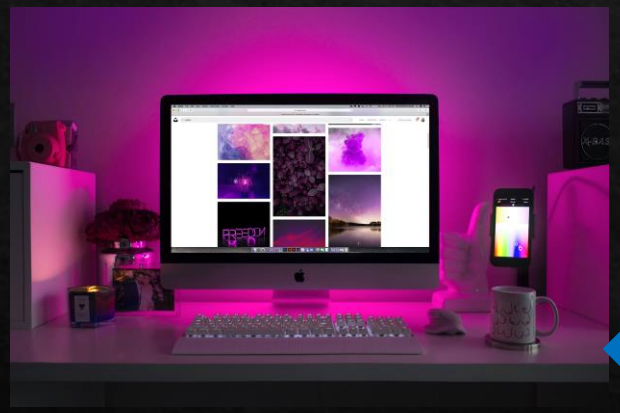
MongoDB

Document Databases

Behind the Scenes

- When a User opens a web page, What really happens??
- First:
  - There is communications between the client and the server
  - Communication is base on HTTP using POST, GET, PUT, DELETE
- Second:
  - There is communications between the server and the database
  - This is done via CRUD, CREATE, READ, UPDATE, DELETE

Behind the Scenes



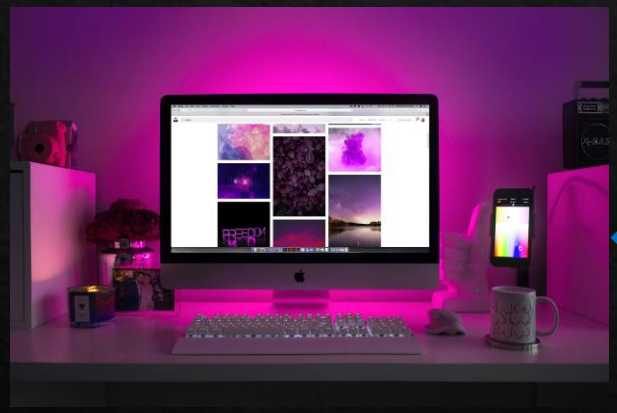
POST
GET
PUT
DELETE

CREATE
READ
UPDATE
DELETE

Behind the Scenes

- Ex 1)
- Lets say you want to open a blog:
  - The client uses the GET to get the blog from the server
  - The server then send a request to the database using the READ method
- Ex 2)
- The user wants to update their password
  - Client uses PUT to contact the server
  - Then the server contacts the database with UPDATE

Behind the Scenes



What is MongoDB

- MongoDB:
  - A NoSQL database system
  - Non-Relational
  - document oriented vs the Tables we have come to know and love
  - The Documents are JSON-like files called BSON which means Binary JSON
  - Created in 2007

How Does it work?

- NoSQL databases are split into collections which are similar to tables
- Each collection stores data called a document
- A document is like a record in SQL
- Documents look like JSON or JavaScript Objects

Table Vs Document

CustomerID	FirstName	LastName	DOB
1	Joe	Smith	1964-05-22
2	Alex	Turner	1987-12-07
3	Sally	Harris	2001-04-30

CommentID	Date	Text	CustomerID
1	June -16	Hi!	1
2	Dec- 7	Life Joy!	3
3	Smarch-4	I'm Happy!	1

```
{
    _id: 1
    login: "Steve",
    password: "123",
    comments:[
                {
                    _id:54,
                    date:"01-01-2022",
                    text: "Hi!",
                },
                {
                    _id:56,
                    date:"06-06-22-022",
                    text: "I'm Happy",

                }
                 ]
}
```

Advantages

- When we add a Column(field) to records in a SQL database we have to add it to all records in the database
- When we add a new field to a record in a NoSQL database we can simply add it to the one person we may want to add it to

Advantages

CustomerId	FirstName	LastName	DOB	Photo
1	Joe	Smith	1964-05-22	Me.jpg
2	Alex	Turner	1987-12-07	Null
3	Sally	Harris	2001-04-30	Null

```
{
    _id:1
    login: “Steve”,
    password: “123”,
    comments:[
                {
                    _id:54,
                    date:”01-01-2022”,
                    text: ”Hi!”,
                },
                {
                    _id:56,
                    date:”06-06-22-022”,
                    text: ”I’m Happy”,

                }
                ],
    Photo:”me.jpg”
}
```

