Course: Advanced Databases (NoSQL)

Assignment: Endterm (project)

Student's name: Tulentay Ramazan

Dataset used: Airbnb.bson

Group: BDA-2407

Student ID: 240463

Instructor: Dinara Zhunissova

Date of submission: 2.01.2026, 2:58 PM

Deadline of submission: 2.02.2026 , 3:00 PM

Team size: 1 student (Solo)

Content

Part I: CRUD operations across multiple collections

Part II: Embedded and referenced data models

Part III: Advanced update/delete operators

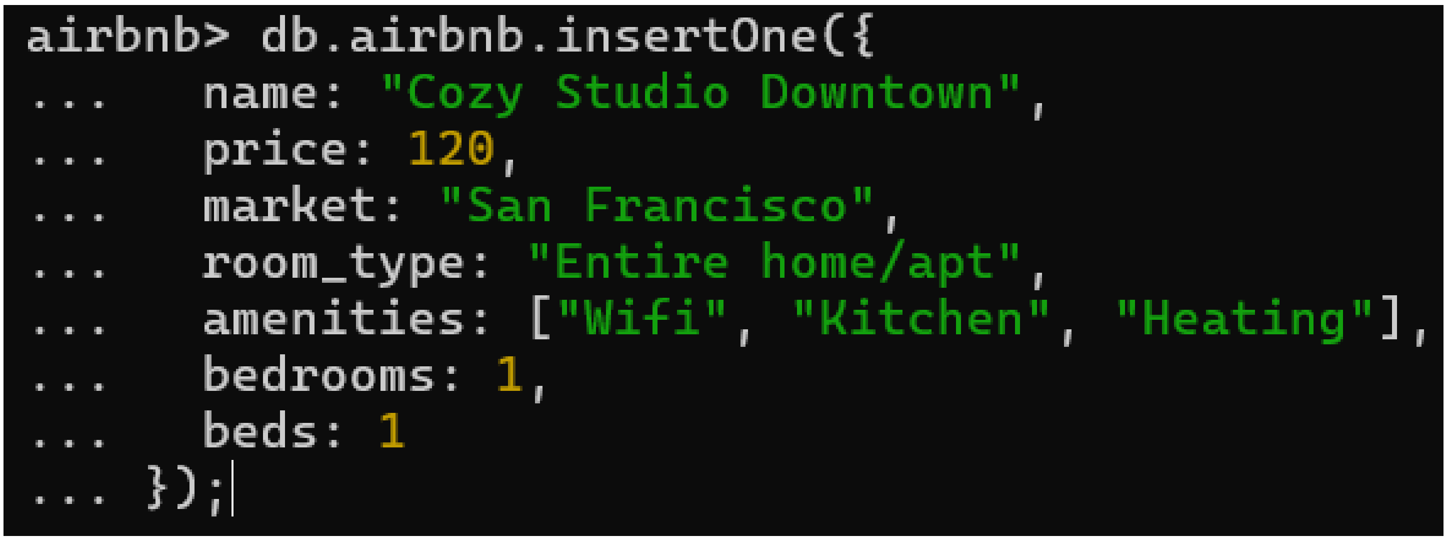
Part IV: Multi-stage aggreagtion pipelines

Part V: Compound indexes and query optimization

PartVI: Authentification and authorization

Part I: CRUD operations across multiple collections

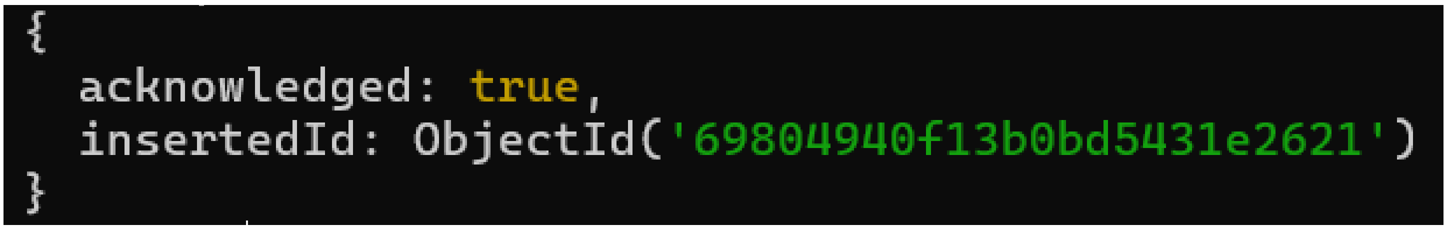
Create command:



```
airbnb> db.airbnb.insertOne({
...    name: "Cozy Studio Downtown",
...    price: 120,
...    market: "San Francisco",
...    room_type: "Entire home/apt",
...    amenities: ["Wifi", "Kitchen", "Heating"],
...    bedrooms: 1,
...    beds: 1
... });
```

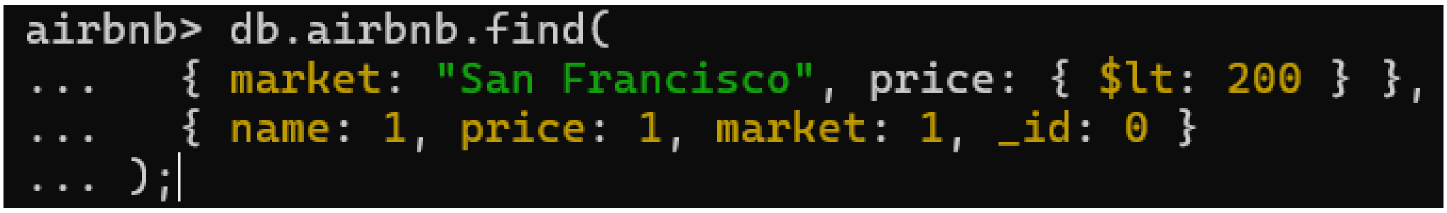
For example, I'd like to add a name: Cozy Studio Downtown, with the market San Francisco to airbnb with the price 120, amenities: Wifi, Kitchen, Heating.

Output:



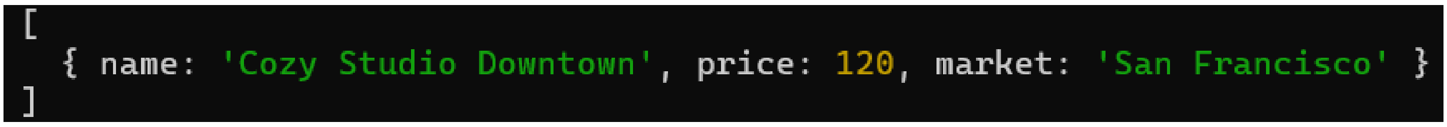
```
{
  acknowledged: true,
  insertedId: ObjectId('69804940f13b0bd5431e2621')
}
```

Read command:



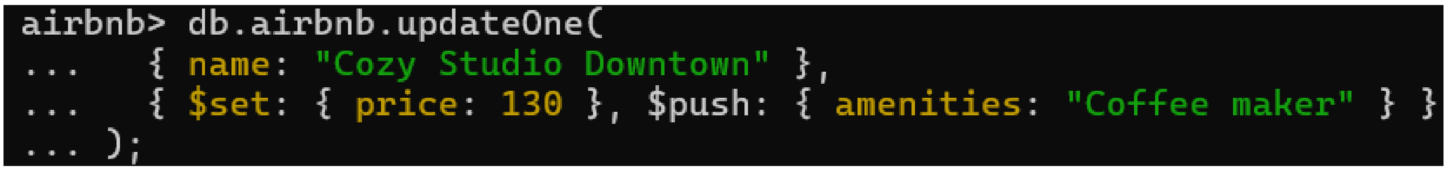
```
airbnb> db.airbnb.find(
...   { market: "San Francisco", price: { $lt: 200 } },
...   { name: 1, price: 1, market: 1, _id: 0 }
... );
```

Output:



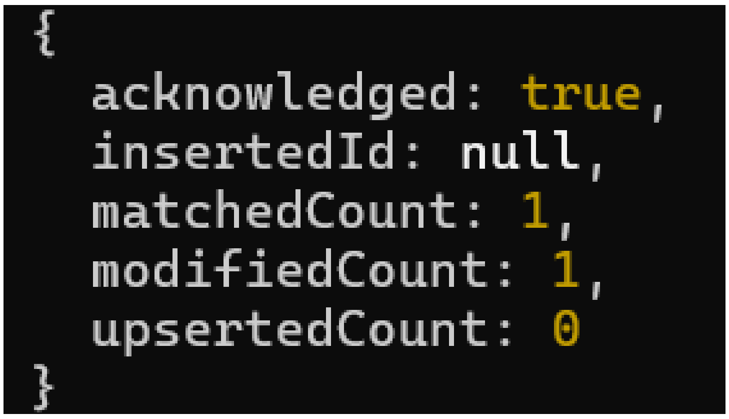
```
[
  { name: 'Cozy Studio Downtown', price: 120, market: 'San Francisco' }
]
```

Update command:



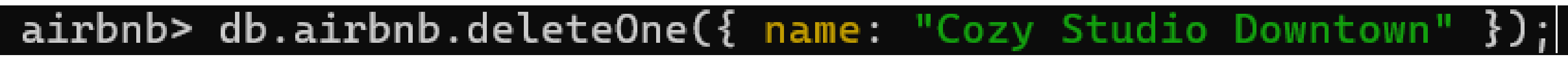
```
airbnb> db.airbnb.updateOne(
...   { name: "Cozy Studio Downtown" },
...   { $set: { price: 130 }, $push: { amenities: "Coffee maker" } }
... );
```

Output:



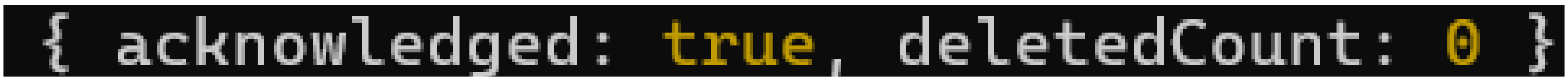
```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Delete command:



```
airbnb> db.airbnb.deleteOne({ name: "Cozy Studio Downtown" });
```

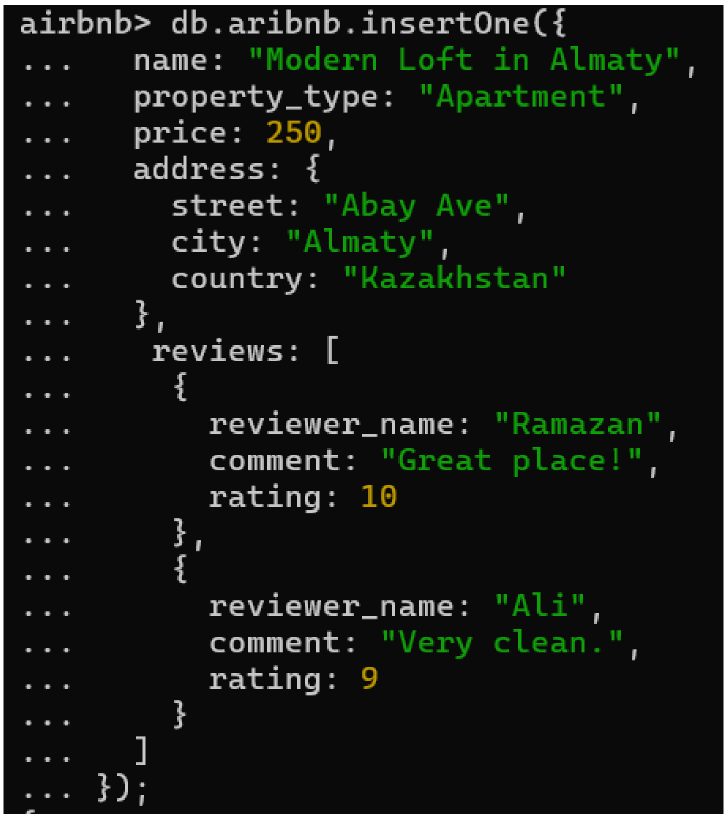
Output:



```
{ acknowledged: true, deletedCount: 0 }
```

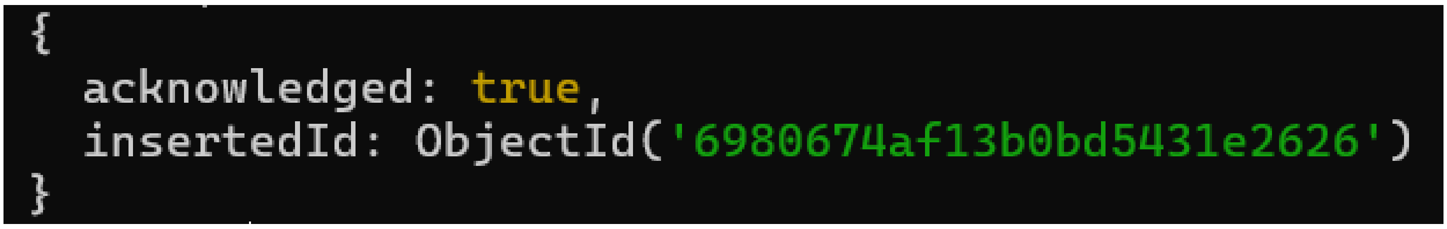
Part II: Embedded and referenced data models

Embedded Data Model command:



```
airbnb> db.aribnb.insertOne({
...    name: "Modern Loft in Almaty",
...    property_type: "Apartment",
...    price: 250,
...    address: {
...      street: "Abay Ave",
...      city: "Almaty",
...      country: "Kazakhstan"
...    },
...     reviews: [
...      {
...        reviewer_name: "Ramazan",
...        comment: "Great place!",
...        rating: 10
...      },
...      {
...        reviewer_name: "Ali",
...        comment: "Very clean.",
...        rating: 9
...      }
...    ]
... });
```

Output:

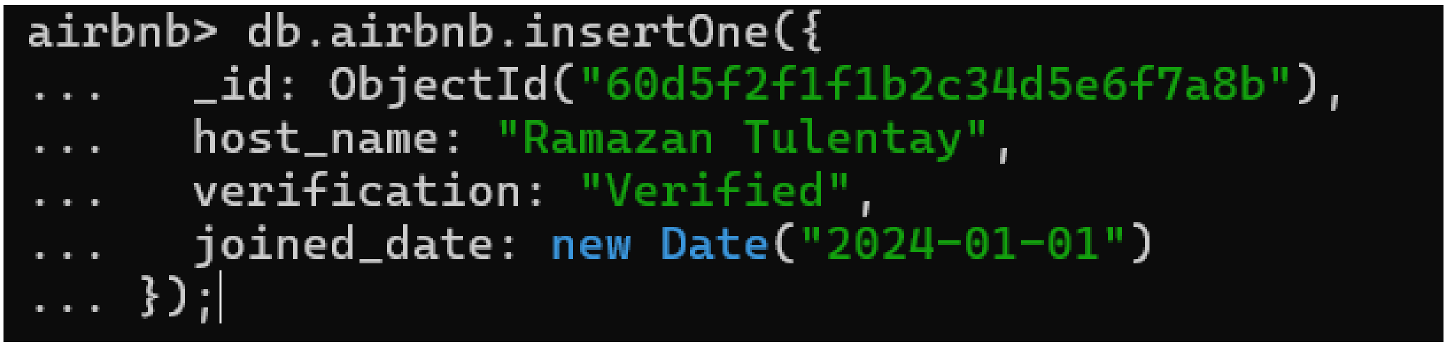


