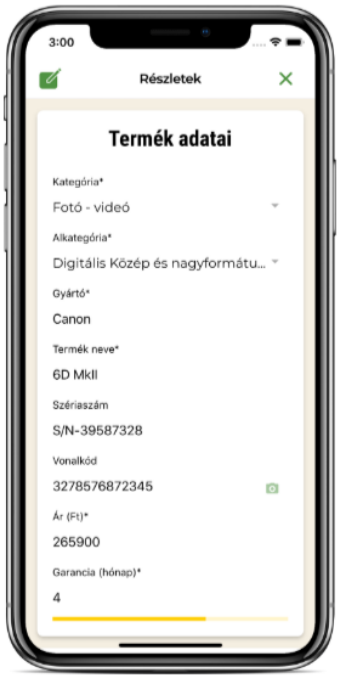
Identity Hungary Informatikai Korlátolt Felelősségű Társaság

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

Product Modal

> Termék modal komponens



Elemek

- **Fejléc**
  - Szerkesztés / Elküldés gomb
  - Bezárás gomb
- **Adatok kártya**
- **Dokumentumok kártya**
  - Új dokumentum gomb
  - Feltöltött dokumentumok listája

Metódusok

ngOnInit

> Inicializálja a megjelenítésre és szerkesztésre használt formot,
> feliratkozik a nézet frissítéséhez szükséges eseményekre,
> kigenerálja a garancia vizuális megjelenítésére szolgáló sáv adatait

> **isLeftSide:** megadja a megnyomott fejléc gomb pozícióját

```typescript
ngOnInit(): void {
  translate.stream('product.header').subscribe(res => {
    headerTitles = res;
  });

  const product = product;
  productEditForm = formBuilder.group({
    barcode: product.barcode,
    files: [],
    has_extra_form_item: product.customer_invoice_extra_item,
    invoice_id: product.invoice_id,
    manufacture: product.manufacture,
    name: product.name,
    price: product.price,
    product_category_id: product.product_category_id,
    product_subcategory_id: null,
    type_no: product.type_no,
    warranty_in_month: product.warranty_in_month,
    warrany: product.warranty
  });

  initProgressBar();

  buttonHandler.subscribe(side => {
    buttonClicked(side);
  });
```

```
    events.subscribe('refreshProduct', () => {
      productsService
        .getProduct(user.user_id, product.invoice_id, product.id)
        .then(res => {
          product = res;
          cdRef.detectChanges();
        });
    });

    events.subscribe('addAttachment', () => {
      editMode = true;
      setTimeout(() => content.scrollToBottom(100), 100);
    });

    productEditForm.get('product_category_id').valueChanges.subscribe(val => {
      getSubCategories(val);
    });

    productEditForm.get('product_subcategory_id').valueChanges.subscribe(val => {
      subCategoryChanged(val);
    });
    getSubCategories(product.product_category_id);
  }
```

initProgressBar

> Kiszámolja a termék adatai alapján a sáv érétkét és színét

typescript

```
initProgressBar(): void {
  const value = calculateWarrantyBar(
    product.invoice.purchased,
    productEditForm.value.warranty_in_month
  );
  product.progressBarValue = value;
  if (value < 0.5) {
    barColor = 'primary';
  }
  if (value > 0.5 && value < 0.75) {
    barColor = 'warning';
  }
  if (value > 0.75) {
    barColor = 'danger';
  }
}
```

calculateWarrantyBar

> Kiszámolja százalékos értékeben a vásárlás óta eltelt időből
> és a garancia hosszából a hátralevő garancia idejét

