Instructor Notes:

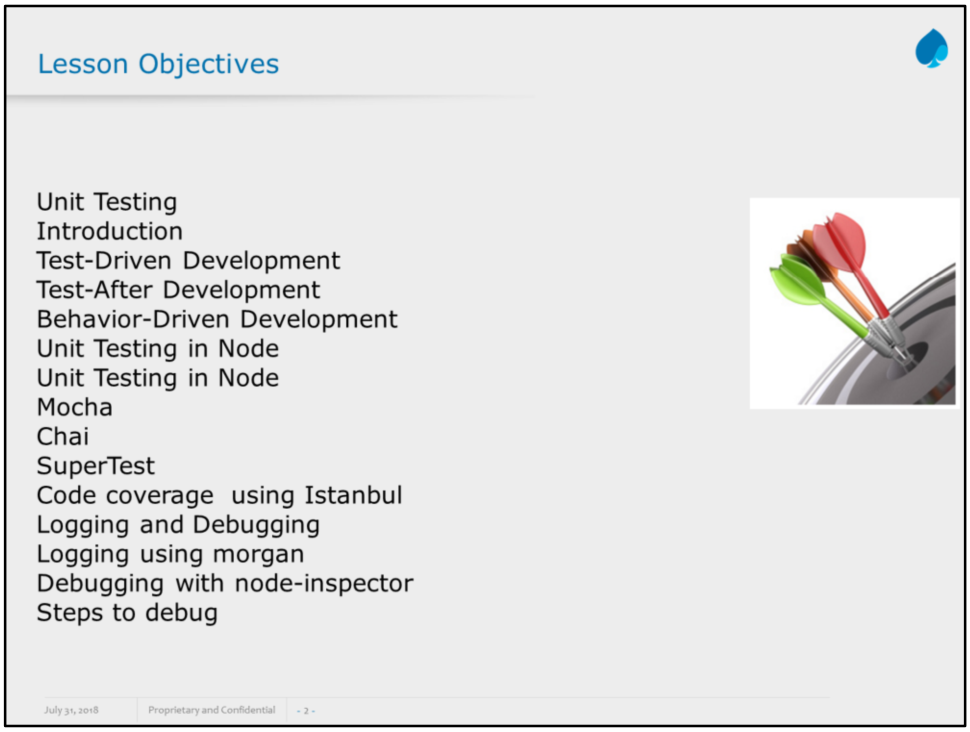
Add instructor notes here.

Node JS

Lesson 05 : Unit Testing, Logging & Debugging

Instructor Notes:

Add instructor notes here.



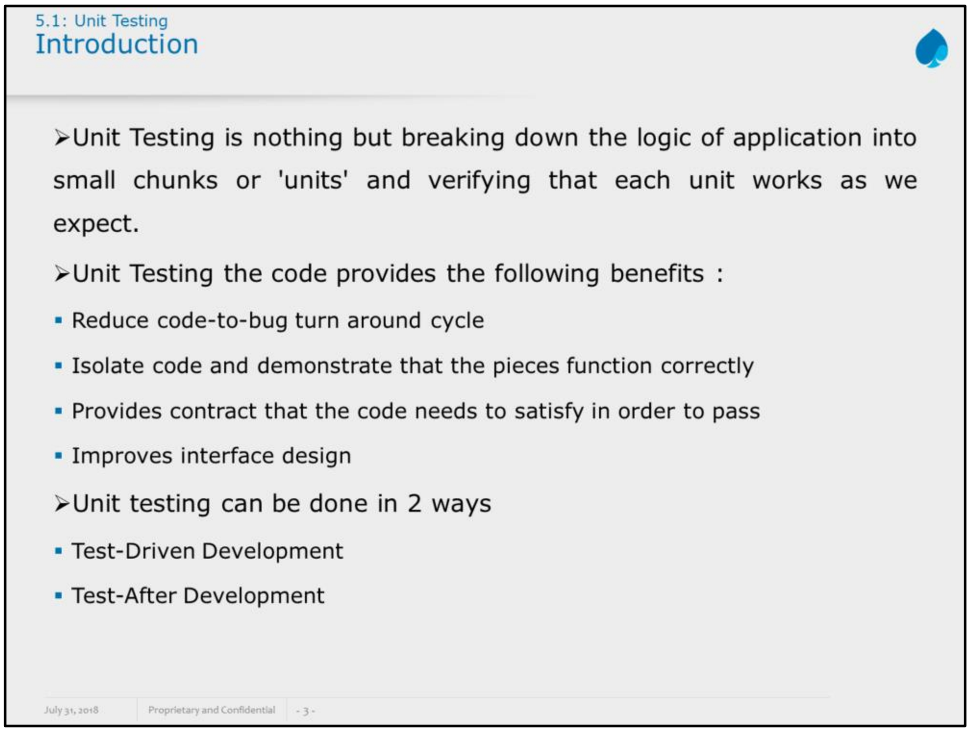
Lesson Objectives

Unit Testing
Introduction
Test-Driven Development
Test-After Development
Behavior-Driven Development
Unit Testing in Node
Unit Testing in Node
Mocha
Chai
SuperTest
Code coverage using Istanbul
Logging and Debugging
Logging using morgan
Debugging with node-inspector
Steps to debug

July 31, 2018 Proprietary and Confidential - 2 -

Instructor Notes:

Add instructor notes here.



5.1: Unit Testing

Introduction

➢Unit Testing is nothing but breaking down the logic of application into small chunks or 'units' and verifying that each unit works as we expect.

➢Unit Testing the code provides the following benefits :

- Reduce code-to-bug turn around cycle
- Isolate code and demonstrate that the pieces function correctly
- Provides contract that the code needs to satisfy in order to pass
- Improves interface design

