Basit Hussain

FC Computer Collegi@te

The Future Creator

Be a Professional by the Professionals...

Week 01

JAVASCRIPT

Deep Dive

Adding JS To a Web

Why Should you take this course?

- ✓ **JavaScript is a Universal Language**
  - ❑ Websites & Apps
  - ❑ Front-end
  - ❑ Server-Side Code (Node.js)
  - ❑ Programming Robots
  - ❑ & Much More

JavaScript on the Front-end

- ✓ JavaScript that runs in a browser
- ✓ Add Interactivity web pages
- ✓ Animations, Web forms, UI Element, Using API's & data
- ✓ Using Modern JavaScript syntax & techniques

Why JavaScript is Amazing

- ✓ Make awesome things in a browser
  - ❑ Digital Clocks
  - ❑ Quizzes
  - ❑ Chat rooms
  - ❑ Weather App
  - ❑ Games…

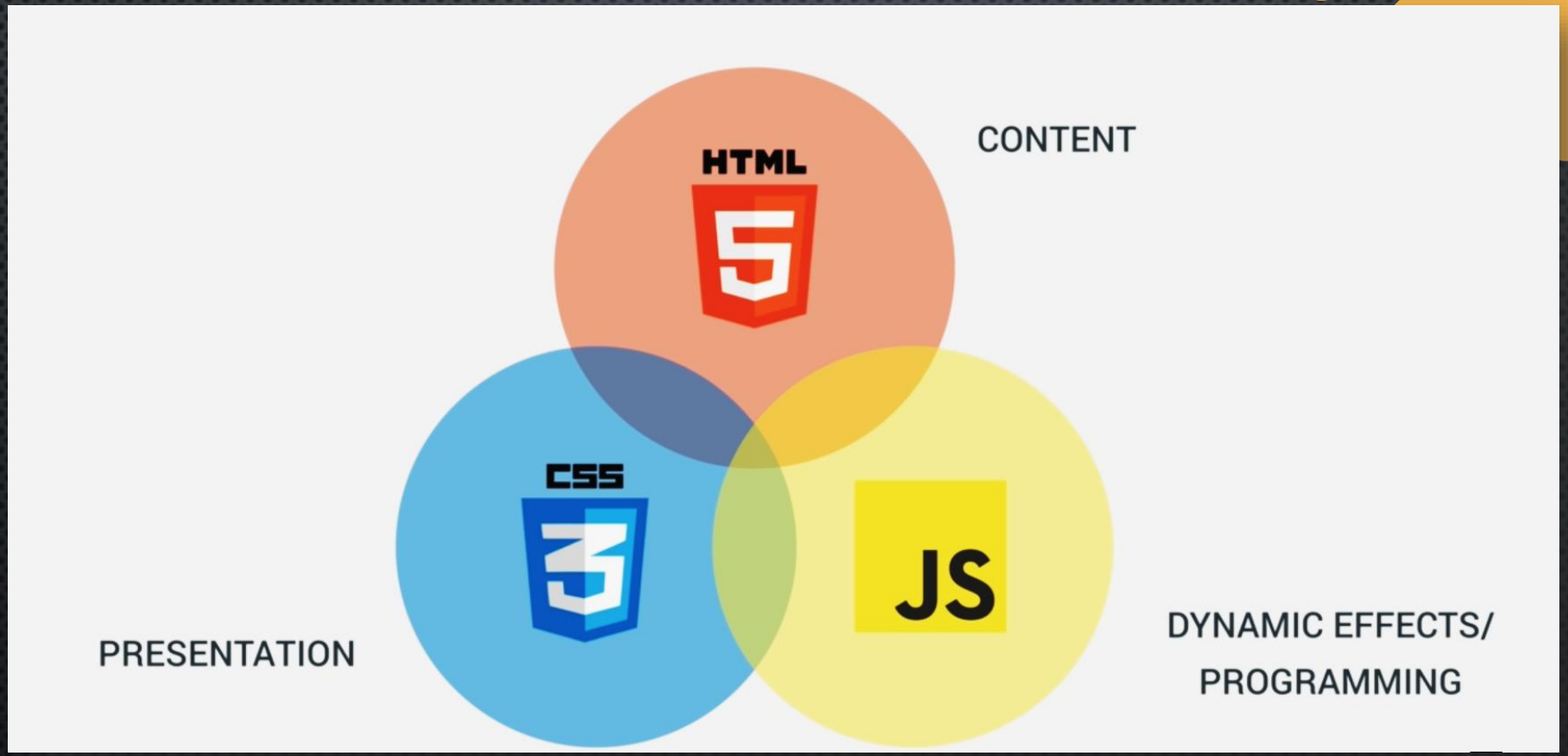
Setting up Your Environment

✓ Download Visual Studio Code

❑ Extensions

- Live Server
- Material Icon Theme
- Auto Close Tag
- Auto Import ES6, TS, JSX, TSX
- Auto Rename Tag
- Bracket Pair Colorizer
- HTML CSS Support
- JavaScript Snippets
- Prettier Code Formatter

Role of JavaScript



Chapter 02

- ✓ Adding JavaScript to a Web Page
- ✓ The Browser Console
- ✓ Data Types
  - ❑ String
  - ❑ Number
  - ❑ Boolean
  - ❑ Undefined
  - ❑ Null

Chapter 02

- **String:** Sequence of characters or text.
  - String Concatenation
  - Getting Characters
  - String Length
  - String Methods
    - toUpperCase()
    - toLowerCase()
    - indexOf()
    - lastIndexOf()
    - Slice(from where to start, form where to end)
    - Substr(from where to start, count the digits which you give then end)
    - Replace()
  - Template String

Chapter 02

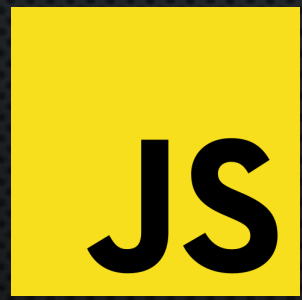
- **Number:** Floating point numbers, for decimal integers.
  - Operators
    - +, -, /, *, **, %
    - ++, --
    - +=, -=, *=, /=
    - NaN – not a number

Chapter 02

- **Boolean:** Logical data type that can only be true or false.
- **Undefined:** Data type of a variable that does not has value yet.
- **Null:** Also means ‘non-existent’.

Chapter 02

- ✓ **Array:**
  - ❑ Array Method:
    - join(,)
    - indexOf()
    - concat([])
    - push()
    - Pop()
    - unshift()
    - shift()



Chapter 02

- Type Conversion
- For Loop
  - for (variable initialize; condition; Increment / Decrement)

```
{
    Statement;
}
```

Chapter 02

While Loop

```
variable initialize;
while (condition) {
        Statement;
        Increment / Decrement;
}
```

Chapter 02

❑ do while Loop

```
variable initialize;
do {
        Statement;
        Increment / Decrement;
}
while (condition)
```

Chapter 02

- if statement

```
if  (condition) {
        Statement;
} else if (condition) {
        Statement;
} else {
        Statement;
}
```

Chapter 02

- Function Declaration

```
function name () {
    Statement;
}
```

- Function Expression

```
const name = function () {
    Statement;
}
```

Chapter 02

❑ Arrow Function

```
const name = () => {
    Statement;
}
```

Chapter 02

❑ Function Declaration

```
function name (Parameter01, Parameter02) {
    Statement;
}

name(Argument01, Argument02);
```

Chapter 02

❑ Function Expression

```
const name = function (Parameter01, Parameter02) {
    Statement;
}

name(Argument01, Argument02);
```

Chapter 02

❑ Arrow Function

```
const name = (Parameter01, Parameter02) => {
    Statement;
}

name(Argument01, Argument02);
```

Chapter 02

❑ Returning Values

```
const add = (a, b) => {
    let c = a + b;
    return c;
}

let sum = add(2, 2);

Console.log(sum);
```

