



Dans une action ngxs de sélection de catégorie,
Je veux récupérer les autres informations de la catégorie à travers une api et ensuite appeler une autre api pour récupérer les template et enfin mettre à jour le state. Comment le faire proprement dans angular

Pour réaliser proprement cette séquence dans NGXS (sélection d'une catégorie → appel API pour détails → appel API pour templates → mise à jour du state), voici une approche recommandée, en gardant ton code modulaire et maintenable :

1. Structure de base

Tu vas créer une **action** de sélection, un **state handler** avec effets asynchrones (`@Actions` ou `@State`) et utiliser des **services** pour appeler tes API.

2. Exemple détaillé

Actions

```
ts

export class SelectCategory {
  static readonly type = '[Category] Select';
  constructor(public categoryId: string) {}
}

export class SetCategoryDetails {
  static readonly type = '[Category] Set Details';
  constructor(public categoryDetails: CategoryDetails) {}
}

export class SetTemplates {
  static readonly type = '[Category] Set Templates';
  constructor(public templates: Template[]) {}
}
```

State

```
ts

@State<CategoryStateModel>({
  name: 'category',
  defaults: {
    selectedCategoryId: null,
    details: null,
    templates: []
  }
})
@Injectable()
export class CategoryState {
  constructor(private categoryService: CategoryService) {}

  @Action(SelectCategory)
  selectCategory(ctx: StateContext<CategoryStateModel>, action: SelectCategory) {
    ctx.patchState({ selectedCategoryId: action.categoryId });
  }
}
```

```
// Enchaîner les appels API
return this.categoryService.getDetails(action.categoryId).pipe(
  tap((details) => {
    ctx.dispatch(new SetCategoryDetails(details));
  }),
  switchMap((details) =>
    this.categoryService.getTemplates(details.typeId) // ou un autre champ
  ),
  tap((templates) => {
    ctx.dispatch(new SetTemplates(templates));
  })
);

@Action(SetCategoryDetails)
setDetails(ctx: StateContext<CategoryStateModel>, action: SetCategoryDetails) {
  ctx.patchState({ details: action.categoryDetails });
}

@Action(SetTemplates)
setTemplates(ctx: StateContext<CategoryStateModel>, action: SetTemplates) {
  ctx.patchState({ templates: action.templates });
}
}
```

3. Services API (exemple)

```
ts

@Injectable({ providedIn: 'root' })
export class CategoryService {
  constructor(private http: HttpClient) {}

  getDetails(categoryId: string): Observable<CategoryDetails> {
    return this.http.get<CategoryDetails>(`/api/categories/${categoryId}`);
  }

  getTemplates(typeId: string): Observable<Template[]> {
    return this.http.get<Template[]>(`/api/templates?type=${typeId}`);
  }
}
```

4. Avantages de cette approche

- **Lisibilité** : chaque action a une seule responsabilité.
- **Réutilisabilité** : `SetCategoryDetails` ou `SetTemplates` peuvent être dispatchées ailleurs si besoin.
- **Testabilité** : chaque handler peut être testé indépendamment.
- **Effet de chaîne** : les appels API s'enchaînent de manière réactive (`switchMap`).

Souhaites-tu également intégrer un système de loading ou gestion d'erreurs dans ce flow ?



Soit une liste de modèles de mail de type observable issue d'un selector ngx. Je souhaite mettre en place une recherche par mot clé pour afficher les modèles de mail qui correspondent au critère. Les résultats doivent apparaître à partir de la 4eme lettre saisie

Pour implémenter proprement une **recherche de modèles de mail à partir du 4^e caractère** dans une liste observable issue d'un selector NGXS, tu peux suivre ces étapes en Angular avec `RxJS` :

1. Selector dans ton State NGXS

Imaginons que tu as un selector qui te retourne tous les modèles :

```
ts

@Selector()
static allTemplates(state: CategoryStateModel): Template[] {
  return state.templates;
}
```

2. Composant : champs de recherche + filtrage

Tu vas utiliser un `FormControl` réactif avec un observable combiné :

```
ts

@Component({
  selector: 'app-template-search',
  template: `
    <input type="text" [formControl]="searchControl" placeholder="Rechercher un modèle" />
    <ul>
      <li *ngFor="let template of filteredTemplates$ | async">
        {{ template.name }}
      </li>
    </ul>
  `
})
export class TemplateSearchComponent implements OnInit {
  searchControl = new FormControl('');
  templates$ = this.store.select(CategoryState.allTemplates);
  filteredTemplates$: Observable<Template[]>;

  constructor(private store: Store) {}

  ngOnInit(): void {
    this.filteredTemplates$ = combineLatest([
      this.templates$,
      this.searchControl.valueChanges.pipe(
        startWith(''),
        debounceTime(300),
        distinctUntilChanged()
      )
    ]).pipe(
      map(([templates, searchTerm]) => {
        if (searchTerm.length < 4) return [];
        const lowerSearch = searchTerm.toLowerCase();
        return templates.filter(t =>
          t.name.toLowerCase().includes(lowerSearch)
        );
      })
    );
  }
}
```

3. Explication des points clés

- `FormControl` : champ de recherche réactif.

