Additional Documents API Documentation

Overview

This API allows users to upload, retrieve, and manage additional documents for Individual and Corporate profiles. All documents are stored in Cloudinary and metadata is saved in MongoDB.

Base URL: `{{baseUrl}}/additional-documents`

Authentication: All endpoints require a valid Bearer token in the Authorization header.

Complete Workflow

Step-by-Step Process

```
1. USER LOGIN
   POST /user/login-user
   → Returns: accessToken

2. SEARCH/FIND CUSTOMER
   GET /profiles?searchType=Core customer number&searchValue=1000659
   → Returns: Array of profiles with id and profileType

3. SELECT CUSTOMER
   User selects a customer from search results
   → Extract: profileId (id field) and profileType

4. UPLOAD ADDITIONAL DOCUMENT
   POST /additional-documents/upload
   → Document uploaded and linked to customer

5. VIEW/UPDATE/DELETE (Optional)
   GET /additional-documents?profileId=XXX
   PUT /additional-documents/:id
   DELETE /additional-documents/:id
```

Finding a Customer (Profile)

Before uploading documents, you need to find the customer profile:

Search by Customer ID:

```
GET /profiles?searchType=Core customer number&searchValue=1000659
```

Search by Name:

```
GET /profiles?searchType=Name&searchValue=Matt
```

Generic Search:

```
GET /profiles?search=Matt
```

Response Example:

```
{
  "success": true,
  "results": [
    {
      "id": "65f1234567890abcdef12345",  // ← Use this as profileId
      "custId": "1000659",
      "name": "Matt Dickerson",
      "profileType": "Individual",  // ← Use this as profileType
      "mobile": "5554544",
      "country": "Philippines"
    }
  ]
}
```

Endpoints

1. Upload Additional Document

Endpoint: `POST /additional-documents/upload`

Description: Upload a new document for an existing profile (Individual or Corporate).

Authentication: Required (Bearer Token)

Content-Type: `multipart/form-data`

Request Body (Form Data)

Field	Type	Required	Description
`file`	File	Yes	The document file to upload (jpg, png, jpeg, pdf)
`profileId`	String	Yes	ObjectId of the Individual or Corporate profile
`profileType`	String	Yes	Either `"Individual"` or `"Corporate"`
`uploadDocumentType`	String	No	Type of upload (e.g., "Profile based")
`documentType`	String	Yes	One of: `"Passport copy"`, `"Nationality ID copy"`, `"Bank statement"`, `"Withdrawal slip"`, `"Invoice"`, `"Others"`
`customerName`	String	No	Customer name (snapshot for display)
`transactionNumber`	String	No	Associated transaction number
`remarks`	String	No	Additional remarks or notes

Success Response (201 Created)

```
{
  "message": "Document uploaded successfully",
  "data": {
    "_id": "65f1234567890abcdef12345",
    "userId": "65e9876543210fedcba98765",
    "profileId": "65f000001111122222333344",
    "profileType": "IndividualProfile",
    "customerName": "John Doe",
    "transactionNumber": "TXN123456",
```

```
    "uploadDocumentType": "Profile based",
    "documentType": "Passport copy",
    "file": {
      "fileName": "passport.pdf",
      "fileUrl": "https://res.cloudinary.com/your-
cloud/image/upload/v1234567890/ebmc_uploads/abc123.pdf",
      "fileType": "application/pdf",
      "publicId": "ebmc_uploads/abc123"
    },
    "remarks": "Additional remarks here",
    "status": "Uploaded",
    "isDeleted": false,
    "deletedAt": null,
    "createdAt": "2024-03-13T10:30:00.000Z",
    "updatedAt": "2024-03-13T10:30:00.000Z"
  }
}
```


Error Responses

400 Bad Request - No file uploaded


```
{
  "message": "No file uploaded"
}
```


400 Bad Request - Invalid profile type


```
{
  "message": "Invalid profile type"
}
```


404 Not Found - Profile not found


```
{
  "message": "Profile not found"
}
```


401 Unauthorized - Missing or invalid token


```
{
  "success": false,
  "message": "Authorization token missing"
}
```


---

2. Get Documents (List)

Endpoint: `GET /additional-documents`

Description: Retrieve all additional documents with optional filtering by profile and pagination.