```
{
  acknowledged: true,
  insertedId: ObjectId('6980674af13b0bd5431e2626')
}
```

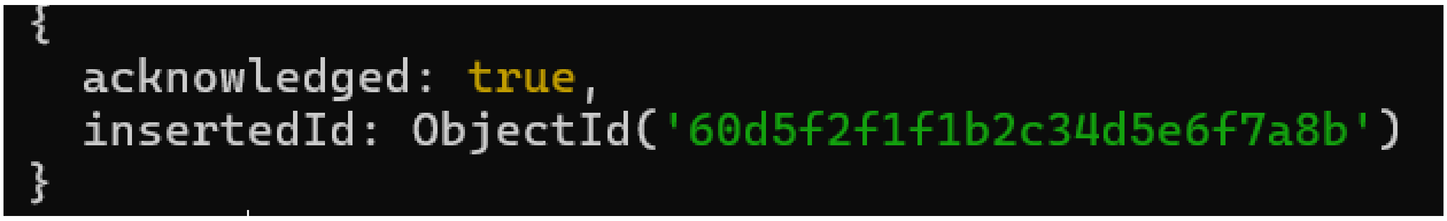
In Airbnb, reviews are typically embedded directly into the listing document. This allows us to find the apartment description and recent reviews in a single query, which is ideal for NoSQL.

Referenced Data Model command:

First of all, we create a host in the hosts collection:

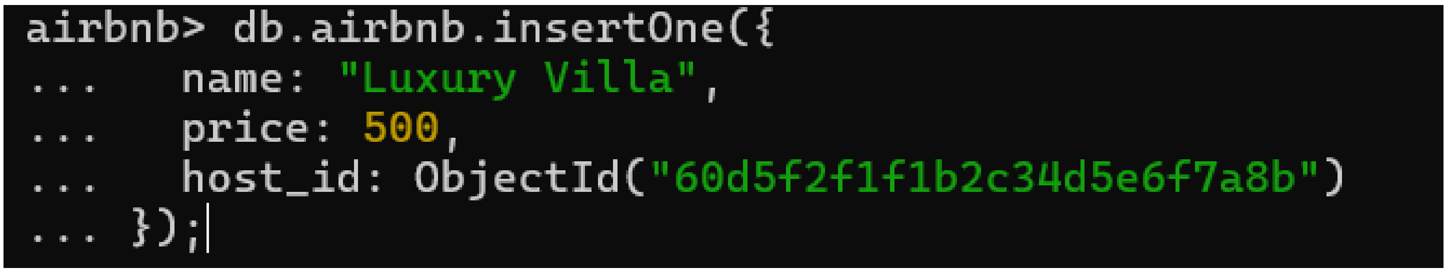


```
airbnb> db.airbnb.insertOne({
...   _id: ObjectId("60d5f2f1f1b2c34d5e6f7a8b"),
...   host_name: "Ramazan Tulentay",
...   verification: "Verified",
...   joined_date: new Date("2024-01-01")
... });
```

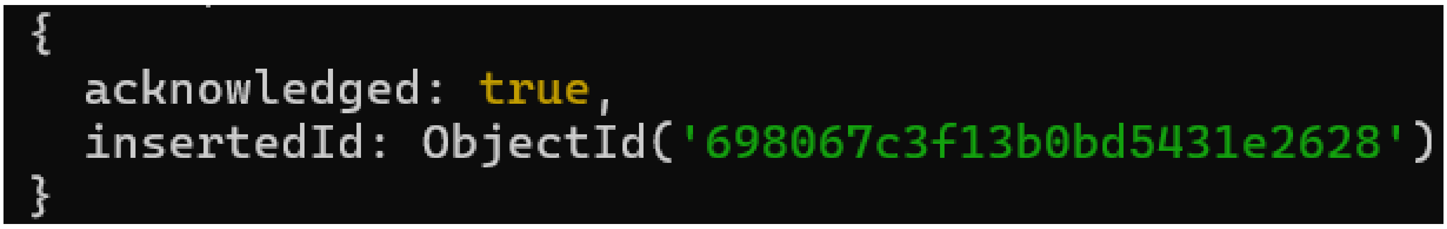


```
{
  acknowledged: true,
  insertedId: ObjectId('60d5f2f1f1b2c34d5e6f7a8b')
}
```

Next, we create a home that references this owner:



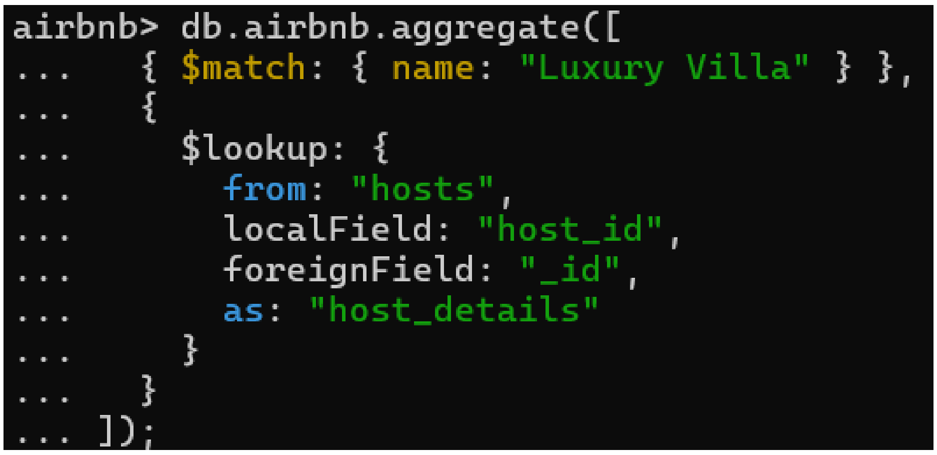
```
airbnb> db.airbnb.insertOne({
...   name: "Luxury Villa",
...   price: 500,
...   host_id: ObjectId("60d5f2f1f1b2c34d5e6f7a8b")
... });
```



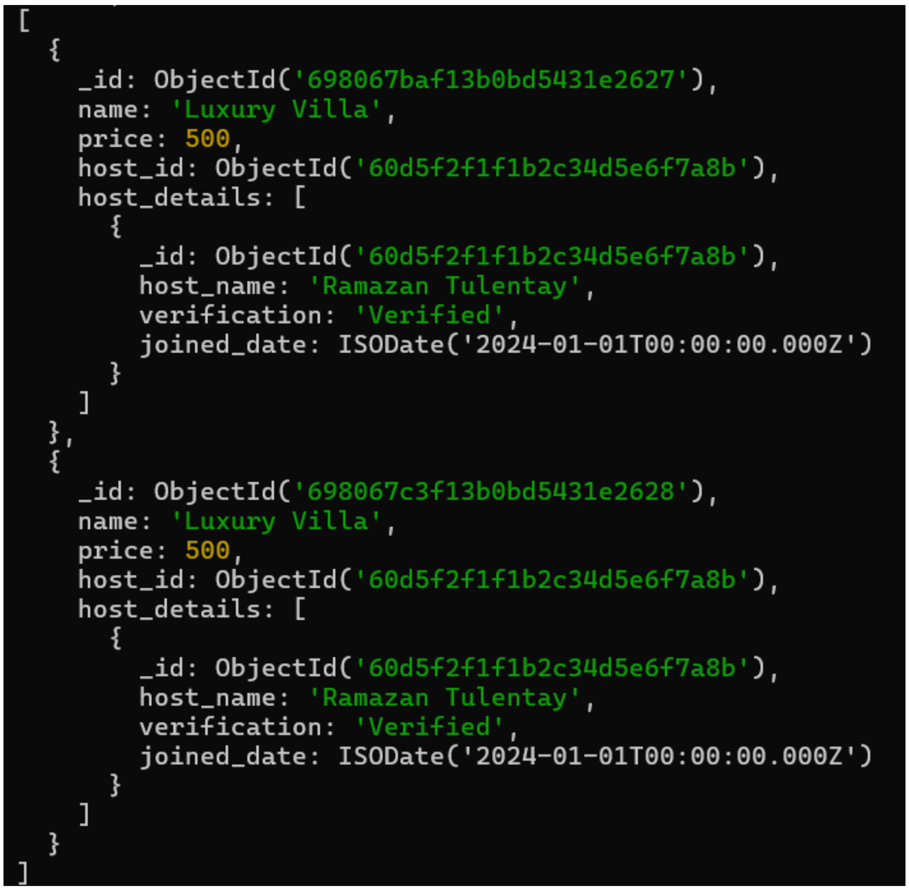
```
{
  acknowledged: true,
  insertedId: ObjectId('698067c3f13b0bd5431e2628')
}
```

How to retrieve data from related collections (Similar to JOIN):

To show that links work during defense, use the **$lookup** stage. This will demonstrate your mastery of advanced modeling.



```
airbnb> db.airbnb.aggregate([
...   { $match: { name: "Luxury Villa" } },
...   {
...     $lookup: {
...       from: "hosts",
...       localField: "host_id",
...       foreignField: "_id",
...       as: "host_details"
...     }
...   }
... ]);
```

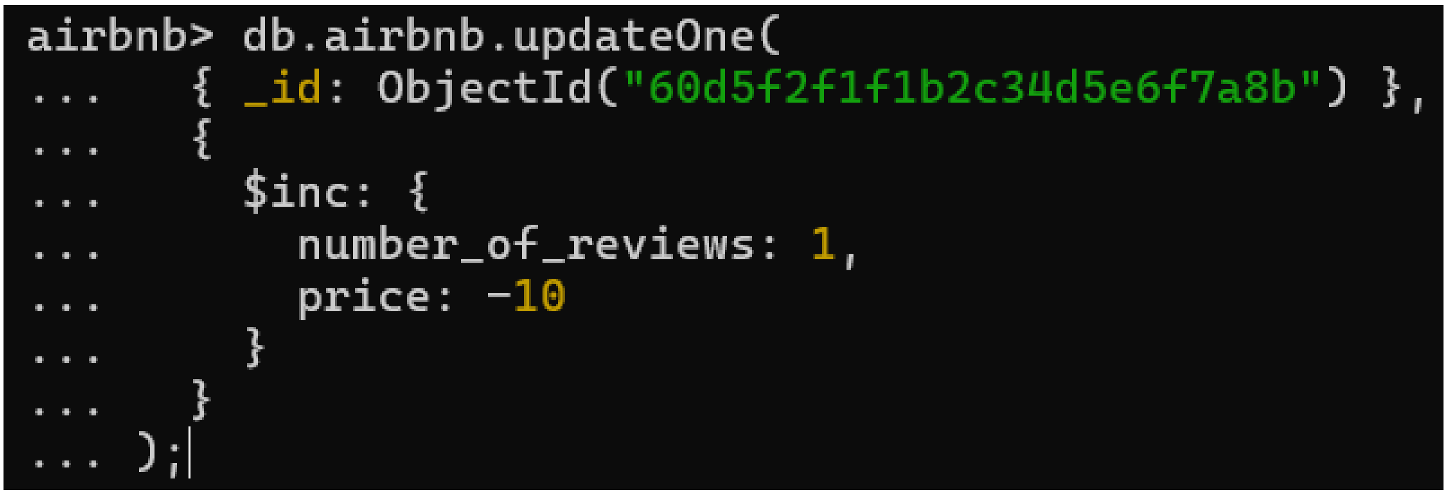


```
[
  {
    _id: ObjectId('698067baf13b0bd5431e2627'),
    name: 'Luxury Villa',
    price: 500,
    host_id: ObjectId('60d5f2f1f1b2c34d5e6f7a8b'),
    host_details: [
      {
        _id: ObjectId('60d5f2f1f1b2c34d5e6f7a8b'),
        host_name: 'Ramazan Tulentay',
        verification: 'Verified',
        joined_date: ISODate('2024-01-01T00:00:00.000Z')
      }
    ]
  },
  {
    _id: ObjectId('698067c3f13b0bd5431e2628'),
    name: 'Luxury Villa',
    price: 500,
    host_id: ObjectId('60d5f2f1f1b2c34d5e6f7a8b'),
    host_details: [
      {
        _id: ObjectId('60d5f2f1f1b2c34d5e6f7a8b'),
        host_name: 'Ramazan Tulentay',
        verification: 'Verified',
        joined_date: ISODate('2024-01-01T00:00:00.000Z')
      }
    ]
  }
]
```

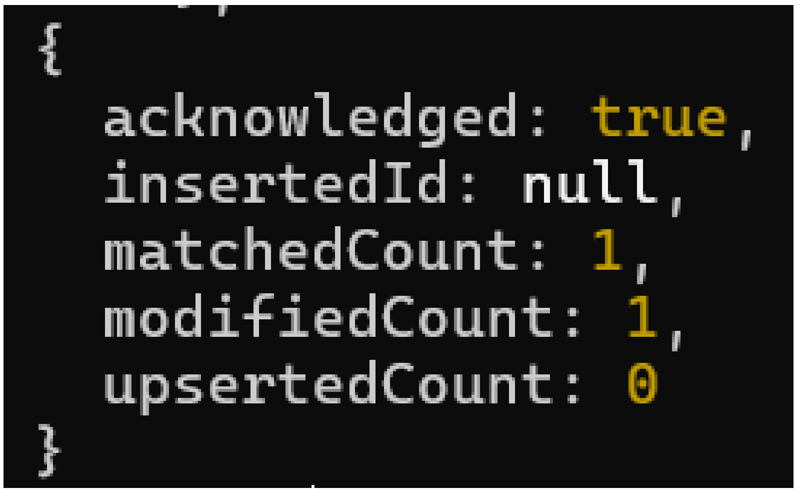
Part III: Advanced update/delete operators

$inc operator (the smart changing the number)

Task: Increase the number of reviews by 1 and decrease the price by 10.



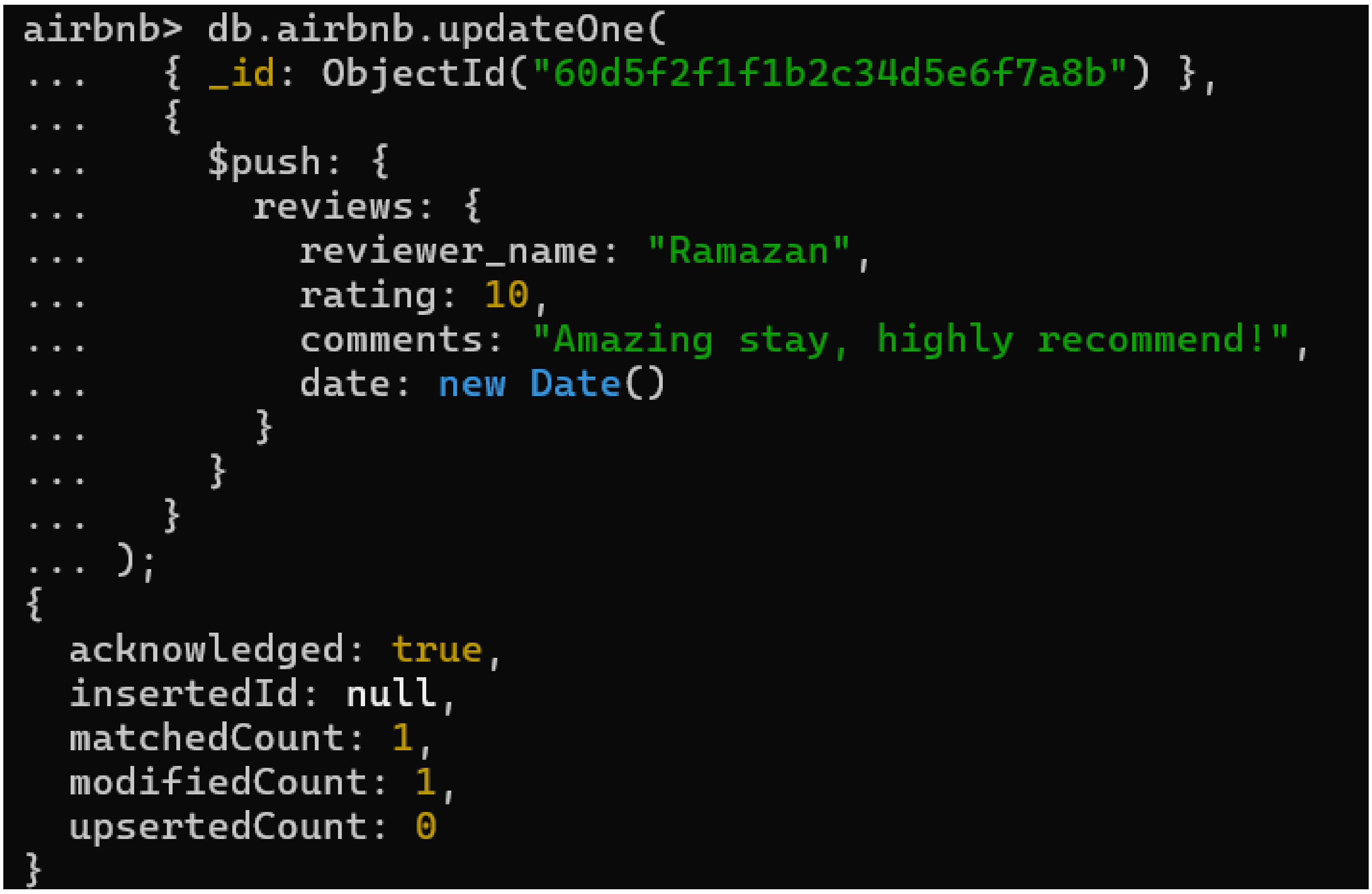
```
airbnb> db.airbnb.updateOne(
...     { _id: ObjectId("60d5f2f1f1b2c34d5e6f7a8b") },
...     {
...       $inc: {
...         number_of_reviews: 1,
...         price: -10
...       }
...     }
... );
```