- ``debounceTime(300)`` : évite d'exécuter le filtre à chaque frappe.
- ``combineLatest`` : combine la liste des modèles avec la recherche.
- ``searchTerm.length < 4`` : condition pour n'afficher les résultats qu'à partir de 4 lettres.

Optionnel : Ajout d'un indicateur de chargement ou "aucun résultat"

Souhaites-tu afficher un message "aucun résultat" ou un spinner de chargement pendant les requêtes API ?



À la sélection d'un modèle de mail, je veux pouvoir l'insérer tout en ajoutant deux fichiers attachés pdf contenu dans le modèle

Pour **insérer un modèle de mail** avec **deux fichiers PDF attachés inclus dans ce modèle**, dans un contexte Angular + NGXS (et probablement dans un Web Add-in Outlook, vu ta stack), tu peux suivre ce plan :

1. Structure du modèle de mail

Assure-toi que chaque modèle contient les fichiers PDF sous une forme exploitable, par exemple :

```
ts

interface MailTemplate {
  id: string;
  name: string;
  subject: string;
  body: string;
  attachments: { fileName: string; base64: string }[]; // ou Blob/ArrayBuffer
}
```

2. Sélection du modèle

Action NGXS :

```
ts

export class SelectMailTemplate {
  static readonly type = '[Mail] Select Template';
  constructor(public template: MailTemplate) {}
}
```

3. Insertion dans Outlook (dans le composant)

Supposons que tu utilises l'API JavaScript d'Outlook (``office.js``) dans un Web Add-in :

ts

```
selectTemplate(template: MailTemplate) {
  // 1. Insertion du contenu
  Office.context.mailbox.item.body.setSelectedDataAsync(
    template.body,
    { coercionType: Office.CoercionType.Html },
    (result) => {
      if (result.status === Office.AsyncResultStatus.Succeeded) {
        console.log('Contenu inséré');
      }
    }
  );

  // 2. Ajout des pièces jointes
  template.attachments.forEach((att) => {
    Office.context.mailbox.item.addFileAttachmentFromBase64Async(
      att.base64,
      att.fileName,
      (result) => {
        if (result.status !== Office.AsyncResultStatus.Succeeded) {
          console.error('Erreur lors de l'ajout de la pièce jointe :', result.error);
        }
      }
    );
  });
});
```

4. Conversion côté serveur (si besoin)

Si les PDF sont stockés sur le serveur (ex : sous forme de `Blob` ou d'URL), tu dois :

- Soit convertir côté backend en **Base64** et les inclure dans la réponse.
- Soit utiliser un appel HTTP pour les récupérer en `blob` et convertir localement en base64 :

ts

```
getBase64FromBlob(blob: Blob): Promise<string> {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.onloadend = () => resolve((reader.result as string).split(',')[1]);
    reader.onerror = reject;
    reader.readAsDataURL(blob);
  });
}
```

Souhaites-tu un exemple complet incluant le téléchargement des PDF depuis une API ou depuis le localStorage ?



J'ai l'erreur Argument of type (result:any) => void is not assignable to paramètre of type string

L'erreur `Argument of type '(result: any) => void' is not assignable to parameter of type 'string'` signifie que tu as probablement passé une **fonction callback** là où une **chaîne** (`string`) était **attendue** — souvent une **confusion sur l'ordre des arguments** d'une méthode Office.js.

Vérifions ensemble le cas typique :

Méthode concernée

Tu as sûrement ce genre de ligne :

```
ts

Office.context.mailbox.item.body.setSelectedDataAsync(
  template.body,
  { coercionType: Office.CoercionType.Html },
  (result) => { console.log(result); }
);
```

Mais parfois on écrit mal les arguments :

```
ts

// ERREUR : callback à la mauvaise place
Office.context.mailbox.item.body.setSelectedDataAsync(
  (result) => { console.log(result); }, // <-- placé comme 2e argument = erreur
  template.body
);
```

Solution

Assure-toi que tu utilises bien **la bonne signature** de la fonction.

Pour `setSelectedDataAsync` :

```
ts

setSelectedDataAsync(data: string, options?: CoercionTypeOptions, callback?: (result: AsyncResult<v
```

Donc :

```
ts

Office.context.mailbox.item.body.setSelectedDataAsync(
  template.body,
  { coercionType: Office.CoercionType.Html },
  (result) => {
    if (result.status === Office.AsyncResultStatus.Succeeded) {
      console.log('Contenu inséré');
    } else {
      console.error(result.error);
    }
  }
);
```

Pour `addFileAttachmentFromBase64Async` :

```
ts

addFileAttachmentFromBase64Async(base64: string, fileName: string, optionsOrCallback?: any, callback
```

Alors si tu n'as **pas d'options à passer**, ton appel doit ressembler à :

```
ts

Office.context.mailbox.item.addFileAttachmentFromBase64Async(
  att.base64,
  att.fileName,
  (result) => {
    if (result.status === Office.AsyncResultStatus.Failed) {
      console.error(result.error.message);
    }
  }
);
```

Mais si tu passes un objet `options`, ton `callback` est en 4^e paramètre :

```
ts

Office.context.mailbox.item.addFileAttachmentFromBase64Async(
  att.base64,
  att.fileName,
  {}, // options
  (result) => { ... }
);
```

Conclusion

Ton erreur vient très probablement d'un **ordre incorrect des paramètres** dans une fonction `office.js`. Peux-tu coller ici la ligne exacte qui cause l'erreur pour que je la corrige ?



