Aptech Limited

Essentials of Node JS

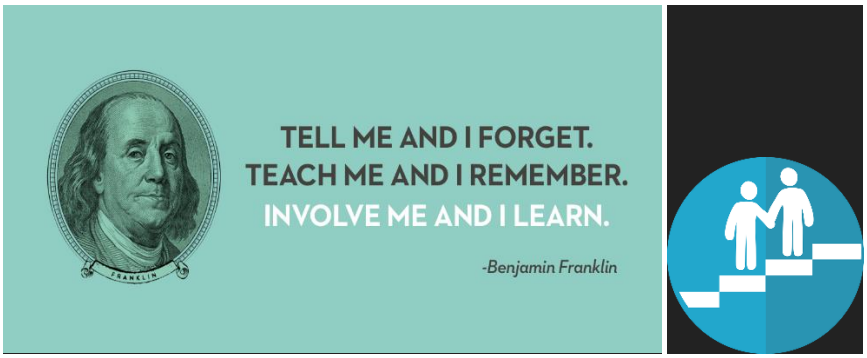
FACILITATOR GUIDE

Aptech
Unleash your potential

About the Facilitator Guide

This guide is designed for the facilitator/faculty to enable the learning objectives. The main goal is to help the participants meet the learning goals through extensive guidance and excellent facilitation in a blended learning mode.

This guide explains in detail the process of conducting the training, mode of delivery at each content point, additional support required, and activities to be carried out to ensure effective and smooth learning.



Goal

This guide is designed to help the facilitator engage students in learning about Node JS. It aims at providing information to students about concepts of Node JS and its applications.

Context

Node JS is a cross-platform, open-source, back-end JavaScript runtime environment that uses the V8 engine to execute JavaScript code outside of a Web browser. Node JS allows developers to utilize JavaScript and create command-line tools and server-side scripting, which involves running scripts on the server before sending the page to a user's browser. As a result, Node JS symbolizes a JavaScript everywhere paradigm, bringing Web-application development together around a single programming language rather than separate languages for server-side and client-side scripts.

Course Structure

Facilitator Guide	Lesson Number	Start Topic	End Topic	Class Duration	Video Content Duration (Approx.)
1	1	Getting Started	Sharing Functions Between Files	2 Hours	00:53:29
2	2	Request and Responses	Serving HTTP Responses	2 Hours	01:15:34
3	3	Express.JS	Handling Parameters	2 Hours	00:37:27
4	4	Introducing Event Loop	Testing API with CURL	2 Hours	00:45:28
5	5	Setting Things Up	Creating Version 2 Rest APIs	2 Hours	00:48:28
6	6	Using MongoDB	Subdocuments and References	2 Hours	01:18:24
7	7	Request Middleware	Bycrypt	2 Hours	00:36:16
8	8	Introduction to Heroku	Adding Performance Monitoring	2 Hours	00:54:02

Course Objectives

After completion of this course, the student will be able to:

- Explain the evolution of Node JS
- Explore Server-side development with Node JS
- Understand Asynchronous Programming
- Explain CURL with MySQL
- Describe CURL with MongoDB
- Explore MongDB
- Explain authentication and security
- Understand deployment with Heroku

Audience

This course is beneficial for professionals who intend to delve into server-side development.

Prerequisite for Students

Fundamental knowledge of Programming is the basic course prerequisite. However, participants are also expected to have knowledge of HTML5, CSS, Bootstrap (Optional), and JavaScript.

Session Preparation Guidelines

1. Using Video Content

Each session of this course is delivered in a blended learning mode. The delivery will be structured around the video content. The faculty/facilitator will ensure effective learning, through various value additions that will be provided in the classroom along with the video content. For each session, detailed delivery guidelines are provided in this guide.

2. Session Material

Following are the basic materials required for a classroom session:

- Whiteboard and Markers
- Be ready with Flip charts or any other material that is specified in each session
- Printed copies of **Participant Worksheets** (if any)
- Projector and pen drive with the video file to be played
- **Check List** (Annexure 1): This checklist is to be used and signed before every session to ensure that everything is in place
- **Self-Assessment Sheet** (Annexure 2): Use this sheet at the end of every session to self-assess the quality of facilitation/teaching

To be printed for each Session:

Annexure 1

Check List

Week Before the Session	Day of the Session
✓ Gone through the session details in this guide. ✓ Gone through the links (if any) mentioned in session details. ✓ Collected supplementary material required as mentioned in session details. ✓ Taken prints of the worksheets (if any) for the students. ✓ Ensured all material required for hands on is in place. ✓ **Check and play the video twice and rehearse timings**.	✓ Projector is working. ✓ Markers are working. ✓ All material is in place (Flip charts, Notes, and so on). ✓ Attendance sheet is printed. ✓ Self-Assessment sheet is printed. ✓ Video is playing with proper audio.

Session Name:
Signature:
Date:

Facilitator Notes:

Self-Assessment Sheet

Session Name:
Duration:
Date:

Assessment Pointers	Assess Yourself
- Was the Quality of Video Played (Audio and visual) OK?	
- Was the timing of the Session appropriate to complete the learning process?	
- Were students actively participating in discussions?	
- Were students able to complete the tasks assigned?	
- Were links added for value addition components of the content appropriate for students?	
- Were the additional explanations adequate for the students?	
- Were the questions asked by students demonstrating their understanding of topics?	
- Rate the quality of teaching and delivery.	

SESSION 1: BASICS OF NODE.JS

1.1 Getting Started

CLASS DURATION: 8 MINUTES

VIDEO DURATION: 5 MINUTES

FACILITATION GUIDELINES -

Greeting: Welcome the students to the course. Introduce yourself and seek the introduction of students. During the introduction, ask students what they expect from this course.

After the introduction, play the course video.

Show: Play Getting Started with Node.js: Session 1 Video – 1.1.

Discussion: Node.js is a free, open-source server environment that is used to run JavaScript on the server-side environment. It runs on multiple platforms including Linux, Mac OS X, Windows, Linux, and UNIX.

NodeJS, Node, and Node.js (as used with these casing) are synonymous with each other and can be used interchangeably.

Node.js can build scalable network applications, create dynamic page content, and collect form data and so on.

JavaScript is a prerequisite to learning Node.js. Since, Node.js uses JavaScript on the server, knowing JavaScript makes it easier to learn Node.

Show: Play the video from 1:18 to 2:40.

Content: Explain to the students that running the command `node` on the terminal (for example, through spotlight) makes it enter the node console. The console is basically the JavaScript console.

Further, ask the students about their familiarity with JavaScript. You may ask if they have some experience working on JavaScript projects. Let them know that there is a JavaScript console in the browser itself. This will be helpful to view outcomes of scripts and troubleshoot or debug, if required.

Additional Reference:

Refer to following link for more information: https://devdocs.io/javascript/

Show: Play the video from 1:33 to 2:40.

Content: Tell the students that they can practically implement Node.js in both the node terminal and the browser console. Moreover, inform them that they can perform basic functions such as addition, increment, display, decrement, and so on.

Show: Play the video from 3:07 to 3:40.

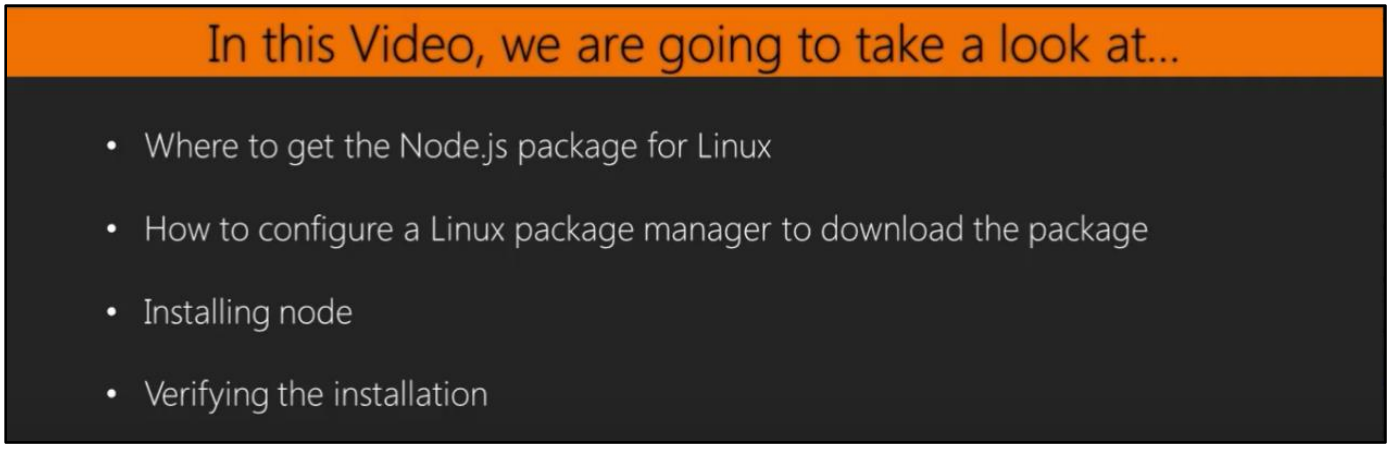
Content: Inform the students about the differences between NodeJS and JavaScript. JavaScript is a programming language while NodeJS is a runtime environment. JavaScript is utilized on the client-side and used in front-end development. On the other hand, NodeJS is used on the server-side and comes in handy for server-side development.

Example: Functions such as `window()` and `document()` are undefined in the node terminal. On the contrary, there is a `detail` object in the browser console displayed using these functions. Similarly, using the `process()` function in the node terminal works. However, it does not work in the browser.

Show: Play the video from 3:57 to 4:54.

Content: Explain to the students that node is used to write JavaScript code that executes in a computer process directly rather than in a browser. There are many text editors available today that can support Node.js. *Sublime Text* editor is one of these. It can be used to execute node files or projects. A 'hello_world.js' file is created on the desktop that prints the `Hello World` statement on the console. In the terminal, the file is navigated by providing the pathname. It is then executed using the 'node file_name.js' file, where the extension of the file is optional. As expected, `Hello World` is displayed on the console.

1.2 Installing Node on Linux



CLASS DURATION: 10 MINUTES

VIDEO DURATION: 4 MINUTES

FACILITATION GUIDELINES -

Show: Play Installing Node on Linux: Session 1 Video – 1.2.

Discussion: In this video, students will learn how to install Node on their Linux systems.

Discuss following installation steps with the students:

Step 1: Go to nodejs.org using a Web browser.

Step 2: As displayed on the Website, there are two versions available to download for Linux - v4.4 LTS and v6.3 Current.

LTS stands for Long Term Support. The LTS version is recommended for organizations with complex environments that find it tedious to constantly upgrade the software. It primarily focuses on stability and security.

The current version contains all the latest features. For this session, the current version would be used.

Step 3: Click the 'Current' green button to download the package. Alternatively, click 'Downloads' on the top menu bar and scroll down to click the 'Installing Node.js via package manager'.

Step 4: Then, click 'Debian and Ubuntu based Linux distributions'.

Step 5: Install version 6 (the latest version) by copying the given command. Open a terminal and paste the command there. The package manager then updates the package sources. Once it is completed, copy the command `apt-get install nodejs`, paste the command, and add `sudo` in front of the command.

Step 6: Now, press Enter and `Y` to continue. If there are no errors displayed, node must be successfully installed. A successful installation can be verified by typing the command `node -v` and pressing Enter. The currently installed version of node in the system will be displayed (v6.3).

Step 7: Node has an interactive shell similar to a console in a browser. You can type the `node` command in the terminal to enter the node shell.

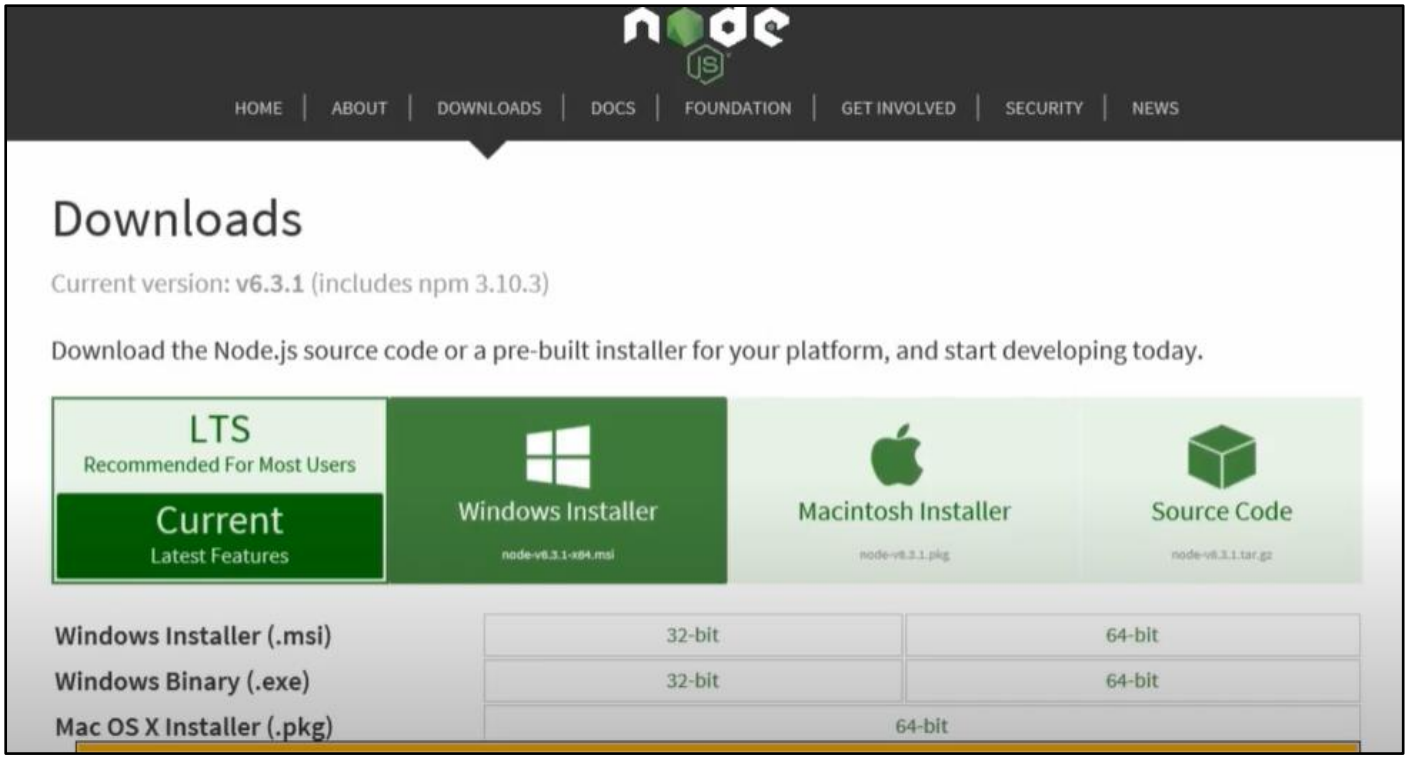
1.3 Installing Node on Windows

CLASS DURATION: 10 MINUTES

VIDEO DURATION: 3 MINUTES

FACILITATION GUIDELINES -

Show: Play Installing Node on Windows: Session 1 Video – 1.3.



Discussion: In this video, students will learn how to install Node on their Windows systems.

Following are the steps to install Node on Windows:

Step 1: Go to nodejs.org using a Web browser.

Step 2: Click 'Downloads' on the top menu. The *Downloads* page will open. There will be two choices, that is, the LTS and the Current version. For this session, the Current version will be installed.

Step 3: Click the 'Current' button. Then, click 'Windows Installer' to download the installation program. Once it is downloaded, run the *msi* package. A graphical setup wizard shall now be displayed. It helps in setting up the configuration and guides a user with the installation steps.

Step 3: Click the 'Next' button to read the End-User License Agreement (EULA). After reading the EULA, accept the terms and click 'Next'.

Step 4: There is no requirement for customizing the installation settings for this session. So, proceed by clicking 'Next' with the default settings. Finally, click 'Install' to install the program. Windows might ask the user for permission to install foreign software. Click 'Yes' to grant permission. The installation now begins.

Step 5: The installation usually takes 1-2 minutes. Click 'Finish' after the installation is complete. This exists the setup wizard.

Step 6: Node.js command line can be launched by searching for node in the system's search bar and then clicking on 'Node.js command prompt'.

Step 7: Successful installation can be verified by typing the command `node -v` and pressing Enter. The version string of node installed in the system will be shown (v6.3).

Node has an interactive shell, similar to a console in a browser. Type `node` command in the terminal to enter the node shell.

1.4 Writing Your First Node App

CLASS DURATION: 15 MINUTES

VIDEO DURATION: 5 MINUTES

FACILITATION GUIDELINES -

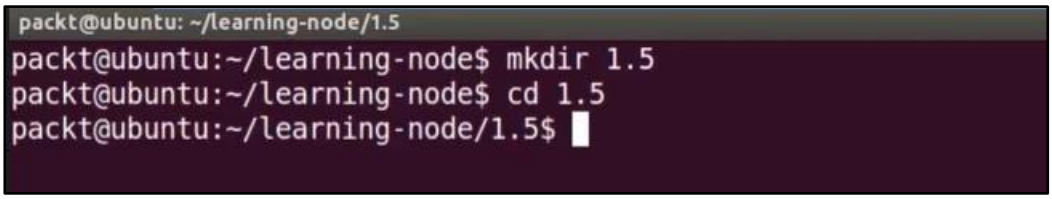
Show: Play Writing Your First Node App: Session 1 Video – 1.4.

Discussion: In this video, students will learn to create their first Node application. Ubuntu will be used as the development environment in this session.

Describe following steps to initialize a new node project:

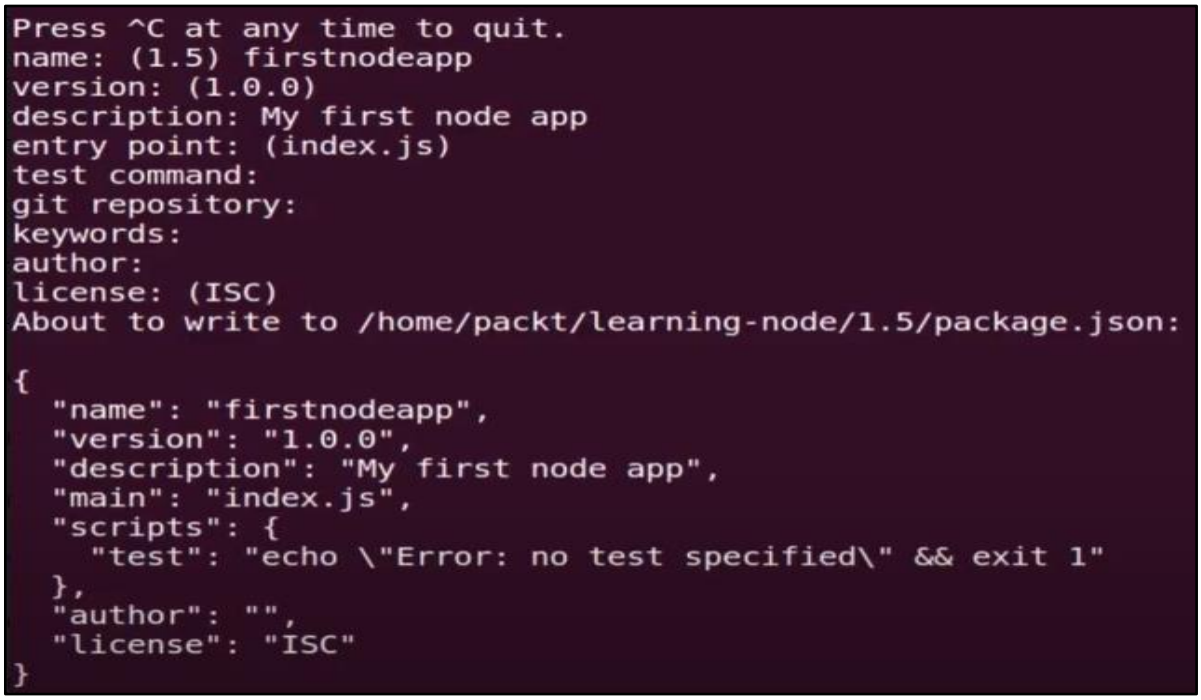
Step 1: Open a terminal. Create a folder (project directory) to store the new project using the `mkdir` command.

Step 2: Go into the newly made directory by using the `cd your_new_directory_name` command.



The Metadata of the project is defined in the *package.json* file. It holds details such as project name, description, entry point, version, test command, Git repository, and so on. It can be created using the `npm init` command.

Show: Show this image to the students.



Step 3: Project name, description, and entry point must be entered while other suggestions can be accepted as default. Type `Yes` to end finish creating the project.

Step 4: Open any text editor such as *Sublime Text*. Now, open the project folder. Click the *package.json* file and observe the Metadata created.

Step 5: Click 'New file' to create a new file with the name 'index.js' in the root directory. Here, the *http* module will be used for setting up a Web server. This Web server will be used to run the project.

Step 6: Import the *http* module using the `require` method. Create a new *http* server instance using the `createFunction` method of the *http* module.

Industry Best Practice:

Always add required libraries or packages in the beginning of the file and avoid adding them inside functions. Explain to students that using required modules inside functions might produce errors during execution of the file.

This function takes a request-response handler function as a parameter. Write an anonymous function that takes two arguments - `res` and `req` as the response and request respectively. Inside the anonymous function, write the business logic. It can be anything.

The response body can hold any sentence and return it. The `res.end` method is used to return the sentence to the client. Now, close the function. The server can now be started by directing it to listen on port 8080.

For this, a `listen` method is required for taking a callback function as a second parameter. It takes the first parameter as `8080`. The callback is invoked when the server successfully listens to the specified port. Print any sentence inside the function to the console for verification.

Step 7: Return to the terminal and enter the `node index` command. For the successful creation of a server instance, the message will be displayed in the console.

Step 8: Open a browser and type in 'localhost:8080'. The intended message must be displayed verifying the successful creation of the first node app.

In-Class Question: Where is a metadata of the project defined?

Answer: Metadata of the project is defined in the *package.json* file.

Additional Reference:

Refer to following link for more information: https://nodejs.dev/learn

1.5 Creating Command Line Utilities

CLASS DURATION: 40 MINUTES

VIDEO DURATION: 23 MINUTES

FACILITATION GUIDELINES -

Show: Play Create Command Lines: Session 1 Video – 1.5.

Discussion: In this video, students will learn how to create a command-line utility in NodeJS. The utility is a simple to-do list only in the command line.

Show: Play the video from 1:20 to 3:17.

Content: Explain the concept of asynchronicity through an example of a person's daily morning routine. The routine includes activities such as waking up in the morning, checking his/her phone, and then eating breakfast regularly.

Three simple functions such as `wakeUp()`, `checkPhone()`, `eatBreakfast()` are written and called one by one to invoke them. The file is saved in the system (desktop, in this session) as 'morning.js'.

The file is then navigated using the `cd path_name` command in the terminal. Once, the terminal is inside the desktop path, all the saved files are displayed. The methods can be invoked by writing the `node morning.js` command and pressing Enter. Finally, the methods print the statements in the console.

Show: Play the video from 3:18 to 7:00.

Content: Explain to the students that these methods are synchronous. This essentially means that the second function is executed only after the first function gets completely executed and so on.

In this example, the `checkPhone()` function will wait for the `wakeUp()` function to finish execution before it begins to execute.

Example: A person might want to check their phone while waking up. Thus, he/she might want to do two things simultaneously. For instance, people like to listen to music while doing other chores. This is an example of asynchronicity.

Show: Play the video from 7:03 to 8:18.

Content: Explain to the students that for asynchronously executing these three functions, each function will now accept a callback function (`callback` or `cb`). This `callback` function can be invoked from inside the main functions.

Show: Play the video from 8:30 to 16:35.

Content: Inform the students that a 'to-do' list app will now be created. This app can add, remove, and move items to/from the list. The editor's mode will now be switched to JavaScript. For this, the `readLine` library must be included to use the utilities of code which will not be present in the main code.

The `readLine.createInterface` ensures that the information is received from the console. This also makes sure to return any results from the code to the console only. The parameters `process.stdin` and `process.stdout` are passed to take some input from the client-side and pass them to the interface. The file is saved as '*todo.js*'.

Show: Play the video from 8:30 to 12:35.

Content: Explain to the students that a `prompt` is set for a better user experience in this step. The `on()` method is called with the `readLine` instance which takes input as `line` and prints it in the console. This verifies the correct working of the function.

Show: Play the video from 8:30 to 16:35.

Content: Explain to the students that a `toDoList` variable is taken as an array to perform operations on the to-do items in this step. The function now performs a push operation on the array and inserts the input given by the user. Then, it prints out the `toDoList` to show the inserted items.

Show: Play the video from 13.16 to 16.57.

Content: Describe to the students that the input given by the user contains both the command and the item to be inserted or removed. The `split()` method is used to separate the command from the item.

The `split()` method creates an array with the split words. `words[0]` contains the command (*`add`*, *`ls`*, and *`rm`* is *add*, *list*, and *remove* respectively). However, adding an `if` statement is the simplest way to do different things based on different commands.

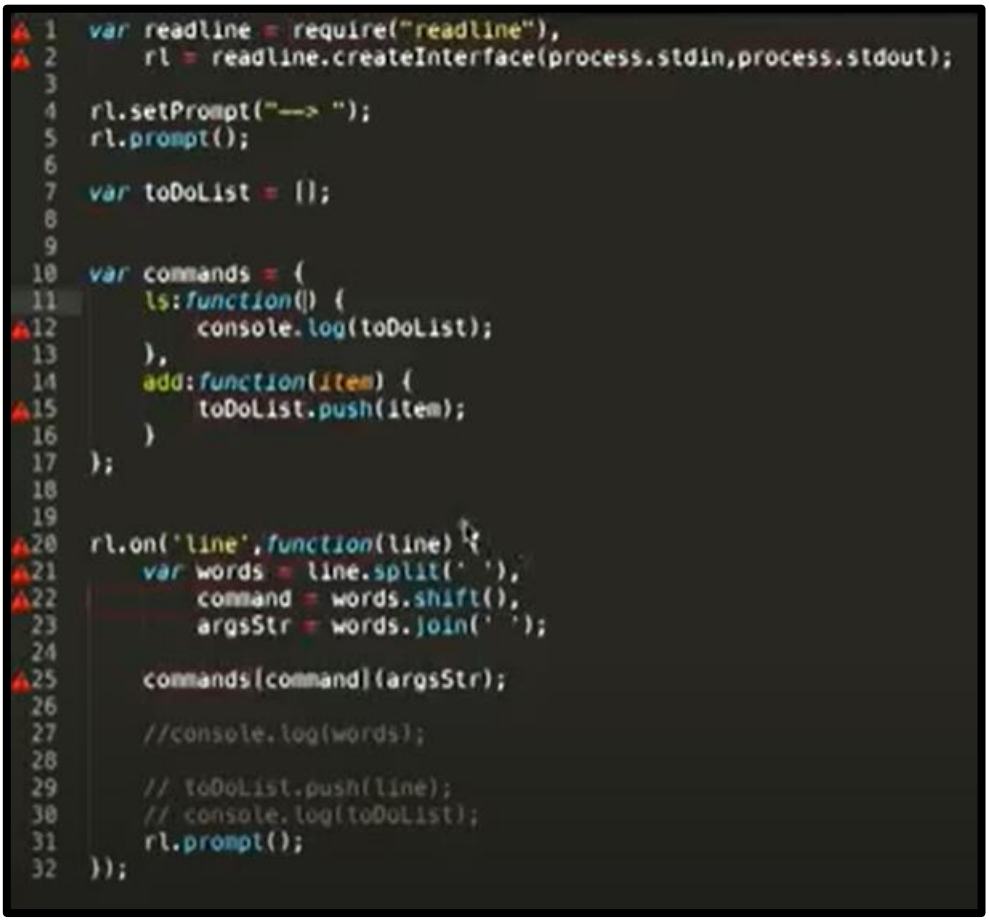
An `if` condition checks which command matches with the input command. Therefore, if the command is `ls`, the to-do list gets printed. If the command is `add`, the input item from the user to the `toDoList` gets inserted.

Show: Play the video from 17:00 to 19:01.

Content: Explain to the students that the `shift()` method is used with the input line instead of the `split()` method. This is because the `shift()` method simply removes the first word (command) from the input that provides the rest of the word as the item.

The `Split()` method can separate the words. Nevertheless, the `shift()` method works correctly for inserting just the item. Now, in the `add` condition, the `item` variable stores the separated words. These words are joined together by the `join()` method that is finally pushed into the array.

Show: Play the video from 20:10 to 21:22.



```
var readline = require("readline"),
    rl = readline.createInterface(process.stdin,process.stdout);

rl.setPrompt("--> ");
rl.prompt();

var toDoList = [];


var commands = {
    ls:function() {
        console.log(toDoList);
    },
    add:function(item) {
        toDoList.push(item);
    }
};


rl.on('line',function(line) {
    var words = line.split(' '),
        command = words.shift(),
        argsStr = words.join(' ');

    commands[command](argsStr);

    //console.log(words);

    // toDoList.push(line);
    // console.log(toDoList);
    rl.prompt();
});
```

Content: Explain to the students that this step involves an effort to make the code more organized. Therefore, an object named `commands` is created that contains two functions. These are `add` and `ls` that perform the same corresponding operations shown previously. Now, the first word (command) of the input is compared to either of the functions.

Show: Play the video from 21:25 to 22:00.

Content: Explain to the students that the `command` variable which stores the actual command along with the item is passed as arguments to the `commands` object. Finally, ask the students to think about how the `remove()` function should be written.

Tips: Try to make use of built-in debugger to debug the code with ease. This tool does a step-by-step debugging of the code and scope injection.

In-Class Question: What parameters are passed to take input from the client-side and pass them to the interface?

Answer: Parameters `process.stdin` and `process.stdout` are passed to take some input from the client-side and pass them to the interface.

Additional Reference:

Refer to following links for more information:

https://thecodebarbarian.com/building-a-cli-tool-with-node-js.html

https://dev.to/dendekky/how-to-build-a-command-line-tool-with-nodejs-a-step-by-step-guide-386k

https://www.youtube.com/watch?v=OO2Oy1VjSB4

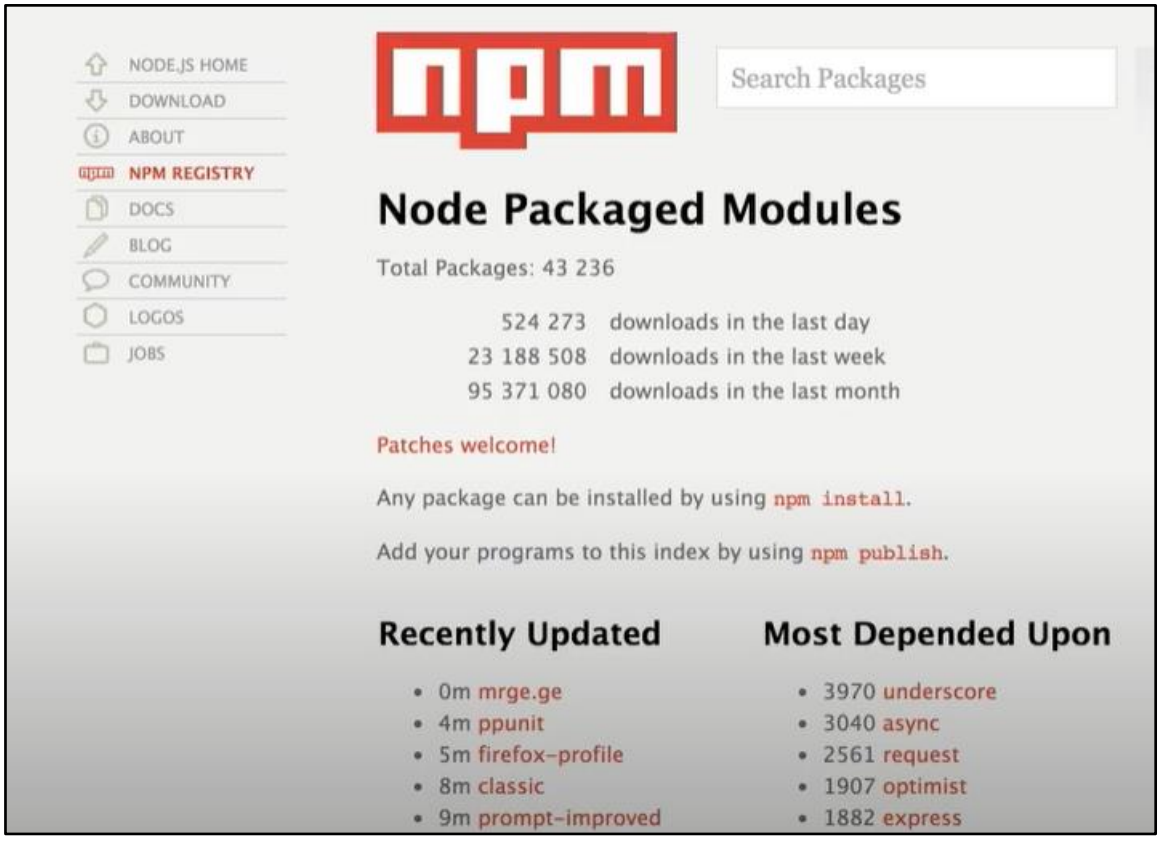
1.6 Using Existing Code in Node.js

CLASS DURATION: 22 MINUTES

VIDEO DURATION: 8 MINUTES

FACILITATION GUIDELINES -

Show: Play Using Existing Code in Node.js: Session 1 Video – 1.6.



Discussion: In this video, students will learn how to use previously existing codes or other people's code in a NodeJS project.

Show: Play the video from 0.28 to 2:17.

Example: In this video, the GitHub project *colors.js* formats the text output. For example, the `.red()` method changes the text color to red.

Ask the students what different effects they would like to test on sample inputs. Elicit replies.

Additional Reference:

Refer to following link for more information: https://www.npmjs.com/package/colors

Content: Explain the most common method to use a NodeJS project from a GitHub profile. This involves downloading the *zip* file and then using it in the user's own file using the `require` function.

The `require` function takes a parameter with the pathname directing to that downloaded file and the extracted *zip* folder. Tell the students that even if this works, this is not the standard way in node.

Show: Play the video from 3:05 to 4:03.

Content: Explain to the students that using the Node Package Manager (*npm*) enables a user to do the same work. NPM is a terminal program that is included in node. It supports registered external source codes.

Using the `npm install colors` command downloads a 'node modules' folder including the 'colors' package after searching it through the registry. Now, using the package name in the `require` parameter is enough for node to recognize the code and work as expected.

Show: Play the video from 4:10 to 6:26.

Content: Explain to the students that there is a better way to deal with all dependencies. The previously discussed technique would not be effective if there is a long list of dependencies. Moreover, downloading them one by one can be cumbersome.

Hence, a better way is to create a node project using the `npm init` command. Executing this command will result in the prompt asking a few questions such as the name of the project, Git repository, and so on.

Filling in the necessary details and entering `Yes` can create a node project with a *package.json* file. This file now includes a dependencies list. A long list of required dependencies can just be added to this file and used in the project.

Each time a new dependency is added in the package.json file, the command `npm install` must be used to update the project with the latest dependencies. Inform the students to always navigate to the node project folder and then, use the node commands such as `npm install`.

Example: The *'underscore'* package is added to the dependency and included in the project using the `require` method.

Additional Reference:

Refer to following link for more information:
https://nodejs.org/en/docs/meta/topics/dependencies/

In-Class Question: What is the ideal way to create a node project?

Answer: Ideal way is to create a node project is using the `npm init` command

1.7 Sharing Functions between Files

CLASS DURATION: 15 MINUTES

VIDEO DURATION: 5 MINUTES

FACILITATION GUIDELINES -

Show: Play Sharing Functions between Files: Session 1 Video – 1.7.

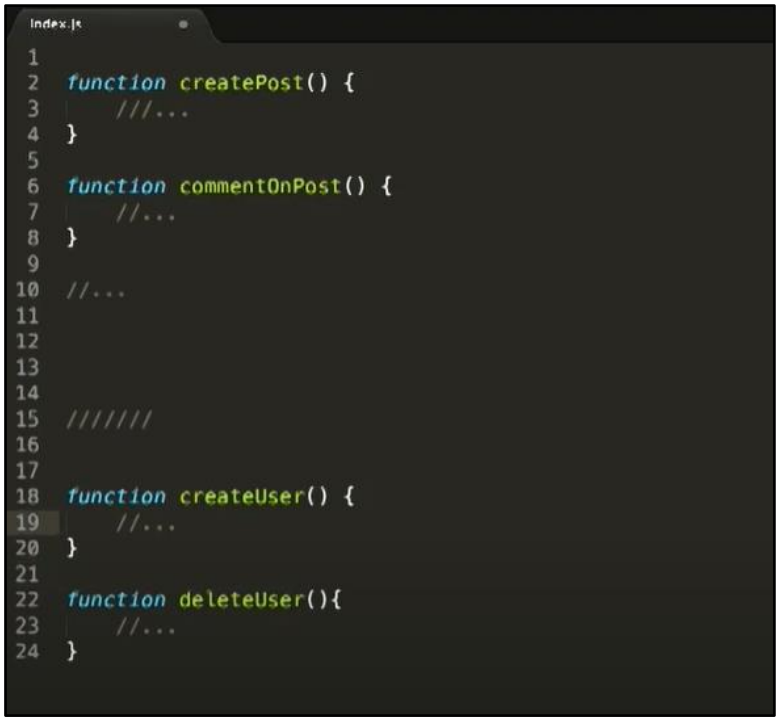
Discussion: In this video, students will learn how to share functions and functionalities between different files. It is crucial to know this as it becomes burdensome to keep all the functions in a single file. This results in the code becoming dirty and unorganized.

Show: Play the video from 0.34 to 1:40.

Content: Explain to the students that creating multiple functions leads to difficulties in managing them in a single file of a project. For instance, there may be more than a hundred functions in a single project.

For example, four functions are created in this project namely, `createPost()`, `commentOnPost()`, `createUser()`, and `deleteUser()`.

Show: Show this image to the students and take their insights on how they think these functions can be more organized.



Show: Play the video from 1:52 to 2:25.

Content: Discuss in detail that two files; namely 'user.js' and 'post.js' are created in this step. 'post.js' deals with the `createPost()` and `commentOnPost()` functions whereas 'user.js' deals with the `createuser()` and `deleteUser()` functions.

Moving further, the respective functions are cut and pasted to their corresponding files. These functions are to be used in the 'index.js' file.

Show: Play the video from 2:31 to 2:53.

Content: Elaborate the fact that the `require` command can be used for enabling the *index.js* file to access the 'post.js' and 'user.js' files. Functions from a file can be easily shared with another file using the `require` command.

Show: Play the video from 2:55 to 4:30.

Content: Explain to the students that functions from a file can be exported using the `exports` module. Once a function is exported, any other file accessing that particular file can use any function from that file.

Conclusion: Give the summary of all the topics that we covered in this session.

QUERIES REGARDING SESSION

CLASS DURATION: 10 MINUTES

FACILITATION GUIDELINES

Ask students if they have any queries regarding the session and resolve the queries.

SESSION 2: DYNAMIC WEB APPLICATIONS USING NODE.JS

2.1 Request and Responses

CLASS DURATION: 15 MINUTES

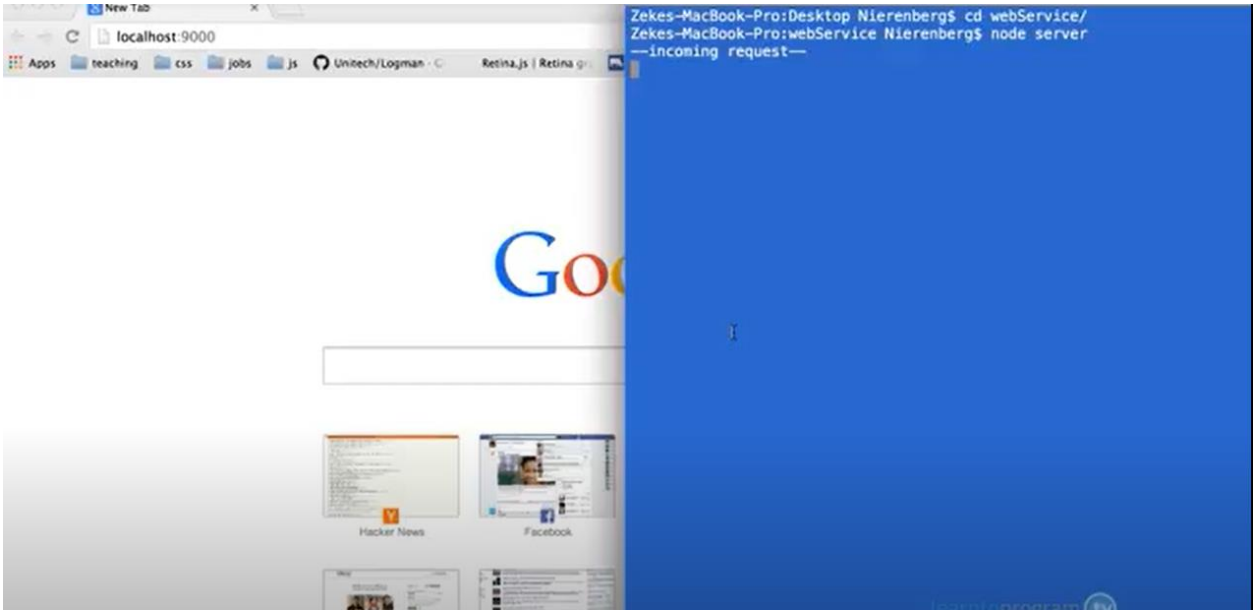
VIDEO DURATION: 12 MINUTES

FACILITATION GUIDELINES -

Greeting: Welcome students to the course. Introduce yourself and seek introduction of students.

During introduction, ask students what they are anticipating from this course.
After the introduction, play the course video.

Show: Play Request and Responses: Session 2 Video – 2.1.



Discussion: Before playing the video, ask students what they know about Node.js basics. Expect some answers from them. Ask them about any real-world example related to Node.js.

Show: Play the video from 0 to 4 MINUTES.

Start the session with the Request and Response topic in Node.js.

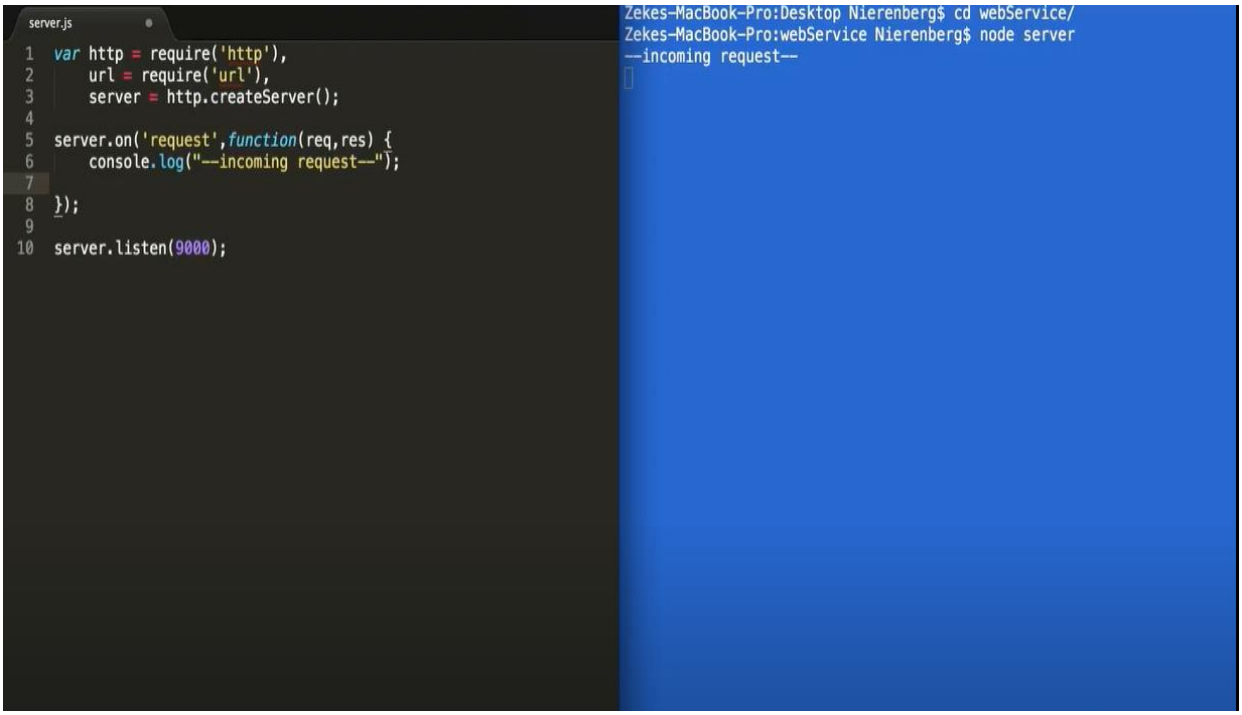
Both the Request and Response objects are callback function arguments in Node.js. The request query, params, content, headers, and cookies are all available to the user. The user can overwrite any value or object in the area. On the other hand, overwriting headers or cookies has no effect on the output to the browser.

Request Body: The request body may be essential to user application when receiving a POST or PUT request. Accessing body data requires a little more effort than accessing request headers. The request object that is provided to a handler implements the `ReadableStream` interface. This stream can be listened to or piped elsewhere. The data can be retrieved directly from the stream by listening to the 'data' and 'end' events.

When dealing with a request, the first thing user should do is look at the method and URL to view what actions should be taken. Node.js makes this relatively straightforward by providing valuable features to the request object.

Show: Use following figure to explain:

- The process of creating an HTTP request
- The importance of using URL and server



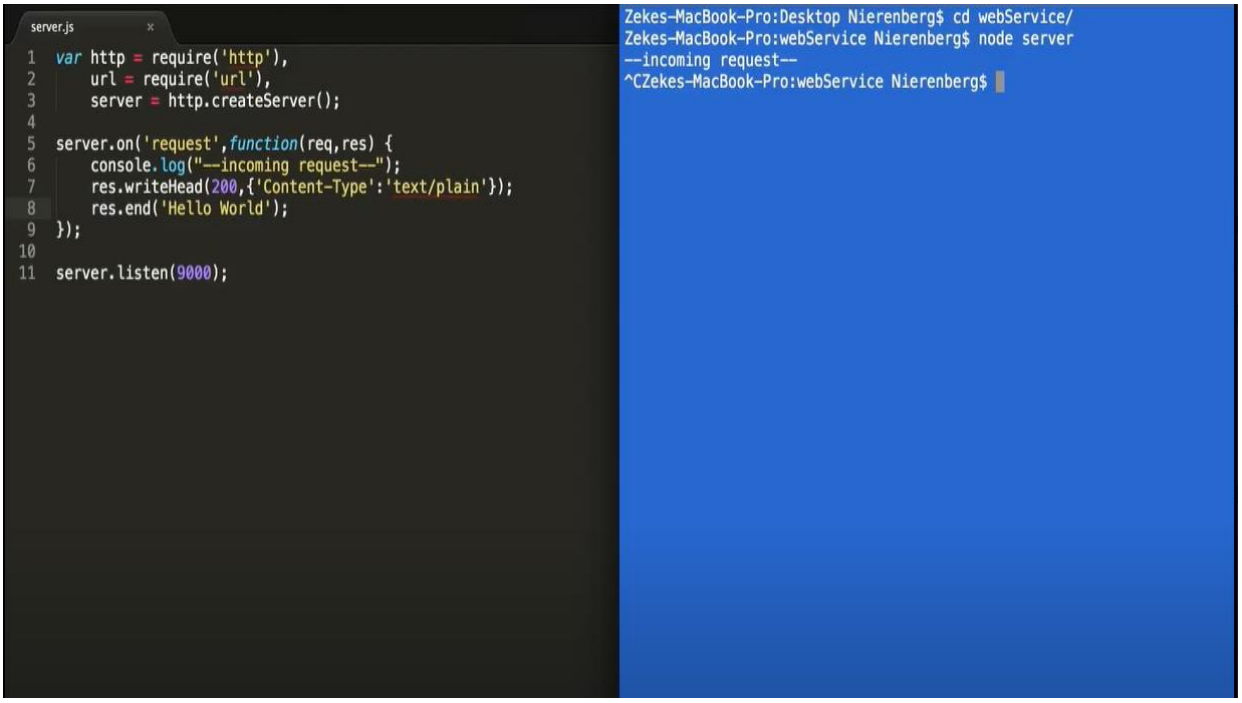
If this code is executed successfully, the user will receive requests but does not respond to them.

If the user tries to visit this code through a Web browser, the request would time out mail nothing is delivered to the client.

The response object, which is a `WritableStream` that is an instance of `ServerResponse`, has not yet been explained. It provides a variety of options for returning data to the client.

Show: Play the video from 4 to 8.

Discussion: Before playing the video, ask students what they understand from the explanation of Request and Response.



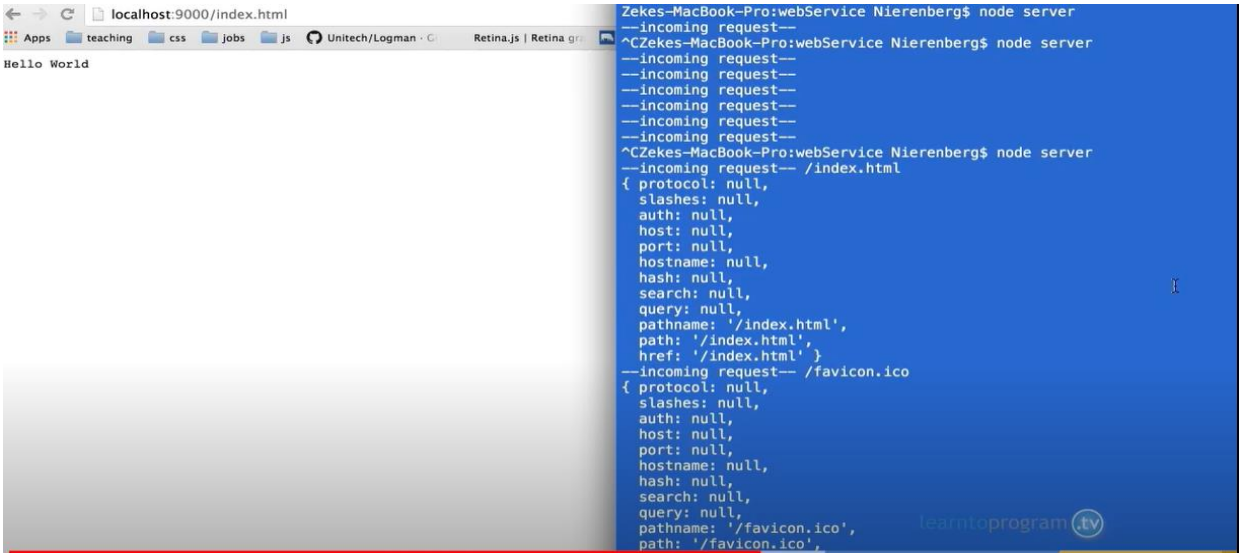
The image attached here explains how to add the response body to the application. The techniques for modifying the headers and status code indicated by the user presume that 'implicit headers' are used. This implies that you are counting on Node to send the headers at the right time before sending the body data.

If the user wishes, the headers can be explicitly written to the response stream. This is accomplished using the `writeHead` method, which publishes the status code and headers to the stream.

After the headers have been set (either implicitly or manually), the user is ready to transmit response data. The end method of a stream can also receive additional data to deliver as the stream's final piece of data.

Make sure the status and headers are set before you start writing data to the body. Since headers arrive before the body in HTTP responses, this makes sense.

