setState({
        extraItems,
        ...itemDefaultState
      })
    }).catch(error => error)
}
```

extraFormItemsValues

Visszaadja a kiválasztott termék alkategóriájához tartozó extra form elemek sateben tárolt kulcs érték párját.

```
extraFormItemsValues = (items) => {
  let values = {}
  if (items.length !== 0) {
    items.forEach(item => item.form_item.forEach(form_item => values = {
      ...values,
      [form_item.name]: this.state[form_item.name]
    }))
  }
  return values
}
```

setMainCategoryId

Főkategória kiválasztása esetén lekérdezi a főkategóriához tartozó alkategóriákat, a **product_category_id** változó értékének a kiválasztott főkategória id-ját a **selectedMainCategory** változónak pedig a kiválasztott objektumot adja értékül.

```
setMainCategoryId = (selected) => {
  const isSelected = selected !== null
  let product_category_id = null
  let selectedMainCategory = null

  if (isSelected) {
    this.getSubCategory(selected.value)
    product_category_id = selected.value
    selectedMainCategory = selected
  }

  this.setState({
    product_category_id,
    selectedMainCategory,
    selectedSubCategory: null,
    product_subcategory_id: null,
    extraItems: [],
    subCategories: []
  })

}
```

getSubCategory

Főkategória kiválasztása esetén a `getProducts()` függvény meghívásával a visszakapott sikeres válaszból feltölti a **subCategories** nevű state tömböt a főkategóriához tartozó alkategóriák id-jával és nevével.

```
getSubCategory = (mainCategoryId) => {
    if (mainCategoryId !== null) {
      getProducts(`?withParent=${mainCategoryId}`)
        .then(res => res.json())
        .then(json => json.success)
        .then(subCategories => {
          const suggestions = subCategories.map(suggestion => ({ value: suggestion.id, label: suggestion.name }))
          this.setState({ subCategories: suggestions })
        })
        .catch(error => error)
    }
  }
```

setSubCategoryId

Alkategória kiválasztása esetén lekérdezi az alkategórához tartozó extra form elemeket, a **product_subcategory_id** változó értékének a kiválasztott alkategória id-ját a **selectedSubCategory** változónak pedig a kiválasztott objektumot adja értékül.

```
setSubCategoryId = (selected) => {
  const isSelected = selected !== null
  let product_subcategory_id = null
  let selectedSubCategory = null

  if (isSelected) {
    this.extraFormItems(selected.value)
    product_subcategory_id = selected.value
    selectedSubCategory = selected
  }

  this.setState({
    product_subcategory_id,
    selectedSubCategory
  })
}
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `createInvoiceItem()` függvénynek átadja paraméterül. Sikeres létrehozás esetén megjelenik egy sikeres létrehozás üzenet és újratölti az. Ha nem sikerült létrehozni a kategóriát, akkor az **errors** state objektum megkapja az aktuális hibaüzeneteket, amik a dialógusablakban megjelennek.

```
submitHandler = (e) => {
  e.preventDefault()
  const user_id = localStorage.getItem('user_id')
  const {
    name,
    product_category_id,
    product_subcategory_id,
    manufacture,
    type_no,
    barcode,
    price,
    warranty_in_month,
    warranty,
    files,
    extraItems
```

```
  } = this.state
  const { invoice_id } = this.props
  const extraItemValues = this.extraFormItemsValues(extraItems)
  let data = {
    invoice_id: parseInt(invoice_id),
    name,
    product_category_id: product_category_id === 0 ? null : product_category_id,
    product_subcategory_id: product_subcategory_id === 0 ? null : product_subcategory_id,
    manufacture,
    type_no,
    price: parseInt(price),
    warranty_in_month: parseInt(warranty_in_month),
    warranty: warranty ? 1 : 0,
    has_extra_form_item: extraItems.length !== 0 ? 1 : 0
  }

  if (barcode !== '') {
    data = {...data, barcode }
  }
  if (warranty) {
    data = { ...data, files }
  }
  if (extraItems.length > 0) {
    data = { ...data, ...extraItemValues }
  }
  const { lang } = this.context
  createInvoiceItem(user_id, invoice_id, data)
    .then(res => res.json())
    .then(json => {
      if (json.success === 'true') {
        this.props.alertHandler('success', lang.common.messages.success.create, lang.common.messages.success.title)
        return json
      } else {
        this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
        this.setState({ errors: json.errors })
      }
    }).then(json => {
      if (json.success === 'true') {
        this.props.updateData(json.item_id)
      }
    }).catch(error => Error(error)
  )
}
```

Fájl törlése

Vásárló felületéhez tartozó dialógus komponens, amely a feltöltött fájlok törléséért felel.

Általános használat

```
import DeleteFileDialog from './components/Dialog/DeleteFileDialog'

<DeleteFileDialog
  updateData={updateData}
  updateRowId={updateRowId}
  maxWidth="md" {...this.props}
  open={openDeleteDialog}
  handleClose={this.handleClickDeleteClose}
/>
```

Properties

updateData	Func	required
updateRowId	String	required
maxWidth	String	= "md"
open	Bool	required
handleClose	Func	required

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását.

```
<Dialog maxWidth={maxWidth} dialogTitle={dialogs.deleteFile.title} open={open} onClose={handleClose} >
  <DialogContent>
    <Typography variant="body1" component="p">
      {dialogs.deleteFile.text}
    </Typography>
  </DialogContent>
  <DialogActions>
    <Button onClick={handleClose} variant="outlined" color="default" size="medium" >
      {common.buttons.cancel}
    </Button>
    <Button variant="outlined" type="submit" color="secondary" onClick={this.submitHandler} size="medium">
      {common.buttons.sure}
    </Button>
  </DialogActions>
</Dialog >
```

Függvények

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `deleteFile()` függvénynek átadja paraméterül. Sikeres törlés esetén bezárja a dialógus ablakot, újratölti az adatokat és megjelenik egy sikeres törlés üzenet. Ha nem sikerült törölni a fájlt, akkor megjelenik egy sikertelen törlés hibaüzenet.

```
submitHandler = () => {
  const { updateRowId } = this.props
  const { lang } = this.context
  const data = {
    user_id: localStorage.getItem('user_id'),
    file_id: updateRowId
  }
  deleteFile(localStorage.getItem('user_id'), updateRowId, data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.props.handleClose()
        this.props.updateData()
        this.props.alertHandler('success', lang.common.messages.success.delete,
lang.common.messages.success.title)
      } else {
        this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
      }
    }).catch(error => Error('deleteInvoice: ', error))
  }
```

Számla törlése

Vásárló felületéhez tartozó dialógus komponens, amely a számlák törléséért felel.

Általános használat

```
1  import DeleteInvoiceDialog from './components/Dialog/DeleteInvoiceDialog'
2
3  <DeleteInvoiceDialog
4    updateData={this.invoicesData}
5    updateRowId={updateRowId}
6    maxWidth="md"
7    {...this.props}
8    open={openDeleteDialog}
9    handleClose={this.handleClickDeleteClose}
10 />
```

Properties

updateData	Func	required
updateRowId	String	required
maxWidth	String	= "md"
open	Bool	required
handleClose	Func	required

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását.

```
<Dialog maxWidth={maxWidth} dialogTitle={dialogs.deleteInvoice.title} open={open} onClose={handleClose} >
  <DialogContent>
    <Typography variant="body1" component="p">{dialogs.deleteInvoice.text}</Typography>
  </DialogContent>
  <DialogActions>
    <Button onClick={handleClose} variant="outlined" color="default" size="medium">
      {common.buttons.cancel}
    </Button>
    <Button variant="outlined" type="submit" color="secondary" onClick={this.submitHandler} size="medium">
      {common.buttons.sure}
    </Button>
  </DialogActions>
</Dialog >
```

Függvények

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `deleteInvoice()` függvénynek átadja paraméterül. Sikeres törlés esetén bezárja a dialógus ablakot, újratölti az adatokat és megjelenik egy sikeres törlés üzenet. Ha nem sikerült törölni a számlát, akkor megjelenik egy sikertelen törlés hibaüzenet.

```
submitHandler = () => {
  const { updateRowId } = this.props
  const { lang } = this.context
  const data = {
    user_id: localStorage.getItem('user_id'),
    invoice_id: updateRowId
  }
  deleteInvoice(localStorage.getItem('user_id'), updateRowId, data)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        this.props.alertHandler('success', lang.common.messages.success.delete, lang.common.messages.success.title)
        this.props.handleClose()
        this.props.updateData()
```

```
15        } else {
16          this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
17        }
18      }).catch(error => Error('deleteInvoice: ', error))
19  }
```

Termék törlése

Vásárló felületéhez tartozó dialógus komponens, amely a számlákhoz tartozó termékek törléséért felel.

Általános használat

```
1  import DeleteItemDialog from './components/Dialog/DeleteItemDialog'
2
3  <DeleteItemDialog
4    updateData={this.invoicesData}
5    updateRowId={updateRowId}
6    maxWidth="md"
7    {...this.props}
8    open={openDeleteDialog}
9    handleClose={this.handleClickDeleteClose}
10 />
```

Properties

updateData	Func	required
updateRowId	String	required
maxWidth	String	= "md"
open	Bool	required
handleClose	Func	required

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását.

```
<Dialog maxWidth={maxWidth} dialogTitle={dialogs.deleteItem.title} open={open} onClose={handleClose} >
  <DialogContent>
    <Typography variant="body1" component="p">{dialogs.deleteItem.text}</Typography>
  </DialogContent>
  <DialogActions>
    <Button onClick={handleClose} variant="outlined" color="default" size="medium" >
      {common.buttons.cancel}
    </Button>
    <Button variant="outlined" type="submit" color="secondary" onClick={this.submitHandler} size="medium">
      {common.buttons.sure}
    </Button>
  </DialogActions>
</Dialog >
```

Függvények

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `deleteInvoiceItem()` függvénynek átadja paraméterül. Sikeres törlés esetén bezárja a dialógus ablakot, újratölti az adatokat és megjelenik egy sikeres törlés üzenet. Ha nem sikerült törölni a terméket, akkor megjelenik egy sikertelen törlés hibaüzenet.

```
submitHandler = () => {
  const { updateRowId, invoice_id } = this.props
  const { lang } = this.context
  const data = {
    user_id: localStorage.getItem('user_id'),
    invoice_id,
    item_id: updateRowId
  }
  deleteInvoiceItem(localStorage.getItem('user_id'), invoice_id, updateRowId, data)
    .then(res => res.json()).then(json => {
      if (json.success) {
        this.props.updateData()
        this.props.alertHandler('success', lang.common.messages.success.delete, lang.common.messages.success.title)
        this.props.handleClose()
```

```
15        } else {
16          this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
17        }
18      }).catch(error => Error('deleteInvoice: ', error))
19    }
```

Vásárló adatainak módosítása

Vásárló felületéhez tartozó dialógus komponens, amely a vásárló adatainak módosításáért felel.