J'ai ajouté la base64 d'un fichier zip. Mais le fichier attaché au mail est de type Fichier pourquoi

Le problème que tu rencontres — où un fichier `.zip` attaché via Base64 dans un Web Add-in Outlook apparaît comme un type générique "Fichier" sans extension ni reconnaissance MIME — est **lié à la façon dont Office.js traite les fichiers Base64 attachés**.

Voici les **causes courantes** et comment les corriger :

1. Problème : Base64 mal formaté ou non reconnu comme ZIP

L'API `addFileAttachmentFromBase64Async()` accepte un **contenu base64 brut sans en-tête MIME**, mais elle **s'appuie sur le nom du fichier** pour déterminer le type.

2. Vérifie ces 3 points clés

A. Extension du fichier

Assure-toi que `fileName` **contient bien l'extension `.zip`** :

```
ts

Office.context.mailbox.item.addFileAttachmentFromBase64Async(
  base64String,           // Sans préfixe "data:..."
  'monfichier.zip',       // L'extension est essentielle
  (result) => {
```

```
if (result.status === Office.AsyncResultStatus.Failed) {
  console.error(result.error.message);
}
);
```

B. Supprime les préfixes base64 (data URI)

Le `base64` doit **exclude** le préfixe `data:application/zip;base64`,`.

Utilise ceci si nécessaire :

```
ts

const base64Clean = base64String.split(',').pop(); // supprime le préfixe data:...
```

C. Taille du fichier

Les fichiers joints en base64 via `Office.js` doivent respecter les limites de taille (environ **3 Mo** par fichier).

3. Exemple complet fonctionnel

```
ts

Office.context.mailbox.item.addFileAttachmentFromBase64Async(
  zipBase64, // Sans préfixe data URI
  'archive-client.zip',
  (result) => {
    if (result.status === Office.AsyncResultStatus.Succeeded) {
      console.log('Fichier ZIP attaché');
    } else {
      console.error('Erreur :', result.error.message);
    }
  }
);
```

4. Résultat attendu

Si :

- Le **contenu est bien en base64 pur**
- Le **nom a une extension** `.zip`

Alors Outlook affichera bien **"archive-client.zip"** avec l'icône ZIP.

Souhaites-tu un utilitaire Angular/TS pour convertir un fichier `Blob` ou `File` en base64 propre à injecter ici ?



Comment office traite les fichiers base64 attaché ?

Office.js traite les fichiers Base64 attachés via l'API `addFileAttachmentFromBase64Async`` de manière spécifique et relativement **basique**. Voici ce qu'il faut comprendre sur **la façon dont Office (notamment Outlook) gère ces fichiers** :

1. L'API attend du Base64 "pur"

Office **n'interprète pas** les métadonnées MIME ou les en-têtes de type ``data:``. Il faut donc fournir **uniquement le contenu encodé** :

Incorrect :

```
text

data:application/pdf;base64,JVBERi0xLjQKJeLjz9MKMiAwIG9iago8PC9...
```

Correct :

```
text

JVBERi0xLjQKJeLjz9MKMiAwIG9iago8PC9...
```

2. Le type de fichier est déterminé par le nom de fichier (et son extension)

Office ne lit pas le contenu du Base64 pour deviner le type MIME.

Si tu attaches un fichier ``fichier`` sans extension :

```
ts

addFileAttachmentFromBase64Async(base64, 'fichier')
```

→ Outlook l'affichera comme un fichier générique ("Fichier") car il **n'a aucune extension** pour l'associer à un type connu.

Si tu ajoutes `` .pdf``, `` .zip``, etc., le type est reconnu automatiquement selon l'extension :

```
ts

addFileAttachmentFromBase64Async(base64, 'monfichier.zip')
```

3. Le fichier est reconstruit côté Outlook

Lorsque tu attaches le base64 via l'API, Outlook :

- Crée un fichier temporaire localement sur le client (navigateur ou client lourd)
- L'associe à l'élément de mail
- L'encode pour le transport en interne (Exchange)

Cela signifie qu'un fichier `` .zip`` correctement nommé sera :

- Téléchargeable comme un fichier `.zip`
- Non interprété à l'ouverture dans Outlook (pas de prévisualisation)

4. Limites imposées par Office.js

- **Taille maximum par pièce jointe** : ~3 Mo
- **Nombre total de pièces jointes** : dépend du client Outlook (Web, Desktop, Mobile)
- **Formats supportés** : tous les formats sont "acceptés" tant que le base64 est valide et bien nommé, mais Outlook n'en prévisualise que certains.