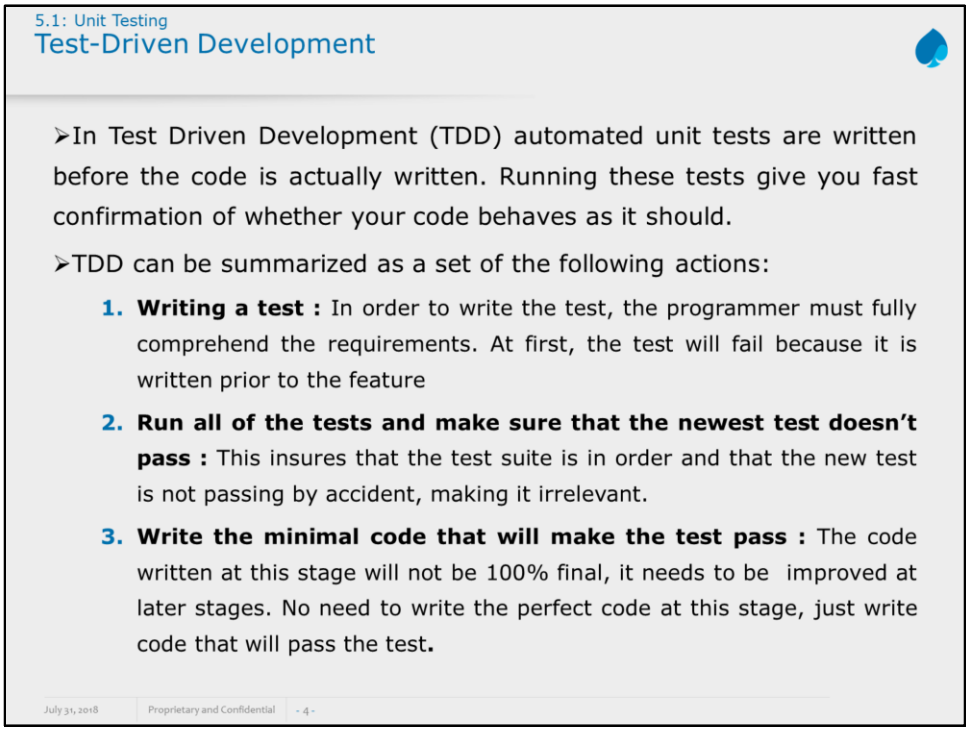
➢Unit testing can be done in 2 ways

- Test-Driven Development
- Test-After Development

July 31, 2018 | Proprietary and Confidential | - 3 -

Instructor Notes:

Add instructor notes here.



5.1: Unit Testing

Test-Driven Development

➢In Test Driven Development (TDD) automated unit tests are written before the code is actually written. Running these tests give you fast confirmation of whether your code behaves as it should.

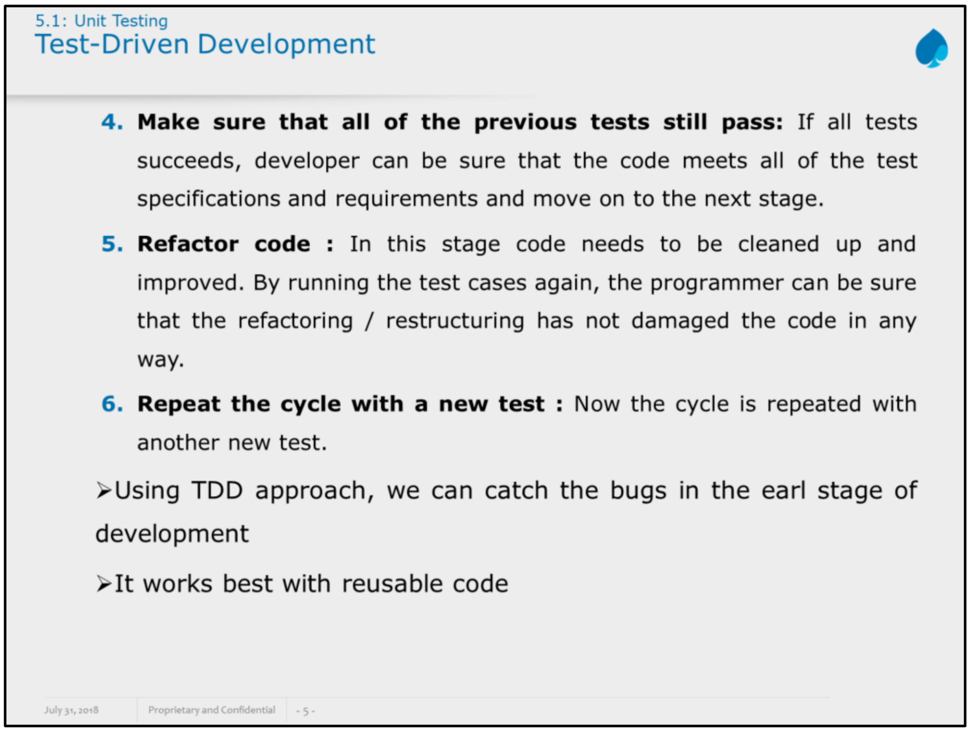
➢TDD can be summarized as a set of the following actions:

1. **Writing a test :** In order to write the test, the programmer must fully comprehend the requirements. At first, the test will fail because it is written prior to the feature
2. **Run all of the tests and make sure that the newest test doesn't pass :** This insures that the test suite is in order and that the new test is not passing by accident, making it irrelevant.
3. **Write the minimal code that will make the test pass :** The code written at this stage will not be 100% final, it needs to be improved at later stages. No need to write the perfect code at this stage, just write code that will pass the test**.**

July 31, 2018 Proprietary and Confidential - 4 -

Instructor Notes:

Add instructor notes here.



5.1: Unit Testing

Test-Driven Development

4. **Make sure that all of the previous tests still pass:** If all tests succeeds, developer can be sure that the code meets all of the test specifications and requirements and move on to the next stage.
5. **Refactor code :** In this stage code needs to be cleaned up and improved. By running the test cases again, the programmer can be sure that the refactoring / restructuring has not damaged the code in any way.
6. **Repeat the cycle with a new test :** Now the cycle is repeated with another new test.

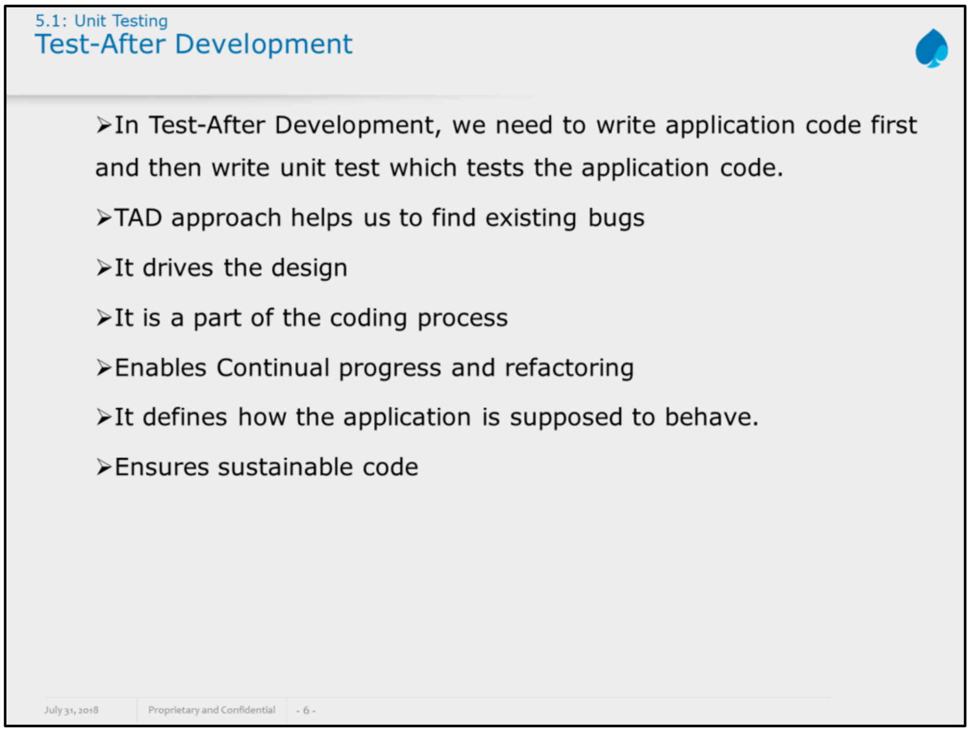
➢Using TDD approach, we can catch the bugs in the earl stage of development

➢It works best with reusable code

July 31, 2018 Proprietary and Confidential - 5 -

Instructor Notes:

Add instructor notes here.