Általános használat

```
1  import UpdateCustomerDataDialog from './components/Dialog/UpdateCustomerDataDialog'
2
3  <UpdateCustomerDataDialog
4    maxWidth="lg"
5    {...this.props}
6    updateData={this.customerData}
7    open={openUpdateDialog}
8    handleClose={this.handleClickClose}
9    data={customerFormData}
10   onChange={this.onChangeHandler}
11   errors={errors}
12   submitHandler={this.submitHandler}
13 />
```

Properties

open	Bool	required
data	Object	required
onChange	Func	required
errors	Object	required
submitHandler	Func	required
updateData	Func	required
handleClose	Func	required
maxWidth	String	= "lg"

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását ill. a módosításhoz szükséges ürlapot.

```
<Dialog maxWidth={maxWidth} dialogTitle={lang.dialogs.updateProfile.title} open={open} onClose={handleClose} >
  <DialogContent>
    <Grid item xs={12}>
      <Typography variant="body1" component="p">{lang.dialogs.updateProfile.text}</Typography>
    </Grid>
    <Grid item xs={12} className="spacing"></Grid>
    <CustomerUpdateForm {...data} onChange={onChange} errors={errors} />
  </DialogContent >
  <DialogActions>
    <Button onClick={handleClose} color='default' variant='outlined'>
      {lang.common.buttons.cancel}
    </Button>
    <Button form="CustomerUpdateForm" onClick={submitHandler} color='secondary' variant='outlined'>
      {lang.common.buttons.save}
    </Button>
  </DialogActions>
</Dialog >
```

Számla módosítása

Vásárló felületéhez tartozó dialógus komponens, amely a számla módosításáért felel.

Általános használat

```
1  import UpdateInvoiceDialog from './components/Dialog/UpdateInvoiceDialog'
2
3  <UpdateInvoiceDialog
4    updateData={this.invoicesData}
5    maxWidth="lg"
6    {...this.props}
7    updateRowId={updateRowId}
8    open={openInvoiceUpdateDialog}
9    handleClose={this.handleClickUpdateClose}
10 />
```

Properties

open	Bool	required
updateRowId	String	required
updateData	Func	required
handleClose	Func	required
maxWidth	String	*= "lg"*

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását, a dátumválasztóhoz tartozó provider-t ill. a létrehozáshoz szükséges ürlapot.

```
<Dialog maxWidth={maxWidth} dialogTitle={dialogs.updateInvoice.title} open={open} onClose={handleClose} >
  <MuiPickersUtilsProvider utils={DateFnsUtils} locale={huLocale}>
    <DialogContent>
      <Typography variant="body1" component="p">{dialogs.updateInvoice.text}</Typography>
      <UpdateInvoiceForm {...this.state}
        close={handleClose}
        submitHandler={this.submitHandler}
        setSellerId={this.setSellerId}
        onChangeHandler={this.onChangeHandler}
        handleDateChange={this.handleDateChange}
      />
    </DialogContent>
    <DialogActions>
      <Button onClick={handleClose} variant="outlined" color="default" size="medium" >{buttons.cancel}</Button>
      <Button form="UpdateInvoiceForm" variant="outlined" type="submit" color="secondary" size="medium">{buttons.save}</Button>
    </DialogActions>
  </MuiPickersUtilsProvider>
</Dialog >
```

Statek

State	Type	Default
sellers	Array	[]
files	Array	[]
invoice_no	String	''
ap_no	String	''
sellerId	Number	null
updateRowId	Number	null
purchased	Date	new Date()
selectedSeller	Object	{}
errors	Object	{}

Függvények

getInvoiceData

A `getInvoiceByID()` függvény meghívásával a paraméterül kapott id segítségével lekérdezi a számla adatait a visszakapott sikeres válaszból a stateknek átadja az értékeket.

```
getInvoiceData = (user_id, updateRowId, params) => {
    getInvoiceByID(user_id, updateRowId, params)
      .then(res => res.json())
      .then(json => json.success[0])
      .then(invoiceData => {
        this.setState({
          sellerId: invoiceData.seller_id,
          selectedSeller: {
            value: invoiceData.seller.id, label: invoiceData.seller.name
          },
          files: invoiceData.uploads,
          ap_no: invoiceData.ap_no === null ? '' : invoiceData.ap_no,
          invoice_no: invoiceData.invoice_no,
          purchased: new Date(invoiceData.purchased)
        })
```

```
16        })
17        .catch(error => Error('getInvoiceByID: ', error))
18    }
```

sellers

A `getSellers()` függvény meghívásával a visszakapott sikeres válaszból feltölti a **sellers** nevű state tömböt az eladók id-jával és nevével.

```
1  sellers = () => {
2    const params = ''
3    return (
4      getSellers(params)
5        .then(res => res.json())
6        .then(json => json.success)
7        .then(sellers => {
8          const suggestions = sellers.map(suggestion => ({ value: suggestion.id, label:
   suggestion.name }))
9          this.setState({ sellers: suggestions })
10       }).catch(error => error)
11   )
12 }
```

setSellerId

Az eladók listából kiválasztott listaelem objektumot állítja be a **selectedSeller** értékének, valamint a **sellerId** a kiválasztott listalem értékét kapja meg. Ha nincs kiválasztott elem, akkor a **sellerId** értéke `null` .

```
1  setSellerId = (selected) => {
2      const isSelected = selected !== null
3      if (isSelected) {
4        this.setState({
5          sellerId: selected.value,
6          selectedSeller: selected
7        })
8      } else {
9        this.setState({
10         sellerId: null
11       })
12     }
13   }
```

onChangeHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
onChangeHandler = (e) => {
  const target = e.target.name
  const value = e.target.value
  this.setState({
    [target]: value
  })
}
```

handleDateChange

Beállítja a dátumválasztó által kiválasztott értéket.

```
handleDateChange = (date) => {
  this.setState({ purchased: date })
}
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `updateInvoice()` függvénynek átadja paraméterül. Sikeres módosítás esetén bezárja a dialógus ablakot, újratölti az adatokat és megjelenik egy sikeres módosítás üzenet. Ha nem sikerült módosítani a számla adatokat, akkor az **errors** state objektum megkapja az aktuális hibaüzeneteket, amiket a dialógusablakban megjelenítünk.

```
submitHandler = (e) => {
    e.preventDefault()
    const { updateRowId } = this.props
    const { purchased, invoice_no, ap_no, sellerId } = this.state
    const { lang } = this.context
    const user_id = localStorage.getItem('user_id')
    const day = purchased.getDate() < 10 ? `0${purchased.getDate()}` : purchased.getDate()
    const month = (purchased.getMonth() + 1) < 10 ? `0${purchased.getMonth() + 1}` :
purchased.getMonth() + 1
    const year = purchased.getFullYear()
    const data = {
      seller_id: sellerId,
      invoice_id: updateRowId,
      user_id,
      purchased: `${year}-${month}-${day}`,
```

```
15        ap_no,
16        invoice_no
17      }
18      updateInvoice(user_id, updateRowId, data)
19        .then(res => res.json())
20        .then(json => {
21          if (json.success) {
22            this.props.updateData()
23            this.props.alertHandler('success', lang.common.messages.success.update,
    lang.common.messages.success.title)
24            this.props.handleClose()
25          } else {
26            this.setState({
27              errors: json.errors
28            })
29          }
30        }).catch(error => Error(error))
31    }
```

Termék módosítása

Vásárló felületéhez tartozó dialógus komponens, amely a termék módosításáért felel.

Általános használat

```
import UpdateItemDialog from './components/Dialog/UpdateItemDialog'

<UpdateItemDialog
  maxWidth="lg"
  updateData={this.getItem}
  {...this.props}
  invoice_id={invoice_id}
  updateRowId={updateRowId}
  open={openItemUpdateDialog}
  handleClose={this.handleClickUpdateClose}
/>
```

Properties

open	Bool	required
updateRowId	String	required
invoice_id	String	required
updateData	Func	required
handleClose	Func	required
maxWidth	String	= "lg"

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását ill. a módosításhoz szükséges ürlapot.

```
<Dialog maxWidth={maxWidth} dialogTitle={dialogs.updateItem.title} open={open} onClose={handleClose} >
  <DialogContent>
    <Typography variant="body1" component="p">{dialogs.updateItem.text}</Typography>
    <UpdateItemForm
      {...this.state}
      selectedMainCategory={selectedMainCategory}
      selectedSubCategory={selectedSubCategory}
      submitHandler={this.submitHandler}
      onChangeHandler={this.onChangeHandler}
    />
  </DialogContent>
  <DialogActions>
    <Button onClick={handleClose} variant="outlined" color="default" size="medium" >{common.buttons.cancel}</Button>
    <Button variant="outlined" form="UpdateItemForm" type="submit" color="secondary" size="medium">{common.buttons.save}</Button>
  </DialogActions>
</Dialog >
```

Statek

State	Type	Default
product_category_id	Number	0
product_subcategory_id	Number	0
manufacture	String	''
warranty	Boolean	false
name	String	''
type_no	String	''
barcode	String	''
price	String	''
warranty_in_month	String	''
extraItems	Array	[]
selectedMainCategory	Object	{}
selectedSubCategory	Object	{}
errors	Object	{}

Függvények

getInvoiceData

A `getItemData()` függvény meghívásával a paraméterül kapott számla id és termék id segítségével lekérdezi a termék adatait a visszakapott sikeres válaszból a stateknek átadja az értékeket.

```
getItemData = () => {
  const { invoice_id, updateRowId } = this.props
  const params = '?withRelation=true'
  const user_id = localStorage.getItem('user_id')

  getItemById(user_id, invoice_id, updateRowId, params)
    .then(res => res.json())
    .then(json => json.success)
    .then(item => {
      this.extraFormItems(item[0].product_subcategory_id, item[0].customer_invoice_extra_item)
      this.setState({
        product_category_id: item[0].product_category_id,
        product_subcategory_id: item[0].product_subcategory_id,
        manufacture: item[0].manufacture,
        warranty: item[0].warranty === 0 ? false : true,
        name: item[0].name,
        type_no: item[0].type_no,
        barcode: item[0].barcode,
        price: item[0].price,
        selectedMainCategory: { value: item[0].product_category_id, label:
item[0].product_category.name },
        selectedSubCategory: { value: item[0].product_subcategory_id, label:
item[0].product_sub_category.name },
        warranty_in_month: item[0].warranty_in_month,
      })
    }).catch(error => error)
}
```

onChangeHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
onChangeHandler = (e) => {
  const target = e.target.name
  const value = e.target.value
  const checked = e.target.checked
  this.setState({
    [target]: value !== 'warranty' ? value : checked
  })
}
```

extraFormItems

A `getExtraFormItems()` függvény meghívásával lekérdezi a kiválaszott alkategória id alapján az összes extra form elemet.

