**MONGO \_DB\_AGGREGATION**

**DATA COLLECTIONS:**

**1.AMENITIES**

1. {

amenityCount: -1,

}

**DESCRIPTION:**

This query uses the $sort stage to arrange amenities in descending order based on their count. By specifying "amenityCount": -1, it brings the most frequently listed amenities to the top. This helps identify the most common or popular facilities across properties. Useful for decision-making in property upgrades or marketing highlights.

2. "$amenities"

**DESCRIPTION:**

This operation uses the **$unwind** stage to break down the amenities array into individual documents.  
Each amenity from the array becomes a separate document for further grouping or analysis.  
It allows detailed processing like counting or filtering specific amenities.  
Useful for identifying trends in feature offerings across multiple properties.

**2. MAINTENANCE\_REQUESTS**

1. {

\_id: {

$dateToString: {

format: "%Y-%m-%d",

date: {

$toDate: "$createdAt",

},

},

},

count: {

$sum: 1,

},

}

**DESCRIPTION:**

This aggregation uses the **$group** stage to count the number of maintenance requests made per day.  
It converts the createdAt field to a date string (YYYY-MM-DD) using $dateToString for grouping.Then, it uses $sum: 1 to count how many requests were submitted on each date.  
Useful for tracking daily maintenance load and identifying peak request periods.

2. {

\_id: "$status",

totalRequests: {

$sum: 1,

},

}

**DESCRIPTION:**

This aggregation uses the **$group** stage to calculate the total number of maintenance requests by their status.  
It groups records based on the status field (e.g., Pending, Resolved) and counts them using $sum: 1.The result shows how many requests exist in each status category.Useful for monitoring service progress and identifying backlog or resolved issues.

**3. OWNERS**

1.{

\_id: "$tower",

numberOfOwners: {

$count: {},

},

}

**DESCRIPTION:**

This aggregation uses the **$group** stage to count the number of owners in each tower.  
By grouping documents using "$tower" and applying $count, it totals how many owners are associated with each tower.  
This helps analyze owner distribution across different buildings.  
Useful for tower-level management, communication, and service planning.

2. {

"bankDetails.accountNumber": {

$concat: [

{

$substr: [

"$bankDetails.accountNumber",

0,

{

$subtract: [

{

$strLenBytes:

"$bankDetails.accountNumber",

},

4,

],

},

],

},

"\*\*\*\*",

{

$substr: [

"$bankDetails.accountNumber",

{

$subtract: [

{

$strLenBytes:

"$bankDetails.accountNumber",

},

4,

],

},

4,

],

},

],

},

}

**DESCRIPTION:**

This operation uses the **$set** stage to mask bank account numbers for privacy.  
It keeps only the last 4 digits visible and replaces the rest with asterisks (\*\*\*\*) using $concat and $substr.  
Helps protect sensitive financial data while still allowing identification.  
Useful for displaying account numbers securely in UI or reports.

**4. PROPERTIES**

1. {

\_id: "$property\_type",

averageRent: {

$avg: "$rentAmount",

},

} **DESCRIPTION:**

This aggregation uses the **$group** stage to calculate the **average rent** for each property type.  
It groups properties by property\_type (e.g., Studio, Affordable) and applies $avg on rentAmount.  
The result shows how rental prices vary across different categories.  
Useful for pricing analysis, budgeting, and market comparison.

2. {

\_id: "$petAllowed",

count: {

$sum: 1,

},

}

**DESCRIPTION:**

This aggregation uses the **$group** stage to count properties based on whether pets are allowed.  
It groups data by the petAllowed field (true or false) and uses $sum: 1 to count each group.  
The result shows how many properties permit or restrict pets.  
Useful for filtering pet-friendly listings and understanding tenant preferenc

**5. RENTAL\_AGREEMENTS**

1. {

status: "active",

}

**DESCRIPTION:**

This query uses the **$match** stage to filter and retrieve only the documents where the status is "active".It is used to find all currently valid or ongoing rental agreements.This helps property managers focus on active leases for monitoring and rent tracking.

2. {

\_id: "$status",

count: {

$sum: 1,

},

}

**DESCRIPTION:**

This aggregation uses the **$group** stage to count the number of rental agreements by their status.  
It groups documents using the status field (e.g., Active, Upcoming, Expired) and uses $sum: 1 to total each group.  
The result shows how many agreements fall under each status category.  
Useful for tracking lease lifecycle stages and managing upcoming or expiring agreements.

**6. TENANTS**

1.

{

\_id: null,

averageRentPaid: {

$avg: "$rentPaid",

},

}

**DESCRIPTION:**

This aggregation uses the $group stage to calculate the average rent paid by all tenants.  
By setting \_id: null, it treats all documents as a single group and applies $avg to the rentPaid field.  
The result gives an overall average rent value across the entire tenants collection.  
Useful for analyzing rent trends and setting competitive pricing.

2.

{

\_id: "$tower",

tenantCount: {

$sum: 1,

},

}

**DESCRIPTION:**

This aggregation uses the **$group** stage to count the number of tenants living in each tower.By grouping documents using "$tower" and applying $sum: 1, it totals tenants per tower.  
The result shows how many tenants reside in each building block.Useful for understanding occupancy distribution and planning tower-specific services.