Project Report - SmartFit

Vladislav Mazur - 658 453 550

Vancouver Island University - Fall 2025

CSCI 485 - Advance General Topics In Computer Science

Professor Kawal Jeet

November 30, 2025

Introduction

Background

SmartFit was created to be a health and wellness application to assist its users with the tasks such as logging their workouts, sports, meals, and personal metrics as well as the ability to find new places to work out at. While there are many applications that do each of these use cases very well such as: MyFitnessPal for food tracking, BoostCamp for tracking workouts, various apps to track sport specific data; this application was built with a jack-of-all-traits, master of none idea in mind.

Motivation

The developer of SmartFit - Vladislav Mazur is a data driven person and records much of his own activity down to the order of the exercises. The more data available, the easier it is to maintain the human body on an upward trend of fitness; doing a little more today than was done yesterday leads to progressive overload. The usual user of this application will have a cluttered phone with a series of apps to track all of this various data, spending time going in between them to record the desired data instead of doing what they enjoy.

With this in mind, the non-rigid schema of NoSQL comes into consideration as well. Each user can vary with the level of detail they want to record or the types of activities being recorded; something that would be difficult / not cost effective in the table like schema of SQL. The data can be input in various ways and polymorphism can be used to cut down on the needed collections.

Objectives & Goals

The application focuses on being the one stop shop for most things health and wellness, letting them decide themselves what they want to log. To allow for this, the application needed to have quick lookups - so that the user can quickly view their last workout and adjust the weight during this one accordingly; similarly with their last few meals. Insertion is also set to be of high importance in this application for the user to spend the least amount of time on the application - the insertions must be quick.

Report Outline

The purpose of this report is to help the outside observer understand the choices and direction that lead the development of SmartFit. It aims to be a guide and explanation to show the understanding of core MongoDB principles, and the roadmap of thinking that took place during this semester.

As a result, multiple database design decisions are highlighted which are accompanied by diagrams and validation rules. Following are the queries the application supports, including basic CRUD as well as complex aggregation and the results of such queries. Performance of the queries, optimization, and other advanced features are all considered in the latter section. Next, is the discussion of challenges faced and the workarounds used. Closing up with the conclusion and various references used within the scope of this project.

Database Design

Collections & Relationships

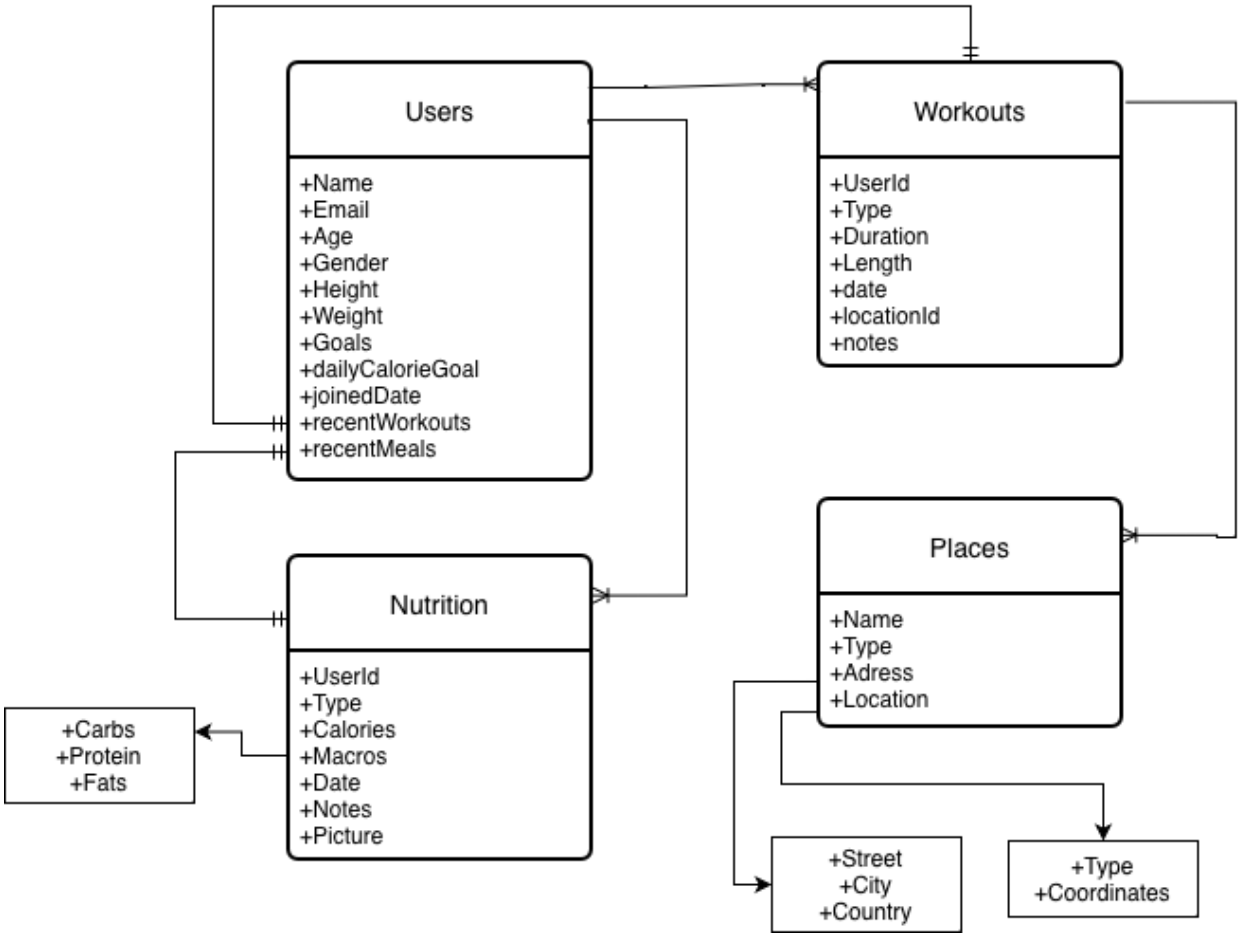
The User collection holds all the information about the actual user of the application, the one who is trying to achieve a goal they had set out for themselves. This collection has a one to many relationship with the workouts and nutrition collection. The thought behind this is that a user can have many logged meals and workouts and those will always be linked back to one user. This collection also has a one to one relation with a subset of the nutrition and workouts collection; mainly the last 3 of each for quick searches.

The Workouts collection has a many to one relationship with the user collection as outlined above, and has an optional many to one relationship with the places collection. If the user decides to record where they worked out this relation is used, where a place may have multiple workouts but a workout only has one place that it was recorded at.

The Nutrition collection has a many to one relationship with the user collection as stated previously, and has an embedded document of macros in a one to one relation.

The Places collection has a one to many relation with workouts as discussed, additionally, two embedded documents. One of the embedded documents is for the address of this place, while the other is a required document for GeoJSON queries; both of which have a one to one relation with the place.