Authentication: Required (Bearer Token)

Query Parameters

Parameter	Type	Required	Default	Description
`profileId`	String	No	-	Filter documents by specific profile ID
`page`	Number	No	1	Page number for pagination
`limit`	Number	No	10	Number of items per page

Success Response (200 OK)


```
{
  "data": [
    {
      "_id": "65f1234567890abcdef12345",
      "userId": "65e9876543210fedcba98765",
      "profileId": "65f000001111122222333344",
      "profileType": "IndividualProfile",
      "customerName": "John Doe",
      "transactionNumber": "TXN123456",
      "uploadDocumentType": "Profile based",
      "documentType": "Passport copy",
      "file": {
        "fileName": "passport.pdf",
        "fileUrl": "https://res.cloudinary.com/...",
        "fileType": "application/pdf",
        "publicId": "ebmc_uploads/abc123"
      },
      "remarks": "Additional remarks",
      "status": "Uploaded",
      "isDeleted": false,
      "createdAt": "2024-03-13T10:30:00.000Z",
      "updatedAt": "2024-03-13T10:30:00.000Z"
    }
  ],
  "pagination": {
    "total": 25,
    "page": 1,
    "limit": 10,
    "pages": 3
  }
}
```


Example Requests

Get all documents (paginated)

```
GET /additional-documents?page=1&limit=10
```

Get documents for a specific profile

```
GET /additional-documents?profileId=65f000001111122222333344&page=1&limit=10
```

Use Cases

The GET endpoint serves multiple purposes:

1. Customer Profile View (Most Common)

```
// Display all documents for a specific customer on their profile page
GET /additional-documents?profileId=65f1234567890abcdef12345
```

Use this when:

- User clicks on a customer profile
- Viewing document history for a specific customer
- Verifying what documents a customer has submitted

2. Admin Dashboard / All Documents View

```
// Display all documents across all customers
GET /additional-documents?page=1&limit=50
```

Use this when:

- Admin wants to monitor all document uploads
- Review pending documents across all customers
- Generate reports or audit trails
- Manage documents from a central dashboard

3. Pagination for Large Datasets

```
// Handle large numbers of documents efficiently
GET /additional-documents?page=2&limit=20
```

Use this when:

- Displaying documents in a table with pagination
- Loading documents incrementally for better performance
- User has many documents to manage

---

3. Update Document

Endpoint: `PUT /additional-documents/:id`

Description: Update an existing additional document's metadata and/or replace the file.

Authentication: Required (Bearer Token)

Content-Type: `multipart/form-data`

URL Parameters

Parameter	Type	Required	Description
`id`	String	Yes	ObjectId of the document to update

Endpoints Summary

Method	Endpoint	Description
POST	`/additional-documents/upload`	Upload a new document
GET	`/additional-documents`	List documents (with pagination & filtering)
PUT	`/additional-documents/:id`	Update an existing document

DELETE	`/additional-documents/:id`	Soft delete a document

Request Body (Form Data)

Field	Type	Required	Description
`file`	File	No	New file to replace the existing one (jpg, png, jpeg, pdf)
`uploadDocumentType`	String	No	Update upload type
`documentType`	String	No	Update document type (must be one of the enum values)
`customerName`	String	No	Update customer name
`transactionNumber`	String	No	Update transaction number
`remarks`	String	No	Update remarks
`status`	String	No	Update status: `"Uploaded"`, `"Verified"`, or `"Rejected"`

Success Response (200 OK)


```
{
  "message": "Document updated successfully",
  "data": {
    "_id": "65f1234567890abcdef12345",
    "userId": "65e9876543210fedcba98765",
    "profileId": "65f000001111122222333344",
    "profileType": "IndividualProfile",
    "customerName": "John Doe Updated",
    "transactionNumber": "TXN123456-UPDATED",
    "uploadDocumentType": "Profile based",
    "documentType": "Bank statement",
    "file": {
      "fileName": "new_bank_statement.pdf",
      "fileUrl": "https://res.cloudinary.com/...",
      "fileType": "application/pdf",
      "publicId": "ebmc_uploads/xyz789"
    },
    "remarks": "Updated remarks",
    "status": "Verified",
    "isDeleted": false,
    "createdAt": "2024-03-13T10:30:00.000Z",
    "updatedAt": "2024-03-13T14:45:00.000Z"
  }
}
```


Error Responses

404 Not Found - Document not found


```
{
  "message": "Document not found"
}
```


400 Bad Request - Cannot update deleted document

```
{
  "message": "Cannot update deleted document"
}
```

Notes

- If a new file is uploaded, the old file is automatically deleted from Cloudinary
- You can update only specific fields without providing all fields
- If no file is provided, only metadata is updated

---

4. Delete Document (Soft Delete)

Endpoint: `DELETE /additional-documents/:id`

Description: Soft delete a document by setting `isDeleted` to `true` and recording the deletion timestamp.