5.1: Unit Testing

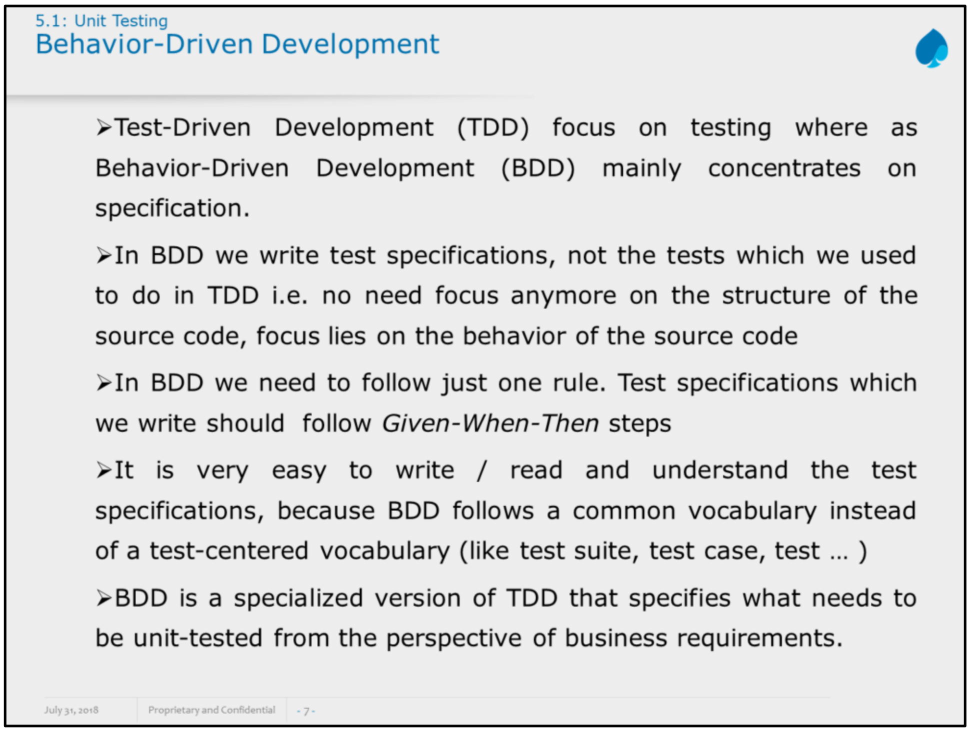
Test-After Development

- ➢In Test-After Development, we need to write application code first and then write unit test which tests the application code.
- ➢TAD approach helps us to find existing bugs
- ➢It drives the design
- ➢It is a part of the coding process
- ➢Enables Continual progress and refactoring
- ➢It defines how the application is supposed to behave.
- ➢Ensures sustainable code

July 31, 2018 | Proprietary and Confidential | - 6 -

Instructor Notes:

Add instructor notes here.



5.1: Unit Testing

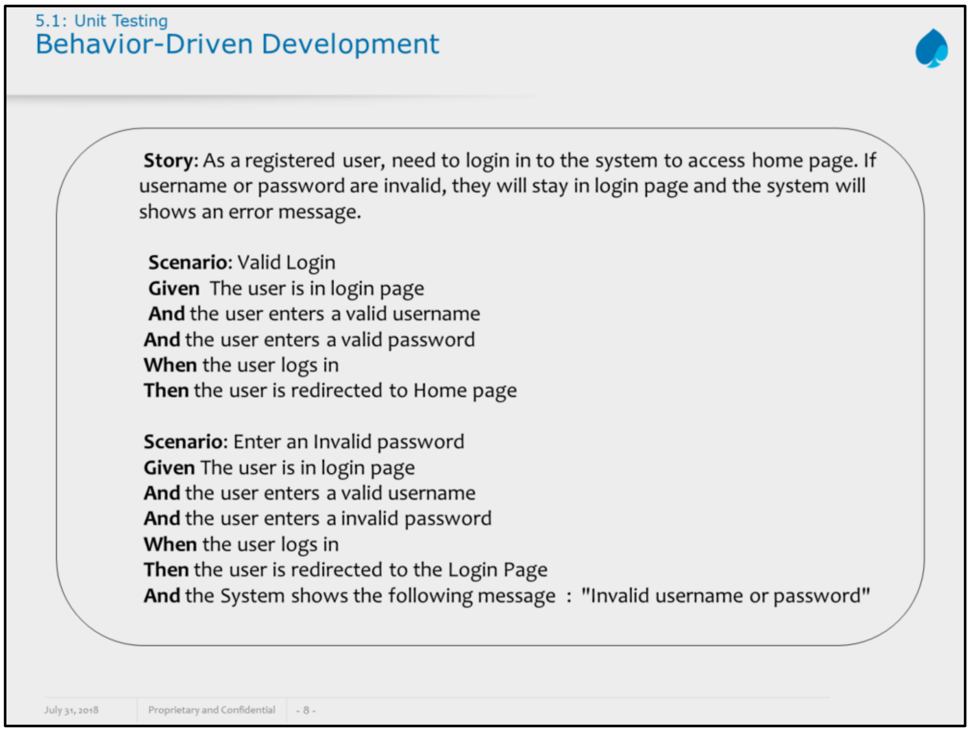
Behavior-Driven Development

- Test-Driven Development (TDD) focus on testing where as Behavior-Driven Development (BDD) mainly concentrates on specification.
- In BDD we write test specifications, not the tests which we used to do in TDD i.e. no need focus anymore on the structure of the source code, focus lies on the behavior of the source code
- In BDD we need to follow just one rule. Test specifications which we write should follow *Given-When-Then* steps
- It is very easy to write / read and understand the test specifications, because BDD follows a common vocabulary instead of a test-centered vocabulary (like test suite, test case, test …)
- BDD is a specialized version of TDD that specifies what needs to be unit-tested from the perspective of business requirements.