Relationship Diagram



Schema Decisions

During the decision-making between embedding and referencing the thought process first went to reference everything - a very SQL style of thinking. After some discussion with professor Kawal Jeet and a deeper understanding of NoSQL principles, was when the real decision making was able to begin. The embedded documents in this application came in 4 different spots: recent workout and meals in the User collection, macros in the Nutrition collection, and address plus location in the Places collection. The idea behind including the recent workouts and meals in the User collection came from trying to limit the amount of joins required for a user. This way the user is able to quickly view their latest data and make appropriate changes without the performance degradation from looking into a different collection. Macros inside the Nutrition collection were chosen to be embedded due to the size of their data and where it would be used. All of the queries that use the macros are also requiring the data from the nutrition collection, therefore, referencing would not make logistical sense. Same logic can be applied to the address document within the Places collection that only features

information that would be valuable with the other info within the document. Finally, locations is a required embedded document for the GeoJSON queries.

The referencing decisions were made when the documents were too large to be embedded or the use case didn't make sense for the information to be stored within the collection. The User collection was referenced inside the Workouts collection to make sure that each workout has a user attached to it and no workouts are created without one. Similar thought processes weren't behind making the User collection referenced from the Nutrition collection. Additionally, Places are referenced in workouts if the user wants to link a place to a workout that they created. An extended reference is within the Workouts collection, not only linking Places but also having a small summary of the place.

Within the validation the focus was on making sure the user had freedom with how they log their data while still making sure that the important data was required for the queries. The schema was carefully considered and changed many times throughout the development of the application. The final iteration seems to be in-depth enough meanwhile leaving the user with freedom.

Starting from the User collection, the interesting parts here are the email regex, gender, max / mins, and the array for the recent documents. The email regex makes sure that the data gathered is within the proper email constraints. The gender field is put as string instead of enum due to the fact that some users might feel more comfortable putting in a certain gender that wasn't thought of during the development. Max and mins were chosen based on the oldest, tallest, and biggest recorded humans plus a couple extra to make sure the app doesn't need to be changed in the near future. Finally the recent arrays were capped at three maximum documents to prevent the bloating of the User collection.

Moving on to the Workouts collection, enum is used here to make sure the user doesn't misspell the type of workout that they just completed - making the queries more uniform. Duration deals with time, while length deals with distance. Letting the user decide whether they want to clock something as "12" or "15.45" seemed important to keep up the freedom, therefore, ints or doubles are allowed. Finally, additional properties are set to true to explicitly allow the user to enter any other information they require.

The Nutrition collection is decently self explanatory when it comes to validation, apart from the picture property. The picture property was put here to allow for GridFS to store images or video of the users meals, this was put towards stretch goals and not implemented within this version.

Lastly, the Places collection once again uses enum to assist the user with the choices of types the queries can understand. Within location it was important that the type was "point" and the coordinates were only two (longitude and latitude), and no polygon or set of coordinates was passed - making the max and min of the array, two.

For SmartFit's indexing strategy, the queries were all written out and ranked based on how often the user would complete each operation, then those were taken into consideration and indexed upon. These indexes were then tested against the data set available at the time and the .explain() command was run to see whether the new index cut down on the execution time. This led to the creation of an index on workouts and nutrition as those were the most used and had the most data within them. The other three indexes were created without the optimization mindset, those were: unique, TTL, and 2dSphere. In this application, the decision was made to let the users have the same name, and without a username field, the email was chosen to be the unique factor of each user. Making sure that all lower case emails were the same as upper case emails led me to use collation with a strength of two. TTL index was useful within the nutrition collection, due to the collection growing at such a rapid rate. The decision was made to delete all the nutritional logs after six months, to make sure the collection did not grow infinitely large. To wrap up, the 2dSphere index was created to work with the Places location document to allow the user to experience GeoJSON queries.

Constraints & Rules

USER COLLECTION:

```
// USERS COLLECTION
db.createCollection("Users", {
 validator: {
   $jsonSchema: {
     bsonType: "object",
     required: ["name", "email", "joinedDate", "dailyCalorieGoal"],
     properties: {
       "name": { bsonType: "string", description: "User's full name" },
       "email": {
         bsonType: "string",
         pattern: "^[\\w.-]+@[\\w.-]+\\.[A-Za-z]{2,}$",
         description: "Valid email format"
       },
       "age": { bsonType: "int", minimum: 0, maximum: 150 },
       "gender": { bsonType: "string" },
       "height": { bsonType: ["int", "double"], minimum: 0, maximum: 300 },
       "weight": { bsonType: ["int", "double"], minimum: 0, maximum: 700 },
       "goals": { bsonType: "string" },
       "dailyCalorieGoal": { bsonType: "int", minimum: 0 },
       "joinedDate": { bsonType: "date" },
       "recentWorkouts": {
         bsonType: "array",
         maxItems: 3,
         description: "Last 3 Workouts",
         items: {
           bsonType: "object",
           required: ["_id", "type", "date"],
           properties: {
             "_id": { bsonType: "objectId" },
             "type": { bsonType: "string" },
             "date": { bsonType: "date" },
             "duration": { bsonType: ["int", "double"] },
             "length": { bsonType: ["int", "double"] }
           }
         }
       },
       "recentNutrition": {
         bsonType: "array",
         maxItems: 3,
```

```
        description: "Last 3 Meals",
        items: {
          bsonType: "object",
          required: ["_id", "type", "date", "calories"],
          properties: {
            "_id": { bsonType: "objectId" },
            "type": { bsonType: "string" },
            "date": { bsonType: "date" },
            "calories": { bsonType: "int" }
          }
        }
      }
    }
  }
});
```

WORKOUTS COLLECTION:

```
//  WORKOUTS COLLECTION
db.createCollection("Workouts", {
 validator: {
   $jsonSchema: {
     bsonType: "object",
     required: ["userId", "type", "date"],
     properties: {
       "userId": { bsonType: "objectId", description: "Refers to user" },
       "type": { bsonType: "string", enum: ["Weights", "Cardio", "Sport", "Other"] },
       "duration": { bsonType: ["int", "double"], description: "Time Based", minimum:
0 },
       "length": { bsonType: ["int", "double"], description: "Distance Based",
minimum: 0 },
       "date": { bsonType: "date" },
       "locationId": { bsonType: "objectId" },
       "locationSummary": {
         bsonType: "object",
         required: ["name", "address"],
         properties: {
           "name": { bsonType: "string", description: "Name of the place" },
           "address": {
             bsonType: "object",
             required: ["street", "city", "country"],
             properties: {
               "street": { bsonType: "string" },
               "city": { bsonType: "string" },
               "country": { bsonType: "string" }
             }
           }
         }
       },
       "notes": { bsonType: "string" }
     },
     // to allow for other sports or metrics
     additionalProperties: true
   }
 }
});
```