> **date:** a termék vásárlási dátuma (a számla alapján) **month:** a garancia hossza hónapban

typescript

```
calculateWarrantyBar(date, month): Number {
  const today: any = new Date();
  const start: any = new Date(date);
  const starTime = new Date(date);
  const endTime = starTime.setMonth(starTime.getMonth() + parseInt(month, 10));
  const end: any = new Date(endTime);
  const value =
    Math.round(((today - start) / (end - start)) * 100) < 0
      ? 0
      : Math.round(((today - start) / (end - start)) * 100);
  return value / 100;
}
```

subCategoryChanged

> Az alkategória változásaira meghívott függvény,
> lekéri a kategóriához tartozó extra mezőket, majd meghívja
> az ezen mezők hozzáadásáért szolgáló függvényeket

> **val**: az alkategória mező értéke

typescript

```
subCategoryChanged(val): void {
  productsService.getExtraFormItems(val).then(res => {
    if (res && res.length > 0) {
      productEditForm.get('has_extra_form_item').setValue(1);
      addExtraField(res);
      loadExtraFields(res);
    } else {
      productEditForm.get('has_extra_form_item').setValue(0);
      clearExtraFields();
    }
  });
}
```

getSubCategories

> A kategória változására lekéri az adott kategóriához tartozó alkategóriákat a backendről

> **val**: a kategória mező értéke

```typescript
getSubCategories(val): void {
  productsService.getProductSubCategories(val).then(res => {
    subCategories = res;
    if (product.product_subcategory_id) {
      productEditForm.patchValue({
        product_subcategory_id: product.product_subcategory_id
      });
      subCategoryChanged(product.product_subcategory_id);
    }
  });
}
```

loadExtraFields

> Az adatok szerkesztésre történő betöltéséhez használt függvény

> **extraFields**: a backend által küldött extra mezők

```typescript
loadExtraFields(extraFields): void {
  const extraItems = product.customer_invoice_extra_item || [];
  extraFields.forEach(item => {
    const value = extraItems.find(el => el.form_group_id === item.id).form_item_value;
    const field = item.form_item[0].name;
    productEditForm.patchValue({
      [field]: value
    });
  });
}
```

addExtraField

> Hozzáadja az extra elemeket a formhoz, és a nézethez

> **items:** a backend által küldött extra form elemek

typescript

```
addExtraField(items): void {
  clearExtraFields();

  items.forEach(item => {
    productEditForm.addControl(item.form_item[0].name, new FormControl(null));
    extraFormItems.push({
      controlName: item.form_item[0].name,
      id: item.form_item[0].form_group_id,
      label: item.name,
      type: item.form_item[0].type
    });
  });

  extraItemsForTemplate = extraFormItems;
}
```

clearExtraFields

> Eltávolítja a formból és a nézetből az összes extra elemet

typescript

```
clearExtraFields(): void {
  extraFormItems.forEach(item => {
    productEditForm.removeControl(item.controlName);
  });
  extraFormItems = [];
}
```

editHandler

> Lekezeli a szerkesztés műveletetét, eldönti, hogy milyen változtatásokat kell elküldeni a backend számára.

```typescript
editHandler(): void {
  const form = productEditForm;
  const formTouched = form.touched;

  if (!formTouched && fileUploader.files.length === 0) {
    editMode = false;
    return;
  }

  if (formTouched) {
    modifyForm();
  }
  if (fileUploader.files.length > 0) {
    modifyAttachments();
  }
}
```

modifyForm

> A termék alap adatainak módosítását kezeli.

> ✅ **Sikeres hívás:** frissítjük a nézetet.
> ❌ **Sikertelen hívás:** megjelenítjük a backendről érkező hibaüzenetet a hibás mezőknél.

```typescript
modifyForm(): void {
  const form = productEditForm;

  form.patchValue({
    invoice_id: parseInt(form.get('invoice_id').value, 10),
    price: parseInt(form.get('price').value, 10),
    warranty_in_month: parseInt(form.get('warranty_in_month').value, 10)
  });

  form.value.warranty = fileUploader.files.length > 0 ||
    product.uploads.length > 0 ? 1 : 0;

  productsService
    .modifyProduct(
      form.value,
      user.user_id,
      product.invoice_id,
      product.id
    )
    .then(() => {
      editMode = false;
      initProgressBar();
      events.publish('refreshProducts');
      events.publish('refreshExpiring');
      events.publish('refreshProduct');
    })
    .catch(err => (formErrors = err.error.errors || {}));
}
```