July 31, 2018 Proprietary and Confidential - 7 -

Instructor Notes:

Add instructor notes here.



5.1: Unit Testing

Behavior-Driven Development

Story: As a registered user, need to login in to the system to access home page. If username or password are invalid, they will stay in login page and the system will shows an error message.

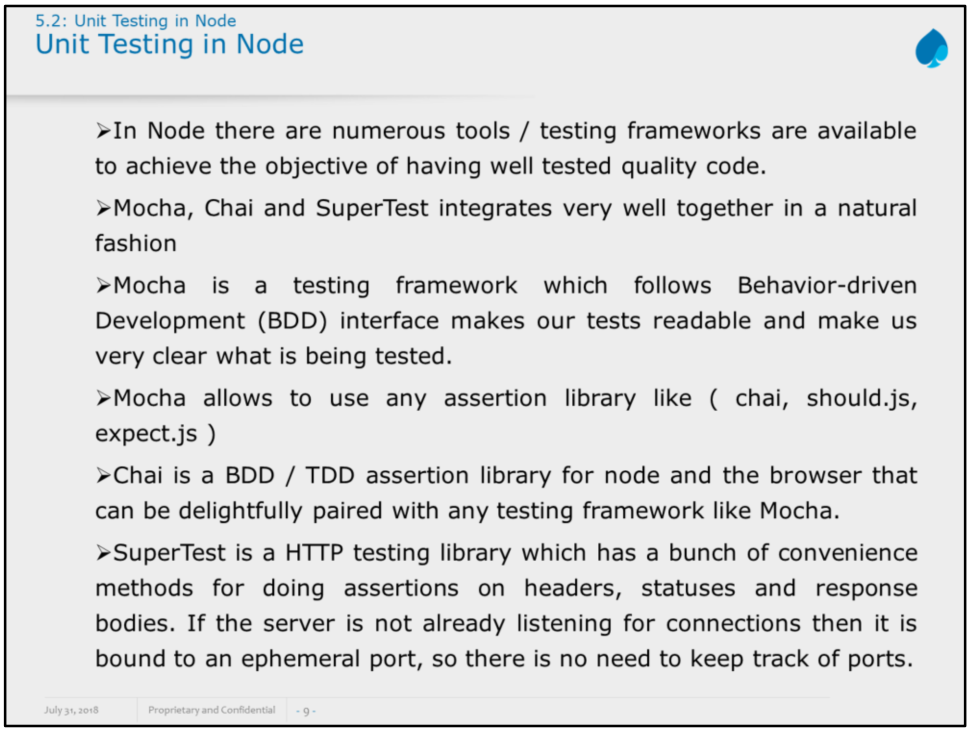
Scenario: Valid Login
Given The user is in login page
And the user enters a valid username
And the user enters a valid password
When the user logs in
Then the user is redirected to Home page

Scenario: Enter an Invalid password
Given The user is in login page
And the user enters a valid username
And the user enters a invalid password
When the user logs in
Then the user is redirected to the Login Page
And the System shows the following message : "Invalid username or password"

July 31, 2018 Proprietary and Confidential - 8 -

Instructor Notes:

Add instructor notes here.



5.2: Unit Testing in Node

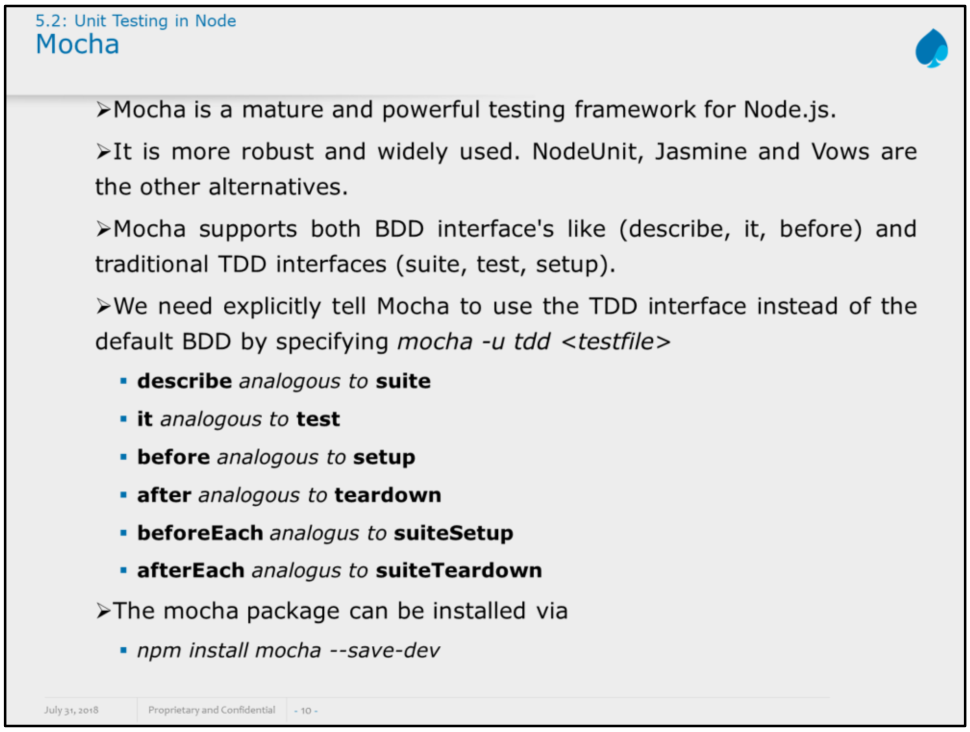
Unit Testing in Node

- ➢In Node there are numerous tools / testing frameworks are available to achieve the objective of having well tested quality code.
- ➢Mocha, Chai and SuperTest integrates very well together in a natural fashion
- ➢Mocha is a testing framework which follows Behavior-driven Development (BDD) interface makes our tests readable and make us very clear what is being tested.
- ➢Mocha allows to use any assertion library like (chai, should.js, expect.js)
- ➢Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any testing framework like Mocha.
- ➢SuperTest is a HTTP testing library which has a bunch of convenience methods for doing assertions on headers, statuses and response bodies. If the server is not already listening for connections then it is bound to an ephemeral port, so there is no need to keep track of ports.

July 31, 2018 | Proprietary and Confidential | - 9 -

Instructor Notes:

Add instructor notes here.