```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

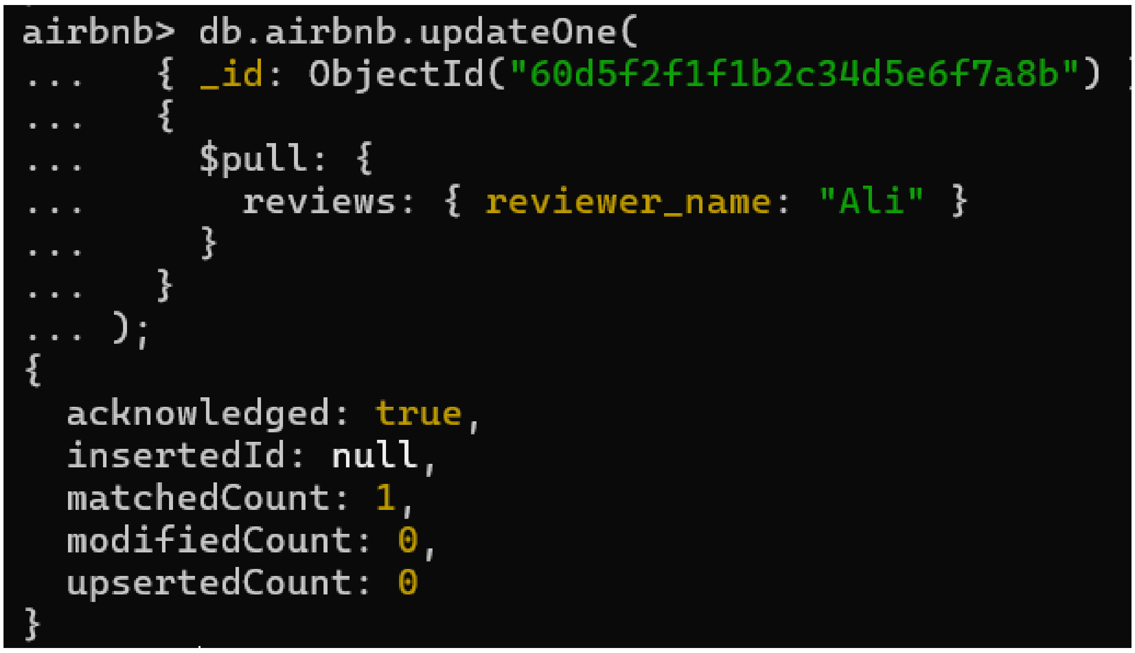
$push Operator (Adding to an Array)

Task: Add a new review to the reviews array.


```
airbnb> db.airbnb.updateOne(
...   { _id: ObjectId("60d5f2f1f1b2c34d5e6f7a8b") },
...   {
...     $push: {
...       reviews: {
...         reviewer_name: "Ramazan",
...         rating: 10,
...         comments: "Amazing stay, highly recommend!",
...         date: new Date()
...       }
...     }
...   }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

$pull (removing from array):

Task: Delete all reviews left by user "Ali".



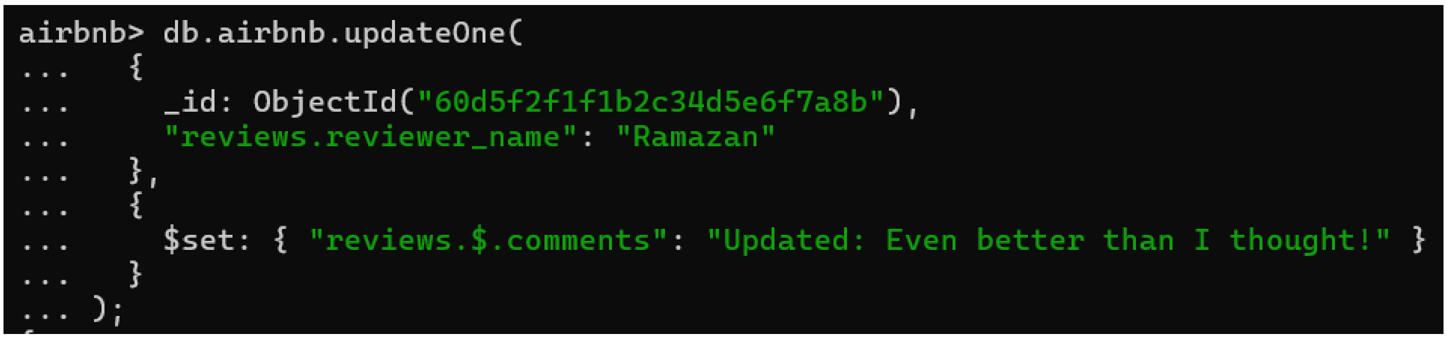
```
airbnb> db.airbnb.updateOne(
...   { _id: ObjectId("60d5f2f1f1b2c34d5e6f7a8b")
...   {
...     $pull: {
...       reviews: { reviewer_name: "Ali" }
...     }
...   }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
```

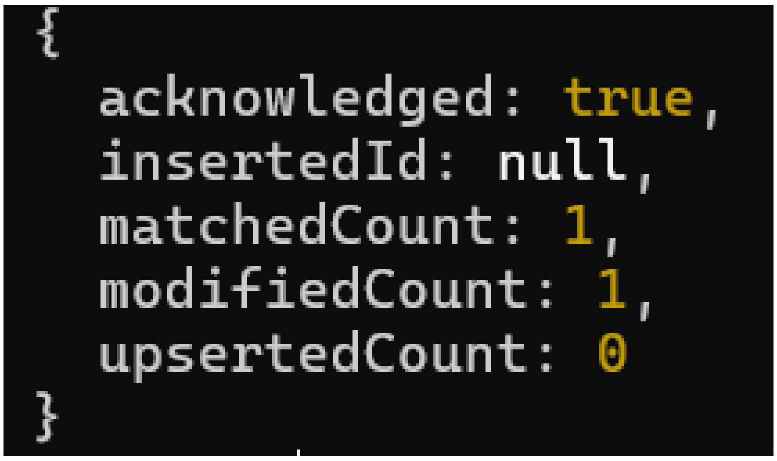
The quantity is matched, but not modified, because this query only removed all correspondent reviews and nothing is modified.

Positional Operator $ (Update a Specific Element)

This is the most advanced part of the task. It allows you to update a specific element in the array that matches the filter condition.

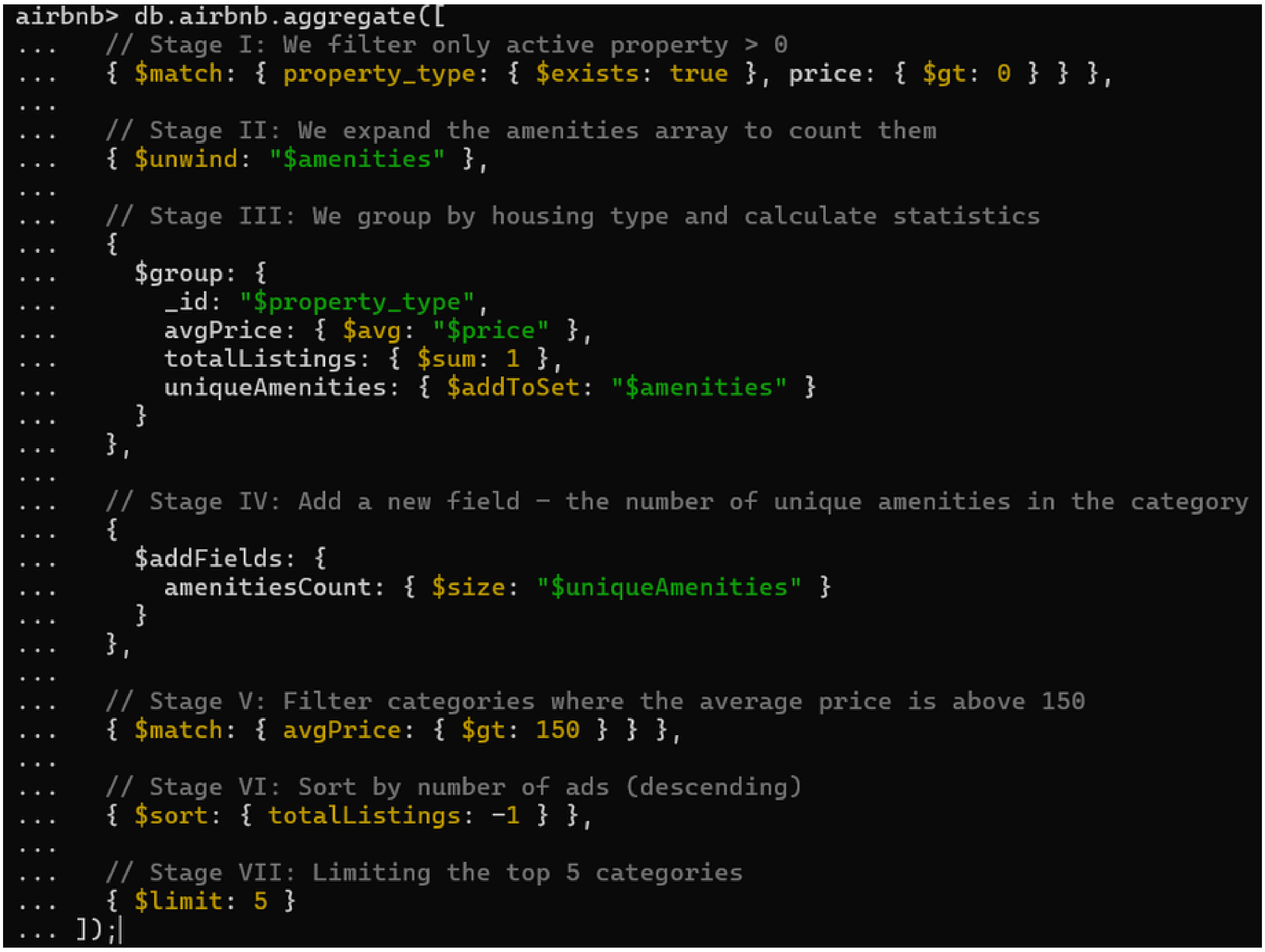
Task: Find a review from "Ramazan" and update only the comment text.