NUTRITION COLLECTION:

```
//  NUTRITION COLLECTION
db.createCollection("Nutrition", {
 validator: {
   $jsonSchema: {
     bsonType: "object",
     required: ["userId", "calories", "date"],
     properties: {
       "userId": { bsonType: "objectId", description: "References which user ate" },
       "type": { bsonType: "string", enum: ["Breakfast", "Lunch", "Dinner", "Snack"]
},
       "calories": { bsonType: "int", minimum: 0 },
       "macros": {
         bsonType: "object",
         required: ["carbs", "protein", "fats"],
         properties: {
           "carbs": { bsonType: "int", minimum: 0 },
           "protein": { bsonType: "int", minimum: 0 },
           "fats": { bsonType: "int", minimum: 0 }
         }
       },
       "date": { bsonType: "date" },
       "notes": { bsonType: "string" },
       "picture": { bsonType: "objectId", description: "Reference to GridFS meal
picture" }
     }
   }
 }
});
```

PLACES COLLECTION:

```
//  PLACES COLLECTION
db.createCollection("Places", {
 validator: {
   $jsonSchema: {
     bsonType: "object",
     required: ["name", "type", "address", "location"],
     properties: {
       "name": { bsonType: "string" },
       "type": { bsonType: "string", enum: ["Arena", "Track", "Gym", "Park",
"Facility", "Stadium", "Other"] },
       "address": {
         bsonType: "object",
         required: ["street", "city", "country"],
         properties: {
           "street": { bsonType: "string" },
           "city": { bsonType: "string" },
           "country": { bsonType: "string" }
         }
       },
       "location": {
         bsonType: "object",
         required: ["type", "coordinates"],
         properties: {
           "type": { enum: ["Point"], description: "Must be 'Point' for geoJSON
queries" },
           "coordinates": {
             bsonType: "array",
             minItems: 2,
             maxItems: 2,
             items: { bsonType: "double" },
             description: "[longitude, latitude]"
           }
         }
       }
     }
   }
 }
});
```

Queries & Operations

CRUD

This project implements a variety of CRUD (Create, Read, Update, Delete) queries to allow for a smooth user experience. These were spread out across the four collections to demonstrate some of the benefits of mongoDB.

Within the User collection the create query can be found by making a test user named Alice with different properties. Queries retrieve user data by name or email, identify users with high calorie goals, or list the five most recently joined users. The update queries work on adjusting calorie goals, incrementing height or weight values, some collation-sensitive updates (makes find by email easier), removing the last workout from a user's "recentWorkouts" array, and setting a "firstWorkoutDate" only if one does not already exist. The system can also delete user profiles based on filters, such as deleting all users under age 22 who are not female.

Workout queries focus on adding and removing workouts, updating their workout details, and deleting old ones. New workouts can be added, associated with a specific user ID, and include details such as type, duration, date, and notes. Find examples include retrieving a user's last 10 workouts or finding the second-longest recorded workout. Updating workout details can be done, such as updating workout notes. Finally, the user is able to delete older data, cleanup is performed by deleting workouts older than one year.

On the Nutrition collection note, Meal entries are inserted with calorie counts, macro breakdowns, timestamps, and notes. A TTL-test entry was created to validate time-to-live index behavior. Here queries retrieve: the most recent meals for a specific user, and the last 10 lunch or dinner meals over 500 calories. Updating allows adjusting calorie counts when meals are logged incorrectly on the first creation. Lastly, nutrition entries missing required macro fields (carbs, protein, fats) can be removed using the query.

The Places collection had the ability to create new locations that can be added with names, types, addresses, and geospatial coordinates. Allows for reading geospatial query, which finds nearby gyms within a 5 km radius using. Places properties can be updated, in this case, renaming a gym. All gyms within a specific region can be removed, in this case, all gyms in Delta.

Representative Queries

Basic

```
// USER QUERIES

//Create alice for testing
const insertAlice = db.Users.insertOne({
   "name": "Alice",
   "email": "alice@example.com",
   "joinedDate": new Date(),
   "dailyCalorieGoal": 2200,
   "age": 28,
   "gender": "Female",
   "height": 165,
   "weight": 60,
   "goals": "Lose 5 kg"
});
// to use alice later
const aliceId = insertAlice.insertedId;

// Update Alices calorie goal
db.Users.updateOne(
   { "name": "Alice" },
   { $set: { "dailyCalorieGoal": 2300 } }
);

// Find user using email or name
db.Users.findOne({ $or: [{ "email": "alice@example.com" }, {"name": "alice"} ]} );

// Get all users whos calorie goal > 2500
db.Users.find({ "dailyCalorieGoal": { $gt: 2500 } });

// Count users with atleast 1 workout
db.Users.find({ "recentWorkouts": { $exists: true, $not: { $size: 0 } } }).count();

// Find the 5 most recently joined users
db.Users.find().sort({ "joinedDate": -1 }).limit(5);

// Add 1 cm to users height
db.Users.updateOne(
   { "email": "alice@example.com" },
```

```
   { $inc: { "height": 1 } }
);

// Remove 1 kg from users weight
db.Users.updateOne(
   { "name": "alice@example.com" },
   { $inc: { "weight": -1 } }
);

//Remove the last workout from recents array
db.Users.updateOne(
   { "email": "alice@example.com" },
   { $pop: { "recentWorkouts": -1 } }
);


// make a new field and store first workout there (if none already there - make first
workout be today)
db.Users.updateOne(
   { "email": "someones email" }, // insert actual email
   { $min: { "firstWorkoutDate": new Date() } },
   { collation: { locale: "en", strength: 2 } }
);

// delete all who are under 22 and are not female
db.Users.deleteMany({
   $and: [
       { "age": { $lt: 22 } },
       { "gender": { $ne: "Female" } }
   ]
});




// WORKOUT QUERIES

//create a workout for alice
const insertWorkout = db.Workouts.insertOne({
   "userId": ObjectId(aliceId),
   "type": "Cardio",
   "date": new Date(),
```

```
   "duration": 45,
   "length": 10,
   "notes": "Felt strong"
});

//use this workout later
const workoutId = insertWorkout.insertedId;

// change note on alices workout
db.Workouts.updateOne(
   { "_id": workoutId },
   { $set: { "notes": "Ran faster today!" } }
);

// Find a specific users last 10 workouts
db.Workouts.find({ "userId": ObjectId('692671f89e21211ca1f71daa') }).sort({ "date": -1
}).limit(10); //Eve Martin

//delete 1 year old or older workouts
db.Workouts.deleteMany({
   "date": { $lt: new Date(Date.now() - 31536000000) }
});

//find the second longest workout for a user
db.Workouts.find({ "userId": ObjectId('692671f89e21211ca1f71dab') }).sort({ duration:
-1 }).skip(1).limit(1); //Mia Martinez




//NUTRITION QUERIES

//give alice a meal
const nutritionId = db.Nutrition.insertOne({
   "userId": aliceId,
   "type": "Lunch",
   "calories": 600,
   "macros": { "carbs": 80, "protein": 25, "fats": 20 },
   date: new Date(),
   "notes": "Healthy meal",
}).insertedId;
```

```
//meal was actually 100 calories more
db.Nutrition.updateOne(
   { "_id": nutritionId },
   { $inc: { "calories": 100 } }
);

// Query to test TTL
db.Nutrition.insertOne({
   "userId": aliceId, // used with alice
   "type": "Lunch",
   "calories": 500,
   "macros": { "carbs": 50, "protein": 20, "fats": 15 },
   "date": new Date(Date.now() - 15778800 * 1000), // ~6 months ago
   "notes": "Old meal for TTL test"
});

//delete any entries that don't have any of the macros
db.Nutrition.deleteMany({
   $or: [
     { "macros": { $exists: false } },
     { "macros.carbs": { $exists: false } },
     { "macros.protein": { $exists: false } },
     { "macros.fats": { $exists: false } }
   ]
});

// Find a specific users last 10 workouts
db.Nutrition.find({ "userId": ObjectId('692671f89e21211ca1f71df3') }).sort({ "date":
-1 }).limit(10); //Eve Anderson

//Find the last 10 meals that were lunch or dinner and gt 500 cals
db.Nutrition.find({
   "type": { $in: ["Lunch", "Dinner"] },
   "calories": { $gt: 500 }
}).sort({"date": -1}).limit(10);



//PLACES QUERIES

//add a place
```

```
const placeId = db.Places.insertOne({
   "name": "Central Gym",
   "type": "Gym",
   "address": { "street": "123 Main St", "city": "Vancouver", "country": "Canada" },
   "location": { "type": "Point", "coordinates": [-123.115, 49.28] }
 }).insertedId;
 // name change
db.Places.updateOne(
   { "_id": placeId },
   { $set: { "name": "Central Fitness Center" } }
);

//find gyms within 5k radius (3 limit)
db.Places.find({
   "type": "Gym",
   "location": {
     $near: {
       $geometry: { "type": "Point", "coordinates": [-123.0990, 49.2825] },
       $maxDistance: 5000
     }
   }
}).limit(3);

//delete all the gyms in delta
db.Places.deleteMany({
   "type": "Gym",
   "address.city": "Delta"
});
```