```
extraFormItems = (selectedId, extra_items) => {
  getExtraFormItems(selectedId)
    .then(res => res.json())
    .then(json => json.success)
    .then(extraItems => {
      let customer_invoice_extra_item = {}
      extraItems.forEach(itemGroup => extra_items.find(item => {
        if (item.form_group_id === itemGroup.id) {
          itemGroup.form_item.forEach(formitem => (
            customer_invoice_extra_item = { ...customer_invoice_extra_item, [formitem.name]: item.form_item_value }
          ))
        }
        return customer_invoice_extra_item
      }))
      this.setState({ extraItems, ...customer_invoice_extra_item })
    })
    .catch(error => error)
}
```

extraFormItemsValues

Visszaadja a kiválasztott termék alkategóriájához tartozó extra form elemek sateben tárolt kulcs érték párját.

```
extraFormItemsValues = (items) => {
  let values = {}
  if (items.length !== 0) {
    items.forEach(item => item.form_item.forEach(form_item => values = {
      ...values,
      [form_item.name]: this.state[form_item.name]
    }))
  }
  return values
}
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `updateInvoiceItem()` függvénynek átadja paraméterül. Sikeres módosítás esetén bezárja a dialógus ablakot, újratölti az adatokat és megjelenik egy sikeres módosítás üzenet. Ha nem sikerült módosítani a termék adatokat, akkor az **errors** state objektum megkapja az aktuális hibaüzeneteket, amiket a dialógusablakban megjelenítünk.

```
submitHandler = (e) => {
  e.preventDefault()
  const user_id = localStorage.getItem('user_id')
  const { lang } = this.context
  const {
    name, product_category_id, product_subcategory_id,
    manufacture, type_no, barcode, price, warranty_in_month,
    warranty, extraItems
  } = this.state
  const { invoice_id, updateRowId } = this.props
  const extraItemValues = this.extraFormItemsValues(extraItems)
  let data = {
    invoice_id,
    name,
    product_category_id,
    product_subcategory_id,
    manufacture,
    type_no,
    barcode,
    price,
    warranty_in_month,
    warranty: warranty ? 1 : 0,
    has_extra_form_item: extraItems.length !== 0 ? 1 : 0
  }

  if (extraItems.length > 0) {
    data = { ...data, ...extraItemValues }
  }
  updateInvoiceItem(user_id, invoice_id, updateRowId, data)
    .then(res => res.json())
```

```
31      .then(json => {
32        if (json.success) {
33          this.props.updateData()
34          this.props.alertHandler('success', lang.common.messages.success.update,
    lang.common.messages.success.title)
35          this.props.handleClose()
36        } else {
37          this.setState({
38            errors: json.errors
39          })
40        }
41      }).catch(error => Error(error))
42  }
```

Fájl feltöltése

Vásárló felületéhez tartozó dialógus komponens, amely a fájlok feltöltéséért felel.

Általános használat

```
1  import UploadFileDialog from './components/Dialog/UploadFileDialog'
2
3  <UploadFileDialog
4    updateData={updateData}
5    maxWidth="lg"
6    {...this.props}
7    open={openNewDialog}
8    handleClose={this.handleClickClose}
9  />
```

Properties

open	Bool	required
updateData	Func	required
handleClose	Func	required
maxWidth	String	= "lg"

Felépítés

A komponens tartalmazza a dialógusablak címét, leírását ill. a feltöltéshez szükséges ürlapot.

```
<Dialog
 maxWidth={maxWidth}
 dialogTitle={dialogs.uploadFiles.title}
 open={open}
 onClose={handleClose}>
 <DialogContent>
   <Typography variant="body1" component="p">{dialogs.uploadFiles.text}</Typography>
   <FormControl fullWidth margin='normal' error={!!errors.hasOwnProperty('files')}>
     <DropzoneWithPreview files={files} dropzoneChange={this.invoiceDropzoneChange} />
     <FormHelperText id='component-error-text'>
       {errors.hasOwnProperty('files') && errors.files[0]}
     </FormHelperText>
   </FormControl>
 </DialogContent>
 <DialogActions>
   <Button
     onClick={handleClose}
     variant="outlined"
     color="default"
     size="medium"
   >{common.buttons.cancel}</Button>
   <Button
     variant="outlined"
     type="submit"
     color="secondary"
     onClick={this.submitHandler}
     size="medium">{common.buttons.upload}</Button>
 </DialogActions>
</Dialog >
```

Statek

State	Type	Default
files	Array	[]
errors	Object	{}

Függvények

invoiceDropzoneChange

Feltölti a **files** tömböt a feltölteni kívánt fájlokkal.

```
1  invoiceDropzoneChange = (files) => {
2    this.setState((prevState) => ({
3      files: [...prevState.files, ...files]
4    }))
5  }
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a `uploadFiles()` függvénynek átadja paraméterül. Sikeres feltöltés esetén bezárja a dialógus ablakot, újratölti az adatokat és megjelenik egy sikeres feltöltés üzenet. Ha nem sikerült feltölteni a fájlokat, akkor az **errors** state objektum megkapja az aktuális hibaüzeneteket, amiket a dialógusablakban megjelenítünk.

```
1  submitHandler = () => {
2    const invoice_id = this.props.match.params.invoice_id
3    const { files } = this.state
4    const { lang } = this.context
5    let data = {
6      invoice_id,
7      user_id: localStorage.getItem('user_id'),
8      files
9    }
10   if (this.props.match.params.hasOwnProperty('item_id')) {
11     const invoice_item_id = this.props.match.params.item_id
12     data = {
13       ...data,
14       invoice_item_id
15     }
```

```
16    }
17
18    uploadFiles(localStorage.getItem('user_id'), data)
19      .then(res => res.json())
20      .then(json => {
21        if (json.success) {
22          this.props.handleClose()
23          this.props.updateData()
24          this.props.alertHandler('success', lang.common.messages.success.upload,
    lang.common.messages.success.title)
25        } else {
26          this.props.alertHandler('error', json.message, lang.common.messages.errors.fail)
27          this.setState({
28            errors: json.errors
29          })
30        }
31      })
32      .catch(error => Error('uploadFiles: ', error))
33  }
```

- **CUSTOMER TABLES**

AllProductsTable

A táblázat a felhasználóhoz tartozó összes terméket megjeleníti.

Általános használat

```
import AllProductsTable from '../componentes/Tables/AllProductsTable'

<AllProductsTable {...this.props} />
```

Függvények

getItem

A `getItems()` függvény segítségével lekérdezzük a felhasználóhoz tartozó összes terméket (oldalanként max 15). Majd előkészítjük a megjelenítendő adatokat a táblázatnak, összefűzzük a kategóriát, a gyártót és típust ill. az objektumhoz hozzáadjuk a feltöltött dokumentumok darabszámát és a szála sorszámát.

```
getItem = (currentPage) => {
  currentPage = currentPage === undefined && this.state.currentPage
  const user_id = localStorage.getItem('user_id')
  const params = `?page=${currentPage}&withPaginate=true&withRelation=true`
  const { lang } = this.context
  getItems(user_id, params)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        return json.success
      } else {
        this.props.alertHandler('error', json.errors.id[0], lang.common.messages.errors.fail)
        this.setState({
          status: 200
        })
```

```
        return {}
      }
    })
    .then(items => {
      let productsData = items.data
      productsData = productsData.filter(product => product.invoice !== null)
      productsData = productsData.map(product => ({
        ...product,
        'category': `${product.product_category.name}<br/>
<small>${product.product_sub_category.name}</small>`,
        'manufacture_type': `${product.manufacture}<br/><small>${product.type_no}</small>`,
        'invoice_prod': `<a
href="/customer/invoices/${product.invoice_id}">${product.invoice.invoice_no}</a>`,
        'prod_uploads': `${product.uploads.length} db`,
        'price_ft': `${product.price} Ft`,
      }))
      this.setState({
        productsData,
        products: items,
        status: 200
      })
    })
    .catch(error => error)
}
```

DocumentsTable

A táblázat a feltöltött dokumentumokat jeleníti meg.

Általános használat

```
import DocumentsTable from '../componentes/Tables/DocumentsTable'

<DocumentsTable
  status={status}
  {...this.props}
  uploads={uploads}
  updateData={this.getItem}
/>
```

Függvények

downloadHandler

A `getFileDownload()` függvény segítségével lekérdezzük az aktuális fájlt, majd a visszakapott választ átalakítjuk egy fájl szerű objektummá ezt átadjuk a `saveAs()` függvénynek amely elindítja a fájl letöltését a böngészőben.

```
downloadHandler = (e) => {
  const rowId = e.currentTarget.dataset.rowId
  const userId = localStorage.getItem('user_id')
  const fileName = e.currentTarget.dataset.rowFilename
  const disposition = 2

  getFileDownload(userId, rowId, disposition)
    .then(response => response.blob())
    .then(blob => saveAs(blob, fileName))
    .catch(error => console.error(error))
}
```

ExpiredProductsTable

A táblázat a felhasználóhoz tartozó összes hamarosan lejáró terméket megjeleníti.

Általános használat

```
import ExpiredProductsTable from '../componentes/Tables/ExpiredProductsTable'

<ExpiredProductsTable {...this.props} />
```

Függvények

getItem

A `getExpiredItems()` függvény segítségével lekérdezzük a felhasználóhoz tartozó összes hamarosan lejáró terméket (oldalanként max 15). Majd előkészítjük a megjelenítendő adatokat a táblázatnak, összefűzzük a kategóriát, a gyártót és típust ill. az objektumhoz hozzáadjuk a feltöltött dokumentumok darabszámát és a szála sorszámát.