Install MongoDB

- Go to https://www.mongodb.com/
- mongoDB Community Server https://www.mongodb.com/try/download/community
- Select Download
- Open file
  - Next>agree>next>Complete>Next>next>install>Finish
- MongoDB is a CLI (Command Line Application) So you will need to use PowerShell

MongoDB On PowerShell

- Open PowerShell
- Open your GUI file explorer and navigate to:
- C:\Program Files\MongoDB\Server\5.0\bin find mongod.exe
- In your task bar open the search and type > view advanced system settings
- Click environment variables
- In system variables > path > double click
- New> paste> C:\Program Files\MongoDB\Server\5.0\bin
- Ok>ok>ok
- Run mongod on PowerShell, if you still get an error, restart computer

MongoDB On PowerShell

- When we run mongod, we will get an exit code shutting down with code 100
- This means that we don't have a database to serve anything to
- In PowerShell > mkdir C:\data\db
- MongoDB is now running on port 27017
- But it is waiting for a connection

MongoDB Compass

- Open MongoDB Compass
- Select Next if you have a getting started pop up
- Select Connect
- Click databases and you will see we have some default databases
- Create a new Database called Users
- Now inside the Users data base we have collections (like a table) called users

Connect Using VS code

- Create a new directory in your ITD Work Folder Called MongoDB
- Add a file called app.js
- Open with vs Code
- Close poweshell
- Open new terminal in vs code
- And run mongod
- Open a second terminal and install mongoose https://mongoosejs.com/docs/connections.html
- npm i mongoose

Connecting to Mongo via Mongoose

- First:
  - `const mongoose = require("mongoose")`
- This connects mongoose the file we are using
- Second:
  - `mongoose.connect("mongodb://localhost/users")`
  - This allows us to connect to the database we created in MongoDB compass
  - *Fun Fact* Connect is an asynchronous command which means it doesn't stop other commands while it connects to the database

Connecting to Mongo via Mongoose

- Then we add some error handling, this is similar to what we did with if err throw
- ```
  mongoose.connect("mongodb://localhost/users").then(() => {
  ```
- ```
    console.log("Connected to MongoDB...")
  ```
- ```
    }).catch((error) => {
  ```
- ```
      console.log("Error connecting to mongoDB" + error);
  ```
- ```
  });
  ```
  ```

Connecting to Mongo via Mongoose

```
mongoose.connect("mongodb://localhost/users",(err)=>{
    if(err){throw err}
    console.log("Connected to mongoDB")
});
```

How to Add Data Mongoose

- First Make a Schema by using mongoose
- Second we will create a Class, also called a Model
- Third we will create an object based on the class
- Fourth and final we will add the Object to the MongoDB database which will be stored as a document

What is a Schema?

- A schema is basically rules and structures that the data in our database has to follow
- In both SQL and NoSql This includes things like:
  - Relationships
  - Tables
  - Fields
  - Procedures
  - Indexes
  - Types
  - Etc...

What is a Schema?

- For example:
  - Lets say we have a document type database like MongoDB
  - In our documents we have

```
{ userId: 6,
 FirstName: “Steve”,
 LastName: “Buscemi”,
 Password: “123FakePass”
}
```

```
{
uId: 88,
 FN: ‘Bob’,
 LN: ‘Loblaws’,
 Pass: ‘LawBlog’
}
```

What is a Schema?

- This would cause issues because the data is stored under different identifiers
- If we instead do this:

  {

  userId: Number

  FirstName: string

  LastName: string

  Password: string

  }
- We can create a schema for our fields which will allow our data to be stored safely and consistently

Create A Schema

- First we start with :
- `const { Schema } = mongoose;`
- `This make a Schema Class from mongoose`

Create A Schema

- Second :

```
const usersSchema = new Schema({
    login:String,
    password:String,
    age:Number,
    student: Boolean,
    country: String,
});
```

Create A Schema

- `new Schema();` is a constructor for a class
- What is being passed through as an argument is a JS object with key value pairs {Key: “Value”}

Create A Schema

The object we passed through :{