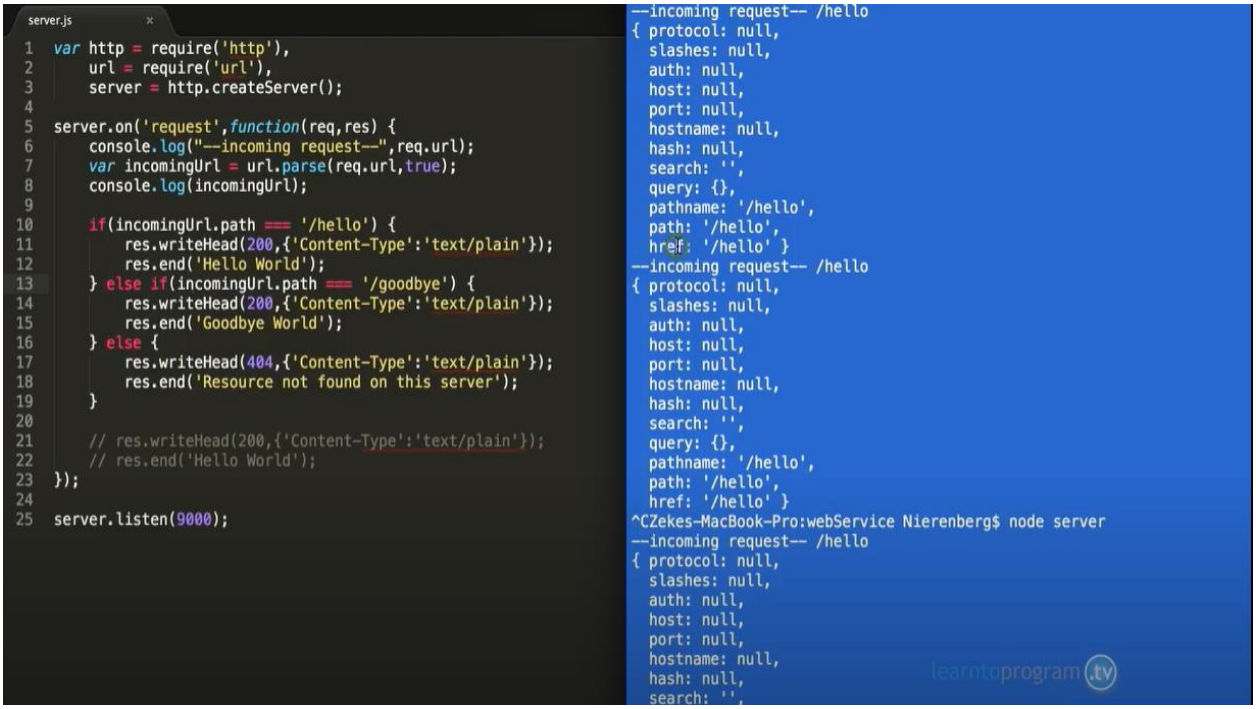
Show: Show image to the students and explain further.



Students can see in the diagram that when a request is submitted and a response is received, it prints 'Hello World'.

For example, create a simple echo server that simply responds with whatever data was received in the request. The user only requires copying data from the request stream to the response stream.

Show: Play the video from 8 to 12.



It is not quite finished yet. As stated multiple times during this video, errors can and do occur, and one must deal with them.

If there are any errors on the request stream, the user will record the error to `stderr` and deliver a 400 status code to indicate a Bad Request. However, in a real-world application, the user wants to look at the error to see what status code and message it should have. When dealing with issues, the user should turn to the Error documentation as usual.

Note:

- Make an HTTP server that listens on a specific port and has a request handler function.
- Headers, URLs, methods, and body data can be retrieved using request objects.
- Make routing decisions based on the URL and/or other data in request objects.
- Headers, HTTP status codes, and body data are sent using response objects.
- Data from request objects to response objects can be piped.
- Handle stream failures in both the request and response streams.

Additional Reference:

Refer to following links for more information:

- https://livecodestream.dev/post/5-ways-to-make-http-requests-in-javascript/
- https://livecodestream.dev/post/5-ways-to-make-http-requests-in-javascript/

In-Class Question: What are the three parts to a response message?

Answer: Each message contains either a client's request or a server's response. There are three elements to them: a start line that describes the message, a block of headers with attributes, and an optional body that contains data.

2.2 Mapping Requests

CLASS DURATION: 20 MINUTES

VIDEO DURATION: 13.50 MINUTES

FACILITATION GUIDELINES -

Show: Play Mapping Requests: Session 2 Video – 2.2.

Discussion:

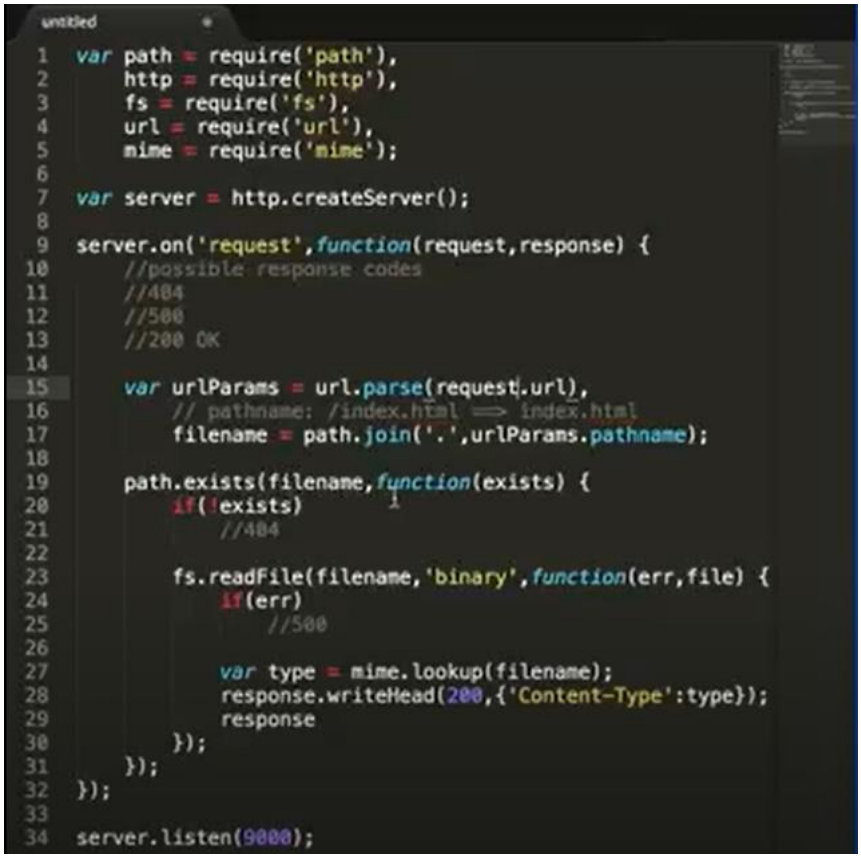
It is crucial to understand that Node.js has a file server before diving into the ideas of Dynamic Web applications. It can be a Web server that recognizes whether the request is HTTP, image, and so on and then sends the request to the client. To accomplish this, the user will require a number of pre-installed programs.

Show: Show following image to the students:

The illustration depicts several packages in the program demonstrating:

- The **path** module that provides utilities for working with file and directory paths.
- That **HTTPS** is the **HTTP** protocol over TLS/SSL. In Node.js, this is implemented as a separate module.
- The **fs** module enables interaction with the file system in a way modeled on standard POSIX functions.
- The **url** module that provides utilities for URL resolution and parsing.
- **Mime** (npm node-mimejs) is a **capturing mock library for Node.js**. It makes use of harmony-reflect Proxy objects (part of the ES 6 JavaScript standard) to make it easy to construct and use capturing fake objects and callbacks in an automated test framework, such as Mocha.

Show: Play the video from 4.15 to 8.20.



```
untitled
var path = require('path'),
    http = require('http'),
    fs = require('fs'),
    url = require('url'),
    mime = require('mime');

var server = http.createServer();

server.on('request',function(request,response) {
    //possible response codes
    //404
    //500
    //200 OK

    var urlParams = url.parse(request.url),
        // pathname: /index.html ==> index.html
        filename = path.join('.',urlParams.pathname);

    path.exists(filename,function(exists) {
        if(!exists)
            //404

        fs.readFile(filename,'binary',function(err,file) {
            if(err)
                //500

            var type = mime.lookup(filename);
            response.writeHead(200,{'Content-Type':type});
            response
        });
    });
});

server.listen(9000);
```

Discussion:

The functions specified in the packages are utilized in the creation of an application.

•.open(filename, [encoding], callback(err, file)): open or create a new file using the built-in File System module.

•.writeFile(filename, data, callback(err)): write data to filename, overwriting any existing data.

•.appendFile(filename, data, callback(err)): append data to the supplied filename.

•.unlink(filename, callback(err)): remove a file.

• .rename(oldFilename, newFilename)

Use `url.parse` to get the supplied filename(). If the file cannot be located, the callback produces a 404 Not Found with the status code 404.

The callback gives a 200 'OK' response code and writes the file data if the file is found.

HOW IT WORKS?

1. If user is unfamiliar with the HTTP protocol, read 'HTTP Basics'.

2. `http.createServer()` returns a new HTTP server created in the application.

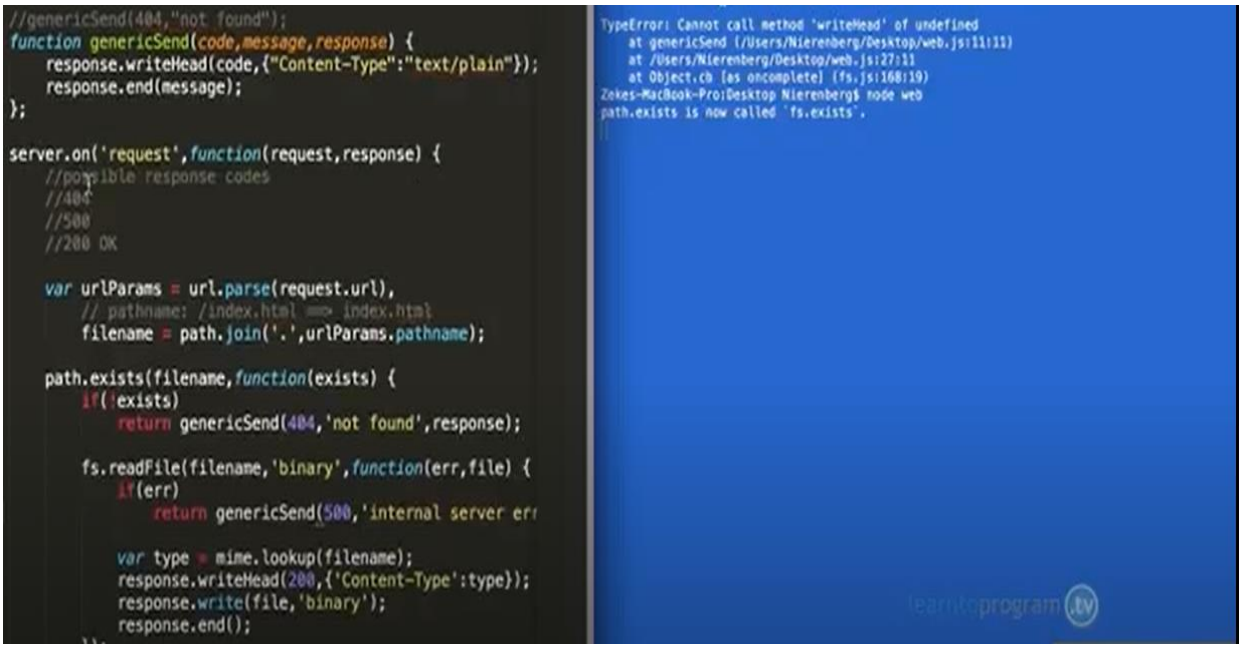
3. Using `the.listen` directive, the server is configured to listen on the provided port and hostname (port, host, callback). The callback method is invoked when the server is ready and it runs `console.log()` to notify us that the server is ready.

4. The request event is triggered whenever a new request is received and it returns two objects: a request (an http.IncomingMessage object) and a response (an http.ServerResponse object).

5. Set the response status Code property to 200 in Code (*OK*) to signal a successful response. Set the response headers Content-Type to MIME 'text/plain' in code. Then write the output, and the response should be closed with an `end ()`.

6. The user may use the F12 Debugger to examine the request and response messages in the 'Network' section.

Show: Play the video from 8.20 to 13.50.



Discussion:

The user has built a parent function in the code called `GenericSend`, which has numerous parameters and sends a plain text message as a response.

If –else condition is used by programmers, it means that the program will respond in some way, such as an error or a message. Finally, when the application has completed successfully, the message is printed.

Additional Reference:

Refer following links for more information:

- https://www.edureka.co/blog/node-js-requests/
- https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/

2.3 Dynamic Web-Applications

CLASS DURATION: 30 MINUTES

VIDEO DURATION: 25 MINUTES

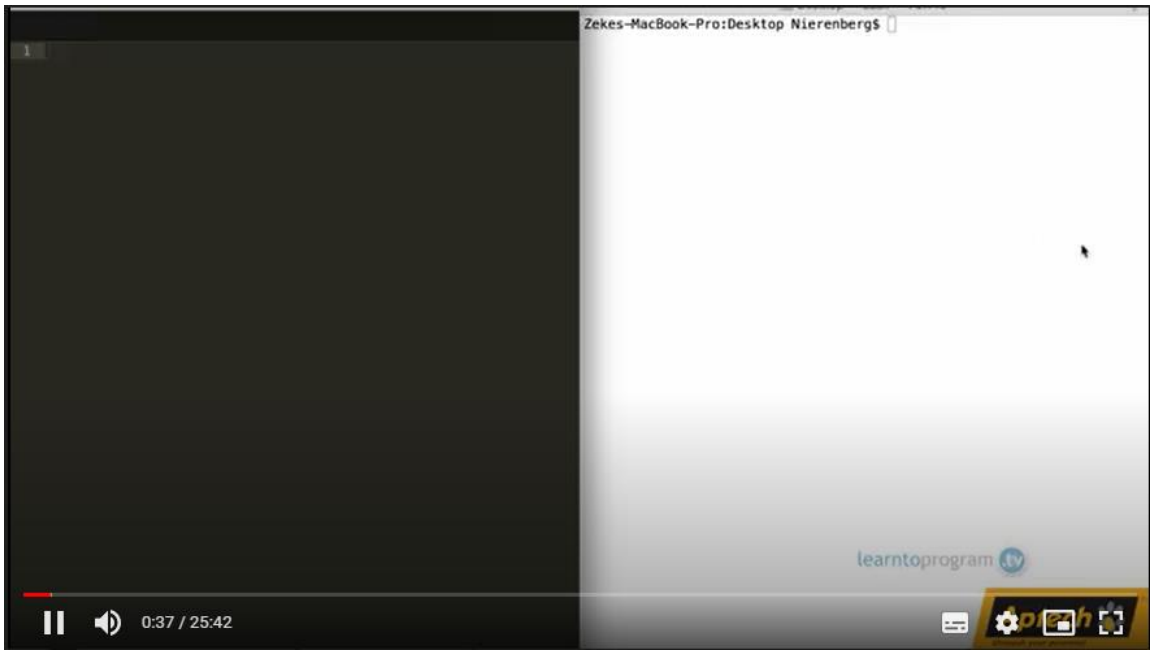
FACILITATION GUIDELINES -

Show: Play Dynamic Web Applications: Session 2 Video – 2.3.

Discussion:

Explain the students the process of creating dynamic Web applications. For understanding the process of creating a dynamic Web application, one must be familiar with installing Node.js, testing Node.js, and creating the first server.

Show: Play the video from 0 to 10.



INSTALLING NODE.JS:

Explain to students how to install on local computer or VPS hosting provider's server. With Node.js, the user may write server-side code in a special variation of JavaScript, allowing them to use a language they are already familiar with.
The Node.js installer includes the package management NPM. NPM is a repository for Node modules, which are reusable code modules that may be used to extend the capabilities of the server. Node modules are code snippets or libraries that work similarly to a plugin repository (depending on how large they are).

Testing the Install:

To check if the installation was successful, ask the students to open a terminal window and type `node -v` and `npm -v`. If the message begins with a v and is followed by a series of digits, the installation was successful (representing a version). The user can now start constructing first server.

Creating First Server:

After creating a static Website, the first step in establishing a Node.js app is to create an Express Web server.

To begin, place all the Website's static files (HTML, CSS, JS, pictures, and so on) in a folder called public and then, create a file called server.js in the root directory of the Website folder.

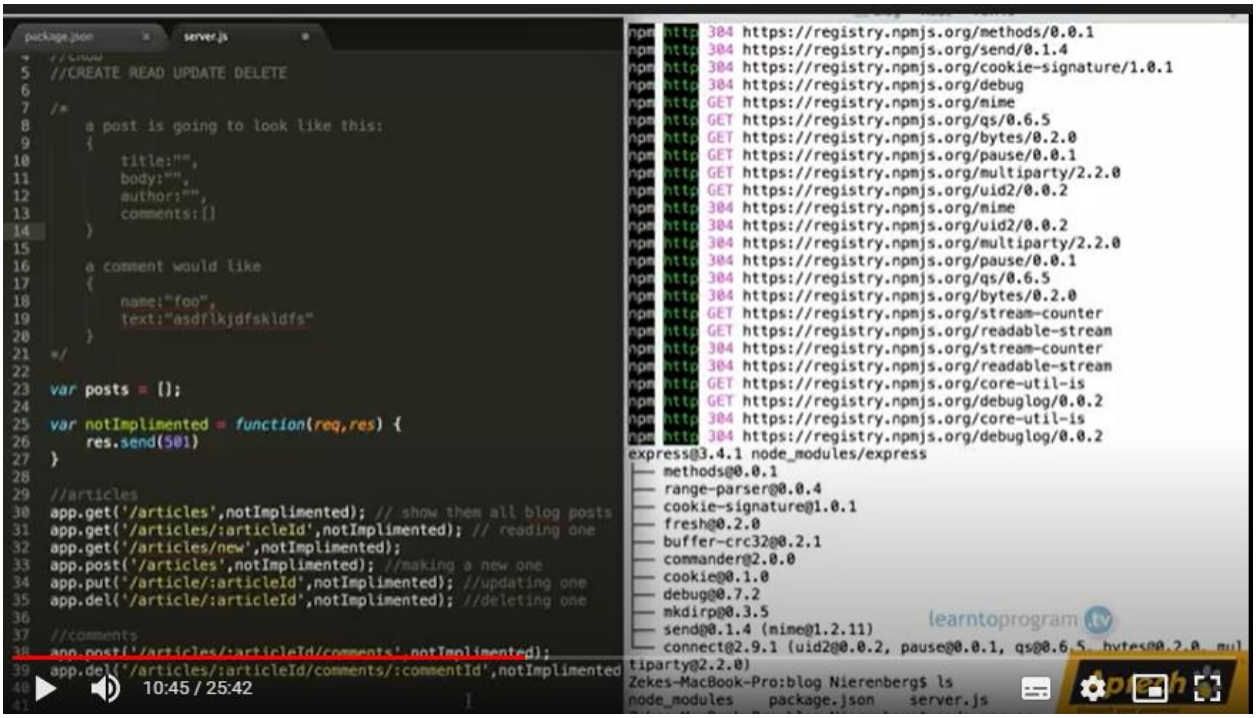
Type `npm init` in the terminal and press Enter to accept the default values for all of the given choices, but make sure server.js is the starting point.

Explain the students how to convert static files to dynamic files using the EJS template engine, as well as how to use partials to transfer code and inject server-side variables into the front-end.

Discussion:

Move forward to explain students how to create a dynamic Web application:

Show: Play the video from 10 to 20.



INSTALLING EJS:

The first step in utilizing EJS is to install it. It would be sufficient to run `npm install ejs –save`. The `–save` parameter saves the module to the `package.json` file so that anyone who clones the git repo (or otherwise downloads the site's files) can use the `npm` install command to install all of the project's required Node modules (known as dependencies) rather than typing `npm install` (module name) for any other modules as well.

CONVERTING STATIC PAGES TO EJS FILES:

After that, turn static HTML pages to dynamic EJS files and organize folders according to EJS's guidelines. In Website's root directory, create a subdirectory named views, and two subfolders called pages and partials inside that folder. Change the.html file extensions to.ejs and place all the HTML files in the pages folder.

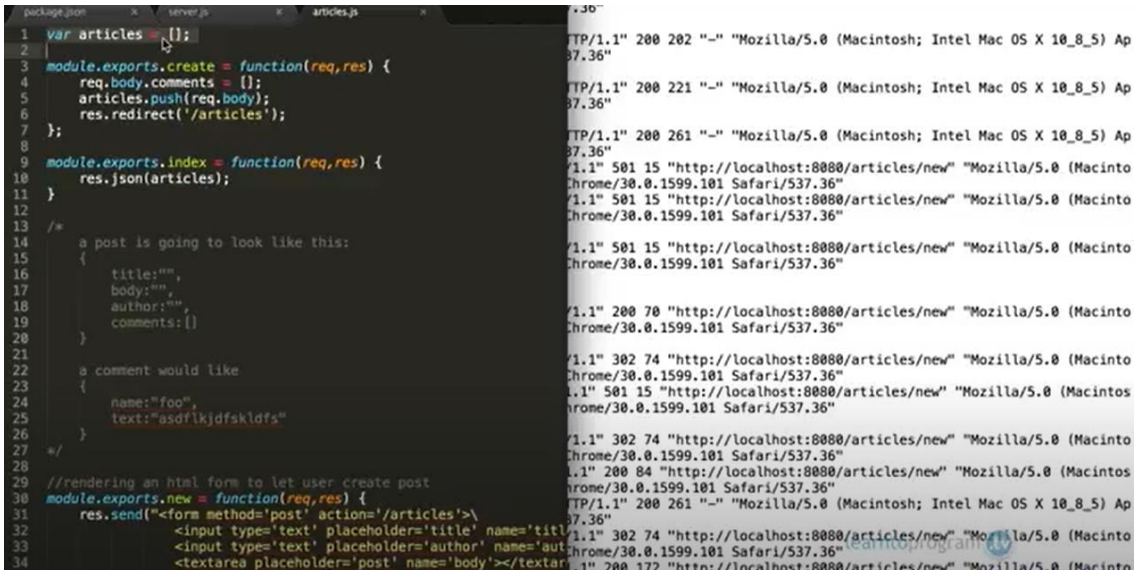
RENDERING EJS PAGES:

Explain to the students how to get the server to render the EJS pages and partials so they can be seen on the front-end.

- Firstly, install the EJS Node module on server.
- Add `constejs = require('ejs');` to the server.js file.
- Second, add `app.set('view engine', 'ejs');` to instruct Express server to utilize EJS.
- Now, it is time to set up the routes. When the user visits a specific URL on the Websites such as **http://testapp.com/login,** routes inform the server on what to do.
- The user will just use the GET routes to show only the EJS pages. After installing the app, add them; server.js has a `listen (8080)` line.

The '/' specifies the Website's URL, while **req** and **res** stand for request and response, respectively. When the user visits `http://testapp.com`, the **pages/index.ejs** page is rendered as the response (shown to the browser). Create routes for the remainder of the EJS pages in the same way.

Show: Play the video from 21 to 25.



Discussion:

PASSING SERVER-SIDE DATA TO THE FRONTEND:

Explain to the students that templating offers the added benefit of allowing the user to pass server-side variables to the frontend, in addition to reusing code. Single variable such as the current user's username, or an array, such as the data of all registered users can be reused. The actual potential of passing server-side variables becomes apparent when using APIs or databases.

The code for constructing a dynamic Web application is shown in the image. It explains how arrays can be used in a Web application. Instead, for a basic array, use this example, which creates a `p` tag for each name in the variable. In the code, the user has used html tags to introduce value instead, Templates can be utilized.

In-Class Question: Is Node.js good for Web development?

Answer: JavaScript became a full-stack technology for Web application development, thanks to Node.js. Node.js is ideal for encoding and broadcasting video and audio, uploading numerous files, and data streaming because to its non-blocking architecture.

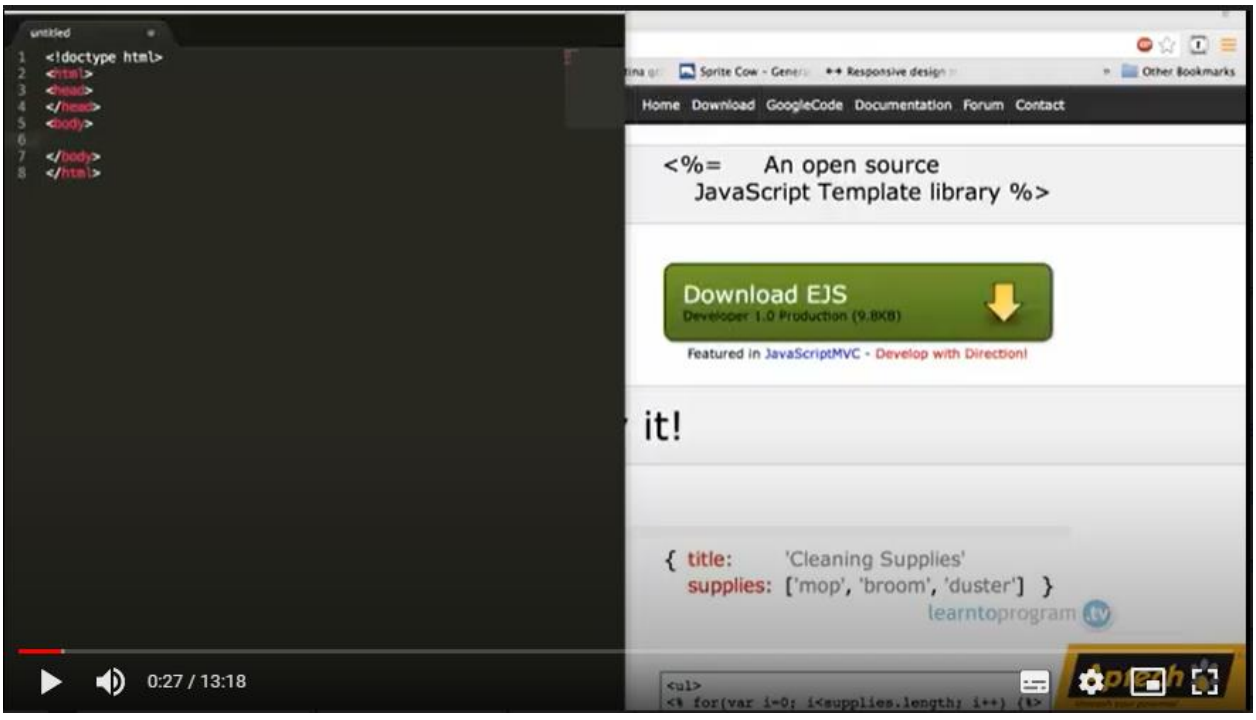
2.4 Embedded JavaScript (EJS)

CLASS DURATION: 20 MINUTES

VIDEO DURATION: 13.18 MINUTES

FACILITATION GUIDELINES -

Show: Play Embedded JavaScript (EJS): Session 2 Video – 2.4.



Discussion:

Now, learn about the very important topic of Embedded JavaScript (EJS) in this video. Embedded JavaScript is abbreviated as EJS. It is one of the most popular template view engines for Node.js and Express.js.

Note: Template engine is a tool that allows the user to create HTML content using specified tags or syntax that will either inject variables into the final output of the template or run programming logic at runtime before delivering it to the browser for display.

Purpose of using EJS:

- When the user requires output HTML with JavaScript, EJS comes in handy.
- If the user is working with dynamic material or content that requires real-time updates, it can drastically reduce the amount of code the user has to write.

Install ejs using npm/yarn: `$npm install ejs`

Common EJS tags:

- <%: This tag is used to include control-flow, conditionals, and no output JS code.
  Note: The Plain Ending tag, often known as percent >, is required to close all tags.

- <%=: The value is written into the template with the HTML escaped. It is used to include JavaScript code that outputs the result of the expression included within the tag.

- <%-: Inserts HTML value into the template without escaping it. When working with partials, it escapes the HTML before it enters the buffer, but percent - does not.

- <%#: To add comments in the file, use the comment tag. During the execution of the script, the content of the tag will be disregarded. The other tags are <%_ ('Whitespace Slurping' Scriptlet tag, it strips all leading whitespace) and <%% (Outputs a literal <%).

Basic server setup with express.js and EJS:

- Use `app.set('view engine', 'ejs')` to make EJS the view engine.
- Create a views directory to house all the ejs files that will render different layouts produced with different partials based on different routes.

Show: Play the video from 7 to 3.18.

Discussion:

Let us learn more about EJS and the platform called Handlebars.

Conditionals:

With the use of <% %>, the user can easily put some conditions in normal HTML.

Loops:

Loops are an essential component of any programming language. It takes the monotony out of doing the same thing over and over.

Handlebars:

Handlebar is a simple templating system. It uses a template and an input object to generate HTML or other text formats. Handlebars templates resemble regular text but incorporate Handlebars expressions.

A handlebars expression is made up of three parts: **Handlebars.js Expressions** some contents and the Handlebars Compile Function. When the template is run, these expressions are replaced with values from an input object.

Additional Reference:

Refer to following links for more information:

https://handlebarsjs.com/
https://devdocs.io/handlebars/

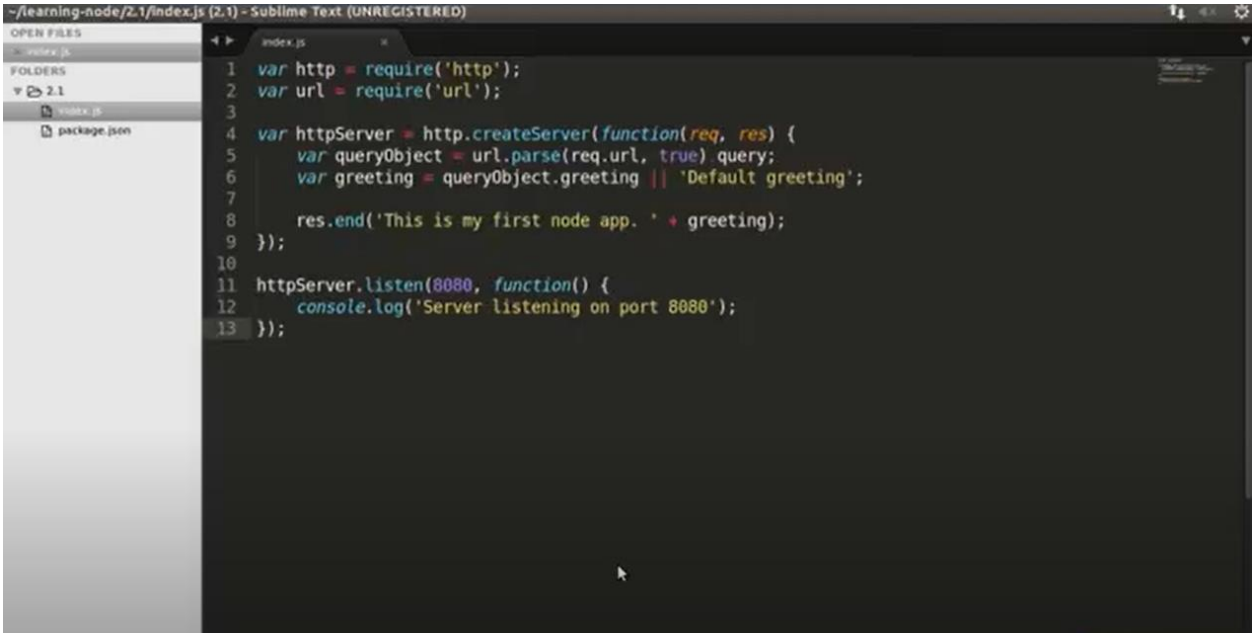
2.5 Creating a Simple Web Server

CLASS DURATION: 5 MINUTES

VIDEO DURATION: 3.52 MINUTES

FACILITATION GUIDELINES -

Show: Play Creating a Simple Web Server: Session 2 Video – 2.5.



Discussion:

In this section, the students will learn how to create a simple Node.js Web server and process HTTP requests.

To access the Web pages of any Web application, the user will require a Web server. For example, the Web server will handle all HTTP requests for the Web application. IIS is an ASP.NET Web server, whereas Apache is for PHP or Java applications.

Node.js allows the user to create their own Web server that can asynchronously handle HTTP requests. A Node.js Web application can be run on IIS or Apache; however, it is recommended that the user uses the Node.js Web server.

Create Node.js Web Server:

Node.js makes it simple to develop an asynchronous Web server that processes incoming requests.

In the example, the http module is imported using the `require()` function. Since the http module is a fundamental Node.js module, the user does not require using NPM to install it. The next step is to run the callback function with the `http createServer()` method and pass request and response parameters. Finally, use the `listen()` function of the server object provided by the `createServer()` method with the port number to begin listening to incoming requests on port 8080. Here, the user can provide any unused port.

To initiate the Web server, ask the students to type `node server.js` in a command prompt or the terminal window.

In-Class Question: How do I create a simple Web app using Node JS?

Answer: In the Scripts object, declare the start object. `node app.js` is the start command. Simply save the file after making changes to the package.json file. Using the `npm start` command, you can now execute the project.

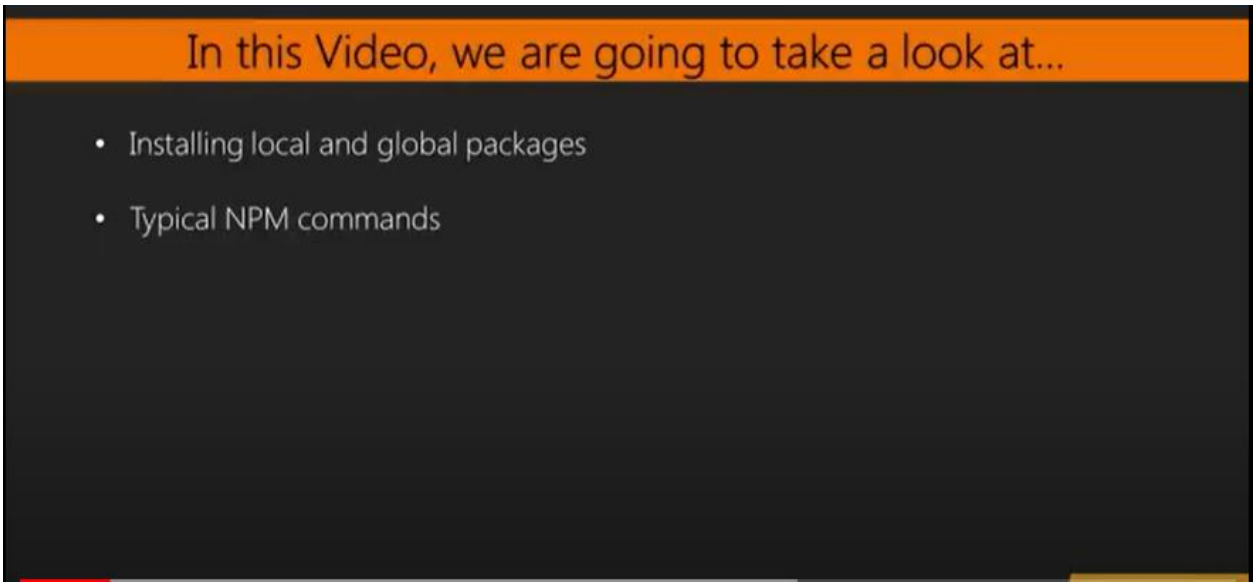
2.6 Using Npm to Install Libraries

CLASS DURATION: 5 MINUTES

VIDEO DURATION: 3.53 MINUTES

FACILITATION GUIDELINES -

Show: Play Using NPM to Install Libraries: Session 2 Video – 2.6.



Discussion:

Explain the students that the NPM command installs a package and any dependencies it has. If a package includes a **package-lock**, **npm shrink wrap**, or **yarn lock** file, those files will be utilized to install dependencies in following order:

- **npm-shrinkwrap.json**
- **package-lock.json**
- **yarn.lock**

npm install (no parameters, in a package directory):

- Install the dependencies to the Node modules folder on local machine.
- It installs the current package context (the current working directory) as a global package in the global mode (with -g or —global applied to the command).
- By default, npm installs all modules mentioned as package dependencies.
- If the —production flag is used, npm will not install modules mentioned in devDependencies (or if the Node ENV environment variable is set to production). The user can use —production=false to install all modules mentioned in both dependencies and devDependencies while the Node ENV environment variable is set to production.

NOTE: When adding a dependency to a project, the production flag has no effect.

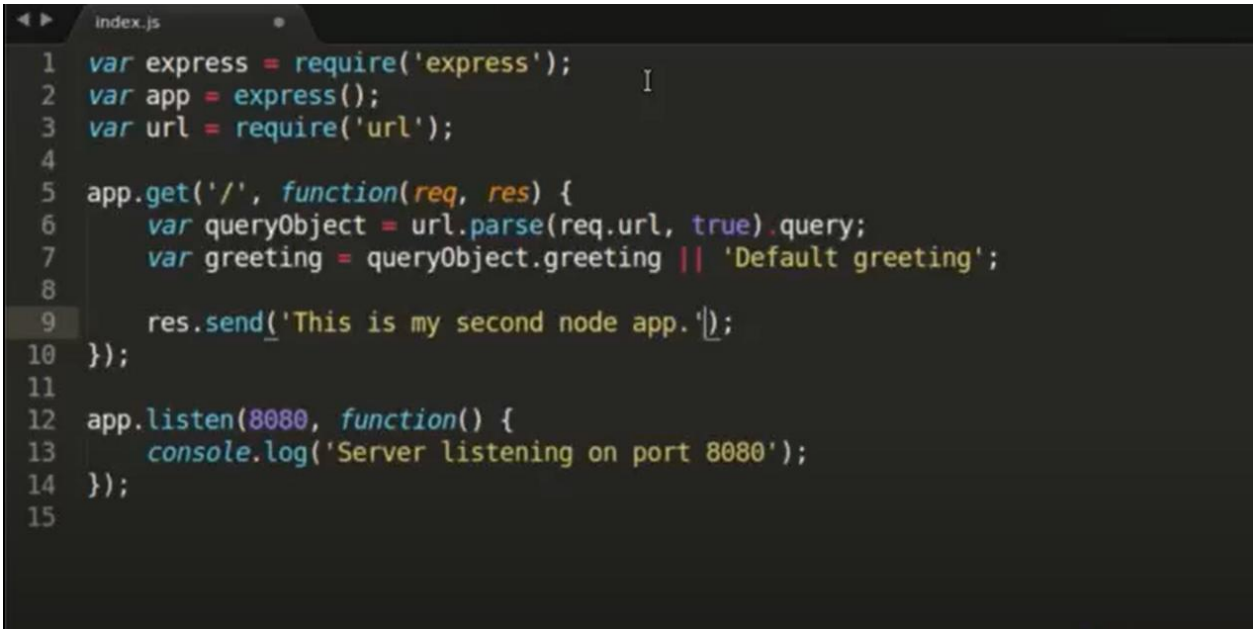
npm install <folder>:

If the user places a folder in the root of the project, its dependencies will be installed. It may be hoisted to the top-level Node modules. `npm` will create a symlink to <folder> instead of installing the package dependencies in that directory if <folder> is not in the root of the project.

2.7 Scaffolding An Express.Js Web App

CLASS DURATION: 5 MINUTES
VIDEO DURATION: 3.37 MINUTES
FACILITATION GUIDELINES -

Show: Play Scaffolding an Express.js Web App: Session 2 Video – 2.7.



Discussion:

Scaffolding makes creating a skeleton for a Web application a breeze. Among other things, the user creates public directories, middleware, and route files manually. A scaffolding tool takes care of all of this for the user, letting them to focus on the application right away.

Getting Started:

Use the express-generator package to install the 'express' command line utility. Express-generator is used to build the application's structure.

Installing express-generator:

- Using the terminal, navigate to the folder where the app will be built out.
- Run following command in the terminal to install express-generator.

```
npm install express-generator -g
```

Scaffolding the app:

The app's scaffolding is depicted in the given diagram. The user can observe that the application's basic structure is being developed if they look closely. The application's structure is being constructed, including public directories, paths, routes, views, and other components.

Explanation: Describes the project's files and folders.

- bin: The www file inside bin is our app's main configuration file.
- public: The public folder contains files that will be made available to the public for use, such as JavaScript scripts, CSS files, and photos.
- routes: The routes folder contains files that contain navigation methods for multiple parts of the map. It contains a number of js files.
- views: The view folder contains numerous files that make up the application's views section.
- For instance, the home page, the registration page, and so on.

Note: At the time of authoring this article, the files' extension was `.jade`. Since the jade project has been renamed to pug, change the file extensions to `.pug`.

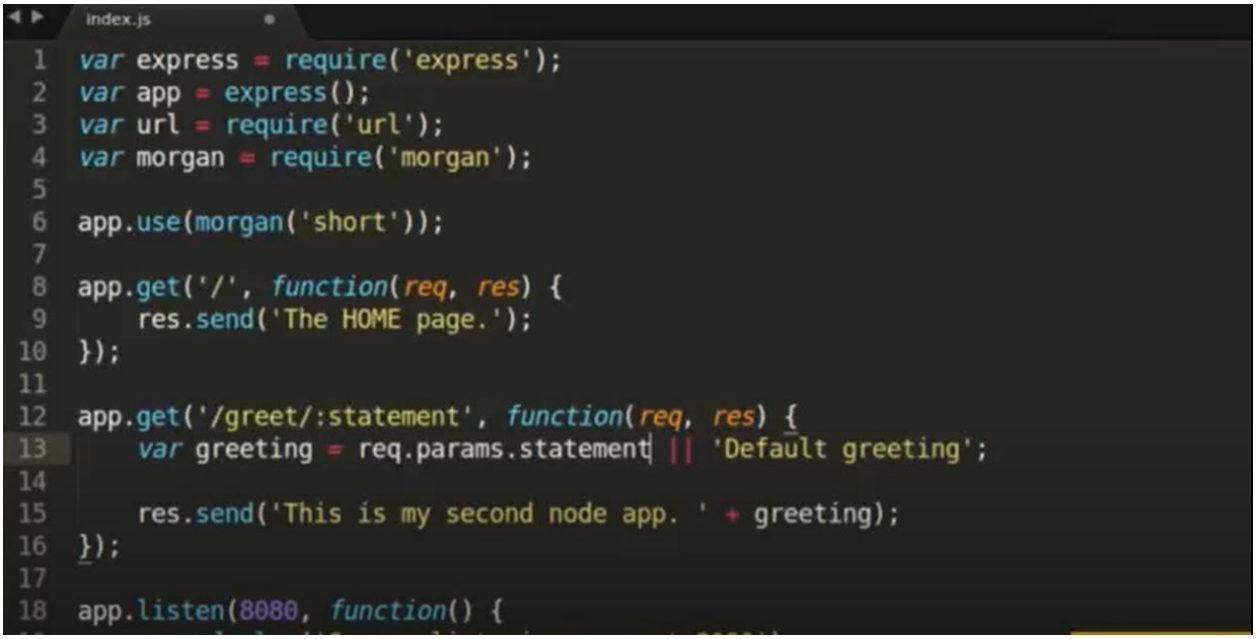
2.8 Understanding Routes and Actions

CLASS DURATION: 5 MINUTES

VIDEO DURATION: 5.25 MINUTES

FACILITATION GUIDELINES -

Show: Play Understanding Routes and Actions: Session 2 Video – 2.8.



Discussion:

Students already know how to make the models because they have watched the previous videos. Following are the main elements that students require to make routes:

- 'Routes' transmit the supported requests (together with any information encoded in request URLs) to the appropriate controller functions.
- Controller functions get data from models, build an HTML page with the data, and give it to the user to view in the browser.
- To render the data, the controllers use views (templates).

A route in Express is a section of code that connects an HTTP verb (GET, POST, PUT, DELETE, and so on.) to a URL path/pattern and a function that handles that pattern.

Routes can be made in a number of different ways. In this video, students will be using the express. Router middleware is useful since it allows you to gather route handlers for a given section of a Website and access them all through a single route-prefix. All the library-related routes will be stored in a 'catalogue' module, and they will be kept separate if the user adds routes for handling user accounts or other functions.

Functions:

The 'about' route, which only responds to HTTP GET queries, is defined using the `Router.get()` method. This method takes two arguments: the URL path and a callback function to be triggered if the path is accepted in an HTTP GET request.

The callback receives three arguments (typically named **req**, **res**, and **next**) that contain the HTTP Request object, HTTP response, and the next middleware chain function.

The Router also provides route methods for all the other HTTP verbs that are mostly used in exactly the same way:

`Post(),put(), delete(), options(), trace(), copy(), lock(), mkcol(), move(), purge(), propfind(), proppatch(), unlock(), report(), mkactivity(), checkout(), merge(), m-search(), notify(), subscribe(), unsubscribe(), patch(), search(), and connect().`

In-Class Question: What does Express `router ()` do?

Answer: To build a new router object, use the express. `Router()` function. When the user creates a new router object in software to process requests, use this function.

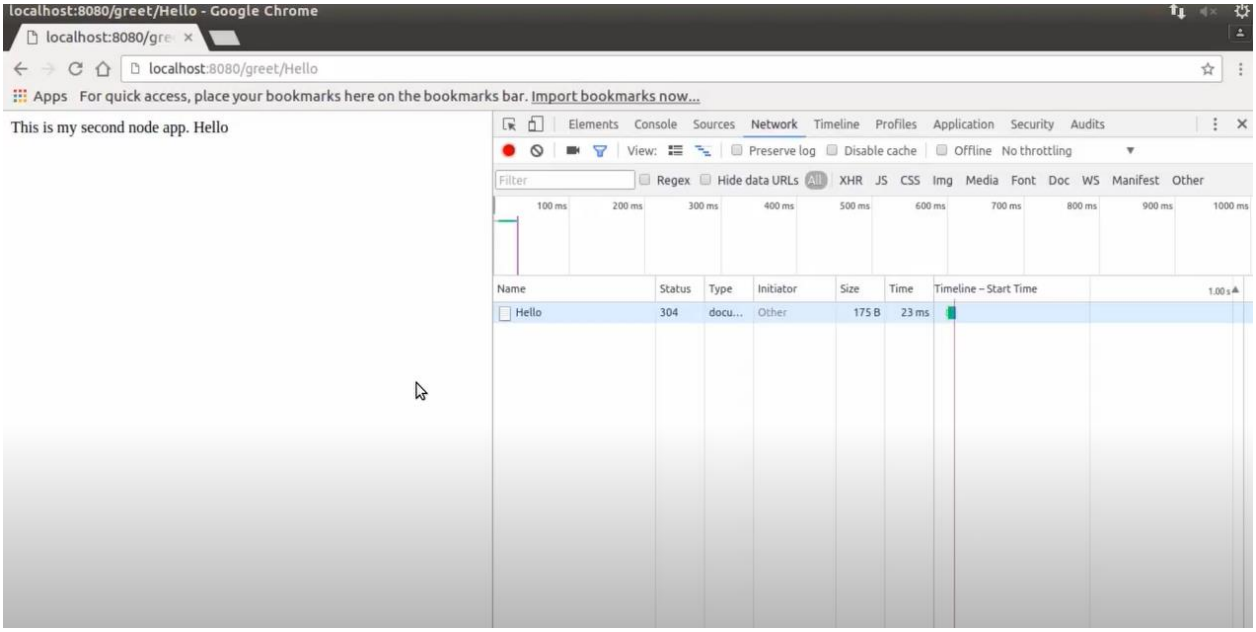
2.9 Serving Http Responses

CLASS DURATION: 5 MINUTES

VIDEO DURATION: 5.11 MINUTES

FACILITATION GUIDELINES -

Show: Play Serving HTTP Responses: Session 2 Video – 2.9.



Discussion:

Explain the students that ('http') is required to use the HTTP server and client. Many of the protocol's previously difficult-to-use capabilities are now handled by Node.js' HTTP interfaces,

especially large, maybe chunk-encoded messages. The user interface ensures that entire queries or responses are not delayed, allowing the user to stream data.

- **Extends: <Stream>**

An HTTP server, not the user, creates this object internally. It is sent to the 'request' event as the second parameter.

- **Event:** `'close'`

The answer has been completed or the underlying connection has been ended prematurely (before the response completion).

- **Event:** `'finish'`

When the response has been sent, this event is emitted. This event is produced when the operating system receives the last section of the response headers and body for transmission over the network. It does not mean the client has not yet got anything.

Response addTrailers(headers)

This method adds HTTP trailing headers to the response (a header at the end of the message).

Trailers will be emitted if chunked encoding is utilized for the response. HTTP requires the Trailer header to be specified, with a set of header fields as its value to emit trailers.

Conclusion: Give the summary of all the topics that are covered in the sessions.

QUERIES REGARDING SESSION

CLASS DURATION: 10 MINUTES

FACILITATION GUIDELINES

Ask students if they have any queries regarding the session and resolve the queries.

SESSION 3: EXPRESS.JS IN DEPTH

Greeting: Welcome the Learners to the third session on Node.js. Inform the learners that this session is in continuation of the previous one in session 2.

QUICK RECAP

Recap: Recall the topics covered in the previous sessions and gauge how much the learners remember. Make the session interactive and open-ended. Urge the learners to ask any doubts that they might have about the previous session. In the end, briefly list down the topics that will be covered in this session.

Start the session.

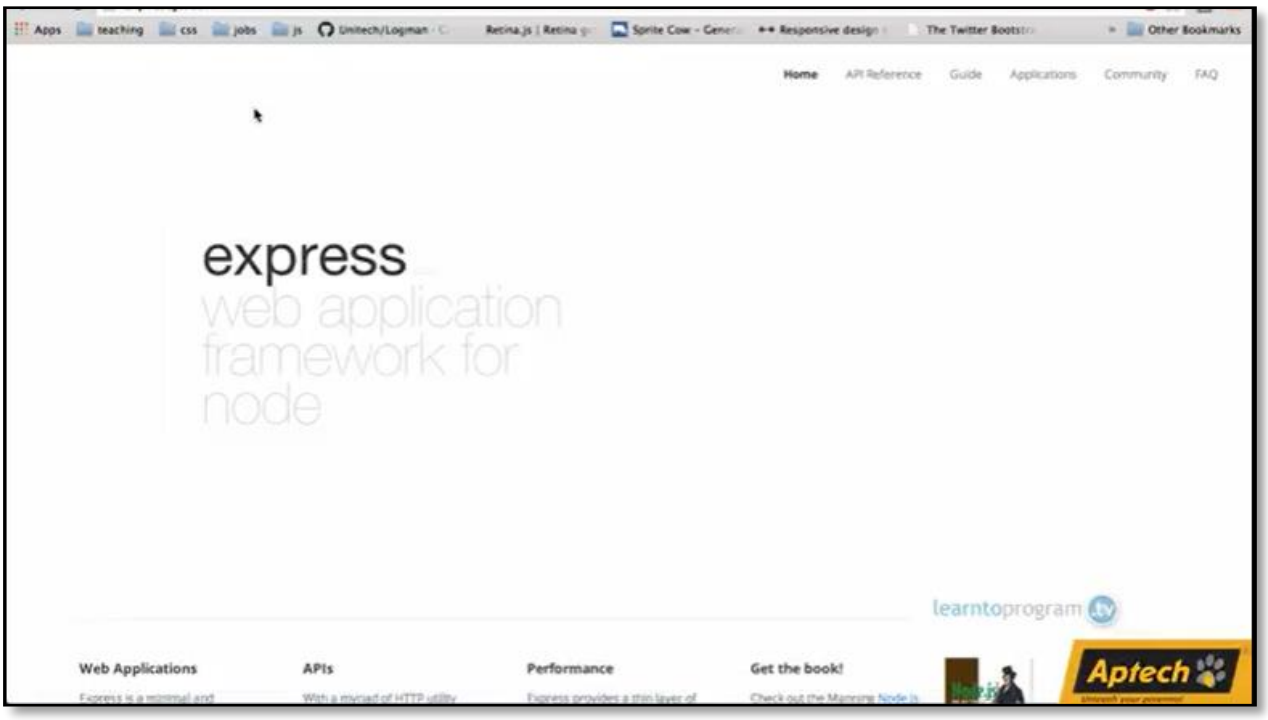
3.1 Express.Js

CLASS DURATION: 20 MINUTES

VIDEO DURATION: 5 MINUTES

FACILITATION GUIDELINES –

Show: Play Express.js: Session 3 Video - 3.1.



