Task 1

a) **Display 3-bedroom apartments in Barcelona.**

Mongo Shell: db.listingsAndReviews.find({ "bedrooms": 3, "address.market": "Barcelona" })
Mongo Compass: On Filter field: { "bedrooms" : 3, "address.market" : "Barcelona" }

b) **Display listings in Barcelona that has both Wifi and Cable TV.**

Mongo Shell: db.listingsAndReviews.find({ "address.market": "Barcelona", "amenities": { $all: ["Wifi", "Cable TV"] } })
Mongo Compass: On Filter field: { "address.market" : "Barcelona", "amenities" : { "$all": ["Wifi", "Cable TV"] } }

c) **Compute and display the average daily rate for Barcelona properties.**

Mongo Shell:
```
db.listingsAndReviews.aggregate([
  { $match: { "address.market": "Barcelona" } },  // Filter for Barcelona properties
  { $group: { _id: null, avgPrice: { $avg: "$price" } } }  // Compute average price
])
```
Mongo Compass:

c.1) Navigate to the listingsAndReviews collection.
c.2) Click on the Aggregations tab.
c.3) Now, set up the following stages:
Stage 1: $match

- Stage Operator: **$match**
- Stage Content:
  {"address.market": "Barcelona"}

Stage 2: $group

- Stage Operator: **$group**
- Stage Content:
  {"_id": null,"avgPrice": { "$avg": "$price" }}

d) **Display the average price for 3-bedroom apartments in Barcelona.**

Mongo Shell:
```
db.listingsAndReviews.aggregate([
  { $match: { "address.market": "Barcelona", "bedrooms": 3 } },  // Filter for Barcelona properties with 3 bedrooms
  { $group: { _id: null, avgPriceFor3Bedrooms: { $avg: "$price" } } }  // Compute average price
])
```
Mongo Compass:
d.1) Navigate to the listingsAndReviews collection.

d.2) Click on the Aggregations tab.
d.3) Now, set up the following stages:
Stage 1: $match

- Stage Operator: **$match**
- Stage Content:

```
{"address.market": "Barcelona", "bedrooms": 3}
```

Stage 2: $group

- Stage Operator: **$group**
- Stage Content:

```
{"_id": null,"avgPriceFor3Bedrooms": { "$avg": "$price" }}
```

e) **Display the median price for 3-bedroom apartments in Barcelona.**

Mongo Shell:

```
db.listingsAndReviews.aggregate([
 // Filter for 3-bedroom apartments in Barcelona
 { $match: { "address.market": "Barcelona", "bedrooms": 3 } },

 // Sort by price
 { $sort: { "price": 1 } },

 // Group and push prices into an array + count them
 { $group: { _id: null, prices: { $push: "$price" }, count: { $sum: 1 } } },

 // Calculate median
 {
   $project: {
     median: {
       $cond: [
         { $mod: ["$count", 2] }, // If odd
         { $arrayElemAt: ["$prices", { $floor: { $divide: ["$count", 2] } }] }, // Get the middle value
         { // If even, average the two middle values
           $avg: [
             { $arrayElemAt: ["$prices", { $subtract: [{ $divide: ["$count", 2] }, 1] }] },
             { $arrayElemAt: ["$prices", { $divide: ["$count", 2] }] }
           ]
         }
       ]
     }
   }
 }
])
```

Mongo Compass:

1. **Open MongoDB Compass** and connect to your database.

2. Navigate to the **listingsAndReviews** collection.
3. Click on the **Aggregations** tab to start a new aggregation pipeline.

Now, add the stages one by one:

Stage 1: $match

- **Stage Operator**: **$match**
- **Stage Content**:

```
{"address.market": "Barcelona", "bedrooms": 3}
```

Stage 2: $sort

- **Stage Operator**: **$sort**
- **Stage Content**:

```
{"price": 1}
```

Stage 3: $group

- **Stage Operator**: **$group**
- **Stage Content**:

```
{"_id": null, "prices": { "$push": "$price" }, "count": { "$sum": 1 }}
```

Stage 4: $project (to calculate median)

- **Stage Operator**: **$project**
- **Stage Content**:

```
{"median": { "$cond": [ { "$mod": ["$count", 2] }, { "$arrayElemAt": ["$prices", { "$floor": { "$divide":
["$count", 2] } }] }, { "$avg": [ { "$arrayElemAt": ["$prices", { "$subtract": [{ "$divide": ["$count", 2] },
1] }] }, { "$arrayElemAt": ["$prices", { "$divide": ["$count", 2] }] } ] } ] }}
```

Display 10 "least-expensive" cities for travelers. The "least expensive" is determined by averaging the prices of all properties in each city.