```
    login:String,
    password:String,
    age:Number,
    student: Boolean,
    country: String,
}
```

ensures that when we create users, every users will have these specific fields and data types for more data types

https://mongoosejs.com/docs/schematypes.html

Additional Constraints

- Lets say we want to make sure people are inputting the correct data
- Age: may be < 0
- this isn't good because very few humans are -1 years old...
- To prevent this, we can have additional constraints in our schema
- First create a JS object as the vale of your key
- {age:

  {type: Number,

  min: 0

  }

  }

Additional Constraints

```
const usersSchema = new Schema({
    login:String,
    password:{
        type: String,
        minlength: 4
    },
    age:{
        type: Number,
        min: 0
    },
    student: Boolean,
    country: String,
});
```

In the Background: Classes

- In OOP, we use classes to create objects
- Objects that come from classes are different than JS objects
- JavaScript objects are more like dictionaries or hash tables because they use key value pairs
- Classes have similarities to JS objects, but are used more as collections of functions and data members, Which are just variables inside the class

Creating Class Objects

```
const User = mongoose.model('User', usersSchema)
^^ is a class that we can instantiate an object from
const user1 = new User({
    login:'Peter',
    password: '1234',
    age:42,
    student: false,
    country: "Canada"
})
```

CRUD

- As mentioned before CRUD is used for server to Database and
- POST, GET, PUT, DELETE is used to connect from client to server
- These two processes basically have the same end goals with different names

C -> Create

- To create a document we simply use the .save() method
- This will allow us to save an object as a document in out database
- Lets create a few documents...
- ```
  user1.save().then(()=>{
  ```
- ```
          console.log('Saved!')
  ```
- ```
  })
  ```
  ```

R - Read

- When we want to find data in our database we can use 5 different functions
  1. find()
  2. limit()
  3. sort()
  4. select()
  5. where()
     -

R - Read

- Find() allows us to locate ALL documents in the collections
  - We do this by making a function, we also need to make this function async which means we will wait till we get all the data before

```
const getUsers = async ()=>{
    const users = await User.find();
    console.log(users)
}
getUsers();
```

R - Read

- Limit()
- For this we just chain limit with find()

```
const getUsers = async () => {
  const users = await User.find().limit(2);
  console.log(users);
};
```

R - Read

- Sort()
  - This allows us to sort items so if we want to get the age of our users from min to max we do this:

```
const getUsers = async () => {
  const users = await User.find().limit(3).sort({age: 1});
  console.log(users);
};
```

  - To sort from max to min we change 1 to -1

R - Read

- Select()

```
const getUsers = async () => {
  const users = await User.find().limit(3).sort({age:
1}).select({login: true, password  : true
});
  console.log(users);
};
```

  - We can add select to our chain and find the data we want for that specific

R - Read

- Where()

Operator	Name	Mongoose Function
>	Greater than	.gt()
>=	Greater than or equal to	.gte()
<	Less than	.ls()
<=	Less than or equal too	.lse()
===	Equality	.equals()
!==	Inequality	.ne()

R - Read

- Where()
- ```
  const getUsers = async () => {
  ```
- ```
    const users = await
  User.find().where('age').gt(30).limit(3).sort({age:
  1}).select({login: true, password  : true});
  ```
- ```
    console.log(users);
  ```
- ```
  };
  ```
- ```
  getUsers();
  ```
  ```

Query

- For more query functions go to this link
- https://mongoosejs.com/docs/api/query.html

U - Update

- Three steps:
  1. Read data we want to update firstName = peter
  2. Change the document information password = 12345
  3. Save the information save peter

U - Update

- Long Way: We can use a findOne() or findById()

```
const updateUser =  async () => {
    const user = await User.findOne({login: "Nancy"});
    user.password = "99633"
   await user.save()
   console.log('Done’)
}
updateUser()
```

U - Update

- Short Way:

```
const updateUser =  async () => {
User.findOneAndUpdate({login: 'Nancy'}, {password: '65412'}, (err)=>{
    if(err) {throw err}
    console.log('Updated data’)

})
}
```

D - Delete

- Delete():
- ```
  const removeUser = async ()=>{
      const result = await User.deleteOne({login:'Nancy'})
      console.log(result)
  }
  ```
  ```

Connect a form to a database

- Connect a form you make to a database like you did with mysql