5. Que fait Office avec le base64 ?

Il :

- Décode le base64
- Écrit un fichier temporaire avec le nom fourni
- L'ajoute à la liste des pièces jointes du mail
- (Optionnel) le synchronise avec le brouillon/serveur selon la session

Résumé :

Élément	Comportement Outlook via Office.js
Base64 avec préfixe <code>data:</code>	Refusé / Erreur
Base64 "pur"	OK
Nom sans extension	Fichier générique ("Fichier")
Nom avec <code>.pdf</code> , <code>.zip</code>	Type reconnu, affiché correctement
Analyse du contenu	Non , pas de détection MIME
Taille max	~3 Mo par fichier

Souhaites-tu un helper Angular pour encoder automatiquement un `Blob`, `File` ou `URL` vers un base64 prêt à être injecté ?



Donc un fichier de 5 mégas ne pourra pas être utilisé ?

Exact : **un fichier de 5 Mo ne pourra pas être attaché via `addFileAttachmentFromBase64Async` dans un Web Add-in Outlook**, car cette API a une **limite stricte de taille**, qui est actuellement d'environ **3 Mo** par fichier.

Détails techniques sur les limites :

API : `addFileAttachmentFromBase64Async`

- **Taille max par fichier** : environ **3 Mo (3 145 728 octets)**
- **Limite imposée par le client Outlook, pas contournable**
- **Pas de contournement via découpage de base64 ou streaming**

Conséquences

Si tu essaies d'attacher un fichier >3 Mo :

- L'API renvoie une erreur : `AttachmentSizeExceeded`
- Le fichier **n'est pas attaché** au mail
- Aucun message d'avertissement natif : c'est à toi de gérer le cas côté client

Solutions de contournement possibles

1. Lien vers un téléchargement externe

Tu peux héberger le fichier (ex : sur un blob Azure, SharePoint, OneDrive...) et insérer un **lien dans le corps du mail** :

```
html
```

```
<a href="https://monstockage.com/fichiers/rapport.zip">Télécharger le rapport</a>
```

2. Réduire la taille du fichier en amont

Si possible :

- Compresser les PDF
- Extraire uniquement les pages nécessaires
- Supprimer les métadonnées lourdes

3. Utiliser un complément COM (non web)

Si tu développes un **Add-in Outlook natif (COM/VSTO)**, ces limitations n'existent pas, car tu as un accès direct au client Outlook et au système de fichiers.

Souhaites-tu que je te génère une fonction qui vérifie la taille du fichier en base64 *avant* d'appeler l'API pour éviter les erreurs ?



Existe-t-il une autre API pour attacher le fichier

Dans un **Web Add-in Outlook** (utilisant ``office.js``), **non**, il **n'existe pas d'autre API pour attacher un fichier directement** au mail si tu veux le faire entièrement côté client. L'unique méthode disponible dans ce contexte est :

``addFileAttachmentFromBase64Async()``

C'est la seule fonction permettant d'attacher des fichiers **de manière dynamique** dans le mail via JavaScript dans un Web Add-in.

Les alternatives dépendantes du scénario

1. [Alternative recommandée] Héberger le fichier & insérer un lien de téléchargement

Si ton fichier dépasse 3 Mo :

- Tu peux le **téléverser** côté serveur ou cloud (OneDrive, SharePoint, Azure Blob, S3, etc.)
- Puis insérer un **lien dans le corps de mail** via ``setSelectedDataAsync`` :

```
ts

const downloadUrl = 'https://mon-serveur.com/fichiers/rapport.zip';
const html = `

Veuillez télécharger le fichier ici : <a href="${downloadUrl}">rapport.zip</a></p>`;

Office.context.mailbox.item.body.setSelectedDataAsync(
    html,
    { coercionType: Office.CoercionType.Html }
);


```

2. Utiliser un complément VSTO/COM (hors Web Add-in)

Si tu as le contrôle sur un environnement Windows avec Outlook Desktop, tu peux créer un **VSTO Add-in** en C# qui :

- A un accès complet au système de fichiers
- Peut attacher **n'importe quel fichier sans limite de taille réelle**

Mais ce n'est pas du tout compatible avec un déploiement Web Add-in ou multiplateforme.

3. Pièce jointe pré-existante depuis le serveur via REST API (non via ``office.js``)

Si tu envoies l'email **depuis ton backend** plutôt que depuis Outlook (ex: via ``Microsoft Graph``, SMTP ou SendGrid), tu peux :

- Construire l'email (sujet, corps, PJ) côté serveur
- L'envoyer en tant qu'email Outlook ou Exchange
- Et donner à l'utilisateur un bouton "envoyer depuis moi" ou "prévisualiser"

Mais cela ne passe plus par **Office.js** ou **Outlook Web Add-in** pour la création finale du mail.

Conclusion

Méthode	Fichier > 3 Mo	Outlook intégré ?	Commentaire
`addFileAttachmentFromBase64Async`	Non	Oui	Limité, seule API JS
Lien vers fichier externe	Oui	Oui	Recommandé
VSTO / COM Add-in (C#)	Oui	Oui (Desktop only)	Complexe, natif
Envoi backend via Graph / SMTP / API	Oui	Non	Hors Office.js

Souhaites-tu un exemple de mise en œuvre pour uploader un fichier trop volumineux sur un serveur et insérer le lien dans Outlook automatiquement ?