Mongo Shell:

```
db.listingsAndReviews.aggregate([
 {
  $group: {
   _id: "$address.market", // Group by city
   avgPrice: { $avg: "$price" } // Compute average price for each city
  }
 },
 {
  $sort: { avgPrice: 1 } // Sort by average price in ascending order
 },
 {
  $limit: 10 // Limit results to top 10 cities
 }
])
```

Mongo Compass:

1. Open MongoDB Compass and connect to **sample_airbnb** database.

2. Navigate to the **listingsAndReviews** collection.
3. Click on the **Aggregations** tab.
4. Add the stages provided in the Mongo shell command.

- Stage 1 (**$group**):

```
{ "_id": "$address.market", "avgPrice": { "$avg": "$price" } }
```

- Stage 2 (**$sort**):

```
{ "avgPrice": 1}
```

- Stage 3 (**$limit**):

```
10
```

2. The owner of the listing named “Be Happy in Porto” wishes to add “Netflix” as a new amenity. Write a MongoDB statement to add this new amenity into the amenities list of the existing listing.

Mongo Shell:

```
db.listingsAndReviews.updateOne(
  { name: "Be Happy in Porto" }, // Filter by the name of the listing
  { $addToSet: { amenities: "Netflix" } } // Add "Netflix" to the amenities array
)
```

Justification: I used **$addToSet** instead of **$push** to ensure that "Netflix" is only added if it isn't already present in the amenities array. This prevents duplicate entries. If "Netflix" is already in the array, it won't be added again.

Mongo Compass:
1. On Filter field: **{ "name": "Be Happy in Porto" }.**
4. Click the pencil icon on the desired document to edit.
5. Manually add **"Netflix"** to the `**amenities**` array.
6. Click "Update" to save changes.

3. Write a MongoDB statement to add this comment into the reviews array of the existing listing of this property.

Mongo Shell:

```
db.listingsAndReviews.updateOne(
   { "name": "Be Happy in Porto" },
   {
      $push: {
         "reviews": {
            "_id": ObjectId(),
            "date": new Date(),
            "listing_id": "10083468",
            "reviewer_id": "19765799",
            "reviewer_name": "Guest",
            "comments": "This holiday accommodation did not meet my expectation. Being in Portugal, I wanted to watch bull-fighting from the balcony. But, neither balcony nor bull-fighting nearby are there."
          }
        }
    }
)
```

Mongo Compass:

1. On Filter field: **{ "name": "Be Happy in Porto" }**.
2. Click the pencil icon on the desired document to edit.
3. Manually add **a comment** to the `**review**` array.
4. **Save**: Click "UPDATE" at the bottom.

4. Write a MongoDB statement to change the daily rate of the existing listing.

Mongo Shell:

```
db.listingsAndReviews.updateOne(
    { "name": "Be Happy in Porto" },
    {
        $set: {
            "price": Decimal128.fromString("40.00")
        }
    }
)
```

Mongo Compass:

1. On Filter field: **{ "name": "Be Happy in Porto" }**.
2. Find the **price** field and change its value to **40.00**.
3. Click "UPDATE" at the top-right to save changes.

5. Write a MongoDB statement to delete the feedback currently stored in the reviews array.

Mongo Shell:

```
db.listingsAndReviews.updateOne(
    { "name": "Be Happy in Porto" },
    {
        $pull: {
            "reviews": {
                "reviewer_id": "19765799",
            "comments": "This holiday accommodation did not meet my expectation. Being in Portugal, I wanted to watch bull-fighting from the balcony. But, neither balcony nor bull-fighting nearby are there."
         }
      }
   }
)
```

Mongo Compass:

1. Connect to your database.
2. Go to **listingsAndReviews** collection.
3. In the search bar, enter: **{ "name": "Be Happy in Porto" }**
4. Locate the document that we have just added for question 3.
5. Click the pencil/edit icon to modify the document.
6. Manually delete the specific review from the **reviews** array.
7. Save/apply the changes by clicking **Update** at the bottom of the screen.

Task 2

2.1)

Making Changes to Airbnb Database:

1. **Changes to listingsAndReviews**:
   - two new times are added:
     - **checkInTime**: The time guests can start checking in.
     - **checkOutTime**: The time guests need to leave by.

Additional configuration for listingAndReview collection.

```
db.listingsAndReviews.updateMany(
   {},
   {
      $set: {
         checkInTime: "2:00 PM",
         checkOutTime: "11:00 AM"
      }
   }
);
```

2. **Creation of the UserandBooking Collection:**

This new collection will maintain comprehensive booking details of our users. A key attribute in this collection would be **listing_id**. This will serve as a reference to the specific listing in the listingsAndReviews collection that a particular booking is associated with. In essence, we're leveraging the **"Referencing"** technique where each booking record points to a specific listing, ensuring data integrity and simplifying future queries or aggregations.