5.2: Unit Testing in Node

Mocha

- Mocha is a mature and powerful testing framework for Node.js.
- It is more robust and widely used. NodeUnit, Jasmine and Vows are the other alternatives.
- Mocha supports both BDD interface's like (describe, it, before) and traditional TDD interfaces (suite, test, setup).
- We need explicitly tell Mocha to use the TDD interface instead of the default BDD by specifying *mocha -u tdd <testfile>*
  - **describe** *analogous to* **suite**
  - **it** *analogous to* **test**
  - **before** *analogous to* **setup**
  - **after** *analogous to* **teardown**
  - **beforeEach** *analogus to* **suiteSetup**
  - **afterEach** *analogus to* **suiteTeardown**
- The mocha package can be installed via
  - *npm install mocha --save-dev*

July 31, 2018 Proprietary and Confidential - 10 -

describe(description, callback): This is the basic method that wraps each test suite with a description. The callback function is used to define test specifications or subsuites.

it(description, callback): This is the basic method that wraps each test specification with a description. The callback function is used to definethe actual test logic.

before(callback): This is a hook function that is executed once before all the tests in a test suite.

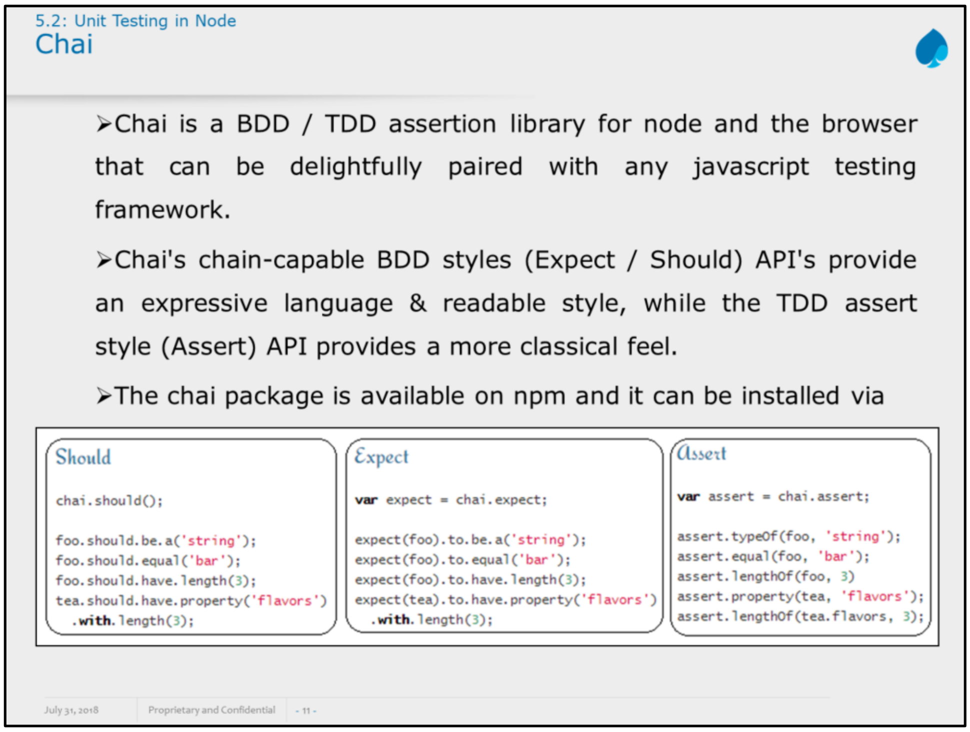
beforeEach(callback): This is a hook function that is executed before each test specification in a test suite.

after(callback): This is a hook function that is executed once after all the tests in a test suite are executed.

afterEach(callback): This is a hook function that is executed after each test specification in a test-suite is executed.

Instructor Notes:

Add instructor notes here.



5.2: Unit Testing in Node

Chai

- Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.
- Chai's chain-capable BDD styles (Expect / Should) API's provide an expressive language & readable style, while the TDD assert style (Assert) API provides a more classical feel.
- The chai package is available on npm and it can be installed via

Should

```
chai.should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.length(3);
tea.should.have.property('flavors')
  .with.length(3);
```

Expect

```
var expect = chai.expect;

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.length(3);
expect(tea).to.have.property('flavors')
  .with.length(3);
```

Assert

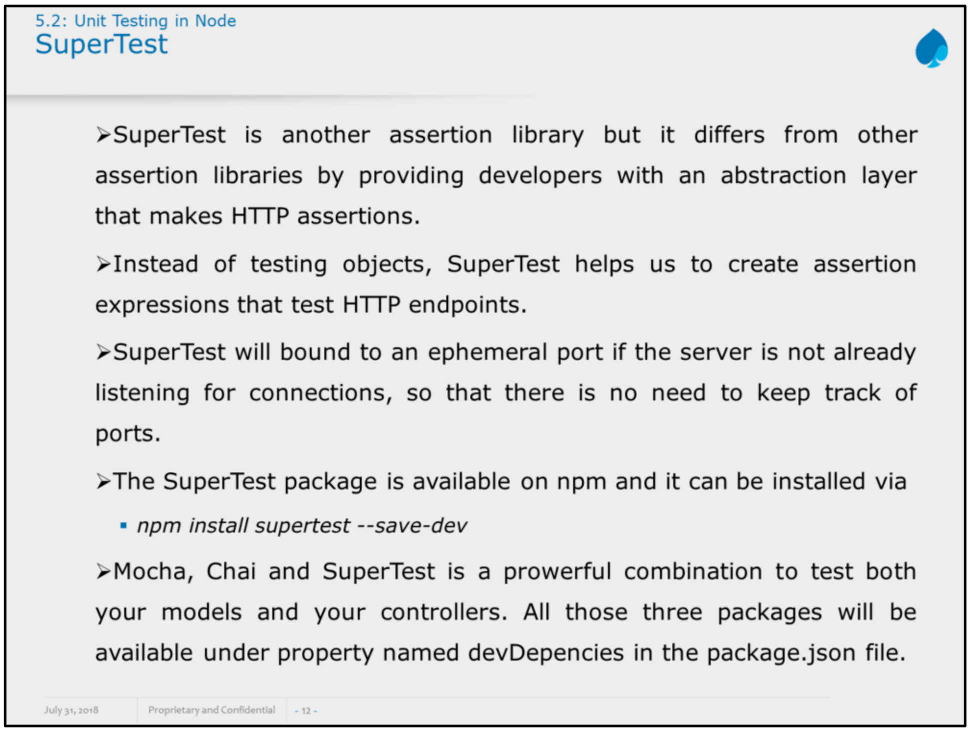
```
var assert = chai.assert;

assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
assert.lengthOf(foo, 3)
assert.property(tea, 'flavors');
assert.lengthOf(tea.flavors, 3);
```

July 31, 2018 Proprietary and Confidential - 11 -

Instructor Notes:

Add instructor notes here.