Chapter 02

❑ Returning Value

```
275  const calcProd = function (products, tax) {
276    let total = 0;
277    for (let i = 0; i < products.length; i++) {
278      total += products[i] + products[i] * tax;
279    }
280    return total;
281  };
282
283  let productC = calcProd([20, 30, 40], 0.2);
284  console.log(productC);
```

Chapter 02

- setTimout(functionName, milliseconds)

```
//Set Time Out
setTimeout(() => {
  let employees = ["Faraz", "Aqib", "Shahab", "Ali"];
  console.log(employees);
}, 3000);
```

Chapter 02

❑ setTimout(functionName, milliseconds)

```
335  let box = document.querySelector("#box");
336
337  setTimeout(() => {
338    box.style.width = "500px";
339  }, 3000);
```

Chapter 02

- forEach

```
Let days = [Mon, Tues, Wed, Thurs, Fri];
days.forEach(function(day) {
    console.log(day);
}
```

Chapter 02

❑ forEach

```
324  list = document.querySelector(".people");
325
326  html = ``;
327
328  const students = ["Aqib", "Aqib", "Aqib", "Aqib"];
329
330  students.forEach((person) => {
331    html += `<li style="color: red">${person}</li>`;
332  });
333
334  list.innerHTML = html;
```

Chapter 02

❑ callback Function

- ❑ Any function that is passed as an argument is called callback function.

- ❑ A callback is a function that is to be executed after another Function has finished executing – hence the name ‘call back’.

Chapter 02

❑ Call Back Function

```
const Func = function(callBackFunc) {
    let value = 20;
    callBackFunc();
}
Func(function(value ) {
    console.log(value);
})
```

Chapter 02

❑ Call Back Function

```
312  const mesgOne = (day, time) => {
313    console.log(`Hello, Today is ${day}`);
314    time();
315  };
316
317  const mesgTwo = (time) => {
318    time = "Morning";
319    console.log(`Good ${time}!`);
320  };
321
322  mesgOne("Monday", mesgTwo);
```

Chapter 02

❑ JavaScript Objects

Objects in JavaScript have properties & things they can do (methods)…

Chapter 02

❑ JavaScript Objects

```
let user = {
  name: 'Basit Hussain',
  age: 21,
  email: 'basit@gmail.com',
  blogs: ['Lorem ipsum dolor sit, amet consectetur adipisicing elit. Voluptas, illo.', '
  location: 'North Karachi',
  login() {
    console.log('User is Log In');
  },
  logOut() {
    console.log('User is Log out');
  },
  blog() {
    console.log('Here are the blogs:');
    console.log('--------------------');
    this.blogs.forEach((blog) => {
      console.log(blog);
    })
  }
};
```

Chapter 02

- This keyword
- Objects in Array

```
381    blogs: [
382      {post: "Hello, Good day", likes: 30},
383      {post: "Hello, Good Night", likes: 20}]
```

Chapter 02

❑ Math Object

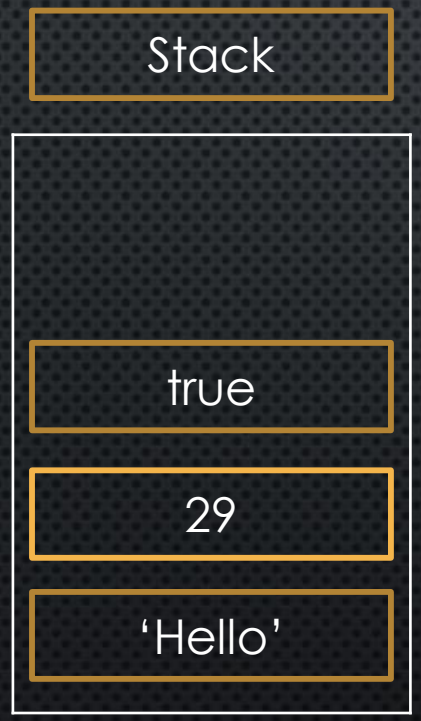
```
409  // Math Objects
410  console.log(Math);
411  console.log(Math.PI);
412  let area = 5.9;
413  console.log(Math.round(area));
414  console.log(Math.floor(area));
415  console.log(Math.ceil(area));
416  console.log(Math.trunc(area));
417
418  // RANDOM VALUES
419  let random = Math.random();
420  console.log(Math.round(random * 100));
```

Chapter 02

Primitive Types	Reference Types
Numbers	All types of objects
Strings	Object literals
Booleans	Arrays
Null	Function
Undefined	Dates
Symbols	All other objects

Chapter 02

❑ Stack & Heap



Chapter 02

- Interacting with a Browser
  - Document Object Model (DOM)
  - Add, change & delete content
  - Make a cool pop-up effect in the web page.

Chapter 02

- Query Selector
- document.querySelector('.Class, #ID, Selector')
- Query Selector All
- document.querySelectorAll('.Class, Selector')

Chapter 02

- Get an Element by ID
- document.getElementById(‘#id’)
- Get an Elements by their class
- document.getElementsByClassName(‘.class’)
- Get an Elements by their tag
- document.getElementsByTagName(‘tag name’)

Chapter 02

- Adding and Changing Page Content
- Text.innerText = ‘Hello World!’
- Text.innerText += ‘Hello World’

```
let content = document.querySelector(".content");
const name = ["Basit", "Aqib", "Hussain"];

name.forEach((a) => {
  content.innerHTML += `<h2>${a}</h2>`;
});
```

Chapter 02

❑ Getting & Setting Attributes

```
// Getting and Setting Attribute
const link = document.querySelector("a");
console.log(link.getAttribute("href"));
link.setAttribute("href", "https://www.google.com.pk");
link.innerText = "Google";

const para = document.querySelector("p");
console.log(para.getAttribute("class"));
para.setAttribute("class", "success");
para.setAttribute("style", "color: green");
```

Chapter 02

❑ Changing CSS Style

```
const ba = document.querySelector("p");
// ba.setAttribute("class", "success");
// ba.setAttribute("style", "color: green");

// ba.setAttribute("style", "color: red");
console.log(ba.style);
ba.style.marginTop = "50px";
ba.style.color = "red";
ba.style.backgroundColor = "lightGreen";
```

Chapter 02

❑ Adding and Removing Class

```
const para = document.querySelectorAll("p");

// console.log(para.classList);
// para.classList.add("error");
// para.classList.remove("error");
// para.classList.add("success");

para.forEach((p) => {
  //   console.log(p.textContent);
  if (p.textContent.includes("error")) {
    p.classList.add("error");
  } else if (p.innerText.includes("Success")) {
    p.classList.add("success");
  } else {
    p.classList.add("normal");
  }
});
```

Chapter 02

❑ Adding and Removing Class

```
const para = document.querySelector(".title");
console.log(para.classList.toggle("test"));
para.classList.toggle("test");
```

Chapter 02

❑ Parents, Children & Siblings

```
const title = document.querySelector("article");
// console.log(title.children);
// console.log(Array.from(title.children));

// Array.from(title.children).forEach((child) => {
//   child.classList.add("hello");
//   child.classList.remove("hello");
// });

const t1 = document.querySelector("h2");
console.log(t1.nextElementSibling);
console.log(t1.parentElement);
console.log(t1.previousElementSibling);
console.log(t1.parentElement.parentElement);
console.log(t1.nextElementSibling.parentElement.children);
```

Chapter 02

❑ Event Basic

```
// Event Basics
// const button = document.querySelector("a");

// button.addEventListener("click", () => {
//   console.log("click times");
// });

const list = document.querySelectorAll("li");
list.forEach((item) => {
  item.addEventListener("click", (e) => {
    // console.log("clicked");
    // console.log(e.target);
    // console.log(item);
    e.target.style.textDecoration = "line-through";
  });
});
```

Chapter 04

❑ Filter Method

```
const arr = [20, 30, 40, 50, 60, 70, 90];

const filterScore = arr.filter((score) => {
  return score > 20;
});

console.log(filterScore);
```