Create User&Booking collection and update with the below example.

```
db.userAndBooking.insertMany([
  {
    userId: 1,
    name: "Jenna Gilbert",
    email: "Jenn21@gmail.com",
    daytimePhoneNumber: "123-456-7890",
    mobileNumber: "123-456-7891",
    postalAddress: "123 Street, City, Country",
    homeAddress: "456 Lane, City, Country",
    bookings: [
      {
        Bookingid: 1,
        listingId: "10021707",
        bookingDate: new Date("2023-11-03"),
        arrivalDate: new Date("2023-11-10"), // Example arrival date
        departureDate: new Date("2023-11-17"), // Example departure date
        depositPaid: 50.00,
        balanceDue: 150.00,
        dueDateForBalance: new Date("2023-10-20"),
        numberOfGuests: 2,
        guests: [
          {
            name: "John Doe",
            age: 28
          },
          {
            name: "Jane Smith",
            age: 25
          }
        ]
      },
      {
        Bookingid: 2,
        listingId: "10006547",
        bookingDate: new Date("2023-12-05"),
        arrivalDate: new Date("2023-12-10"), // Example arrival date
        departureDate: new Date("2023-12-15"), // Example departure date
```

```
            depositPaid: 60.00,
            balanceDue: 140.00,
            dueDateForBalance: new Date("2023-11-20"),
            numberOfGuests: 1,
            guests: [
              {
                name: "Alice Johnson",
                age: 30
              }
            ]
          }
        ]
      },
      {
        userId: 2,
        name: "Tom Smith",
        email: "tomsmith@gmail.com",
        daytimePhoneNumber: "123-456-7892",
        mobileNumber: "123-456-7893",
        postalAddress: "789 Avenue, City, Country",
        homeAddress: "012 Road, City, Country",
        bookings: [
          {
            Bookingid: 3,
            listingId: "10009999",
            bookingDate: new Date("2023-12-10"),
            arrivalDate: new Date("2023-12-15"), // Example arrival date
            departureDate: new Date("2023-12-22"), // Example departure date
            depositPaid: 70.00,
            balanceDue: 130.00,
            dueDateForBalance: new Date("2023-11-25"),
            numberOfGuests: 3,
            guests: [
              {
                name: "Bob Brown",
                age: 35
              },
              {
                name: "Charlie Clark",
                age: 40
              },
              {
                name: "David Davis",
                age: 45
              }
            ]
          }
        ]
      },
      {
        userId: 3,
        name: "Anna Johnson",
        email: "anna.johnson@gmail.com",
        daytimePhoneNumber: "123-456-7894",
        mobileNumber: "123-456-7895",
        postalAddress: "345 Boulevard, City, Country",
        homeAddress: "678 Street, City, Country",
        bookings: [] // No bookings for Anna in the provided data
      }
      ]);
```

2.2)
Justification for Choosing Referencing:

1. **Using reference When the "many" side in one-to-many relationships is very large or changes frequently:** The listingsAndReview collection contains property listings, each with a unique listing_id. The userAndBooking collection tracks user bookings, where each booking references a specific listing through the **listing_id**. This establishes a one-to-many relationship: a single listing can have multiple bookings.

2. **Scalability:** I expected many more bookings in the future. To handle this, I used the referencing method. Instead of putting booking details directly inside each listing (which could have made listings large and slow to access), I kept bookings in a separate collection. Each booking pointed to its related listing. This way, as the number of bookings increased, our system remained fast and manageable.

3. **Optimized Data Access**: Keeping bookings separate means when users are browsing listings in listingAndReview collection, the system doesn't load all the booking details unnecessarily. This streamlines the data access process based on actual needs.

4. **Efficient Updates**: Separate booking records allow for quicker write operations, especially crucial for popular listings. In contrast, embedding booking within listings might have required frequent edits to big documents, which would have impacted speed and efficiency.

5. **Future-proof Flexibility**: A separate bookings collection ensures flexibility for future data expansion without restructuring the main listings, accommodating evolving business needs.

6. **Simplified Queries**: With a separate bookings collection, certain queries, like fetching all bookings of a specific user or date range, become more straightforward.

7. **Data Integrity**: By referencing, there's a reduction in data redundancy. For instance, user details don't need to be repeated across multiple embedded bookings, ensuring data consistency and reduced storage needs.

8. **Handling Detailed Bookings:** Bookings can have a lot of details, like lists of guests or payment activities. Using referencing makes it easier to work with and update these detailed pieces of information.

9. **Consistency & Atomic Updates**: Keeping bookings separate ensures each booking's data remains consistent and can be atomically updated, reducing the risk of concurrency issues.