scanBarcode

> Megnyitja a vonalkód olvasót

> ✔️ **Sikeres olvasás:** beállítjuk az olvasott értéket a vonalkód mezőbe.
> ❌ **Sikertelen olvasás:** nem történik semmi.

```typescript
scanBarcode() {
  barcodeScanner
    .scan()
    .then(barcodeData => {
      productEditForm.patchValue({
        barcode: barcodeData.text
      });
    })
    .catch();
}
```

modifyAttachments

> A termékhez tartozó csatolmányok feltöltéséért felel.

> ✅ **Sikeres hívás:** frissítjük a nézetet az új dokumentumokkal.
> ❌ **Sikertelen hívás:** megjelenítjük a backendről érkező hibaüzenetet.

typescript

```
modifyAttachments(): void {
    const formData = new FormData();
    formData.append('invoice_id', product.invoice_id.toString());
    formData.append('invoice_item_id', product.id.toString());
    formData.append('user_id', user.user_id.toString());
    fileUploader.formFiles.forEach((file: any) => {
      formData.append('files[]', file, file.name);
    });

    productsService.addAttachment(formData, user.user_id).then(() => {
      productsService
        .getProduct(user.user_id, product.invoice_id, product.id)
        .then(res => {
          product = res;
          events.publish('refreshProducts');
          events.publish('refreshProduct');
          editMode = false;
        });
    });
  }
```

buttonClicked

> A fejlécben található gombok megnyomásának kontextus szerinti lekezelése.

> **isLeftSide**: megadja a megnyomott fejléc gomb pozícióját

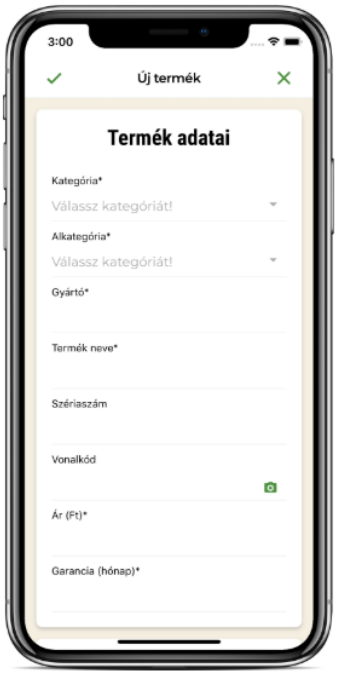
```typescript
buttonClicked(isLeftSide): void {
  if (!isLeftSide) {
    modalCtrl.dismiss();
    return;
  }
  if (isLeftSide && editMode === false) {
    editMode = true;
    return;
  }
  if (isLeftSide && editMode === true) {
    editHandler();
  }
}
```

New Product

> Új termék modal komponens

Elemek

- **Fejléc**
  - Elküldés gomb
  - Bezárás gomb
- **Adatok kártya**
- **Dokumentumok feltöltése kártya**
  - Új dokumentum gomb
  - Feltöltésre kiválasztott dokumentumok listája



Metódusok

ngOnInit

> Inicializálja az új termék modal ablakban található formot, és feliratkozik a nézet frissítéséhez szükséges eseményekre, valamint a fejlécben találhatógombok lenyomására