```
airbnb> db.airbnb.updateOne(
...   {
...     _id: ObjectId("60d5f2f1f1b2c34d5e6f7a8b"),
...     "reviews.reviewer_name": "Ramazan"
...   },
...   {
...     $set: { "reviews.$.comments": "Updated: Even better than I thought!" }
...   }
... );
```

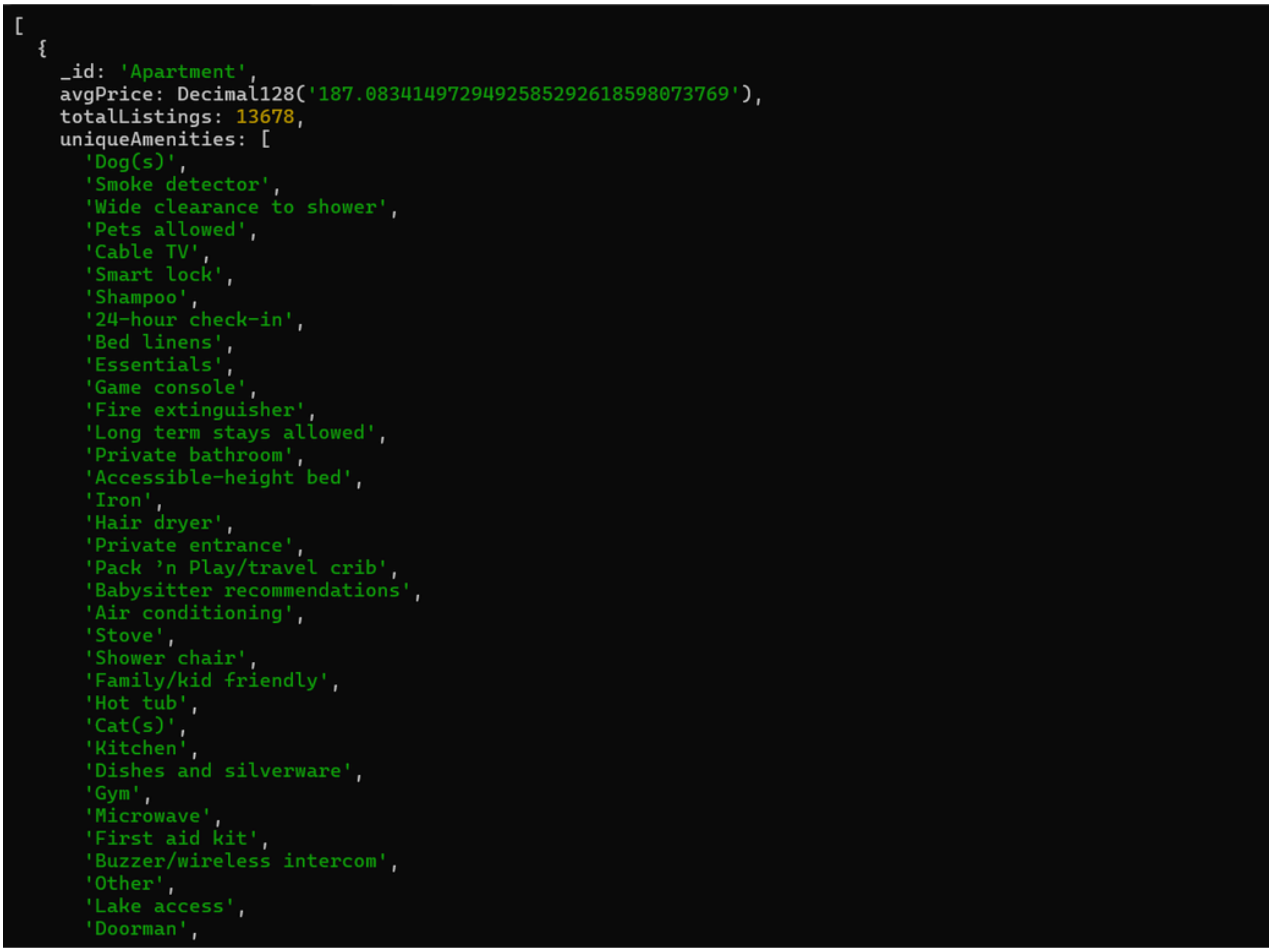

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Part IV: Multi-stage aggreagtion pipelines

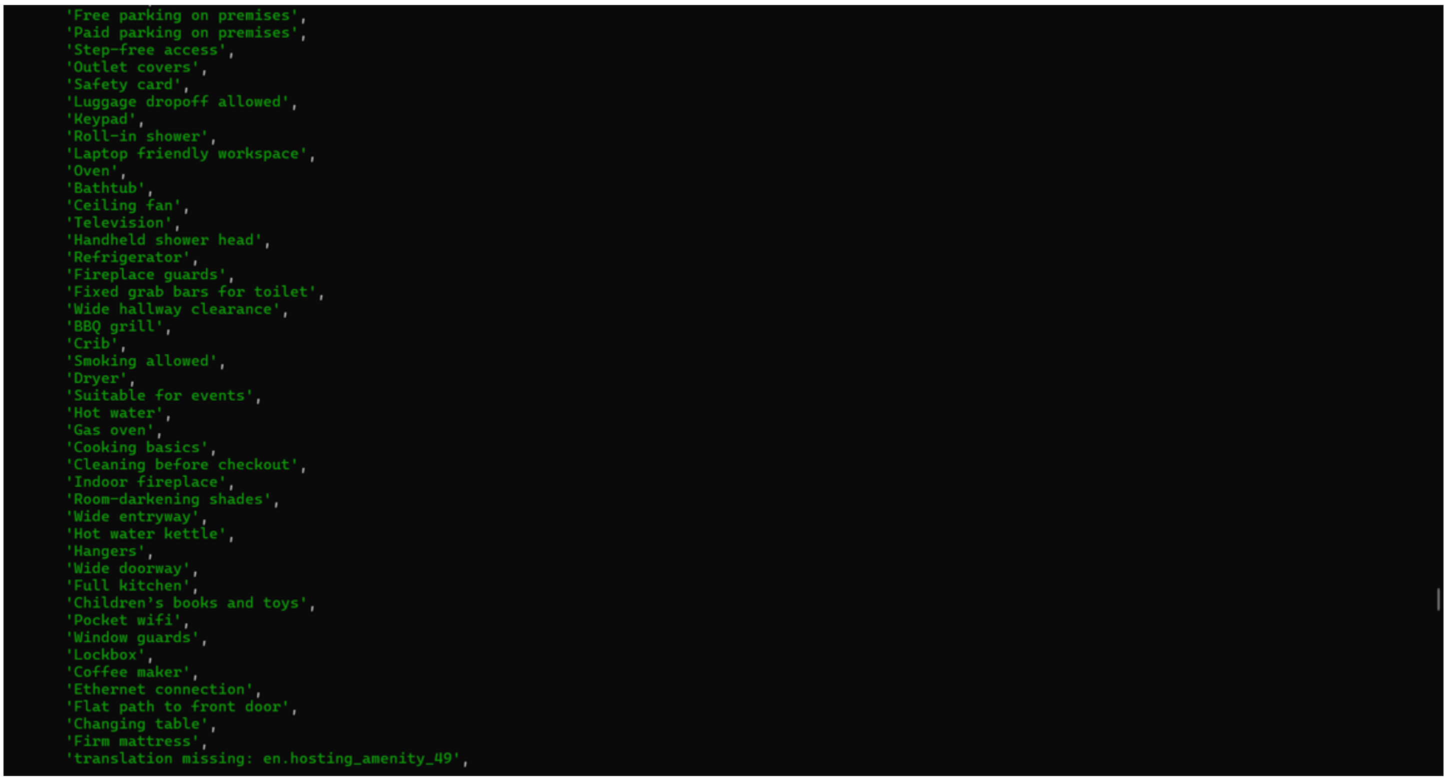
This pipeline does the following: filters the data, unwraps the arrays, calculates averages, filters the grouping result and sorts it.



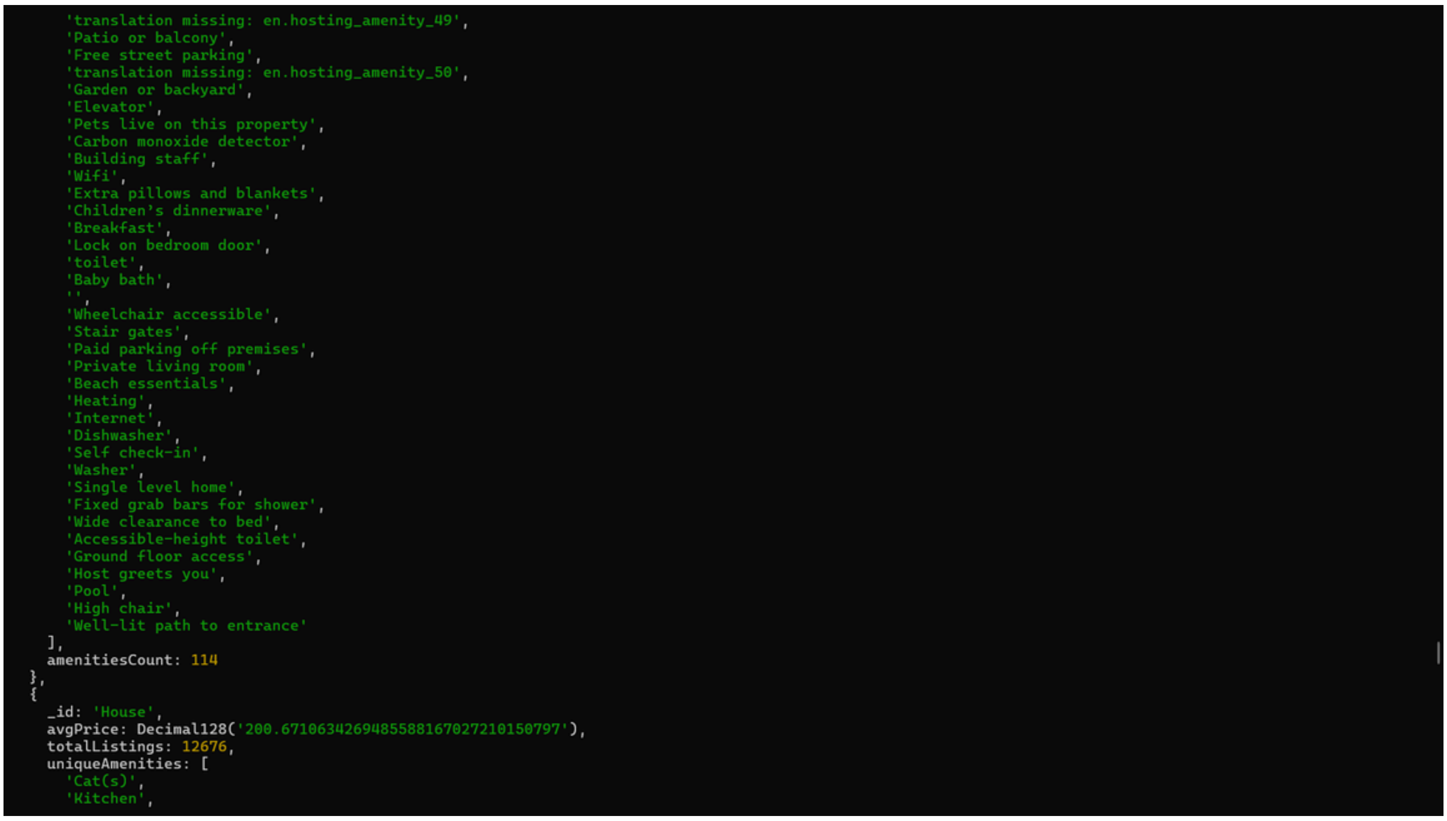
```
airbnb> db.airbnb.aggregate([
...   // Stage I: We filter only active property > 0
...   { $match: { property_type: { $exists: true }, price: { $gt: 0 } } },
...
...   // Stage II: We expand the amenities array to count them
...   { $unwind: "$amenities" },
...
...   // Stage III: We group by housing type and calculate statistics
...   {
...     $group: {
...       _id: "$property_type",
...       avgPrice: { $avg: "$price" },
...       totalListings: { $sum: 1 },
...       uniqueAmenities: { $addToSet: "$amenities" }
...     }
...   },
...
...   // Stage IV: Add a new field – the number of unique amenities in the category
...   {
...     $addFields: {
...       amenitiesCount: { $size: "$uniqueAmenities" }
...     }
...   },
...
...   // Stage V: Filter categories where the average price is above 150
...   { $match: { avgPrice: { $gt: 150 } } },
...
...   // Stage VI: Sort by number of ads (descending)
...   { $sort: { totalListings: -1 } },
...
...   // Stage VII: Limiting the top 5 categories
...   { $limit: 5 }
... ]);
```



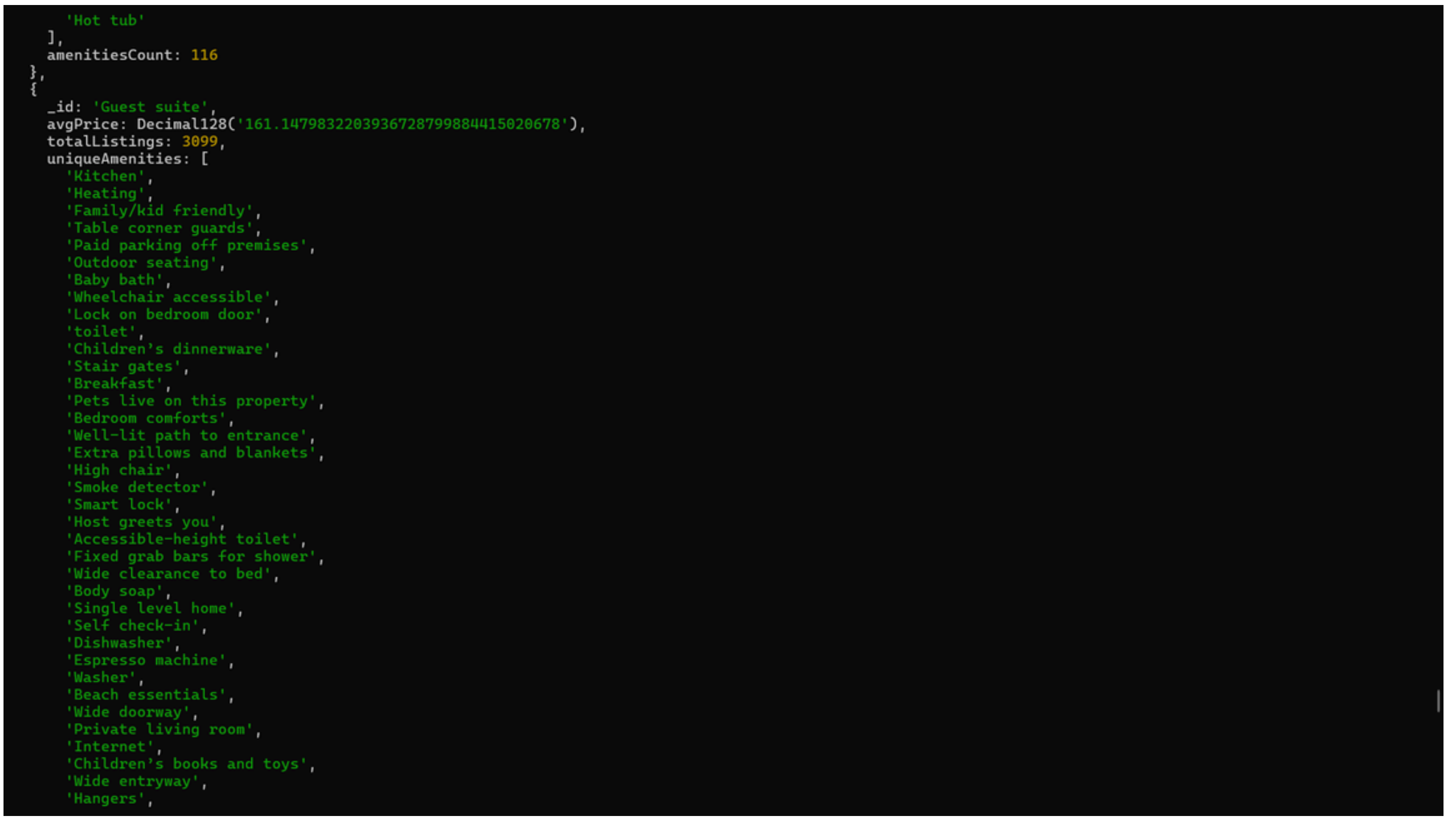
```
[
  {
    _id: 'Apartment',
    avgPrice: Decimal128('187.083414972949258529261859807376 9'),
    totalListings: 13678,
    uniqueAmenities: [
      'Dog(s)',
      'Smoke detector',
      'Wide clearance to shower',
      'Pets allowed',
      'Cable TV',
      'Smart lock',
      'Shampoo',
      '24-hour check-in',
      'Bed linens',
      'Essentials',
      'Game console',
      'Fire extinguisher',
      'Long term stays allowed',
      'Private bathroom',
      'Accessible-height bed',
      'Iron',
      'Hair dryer',
      'Private entrance',
      'Pack 'n Play/travel crib',
      'Babysitter recommendations',
      'Air conditioning',
      'Stove',
      'Shower chair',
      'Family/kid friendly',
      'Hot tub',
      'Cat(s)',
      'Kitchen',
      'Dishes and silverware',
      'Gym',
      'Microwave',
      'First aid kit',
      'Buzzer/wireless intercom',
      'Other',
      'Lake access',
      'Doorman',
```



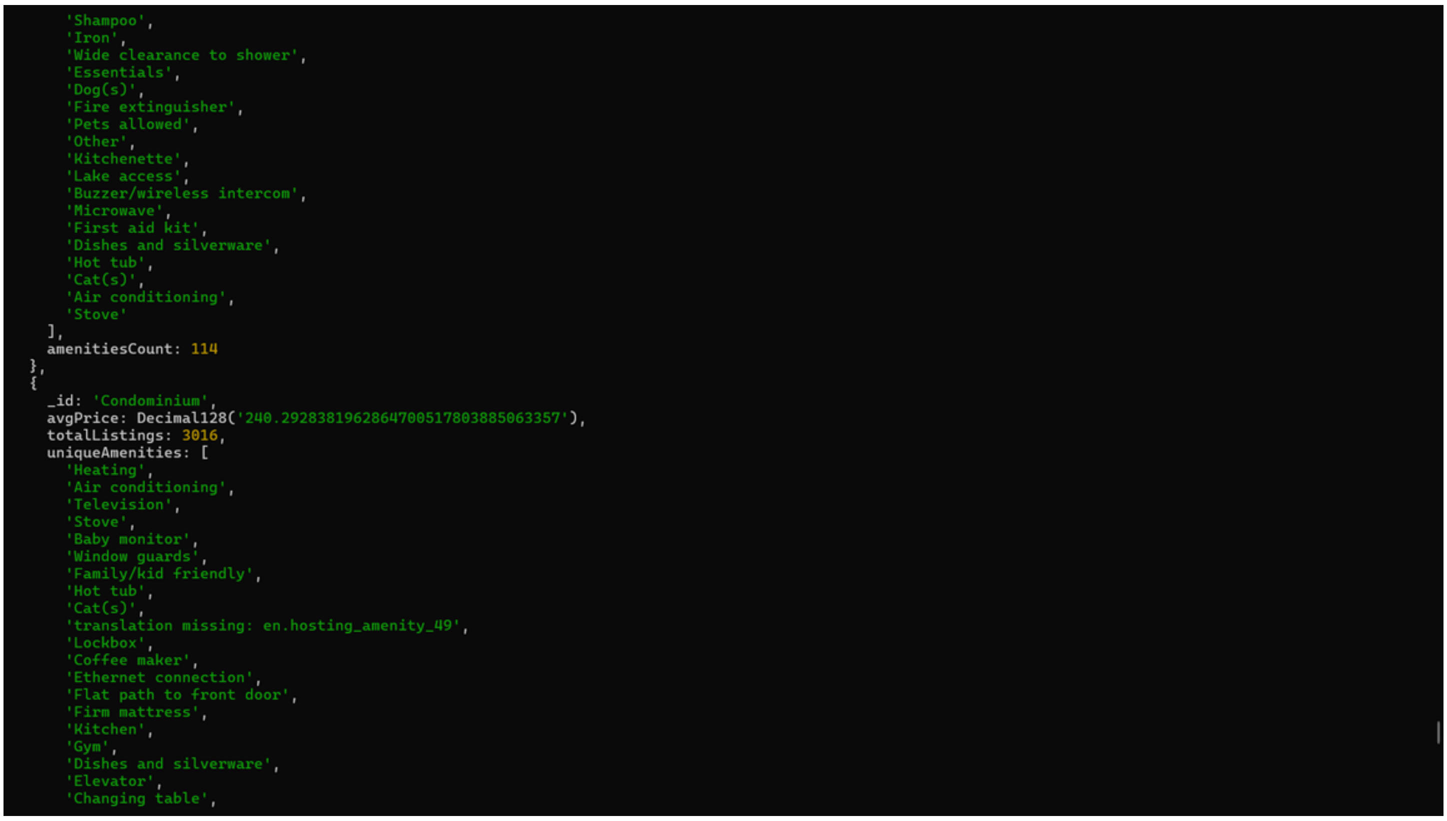
```
      'Free parking on premises',
      'Paid parking on premises',
      'Step-free access',
      'Outlet covers',
      'Safety card',
      'Luggage dropoff allowed',
      'Keypad',
      'Roll-in shower',
      'Laptop friendly workspace',
      'Oven',
      'Bathtub',
      'Ceiling fan',
      'Television',
      'Handheld shower head',
      'Refrigerator',
      'Fireplace guards',
      'Fixed grab bars for toilet',
      'Wide hallway clearance',
      'BBQ grill',
      'Crib',
      'Smoking allowed',
      'Dryer',
      'Suitable for events',
      'Hot water',
      'Gas oven',
      'Cooking basics',
      'Cleaning before checkout',
      'Indoor fireplace',
      'Room-darkening shades',
      'Wide entryway',
      'Hot water kettle',
      'Hangers',
      'Wide doorway',
      'Full kitchen',
      'Children's books and toys',
      'Pocket wifi',
      'Window guards',
      'Lockbox',
      'Coffee maker',
      'Ethernet connection',
      'Flat path to front door',
      'Changing table',
      'Firm mattress',
      'translation missing: en.hosting_amenity_49',
```



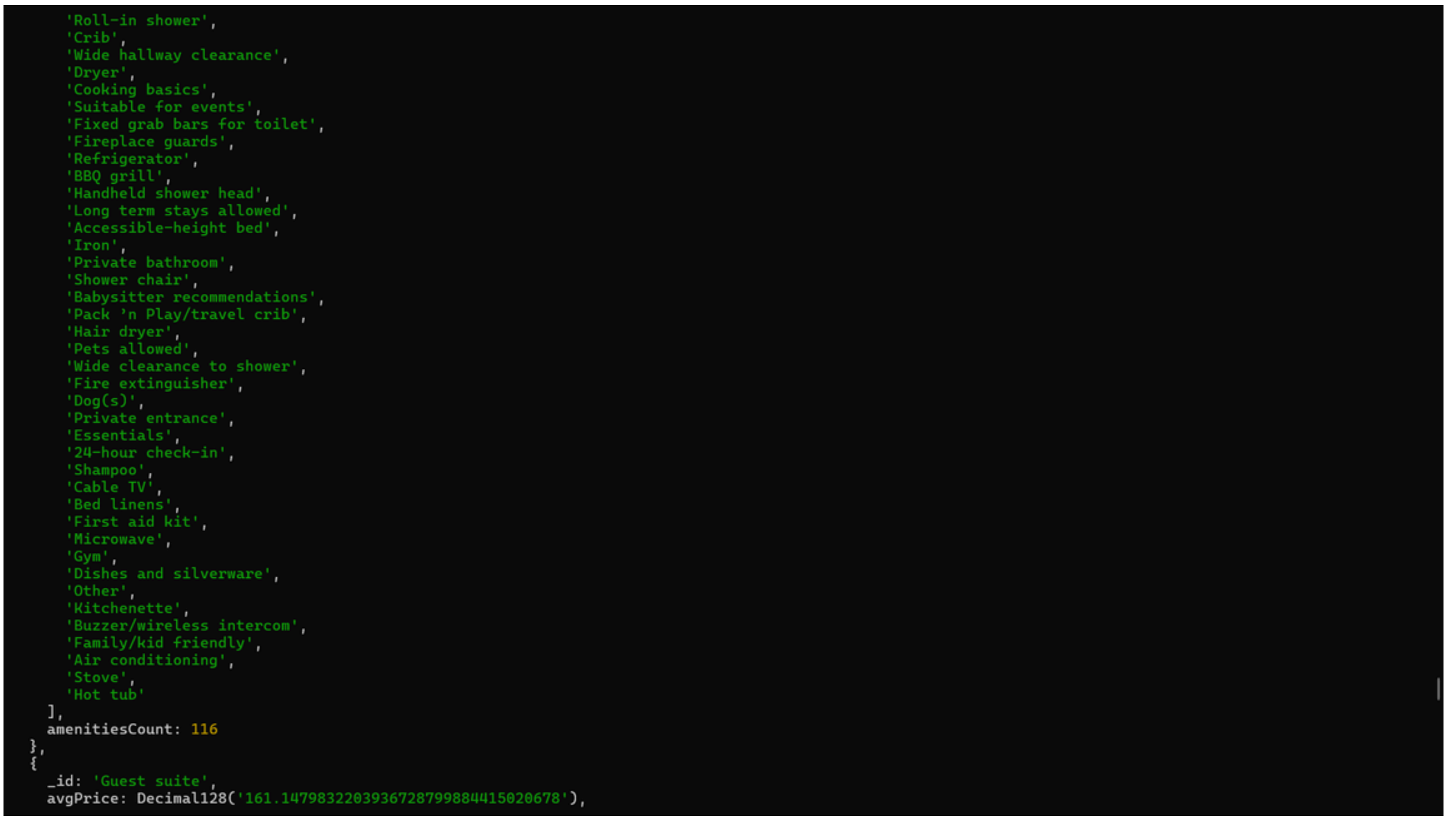
```
      'translation missing: en.hosting_amenity_49',
      'Patio or balcony',
      'Free street parking',
      'translation missing: en.hosting_amenity_50',
      'Garden or backyard',
      'Elevator',
      'Pets live on this property',
      'Carbon monoxide detector',