10. **Flexible Booking Look-Up:** Booking information, being versatile, can be referenced from multiple angles - by the listing, by users, by date, etc. This multi-referencing scenario is better managed when bookings are kept in a dedicated collection.

In conclusion, while embedding can provide quick read operations by grouping related data together, the referencing approach in this scenario offers superior flexibility in querying or updating one collection without affecting another, scalability, and maintainability, especially given the dynamic and expansive nature of booking data.

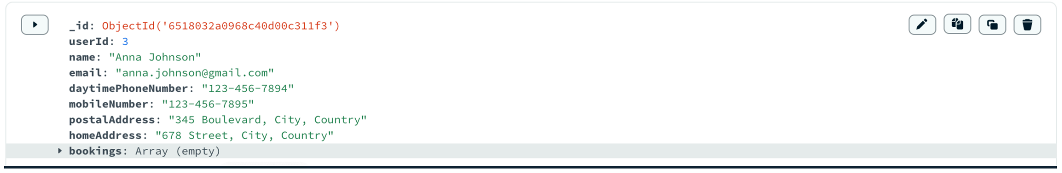
Disadvantages of embedding

1. **Document Size Limit**: MongoDB has a limit on the document size (currently 16 MB). Extensive embedding can result in large documents that might hit this limit.
2. **Data Redundancy**: If the embedded data needs to be repeated across many documents, it will cause redundancy, which can consume extra storage and make updates more complex.
3. **Complex Updates**: If we frequently need to update or query based on the embedded data, these operations can become more complex compared to using references.

2.3)

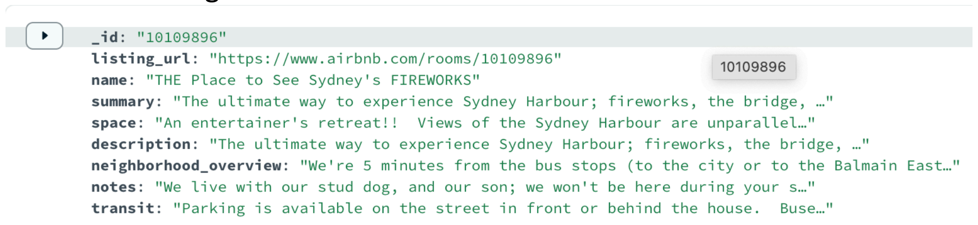
2.3.a) Make a new booking for a chosen property

Before: UserId 3 "Anna Johnson" does not have any booking



```
_id: ObjectId('6518032a0968c40d00c311f3')
userId: 3
name: "Anna Johnson"
email: "anna.johnson@gmail.com"
daytimePhoneNumber: "123-456-7894"
mobileNumber: "123-456-7895"
postalAddress: "345 Boulevard, City, Country"
homeAddress: "678 Street, City, Country"
bookings: Array (empty)
```

This is the listing that Anna will book



```
_id: "10109896"
listing_url: "https://www.airbnb.com/rooms/10109896"
name: "THE Place to See Sydney's FIREWORKS"
summary: "The ultimate way to experience Sydney Harbour; fireworks, the bridge, …"
space: "An entertainer's retreat!!  Views of the Sydney Harbour are unparallel…"
description: "The ultimate way to experience Sydney Harbour; fireworks, the bridge, …"
neighborhood_overview: "We're 5 minutes from the bus stops (to the city or to the Balmain East…"
notes: "We live with our stud dog, and our son; we won't be here during your s…"
transit: "Parking is available on the street in front or behind the house.  Buse…"
```

Mongo Shell:

```
db.userAndBooking.updateOne(
  { userId: 3 },
  {
    $push: {
      bookings: {
        Bookingid: 4, // Assuming the next available Bookingid is 4
        listingId: "10109896",
        bookingDate: new Date(),
        arrivalDate: new Date("2023-12-20"), // Example arrival date
        departureDate: new Date("2023-12-27"), // Example departure date
        depositPaid: 400.00, // Example deposit amount
        balanceDue: 120.00, // Example balance amount
        dueDateForBalance: new Date("2023-12-10"), // Example due date for balance
        numberOfGuests: 2,
        guests: [
          {
            name: "Eva Green",
            age: 32
          },
          {
            name: "Liam Neeson",
            age: 65
          }
        ]
      }
    }
  }
);
```

Mongo Compass:

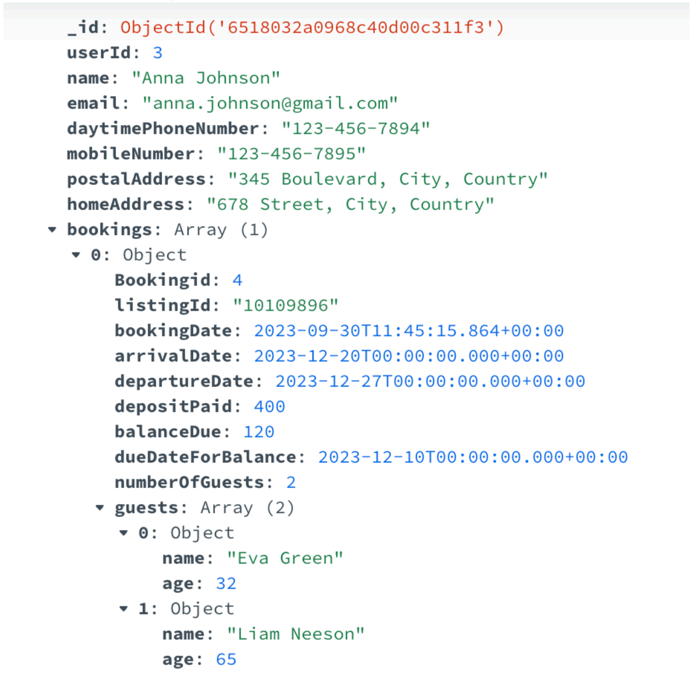
1. On Filter field: { "userId": 3 }
- Filter userID of Anna Johnson

2. Click edit and manually add the below code

```
{
 "Bookingid": 4, // Input the appropriate Bookingid.
 "listingId": "10109896",
 "bookingDate": "2023-09-30T11:45:15.864Z", // Use the current date in the appropriate format.
 "arrivalDate": "2023-12-20T00:00:00.000Z",
 "departureDate": "2023-12-27T00:00:00.000Z",
 "depositPaid": 400.00,
 "balanceDue": 120.00,
 "dueDateForBalance": "2023-12-10T00:00:00.000Z",
 "numberOfGuests": 2,
 "guests": [
  {
   "name": "Eva Green",
   "age": 32
  },
  {
   "name": "Liam Neeson",
   "age": 65
  }
 ]
}
```

3. Click update

After running command, Anna Johnson has one booking.



```
_id: ObjectId('6518032a0968c40d00c311f3')
userId: 3
name: "Anna Johnson"
email: "anna.johnson@gmail.com"
daytimePhoneNumber: "123-456-7894"
mobileNumber: "123-456-7895"
postalAddress: "345 Boulevard, City, Country"
homeAddress: "678 Street, City, Country"
bookings: Array (1)
  0: Object
    Bookingid: 4
    listingId: "10109896"
    bookingDate: 2023-09-30T11:45:15.864+00:00
    arrivalDate: 2023-12-20T00:00:00.000+00:00
    departureDate: 2023-12-27T00:00:00.000+00:00
    depositPaid: 400
    balanceDue: 120
    dueDateForBalance: 2023-12-10T00:00:00.000+00:00
    numberOfGuests: 2
    guests: Array (2)
      0: Object
        name: "Eva Green"
        age: 32
      1: Object
        name: "Liam Neeson"
        age: 65
```

2.3.b) cancel a booking, identified by the listing name and the arrival date

- Cancel the booking that we have just booked in a)

Mongo Shell:

```
var name = "THE Place to See Sydney's FIREWORKS";
var listingId = db.listingsAndReviews.findOne({name: name})._id;

db.userAndBooking.updateOne(
  {
    userId: 3,
    "bookings.listingId": listingId.toString(),
    "bookings.arrivalDate": new Date("2023-12-20")
  },
  {
    $pull: {
      bookings: {
        listingId: listingId.toString(),
        arrivalDate: new Date("2023-12-20")
      }
    }
  }
);
```

Mongo Compass:

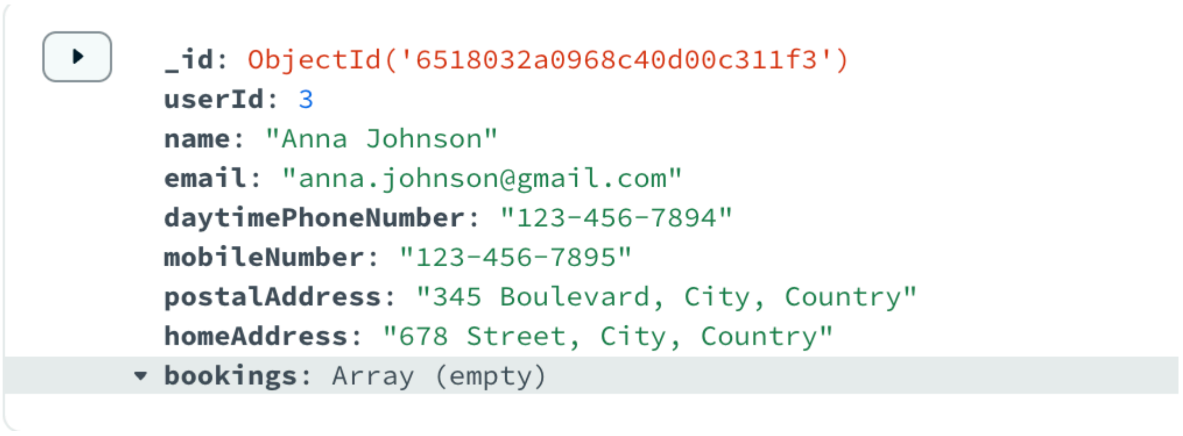
1.Open listingsAndReviews collection → On Filter field: `{ "name": "THE Place to See Sydney's FIREWORKS" }`