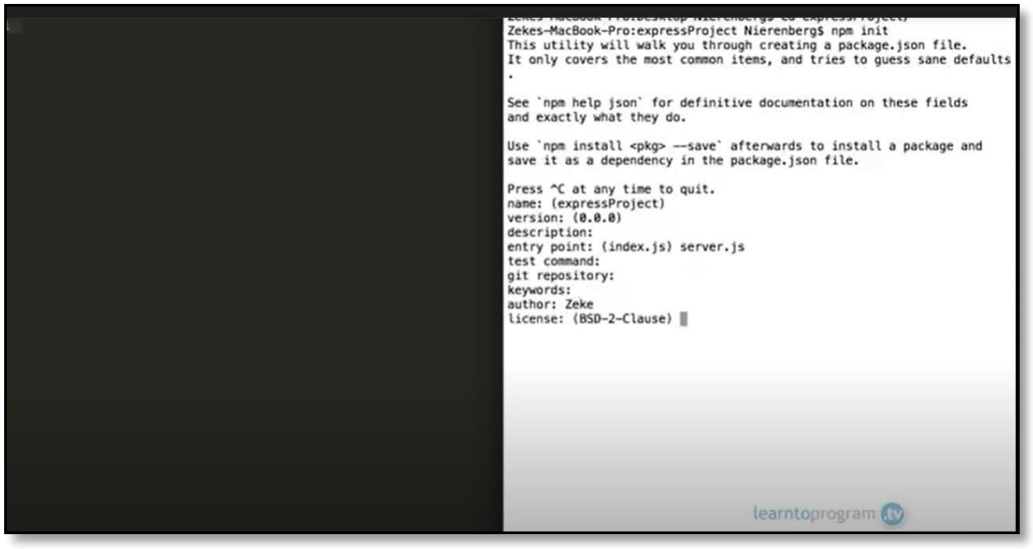
Content: Explain to the students that *Express JS* is an open-source framework for building Web applications and services in Node.js.

Discussion: *Express* expedites the process of writing Web services in Node.js. Discuss the steps for setting up a new project.

Show: Play the video from 0.25 to 3.01.

Content: Describe how a new project is created on the desktop. Tell the students about how an *Express* project is created and the directory is changed to navigate to the project. The `npm init` command is run to initiate the 'Nodes Package Manager' and a `package.json` file that defines dependencies is set up.

It is noteworthy that this step displays the project's details, such as name, version, entry point, and lists all the test commands that can be used.

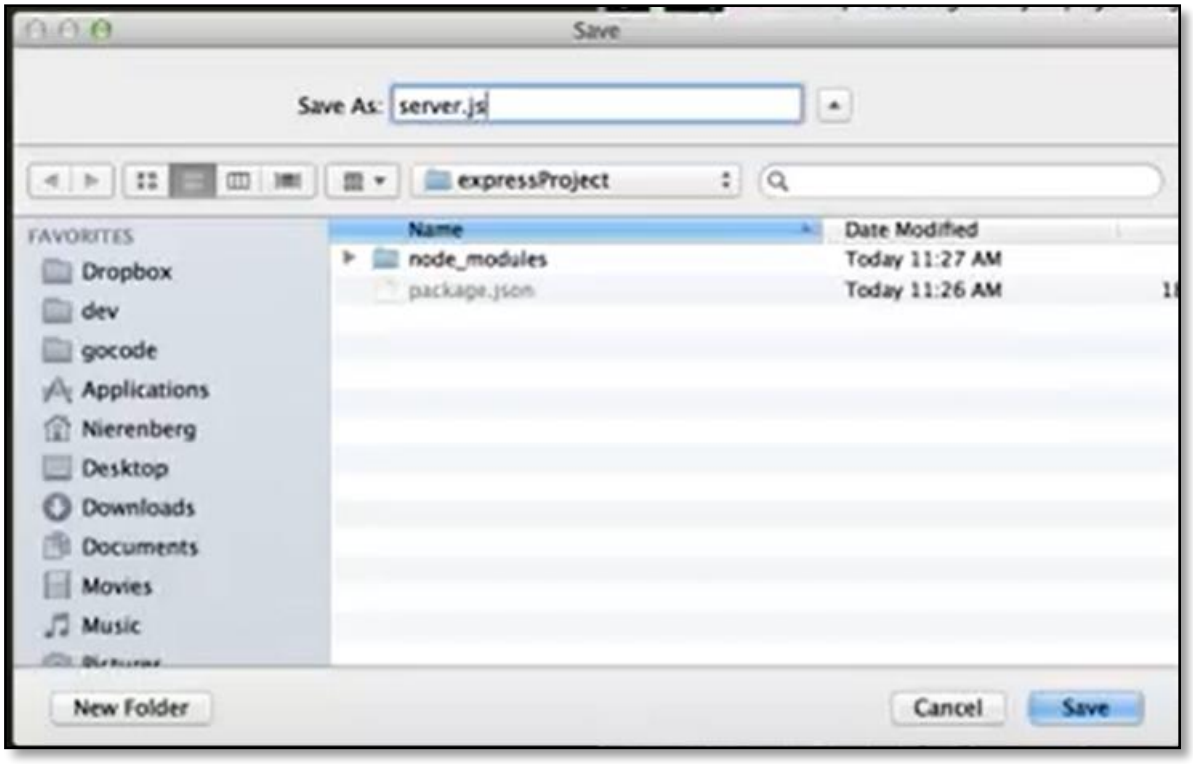


Show: Play the video from 3.10 to 6.56.

Content: Explain that the *package.json* file is opened from the desktop and the displayed details are reviewed. The dependency *Express* has to be specified. The command `npm install` must then be run. This command fetches all the projects and libraries that the framework *Express* depends upon for functioning.

Show: Play the video from 7.03 to 11.00.

Content: Elaborate on the generation of the *package.json* file and a 'node_modules' folder. All the directories under *Express* can be listed. The directory can also be changed to enter 'node_modules' to view its contents. However, the server file must be enabled to specify *package.json*. Next, tell the students that the file must be named and then saved.



Show: Play the video from 11.30 to 11.45.

Content: Discuss with the students that *Express* can be used in the following manner:

1. *Express* is defined in the code. *Node* investigates the modules folder.
2. *Express* is next instantiated. A suitable name is given for the instance. In this case, it is *app*.
3. In *Express*, routes that will go to some actions are defined. A generic example would be *'users/me'* route, which will look up the user and render a profile. *Express* takes a request, checks the route, and then, gives a suitable response for the same.

Examples: Give the students some examples for a deeper understanding. `App.get()` is an example request where '`get`' implies one of the many types of requests that the server receives. When the '`get`' request arrives on the route 'hello', you will respond with the *`request response`* function. The final response sent thus, becomes `Hello World`.

Make the students repeat this for the route 'goodbye' and the final response is 'goodbye world'. To turn the server on, let the students know that they will have to request 'listen'. The request 'listen' requires only one parameter, which is the port to listen on.

The outcome response will be *server is running on 8080*. The 'all' request includes all requests to the server on all routes, such as 'Post' and 'Delete' and the `request response` will be '404'.

Although 'all' is an all-inclusive request, independent requests are essential since *Express* processes requests in the order in which it receives them.

Loggers:

1. The *Express* requests cannot be seen in the Terminal. However, *Express* has a few built-in loggers.
2. Use the method '`us`' that defines the middleware to be used for other packages.

Example:

In *Express*, the *express.logger* request outcomes on the server as the localhost address information and others. More colorful loggers are expressed in a similar manner.

Additional Reference:

Refer to the following link for more information: https://expressjs.com/

3.2 Middleware and Serving Static Files

CLASS DURATION: 20 MINUTES

VIDEO DURATION: 5.36 MINUTES

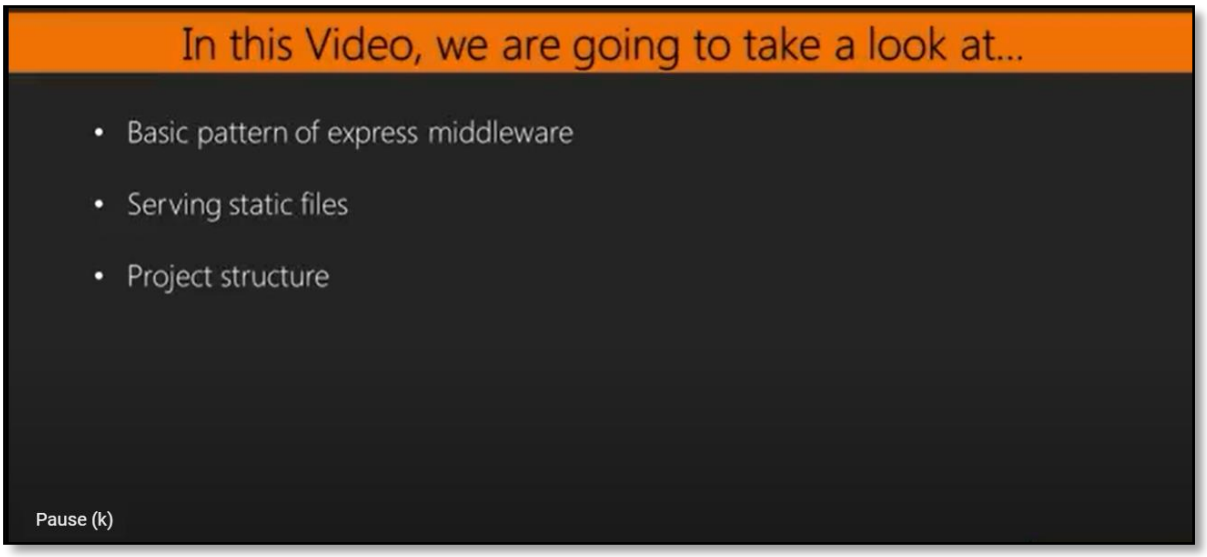
FACILITATION GUIDELINES -

Show: Play Middleware and Serving Static Files: Session 3 Video - 3.2.

Content: Explain to the students that *Express* is an important and powerful framework due to the following two features:

- Flexibility
- Middleware functionality

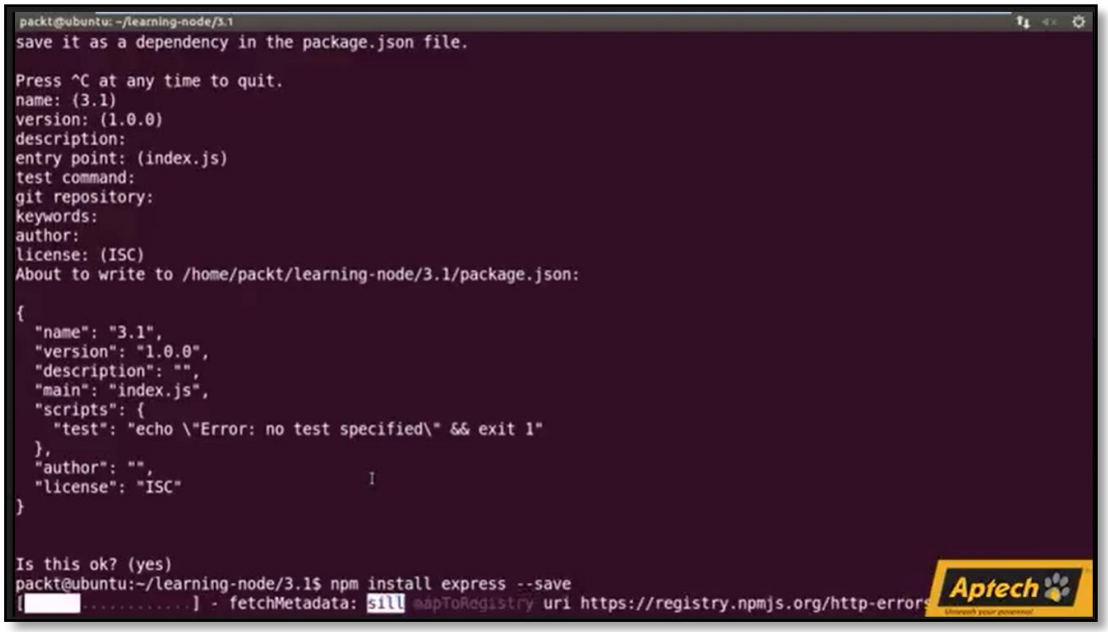
Description: Explain that almost anything can be achieved while creating Web applications with these two features. In this video, students will learn how to structure a Node project. They will also learn how to structure both dynamic and static Web assets along with examples and static HTML pages.



Show: Play the video from 1:10 to 3:10.

Content: Discuss the following steps with students on how to kick start the project:

1. The first step is to create a new video, *3.1*, and run `npm` in it. Then, install the *Express framework*.



2. Change to *Sublime Text* and open the *3.1* project folder. Create the file *index.js* and start creating a skeleton code for an *Express* Web application.

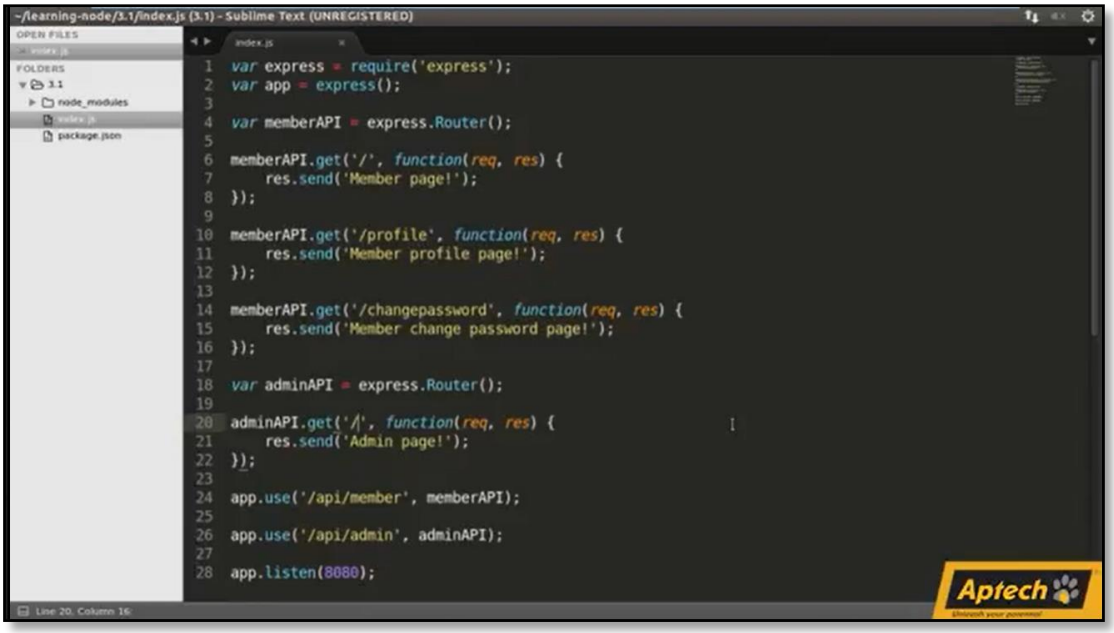
3. There are two user groups to access the application. One is the **member** and the other is the **admin**. Hence, there are two `get` method handlers. The first one is */api/member* and the second one is */api/admin*. Finally, make `app.listen(8080)`.

4. The next step is to add a few more URLs. Copy 'get' and change the path to *api/member/profile*. Change the text associated with the message to 'Member profile page'. Add another 'get' method.

5. *Express Middleware* enables the creation of a router for all member functions and assigns it to the variable member API. Change the app to *memberAPI*. Make one single router for the admin functions.

   ***Note**: Currently, only one member function is obtainable, but this method will provide future flexibility.*

6. Next, group the **member** and **admin** functions to their routers and link them to the *Express*. In *app.listen*, link the *memberAPI* to the */api/member/* using `app.use`. Repeat the same procedure for the *adminAPI* router.
7. Remove the repetitive path slash from *API/member*. Use the same process to change password methods in the profile. Remove */api/admin* in the `admin.get` method.

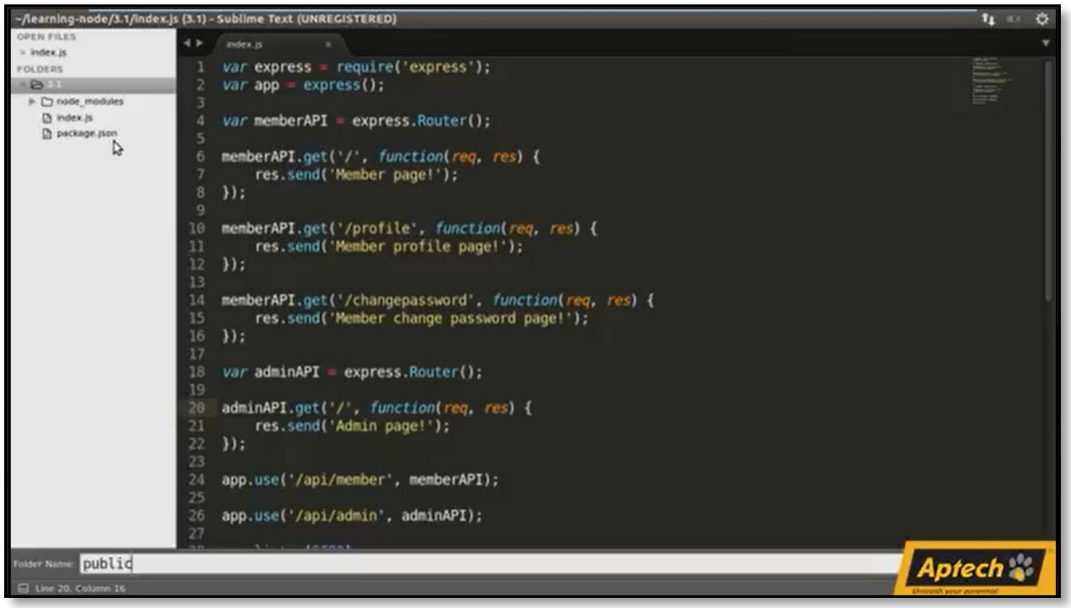


Show: Play the video from 3:10 to 4:10.

Content: Illustrate to the students how to test the member functions.

Explain to the students that they have to run the app in the Terminal by typing `nodemonindex` and pressing Enter. Inform them that they have to open the browser and type *localhost: 8080/api/member*. This will result in the successful opening of the member page. Likewise, explain to them that all member functions `apiadmin`, `profile`, and `changepassword` must be tested.

Furthermore, explain that the routers must be utilized to map a request to the method handlers. Tell them that every Web application has static content, such as static HTML pages, images, CSS, and so on. Let them know that *Express* enables to serve the static assets too.



For the app:

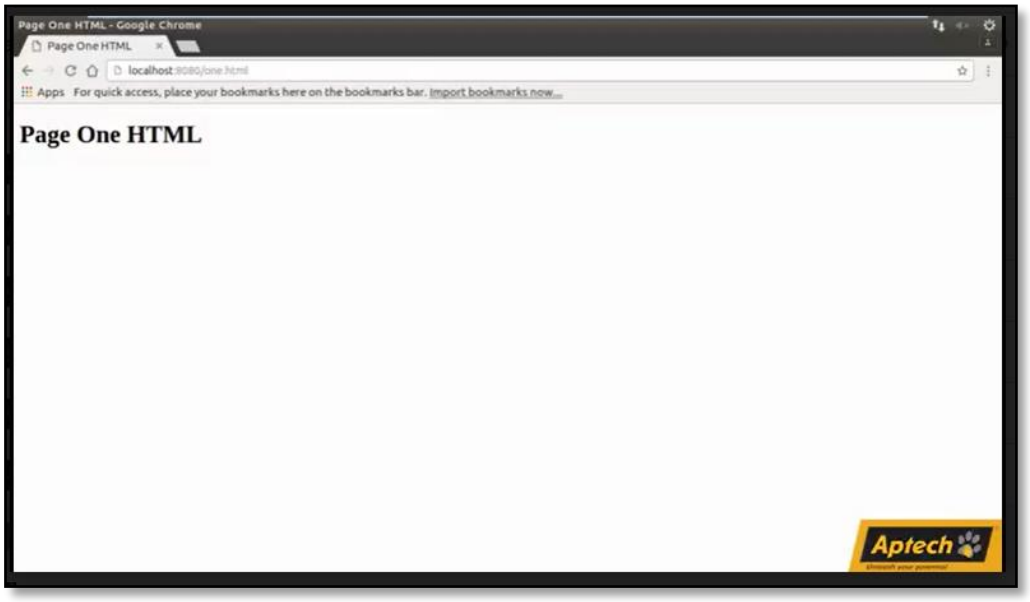
Show: Play the video from 4:10 to 4:38.

Content: Discuss the procedure for serving static resources.

Following is the procedure to serve static resources:

1. First, instruct the students to create a new folder 'public'. Make an HTML page named *one.html*. Then, configure *Express* and search static resources in all public resources with the use method.

2. Now, indicate the folder that contains static resources by using the `static` method of *Express*. This method acts as a router. Open Chrome Web browser and type *one.html*.

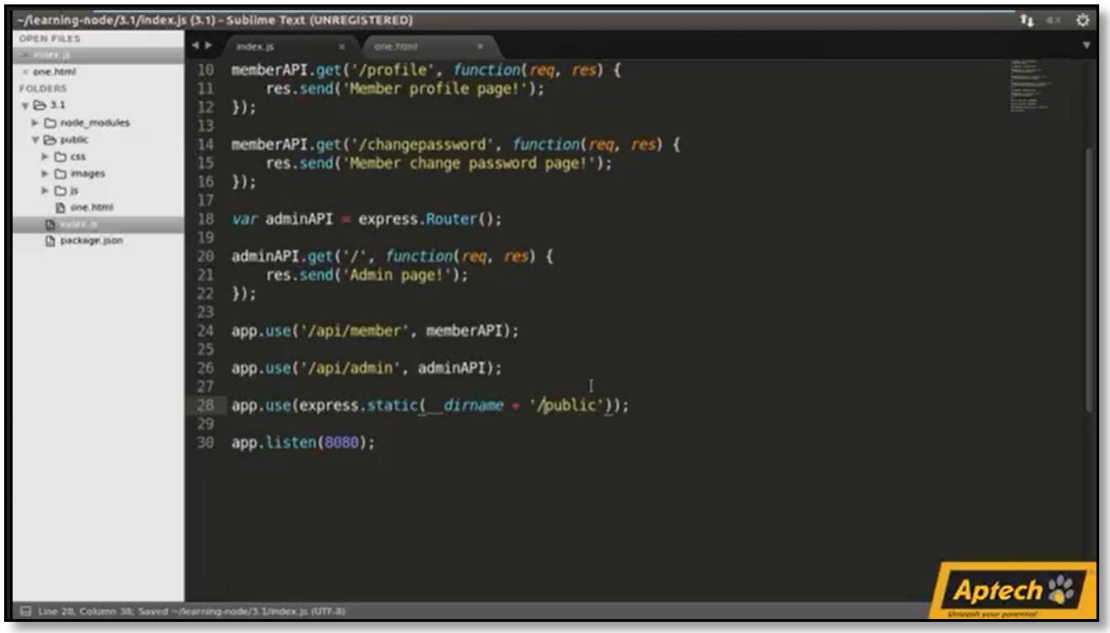


It is important to let the students know that *one.html* serves the static HTML page. Moreover, tell them that new subfolders are usually created publicly that hold different assets.

Example:

css for cascading style sheet files, *js* for JavaScript, images for *png*, *jpeg*, and so on.

Finally, the path for the static function corresponds to the directory from where the Node process is run. If the *Express* app is run from another directory, it is safer to use the absolute directory path. Therefore, tell the students that they must modify the path to double underscore.



In-Class Question: What is *Express Middleware*?

Answer: Middleware is a series of services that caters to applications for several services like Data Management, Messaging, and API Management. In the *Express* router, the next function triggers the Middleware, which in turn controls the request object and response object cycles.

In-Class Question: Explain static content with examples?

Answer: Static content does not change and delivers faster than dynamic content. Content that does not change in response to the input from users is static. For example, JavaScript files and CSS files.

In-Class Question: How is static content served?

Answer: The Middleware functionality in the *Express* router serves the static files. For example, NGINX and NGINX Plus, once configured, serves static content with type-specific root directories with performance optimizations.

Additional References:

https://expressjs.com/en/guide/writing-middleware.html

https://blog.stackpath.com/static-content/

https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/

3.3 Using Template Engine - Jade

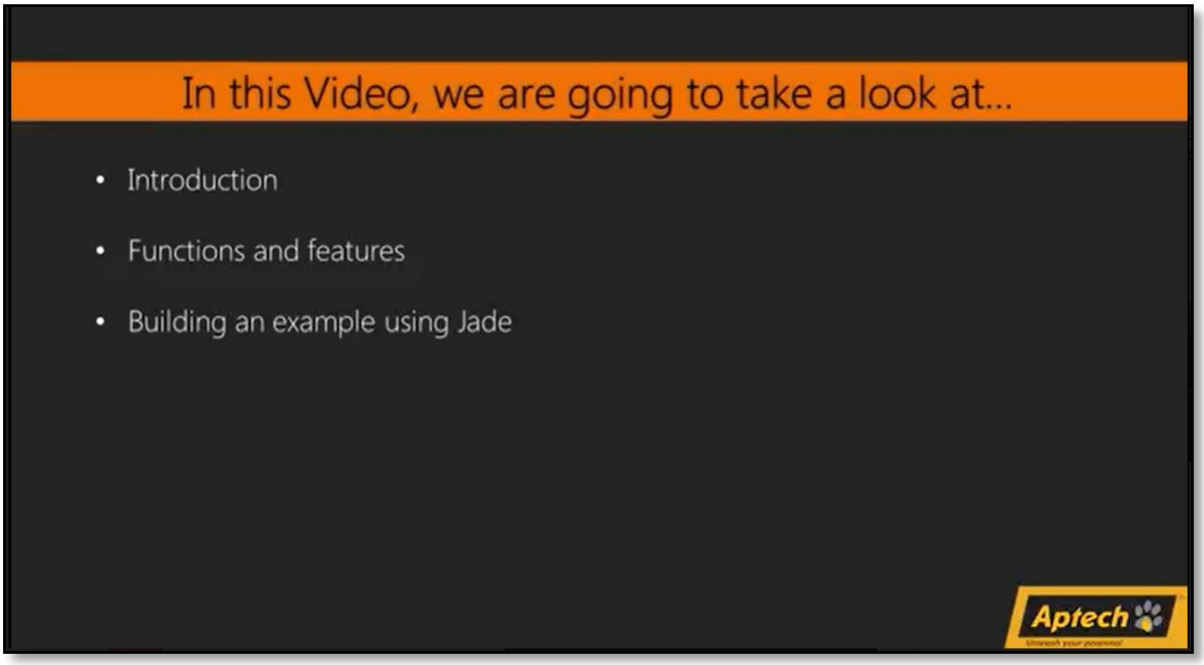
CLASS DURATION: 20 MINUTES

VIDEO DURATION: 4.16 MINUTES

FACILITATION GUIDELINES –

Show: Play Using Template Engine – Jade: Session 3 Video - 3.3.

Content: Explain to the students that **Jade** is a well-known Web template engine for Node. Provide an outline of Jade, its features, and how to configure *Express* to use it and render HTML pages.

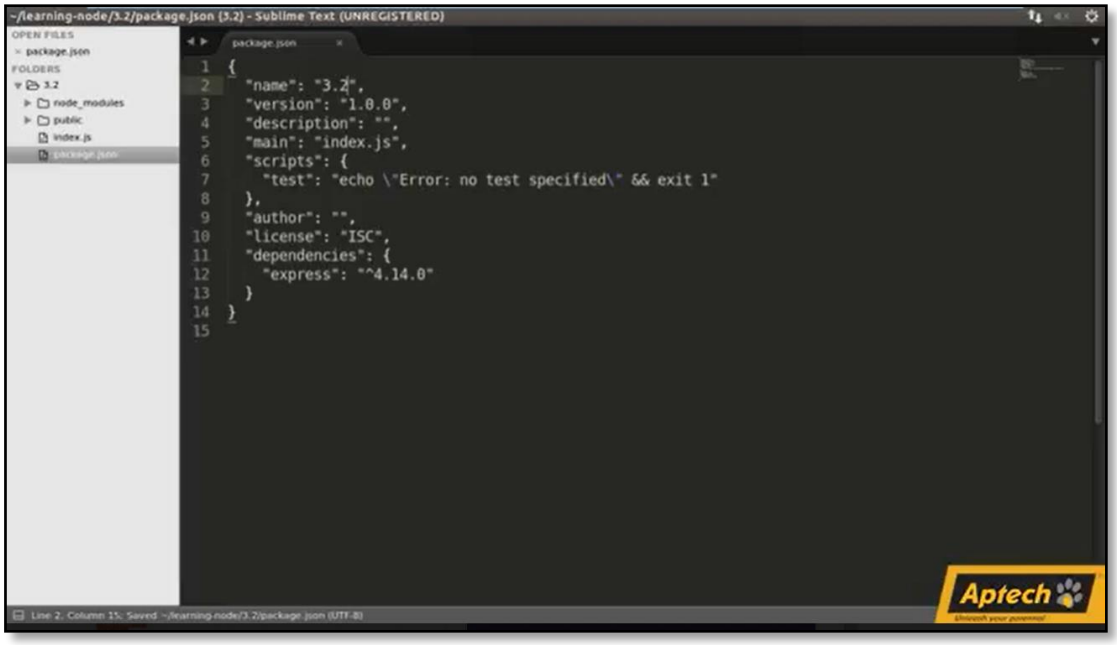


Show: Play the video from 0.36 to 1.28.

Content: Discuss how to get started with Jade.

Process: Tell the students that they must go through the following steps to get started with Jade:

1. Create a folder *3.2* and change the navigation directory to *3.2*.
2. Copy the files created from the Middleware and serve the static section to this directory.
3. Switch to *Sublime Text* and open the *3.2* directory.
4. Navigate to *Open.Json* and change the name to *3.2*.



5. Browse the official Website of Jade. You will notice that two pages, Jade template, and HTML, open in parallel.

Jade has following advantages over HTML:

Jade	HTML
Jade removes the necessity for **ntags** that are essential in HTML	HTML requires **ntags**
Jade is easier to use and is much cleaner	HTML is not as developer-friendly as Jade

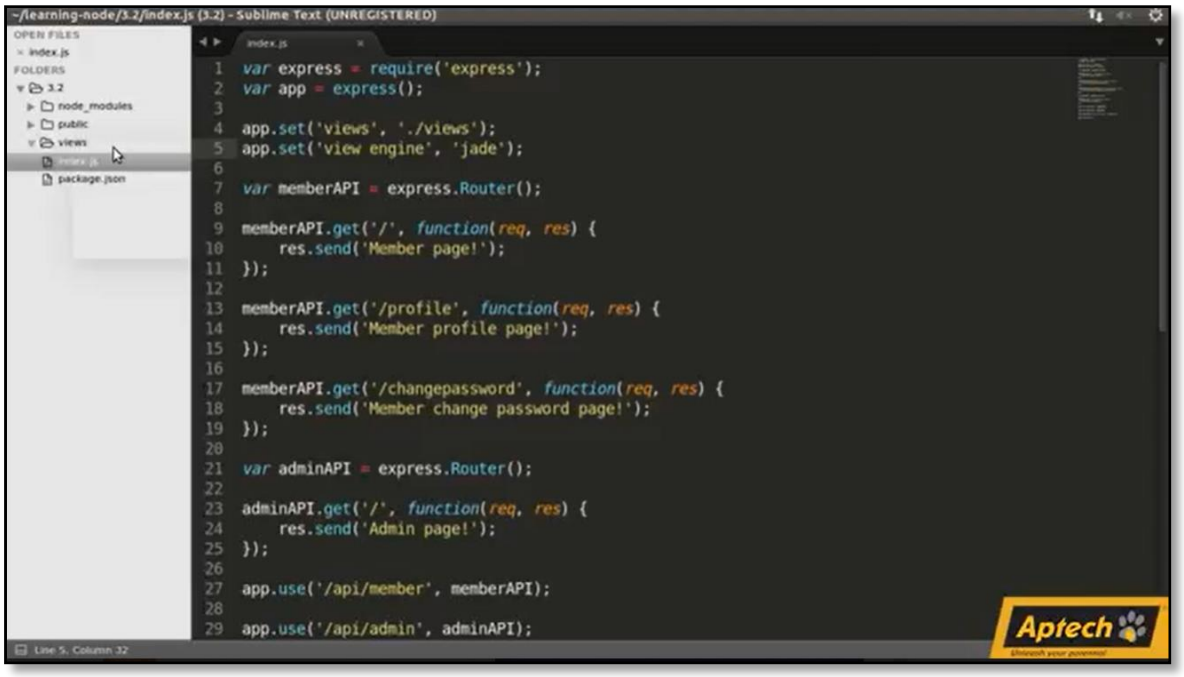
Show: Play the video from 1.28 to 4.16.

Content: Discuss the steps of using Jade in Web apps with the students.

1. Instruct them to install Jade by changing to the terminal and typing `npm install jade - - save`

2. Then, they must create a folder named *Views* in which, all Jade templates will be saved. Inform students that Jade can be enabled in their app through these two sub-steps:

- Configure *Express* to use Jade.
- To locate the jade template, type `app.set`, and set the 'viewengine' to 'jade'.



3. In the *Views* folder, tell them to create a template *two.jad*. They have to start with `html`. Further, ask them to press Tab, then type `head`, again press Tab, and type `title = title`.

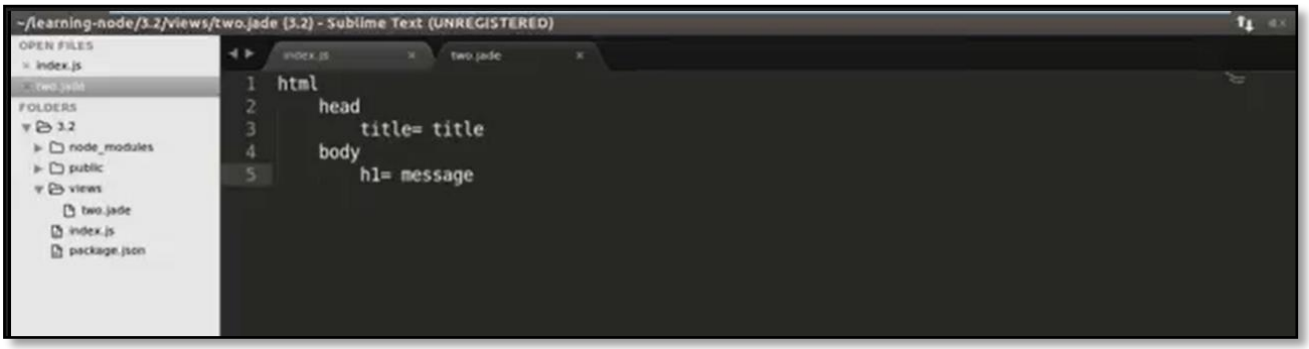
***Note**: The title will not be passed from the Web app to this template.*

4. Explain the last step that involves pressing SHIFT+TAB, typing `body`, pressing Tab again, and typing `h1=message`.

***Note**: The message will be passed from the Web application to this template.*

Explain to the students that the first Jade template is ready. Ask them to save it.

***Note**: Jade can perform complicated functions.*



Show: Play the video from 2.27 to 4.05.

Content: Explain to students about linking a template to the middleware chain.

Let them know that they must use 'render' to make use of the template. Moreover, the title and message variables are defined next.

Following are the parameters used in the render statement:

- Template name
- Object to the values that pass to the template

Discuss the following steps to test the process:

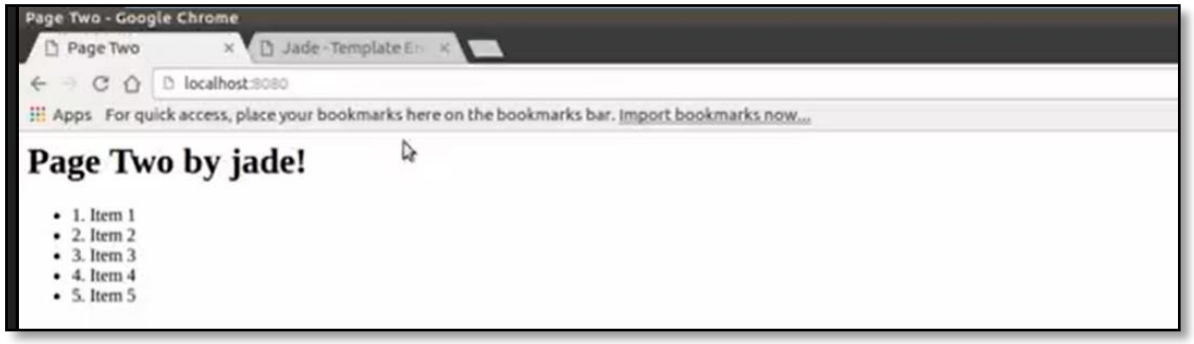
1. Tell the students to change to the terminal `nodemon` index and navigate to the browser *localhost: 8080*. Let them notice that Jade is now active and running with the app.

2. Next step is to define `div`, and set its CSS class, and ID respectively. Tell students that they can also write JavaScript code in a Jade template.

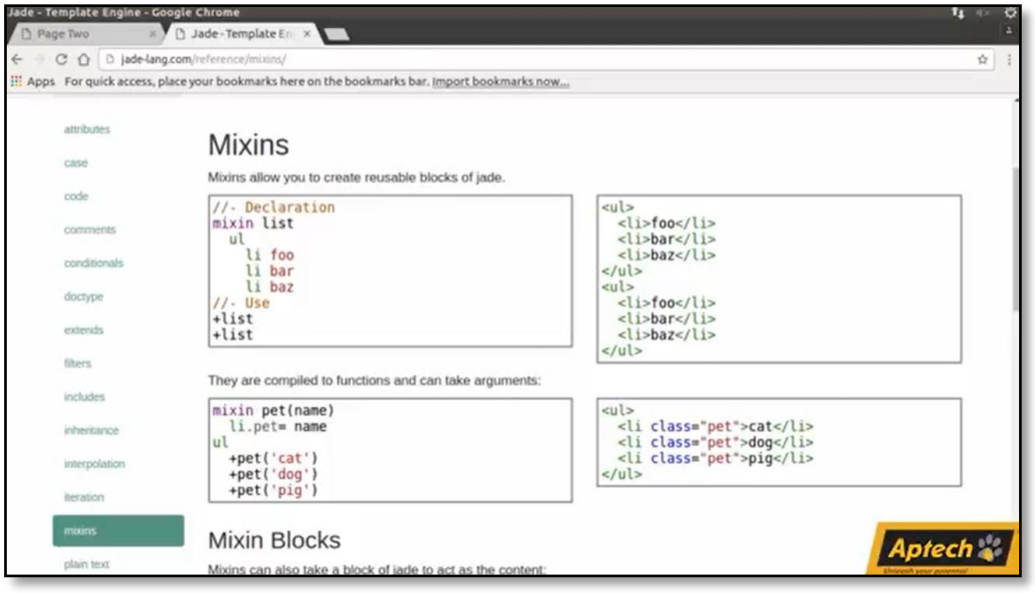
3. Now, an unordered list must be defined. Then, tell students to indent the next line, and start with a hyphen that tells Jade that the following is a block in JavaScript code. Lastly, they must write *for loop* five items and add another hyphen in the final lines.

***Note**: JavaScript is mixed into the Jade language.*

4. Inform students that testing the process again will display the five items.



Explain to the students that Jade has some effective features that can be seen in the reference section of its Website. These are *Mixins* that enable the creation of a reusable block of Jade.



In-Class Question: Why is Jade a preferred choice over HTML and other engines?

Answer: Some of the advantages of Jade over others are as follows:

a) Jade can be compiled into reusable functions that can run on both client and server-side. We can play around with data sets and reuse the template engine.
b) Jade enables the replication of the contents of one template onto another that resembles object-oriented programming techniques.
c) Jade removes all problems associated with output, such as iterating conditions and encoding issues, and improves template productivity.

Reference Links:

https://ezeelive.com/advantages-jade-template-engine/

https://www.developer.com/open-source/build-express-js-web-applications-faster-with-jade-and-stylus/

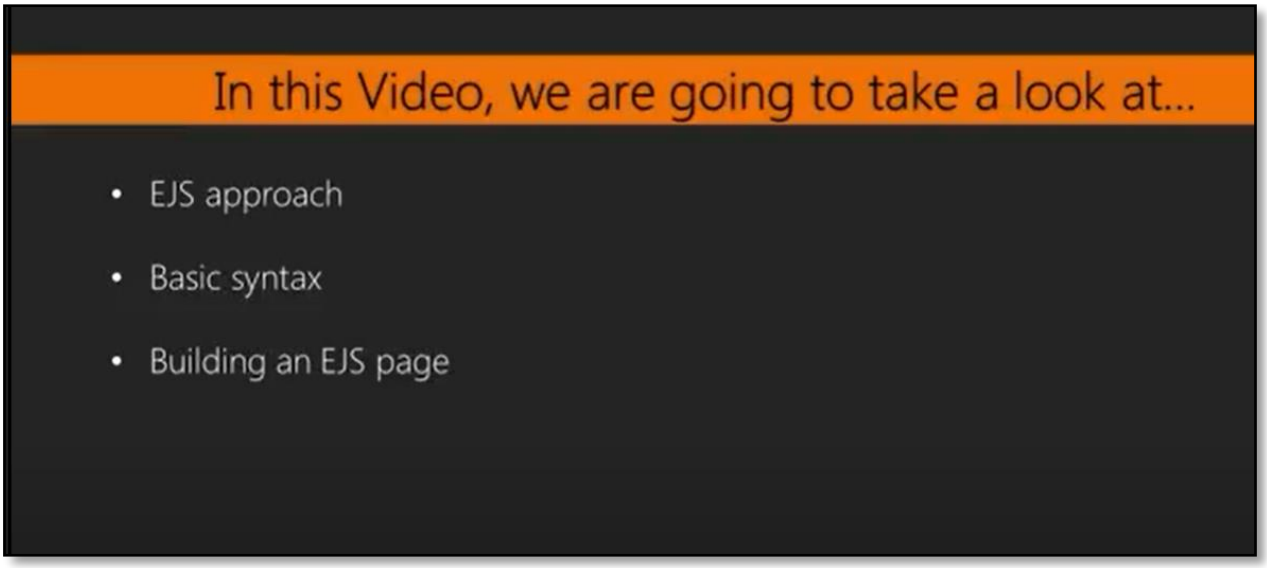
3.4 Using Template Engine – Ejs

Class DURATION: 10 MINUTES

VIDEO DURATION: 4.40 MINUTES

FACILITATION GUIDELINES –

Show: Play Using Template Engine – EJS: Session 3 Video - 3.4.

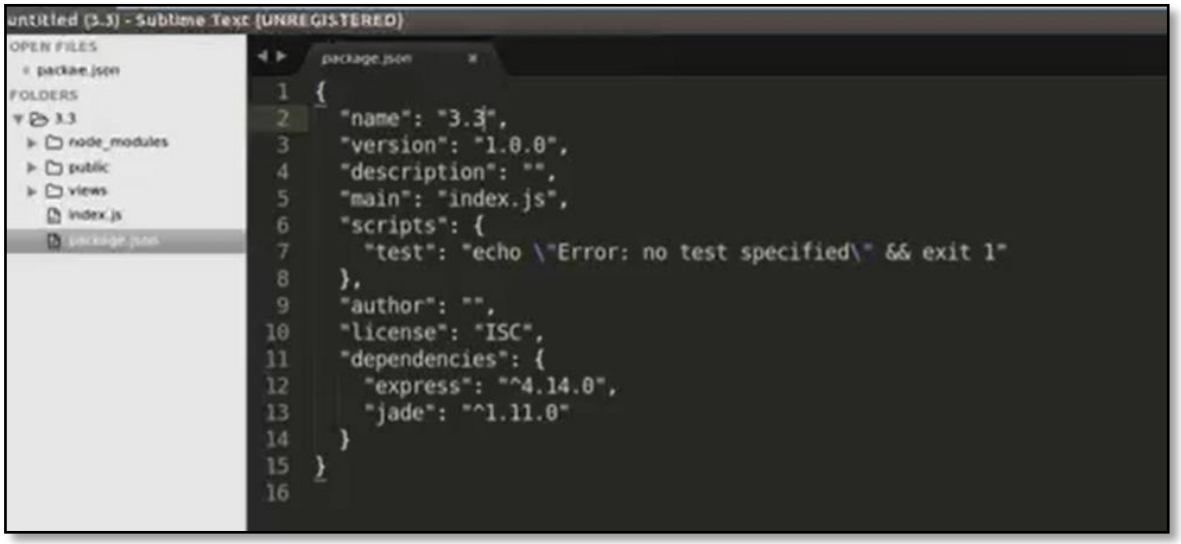


Show: Play the video from 0.43 to 1.59.

Content: Introduce the students to the template engine EJS for *Express* after explaining the Jade template engine. Let them know that EJS embeds logic into an HTML page and instructs them to build a sample page with EJS and Express.

Following are the steps to build an HTML page with EJS and Express:

1. As a first step, students have to create a folder and name it *3.3*. Then, they must copy data from the one created for Jade and open the project folder, change *package.json* to *3.3*. Finally, click 'Save'.



2. Next, ask them to uninstall the Jade package and replace it with the EJS package. For this, they must open the EJS Web site - embeddedjs.com. EJS is described as a "client-side templating language that was originally part of JavaScript NVC, which is a complete framework for client-side development." EJS is very similar to JHP and PHP, the Java pseudo pages. EJS combines data in a template that generates the HTML.

Creating code:

Show: Play the video from 2.00 to 4.18.

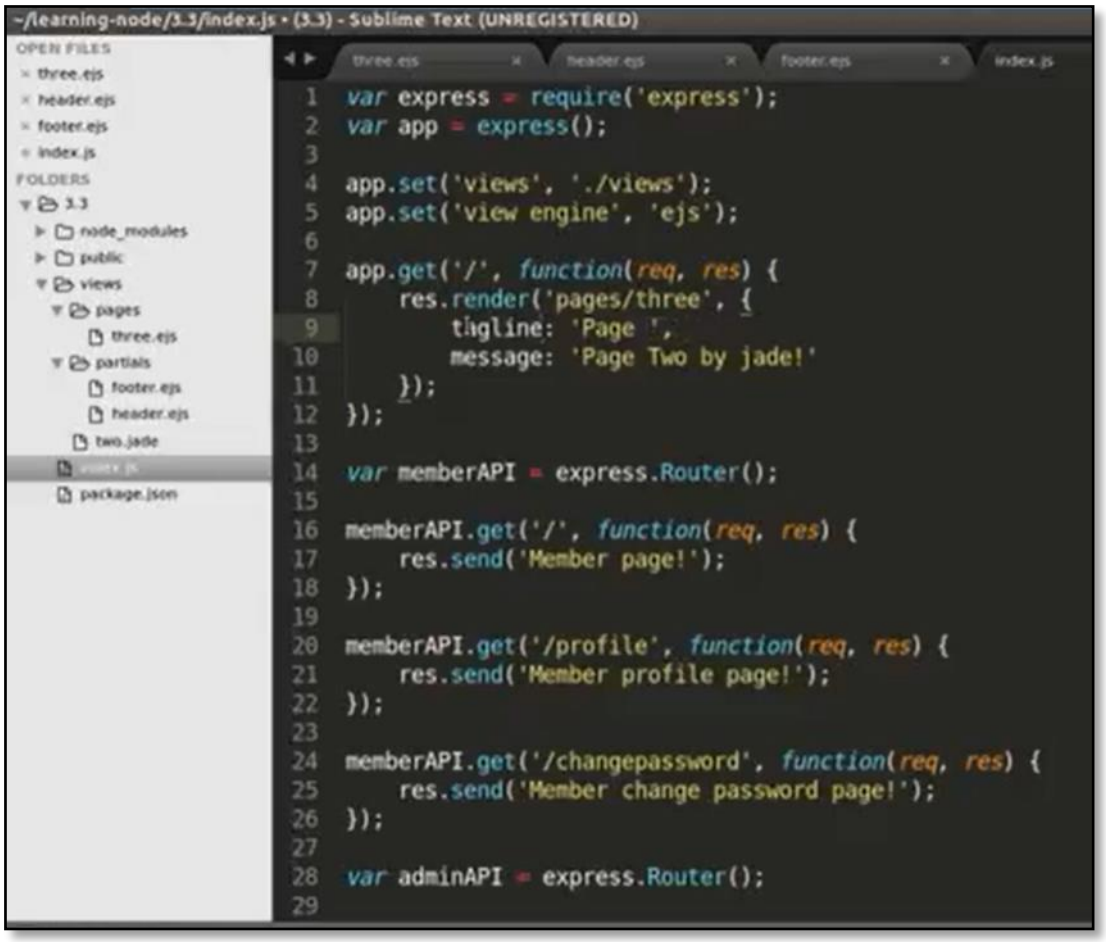
Content: Explain to the students and instruct them to create two folders named 'pages' and 'partials' under 'Views'.

- Pages: Stores whole EJS pages
- Partials: Stores only partial pages

***Note**: Partial folders are building blocks of whole pages and can be used again and again.*

Discuss the following steps with the students to create the blank *ejs* files:

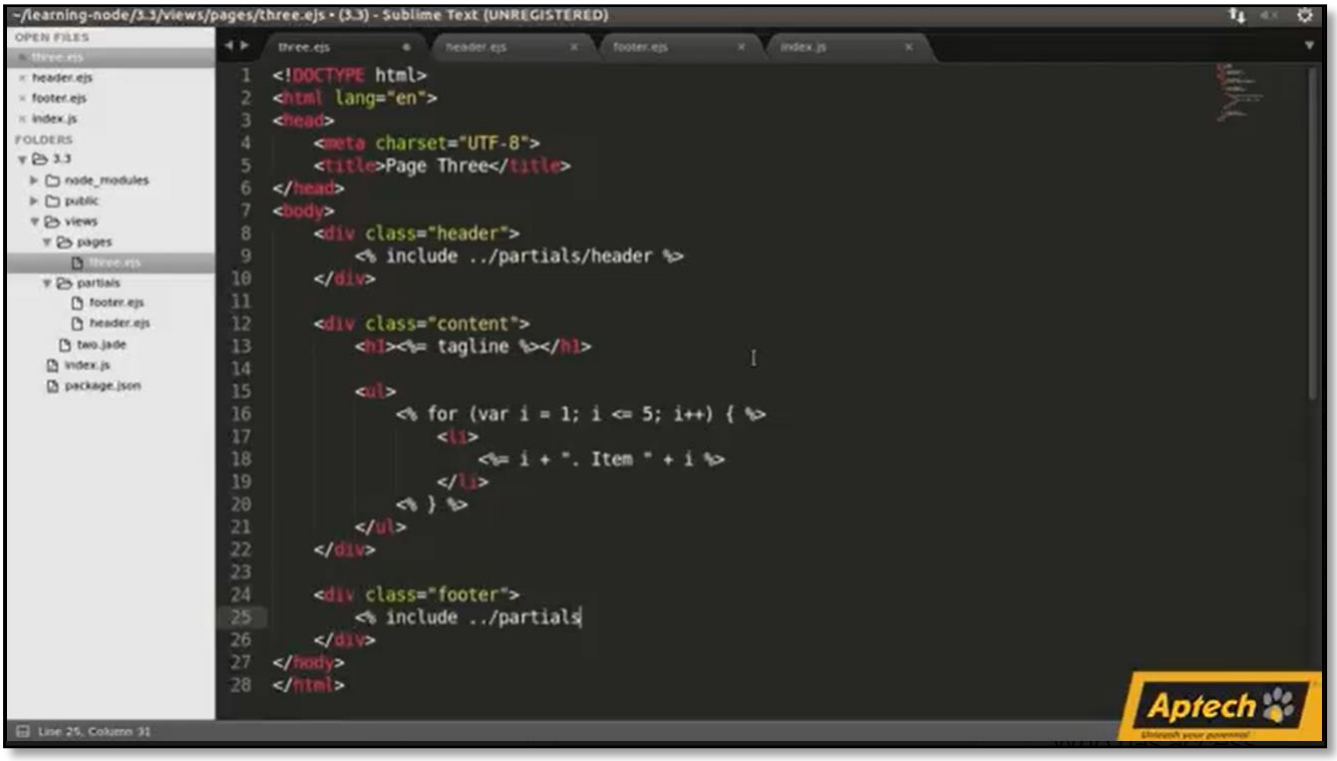
1. As a first step, students have to create three *.ejs* in Pages and *header.ejs* and *footer.ejs* in partials. The codes can be filled in these files later. Then, the `viewengine` property in *index.js* has to be changed so that *Express* can interpret EJS. Furthermore, tell the class to change the render methods of the route to 'pages/three.' Let them pass the 'tagline' variable to the EJS page. Lastly, ask the students to set the tagline.



2. Secondly, tell students to navigate back to *three.ejs* and create a page similar to *html*. In the `body` tag, they must insert a placeholder `div` of the class `header` for *header.ejs*.