Aggregation & Analysis

```
//get last 14 days of cals
const fourteenDaysAgo = new Date();
fourteenDaysAgo.setDate(fourteenDaysAgo.getDate() - 14);

db.Nutrition.aggregate([
   {
       $match: {
           "userId": ObjectId('692671f89e21211ca1f71d98'), //Tina Anderson
           "date": { $gte: fourteenDaysAgo, $lte: new Date() } // Last 14 days
       }
   },
   {
```

```
        $group: {
            _id: null,
            totalCalories: { $sum: "$calories" } // Sum up the calories that alice had
into "totalCalories"
        }
    },
    {
        $project: { _id: 0, totalCalories: 1 } // Show just the calories
    }
]);

// Get total duration and length grouped by workout type for the last 30 days
const thirtyDaysAgo = new Date();
thirtyDaysAgo.setDate(thirtyDaysAgo.getDate() - 30);

db.Workouts.aggregate([
    {
        $match: {
            "userId": ObjectId('692671f89e21211ca1f71dbe'), // Jack Anderson
            "date": { $gte: thirtyDaysAgo, $lte: new Date() }
        }
    },
    {
        $group: {
            _id: "$type", // Group by the workout type
            totalDuration: { $sum: "$duration" }, // Sum up the duration in time
            totalLength: { $sum: "$length" },     // Sum up the lenght in distance
            count: { $sum: 1 }                    // Count how many workouts of this
type were done
        }
    },
    {
        $project: {
            _id: 0,
            workoutType: "$_id",
            totalDuration_minutes: "$totalDuration",
            totalLength_units: "$totalLength",
            numberOfWorkouts: "$count"
        }
    }
]);
```

```
//Find the avg calories of a meal eaten accross all users
db.Nutrition.aggregate([
   {
       $group: {
           _id: null, // All documents together
           averageMealCalories: { $avg: "$calories" } // Calculate the average cals
       }
   },
   {
       $project: {
           _id: 0,
           averageMealCalories: { $round: ["$averageMealCalories", 0] } // Round to
get an whole number
       }
   }
]);
```

Results of Queries

These query results are first, only the find queries are used; it is then followed by the aggregation queries:

FIND:

```
mazurv_practice> db.Users.findOne({ $or: [{ "email": "alice@example.com" }, {"name": "alice"} ]} );
{
  _id: ObjectId('692675309e21211ca1f75599'),
  name: 'Alice',
  email: 'alice@example.com',
  joinedDate: ISODate('2025-11-26T03:34:08.199Z'),
  dailyCalorieGoal: 2300,
  age: 28,
  gender: 'Female',
  height: 165,
  weight: 60,
  goals: 'Lose 5 kg'
}
```

—

```
db.Users.find({ "dailyCalorieGoal": { $gt: 2500 } }).limit(10);
[
  {
    _id: ObjectId('692671f89e21211ca1f71d9b'),
    name: 'Bob Wilson',
    email: 'bob.wilson3@example.com',
    age: 25,
    gender: 'Male',
    height: 174,
    weight: 92,
    dailyCalorieGoal: 2674,
```

```
  goals: 'Improve endurance',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
},
{
  _id: ObjectId('692671f89e21211ca1f71d9c'),
  name: 'Jack Thompson',
  email: 'jack.thompson4@example.com',
  age: 45,
  gender: 'Male',
  height: 166,
  weight: 93,
  dailyCalorieGoal: 2676,
  goals: 'Run a marathon',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
},
{
  _id: ObjectId('692671f89e21211ca1f71d9e'),
  name: 'Olivia Hernandez',
  email: 'olivia.hernandez6@example.com',
  age: 36,
  gender: 'Male',
  height: 178,
  weight: 54,
  dailyCalorieGoal: 2747,
  goals: 'Gain muscle',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
},
{
  _id: ObjectId('692671f89e21211ca1f71da6'),
  name: 'Noah Garcia',
  email: 'noah.garcia14@example.com',
  age: 30,
  gender: 'Female',
  height: 194,
  weight: 57,
  dailyCalorieGoal: 2768,
  goals: 'Improve endurance',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
},
{
  _id: ObjectId('692671f89e21211ca1f71da7'),
  name: 'Jack Perez',
  email: 'jack.perez15@example.com',
  age: 20,
  gender: 'Female',
  height: 172,
  weight: 93,
  dailyCalorieGoal: 2511,
  goals: 'Gain muscle',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
},
{
  _id: ObjectId('692671f89e21211ca1f71da9'),
  name: 'Charlie Hernandez',
  email: 'charlie.hernandez17@example.com',
  age: 54,
  gender: 'Male',
```

```
  height: 158,
  weight: 99,
  dailyCalorieGoal: 2645,
  goals: 'Get Stronger',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
},
{
  _id: ObjectId('692671f89e21211ca1f71daf'),
  name: 'Liam Jones',
  email: 'liam.jones23@example.com',
  age: 41,
  gender: 'Male',
  height: 191,
  weight: 61,
  dailyCalorieGoal: 2685,
  goals: 'Gain muscle',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
},
{
  _id: ObjectId('692671f89e21211ca1f71db0'),
  name: 'Grace Lee',
  email: 'grace.lee24@example.com',
  age: 43,
  gender: 'Female',
  height: 198,
  weight: 90,
  dailyCalorieGoal: 2590,
  goals: 'Get Stronger',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
},
{
  _id: ObjectId('692671f89e21211ca1f71db8'),
  name: 'Mia Williams',
  email: 'mia.williams32@example.com',
  age: 22,
  gender: 'Male',
  height: 160,
  weight: 92,
  dailyCalorieGoal: 2712,
  goals: 'Gain muscle',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
},
{
  _id: ObjectId('692671f89e21211ca1f71dbc'),
  name: 'Quinn Garcia',
  email: 'quinn.garcia36@example.com',
  age: 47,
  gender: 'Male',
  height: 167,
  weight: 71,
  dailyCalorieGoal: 2525,
  goals: 'Get Stronger',
  joinedDate: ISODate('2025-11-26T03:20:24.507Z')
 }
]
```

