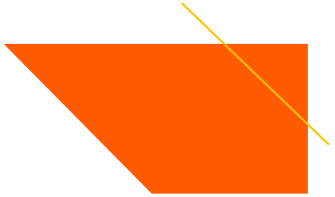
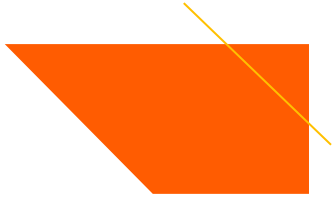
Web Application Development

COSI 152A

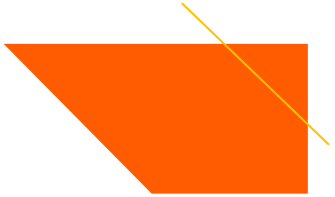


Main Point Preview

- In this lesson, you will:
  - set up MongoDB, the database system that you'll use to store persistent data.
  - explore what makes MongoDB particularly convenient for Node.js applications.
  - have a database set up and connected to your application.

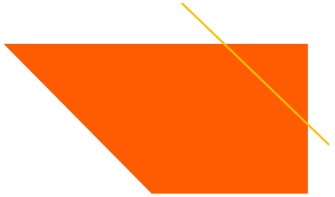


MongoDB



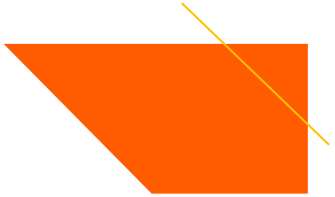
Why Database?

- Storing data is the most important part of application development.
  - Without long-term storage the data disappear.
  - Imagine a social network where the data disappear every time a user close the browser.
- A database help us to save information for a long-term.
  - Our data will persist, even if the application is shut down.
- A database is an organization of your data
  - Provide easy access and efficient changes.
- A database is like a warehouse:
  - Easy to find an item with an organized system withe more items.



MongoDB

- MongoDB is a popular open-source NoSQL database management system
  - designed to store, retrieve, and manage data in a flexible and scalable manner.
- Here are some key features and concepts associated with MongoDB
  - Document-Oriented
  - Collections
  - Flexible Schema



Document-Oriented

- MongoDB is a document database:
  - stores data in JSON-like documents.
- Each document is a collection of key-value pairs

```
{
"title": "The Pragmatic Programmer",
"author": "Andrew Hunt, David Thomas",
"publication_year": 1999,
"ISBN": "978-0201616224"
}
```

Collections

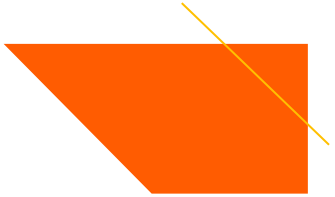
- Documents are organized into collections, which are like the tables in relational databases.
- Collection: books

```
{
"title": "Introduction to Algorithms",
"author": "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein",
"publication_year": 2009,
"ISBN": "978-0262033848"
},
{
"title": "The Pragmatic Programmer",
"author": "Andrew Hunt, David Thomas",
"publication_year": 1999,
"ISBN": "978-0201616224"
}
```

Flexible Schema

- MongoDB documents within a collection can have varying structures.
  - you can have documents with different fields, and fields within a document can have different data types.
  - unlike traditional relational databases, where each table has a fixed schema.

```
{
  "_id": 1,
  "name": "John Doe",
  "email": "john@example.com",
  "age": 30,
  "address": {
      "street": "123 Main St",
      "city": "New York"
  },
  "interests": ["programming", "hiking", "photography"]
}
```



Why MongoDB?

- Unlike relational databases used by most applications, MongoDB's nonrelational database system leads the Node.js application community.
- It is possible to set up a relational database with Node.js but the MongoDB query language is simpler to understand for people who have a Java-Script background.

Starting MongoDB Server

- Having MongoDB installed on your computer, start it by following command:

```
brew services start mongodb (mac)
```

```
mongod (windows)
```

- To test, run:

```
mongo        or        mongosh
```

  - this will bring up the MongoDB shell where you can use MongoDB

MongoDB Shell and Commands

- Now MongoDB is ready to receive commands to add, view, delete, or otherwise change data.
- To list your current database, enter: `db` (You are inside test database if it is your fist time).
- To create a new database and switch into it, run: `use <new_db_name>`

  Example: `use recipe_db`
- To show list of all databases, run: `show dbs`

You won't see your new database in the list of databases until data is added

MongoDB Shell and Commands

- To add data to your database, specify a collection name with which that data is associated.
- Adding data to the books:

```
db.books.insertOne({
  title: "The Catcher in the Rye",
  author: "J.D. Salinger",
  publication_year: 1951,
  ISBN: "978-0316769174"
})
```

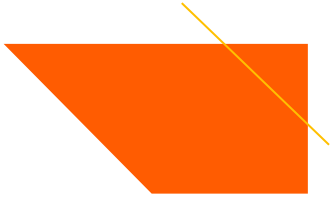
- The insert method runs on a MongoDB collection to add elements of a JavaScript object to a new document.