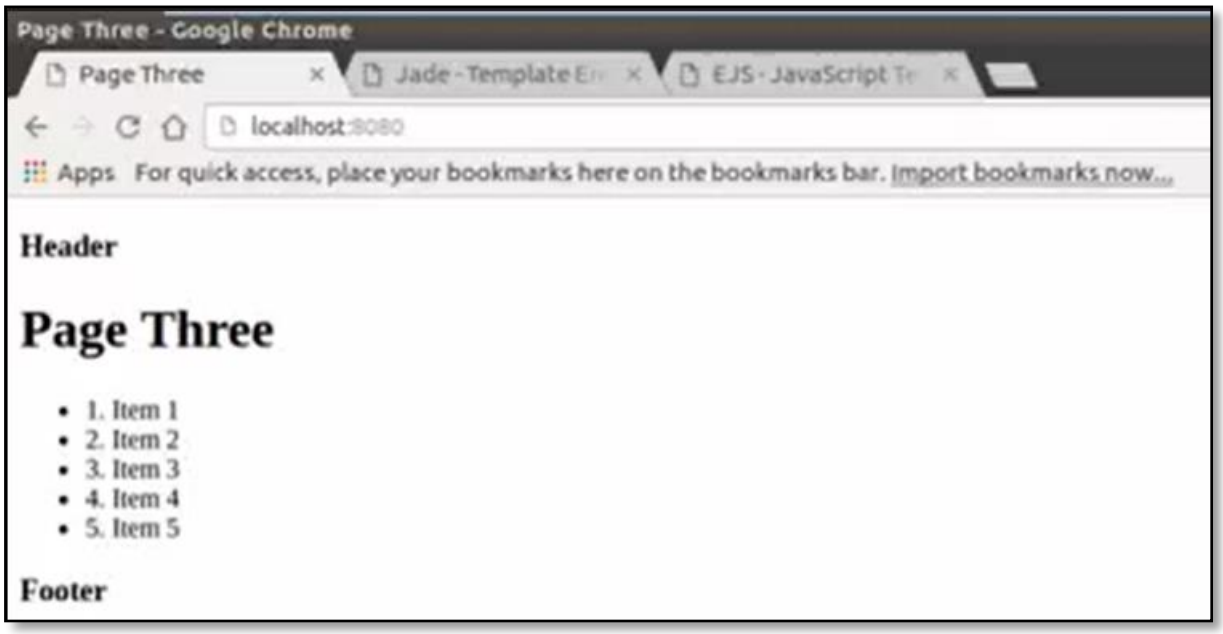
***Note**: EJS enables insertion of the placeholder by the include directive.*

3. Then, ask the class to mention the path of *folder.ejs* that is saved in the *partials* folder. Let them insert another `div` under the header as the location of primary content. A tagline and the `for` loop will list out five items as an unordered list. Tell them to follow the same procedure for *header.ejs* and *footer.ejs*.



4. Students must open *footer.ejs* and create a `div` with 'h3' tag that says 'footer'. Similarly, for the header, tell them to create a `div` with 'h3' tag that says 'header'. Then, let them run the application.

5. Now, ask them to run the *Node1* index. Discuss that the outcome shows 'localhost:8080' with a header, footer, and main content with page three as the tagline passed from *index.js* and the list of five items generated from the `for` loop.



***Note**: If you are more familiar with JSP, you will usually prefer EJS over JADE.*

In-Class Question: Why is EJS preferred over Jade?

Answers: Jade as a template engine has its own perks. However, here are a few things that make EJS the preferred template engine for many developers:

- When HTML templates are received, you do not have to convert them to EJS. Since EJS is very close to HTML, you only have to replace dynamic parts with variables passed from Express.
- EJS is more designer-friendly. It can combine with Zen-coding and speed up processes in general.

Additional References:

https://stackoverflow.com/questions/16513168/what-are-the-pros-and-cons-of-both-jade-and-ejs-for-node-js-templating

3.5 Adding Responsiveness With Bootstrap

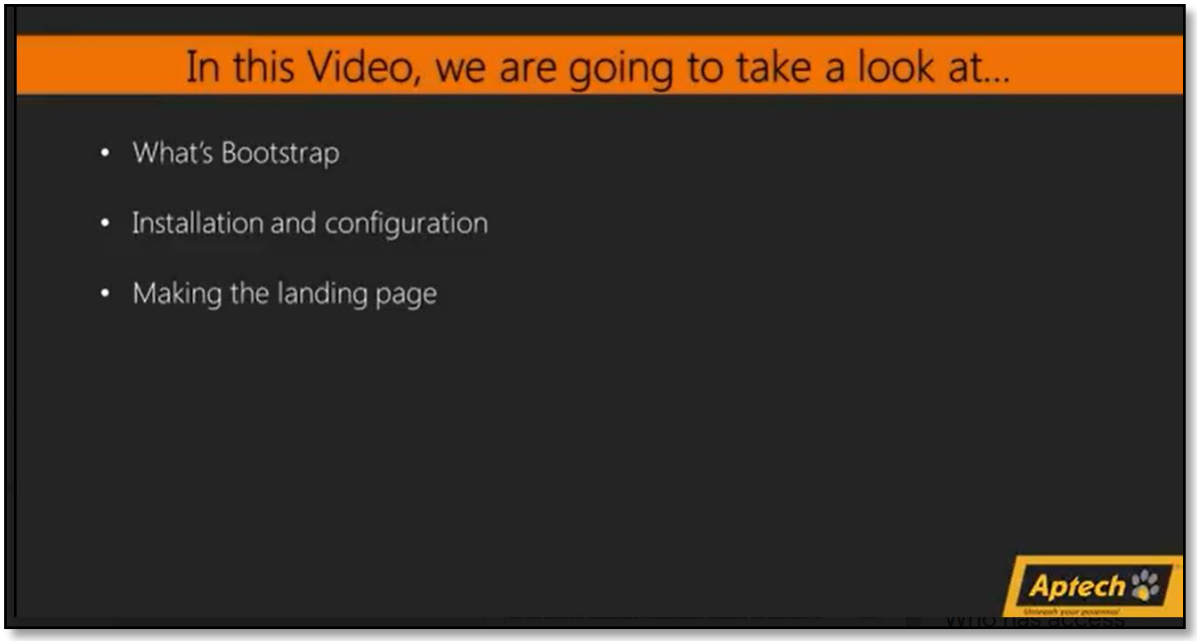
CLASS DURATION: 20 MINUTES

VIDEO DURATION: 5.41 MINUTES

FACILITATION GUIDELINES -

Show: Play Adding Responsiveness with Bootstrap: Session 3 Video - 3.5.

Content: Explain to the students that the Bootstrap Framework enables them to make responsible Web apps for different screen sizes.

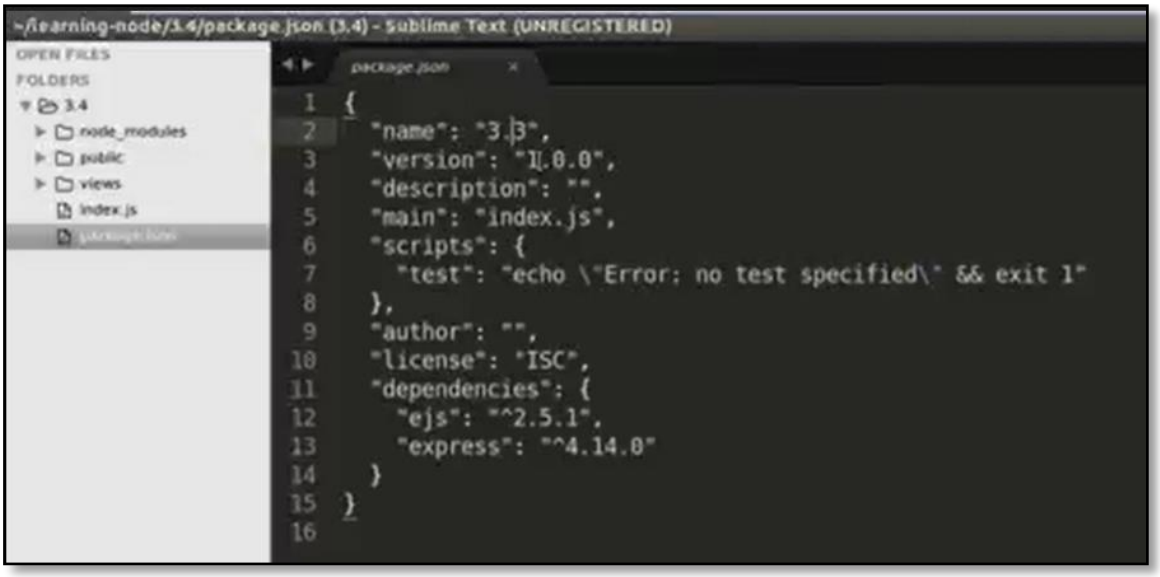


Show: Play the video from 0.48 to 1.04.

Content: Discuss the steps required to create a responsive bootstrap application.

Following are the steps to create a responsive bootstrap and reuse it for making a landing page:

1. Firstly, tell students to create a new folder and name it *3.4*.
2. Then, instruct them to copy data from the one created for EJS and open the project folder *3.4* in the sublime text.
3. Finally, ask them to change the name to *3.4* in *package.json*.



Following are the steps to install Bootstrap:

1. As a first step, tell students to navigate to *getbootstrap.com*. Explain that the Website describes bootstrap as "the most popular HTML, CSS, and JS framework for developing responsive, mobile-first projects on the Web."

   ***Note**: Bootstrap technology enables designing apps based on mobile devices. Mobile devices have simpler designs than desktop computers.*

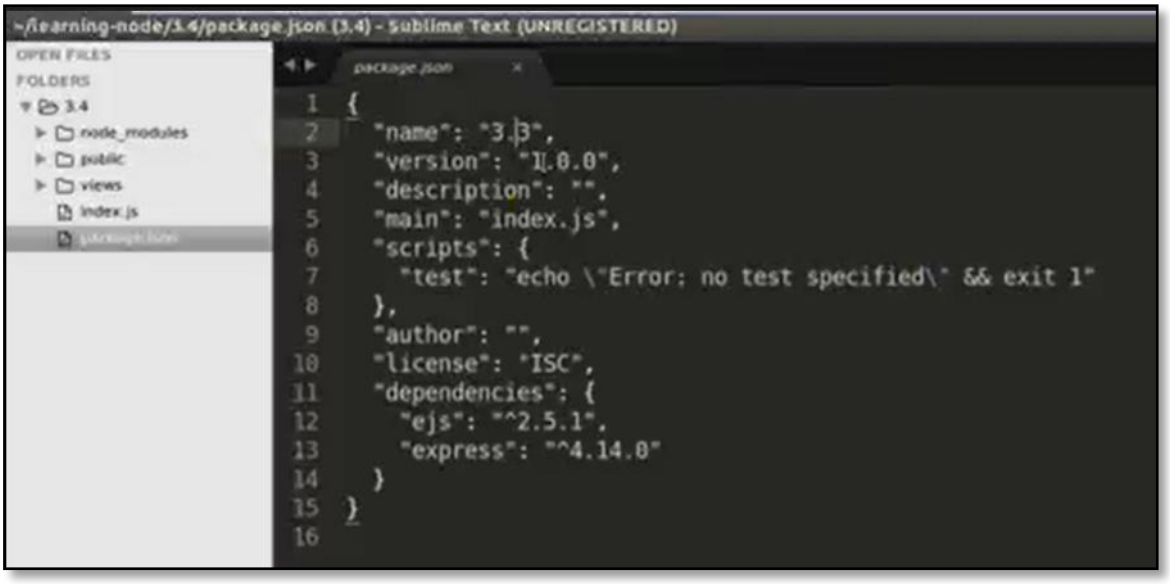
2. Further, discuss with students that Bootstrap can be downloaded in multiple ways. However, for this demo, they must click 'Download Bootstrap'. Inform them that once the download completes, they must extract the file in the project's public folder. Bootstrap artifacts are downloaded and ready for use under CSS → fonts → js folders.

   Note: Bootstrap also provides some basic and standard, ready-to-use templates that can be used in any Web application.

Show: Play the video from 2.44 to 5.00.

Choose Jumbotron

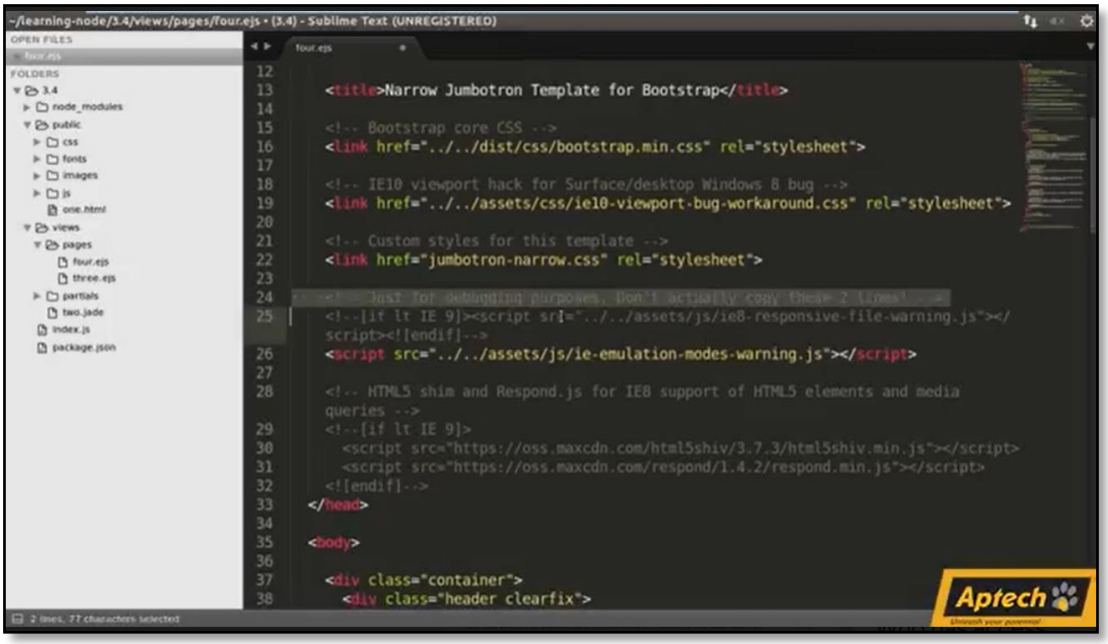
Explain to the students about the Chrome developer. Instruct them to launch Chrome developer, turn ON the Toggle toolbar, and select the device screen size. The size must be as per preference and depending on view as to how the template responds to different screen sizes.



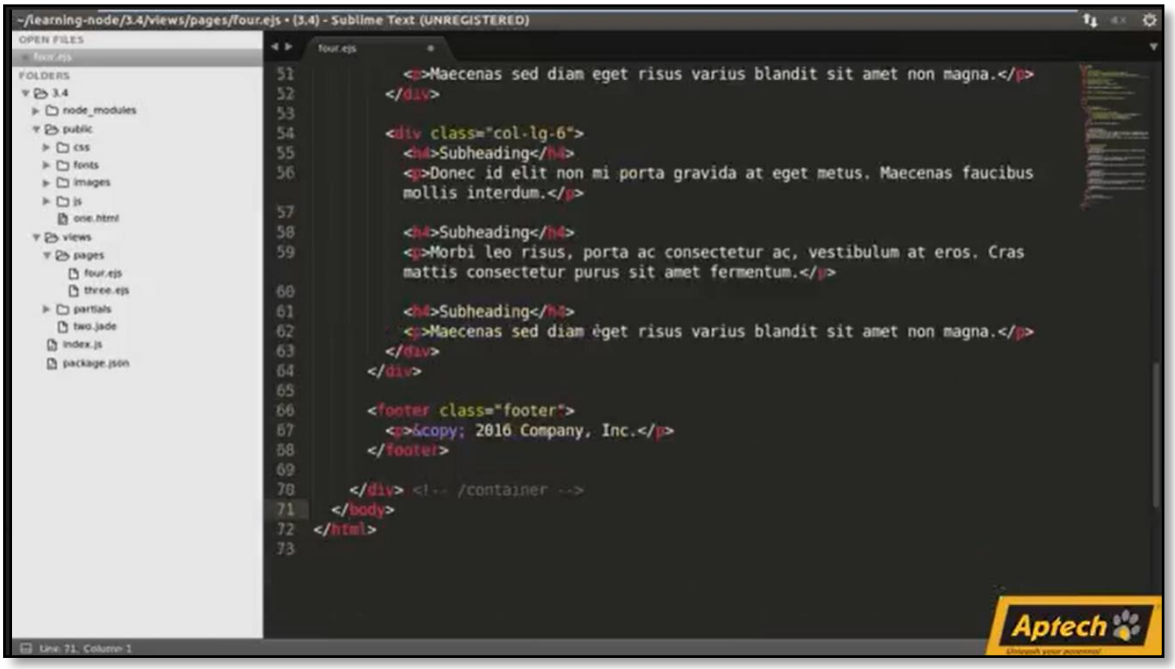
Explain to the student that they can use this template for their use.

Instruct the students to follow these steps to copy the source code:

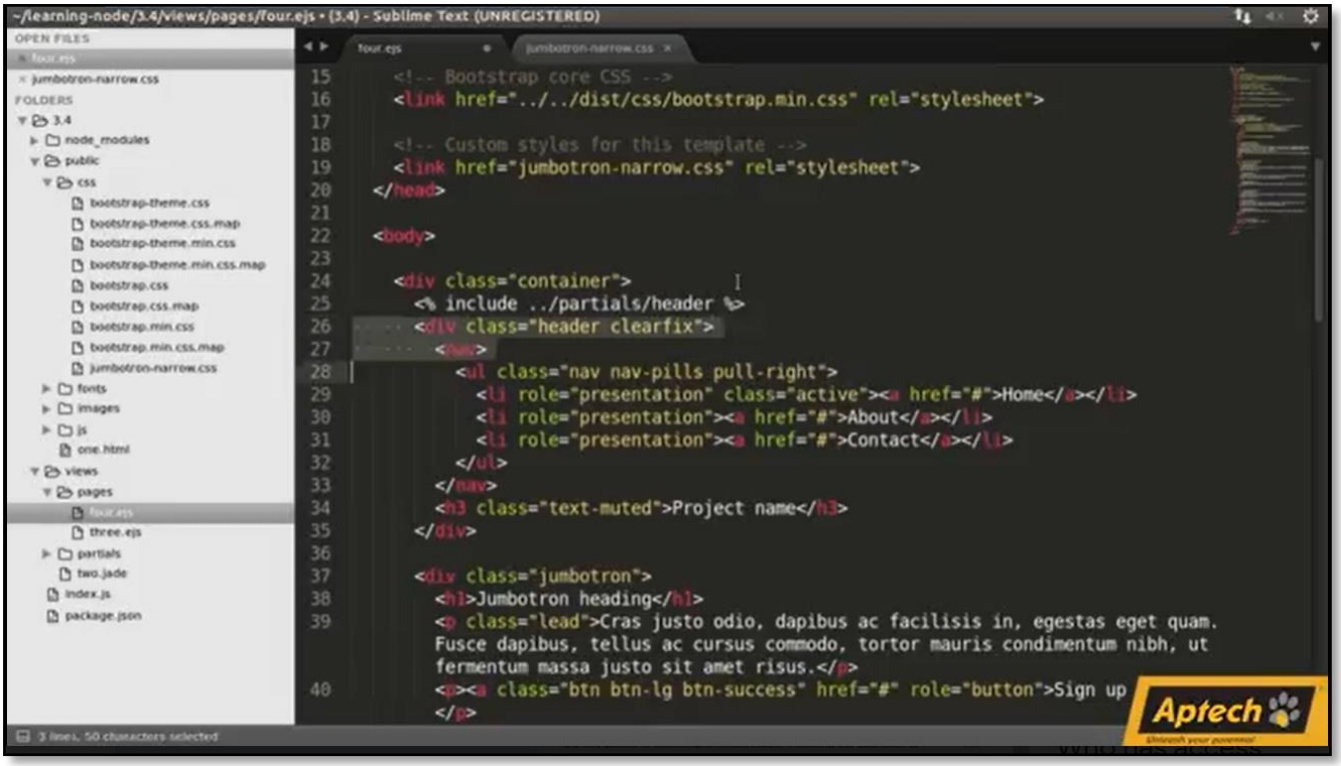
1. Firstly, tell them to select *ViewSource*.
2. Then, select 'All' and 'copy'.
3. In the Pages folder, ask them to create a new file *four.ejs* as the landing page.
4. Further, they must paste the narrow **Jumbotron** source code into it.



5. Tell them to delete all the scripts that cater to Internet Explorer.



6. *Jumbotronarrow.css* file must be copied.
7. Then, students must browse to and click *jumbotron-narrow.css*.
8. Select 'All' and 'copy'.
9. Subsequently, students must create a new file in CSS.
10. Tell them to paste the new CSS code and save it as *jumbotron-narrow.css*. For better usage, instruct them to divide *four.ejs* into some parts as done in the EJS approach. Underneath the container, the programmer must write `include for header.ejs`.



11. Instruct the students to move the code block to *header.ejs*. At the top of the `div` container, add another `include for footer.ejs`. Shift this code block to *footer.ejs*.
12. Moving further, they must create a new ejs file called *index.ejs* in partials. Then, shift the primary content to *index.ejs*.

```
        <h4>Subheading</h4>
        <p>Morbi leo risus, porta ac consectetur ac, vestibulum at eros. Cras
        mattis consectetur purus sit amet fermentum.</p>

        <h4>Subheading</h4>
        <p>Maecenas sed diam eget risus varius blandit sit amet non magna.</p>
      </div>

      <div class="col-lg-6">
        <h4>Subheading</h4>
        <p>Donec id elit non mi porta gravida at eget metus. Maecenas faucibus
        mollis interdum.</p>

        <h4>Subheading</h4>
        <p>Morbi leo risus, porta ac consectetur ac, vestibulum at eros. Cras
        mattis consectetur purus sit amet fermentum.</p>

        <h4>Subheading</h4>
        <p>Maecenas sed diam eget risus varius blandit sit amet non magna.</p>
      </div>
```

13. Another `include` must be added for *index.ejs*.

***Note**: Always add an 'include' for the removed code.*

14. Then, they have to change the route to 'four' in *index.js*.

```
app.get('/', function(req, res) {
    res.render('pages/four');
});
```

15. Ask students to remove the tagline variable. In localhost: 8080 – they must launch the developer tools.
16. Finally, the class has to switch on the Toggle device toolbar and experiment with different screen sizes. Describe that they can use the developer tools for app testing.

In-Class Question: What are the advantages of using the Bootstrap framework?

Answers:

- Bootstrap saves time on getting the work done and improves the speed of development.
- Bootstrap is easier to use because it only requires foundational knowledge of HTML and CSS
- Bootstrap enables the creation of a response Web application, which adapts to all platforms such as desktop, laptop, and mobile.

3.6 Handling Parameters

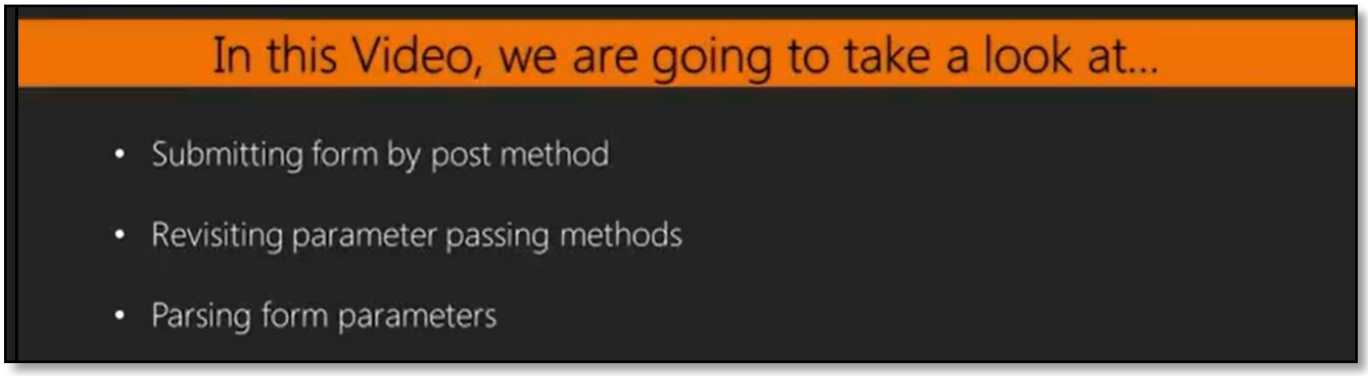
CLASS DURATION: 20 MINUTES

VIDEO DURATION: 6.02 MINUTES

FACILITATION GUIDELINES -

Show: Play Handling Parameters: Session 3 Video - 3.6.

Content: Inform the students about building a data entry form and parsing the form parameters. Further, tell them how to create a data entry form for user input and parse the node parameter with the *Express Middleware*.



Show: Play the video from 0.38 to 1.31.

Content: Discuss the following steps to create a Web page with forms:

1. As a first step, tell students to create a new folder and name it *3.5*.
2. They have to copy data from the one created for Bootstrap and open the project folder *3.5* in *package.json*. Explain that with bootstrap, they can mark up the most often used Web pages such as a data entry form.

Example:

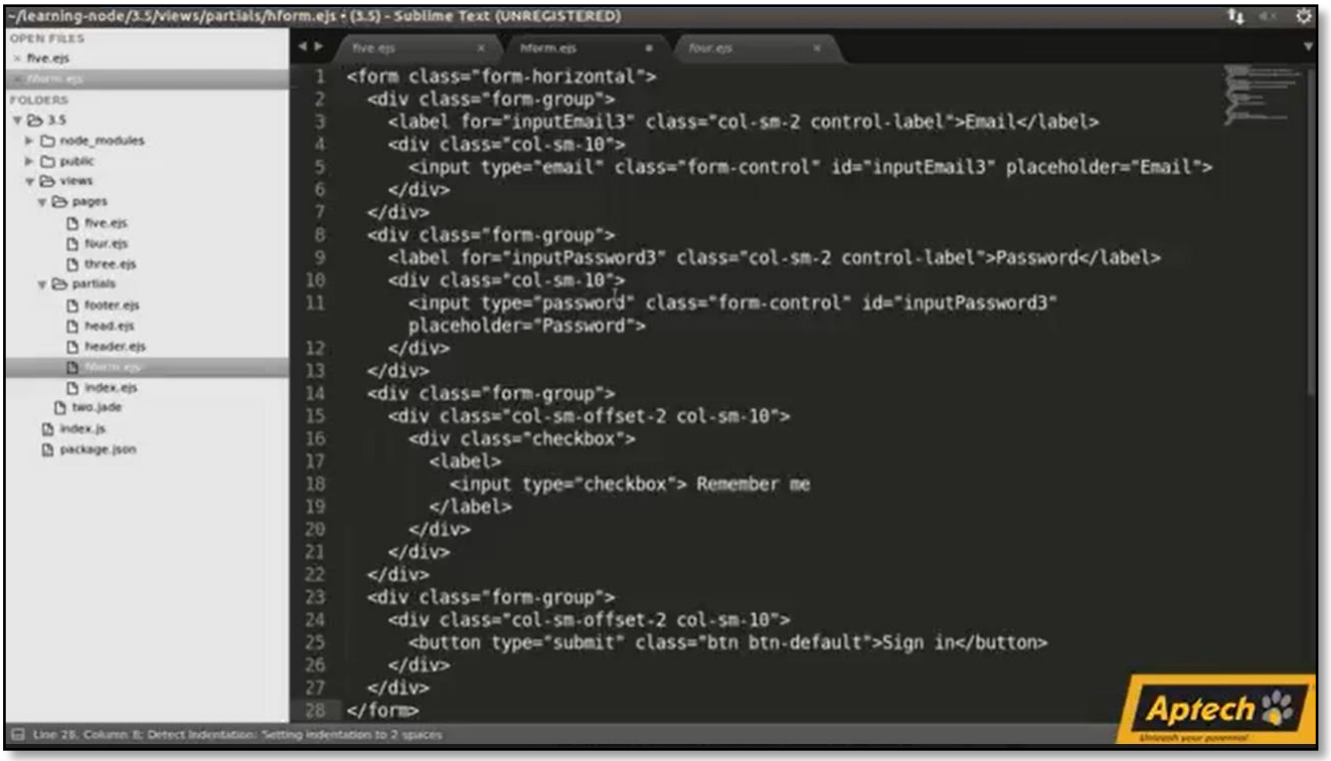
On the Bootstrap Website, click 'Forms'. Under that, there are prepopulated templates for reference.

Show: Play the video from 1.30 to 3.10.

Content: Ask the students to follow the steps as a prerequisite to copying the code.

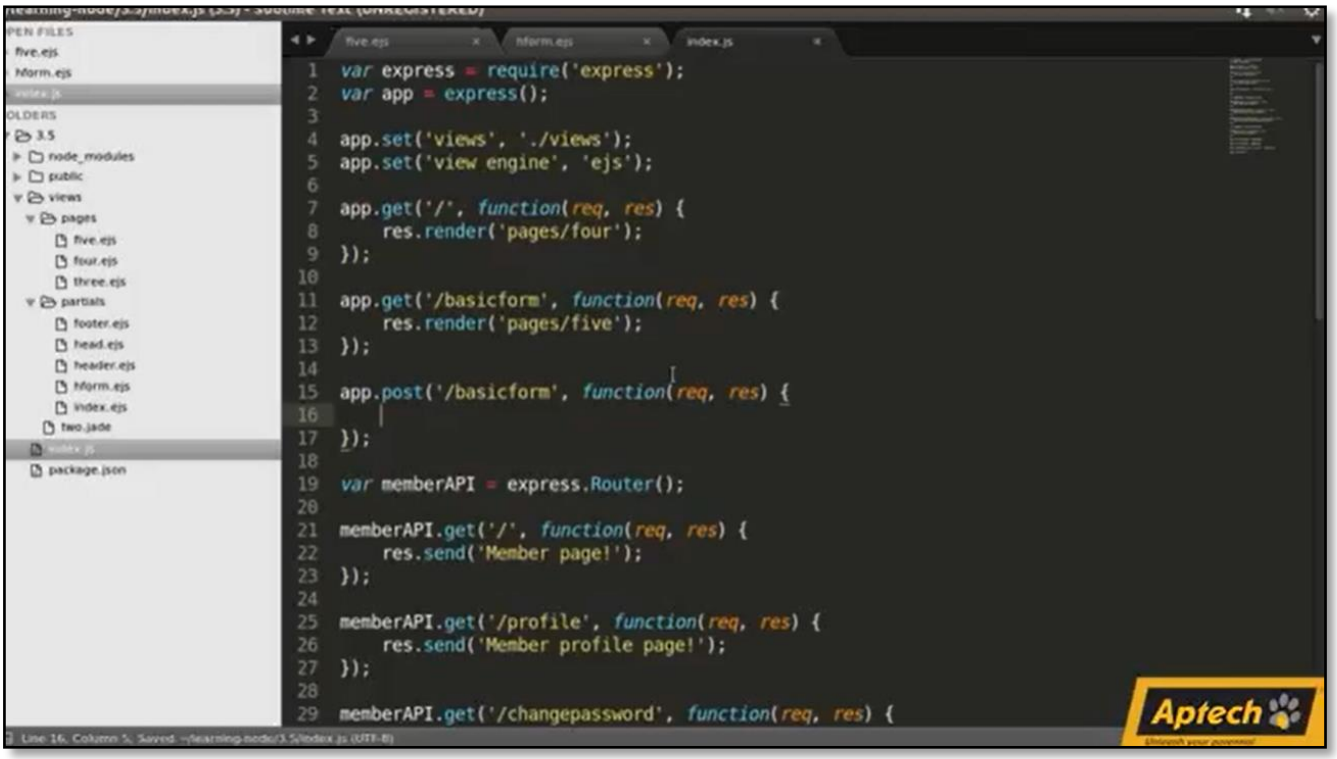
Following are the steps involved in creating *hform*:

1. In *pages*, students must create a new EJS file, *five.ejs*. Then, they must copy the HTML code from *four.ejs* and paste it in the *five.ejs*.
2. Tell them to paste the content part to hform which must be created. In partials, they must create *hform.ejs*.
3. On the bootstrap Website, students must click *copy*. Moreover, they have to paste this code to hform and save.



4. *five.ejs* must be linked to *index.js* by adding a new 'basicform' for *five.ejs*. Explain that the login application form is ready. Now, they must run the app in 'localhost:8080/basicform'.
5. Finally, the submitted parameters must be processed at the backend. Discuss that since the POST method submits parameters to extract them, a POST method handler in **index.handler** must be added. Tell students that the following handlers are required for a data entry form:
   - Get the handler to render the form.
   - POST handler to process the form submission.

Add business logic in the callback function of the POST method handler.



Explain to the students that parameters can be parsed by two `main` methods. One is in query strain and the other is in `urlpath`. There is a third method as well. This includes the parameters parsed in a POST request. Discuss with students the ways to post the parameters.

BodyParser:

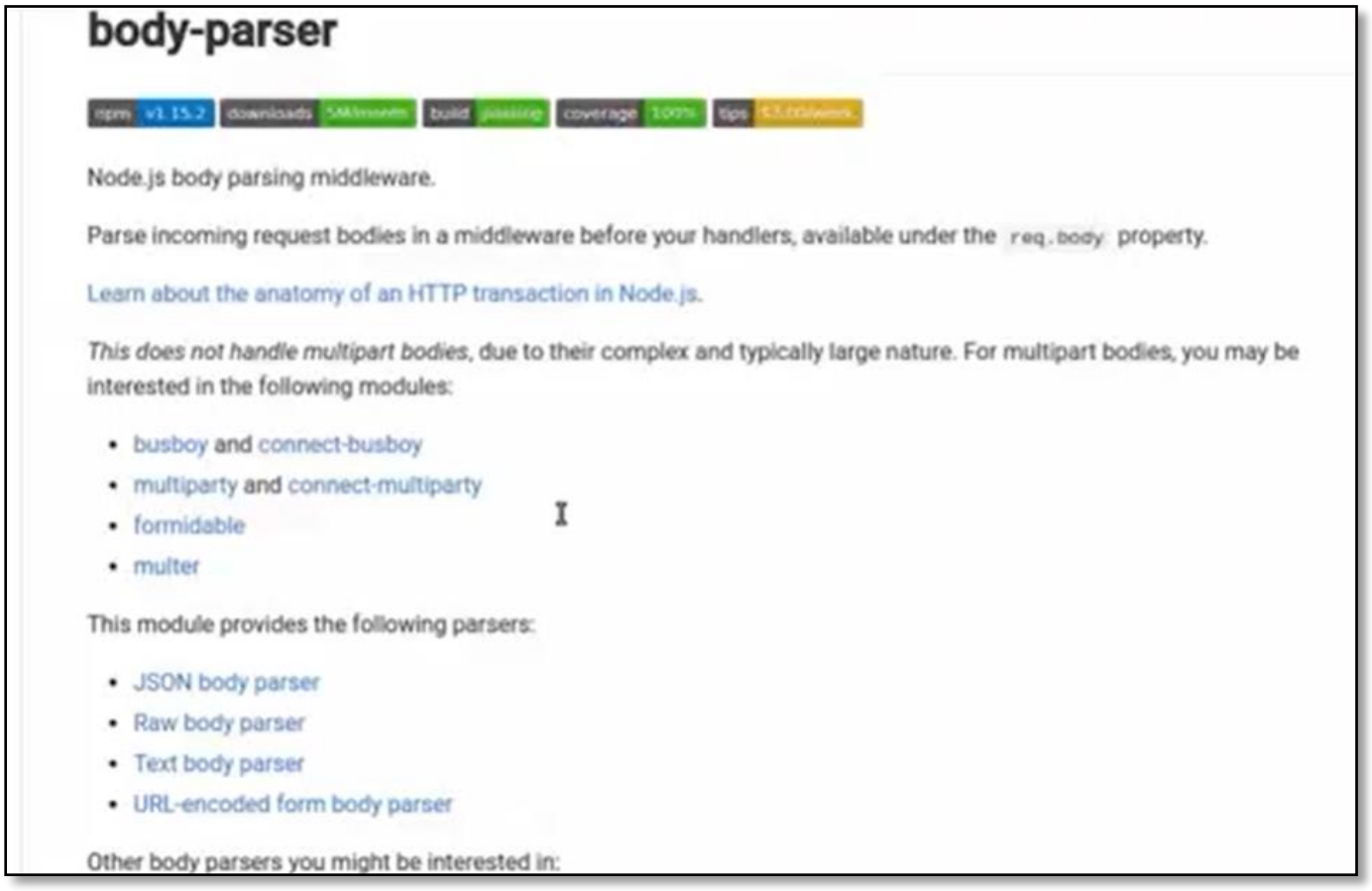
Explain that the Middleware can parse various parameters in a POST request.

Middleware can complete parsing in one of the three formats:

- raw format
- json
- plain text

***Note**: BodyParser cannot parse multipart bodies as they are bulky and large. One example is a file upload.*

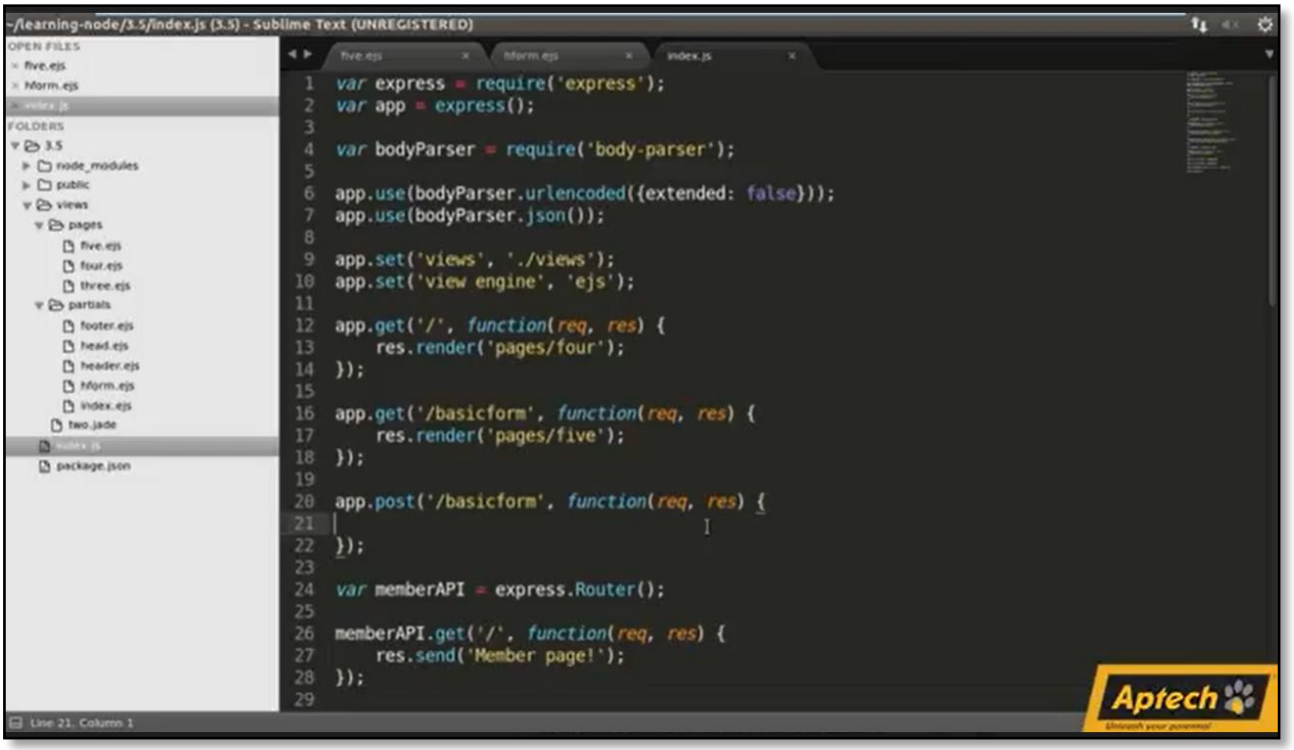
Show: Play the video from 3.30 to 3.56



Steps to install BodyParser by npm:

Show: Play the video from 4.00 to 5.33

Illustrate to the students by starting `nodemon`, and then, install BodyParse. Restart nodemon again.



To understand the usage of Body Parser in the *Express* app, a developer must import and instantiate a Body Parser by adding a statement as required. The next step is to configure the Body Parser by typing in the URL in the coded method. This will return parsed parameters in key-value pairs.

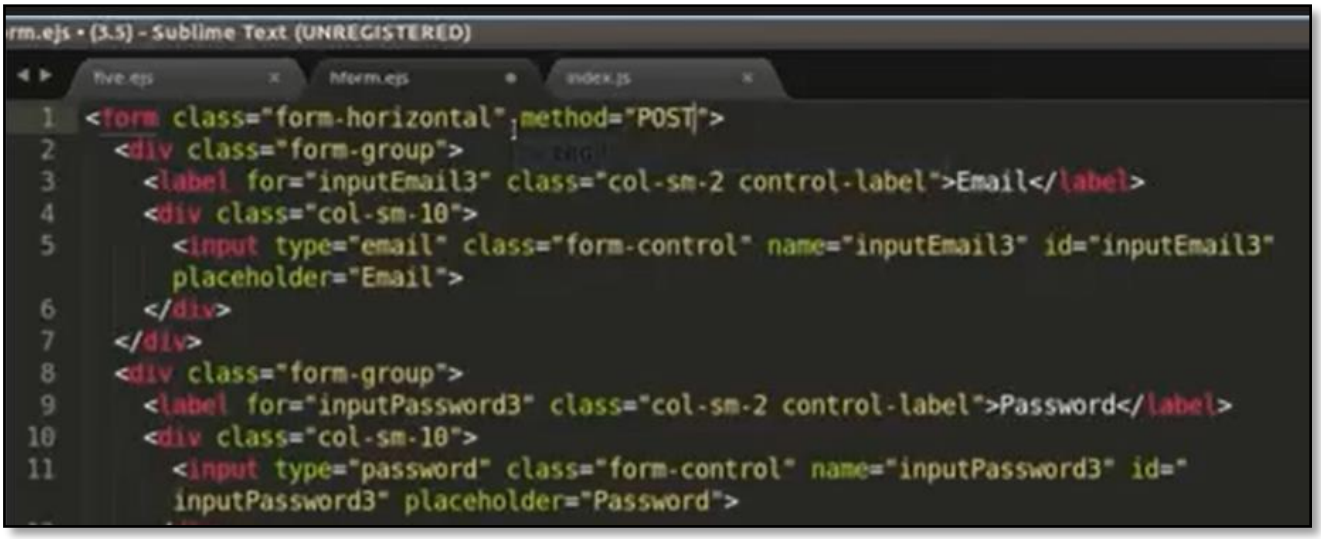
Additionally, let them know that they can call the `JSON` method to handle parameters parsed in a *JSON* file. Furthermore, the `body` attribute of the `Request` Object can retrieve parameters submitted by a POST request. Let them test the output by typing in `login successful` in a console and getting back to the browser.

```
app.post('/basicform', function(req, res) {
    console.log(req.body.inputEmail3 + ", " + req.body.inputPassword3);
    res.send('Login successful')
});
```

Discuss with the students about adding attributes in *hform.ejs*.

Following attributes are to be added in *hform.ejs* for input controls:

- Add a name to the email input control
- Add a password to the password input control
- Add POST as the form method



Now, make the students test the login page by browsing to 'localhost:8080/basicform' and logging in with the following credentials:

- Email: a@a.com
- Password: 123456

Finally, make them check the console log and notice that the same login details are displayed there.

You can use *Bootstrap* and *Express* to build data entry forms.

Conclusion: Give a summary of all the topics that are covered in the session.

QUERIES REGARDING SESSION

CLASS DURATION: 10 MINUTES

FACILITATION GUIDELINES –

Ask students if they have any queries regarding the session and resolve the queries.

ASYNCHRONOUS PROGRAMMING AND API SERVER

4.1 Introducing Event Loop

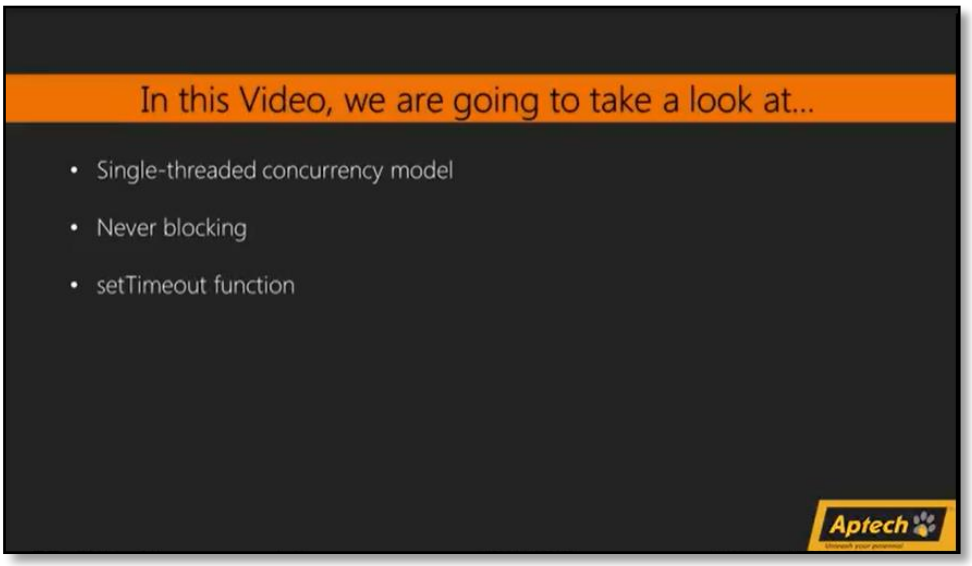
CLASS DURATION: 15 MINUTES

VIDEO DURATION: 4.45 MINUTES

FACILITATION GUIDELINES -

Greeting: Welcome the students back to the course. Recap the previous session and summarize all the main points covered in it. Present a question or two to the students to test their understanding of the previous session.

Show: Play Introducing Event Loop: Session 4 Video – 4.1.



Discussion: Let us look at how one of the technologies most important components functions. This video provides an outline of the Node.js Event Loop concept.

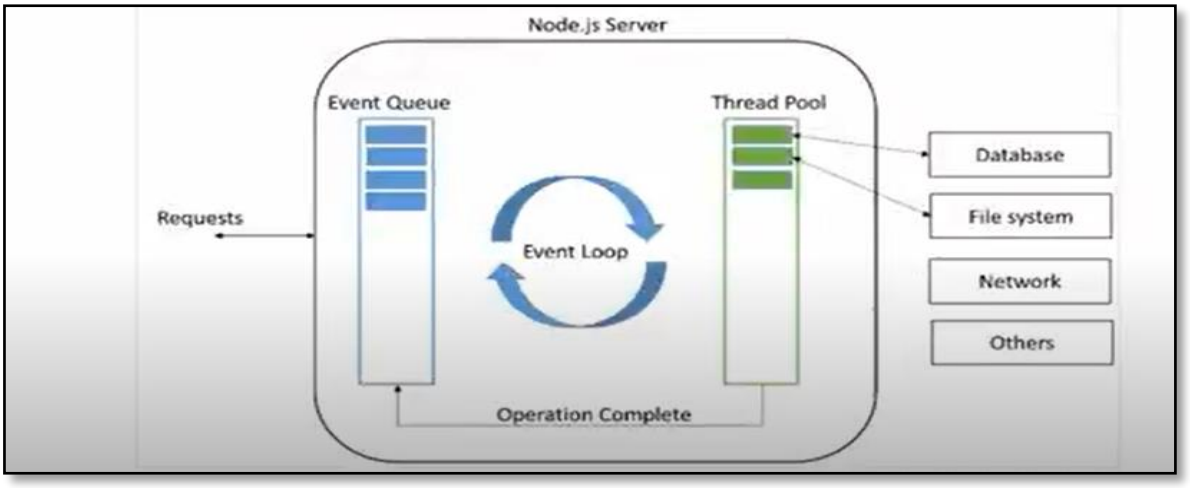
Show: Play the video from 1:00 to 3:00.

Content: Explain how Event Loop Software allows students to subscribe to system resource events. It includes a callback stack that may be attached to resources and run when appropriate descriptors are available.

JavaScript developers might be familiar with the event Loop abstraction because it is commonly used in browser programming applications. Frontend programming that involves attaching callbacks to user activities is structured using the Event-Driven paradigm.

Each call to I/O operations blocks the program main thread in traditional Web Server-side architecture. When an I/O event happens in Node.js, it schedules a task on the `libuv` stack to process it, and the Event Loop continues to execute.

Show: Show following image to students.



Following process occurs when Node.js server gets started:

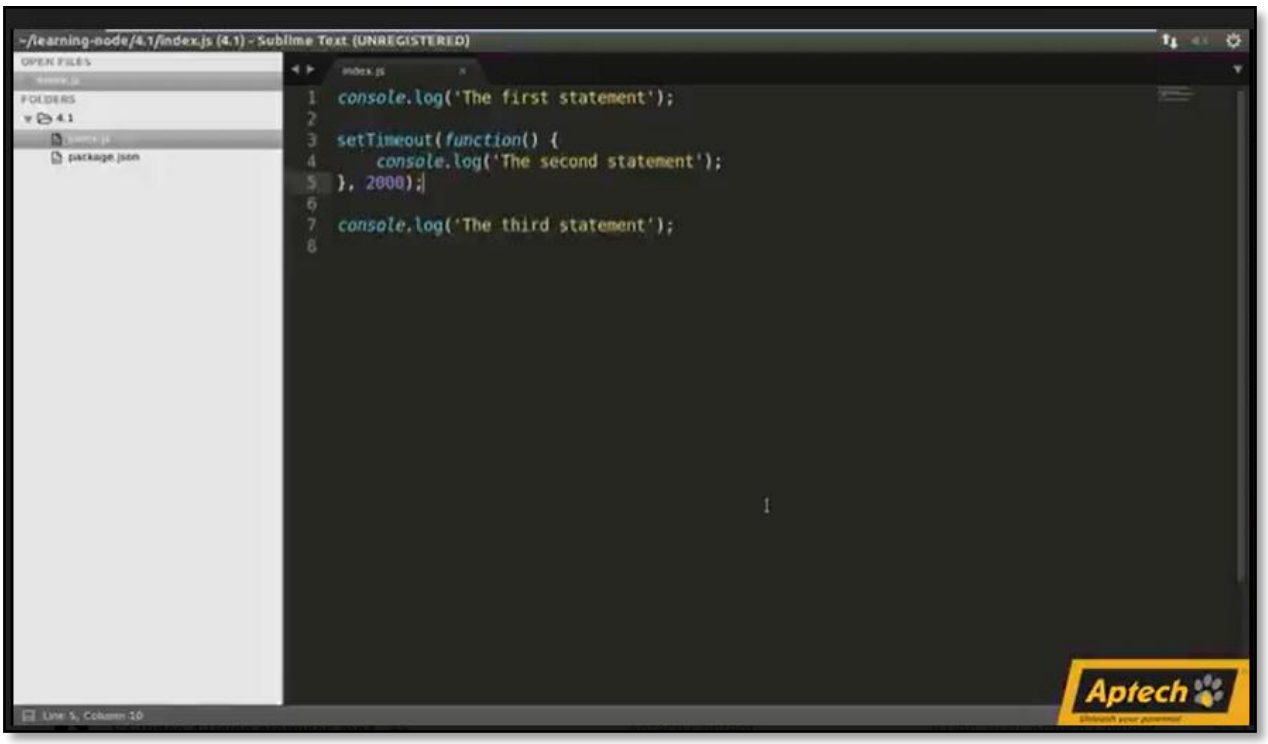
1. The `Event Loop` starts up.
2. The script is then run using dependent libraries and synchronous calls (API, `setTimeout`, `process.nextTick`).
3. Timeouts and callbacks for async actions are defined.
4. Following that, Event Loop waits for events to occur.

An `Event Loop` begins processing events after an initial script completes. When there is nothing to process on a main thread, it is set to sleep. However, main thread wakes up during the following:

- When a system event occurs.
- When an operation completes.
- When a timer goes off.

Everything comes to an end with the process `#exit` event.

Show: Play the video from 3:00 to 4.45.



Content: Following native modules use `Event` Loop:

- fs
- http
- dns and so on.

The callback that Node.js environment reads is linked to the action event.js.

