Chapter 8. Working with Data

COMP7270 Web and Mobile Programming
& COMP7980 Dynamic Web and Mobile Programming

12th March 2025, Dr Shichao MA

JavaScript Methods

- The push() method adds one or more elements to the end of an array and returns the new length of the array.

```
const fruits = ['apple', 'banana'];
fruits.push('orange');
console.log(fruits); // Output: ['apple', 'banana', 'orange']
```

- concat(): **Concatenates** two or more arrays and returns a new array.

```
const fruits = ['apple', 'banana'];
const vegetables = ['carrot', 'potato'];
const combined = fruits.concat(vegetables);
console.log(combined); // Output: ['apple', 'banana', 'carrot', 'potato']
```

JavaScript Methods

- map(): Creates a new array by applying a function to each element in the original array.

```
const numbers = [1, 2, 3];
const doubled = numbers.map((number) => number * 2);
console.log(doubled); // Output: [2, 4, 6]
```

- filter(): Creates a new array with all elements that **pass a test specified by a function**.

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter((number) => number % 2 === 0);
console.log(evenNumbers); // Output: [2, 4]
```

JavaScript Methods

- join(): Joins all elements of an array into **a string**, optionally separated by a specified delimiter.

```
const fruits = ['apple', 'banana', 'orange'];
const joinedString = fruits.join(', ');
console.log(joinedString); // Output: 'apple, banana, orange'
```

- includes(): Checks **whether an array contains a specific element** and returns true or false.

```
const fruits = ['apple', 'banana', 'orange'];
console.log(fruits.includes('banana')); // Output: true
console.log(fruits.includes('grape')); // Output: false
```


```javascript
async function fetchData() {
  try {
    const response = await fetch(`/bookings?${params}`);
    const result = await response.json();

    perPage.value = result.perPage;
    total.value = result.total;
    data.value = result.bookings;

    loading.value = false;
  } catch (error) {
    // Handle any errors
    // that occur during the fetch
    console.error(error);
  }
}

fetchData();
```

Async/Await Approach

Promise Chain Approach

```javascript
fetch(`/bookings?${params}`)
  .then((response) => response.json())
  .then((result) => {
    perPage.value = result.perPage;
    total.value = result.total;
    data.value = result.bookings;

    loading.value = false;
  })
  .catch((error) => {
    // Handle any errors
    // that occur during the fetch
    console.error(error);
  });
```

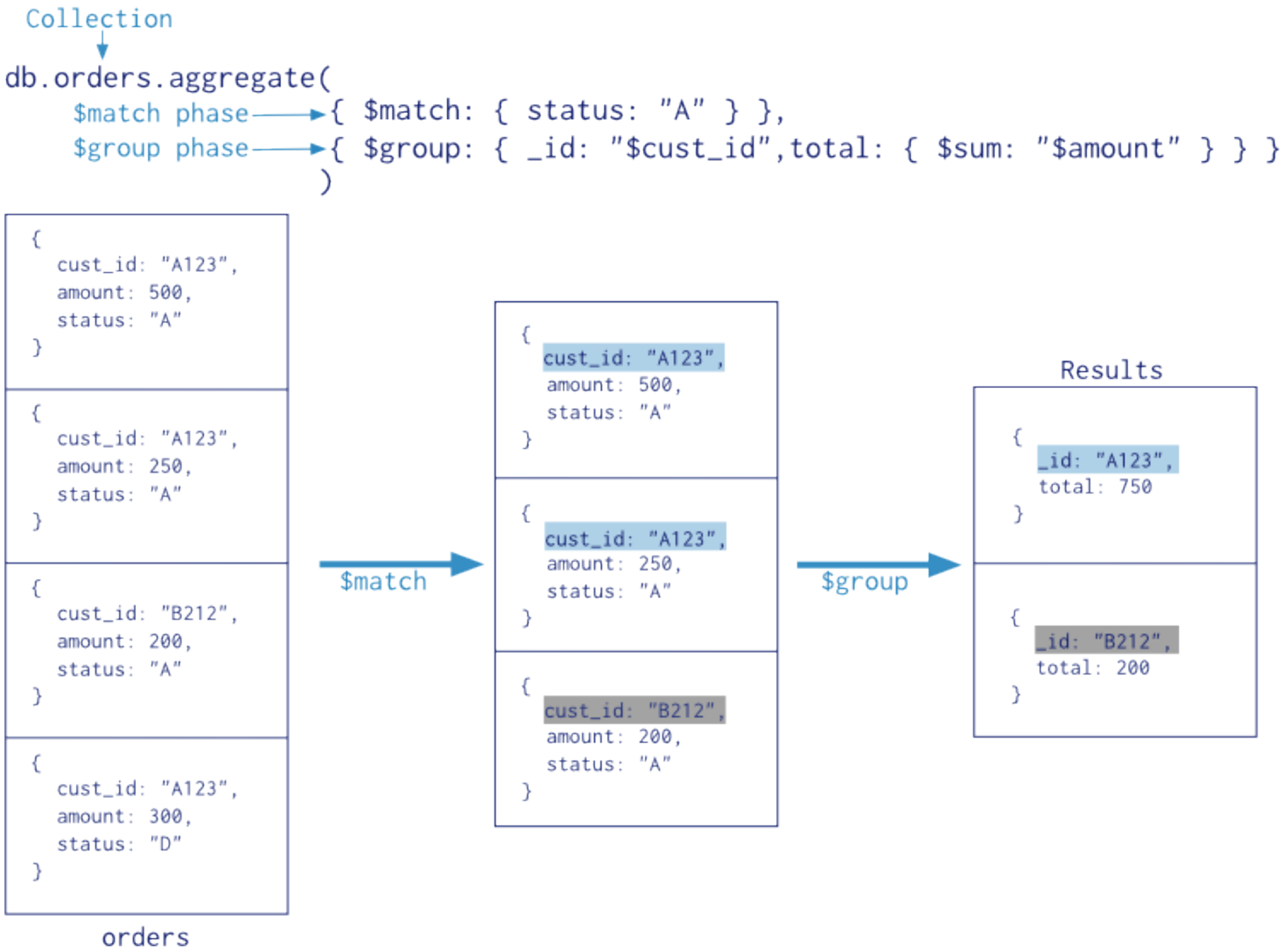