typescript

```typescript
ngOnInit(): void {
  newProductForm = formBuilder.group({
    barcode: [null],
    files: [],
    has_extra_form_item: [0],
    invoice_id: invoiceId,
    manufacture: [null],
    name: [null],
    price: [null],
    product_category_id: [null],
    product_subcategory_id: [null],
    type_no: [null],
    warranty: [false],
    warranty_in_month: [null]
  });

  buttonHandler.subscribe(side => {
    buttonClicked(side);
  });
```

```
    newProductForm.get('product_category_id').valueChanges.subscribe(val => {
      productsService.getProductSubCategories(val).then(res => {
        subCategories = res;
      });
    });

    newProductForm.get('product_subcategory_id').valueChanges.subscribe(val => {
      productsService.getExtraFormItems(val).then(res => {
        if (res && res.length > 0) {
          newProductForm.get('has_extra_form_item').setValue(1);
          addExtraField(res);
        } else {
          newProductForm.get('has_extra_form_item').setValue(0);
          clearExtraFields();
        }
      });
    });
  }
```

addExtraField

> Amennyiben a kiválaszottt alkategóriának vannak extra form elemei, hozzáadja azokat a nézethez és a form vezérléshez

> **items:** a kiválasztott alkategóriához tartozó extra form elemek

typescript

```
addExtraField(items): void {
  clearExtraFields();

  items.forEach(item => {
    newProductForm.addControl(item.form_item[0].name, new FormControl(null));
    extraFormItems.push({
      controlName: item.form_item[0].name,
      label: item.name,
      type: item.form_item[0].type
    });
  });
}
```

clearExtraFields

> Kiüríti a nézetből és a form vezérlésből az extra elemeket

typescript

```
clearExtraFields(): void {
  extraFormItems.forEach(item => {
    newProductForm.removeControl(item.controlName);
  });
  extraFormItems = [];
}
```

scanBarcode

> Megnyitja a vonalkód olvasót

> ✔️ **Sikeres olvasás**: beállítjuk az olvasott értéket a vonalkód mezőbe.
> ❌ **Sikertelen olvasás**: nem történik semmi.

```typescript
scanBarcode(): void {
  barcodeScanner
    .scan()
    .then(barcodeData => {
      newProductForm.patchValue({
        barcode: barcodeData.text
      });
    })
    .catch();
}
```

saveProduct

> Összegyűjti a formba bevitt adatokat, a backend által megkövetelt formátumra alakítja, majd elküldi a formot.

> ✔️ **Sikeres hívás**: visszanavigáljuk a usert a listanézetre.
> ❌ **Sikertelen hívás**: megjelenítjük a hibás mezők alatt a backendről érkező hibaüzenetet.

```typescript
saveProduct(): void {
  newProductForm.patchValue({
    files: fileUploader.formFiles
  });

  const form = newProductForm.value;
  form.warranty = fileUploader.formFiles.length > 0 ? 1 : 0;

  newProductForm.patchValue({
    invoice_id: parseInt(newProductForm.get('invoice_id').value, 10),
    price: parseInt(newProductForm.get('price').value, 10),
    warranty_in_month: parseInt(newProductForm.get('warranty_in_month').value, 10)
  });

  const formData = new FormData();
  Object.keys(form).forEach(el => {
    if (el === 'files') {
      Array.from(form[el]).forEach((file: any) => {
        formData.append('files[]', file, file.name);
      });
    } else {
      formData.append(el, form[el]);
    }
  });

  productsService
    .newProduct(formData, user.user_id, invoiceId)
    .then(() => {
      modalCtrl.dismiss();
      events.publish('refreshInvoice');
      events.publish('refreshInvoices');
      events.publish('refreshProducts');
      events.publish('updateHeight');
    })
    .catch(err => {
      formErrors = err.error.errors || {};
    });
}
```

buttonClicked

> A fejlécben található gombok megnyomásának kontextus szerinti lekezelése.