Au cas où office-js n'interprète pas le base64. Propose moi une fonction local pour traiter la base64 et ressortir le nom et l'extension

Office.js **ne lit pas le contenu base64** pour deviner le type du fichier. Donc si tu reçois un fichier base64 (par ex. depuis une API ou un input `sans nom ni extension, il faut :

- **Décoder la base64**
- **Analyser les premiers octets** du fichier (signature/magic number)
- En déduire le type MIME et l'extension

Fonction : Analyse base64 localement pour détecter l'extension et proposer un nom

```
ts

function detectFileTypeFromBase64(base64: string): { extension: string, mime: string, name: string } {
  const magicNumbers: { [key: string]: { mime: string; extension: string } } = {
    // Signatures (en hex) : https://en.wikipedia.org/wiki/List_of_file_signatures
    '504b0304': { mime: 'application/zip', extension: 'zip' },
    '25504446': { mime: 'application/pdf', extension: 'pdf' },
    'ffd8ffe0': { mime: 'image/jpeg', extension: 'jpg' },
    'ffd8ffe1': { mime: 'image/jpeg', extension: 'jpg' },
    '89504e47': { mime: 'image/png', extension: 'png' },
    '47494638': { mime: 'image/gif', extension: 'gif' },
  };

  try {
    const binary = atob(base64.substring(0, 50)); // décode les premiers octets
    const bytes = Array.from(binary).map(c => c.charCodeAt(0).toString(16).padStart(2, '0')).join('');
    const signature = bytes.substring(0, 8);

    for (const key in magicNumbers) {
      if (signature.startsWith(key)) {
        const { mime, extension } = magicNumbers[key];
        return {
          mime,
          extension,
          name: `fichier.${extension}`,
        };
      }
    }
  } catch (err) {
    console.error('Erreur de décodage base64 :', err);
  }
}
```

```
}
  return null;
}
```

Utilisation :

```
ts

const result = detectFileTypeFromBase64(myBase64);

if (result) {
  console.log(`Type : ${result.mime}`);
  console.log(`Extension : ${result.extension}`);
  console.log(`Nom par défaut : ${result.name}`);

  // Exemple pour Outlook :
  Office.context.mailbox.item.addFileAttachmentFromBase64Async(
    myBase64,
    result.name
  );
} else {
  console.warn('Type de fichier inconnu. Utilisation d'un nom par défaut.');
```

Extensions supportées

Tu peux facilement enrichir `magicNumbers` pour supporter plus de formats : `.docx`, `.xlsx`, etc.

Souhaites-tu que j'ajoute le support pour les fichiers Office (docx, xlsx, pptx) ?



Oui

Parfait ! Voici la version **étendue** de la fonction avec **prise en charge des fichiers Office** (`.docx`, `.xlsx`, `.pptx`) — qui sont tous en réalité des fichiers `.zip` avec un contenu spécifique :

Fonction complète : détecter l'extension depuis un base64

```
ts

function detectFileTypeFromBase64(base64: string): { extension: string, mime: string, name: string } {
  const magicNumbers: { [key: string]: { mime: string; extension: string; officeType?: string } } =
    // PDF
    '25504446': { mime: 'application/pdf', extension: 'pdf' },
    // JPEG
    'ffd8ffe0': { mime: 'image/jpeg', extension: 'jpg' },
    'ffd8ffe1': { mime: 'image/jpeg', extension: 'jpg' },
    // PNG
    '89504e47': { mime: 'image/png', extension: 'png' },
    // GIF
    '47494638': { mime: 'image/gif', extension: 'gif' },
    // ZIP / DOCX / XLSX / PPTX
    '504b0304': { mime: 'application/zip', extension: 'zip' },
  };
}
```

```
try {
  const binary = atob(base64.substring(0, 100)); // lire les premiers octets
  const bytes = Array.from(binary).map(c => c.charCodeAt(0).toString(16).padStart(2, '0')).join('');
  const signature = bytes.substring(0, 8);

  if (signature.startsWith('504b0304')) {
    // Possible ZIP, DOCX, XLSX, PPTX → inspecter contenu
    if (binary.includes('word/')) {
      return { mime: 'application/vnd.openxmlformats-officedocument.wordprocessingml.document', extension: 'docx' };
    } else if (binary.includes('xl/')) {
      return { mime: 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet', extension: 'xlsx' };
    } else if (binary.includes('ppt/')) {
      return { mime: 'application/vnd.openxmlformats-officedocument.presentationml.presentation', extension: 'pptx' };
    } else {
      return { mime: 'application/zip', extension: 'zip', name: 'archive.zip' };
    }
  }

  for (const key in magicNumbers) {
    if (signature.startsWith(key)) {
      const { mime, extension } = magicNumbers[key];
      return { mime, extension, name: `fichier.${extension}` };
    }
  }
} catch (err) {
  console.error('Erreur de décodage base64 :', err);
}

return null;
}
```

Exemple d'utilisation dans un Outlook Web Add-in

```
ts

const result = detectFileTypeFromBase64(base64Data);
const filename = result?.name ?? 'fichier_sans_extension';

Office.context.mailbox.item.addFileAttachmentFromBase64Async(
  base64Data,
  filename,
  (res) => {
    if (res.status === Office.AsyncResultStatus.Succeeded) {
      console.log('Fichier attaché :', filename);
    } else {
      console.error('Erreur Office:', res.error.message);
    }
  }
);
```

Tu veux aussi que je te fournisse une version inverse qui convertit un fichier local (`File` ou `Blob`) vers base64 et utilise cette détection automatiquement ?