`Libuv` is a cross-platform asynchronous I/O library which includes Event Loops, Timers, Sockets, File System operations, Signal handling, Child processes, Threads, and among other features.

Polling for I/O and scheduling callbacks based on various sources of events are handled by it.

Following are the stages supported by `libuv`:

- Timers: Timers is a phase where callbacks arranged by `setTimeout()` and `setInterval()` are executed.
- Pending callbacks: Used in nearly all built-in functions callbacks, but not for timers, immediate, and close types.
- Poll: It is used for connections.
- Close: Used in events such as `socket.on('close')`.

Timeouts, Immediate, and process.nextTick()

- `setTimeout()` and `setInterval()` functionality is almost similar to a browser.
- `setImmediate()` is executed during the poll phase.
- `process.nextTick()` schedules a callback to execute inside phases.

In-Class Question: What is an event loop example?

Answer: The `setTimeout` method is an example of this. The `setTimeout` operation is send to associated API after the operation is completed in the stack and waits until the specified time has over before returning the operation to be executed.

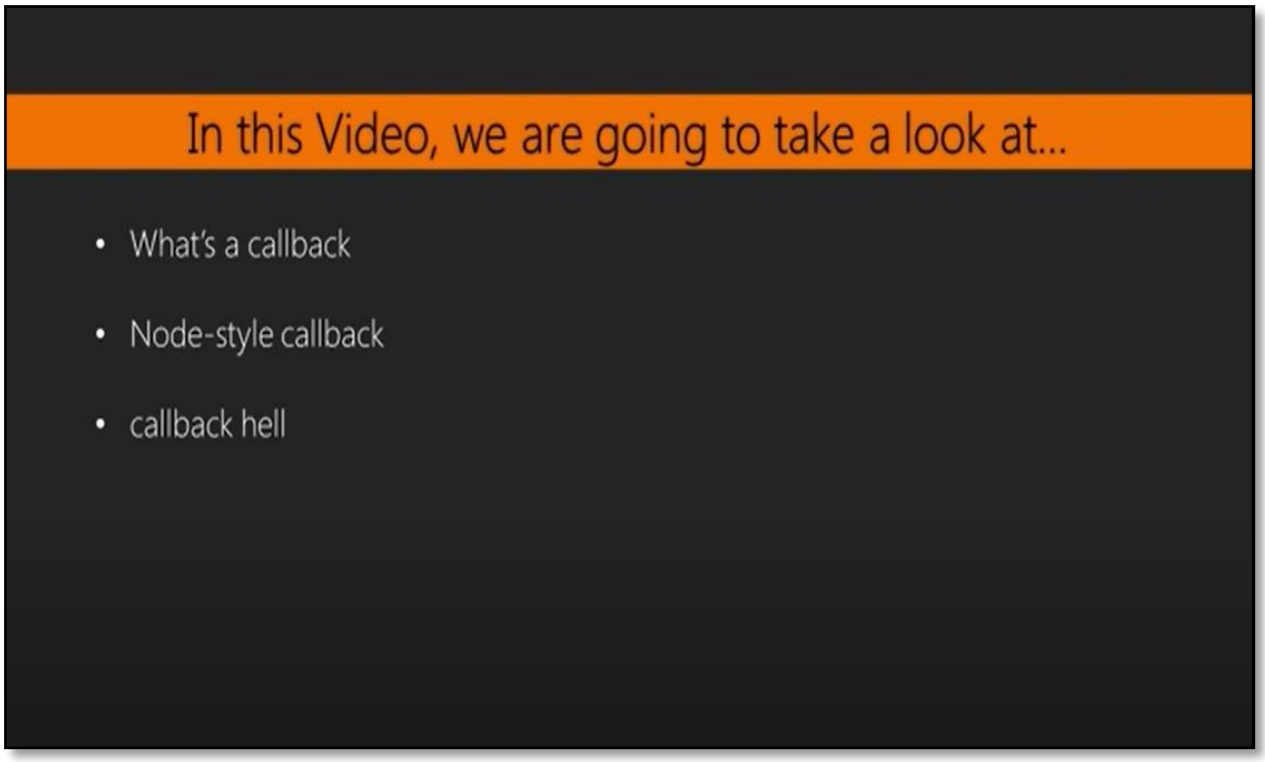
4.2 Understanding Callbacks and Error-First Pattern

CLASS DURATION: 15 MINUTES

VIDEO DURATION: 6.03

FACILITATION GUIDELINES -

Show: Play Understanding Callbacks and Error-First Pattern: Session 4 Video – 4.2.



Discussion: If the heart of Node.js application is the Google's V8 Engine then, veins of Node.js application are the callbacks. They allow the flow of nonsynchronous control between modules and applications to be balanced and non-blocking. However, a user requires a common, dependable protocol to make callbacks function at scale. The 'error-first' callback was established to address this issue and it has subsequently become the industry standard for Node.js callbacks.

Content: Explain students about the extensive usage of callbacks in Node stems back to a programming technique that introduce JavaScript. Continuation-Passing Style, now CPS - former handle to understand how Node.js uses callbacks. In CPS, an argument, 'continuation

function' (`read: 'callback'`) is passed, and it is called after remaining code has finished. This permits different functions in an application to exchange control at the same time.

Node.js is based on asynchronous programming, and hence, a reliable callback mechanism is required. Developers are obliged to keep separate signatures and styles for each module if they did not have one.

The error-first approach was added into Node core to remedy this issue and it has become the industry standard.

DEFINING AN ERROR-FIRST CALLBACK

There are two rules to follow when it comes to describe an error-first callback:

- The foremost argument of the callback is for a fault object. When an error occurs, the foremost *err* parameter will return it.
- The second parameter of the callback is for any response data that was successful. The *err* is set to null if there was no error and the second argument will contain the successful data.

Callback Hell

- Pyramid of Doom, also known as Callback Hell, is an anti-pattern found in asynchronous computer code.
- A slang phrase for a large number of interconnected 'if' statements or functions.
- Some callbacks appear harmless when users do not expect the logic of application to become too sophisticated.

A callback function 'A' act as a parameter in another function 'B'. At some time, the function 'B' runs the code 'A.' The callback function 'A' can be invoked immediately, as in a synchronous or later, as in an asynchronous callback.

Writing and maintaining programming becomes more difficult with the usage of callbacks. It also makes identifying the application flow more difficult, which is a debugging barrier, hence, the problem well-known as Callback Hell.

Additional Reference:

Refer to following links for more information:

http://callbackhell.com/

https://wesbos.com/javascript/12-advanced-flow-control/66-the-event-loop-and-callback-hell

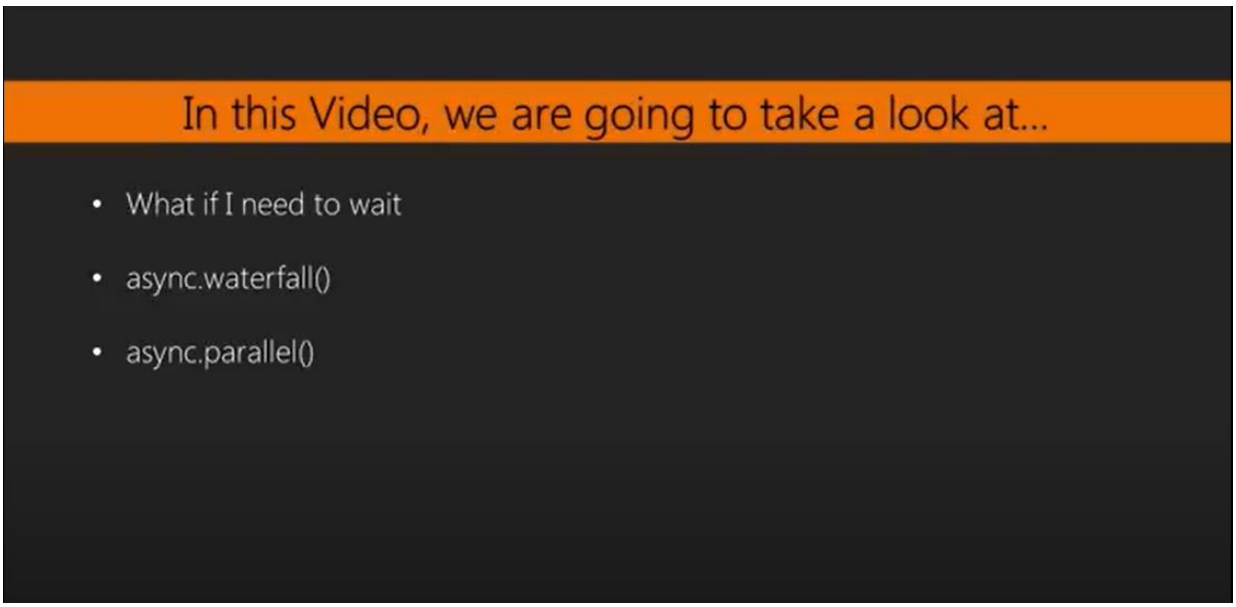
4.3 Using Async.Js Library

CLASS DURATION: 10 MINUTES

VIDEO DURATION: 3.35 MINUTES

FACILITATION GUIDELINES -

Show: Play Using Async.js Library: Session 4 Video – 4.3.



Discussion: `Async` is a utility module for working with asynchronous JavaScript that provides simple yet powerful capabilities. It may be used straight in the browser while being developed for use with Node.js and installable with `npm I async`.

Content: Async Waterfall

Explain students about the `Async` Waterfall approach. It is an easy and effective way to break out of a callback nightmare. Furthermore, it turns the code readable, understandable, and much easier to maintain and debug.

Show: Play the video from 1:30 to 2:00.

Content: Tell students that they can perform following steps before starting:

- Break the code down into basic asynchronous step functions that must be used to complete a task. In case of reading a JSON file, reading the file and processing it are the steps included.

- Each step function accepts a callback as a parameter. The error object is the foremost parameter to the callback. Waterfall stops processing and the error handler is called along with the error object when the error object is not null.

- The error handler takes the parameters as arguments to the next step function along with any other arguments required.

    - Read the file.
    - Process the file.
    - Plug everything with `async.waterfall`.

Show: Play the video from 2.00 to 3.30.

Content: AsyncParrallel

Explain students that numerous asynchronous operations can be done simultaneously using `async.parallel()` function.

- The foremost argument to `async.parallel()` is a group of asynchronous procedures to run (an array, object, or other iterable). After a function is finished, it must call callback (err, result) with an error err (that can be null) and an optional results value.

- The second argument to `async.parallel()` is a callback function that will be called after finishing all of the functions in the first argument. The callback is executed with an error parameter and a result collection containing the outcomes of the respective asynchronous operations. The result collection type is the same as the first parameters.

In-Class Question: What is async and await in JS?

Answer: Promises are easier to write using async and await. The async makes a function return a Promise. The await keyword causes a function to wait for a Promise.

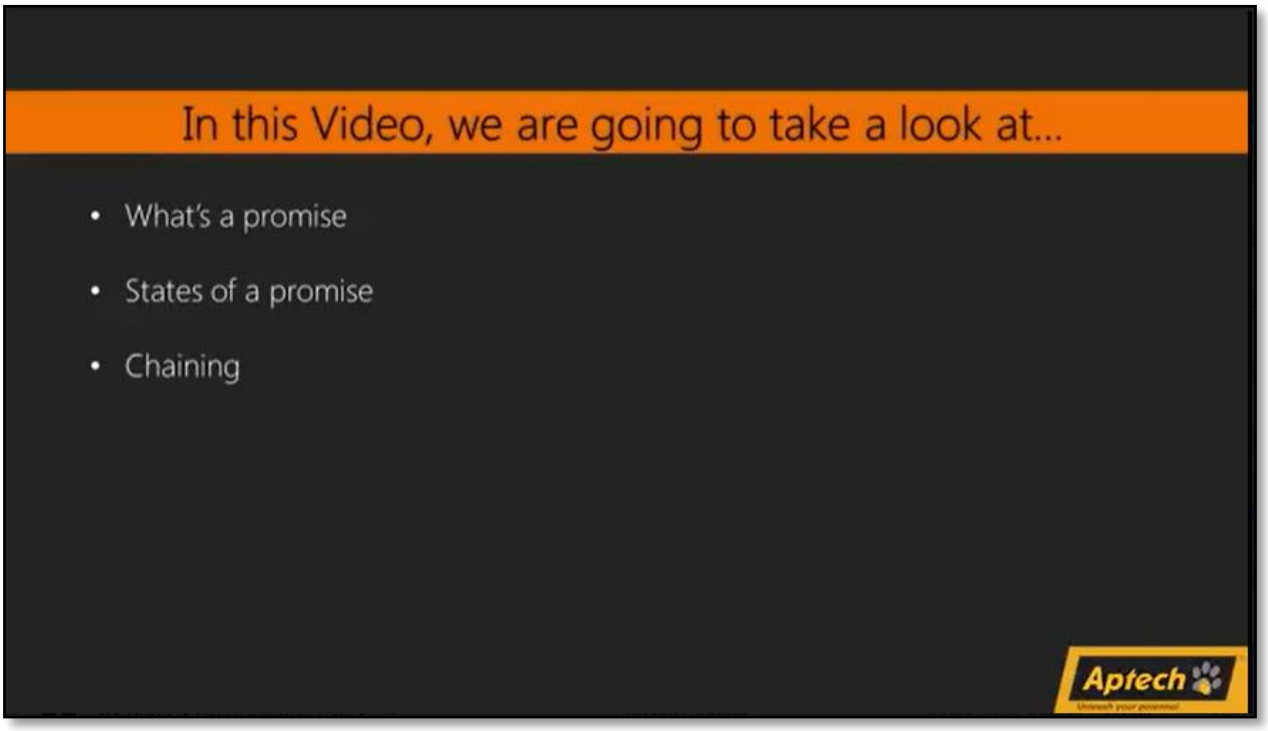
4.4 Using Promises

CLASS DURATION: 10 MINUTES

VIDEO DURATION: 4.18 MINUTES

FACILITATION GUIDELINES -

Show: Play Using Promises: Session 4 Video – 4.4.



Discussion: A callback progression is a promise in Node. The user may notice many nested callback functions when developing an application.

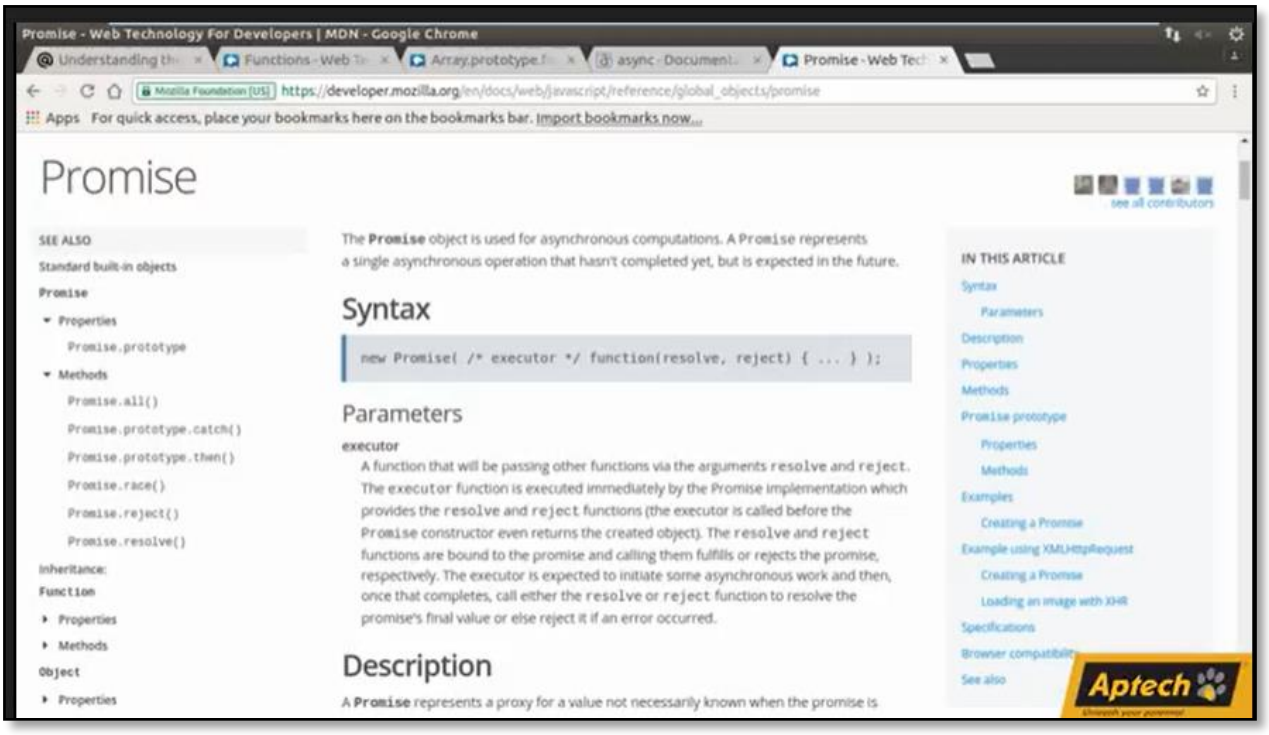
Content: Tell students that Promise in Node is a one-of-a-kind activity that can be accomplished or rejected. The promise is kept if the task is completed else, the promise is broken. As the name implies, the promise is either honored or broken. Promises, unlike callbacks, can be chained.

Explain students following advantages of promise in Node:

- Enhances the readability of code.
- A better way to handle asynchronous operations.

- In asynchronous logic, a better flow of control specification is required.
- A Promise of Better Error Handling.

Show: Play the video from 1:00 to 2:40.



Content: Tell the students about the four states of Promise. Following are the states of Promise:

1. **Fulfilled**: Action related to the promise succeeded.
2. **Rejected**: Action related to the promise failed.
3. **Pending**: Promise is pending and not addressed or declined yet.
4. **Settled**: Promise has addressed or declined.

Promise constructor is used to create a promise.

Syntax:

```
var promise = new Promise(function(resolve, reject){
     //do something
})
```

Content: Explain students about different parameters of Promise.

- The promise function `Object(){ [native code] }` only accepts one argument: a callback function.
- The resolve and refuse arguments are passed to the callback function.
- Execute operations within the callback function. Later call resolve when everything goes well.
- Call reject if the expected operations is not working as planned.

Promise-Consumers

Register functions using `.then` and `.catch` methods to use promises.

- `Then()`:

If a promise is either resolved or refused, `then()` is called. It can also be characterized as a profession that takes data from a promise and successfully performs it. Two functions are passed as inputs to the `then()` method.

If the promise is addressed and a result is obtained, the foremost function is called.

If the promise is refused and resulted in an error, the second function is called. The `catch()` is a method for catching something.

- `catch()`:

`catch()` is triggered when a promise is either refused or an error occurs during execution. If there is a possibility of receiving an error at any point, it is used as an Error Handler.

Parameters

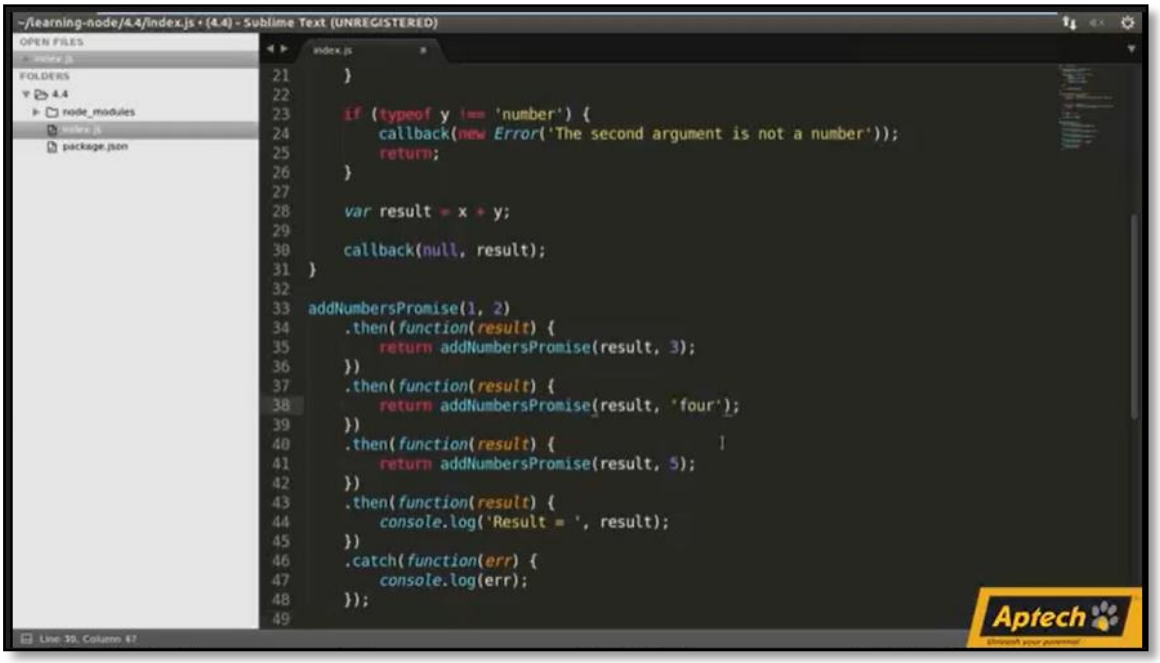
One function is passed as an argument to the `catch()` method.

Errors or promise rejections are handled by this function (the `.catch()` function calls `.then(null, errorHandler)` internally, so `.catch()` is merely a shortcut for `.then(null, errorHandler)`.

Applications

- Promises are used to handle asynchronous events.
- Asynchronous http requests are handled using promises.

Show: Play the video from 2.40 to 4.18.



Content: Tell students that Promise Chaining is syntax for chaining together many asynchronous activities in a specific order. Only after an asynchronous activity is executed another asynchronous task is completed. This is useful in large Web applications.

Note: The async npm module in Node.js can be used to chain commands. The most generally used approaches for chaining functions are provided by the async module:

- Parallel (tasks, callback): These tasks are a set of functions that, in practice, operate in parallel due to I/O switching. The callback function is called when any function in the collection tasks returns an error. The data is sent to the callback function as an array once all functions have been performed. It is not required to use the callback function.

- Series (tasks, callback): Each task function runs only after the previous one has finished. When any of the functions throw an error, the callback is fired with an error value and the following functions are not run. The data is sent as an array to the callback method after the tasks are completed.

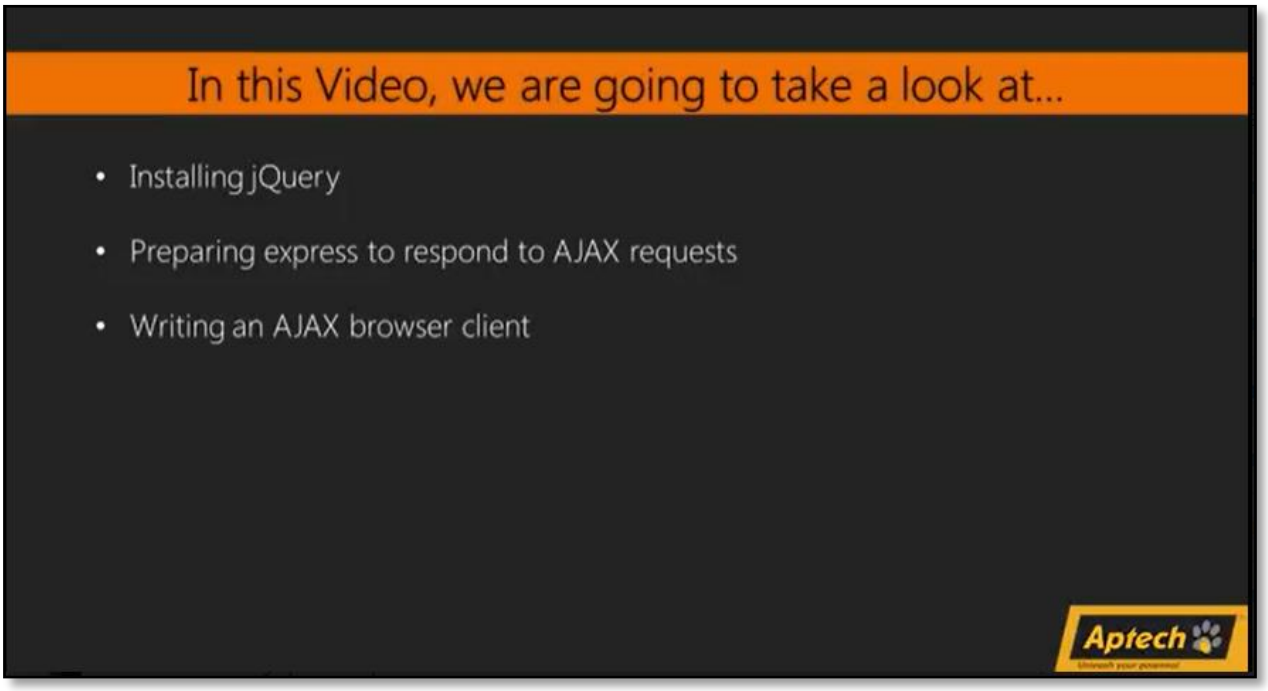
4.5 Making Ajax Calls

CLASS DURATION: 10 MINUTES

VIDEO DURATION: 4.43 MINUTES

FACILITATION GUIDELINES -

Show: Play Making Ajax Calls: Session 4 Video – 4.5.



Discussion:

Ajax refers to a combination of asynchronous JavaScript and XML. It is used to communicate with the server in an asynchronous manner. Ajax is a technology that allows a user to obtain data from a server and update the page, as well as send data to the server without changing the current client page. The term 'Ajax' refers to a programming technique.

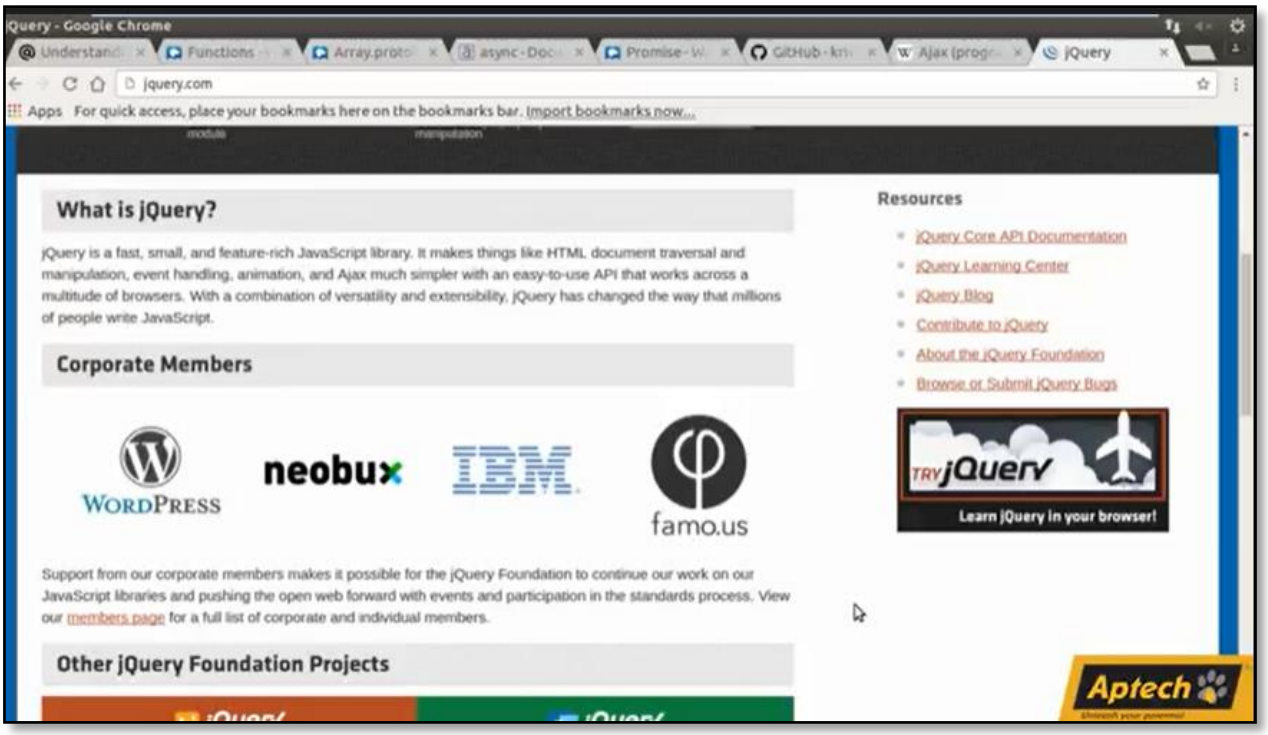
Show: Play the video from 0:33 to 1.06.

Content: Explain to the students that jQuery is a JavaScript library, which performs the same functions as vanilla JavaScript, but with less code. Doing things in plain JavaScript has become easier; upgrades have reduced the requirement for jQuery. It is still used in roughly 76 percent of projects, despite its waning popularity.

Our goal is to combine jQuery and Node.js. The jQuery module in Node.js allows users to use jQuery.

Note: The 'jquery' module should be used instead of the 'jQuery' module, which is deprecated.

Show: Play the video from 1:50 to 3:00.



Content: Explain students on how jQuery works with Node.js.

- Creating *package.json* file: The *package.json* file, that keeps track of the modules and dependencies, is created with following command.

  ```
  npm init -y
  ```

  The '-y' tag makes yes, the default answer for all the questions asked while creating the *package.json* file.

- Installing the jquery module: To install the jquery module, use the following command.

  ```
  npm install jquery
  ```

- The jsdom module is being installed: However, jQuery is a frontend JavaScript library and requires a document window to work in the backend. The library 'jsdom' is used to parse and interact with HTML. It is not a Web browser, just close to that. To install the jsdom module, use following command.

```
npm install jsdom
```

- Importing the jsdom module: Use the require method to import the jsdom module.

```
const jsdom = require('jsdom')
```

- Creating a new window: Using HTML code as the parameter, create a JSDOM object for creating a window with a document. Following is the code to do this:

```
const dom = new jsdom.JSDOM('')
```

- Importing jQuery and providing it with a window: After the window with a document is generated, jquery module can be used by providing it with the window created. To import the jquery module, following is the code:

```
const jquery = require('jquery')(dom.window)
```

Show: Play the video from 3:00 to 4:30.

Content: Explain the working of jQuery with Node.js:

1. An event occurs in a Web page (the page is loaded; a button is clicked)
2. An *XMLHttpRequest* object is created by JavaScript.
3. The *XMLHttpRequest* object sends a request to a Web server.
4. The server then processes the request.
5. The server sends back the response to the Web page.
6. The JavaScript reads the response.
7. The JavaScript takes a proper action (such as page update).

Additional Reference:

Refer to following links for more information:

https://www.npmjs.com/package/jquery’https://www.npmjs.com/package/jquery

4.6 Building Restful Web Services

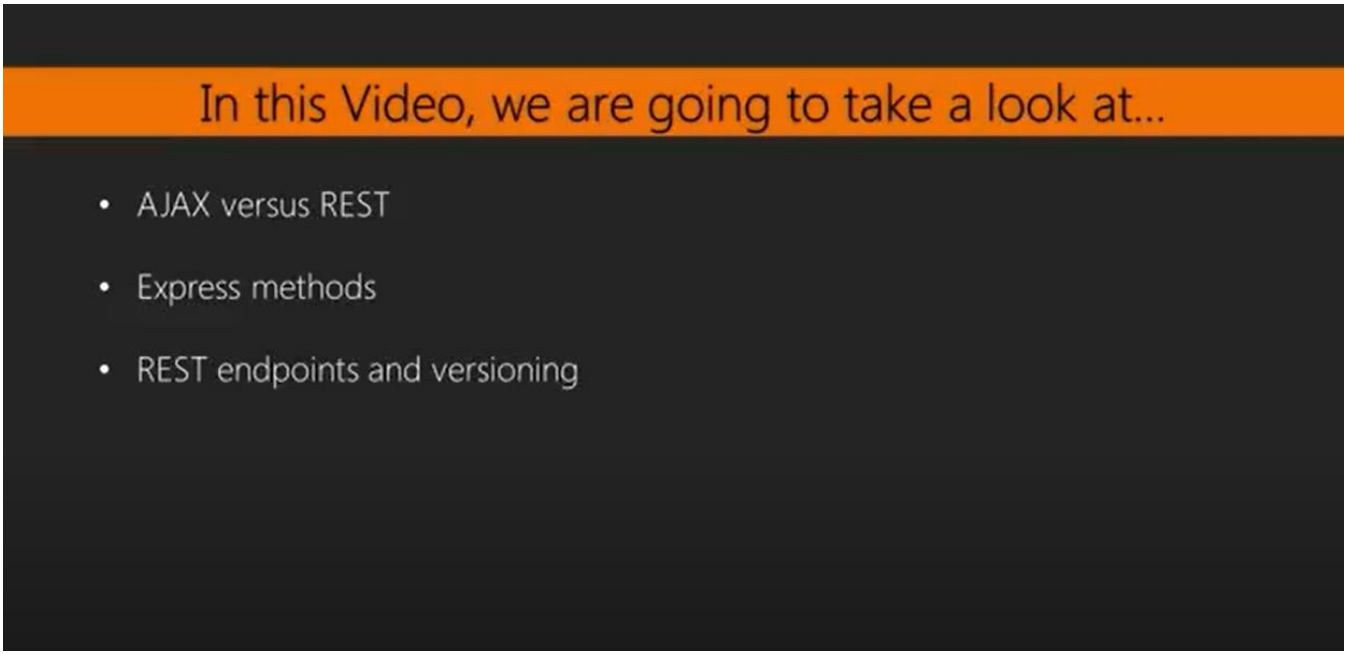
CLASS DURATION: 10 MINUTES

VIDEO DURATION: 6.06 MINUTES

FACILITATION GUIDELINES -

Show: Play Building Restful Web Services: Session 4 Video – 4.6.

Discussion: In this video, students will know about the difference between AJAX and REST, Express methods, and REST endpoints.



Show: Play the video from 1:00 to 2:30.

Content: AJAX Vs REST

The terms AJAX and REST (Representational State Transfer) API are not interchangeable. There is a lot more to say about REST, but here are some key differences between the two in two minutes.

The use of a group of technologies to make asynchronous server requests and process the client response, such as altering the HTML of a shopping list after adding something to the cart, is an example of REST API.

- REST is a server-side architecture standard for organizing applications. It specifies which server endpoints are available and how a client can communicate with them.
- REST allows you to interface with a resource by using URLs to identify it and several HTTP verbs to interact with it.
- REST covers, among other things, the creation of URLs (that are used to uniquely identify a resource in the system) and the connection of CRUD activities over that resource to HTTP verbs.

Show: Play the video from 2:30 to 3:30.

Content: Express Methods

Explain students that Express is a Node.js Web application framework that includes a wide range of features for developing Web and mobile applications. Features make it easy to swiftly build Node-based Web apps.

Following are a few core elements of the Express framework.

- Allows middleware to respond to incoming HTTP requests.
- Creates a routing table that may be used to do various actions based on the HTTP Method and URL.
- Allows the user to dynamically render HTML pages by giving variables to templates.

Methods

However, data is sent in the body, the post method allows user to communicate enormous amounts of data. Although the Post method is secure because the data is not visible in the URL bar, it is not as widely utilized as the GET method. The GET method, on the other hand, is more efficient and popular than the POST approach.

The `router()` method in the Express framework is used to create HTTP endpoints.

Rest endpoints and versioning

The evolution of all application APIs is unavoidable. The evolution of public APIs with an undetermined number of clients, such as RESTful services, is, on the other hand, a problematic matter. Customers may not be able to handle the modified data effectively, and there is no way to notify them all, so the user must ensure that our APIs are backward-compatible. It can be challenging to version a Node.js API, especially if it has a high number of endpoints and must go through API breaking changes.

When defining routes in index.js, the functions in routes are called. A user manages migrating between API versions for each endpoint here. Requests contain the actual implementation for each endpoint and separated into numerous versions in the future to accommodate API breaking changes between two versions of our API.

The following step is to route API requests to the correct implementation version. The user will change index.js routes to add the v1 and v2 version numbers. However, it is possible that the user will discover a problem.

If further versions of the API required to be developed for future updates, more endpoints required to be introduced but only to specific versions of the API, and earlier API versions are gradually deprecated and eventually unsupported. This might create a routing system that is very difficult to maintain.

In-Class Question: Is AJAX an API call?

Answer: AJAX (Asynchronous JavaScript and XML) is a set of tools for making server calls and retrieving data. In this post, let us look at how to use AJAX to make a simple API call.

4.7 Developing Rest API

CLASS DURATION: 10 MINUTES

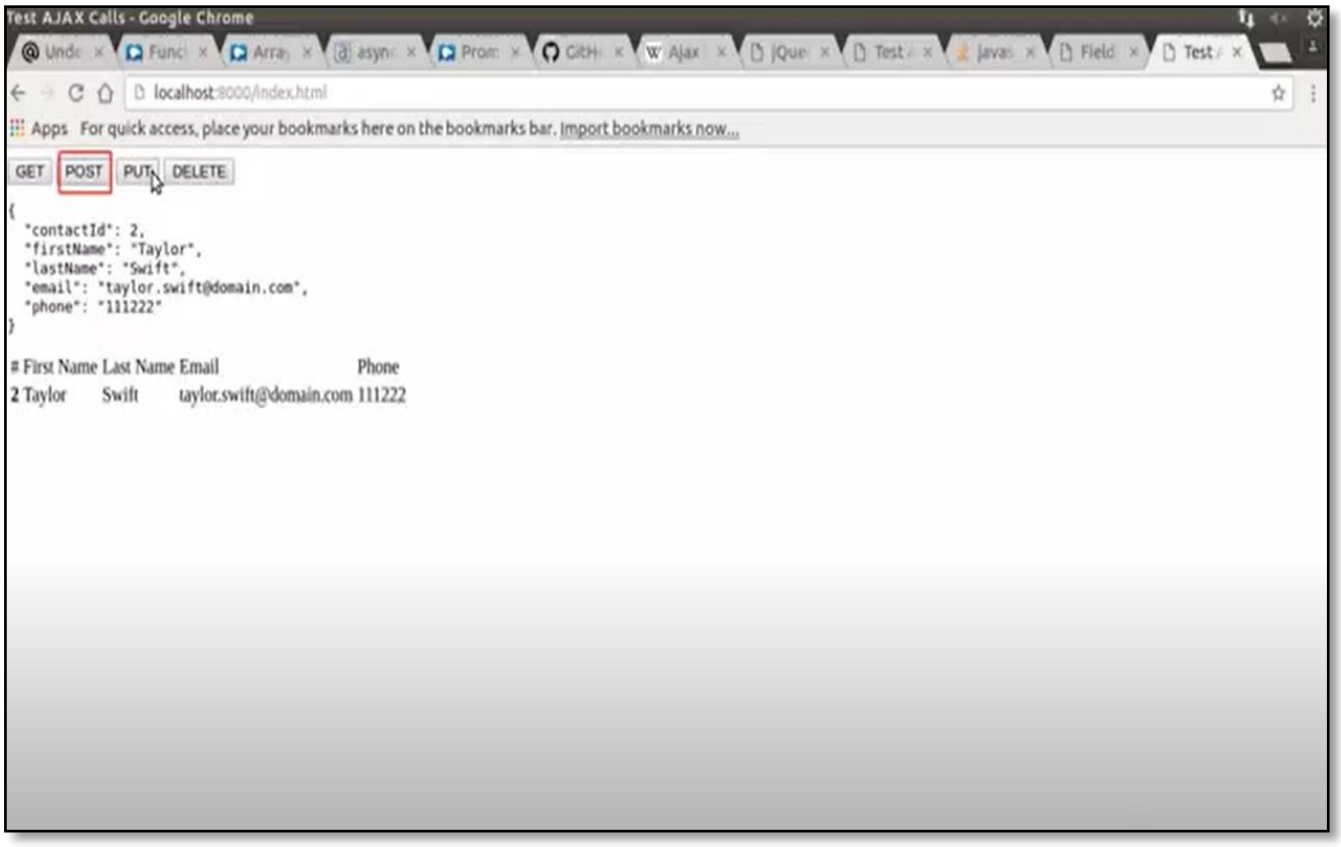
VIDEO DURATION: 4.58 MINUTES

FACILITATION GUIDELINES -

Show: Play Developing Rest APIs: Session 4 Video – 4.7.

Discussion: Application Programming Interfaces (APIs) allow software to communicate with other software, both internal and external, in a uniform manner, which is essential for scalability and reusability.

Content: Explain to the students that public-facing APIs are becoming more widespread in online services. These public-facing APIs can be used by other developers to add features such as login pages for social media, payments, and behavior tracking to their apps. The de facto standard for this is Representational State Transfer (REST).



Show: Play the video from 1:00 to 2.30.

Content: Express is a common fare in developing a REST API back-end Mongoose that will connect the user back-end to a MongoDB database that a reader supposed to be familiar with.

REST APIs are used to access and update data using a common set of stateless activities. These actions are vital to the HTTP protocol and constitute essential Create, Read, Update, and Delete (CRUD) capability in a one-to-one manner:

- `OST` (create a resource or provide data)
- `GET` (recover an index of resources or an individual resource)
- `PUT` (create or replace a resource)
- `PATCH` (update/modify a resource)
- `DELETE` (remove a resource)

By creating an endpoint for each action, users can develop a REST API utilizing these HTTP operations and a resource name as an address. Furthermore, by following the pattern, the user will have a solid and easy-to-understand foundation on which to quickly evolve and maintain the code. As previously indicated, the same foundation will be used to integrate third-party features, with the bulk of them relying on REST APIs to expedite integration.

Following operations user can perform on Rest APIs:

- POST on the endpoint /users (create a new user)
- GET on the endpoint /users (list all users)
- GET on the endpoint /users/:userId (get a specific user)
- PATCH on the endpoint /users/:userId (update the data for a specific user)
- DELETE on the endpoint /users/:userId (remove a specific user)

Our project has three module folders:

- common (dealing all shared services and information shared between user modules)
- users (all details regarding users)
- auth (dealing JWT generation and the login flow)

Note: Run `npm` install now (or yarn if you have it).

Show: Play the video from 2:30 to 4:50.

Content: Tell students about Middleware for Permissions and Validations. The foremost part is to identify who all has access to the user resource.

Following are the situations a user has to deal with:

- Users can be created in the open (registration process). For this instance, JWT is not used.
- The logged-in user information is kept private, but admins can edit that information.
- Only the administrator has access to the section that is used to delete user accounts.

After user have been defined these scenarios, user must require a middleware that constantly checks to see whether a valid JWT is using or not.

4.8 Mocking Up CRUD

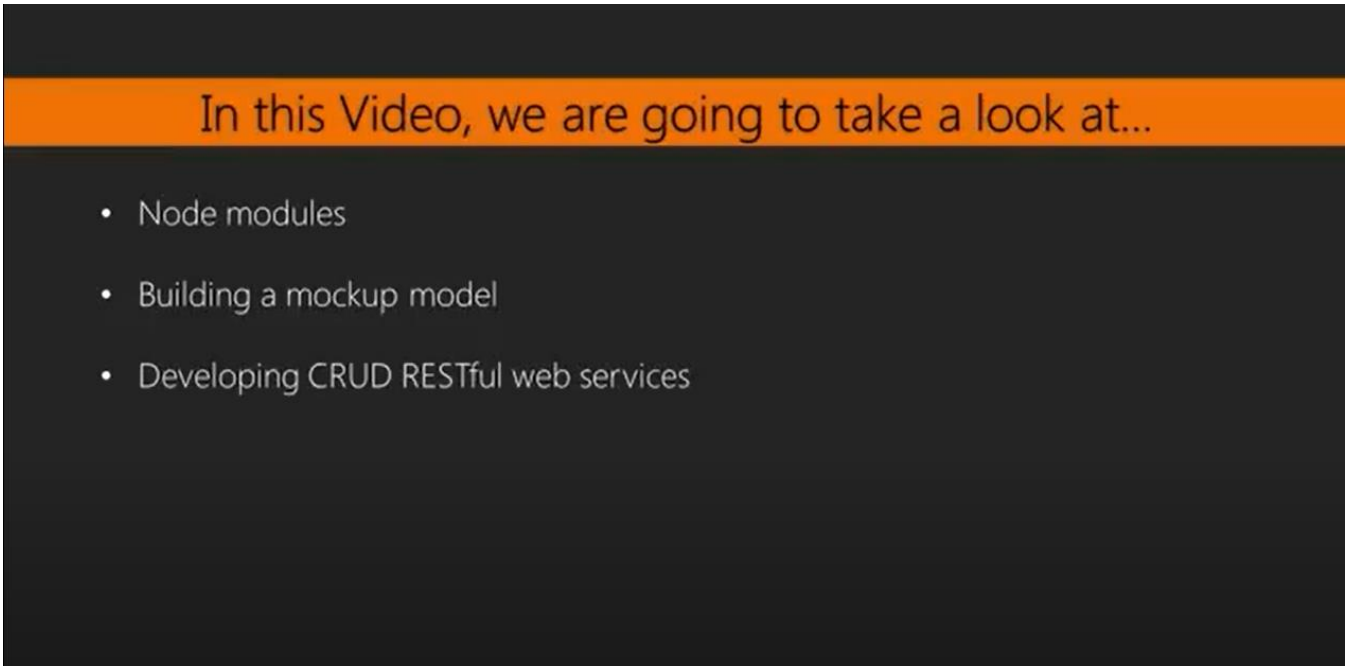
CLASS DURATION: 20 MINUTES

VIDEO DURATION: 8.24 MINUTES

FACILITATION GUIDELINES -

Show: Play Mocking up Crud: Session 4 Video – 4.8.

Discussion: In this video, students will learn about Node modules, building a mockup model and developing CRUD RESTful Web services.



Content: Node modules

Modules are wrapped code units that interface with an external application based on their relevant functionality in Node.js. Modules might consist of a single file or a group of files/folders.

Following are the three sorts of modules:

Core Modules: Node.js comes with some built-in modules that are included with the platform. The require function is used to load these modules into the program.

Syntax:

```
var module = require('module_name');
```

Local Modules: Local modules are developed locally in your Node.js application, unlike built-in and external modules. Explain students how to make a simple calculating module that performs a variety of calculations.

Note: This module also hides functionality that is not required outside of it.

Third-party modules: Third-party modules installed using the Node Package Manager (NPM). These modules can be installed locally or globally in the project folder. Mongoose, Express, Angular, and React are a few most popular third-party modules.

Example:

- `npm install express`
- `npm install mongoose`
- `npm install -g @angular/cli`

Show: Play the video from 1:00 to 3:00.

Content:

Students should be informed that the mockup is a static image with medium to high fidelity. Color palettes, content layouts, typefaces, icons, navigation visualizations, pictures, and the general feel of future software product design and user experience are all included.

Every UI/UX designer is a one-of-a-kind individual. Mockup development has as many ways as there are UI/UX designers. For mockup fidelity or development times, there are no agreed criteria. Some designers begin on mobile devices, while others prefer to begin with desktop computers. Everything should be conveyed to the designer.

Each Website or mobile app is distinct. Even if you know the designer you are working with, you still cannot expect a great mockup development estimate immediately. Each case is distinct. The Wireframes should be produced from UI/UX designer for your application or Web platform. After that, the wireframes can be turned into actual mockups.

Mockups are accurate representations of the real world. It is an excellent opportunity to examine how all of your design choices interact. What if the color scheme clashes with the form user have created? Mockups allow you to see the final result before the development process begins.

Mockups are simple to change. It is painless to make changes in mockup tools during the mockup stage than it is to make changes later in the coding stage. Developers will appreciate not having to make any changes to the product design. They are simple to comprehend and closely match the finished software product.

Conduct thorough business analysis and competitor study before asking for Website mockups development. Users should also get advice from your development and marketing teams on how to create mockups. Users will learn numerous helpful lessons during the process that users might not have considered at first. The interaction is where the action is.

Show: Play the video from 3:00 to 6:00.

Content: Students will learn about Developing CRUD Restful Web services. Before construct a REST API, the user must understand a few different sorts of HTTP methods.

The Create, Read, Update, and Delete (CRUD) actions are represented by following methods:

- `POST`: Used to submit data, and it is generally used to create new entities or edit existing ones.
- `GET`: Used to request data from a server, and it is generally used to read data.
- `PUT`: This method replaces the resource with the one that was submitted and is commonly used to update data.
- `DELETE`: This method is used to remove an entity from the server.

To edit stored data, you can use either `POST` or `PUT` because `PUT` may be omitted and you can select whether or not to utilize it. Do not use the `PUT` technique when utilizes `POST` for both creation and updating.

What is the difference between a Restful Web Service and an API?

API stands for Application Programming Interface and it describes how one piece of code communicates with another. API is a term used in Web development to describe the process of data retrieval from an Internet service. The API documentation offers you a list of URLs, query parameters, and other details on how to use the API and what kind of response you can expect from each endpoint.

REST, instead, is a set of guidelines, standards, and guidelines for developing a Web API because there are so many alternatives, having a system for defining an API that everyone agrees on saves time when making decisions during the development process.

In Web apps, the term 'routing' refers to how endpoints (URIs) respond to client requests. `App.get()` and `app.post()`, for example, handle `GET` and `POST` requests, respectively, and are defined using Express app object methods that correspond to HTTP methods. The

program calls the supplied `callback` function when it finds a match for the provided route(s) and method(s).

Additional Reference:

Refer to following link for more information:

https://riptutorial.com/node-js/example/22918/a-simple-implementation-of-ajax

4.9 Testing API with CURL

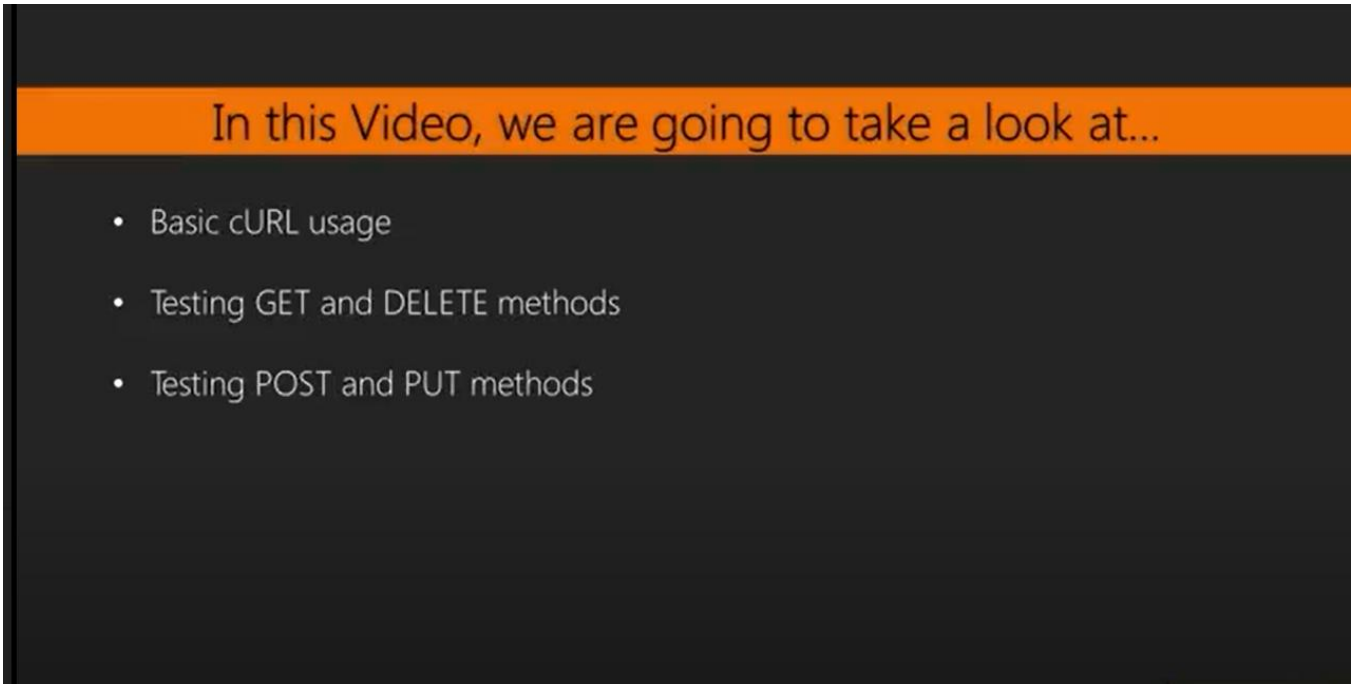
CLASS DURATION: 10 MINUTES

VIDEO DURATION: 3.56 MINUTES

FACILITATION GUIDELINES -

Show: Play Testing APIs with Curl: Session 4 Video – 4.9.