- locate the desired listing and derive listingid as “10109896”

2. Open usersAndBooking collection → On Filter field:
`{ "userId": 3, "bookings.listingId": "10109896"}`

3. Manually delete the booking and click update

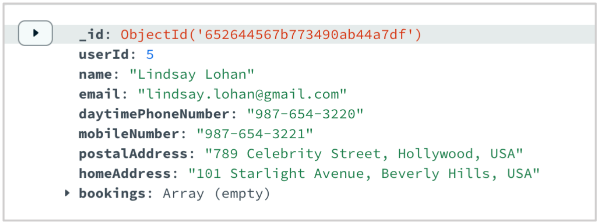
After running command, Anna Johnson’s booking has been deleted.



```
_id: ObjectId('6518032a0968c40d00c311f3')
userId: 3
name: "Anna Johnson"
email: "anna.johnson@gmail.com"
daytimePhoneNumber: "123-456-7894"
mobileNumber: "123-456-7895"
postalAddress: "345 Boulevard, City, Country"
homeAddress: "678 Street, City, Country"
bookings: Array (empty)
```

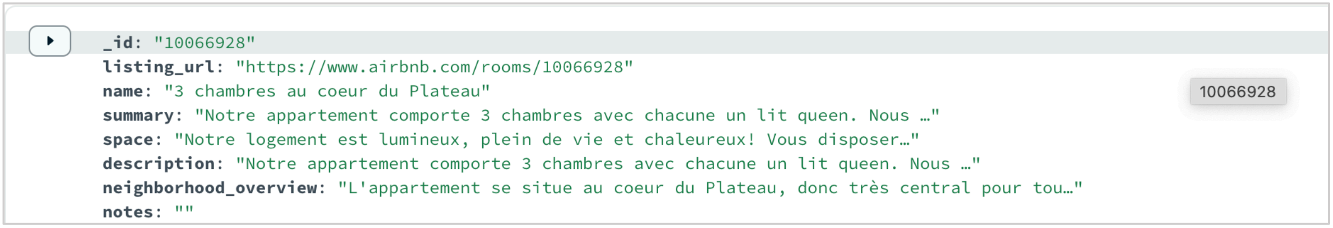
Bonus Task

Before: UserId - 5 “Lindsay” does not have any booking.



```
_id: ObjectId('652644567b773490ab44a7df')
userId: 5
name: "Lindsay Lohan"
email: "lindsay.lohan@gmail.com"
daytimePhoneNumber: "987-654-3220"
mobileNumber: "987-654-3221"
postalAddress: "789 Celebrity Street, Hollywood, USA"
homeAddress: "101 Starlight Avenue, Beverly Hills, USA"
bookings: Array (empty)
```

This is the listing that we will use in booking for the bonus task.



```
_id: "10066928"
listing_url: "https://www.airbnb.com/rooms/10066928"
name: "3 chambres au coeur du Plateau"
summary: "Notre appartement comporte 3 chambres avec chacune un lit queen. Nous …"
space: "Notre logement est lumineux, plein de vie et chaleureux! Vous disposer…"
description: "Notre appartement comporte 3 chambres avec chacune un lit queen. Nous …"
neighborhood_overview: "L'appartement se situe au coeur du Plateau, donc très central pour tou…"
notes: ""
```

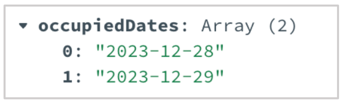
Changes made to the listingAndReviews

- **Add a Booking Calendar to the Specific Listing**

For every listing, we need a way to know which dates are booked. we should have a calendar that keeps track of occupied dates. This can be represented as an array of dates within each listing document.

```
db.listingsAndReviews.updateOne(
  { _id: "10066928" },
  {
    $addToSet: {
      occupiedDates: {
        $each: ["2023-12-28", "2023-12-29"]
      }
      }
    }
);
```

Result:



```
▾ occupiedDates: Array (2)
    0: "2023-12-28"
    1: "2023-12-29"
```

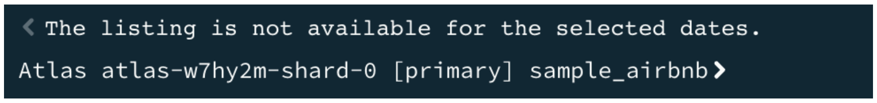
Updated 2.3.a)

2.3.a) Make a new booking for a chosen property

Case I) when user book the dates that are unavailable.