```
getItem = (currentPage) => {
  currentPage = currentPage === undefined && this.state.currentPage
  const user_id = localStorage.getItem('user_id')
  const params = `?page=${currentPage}&withPaginate=true&withRelation=true`
  const { lang } = this.context
  getExpiredItems(user_id, params)
    .then(res => res.json())
    .then(json => {
      if (json.success) {
        return json.success
      } else {
        this.props.alertHandler('error', json.errors.id[0], lang.common.messages.errors.fail)
        this.setState({
          status: 200
        })
```

```
16          return {}
17        }
18      })
19      .then(items => {
20        let expiredProductsData = items.data
21        expiredProductsData = expiredProductsData.filter(product => product.item.invoice !== null)
22        expiredProductsData = expiredProductsData.map(product => ({
23          ...product.item,
24          'category': `${product.item.product_category.name}<br/>
<small>${product.item.product_sub_category.name}</small>`,
25          'manufacture_type': `${product.item.manufacture}<br/><small>${product.item.type_no}
</small>`,
26          'invoice_prod': `<a
href="/customer/invoices/${product.item.invoice_id}">${product.item.invoice.invoice_no}</a>`,
27          'prod_uploads': `${product.item.uploads.length} db`,
28          'price_ft': `${product.item.price} Ft`,
29        }))
30        this.setState({
31          expiredProductsData,
32          products: items,
33          status: 200
34        })
35      })
36      .catch(error => console.log(error))
37  }
```

InvoicesTable

A táblázat a felhasználóhoz tartozó összes számlát megjeleníti.

Általános használat

```
import InvoicesTable from '../componentes/Tables/InvoicesTable'

<InvoicesTable {...this.props} />
```

Függvények

invoicesData

A `getInvoices()` függvény segítségével lekérdezzük a felhasználóhoz tartozó összes számlát (oldalanként max 15).

```
invoicesData = (currentPage) => {
  currentPage = currentPage === undefined && this.state.currentPage
  const params = `?page=${currentPage}&withPaginate=true`
  getInvoices(localStorage.getItem('user_id'), params).then(res => res.json())
    .then(json => {
      const { lang } = this.context
      if (json.success) {
        return json.success
      } else {
        this.props.alertHandler('error', json.errors.id[0], lang.common.messages.errors.fail)
        this.setState({
          status: 200
        })
        return {}
      }
    })
    .then(invoices => {
      this.setState({ invoices })
      return invoices
    })
    .then(invoices => this.setState({ invoicesData: invoices.data, status: 200 }))
    .catch(error => Error('invoicesData: ', error))
}
```

ProductsTable

A táblázat az aktuális számlához tartozó összes terméket megjeleníti.

Általános használat

```
import ProductsTable from '../componentes/Tables/ProductsTable'

<ProductsTable {...this.props} />
```

Függvények

getItem

A `getInvoiceItems()` függvény segítségével lekérdezzük az akutális számlához tartozó összes terméket (oldalanként max 15). Majd előkészítjük a megjelenítendő adatokat a táblázatnak, összefűzzük a kategóriát, a gyártót és típust.

```
getItem = (currentPage) => {
 currentPage = currentPage === undefined && this.state.currentPage
 const { match } = this.props
 const invoice_id = match.params.invoice_id
 const user_id = localStorage.getItem('user_id')
 const params = `?page=${currentPage}&withPaginate=true&withRelation=true`

 getInvoiceItems(user_id, invoice_id, params)
   .then(res => {
     this.setState({
       status: res.status
     })
     return res.json()
   })
   .then(json => {
```

```
      const { lang } = this.context
      if (json.success) {
        return json.success
      } else {
        this.props.alertHandler('error', json.errors.id[0], lang.common.messages.errors.fail)
        this.setState({
          status: 200
        })
        return {}
      }
    })
    .then(products => {
      this.setState({ products })
      return products
    })
```

```
    .then(products => {
      let formattedProducts = products.data
      formattedProducts = formattedProducts.map(product => (
        {
          ...product,
          'category': `${product.product_category.name}<br/>
<small>${product.product_sub_category.name}</small>`,
          'manufacture_type': `${product.manufacture}<br/><small>${product.type_no}</small>`,
          'price_ft': `${product.price} Ft`
        }
      ))
      this.setState({
        formattedProducts
      })
    }).catch(error => error)
}
```

- **FORMS**

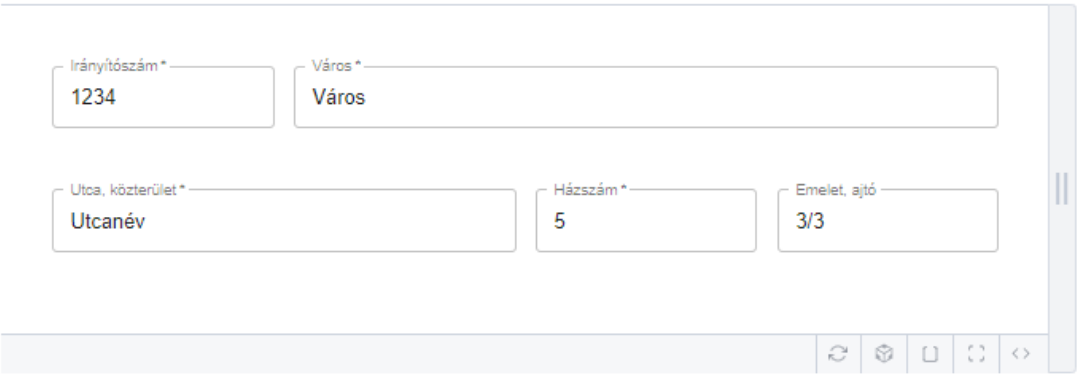
Address Form

Kiegészítő form komponens, a címek megjelenítéséhez szükséges.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import AddressForm from './components/Forms/AddressForm'
```



Properties

data	Object	required
errors	Object	required
id	Number	required
onChange	Func	required
addressType	String	required

Autocomplete select

Kiegészítő form komponens. Beviteli mező amely gépelés hatására találati lehetőségeket javasol.

Általános használat

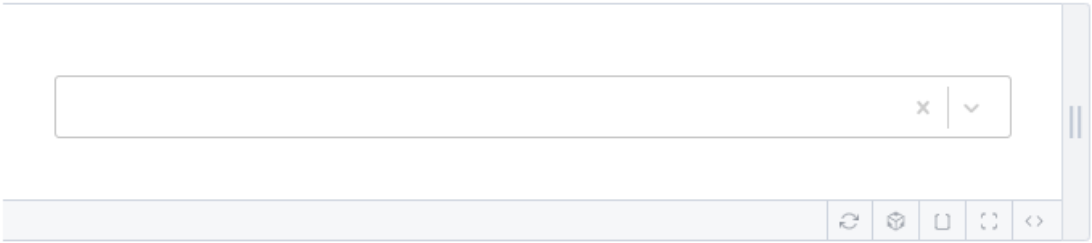
Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
import AutocompleteSelect from './components/Forms/AutocompleteSelect'

<AutocompleteSelect
  suggestions
  selected
  setValue
  placeholder
/>
```

Properties

Property	Type	Required	Default
classes	Object	true	-
theme	Object	true	-
suggestions	Array	true	-
selected	Object	false	{}
placeholder	String	false	[Empty String]
disabled	Boolean	false	false

Felépítés



Change Email Form

Kiegészítő form komponens, az email cím megváltoztatásához szükséges.

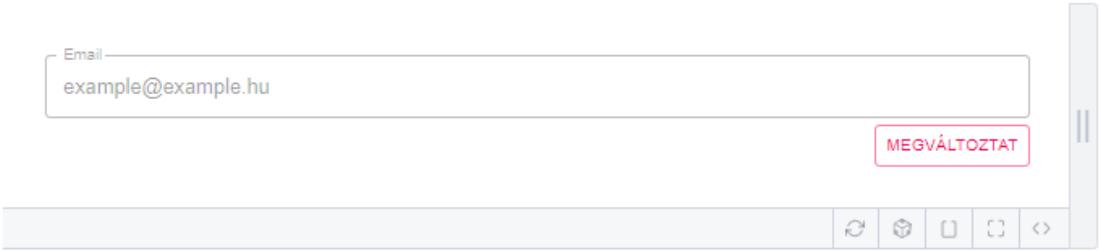
Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import ChangeEmailForm from './components/Forms/ChangeEmailForm'
2
3  <ChangeEmailForm onChange={onChange} id={id} user_email={user_email}/>
```

Properties

onChange	Func	= () => { }
id	Number	= 0
user_email	String	= [Empty String]



Change Password Form

Kiegészítő form komponens, a jelszó megváltoztatásához szükséges.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import ChangePasswordForm from './components/Forms/ChangePasswordForm'
2
3  <ChangePasswordForm onChange={onChange} id={id} />
```

Properties

onChange	Func	= () => { }
id	Number	= 0

Contact Person Form

Kiegészítő form komponens, a kapcsolattartó adatainak megjelenítéséhez szükséges.

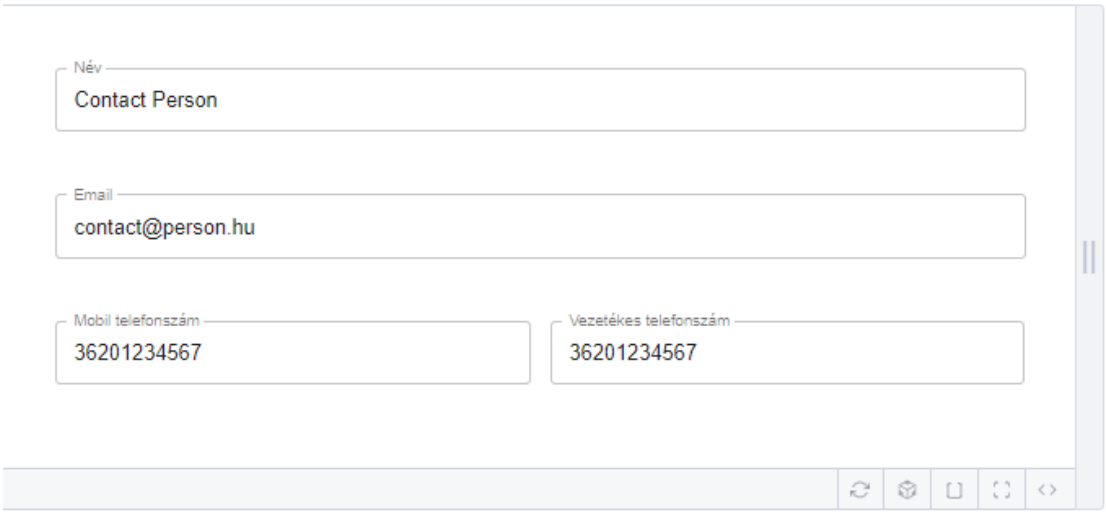
Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
import ContactPersonForm from './components/Forms/ContactPersonForm'

<ContactPersonForm
  onChange={onChange}
  id={id}
  contactPersonData={{
    contact_person_name,
    contact_person_email,
    contact_person_mobile_phone,
    contact_person_wired_phone
  }}
/>
```