—

```
db.Users.find({ "recentWorkouts": { $exists: true, $not: { $size: 0 } } }).count();
```

0

```
db.Users.find().sort({ "joinedDate": -1 }).limit(5);
[
  {
    _id: ObjectId('692675309e21211ca1f75599'),
    name: 'Alice',
    email: 'alice@example.com',
    joinedDate: ISODate('2025-11-26T03:34:08.199Z'),
    dailyCalorieGoal: 2300,
    age: 28,
    gender: 'Female',
    height: 165,
    weight: 60,
    goals: 'Lose 5 kg'
  },
  {
    _id: ObjectId('692671f89e21211ca1f71dcc'),
    name: 'Charlie Wilson',
    email: 'charlie.wilson_req52@example.com',
    joinedDate: ISODate('2025-11-26T03:20:24.510Z'),
    dailyCalorieGoal: 2421
  },
  {
    _id: ObjectId('692671f89e21211ca1f71dcb'),
    name: 'Bob Smith',
    email: 'bob.smith_req51@example.com',
    joinedDate: ISODate('2025-11-26T03:20:24.510Z'),
    dailyCalorieGoal: 2728
  },
  {
    _id: ObjectId('692671f89e21211ca1f71dce'),
    name: 'Quinn Williams',
    email: 'quinn.williams_req54@example.com',
    joinedDate: ISODate('2025-11-26T03:20:24.510Z'),
    dailyCalorieGoal: 2621
  },
  {
    _id: ObjectId('692671f89e21211ca1f71dcd'),
    name: 'Alice Lee',
    email: 'alice.lee_req53@example.com',
    joinedDate: ISODate('2025-11-26T03:20:24.510Z'),
    dailyCalorieGoal: 2261
  }
]
```

```
db.Workouts.find({ "userId": ObjectId('692671f89e21211ca1f71daa') }).sort({ "date": -1 }).limit(10); //Eve Martin
...
[
  {
    _id: ObjectId('692671f89e21211ca1f72482'),
    userId: ObjectId('692671f89e21211ca1f71daa'),
    type: 'Cardio',
    date: ISODate('2025-11-26T03:20:24.547Z'),
    duration: 89,
    length: 5,
```

```
  notes: 'Felt strong running in the park'
},
{
  _id: ObjectId('692671f89e21211ca1f72484'),
  userId: ObjectId('692671f89e21211ca1f71daa'),
  type: 'Cardio',
  date: ISODate('2025-11-23T03:20:24.547Z'),
  duration: 70,
  length: 1,
  notes: 'Exhausted cycling at home'
},
{
  _id: ObjectId('692671f89e21211ca1f724af'),
  userId: ObjectId('692671f89e21211ca1f71daa'),
  type: 'Other',
  date: ISODate('2025-11-18T03:20:24.547Z'),
  duration: 38,
  length: 10,
  notes: 'Relaxed playing hockey at home'
},
{
  _id: ObjectId('692671f89e21211ca1f724b8'),
  userId: ObjectId('692671f89e21211ca1f71daa'),
  type: 'Weights',
  date: ISODate('2025-11-16T03:20:24.547Z'),
  duration: 86,
  length: 1,
  notes: 'Exhausted playing hockey at the gym'
},
{
  _id: ObjectId('692671f89e21211ca1f724ab'),
  userId: ObjectId('692671f89e21211ca1f71daa'),
  type: 'Sport',
  date: ISODate('2025-11-14T03:20:24.547Z'),
  duration: 31,
  length: 4,
  notes: 'Relaxed cycling in the park'
},
{
  _id: ObjectId('692671f89e21211ca1f72476'),
  userId: ObjectId('692671f89e21211ca1f71daa'),
  type: 'Sport',
  date: ISODate('2025-11-09T03:20:24.547Z'),
  duration: 34,
  length: 7,
  notes: 'Motivated lifting weights with friends'
},
{
  _id: ObjectId('692671f89e21211ca1f72478'),
  userId: ObjectId('692671f89e21211ca1f71daa'),
  type: 'Other',
  date: ISODate('2025-11-07T03:20:24.547Z'),
  duration: 37,
  length: 7,
  notes: 'Relaxed playing hockey at the gym'
},
{
```

```
    _id: ObjectId('692671f89e21211ca1f72488'),
    userId: ObjectId('692671f89e21211ca1f71daa'),
    type: 'Cardio',
    date: ISODate('2025-11-04T03:20:24.547Z'),
    duration: 115,
    length: 5,
    notes: 'Relaxed playing hockey on the track'
  },
  {
    _id: ObjectId('692671f89e21211ca1f7245d'),
    userId: ObjectId('692671f89e21211ca1f71daa'),
    type: 'Cardio',
    date: ISODate('2025-10-31T03:20:24.544Z'),
    duration: 110,
    length: 1,
    notes: 'Energetic running in the park'
  },
  {
    _id: ObjectId('692671f89e21211ca1f724dd'),
    userId: ObjectId('692671f89e21211ca1f71daa'),
    type: 'Other',
    date: ISODate('2025-10-30T03:20:24.547Z'),
    duration: 90,
    length: 1,
    notes: 'Felt strong doing yoga in the park'
  }
]
```

```
db.Workouts.find({ "userId": ObjectId('692671f89e21211ca1f71dab') }).sort({ duration: -1 }).skip(1).limit(1);
[
  {
    _id: ObjectId('692671f89e21211ca1f72536'),
    userId: ObjectId('692671f89e21211ca1f71dab'),
    type: 'Weights',
    date: ISODate('2025-05-11T03:20:24.550Z'),
    duration: 117,
    length: 6,
    notes: 'Energetic lifting weights with friends'
  }
]
```

```
db.Nutrition.find({ "userId": ObjectId('692671f89e21211ca1f71df3') }).sort({ "date": -1 }).limit(10);
[
  {
    _id: ObjectId('692671f89e21211ca1f7529f'),
    userId: ObjectId('692671f89e21211ca1f71df3'),
    calories: 533,
    date: ISODate('2025-11-18T03:20:24.775Z')
  },
  {
    _id: ObjectId('692671f89e21211ca1f75259'),
    userId: ObjectId('692671f89e21211ca1f71df3'),
    type: 'Lunch',
    calories: 752,
    macros: { carbs: 142, protein: 33, fats: 35 },
    date: ISODate('2025-11-16T03:20:24.775Z'),
```

```
  notes: 'Light snack'
},
{
  _id: ObjectId('692671f89e21211ca1f7527f'),
  userId: ObjectId('692671f89e21211ca1f71df3'),
  type: 'Dinner',
  calories: 348,
  macros: { carbs: 44, protein: 21, fats: 8 },
  date: ISODate('2025-11-14T03:20:24.775Z'),
  notes: 'Healthy meal'
},
{
  _id: ObjectId('692671f89e21211ca1f752a6'),
  userId: ObjectId('692671f89e21211ca1f71df3'),