Chapter 04

❑ Filter Method

```
const user = [
  { name: "Basit", premium: true },
  { name: "Ali", premium: false },
  { name: "Askari", premium: true },
  { name: "Aqib", premium: false },
  { name: "Saqib", premium: false },
];

const premiumUser = user.filter((user) => {
  return user.premium;
});

console.log(premiumUser);
```

Chapter 04

❑ Map Method:

```
const price = [20, 30, 40, 50, 60, 70, 80];

const salesPrice = price.map((price) => {
  return price / 2;
});

console.log(salesPrice);
```

Chapter 04

❑ Map Method:

```
const user = [
  { name: "Basit", fees: 30 },
  { name: "Ali", fees: 40 },
  { name: "Askari", fees: 50 },
  { name: "Aqib", fees: 60 },
  { name: "Saqib", fees: 80 },
];
const modifiedPrice = user.map((student) => {
  if (student.fees > 50) {
    return { name: student.name, fees: student.fees / 2 };
  } else {
    return student;
  }
});
console.log(modifiedPrice);
```

Chapter 04

❑ Reduce Method:

```
const score = [20, 30, 40, 50];
// 0 + 20 = 20;
// 20 + 30 = 50;
// 50 + 40 = 90;
// 90 + 50 = 130;
const result = score.reduce((acc, curr) => {
  return acc + curr;
}, 0);
console.log(result);
```

Chapter 04

❑ Reduce Method:

```
const score = [20, 30, 40, 50];

const result = score.reduce((acc, curr) => {
  if (curr > 30) {
    acc++;
  }
  return acc;
}, 0);
console.log(result);
```

Chapter 04

❑ Reduce Method:

```
const user = [
  { name: "Basit", fees: 30 },
  { name: "Ali", fees: 40 },
  { name: "Basit", fees: 50 },
  { name: "Aqib", fees: 60 },
  { name: "Basit", fees: 80 },
];
// 0 + 30 = 30;
// 30 + 50 = 80;
// 80 + 80 = 160
const result = user.reduce((acc, curr) => {
  if (curr.name === "Basit") {
    acc += curr.fees;
  }
  return acc;
}, 0);
console.log(result);
```

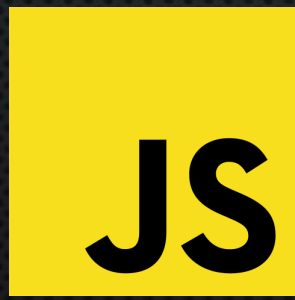
Chapter 04

❑ Find Method:

```
const price = [20, 30, 40, 20, 60, 70, 80];

const findPrice = price.find((p) => {
  return p > 40;
});

console.log(findPrice);
```



Chapter 04

❑ Sort Method:

```
const price = [20, 30, 40, 20, 50, 5, 60, 70, 80];
price.sort();
price.reverse();
console.log(price);
```

Chapter 04

❑ Sort Method:

```
const name = ["basit", "kashif", "ammar", "latif", "fatima"];
name.sort();
name.reverse();
console.log(name);
```

Chapter 04

❑ Sort Method:

```
const user = [
  { name: "Basit", fees: 30 },
  { name: "Ali", fees: 40 },
  { name: "Basit", fees: 5 },
  { name: "Aqib", fees: 20 },
  { name: "Basit", fees: 10 },
];

const a = user.sort((a, b) => {
  if (a.fees > b.fees) {
    return 1;
  } else if (b.fees > a.fees) {
    return -1;
  } else {
    return 0;
  }
});

console.log(a);
```

```
scores.sort((a,b) => a - b);
console.log(scores);
```

Chapter 04

❑ Change Array:

```
const users = [
  { name: "Basit", fees: 30 },
  { name: "Ali", fees: 40 },
  { name: "Basit", fees: 5 },
  { name: "Aqib", fees: 20 },
  { name: "Basit", fees: 10 },
];
const filteredPrice = users.filter((user) => {
  return user.fees > 20;
});
const modifiedFees = filteredPrice.map((user) => {
  return `The ${user.name} fees is ${user.fees}Rs.`;
});
console.log(modifiedFees);
```

Chapter 04

❑ Change Array:

```
const users = [
  { name: "Basit", fees: 30 },
  { name: "Ali", fees: 40 },
  { name: "Basit", fees: 5 },
  { name: "Aqib", fees: 20 },
  { name: "Basit", fees: 10 },
];

const changeArray = users
  .filter((user) => {
    return user.fees > 20;
  })
  .map((user) => {
    return `The ${user.name} fees is ${user.fees / 2}Rs.`;
  });

console.log(changeArray);
```

Chapter 05

❑ Dates (Object Data Type):

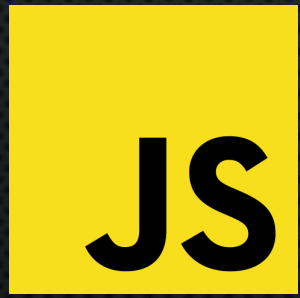
```
const now = new Date();
console.log(now);
console.log(typeof now);

//year, month, day and time
console.log("getFullYear: ", now.getFullYear());
console.log("getMonth: ", now.getMonth());
console.log("getDay: ", now.getDay());
console.log("getHours: ", now.getHours());
console.log("getMinutes: ", now.getMinutes());
console.log("getSeconds: ", now.getSeconds());
```

Chapter 05

❑ Dates (Object Data Type):

```
13  //Time Stamp
14  console.log("TimeStamp: ", new Date().getTime());
15
16  //Date String
17  console.log("DateString", now.toDateString());
18  console.log("TimeString", now.toTimeString());
19  console.log("LocaleString", now.toLocaleString());
```



Chapter 05

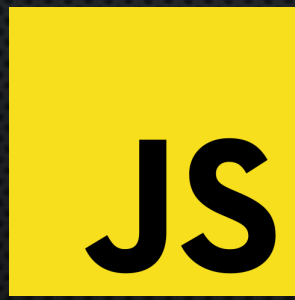
❑ TimeStamp (Difference)

```
21  //Time Stamp
22
23  const before = new Date("Wed Dec 02 2020 21:56:31");
24  const now1 = new Date();
25
26  const diff = now1.getTime() - before.getTime();
27  console.log(diff);
28  const mins = Math.round(diff / 1000 / 60);
29  const hours = Math.round(mins / 60);
30  const day = Math.round(hours / 24)
31  console.log(mins, hours, day);
```

Chapter 05

❑ Digital Clock

```
41  const tick = () => {
42    const now = new Date();
43    const h = now.getHours();
44    const m = now.getMinutes();
45    const s = now.getSeconds();
46
47    //   console.log(h, m, s);
48    const html = `
49   <span>${h}</span> :
50   <span>${m}</span> :
51   <span>${s}</span>
52   `;
53    clock.innerHTML = html;
54  };
55
56  setInterval(tick, 1000);
```



Chapter 06

❑ Synchronous Programming

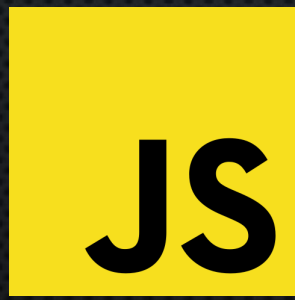
JS async.js > ...

```
1  // Sync Programming
2
3  function hello() {
4    console.log("Good Morning");
5    console.log("Good Night");
6  }
7
8  console.log("start");
9  hello();
10 console.log("end");
11
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
USER@DESKTOP-BHST7O7 MINGW64 ~/Desktop/Module I/js prac (master)
$ node async.js
start
Good Morning
Good Night
end
```

JavaScript can run ONE statement at a time



Chapter 06

❑ Asynchronous Programming

Start some thing now
and
Finish it later

```
JS async.js > funct
1  // Async Programming
2  function funct() {
3    console.log("Good Morning");
4    console.log("Good Night");
5  }
6  console.log("start");
7  setTimeout(funct, 2000);
8  console.log("end");
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
USER@DESKTOP-BHST7O7 MINGW64 ~/Desktop/Module I/js prac (master)
$ node async.js
start
end
Good Morning
Good Night
```