On peut pas détecter le nom d'origine du fichier?

Non, **il est impossible de détecter le nom d'origine d'un fichier à partir de son contenu Base64 seul** — car :

1. Le nom de fichier n'est pas encodé dans le Base64

Le Base64 ne contient que les **octets du fichier lui-même** (le contenu binaire brut). Il **n'inclut pas de métadonnées** comme :

- Le nom original
- Le chemin du fichier
- La date de création/modification

2. Exceptions : certains formats contiennent un nom en interne

Il y a des cas très spécifiques où un **nom de fichier ou de document est embarqué dans le contenu** :

- Les fichiers **Word (DOCX)** peuvent avoir un titre de document dans les métadonnées (``docProps/core.xml``)
- Les **archives ZIP** peuvent contenir plusieurs fichiers avec leur nom dans la structure interne
- Les **PDF** peuvent avoir un ``Title``, mais ce n'est **pas forcément le nom du fichier original**

Mais :

- Ce n'est **pas fiable**
- C'est **complexe à parser** manuellement depuis du base64
- Et ce **n'est pas le "nom de fichier"**, mais un "titre interne"

3. Conclusion

Si tu veux conserver le nom d'origine, tu dois :

- Le capturer **au moment de l'envoi ou du téléchargement**
- L'envoyer en **paramètre séparé** avec le base64 (ex : ``{ fileName, base64 }``)

Exemple pour un input HTML (utilisateur charge un fichier)

```
ts

function handleFileInput(event: Event) {
  const input = event.target as HTMLInputElement;
  const file = input.files?.[0];

  if (file) {
    const reader = new FileReader();
    reader.onload = () => {
      const base64 = (reader.result as string).split(',')[1]; // retire "data:..."
      const originalName = file.name;

      // Tu as : base64 + nom original
      console.log('Nom:', originalName);
      console.log('Base64:', base64);
    };
    reader.readAsDataURL(file);
  }
}
```


Souhaites-tu que je t'aide à créer une structure de données `{ name: string, base64: string, mimeType: string }` pour gérer tout ça proprement dans ton add-in ?



est-ce que la somme de la taille des fichiers peut quand même dépasser les 3 méga ?

Non — **dans les Outlook Web Add-ins, la somme des fichiers attachés via `addFileAttachmentFromBase64Async` ne peut *pas* dépasser une certaine limite globale.**

✅ Limite cumulée stricte imposée par Outlook (via `Office.js`) :

Plateforme Outlook	Limite totale de pièces jointes (Base64)	Limite par fichier
Outlook Desktop	~3 Mo cumulés (approximatif)	~3 Mo
Outlook Web	~3 Mo cumulés	~3 Mo
Outlook Mobile	Moins (pas officiellement supporté)	Moins que 3 Mo

Ces 3 Mo concernent la totalité de ce que tu ajoutes via Office.js, que ce soit :

- Un seul fichier de 3 Mo, ou
- Trois fichiers de 1 Mo, ou
- Six fichiers de 500 Ko...

🔍 Pourquoi cette limite ?

- Elle est **technique et de sécurité**, car les add-ins Office fonctionnent dans un bac à sable web.
- La base64 **augmente la taille réelle** d'un fichier de ~33% (1 Mo → 1,33 Mo).
- Outlook doit charger et intégrer ces fichiers **en mémoire côté client**, ce qui impacte la stabilité.

🧠 Bonnes pratiques :

- Toujours **calculer la taille cumulée** des fichiers en base64 avant de les attacher.
- Si la taille dépasse la limite, proposer :
 - Un lien vers un téléchargement externe
 - Un message d'avertissement
 - Une pièce jointe unique réduite/compressée

📎 Exemple de calcul de taille en JS

```
ts
function estimateBase64SizeInBytes(base64: string): number {
  return Math.ceil((base64.length * 3) / 4);
}
```

```
function formatBytes(bytes: number): string {
  return (bytes / (1024 * 1024)).toFixed(2) + ' Mo';
}
```

Et ensuite, tu peux faire :

```
ts

const totalSize = fichiers.reduce((sum, f) => sum + estimateBase64SizeInBytes(f.base64), 0);

if (totalSize > 3 * 1024 * 1024) {
  console.warn('Trop lourd :', formatBytes(totalSize));
}
```

Souhaites-tu que je t'aide à créer un wrapper TypeScript qui gère automatiquement la limite cumulée avant de faire les appels à `addFileAttachmentFromBase64Async` ?



Rédige moi un document technique clair à partir de ce texte

L'API Office-js addFileAttachmentFromBase64Async qui permet d'attacher des pièces jointes prend quatre éléments en paramètre :

- La donnée base64 du fichier
- Le nom du fichier
- L'option d'alignement
- Le callback

Les deux premiers paramètres nous intéresseront :

Dans chaque document contenu dans le modèle de mail, Il nous faudra donc en plus de la donnée en base64 du fichier, le nom de ce dernier et l'extension.