      'Building staff',
      'Wifi',
      'Extra pillows and blankets',
      'Children's dinnerware',
      'Breakfast',
      'Lock on bedroom door',
      'toilet',
      'Baby bath',
      '',
      'Wheelchair accessible',
      'Stair gates',
      'Paid parking off premises',
      'Private living room',
      'Beach essentials',
      'Heating',
      'Internet',
      'Dishwasher',
      'Self check-in',
      'Washer',
      'Single level home',
      'Fixed grab bars for shower',
      'Wide clearance to bed',
      'Accessible-height toilet',
      'Ground floor access',
      'Host greets you',
      'Pool',
      'High chair',
      'Well-lit path to entrance'
    ],
    amenitiesCount: 114
  },
  {
    _id: 'House',
    avgPrice: Decimal128('200.6710634269485588167027210150797'),
    totalListings: 12676,
    uniqueAmenities: [
      'Cat(s)',
      'Kitchen',
```



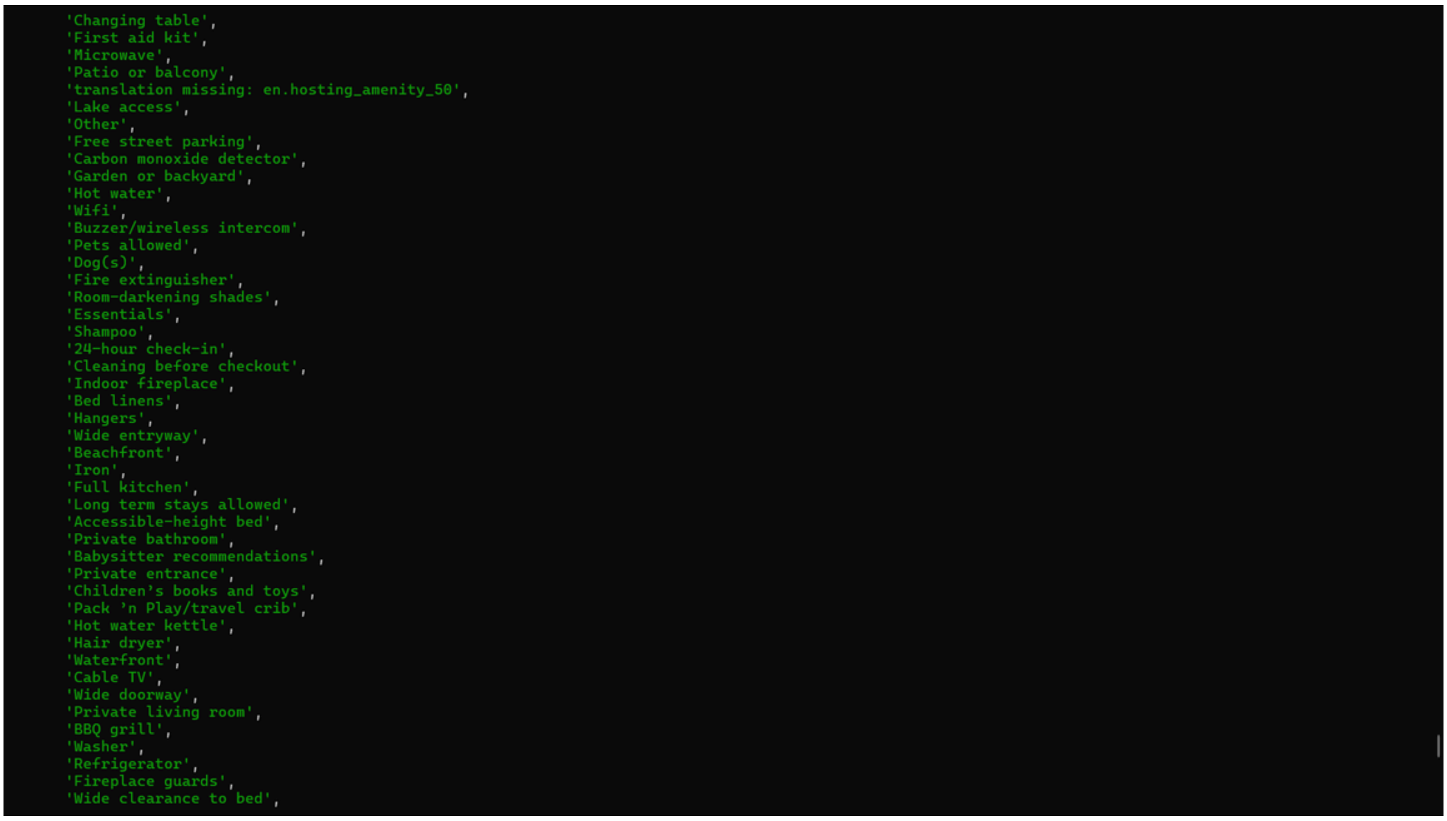
```
    'Hot tub'
  ],
  amenitiesCount: 116
},
{
  _id: 'Guest suite',
  avgPrice: Decimal128('161.1479832203936728799884415020678'),
  totalListings: 3099,
  uniqueAmenities: [
    'Kitchen',
    'Heating',
    'Family/kid friendly',
    'Table corner guards',
    'Paid parking off premises',
    'Outdoor seating',
    'Baby bath',
    'Wheelchair accessible',
    'Lock on bedroom door',
    'toilet',
    'Children's dinnerware',
    'Stair gates',
    'Breakfast',
    'Pets live on this property',
    'Bedroom comforts',
    'Well-lit path to entrance',
    'Extra pillows and blankets',
    'High chair',
    'Smoke detector',
    'Smart lock',
    'Host greets you',
    'Accessible-height toilet',
    'Fixed grab bars for shower',
    'Wide clearance to bed',
    'Body soap',
    'Single level home',
    'Self check-in',
    'Dishwasher',
    'Espresso machine',
    'Washer',
    'Beach essentials',
    'Wide doorway',
    'Private living room',
    'Internet',
    'Children's books and toys',
    'Wide entryway',
    'Hangers',
```



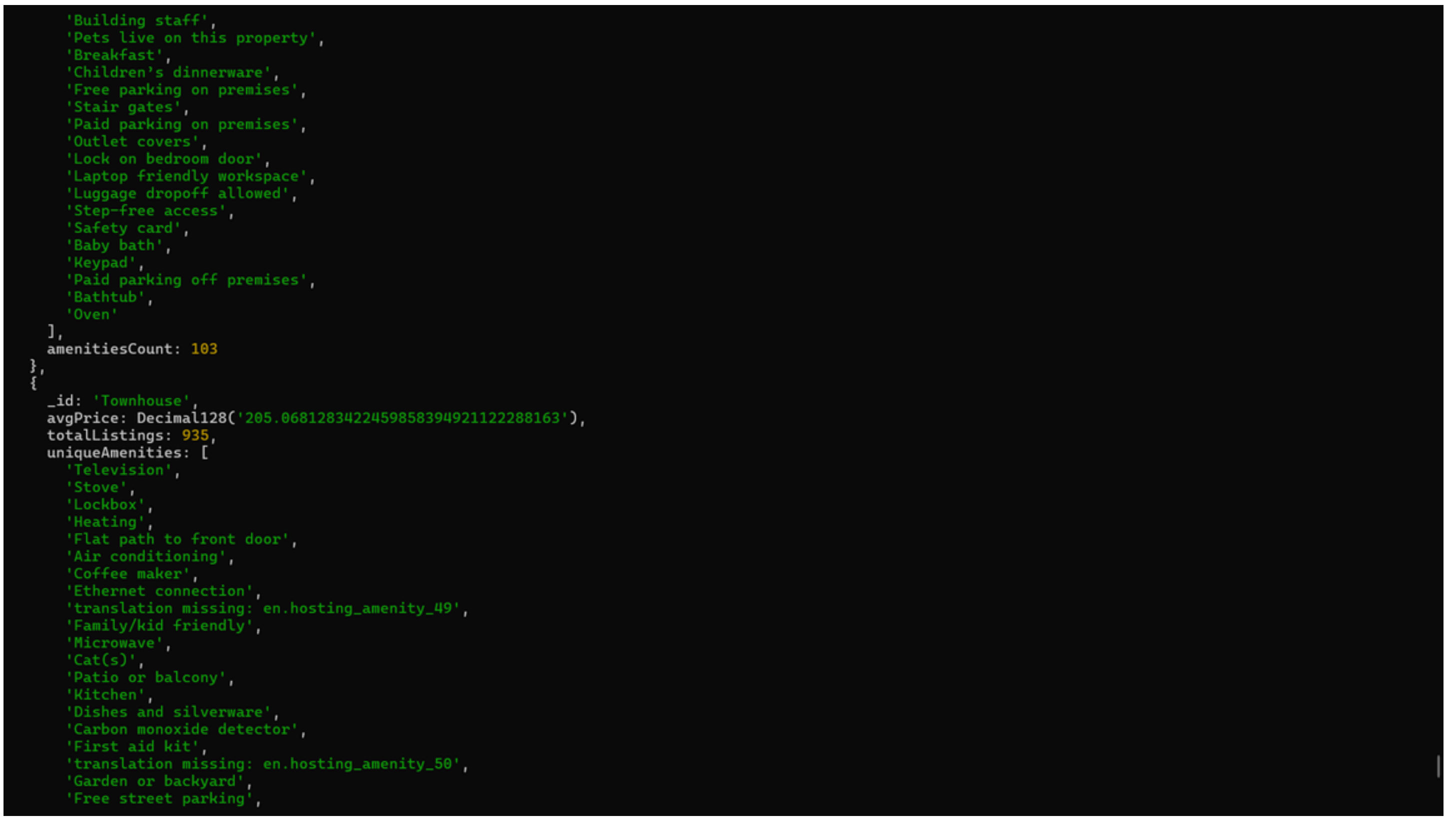
```
    'Shampoo',
    'Iron',
    'Wide clearance to shower',
    'Essentials',
    'Dog(s)',
    'Fire extinguisher',
    'Pets allowed',
    'Other',
    'Kitchenette',
    'Lake access',
    'Buzzer/wireless intercom',
    'Microwave',
    'First aid kit',
    'Dishes and silverware',
    'Hot tub',
    'Cat(s)',
    'Air conditioning',
    'Stove'
  ],
  amenitiesCount: 114
},
{
  _id: 'Condominium',
  avgPrice: Decimal128('240.2928381962864700517803885063357'),
  totalListings: 3016,
  uniqueAmenities: [
    'Heating',
    'Air conditioning',
    'Television',
    'Stove',
    'Baby monitor',
    'Window guards',
    'Family/kid friendly',
    'Hot tub',
    'Cat(s)',
    'translation missing: en.hosting_amenity_49',
    'Lockbox',
    'Coffee maker',
    'Ethernet connection',
    'Flat path to front door',
    'Firm mattress',
    'Kitchen',
    'Gym',
    'Dishes and silverware',
    'Elevator',
    'Changing table',
```



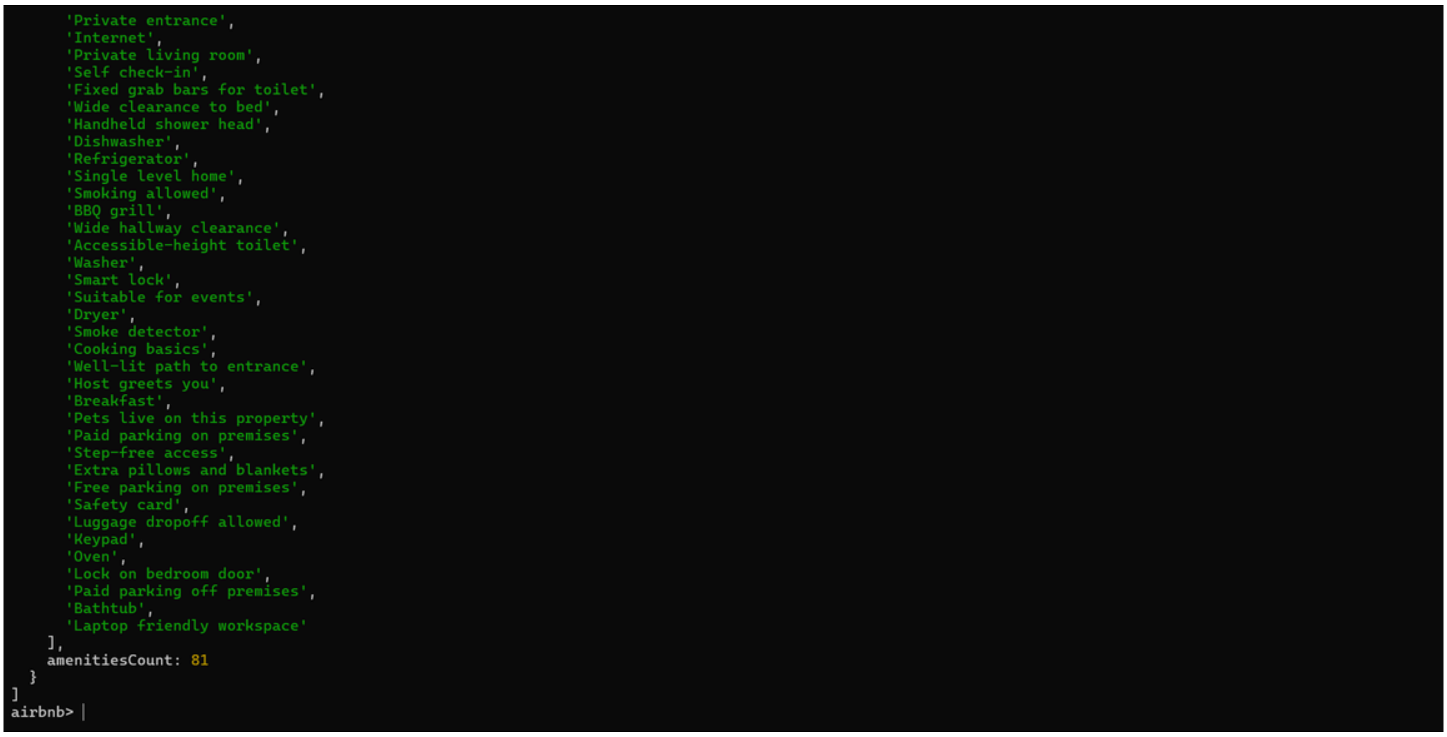
```
      'Roll-in shower',
      'Crib',
      'Wide hallway clearance',
      'Dryer',
      'Cooking basics',
      'Suitable for events',
      'Fixed grab bars for toilet',
      'Fireplace guards',
      'Refrigerator',
      'BBQ grill',
      'Handheld shower head',
      'Long term stays allowed',
      'Accessible-height bed',
      'Iron',
      'Private bathroom',
      'Shower chair',
      'Babysitter recommendations',
      'Pack 'n Play/travel crib',
      'Hair dryer',
      'Pets allowed',
      'Wide clearance to shower',
      'Fire extinguisher',
      'Dog(s)',
      'Private entrance',
      'Essentials',
      '24-hour check-in',
      'Shampoo',
      'Cable TV',
      'Bed linens',
      'First aid kit',
      'Microwave',
      'Gym',
      'Dishes and silverware',
      'Other',
      'Kitchenette',
      'Buzzer/wireless intercom',
      'Family/kid friendly',
      'Air conditioning',
      'Stove',
      'Hot tub'
    ],
    amenitiesCount: 116
  },
  {
    _id: 'Guest suite',
    avgPrice: Decimal128('161.147983220393672879988441502067 8'),
```



