# MongoDB

# Clone Recitation 4 Repo

https://github.com/61040-fa23/rec-4

# Walk through
# Prep solution

# Motivation

- How can we create a robust web application?
  - Problems: server shutdowns/crashes, user clearing of browser cache, etc.
- How can we store data persistently?
- Other considerations for data storage:
  - Modifiability: create/modify/delete data concurrently
  - Extensibility: easily redefine data attributes for organization
  - Searchability: quickly find and deliver data in a usable format

# Document-Oriented Databases

- Subset of NoSQL databases
- Stores all information for a given object in a single **document** in the database
- What is a document?
  - A data structure composed of field and value pairs
  - MongoDB documents are similar to JSON objects
    - The values of fields may include other documents, arrays, and arrays of documents.

# Schemas/Models

## user.ts

```ts
export interface UserDoc extends BaseDoc {
  username: string;
  password: string;
}
```

# _id

```ts
/**
 * Add `item` to the collection. Returns the _id of the inserted document.
 */
async createOne(item: Partial<Schema>): Promise<ObjectId> {
  this.sanitizeItem(item);
  item.dateCreated = new Date();
  item.dateUpdated = new Date();
  return (await this.collection.insertOne(item as OptionalUnlessRequiredId<Schema>)).insertedId;
}
```

# ObjectId

```
export interface PostDoc extends BaseDoc {
  author: ObjectId;
  content: string;
}
```

# Data Modeling Exercise

```
concept Label[Item, User]

  state
    name: Label -> one String
    owns: User -> set Label
    labels: Item -> set Label
  actions
    createLabel (s: String, u: User, out p: Label)
      no u: User | p in u.owns // p is fresh, not owned by existing user; or "no p.~owns" or "p.~owns = {}"
      p.name := s; // give p the name s
      u.owns += p // add p to labels owned by u

    getUserLabels(u: User, out ps: set Label)
      ps := u.owns // ps is labels that u owns

    assignLabel (i: Item, p: Label)
      i.labels += p // add p to labels for i

    deleteLabel (p: Label)
     // remove p from all relations
     p.name = none
     p.~owns := none // ~owns is owns backwards
     p.~labels := none
```

# Filtering

# Running Example

```
const myDB = client.db("myDB");
const myColl = myDB.collection("fruits");
await myColl.insertMany([
{ "_id": 1, "name": "apples", "qty": 5, "rating": 3 },
{ "_id": 2, "name": "bananas", "qty": 7, "rating": 1, "color":
"yellow" },
{ "_id": 3, "name": "oranges", "qty": 6, "rating": 2 },
{ "_id": 4, "name": "avocados", "qty": 3, "rating": 5 },
]);
```

# Literal Value Queries

```
const query = { "name": "apples" };
const cursor = myColl.find(query);
for await (const doc of cursor) {
console.dir(doc);
}
```

```
{ "_id": 1, "name": "apples", "qty": 5, "rating": 3 }
```

# Comparison Operators

```
// $gt means "greater than"
const query = { qty: { $gt : 5 } };
const cursor = myColl.find(query);
for await (const doc of cursor) {
console.dir(doc);
}
{ "_id": 2, "name": "bananas", "qty": 7, "rating": 1 }
{ "_id": 3, "name": "oranges", "qty": 6, "rating": 2 }
```

https://www.mongodb.com/docs/manual/reference/operator/query-comparison/

# Logical Operators

```
const query = { qty: { $not: { $gt: 5 }}};
const cursor = myColl.find(query);
for await (const doc of cursor) {
console.dir(doc);
}
```

```
{ "_id": 4, "name": "avocados", "qty": 3, "rating": 5 }
{ "_id": 1, "name": "apples", "qty": 5, "rating": 3 }
```

https://www.mongodb.com/docs/manual/reference/operator/query-logical/

# Evaluation Operators

```
// $mod means "modulo" and returns the remainder after division
const query = { qty: { $mod: [ 3, 0 ] } };
const cursor = myColl.find(query);
for await (const doc of cursor) {
console.dir(doc);
}
```

```
{ "_id": 3, "name": "oranges", "qty": 6, "rating": 2 }
{ "_id": 4, "name": "avocados", "qty": 3, "rating": 5 }
```

https://www.mongodb.com/docs/manual/reference/operator/query-evaluation/

# Filtering Resources

https://www.mongodb.com/docs/drivers/node/current/fundamentals/crud/query-document/#std-label-node-fundamentals-query-document

# Filtering Exercise

# Cursor Methods

# options? in doc.ts

```ts
/**
 * Read the document that matches `filter`. Returns `null` if no document matches.
 */
async readOne(filter: Filter<Schema>, options?: FindOptions): Promise<Schema | null> {
  this.sanitizeFilter(filter);
  return await this.collection.findOne<Schema>(filter, options);
}
```

# sort()

```
async read(query: Filter<PostDoc>) {
  const posts = await this.posts.readMany(query, {
    sort: { dateUpdated: -1 },
  });
  return posts;
}
```

https://www.mongodb.com/docs/manual/reference/method/cursor.sort/

# Cursor Methods Exercise

# Testing!

# Testing Exercise

# Wrap Up

```
  private async canSendRequest(u1: ObjectId, u2: ObjectId) {
    await this.isNotFriends(u1, u2);
    // check if there is pending request between these users
    const request = await this.requests.readOne({
      from: { $in: [u1, u2] },
      to: { $in: [u1, u2] },
      status: "pending",
    });
    if (request !== null) {
      throw new FriendRequestAlreadyExistsError(u1, u2);
    }
  }
}
```

https://www.mongodb.com/docs/manual/reference/operator/query/in/