Exemple :

```
{
  nom : 'Conditions-générales',
  extension : '.zip',
  base64 : 'xxxxxxxxxxxxxxxxxx'
}
```

L'API addFileAttachmentFromBase64Async() accepte un contenu base64 brut sans en-tête MIME, mais elle s'appuie sur le nom du fichier pour déterminer le type.

Le base64 doit exclure le préfixe. Exemple pour un fichier zip. data:application/zip;base64,

- Point d'attention :

Taille du fichier :

Les fichiers joints en base64 via Office.js doivent respecter les limites de taille (environ 3 Mo par fichier).

Comment office traite les fichiers base64 attachés ?

1. L'API attend du Base64 "pur"

Office n'interprète pas les métadonnées MIME ou les en-têtes de type data:. Il faut donc fournir uniquement le contenu encodé :

Incorrect :

data:application/pdf;base64,JVBERi0xLjQKJeLjz9MKMiAwIG9iago8PC9...

Correct :

JVBERi0xLjQKJeLjz9MKMiAwIG9iago8PC9...

2. Le type de fichier est déterminé par le nom de fichier (et son extension)

Office ne lit pas le contenu du Base64 pour deviner le type MIME.

Si tu attaches un fichier « fichier » sans extension :

`addFileAttachmentFromBase64Async(base64, 'fichier')`

→ Outlook l'affichera comme un fichier générique ("Fichier") car il n'a aucune extension pour l'associer à un type connu.

Si tu ajoutes .pdf, .zip, etc., le type est reconnu automatiquement selon l'extension :

`addFileAttachmentFromBase64Async(base64, 'monfichier.zip')`

3. Le fichier est reconstruit côté Outlook

Lorsque tu attaches le base64 via l'API, Outlook :

- Crée un fichier temporaire localement sur le client (navigateur ou client lourd)
- L'associe à l'élément de mail
- L'encode pour le transport en interne (Exchange)

Cela signifie qu'un fichier .zip correctement nommé sera :

- Téléchargeable comme un fichier .zip
- Non interprété à l'ouverture dans Outlook (pas de prévisualisation)

4. Limites imposées par Office.js

- Taille maximum par pièce jointe : ~3 Mo
- Nombre total de pièces jointes : dépend du client Outlook (Web, Desktop, Mobile)
- Formats supportés : tous les formats sont "acceptés" tant que le base64 est valide et bien nommé, mais Outlook n'en prévisualise que certains.

5. Que fait Office avec le base64 ?

Il :

- Décode le base64
- Écrit un fichier temporaire avec le nom fourni
- L'ajoute à la liste des pièces jointes du mail
- (Optionnel) le synchronise avec le brouillon/serveur selon la session

Élément Comportement Outlook via Office.js

Base64 avec préfixe data: Refusé / Erreur

Base64 "pur" OK

Nom sans extension Fichier générique ("Fichier")

Nom avec .pdf, .zip Type reconnu, affiché correctement

Analyse du contenu Non, pas de détection MIME

Taille max ~3 Mo par fichier

Détails techniques sur les limites :

API : `addFileAttachmentFromBase64Async`

- Taille max par fichier : environ 3 Mo (3 145 728 octets)
- Limite imposée par le client Outlook, pas contournable
- Pas de contournement via découpage de base64 ou streaming

Conséquences

Si tu essaies d'attacher un fichier >3 Mo :

- L'API renvoie une erreur : `AttachmentSizeExceeded`
- Le fichier n'est pas attaché au mail
- Aucun message d'avertissement natif : c'est à toi de gérer le cas côté client

Solutions de contournement possibles

1. Lien vers un téléchargement externe

Héberger le fichier et insérer un lien dans le corps du mail :

2. Réduire la taille du fichier en amont

Si possible :

- Compresser les PDF
- Extraire uniquement les pages nécessaires
- Supprimer les métadonnées lourdes

3. Utiliser un complément COM (non web)

Si tu développes un Add-in Outlook natif (COM/VSTO), ces limitations n'existent pas, car tu as un accès direct au client Outlook et au système de fichiers.

La somme de la taille des fichiers peut quand même dépasser les 3 méga ?

Non — dans les Outlook Web Add-ins, la somme des fichiers attachés via `addFileAttachmentFromBase64Async` ne peut pas dépasser une certaine limite globale.

✅ Limite cumulée stricte imposée par Outlook (via Office.js) :

Plateforme Outlook Limite totale de pièces jointes (Base64) Limite par fichier

Outlook Desktop ~3 Mo cumulés (approximatif) ~3 Mo

Outlook Web ~3 Mo cumulés ~3 Mo

Outlook Mobile Moins (pas officiellement supporté) Moins que 3 Mo

Ces 3 Mo concernent la totalité de ce que tu ajoutes via Office.js, que ce soit :

- Un seul fichier de 3 Mo, ou
- Trois fichiers de 1 Mo, ou
- Six fichiers de 500 Ko...

🔍 Pourquoi cette limite ?