Authentication: Required (Bearer Token)

URL Parameters

Parameter	Type	Required	Description
`id`	String	Yes	ObjectId of the document to delete

Success Response (200 OK)

```
{
  "message": "Document deleted successfully"
}
```

Error Responses

404 Not Found - Document not found

```
{
  "message": "Document not found"
}
```

Data Models

AdditionalDocument Schema

```
{
  userId: ObjectId,              // Reference to User
  profileId: ObjectId,           // Reference to IndividualProfile or CorporateProfile
  profileType: String,           // "IndividualProfile" or "CorporateProfile"
  customerName: String,          // Customer name snapshot
  transactionNumber: String,     // Optional transaction number
  uploadDocumentType: String,    // Type of upload
  documentType: String,          // Enum: ["Passport copy", "Nationality ID copy", "Bank statement",
"Withdrawal slip", "Invoice", "Others"]
  file: {
    fileName: String,
    fileUrl: String,
```

```
    fileType: String,
    publicId: String
  },
  remarks: String,              // Optional remarks
  status: String,               // Enum: ["Uploaded", "Verified", "Rejected"], default: "Uploaded"
  isDeleted: Boolean,           // Soft delete flag, default: false
  deletedAt: Date,              // Deletion timestamp
  createdAt: Date,              // Auto-generated
  updatedAt: Date               // Auto-generated
}
```

Authentication

All endpoints require a valid JWT Bearer token obtained from the user login endpoint.

Header Format:

```
Authorization: Bearer <your_access_token>
```

The token should contain:

- `userId` : User's ObjectId
- `role` : User's role

File Upload Specifications

Supported File Types

- Images: `jpg` , `jpeg` , `png`
- Documents: `pdf`

Storage

- Files are uploaded to **Cloudinary** in the `ebmc_uploads` folder
- The file URL is returned in the response for frontend display

File Size

- Maximum file size is determined by Cloudinary configuration

---

Error Handling

All endpoints return consistent error responses:

```
{
  "message": "Error description",
  "error": "Detailed error message (in development mode)"
}
```

Common HTTP Status Codes

Code	Description
200	Success (GET, DELETE)
201	Created (POST)

400	Bad Request (validation errors)
401	Unauthorized (missing/invalid token)
403	Forbidden (account blocked/deleted)
404	Not Found (resource doesn't exist)
500	Internal Server Error

Frontend Integration Guide

Complete Workflow Example

```
// ============================================
// STEP 1: Login
// ============================================
const loginResponse = await fetch('http://localhost:3000/user/login-user', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    email: 'user@example.com',
    password: 'password123'
  })
});
const { accessToken } = await loginResponse.json();

// ============================================
// STEP 2: Search for Customer
// ============================================
const searchResponse = await fetch(
  'http://localhost:3000/profiles?searchType=Core customer number&searchValue=1000659',
  {
    headers: { 'Authorization': `Bearer ${accessToken}` }
  }
);
const searchData = await searchResponse.json();

// searchData.results = [
//   {
//     id: "65f1234567890abcdef12345",  ← This is the profileId
//     custId: "1000659",
//     name: "Matt Dickerson",
//     profileType: "Individual",  ← This is the profileType
//     mobile: "5554544",
//     ...
//   }
// ]

// ============================================
// STEP 3: User selects a customer
// ============================================
const selectedCustomer = searchData.results[0];
const profileId = selectedCustomer.id;
const profileType = selectedCustomer.profileType;
```

```
// ============================================
// STEP 4: Upload Document for Selected Customer
// ============================================
const formData = new FormData();
formData.append('file', fileInput.files[0]);
formData.append('profileId', profileId);  // From step 3
formData.append('profileType', profileType);  // From step 3
formData.append('documentType', 'Passport copy');
formData.append('customerName', selectedCustomer.name);
formData.append('transactionNumber', 'TXN123456');
formData.append('remarks', 'Additional passport copy');

const uploadResponse = await fetch(
  'http://localhost:3000/additional-documents/upload',
  {
    method: 'POST',
    headers: { 'Authorization': `Bearer ${accessToken}` },
    body: formData
  }
);
const uploadResult = await uploadResponse.json();
console.log('Document uploaded:', uploadResult.data);

// ============================================
// STEP 5: View All Documents for This Customer
// ============================================
const docsResponse = await fetch(
  `http://localhost:3000/additional-documents?profileId=${profileId}`,
  {
    headers: { 'Authorization': `Bearer ${accessToken}` }
  }
);
const docsData = await docsResponse.json();
console.log('Customer documents:', docsData.data);

// ============================================
// STEP 6: Update a Document (if needed)
// ============================================
const documentId = docsData.data[0]._id;
const updateFormData = new FormData();
updateFormData.append('remarks', 'Updated remarks');
updateFormData.append('status', 'Verified');
// Optional: updateFormData.append('file', newFile);

const updateResponse = await fetch(
  `http://localhost:3000/additional-documents/${documentId}`,
  {
    method: 'PUT',
    headers: { 'Authorization': `Bearer ${accessToken}` },
    body: updateFormData
  }
);
const updateResult = await updateResponse.json();
console.log('Document updated:', updateResult.data);
```

```
// ==========================================
// STEP 7: Delete a Document (if needed)
// ==========================================
const deleteResponse = await fetch(
  `http://localhost:3000/additional-documents/${documentId}`,
  {
    method: 'DELETE',
    headers: { 'Authorization': `Bearer ${accessToken}` }
  }
);
const deleteResult = await deleteResponse.json();
console.log('Document deleted:', deleteResult.message);
```