  calories: 273,
  date: ISODate('2025-11-01T03:20:24.775Z')
},
{
  _id: ObjectId('692671f89e21211ca1f75271'),
  userId: ObjectId('692671f89e21211ca1f71df3'),
  calories: 547,
  date: ISODate('2025-10-19T03:20:24.775Z')
},
{
  _id: ObjectId('692671f89e21211ca1f75263'),
  userId: ObjectId('692671f89e21211ca1f71df3'),
  type: 'Dinner',
  calories: 785,
  macros: { carbs: 167, protein: 42, fats: 39 },
  date: ISODate('2025-10-15T03:20:24.775Z'),
  notes: 'Fast food'
},
{
  _id: ObjectId('692671f89e21211ca1f7528c'),
  userId: ObjectId('692671f89e21211ca1f71df3'),
  type: 'Breakfast',
  calories: 464,
  macros: { carbs: 77, protein: 51, fats: 19 },
  date: ISODate('2025-10-10T03:20:24.775Z'),
  notes: 'Healthy meal'
},
{
  _id: ObjectId('692671f89e21211ca1f75277'),
  userId: ObjectId('692671f89e21211ca1f71df3'),
  calories: 439,
  date: ISODate('2025-10-04T03:20:24.775Z')
},
{
  _id: ObjectId('692671f89e21211ca1f7526e'),
  userId: ObjectId('692671f89e21211ca1f71df3'),
  type: 'Snack',
  calories: 507,
  macros: { carbs: 80, protein: 48, fats: 20 },
  date: ISODate('2025-09-27T03:20:24.775Z'),
  notes: 'Healthy meal'
},
{
```

```
  _id: ObjectId('692671f89e21211ca1f752a1'),
  userId: ObjectId('692671f89e21211ca1f71df3'),
  type: 'Dinner',
  calories: 791,
  macros: { carbs: 138, protein: 43, fats: 38 },
  date: ISODate('2025-09-20T03:20:24.775Z'),
  notes: 'Light snack'
 }
]
```

```
db.Nutrition.find({
...     "type": { $in: ["Lunch", "Dinner"] },
...     "calories": { $gt: 500 }
... }).sort({"date": -1}).limit(10);
[
 {
  _id: ObjectId('692675309e21211ca1f7559b'),
  userId: ObjectId('692675309e21211ca1f75599'),
  type: 'Lunch',
  calories: 600,
  macros: { carbs: 80, protein: 25, fats: 20 },
  date: ISODate('2025-11-26T03:34:08.206Z'),
  notes: 'Added extra protein'
 },
 {
  _id: ObjectId('692671f89e21211ca1f7496f'),
  userId: ObjectId('692671f89e21211ca1f71ddb'),
  type: 'Dinner',
  calories: 647,
  macros: { carbs: 124, protein: 26, fats: 24 },
  date: ISODate('2025-11-26T03:20:24.734Z'),
  notes: 'Light snack'
 },
 {
  _id: ObjectId('692671f89e21211ca1f73c5d'),
  userId: ObjectId('692671f89e21211ca1f71db6'),
  type: 'Dinner',
  calories: 780,
  macros: { carbs: 116, protein: 81, fats: 26 },
  date: ISODate('2025-11-26T03:20:24.676Z'),
  notes: 'Protein shake'
 },
 {
  _id: ObjectId('692671f89e21211ca1f75235'),
  userId: ObjectId('692671f89e21211ca1f71df2'),
  type: 'Dinner',
  calories: 699,
  macros: { carbs: 125, protein: 64, fats: 21 },
  date: ISODate('2025-11-25T03:20:24.773Z'),
  notes: 'Fast food'
 },
 {
  _id: ObjectId('692671f89e21211ca1f75226'),
  userId: ObjectId('692671f89e21211ca1f71df2'),
  type: 'Lunch',
  calories: 601,
  macros: { carbs: 89, protein: 46, fats: 20 },
```

```
  date: ISODate('2025-11-25T03:20:24.773Z'),
  notes: 'Fast food'
 },
 {
  _id: ObjectId('692671f89e21211ca1f73ee6'),
  userId: ObjectId('692671f89e21211ca1f71dbe'),
  type: 'Dinner',
  calories: 601,
  macros: { carbs: 105, protein: 31, fats: 21 },
  date: ISODate('2025-11-25T03:20:24.686Z'),
  notes: 'Light snack'
 },
 {
  _id: ObjectId('692671f89e21211ca1f73906'),
  userId: ObjectId('692671f89e21211ca1f71dae'),
  type: 'Dinner',
  calories: 780,
  macros: { carbs: 139, protein: 55, fats: 27 },
  date: ISODate('2025-11-25T03:20:24.659Z'),
  notes: 'Fast food'
 },
 {
  _id: ObjectId('692671f89e21211ca1f74ec9'),
  userId: ObjectId('692671f89e21211ca1f71dea'),
  type: 'Dinner',
  calories: 749,
  macros: { carbs: 125, protein: 35, fats: 17 },
  date: ISODate('2025-11-24T03:20:24.759Z'),
  notes: 'Healthy meal'
 },
 {
  _id: ObjectId('692671f89e21211ca1f74ef7'),
  userId: ObjectId('692671f89e21211ca1f71dea'),
  type: 'Lunch',
  calories: 695,
  macros: { carbs: 73, protein: 32, fats: 25 },
  date: ISODate('2025-11-24T03:20:24.759Z'),
  notes: 'Fast food'
 },
 {
  _id: ObjectId('692671f89e21211ca1f745c2'),
  userId: ObjectId('692671f89e21211ca1f71dd1'),
  type: 'Dinner',
  calories: 510,
  macros: { carbs: 62, protein: 37, fats: 21 },
  date: ISODate('2025-11-24T03:20:24.718Z'),
  notes: 'Protein shake'
 }
]
—
db.Places.find({
...     "type": "Gym",
...     "location": {
...       $near: {
...         $geometry: { "type": "Point", "coordinates": [-123.0990, 49.2825] },
...         $maxDistance: 5000
...       }
```

```
...     }
... }).limit(3);
[
  {
    _id: ObjectId('692671f89e21211ca1f7556c'),
    name: 'IronWorks Gym',
    type: 'Gym',
    address: { street: '12 Powell St', city: 'Vancouver', country: 'Canada' },
    location: { type: 'Point', coordinates: [ -123.099, 49.2825 ] }
  },
  {
    _id: ObjectId('692675309e21211ca1f7559c'),
    name: 'Central Fitness Center',
    type: 'Gym',
    address: { street: '123 Main St', city: 'Vancouver', country: 'Canada' },
    location: { type: 'Point', coordinates: [ -123.115, 49.28 ] }
  },
  {
    _id: ObjectId('692671f89e21211ca1f75571'),
    name: 'North Van Powerhouse',
    type: 'Gym',
    address: {
      street: '123 Bewicke Ave',
      city: 'North Vancouver',
      country: 'Canada'
    },
    location: { type: 'Point', coordinates: [ -123.084, 49.316 ] }
  }
]
```