- Elle est technique et de sécurité, car les add-ins Office fonctionnent dans un bac à sable web.
- La base64 augmente la taille réelle d'un fichier de ~33% (1 Mo → 1,33 Mo).
- Outlook doit charger et intégrer ces fichiers en mémoire côté client, ce qui impacte la stabilité.

tu pourras ajouter des détails que tu trouve utiles

Voici un **document technique clair, structuré et prêt à être partagé** concernant l'utilisation de l'API ``addFileAttachmentFromBase64Async`` dans un Outlook Web Add-in :

Utilisation de l'API ``addFileAttachmentFromBase64Async`` dans un Add-in Outlook

Présentation de l'API

L'API `Office.context.mailbox.item.addFileAttachmentFromBase64Async`` permet d'attacher un fichier à un mail en utilisant son contenu encodé en base64. Elle prend en paramètre :

1. ``base64`` : contenu du fichier (encodé sans préfixe MIME)
2. ``name`` : nom du fichier avec son extension
3. ``options`` : (*facultatif*) options d'alignement (`{ asyncContext }`)
4. ``callback`` : fonction appelée après l'opération

✓ Paramètres obligatoires

Pour attacher correctement un fichier, il faut fournir au minimum :

```
ts
{
  nom: 'Conditions-générales',
  extension: '.zip',
  base64: 'xxxxxxxxxxxxxxxx' // sans préfixe data:
}
```

✗ Le paramètre `base64` doit **exclure tout préfixe** MIME (`data:application/...;base64,`).

📦 Comment Outlook traite le base64 ?

1. Base64 "pur" uniquement

L'API n'accepte **pas** de chaînes contenant `data:application/pdf;base64,...`.

✓ **Correct :**

```
ts
addFileAttachmentFromBase64Async("JVBERi0xLjQKJeLj...", "document.pdf", ...);
```

✗ **Incorrect :**

```
ts
addFileAttachmentFromBase64Async("data:application/pdf;base64,JVBER...", "document.pdf", ...);
```

2. Le type est déterminé par le nom du fichier

Office ne détecte **pas** le type MIME à partir du contenu du base64. Le fichier doit être **nommé correctement** :

- `document` → Outlook affiche : **"Fichier" générique**
- `document.pdf` → Outlook affiche : **PDF avec icône correcte**


3. Reconstruction côté client

Le fichier attaché est :

- Reconstitue **localement** sur le client (navigateur ou Outlook Desktop)
- Attaché en tant que pièce jointe à l'élément de mail
- Envoyé via Exchange comme un fichier ordinaire

Limites techniques imposées par Office.js

Plateforme	Taille max par fichier	Taille cumulée max
Outlook Desktop	~3 Mo	~3 Mo
Outlook Web	~3 Mo	~3 Mo
Outlook Mobile	<3 Mo (non officiel)	Moins que 3 Mo

 **Important** : le total des fichiers ajoutés via `Office.js` ne doit **pas dépasser 3 Mo cumulés**, quelle que soit la combinaison (1×3Mo, 3×1Mo, etc.)

Pourquoi cette limite ?

- Sécurité du bac à sable web
- Base64 augmente la taille réelle (+33%)
- Optimisation mémoire dans Outlook

Comportements à connaître

Élément	Comportement
Base64 avec préfixe <code>`data:`</code>	Refusé
Base64 "pur"	Accepté
Nom sans extension	Affiché comme "Fichier" générique
Nom avec <code>`.pdf`</code> , <code>`.zip`</code> , etc.	Type reconnu automatiquement
Détection MIME à partir du contenu	Non
Taille max par fichier	~3 Mo

Erreurs possibles

``AttachmentSizeExceeded``

Déclenchée si un fichier dépasse ~3 Mo ou si le total dépasse la limite autorisée.

- Aucun message d'erreur natif n'est affiché à l'utilisateur
- C'est à l'add-in de gérer la validation **avant** l'appel à l'API

Bonnes pratiques & contournements

Avant de joindre un fichier :

1. Calculer la taille du base64 :

```
ts

function estimateBase64Size(base64: string): number {
    return Math.ceil((base64.length * 3) / 4);
}
```

2. Valider que la **somme totale des fichiers** ne dépasse pas 3 Mo :

```
ts

if (totalSize > 3 * 1024 * 1024) {
    // afficher un message d'erreur ou proposer un lien de téléchargement
}
```

Solutions alternatives si la limite est atteinte

1. Lien de téléchargement externe

Hébergez le fichier sur un serveur ou service cloud et insérez un lien dans le corps du mail.

2. Réduction de taille

- Compresser les fichiers PDF
- Extraire uniquement les éléments utiles
- Supprimer les métadonnées inutiles

3. Utiliser un Add-in natif (VSTO / COM)

En environnement lourd (non web), il n'y a pas de limite stricte de 3 Mo.

À retenir

- Le nom du fichier est **indispensable** pour que Outlook identifie correctement le type
- Le contenu doit être en **base64 pur**, sans en-tête MIME
- La **taille totale** de tous les fichiers encodés ne doit **jamais dépasser 3 Mo**
- Outlook **n'analyse pas** le contenu du fichier pour en déduire son type
- Il n'existe **aucune méthode pour retrouver le nom d'origine** d'un fichier à partir de son base64

Souhaites-tu aussi un exemple de composant Angular qui affiche la taille cumulée des fichiers et bloque l'ajout si on dépasse la limite ?