Properties

onChange	Func	= () => { }
contactPersonData	Object	= { contact_person_name: "", contact_person_email: "", contact_person_mobile_phone: "", contact_person_wired_phone: "" }
id	Number	= 0
errors	Object	= {}


Név
Contact Person
Email
contact@person.hu
Mobil telefonszám
36201234567
Vezetékes telefonszám
36201234567

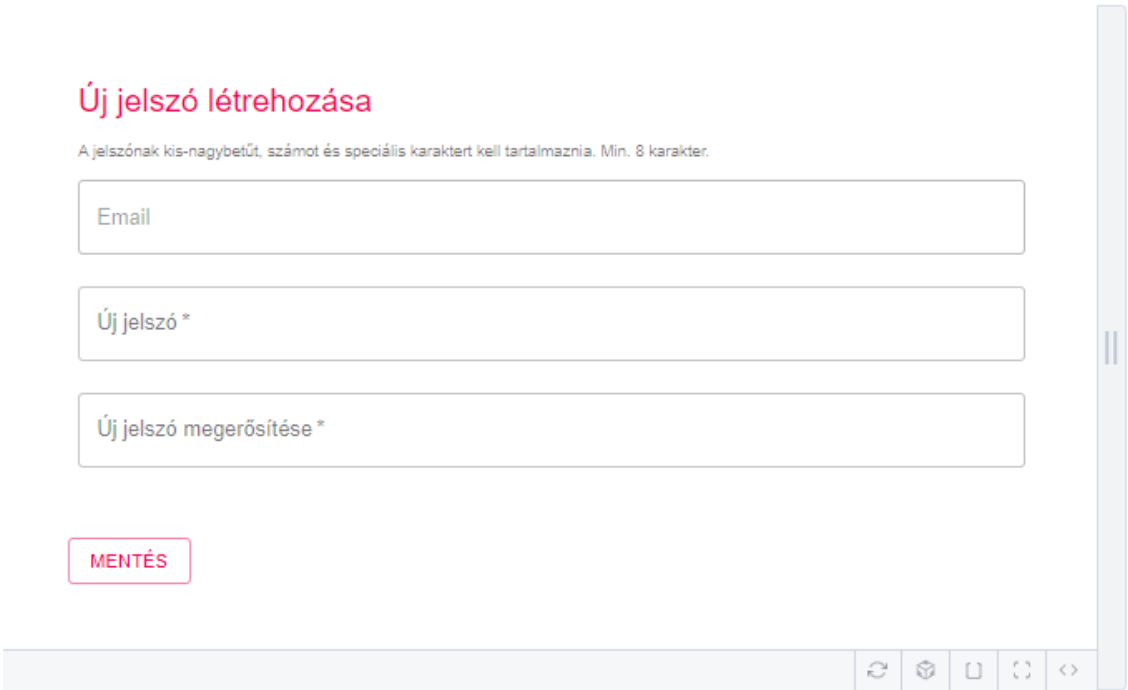
Create New Password Form

Általános használat

```
import CreateNewPassForm from './components/Forms/CreateNewPassForm'

<CreateNewPassForm
  token={token}
  email={email}
  error={error}
  createNewPass={this.createNewPass}
  confirmPass={this.confirmPass}
/>
```

Properties

createNewPass	Func	= () => { }
token	String	= [Empty String]
email	String	= [Empty String]
confirmPass	Func	= () => { }
error	Bool	= false



Dropzone

Kiegészítő form komponens, a fájlok feltöltéséhez szükséges.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import DropzoneWithPreview from './components/Forms/DropzoneWithPreview'
```

Email Form

Kiegészítő form komponens, az emailcímek beírásához szükséges.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import EmailForm from './components/Forms/EmailForm'
```

Properties

id	Number	required
errors	Object	= {}
onChange	Func	= () => { }

Statek

State	Type	Default
email_confirmation	String	''
email	String	''

Függvények

changeEmailHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
changeEmailHandler = (e) => {
    const target = e.target.name
    const value = e.target.value
    this.props.onChange(e, this.props.id)

    this.setState({
        [target]: value
    })
}
```

Forgot Password Form

Kiegészítő form komponens, elfelejtett jelszó igényléséhez szükséges.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import ForgotPassForm from './components/Forms/ForgotPassForm'
```

Properties

resetPasswordRequest	Func	required
error	Bool	= false
errormessage	String	= [Empty String]

```
Invariant Violation: You should not use <Link> outside a <Router>
```

Statek

State	Type	Default
email	String	''

Függvények

valueChangeHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
1  valueChangeHandler = (event) => {
2    const target = event.target.name
3    const value = event.target.value
4
5    this.setState({
6      [target]: value
7    })
8  }
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a porpertyben kapott függvénynek átadja paraméterül.

```
submitHandler = (event) => {
  const { email } = this.state
  const { resetPasswordRequest } = this.props
  event.preventDefault()
  resetPasswordRequest({ email })
}
```

Login Form

Kiegészítő form komponens, bejelentkezéshez szükséges.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import LoginForm from './components/Forms/LoginForm'
```

Properties

login	Func	required
error	Bool	required
errormessage	String	required

```
Invariant Violation: You should not use <Link> outside a <Router>
```

Statek

State	Type	Default
email	String	''
password	String	''
rememberMe	Boolean	false

Függvények

valueChangeHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
valueChangeHandler = (event) => {
  const target = event.target.name
  const value = event.target.value

  this.setState({
    [target]: value
  })
}
```

handleChange

Az checkbox értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja a checkbox értékére.

```
handleChange = name => event => {
  this.setState({ [name]: event.target.checked })
}
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a porpertyben kapott függvénynek átadja paraméterül.

```
submitHandler = (event) => {
  const { login } = this.props
  const { email, password, rememberMe } = this.state

  event.preventDefault()
  const formValues = {
    email,
    password,
    remember_me: rememberMe
  }
  login(formValues)
}
```

Password Form

Kiegészítő form komponens, jelszó beírásához szükséges.

Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import PasswordForm from './components/Forms/PasswordForm'
```

Properties

id	Number	
errors	Object	= {}
onChange	Func	required

Statek

State	Type	Default
password_confirmation	String	''
password	String	''
showPassword	Boolean	false

Függvények

changepasswordHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
changepasswordHandler = (e) => {
    const target = e.target.name
    const value = e.target.value
    this.props.onChange(e, this.props.id)

    this.setState({
      [target]: value
    })
  }
```

handleClickShowPassword

Klikk eseményre a **showPassword** nevű state változó értékét negálja.

```
handleClickShowPassword = () => {
    this.setState(state => ({ showPassword: !state.showPassword }))
}
```

Permissions checkboxok

Kiegészítő form komponens, jogosultságok jelölőmezőinek megjelenítéséhez szükséges.

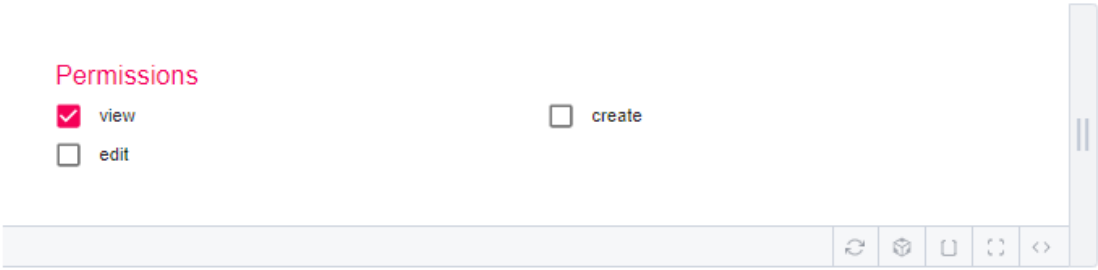
Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
import PermissionsCheckboxs from './components/Forms/PermissionsCheckboxs'

<PermissionsCheckboxs
  permissions={permissions}
  permissionStatus={permissionStatus}
  handleChange={this.handleChange}
/>
```

Properties

permissions	Array	= []
permissionStatus	Object	= {}
handleChange	Func	required



registration Form

Kiegészítő form komponens, regisztrációhoz szükséges.

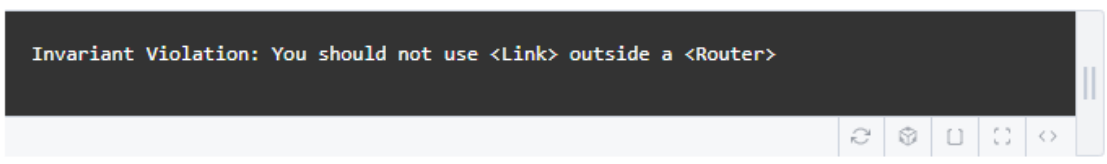
Általános használat

Az űrlap megjeleníti az átadott adatokat a megfelelő mezőkben.

```
1  import RegistrationForm from './components/Forms/RegistrationForm'
2
3  <RegistrationForm registration={registration} error={error} errormessage={errormessage}
   confirmPass={confirmPass} />
```

Properties

registration	Func	= () => {}
confirmPass	Func	= () => {}
error	Bool	= false
userErrorMessage	String	= [Empty String]



Statek

State	Type	Default
errormessage	String	''
email	String	''
password	String	''
confpassword	String	''
roleId	String	'2'

Függvények

valueChangeHandler

Az inputmező értékének változásakor az aktuális mező nevével tárolt értéket megváltoztatja az inputmező értékére.

```
valueChangeHandler = (event) => {
  const target = event.target.name
  const value = event.target.value

  this.setState({
    [target]: value
  })
}
```

submitHandler

A form elküldésekor hívódik meg. Összeállítja az elküldeni kívánt adatokat egy objektumba, majd a porpertyben kapott függvénynek átadja paraméterül.

```
1  submitHandler = (event) => {
2    const { email, password, confpassword, roleId } = this.state
3    const { confirmPass, registration } = this.props
4    const { lang } = this.context
5    event.preventDefault()
6    const formValues = {
7      email,
8      password,
9      password_confirmation: confpassword,
10     role_id: parseInt(roleId)
11   }
12
13   if (password !== confpassword) {
14     confirmPass(true)
15     this.setState({
16       errormessage: lang.common.messages.errors.passMatch
17     })
18   } else {
19     confirmPass(false)
20     this.setState({
21       errormessage: ''
22     })
23     registration(formValues)
24   }
25 }
```

- **HELPERS**

Helpers

```
import { addressConcatHelper, nameConcatHelper, formDataCreator, addStatus } from
'./utils/helpers'
```

addressConcatHelper

Összefűzi a különböző objektumokban lévő címadatokat a táblázathoz.