MongoDB Shell and Commands

- There’s no strict collection structure; you can add any values to new documents without needing to follow previous data patterns.

```
db.books.insertOne ({
first_name: "Jon",
favoriteSeason: "spring",
countries_visited: 42
})
```

- don’t insert inconsistent data, it is a bad habit



MongoDB Shell and Commands

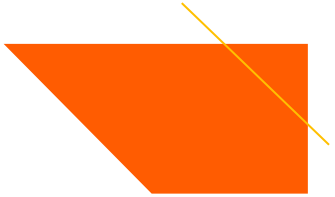
- To list the collection's contents, you can run:

```
db.<collection_name>.find()
```

e.g.

```
db.books.find()
```

- You should see a response with inserted items and with an extra "_id" property added by MongoDB.
  - The id property stores a unique value
  - Use it to differentiate and locate specific items in your database.



MongoDB Shell and Commands

- MongoDB uses an ObjectId class to record some meaningful information about its database documents.
- ObjectId("5941fe7acda203f026856a5e"), for example, constructs a new ObjectId representing a document in your database.
- The hexadecimal value passed into the ObjectId constructor references the document, a timestamp of the record's creation, and some information about your database system.
- The resulting ObjectId instance provides many useful methods that you can use to sort and organize data in your database.

MongoDB Shell and Commands

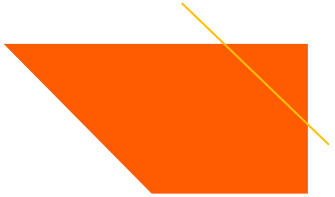
- Try searching for a specific item in the books collection by entering:
  - Replace the ObjectId in this example with one from your own database results.

```
db.books.find({_id: ObjectId(”<5941fce5cda203f026856a5d>")})
```

MongoDB Shell and Commands

- You can use many MongoDB commands. Table below lists a few that you should know about.

Command	Description
`show collections`	Displays all the collections in your database. Later, these collections should match your models.
`db.contacts.findOne`	Returns a single item from your database at random or a single item matching the criteria passed in as a parameter, which could look like findOne({name: 'Jon'}).
`db.contacts.update({name: "Jon"}, {name: "Jon Wexler"})`	Updates any matching documents with the second parameter's property values.
`db.contacts.delete({name: "Jon Wexler"})`	Removes any matching documents in the collection.
`db.contacts.deleteMany({})`	Removes all the documents in that collection. These commands can't be undone.



MongoDB Compass

- MongoDB's graphical user interface called MongoDB Compass
- MongoDB Compass is straightforward to use.
- To view the databases that you set up for your application, follow these steps:
  - Download the software from https://www.mongodb.com/download/compass.
  - Follow the installation steps to add MongoDB Compass to your applications folder.
  - Run MongoDB Compass and accept the default connection settings to your existing MongoDB setup.
  - See your databases listed with options to view the collections and documents within them.

Connecting MongoDB to Your Application

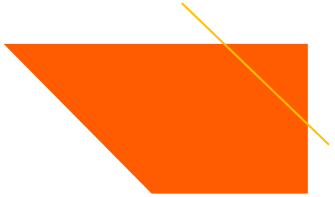
- To add MongoDB to your Node.js application, navigate to your project folder in terminal, and install the mongodb package by running:

```
npm i mongodb@3.6.3 -S
```

- At the top of your index.js file, add the code in the next slide

Connecting MongoDB to Your Application

```
const MongoClient = require("mongodb").MongoClient;

const url = "mongodb://localhost:27017/the_kitchen";

MongoClient.connect(url, (err, client) => {

  if (err) {

    console.error("Failed to connect to MongoDB:", err);

    return;}

  const db = client.db();

  const collection = db.collection("books");

collection.find({}).toArray((err, documents) => {

    if (err) {

      console.error("Error finding documents:", err);

    } else {

      console.log("Found documents:", documents);  }

  });

});
```



Thank You!