- Validation to check availability of the dates before booking.

```
var listingId = "10066928";
var listingDoc = db.listingsAndReviews.findOne({_id: listingId});
var arrivalDate = new Date("2023-12-28");
var departureDate = new Date("2023-12-30");
var datesToBook = [];

// Generate the dates to book
for (var d = arrivalDate; d < departureDate; d.setDate(d.getDate() + 1)) {
   datesToBook.push(new Date(d));
}

// Check if any of the datesToBook is in listingDoc.occupiedDates
var isAvailable = datesToBook.every(date => {
   return !listingDoc.occupiedDates.includes(date.toISOString().split("T")[0]);
});

if (isAvailable) {
   print("The listing is available for the selected dates. Proceeding to booking.");
} else {
   print("The listing is not available for the selected dates.");

}
```

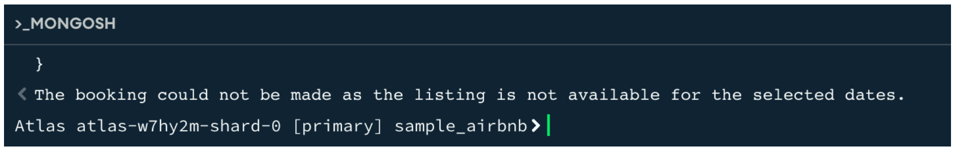


```
< The listing is not available for the selected dates.
Atlas atlas-w7hy2m-shard-0 [primary] sample_airbnb>
```

- Book the property when the date is unavailable.

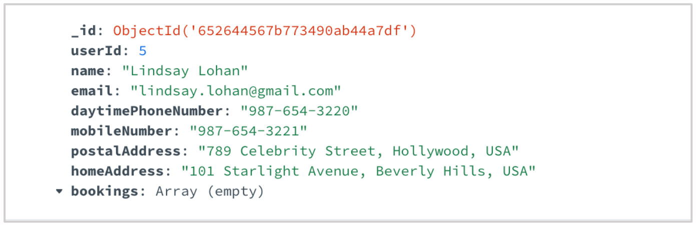
```
if (isAvailable) {
  const bookingDetails = {
    Bookingid: 5,
    listingId: listingDoc._id,
    bookingDate: new Date(),
    arrivalDate: new Date("2023-12-28"),
    departureDate: new Date("2023-12-30"),
    depositPaid: 400.00,
    balanceDue: 120.00,
    dueDateForBalance: new Date("2023-11-03"),
    numberOfGuests: 2,
    guests: [
      { name: "Eva Green", age: 32 },
      { name: "Liam Neeson", age: 65 }
    ]
  };

  // Add the booking for the user
  db.userAndBooking.updateOne(
    { userId: 5 },
    {
      $push: {
        bookings: bookingDetails
      }
    },
    { upsert: true }  // This option will insert a new document if userId 5 isn't found.
  );

  // Update the occupied dates for the listing
  db.listingsAndReviews.updateOne(
    { _id: listingId },
    {
      $addToSet: {
        occupiedDates: {
          $each: datesToBook.map(date => date.toISOString().split("T")[0])
        }
```

```
          }
        }
    );

    print("Booking has been successfully made.");
} else {
    print("The booking could not be made as the listing is not available for the selected dates.");
}
```



```
>_MONGOSH
  }
< The booking could not be made as the listing is not available for the selected dates.
Atlas atlas-w7hy2m-shard-0 [primary] sample_airbnb>
```

After running code: User 5 ‘Lindsay’ still have no booking.

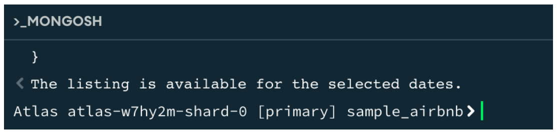


```
    _id: ObjectId('652644567b773490ab44a7df')
    userId: 5
    name: "Lindsay Lohan"
    email: "lindsay.lohan@gmail.com"
    daytimePhoneNumber: "987-654-3220"
    mobileNumber: "987-654-3221"
    postalAddress: "789 Celebrity Street, Hollywood, USA"
    homeAddress: "101 Starlight Avenue, Beverly Hills, USA"
  ▾ bookings: Array (empty)
```

Case II) when user book the dates that are available.