5.2: Unit Testing in Node

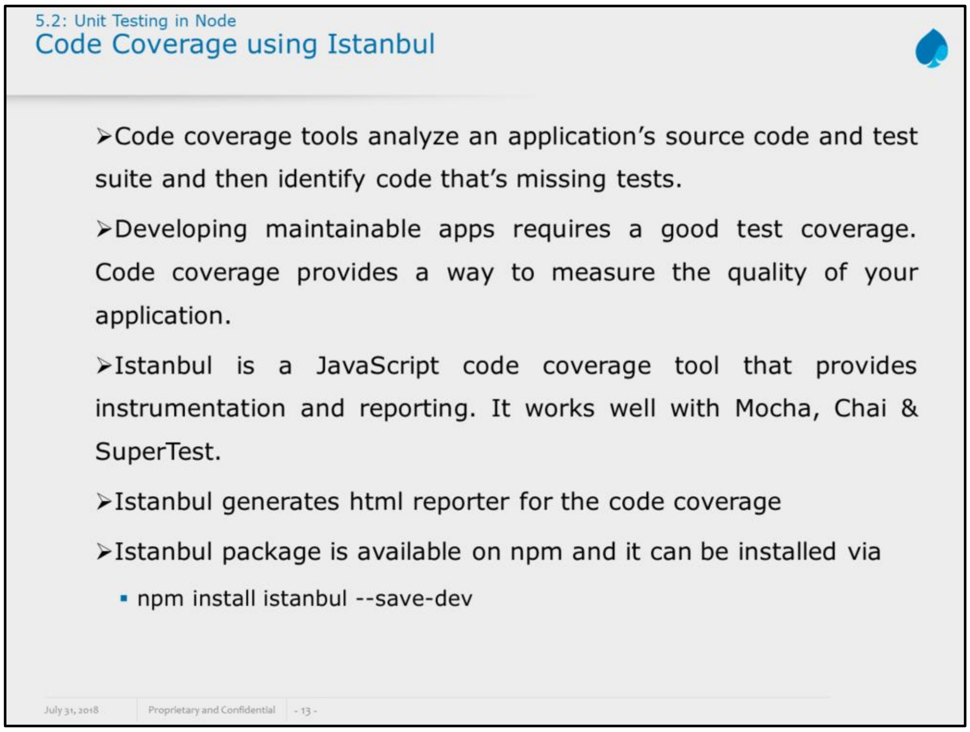
SuperTest

- ➢SuperTest is another assertion library but it differs from other assertion libraries by providing developers with an abstraction layer that makes HTTP assertions.
- ➢Instead of testing objects, SuperTest helps us to create assertion expressions that test HTTP endpoints.
- ➢SuperTest will bound to an ephemeral port if the server is not already listening for connections, so that there is no need to keep track of ports.
- ➢The SuperTest package is available on npm and it can be installed via
  - *npm install supertest --save-dev*
- ➢Mocha, Chai and SuperTest is a prowerful combination to test both your models and your controllers. All those three packages will be available under property named devDepencies in the package.json file.

July 31, 2018 | Proprietary and Confidential | - 12 -

Instructor Notes:

Add instructor notes here.



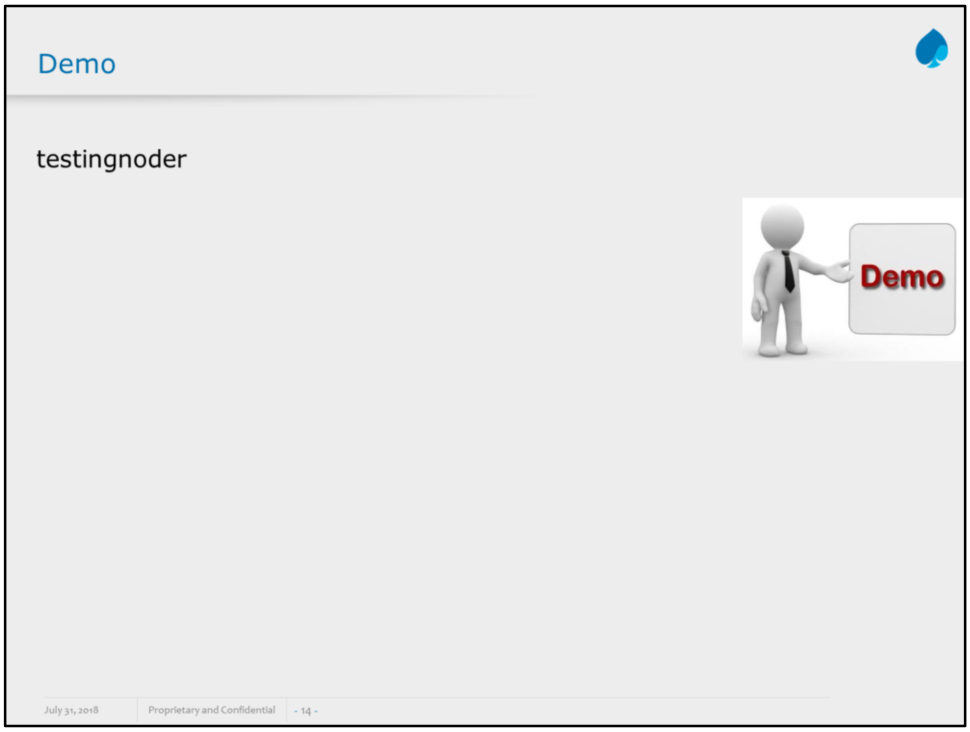
5.2: Unit Testing in Node

Code Coverage using Istanbul

- ➢Code coverage tools analyze an application’s source code and test suite and then identify code that’s missing tests.
- ➢Developing maintainable apps requires a good test coverage. Code coverage provides a way to measure the quality of your application.
- ➢Istanbul is a JavaScript code coverage tool that provides instrumentation and reporting. It works well with Mocha, Chai & SuperTest.
- ➢Istanbul generates html reporter for the code coverage
- ➢Istanbul package is available on npm and it can be installed via
  - npm install istanbul --save-dev

July 31, 2018 | Proprietary and Confidential | - 13 -

Instructor Notes:



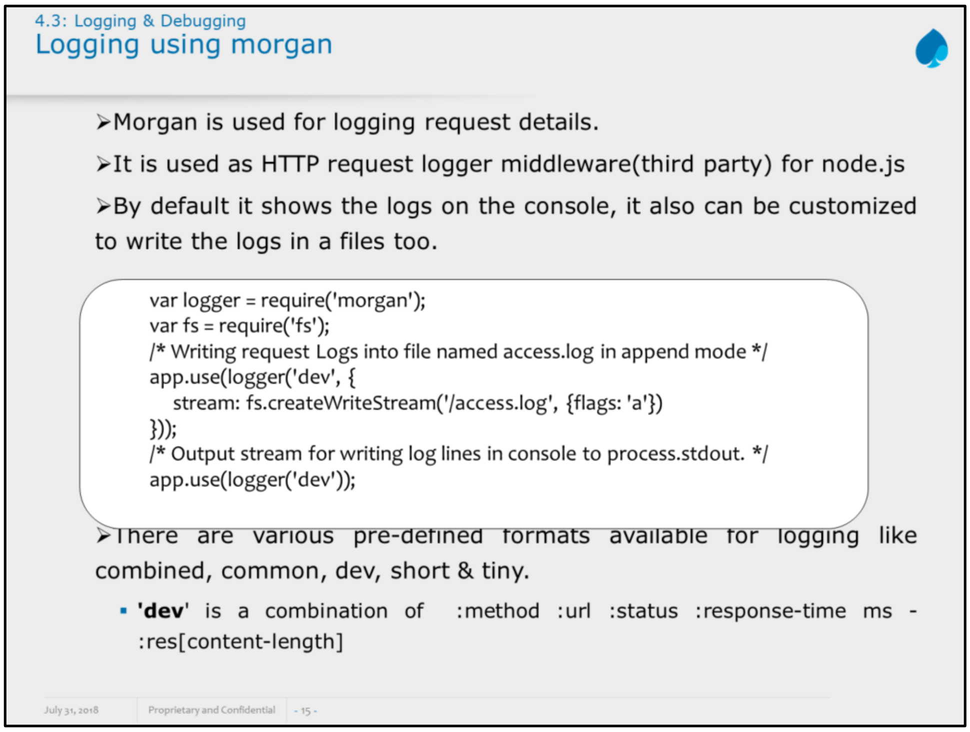
Demo

testingnoder

July 31, 2018 Proprietary and Confidential - 14 -

Instructor Notes:

Add instructor notes here.



4.3: Logging & Debugging

Logging using morgan

➢Morgan is used for logging request details.

➢It is used as HTTP request logger middleware(third party) for node.js

➢By default it shows the logs on the console, it also can be customized to write the logs in a files too.

```
var logger = require('morgan');
var fs = require('fs');
/* Writing request Logs into file named access.log in append mode */
app.use(logger('dev', {
   stream: fs.createWriteStream('/access.log', {flags: 'a'})
}));
/* Output stream for writing log lines in console to process.stdout. */
app.use(logger('dev'));
```

➢There are various pre-defined formats available for logging like combined, common, dev, short & tiny.

- **'dev'** is a combination of :method :url :status :response-time ms - :res[content-length]

July 31, 2018 Proprietary and Confidential - 15 -

:date[format] - The current date and time in UTC.

:http-version - The HTTP version of the request.

:method - The HTTP version of the request.

:referrer - The Referrer header of the request. This will use the standard mis-spelled Referer header if exists, otherwise Referrer.

:remote-addr - The remote address of the request. This will use req.ip, otherwise the standard req.connection.remoteAddress value (socket address).

:remote-user - The user authenticated as part of Basic auth for the request.

:req[header] - The given header of the request.

:res[header] - The given header of the response.

:response-time - The time between the request coming into morgan and when the response headers are written, in milliseconds.

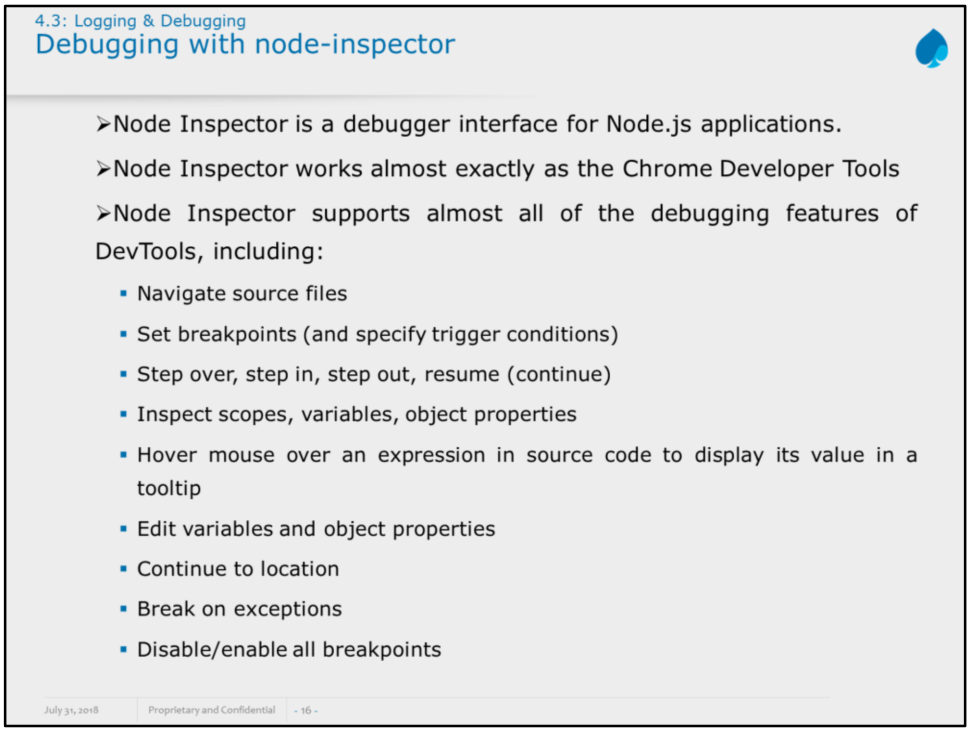
:status - The status code of the response.

:url - The URL of the request. This will use req.originalUrl if exists, otherwise req.url.

:user-agent - The contents of the User-Agent header of the request.

Instructor Notes:

Add instructor notes here.