Key Points

1. **Finding a Customer**: Always use `/profiles` endpoint with search parameters first
2. **ProfileId**: The `id` field from the profile search results (MongoDB ObjectId)
3. **ProfileType**: The `profileType` field (`"Individual"` or `"Corporate"`)
4. **Authentication**: All requests require Bearer token from login
5. **File Upload**: Use FormData for file uploads (multipart/form-data)

Helper Functions

```
// Helper: Upload document
const uploadDocument = async (accessToken, profileId, profileType, file, metadata) => {
  const formData = new FormData();
  formData.append('file', file);
  formData.append('profileId', profileId);
  formData.append('profileType', profileType);

  Object.keys(metadata).forEach(key => {
    if (metadata[key]) formData.append(key, metadata[key]);
  });

  const response = await fetch(`${baseUrl}/additional-documents/upload`, {
    method: 'POST',
    headers: { 'Authorization': `Bearer ${accessToken}` },
    body: formData
  });

  return await response.json();
};

// Helper: Get documents for a profile
const getDocuments = async (accessToken, profileId, page = 1, limit = 10) => {
  const response = await fetch(
    `${baseUrl}/additional-documents?profileId=${profileId}&page=${page}&limit=${limit}`,
    {
      headers: { 'Authorization': `Bearer ${accessToken}` }
    }
  );

  return await response.json();
};

// Helper: Update document
```

```
const updateDocument = async (accessToken, documentId, updates) => {
  const formData = new FormData();

  Object.keys(updates).forEach(key => {
    if (updates[key]) formData.append(key, updates[key]);
  });

  const response = await fetch(`${baseUrl}/additional-documents/${documentId}`, {
    method: 'PUT',
    headers: { 'Authorization': `Bearer ${accessToken}` },
    body: formData
  });

  return await response.json();
};

// Helper: Delete document
const deleteDocument = async (accessToken, documentId) => {
  const response = await fetch(`${baseUrl}/additional-documents/${documentId}`, {
    method: 'DELETE',
    headers: { 'Authorization': `Bearer ${accessToken}` }
  });

  return await response.json();
};
```

Testing with Postman

1. Import the provided Postman collection: `Additional_Documents_API.postman_collection.json`
2. Set environment variables:
   - `baseUrl` : Your API base URL (e.g., `http://localhost:3000`)
   - `userAccessToken` : Your JWT access token from login
   - `profileId` : A valid profile ID from your database
3. Test the endpoints in order:
   - Login to get access token
   - Upload a document
   - List documents
   - Delete a document

Notes

- All timestamps are in ISO 8601 format (UTC)
- Documents are soft-deleted (not permanently removed from database)
- The `profileType` field accepts both "Individual"/"Corporate" and "IndividualProfile"/"CorporateProfile" for flexibility
- File uploads are handled via Cloudinary, so ensure proper Cloudinary configuration in your environment variables