> **isLeftSide**: megadja a megnyomott fejléc gomb pozícióját

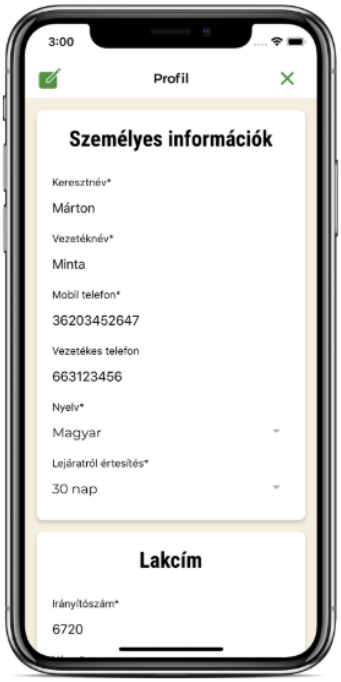
```typescript
buttonClicked(isLeftSide): void {
  if (isLeftSide) {
    saveProduct();
  } else {
    modalCtrl.dismiss();
  }
}
```

Profile

> Profil komponens

Elemek

- **Fejléc**
  - Szerkesztés / Elküldés gomb
  - Modal bezárása gomb
- Személyes információk kártya
- Lakcím kártya
- Postai cím kártya



Metódusok

ngOnInit

> Inicializálja a profil modal ablakot, betölti a meglévő adatokat, lekéri a fejlécben található szöveg adott nyelvhez tartozó értékeit.

```typescript
ngOnInit(): void {
  profileForm = formBuilder.group({
    default_language: profile.user.default_language,
    expire_alert_in_day: profile.expire_alert_in_day.toString(),
    first_name: profile.first_name,
    home_address_city: profile.home_address.city,
    home_address_door_number: profile.home_address_door_number,
    home_address_postal_code: profile.home_address.postal_code,
    home_address_street: profile.home_address.street,
    home_address_street_number: profile.home_address.street_number,
    last_name: profile.last_name,
    mobile_phone: profile.mobile_phone,
    postal_address_city: profile.postal_address.city,
    postal_address_door_number: profile.postal_address_door_number,
    postal_address_postal_code: profile.postal_address.postal_code,
    postal_address_street: profile.postal_address.street,
    postal_address_street_number: profile.postal_address.street_number,
    user_id: user.user_id,
    wired_phone: profile.wired_phone
  });

  translate.stream('profile.header').subscribe(res => (headerTitles = res));
  buttonHandler.subscribe(side => {
    buttonClicked(side);
  });
}
```

sendForm

> Összegyűjti a form értékeit, és elküldi a backendnek.

> ✅ **Sikeres hívás:** visszanavigáljuk a usert a megtekintési nézetre.
> ❌ **Sikertelen hívás:** megjelenítjük a hibás mezők alatt a backendről érkező hibaüzenetet.

```typescript
sendForm(): void {
  if (!profileForm.touched) {
    editMode = false;
    return;
  }

  formErrors = {};
  userService
    .setProfile(user.user_id, profileForm.value)
    .then(() => {
      storage.set('lang', profileForm.value.default_language);
      translate.use(profileForm.value.default_language);
      editMode = false;
    })
    .catch(err => (formErrors = err.error.errors));
}
```

buttonClicked

> A fejlécben található gombok megnyomásának kontextus szerinti lekezelése.

> **isLeftSide:** megadja a megnyomott fejléc gomb pozícióját

```typescript
buttonClicked(isLeftSide): void {
  if (!isLeftSide) {
    modalCtrl.dismiss();
    return;
  }
  if (isLeftSide && editMode === false) {
    editMode = true;
    return;
  }
  if (isLeftSide && editMode === true) {
    sendForm();
  }
}
```

openPrompt

> Megnyitja a kijelentkezés megerősítésére szolgáló felugró ablakot.

```typescript
openPrompt(): void {
  modalService.openModal('logout').then(success => {
    if (success) {
      logoutHandler();
    }
  });
}
```

logoutHandler

> Kijelentkezteti a usert.