```
      'Changing table',
      'First aid kit',
      'Microwave',
      'Patio or balcony',
      'translation missing: en.hosting_amenity_50',
      'Lake access',
      'Other',
      'Free street parking',
      'Carbon monoxide detector',
      'Garden or backyard',
      'Hot water',
      'Wifi',
      'Buzzer/wireless intercom',
      'Pets allowed',
      'Dog(s)',
      'Fire extinguisher',
      'Room-darkening shades',
      'Essentials',
      'Shampoo',
      '24-hour check-in',
      'Cleaning before checkout',
      'Indoor fireplace',
      'Bed linens',
      'Hangers',
      'Wide entryway',
      'Beachfront',
      'Iron',
      'Full kitchen',
      'Long term stays allowed',
      'Accessible-height bed',
      'Private bathroom',
      'Babysitter recommendations',
      'Private entrance',
      'Children's books and toys',
      'Pack 'n Play/travel crib',
      'Hot water kettle',
      'Hair dryer',
      'Waterfront',
      'Cable TV',
      'Wide doorway',
      'Private living room',
      'BBQ grill',
      'Washer',
      'Refrigerator',
      'Fireplace guards',
      'Wide clearance to bed',
```



```
      'Building staff',
      'Pets live on this property',
      'Breakfast',
      'Children's dinnerware',
      'Free parking on premises',
      'Stair gates',
      'Paid parking on premises',
      'Outlet covers',
      'Lock on bedroom door',
      'Laptop friendly workspace',
      'Luggage dropoff allowed',
      'Step-free access',
      'Safety card',
      'Baby bath',
      'Keypad',
      'Paid parking off premises',
      'Bathtub',
      'Oven'
    ],
    amenitiesCount: 103
  },
  {
    _id: 'Townhouse',
    avgPrice: Decimal128('205.0681283422459858394921122288163'),
    totalListings: 935,
    uniqueAmenities: [
      'Television',
      'Stove',
      'Lockbox',
      'Heating',
      'Flat path to front door',
      'Air conditioning',
      'Coffee maker',
      'Ethernet connection',
      'translation missing: en.hosting_amenity_49',
      'Family/kid friendly',
      'Microwave',
      'Cat(s)',
      'Patio or balcony',
      'Kitchen',
      'Dishes and silverware',
      'Carbon monoxide detector',
      'First aid kit',
      'translation missing: en.hosting_amenity_50',
      'Garden or backyard',
      'Free street parking',
```

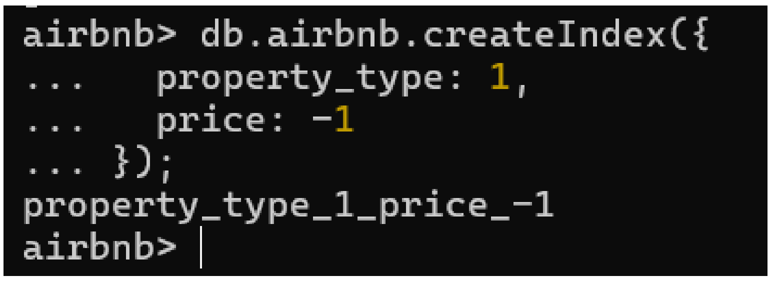


```
      'Private entrance',
      'Internet',
      'Private living room',
      'Self check-in',
      'Fixed grab bars for toilet',
      'Wide clearance to bed',
      'Handheld shower head',
      'Dishwasher',
      'Refrigerator',
      'Single level home',
      'Smoking allowed',
      'BBQ grill',
      'Wide hallway clearance',
      'Accessible-height toilet',
      'Washer',
      'Smart lock',
      'Suitable for events',
      'Dryer',
      'Smoke detector',
      'Cooking basics',
      'Well-lit path to entrance',
      'Host greets you',
      'Breakfast',
      'Pets live on this property',
      'Paid parking on premises',
      'Step-free access',
      'Extra pillows and blankets',
      'Free parking on premises',
      'Safety card',
      'Luggage dropoff allowed',
      'Keypad',
      'Oven',
      'Lock on bedroom door',
      'Paid parking off premises',
      'Bathtub',
      'Laptop friendly workspace'
    ],
    amenitiesCount: 81
  }
]
airbnb> |
```

Part V: Compound indexes and query optimization

Creating a Compound Index

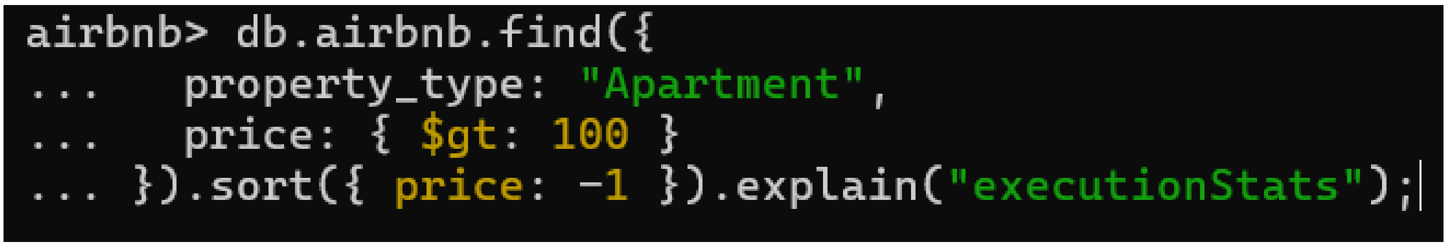
A standard single-field index isn't always effective when the user filters data by multiple criteria. We’re going to create an index that first groups the data by housing type and then sorts it by price within that type.



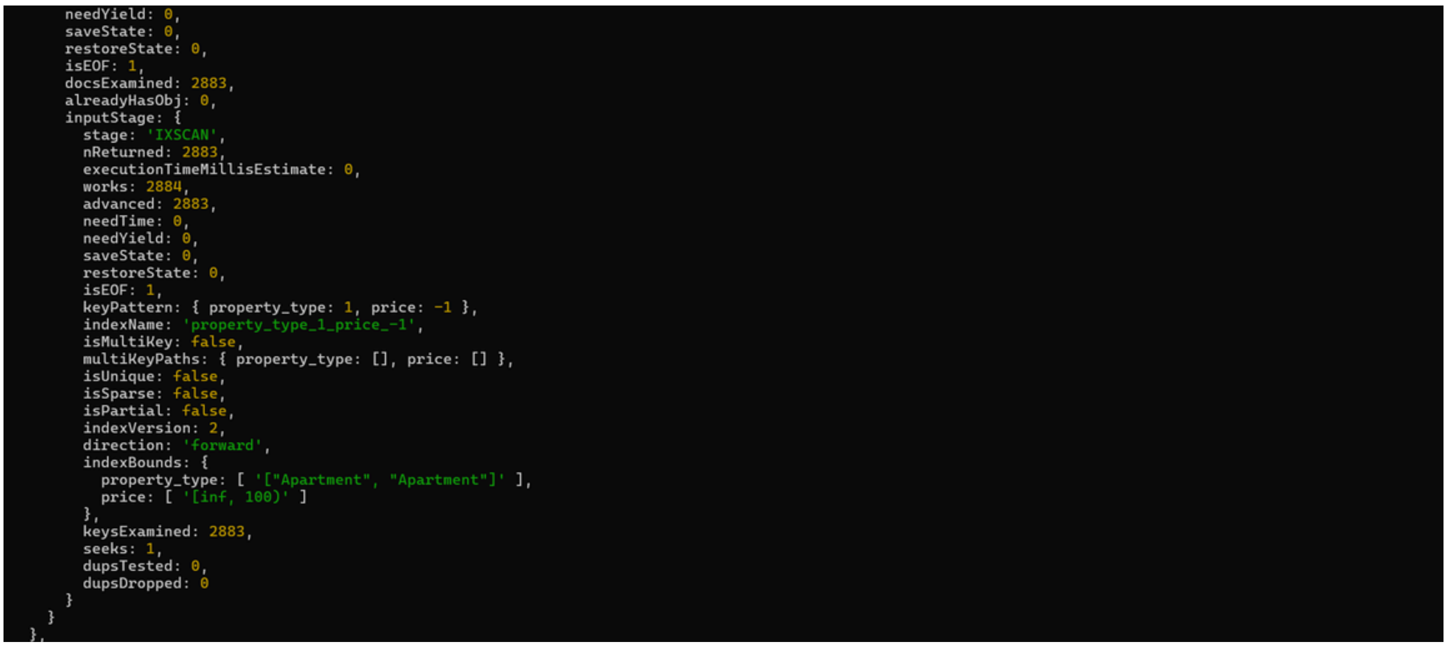
```
airbnb> db.airbnb.createIndex({
...   property_type: 1,
...   price: -1
... });
property_type_1_price_-1
airbnb> 
```

Using explain() command

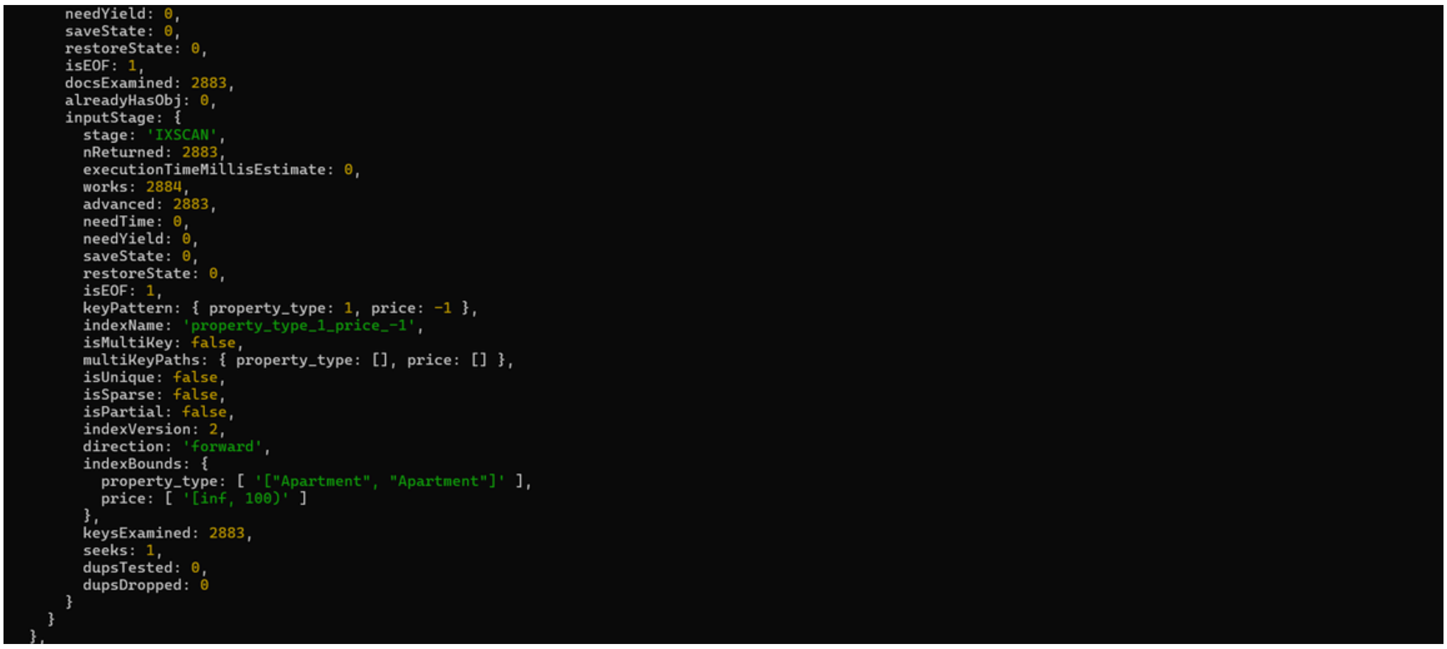
To make sure the index is working, we run a query with the .explain("executionStats") method.