4.3: Logging & Debugging

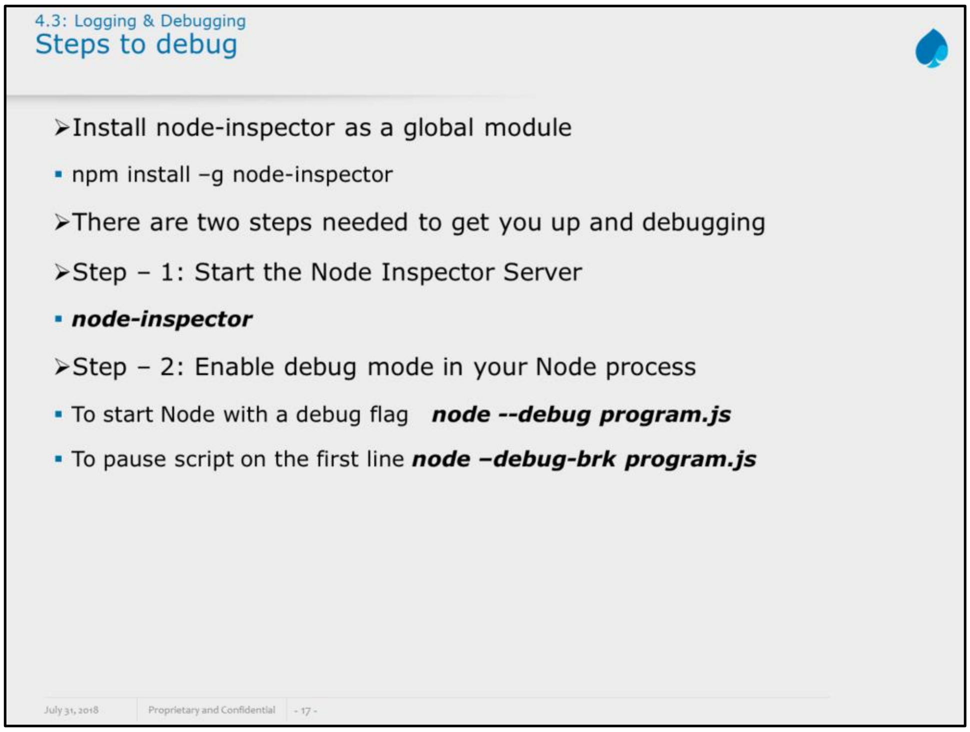
Debugging with node-inspector

- ➢Node Inspector is a debugger interface for Node.js applications.
- ➢Node Inspector works almost exactly as the Chrome Developer Tools
- ➢Node Inspector supports almost all of the debugging features of DevTools, including:
  - Navigate source files
  - Set breakpoints (and specify trigger conditions)
  - Step over, step in, step out, resume (continue)
  - Inspect scopes, variables, object properties
  - Hover mouse over an expression in source code to display its value in a tooltip
  - Edit variables and object properties
  - Continue to location
  - Break on exceptions
  - Disable/enable all breakpoints

July 31, 2018 | Proprietary and Confidential | - 16 -

Instructor Notes:

Add instructor notes here.



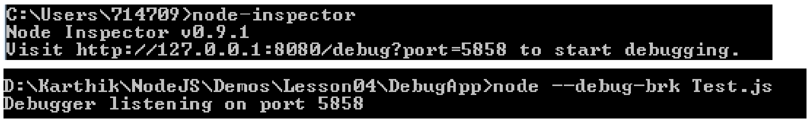
4.3: Logging & Debugging

Steps to debug

- ➢Install node-inspector as a global module
  - npm install –g node-inspector
- ➢There are two steps needed to get you up and debugging
- ➢Step – 1: Start the Node Inspector Server
  - ***node-inspector***
- ➢Step – 2: Enable debug mode in your Node process
  - To start Node with a debug flag ***node --debug program.js***
  - To pause script on the first line ***node –debug-brk program.js***

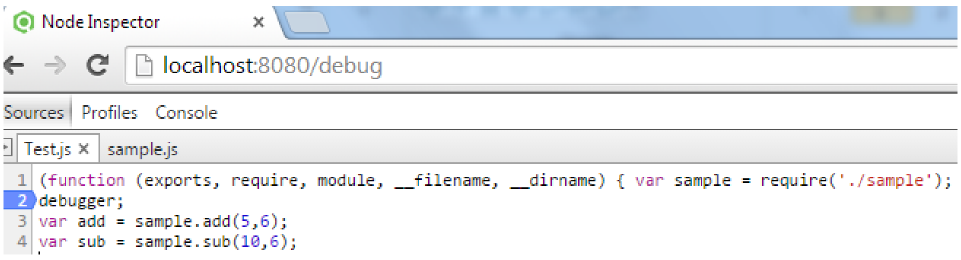
July 31, 2018 Proprietary and Confidential - 17 -

Debugging node with node-inspector



```
C:\Users\714709>node-inspector
Node Inspector v0.9.1
Visit http://127.0.0.1:8080/debug?port=5858 to start debugging.
```

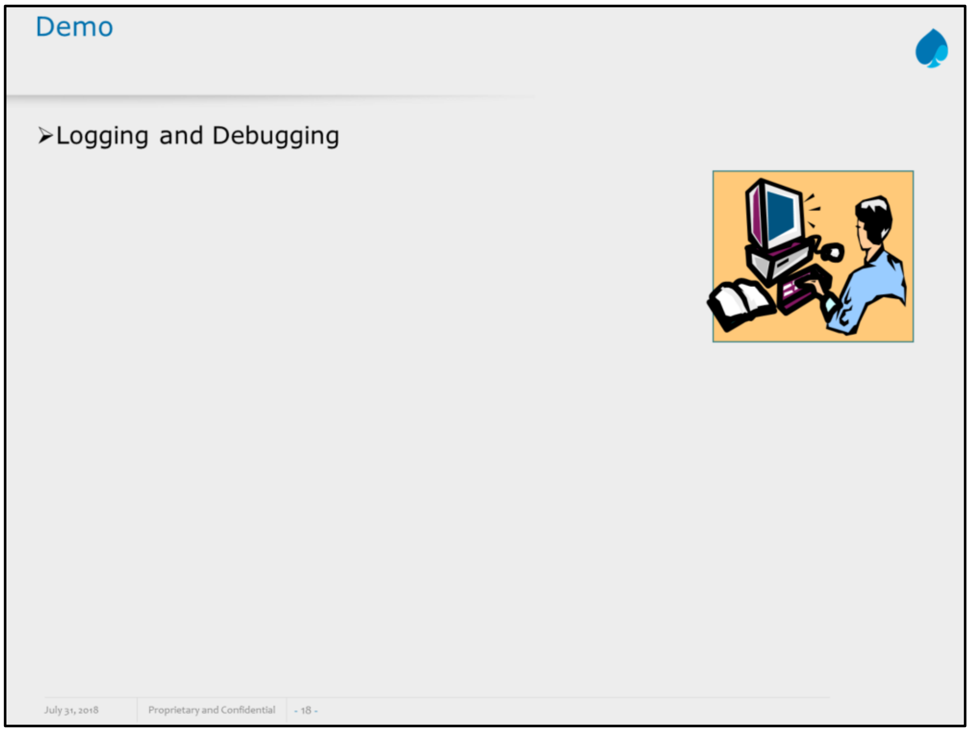
```
D:\Karthik\NodeJS\Demos\Lesson04\DebugApp>node --debug-brk Test.js
Debugger listening on port 5858
```



```
(function (exports, require, module, __filename, __dirname) { var sample = require('./sample');
debugger;
var add = sample.add(5,6);
var sub = sample.sub(10,6);
```

Instructor Notes:

Add instructor notes here.



Demo

➢Logging and Debugging

July 31, 2018 Proprietary and Confidential - 18 -