Mongoose does not provide built-in pagination for aggregation pipelines. However, you can implement it manually or use libraries like **mongoose-aggregate-paginate-v2** for an out-of-the-box solution.

**Using mongoose-aggregate-paginate-v2**

**What is mongoose-aggregate-paginate-v2?**

It’s a plugin that adds pagination capabilities to Mongoose aggregation pipelines. It simplifies the process of paginating data returned by Mongoose's aggregate() function.

**Installation**

To use it, first install the package:

npm install mongoose-aggregate-paginate-v2

**Setup**

**Schema Plugin**

You need to apply the plugin to your schema:

import mongoose,{Schema} from "mongoose";

import aggregatePaginate from 'mongoose-aggregate-paginate-v2';

const userSchema = new Schema(

    {

        userName : {

            type : String,

            required : true,

            unique : true,

            lowerCase : true,

            trim : true,

            index : true

        },

        email : {

            type : String,

            required : true,

            unique : true,

            lowerCase : true,

            trim : true

        },

        fullName : {

            type : String,

            required : true,

            trim : true

        },

        avatar : {

            type : String, // cloudinary url

            required : true,

        },

        coverImage : {

            type : String

        },

        watchHistory : [{

            type : Schema.Types.ObjectId ,

            ref : 'Video'

            }

        ],

        password : {

            type : String,

            required : [true, 'Password is required']

        },

        refreshToken : {

            type : String,

        }

    },

    {

        timestamps : true

    }

)

userSchema.plugin(aggregatePaginate)

export const User = mongoose.model('User',userSchema)

**Basic Example**

**Aggregation with Pagination**

const paginateAggregation = async (page, limit) => {

    try {

        const aggregateQuery = MyModel.aggregate([

            { $match: { age: { $gte: 18 } } }, // Match documents where age is >= 18

            { $sort: { age: -1 } },           // Sort by age descending

        ]);

        const options = {

            page: page || 1,                 // Default to the first page

            limit: limit || 10,              // Default to 10 items per page

        };

        const result = await MyModel.aggregatePaginate(aggregateQuery, options);

        console.log(result);

    } catch (error) {

        console.error(error);

    }

};

// Call the function with desired page and limit

paginateAggregation(1, 5);

**Result**

The result will be a paginated object like:

{

    "docs": [

        { "name": "John", "age": 25, "city": "New York" },

        { "name": "Jane", "age": 22, "city": "Los Angeles" }

    ],

    "totalDocs": 50,   // Total documents matching the query

    "limit": 5,        // Items per page

    "page": 1,         // Current page

    "totalPages": 10,  // Total number of pages

    "hasNextPage": true,

    "hasPrevPage": false,

    "nextPage": 2,

    "prevPage": null,

    "pagingCounter": 1

}

**Advanced Options**

You can pass additional options to customize pagination:

**Example**

const options = {

    page: 2,              // Specify the page number

    limit: 10,            // Specify the number of documents per page

    customLabels: {       // Custom labels for pagination metadata

        totalDocs: 'totalItems',

        docs: 'items',

        limit: 'perPage',

        page: 'currentPage',

        totalPages: 'pageCount',

        nextPage: 'next',

        prevPage: 'prev'

    },

    collation: { locale: 'en' }, // Set collation for sorting

};

const result = await MyModel.aggregatePaginate(aggregateQuery, options);

**Manual Implementation of Aggregation Pagination**

If you don't want to use mongoose-aggregate-paginate-v2, you can implement pagination manually using $skip and $limit in your aggregation pipeline.

**Example**

const paginateAggregationManually = async (page, limit) => {

    try {

        const skip = (page - 1) \* limit;

        const result = await MyModel.aggregate([

            { $match: { age: { $gte: 18 } } },

            { $sort: { age: -1 } },

            { $skip: skip },

            { $limit: limit }

        ]);

        const totalDocs = await MyModel.countDocuments({ age: { $gte: 18 } });

        const totalPages = Math.ceil(totalDocs / limit);

        console.log({

            docs: result,

            totalDocs,

            totalPages,

            currentPage: page,

            hasNextPage: page < totalPages,

            hasPrevPage: page > 1

        });

    } catch (error) {

        console.error(error);

    }

};

// Call the function with desired page and limit

paginateAggregationManually(1, 5);

**Comparison: Plugin vs. Manual**

| **Aspect** | **Plugin (mongoose-aggregate-paginate-v2)** | **Manual Implementation** |
| --- | --- | --- |
| **Ease of Use** | Easy; minimal setup required | Requires manual implementation |
| **Customization** | Supports custom labels, collation, etc. | Customizable, but more complex |
| **Performance** | Optimized for common use cases | Direct control over pipeline |
| **Dependencies** | Requires an external library | No extra dependencies |

**Conclusion**

* Use **mongoose-aggregate-paginate-v2** if you want an easy-to-use and feature-rich pagination solution.
* Implement manual pagination if you need fine-grained control or want to avoid external dependencies.