```
airbnb> db.airbnb.find({
...   property_type: "Apartment",
...   price: { $gt: 100 }
... }).sort({ price: -1 }).explain("executionStats");
```



```
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      docsExamined: 2883,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 2883,
        executionTimeMillisEstimate: 0,
        works: 2884,
        advanced: 2883,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        keyPattern: { property_type: 1, price: -1 },
        indexName: 'property_type_1_price_-1',
        isMultiKey: false,
        multiKeyPaths: { property_type: [], price: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: {
          property_type: [ '["Apartment", "Apartment"]' ],
          price: [ '[inf, 100)' ]
        },
        keysExamined: 2883,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
```



```
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        docsExamined: 2883,
        alreadyHasObj: 0,
        inputStage: {
          stage: 'IXSCAN',
          nReturned: 2883,
          executionTimeMillisEstimate: 0,
          works: 2884,
          advanced: 2883,
          needTime: 0,
          needYield: 0,
          saveState: 0,
          restoreState: 0,
          isEOF: 1,
          keyPattern: { property_type: 1, price: -1 },
          indexName: 'property_type_1_price_-1',
          isMultiKey: false,
          multiKeyPaths: { property_type: [], price: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'forward',
          indexBounds: {
            property_type: [ '["Apartment", "Apartment"]' ],
            price: [ '[inf, 100)' ]
          },
          keysExamined: 2883,
          seeks: 1,
          dupsTested: 0,
          dupsDropped: 0
        }
      }
    },
```

According to the .explain() output, the query execution stage changed from COLLSCAN to IXSCAN. This reduces the number of documents scanned from 1.000+ to only the relevant ones, significantly improving the API response time.

Part VI: Authentification and authorization

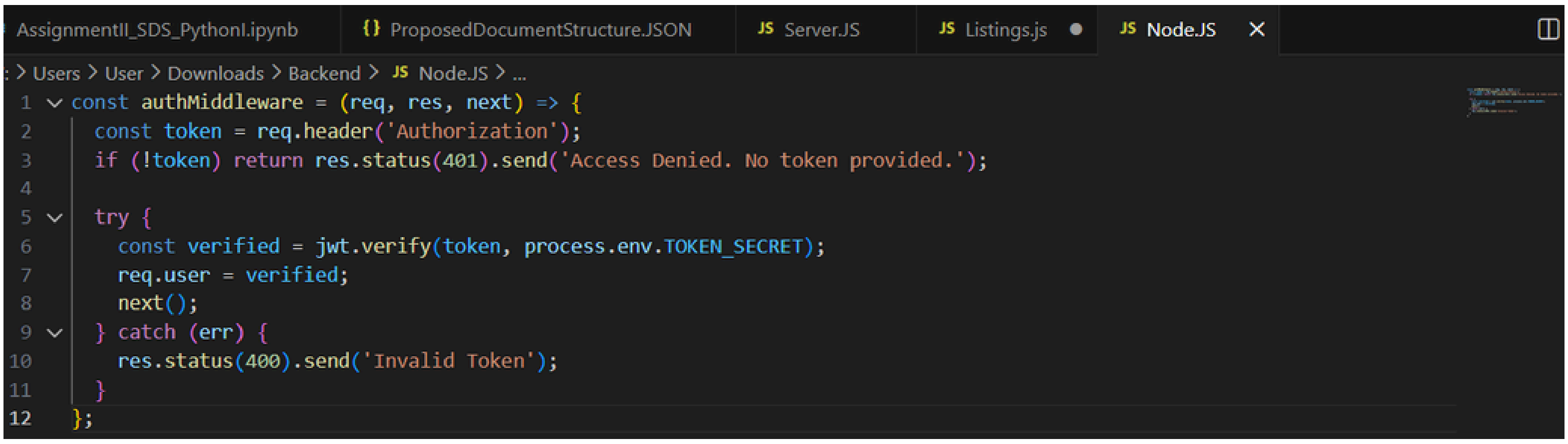
We create some JS-files to provide the authentification and authorization to

The following measures are used to secure the MongoDB Atlas cloud database:

IP Whitelisting: Access to the database is permitted only from specific IP addresses (for example, your computer and the hosting server).

Environment Variables (.env): All database access (URIs) and secret keys are stored in a secure .env file, which is added to .gitignore and is not included in the public GitHub repository.

Node.js:

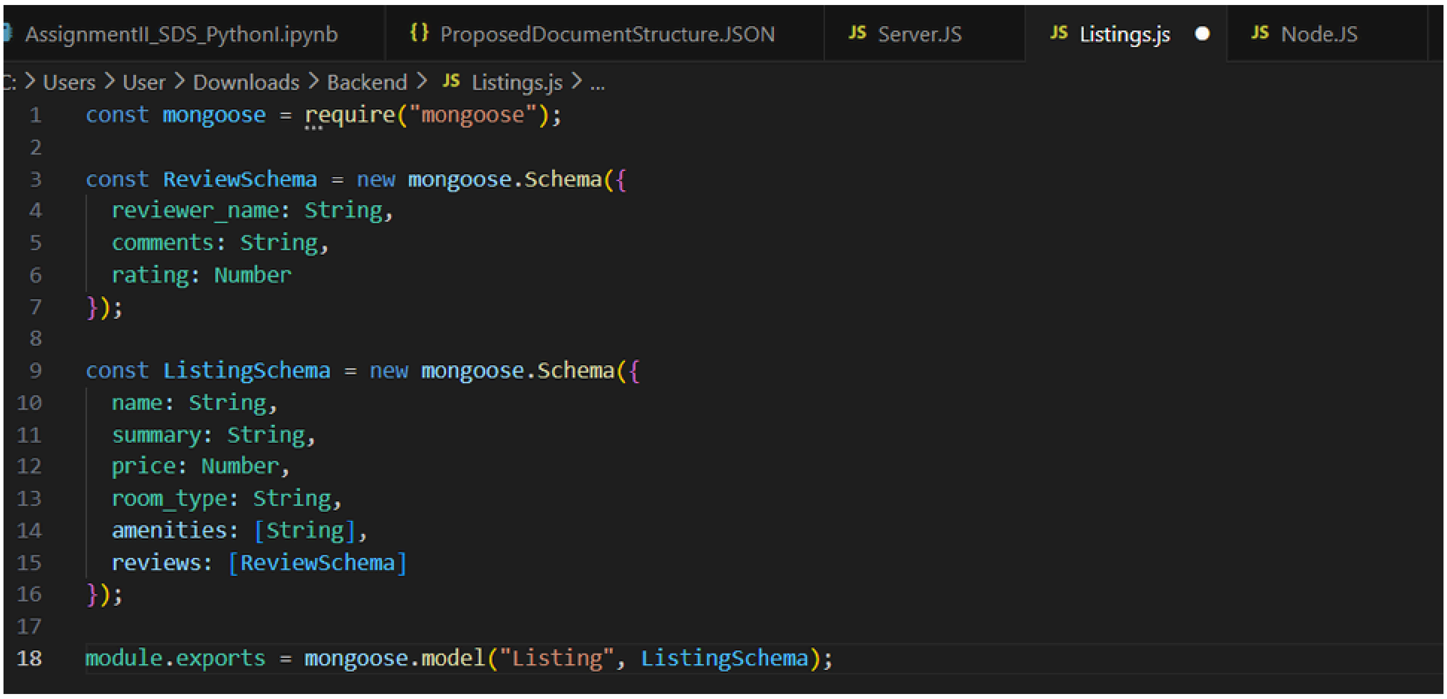


AssignmentII_SDS_PythonI.ipynb | ProposedDocumentStructure.JSON | Server.JS | Listings.js | Node.JS

: > Users > User > Downloads > Backend > Node.JS > ...

```javascript
const authMiddleware = (req, res, next) => {
  const token = req.header('Authorization');
  if (!token) return res.status(401).send('Access Denied. No token provided.');

  try {
    const verified = jwt.verify(token, process.env.TOKEN_SECRET);
    req.user = verified;
    next();
  } catch (err) {
    res.status(400).send('Invalid Token');
  }
};
```

Listings.js:

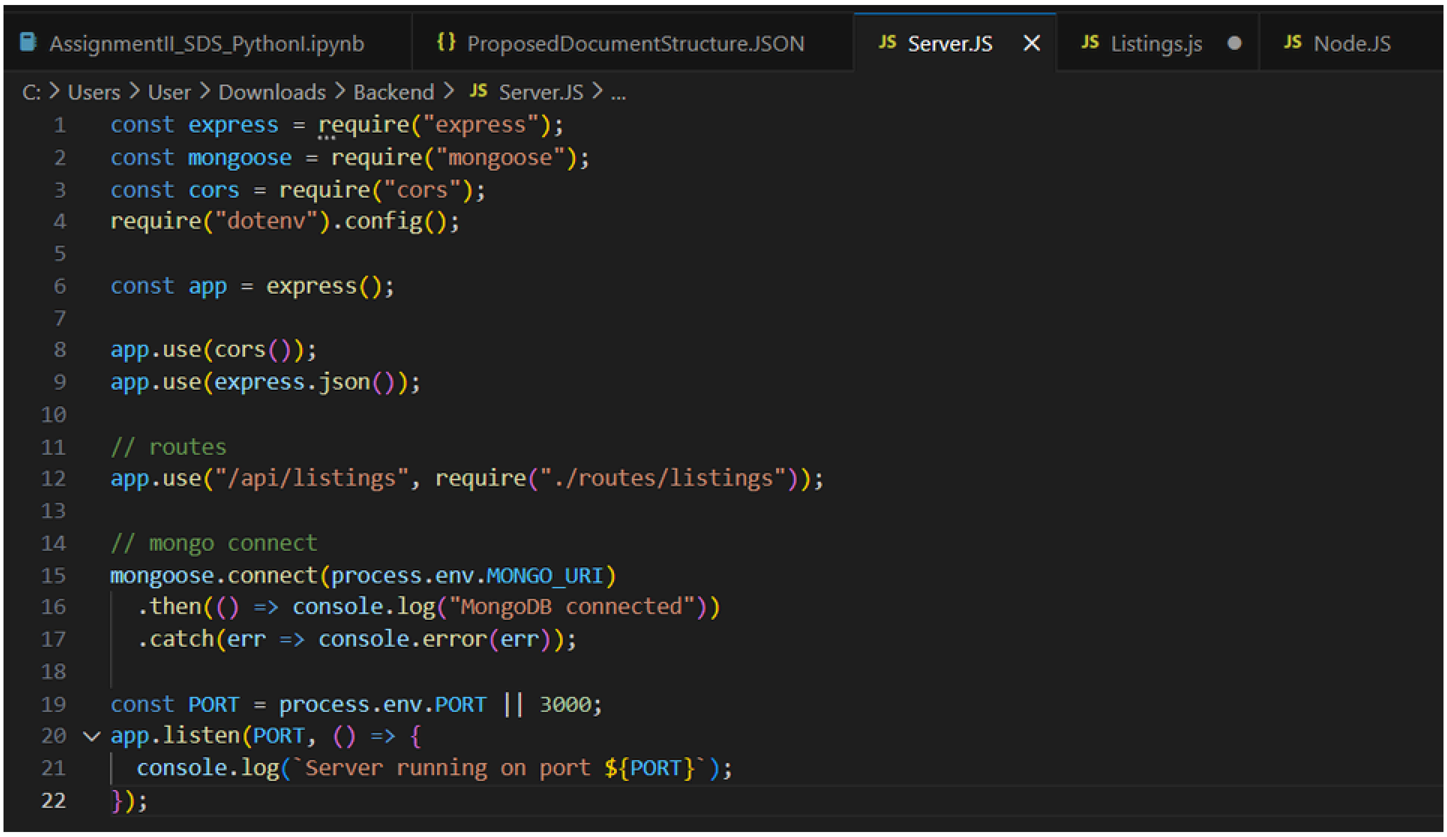


AssignmentII_SDS_PythonI.ipynb | ProposedDocumentStructure.JSON | Server.JS | Listings.js | Node.JS

C: > Users > User > Downloads > Backend > Listings.js > ...

```javascript
const mongoose = require("mongoose");

const ReviewSchema = new mongoose.Schema({
  reviewer_name: String,
  comments: String,
  rating: Number
});

const ListingSchema = new mongoose.Schema({
  name: String,
  summary: String,
  price: Number,
  room_type: String,
  amenities: [String],
  reviews: [ReviewSchema]
});

module.exports = mongoose.model("Listing", ListingSchema);
```

Server.js:


AssignmentII_SDS_PythonI.ipynb
ProposedDocumentStructure.JSON
Server.JS
Listings.js
Node.JS
C: > Users > User > Downloads > Backend > Server.JS > ...

```javascript
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
require("dotenv").config();

const app = express();

app.use(cors());
app.use(express.json());

// routes
app.use("/api/listings", require("./routes/listings"));

// mongo connect
mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log("MongoDB connected"))
  .catch(err => console.error(err));

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

1. Node.js (usually called db.js or connection.js)

Purpose: MongoDB client configuration: Creating a MongoClient instance.

2. Listings.js

This file contains all the logic related specifically to Airbnb data (listings).

Purpose: Defines endpoints: Paths are defined here, such as router.get('/'), router.post('/add'), router.put('/:id').

3. Servers.js

This is the "heart" of the application. It starts the server and ties everything together.

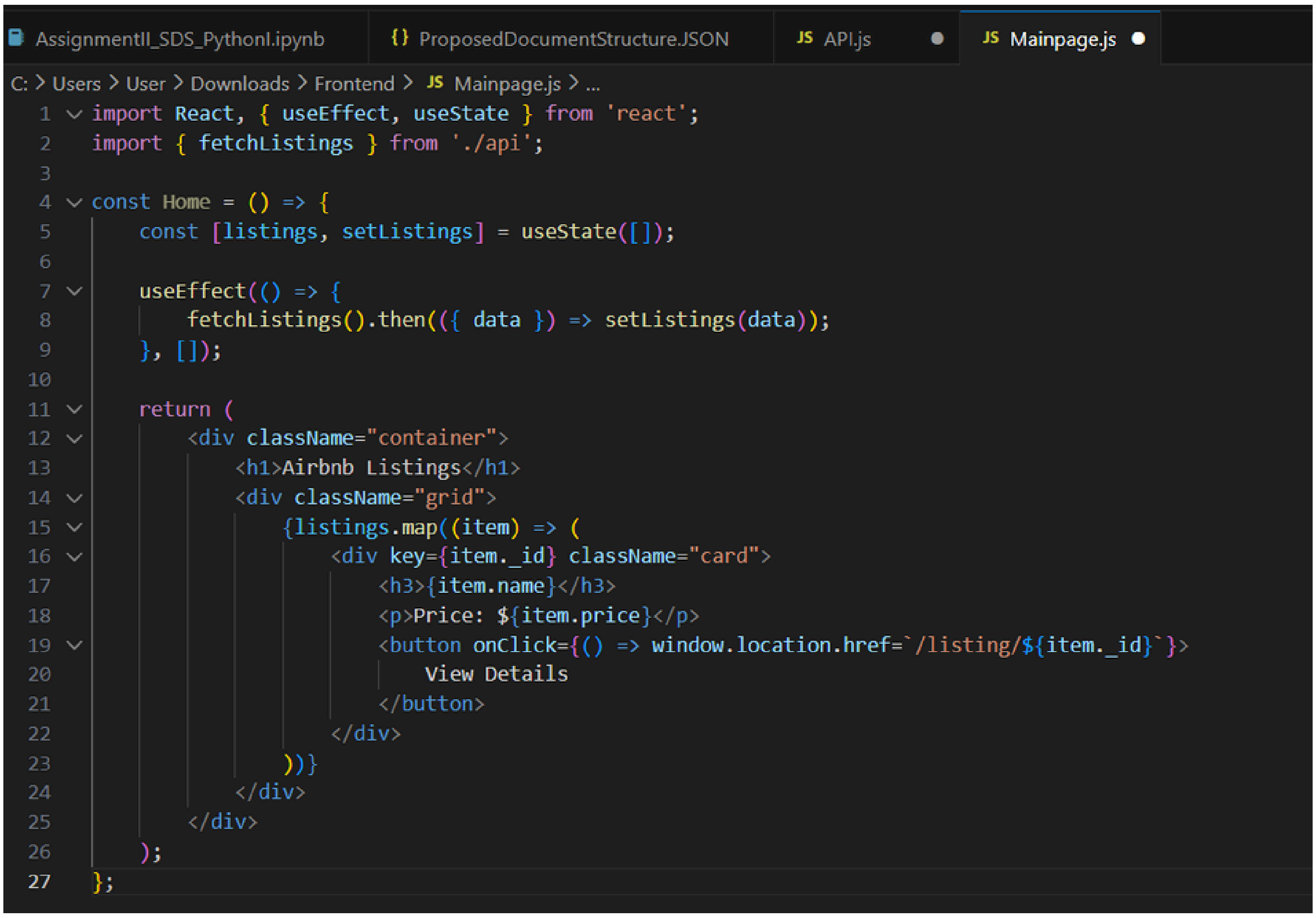
Purpose: Initializes the Express server.