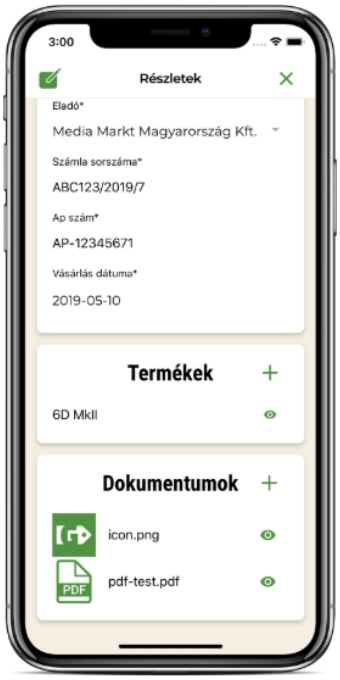
```typescript
logoutHandler(): void {
    modalCtrl.dismiss();
    userService.logout();
  }
```

Uploads

> Feltöltések komponens

Elemek

- Feltöltések kártya
  - **Feltöltött dokumentumok listája**
    - Fájl logó / előnézeti kép
    - Fálj neve
    - Megtekintés / Törlés gomb



Metódusok

openPreview

> Ha a csatolmány már le van töltve, megnyitja megtekintésre, ha nincs, letölti.

> **idx:** a csatolmány indexe
> **uploadId:** a csatolmány id-ja

typescript

```
openPreview(idx, uploadId) {
  const file = uploads[idx];
  if (!file.thumbnail) {
    downloadFile(idx, uploadId);
  } else {
    if (file.extension === 'pdf') {
      saveAndOpenPdf(file);
    } else {
      imageModal.showImageModal(
        sanitizer.bypassSecurityTrustResourceUrl(file.thumbnail),
        file.filename
      );
    }
  }
}
```

triggerAttachment

> Jelzi az eseményre feliratkozott komponenseknek,
> hogy új csatolmányt szeretnénk hozzáadni, amire az adott komponens szerkesztő módra vált.

```typescript
triggerAttachment(): void {
    events.publish('addAttachment');
}
```

saveAndOpenPdf

> Lementi a pdf fájlt a készülék átmeneti tárolójába, majd megnyitja azt megtekintésre.

> **file:** a pdf fájl

```typescript
saveAndOpenPdf(file) {
  const writeDirectory = platform.is('ios')
    ? file.dataDirectory
    : file.externalDataDirectory;

  file
    .writeFile(writeDirectory, file.filename, file.pdf, { replace: true })
    .then(() => {
      opener
      .open(writeDirectory + file.filename, 'application/pdf')
      .catch();
    })
    .catch();
}
```

downloadFile

> Letölti a csatolmányt, majd beállítja azok előnézeti képét, vagy pdf esetében egy pdf logót

> **idx**: a csatolmány indexe
> **uploadId**: a csatolmány id-ja

typescript

```
downloadFile(idx, uploadId) {
  const file = uploads[idx];
  filesService.downloadFile(user.user_id, uploadId)
  .then(res => {
    if (file.extension !== 'pdf') {
      const reader = new FileReader();
      reader.readAsDataURL(res);
      reader.onloadend = () => {
        file.thumbnail = reader.result;
      };
    } else {
      file.thumbnail = 'assets/img/pdf.svg';
      file.pdf = res;
    }
  });
}
```

showRemoveAlert

> Megnyitja a törlés megerősítése felugró ablakot.

- **idx**: A kiválasztott csatolmány indexe

```typescript
showRemoveAlert(idx): void {
  modalService.openModal('delete').then(success => {
    if (success) {
      deleteHandler(idx)
    }
  });
}
```

deleteHandler

> Meghívja a csatolmányok törlése végpontot, majd frissíti a nézetet.

- **idx**: A kiválasztott csatolmány indexe

```typescript
async deleteHandler(idx) {
  const attachment = uploads[idx];
  const body = { user_id: user.user_id, file_id: attachment.id };

  productsService.deleteAttachment(body).then(() => {
    uploads.splice(idx, 1);
  });
}
```