Chapter 06

❑ HTTP Request

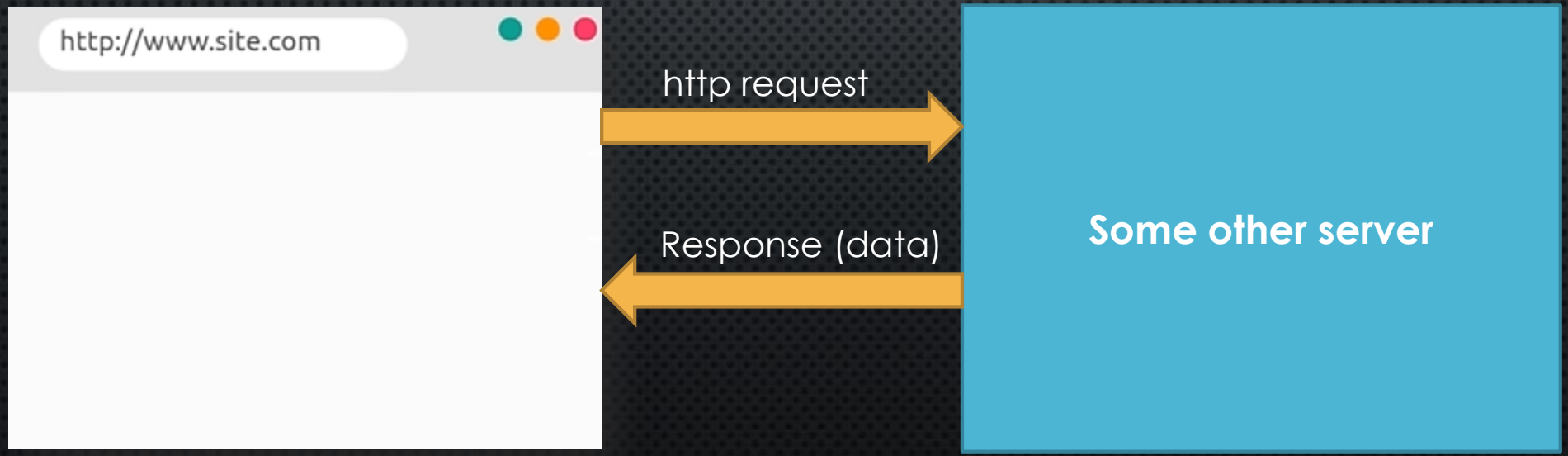
Make HTTP Requests to get the data from the server.

We make these request to API endpoints.

Chapter 06

❑ API Endpoints

Example API Endpoint:
http://www.musicapi.com/artist/atif



Chapter 06

What is AJAX

- AJAX is a Asynchronous JavaScript and XML
- AJAX is not a programming language. Rather it's set of existing technologies.
- AJAX helps fetching data asynchronously without interfering with the existing page.
- No page reload/ refresh
- Modern JS use JSON or XML for data transfer

Chapter 06

Why use AJAX?

- No page reload/ refresh
- Better user experience
- Saves network bandwidth
- Very interactive

Chapter 06

How it works?

- AJAX uses XMLHttpRequest Object (also called xhr object) to achieve this.
- Modern websites uses JSON or XML for data transfer.
- Data can be transferred in any format and protocol (Not always https necessarily)

Chapter 06

```
46  const fetchBtn = document.querySelector("#btnfetch");
47  const request = new XMLHttpRequest();
48
49  fetchBtn.addEventListener("click", () => {
50    console.log("click");
51    // console.log(request, request.readyState);
52    if (request.readyState === 4) {
53      if (request.status === 200) {
54        console.log(request.responseText);
55      }
56
57      if (request.status === 404) {
58        console.error("There is a server server");
59      }
60    }
61  });
62
63  request.open("GET", "basit.txt", true);
64  request.send();
```

Chapter 06

JS async.js > buttonClickHandler

```
1  const fetchBtn = document.querySelector("#btnfetch");
2
3  fetchBtn.addEventListener("click", buttonClickHandler);
4
5  function buttonClickHandler() {
6    console.log("Clicked");
7
8    // create xhr object
9    const xhr = new XMLHttpRequest();
10
11   // open the object
12   xhr.open("GET", "https://jsonplaceholder.typicode.com/todos/1", true);
13
14   //on progress
15   xhr.onprogress = function () {
16     console.log("on progress");
17   };
18
19   //on load
20   xhr.onload = function () {
21     console.log(this.responseText);
22   };
23
24   // send the request
25   xhr.send();
26
27   console.log("Request Done");
28 }
```

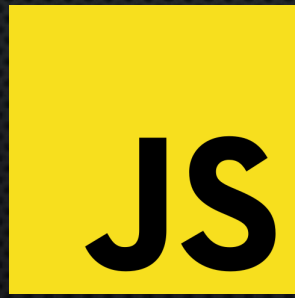
Chapter 06

```
const getTodos = (callBack) => {
  const req = new XMLHttpRequest();

  req.addEventListener("readystatechange", () => {
    if (req.readyState === 4 && req.status === 200) {
      callBack(undefined, req.responseText);
    } else if (req.readyState === 4) {
      callBack("Could not fetch data", undefined);
    }
  });

  req.open("GET", "./basit.txt");
  req.send();
};

getTodos((err, data) => {
  console.log("Fired");
  if (err) {
    console.log(err);
  } else {
    console.log(data);
  }
});
```



Chapter 06

- Convert JSON to JS Object

- `const data = JSON.parse(req.responseText)`

Chapter 06

```
17  getTodos("./json/basit.json", (err, data) => {
18    console.log("Fired");
19    console.log(data);
20    getTodos("./json/askari.json", (err, data) => {
21      console.log(data);
22      getTodos("./json/ali.json", (err, data) => {
23        console.log(data);
24      });
25    });
26  });
```

Chapter 06

❑ PROMISE BASICS

```
const getSomeThing = () => {
  return new Promise((resolve, reject) => {
    resolve("some data");
    reject("some error");
  });
};

// getSomeThing().then(
//   (data) => {
//     console.log(data);
//   },
//   (err) => {
//     console.log(err);
//   },
// );
```

Chapter 06

❑ PROMISE BASICS

```
44  getSomeThing()
45    .then((data) => {
46      console.log(data);
47    })
48    .catch((err) => {
49      console.log(err);
50    });
51
```

Chapter 06

```
const getTodos = (resource) => {
  return new Promise((resolve, rejects) => {
    const req = new XMLHttpRequest();

    req.addEventListener("readystatechange", () => {
      if (req.readyState === 4 && req.status === 200) {
        const data = JSON.parse(req.responseText);
        resolve(data);
      } else if (req.readyState === 4) {
        rejects("Could not fetch data");
      }
    });

    req.open("GET", resource);
    req.send();
  });
};
```

Chapter 06

```
19  getTodos("./json/ali.json")
20    .then((data) => {
21      console.log("Promise Resolved: ", data);
22    })
23    .catch((err) => {
24      console.log("Promise Rejected: ", err);
25    });
```

Chapter 06

```
19  getTodos("./json/ali.json")
20    .then((data) => {
21      console.log("Promise Resolved: ", data);
22      return getTodos("./json/askari.json");
23    })
24    .then((data) => {
25      console.log("Promise 1 Resolved: ", data);
26      return getTodos("./json/basit.json");
27    })
28    .then((data) => {
29      console.log("Promise 2 Resolved: ", data);
30    })
31    .catch((err) => {
32      console.log("Promise Rejected: ", err);
33    });
```

Chapter 06

- **Weather App**
- First Create out the Layout of the PROJECT
- Create New App
- Copy the key and paste it on forecast.js

```
const key = "Qy6yjd8XgVRvOHAcWrnR8jUqJVLIGhdy";
```

Chapter 06

❑ Weather App

```
12  const getCity = async (city) => {
13    const base = "http://dataservice.accuweather.com/locations/v1/cities/search";
14    const query = `?apikey=${key}&q=${city}`;
15
16    const response = await fetch(base + query);
17    const data = await response.json();
18
19    return data[0];
20  };
```