```
// Get the total number of bookings per superhero
router.get('/stats/superhero', async function (req, res) {
    const db = await connectToDB();
    try {
        let result = await db.collection("bookings").aggregate([
            // non null superhero
            { $match: { superhero: { $ne: null } } },
            { $group: { _id: "$superhero", total: { $sum: 1 } } }
        ]).toArray();

        res.json(result);
    } catch (err) {
        res.status(400).json({ message: err.message });
    } finally {
        await db.client.close();
    }
});
```

Aggregate

- In MongoDB, the aggregate is a powerful method that allows you to perform advanced data aggregation operations on collections. It provides a flexible and expressive way to process and transform data within MongoDB.

- The aggregate method **takes an array of stages as its parameter**. Each stage represents a specific operation that is applied to the documents in the collection. These stages are executed in sequence, allowing you to build complex data pipelines.

Aggregate


Collection
db.orders.aggregate(
$match phase
{ $match: { status: "A" } },
$group phase
{ $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
)
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
$match
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
$group
Results
{ _id: "A123", total: 750 }
{ _id: "B212", total: 200 }

Aggregate Operations

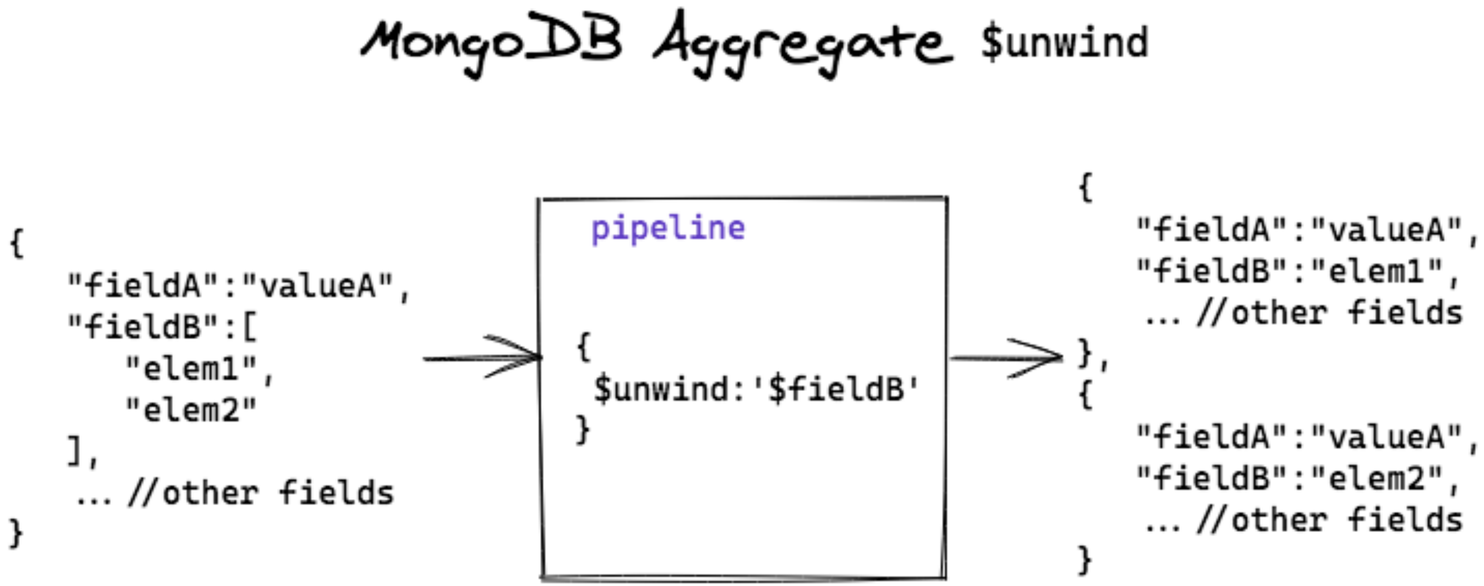
- $match: Filters the documents based on specific criteria, similar to the find method. It allows you to select a subset of documents for further processing.
- $group: Groups the documents by a specified key and performs aggregations on each group. It is commonly used for tasks such as **calculating sums, averages, or counts**.
- $project: Specifies **which fields to include or exclude** from the output documents. It allows you to reshape the documents and create new computed fields.