Discussion: In this video, students will learn about usage of cURL, test `GET` and `DELETE` methods and test `POST` and `PUT` methods.



Content: cURL (Client for URLs) is a command-line program for Linux, Windows, and macOS for transporting data over the Internet via HTTP, HTTPS, FTP, and SFTP protocols. Sending `GET`, `POST`, and `HEAD` requests, as well as getting HTTP headers, downloading HTML pages, uploading files, and submitting forms are all possible with the server.

The documentation for cURL lists 383 command-line flags, making it hard to find what you are looking for.

- The `GET` method is used to retrieve or obtain data from a server using a URL. It conducts the read operation in `REST CURD`.
- The post method is used to transmit data to the server, such as uploading a file, transferring data, or adding a new row to the back-end table of any online form. In a nutshell, the post method is used to put new things into the back-end server. It conducts the create operation in the `REST CRUD` operation.
- It is worth noting that users have to provide the request method, headers, and body. These attributes are defined by default for the `GET` request, but the user must specify them for all other types of requests, therefore, user does not pass them in the `GET` Method.
- The general usage of the `PUT` method is to update an existing resource.
- The resource is initially recognized by the URL in the HTTP.`PUT` method, and if it already exists, it is updated; otherwise, a new resource is generated. To be precise, when the resource already exists, update it; or else, build a new resource.
- The `PATCH` technique is used to update the resource properties values.
- The `DELETE` method is used to remove a resource whose URI is supplied.

Conclusion: Give the summary of all the topics that is covered in this session.

QUERIES REGARDING SESSION

CLASS DURATION: 10 MINUTES

FACILITATION GUIDELINES

Ask students if they have any queries regarding the session and resolve the queries.

USING MYSQL AND MONGODB WITH NODE.JS

Greeting: Welcome the students back to the course, greet them, and ask if they are ready to get started with the next session of the course. Tell them that this session will carry forward from the previous session and continue explaining Session 5.

Quick Recap

Recap: Give a quick recap of the topics covered in the last session and ask the students if they have any doubts. Resolve their doubts and then, inform them that this session will be a long one as multiple topics will have to be covered in it.

Start the session.

5.1 Setting Things Up

CLASS DURATION: 10 MINUTES
VIDEO DURATION: 6 MINUTES

Show: Play Using MySQL and MongoDB with Node.Js: Session 5 Video – 1.1.

Discussion: In this video, students will learn how to install and set up MySQL database. They will also learn how to install a MySQL connector to connect with Node and to creating a structure for an application. MySQL will be installed in Ubuntu for this session.

MySQL:

MySQL is a popular and widely used database system. It is commonly used with PHP and Apache server for communicating with the databases. It is also used with Node. The installation process is quite convenient for each operating system such as Ubuntu, Windows, Mac, and so on.

Question: Ask the students which other database systems are they familiar with and provide a brief idea about pros and cons of those databases compared to MySQL.

Show: Play the video from 0:54 to 1:43.

Content: Explain to the students that in this step MySQL is being installed in an Ubuntu operating system.

Installation:

Step 1: Go to the Ubuntu terminal.

Step 2: Type `sudo apt-get update` command and click Enter.

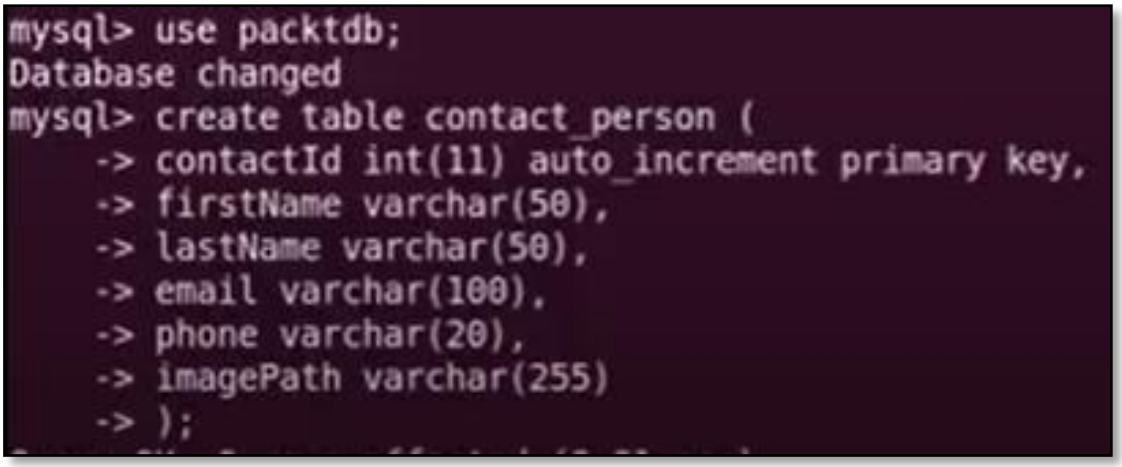
Step 3: Type `sudo apt-get install mysql-server-5.5` and click Enter. A prompt pops up asking for the password for the administrator user account. Type again to confirm the password and press Enter.

Step 4: Type `sudo mysql_install_db`. The MySQL server and the data dictionary are now installed on the system.

Step 5: For a successful installation confirmation, type `mysqladmin -p -uroot version`. Type in the password which was asked in Step 3 (administrator account password) and press Enter. It displays a brief description of the MySQL server installed.

Show: Play the video from 1:45 to 2:20.

Content: Explain to the students that in this step they will learn how to create a database and a table through the command line. The command `mysql -p -uroot` is typed in the terminal to enter the MySQL command line. Then, a database named *packtdb* is created using the command `create database packtdb`. The database is used for further operations using the command `use packtdb`. A table `contact_person` is created using the following codes:



Show: Play the video from 2:47 to 3:15.

Content: Explain to the students that a new directory *6.1* is created from the terminal then, the path is changed to the directory. The project is initialized using `npm init`. Few basic packages such as express ejs body-parser, morgan, and mysql are installed. The project folder is opened in the sublime text editor, the source code of which is provided with the session.

Show: Play the video from 3:20 to 4:10.

Content: Inform the students that *index.js* is the starting point of any nodejs project.

Description of significant files in a Node Project:

- The folder called app contains all files for server side programs. It has two sub-folders namely, *model* and *router*.
- The *public* folder holds files for client side programs.
- The *assets* folder contains data related to CSS, JS, images, and so on. The *uploads* folder contains the images of the contacts regarding the project.
- The *views* folders has *pages* with ejs templates and *partials* sub-folders.

Show: Play the video from 4:30 to 5:29.

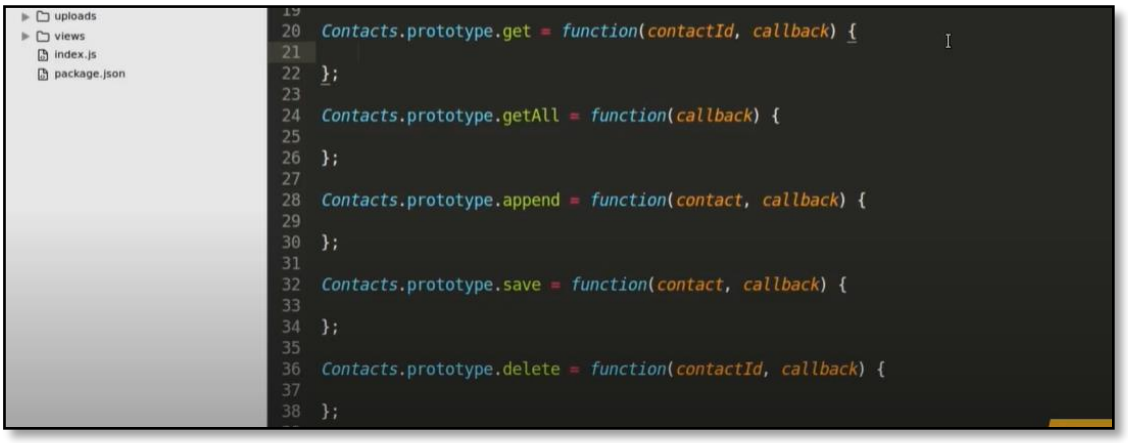
Content: Explain to the students that a file named *contacts-router.js* is made under the folder *router* to map with the *contacts.ejs* template in the *pages* folder. Similarly, *profiles-router* file is made to map with the *profiles.ejs* template in the *pages* folder. An express router is created which manages the landing page. Both files are imported in the *index.js* file using the `require()` method and the paths are mapped to their respective pages.

5.2 Connecting To MySql

CLASS DURATION: 10 MINUTES
VIDEO DURATION: 5 MINUTES

Show: Play Connecting to MySQL with Node.Js: Session 5 Video – 1.2.

Discussion: In this video, students will learn how to connect to the MySQL database server and perform Create, Read, Update, and Delete (CRUD) operations on tables wrapped in a data model. These are the four basic and major operations required in any database or table.



Show: Play the video from 0:37 to 0:45.

Content: Explain to the students that in this step a new directory *6.2* is created in the terminal and its path is changed to this directory. The project used in video 1.1 *contacts_person* is used here. The contents of the project are copied using `cp -Rp /6.1/*`. The project is then opened in Sublime Text Editor.

Show: Play the video from 0:46 to 1:54.

Content: Explain to the students that a file *contact-model-mysql.js* is being created in the *model* folder. Much of the things are similar to what was done in video 1.1 regarding files stored under the *router* folder. In this file, mysql is imported and a connection is being set up using the `createConnection()` method of MySQL. The values passed in the parameters passed are relative to each user. Then, a class contacts the data model to perform CRUD operations. A connection to the database server must be made when the class Contacts is initiated. A `recordNotFoundError` class and getter-setter methods are used.

Show: Play the video from 0:57 to 3:24.

Content: Explain to the students that in this step, the getter-setter methods are being implemented with the same logic as was observed in the mockup data model. The addition is setting up the connection to the database server and calling a query() method.

Working of CRUD Operations

For the `get()` method, the query is made to return the results of a single contact using the SELECT statement after passing the required `contactId` as a parameter for searching.

The `getAll()` method just requires the `SELECT * from table_name` as a query to return records of all contacts.

The `append()` method allows the user to insert new records in the table. Thus, the values of the columns are passed as parameters along with the insert statement in the query.

The `save()` method is similar to the append method. It takes the `contactId` of the contact along with the values to be updated to finally update the record of that particular contact.

The `delete()` method requires a query with the `contactId` of the contact to be deleted passed as a parameter.

These functions must be exported using the keyword exports so that they can be used by other programs.

Show: Play the video from 3:27 to 4:31.

Content: Explain to the students that in this step a test model is created to test the functionalities of the data models implemented. A new file *test-mysql.js* is created in the project root folder. Async is imported in this file to form the test cases. The *Contacts* data model is imported as well and a new instance of it is created. Two sample records namely `newContact` and `oldContact` are created to test CRUD operations on them. A helper function is also created to display the test case results. Now, a series of functions are created using `async.series()` method.

The sample record `newContact` is appended/inserted into the table followed by the `getAll` methods which would return only one record. Consequently, add the `save` method passing the values to update the phone number of the respective contact. Now, the `get` method is added with the id as 1 (as only one record is there) followed by the delete method to delete the `contactId` of 1 itself. Finally, the `getAll` method is run again to check whether the contact had been successfully deleted.

Tips: Tell the students to make use of plug-ins in their preferred choice of IDE so that their code writing process is made more comfortable. With writing more than 3000 lines of code, it becomes cumbersome to maintain the neatness and check on closing of brackets, semi-colons, and so on. For sublime text IDE, plug-in such as JSHint, BracketHighter, and so on proves to be very useful in this case.

Refer to this link for more details on plug-ins:

https://www.sitepoint.com/essential-sublime-text-javascript-plugins/

Show: Play the video from 4:30 to 5:04.

Content: Explain to the students that before the scripts are executed, the project must be installed with the async dependency as it was not previously installed and would result in an error. Once the dependency is successfully installed, the project is executed. The results are as expected.

5.3 Providing REST APIS

CLASS DURATION: 16 MINUTES
VIDEO DURATION: 7 MINUTES

Show: Play Providing Rest APIs: Session 5 Video – 1.3.

Discussion: In this video, students will learn about REST APIS, how they are used and what their advantages are. REST APIS will be created in this video relating to the MySQL data model. This video also deals with multipart data submission through forms from the client side.

Show: Play the video from 0:41 to 0:50.

Content: Explain to the students that firstly, a new directory *6.3* is created in the terminal and its path is changed to this directory. The contents of the project are copied using `cp -Rp /6.2/*`. The project is then opened in Sublime Text Editor.

Show: Play the video from 0:51 to 1:45.

Content: Explain to the students why Multer is being used for this project.

Why is Multer used?

Creating REST APIS in the backend server includes handling multiple HTTP requests such as POST, PUT, GET, and DELETE. For the POST request type, data payload must be passed from the client front-end to the server. The type of data payload is originally in the HTML form in which the content type might be URL-encoded, JSON or multi-part.

For this project, images of contacts are required to be uploaded. Thus, the content type of an image used is multipart. As it was mentioned in the earlier sessions that body-parser can handle URl-encoded and JSON content type, but it cannot handle Multipart content type. Therefore, another express middleware library called Multer can be used here. Multer is commonly used to upload files.

Show: Play the video from 1:47 to 6:02.

Content: Explain to the students that in this step the REST API server is being created. Multer is firstly, installed in the node project and a new file called *contact-api-router.js* is created. Express, multer, path, and fs are imported in this file.

API Requests

- **GET Request**: The GET request is used to retrieve all records from a particular table. In this case, records of all contacts are returned from the *contacts_table* through this request. The results are returned in the JSON format with a status of 200 as the successful response status code.
- **POST Request**: The POST request is used to post/insert new records to the server. Multer is used here with the configured storage settings to aid the `imagePath` variable which holds the uploaded image in the payload. As Multer handles only Multipart content type, a body-parser object is created to handle the data of the other content type. The append method is called inside the POST method and the new contact object created is passed as a parameter.
- **PUT Request**: The PUT request is used to update records in a table. Similar to the POST request, the `contactId` parameter is passed in the path and a new contact object is created with the updated record. The method checks whether the image is updated or not and accordingly the changes are made. The save method is then used to update the corresponding contact.
- **DELETE Request**: The DELETE request simply deletes records. It is passed the `contactId` value in its path and performs a delete operation.

Additional Reference:

Refer to this link to know more in details about HTTP Requests: https://nodejs.dev/learn/making-http-requests-with-nodejs

Show: Play the video from 6:03 to 7:03.

Content: Explain to the students that the router now must be linked to the main *index.js* page. For this, the body-parser must be set up. The *contactApiRouter* is imported and is mapped to the */v1/contact* path. Tell the students that the APIS can also be tested using CURL. For that, the command `curl http://localhost:/3000/v1/contact` is typed. This displays records of all contacts in the table. The request types are tested one by one with sample data and hence, the results display that all request types were handled correctly.

In-Class Question: Data payload passed from client front-end to the server is of which type?

Answer: For the POST request type, data payload must be passed from the client front-end to the server. The type of data payload is originally in the HTML form in which the content type might be URL-encoded, JSON, or multi-part.

5.4 Linking Up the Client

CLASS DURATION: 10 MINUTES
VIDEO DURATION: 6 MINUTES

Show: Play Linking Up the Client: Session 5 Video – 1.4.

Discussion: In this video, students will understand how to develop a front-end client side template view which connects with the APIS to perform HTTP requests.

Show: Play the video from 0:32 to 0:37.

Content: Explain to the students that firstly, a new project folder *6.4* is created in the terminal and its path is changed to this corresponding folder. The contents of the project are copied using `cp -Rp /6.3/*`. The project is then opened in Sublime Text Editor.

Show: Play the video from 0:40 to 1:20.

Content: Explain to the students that REJS pages are static pages and not dynamic and interactive. In this step, the pages are made dynamic.

Explain Tight Coupling and its solution:

The contact person records must be passed to the *contacts.ejs* template in the `GET` method handler. This can be done by making direct calls to the MySQL data models as it is in the same project, but it would avoid tight coupling between the router and the data model. Therefore, a better way is used. That is, calls are made to the REST APIS. Thus, HTTP requests are to be made in the express router. For doing that, the request library must be installed.

Show: Play the video from 1:26 to 2:08.

Content: Explain to the students that in this step the request library is installed. Then, the *contacts-router* is opened and the request library is imported. A GET method is created and a request is made to the REST endpoint to get records of all contacts in JSON format. If there is no error and the status code is 200 then, the response body is returned to the EJS template using the rows variable. Again, a GET method is created in the *profiles-router.js* file and the request library is also imported there. An empty new contact object is created and passed it to the *profile.ejs* template file.

Show: Play the video from 2:10 to 4:10.

Content: Explain to the students that in this step the EJS templates are being modified to remove the hardcoded data values and replaced by the values passed to the templates. In the *profile.ejs* template, the hardcoded values of `contactId`, `firstname`, `lastname`, `email`, `phone`, and so on are replaced with variables holding data of each contact as `contact.contactID`, `contact.firstname`, and so on.

BootBox.js:

Dialog boxes are used for better user experience and interaction. As pure bootstrap cannot provide customized effects of dialog boxes, BootBox.js is used instead.

It is included in the *profile.ejs* file. Now, jQuery is used to create the logic for the form submit button. In the script block, a function is created to first handle a default submit and then, an ajax POST request is created to the contact API endpoint. Highlight on the fact that the contentType, cache and processData must be set to false to avoid any content type error as the data being passed contains a Multipart content type data. An error message is displayed with bootbox in case any error happens.

Show: Play the video from 4:14 to 5:00.

Content: Explain to the students that in this step the file *Contact.ejs* is modified. The number of records of contacts can be denoted by rows.length.

An IF condition is used to check whether there are any records to display or not. If yes, a FOR loop is used to iterate through the array rows passed.

Each array element is assigned to a contact and a `fullName` variable is created to store the first and last name of each contact iterated.

Under the div of class profile-image, an If condition is added to check whether a contact has any `imagePath` value or not. If yes, the image path is added else a default no-image path is added.

The other hardcoded values are change to the contact objects attributes. The results are then tested by typing the command `nodemon index` in the terminal and run. The browser is refreshed and after adding a sample contact the results are displayed as expected.

Additional Reference:

Refer to this link to learn the significance of nodemon:

https://www.npmjs.com/package/nodemon

In-Class Question: What is used to create the logic for the form submit button?

Answer: JQuery is used to create the logic for the form submit button.

5.5 Finishing Touch

CLASS DURATION: 15 MINUTES
VIDEO DURATION: 6 MINUTES

Show: Play Finishing Touch: Session 5 Video – 1.5.

Discussion: In this video, students will learn to use the PUT and DELETE AJAX calls and how to add interface features to the application.

Show: Play the video from 0:35 to 0:40.

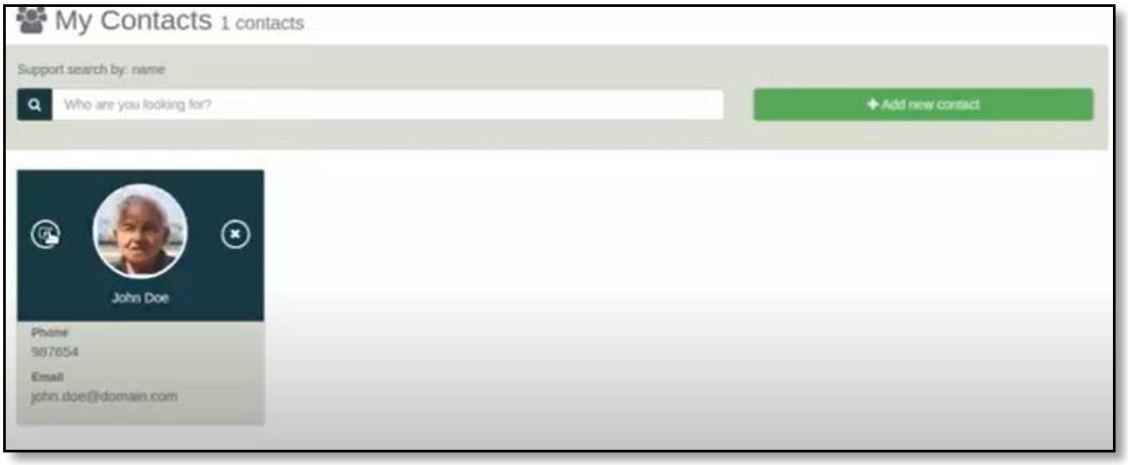
Content: Explain to the students that firstly, a new project folder 6.5 is created in the terminal and its path is changed to this corresponding folder. The contents of the project are copied using `cp -Rp /6.4/*`. The project is then opened in Sublime Text Editor.

Show: Play the video from 0:41 to 2:56.

Content: Explain to the students that they will learn how to add a feature in the application which will allow users to edit an existing contact record.

Edit Button Feature:

Show: Show the image to the students to identify the edit button on the left side of the contact card.



First Part:

For this, the profile template can be reused. In the *Profile-routers* a GET calmethod handler is created to handle this editing feature. The parameter passed in the URL must be the `contactId` attribute of the contact record which will specify the record to be edited. Then, an AJAX call is made to the REST API where the result returned is assigned to the `oldContact` object variable.

The handling of a new record and an existing record is different. Thus, to notify the *profile.ejs* of this difference, a new variable `newRecord` is added and set to false for the concerned GET method whereas it is set to true for the method handling the new record.

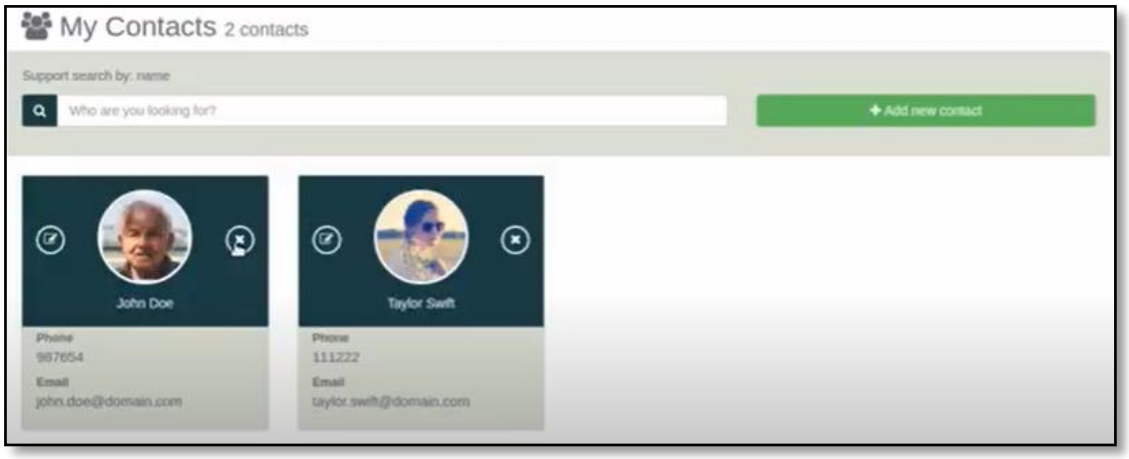
Second Part:

Changes are made in the *profile-router.js* file to make an AJAX call to the POST and PUT Requests. An IF condition is set to check that if the `newRecord` is true, the HTTP request type is set to POST and the URL to the POST endpoint otherwise it is set to PUT and the URL to the PUT endpoint with the `contactId` to specify the record to be edited. In the *contacts.ejs* file, the path of the EDIT button is changed accordingly. The results are tested and displayed as expected.

Show: Play the video from 2:57 to 4:10.

Delete Button Feature:

Show: Show the image to the students to identify the delete button on the right side of the contact card.



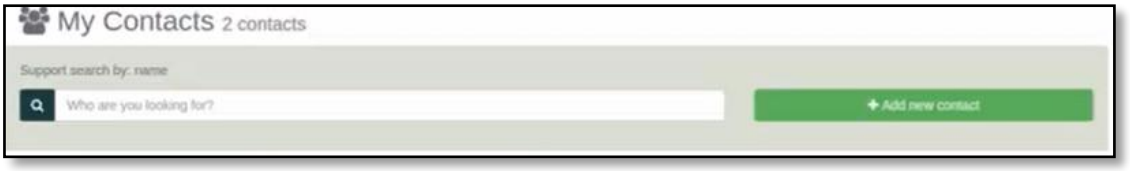
In the DELETE button, the `contactId` is passed to the onClick event handler. That is, when the user clicks the DELETE button of a particular contact record, the next series of events is determined by the `onClick` event handler. Now, for a double-check of deletion, a bootbox is used to create a confirmation dialog box in the script block. The `onClick` event is handled and the `contactId` passed is used to identify the record to be deleted. In the *contacts.ejs* file, a delete function is written and an AJAX call is made to the delete endpoint. The results are tested and displayed as expected.

Show: Play the video from 4:23 to 5:35.

Content: Explain to the students that now, another feature functionality is being added to the application, that is, the search box.

Search Box Feature:

Show: Show the image to the students to identify the search box on the landing page.



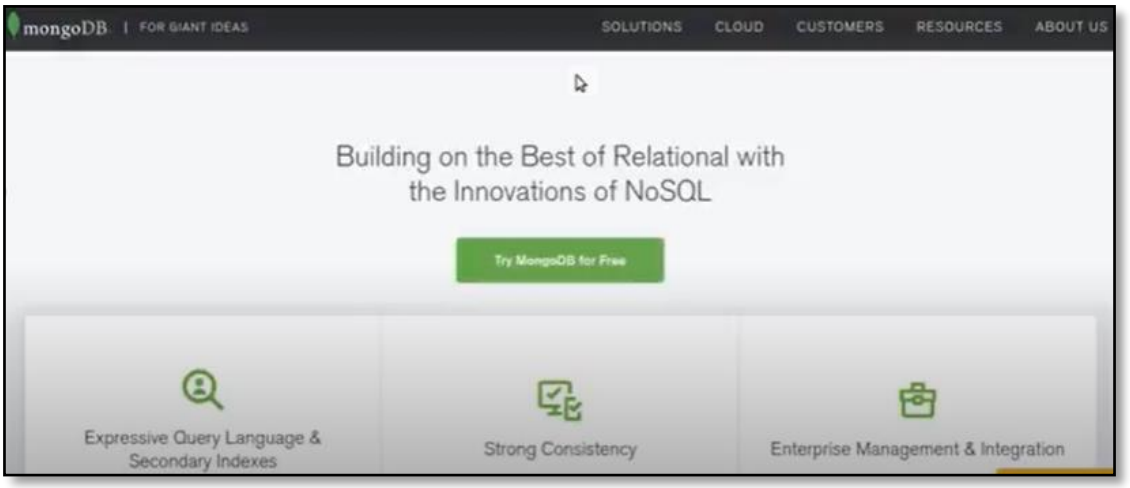
For this, in the *contacts.ejs* file, a keyup event handler is created for the search input box.JQuery may be used to select all the contact records with class contact-card and store in the rows variable. Now, the text entered in the search box is trimmed and converted to lowercase. This value is then compared with every contacts full name and consequently the results are filtered. The contact name matching the input text id displayed and the rest are hidden.

5.6 Preparing MongoDb

CLASS DURATION: 14 MINUTES

VIDEO DURATION: 4 MINUTES

Show: Play Preparing MongoDB: Session 5 Video – 1.6.



Discussion: In this video, students will learn how to install and configure MongoDB. They will understand the use of MongoDB command line interface and how to create collections for future uses in the application.

MongoDB:

It is a NoSQL database. It provides strong consistency, flexibility, scalability, and performance. It is used as a cross-platform document oriented database, that is, it stores JSON in the form of document in dynamic schemas. It is quite different to the relational database MySQL.

Show: Play the video from 1:08 to 2:00.

Content: Explain to the students that in this step MongoDB is installed in the Ubuntu system.

Installation:

- Open the terminal and type the command `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10` to import the public key of the server.
- Change the package repository URL to the aptitude source list. The aptitude source list is then refreshed.
- The MongoDB is installed using the command `sudo apt-get install -y mongodb-org`.
- Download the package and install.
- MongoDB comes with a command line tool which can be started by simply typing the command `mongo`.

Show: Play the video from 2:06 to 2:40.

Content: Explain to the students that in this step a database called *packtdb* is created by executing the command `use packtdb`. Existing databases can be viewed by the command `show dbs`, but unless a collection is created in a database, the database name is not displayed. The current database can be displayed by using the command `db`. A collection called `contactPerson` is then created.

Show: Play the video from 2:42 to 3:40.

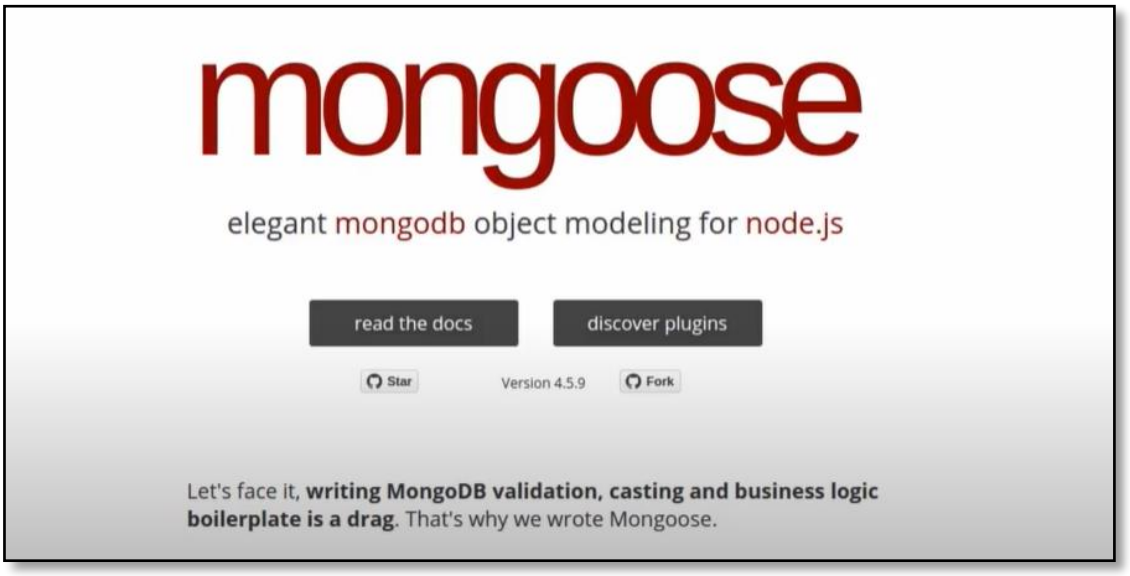
Content: Tell the students that a record is inserted in the collection using the `insert()` method. The data inserted is displayed in JSON format. Records of all contacts is then displayed using the `find()` method. A record is also deleted by passing the `contactId` of the particular contact to be deleted.

5.7 Using Mongoose

CLASS DURATION: 15 MINUTES

VIDEO DURATION: 5 MINUTES

Show: Play Using Mongoose: Session 5 Video – 1.7.



Discussion: In this video, students will learn how to download and install the mongoose library for Node.js. Mongoose will be used to perform basic queries on the MongoDB and also to migrate the records from MySQL database to the MongoDB collection.

Show: Play the video from 0:50 to 0:58.

Content: Explain to the students that firstly, a new project folder *7.2* is created in the terminal and its path is changed to this corresponding folder. The contents of the project are copied using `cp -Rp /6.5/*`. The project is then opened in Sublime Text Editor.

Mongoose:

Connection of a Node Application to the MongoDB collection can be done by a connector library such as mongoose. Instead of using native MongoDB database calls, mongoose packages those APIS into elegant object models. It provides a simple schema based solution to model the data.

Additional Reference:

Refer to the official site of mongoose to explore more: https://mongoosejs.com/

Show: Play the video from 1:30 to 2:47.

Content: Explain to the students that mongoose is installed in the node project using the command `npm install mongoose`. In the sublime text editor, the project is opened and a new file called *mysql-mongoose-conversion.js* is created.

In that file, mongoose is imported and a connection is made to the MongoDB server. An error message is displayed if any error is encountered. A *contactPersonSchema* is made using the new operator. The properties of the schema are the same such as `contactId`, `first name`, `lastname`, and so on.

Show: Play the video from 2:50 to 3:31.

Content: Explain to the students that in this step the schema is compiled into a model. A model is a class with which documents can be constructed. In this case, each record or document is a contact with properties as declared previously.

Show: Play the video from 3:40 to 5:23.

Content: Explain to the students that MySQL must be imported now to migrate the data from the MySQL database to the MongoDB collection. Then, a connection is created and set up to the MySQL server. A function `readMySQLContactPerson` is created which basically returns all the contacts records from the `contacts_person` table. These results are saved to the MongoDB by using the function `saveMongoDBContactPerson` using the FOR loop to iterate over the rows returned. Finally, the `saveMongoDBContactPerson` function is implemented and a document *newContact* is created according to the contact model. Finally, the contact passed is saved to MongoDB in this function. The codes are tested and the results are displayed as expected.

Tips: Database manipulations are common when JS Strings or concatenations are frequently used. Thus, the information is vulnerable to SQL injection attacks and invalidated. Using an in-built security library such as mongoose prevents this.

5.8 Implementing Auto-Increment Counter

CLASS DURATION: 15 MINUTES

VIDEO DURATION: 5 MINUTES

Show: Play Implementing Auto-Increment Counter: Session 5 Video – 1.8.

Discussion: In this video, students will learn how to create a counter schema to generate the auto-increment number sequence. Students will use a pre hook in mongoose and modify the data model.

Show: Play the video from 0:37 to 0:45.

Content: Explain to the students that firstly, a new project folder *7.3* is created in the terminal and its path is changed to this corresponding folder. The contents of the project are copied using `cp -Rp /7.2/*`. The project is then opened in Sublime Text Editor.

Show: Play the video from 0:46 to 3:17.

Why is Auto-Increment Sequence required?

MongoDB does not have any auto-increment sequence available unlike MySQL. Thus, a collection must be created in MongoDB to generate an auto-increment sequence each time a new record is created. MongoDB provides an `objectId` to a new record, but that is not an auto-incremented number.

How is an Auto-Increment Sequence Generated in MongoDB?

First Part:

The terminal is started and the mongo shell is entered by typing the command `mongo`. The database `packtdb` is used and a new collection called counters is created. This collection has two attributes only, namely, `_id` and `sequence_value`. The `id` property of the contact records already holds the auto-incremented numbers in MySQL. So, as the records were migrated from

MySQL to MongoDB, a record is inserted in the counters collection which holds the next value of the `id` of a contact.

Second Part:

A new file *contact-model-mongo.js* is created in the node project. The codes from the file *mysql-mongo-conversion.js* are copied and pasted in this file. Now, a new model called *counters* is created with the properties `_id` and `sequence_value` as discussed. The `_id` property is a required field and the `sequence_value` is a number field with a default value 0. Then, a serial pre-hook is used to find and update the sequence in the contact.

Show: Play the video from 3:18 to 5:20.

Content: Explain to the students that in this step modification of the data model is being done in the *contact-model-mongo.js* file. Instead of the contact person model used by the other programs, the same approach is followed as the MySQL data model to encapsulate the MongoDB specific operations in an object. Thus, the codes are copied and pasted from *contact-model-mysql.js*. A `ContactsMongo` function is created which contains the same operations such as retrieval, insertion, updation, and deletion. Explain to the students the minor changes made in the methods. Finally, as the records were migrated from the MySQL database, the `sequence_value` property is reset. The `contactId` of the last record inserted is retrieved using the `find()` method and then, the `sequence_value` is updated to that number plus one to start the next record with that number.

Industry Best Practice: Always use const instead of let. Do not use var.

In-Class Question: What data types are supported as values in documents by MongoDB?

Answer: String, Integer, Arrays, Timestamp, and so on.

5.9 Creating Version 2 REST APIS

CLASS DURATION: 15 MINUTES

VIDEO DURATION: 4 MINUTES

Show: Play Creating Version 2 REST APIS: Session 5 Video – 1.9.

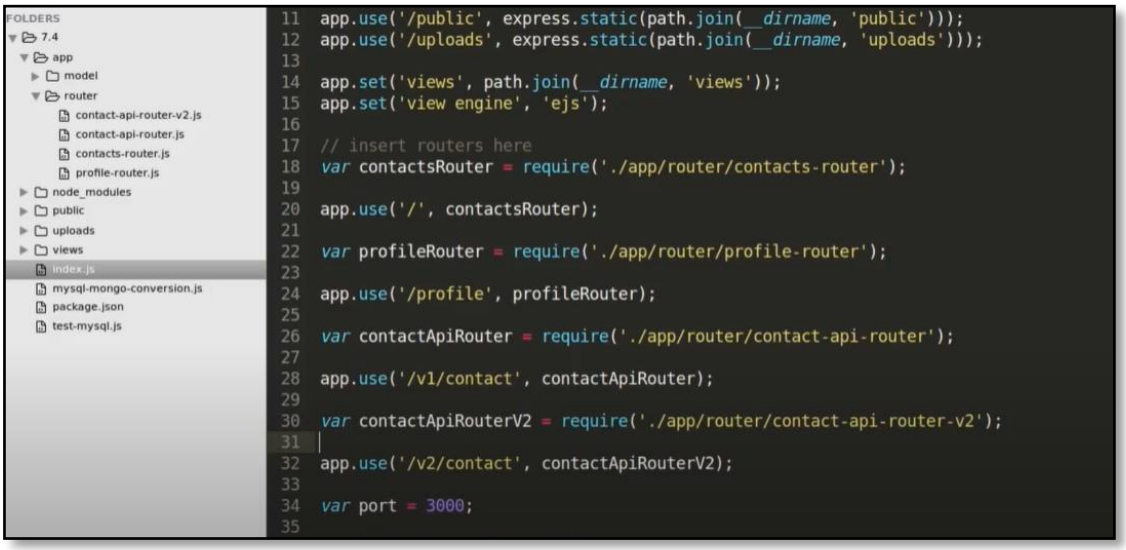
Discussion: In this video, students will learn how to create a second version of the REST APIS with MongoDB as the backend server and modify the server endpoints to use these APIS. Finally, the application will go through an integration testing.

Show: Play the video from 0:41 to 0:50.

Content: Explain to the students that firstly, a new project folder *7.4* is created in the terminal and its path is changed to this corresponding folder. The contents of the project are copied using `cp -Rp /7.3/*`. The project is then opened in Sublime Text Editor.

Show: Play the video from 0:54 to 2:31.

Content: Explain to the students that a second version of APIS is created here. A file called *contact-api-router-v2.js* is created. All contents of the file *contact-api-router.js* are copied to this file. The only modification required done is that the file imported is changed from *contact-model-mysql* to *contact-model-mongo*. In *index.js*, the version 2 router is hooked upto the endpoint `/v2/contact`. The APIS are tested in the terminal using `curl`. It is seen that all the tests have passed.



Show: Play the video from 2:35 to 4:00.

Content: Explain to the students that now, the endpoints in the ejs template are to be changed. In the ajax call of record deletion in the *contacts.ejs* file, the url is modified to v2. The urls of the ajax calls of the *profile.ejs* file are also modified similarly. The endpoints in the *contacts-router.js* and *profile-router.js* are modified in the same way. The application is now tested thoroughly in the browser.

Conclusion: Give the summary of all the topics that is covered in this session.

QUERIES REGARDING SESSION

CLASS DURATION: 10 MINUTES

FACILITATION GUIDELINES

Ask students if they have any queries regarding the session and resolve the queries.

SESSION 6: WORKING WITH MONGODB

Greeting: Welcome students back to the course, greet them, and ask if they are ready to get started with the next session of the course. Explain to them that this session will carry forward from the previous session and will continue explaining session 6.

Quick Recap

Recap: Give a quick recap of the topics covered in the last session and ask students if they have any doubts. Resolve their doubts and then, inform them that this session will be a long one because multiple topics have to be covered.

Start the session.

6.1 Using MongoDb

CLASS DURATION: 20 MINUTES
VIDEO DURATION: 12:55 MINUTES
FACILITATION GUIDELINES -
Show: Play Using MongoDB: Session 6 video 6.1.

Discussion: MongoDB is a popular database solution that is specifically used for Node-based projects.

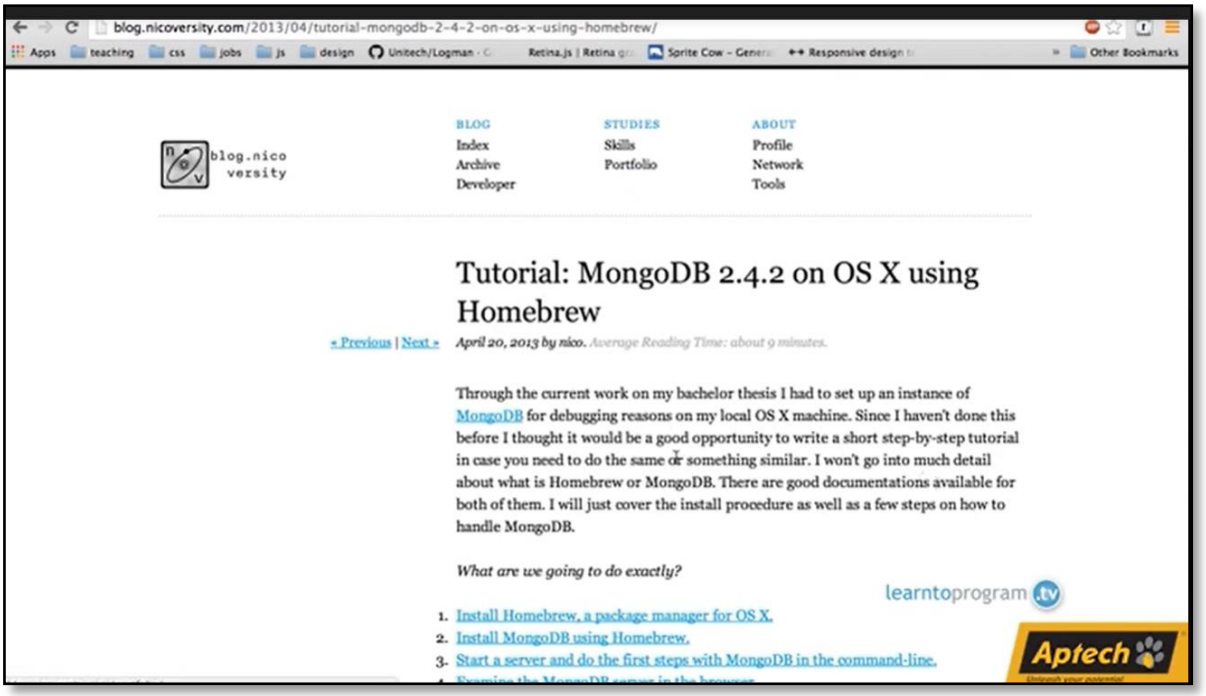
Following are some benefits that MongoDB provides over other solutions:

- It is easy to understand
- It is flexible and easy to adapt
- Queries in Mongo are based in JavaScript

Note: Although, It is possible to write queries in MySQL, but with Node, Mongo aligns better.

Installing MongoDB:

Show: Play the video from 0:25 to 1:36.

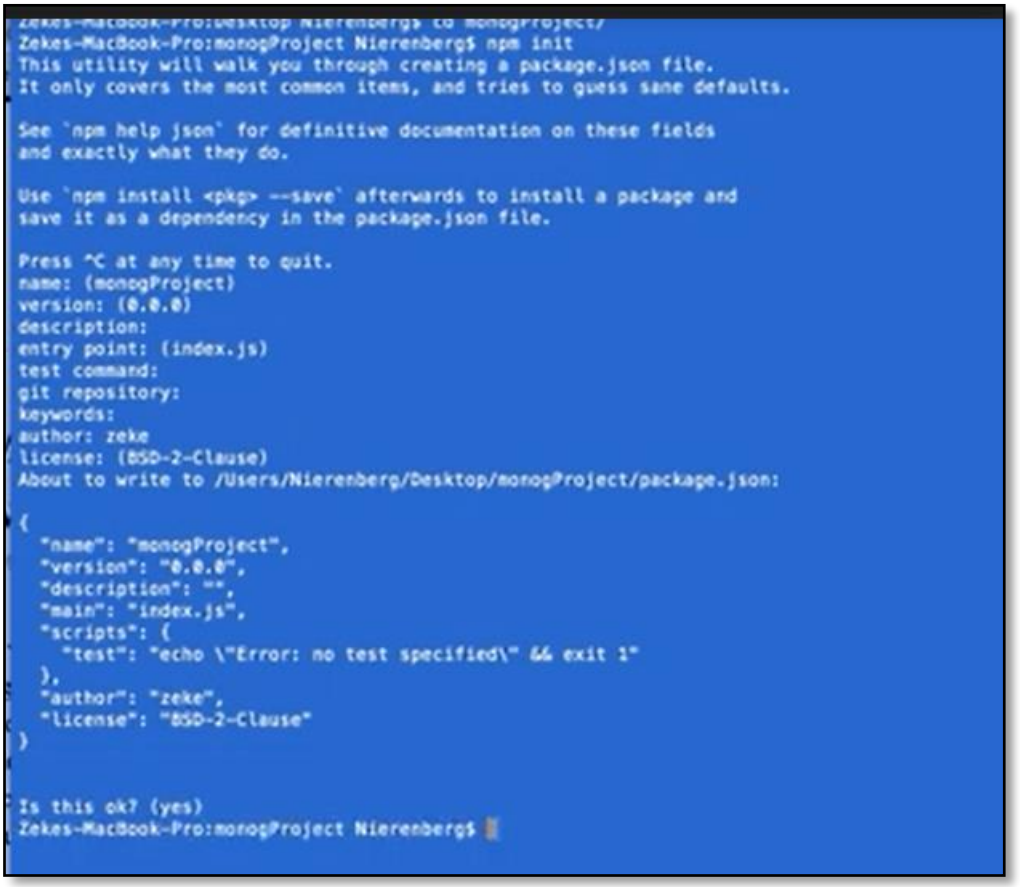


Content: Installing MongoDB can be difficult. The guide on Homebrew assists you and eases the process of installation MongoDB. The URL to access Homebrew guide is provided in the reference links section. These instructions are also available in the Notes section of this TG.

Creating a New Project in MongoDB

Show: Play the video from 1:47 to 3.34.

To begin, create a new project in MongoDB by configuring a *package.json* file (refer to previous sessions on *package.json* configuration).



Open *package.json* and find some dependencies, primarily, MongoDB node driver instead of MongoDB software.

After saving *index.js*, connect *index.js* to the database, insert a document in a collection (row in a table for other databases), and query it.

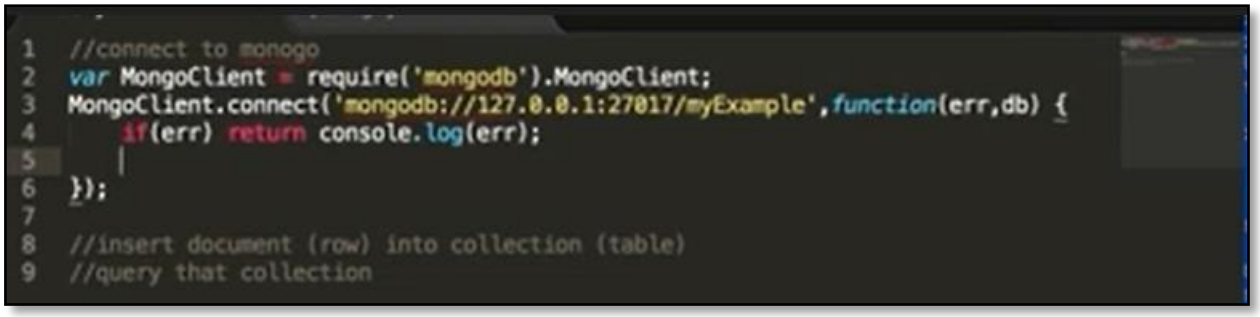
Show: Play the video from 3.37 to 5.46.

First, get MongoClient using MongoDB protocol.

Now, the user can run the MongoClient on LocalHost and connect it to MongoDB. It is possible to specify any database on the server by adding '/' and the name of the database.

Note: Mongo facilitates easy creation of any database and table at any time as required, unlike MySQL *in which, the database has to be created in advance.*

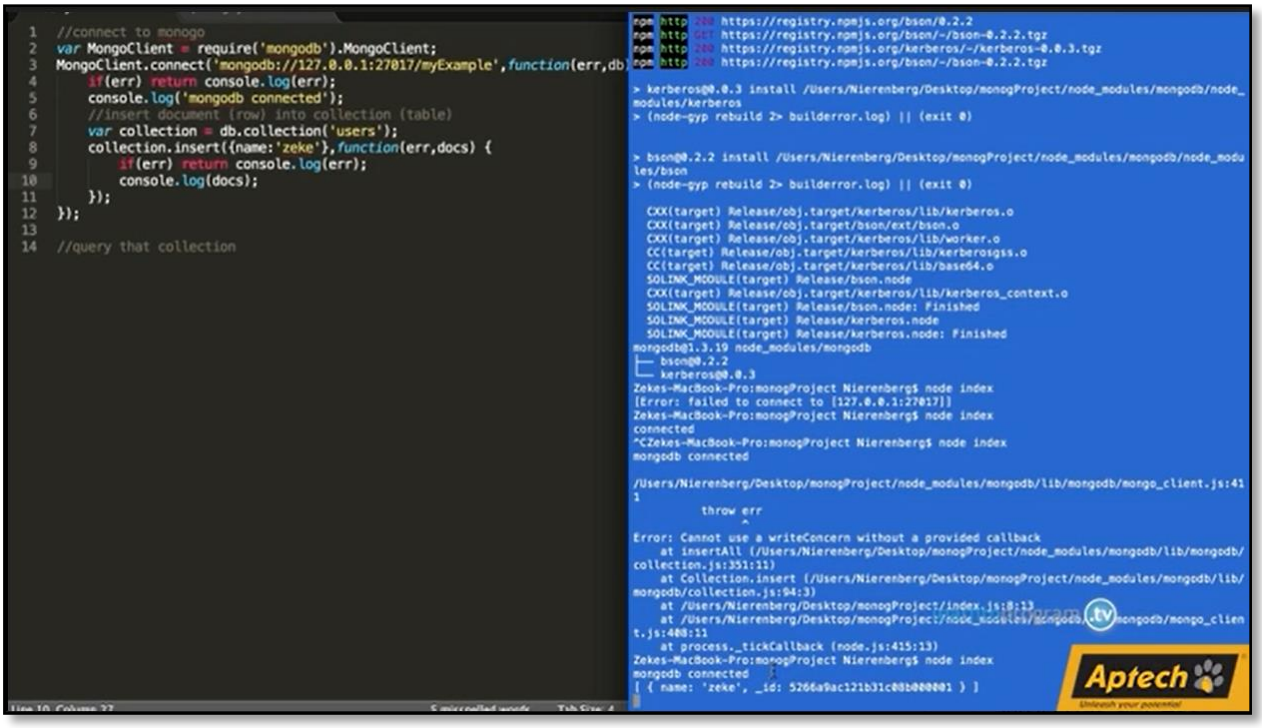
- The `callback` function mentioned in the following image takes two arguments namely, error and database. If there is no error, connect will not pass any argument. If there is an error, entering the `return` function will stop the error and everything in the function.



Show: Play the video from 5.48 to 9.30.

Explain the students how to get the MongoDB collection table and to create collections.

For example, create a collection of 'Users' and insert details, such as a name.



- Now, test this collection and add **npm install** and see the output.
- This results in failing the connection. To establish the connection, initiate the Mongo process.
- As a next step, type **MongoDB** in a new terminal window.
- When MongoDB is installed correctly, it will log the file and connection will succeed.
- Now, insert a `callback` function.
- In case of error, request to return it and not log the docs.
- Start again to ensure the process of insertion of a document is complete. After inserting the document, a different id gets created.

This id embeds a timestamp and denotes when the document was created.

Note: The MongoIds that are generated are called objectids.

Show: Play the video from 9.32 to 12.55.

To query the collection, write `collection.find`. Here, `find` can take any object. Depending on the object entered, records (if any are present) will be shown.

Now, using the `toarray` function and another callback method, look for any errors and users/results.

As the final step, ask the students to run the collection once again, using the following code.

Explain the output to the students.

This concludes the basic introduction of MongoDB.

In-Class Question: How do you create collection in MongoDB?