Chapter 06

❑ Weather App

```
3  const getWeather = async (id) => {
4    const base = "http://dataservice.accuweather.com/currentconditions/v1/";
5    const query = `${id}?apikey=${key}`;
6
7    const response = await fetch(base + query);
8    const data = await response.json();
9    console.log(data);
10 };
```

Chapter 06

❑ Weather App

```
22  getCity("karachi")
23    .then((data) => {
24      return getWeather(data.Key);
25    })
26    .then((data) => {
27      console.log(data);
28    })
29    .catch((err) => {
30      console.log("Error");
31    });
```

Chapter 06

❑ Weather App

```
1   const cityForm = document.querySelector("form");

13  cityForm.addEventListener("submit", (e) => {
14    e.preventDefault();
15
16    //get the city value
17
18    const city = cityForm.city.value.trim();
19    cityForm.reset();
20
21    updatecity(city)
22      .then((data) => {
23        console.log(data);
24      })
25      .catch((err) => {
26        console.log("There is an Error");
27      });
28  });
```

Chapter 06

❑ Weather App

```
3  const updatecity = async (city) => {
4    const citydet = await getCity(city);
5    const weather = await getWeather(citydet.Key);
6
7    return {
8      cityDet: citydet,
9      weather: weather,
10   };
11 };
```

Chapter 06

❑ Weather App

```
31  const card = document.querySelector(".card");
32  const details = document.querySelector(".details");
33
34  const updateUI = (data) => {
35    const cityDets = data.cityDets;
36    const weather = data.weather;
37
38    details.innerHTML = `
39              <h5 class="my-3">${cityDets.EnglishName}</h5>
40              <div class="my-3">${weather.WeatherText}</div>
41              <div class="display-4 my-4">
42                <span>${weather.Temperature.Metric.Value}</span>
43                <span>&deg;C</span>
44              </div>`;
45
46    if (card.classList.contains("d-none")) {
47      card.classList.remove("d-none");
48    }
49  };
```

Chapter 06

❑ Destructuring

```
let x, y;
[x, y] = [23, "Basit"];
console.log(x, y);

let [a, b, c, d, ...e] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 4, 2, 4, 5, 6, 7, 7, 8];
console.log(a);
console.log(b);
console.log(c);
console.log(d);
console.log(e);
```

Chapter 06

❑ Destructuring

```
12  ({ a, b, ...c } = { a: 12, b: 23, c: 24, d: 33, e: 34, f: 33 });
13  console.log(a, b, c);
14
15  const stud = ["Ali", "Basit", "Hussain"];
16  [a, b, c] = stud;
17  console.log(a, b, c);
```

Chapter 06

❑ Destructuring

```
19  let studDetails = {
20    studName: "Basit",
21    studAge: 21,
22    studEmail: "basit@gmail.com",
23    studFee: 2000,
24    studFunc() {
25      console.log("Hello Basit");
26    },
27  };
28
29  const { studName, studAge, studEmail } = studDetails;
30  console.log("Name: ", studName, " Age: ", studAge, " Email: ", studEmail);
```

Chapter 06

❑ Weather App

```
21  //update the night and day image
22  let imgSrc = null;
23  if (weather.IsDayTime) {
24    imgSrc = "../img/day.svg";
25  } else {
26    imgSrc = "../img/night.svg";
27  }
28  time.setAttribute("src", imgSrc);
29
30  // Update the weather icon
31  const iconSrc = `../img/icons/${weather.WeatherIcon}.svg`;
32  icon.setAttribute("src", iconSrc);
```

Chapter 06

❑ Application

❑ Set up database to store or retrieve data.

❑ Use local storage to store and retrieve data.

Chapter 06

❑ Local Storage

❑ Store data in local storage

❑ localStorage.setItem(‘key’, ‘value’)

❑ For Example

```
1  // store data in local storage
2  localStorage.setItem("name", "Basit");
3  localStorage.setItem("age", "21");
```

Chapter 06

- Local Storage

- Get data from local storage
  - localStorage.getItem('key')
  - For Example

```
5  //get data from local storage
6  let name = localStorage.getItem("name");
7  let age = localStorage.getItem("age");
8  console.log(name, age);
```

Chapter 06

- Local Storage
- Updating data of local storage
  - localStorage.getItem('key', 'value')
  - localStorage.age = 40;

```
10  //updating data from local storage
11  localStorage.setItem("name", "hussain");
12  name = localStorage.getItem("name");
13  localStorage.age = 30;
14  age = localStorage.getItem("age");
15  console.log(name, age);
```

Chapter 06

❑ Local Storage

❑ Store Array in Local Storage

```
25  let todos = [
26    { name: "Basit", age: 21 },
27    { name: "Aqib", age: 29 },
28    { name: "Ali", age: 23 },
29  ];
30
31  // let stored = JSON.stringify(todos);
32  localStorage.setItem("todo", JSON.stringify(todos));
33  let stored = localStorage.getItem("todo");
34  console.log(JSON.parse(stored));
```

Chapter 15

❑ Object Oriented Programming

- Constructor Objects

The **constructor** property returns a reference to the **Object** constructor function that created the instance object.

Chapter 15

❑ Object Oriented Programming

- Constructor Objects

```
let o = {}
o.constructor === Object // true

let o = new Object
o.constructor === Object // true

let a = []
a.constructor === Array // true

let a = new Array
a.constructor === Array // true

let n = new Number(3)
n.constructor === Number // true
```

Chapter 15

❑ Object Oriented Programming

- **Class**

ECMAScript 2015, also known as ES6, introduced JavaScript Classes.

JavaScript Classes are **templates** for JavaScript Objects.

JavaScript Class Syntax:

Use the keyword `class` to create a class.
Always add a method named `constructor()`:

Chapter 15

❑ Object Oriented Programming

- Class
- Syntax

```
class ClassName {
  constructor() { ... }
}
```

Chapter 15

❑ Object Oriented Programming

- The ‘new’ keyword

1. It creates a new empty object {}
2. It bind the value of ‘this’ to the new empty object.
3. It calls the constructor function to ‘build’ the object

```
1  class Stud{
2      constructor() {}
3  }
4  const stud1 = new Stud()
```

Chapter 15

CLASSES

User Class

```
{
    username,
    email,
    login(),
    logout(),
}
```

→

```
new User ("Basit", "basit@gmail.com")
new User ("Aammar", "aammar@gmail.com")
new User ("Saad", "saad@gmail.com")
```

Chapter 15

❑ Object Oriented Programming

```
class Stud {
  constructor(name, email) {
    this.name = name;
    this.email = email;
    this.score = 0;
  }

  login() {
    console.log(`${this.name} logged in`);
    return this;
  }

  logOut() {
    console.log(`${this.name} logged out`);
    return this;
  }

  incScore() {
    this.score += 1;
    console.log(`${this.name} has ${this.score}`);
    return this;
  }
}
```

Chapter 15

❑ Object Oriented Programming - Class

```
34  class Student {
35    constructor(name, cls) {
36      this.name = name;
37      this.cls = cls;
38    }
39
40    login() {
41      console.log(`${this.name} logged in`);
42    }
43
44    logout() {
45      console.log(`${this.name} logged out`);
46    }
47  }
48
```

Chapter 15

❑ Object Oriented Programming - Inheritance

```
49  class Teacher extends Student {
50    removeStud(stud) {
51      studs = studs.filter((s) => {
52        return s.name !== stud.name;
53      });
54    }
55  }
56
57  let studOne = new Student("Basit", "Bachelor");
58  let studTwo = new Student("Aqib", "Bachelor");
59  let teacherOne = new Teacher("Hussain", "Bachelor");
60
61  let studs = [studOne, studTwo, teacherOne];
62
63  teacherOne.removeStud(studOne);
64  console.log(studs);
```

Chapter 15

❑ Object Oriented Programming – Inheritance

❑ super()

```
23  class Admin extends User{
24    constructor(username, email, title){
25      super(username, email);
26      this.title = title;
27    }
```

Chapter 15

❑ Object Oriented Programming – Inheritance

❑ Constructor Functions

```
1  // constructor functions
2
3  function User(username, email){
4    this.username = username;
5    this.email = email;
6    this.login = function(){
7      console.log(`${this.username} has logged in`);
8    };
9  }
```

Chapter 15

- Object Oriented Programming - Prototypes
- Every Object in JavaScript has a prototype.
- Prototype contain all the methods for that object type.