- Validation to check availability of the dates before booking.

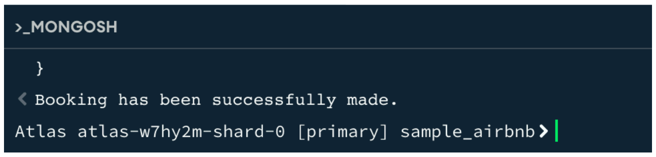
```
var listingId = "10066928";
var listingDoc = db.listingsAndReviews.findOne({_id: listingId});
var arrivalDate = new Date("2023-11-01");
var departureDate = new Date("2023-11-03");
var datesToBook = [];

// Generate the dates to book
for (var d = arrivalDate; d < departureDate; d.setDate(d.getDate() + 1)) {
    datesToBook.push(new Date(d));
}

// Check if any of the datesToBook is in listingDoc.occupiedDates
var isAvailable = datesToBook.every(date => {
    return !listingDoc.occupiedDates.includes(date.toISOString().split("T")[0]);
});

if (isAvailable) {
    print("The listing is available for the selected dates.");
} else {
   print("The listing is not available for the selected dates.");
}
```



```
>_MONGOSH
   }
< The listing is available for the selected dates.
Atlas atlas-w7hy2m-shard-0 [primary] sample_airbnb>
```

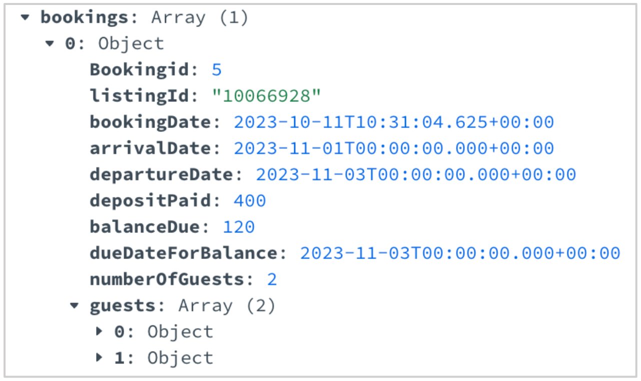
- Book the property when the date is available.

```
if (isAvailable) {
   // Define the booking details
   const bookingDetails = {
      Bookingid: 5,
      listingId: listingDoc._id,
      bookingDate: new Date(),
      arrivalDate: new Date("2023-11-01"),
      departureDate: new Date("2023-11-03"),
      depositPaid: 400.00,
      balanceDue: 120.00,
      dueDateForBalance: new Date("2023-11-03"),
      numberOfGuests: 2,
      guests: [
         { name: "Eva Green", age: 32 },
         { name: "Liam Neeson", age: 65 }
      ]
   };

   // Add the booking for the user
   db.userAndBooking.updateOne(
      { userId: 5 },
      {
         $push: {
            bookings: bookingDetails
         }
      },
      { upsert: true }  // This option will insert a new document if userId 5 isn't found.
   );

   // Update the occupied dates for the listing
   db.listingsAndReviews.updateOne(
      { _id: listingId },
      {
          $addToSet: {
            occupiedDates: {
              $each: datesToBook.map(date => date.toISOString().split("T")[0])
            }
          }
        }
    );

    print("Booking has been successfully made.");
} else {
    print("The booking could not be made as the listing is not available for the selected dates.");
}
```



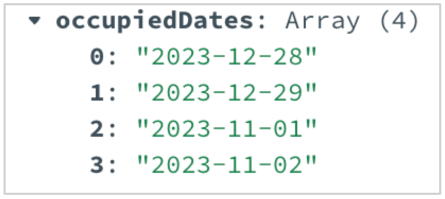
```
>_MONGOSH
  }
< Booking has been successfully made.
Atlas atlas-w7hy2m-shard-0 [primary] sample_airbnb>
```

After running code: booking has been made successfully for User 5.



```
▾ bookings: Array (1)
  ▾ 0: Object
      Bookingid: 5
      listingId: "10066928"
      bookingDate: 2023-10-11T10:31:04.625+00:00
      arrivalDate: 2023-11-01T00:00:00.000+00:00
      departureDate: 2023-11-03T00:00:00.000+00:00
      depositPaid: 400
      balanceDue: 120
      dueDateForBalance: 2023-11-03T00:00:00.000+00:00
      numberOfGuests: 2
    ▾ guests: Array (2)
      ▸ 0: Object
      ▸ 1: Object
```

Date 2023-10-01 and 2023-10-02 has been added to the Occupied Dates array in Listing property with ID "10066928" as below.



```
▾ occupiedDates: Array (4)
    0: "2023-12-28"
    1: "2023-12-29"
    2: "2023-11-01"
    3: "2023-11-02"
```