Answer: MongoDB creates collections when data is first stored, but if there is a requirement for customized collections, such as setting the document validation rules, MongoDB provides the `dbcreateCollection()` that enables the creation of collections.

Reference links:

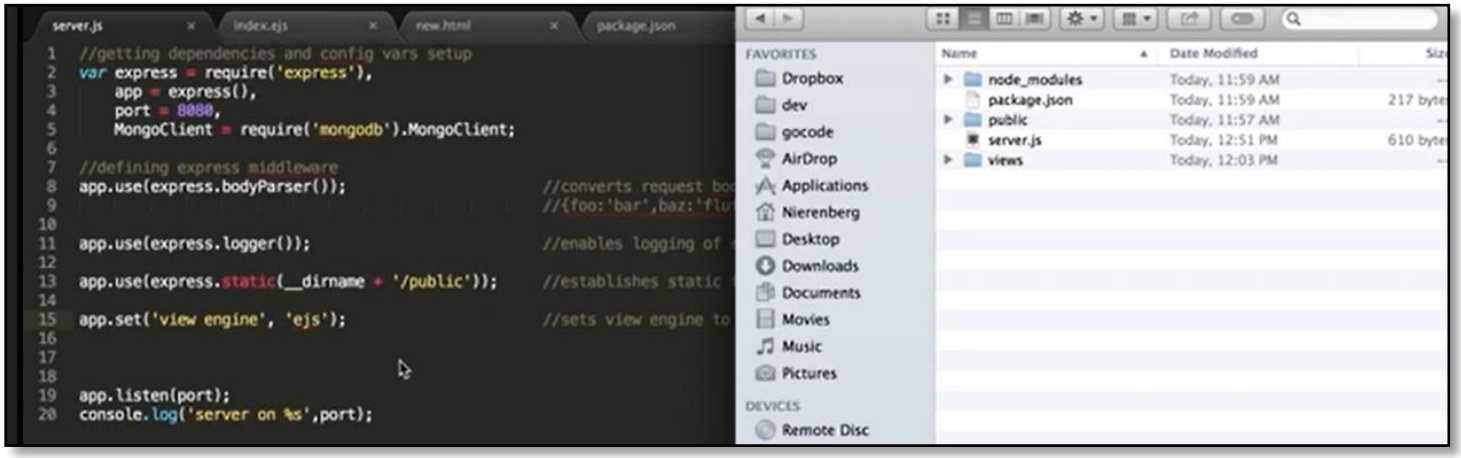
Refer to following links for more information:

https://docs.mongodb.com/manual/core/databases-and-collections/
https://docs.mongodb.com/manual/core/databases-and-collections/

https://switch.homebrew.guide/

6.2 Express App Interfacing With MongoDb

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 17:17 MINUTES
FACILITATION GUIDELINES -
Show: Play Express App Interfacing with MongoDB: Session 6 Video 6.2.



Discussion: The previous video covered the basics of MongoDB. This video talks about the Express app that interfaces with MongoDB.

Show: Play the video from 0:29 to 2:30.

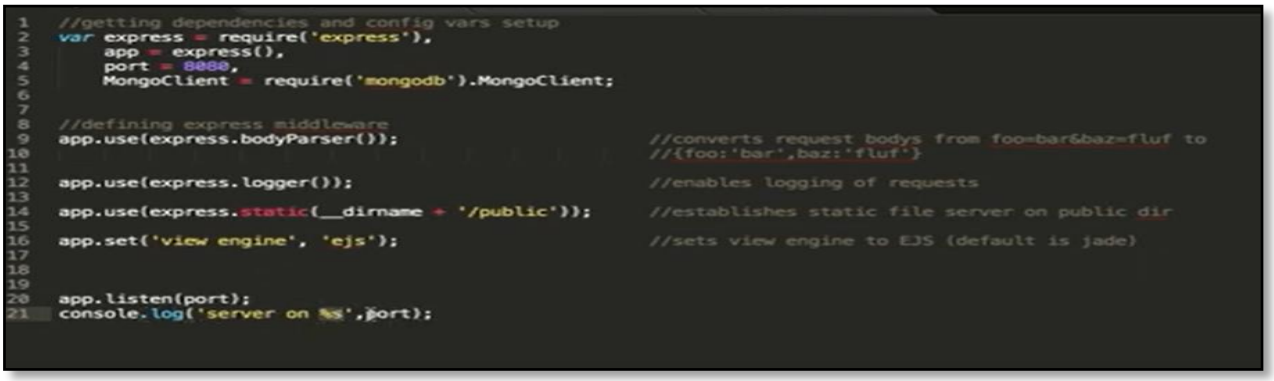
Content: MongoDB interfaces with Express app to store content.

Prerequisites:

a. Creation of *package.json* file that defines dependencies, such as MongoDB, Express, and ejs.
b. Running **npm.install** in the terminal that installs dependencies in a *node_modules* folder.
c. Creation of a server file, server.js in which several dependencies were defined, such as Express, instantiating app, port 8080, and Mongoclient.

```
//getting dependencies and config vars setup
var express = require('express'),
    app = express(),
    port = 8080,
    MongoClient = require('mongodb').MongoClient;
```

d. Defining Express Middleware.
e. BodyParser that converts querystrings in URL format to JavaScript objects.
f. Loggers to which requests are logged.
g. Setting a public static-file server.
h. Setting up ejs as a new engine that will enable rendering of views as ejs instead of Jade.
i. Other conditions as mentioned in the following image.



```
//getting dependencies and config vars setup
var express = require('express'),
    app = express(),
    port = 8080,
    MongoClient = require('mongodb').MongoClient;


//defining express middleware
app.use(express.bodyParser());                  //converts request bodys from foo=bar&baz=fluf to
                                                //{foo:'bar',baz:'fluf'}

app.use(express.logger());                      //enables logging of requests

app.use(express.static(__dirname + '/public')); //establishes static file server on public dir

app.set('view engine', 'ejs');                  //sets view engine to EJS (default is jade)


app.listen(port);
console.log('server on %s',port);
```

Show: Play the video from 2:30 to 6.40.

To begin, run the server on 8080.

Note: The Express app is a list of members in a group.

How to interface Express with MongoDB?

Make an HTML form; this form is just a member name. Now, create a view that lists all the members. Create an Express mongo code to populates the list and submit the form. First, define collections in the form of members.

Note: Mongo does not require any setup.

Create *get* and *Post* routes and set up members in that to add a new member and to list all members. Request the following functions:

```
var index = function(req,res) {};
var addMember = function(req,res) {};

app.get('/members',index);          //list all members
app.post('/members',addMember)      //add a new member
```

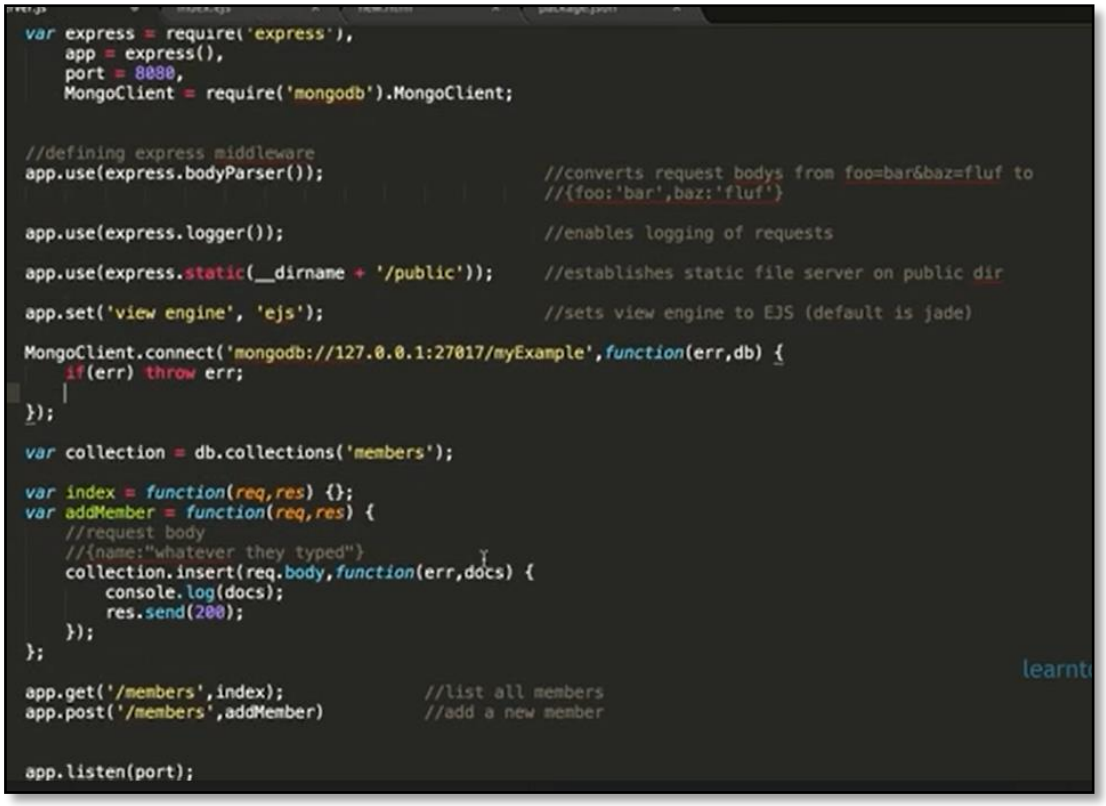
Adding a member provides member information from the new file in the request body. The goal is to insert the request body in the collection named members.

Show: Play the video from 6.42 to 9.30.

Define the collection as follows:

Run `collection.insert` and first, insert the search object, `req.body`. Next, insert a callback with the functions `err` and the actual inserted documents. After insertion, type `console.log(docs)` and respond to the client that program is working correctly. Run Connect on Mongo.

Next, insert callback for err and db. If there are errors, discard the callback.



```
var express = require('express'),
    app = express(),
    port = 8080,
    MongoClient = require('mongodb').MongoClient;


//defining express middleware
app.use(express.bodyParser());                    //converts request bodys from foo=bar&baz=fluf to
                                                  //{foo:'bar',baz:'fluf'}

app.use(express.logger());                        //enables logging of requests

app.use(express.static(__dirname + '/public'));   //establishes static file server on public dir

app.set('view engine', 'ejs');                    //sets view engine to EJS (default is jade)

MongoClient.connect('mongodb://127.0.0.1:27017/myExample',function(err,db) {
    if(err) throw err;

});

var collection = db.collections('members');

var index = function(req,res) {};
var addMember = function(req,res) {
    //request body
    //{name:"whatever they typed"}
    collection.insert(req.body,function(err,docs) {
        console.log(docs);
        res.send(200);
    });
};

app.get('/members',index);              //list all members
app.post('/members',addMember)          //add a new member


app.listen(port);
```

Show: Play the video from 9.33 to 15.30.

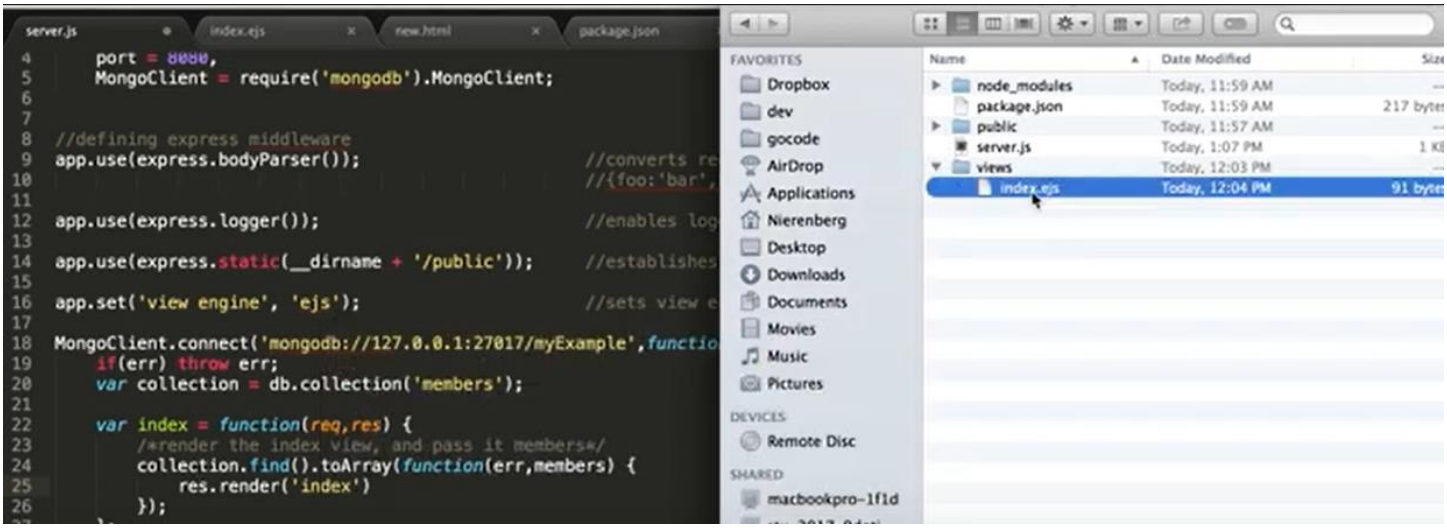
After the database is connected, explain the students that they can process the application request. Save it and restart the server. The server does not connect due to uncaught exceptions. To resolve this, catch the uncaught exception. The server is on 8080 now. The post members will run the following code:

```
MongoClient.connect('mongodb://127.0.0.1:27017/myExample',function(err,db) {
    if(err) throw err;
    var collection = db.collection('members');

    var index = function(req,res) {};
    var addMember = function(req,res) {
        //request body
        //{name:"whatever they typed"}
        collection.insert(req.body,function(err,docs) {
            console.log(docs);
            res.send(200);
        });
```

A new file will post to members and now, browse to *LocalHost:8080/new.html* and try logging in. In the Console, it not only logs in requests, but also documents with a mongoID.

Next, ask students to make the `index` function. The goal of the `index` function is to run the view that was made.

Run the MongoDB query by adding a passage query. Render index with Express and notice it in the Views folder.



The index wants to recall members. Now, if the user restarts the server on localhost, he/she will get the desired output. Once the 'OK' output is seen, add to the flow by redirecting to the members.

Show: Play the video from 15.39 to 17.17.

Restarting the server will show list of members defined. Add more entities and get organized; it is possible to move the members' functions in a members file. A user can also move Middleware to another file and render the server lightweight.

In-Class Questions: What is MongoDB Express?

Answer: Express JS server uses the MongoDB driver to access MongoDB because MongoDB is the basic driver for Node JS applications.

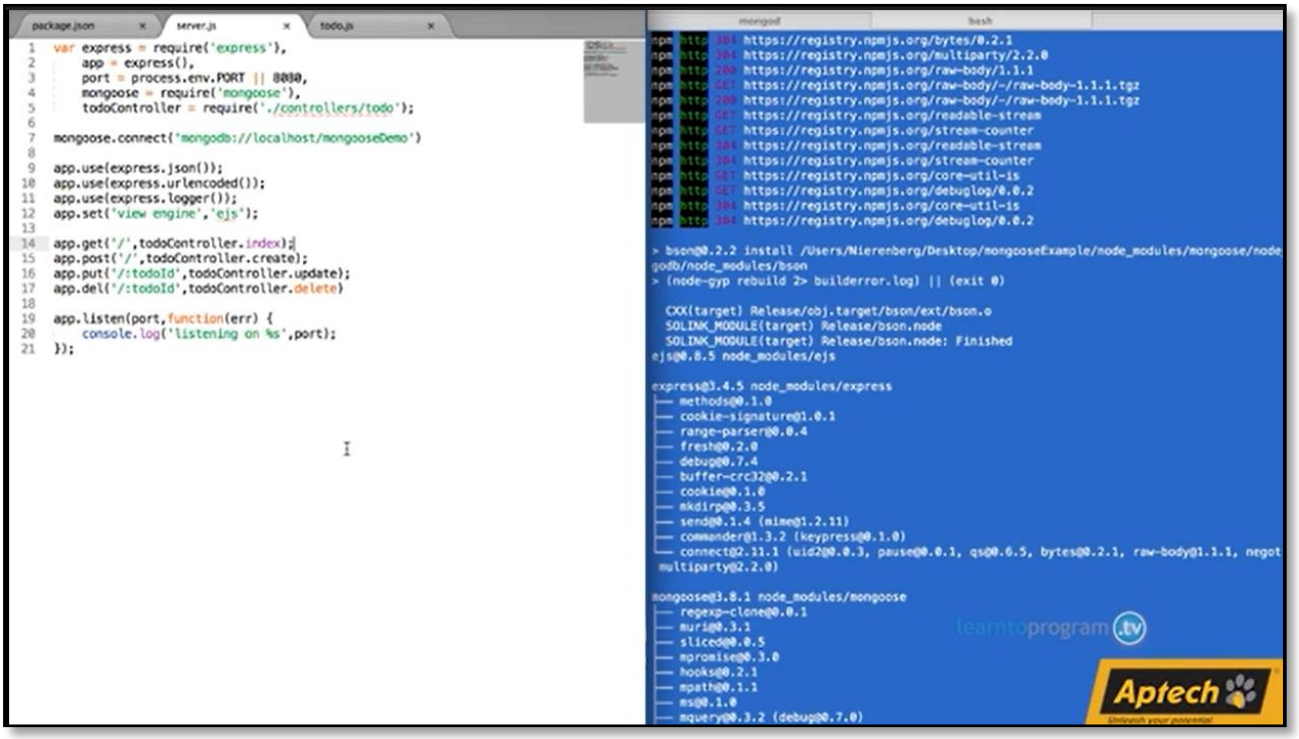
Additional Reference:

Refer to following link for more information:

https://www.mongodb.com/languages/express-mongodb-rest-api-tutorial

6.3 MongoDb's ORM, Mongoose

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 20:47 MINUTES
FACILITATION GUIDELINES
Show: Play MongoDB's ORM, Mongoose: Session 6 video 6.3.



Discussion: In this video, you will learn to use the Mongoose ORM application for MongoDB.

Content: The Mongoose application provides support to MongoDB, controls it, and simplifies the process of working with MongoDB.

Setting up Mongoose ORM

Show: Play the video from 1:15 to 5:44.

Start with a *package.json* file that defines dependencies. Install **npm.install**. This will set up the Express app that will build Mongoose.

Ask students to familiarize themselves with controllers, models, and views.

Todos is the app that stores todos items in MongoDB through Mongoose. The controller is made for Todos. Todos can create, update, index, and delete controllers. Express routes are wired to them.

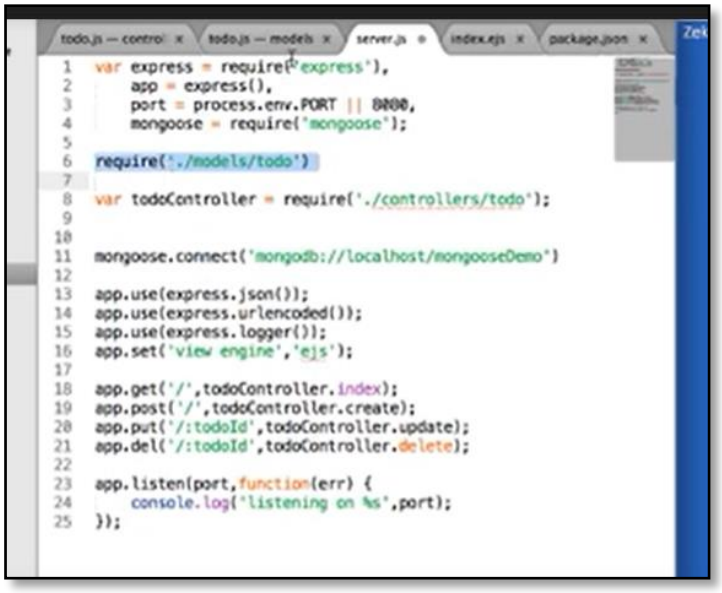
When the user goes to a host, it goes to the todos controller and runs a series of functions mentioned. These functions are run corresponding to requests made in server.js.

For example, in response to a 'put' request made, the update function is run.

This is a part of setting up the skeleton for todo.js. In the models, first, pull out the Schema variable.

a. Define the data that is required; add details, such as create data and name.
b. Add mongoose.model and attach a model with this schema; a model will be set up globally.

Back in the server, this is how requirements will look like.



This model only defines the Mongoose model. The model is always placed before the controller as the controller gets todos through Mongoose.

Show: Play the video from 5:55 to 13:37.

Following steps make controllers interactive with models and send responses:

a. Render non-existent data in the index.js file and run it.
b. Now, save and run the server.

Note: MongoDB is not run here.

c. 'Listening on 8080' is displayed.

The data gets rendered.

d. Next, make the form that posts to the root work.
e. First, the user has to check whether the name is in the request button.

When the name is in the req.body and req.body is not valid, then, 'next' will trigger another function.

f. A callback function is triggered.

Notice the `return` function that will stop the function if any error case persists.

g. The 'next (err)' refers to the next function that handles requests. If you pass parameter one to the next function and start Express, it understands that this might be an error.

Note: Express handles errors in a special way.

h. Here, no error is present and the same is validated. For now, send back the json and save it.

In the output, it sends back to json as expected.

This output consists of a MongoDB id and the document tracking versions.

Redirect the root instead of sending json.

i. Get the functions `err` and todos, get the run, and check again for the error.
j. Restart the server and it works properly.

Show: Play the video from 13:45 to 18:16.

Update function: Next, perform the `update` function. In server.js, place the put request for redirectory/'id.' The request body specifies the changes. Next, load the created id and add the callback function. Check the error condition and the id. Call Next. To utilize the Next functionality, pass the loaded thing to the next function.

Note: The next function will ensure the code will be organized even during asynchronous callbacks.

Notice the request body and for each change, update the todo. Each object in JavaScript is looked through the `for(key)` loop. Refer to the following example:

```
];

controller.create = [
    function(req,res,next) {
        if("name" in req.body && req.body.name !== '') {
            next();
        } else {
            res.send(400);
        }
        //function to validate that the todo isn't empty
    },
    function(req,res,next) {
        //function to create the todo, send back the json
        Todo.create(req.body,function(err,todo) {
            if(err) return next(err);
            res.redirect("/");
        });
    }
];

controller.update = [
    function(req,res,next) {
        //load todo in question
        Todo.findById(req.params['todoId'])
```

Notice this ajax request that makes the put request for data and URL. Now, input the information and notice the parameter for the request body, which is the name.

Note: Some libraries such as _1dash can do this in a one-liner.

Save the callback. Refresh the output and it displays the 500 error. Now, go back and correct the error where 'param' was written as 'params.' Confirm back in the Express log. Notice the error is rectified now. This completes the demonstration.

In-Class Question: What is Mongoose?

Answer: It is an Object Data Modeling (ODM) library based on Node.js for MongoDB.

It enables developers to enforce particular schemas in the application layers.

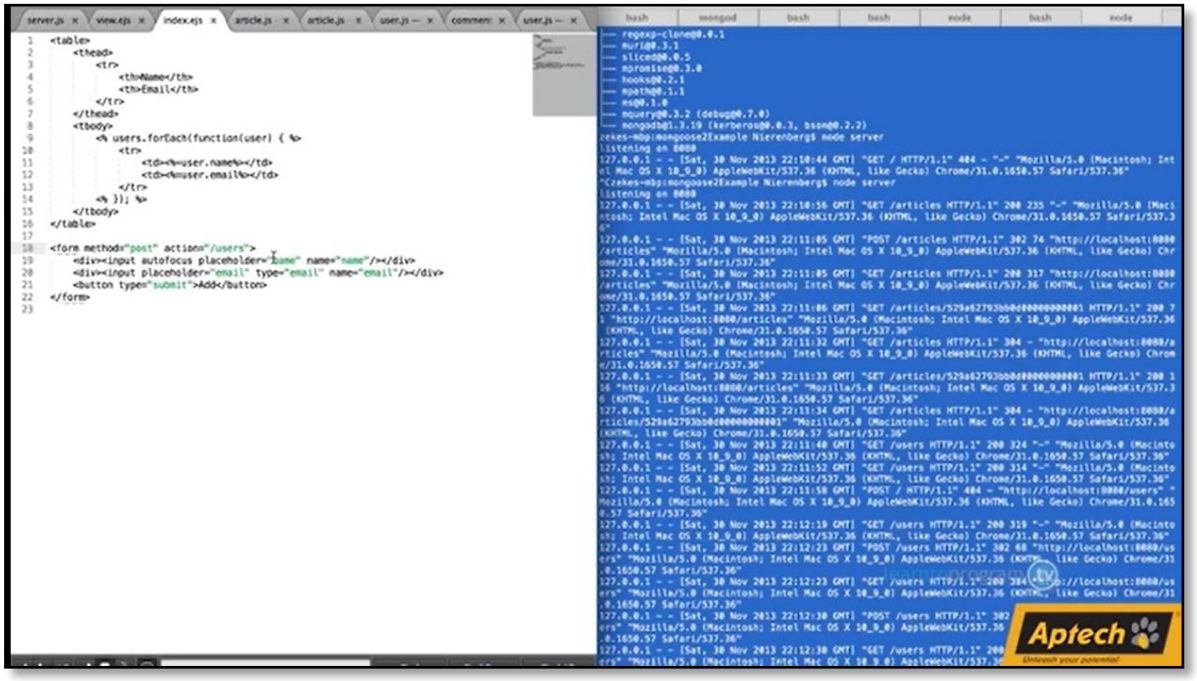
Additional Reference:

Refer to following link for more information:

https://www.mongodb.com/developer/article/mongoose-versus-nodejs-driver/

6.4 Subdocuments and References

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 28:05 MINUTES
FACILITATION GUIDELINES
Show: Play Subdocuments and References: Session 6 video 6.4.



Discussion: In this video, you will learn some tricks on working with Mongoose.

Show: Play the video from 0:35 to 1:36.

Project Walkthrough:

Show: Play the video from 1:47 to 4:02.

Content: For a standard node project, open the *package.json* file. There will be a server.js file. The user must set up Express and get the Mongoose models, users, and articles.

Note: The models define the schema as an attachment to the models.

For example, this article.js image shows all the things that an article can have.

Some controllers lie between the routes, views, and models.

Connect to the database and set up Express. There will be several articles and you can look into each of them and update them.

It is also possible to create, update, and delete 'users.'

Show: Play the video from 4:04 to 5:40.

Describe Express before going further. Instruct Express to run a function whenever it locates something with article id. This way, parameters such as articleId can be utilized for productive tasks. The function of articleId is a load function in the article controller. It requests, responds, and expresses the next function.

This loads the article and finds your id, even if there are multiple ones. This id, then, attach to a request that will be passed to the next function. This way, the article is ready for use in the next request.

Note: This will run before all other controllers.

Show: Play the video from 5:46 to 14.29.

Associating Users and Articles

Navigate to the article model and assign it to a user as the creator. The user is of the type 'Objectid'; Type 'schema' as mentioned.

```
1  var mongoose = require('mongoose');
2  var Schema = mongoose.Schema;
3
4  var ArticleSchema = new Schema({
5      title:String,
6      body:String,
7      createdAt:{type:Date,default:Date.now},
8      user:{type:Schema.ObjectId}
9  });
10
11 mongoose.model('article',ArticleSchema);
```

Browse for 'objectid type' and open the first link that appears. Notice the 'schematypes.objectid.'
Give the 'ref' name. In this case, it is 'user.'

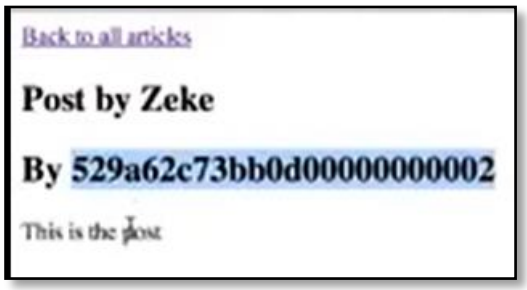
The reference now exists in the database. Select one article from the list of all articles. For the same, specify the user.

- As a first step, navigate to Views → index.ejs.
- Now, in the index, locate the articles but not the users. For the same, type the following code.

```
var mongoose = require('mongoose'),
    Article = mongoose.model('article'),
    User = mongoose.model('user'),
    controller = {};

controller.load = function(req,res,next,id) {
    Article.findById(id,function(err,article) {
        if(err) return next(err);
        if(!article) return res.send(404);
        req.article = article;
        next();
    });
}

controller.index = [
    function(req,res,next) {
        //find all articles, render the index.ejs view
        Article.find({},function(err,articles) {
            if(err) return next(err);
            res.render('article/index',{articles:articles});
        });
    }
];

controller.view = [
    function(req,res,next) {
        res.render('article/view',{article:req.article});
    }
];

controller.create = [
    //validate article
    function(req,res,next) {
        if("title" in req.body && req.body.name !== '') {
            next();
        } else {
            res.send(400);
        }
    },
    function(req,res,next) {
        //function to create the article
        Article.create(req.body,function(err) {
            if(err) return next(err);
            res.redirect("/articles");
        });
```

Refresh the host and check the output.

The MongoIDs are the values. Notice that the output does not contain the name as desired. Instead, it contains output as shown.



This id is the association of the database and the reference.

Convert this id to a user object. After that navigate to article control → view and change the way to load articles and also to load a user. Now, populate the user and save the file.

In view.ejs, change the name to user.name. Restart the server to see the correct name.

Recap:

Following was covered so far:

- Referenced a foreign document type in a model and referenced one model in another.
- Loaded the foreign document with populate in the controller.

Show: Play the video from 15.00 to 19.00.

Sub documents:

In a relational database, define a new schema in the same article file first if it requires articles with multiple comments. In the comment controller, the article id in a post is already wired

up. When a comment is posted, the comment controller creates a node work article because it has been through articlecontroller.load. Browser submits the form that goes to a post request.

Following is the sequence of calling:

1. articlecontroller.load
2. next
3. commentcontroller.create.

Add the push array prototype, which is a Mongoose subdocument array type.

Show: Play the video from 19.03 to 27.00.

Add Comment Schema:

The subdocument will get a MongoDB id of its own. Push following req body:

```
controller.create = [
    function(req,res,next) {
        //push a new comment into the loaded article
        req.article.comments.push(req.body);
```

Save it and add the callback error and redirect the article. Now, delete and filter out the comments array.

Note: This page does not show comments and cannot give comments. Both can be done separately.

1. 'Loading fails' will be displayed after restarting the server.
2. Try the following code to avoid 'loading fails' error.

```
<hr>
<h2>Comments</h2>
<% article.comments.forEach(function(comment) { %>
    <div>
```

3. To load the comment, look through them and fill out a comment form.
4. Now, restart the server to display the comments corresponding to the user.
5. Apart from comment loading, another problem is populating a foreign object on a subdocument. This can be done as shown.

```
var loadAllUsers = function(req,res,next) {
    User.find({},function(err,users) {
        if(err) return next(err);
        req.users = users;
        next();
```

6. Now restart the server and notice the subdocument getting populated successfully.

Summary:

This module covered following topics.

- How to store relationships and not just a single document and
- How to build sophisticated sub-documents

This completes the video.

In-Class Question: How is a Mongoose model created?

Answer: Mongoose model is a wrapper of its schema. To create a model, the first step is to reference Mongoose. When you connect to the database, the reference that returns is the one used here too. Next, define the schema where the key name corresponds to the property name. Finally, export the model and call the model constructor and pass it the collection name and reference.

Additional Reference:

Refer to following link for more information:

https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

Conclusion: Give the summary of all the topics that have been covered in this session.

QUERIES REGARDING SESSION

CLASS DURATION: 10 MINUTES

FACILITATION GUIDELINES

Ask students if they have any queries regarding the session and resolve the queries.

SESSION 7: AUTHENTICATION AND SECURITY

Greeting: Welcome students back to the course, greet them, and ask if they are ready to get started with the next session of the course. Tell them that this session will carry forward from the previous session and will continue explaining further lesson.

Quick Recap

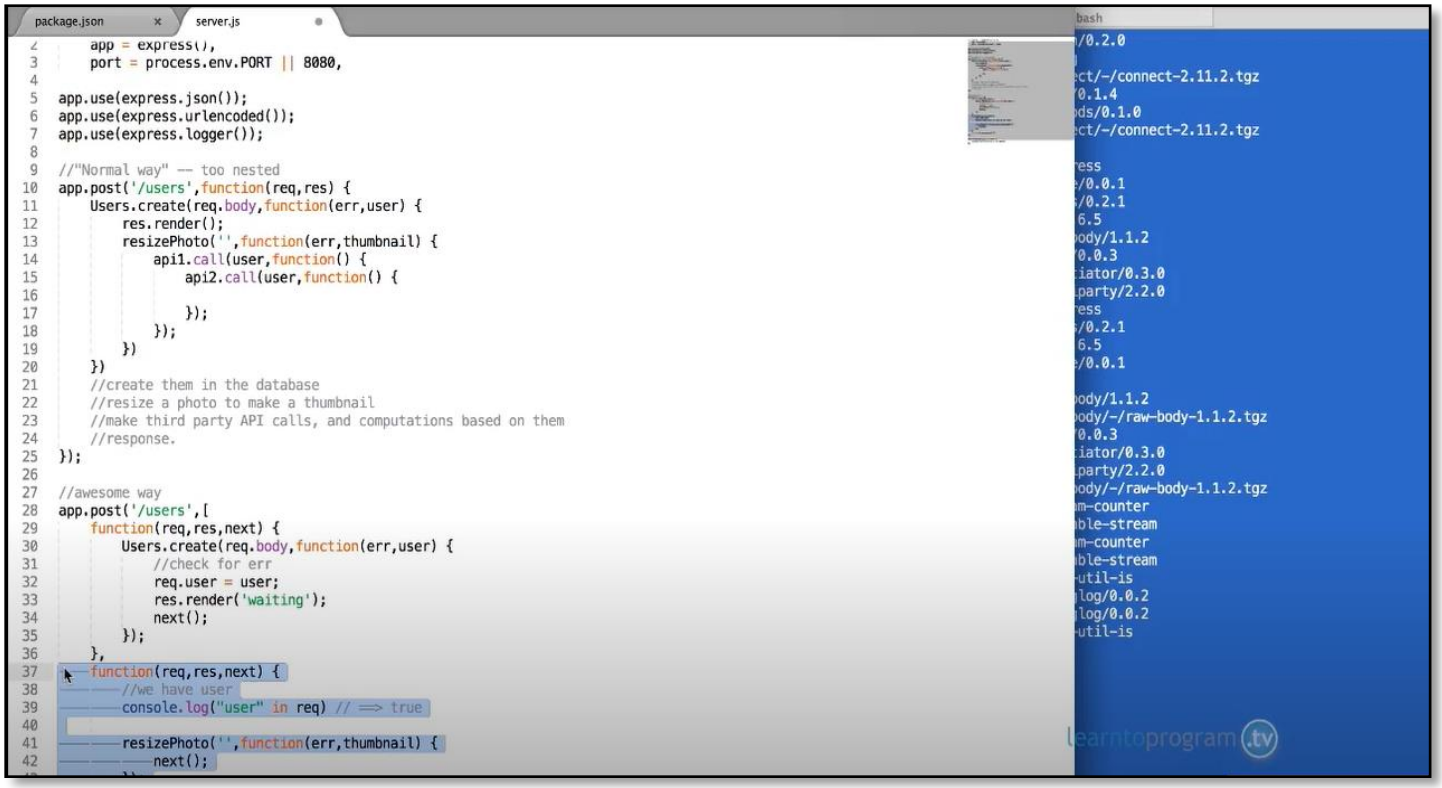
Recap: Give a quick recap of the topics covered in the last session and ask the students if they have any doubts. Resolve their doubts and then, inform them that this session will be a long one as multiple topics will have to be covered in it.

Start the session.

7.1 Request Middleware

CLASS DURATION: 25 MINUTES
VIDEO DURATION: 9.24 MINUTES
FACILITATION GUIDELINES -

Show: Play Request Middleware: Session 7 Video – 7.1



Content: Explain to the students that Express is a Web framework for routing and middleware with just rudimentary functionality and that a succession of middleware function calls makes up an Express application.

Further, let them know that in the request-response cycle of an application, middleware functions have access to the request object (`req`), the response object (`res`), and the next middleware function. Tell them that a variable named next is typically used to identify the next middleware function and discuss the tasks that can be performed by the middleware functions.

Following tasks can be performed by middleware functions:

- Any code can be run.
- Change the request and response objects as required.
- Bring the request-response cycle to a close.
- Invoke the next middleware function of the stack.

Let the students know that the current middleware function must call `next()` to handover control to the next middleware function if the request-response cycle is not yet complete. Otherwise, the request will go unanswered. Further, discuss the types of middleware that the Express application can utilize.

Show: Play the video from 5:00 to 9:24.

Following types of middleware can be used by an Express application:

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware

- Third-party middleware

Application-level middleware:

Following are the key properties to keep in mind while using the application-level middleware:

- Using the app function, you can bind application-level middleware to an instance of the app object.
- The `use()` function and app function are two functions that you can use in application level middleware.
- `METHOD()` functions, where METHOD is the lowercase HTTP method of the request that the middleware function handles (for example, GET, PUT, or POST).

NOTE: Only middleware functions loaded via the `app.METHOD() or router.METHOD()` functions will operate with `next('route')`.

Router-level-middleware:

Explain how Router-level middleware functions are similar to application-level middleware, with the exception that it is connected to an `express.Router()` instance.

Built-in-middleware:

Elaborate on the fact that Express is no longer dependent on Connect as of version 4.x and that the previously bundled middleware functions have been separated into distinct modules. Additionally, discuss about the middleware functions that are already part of Express.

Following are the middleware functions built into Express:

- The `express.static` is a static asset server that provides HTML files, pictures, and other media.
- Incoming requests with JSON payloads are parsed by *express.json*. It is important to note that Express 4.16.0+ is required.
- Inbound requests with URL-encoded payloads are parsed by `express.urlencoded`. Once again, Express 4.16.0+ is required.

Third-party- middleware:

Describe to the students as to how to add functionality to Express programs by using third-party middleware.

Install appropriate Node.js module, and then, load it in your app either at the application level or at the router level.

```
$ npm install cookie-parser
```

In-Class Question: Which function is used to modify requests for each middleware?

Answer: The `next()` function is used.

Additional Reference:

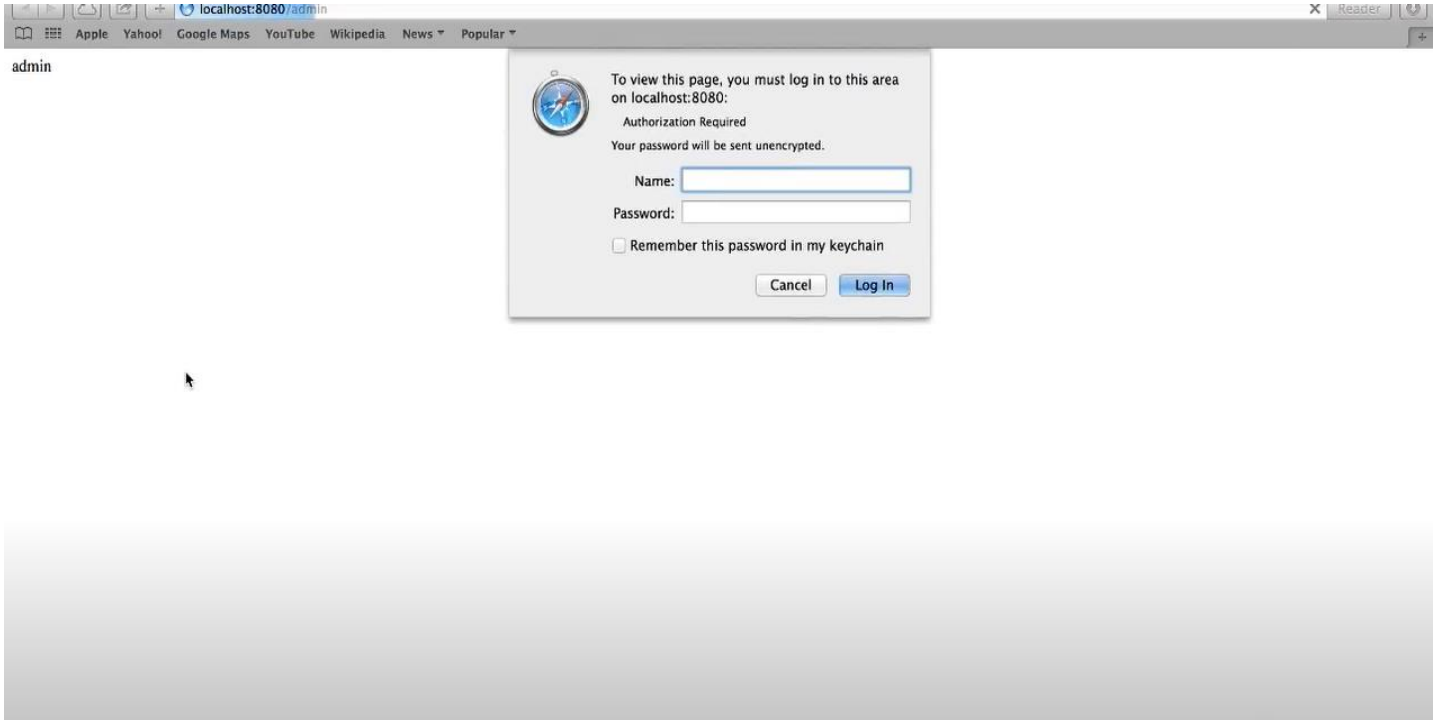
Refer to the following links for more information:

https://expressjs.com/en/guide/writing-middleware.html

https://selvaganesh93.medium.com/how-node-js-middleware-works-d8e02a936113

7.2 Basic Authentication Systems

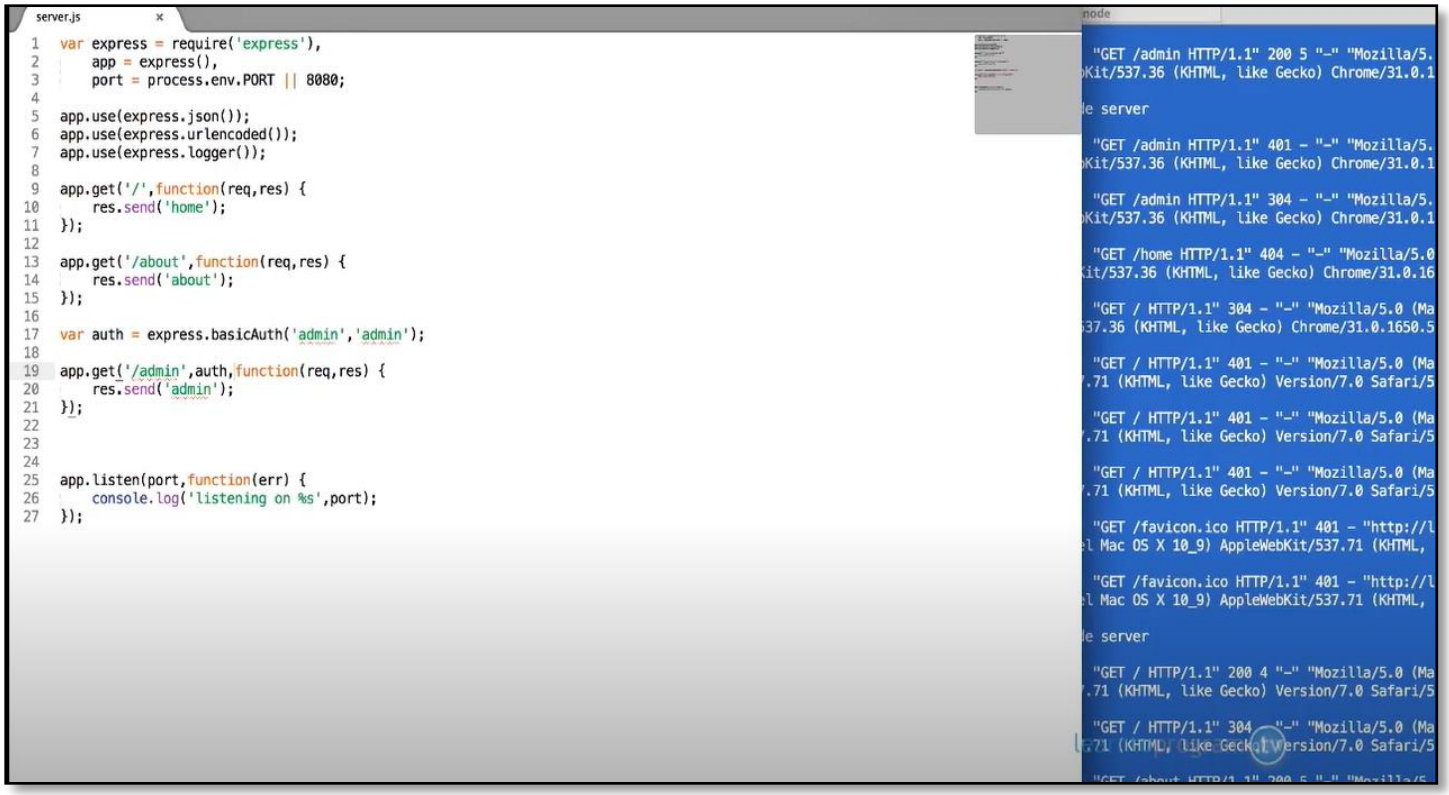
CLASS DURATION: 25 MINUTES
VIDEO DURATION: 6.06 MINUTES
FACILITATION GUIDELINES -
Show: Play Basic Authentication Systems: Session 7 Video – 7.2.



Content: Elaborate to the students that the goal of this session is to construct a basic login system with a backend written in Node.js and the Express framework that renders HTML pages for registering users and enables them to log in.

Let them know that this type of project usually necessitates the use of a database to store the credentials. However, it will be avoided as the purpose of this project is to provide an overview of the login system. The user credentials will be stored in arrays as a result and all credentials will be destroyed once the server is shut down.

Show: Show the image to the students.



Open a terminal and go to this directory, where you will run the command '`npminit`', which will prompt for some information. *Package-lock.json and package.json* files will be created after inputting the information.

Following is the command to install Express, bcrypt, and body-parser.

```
$ npm install express bcrypt body-parser -save
```

Discuss about the major points involving Express, bycrypt, and body-parser.

Following are important points about Express, bycrypt, and body-parser:

- Express is a Node.js Web application framework that offers a comprehensive range of functionality for both Web and mobile apps. APIs can be built quickly and easily with the help of Express.
- Bcrypt is a library that makes hashing passwords easier.
- In Express.js, a body-parser is necessary to handle HTTP POST requests. It extracts and exposes the whole body portion of an incoming request stream on `req.body`.
- The node modules folder will be created when the preceding step is executed.

Creating our project directory:

Establish a directory called project to start off. All the files and folders will be created in this directory in the next steps.

Client-side programming:

Explain to the students that the next step is to work on the client side. In the project directory, make a public folder and instruct to make the index.html file inside this subdirectory. This is the file that our server will display when a user connects to our IP address. The code for this file is basic and it is shown in image. Outline to the students that it will allow users to get to the registration and login pages.

Server-side programming:
Illustrate that the next step is to begin working on the backend. You will do this by writing a server.js file that stores the user credentials in an array as shown in the image.

In case of Error:

Case 1: Explain that if the entered name is found in the memory but the password entered is incorrect, our code will attempt to compare the entered and stored passwords. Finally, it will show a message that says, 'Invalid name or password.'

Case 2: Additionally, elaborate that if the input name does not exist in the database, our application will compare a fake password to the entered password before displaying — 'Invalid name or password.'

This way, the program will take the same amount of time to display the message to the user in both circumstances. Summarize by saying that in the end, you avoid revealing which option is incorrect to the user. This will improve the security of our data as well.

In-Class Question: What is basic authentication method?

Answer: Basic authentication works by prompting a Website visitor for a username and password. This method is widely used because most browsers and Web servers support it.

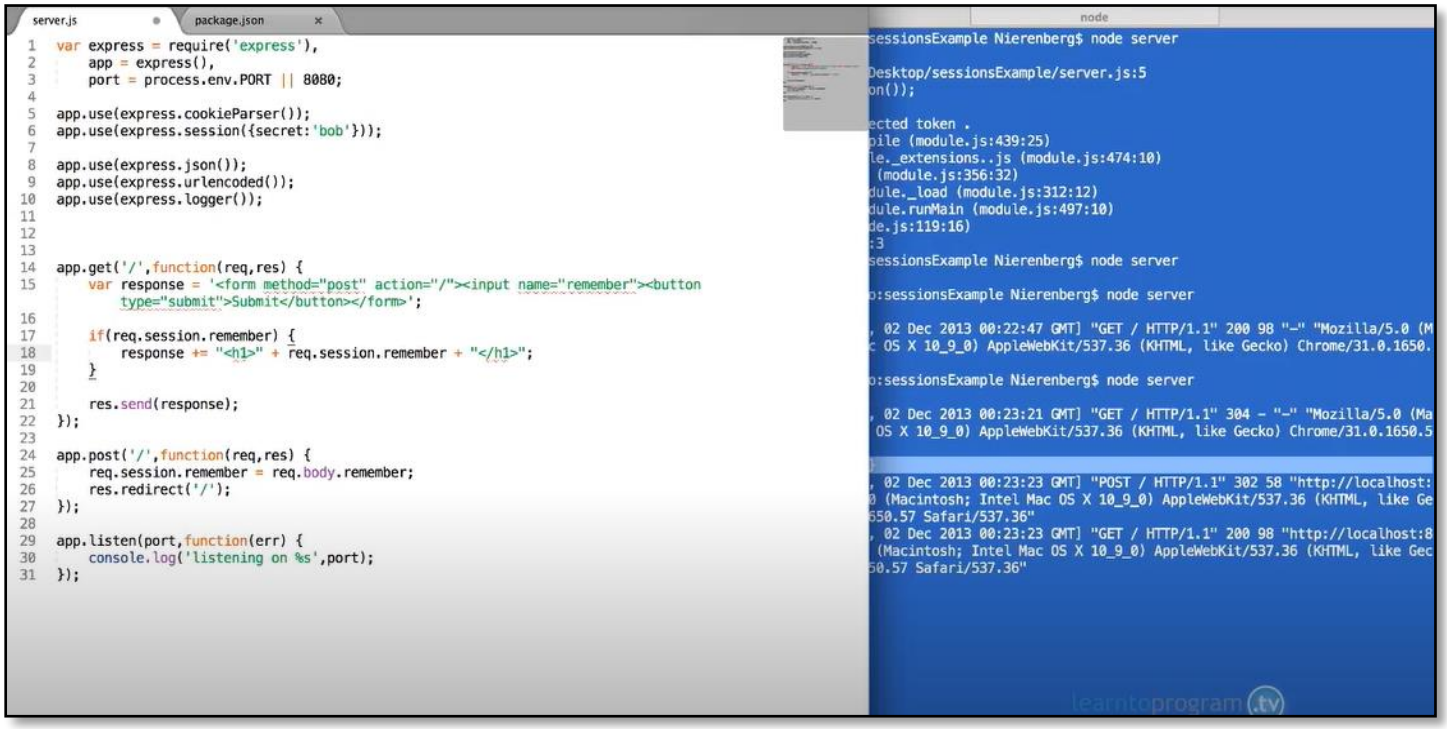
Additional Reference:

Refer to following links for more information:

https://stackoverflow.com/questions/34860814/basic-authentication-using-javascript
https://jasonwatmore.com/post/2018/09/24/nodejs-basic-authentication-tutorial-with-example-api

7.3 Express Sessions

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 7.08 MINUTES
FACILITATION GUIDELINES -
Show: Play Express Sessions: Session 7 Video – 7.3



Content: Explain to the student that HTTP protocol is the foundation of a Website. HTTP is a stateless protocol, which means the client and server forget about each other at the end of each request and answer cycle.

Describe to them that this is where the session enters the picture and that to allow the server to keep track of the state of the user, each session will carry some unique data about that client. Further, elaborate on the fact that the state of the user is saved in the server memory or a database in session-based authentication.

Working

Illustrate the working by elaborating on the fact that when a client requests a login, the server creates a session and stores it on the server. A cookie is sent by the server when it responds to the client. This cookie will hold the unique id, which was previously kept on the server and is now stored on the client. Every time a request is made to the server, this cookie will be transmitted.

Further, discuss that to preserve a one-to-one match between a session and a cookie, this session ID is used to look for the session maintained in the database or the session store which makes the HTTP protocol connections stateful.