Chapter 15

❑ Object Oriented Programming – Prototypes

```
1  // constructor functions
2
3  function User(username, email){
4    this.username = username;
5    this.email = email;
6  }
7
8  User.prototype.login = function(){
9    console.log(`${this.username} has logged in`);
10   return this;
11 };
```

Chapter 15

❑ Object Oriented Programming – Prototype Inheritance

```
74  function Admin(name, email, title) {
75    User.call(this, name, email);
76    this.title = title;
77  }
```

Chapter 15

❑ Object Oriented Programming – Prototype Inheritance

```
79   Admin.prototype = Object.create(User.prototype);
```

Chapter 15

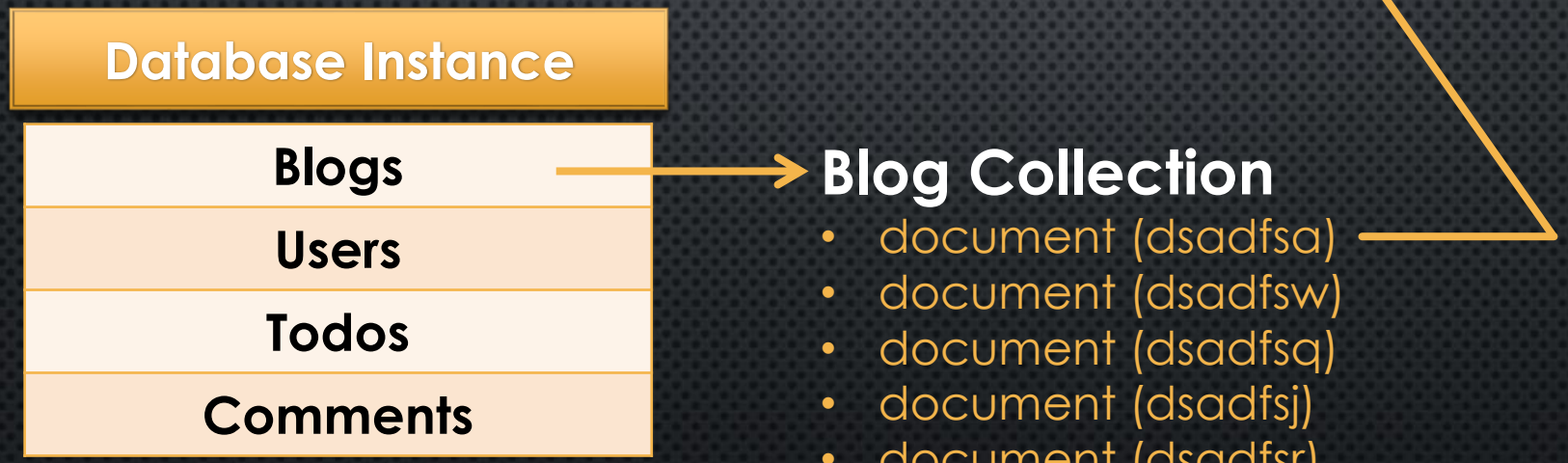
❑ NoSQL Database

SQL Databases	NoSQL Databases
Tables	Collections
Rows	Documents
Columns	Properties

Chapter 15

❑ NoSQL Database

```
Document {
title: 'basit',
likes: 50,
author: 'Hussain'
}
```



Chapter 15

- ❑ Why We Use Firebase
- ❑ Authentication
- ❑ Hosting
- ❑ Cloud Functions
- ❑ Cloud Storage
- ❑ Real Time Database

Chapter 15

❑ How to add Document

```
form.addEventListener("submit", (e) => {
  e.preventDefault();

  let now = new Date();

  const stud = {
    name: form.stud.value,
    admission_at: firebase.firestore.Timestamp.fromDate(now),
    class: "Matric",
  };
```

Chapter 15

❑ How to add Document

```
db.collection("fc_Stud")
    .add(stud)
    .then(() => {
        console.log("Student Added");
    })
    .catch((err) => {
        console.log("Error");
    });
});
```

Chapter 15

❑ How to Delete the Document

```
list.addEventListener("click", (e) => {
  // console.log(e);
  if (e.target.tagName === "BUTTON") {
    const id = e.target.parentElement.getAttribute("data-id");
    // console.log(id);
    db.collection("fc_Stud")
      .doc(id)
      .delete()
      .then(() => {
        console.log("Document Deleted");
      })
      .catch(() => {
        console.log("Error");
      });
  }
});
```

Chat Room Project

Chat Room Project

- ❑ Create HTML Template
- ❑ Connect Firebase
- ❑ Create Chat Collection on Firestore (minimum 2 different users)
- ❑ Copy the CND and pase it to index.html bottom of the body

Chat Room Project

❑ Step 01 – firebase CDN and JS file connect

```
<!-- Firebase CDN -->
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.2.9/firebase-firestore.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use
 https://firebase.google.com/docs/web/setup#available-libraries -->

<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyBW5rvQfwhptNk2qlL-Q16RxnM_vw8UTpE",
    authDomain: "chat-room-app-21.firebaseapp.com",
    projectId: "chat-room-app-21",
    storageBucket: "chat-room-app-21.appspot.com",
    messagingSenderId: "507916626765",
    appId: "1:507916626765:web:5da55b9bb877937e30a504",
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
  const db = firebase.firestore();
</script>

<script src="./chat.js"></script>
<script src="./app.js"></script>
<script src="./ui.js"></script>
```

Chat Room Project

- Step 02 – go to the chat.js
  - // adding new chat documents

```
class Chatroom {
    constructor(room, username) {
      this.room = room;
      this.username = username;
      this.chats = db.collection("chats");
    }
  }
```

```
const chatroom = new Chatroom("gaming", "basit");

console.log(chatroom);
```

Chat Room Project

❑ Step 03 – addChat method in the class Chatroom

❑ // adding new chat documents

```
//add chat
async addChat(message) {
  //format a chat object
  const now = new Date();
  const chat = {
    message,
    username: this.username,
    room: this.room,
    created_at: firebase.firestore.Timestamp.fromDate(now),
  };
  //   save the chat document
  const response = await this.chats.add(chat);
  return response;
}
```

Chat Room Project

❑ Step 04 – getChat method in the class Chatroom

❑ // getting new chat documents

```
29  getChats(callBack) {
30    this.chats.onSnapshot((snapshot) => {
31      //   console.log(snapshot.docChanges());
32      snapshot.docChanges().forEach((change) => {
33        // console.log(change);
34        if (change.type === "added") {
35          //Update UI
36          callBack(change.doc.data());
37        }
38      });
39    });
40  }
```

Chat Room Project

❑ Step 05 – getChat method in the class Chatroom

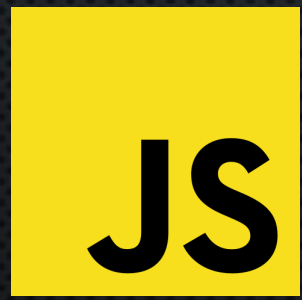
❑ // Complex Queries

```
getChats(callBack) {
  this.chats
    .where("room", "==", this.room)
    .orderBy("created_at")
    .onSnapshot((snapshot) => {
      //   console.log(snapshot.docChanges());
      snapshot.docChanges().forEach((change) => {
        // console.log(change);
        if (change.type === "added") {
          //Update UI
          callBack(change.doc.data());
        }
      });
    });
}
```

Chat Room Project

- Step 06 – updateName method in the class Chatroom
  - // Update Name

```
updateName(username) {
  this.username = username;
}
```