Frontend’s role

The frontend is a Single Page Application (SPA) that communicates with the Node.js backend via REST API.

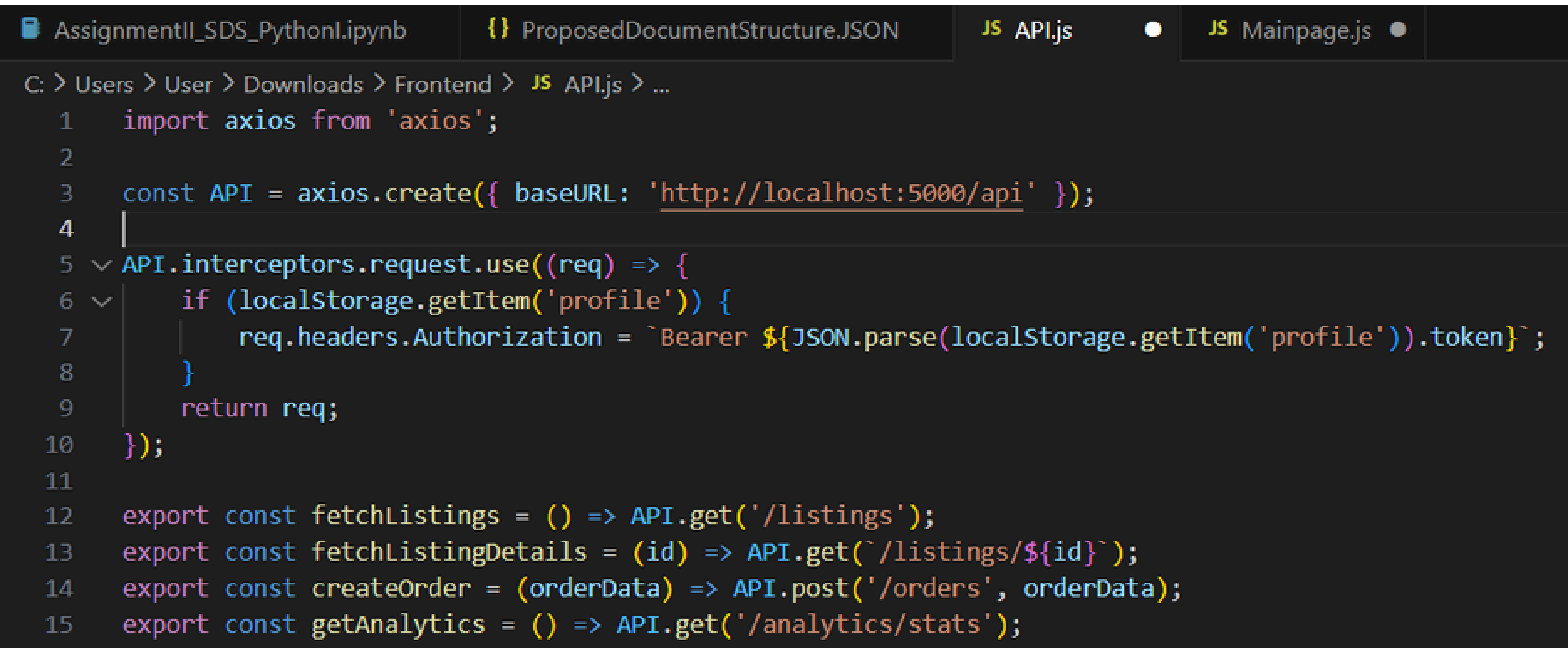
It includes a dynamic search engine, a booking system with real-time updates, and an admin panel for viewing MongoDB aggregation reports.

Main page:


AssignmentII_SDS_PythonI.ipynb
ProposedDocumentStructure.JSON
API.js
Mainpage.js
C: > Users > User > Downloads > Frontend > Mainpage.js > ...

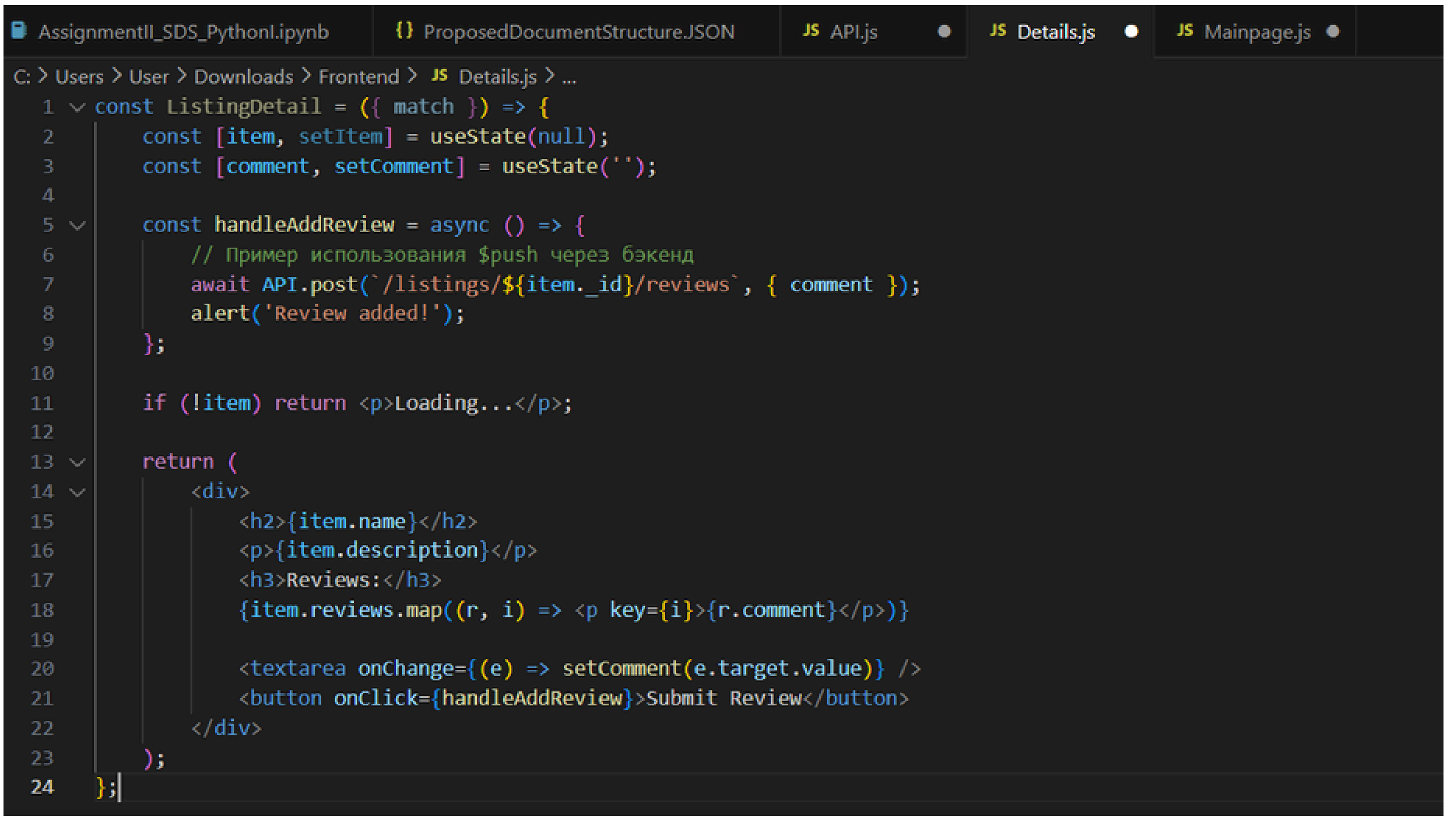
```
import React, { useEffect, useState } from 'react';
import { fetchListings } from './api';

const Home = () => {
    const [listings, setListings] = useState([]);

    useEffect(() => {
        fetchListings().then(({ data }) => setListings(data));
    }, []);

    return (
        <div className="container">
            <h1>Airbnb Listings</h1>
            <div className="grid">
                {listings.map((item) => (
                    <div key={item._id} className="card">
                        <h3>{item.name}</h3>
                        <p>Price: ${item.price}</p>
                        <button onClick={() => window.location.href=`/listing/${item._id}`}>
                            View Details
                        </button>
                    </div>
                ))}
            </div>
        </div>
    );
};
```

API.js:


AssignmentII_SDS_PythonI.ipynb
ProposedDocumentStructure.JSON
API.js
Mainpage.js
C: > Users > User > Downloads > Frontend > API.js > ...

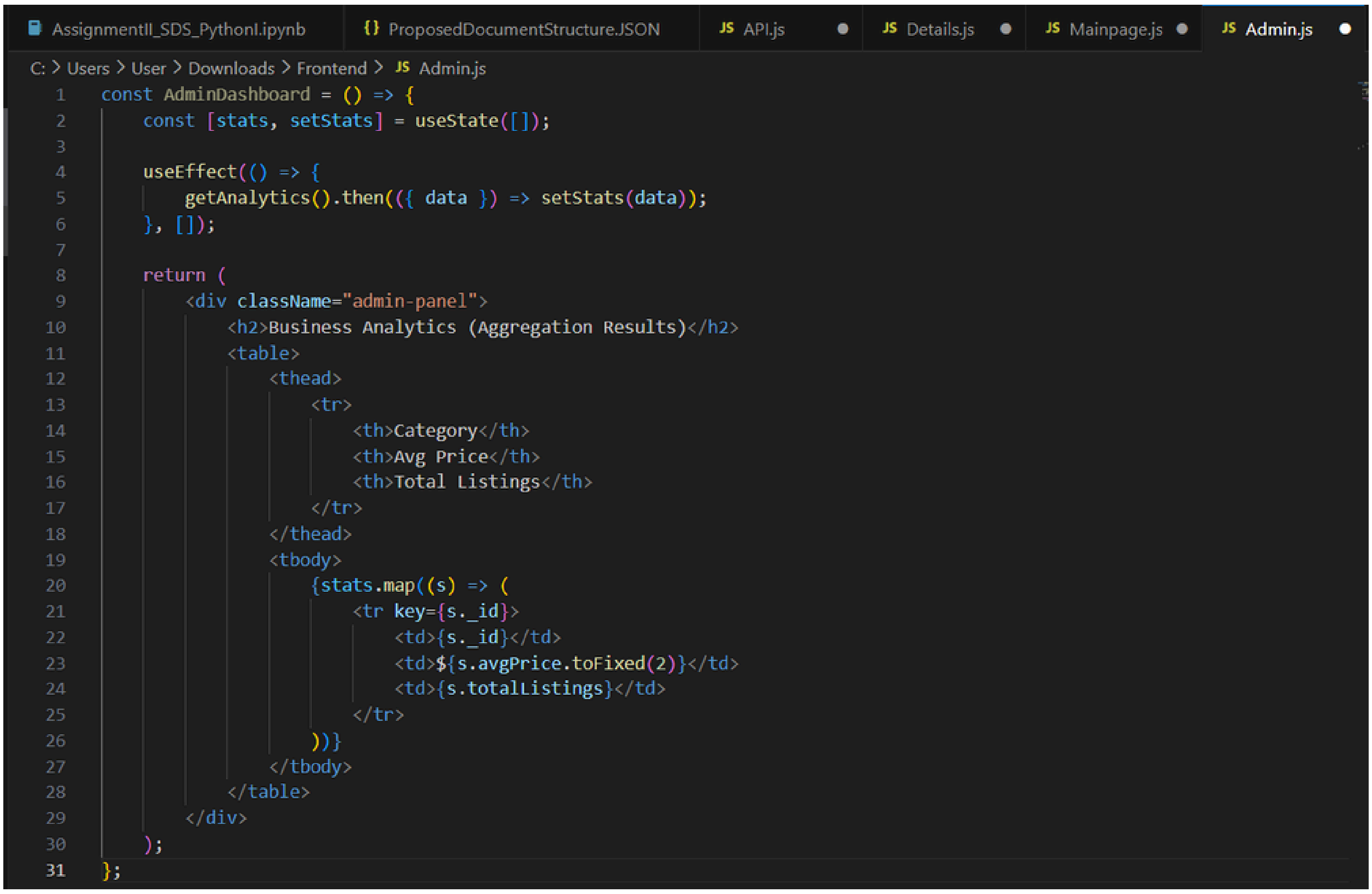
```javascript
import axios from 'axios';

const API = axios.create({ baseURL: 'http://localhost:5000/api' });

API.interceptors.request.use((req) => {
    if (localStorage.getItem('profile')) {
        req.headers.Authorization = `Bearer ${JSON.parse(localStorage.getItem('profile')).token}`;
    }
    return req;
});

export const fetchListings = () => API.get('/listings');
export const fetchListingDetails = (id) => API.get(`/listings/${id}`);
export const createOrder = (orderData) => API.post('/orders', orderData);
export const getAnalytics = () => API.get('/analytics/stats');
```

Details.js:


AssignmentII_SDS_PythonI.ipynb
ProposedDocumentStructure.JSON
API.js
Details.js
Mainpage.js
C: > Users > User > Downloads > Frontend > Details.js > ...

```javascript
const ListingDetail = ({ match }) => {
    const [item, setItem] = useState(null);
    const [comment, setComment] = useState('');

    const handleAddReview = async () => {
        // Пример использования $push через бэкенд
        await API.post(`/listings/${item._id}/reviews`, { comment });
        alert('Review added!');
    };

    if (!item) return <p>Loading...</p>;

    return (
        <div>
            <h2>{item.name}</h2>
            <p>{item.description}</p>
            <h3>Reviews:</h3>
            {item.reviews.map((r, i) => <p key={i}>{r.comment}</p>)}

            <textarea onChange={(e) => setComment(e.target.value)} />
            <button onClick={handleAddReview}>Submit Review</button>
        </div>
    );
};
```

Admin.js:


AssignmentII_SDS_PythonI.ipynb
ProposedDocumentStructure.JSON
API.js
Details.js
Mainpage.js
Admin.js
C: > Users > User > Downloads > Frontend > Admin.js

```
const AdminDashboard = () => {
    const [stats, setStats] = useState([]);

    useEffect(() => {
        getAnalytics().then(({ data }) => setStats(data));
    }, []);

    return (
        <div className="admin-panel">
            <h2>Business Analytics (Aggregation Results)</h2>
            <table>
                <thead>
                    <tr>
                        <th>Category</th>
                        <th>Avg Price</th>
                        <th>Total Listings</th>
                    </tr>
                </thead>
                <tbody>
                    {stats.map((s) => (
                        <tr key={s._id}>
                            <td>{s._id}</td>
                            <td>${s.avgPrice.toFixed(2)}</td>
                            <td>{s.totalListings}</td>
                        </tr>
                    ))}
                </tbody>
            </table>
        </div>
    );
};
```

Frontend JS Directory Structure

- **api** This folder serves as the **Data Communication Layer**. It contains the central logic for all HTTP requests (using Axios or Fetch) to your Node.js backend. Instead of writing URLs in every component, you define them here. It also handles the injection of **JWT tokens** into request headers to ensure that "Authorized" requests reach the database safely.

- **mainpage** This is the **Primary View Layer**. Its purpose is to handle the logic for the landing page where the user interacts with the bulk of the data. It manages the state for search inputs, category filters, and the initial rendering of documents from MongoDB. This is where your **Compound Index** performance will be most visible to the user as they filter through listings.

- **admin** This folder contains the **Analytics and Management Logic**. Its main purpose is to display the results of your **Multi-stage Aggregation Pipelines**. It fetches complex data reports (like sales trends or category averages) and presents them in tables or charts. This folder is restricted, meaning the code here should check if the logged-in user has "Admin" privileges before rendering.

- **details** This is the **Granular Data Layer**. Its purpose is to fetch and display a single, specific document from MongoDB by its ID. It handles the logic for viewing **Embedded Data** (like a list of reviews stored inside a listing) and provides the interface for **Advanced Updates**, such as allowing a user to "Push" a new comment into a review array or "Increment" a view counter.