Session vs. Cookie

Explain to the students that a cookie is a key-value pair that the browser stores. Every HTTP request delivered to the server is accompanied by cookies from the browser.

Elaborate on the fact that it is possible to save only small amounts of information in a cookie and that it is inadvisable to store any kind of user credentials or secret information in a cookie. If done, a hacker can simply gain access to that information and use it for bad purposes.

Explain that the data is stored on the server side, in a database or a session store and that it can handle higher amounts of data as a result. Further, summarize by saying that a session is authenticated with a secret key or a session id that you acquire from the cookie on every request to access data from the server-side.

Session (options)

Create a session middleware using the options provided.

Note: Only the session ID is retained in the cookie, not the session data. Session data is saved on the server. It is also important to understand that the cookie-parser middleware is no longer required for this module to work since version 1.5.0. On request/response, this module now reads and writes cookies directly. If the secret between this module and cookie-parser differs, using cookie-parser may cause problems.

Show: Play the video from 4:00 to 7:08.

Following are the options that can be used to create a session middleware:

`req.session`

Examine the request parameter `req.session` to store or access session data, which is (normally) encoded as JSON by the store, thus, nested objects are usually fine.

`Session.regenerate(callback)`

Examine the `session.regeneratecallback` method to regenerate the session. At `req.session`, a new SID and Session instance will be created, and the callback will be triggered.

Session.destroy(callback)

Utilizing session.destroy, session will be destroyed, and the `req.session` field will be unset. The callback will be triggered once all the execution in the page is finished.

Session.reload(callback)

Utilize this method for the session data to reload from the store and the `req.session` object to be repopulated. The callback is triggered once everything is finished.

Session.save(callback)

If the session data has been changed, this function is automatically called at the conclusion of the HTTP response (through this, behavior can be altered with various options in the middleware constructor). This procedure is usually not required to be called as a result.

Session.touch()

Examine `session.touch()` and note that `the.maxAge` parameter is updated via `Session.touch()`. In most cases, it is unnecessary to call this because the session middleware will do it for you.

Req.session.id

Each session is identified by a unique ID. This is an alias for `req.sessionID` and it cannot be changed. It was included so that the session ID could be accessed from the session object.

Req.session.cookie

Each session comes with its own cookie object. Utilize this to change the session cookie for each individual visitor. Set `req.session.cookie.expires` to false, for example, to make the cookie last only as long as the user-agent is activ

SESSION STORE IMPLEMENTATION

Every session store must implement methods and be an Event Emitter. Discuss the different types of approaches of session store.

Following are the essential, suggested, and optional approaches:

- This module will always call the required methods on the store.
- If the store is available, this module will use the recommended ways.
- Optional methods are those that this module does not employ, but which aid in the presentation of uniform stores to users.

In-Class Question: What is session fixation vulnerability?

Answer: Session fixation is an attack that permits an attacker to hijack a valid user session.

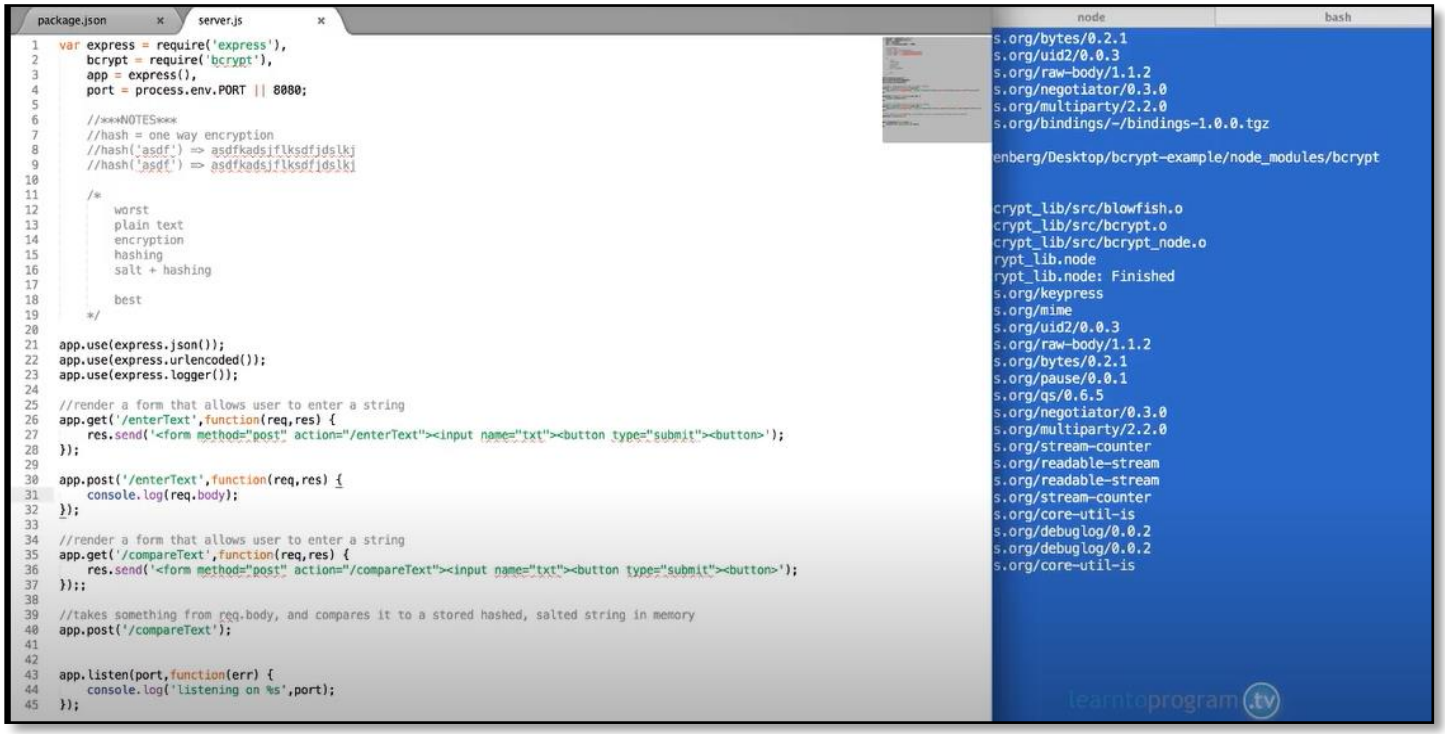
Additional Reference:

Refer to the following links for more information:

https://www.tutorialspoint.com/expressjs/expressjs_sessions.htm
https://www.geeksforgeeks.org/session-management-using-express-session-module-in-node-js/

7.4 Bycrypt

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 13.38 MINUTES
FACILITATION GUIDELINES -
Show: Play Bycrypt: Session 7 Video – 7.4.



Content: Explain to the students that in JavaScript, bcrypt has no dependencies. It is compatible with the Node.js C++ bcrypt binder which also works in the browser.

Further, elaborate that apart from integrating a salt to guard against rainbow table attacks, bcrypt is an adaptive function and that the iteration count may be increased over time to make it slower, making it resistant to brute-force search attempts even as computing power increases.

Outline the fact that while bcrypt.js is compatible with the C++ bcrypt binding, it is written in pure JavaScript and hence slower thereby limiting the number of iterations that can be processed in the same amount of time.

Let them know that the maximum input length is 72 bytes, while the output hashes are 60 characters long and that the library is compatible with both CommonJS and AMD loaders, and if none is available, it is exposed globally as `dcodeIO.bcrypt`.

Show: Show the following image to the students.

Node.js:

Explain that the `randomBytes` interface in the Node.js crypto module is used to generate secure random integers.

```
$ npm install bcryptjs
```

Describe to the students that to retrieve safe random numbers in the browser, bcrypt.js uses the Web Crypto `APIgetRandomValues` method. If no cryptographically safe source of randomization is available, use `bcrypt.setRandomFallback` to define one.

```
varbcrypt = dcodeIO.bcrypt;
```

Note: Asynchronization breaks a crypto process into small parts under the hood. After a chunk is completed, the execution is pushed to the rear of the JS event loop queue, effectively sharing computational resources with the other operations in the queue.

API:

`setRandomFallback(random)`

Explain that if neither the crypto module of the Node nor the Web Crypto API is available, this specifies the pseudo random number generator to use as a fallback. Additionally, make them note that it is critical that the PRNG used is cryptographically secure and properly seeded.

In-Class Question: After installing bcryptjs module how can you check your request version in the command prompt using the command?

Answer: `npm version bcryptjs`

Conclusion: Give the summary of all the topics that have been covered in this session.

QUERIES REGARDING SESSION

CLASS DURATION: 10 MINUTES

FACILITATION GUIDELINES

Ask students if they have any queries regarding the session and resolve the queries.

SESSION 8: DEPLOYMENT

Greeting: Welcome students back to the course, greet them, and ask if they are ready to get started with the next session of the course. Tell them that this session will carry forward from the previous session and will continue explaining further lesson.

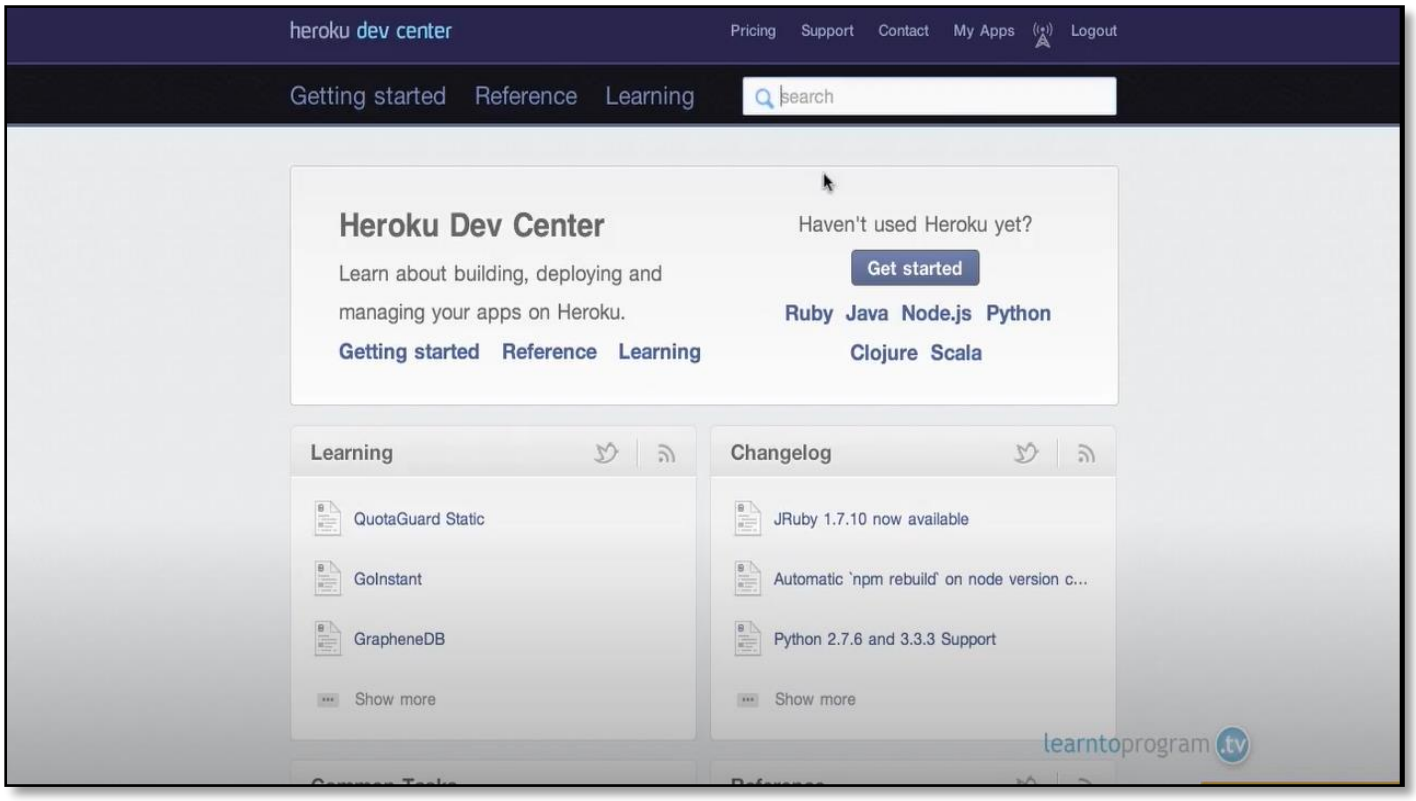
Quick Recap

Recap: Give a quick recap of the topics covered in the last session and ask the students if they have any doubts. Resolve their doubts and then, inform them that this session will be a long one as multiple topics will have to be covered in it.

Start the session.

8.1 Intro To Heroku

CLASS DURATION: 25 MINUTES
VIDEO DURATION: 5.41MINUTES
FACILITATION GUIDELINES -
Show: Play Intro to Heroku: Session 8 Video – 8.1.



Content:

Explain to the students that cloud computing lets users connect to a shared pool of adjustable computing resources whenever required.

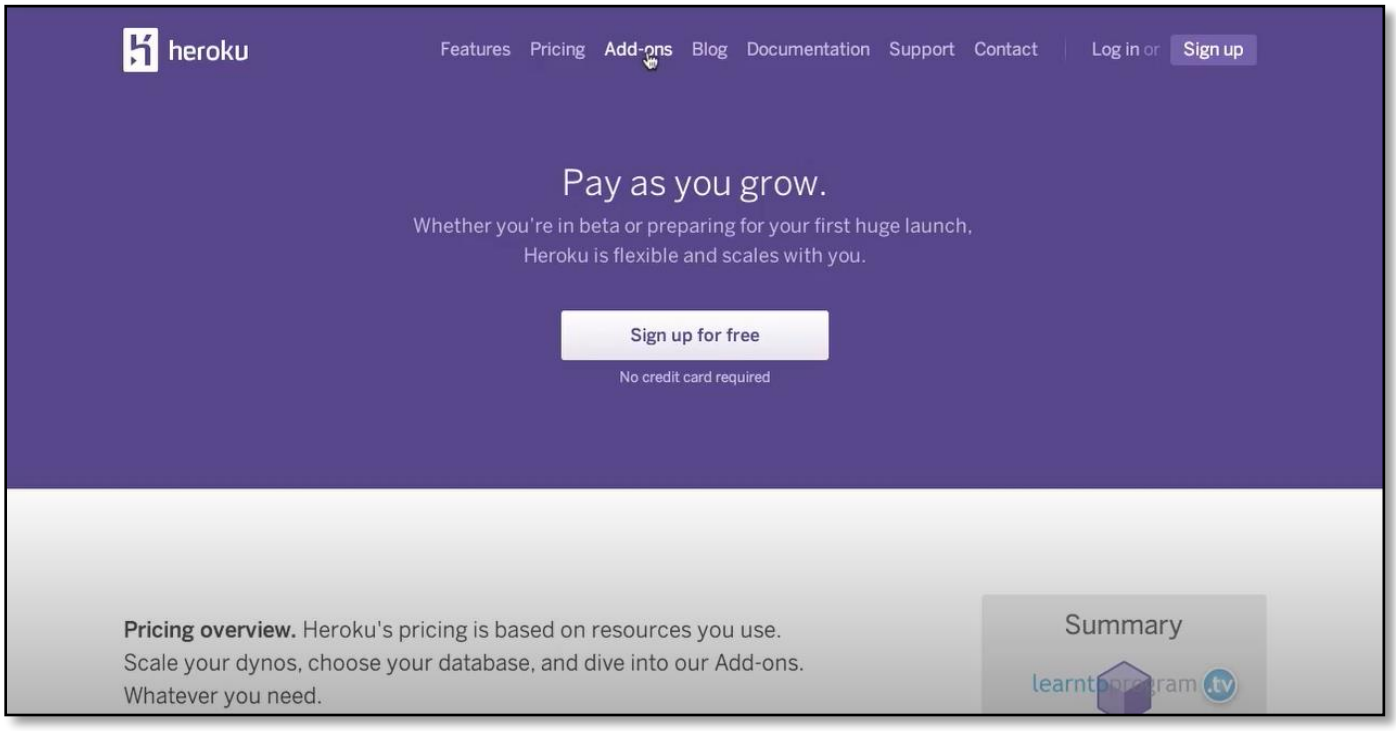
Let them know that Heroku is a Platform as a Service (PaaS) that provides software development tools and that as a PaaS, Heroku enables businesses to swiftly deploy, build, manage, and scale enterprise-level applications while avoiding the infrastructure difficulties that come with hosting a high-quality software. Explain that it acts as a go-between for hosting/infrastructure and Salesforce.

Amazon Web Service (AWS), an Infrastructure as a Service (IaaS) provider and the foundation of our pyramid, hosts all Heroku-based applications. Let the students know that AWS handles computing resources and cuts setup time in new app development in half. It subsequently handles load balancing, operating system selection, servers, networks, storage, server logging and monitoring, application health monitoring, and container organization.

Outline the fact that Amazon Web Services (AWS) is the gold standard for IaaS and is used by just about everyone for Cloud purposes. Netflix, Dow Jones, The Weather Channel, sections of numerous federal governments, and Citrix are among the companies on the list.

Next, let the students know about Salesforce, the pinnacle of the pyramid. Explain to them that Salesforce is Software as a Service (SaaS) company that sells licenses for its software to customers and delivers it via the internet. Salesforce provides full support for Heroku-based applications (a SaaS).

Show: Show following image to the students:



Discussion:

No Language Constraints

Illustrate that Heroku is never a one-size-fits-all solution and that it supports most open-source languages, allowing user to utilize the strengths. Explain that user has the faculty to create a cluster of applications, each in an autonomous language, or has the ability to create a single language that is already strongly supported inside the company. Let them know that the broad ecosystem of Heroku, backed by AWS support, makes moving to the cloud a breeze and that locally hosted applications are intrinsically tough to scale, whether they reside on server or another machine. Moving the application to the cloud qualifies to scale with the organization.

Discuss the real time example about Heroku with the students.

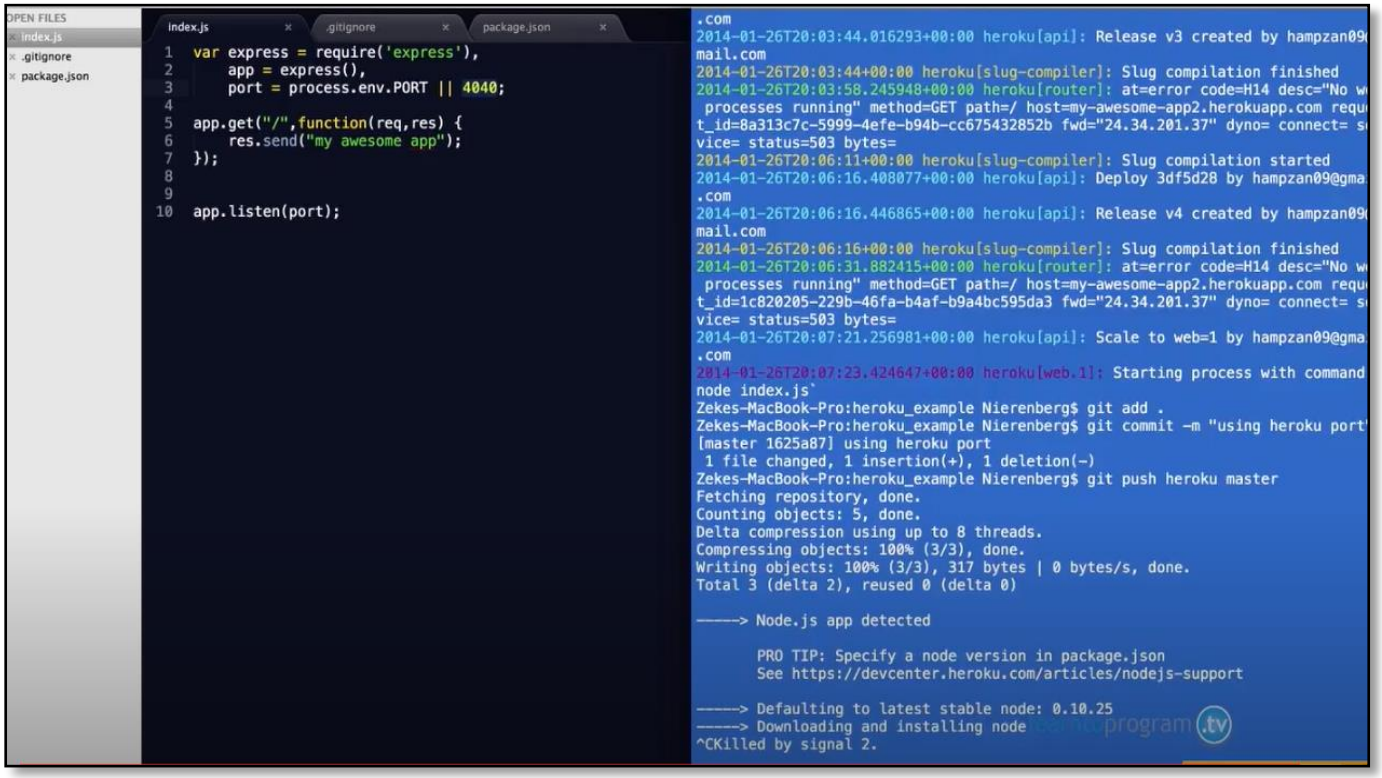
Example:
Assume a retailer who keeps track of inventories on a server in the back office using a custom-built Java application. Business is booming and so the owner decides to add a few more stores across the country. The new locations of the experiencer will VPN into that back office server to check product availability and maintain accurate inventory records to manage inventory consistently. The entirety of the business can suffer if the internet goes down at the server location, or if server struggles to accommodate the added burden of several processes and connections.

The inventory management of the User application will migrate to Heroku and deploy globally from a powerful network of cloud-based servers maintained by the squad at Amazon for

security and infrastructure specialists with just a few minor changes to the backend. The common inventory management data is now accessible to each of the sites independently.

8.2 Deploying an App on Heroku

CLASS DURATION: 40 MINUTES
VIDEO DURATION: 16.33 MINUTES
FACILITATION GUIDELINES -
Show: Play Deploying an App on Heroku: Session 8 Video – 8.2.



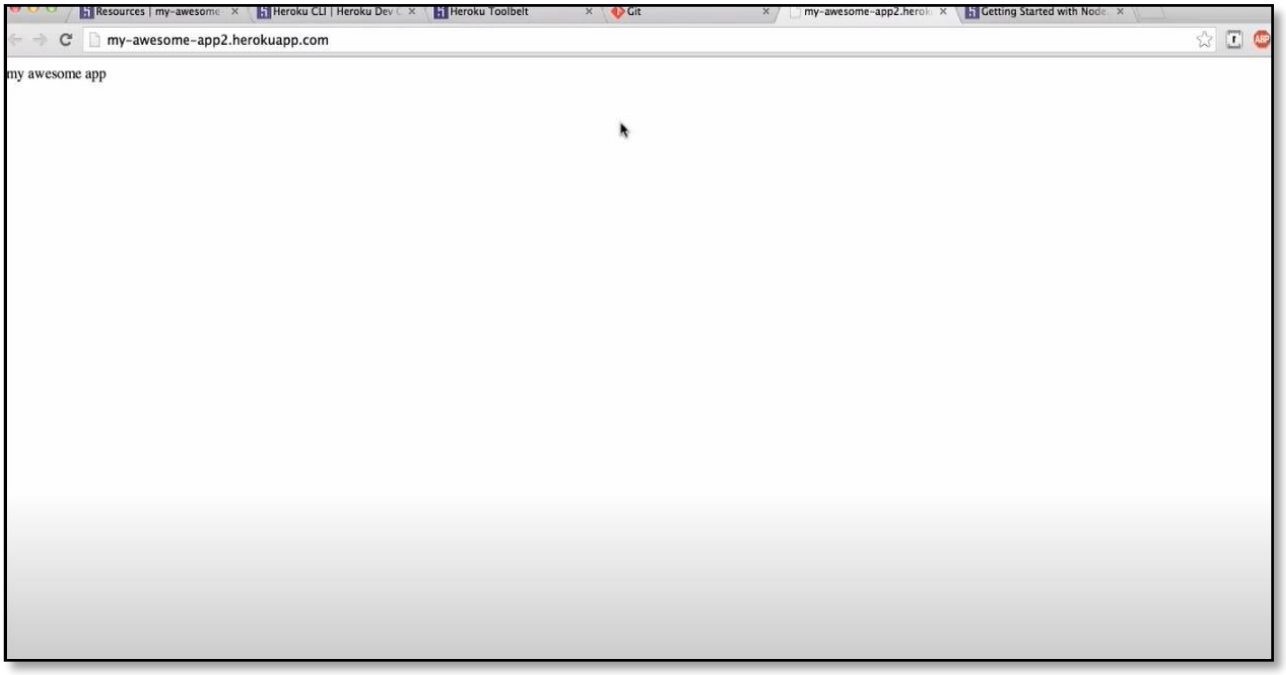
Content:
Discuss with the students the steps involved in deploying an app on Heroku.
Following are the steps to deploy an app on Heroku:

- The Heroku Command Line Interface (CLI) will be installed first. The CLI can definitely be utilized to run and enhance the applications, as well as provision add-ons, view logs, and run them locally.
- User can use the Heroku command from the terminal once the installation is complete.
- To utilize the command shell on Windows, open the Command Prompt (cmd.exe) or Powershell.
- To utilize the CLI for Heroku, make sure to employ the login command for Heroku:
  `$heroku login`
- User will create a sample application that is geared up to be commissioned to Heroku in this phase.
- User can deploy project to Heroku after committing modifications to git.

```
$ git add.
$ git commit –m ``
$ heroku login
$ heroku create
```

```
$ git push heroku main
```

Show: Show following image as the output that the user gets after deploying the application:

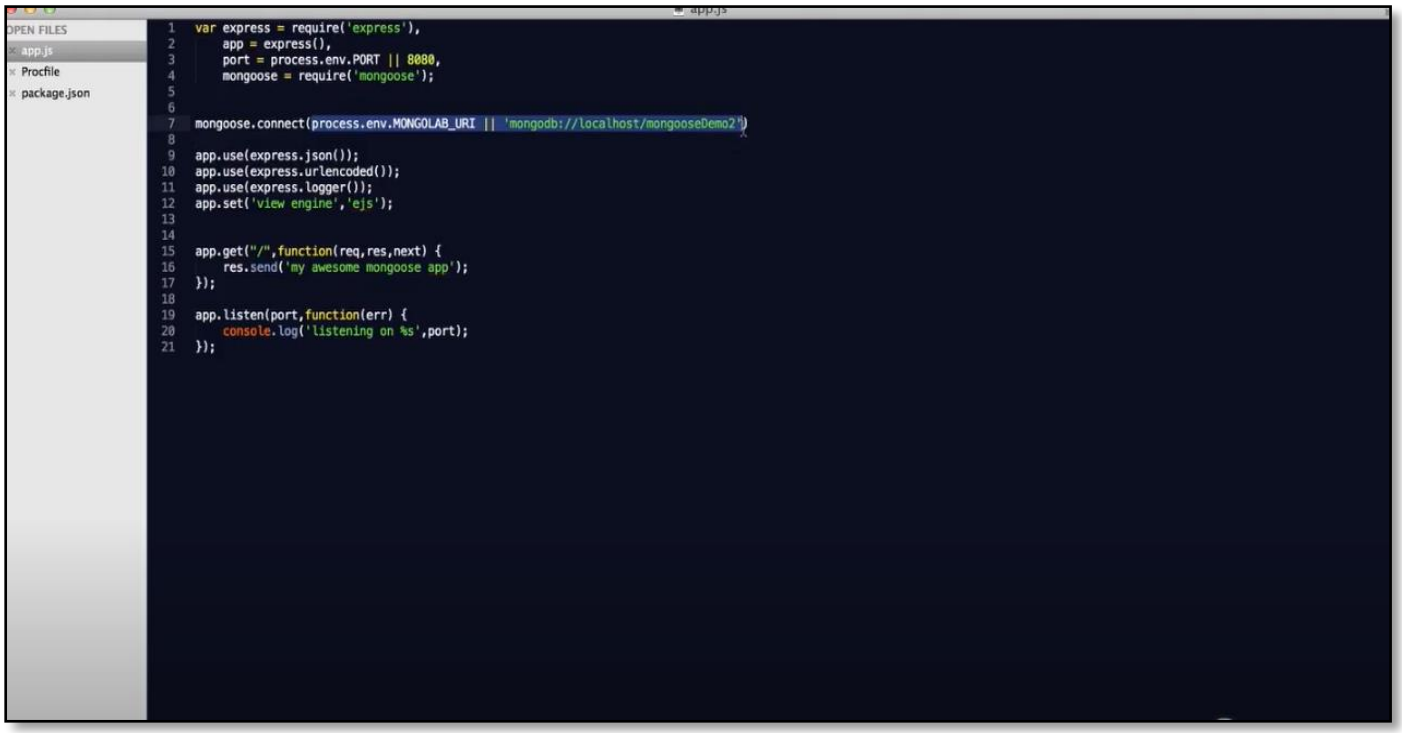


- For opening the app in browser, type `heroku open`.

  In-Class Question: Is Heroku good for deployment?
  Answer: Heroku is a popular solution for many development projects since, it is simple to utilize. It offers straightforward application creation and deployment, with an exceptional basis on facilitating customer-focused apps.

8.3 Environment Configuration

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 9.51 MINUTES
FACILITATION GUIDELINES -
Show: Play Environment Configuration: Session 8 Video – 8.3.



Discussion:

Discuss how a single program runs in several settings at all times, both on the development workstation and in production on Heroku, with the students. A single open-source application could be used in dozens of different ways.

Explain that, in spite of the certainty that all of these environments run the same code, their settings are frequently different. For example, a staging and production environments of the app may use different Amazon S3 buckets, requiring the use of multiple bucket credentials.

Explain to the students that environment variables (rather than the app source code) should be utilized to store the app environment-specific configuration. This allows users to customize each environment independently, while also preventing secure passwords from being stored under version control.

Illustrate how the user when operating domestically or on a conventional host, frequently sets environment variables in `.bashrc` file, and `.Config` variables are used on Heroku.

Explain that if a user sets or removes a configvar using any method after the app is restarted, a new release is made, and that configvar values are durable, meaning they do not change when the application is published or restarted. Unless there is a requirement to change it, the user should only set a value once.

Discuss about utilizing the CLI for Heroku, Dashboard, and configvars with the students.

Using Heroku CLI:

The CLI for Heroku has `herokuconfig` commands manage the configvars of the app in a simple manner.

- Current config values: `$herokuconfig`
- Set config values: `$herokuconfig: set GITHUB_USERNAME = abc`
- Remove config values : `$herokuconfig: unset GITHUB_USERNAME = abc`

Using Heroku Dashboard:

User can also update config variables in the Settings tab of the Heroku Dashboard for app.

Access configvar-values from code:

`Config` variables are available as environment-based variables to the code of the app. In Node.js, for example, user can use `process.env.DATABASE` URL in order to facilitate the DATABASE URL `config` property of the app.

Configvar policies:

- To warrant that they are available from all programming languages, configvar keys must only contain alphanumeric letters and the underscore character. The hyphen character must not be used in configvar keys.
- Each configvar data of the app (the sum of all keys and values) cannot exceed 32 KB.
- A double underscore (__) must not be used to start a configvar key.
- The key of a configvar must avoid beginning with `HEROKU_` except if the Heroku platform has established it.

Add-on configvars:

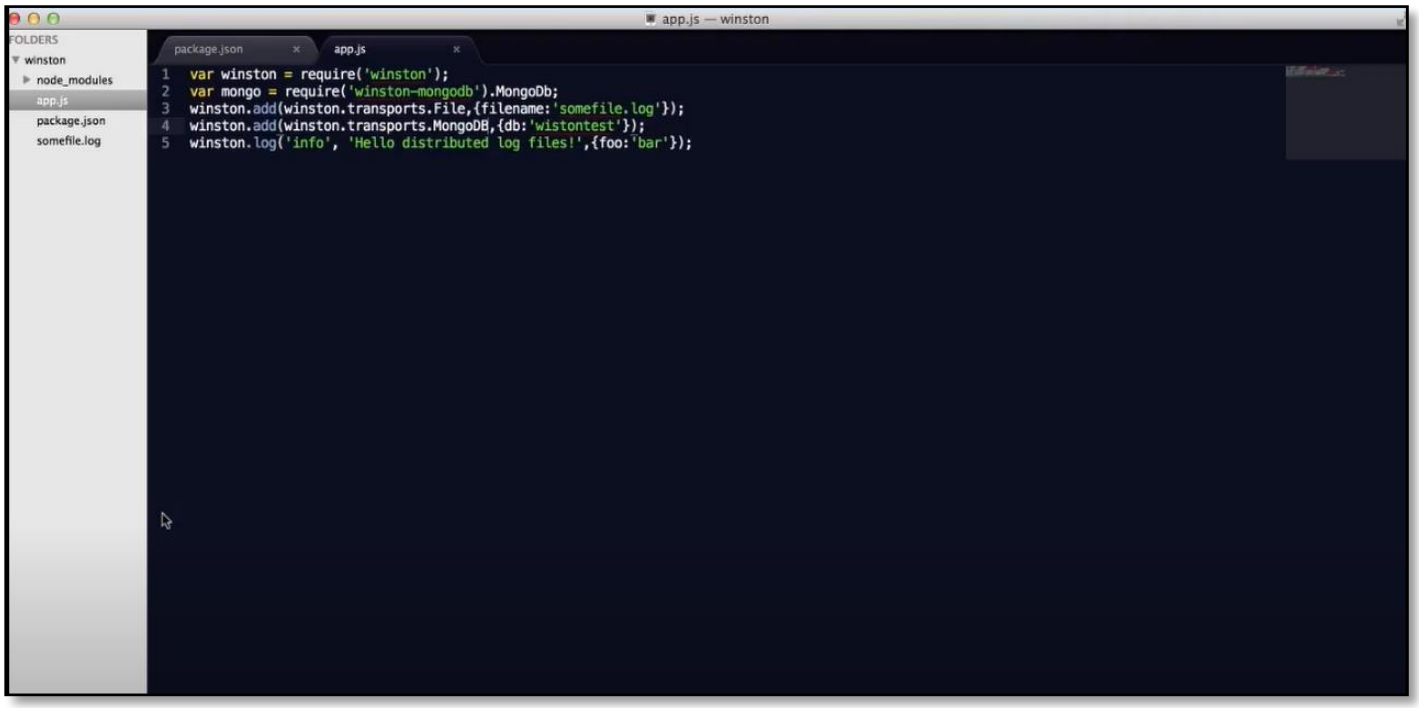
Explain to the students that when users install an add-on to the app, it normally annexes one or more config variables. The add-on supplier can change the values of these configvars at any moment.

In-Class Question: What is the *.env* file?

Answer: The .*env* file, often known to be the *dotenv* file, is a basic text configuration file used to control the environment constants of applications. Majority of applications will not change between the Local, Staging, and Production environments.

8.4 Logging in Production

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 7.45 MINUTES
FACILITATION GUIDELINES -
Show: Play Logging in Production: Session 8 Video – 8.4.



Discussion:

Explain to the students that logs are generated from the output streams of all operational processes, system components, and backing services. Logplex is a Heroku feature that combines log streams from multiple sources into a single channel, laying the foundation for comprehensive logging.

Runtimelogs:

Illustrate that for a deployed app, Heroku collects the log categories.

Following are the log categories collected by Heroku:

1. App logs: This includes logs spawned by the code along with the dependencies of the app, as well as logs spawned by the app itself.

2. System logs: Messages describe actions made in the name of the user app by the Heroku platform architecture, such as resuming a dead process, snoozing or waking a Web dyno, or providing an error page because of a fault in user app.

3. API logs: Messages regarding administrative operations done by user and other app developers, such as releasing new code, scaling the process formation, or toggling maintenance mode.

Buildlogs:

Describe to the students that the logs created during the development and deployments of the user app is distinct from the runtime logs. The Activity tab in Heroku Dashboard of the user app contains logs for both successful and unsuccessful builds.

Log History Limits:
Explain that Logplex is not meant to be used for archiving log messages. Its purpose is to aggregate and route them. It saves the combined logs' last 1,500 lines for a week before they expire.
Allow them to add one of the Heroku platforms logging add-ons to the user project for better production-ready log durability. Most of these add-ons come with a free plan to get you started.

If the user wants complete control over what happens to logs, they should install their own log drains.

Writinglogs:

To access the most recent logs for the app, have the students use the heroku logs command.

Make a point about how the logs are not in any specific order. Logplex gathers logs from multiple sources and merges them into a single log stream. Logplex, on the other hand, is built on a distributed architecture for high availability, so log messages may appear out of order as they are gathered across multiple Logplex nodes.

Following is the format of the Heroku logs commands output:

1. Timestamp: The date and time when the dyno or component generated the log line. The timestamp format, which includes microsecond precision, is defined by RFC5424. All of the app dynos have the source, app (Web dynos, background workers, and cron). All Heroku system components use Heroku as a source (HTTP router, dyno management).

timestamp source[dyno]: message

2. Source: The app is present in all of the app dynos (Web dynos, background workers, and cron). All of the system components use Heroku as a source (HTTP router, dyno management). The dyno refers to the dyno or component that created the log line. For example, worker.3 represents Worker #3, whereas router represents the Heroku HTTP router.

3. Message: Any lines created by dynos that exceed 10,000 bytes are broken into 10,000-byte chunks by Logplex, with no extra trailing newlines. Each chunk is sent as a separate line in the log file.

Additional Reference:

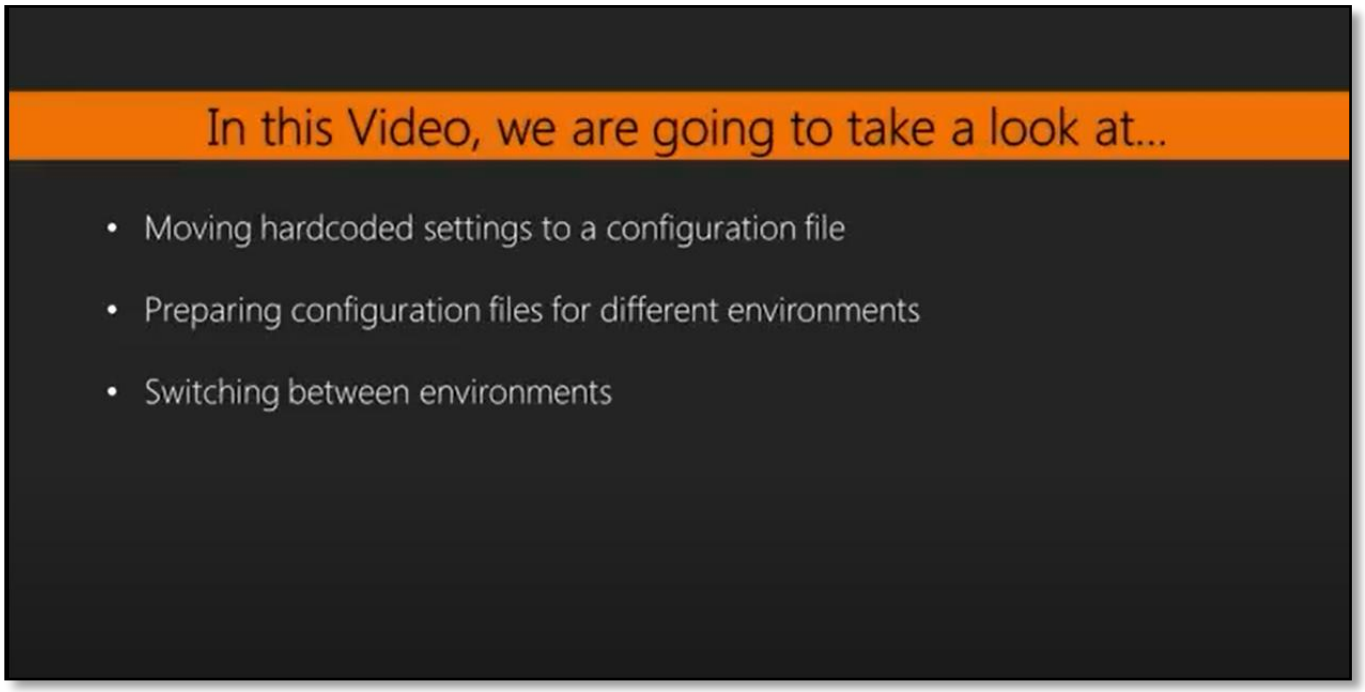
Please refer to following links for more information:

https://devcenter.heroku.com/articles/logging

https://devcenter.heroku.com/articles/production-check

8.5 Adding Configuration

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 6.49 MINUTES
FACILITATION GUIDELINES -
Show: Play Adding Configuration: Session 8 Video – 8.5.



Discussion:

Production and development modes:

Explain to the students that a development mode is supported by several languages and frameworks. More debugging is usually possible, as well as dynamic reloading or recompilation of updated source files.

Let them know that in a Ruby environment, for example, user can activate this mode by setting the `RACK ENV`configvar to development.

Further, explain to them that on a production Heroku app, it is critical to comprehend and make a note of these `config` vars. While a mode for development is ideal for testing, it is not ideal for production or deployment since, it has the ability to reduce performance. At least two environments operate Heroku app.

Following are the two environments that operate the Heroku app:

- On the personal computer.
- Heroku platform used for deployment.

Software should ideally run in two more environments:

1. Test, which allows user to run the test suite of the app in a secure environment.
2. Staging allows user to test a new version of the program in a production-like environment before releasing it. Maintaining these discrete environments as separate Heroku apps and using each environment strictly for its intended purpose will help prevent buggy code from being deployed to production.

Summarize, that Linux stacks in Heroku are likely to differ from the operating system of the development machine and that this method is very crucial. Finally, explain that user cannot be sure that code that works in user local development environment will operate in production exactly the same way.

Linking environments with pipelines:

Illustrate that Pipelines on Heroku make linking the environments of the app and promoting code from staging to production a breeze.

Managing staging and production configurations:

Several programming languages and programming frameworks allow user to switch between development and production mode. When in development mode, for example, user might utilize another database, have higher logging levels, and send all emails to user rather than end users.

From the Dashboard that belongs to Heroku, create a new pipeline and add staging and production environments to it.

User applications services and libraries require their own configuration variables, which must match those on production. For example, user might utilize an alternative S3 bucket for staging than you use for production, therefore the values for the keys will be divergent.

User said git push rather than git push staging staging:master. This is made feasible by the push.default configuration, which, when set to tracking, causes local branches to automatically push changes to the remote repositories that they track.

Additional Reference:

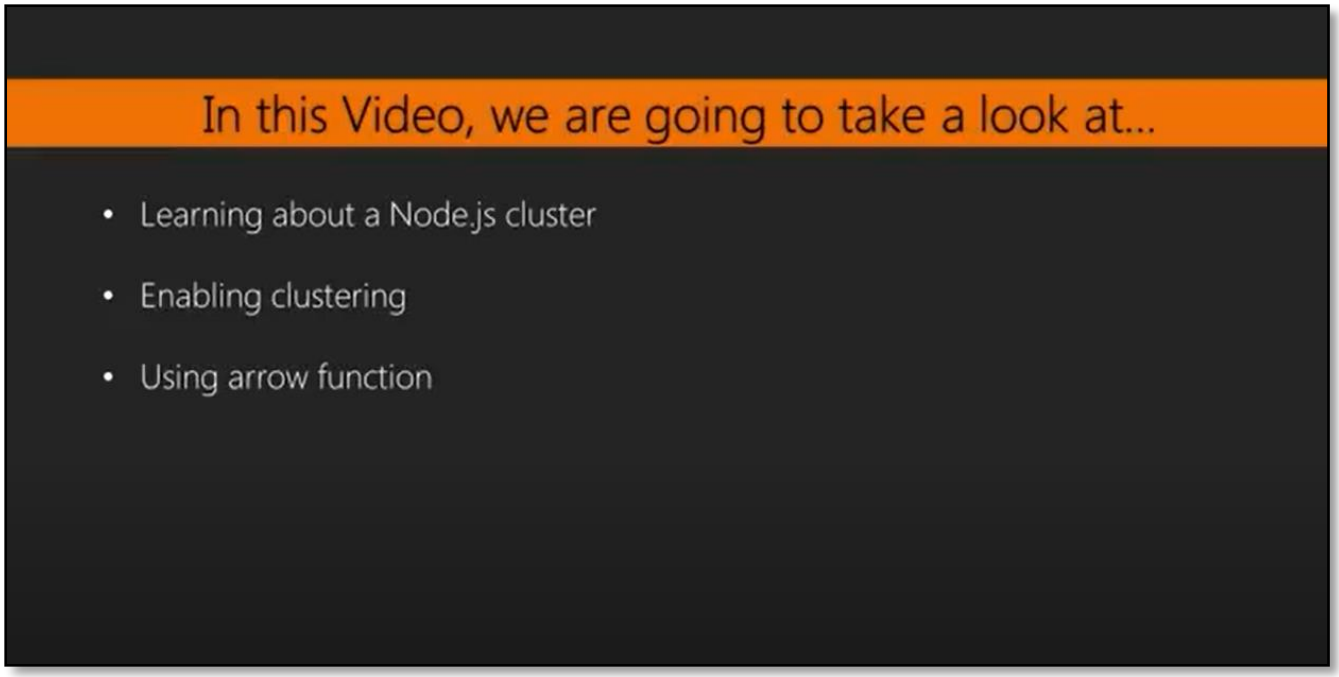
Please refer to following links for more information:

https://catalins.tech/heroku-environment-variables

https://owenconti.com/posts/copying-heroku-environment-variables-across-projects

8.6 Scaling Out Of With a Cluster

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 4.30 MINUTES
FACILITATION GUIDELINES -
Show: Play Scaling out of With a Cluster: Session 8 Video – 8.6.



Discussion:
In this video, students will learn that when a user thinks about speedier programs, usually they think of code optimizations and occasionally concurrency, which Node handles with its Event Loop via callbacks and abstractions such as Promises, async-await, and so on.

Everything runs on a single processor on a computer with multiple processors since the Event Loop is a single-threaded paradigm. This implicates that by using Clustering to benefit from these other processors, one can boost the performance of the programs.

Further, discuss with the students about cluster, process, port, and arrow functions to get them to understand more about it.

- **Cluster:** A cluster in Node is a congregation of processes that share a single server port. In cluster mode, each process is capable of functioning on its own CPU. Due to this, the optimal number of processes in a cluster is determined by the number of CPU cores available on the executing computer.

- **Process:** A process is an instance of software that runs on a single CPU. When one runs a program code that has written, it is called a process. A process can be either a Master or a Fork process when running in cluster mode and that the master process is always the first, and it forks into Fork/Worker processes. When a process recognizes itself as a fork, it launches an express instance and starts serving HTTP requests. It is vital to remember that fork processes can crash due to exceptions and it is sometimes necessary to restart them.

- **Port:** A network port is a software construct used to designate a process or application in a network as a communication endpoint. A port is frequently associated with a certain operation or application. This implies that while one process is running on port 3000, no alternate process can use it as long as the original one still runs.

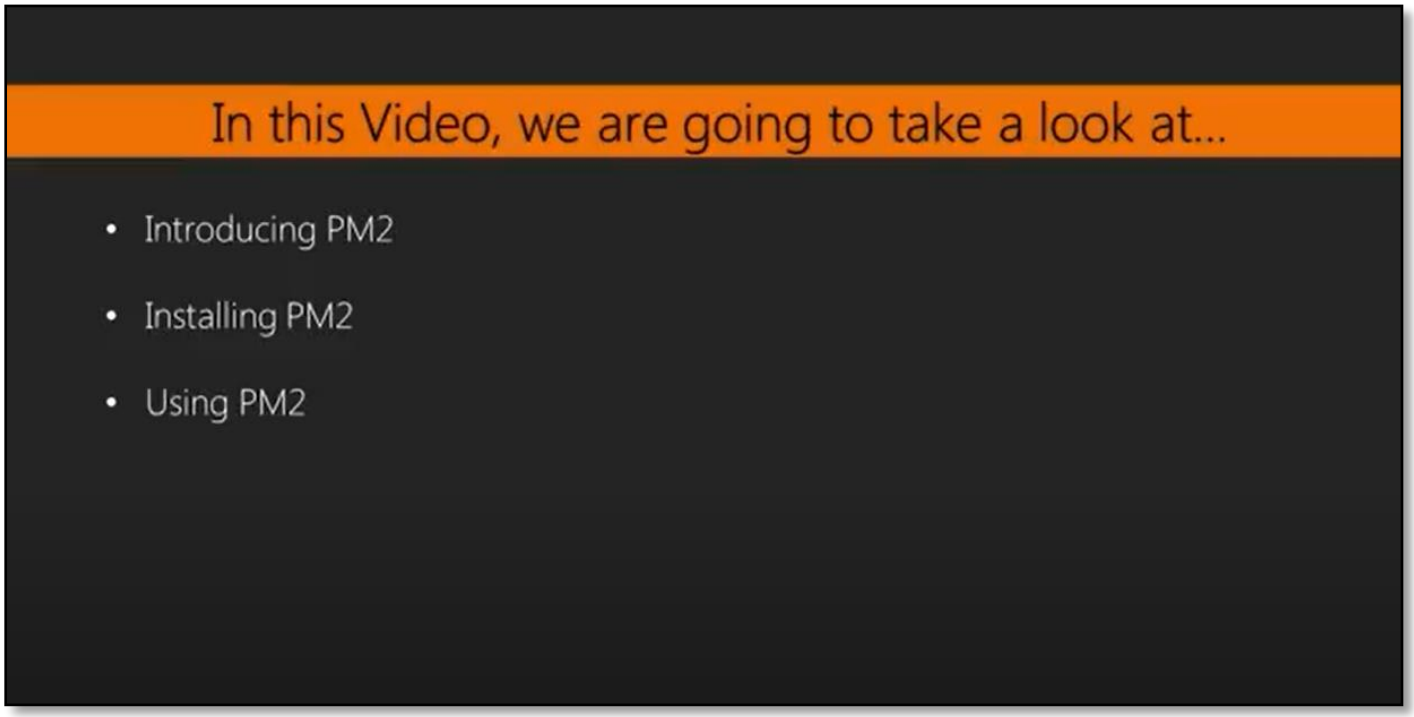
- **Arrow Functions:** Although an arrow function expression is a more compact version of a regular function expression, it has limitations and cannot be utilized in all cases.

Following are the differences and constraints of the arrow function:

1. It should not be used as a method because it lacks its own bindings to this or super.
2. There is nothing new.
3. keyword to be targeted.
4. Not suited for methods such as call, apply, and bind, which all require on setting a scope.
5. They are unable to be utilized as constructors.
6. Within its body, it is unable to use yield.

8.7 Adding Performance Monitoring

CLASS DURATION: 30 MINUTES
VIDEO DURATION: 4.43 MINUTES
FACILITATION GUIDELINES -
Show: Play Adding Performance Monitoring: Session 8 Video – 8.7.



Discussion:

In this video, students will learn that PM2 is a load balancer-assisted production process manager for Node.js applications. It enables one to keep applications running forever, reload them without incurring system downtime, and ease common system management tasks. Explain to them that PM2 is a daemon process manager that will assist in managing and maintaining application and that it is extremely easy to begin with because it comes with a simple and intuitive CLI that can be installed using NPM.

Following is the command to put an application into production mode:

```
$pm2 start app.js
```

Installation:
Following is the way to install the newest version of PM2 via NPM or Yarn:

```
$npm install pm2@latest
```

Following is the way a user can start any application (Node.js, Python, Ruby, binaries in `$PATH`...):

```
$pm2 start app.js
```

Illustrate that utilizing the drop-in replacement command for node, called pm2-runtime, they can run the Node.js application in a hardened production environment.

Explain that there exists a starting script that launches the Node applications, but understanding how it works will free a lot of time. '`pm2-init.sh`' is the name of the script and it is located in the directory '`etc/init.d/`', however, it does not start app.js. Rather, it restarts the process that once ran under PM2 which was about the time the server was last shut down.

Summarize that this is critical and that the node application will refuse restart while the server restarts if it does not appear in the list when one types `pm2 list`. To establish that the programs will resume, ask the students to follow the necessary requirements for starting them using pm2. To rephrase, start the PM2 app.js.

Conclusion: Give the summary of all the topics that we covered in this session.

QUERIES REGARDING SESSION

CLASS DURATION: 10 MINUTES

FACILITATION GUIDELINES

Ask students if they have any queries regarding the session and resolve the queries.