Mobile Application Development

LECTURE 7

Synchronous and Asynchronous Programming

- A Synchronous program is executed line by line, one line at a time.

- In a synchronous programming model, things happen one at a time. When you call a function that performs a long-running action, it returns only when the action has finished and it can return the result.

- An asynchronous model allows multiple things to happen at the same time. When you start an action, your program continues to run. When the action finishes, the program is informed and gets access to the result

Synchronous and Asynchronous Programming

- A simple example of Asynchronous would be a person reading a newspaper while also washing clothes in a dryer. He puts clothes in the dryer, turns it on, starts reading newspaper, when the dryer stops, he takes the clothes out. Multiple things at a time.

- If this was Synchronous, he would first put clothes in dryer, wait until the dryer finishes and stops and then starts reading newspaper.

Synchronous and Asynchronous Programming

- Normally, programming languages are synchronous. C, Java, C#, PHP, Go, JavaScript, Ruby, Swift, Python, they are all synchronous by default.
- These languages however provide ways to manage asynchronicity, in the language or through libraries.
- Some of them handle async by using threads, spawning a new process.

Asynchronous Programming in JavaScript

- JavaScript is synchronous by default and is single threaded. This means that code cannot create new threads and run in parallel.
- It provides asynchronous behaviour through Callbacks, Promises, Generators, Async Await.
- We need Asynchronous for network requests, database requests, I/O etc

Callbacks

- **Callback** is a simple function that's passed as a value to another function, and will only be executed when the event happens.
- We can do this because JavaScript has **first-class functions**, which can be assigned to variables and passed around to other functions (called higher-order functions).
- **higher-order functions** mean passing a function as an object. (Functional Programming)
- Remember, JavaScript is a hybrid language. Object Oriented and Functional Language.

Callbacks

- Examples of Callback Functions

```
$("body").click(function() {
    Alert'hey there');
});
```

```
document.getElementById('button').addEventListener('click', () => {
  //item clicked
})
```

Callbacks

- API request

```
const request = require('request');
let result;

request('https://random-quotes.now.sh', function (error, response, body) {
  if(error){
    // Handle error.
    console.log('error',error);
  }
  else {

    // Successful, do something with the result.
    result = response.body;
  }
});
```

Callbacks

- But why we avoid callbacks?
- Code becomes harder to read.
- Christmas Tree Problem / Callback Hell. (http://callbackhell.com/)
- In comes, **Promises**

"Some people make your laugh a little louder, your smile a little brighter and your life a little better. Try to be one of those people."

Promises

- Promises were introduces in ES6.
- A promise is an object that wraps an asynchronous operation and notifies when it’s done.
- Promises have three states: fulfilled, rejected, or pending.

```
let promise = new Promise((resolve, reject) => {
  console.log("Hello");
  resolve("Promise complete");
});
promise.then(result => console.log(result));
```

Async Await

- Async await example.

```
async function main() {
let promiseResult = new Promise((resolve, reject) => {
  console.log("Hello");
  resolve("Promise complete");
});


const result = await promiseResult();

}

main();
```

Async Await vs Promises

```
// Promise

fetch('https://random-quotes.now.sh')
  .then(
    function(response) {
          console.log(response);
    }
  )
  .catch(function(err) {
    console.log('Err', err);
  });

// Async Await

try {

  const response = await fetch('https://random-quotes.now.sh')
  }

  catch(err){

  console.log('err', err)
  }
```

Rxjs (Reactive Extensions)

- RXJS Is a library for reactive programming using observables that makes it easier to compose asynchronous or callback-based code. Code becomes harder to read.
- Ideal for streams of data (Continuous data e.g like in sockets / chat)
- **Observables** are the key feature of rxjs. They will be coming to JavaScript language very soon.
- Observables provide more power compared to promises.

Rxjs (Reactive Extensions)

- Observables open up a continuous channel of communication in which multiple values of data can be emitted over time. You will use them alot for HTTP requests and for services.

```
getData(): Observable {
  const url = 'https://random-quotes.now.sh';
  return this.http.get(apiURL)
}

  this.getData().subscribe( data => {
      this.loading = false;
      this.results = data (1)
    },
error => {
    console.log('err',err);
  });
```