Bemenet

```
{
    "home_address" : {
        "postal_code": "1111",
        "city": "Település",
        "street": "Utca",
        "street_number": "1"
    },
    "home_address_door_number": "1/1",
}
```

```
export const addressConcatHelper = (data) => {
  const regex = /_address\b/g
  let newAddress = {}
  return data.map(item => {
    if (item.user.email_verified_at !== null) {
      for (const key in item) {
        const doorNum = item[`${key}_door_number`] === null ? '' :
`${item[`${key}_door_number`]}.`
        if (key.match(regex)) {
          Object.assign(newAddress, { [`${key}`]: `${item[key].postal_code} ${item[key].city} <br
/>${item[key].street} ${item[key].street_number}. ${doorNum}` })
        }
      }
      return Object.assign({ ...item, ...newAddress })
    } else {
```

Visszatérési érték

```
{
    "home_address": "1111 Település, Utca 1. 1/1."
}
```

nameConcatHelper

Összefűzi a vezeték és keresztnevet egy név objektumba.

Bemenet

```
{
    "first_name" : "Keresztnév",
    "last_name": "Vezetéknév"
}
```

```
export const nameConcatHelper = (data) => {
  return data.map(item => {
    if (item.user.email_verified_at !== null) {
      const newName = { name: `${item.last_name} ${item.first_name}` }
      return Object.assign({ ...item, ...newName })
    } else {
      const email = { name: `${item.user.email}` }
      return Object.assign({ ...item, ...email })
    }

  })
}
```

Visszatérési érték

```
{
    "name": "Vezetéknév Keresztnév"
}
```

formDataCreator

Előkészíti a Formokhoz az objektumok egyszerűbb elérését.

Bemenet

```
{
    "user" : {
        "id" : "1",
        "email": "example@identity-hungary.hu"
    },
    "home_address" : {
        "postal_code": "1111",
        "city": "Település",
        "street": "Utca",
        "street_number": "1"
    }
}
```

```
export const formDataCreator = (data) => {
  if (data && Array.isArray(data)) {
    return data.map(element => {
      let newDataElement = Object.assign({}, element)
      for (const key in newDataElement) {
        if (newDataElement.hasOwnProperty(key)) {
          const item = newDataElement[key]

          if (typeof item === 'object') {
            for (const itemKey in item) {
              if (item.hasOwnProperty(itemKey)) {
                const newItems = { [`${key}_${itemKey}`]: item[itemKey] }
                Object.assign(newDataElement, newItems)
              }
            }
```

```
          }
        }
      }
      return newDataElement
    })
  }
}
```

Visszatérési érték

```
{
    "user" : { "id" : "1", "email": "example@identity-hungary.hu" },
    "user_id" : "1",
    "user_email": "example@identity-hungary.hu",
    "home_address_postal_code": "1111",
    "home_address_city": "Település",
    "home_address_street": "Utca",
    "home_address_street_number": "1"

}
```

addStatus

A bemeneti adatokhoz hozzáfűzi a status objektumot. A status értéke az user objektum értékétől függ.

Bemenet

```
{
    "user": null
}
```

```
export const addStatus = (data) => {
  let status = {}
  if (data) {
    return data.map(element => {
      let newElement = Object.assign({}, element)
      if (element.user === null || element.deleted_at !== null) {
        status = { status: '<span class="table-status passive">inactive</span>' }
        newElement = { ...newElement, ...status }
      } else if (element.hasOwnProperty('user') && element.user.email_verified_at === null) {
        status = { status: '<span class="table-status warning">pending</span>' }
        newElement = { ...newElement, ...status }
      } else {
        status = { status: '<span class="table-status active">active</span>' }
        newElement = { ...newElement, ...status }
      }
```

```
16        return newElement
17      })
18    }
19  }
```

Visszatérési érték

```
1  {
2      "status": "<span class='table-status passive'>inactive</span>"
3  }
```

permissionsStatus

Visszaadja, hogy az adott szerepkörhöz mely jogosultság tartozik.

Bemenet

```
1  "allowedPermissions": [
2    {
3      "created_at": "2019-04-29 06:16:27",
4      "guard_name": "api",
5      "id": 1,
6      "name": "view",
7      "updated_at": "2019-04-29 06:16:27",
8    },
9    {
10     "created_at": "2019-04-29 06:16:36",
11     "guard_name": "api",
12     "id": 2,
13     "name": "edit",
14     "updated_at": "2019-04-29 06:16:36",
15   }
16 ]
```

```
export const permissionsStatus = (roleData) => {
  let permissionsStatus = {}
  roleData.allowedPermissions.forEach(allowedPermission => {
    const filteredRoles = roleData.rolePermissions.filter(rolePermission => allowedPermission.id
=== rolePermission.id)
    const isRolePermission = filteredRoles.length > 0
    permissionsStatus = { ...permissionsStatus, [allowedPermission.id]: !!isRolePermission }
  })
  return permissionsStatus
}
```

Visszatérési érték

```
{
  "1": true,
  "2": true
}
```

permissionFilter

Visszaadja, hogy az adott szerepkörhöz mely jogosultság tartozik.

Bemenet

```
{
  "1": true,
  "2": true
}
```

```
export const permissionFilter = (data, target) => {
  const keys = Object.keys(data)
  const filtered = keys.filter((key) => {
    if (key === target) {
      return !data[key]
    } else {
      return data[key]
    }
  })
  return filtered
}
```

Visszatérési érték

```
[
  "1",
  "2"
]
```

- **LAYOUTS**

Admin layout

A layout meghatározza az adminfelület elrendezését, tartalmazza a fejlécet, a navigációt és a beburkolt oldal komponenseket valamint a figyelmeztető üzenetek komponensét.

```
<TranslationContext>
  <Alert className='alert' open={open} type={type} message={message} header={header} timeout={3000} />
  <Header {...this.props} >
    <Navigation
      {...this.props}
      lang={lang}
      openMenulist={openMenulist}
      handleClose={this.handleClose}
      handleToggle={this.handleToggle}
    />
  </Header>
  <section className='content' >
    <Content {...this.props} value={value} alertHandler={this.alertHandler} />
  </section>
```

Layout használata

```
import Layout from './layouts/Admin/Layout'

export default Layout(Admin)
```

Admin felület navigációja

Az admin felület navigációjára egy lenyíló úgynevezett dropdown menüt használunk. Ebben a lenyíló menüben találhatóak a menüpontok, amelyek a különböző tabok között navigálnak.

Adminisztárciós menüpontok

- Eladók
- Vásárlók
- Jogosultságok
- Szerepkörök
- Termékkategóriák
- Extra form elemek

Properties

openMenulist	Object	required
handleClose	Func	required
handleToggle	Func	required

Navigation használata

```
import Navigation from './layouts/Admin/Navigation'

<Navigation
  {...this.props}
  lang={lang}
  openMenulist={openMenulist}
  handleClose={this.handleClose}
  handleToggle={this.handleToggle}
/>
```

OpenMenulist

```
this.state = {
  openMenulist: {
    administration: false
  }
}
```

handleToggle

Az aktuális lenyló menüt ki- és becsukja

```
handleToggle = (e) => {
  const target = e.currentTarget.id
  this.setState(state => ({
    openMenulist: {
      ...state.openMenulist,
      [target]: true
    }
  }))
}
```

tabHandleChange

Az aktuális tabra vált

```
tabHandleChange = (event) => {
  const id = event.currentTarget.id
  if (id) {
    const value = parseInt(id.split('#')[1])
    this.setState({ value })
  }
}
```

handleClose

Bezárja a lenyíló menüt

```
handleClose = (event) => {
  this.tabHandleChange(event)
  this.props.history.push(`/admin${event.target.id}`)
  this.setState({
    openMenulist: {
      administration: false
    }
  })
}
```

Customer layout

A layout meghatározza a vásárló felhasználói felületének elrendezését, tartalmazza a fejlécet, a navigációt és a beburkolt oldal komponenseket valamint a figyelmeztető üzenetek komponensét.

```
<TranslationContext {...this.props}>
  <Alert className='alert' open={open} type={type} message={message} header={header} timeout={3000} />
  <Header {...this.props} >
    <Navigation {...this.props} />
  </Header>
  <section className='content' >
    <Content {...this.props} alertHandler={this.alertHandler} />
  </section>
</TranslationContext>
```

Layout használata

```
import Layout from './layouts/Customer/Layout'

export default Layout(Customer)
```

Vásárlói felület navigációja

A vásárlói felület menüszerkezete listaelemekből épül fel. Ezek linkeket tartalmaznak így navigálhatunk az oldalak között.

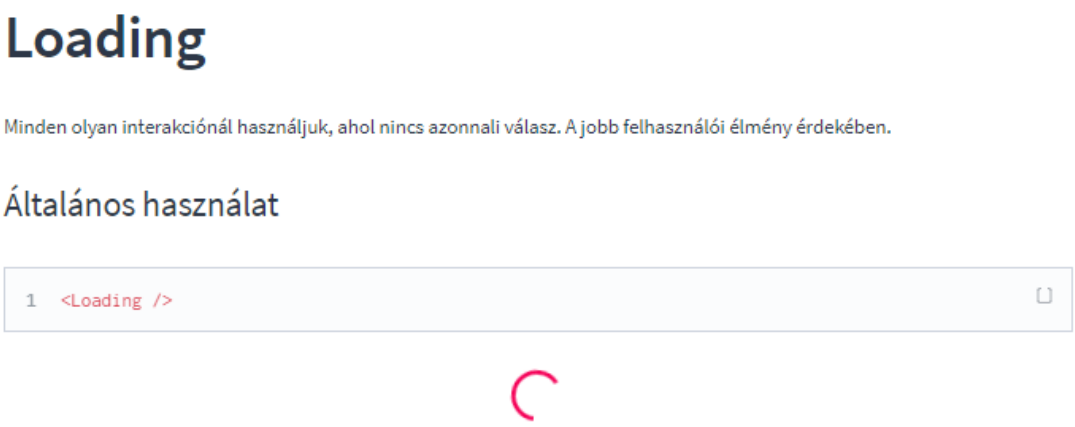
Vásárlói felület menüpontjai

- Főoldal
- Számlák
- Garanciás termékek
- Lejáró termékek

Navigation használata

```
import Navigation from './layouts/Customer/Navigation'

<Navigation {...this.props} />
```

- **LOADING**


Loading
Minden olyan interakciónál használjuk, ahol nincs azonnali válasz. A jobb felhasználói élmény érdekében.
Általános használat
1 <Loading />

- **PACKAGES**

Telepített npm csomagok

React

Egy JavaScript library felhasználói felületek felépítéséhez.