Chat Room Project

❑ Step 07 – updateRoom method in the class Chatroom

❑ // Update Room

```
updateRoom(room) {
  this.room = room;
  console.log("Room Updated");
  if (this.unsub) {
    this.unsub();
  }
}
```

Chat Room Project

❑ Step 08 – Creating ChatUI class in the ui.js

❑ // Creating UI

```
class ChatUI {
  constructor(list) {
    this.list = list;
  }

  render(data) {
    // console.log(data);
    const html = `
     <li class="list-group-item">
      <span class="username">${data.username}</span>
      <span class="message">${data.message}</span>
      <div class="username">${data.created_at.toDate()}</div>
    </li>`;

    this.list.innerHTML += html;
  }
}
```

Chat Room Project

❑ Step 09 – Get Chats in the Browser

❑ // call getChats method on app.js

```
44  const chatList = document.querySelector(".chat-list");
45
46  //class instances
47  const chatUI = new ChatUI(chatList);
48  const chatroom = new Chatroom("general", "kashan");
49
50  //get chats and render
51  chatroom.getChats((data) => {
52    chatUI.render(data);
53  });
```

Chat Room Project

❑ Step 10 – Set Date Fromat using date.fns

```
const when = dateFns.distanceInWordsToNow(
  data.created_at.toDate(),
  {addSuffix: true,}
  );

// console.log(data);
const html = `
 <li class="list-group-item">
  <span class="username">${data.username}</span>
  <span class="message">${data.message}</span>
  <div class="username">${when}</div>
</li>`;
```

Chat Room Project

❑ Step 11 – chat added using text input

```
43  newChatForm.addEventListener("submit", (e) => {
44    e.preventDefault();
45    const message = newChatForm.message.value.trim();
46    chatroom
47      .addChat(message)
48      .then(() => {
49        newChatForm.reset();
50      })
51      .catch((err) => {
52        console.log("err");
53      });
54  });
```

Chat Room Project

❑ Step 12 – Name Update using text input

```
55 newNameForm.addEventListener("submit", (e) => {
56   e.preventDefault();
57   //update name
58   const newName = newNameForm.name.value.trim();
59   chatroom.updateName(newName);
60   newNameForm.reset();
61
62   updateName.innerText = `Your name was sucessfully Updated ${newName}`;
63   setTimeout(() => {
64     updateName.innerText = " ";
65   }, 3000);
66   console.log("Name Updated", newName);
67 });
```

Chat Room Project

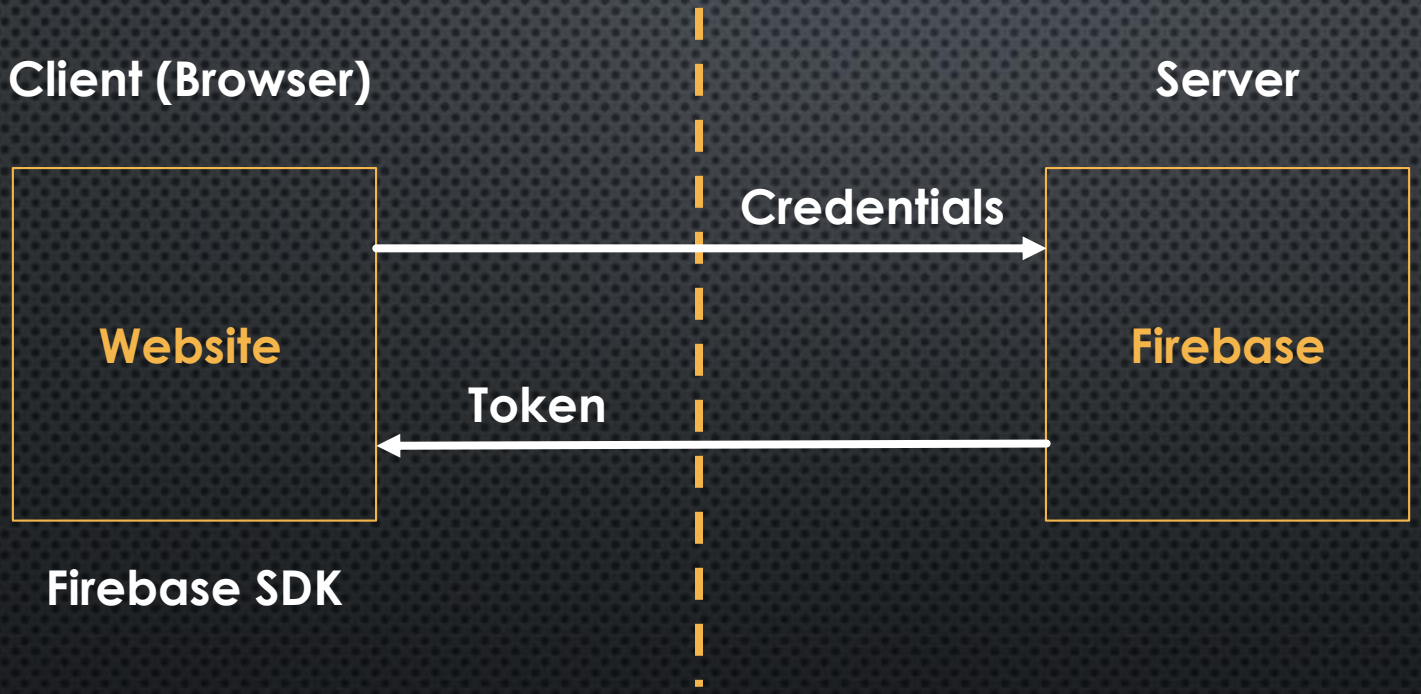
- ❑ Step 13 – Name store on the local storage

Chat Room Project

❑ Step 14 – Update Room

```
70  //update rooms
71  rooms.addEventListener("click", (e) => {
72    if (e.target.tagName === "BUTTON") {
73      chatUI.clear();
74      chatroom.roomUpdate(e.target.getAttribute("id"));
75      chatroom.getChats((data) => chatUI.render(data));
76    }
77  });
```

Firebase Auth



JavaScript Authentication Methods

- createUserWithEmailAndPassword(email, password)
- signOut()
- signIn(email, password)
- onAuthStateChanged()

JavaScript Authentication

- ❑ Step 1 – Go to the Firebase Console and Create the Project.
- ❑ Step 2 - Go to the Authentication and Enable Email/Password.
- ❑ Step 3 – Copy the Firebase SDK and Past it on index.html bottom of the body.
- ❑ Step 4 – Link these CDK

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/8.2.10/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.2.10/firebase-auth.js"></script
<script src="https://www.gstatic.com/firebasejs/8.2.10/firebase-firestore.js"></s
```

JavaScript Authentication

❑ Step 5 – Now Initialize these SDK in variables.

```
//make auth and firestore references
const auth = firebase.auth();
const db = firebase.firestore();
```

❑ Step 6 – Go to the firebase setting and TimestampsInSnapshots:

true;

```
//update firestore settings
db.settings({ timestampsInSnapshots: true });
```

JavaScript Authentication

- Step 7 – When all are done then go to the Auth.js and Create Sign up Method.