AGGREGATION:

```
const fourteenDaysAgo = new Date();
... fourteenDaysAgo.setDate(fourteenDaysAgo.getDate() - 14);
...
... db.Nutrition.aggregate([
...     {
...         $match: {
...             "userId": ObjectId('692671f89e21211ca1f71d98'), //Tina Anderson
...             "date": { $gte: fourteenDaysAgo, $lte: new Date() } // Last 14 days
...         }
...     },
...     {
...         $group: {
...             _id: null,
...             totalCalories: { $sum: "$calories" } // Sum up the calories that alice had into "totalCalories"
...         }
...     },
...     {
...         $project: { _id: 0, totalCalories: 1 } // Show just the calories
...     }
... ]);
[ { totalCalories: 1095 } ]
```

—

```
const thirtyDaysAgo = new Date();
... thirtyDaysAgo.setDate(thirtyDaysAgo.getDate() - 30);
...
... db.Workouts.aggregate([
...     {
...         $match: {
...             "userId": ObjectId('692671f89e21211ca1f71dbe'), // Jack Anderson
...             "date": { $gte: thirtyDaysAgo, $lte: new Date() }
...         }
...     },
...     {
...         $group: {
...             _id: "$type", // Group by the workout type
...             totalDuration: { $sum: "$duration" }, // Sum up the duration in time
...             totalLength: { $sum: "$length" },     // Sum up the lenght in distance
...             count: { $sum: 1 }                    // Count how many workouts of this type were done
...         }
...     },
...     {
...         $project: {
...             _id: 0,
...             workoutType: "$_id",
...             totalDuration_minutes: "$totalDuration",
...             totalLength_units: "$totalLength",
...             numberOfWorkouts: "$count"
...         }
...     }
... ]);
[
  {
    workoutType: 'Weights',
    totalDuration_minutes: 252,
    totalLength_units: 16,
    numberOfWorkouts: 3
  },
  {
    workoutType: 'Cardio',
    totalDuration_minutes: 60,
    totalLength_units: 2,
    numberOfWorkouts: 1
  },
  {
    workoutType: 'Other',
    totalDuration_minutes: 287,
    totalLength_units: 11,
    numberOfWorkouts: 3
  },
  {
    workoutType: 'Sport',
    totalDuration_minutes: 455,
    totalLength_units: 35,
    numberOfWorkouts: 6
  }
]
```

```
db.Nutrition.aggregate([
...     {
...         $group: {
...             _id: null, // All documents together
...             averageMealCalories: { $avg: "$calories" } // Calculate the average cals
...         }
...     },
...     {
...         $project: {
...             _id: 0,
...             averageMealCalories: { $round: ["$averageMealCalories", 0] } // Round to get an whole number
...         }
...     }
... ]);
[ { averageMealCalories: 499 } ]
```

Explanation

All of the queries have comments explaining what they aim to accomplish, the code is also simple enough for most readers with experience within mongoDB to be able to understand. The idea behind why these queries were chosen lies on two main pillars: showing the understanding for most of the operators taught during the lectures, and the use case of a health and wellness application. Variables were used to assist and eliminate the need to create the object and get the id before trying to create a query that then uses said id.

Performance Optimization

The explain plans and indexes used can be found by running the “queriesWithExplain.js” file. The file was created to declutter the already large report, any clarification for all the queries can be found through there.

Advanced Features Used

There were three advanced features attempted, and only one that made it to the final version. The three consisted of: GridFS, 2dSphere index, and a text search index. GridFS was scrapped before being attempted due to being outside the scope of the project during the given timeline. A picture property remains in the Nutrition collection for future implementation. The 2dSphere index was successfully implemented and can be found in use in the “PLACES QUERIES” part of the “basic_CRUD.js” file. It was used to calculate the closest gyms within a certain radius (five km in the query). The text search was removed due to slowing down the execution time of my queries that used them. The data set was five thousand Places with just the type as the one being searched for in the query, five thousand with the name, five thousand with both, and five thousand with neither, along with some places entries left over from earlier testing.

The following is the result of running the query in regex:

```
executionStats: {
  executionSuccess: true,
  nReturned: 20000,
  executionTimeMillis: 17,
  totalKeysExamined: 0,
  totalDocsExamined: 30000,
```

After adding a text index, the query provided the following result:

```
executionStats: {
  executionSuccess: true,
  nReturned: 20000,
  executionTimeMillis: 30,
  totalKeysExamined: 20000,
  totalDocsExamined: 20000,
```

Even though total documents examined went down, the execution time ramp-up pointed towards disregarding this index.

Results & Analysis

Findings & Insights

Throughout this project there were many interesting findings and insights, many of which became second nature and aren't thought twice about at this point. Some that stuck out to me were: how collation worked within indexes and finds, and how text search (among other queries) can assist or slow down the application depending on data.