Csomagok:

- react: ^16.6.3,
- react-dom: ^16.6.3,
- react-scripts: ^2.1.8,

Dokumentáció

React router

A React Router navigációs komponensek gyűjteménye, amelyek deklaratívan összeállíthatók az alkalmazással. Könyvjelzőzhető URL-ek létrehozásához.

Csomagok:

- react-router-dom: ^4.3.1

Dokumentáció

Material UI

React UI framework amelybe implementálták a Google Material Design-ját.

Csomagok:

- @material-ui/core: ^3.6.1,
- @material-ui/icons: ^3.0.1,
- @material-ui/lab: ^3.0.0-alpha.30
- material-ui-pickers: ^2.2.1

Dokumentáció

Material UI Pickers

Testreszabható dátumválasztó

Csomagok:

- @date-io/date-fns: ^1.1.0
- date-fns: ^2.0.0-alpha.25

Dokumentáció

Devexpress React Grid

A DevExtreme React Grid egy olyan komponens, amely helyi vagy távoli forrásból származó adatokat táblázatban jelenít meg. Támogatja a lapozás, a válogatás, a szűrés, a csoportosítás és az egyéb adatformálás lehetőségeit, a sorok kiválasztását és az adatok szerkesztését.

Csomagok:

- @devexpress/dx-react-core: ^1.9.1
- @devexpress/dx-react-grid: ^1.9.1
- @devexpress/dx-react-grid-material-ui: ^1.9.1

Dokumentáció

React select

Rugalmas Select input React-hez. Multiselect, autocomplete, async és kreatív támogatással.

Csomagok:

- react-select: ^2.4.0

Dokumentáció

React-svg

React komponens amely az SVG fájlokat beilleszti a DOM-ba.

Csomagok:

- react-svg: ^7.2.3

Dokumentáció

File-saver

Kliensoldali fájlok mentésére szoltál.

Csomagok:

- file-saver: ^2.0.1

Dokumentáció

PrettySize

Segít a fájlméretek értelmezhetőbb kiírásában.

Csomagok:

- prettysize: ^2.0.0

Dokumentáció

React context alerts

Egyszerű és konfigurálható figyelmeztető library.

Csomagok:

- react-context-alerts: ^0.7.2

Dokumentáció

React currency format

Segít az árak értelmezhetőbb kiírásában.

Csomagok:

- react-currency-format: ^1.0.0

Dokumentáció

React dropzone

Egyszerű React hook HTML5-kompatibilis drag'n'drop zóna létrehozásához.

Csomagok:

- react-dropzone: ^8.1.0

Dokumentáció

Docz

Egyszerű dokumentáció megjelenítő React frameworkhöz

Csomagok:

- docz: ^0.13.7
- docz-plugin-css: ^0.11.0
- docz-theme-default: ^0.13.7

Dokumentáció

ESLint

JavaScript és JSX linter.

Csomagok:

- eslint: ^5.13.0

Dokumentáció

Node-sass

Stíluslap preprocesszor node.js-hez.

Csomagok:

- node-sass: ^4.10.0

Dokumentáció

- **PAGES**

Activate

```
1  <Route path="/signup-activate/:token" render={props => <Activate {...props} />} />
```

Regisztráció aktiválásához szükséges oldal. Ha az url-ben lévő token létezik és az aktiváció sikeres, akkor egy megerősítő üzenetet kapunk a regisztráció aktiválásáról. Különben hibaüzenet jelenik meg.

Admin

```
1  <PrivateRoute path="/admin" exact component={AdminAuth(Admin)} />
```

Csak **admin** szerepkörrel rendelkező felhasználó láthatja ezt az oldalt. Ez az adminisztrátor adminisztrációs felülete. Amelyen látjhatja és kezelheti az összes **eladó, vásárló, jogosultság, szerepkör, kategória, extra form elem** adatát.

Az admin felhasználó csak ezt a kezelőfelületet láthatja.

Customer

```
1  <PrivateRoute path="/customer" exact component={CustomerAuth(Customer)} />
```

Csak **customer** szerepkörrel rendelkező felhasználó láthatja ezt az oldalt. Amely a customer kezelőfelületén egy áttekintő oldal (kezdőoldal). Tartalmazza a felhasználó számára legfontosabb adatokat. Számláinak, termékeinek és lejáró termékeinek darabszámát.

Erről a felületről egyből tud számlát létrehozni. Áttekintheti számláit, az összes rögzített termékét és kilistázhatja a hamarosan lejáró garanciás termékeit.

ExpiredProducts

```
1  <PrivateRoute path="/customer/expired-products" component={CustomerAuth(ExpiredProducts)} />
```

Csak **customer** szerepkörrel rendelkező felhasználó láthatja ezt az oldalt.

Ezen az oldalon a felhasználó a hamarosan lejáró garanciás termékei láthatók.

Elfelejtett jelszó oldal

```
1  <Route path="/forgot-password" render={props => <ForgotPass {...props} />} />
```

Ezen az oldalon a felhasználó új jelszót igényelhet. Regisztrált emailcíme beírásával emailben kap egy egyedi linket ahol beírhatja új jelszavát.

InvoiceDetail

```
1  <PrivateRoute path="/customer/invoices/:invoice_id" component={CustomerAuth(InvoiceDetail)} />
```

Csak **customer** szerepkörrel rendelkező felhasználó láthatja ezt az oldalt.

Ezen az oldalon az url-ben található számla id alapján a számla részletei, a hozzá tartozó feltöltött dokumentumok és a számlához rendelt termékek láthatóak.

Ha nem létező id-val próbálkozunk akkor a 404-es hibaoldal jön be.

Invoices

```
1  <PrivateRoute path="/customer/invoices" component={CustomerAuth(Invoices)} />
```

Csak **customer** szerepkörrel rendelkező felhasználó láthatja ezt az oldalt.

Ezen az oldalon a felhasználó számlái láthatók.

ItemDetail

```
1  <PrivateRoute path="/customer/invoices/:invoice_id/warranty-products/:item_id" component=
   {CustomerAuth(ItemDetail)} />
```

Csak **customer** szerepkörrel rendelkező felhasználó láthatja ezt az oldalt.

Ezen az oldalon az url-ben található termék id alapján a garanciás termék részletei és a hozzá tartozó feltöltött dokumentumok láthatóak.

Ha nem létező id-val próbálkozunk akkor a 404-es hibaoldal jön be.

Bejelentkezés oldal

```
1  <Route exact path="/login" render={props => <Login {...props} />} />
```

Ezen az oldalon a felhasználó bejelentkezhet a szerepkörének megfelelő adminisztrációs felületre.

Page404

```
1  <Route component={Page404} />
```

404-es hiba oldal. Ha az alkalmazásban rossz útvonalra navigálunk amely nem létezik, akkor ez a hibaoldal fog megjelenni. Az oldal tartalmaz egy hibaüzenetet és egy visszagombot a főoldalra.

Products

```
1  <PrivateRoute path="/customer/warranty-products" component={CustomerAuth(Products)} />
```

Csak **customer** szerepkörrel rendelkező felhasználó láthatja ezt az oldalt.

Ezen az oldalon a felhasználó termékei láthatók.

Profile

```
1  <PrivateRoute path="/customer/invoices/:invoice_id/warranty-products/:item_id" component=
   {CustomerAuth(ItemDetail)} />
```

Csak **customer** szerepkörrel rendelkező felhasználó láthatja ezt az oldalt.

Ezen az oldalon a felhasználó beállításai és adatai láthatóak.

Ha nem létező id-val próbálkozunk akkor a 404-es hibaoldal jön be.

Regisztráció oldal

```
1  <Route path="/registration" render={props => <Registration {...props} />} />
```

Ezen az oldalon a látogató regisztrálhat az adminisztrációs felületre mint vásárló.

Regisztráció oldal

```
1  <Route path="/reset-password/:token" render={props => <ResetPass {...props} />} />
```

Ezen a felületen a felhasználó be tudja írni új jelszvát új jelszó igénylése után.

- **ROOTS**

ReactRoot Komponens

A GarNet frontend alapja. Meghatározza az elérési útvonalakat és a hozzá tartozó komponenseket.

Főoldal

A főoldal mindig arra az oldalra irányítja a felhasználót amilyen jogosultsága van. Ha nincs jogosultsága akkor a **/login** oldalra ha viszont van akkor a **/admin** vagy a **/customer** oldalak valamelyikére.

Elérési útvonal	Komponens	Leírás
`"/"`	`<Redirect to='/${path}' />`	Főoldal

Autentikáció nélkül elérhető útvonalak

Elérési útvonal	Komponens	Leírás
`"/login"`	`<Login {...props} />`	Bejelentkezés oldal
`"/forgot-password"`	`<ForgotPass {...props} />`	Új jelszó igénylése oldal
`"/registration"`	`<Registration {...props} />`	Regisztrációs oldal

Egyedi útvonalak

Autentikáció nélkül elérhető útvonalak, viszont egyedi paramétterrel rendelkeznek `:token` ami egy generált karakterlánc. Ezt a backend generálja.

Elérési útvonal	Komponens	Leírás
`"/reset-password/:token"`	`<ResetPass {...props} />`	Régi jelszó törlése, új jelszó megadása
`"/signup-activate/:token"`	`<Activate {...props} />`	Regisztráció aktiválása oldal

Privát elérési útvonal

Sikeres bejelentkezés után a felhasználót jogosultságának megfelelelő kezelőfelületre irányítja át.