```
//sign up
const signup = document.querySelector("#signup-form");

signup.addEventListener("submit", (e) => {
  e.preventDefault();

  const email = signup["signup-email"].value.trim();
  const password = signup["signup-password"].value;

  console.log(email, password);

  //signup the user
  auth.createUserWithEmailAndPassword(email, password).then((data) => {
    // console.log(data.user);
    const modal = document.querySelector("#modal-signup");
    M.Modal.getInstance(modal).close();
    signup.reset();
  });
});
```

JavaScript Authentication

❑ Step 8 – Now Create Log Out Method

```
//logout
const logout = document.querySelector("#logout");

logout.addEventListener("click", (e) => {
  e.preventDefault();

  auth.signOut();
});
```

JavaScript Authentication

❑ Step 9 – Now Create Log In Method

```
//login
const login = document.querySelector("#login-form");

login.addEventListener("submit", (e) => {
  e.preventDefault();

  const email = login["login-email"].value;
  const password = login["login-password"].value;

  //auth sign in method
  auth.signInWithEmailAndPassword(email, password).then((data) => {
    // console.log(data);
    const modal = document.querySelector("#modal-login");
    M.Modal.getInstance(modal).close();
    login.reset();
  });
});
```

JavaScript Authentication

❑ Step 10 – Now Create onAuthState Change Method (Real Time Listeners)

```
// listen auth for state changes
auth.onAuthStateChanged((user) => {
  if (user) {
    console.log("User Logged In: ", user);
  } else {
    console.log("User Logged Out!");
  }
});
```

JavaScript Authentication

❑ Step 10 – Now Create onAuthState Change Method (Real Time

```
auth.onAuthStateChanged((user) => {
  if (user) {
    // console.log("The User Logged In");
    // Getting Data
    db.collection("guides")
      .get()
      .then((snapshot) => {
        setupGuides(snapshot.docs);
        // console.log(snapshot.docs);
      });
  } else {
    setupGuides([]);
  }
});
```

JavaScript Authentication

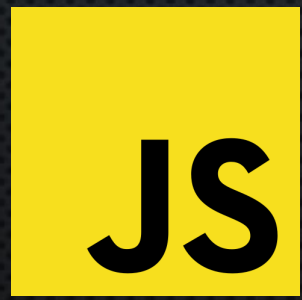
❑ Step 11 – setupGuides Method to Get Data on the Browser

```
const guides = document.querySelector(".guides");

const setupGuides = (data) => {
  // console.log(data);

  if (data.length) {
    let html = ``;
    data.forEach((doc) => {
      const guide = doc.data();
      const li = `
      <li>
      <div class="collapsible-header">${guide.title}</div>
      <div class="collapsible-body"><span>${guide.content}</span></div>
      </li>`;
      html += li;
    });

    guides.innerHTML = html;
  } else {
    guides.innerHTML = `<h5 class="center-align h5">Log in to view guides</h5>`;
  }
};
```



JavaScript Authentication

- Step 12 – Go to the firebase console then firestore database go to the rules and create rule.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // match /{document=**} {
    //   allow read, write: if
    //       request.time < timestamp.date(2021, 4, 6);
    // }
    // match doc in the guides collection
    match /guides/{guideId} {
      allow read, wirte: if request.auth.uid != null;
    }
  }
}
```

JavaScript Authentication

- Step 13 – go to the links which you created (logged-in and logged-out) and reference it on index.js

```
const loggedInLinks = document.querySelectorAll(".logged-in");
const loggedOutLinks = document.querySelectorAll(".logged-out");

const setupUI = (user) => {
  if (user) {
    loggedInLinks.forEach((item) => (item.style.display = "block"));
    loggedOutLinks.forEach((item) => (item.style.display = "none"));
  } else {
    loggedInLinks.forEach((item) => (item.style.display = "none"));
    loggedOutLinks.forEach((item) => (item.style.display = "block"));
  }
};
```

JavaScript Authentication

- Step 13 – now call it setupUI onAuthStateChange method

```
// on Auth State Changed
auth.onAuthStateChanged((user) => {
  if (user) {
    console.log("The User Logged In");
    // Getting Data
    db.collection("guides").onSnapshot((snapshot) => {
      setupGuides(snapshot.docs);
      // console.log(snapshot.docs);
      setupUI(user);
    });
  } else {
    setupUI();
    setupGuides([]);
  }
});
```

JavaScript Authentication

- Step 14 – Now we will create guides first reference on the auth.js (create-form) which we created.

```
19  //Create Guide
20  const createForm = document.querySelector("#create-form");
21  createForm.addEventListener("submit", (e) => {
22    e.preventDefault();
23
24    db.collection("guides")
25      .add({
26        title: createForm.title.value,
27        content: createForm.content.value,
28      })
29      .then(() => {
30        //close modal
31        const modal = document.querySelector("#modal-create");
32        M.Modal.getInstance(modal).close();
33        createForm.reset();
34      })
35      .catch((err) => {
36        console.log(err.message);
37      });
38  });
```

JavaScript Authentication

❑ Step 14 – Now add Real Time Listener Firebase Method

onSnapshot to get the data

```
auth.onAuthStateChanged((user) => {
  if (user) {
    console.log("The User Logged In");
    // Getting Data
    db.collection("guides").onSnapshot((snapshot) => {
      setupGuides(snapshot.docs);
      // console.log(snapshot.docs);
      setupUI(user);
    });
  } else {
    setupUI();
    setupGuides([]);
  }
});
```

JavaScript Authentication

❑ Step 15 – Now add User Account Details

const accountDetails = document.querySelector(".account-details");

```
6   //Set Up Links
7   const setupUI = (user) => {
8     if (user) {
9       //account info
10      const html = `
11        <div>Logged in as ${user.email}</div>
12      `;
13      accountDetails.innerHTML += html;
14      // toggle UI elements
15      loggedInLinks.forEach((item) => (item.style.display = "block"));
16      loggedOutLinks.forEach((item) => (item.style.display = "none"));
17    } else {
18      accountDetails.innerHTML = "";
19      loggedInLinks.forEach((item) => (item.style.display = "none"));
20      loggedOutLinks.forEach((item) => (item.style.display = "block"));
21    }
22  };
```

JavaScript Authentication

❑ Step 15 – User Collections

- **Firebase Auth**

❑ User Id

❑ Email

❑ Photo URL

❑ Display Name

JavaScript Authentication

- Step 16 – Go to the sign up form and add one more field sign up bio

```
<!-- Signup Modal -->
<div id="modal-signup" class="modal">
  <div class="modal-content">
    <h4>Sign Up</h4>
    <form id="signup-form" style="margin-top: 40px">
      <div class="input-field">
        <input id="signup-email" type="email" class="validate" required />
        <label for="signup-email">Email address</label>
      </div>
      <div class="input-field">
        <input
          id="signup-password"
          type="password"
          class="validate"
          required
        />
        <label for="signup-password">Choose Password</label>
      </div>
      <div class="input-field">
        <input id="signup-bio" type="text" class="validate" required />
        <label for="signup-bio">One Line Bio</label>
      </div>
      <button class="btn yellow darken-2 z-depth-0">Sign up</button>
    </form>
  </div>
</div>
```

JavaScript Authentication

- Step 16 – Now, Go to the sign up event listener and add users collection.

```
//signup the user
auth
  .createUserWithEmailAndPassword(email, password)
  .then((data) => {
    // console.log(data.user);
    return db.collection("users").doc(data.user.uid).set({
      bio: signup["signup-bio"].value,
    });
  })
  .then(() => {
    const modal = document.querySelector("#modal-signup");
    M.Modal.getInstance(modal).close();
    signup.reset();
  });
```

JavaScript Authentication

- Step 16 – Now, Go to the fire store rules allow them to read and write all users.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
          request.time < timestamp.date(2021, 4, 6);
    }
    // match doc in the guides collection
    match /guides/{guideId} {
      allow read, write: if request.auth.uid != null;
    }
  }
}
```

JavaScript Authentication

- Step 16 – Now, show the bio to the account modal

```
//Set Up Links
const setupUI = (user) => {
  if (user) {
    //account info
    db.collection("users")
      .doc(user.uid)
      .get()
      .then((doc) => {
        const html = `
        <div>Logged in as ${user.email}</div>
        <div>Bio: ${doc.data().bio}</div>
      `;
        accountDetails.innerHTML += html;
      });
    // toggle UI elements
    loggedInLinks.forEach((item) => (item.style.display = "block"));
    loggedOutLinks.forEach((item) => (item.style.display = "none"));
  } else {
    accountDetails.innerHTML = "";
    loggedInLinks.forEach((item) => (item.style.display = "none"));
    loggedOutLinks.forEach((item) => (item.style.display = "block"));
  }
};
```

JavaScript Authentication

- Step 17 – Secure the firebase auth

```
// match logged in user doc in users collection
match /users/{userId} {
  allow create: if request.auth.uid != null;
  allow read: if request.auth.uid == userId;
}
```

THANKS FOR YOUR ATTENTION