Aggregate Operations

- $sort: Sorts the documents based on one or more fields, either in ascending or descending order.
- $limit and $skip: Controls the number of documents returned by the aggregation pipeline. $limit restricts the number of documents, while $skip skips a specified number of documents.
- $unwind: Deconstructs an array field from the input documents and **outputs one document for each element of the array**. It is useful when you need to perform further operations on individual elements of an array.

Aggregate Operations

$unwind



Group By Example

- This shows the result of a “group by” operation, with **total** showing the number of documents belonging to a particular superhero.

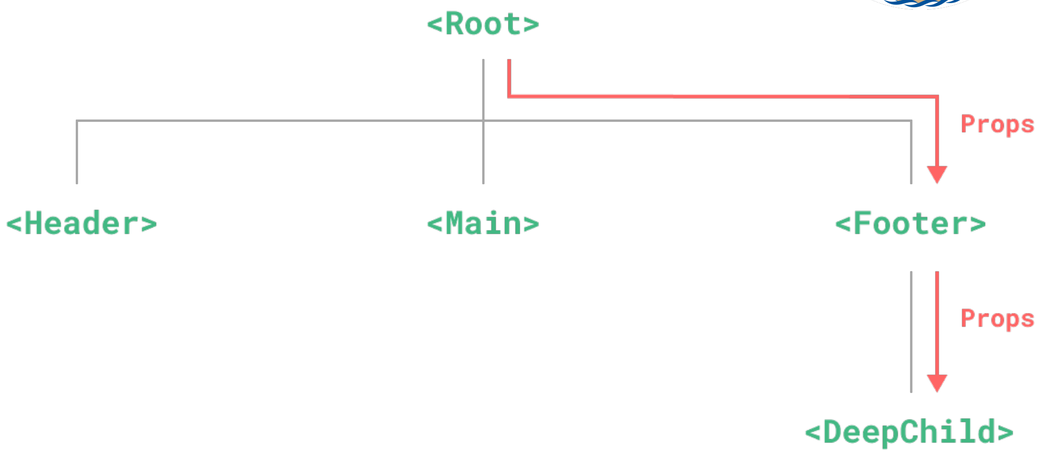

```
[
  {
    "_id": "Hulk",
    "total": 209
  },
  {
    "_id": "Captain America",
    "total": 222
  },
  {
    "_id": "Thor",
    "total": 221
  },
  {
    "_id": "Iron Man",
    "total": 229
  },
  {
    "_id": "Hawkeye",
    "total": 248
  },
  {
    "_id": "Black Widow",
    "total": 273
  }
]
```

Navigation in the Vue.js SPA

- To navigate the frontend components, we use RouterLink
- The navigated component will be rendered in **<RouterView />**

```
<template>
  <nav>
    <RouterLink to="/">Home</RouterLink>
    <RouterLink to="/about">About</RouterLink>
  </nav>
  <RouterView />
</template>
```

Props



- Usually, when we need to pass data from the parent to a child component, we use **Props**.
- Here, we provide **Props** (properties) via the template.

```
<DonutChart team="" />
<DonutChart team="Avengers" />
<DonutChart team="JLA" />
```

ChartsView,

The parent

DonutChart, the child element

```
<script setup>
import { ref, onMounted, defineProps } from "vue";

const props = defineProps({
    team: String,
});
```