Admin felület elérési útvonalai

Elérési útvonal	Komponens	Leírás
`"/admin"`	`AdminAuth(Admin)`	Adminisztrátor főoldala

Customer felület elérési útvonalai

Elérési útvonal	Komponens	Leírás
`"/customer"`	`CustomerAuth(Customer)`	Vásárló főoldala
`"/customer/profile"`	`CustomerAuth(Profile)`	Vásárló profil oldala
`"/customer/invoices/:invoice_id/warranty-products/:item_id"`	`CustomerAuth(ItemDetail)`	Termék részletes oldala
`"/customer/invoices/:invoice_id"`	`CustomerAuth(InvoiceDetail)`	Számla részletes oldala
`"/customer/invoices"`	`CustomerAuth(Invoices)`	Számlák lista oldala
`"/customer/warranty-products"`	`CustomerAuth(Products)`	Termékek lista oldala
`"/customer/expired-products"`	`CustomerAuth(ExpiredProducts)`	Lejáró termékek lista oldala

- **TABLES**

BasicTable

Általános használat

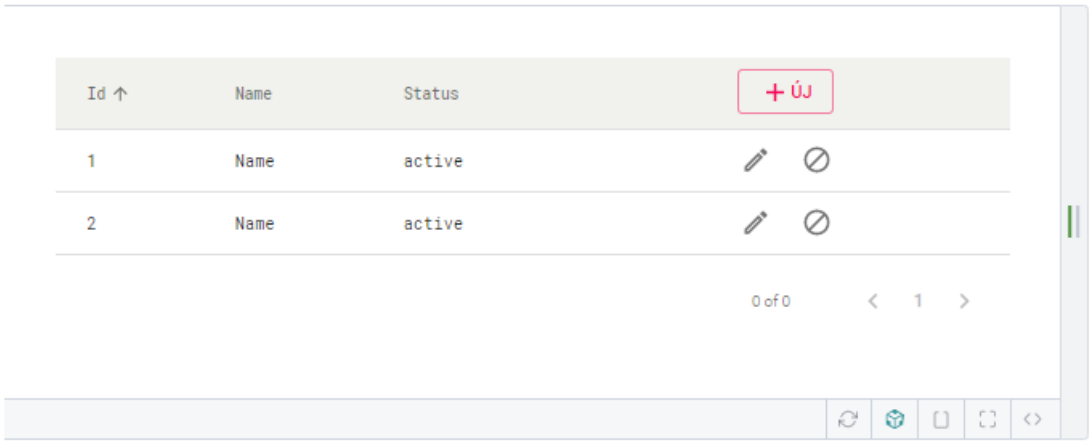
```
import BasicTable from './components/Tables/BasicTable'

<BasicTable
  rows={customersData}
  columns={tableHeaders}
  defaultSorting={defaultSorting}
  currentPage={customers.current_page}
  pageSize={customers.per_page}
  totalCount={customers.total}
  showSortingControls={true}
  canEdit={editable}
  errors={errors}
  updateData={this.updateCustomerPromise}
  createData={this.createCustomerPromise}
  toggleStatus={this.toggleStatusCustomerPromise}
  data={customers.data}
  tableType="Customer"
  cancelData={this.cancelData}
/>
```

Táblázat

Id ↑	Name	Status
1	Name	active
2	Name	active

0 of 0 1

Szerkeszthető táblázat



rows	Array	= []
columns	Array	required
defaultSorting	Array	= [{ columnName: "id", direction: "asc" }]
currentPage	Number	= 1
pageSize	Number	= 0
totalCount	Number	= 0
showSortingControls	Bool	required
canEdit	Bool	= false
errors	Object	= {}
updateData	Func	= () => { }
createData	Func	= () => { }
toggleStatus	Func	= () => { }

data	Array	= []
tableType	String	= [Empty String]
cancelData	Func	= () => { }
changeCurrentPage	Func	= () => { }

Táblázathoz tartozó további komponensek

TableComponent

```
import TableComponent from './components/Tables/BasicTable/TableComponent'


<Grid
  rows={rows}
  columns={columns}
  getRowId={getRowId}
>
  <SortingState defaultSorting={defaultSorting} />
  <EditingState
    onCommitChanges={commitChanges}
    columnExtensions={[{ columnName: 'id', editingEnabled: false }]}
  />
  <IntegratedSorting />
  <PagingState
    currentPage={currentPage - 1}
    onCurrentPageChange={changePage}
    pageSize={pageSize}
  />
  <CustomPaging totalCount={totalCount} />
```

```
  <Table cellComponent={Cell} />
  <TableHeaderRow showSortingControls={showSortingControls ? showSortingControls : false} />
  {canEdit ? (<TableEditColumn
    showAddCommand
    showEditCommand
    showDeleteCommand
    commandComponent={Command}
    cellComponent={EditRowCell}
  />) : (<></>)}
  {canEdit && <Getter
    name="tableColumns"
    computed={({ tableColumns }) => [
      ...tableColumns.filter(c => c.type !== TableEditColumn.COLUMN_TYPE),
      { key: 'editCommand', type: TableEditColumn.COLUMN_TYPE, width: 140 }
    ]}
  />}
  <RowDialogPlugin
    tableType={tableType}
    data={data}
    deletedRowId={deletedRowId}
    cancelData={cancelData}
```

```
42      updateData={updateData}
43      createData={createData}
44      toggleStatus={toggleStatus}
45      cancelDeletedRow={cancelDeletedRow}
46      errors={errors} />
47    {pageSize !== totalCount ? <PagingPanel /> : <></>}
48  </Grid>
```

updateData	Func	= () => { }
currentPage	Number	required
pageSize	Number	required
totalCount	Number	required
changePage	Func	required
tableType	String	= [Empty String]
data	Array	required
errors	Object	= {}
cancelData	Func	= () => { }
createData	Func	= () => { }
toggleStatus	Func	= () => { }
deletedRowId	Array	= []
cancelDeletedRow	Func	= () => { }

rows	Array	required
columns	Array	required
defaultSorting	Array<{ "columnName": "String", "direction": "String" }>	= [{ columnName: "id", direction: "asc" }]
showSortingControls	Bool	= false
commitChanges	Func	= () => { }
canEdit	Bool	= false

RowDialogPlugin

```
import RowDialogPlugin from '.components/Tables/BasicTable/RowDialogPlugin'

<TableComponent>
  ...
   <RowDialogPlugin
      tableType={tableType}
      data={data}
      deletedRowId={deletedRowId}
      cancelData={cancelData}
      updateData={updateData}
      createData={createData}
      toggleStatus={toggleStatus}
      cancelDeletedRow={cancelDeletedRow}
      errors={errors} />
</TableComponent>
```

Sor hozzáadása

```
import AddPopup from '.components/Tables/BasicTable/RowDialogPlugin/AddPopup'

<Plugin>
  <Template name="editPopup">
    <TemplateConnector>
      {({ rows, getRowId ... }, { ... }) => {
        if (openAdd) {
          return (
            <AddPopup
              open={openAdd}
              data={newData}
              errors={errors}
              tableType={tableType}
              onChange={this.addNewInputChangeHandler}
              onApplyChanges={applyNew}
              onCancelChanges={cancelChanges}
            />
          )
         }
      }}
    </TemplateConnector>
    </Template>
    <Template name="root">
      <TemplatePlaceholder />
    <TemplatePlaceholder name="editPopup" />
  </Template>
</Plugin>
```

onApplyChanges	Func	= () => { }
onCancelChanges	Func	= () => { }
onChange	Func	= () => { }
open	Bool	= false
tableType	String	= [Empty String]
data	Object	= {}
errors	Object	= {}

Sor szerkesztése

```
import EditPopup from '.components/Tables/BasicTable/RowDialogPlugin/EditPopup'

<Plugin>
  <Template name="editPopup">
    <TemplateConnector>
      {({ rows, getRowId ... }, { ... }) => {
        if (openEdit) {
          return (
            <EditPopup
              open={openEdit}
              row={changedRow}
              data={targetRowData}
              errors={errors}
              tableType={tableType}
              onChange={this.inputChangeHandler}
              onApplyChanges={applyUpdateChanges}
              onCancelChanges={cancelChanges}
            />
          )
         }
      }}
    </TemplateConnector>
    </Template>
    <Template name="root">
      <TemplatePlaceholder />
    <TemplatePlaceholder name="editPopup" />
  </Template>
</Plugin>
```

row	Object	= {}
onApplyChanges	Func	= () => { }
onCancelChanges	Func	= () => { }
onChange	Func	= () => { }
open	Bool	= false
tableType	String	= [Empty String]
data	Object	= {}
errors	Object	= {}

Státusz állítás

```
import StatusPopup from '.components/Tables/BasicTable/RowDialogPlugin/StatusPopup'

<Plugin>
  <Template name="editPopup">
    <TemplateConnector>
      {({ rows, getRowId ... }, { ... }) => {
          return (
            <StatusPopup
              open={openDelete}
              onApplyChanges={toggleStatus}
              onCancelChanges={cancelChanges}
            />
        )
      }}
    </TemplateConnector>
    </Template>
    <Template name="root">
      <TemplatePlaceholder />
    <TemplatePlaceholder name="editPopup" />
  </Template>
</Plugin>
```

onApplyChanges	Func	= () => { }
onCancelChanges	Func	= () => { }
open	Bool	= false

TreeDataTable

Általános használat

```
import TreeDataTable from './components/Tables/TreeDataTable';
```

Táblázathoz tartozó további komponensek

TreeTableComponent

Az átadott adatokat megjeleníti táblázatban.

```
import TreeTableComponent from './components/Tables/TreeDataTable/TreeTableComponent';

<Grid rows={rows} columns={columns}>
  <EditingState onCommitChanges={commitChanges} columnExtensions={[{ columnName: 'id', editingEnabled: false }]} />
  <TreeDataState />
  <CustomTreeData getChildRows={getChildRows} />
  <Table columnExtensions={tableColumnExtensions} noDataCellComponent={customLoad ? customLoad : Load} />
  <TableEditColumn
    showAddCommand
    showEditCommand
    showDeleteCommand
    commandComponent={customCommand ? customCommand : Command}
    cellComponent={customEditRowCell ? customEditRowCell : EditRowCell}
  />
  <Getter
    name="tableColumns"
    computed={({ tableColumns }) => [
      ...tableColumns.filter(c => c.type !== TableEditColumn.COLUMN_TYPE),
      { key: 'editCommand', type: TableEditColumn.COLUMN_TYPE, width: 150 }
```

```
    ]}
  />
  <TableHeaderRow />
  <TableTreeColumn for="name" />
</Grid>;
```

Properties

rows	Array	required
columns	Array	required
tableColumnExtensions	Array	required
commitChanges	Func	required
customCommand	Func	= null
customEditRowCell	Func	= null
customLoad	Func	= null