Firstly, the presumption was that the collation index would then let the query ask for the email in using any combination of upper and lower case lettering. This was not the case and another collation was needed during the searches, the index was only there to enforce the

uniqueness during insertion operations. This can be seen in this query, first without collation:

```
mazurv_practice> db.Users.updateOne(
...     { email: "LIAM.tHOmas18@example.COM" },
...     { $min: { firstWorkoutDate: new Date() } }
...   );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
```

And then with:

```
mazurv_practice> db.Users.updateOne(
...   { email: "LIAM.tHOmas18@example.COM" },
...   { $min: { firstWorkoutDate: new Date() } },
...   { collation: { locale: "en", strength: 2 } }
... );
...
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Secondly, for this application and the data set used during testing (which can be found in "extras/testingTextIndex.js") the execution time slowed down. This was a surprise since it was assumed that the REGEX find would take up more resources. The revelation occurred that an index that works in one application might not work in the application in the same domain, or two different data sets. Therefore, each index must be made on a case by case basis.

Patterns / Trends / Anomalies

All data seems to be inserted properly and allows for the queries and aggregation to run. Trends such as the relationships between collections seem to be consistent and refer back to the proper user. No blatant anomalies detected from the testing done on the application. However, there are some users that are missing food for a range of days due to the way the populate script is set up to not overload the system.

Challenges & Solutions

Problems Encountered

The main challenges encountered during the development of SmartFit could be put into three different areas: shallow data, schema changing, and getting rid of the SQL ideas. Shallow data made it difficult to run all of the queries when some fields were missing, see the improvement / deterioration of performance before / after indexing was used, and the overall hassle of having to add more data throughout the project. The schema going through multiple reiterations meant that the queries had to do the same thing to match the newly made / deleted properties. Finally, getting away from the idea that everything has to be normalized, joined, and in a table proved to be rather difficult during the beginning stages of the project.

Addressing Problems

To address the problems a populate script, and schema .js files were introduced, and the production embraced mongoDB principles. The populate script quickly made sure that the application had enough (and different) data, playing around with the number of each collection, once settled, eliminating the problem of shallow data. Finally, understanding that things are the way they are in mongoDB is for a reason and embedding speeds up various queries significantly. Focusing on the performance of the program and putting what should be two tables in SQL into one using the polymorphic approach. One such example, was my initial idea of having two collections: one for weight lifting and one for cardio into one - Workouts, showed such understanding.

Conclusion

Summary

Overall, the dedication and effort poured into this project resulted in a final product of decent quality. While by no means anything extraordinary, the understanding of NoSQL concepts along with other database related learnings surely suffices the time provided over the course of the semester. Queries and indexing greatly improved from the previous deliverables and the application has completed the minimal goals that it set out to do, as well as some of the extending goals such as GeoJSON. The learning is evident and there now exists a deeper desire to continue the path that is database design and development.

Lessons Learned / Improvements

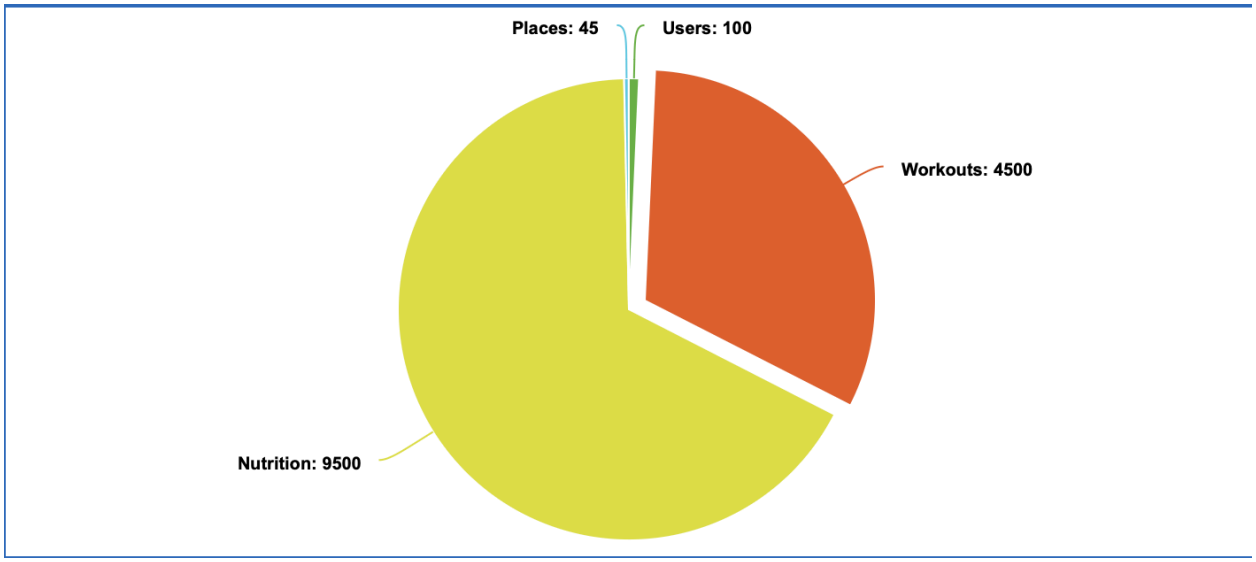
First things first, the biggest lesson taken away from this project is to ask the client for the queries as that would save lots of time on rebuilding the schema over and over again. The flexibility of mongoDB lies in the fact that it is very easy to add more properties and knowing the queries from the get go can help the developer start from a more optimized spot. Second, the quickest improvement and easiest way to see the indexes in action was after lining up all the queries and putting how often they are run. Once they are ranked, indexing on those that are used the most often (or have the most data oftentimes) proved to provide a quicker execution most of the time. Lastly, the lesson that data is king and should not only be in great abundance but also valid data with a mix of required and not required fields. Once the data set was well… set, the rest of production witnessed a great speed up - since the queries were now testable. Prior to that, each query required manual inserts to get enough data to test said query, with the populate script - the application now had enough data to not need manual insertions.

Improvements that could be added to the project would be having some sort of event that would fill the recent arrays in the User collection, adding GridFS for media, and streamlining userId and placeId referencing within the Workouts and Nutrition collections. Something mentioned during the presentation “Clean up the .js files” was completed before the final submission. Currently the recent arrays are empty since the only way to fill them would be manually, if a GUI was attached, on an event such as every workout and meal log the application could update said arrays. GridFS would not be a hard edition and could pose to be a good sweetener of the program, but was not required and out of scope for this iteration. Finally, another problem that a GUI would help with is referencing the user and place id within Workouts and Nutrition. Having a session that keeps track of the current user's id and automatically adds it during the logging of the meal or workout, and a variable that keeps the info of the place to add if the user decides to include the location at which they worked out at.

References

Dataset Sources

Data for users, workouts, and nutrition collections were all randomly generated from either a certain value or array of plausible entries created by the developer Vladislav Mazur. Places were manually inputted to authenticate each one, all from in and around the greater Vancouver area.



Documentation / Tutorials / Libraries

Most of the documentation is provided in this report, along with code comments and the README that accompanies the project. Tutorials were sourced from mongodb.com and their documentation; as well as the works of professor Kawal Jeet in the fall 2025 CSCI 485 class and lecture slides. The math library was used for the randomization of the provided data set in the application.